Chapter 1

Reinforcement Learning

1.1 The Reinforcement Learning Problem

1.2 Markov Decision Processes

The reinforcement learning problem is modeled using Markov decision processes.

Definition 1.2.1 (Markov Decision Process)**.** *A Markov decision process (MDP) is a tuple* $< \mathbb{S}, \mathbb{A}, \mathcal{P}, \mathcal{R}, \gamma >$*, where*

i) $(\mathbb{S}, \mathcal{S})$ *is a measurable state space.*

ii) $(\mathbb{A}, \mathcal{A})$ *is a measurable 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.*

The kernel $\mathcal{P}$ describes the random evolution of the system: suppose that at time t the system is in state s_t and that the agent takes action a_t, 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 $\mathcal{P}(s_t, a_t, B)$, i.e.

$$\mathcal{P}(s_t, a_t, B) = \mathbb{P}\left(S_{t+1} \in B | S_t = s_t, A_t = a_t\right) \tag{1.1}$$

Following this random transition, the agent receives a stochastic reward R_{t+1}. The reward function $\mathcal{R}(s_t, a_t)$ gives the expected reward obtained when ac-

tion a_t is taken in state s_t, i.e.

$$\mathcal{R}(s_t, a_t) = \mathbb{E}\left[R_{t+1} | S_t = s_t, A_t = a_t\right] \tag{1.2}$$

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

Definition 1.2.2 (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 $s_0 \in \mathbb{S}$ together 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. We introduce the useful concept of history of an MDP

Definition 1.2.3 (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 on $\mathbb{S} \times \mathbb{A}$, defined on some probability space $(\Omega, \mathcal{F}, \mathbb{P})$, such that for $t = 0, 1, \ldots$*

$$\begin{cases} S_0 = s_0 \\ A_t \sim \pi(S_t, \cdot) \\ S_{t+1} \sim \mathcal{P}(S_t, A_t, \cdot) \end{cases} \tag{1.3}$$

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

$$\begin{aligned} \mathcal{P}_\pi(s, s') &= \int_{\mathbb{A}} \pi(s, a) \mathcal{P}(s, a, s') da \\ \mathcal{R}_\pi(s) &= \int_{\mathbb{A}} \pi(s, a) \mathcal{R}(s, a) da \end{aligned} \tag{1.4}$$

The goal of the agent is to maximize his expected return.

Definition 1.2.4 (Return)**.** *The return is the total discounted reward obtained by the agent starting from t*

$$G_t = \sum_{t=0}^{\infty} \gamma^t R_{t+k+1} \tag{1.5}$$

where $0 < \gamma < 1$ is the discount factor. The discount factor 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. Finally, the financial interpration is that discounting gives the present value of future rewards. Since the return are stochastic, we consider their expected value.

Definition 1.2.5 (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

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s \right] \tag{1.6}$$

where $\mathbb{E}_\pi$ indicates that all the actions are selected according to policy π. In reinforcement learning, it is useful to consider another function

Definition 1.2.6 (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

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right] \tag{1.7}$$

Definition 1.2.7 (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*

$$V_*(s) = \sup_\pi V_\pi(s) \tag{1.8}$$

Definition 1.2.8 (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*

$$Q_*(s, a) = \sup_\pi Q_\pi(s, a) \tag{1.9}$$

The agent goal is to select a policy π_* that maximize his expected return in all possible states. Such a policy is called *optimal*. More formally, we introduce the following partial ordering in the policy space

$$\pi \succeq \pi' \Leftrightarrow V_\pi(s) \geq V_{\pi'}(s) \quad \forall s \in \mathbb{S} \tag{1.10}$$

Then the optimal policy $\pi_* \succeq \pi$, $\forall \pi$.

1.2.1 Bellman Equations

1.2.2 Risk-Sensitive MDP

1.3 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.

1.3.1 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 $J : \Theta \to \mathbb{R}$

$$\max_{\theta \in \Theta} J(\theta) \tag{1.11}$$

There are various choices for the objective function.

Definition 1.3.1 (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*

$$J_{start}(\theta) = V_{\pi_\theta}(s^*) = \mathbb{E}_{\pi_\theta}\left[G_t | S_t = s^*\right] \tag{1.12}$$

Definition 1.3.2 (Average Value)**.** *In a continuing environment, the average value is the expected value that can be obtained following policy π_θ*

$$J_{avV}(\theta) = \mathbb{E}_{S\sim\mu}\left[V_{\pi_\theta}(S)\right] = \int_{\mathbb{S}} V_{\pi_\theta}(s)\mu(s)ds \tag{1.13}$$

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

Definition 1.3.3 (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*

$$J_{avR}(\theta) = \mathbb{E}_{\substack{S\sim\mu \\ A\sim\pi_\theta}}\left[\mathcal{R}(S, A)\right] = \int_{\mathbb{S}} \mu(s) \int_{\mathbb{A}} \pi_\theta(s, a)\mathcal{R}(s, a)dads \tag{1.14}$$

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

$$\theta_{k+1} = \theta_k + \alpha_k \nabla_\theta J|_{\theta=\theta_k} \tag{1.15}$$

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*

$$\sum_{k=0}^{\infty} \alpha_k = \infty \qquad \sum_{k=0}^{\infty} \alpha_k^2 < \infty \tag{1.16}$$

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.

1.3.2 Finite Differences

1.3.3 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

$$G(h) = \mathbb{E}\left[G_0 | H_t = h\right] = \sum_{t=1}^{\infty} \gamma^k \mathcal{R}(s_{t-1}, a_{t-1}) \tag{1.17}$$

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

$$J_{\text{avV}}(\theta) = \int_{\mathbb{H}} p_\theta(h) G(h) dh \tag{1.18}$$

We can compute its gradient using the likelihood ratio trick

$$\begin{aligned} \nabla_\theta J(\theta) &= \int_{\mathbb{H}} \nabla_\theta p_\theta(h) G(h) dh \\ &= \int_{\mathbb{H}} p_\theta(h) \nabla_\theta \log p_\theta(h) G(h) dh \\ &= \mathbb{E}\left[\nabla_\theta \log p_\theta(H) G(H)\right] \end{aligned} \tag{1.19}$$

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

$$\nabla_\theta \log p_\theta(H) = \sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(s_t, a_t) \tag{1.20}$$

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

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\nabla_\theta \log p_\theta(H)(G(H) - b)\right] \tag{1.21}$$

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

$$g_{\text{RF}} = \frac{1}{M} \sum_{m=1}^{M} \left[\sum_{i=0}^{T^{(m)}} \nabla_\theta \log \pi_\theta(s_i^{(m)}, a_i^{(m)})\right] \left[\sum_{j=0}^{T^{(m)}} \gamma^j r_{j+1}^{(m)} - b\right] \tag{1.22}$$

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

$$g_{\text{SRF}} = \left[\sum_{i=0}^{T} \nabla_\theta \log \pi_\theta(s_i, a_i)\right] \left[\sum_{j=0}^{T} \gamma^j r_{j+1} - b\right] \tag{1.23}$$

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]

1.3.4 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,

$$\mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(S_t, A_t) \mathcal{R}(S_s, A_s)\right] = 0 \quad \forall t > s \tag{1.24}$$

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

$$g_{\text{PG}} = \frac{1}{M} \sum_{m=1}^{M} \sum_{i=0}^{T^{(m)}} \nabla_\theta \log \pi_\theta(s_i^{(m)}, a_i^{(m)}) \left(\sum_{j=i}^{T^{(m)}} \gamma^j r_{j+1}^{(m)} - b\right) \tag{1.25}$$

$$g_{\text{GMDP}} = \frac{1}{M}\sum_{m=1}^{M}\sum_{j=0}^{T^{(m)}}\left[\sum_{i=j}^{T^{(m)}}\nabla_\theta \log \pi_\theta(s_i^{(m)}, a_i^{(m)})\right]\left(\gamma^j r_{j+1}^{(m)} - b\right) \tag{1.26}$$

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 1.3.1 (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*

$$\nabla_\theta J(\theta) = \mathbb{E}_{\substack{S\sim\mu \\ A\sim\pi_\theta}}\left[\nabla_\theta \log \pi_\theta(S, A) Q_{\pi_\theta}(S, A)\right] \tag{1.27}$$

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.

1.3.4.1 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

$$\theta_{k+1} = \theta_k + \alpha_k \nabla_\theta \pi_\theta(s_t, a_t)\widehat{Q}_\psi(s_t, a_t) \tag{1.28}$$

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]

1.3.5 Natural Policy Gradient

1.3.6 Policy Gradient with Parameter Exploration

In REINFORCE, trajectories are generated by sampling an action from a stochastic policy π_θ at each time step. This process leads to a large variance

in the gradient estimate which can slow down the convergence of the learning process. To address this issue, in [48] 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 to the infinite horizon setting.

1.3.6.1 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. $\theta \sim p_\xi(\theta)$. Combining these two hypotheses, we obtain a stochastic policy parametrized by the hyperparameters ξ

$$\pi_\xi(s,a) = \pi(s,a;\xi) = \int_\Theta p_\xi(\theta)\mathbb{1}_{F_\theta(s)=a}d\theta \tag{1.29}$$

where $\mathbb{1}_A$ denotes the indicator function of a set A. 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 period and then generate an entire trajectory using the deterministic policy F_θ. The hyperparameters ξ will be updated following the gradient of the expected reward, which can be rewritten as

$$J(\xi) = \int_\Theta \int_\mathbb{H} p_\xi(\theta,h)G(h)dhd\theta \tag{1.30}$$

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

$$\begin{aligned}\nabla_\xi J(\xi) &= \int_\Theta \int_\mathbb{H} \nabla_\xi p_\xi(\theta)p_\theta(h)G(h)dhd\theta \\ &= \int_\Theta \int_\mathbb{H} p_\xi(\theta)p_\theta(h)\nabla_\xi \log p_\xi(\theta)G(h)dhd\theta \\ &= \mathbb{E}\left[\nabla_\xi \log p_\xi(\theta)G(H)\right]\end{aligned} \tag{1.31}$$

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

$$\nabla_\xi J(\xi) = \mathbb{E}\left[\nabla_\xi \log p_\xi(\theta)\left(G(H) - b\right)\right] \tag{1.32}$$

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, this gradient can be approximated via Monte Carlo with

$$g_{\text{PGPE}} = \frac{1}{M}\sum_{m=1}^{M} \nabla_\xi \log p_\xi(\theta) \left[G\left(h^{(m)}\right) - b\right] \tag{1.33}$$

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

$$g_{\text{PGPE}} = \nabla_\xi \log p_\xi(\theta) \left[G\left(h^{(m)}\right) - b\right] \tag{1.34}$$

If we 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

$$\begin{aligned} \frac{\partial \log p_\xi(\theta)}{\partial \mu_i} &= \frac{\theta_i - \mu_i}{\sigma_i^2} \\ \frac{\partial \log p_\xi(\theta)}{\partial \sigma_i} &= \frac{(\theta_i - \mu_i)^2 - \sigma_i^2}{\sigma_i^3} \end{aligned} \tag{1.35}$$

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

$$\begin{aligned} \mu_i^{k+1} &= \mu_i^k + \alpha \left[G(h) - b\right](\theta_i - \mu_i) \\ \sigma_i^{k+1} &= \sigma_i^k + \alpha \left[G(h) - b\right] \frac{(\theta_i - \mu_i)^2}{\sigma_i} \end{aligned} \tag{1.36}$$

where b can be computed as a moving average of the past returns. In the original paper, the authors also propose a symmetric sampling technique similar to antithetic variates that further improves the convergence of the method. Finally, let us notice that more complex distributions for the parameters θ could be used.

1.3.6.2 Infinite Horizon PGPE

In this section we discuss the extension of the PGPE method to the infinite horizon case [47], which is relevant to the asset allocation problem. 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.

Chapter 2

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.

2.1 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

$$X_{t+1} = \frac{\Delta W_{t+1}}{W_t} = \sum_{i=0}^{I} \left[a_t^i X_{t+1}^i - \delta_i \left| a_t^i - \tilde{a}_t^i \right| - \delta_s (a_t^i)^- \right] - \delta_f \mathbb{1}_{a_t \neq \tilde{a}_{t-1}} \quad (2.1)$$

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

$$\sum_{i=0}^{I} a_t^i = 1 \qquad \forall t \in \{0, 1, 2, \ldots\} \quad (2.2)$$

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. (2.1), we obtain

$$X_{t+1} = X + \sum_{i=1}^{I} a_t^i (X_{t+1}^i - X) - \sum_{i=0}^{I} \left[\delta_i \left| a_t^i - \tilde{a}_t^i \right| - \delta_s^i (a_t^i)^- \right] - \delta_f \mathbb{1}_{a_t \neq \tilde{a}_{t-1}} \quad (2.3)$$

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

$$R_{0,T} = \log \frac{W_T}{W_0} = \sum_{t=1}^{T} \log(1 + X_t) = \sum_{t=1}^{T} R_t \quad (2.4)$$

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

$$R_{t+1} = \log \left\{ 1 + \sum_{i=0}^{I} \left[a_t^i X_{t+1}^i - \delta_i \left| a_t^i - \tilde{a}_t^i \right| - \delta_s (a_t^i)^- \right] - \delta_f \mathbb{1}_{a_t \neq \tilde{a}_{t-1}} \right\} \quad (2.5)$$

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.

2.2 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

$$S_t = \{X_t, X_{t-1}, \dots, X_{t-P}, \tilde{a}_t, Y_t, Y_{t-1}, \dots, Y_{t-P}\} \tag{2.6}$$

2.3 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

$$\begin{cases} a_t^i \in \mathbb{R} \ , \ \forall i \in \{0, \dots, I\} \\ \sum_{i=0}^I a_t^i = 1 \end{cases} \tag{2.7}$$

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

$$\begin{cases} a_t^i \in \mathbb{R} \ , \ \forall i \in \{1, \dots, I\} \\ a_t^0 = 1 - \sum_{i=1}^I a_t^i \end{cases} \tag{2.8}$$

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

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

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

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

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

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

[7] Bertoluzzo, F., and Corazza, M. Reinforcement learning for automated financial trading: Basics and applications. In *Recent Advances of Neural Network Models and Applications*. Springer, 2014, pp. 197–213.

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

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

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

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

[12] Casqueiro, P. X., and Rodrigues, A. J. Neuro-dynamic trading methods. *European Journal of Operational Research 175*, 3 (2006), 1400–1412.

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

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

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

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

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

[18] Dempster, M. A., and Leemans, V. An automated fx trading system using adaptive reinforcement learning. *Expert Systems with Applications 30*, 3 (2006), 543–552.

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

[20] Deng, Y., Bao, F., Kong, Y., Ren, Z., and Dai, Q. Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning Systems* (2016).

[21] Deng, Y., Kong, Y., Bao, F., and Dai, Q. Sparse coding inspired optimal trading system for hft industry. *IEEE Transactions on Industrial Informatics 11*, 2 (2015), 467–475.

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

[23] Elder, T. Creating algorithmic traders with hierarchical reinforcement learning.

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

[25] Ganchev, K., Nevmyvaka, Y., Kearns, M., and Vaughan, J. W. Censored exploration and the dark pool problem. *Communications of the ACM 53*, 5 (2010), 99–107.

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

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

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

[29] Hendricks, D., and Wilcox, D. 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.

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

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

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

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

[34] Laruelle, S., Lehalle, C.-A., and Pages, G. Optimal split of orders across liquidity pools: a stochastic algorithm approach. *SIAM Journal on Financial Mathematics 2*, 1 (2011), 1042–1076.

[35] Laruelle, S., Lehalle, C.-A., and Pages, G. Optimal posting price of limit orders: learning by trading. *Mathematics and Financial Economics 7*, 3 (2013), 359–403.

[36] Li, H., Dagli, C. H., and Enke, D. 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.

[37] Moody, J., and Saffell, M. Learning to trade via direct reinforcement. *Neural Networks, IEEE Transactions on 12*, 4 (2001), 875–889.

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

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

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

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

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

[43] O, J., Lee, J., Lee, J. W., and Zhang, B.-T. Adaptive stock trading with dynamic asset allocation using reinforcement learning. *Information Sciences 176*, 15 (2006), 2121–2147.

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

[45] Peters, J., and Schaal, S. Reinforcement learning of motor skills with policy gradients. *Neural networks 21*, 4 (2008), 682–697.

[46] Saad, E. W., Prokhorov, D. V., and Wunsch, D. C. Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks. *IEEE Transactions on Neural Networks 9*, 6 (1998), 1456–1470.

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

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

[49] Sehnke, F., Osendorfer, C., Rückstiess, T., Graves, A., Peters, J., and Schmidhuber, J. Parameter-exploring policy gradients. *Neural Networks 23*, 4 (2010), 551–559.

[50] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. Deterministic policy gradient algorithms. In *ICML* (2014).

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

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

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

[54] Tamar, A., Castro, D. D., and Mannor, S. 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.

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

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

[57] Tan, Z., Quek, C., and Cheng, P. Y. Stock trading with cycles: A financial application of anfis and reinforcement learning. *Expert Systems with Applications 38*, 5 (2011), 4741–4755.

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

[59] Werbos, P. J. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE 78*, 10 (1990), 1550–1560.

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

[61] Williams, R. J., and Zipser, D. A learning algorithm for continually running fully recurrent neural networks. *Neural computation 1*, 2 (1989), 270–280.

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