Contents

Chapter 1

Introduction

Chapter 2

Discrete-Time Stochastic Optimal Control

2.1 Sequential Decision Problems

In sequential decision problems, an agent interacts with an environment by selecting a series of actions in order to complete a specific task. During this interaction, the agent receives a numerical reward from the environment and his goal is to find the best strategy in order to maximize a certain measure of his performance. The environment evolves stochastically and may be influenced by the interaction with the agent, so that each action taken by the agent may influence the circumstances under which future decisions will be made. Therefore, the agent must balance his desire to obtain a large reward today by acting greedily and the opportunities that will be available in the future. While this setting appears quite simple, it is general enough to encompasses a wide range of applications in different fields. A classical example is portfolio management, where an investor must allocate his capital so as to maximize his long-term profits. Another standard example is chess, where two players successively move pieces around the chessboard to checkmate the opponent's king.
The purpose of the following sections is to introduce the notation that will be used in the rest of this work and to recall the fundamental concepts and results of the discrete-time stochastic optimal control theory, which is the standard framework to study sequential decisions problems in mathematical terms. Since our discussion will be far from being comprehensive, we refer the reader to the extensive literature on the subject, such as [11], [59], [10].

2.2 Markov Decision Processes

A sequential decision problem under uncertainty can be schematized as in Figure 2.1: at a given time t, the agent (also known as decision maker or controller) observes the state s_t of the system (also know as environment) and subsequently performs an action a_t. Following this action, the agent receives an immediate reward r_{t+1} (or incurs an immediate cost) and the system evolves to a new state according to a probability distribution which depends on the action selected by the agent. At the subsequent time $t+1$, the agent selects a new action given the new state of the system and the process repeats. This interaction can be modeled rigorously using Markov decision processes.

Definition 2.2.1 (Markov Decision Process)**.** *A Markov decision process (MDP) is a stochastic dynamical system specified by the tuple* $< \mathbb{S}, \mathbb{A}, \mathcal{P}, \mathcal{R}, \gamma >$*, where*

i) $(\mathbb{S}, \mathcal{S})$ *is a measurable space, called the state space.*
ii) $(\mathbb{A}, \mathcal{A})$ *is a meausrable space, called the action space.*
iii) $\mathcal{P} : \mathbb{S} \times \mathbb{A} \times \mathcal{S} \to \mathbb{R}$ *is a Markov transition kernel, i.e.*
  a) for every $s \in \mathbb{S}$ *and* $a \in \mathbb{A}$, $B \mapsto \mathcal{P}(s, a, B)$ *is a probability distribution over* $(\mathbb{S}, \mathcal{S})$.
  b) for every $B \in \mathcal{S}$, $(s, a) \mapsto \mathcal{P}(s, a, B)$ *is a measurable function on* $\mathbb{S} \times \mathbb{A}$.
iv) $\mathcal{R} : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ *is a reward function.*
v) $\gamma \in (0, 1)$ *is a discount factor.*

Typically, the state space (and similarly the action space) will be either finite, i.e. $\mathbb{S} = \{s_1, \dots, s_d\}$, or continuous, i.e. $\mathbb{S} \subseteq \mathbb{R}^{D_s}$. The kernel $\mathcal{P}$ describes the random evolution of the system: suppose that at time t the system is in state s and that the agent takes action a, then, regardless of the previous history of the system, the probability to find the system in a state belonging to $B \in \mathcal{S}$ at time $t+1$ is given by $\mathcal{P}(s, a, B)$, i.e.

$$\mathcal{P}(s, a, B) = \mathbb{P}\left(S_{t+1} \in B | S_t = s, A_t = a\right) \tag{2.1}$$

Following this random transition, the agent receives a stochastic reward R_{t+1}. The reward function $\mathcal{R}(s, a)$ gives the expected reward obtained when action a is taken in state s, i.e.

$$\mathcal{R}(s, a) = \mathbb{E}\left[R_{t+1} | S_t = s, A_t = a\right] \tag{2.2}$$

This setting can be easily generalized to the following cases

1. The initial state of the system is a random variable $S_0 \sim \mu$.
2. The actions that an agent can select depend on the state of the system.

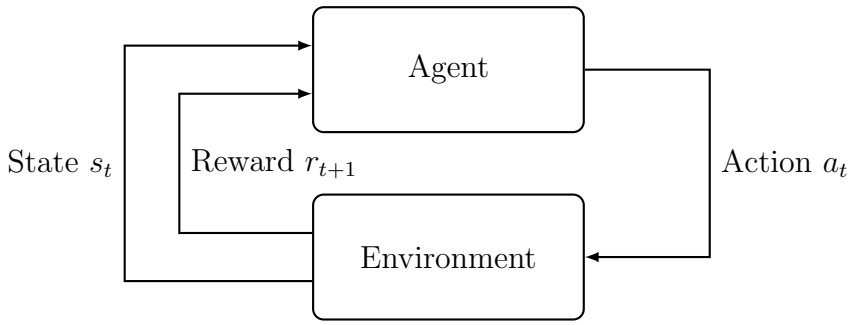

Figure 2.1: Agent-environment interaction in sequential decision problems.

2.3 Policies

At any time step, the agent selects his actions according to a certain policy.

Definition 2.3.1 (Policy)**.** *A policy is a function* $\pi : \mathbb{S} \times \mathcal{A} \to \mathbb{R}$ *such that*
i) for every $s \in \mathbb{S}$, $C \mapsto \pi(s, C)$ *is a probability distribution over* $(\mathbb{A}, \mathcal{A})$.
ii) for every $C \in \mathcal{A}$, $s \mapsto \pi(s, C)$ *is a measurable function.*

Intuitively, a policy represents a stochastic mapping from the current state of the system to actions. Deterministic policies are a particular case of this general definition. We assumed that the agent's policy is stationary and only depends on the current state of the system. We might in fact consider more general policies that depends on the whole history of the system. However, as we will see, we can always find an optimal policy that depends only on the current state, so that our definition is not restrictive. A policy π and an initial state $s_0 \in \mathbb{S}$ determine a random state-action-reward sequence $\{(S_t, A_t, R_{t+1})\}_{t\geq 0}$ with values on $\mathbb{S} \times \mathbb{A} \times \mathbb{R}$ following the mechanism described above.

Definition 2.3.2 (History)**.** *Given an initial state* $s_0 \in \mathbb{S}$ *and a policy* π*, a history (or equivalently trajectory or roll-out) of the system is a random sequence* $H_\pi = \{(S_t, A_t)\}_{t\geq 0}$ *with values in* $\mathbb{S} \times \mathbb{A}$*, defined on some probability space* $(\Omega, \mathcal{F}, \mathbb{P})$*, such that for* $t = 0, 1, \ldots$

$$\begin{cases} S_0 = s_0 \\ A_t \sim \pi(S_t, \cdot) \\ S_{t+1} \sim \mathcal{P}(S_t, A_t, \cdot) \end{cases} \tag{2.3}$$

we will denote by $(\mathbb{H}, \mathcal{H})$ *the measurable space of all possible histories.*

Moreover, we observe that
i) the state sequence $\{S_t\}_{t\geq 0}$ is a Markov process $< \mathbb{S}, \mathcal{P}_\pi >$

ii) the state-reward sequence $\{(S_t, R_t)\}_{t\geq 0}$ is a Markov reward process $< \mathbb{S}, \mathcal{P}_\pi, \mathcal{R}_\pi, \gamma >$

where we denoted

$$\begin{aligned} \mathcal{P}_\pi(s, s') &= \int_{\mathbb{A}} \pi(s, a)\mathcal{P}(s, a, s')da \\ \mathcal{R}_\pi(s) &= \int_{\mathbb{A}} \pi(s, a)\mathcal{R}(s, a)da \end{aligned} \tag{2.4}$$

In stochastic optimal control, the goal of the agent is to find a policy that maximizes a measure of the agent's long-term performance. In the next sections we discuss some objective functions that are commonly used in the infinite horizon framework.

2.4 Discounted Reward Formulation

In the discounted reward formulation, the agent's performance is measured as the expected return obtained following a specific policy.

Definition 2.4.1 (Return)**.** *The return is the total discounted reward obtained by the agent starting from t*

$$G_t = \sum_{t=0}^{\infty} \gamma^t R_{t+k+1} \tag{2.5}$$

where $0 < \gamma < 1$ *is the discount factor.*

In some domains, such as economics, discounting can be used to represent interest earned on rewards, so that an action that generates an immediate reward will be preferred over one that generates the same reward some steps into the future. Discounting thus models the trade-off between immediate and delayed reward: if $\gamma = 0$ the agent selects his actions in a myopic way, while if $\gamma \to 1$ he acts in a far-sighted manner. There are other possilble reasons for discounting future rewards. The first is because it is mathematically convenient, as it avoids infinite returns and it solves many convergence issues. Another interpretation is that it models the uncertainty about the future, which may not be fully represented. Indeed, the discount factor could be seen as the probability that the world does not stop at a given time step. Since the return is stochastic, we consider its expected value.

Definition 2.4.2 (State-Value Function)**.** *The state-value function* $V_\pi : \mathbb{S} \to \mathbb{R}$ *is the expected return that can be obtained starting from a state and following policy* π

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s\right] \tag{2.6}$$

where $\mathbb{E}_\pi$ indicates that all the actions are selected according to policy π. The state-value function measures how good it is for the agent to be in a given state and follow a certain policy. Similarly, we can introduce an action-value function that measures how good it is for the agent to be in a state, take a certain action and then follow the policy.

Definition 2.4.3 (Action-Value Function)**.** *The action-value function* $Q_\pi : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ *is the expected return that can be obtained starting from a state, taking an action and then following policy* π

$$Q_\pi(s,a) = \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right] \tag{2.7}$$

We have the following relationship between V_π and Q_π

$$V_\pi(s) = \int_{\mathbb{A}} \pi(s,a) Q_\pi(s,a) da \tag{2.8}$$

Almost all reinforcement learning algorithms are designed for estimating these value functions and are typically based on the Bellman equations.

Proposition 2.4.1 (Bellman Expectation Equations)**.**

$$V_\pi(s) = \mathcal{R}_\pi(s) + \gamma T_\pi V_\pi(s) \tag{2.9}$$

$$\begin{aligned} Q_\pi(s,a) &= \mathcal{R}(s,a) + \gamma T_a V_\pi(s) \\ &= \mathcal{R}(s,a) + \gamma \int_{\mathbb{S}} \mathcal{P}(s,a,s') \int_{\mathbb{A}} \pi(s',a') Q_\pi(s',a') da' ds' \end{aligned} \tag{2.10}$$

where we denoted by T_a *(resp.* T_π*) the transition operator for action* a *(resp. for policy* π*)*

$$\begin{aligned} T_a V(s) &= \int_{\mathbb{S}} \mathcal{P}(s,a,s') V_\pi(s') ds' \\ T_\pi V(s) &= \int_{\mathbb{A}} \pi(s,a) \int_{\mathbb{S}} \mathcal{P}(s,a,s') V_\pi(s') ds' da \end{aligned} \tag{2.11}$$

If we introduce the Bellman expection operator B_π, defined as

$$B_\pi V_\pi(s) = \mathcal{R}_\pi(s) + \gamma T_\pi V_\pi(s) \tag{2.12}$$

Then Eq. (2.9) can be written as a fixed-point equation

$$V_\pi(s) = B_\pi V_\pi(s) \tag{2.13}$$

which, under some simple assumptions on the reward functions, admits a unique solution by the contraction mapping theorem. A similar argument applies to Eq. (2.10). The agent's goal is to select a policy π_* that maximizes his expected return in all possible states. Such a policy is called *optimal*.

Definition 2.4.4 (Optimal State-Value Function)**.** *The optimal state-value function $V_* : \mathbb{S} \to \mathbb{R}$ is the largest expected return that can be obtained starting from a state*

$$V_*(s) = \sup_\pi V_\pi(s) \tag{2.14}$$

Definition 2.4.5 (Optimal Action-Value Function)**.** *The optimal action-value function $Q_* : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ is the largest expected return that can be obtained starting from a state and taking an action*

$$Q_*(s,a) = \sup_\pi Q_\pi(s,a) \tag{2.15}$$

The optimal value functions satisfy the following Bellman equations.

Proposition 2.4.2 (Bellman Optimality Equations)**.**

$$V_*(s) = \sup_a Q_*(s,a) = \sup_a \{\mathcal{R}(s,a) + \gamma T_a V_*(s)\} \tag{2.16}$$

$$\begin{aligned} Q_*(s,a) &= \mathcal{R}(s,a) + \gamma T_a V_*(s) \\ &= \mathcal{R}(s,a) + \gamma \int_{\mathbb{S}} \mathcal{P}(s,a,s') \sup_a Q_*(s',a') ds' \end{aligned} \tag{2.17}$$

Again, these two equations are fixed-point equations and the existence and uniqueness of a solution is guaranteed, under some technical assumptions, by the contraction mapping theorem. Starting from the optimal value functions, we can easily derived an optimal policy. Let us define a partial ordering in the policy space

$$\pi \succeq \pi' \Leftrightarrow V_\pi(s) \geq V_{\pi'}(s) \quad \forall s \in \mathbb{S} \tag{2.18}$$

Then the optimal policy $\pi_* \succeq \pi$, $\forall \pi$. We have the following result

Theorem 2.4.1 (Optimal Policy)**.** *For any Markov decision process,*

i) It exists an optimal policy π_* such that $\pi_* \succeq \pi$, $\forall \pi$.
ii) $V_{\pi_*}(s) = V_*(s)$.
iii) $Q_{\pi_*}(s,a) = Q_*(s,a)$.

An optimal policy can be found by acting greedily with respect to Q_*, i.e. in each state s the agent selects the action that maximizes the action-value function

$$a = \arg\sup_a Q_*(s,a) \tag{2.19}$$

We see that this policy is deterministic and only depends on the current state of the system. The Bellman equations provide the basis for the *value iteration* and *policy iteration* algorithms [68], which however require knowledge of the Markov transition kernel.

2.5 Average Reward Formulation

Most of the research in RL has studied a problem formulation where agents maximize the cumulative sum of rewards. However, this approach cannot handle infinite horizon tasks, where there are no absorbing goal states, without discounting future rewards. Clearly, discounting is only necessary in cyclical tasks, where the cumulative reward sum can be unbounded. More natural long-term measure of optimality exists for such cyclical tasks, based on maximizing the average reward per action. For a more in-depth presentation, the reader may again refer to the extensive literature on the subject, such as [3], [46] and the references therein. In the average reward setting, also known as long-run reward or ergodic reward, the goal of the agent is to find a policy that maximizes the expected reward per step.

Definition 2.5.1 (Average Reward)**.** *The average reward ρ_π associated to a policy π is defined as*

$$\begin{aligned}\rho_\pi &= \lim_{T\to\infty} \frac{1}{T}\mathbb{E}_\pi \left[\sum_{t=0}^{T-1} R_{t+1}\right] \\ &= \mathbb{E}_{\substack{S\sim d_\pi \\ A\sim\pi}} \left[\mathcal{R}(S,A)\right] \\ &= \int_{\mathbb{S}} d_\pi(s) \int_{\mathbb{A}} \pi(s,a)\mathcal{R}(s,a) da ds\end{aligned} \tag{2.20}$$

where d_π is the stationary distribution of the Markov process induced by π.

The agent aims to find an *average optimal* policy

$$\pi_* = \arg\sup_\pi \rho_\pi \tag{2.21}$$

In this setting, we introduce the *average adjusted* value and action-value functions.

Definition 2.5.2 (Average Adjusted State-Value Function)**.** *The average adjusted state-value function $V_\pi : \mathbb{S} \to \mathbb{R}$ is the expected residual return that can be obtained starting from a state and following policy π*

$$V_\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \left(R_{t+1} - \rho_\pi\right) \middle| S_0 = s\right] \tag{2.22}$$

The term $V_\pi(s)$ is usually referred to as the *bias* value, or the *relative* value, since it represents the relative difference in total reward gained starting from a state s as opposed to a generic state. ρ_π serves as a baseline that allows to avoid divergence in the value function definition.

Definition 2.5.3 (Average Adjusted Action-Value Function)**.** *The average adjusted action-value function $Q_\pi : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ is the expected residual return that can be obtained starting from a state, taking an action and then following policy π*

$$Q_\pi(s,a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} (R_{t+1} - \rho_\pi) \,\middle|\, S_0 = s, A_0 = a \right] \tag{2.23}$$

We have the following relation between the state-value function and the action-value function

$$V_\pi(s) = \int_{\mathbb{A}} \pi(s,a) Q_\pi(s,a) \tag{2.24}$$

The value functions satisfy the following Bellman equation

Proposition 2.5.1 (Bellman Expectation Equations)**.**

$$\rho_\pi + V_\pi(s) = \mathcal{R}_\pi(s) + T_\pi V_\pi(s) \tag{2.25}$$

$$\begin{aligned} \rho_\pi + Q_\pi(s,a) &= \mathcal{R}(s,a) + T_a V_\pi(s) \\ &= \mathcal{R}(s,a) + \int_{\mathbb{S}} \mathcal{P}(s,a,s') \int_{\mathbb{A}} \pi(s',a') Q_\pi(s',a') da' ds' \end{aligned} \tag{2.26}$$

Again, by introducing opportune Bellman operators, these equations can be rewritten as fixed-point equations. In the discrete case, where the transition operator correspond to matrices, these Bellman equations become linear systems that can be solved to obtain the value functions.

2.6 Risk-Sensitive Formulation

In many application, in addition to maximizing the average reward, the agent may want to control risk by minimizing some measure of variability in rewards. In [58], the authors consider the long-run variance of π

Definition 2.6.1 (Long-Run Variance)**.** *The long-run variance Λ_π under policy π is defined as*

$$\Lambda_\pi = \lim_{T \to \infty} \frac{1}{T} \mathbb{E}_\pi \left[\sum_{t=0}^{T-1} (R_{t+1} - \rho_\pi)^2 \right] \tag{2.27}$$

The long-run variance can be decomposed as follows

$$\Lambda_\pi = \eta_\pi - \rho_\pi^2 \tag{2.28}$$

where η_π is the average square reward per step

Definition 2.6.2 (Average Square Reward)**.**

$$\begin{aligned}\eta_\pi &= \lim_{T\to\infty}\frac{1}{T}\mathbb{E}_\pi\left[\sum_{t=0}^{T-1} R_{t+1}^2\right] \\ &= \mathbb{E}_{\substack{S\sim d_\pi \\ A\sim\pi}}\left[\mathcal{M}(S,A)\right] \\ &= \int_{\mathbb{S}} d_\pi(s)\int_{\mathbb{A}}\pi(s,a)\mathcal{M}(s,a)\end{aligned} \tag{2.29}$$

where we denoted by $\mathcal{M}(s,a)$ *the square reward function*

$$\mathcal{M}(s,a) = \mathbb{E}\left[R_{t+1}^2 | S_t = s, A_t = a\right] \tag{2.30}$$

As before, we introduce the residual state-value and action-value functions associated with the square reward under policy π

Definition 2.6.3 (Average Adjusted Square State-Value Function)**.** *The average adjusted square state-value function* $U_\pi : \mathbb{S}\to\mathbb{R}$ *is the expected square residual return that can be obtained starting from a state and following policy* π

$$U_\pi(s) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\left(R_{t+1}^2 - \eta_\pi\right)\middle| S_0 = s\right] \tag{2.31}$$

Definition 2.6.4 (Average Adjusted Square Action-Value Function)**.** *The average adjusted square action-value function* $Q_\pi : \mathbb{S}\times\mathbb{A}\to\mathbb{R}$ *is the expected residual square return that can be obtained starting from a state, taking an action and then following policy* π

$$W_\pi(s,a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\left(R_{t+1}^2 - \eta_\pi\right)\middle| S_0 = s, A_0 = a\right] \tag{2.32}$$

The following relation between square state-value function and the square action-value holds

$$U_\pi(s) = \int_{\mathbb{A}}\pi(s,a)W_\pi(s,a) \tag{2.33}$$

The average adjusted square value functions satisfy the following Bellman equations

Proposition 2.6.1 (Bellman Expectation Equations)**.**

$$\eta_\pi + U_\pi(s) = \mathcal{M}_\pi(s) + T_\pi U_\pi(s) \tag{2.34}$$

$$\begin{aligned}\eta_\pi + W_\pi(s,a) &= \mathcal{M}(s,a) + T_a U_\pi(s) \\ &= \mathcal{M}(s,a) + \int_{\mathbb{S}}\mathcal{P}(s,a,s')\int_{\mathbb{A}}\pi(s',a')W_\pi(s',a')da'ds'\end{aligned} \tag{2.35}$$

In the risk-sensitive setting, the agent wants to find a policy that solves the following mean-variance optimization problem

$$\begin{cases} \max_\pi \rho_\pi \\ \text{subject to } \Lambda_\pi \leq \alpha \end{cases} \tag{2.36}$$

for a given $\alpha > 0$. Using the Lagrangian relaxation procedure, we can recast (2.36) to the following uncostrained problem

$$\max_\lambda \min_\pi L(\pi, \lambda) = -\rho_\pi + \lambda(\Lambda_\theta - \alpha) \tag{2.37}$$

Alternatively, the agent may want to optimize the Sharpe ratio, a risk-sensitive performance measure commonly used in finance

$$\mathbf{Sh}_\pi = \frac{\rho_\pi}{\sqrt{\Lambda_\pi}} \tag{2.38}$$

The algorithms that will be discussed in the next chapters apply equally to both optimization problems with only minor modifications.

2.7 Policy Evaluation and Policy Iteration

TODO

Chapter 3

Reinforcement Learning

The standard way to solve Markov decision processes is through dynamic programming, which simply consists in solving the Bellman fixed-point equations discussed in the previous chapter. Following this approach, the problem of finding the optimal policy is transformed into the problem of finding the optimal value function. However, apart from the simplest cases where the MDP has a limited number of states and actions, dynamic programming becomes computationally infeasible. Moreover, this approach requires complete knowledge of the Markov transition kernel and of the reward function, which in many real-world applications might be unknown or too complex to use. Reinforcement Learning (RL) is a subfield of Machine Learning which aims to turn the infeasible dynamic programming methods into practical algorithms that can be applied to large-scale problems. RL algorithms are based on two key ideas: the first is to use samples to compactly represent the unknown dynamics of the controlled system. The second idea is to use powerful function approximation methods to compactly estimate value functions and policies in high-dimensional state and action spaces. In the following sections, we present the reinforcement learning problem in more depth and discuss how it relates to the standard discrete-time stochastic optimal control theory and to the other subfields of machine learning, such as supervised learning. In particula, we will see how RL extends ideas from optimal control theory and stochastic approximation to address the broader goal of artificial intelligence. This quick overview of RL is propaedeutic to the following chapters, where we will present in more detail a particular class of algorithms, called policy gradient methods, which are well suited for continuous action spaces. For a more thorough presentation, the reader may consult [68], [70] or [78].

3.1 The Reinforcement Learning Problem

Reinforcement Learning (RL) is a general class of algorithms in the field of machine learning that allows an agent to learn how to behave in a stochastic and possibly unknown environment, where the only feedback consists of a scalar reward signal. In order to maximize the long-run reward, the agent must learn which actions are the most profitable by trial-and-error. Therefore, RL algorithms can be seen as computational methods to solve Markov decision processes by directly interacting with the environment, for which a model may or may not be available. Trial-and-error search and a delayed reward signal can be seen as the most characteristic features of reinforcement learning.
Compared to supervised learning, one of the main branches of machine learning, the feedback the learner receives is much less. In supervised learning, the agent is provided with examples of the correct or expected behavior by a knowledgeable external supervisor and his goal is to learn how to replicate these examples as well as possible and possibly generalize this knowledge to new examples. In reinforcement learning, the agent only receives a numerical reward that only gives a partial feedback of the goodness of his actions. Therefore, this feedback system is evaluative rather than instructive and it is much more difficult for the agent to learn how to behave in uncharted territory without any external guidance.
This particular framework generates some challenges that are not present in other kinds of learning. The first one is the trade-off between exploration and exploitation. In order to maximize his reward, an agent would greedily select actions that he has already tried in the past and found to be effective in producing rewards. However, to find these actions, he has to test actions that haven't been selected before in order to evaluate their potential. Clearly, this might result in worse performance in the short-term because the actions might be suboptimal. However, without trying them, the agent might not be able to find possible improvements. Thus, an agent must exploit what he already knows to obtain rewards, but he also needs to explore to select better actions in the future. A second challenge is the credit assignment problem. Since rewards might be delayed in time, it will be difficult for the agent to understand which actions are mostly responsible for the outcome.

3.2 Model-Free RL Methods

In Section 2.7, we discussed the policy iteration method for computing an optimal policy for an MDP in a finite state and action spaces. This algorithm belongs to the class of *model-based* methods, since it requires perfect

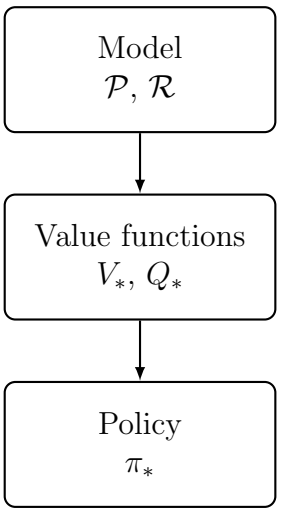

Figure 3.1: Solution process for the control problem.

knowledge of the Markov transition kernel and reward function, i.e. the model of the MDP. RL is primarily concerned with how to obtain an optimal policy when such a model is not available. In this section, we discuss some classes of *model-free* methods which do not rely on the MDP model. The lack of a model generates the need to sample the MDP to gather statistical knowledge about this unknown model. In the control setting, the goal is to approximate the optimal policy, which depends on the optimal value, which in turn depends on the model of the MDP, as shown in Figure 3.1. Indeed, we have already seen that a policy which is greedy with respect to the optimal action-value funtion Q_*, i.e. such that

$$\int_{\mathbb{A}} \pi_*(s,a) Q_*(s,a) da = \sup_a Q_*(s,a) \tag{3.1}$$

is optimal. Therefore, we can derive the following three methodologies that differ in which part of the solution process is approximated

i) *Model-approximation* algorithms approximate the MDP model and compute an estimate of the optimal policy by dynamic programming.
ii) *Value-approximation* algorithms use samples to directly approximate V_* or Q_*, from which an estimate of π_* can be derived by acting greedily.
iii) *Policy-approximation* algorithms directly try to estimate the optimal policy.

It should be noticed that these approaches are not mutually exclusive and can be combined to derive hybrid algorithms. In the following sections we discuss these three classes of algorithm in more detail, following closely [78].

3.2.1 Model Approximation

Model-approximation algorithms approximate the MDP model and compute an optimal policy by dynamic programming, using the techniques discussed in the previous chapter. Since $\mathbb{S}$, $\mathbb{A}$ and γ are assumed to be known, these methods are based on learning an approximation of the Markov transition kernel $\mathcal{P}$ and the reward function $\mathcal{R}$. Thanks to the Markov property, these quantities only depend on the current state and action, so that their approximation corresponds to a density estimation problem and a regression problem respectively, which are fairly standard supervised learning problems. Learning the model may not be trivial, but it is in general easier than learning the value of a policy or optimizing the policy directly. The major drawback of model-based algorithms in continuous-state MDPs is that, even if the model is available, it is in general infeasible to compute the value functions by dynamic programming and to extract an optimal policy for all states by acting greedily. Alternatively, a transition model estimate may be used to generate sample trajectories from the MDP, which can then be used to estimate the value function or directly improve the policy. However, the value function and the policy estimated using these samples can only be as accurate as the learned model, so that in many cases it may be easier to directly approximate the value function or the policy using the methods described below.

3.2.2 Value Approximation

Value-approximation algorithms use samples from the MDP to approximate V_* or Q_* directly and then derive an estimate of the optimal policy by acting greedily with respect to Q_*. Typically, when the state and action spaces are large, the value functions are estimated using a parametric function approximator, whose parameters are iteratively updated given the observed samples. Many reinforcement learning algorithms fall into this category and they can be distinguishes based on whether they are on-policy or off-policy and whether they update the value function estimates online or offline. *On-policy* algorithms approximate the state-value function V_π or the action-value function Q_π from samples of the MDP obtained by following the same policy π to be evaluated. Although the optimal policy π_* is initially unknown, such algorithms can eventually approximate the optimal value functions V_* or Q_* from which an approximation of the optimal policy can be derived. On the other hand, *off-policy* algorithms can learn the value of a policy different from the one use for obtaining the MDP samples. *Online* algorithms adapt their value approximation after each observed sample while *Offline* algorithms operate on batches of samples. Online algorithms typically re-

quire less computation per sample but their convergence is slower. Online on-policy algorithms include temporal-difference (TD) algorithms, such as TD-learning, Sarsa. Offline on-policy algorithms include least-squares approaches, such least-squares temporal difference (LSTD), least-squares policy evaluation (LSPE) and least-squares policy iteration (LSPI). The most known model-free online off-policy algorithm is Q-learning.

3.2.3 Policy Approximation

Policy-approximation algorithms only store a policy and update this policy to maximize a given performance measure and eventually converge to the optimal policy. Since these algorithms only use a policy estimate and don't rely on a value function approximation, they are also referred to as *direct policy-search* or *actor-only* algorithms. Algorithms that store both a policy and a value function are commonly known as *actor-critic* algorithms [69], [41]. Policy gradient algorithms, the most studied policy-approximation methods, will be discussed in much more detail in the next chapter.

Chapter 4

Policy Gradient Methods

4.1 Policy Gradient

In policy gradient methods, we directly store and iteratively improve an approximation of the optimal policy. More formally, we consider a parametrized policy $\pi : \mathbb{S} \times \mathcal{A} \times \Theta \to \mathbb{R}$ such that, for every $s \in \mathbb{S}$, $B \in \mathcal{A}$ and a given policy parameter vector $\theta \in \Theta \subseteq \mathbb{R}^{D_\theta}$, $\pi(s, B; \theta) = \pi_\theta(s, B)$ gives the probability of selecting an action in B when the system is in state s. This policy is also called an *actor* and methods that directly approximate the policy without exploiting an approximation of the optimal value function are called *actor-only*. We will also see how to combine an approximation of the optimal policy with an approximation of the value function, in what are commonly called *actor-critc* methods. As we have seen in the previous sections, an optimal policy can be derived by simply acting greedily with respect to the optimal action-value function. However, in large or continuous action spaces, this leads to a complex optimization problem that is computationally expensive to solve. Therefore, it can be beneficial to store an explicit estimation of the optimal policy from which we can select actions. Policy gradient methods have other advantages compared to standard value-based approaches

i) these methods have better convergence properties and are guaranteed to converge at least to a local optimum, which may be good enough in practice.
ii) they are effective in high-dimensional or continuous action spaces.
iii) they can learn stochastic policies and not only deterministic ones.
iv) In many applications, the optimal policy has a more compact representation that the value function, so that it might be easier to approximate.

On the other hand, policy gradient methods have a large variance which may hinder the converge speed. We will see in the following sections how different methods address this problem.

4.1.1 Basics of Policy Gradient

The general goal of policy optimization in reinforcement learning is to optimize the policy parameters $\theta \in \Theta$ so as to maximize a certain objective function $J : \Theta \to \mathbb{R}$

$$\max_{\theta \in \Theta} J(\theta) \tag{4.1}$$

There are various choices for the objective function.

Definition 4.1.1 (Start Value)**.** *In an episodic environment, the start value is the expected return that can be obtained starting from the start state $s^* \in \mathbb{S}$ and following policy π_θ*

$$J_{start}(\theta) = V_{\pi_\theta}(s^*) = \mathbb{E}_{\pi_\theta}\left[G_t | S_t = s^*\right] \tag{4.2}$$

Definition 4.1.2 (Average Value)**.** *In a continuing environment, the average value is the expected value that can be obtained following policy π_θ*

$$J_{avV}(\theta) = \mathbb{E}_{S\sim\mu}\left[V_{\pi_\theta}(S)\right] = \int_{\mathbb{S}} V_{\pi_\theta}(s)\mu(s)ds \tag{4.3}$$

where μ is a probability distribution over $(\mathbb{S}, \mathcal{S})$.

Definition 4.1.3 (Average Reward per Time Step)**.** *The average reward per time step is the expected reward that can be obtained over a single time step by following policy π_θ*

$$J_{avR}(\theta) = \mathbb{E}_{\substack{S\sim\mu \\ A\sim\pi_\theta}}\left[\mathcal{R}(S, A)\right] = \int_{\mathbb{S}} \mu(s) \int_{\mathbb{A}} \pi_\theta(s, a)\mathcal{R}(s, a)dads \tag{4.4}$$

where μ is a probability distribution over $(\mathbb{S}, \mathcal{S})$.

Fortunately, the same methods apply to the three formulations. In the following, we will focus on gradient-based and model-free methods that exploit the sequential structure of the the reinforcement learning problem. The idea of policy gradient algorithms is to update the policy parameters using the gradient ascent direction of the objective function

$$\theta_{k+1} = \theta_k + \alpha_k \nabla_\theta J|_{\theta=\theta_k} \tag{4.5}$$

where $\{\alpha_k\}_{k\geq 0}$ is a sequence of learning rates. Typically, the gradient of the objective function is not know and needs to be estimated. It is a well-know result from stochastic optimization that, if the gradient estimate is unbiased and the learning rates satisfy the *Robbins-Monro conditions*

$$\sum_{k=0}^{\infty} \alpha_k = \infty \qquad \sum_{k=0}^{\infty} \alpha_k^2 < \infty \tag{4.6}$$

the learning process is guaranteed to converge at least to a local optimum of the objective function. In the following sections, we describe various methods of approximating the gradient.

4.1.2 Finite Differences

4.1.3 Monte Carlo Policy Gradient

Let $h = \{(s_t, a_t)\}_{t\geq 0} \in \mathbb{H}$ be a given trajectory and let us denote by $p_\theta(h) = \mathbb{P}_{\pi_\theta}(H = h)$ the probability of obtaining this trajectory by following policy π_θ. Let $G(h)$ denote the expect return obtained on trajectory h

$$G(h) = \mathbb{E}\left[G_0 | H_t = h\right] = \sum_{t=1}^{\infty} \gamma^k \mathcal{R}(s_{t-1}, a_{t-1}) \tag{4.7}$$

For simplicity, let us consider the average value objective function which can be rewritten as an expectation over all possible trajectories

$$J_{\text{avV}}(\theta) = \int_{\mathbb{H}} p_\theta(h) G(h) dh \tag{4.8}$$

We can compute its gradient using the likelihood ratio trick

$$\begin{aligned} \nabla_\theta J(\theta) &= \int_{\mathbb{H}} \nabla_\theta p_\theta(h) G(h) dh \\ &= \int_{\mathbb{H}} p_\theta(h) \nabla_\theta \log p_\theta(h) G(h) dh \\ &= \mathbb{E}\left[\nabla_\theta \log p_\theta(H) G(H)\right] \end{aligned} \tag{4.9}$$

where the expectation is taken over all possible trajectories. The crucial point is that $\nabla_\theta \log p_\theta(H)$ can be computed without knowledge of the transition probability kernel $\mathcal{P}$. Indeed

$$p_\theta(h) = \mathbb{P}\left(S_0 = s_0\right) \prod_{t=0}^{\infty} \pi_\theta(s_t, a_t) \mathcal{P}(s_t, a_t, s_{t+1})$$

$$\log p_\theta(h) = \log \mathbb{P}\left(() S_0 = s_0\right) + \sum_{t=0}^{\infty} \log \pi_\theta(s_t, a_t) + \sum_{t=0}^{\infty} \log \mathcal{P}(s_t, a_t, s_{t+1})$$

The only term depending on the parameters θ is the policy term, so that

$$\nabla_\theta \log p_\theta(H) = \sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(s_t, a_t) \tag{4.10}$$

So we do not need the transition model to compute the $\nabla_\theta \log p_\theta(H)$. However, this trick only works if the policy is stochastic. In most cases this is not a big problem, since stochastic policies are needed anyway to ensure sufficient exploration. There are two important classes of stochastic policies that

work well in this framework: softmax policies and Gaussian policies [TODO]. Moreover, since

$$\int_{\mathbb{H}} \nabla_\theta p_\theta(h) dh = 0$$

a constant baseline $b \in \mathbb{R}$ can always be added in the gradient formula

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\nabla_\theta \log p_\theta(H)(G(H) - b)\right] \tag{4.11}$$

We will see how this baseline can be chosen in order to minimize the variance of the estimator. In an episodic environment, we can derive an estimate of the objective function gradient by sampling M trajectories $h^{(m)} = \{(s_t^{(m)}, a_t^{(m)})\}_{t=0}^{T^{(m)}}$ from the MDP and by approximating the expected value via Monte Carlo

$$g_{\text{RF}} = \frac{1}{M} \sum_{m=1}^{M} \left[\sum_{i=0}^{T^{(m)}} \nabla_\theta \log \pi_\theta(s_i^{(m)}, a_i^{(m)})\right] \left[\sum_{j=0}^{T^{(m)}} \gamma^j r_{j+1}^{(m)} - b\right] \tag{4.12}$$

This method is known in the literature as the REINFORCE algorithm and is guaranteed to converge to the true gradient at a pace of $O(M^{-1/2})$. In practice, we can obtain an approximation of the gradient using only one sample which leads to a stochastic gradient ascent method

$$g_{\text{SRF}} = \left[\sum_{i=0}^{T} \nabla_\theta \log \pi_\theta(s_i, a_i)\right] \left[\sum_{j=0}^{T} \gamma^j r_{j+1} - b\right] \tag{4.13}$$

This method is very easy and works well on many problems. However, the gradient estimate is characterized by a large variance which can hamper the convergence rate of the algorithm. A first approach to adress this issue is to optimally set the benchmark to reduce the estimate variance. [TODO]

4.1.4 Policy Gradient Theorem

In the last section, we said that the REINFORCE gradient estimate is characterized by a large variance, which may slow the method's convergence. To improve the estimate, it is sufficient to notice that future actions do not depend on past rewards, unless the policy has been changed. Therefore,

$$\mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(S_t, A_t)\mathcal{R}(S_s, A_s)\right] = 0 \quad \forall t > s \tag{4.14}$$

From this trivial remark, we can derive two estimates of the objective function gradient

$$g_{\text{PG}} = \frac{1}{M} \sum_{m=1}^{M} \sum_{i=0}^{T^{(m)}} \nabla_\theta \log \pi_\theta(s_i^{(m)}, a_i^{(m)}) \left(\sum_{j=i}^{T^{(m)}} \gamma^j r_{j+1}^{(m)} - b\right) \tag{4.15}$$

$$g_{\text{GMDP}} = \frac{1}{M} \sum_{m=1}^{M} \sum_{j=0}^{T^{(m)}} \left[\sum_{i=j}^{T^{(m)}} \nabla_\theta \log \pi_\theta(s_i^{(m)}, a_i^{(m)}) \right] \left(\gamma^j r_{j+1}^{(m)} - b \right) \tag{4.16}$$

The two estimates are exactly equivalent. This simple trick greatly reduces the estimate variance and this can speed up convergence. These algorithms can all be derived from a more general result: the policy gradient theorem.

Theorem 4.1.1 (Policy Gradient)**.** *For any differentiable policy π_θ, for any of the policy objective functions $J = J_{start}$, J_{avV}, J_{avR}, $\frac{1}{1-\gamma} J_{avV}$, the policy gradient is*

$$\nabla_\theta J(\theta) = \mathbb{E}_{\substack{S \sim \mu \\ A \sim \pi_\theta}} \left[\nabla_\theta \log \pi_\theta(S, A) Q_{\pi_\theta}(S, A) \right] \tag{4.17}$$

The problem is that the action-value function is typically unknown and needs to be approximated. For instance, REINFORCE replaces it with the realized return achieved on a sample trajectory, that are an unbiased estimate of the action-value function. However, the theorem can be used as the starting point to derive many other policy gradient methods that use different approximation of the action-value function.

4.1.4.1 Actor-Critic Policy Gradient

Actor-Critic Policy Gradient employs a *critic*, that is a parametric approximation $\widehat{Q} : \mathbb{S} \times \mathbb{A} \times \Psi \to \mathbb{R}$, where $\Psi \in \mathbb{R}^{D_\psi}$ such that $\widehat{Q}_\psi(s, a) = \widehat{Q}(s, a; \psi) \approx Q_{\pi_\theta}(s, a)$. Therefore these methods maintain two sets of parameters: a *critic* that updates the action-value function parameters ψ and an *actor* that updates the policy parameters θ in the direction suggested by the critic. More formally, given $\widehat{Q}_\psi$, the current state of the system s_t and the action a_t selected using policy π_θ, the policy parameters are updated in the approximated gradient direction

$$\theta_{k+1} = \theta_k + \alpha_k \nabla_\theta \pi_\theta(s_t, a_t) \widehat{Q}_\psi(s_t, a_t) \tag{4.18}$$

On the other hand, the action-value function parameters ψ can be updated using any value-based approach, for instance TD(0). This leads to the QAC algorithm, for which the pseudo code is reported in **??**.

[TODO: TD policy gradient, compatible function approximation, advantage function and actor-critic policy gradient]

4.1.5 Natural Policy Gradient

4.1.6 Policy Gradient with Parameter Exploration

In Monte Carlo Policy Gradient, trajectories are generated by sampling at each time step an action according to a stochastic policy π_θ and the objec-

tive function gradient is estimated by differentiating the policy with respect to the parameters. However, sampling an action from the policy at each time step leads to a large variance in the sampled histories and therefore in the gradient estimate, which can in turn slow down the convergence of the learning process. To address this issue, in [65] the authors propose the *policy gradient with parameter-based exploration* (PGPE) method, in which the search in the policy space is replaced with a direct search in the model parameter space. We start by presenting the episodic case and we will later generalize this approach to the infinite horizon setting.

4.1.6.1 Episodic PGPE

Given an episodic MDP, PGPE considers a determinstic policy $F : \mathbb{S}\times\Theta \to \mathbb{A}$ that, given a set of parameters $\theta \in \Theta \subseteq \mathbb{R}^{D_\theta}$, maps a state $s \in \mathbb{S}$ to an action $a = F(s;\theta) = F_\theta(s) \in \mathbb{A}$. The policy parameters θ are random variables distributed according to a probability distribution parametrized by some hyperparameters $\xi \in \Xi \subseteq \mathbb{R}^{D_\xi}$, i.e. $\theta \sim p(\theta;\xi) = p_\xi(\theta)$. Combining these two hypotheses, we obtain a stochastic policy parametrized by the hyperparameters ξ

$$\pi_\xi(s,a) = \pi(s,a;\xi) = \int_\Theta p_\xi(\theta)\mathbb{1}_{F_\theta(s)=a}d\theta \tag{4.19}$$

The advantage of this approach is that the policy is deterministic and therefore the actions do not need to be sampled at each time step, with a consequent reduction of the gradient estimate variance. Indeed, It is sufficient to sample the parameters θ once at the beginning of the episode and then generate an entire trajectory following the deterministic policy F_θ. As an additional benefit, the parameter gradient is estimated by direct parameter perturbations, without having to backpropagate any derivatives, which allows to use non-differentiable controllers. The hyperparameters ξ will be updated following the gradient of the expected reward, which can be rewritten as

$$J(\xi) = \int_\Theta \int_\mathbb{H} p_\xi(\theta,h)G(h)dhd\theta \tag{4.20}$$

Hence, using the fact that h is conditionally independent from ξ given θ, so that $p_\xi(\theta,h) = p_\xi(\theta)p_\theta(h)$, and the likelihood trick

$$\begin{aligned}\nabla_\xi J(\xi) &= \int_\Theta \int_\mathbb{H} \nabla_\xi p_\xi(\theta)p_\theta(h)G(h)dhd\theta \\ &= \int_\Theta \int_\mathbb{H} p_\xi(\theta)p_\theta(h)\nabla_\xi \log p_\xi(\theta)G(h)dhd\theta \\ &= \mathbb{E}\left[\nabla_\xi \log p_\xi(\theta)G(H)\right]\end{aligned} \tag{4.21}$$

Again, we can subtract a constant baseline $b \in \mathbb{R}$ from the total return

$$\nabla_\xi J(\xi) = \mathbb{E}\left[\nabla_\xi \log p_\xi(\theta)\,(G(H) - b)\right] \tag{4.22}$$

the hyperparameters can then be updated by gradient ascent

$$\xi_{k+1} = \xi_k + \alpha_k \nabla_\xi J(\xi) \tag{4.23}$$

The gradient can be approximated via Monte Carlo by sampling $\theta \sim p_\xi$ and then generating M trajectories $h^{(m)} = \{(s_t^{(m)}, a_t^{(m)})\}_{t\geq 0}$ following the deterministic policy F_θ

$$g_{\text{PGPE}} = \frac{1}{M}\sum_{m=1}^{M} \nabla_\xi \log p_\xi(\theta)\left[G\left(h^{(m)}\right) - b\right] \tag{4.24}$$

Alternatively, we can use a fully stochastic approximation by sampling a single trajectory and approximating the gradient as

$$g_{\text{PGPE}} = \nabla_\xi \log p_\xi(\theta)\,[G(h) - b] \tag{4.25}$$

In the following paragraphs, we discuss some possible choices for the policy parameters distribution p_ξ, which is the last component of the algorithm.

Gaussian Parameter Distribution A simple approach is to assume that all the components of the parameter vector θ are independent and normally distributed with mean μ_i and variance σ_i^2, i.e. $\theta_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$, the gradient with respect to the hyperparameters $\xi = (\mu_1, \ldots, \mu_{D_\theta}, \sigma_1, \ldots, \sigma_{D_\theta})^T$ is given by

$$\begin{aligned} \frac{\partial \log p_\xi(\theta)}{\partial \mu_i} &= \frac{\theta_i - \mu_i}{\sigma_i^2} \\ \frac{\partial \log p_\xi(\theta)}{\partial \sigma_i} &= \frac{(\theta_i - \mu_i)^2 - \sigma_i^2}{\sigma_i^3} \end{aligned} \tag{4.26}$$

Using a constant learning rate $\alpha_i = \alpha\sigma_i^2$, the gradient updates takes the following form

$$\begin{aligned} \mu_i^{k+1} &= \mu_i^k + \alpha\,[G(h) - b]\,(\theta_i - \mu_i) \\ \sigma_i^{k+1} &= \sigma_i^k + \alpha\,[G(h) - b]\,\frac{(\theta_i - \mu_i)^2}{\sigma_i} \end{aligned} \tag{4.27}$$

where b can be computed as a moving average of the past returns. Intuitively, if $G(h) > b$ we adjust ξ so as to increase the probability of θ while if $G(h) < b$ we do the opposite.

Symmetric Sampling and Gain Normalization In some settings, comparing the gain with a baseline can be misleading. In their original work, the authors propose a symmetric sampling technique similar to antithetic variates that further improves the convergence of the method. More in detail, a more robust gradient estimate can be obtained by measuring the difference in reward between two symmetric samples on either side of the current mean. That is, we sample a random perturbation $\epsilon \sim \mathcal{N}(0, \Sigma)$, where $\Sigma = \text{diag}(\sigma_1^2, \dots, \sigma_{D_\theta}^2)$, and we define $\theta^+ = \mu + \epsilon$ and $\theta^- = \mu - \epsilon$. Denoting by G^+ (resp. G^-) the gains obtained on the trajectory associated to θ^+ (resp. θ^-), the objective function gradient can be approximated with

$$\begin{aligned} \nabla_{\mu_i} J(\xi) &\approx \frac{\epsilon_i (G^+ - G^-)}{2\sigma_i^2} \\ \nabla_{\sigma_i} J(\xi) &\approx \frac{\epsilon_i^2 - \sigma_i^2}{\sigma_i^3} \left(\frac{G^+ + G^-}{2} - b \right) \end{aligned} \tag{4.28}$$

Hence, by choosing $\alpha_i^k = 2\alpha\sigma_i^2$, we have the following update rules

$$\begin{aligned} \mu_i^{k+1} &= \mu_i^k + \alpha \epsilon_i (G^+ - G^-) \\ \sigma_i^{k+1} &= \sigma_i^k + \alpha \frac{\epsilon_i^2 - \sigma_i^2}{\sigma_i} \left(G^+ + G^- - 2b \right) \end{aligned} \tag{4.29}$$

In addition, the authors propose to normalize the gains in order to make the updates independent of the scale of the rewards. For instance, we could modify the hyperparameters updates as follows

$$\begin{aligned} \mu_i^{k+1} &= \mu_i^k + \alpha \epsilon_i \frac{(G^+ - G^-)}{2m - G^+ - G^-} \\ \sigma_i^{k+1} &= \sigma_i^k + \alpha \frac{\epsilon_i^2 - \sigma_i^2}{\sigma_i} \frac{(G^+ + G^- - 2b)}{m - b} \end{aligned} \tag{4.30}$$

where m might be the maximum gain the agent can receive, if known, or alternatively the maximum gain achieved so far. Symmetric sampling and gain normalization can drastically improve the gradient estimate quality and consequently the convergence time.

4.1.6.2 Infinite Horizon PGPE

In this section we discuss the extension of the PGPE method to the infinite horizon case [64]. While in the episodic PGPE the parameters θ are sampled only at the beginning of each episode, in *Infinite Horizon PGPE* (IHPGPE) the parameters and learning are carried out simultaneously, while interacting

with the environmnent. Let $0 < \varepsilon < 1$ the probability of updating the policy parameters, the parameters θ_t can be sampled consecutively as follows

$$p_\xi(\theta_{i,t+1}) = \varepsilon \mathcal{N}(\mu_{i,t}, \sigma_{i,t}^2) + (1 - \varepsilon)\delta_{\theta_{i,t}} \tag{4.31}$$

In practice, ε should be chosen so that the expected frequency of changing a single parameter is coherent with the typical episode length in the episodic framework. Alternatively, one could sample all the parameters at a certain time step simultaneously

$$p_\xi(\theta_{t+1}) = \varepsilon \mathcal{N}(\mu_t, \Sigma_t) + (1 - \varepsilon)\delta_{\theta_t} \tag{4.32}$$

This is equivalent to splitting the state-action space into artificial episodes. However, updating parameters asynchronously changes the policy only slightly thus introducing less noise in the process. Again, parameters can be updated at every time step by gradient ascent

$$\begin{aligned} \mu_{i,t+1} &= \mu_{i,t} + \alpha \left[G_t(h) - b \right] (\theta_{i,t} - \mu_{i,t}) \\ \sigma_{i,t+1} &= \sigma_{i,t} + \alpha \left[G_t(h) - b \right] \frac{(\theta_{i,t} - \mu_{i,t})^2}{\sigma_{i,t}} \end{aligned} \tag{4.33}$$

Similarly to the episodic case, we can improve the gradient estimate by symmetric sampling and gain normalization.

Chapter 5

Asset Allocation

The asset allocation problem consists of determining how to dynamically invest the available capital in a portfolio of different assets in order to maximize the expected total return or another relevant performance measure. Let us consider a financial market consisting of $I+1$ different stocks that are traded only at discrete times $t \in \{0, 1, 2, \ldots\}$ and denote by $Z_t = (Z_t^0, Z_t^1, \ldots, Z_t^I)^T$ their prices at time t. Typically, Z_t^0 refers to a riskless asset whose dynamic is given by $Z_t^0 = (1+X)^t$ where X is the deterministic risk-free interest rate. The investment process works as follows: at time t, the investor observes the state of the market S_t, consisting for example of the past asset prices and other relevant economic variables, and subsequently chooses how to rebalance his portfolio, by specifying the units of each stock $n_t = (n_t^0, n_t^1, \ldots, n_t^I)^T$ to be held between t and $t+1$. In doing so, he needs to take into account the transaction costs that he has to pay to the broker to change his position. At time $t+1$, the investor realizes a profit or a loss from his investment due to the stochastic variation of the stock values. The investor's goal is to maximize a given performance measure.

5.1 Reward Function

Let W_t denote the wealth of the investor at time t. The profit realized between t and $t+1$ is simply given by the difference between the trading P&L and the transaction costs payed to the broker. More formally

$$\Delta W_{t+1} = W_{t+1} - W_t = \mathrm{PL}_{t+1} - \mathrm{TC}_t$$

where PL_{t+1} denotes the profit due to the variation of the portfolio asset prices between t and $t+1$

$$\mathrm{PL}_{t+1} = n_t \cdot \Delta Z_{t+1} = \sum_{i=0}^{I} n_t^i (Z_{t+1}^i - Z_t^i)$$

and TC_t denotes the fees payed to the broker to change the portfolio allocation and on the short positions

$$\mathrm{TC}_t = \sum_{i=0}^{I} \delta_p^i \left| n_t^i - n_{t-1}^i \right| Z_t^i - \delta_f W_t \mathbb{1}_{n_t \neq n_{t-1}} - \sum_{i=0}^{I} \delta_s^i (n_t^i)^- Z_t^i$$

The transaction costs consist of three different components. The first term represent a transaction cost that is proportional to the change in value of the position in each asset. The second term is a fixed fraction of the total value of the portfolio which is payed only if the allocation is changed. The last term represents the fees payed to the broker for the shares borrowed to build a short position. The portfolio return between t and $t+1$ is thus given by

$$X_{t+1} = \frac{\Delta W_{t+1}}{W_t} = \sum_{i=0}^{I} \left[a_t^i X_{t+1}^i - \delta_i \left| a_t^i - \tilde{a}_t^i \right| - \delta_s (a_t^i)^- \right] - \delta_f \mathbb{1}_{a_t \neq \tilde{a}_{t-1}} \tag{5.1}$$

where

$$X_{t+1}^i = \frac{\Delta Z_{t+1}^i}{Z_t^i}$$

is the return of the i-th stock between t and $t+1$,

$$a_t^i = \frac{n_t^i Z_t^i}{W_t}$$

is the fraction of wealth invested in the i-th stock between time t and $t+1$ and finally

$$\tilde{a}_t^i = \frac{n_{t-1}^i Z_t^i}{W_t} = \frac{a_{t-1}^i (1 + X_t^i)}{1 + X_t}$$

is the fraction of wealth invested in the i-th stock just before the reallocation. We assume that the agent invests all his wealth at each step, so that W_t can be also interpreted as the value of his portfolio. This assumption leads to the following constraint on the portfolio weights

$$\sum_{i=0}^{I} a_t^i = 1 \qquad \forall t \in \{0, 1, 2, \ldots\} \tag{5.2}$$

We notice that we are neglecting the typical margin requirements on the short positions, which would reduce the available capital at time t. Considering margin requirements would lead to a more complex constraint on the portfolio weights which would be difficult to treat in the reinforcement learning framework. Plugging this constraint into Eq. (5.1), we obtain

$$X_{t+1} = X + \sum_{i=1}^{I} a_t^i (X_{t+1}^i - X) - \sum_{i=0}^{I} \left[\delta_i \left| a_t^i - \tilde{a}_t^i \right| - \delta_s^i (a_t^i)^- \right] - \delta_f \mathbb{1}_{a_t \neq \tilde{a}_{t-1}} \quad (5.3)$$

which highlights the role of the risk-free asset as a benchmark for the portfolio returns. The total profit realized by the investor between $t = 0$ and T is

$$\Pi_T = W_T - W_0 = \sum_{t=1}^{T} \Delta W_t = \sum_{t=1}^{T} W_t X_t$$

The portfolio return between $t = 0$ and T is given by

$$X_{0,T} = \frac{W_T}{W_0} - 1 = \prod_{t=1}^{T} (1 + X_t) - 1$$

In order to cast the asset allocation problem in the reinforcement learning framework, we consider the log-return of the portfolio between $t = 0$ and T

$$R_{0,T} = \log \frac{W_T}{W_0} = \sum_{t=1}^{T} \log(1 + X_t) = \sum_{t=1}^{T} R_t \quad (5.4)$$

where R_{t+1} is the log-return of the portfolio between t and $t + 1$

$$R_{t+1} = \log \left\{ 1 + \sum_{i=0}^{I} \left[a_t^i X_{t+1}^i - \delta_i \left| a_t^i - \tilde{a}_t^i \right| - \delta_s (a_t^i)^- \right] - \delta_f \mathbb{1}_{a_t \neq \tilde{a}_{t-1}} \right\} \quad (5.5)$$

The portfolio return and log-return can be used as the reward function of a RL algorithm, either in a offline or in an online approach.

5.2 States

In this section, we specify the state of the system that the agent observes in order to make his decisions. First, we consider the $P + 1$ past returns of all risky assets, i.e. $\{X_t, X_{t-1}, \ldots, X_{t-P}\}$. These input variables may be used to construct more complex features for example using some Deep Learning techniques, such as a deep auto-encoder. In order to properly incorporate

the effects of transaction costs into his decision process, the agent must keep track of its current position $\tilde{a}_t$. Finally, we might consider some external variables Y_t that may be relevant to the trader. Summing up, we assume that the state of the system is composed in the following way

$$S_t = \{X_t, X_{t-1}, \dots, X_{t-P}, \tilde{a}_t, Y_t, Y_{t-1}, \dots, Y_{t-P}\} \tag{5.6}$$

5.3 Actions

In this section, we specify the actions that the trading system may take. We assume that the agent only specifies the portfolio weights $a_t = (a_t^0, \dots, a_t^I)^T$ and therefore determines the allocation for the time interval $[t, t+1)$. If we assume that the agent invests all of his capital at each time step and that short-selling is not allowed, then the portfolio weights should satisfy, for every $t \in \{0, 1, \dots\}$, the following constraints

$$\begin{cases} a_t^i \geq 0 \ , \ \forall i \in \{0, \dots, I\} \\ \sum_{i=0}^I a_t^i = 1 \end{cases} \tag{5.7}$$

This constraint can be easily enforced by considering a parametric softmax policy, which has been introduced in Section **??**. If short selling is allowed, weights might also be negative. A simple approach is to assume that, for every $t \in \{0, 1, \dots\}$, the weights satisfy

$$\begin{cases} a_t^i \in \mathbb{R} \ , \ \forall i \in \{1, \dots, I\} \\ a_t^0 = 1 - \sum_{i=1}^I a_t^i \end{cases} \tag{5.8}$$

Since a_t^0 is uniquely determined by the other weights, it is enough to define a policy that specifies the allocation in the risky assets, e.g. a Gaussian policy in $\mathbb{R}^I$. The obvious shortcoming of this approach is that the agent might enter in huge short positions, which is not realistic. A first observation is that, if the stochastic policy is concentrated around the origin, huge long or short positions would have very small probabilities of being selected. Moreover, we notice that the agent would pay large fees for entering into large short positions, independently of the trading profits. Therefore, we expect the agent to learn that short positions are very expensive and would therefore try to avoid them.
Working in a constinuous action space is computationally difficult and only few reinforcement learning algorithm are well-suited to this setting, e.g. policy gradient methods. A simpler approach is to reduce the action space to a discrete space. For instance, in the two assets scenario we might assume

that $a_t \in \{-1, 0, +1\}$. Thus the agent may be long $(+1)$, neutral (0) or short (-1) on the risky-asset. Working in a discrete action space is more simple, and standard value-based approaches might be employed.

Bibliography

[1] A. Agarwal, P. Bartlett, and M. Dama, *Optimal allocation strategies for the dark pool problem*, arXiv preprint arXiv:1003.2245, (2010).

[2] R. Almgren and N. Chriss, *Optimal execution of portfolio transactions*, Journal of Risk, 3 (2001), pp. 5–40.

[3] A. Arapostathis, V. S. Borkar, E. Fernández-Gaucherand, M. K. Ghosh, and S. I. Marcus, *Discrete-time controlled markov processes with average cost criterion: a survey*, SIAM Journal on Control and Optimization, 31 (1993), pp. 282–344.

[4] A. Basu, T. Bhattacharyya, and V. S. Borkar, *A learning algorithm for risk-sensitive cost*, Mathematics of Operations Research, 33 (2008), pp. 880–898.

[5] N. Bäuerle and U. Rieder, *Markov decision processes with applications to finance*, Springer Science & Business Media, 2011.

[6] S. D. Bekiros, *Heterogeneous trading strategies with adaptive fuzzy actor–critic reinforcement learning: A behavioral approach*, Journal of Economic Dynamics and Control, 34 (2010), pp. 1153–1170.

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

[8] ——, *Q-learning-based financial trading systems with applications*, University Ca'Foscari of Venice, Dept. of Economics Working Paper Series No, 15 (2014).

[9] ——, *Reinforcement learning for automated financial trading: Basics and applications*, in Recent Advances of Neural Network Models and Applications, Springer, 2014, pp. 197–213.

[10] D. P. Bertsekas, *Dynamic programming and optimal control*, vol. 1, Athena Scientific, Belmont, 1995.

[11] D. P. Bertsekas and S. E. Shreve, *Stochastic optimal control: the discrete-time case*, vol. 23, Academic Press New York, 1978.

[12] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*, Optimization and neural computation series, Athena Scientific, Belmont, 1 ed., 1996.

[13] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee, *Natural actor–critic algorithms*, Automatica, 45 (2009), pp. 2471–2482.

[14] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[15] V. S. Borkar, *A sensitivity formula for risk-sensitive cost and the actor–critic algorithm*, Systems & Control Letters, 44 (2001), pp. 339–346.

[16] ——, *Q-learning for risk-sensitive control*, Mathematics of operations research, 27 (2002), pp. 294–311.

[17] V. S. Borkar and S. P. Meyn, *Risk-sensitive optimal control for markov decision processes with monotone cost*, Mathematics of Operations Research, 27 (2002), pp. 192–209.

[18] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst, *Reinforcement learning and dynamic programming using function approximators*, vol. 39, CRC press, 2010.

[19] P. X. Casqueiro and A. J. Rodrigues, *Neuro-dynamic trading methods*, European Journal of Operational Research, 175 (2006), pp. 1400–1412.

[20] N. Chapados and Y. Bengio, *Cost functions and model combination for var-based asset allocation using neural networks*, IEEE Transactions on Neural Networks, 12 (2001), pp. 890–906.

[21] M. Choey and A. S. Weigend, *Nonlinear trading models through sharpe ratio maximization*, International Journal of Neural Systems, 8 (1997), pp. 417–431.

[22] Y. Chow, A. Tamar, S. Mannor, and M. Pavone, *Risk-sensitive and robust decision-making: a cvar optimization approach*, in Advances in Neural Information Processing Systems, 2015, pp. 1522–1530.

[23] M. Corazza and A. Sangalli, *Q-learning vs. sarsa: comparing two intelligent stochastic control approaches for financial trading.*

[24] J. Cumming, D. Alrajeh, and L. Dickens, *An investigation into the use of reinforcement learning techniques within the algorithmic trading domain*, (2015).

[25] M. A. Dempster and V. Leemans, *An automated fx trading system using adaptive reinforcement learning*, Expert Systems with Applications, 30 (2006), pp. 543–552.

[26] M. A. H. Dempster and Y. S. Romahi, *Intraday FX trading: An evolutionary reinforcement learning approach*, Springer, 2002.

[27] 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, (2016).

[28] Y. Deng, Y. Kong, F. Bao, and Q. Dai, *Sparse coding inspired optimal trading system for hft industry*, IEEE Transactions on Industrial Informatics, 11 (2015), pp. 467–475.

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

[30] T. Elder, *Creating algorithmic traders with hierarchical reinforcement learning*, (2008).

[31] L. A. Feldkamp, D. V. Prokhorov, C. F. Eagen, and F. Yuan, *Enhanced multi-stream kalman filter training for recurrent networks*, in Nonlinear Modeling, Springer, 1998, pp. 29–53.

[32] K. Ganchev, Y. Nevmyvaka, M. Kearns, and J. W. Vaughan, *Censored exploration and the dark pool problem*, Communications of the ACM, 53 (2010), pp. 99–107.

[33] C. Gold, *Fx trading via recurrent reinforcement learning*, in IEEE 2003 IEEE International Conference on Computational Intelligence for Financial Engineering. Proceedings, 2003.

[34] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning.* Book in preparation for MIT Press, 2016.

[35] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*, Springer, 2009.

[36] D. Hendricks and D. Wilcox, *A reinforcement learning extension to the almgren-chriss framework for optimal trade execution*, in IEEE Conference on Computational Intelligence for Financial Engineering & Economics (CIFEr), 2014, pp. 457–464.

[37] H. Jaeger, *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the echo state network approach*, GMD-Forschungszentrum Informationstechnik, 2002.

[38] M. S. Joshi, *C++ design patterns and derivatives pricing*, vol. 2, Cambridge University Press, 2008.

[39] K. Kamijo and T. Tanigawa, *Stock price pattern recognition-a recurrent neural network approach*, in IEEE 1990 IJCNN International Joint Conference on Neural Networks, 1990.

[40] M. Kearns and Y. Nevmyvaka, *Machine learning for market microstructure and high frequency trading*, High-Frequency Trading–New Realities for Traders, Markets and Regulators, (2013), pp. 91–124.

[41] V. R. Konda and J. N. Tsitsiklis, *Actor-critic algorithms.*, in NIPS, vol. 13, 1999, pp. 1008–1014.

[42] P. L.A. and M. Ghavamzadeh, *Actor-critic algorithms for risk-sensitive mdps*, in Advances in Neural Information Processing Systems 26, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, eds., Curran Associates, Inc., 2013, pp. 252–260.

[43] S. Laruelle, C.-A. Lehalle, and G. Pages, *Optimal split of orders across liquidity pools: a stochastic algorithm approach*, SIAM Journal on Financial Mathematics, 2 (2011), pp. 1042–1076.

[44] ——, *Optimal posting price of limit orders: learning by trading*, Mathematics and Financial Economics, 7 (2013), pp. 359–403.

[45] H. Li, C. H. Dagli, and D. Enke, *Short-term stock market timing prediction under reinforcement learning schemes*, in 2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning, 2007, pp. 233–240.

[46] S. Mahadevan, *Average reward reinforcement learning: Foundations, algorithms, and empirical results*, Machine learning, 22 (1996), pp. 159–195.

[47] A. Miyamae, Y. Nagata, I. Ono, and S. Kobayashi, *Natural policy gradient methods with parameter-based exploration for control tasks*, in Advances in neural information processing systems, 2010, pp. 1660–1668.

[48] J. Moody and M. Saffell, *Learning to trade via direct reinforcement*, Neural Networks, IEEE Transactions on, 12 (2001), pp. 875–889.

[49] J. Moody, M. Saffell, Y. Liao, and L. Wu, *Reinforcement learning for trading systems*, in Decision Technologies for Computational Finance: Proceedings of the fifth International Conference Computational Finance, vol. 2, Springer Science & Business Media, 2013, p. 129.

[50] J. Moody and L. Wu, *Optimization of trading systems and portfolios*, in Proceedings of the IEEE/IAFE 1997 Computational Intelligence for Financial Engineering (CIFEr), 1997, pp. 300–307.

[51] J. Moody, L. Wu, Y. Liao, and M. Saffell, *Performance functions and reinforcement learning for trading systems and portfolios*, Journal of Forecasting, 17 (1998), pp. 441–470.

[52] Y. Nevmyvaka, Y. Feng, and M. Kearns, *Reinforcement learning for optimized trade execution*, in Proceedings of the 23rd international conference on Machine learning, ACM, 2006, pp. 673–680.

[53] J. Nocedal and S. Wright, *Numerical optimization*, Springer Science & Business Media, 2006.

[54] J. O, J. Lee, J. W. Lee, and B.-T. Zhang, *Adaptive stock trading with dynamic asset allocation using reinforcement learning*, Information Sciences, 176 (2006), pp. 2121–2147.

[55] J. Peters, K. Mülling, and Y. Altun, *Relative entropy policy search.*, in AAAI, Atlanta, 2010.

[56] J. Peters and S. Schaal, *Policy gradient methods for robotics*, in Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on, IEEE, 2006, pp. 2219–2225.

[57] ——, *Reinforcement learning of motor skills with policy gradients*, Neural networks, 21 (2008), pp. 682–697.

[58] L. Prashanth and M. Ghavamzadeh, *Actor-critic algorithms for risk-sensitive reinforcement learning.*, arXiv preprint arXiv:1403.6530, (2014).

[59] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*, John Wiley & Sons, 1994.

[60] E. W. Saad, D. V. Prokhorov, and D. C. Wunsch, *Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks*, IEEE Transactions on Neural Networks, 9 (1998), pp. 1456–1470.

[61] C. Sanderson, *Armadillo: An open source c++ linear algebra library for fast prototyping and computationally intensive experiments*, (2010).

[62] M. Sato and S. Kobayashi, *Variance-penalized reinforcement learning for risk-averse asset allocation*, in Intelligent Data Engineering and Automated Learning—IDEAL 2000. Data Mining, Financial Engineering, and Intelligent Agents, Springer, 2000, pp. 244–249.

[63] ——, *Average-reward reinforcement learning for variance penalized markov decision problems*, in Proceedings of the Eighteenth International Conference on Machine Learning, ICML '01, San Francisco, CA, USA, 2001, Morgan Kaufmann Publishers Inc., pp. 473–480.

[64] F. Sehnke et al., *Parameter exploring policy gradients and their implications*, PhD thesis, Technische Universität München, 2012.

[65] F. Sehnke, C. Osendorfer, T. Rückstiess, A. Graves, J. Peters, and J. Schmidhuber, *Policy gradients with parameter-based exploration for control*, in Artificial Neural Networks-ICANN 2008, Springer, 2008, pp. 387–396.

[66] ——, *Parameter-exploring policy gradients*, Neural Networks, 23 (2010), pp. 551–559.

[67] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, *Deterministic policy gradient algorithms*, in ICML, 2014.

[68] R. S. Sutton and A. G. Barto, *Introduction to reinforcement learning*, vol. 135, MIT Press Cambridge, 1998.

[69] R. S. Sutton, D. A. McAllester, S. P. Singh, Y. Mansour, et al., *Policy gradient methods for reinforcement learning with function approximation.*, in NIPS, vol. 99, 1999, pp. 1057–1063.

[70] C. Szepesvári, *Algorithms for reinforcement learning*, Synthesis lectures on artificial intelligence and machine learning, 4 (2010), pp. 1–103.

[71] A. Tamar, D. D. Castro, and S. Mannor, *Temporal difference methods for the variance of the reward to go*, in Proceedings of the 30th International Conference on Machine Learning (ICML-13), 2013, pp. 495–503.

[72] A. Tamar, Y. Chow, M. Ghavamzadeh, and S. Mannor, *Policy gradient for coherent risk measures*, in Advances in Neural Information Processing Systems, 2015, pp. 1468–1476.

[73] A. Tamar, D. Di Castro, and S. Mannor, *Policy gradients with variance related risk criteria*, in Proceedings of the 29th International Conference on Machine Learning, 2012, pp. 387–396.

[74] A. Tamar and S. Mannor, *Variance adjusted actor critic algorithms*, arXiv preprint arXiv:1310.3697, (2013).

[75] Z. Tan, C. Quek, and P. Y. Cheng, *Stock trading with cycles: A financial application of anfis and reinforcement learning*, Expert Systems with Applications, 38 (2011), pp. 4741–4755.

[76] R. S. Tsay, *Analysis of financial time series*, vol. 543, John Wiley & Sons, 2005.

[77] P. J. Werbos, *Backpropagation through time: what it does and how to do it*, Proceedings of the IEEE, 78 (1990), pp. 1550–1560.

[78] M. Wiering and M. Van Otterlo, *Reinforcement Learning: State-of-the-Art*, vol. 12 of Adaptation, Learning, and Optimization, Springer, 1 ed., 2012.

[79] R. J. Williams and D. Zipser, *A learning algorithm for continually running fully recurrent neural networks*, Neural computation, 1 (1989), pp. 270–280.

[80] S. Yang, M. Paddrik, R. Hayes, A. Todd, A. Kirilenko, P. Beling, and W. Scherer, *Behavior based learning in identifying high frequency trading strategies*, in IEEE Conference on Computational Intelligence for Financial Engineering & Economics (CIFEr), 2012.

[81] T. Zhao, H. Hachiya, G. Niu, and M. Sugiyama, *Analysis and improvement of policy gradient estimation*, in Advances in Neural Information Processing Systems, 2011, pp. 262–270.

[82] T. Zhao, G. Niu, N. Xie, J. Yang, and M. Sugiyama, *Regularized policy gradients: Direct variance reduction in policy gradient estimation*, in Proceedings of The 7th Asian Conference on Machine Learning, 2015, pp. 333–348.