Contents

Chapter 1

Preliminaries

In this chapter we provide an introduction to the Multi-Armed Bandit framework and then Reinforcement Learning framework, which are the two frameworks of reference of our research project. The Reinforcement Learning section is opportunely preceded by an overview of Markov Decision Processes, which are the building blocks of Reinforcement Learning. Then, we present a particular class of strategies to solve the Reinforcement Learning problem: Policy Search methods. The aim is to introduce concepts that will be used extensively in the following chapters.

1.1 Multi Armed Bandits

Consider a situation in which a gambler (the *agent*) is sitting in a casino, staring at a number of slot machines from which he wants to pick the most profitable one and play with it all night long. Each slot machine is characterized by a certain *mean reward*. The objective of the gambler is to maximize the gains that he will have won by the end of the night, but, unfortunately, he can not know *a priori* the mean reward associated to each slot machine. Hence, the gambler must find a suitable strategy in order to effectively *explore* his possibilities, by testing the different slot machines, and, then, *exploit* the ones that prove to be more profitable. In fact, the gambler would face an equivalent problem if he was to play a single machine with as many *arms* (levers) as the number of slot machines present in the casino, each with its own specific mean reward. Hence the name Multi Armed Bandit (MAB) [58] given to this problem, in homage to the one-armed bandit, an old-fashioned name for a lever operated slot machine

("bandit" because it steals your money). In the artificial intelligence literature, MAB refers to a broad class of problems that can be formulated as this gambling learning problem. Of course, MABs have other, more valuable, applications other than gambling. Current applications span from web interfaces configuration, news recommendation, dynamic pricing, ad placement, networking routing and game design.

Definition 1.1.1. (Multi Armed Bandit) *A MAB is a tuple* $\langle \mathcal{X}, \mathcal{R} \rangle$*, where:*

(i) $\mathcal{X}$ *is the* $d_{\mathcal{X}}$*-dimensional set of possible arms, or actions that the agent can pick;* $\mathcal{X}$ *can be discrete or continuous, one dimensional or multi-dimensional, finite or infinite.*

(ii) $\mathcal{R} : \mathcal{X} \to \Delta([-R_{\max}, R_{\max}])$ *is an unknown function of rewards such that for every* $x \in \mathcal{X}$ *it assigns a probability measure* $\mathcal{R}(\cdot|x)$ *over* $[-R_{\max}, R_{\max}]$*, a bounded set of real-valued rewards.* $R_{\max}$ *is the maximum absolute reward that the agent can receive.*

One can think to the set of possible arms and reward functions associated to them as the *environment* the learner interacts with. According to the specific characteristics of the environment, there exist a broad taxonomy of MABs. For our reader, it is sufficient to know that in this work we deal with *stochastic stationary bandits*. In this case, the environment is restricted to generate the reward in response to each action from a *probability distribution* $\mathcal{R}(\cdot|x)$ that is specific to that action, stationary along time and independent of the previous action choices and rewards. For example, the reward distribution could be Gaussian or Bernoulli. The expected reward of the stationary distribution associated to arm x is noted $\mu(x) = \mathop{\mathbb{E}}_{r \sim \mathcal{R}(\cdot|x)}[r]$.
The MAB game is played sequentially by a *learner* over multiple rounds $t = 1, 2, \ldots, T$, up to the *horizon* $T \in \mathbb{N}$, which depends on the problem at hand. Evidently, the agent can have memory but can not foresee the future. Thus, the current action x_t should only depend upon the sequence of previous actions and rewards, the *history* $\mathcal{I} = \{x_0, r_0, x_1, r_1, \ldots, x_{t-1}, r_{t-1}\}$. A *policy* is a mapping from histories to actions which represents the behaviour of the agent, its strategy. the first question which springs to mind is then: how can we evaluate the quality of a policy? To this aim, the literature extensively adopts a performance measure called the *regret*;

Definition 1.1.2. (Immediate Regret) *The immediate regret suffered by a learner*

at round t is:

$$\Delta_t = \mu(x^*) - \mu(x_t) \tag{1.1}$$

where $x^* = \arg\max_{x \in \mathcal{X}} \mu(x)$ *is the arm with the highest expected reward i.e., the optimal arm.*

Definition 1.1.3. (Regret) *The regret of a learner is the cumulated sum of the instantaneous regrets:*

$$Regret(T) = \sum_{t=0}^{T} \Delta_t \tag{1.2}$$

The quality of a certain policy is then given by the rate of growth of the regret as the horizon T grows. A good learner achieves sub-linear regret, which means that $Regret(T) = o(T)$ or, equivalently, that $\lim_{T \to \infty} Regret(T)/T = 0$. For example, some state of the art policies for different bandit settings have regrets close to $O(\sqrt{T})$ or $O(\log(n))$ [62]. The regret has two characteristics that make it a good performance metric. First, it supplies a degree of normalization because it is invariant under translation of rewards. Second, it represents the price paid by the learner for not knowing the true environment. Anyway, a crucial aspect for designing a sub-linear regret policy is to carefully balance the trade-off between exploration and exploitation that characterizes all bandits.

1.1.1 Exploration and exploitation

In order to maximize its cumulative reward, a bandit agent must prefer high-reward arms discovered in past rounds. But to discover such arms, it has to spend a certain amount of rounds trying arms that it has not selected before. In other words, the agent has to *exploit* the most profitable actions revealed in the past, but is also has to *explore* in order to make better action selections in the future. If the reward function is stochastic as in stationary stochastic bandits, even more resources should be dedicated to exploration, because one trial is not enough to have a good estimate of the expected reward of one arm. In fact, the exploration-exploitation trade-off is a well known mechanisms that goes back to psychology and it affects us all in our daily lives. Take for example a typical lunch break on a working day. Say, that your favourite restaurant is right around the corner. If you go there every day (*exploitation*), you would be confident of what you will get, but miss the chances of discovering an even better option. On the contrary, if you try new places all the time (*exploration*), you are very likely gonna have to eat unpleasant food from time to time. The sweet spot is

usually in between these two extreme options and a crucial challenge for any bandit algorithm is to properly balance between exploration and exploitation. As we will see, this dilemma arises in *reinforcement learning* too and will be at the very core of our discussion.
Now, we proceed by briefly sketching out two bandit settings that are of central interest to this work.

1.1.2 Stochastic Bandits With Finitely Many Arms

An important declination of the bandit problem, much studied in the literature [62], is the stationary stochastic bandit problem with *finitely many arms*, i.e., stationary stochastic bandits whose action space consists of a discrete arm set $|\mathcal{X}| = K \in \mathbb{N}_+$. This setting is particularly important because of its semplicity, which makes it a perfect starting point for understanding the exploration-exploitaiton trade-off and for designing algorithms that can be extended to more complex settings afterwards. Also, many real world applications can be modeled as finitely-armed stationary stochastic bandits.
In order to deepen our understanding of this setting, and of the bandit problem in general, let's discuss the most simple policy that one can imagine: the *explore-then-commit* algorithm. Basically, this policy consists in choosing each arm a certain number of times m and subsequently exploit by playing the arm that appeared best after exploration. Because there are K actions, the algorithm will explore for mK rounds before choosing a single action for the remaining rounds. The choice of the agent goes to the arm i with the highest average pay-off received up to round t:

$$\widehat{\mu}_i(t) = \frac{1}{T_i(t)} \sum_{s=1}^{t} \mathbb{1}_{\{x_s=i\}} r_s \tag{1.3}$$

where $T_i(t) = \sum_{s=1}^{t} \mathbb{1}_{\{x_s=i\}}$ is the number of times up to round t the agent picks action i. Algorithm 1.1 shows the pseudo-code of the explore-then-commit policy. Recall $\forall a, b \in \mathbb{N}^+, a \bmod b = a - b\lfloor a/b \rfloor$.
It suffices a quick glance to the algorithm to have a tangible demonstration of the importance of the trade-off between exploration and exploitation. This is confirmed by the analysis of the regret.

Theorem 1.1.1. *[62] The expected regret of the explore-then-commit policy is*

Algorithm 1.1 Explore-then-commit

1: **Input:** $m \in \mathbb{N}$
2: **for** $t = 1, \dots, T$ **do**
3: Choose arm

$$x_t = \begin{cases} i, & \text{if } (t \bmod K) + 1 = i \text{ and } t \leqslant mK \\ \arg\max \widehat{\mu}_i(mK), & \text{if } t > mK \end{cases}$$

4: **end for**

bounded by:

$$Regret(T) \leqslant \min\left(m, \lceil \tfrac{T}{K} \rceil\right) \sum_{i=1}^{K} \Delta_i + \max\left(T - mK, 0\right) \sum_{i=1}^{K} \Delta_i \exp\left(-\frac{m\Delta_i^2}{4}\right) \quad (1.4)$$

where Δ_i is the immediate regret of arm i as defined in 1.1.2.

Evidently, if m is large the policy explores for too long and the first term will be eventually too large. On the other hand, if m is too small, then the probability that the algorithm exploits the wrong arm will grow and the second term will become too large.

1.1.3 $\mathcal{X}$-armed bandits

Another stationary stochastic bandit setting interesting to our scope is the *continuum-armed bandit* setting [2]. Here, the arms are chosen from a subset of the real numbers $\mathcal{X} \in \mathbb{R}^{d_\mathcal{X}}$, hence, the action space is infinite. Moreover, the mean rewards are assumed to be a continuous function of the arms. An example of such bandits are continuous *Lipschitz bandits* [63], where the expected reward is a Lipschitz function of the arm. This assumption is extremely useful because it implies that the information obtained by selecting one arm can be shared to its neighbouring arms. Thus, exploration can be made more efficient, and possibly directed, by exploiting the correlation between arms. *$\mathcal{X}$-armed bandits* [22] are a further generalization of this setting where the set of arms, $\mathcal{X}$, is allowed to be a generic measurable space and the mean-payoff function is *weak Lipschitz* with respect to a dissimilarity function that is known to the decision maker.

By now, the reader should have noticed that a distinguishing feature of bandit problems is that the learner never needs to plan for the future. More precisely, in bandits we invariably make the assumption that the learner's choices and

rewards tomorrow are not affected by its decision today. In the next chapter, we discuss a more general framework that includes this kind of long-term planning.

1.2 Markov Decision Processes

A Markov Decision Process (MDP) is another way to model the interaction between an agent, which is the learner and the decision maker, and an environment. When the agent performs an action a, the environment responds presenting a new state s to the agent and rewarding the agent with a certain scalar signal called *immediate reward*. Importantly, the environment dynamics of a MDP are *stationary*, i.e., they do not depend upon time. Moreover, the state s_{h+1} at time step $h+1$ must depend only on the previous state s_h and action a_h. This property, called *Markovian Property*, entails that the agent is somehow pushed to forget the states and actions of the past. Still, differently from what happened in bandits, the current state and set of possible actions and subsequent rewards the agent faces, are determined by his previous choice. This is a strong mechanism that encourages the agent to take into account the consequences of its choices over time. On top of this, there is another mechanism that makes the agent even more concerned by long term consequences. In fact, the objective of the agent is to maximize over time the *cumulative reward* or *return*, which is the cumulated sum of the immediate rewards obtained after each action undertaken by the agent. Evidently, sometimes it is more profitable to sacrifice immediate reward in order to reach a higher cumulative reward in the long term. This mechanism pushes the agent to take into account the future in its current decisions.
Hence, MDPs are a powerful tool capable of modelling many challenges that we encounter in life, science and engineering. Take, for example, a hungry newborn infant in his mother's arms. Through various attempts, the infant moves around his arms, head and mouth looking for food. In this way, he changes its state until the moment he eventually receives a positive reward, the mother's milk. Little by little, the baby will learn the precise sequence of actions and states that rewards him with milk. Having understood the general principles underlying MDPs, we can now move to a formal definition.

Definition 1.2.1. (Markov Decision Process) *A continuous MDP [83, 96] is a tuple* $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mu \rangle$*, where:*

(i) $\mathcal{S} \in \mathbb{R}^{d_{\mathcal{S}}}$ *is the* $d_{\mathcal{S}}$*-dimensional* state space*, i.e., the set of all possible observable states of the environment.*

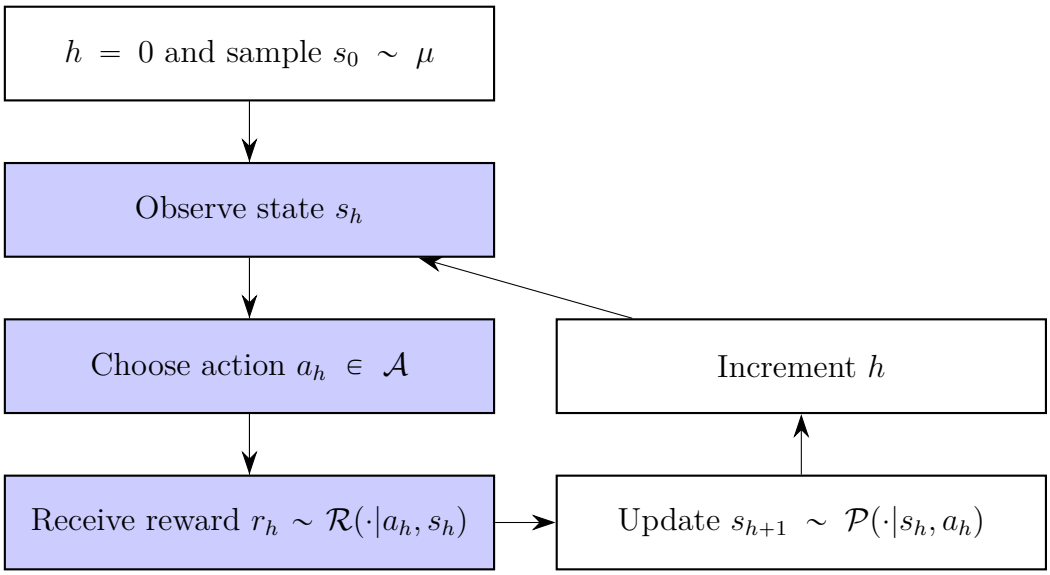

Figure 1.1: Interaction protocol for Markov decision processes

(ii) $\mathcal{A} \in \mathbb{R}^{d_{\mathcal{A}}}$ *is the* $d_{\mathcal{A}}$*-dimensional* action space*, i.e., the set of all possible actions that the agent can perform. Sometimes, not all the actions are performable in all states. In such cases, we can define the set* $\mathcal{A}(s)$ *for all* $s \in \mathcal{S}$*, s.a* $\mathcal{A} = \bigcup_{s \in \mathcal{S}} \mathcal{A}(s)$*.*

(iii) $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ *is a function called* transition model *such that, for every* $s \in \mathcal{S}$ *and* $a \in \mathcal{A}$*, it assigns a probability measure* $\mathcal{P}(\cdot|s, a)$ *over* $\mathcal{S}$*. In general, we denote with* $\Delta(\mathcal{X})$ *the set of probability measures over a measurable space* $\mathcal{X}$*; The corresponding probability density function is* $P(\cdot|s, a)$*.*

(iv) $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \Delta[-R_{\max}, R_{\max}]$ *is a bounded* reward *function such that, every time the agent chooses action* $a \in \mathcal{A}$ *in state* $s \in \mathcal{S}$*, it assigns a probability measure* $\mathcal{R}(\cdot|s, a)$ *over* $[-R_{\max}, R_{\max}]$*. With little abuse of notation, we will denote* $\mathcal{R}(a, s)$ *its expected return too.* $R_{\max}$ *is the maximum absolute reward that the agent can receive.*

(v) $\gamma \in (0, 1]$ *is a discount factor to apply to future rewards. From an economical point of view, it accounts for the fact that an agent might be more interested in a pay-off obtained in the near future rather than a pay-off obtained far in the future. From a statistical point of view the discount factor is related to the probability that the process continues for another decision epoch.*

(vi) $\mu \in \Delta(\mathcal{S})$ *is the initial state distribution, such that the initial state is drawn as* $s_0 \sim \mu$*.*

In the more general case, the state space $\mathcal{S}$ and the action space $\mathcal{A}$, which are the sensor and actuator possibilities of the agent, respectively, can be both continuous and discrete, finite or infinite. In what follows, we will focus on the continuous case because it is the most relevant to our work. The time is typically modelled as a discrete sequence of decision steps represented by the natural numbers: $\mathcal{H} = \{0, 1, \dots, H\}$, where $H \in \mathbb{N}$ is the *horizon* of a given task, which can be either infinite $H = \infty$ or finite. The agent-environment interaction ends when either the horizon is reached or a *terminal state* is reached, i.e., a state from which no other state can be reached. Tasks are called *episodic* when there exists a terminal state. After having reached a terminal state, usually all the rewards are considered to be zero. A *trajectory* is the sequence of states, actions and rewards $r_{h+1} \sim \mathcal{R}(\cdot|a_h, s_h)$ up to the last time step of the episode: $\tau = \{s_h, a_h, r_{h+1}\}_{h=0}^{H-1}$.

1.2.1 Policies

The core of a MDP agent is the *policy* π, a mapping from perceived states of the environment to possible actions.

Definition 1.2.2. (Policy) *A policy is a stochastic function* $\pi : \mathcal{S} \to \Delta(\mathcal{A})$*, such that the current action is drawn as* $a \sim \pi(\cdot|s)$

Hence, deterministic policies $\pi : \mathcal{S} \to \mathcal{A}$, such that the current action is prescribed as $a = \pi(s)$, are a particular case. As for bandits, in the most general case a policy takes as input the past history of states and actions, but in our work we only consider *memoryless policies*. In fact, we can always find an optimal policy that depends only on the current state. Moreover, we assumed that the agent's policy is stationary.

Remark 1.2.1. Note that we are adopting different notations for time steps in bandits and MDPs, t and h respectively. The reason is that we can think of a full MDP episode performed with policy π as a unique decision epoch t of a bandit. In other words, one could consider a bandit in which, at iteration t, the *bandit agent* pulls a policy π. The *MDP agent* perform a trajectory by interacting with its environment according to policy π. Then, the bandit agent obtains a reward $r_t = \sum_{h=1}^{H} r_h$. This is the modelling approach that we will adopt in our work. For all details see IternalRef.

1.2.2 Performance

A we stated above, the goal of an MDP agent is usually to maximize the total discounted reward, also called *return*:

$$\nu = \sum_{h=0}^{\infty} \gamma^h r_{h+1}, \tag{1.5}$$

where $\gamma \in (0, 1]$ is the discount rate. Given this objective, how can we evaluate the agent's policy ? We need to design a *utility function* which represents the performance of the agent's policy with respect to its objective. There are two common formulations of the *performance* $J(\pi)$ in the reinforcement learning literature [97]. In the *start state formulation* the performance is calculated as the long-term discounted reward obtained by starting from a designated start state s_0:

$$J(\pi) = \mathbb{E}[\nu \mid s_0 \sim \mu, \pi]. \tag{1.6}$$

Note that $\gamma = 1$ is allowed in episodic tasks only, otherwise the convergence of the performance is not guaranteed. We can now conveniently introduce the *stationary distribution of states under* π:

$$d^{\pi}(s) = (1 - \gamma) \sum_{h=0}^{\infty} \gamma^h Pr(s_h = s \mid s_0 \sim \mu, \pi), \tag{1.7}$$

which is the probability induced by the policy π of having $s_h = s$, discounted by γ^h, when $t \to \infty$. This allows us to rewrite the performance measure as:

$$J(\pi) = \int_{\mathcal{S}} d^{\pi}(s) \int_{\mathcal{A}} \pi(a|s) \mathcal{R}(s, a) da ds. \tag{1.8}$$

In the *average reward formulation* policies are ranked according to their long-term expected reward per step:

$$\begin{aligned} J(\pi) &= \lim_{n \to \text{inf}} \frac{1}{n} \mathbb{E}\left[\sum_{h=1}^{n} r_h \mid \pi\right] \\ &= \int_{\mathcal{S}} d^{\pi}(s) \int_{\mathcal{A}} \pi(a|s) \mathcal{R}(s, a) da ds, \end{aligned} \tag{1.9}$$

where $d^\pi(s) = \lim_{n\to\inf} Pr(s_h = s \mid s_0 \sim \mu, \pi)$. In this work, we mostly adopt the first formulation of the performance utility function on episodic tasks, i.e., tasks with an absorbing (terminal) state.

1.2.3 Value functions

From the start state formulation of performance we can derive the value associated to a state s by following policy π, i.e., the *state-value function* $V^\pi(s) : \mathcal{S} \to \mathbb{R}$ defined recursively as:

$$V^\pi(s) = \mathbb{E}[\nu \mid s_0 = s, \pi] \tag{1.10}$$

$$= \int_{\mathcal{A}} \pi(a|s) \left(\mathcal{R}(s,a) + \gamma \int_{\mathcal{S}} P(s'|s,a) V^\pi(s') ds' \right) da, \tag{1.11}$$

which allows to re write the performance as:

$$J(\pi) = \int_{\mathcal{S}} \mu(s) V^\pi(s) ds. \tag{1.12}$$

Equation 1.11 is known as *Bellman's equation* of the state-value function. Thanks to another Bellman's equation we can define another value function, namely $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. This value function is more practical for control problems, in which we prefer to reason in terms of which actions are more valuable in a given state. Thus, we define the *action-value function* $Q^\pi(s,a)$ as:

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}[\nu \mid s_0 = s, a_0 = a, \pi] \\ &= \mathcal{R}(s,a) + \gamma \int_{\mathcal{S}} P(s'|s,a) \int_{\mathcal{A}} \pi(a'|s') Q^\pi(s',a') da' ds', \end{aligned} \tag{1.13}$$

The difference between the two value functions is known as advantage function [12]:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s) \tag{1.14}$$

Intuitively, the advantage function represents how much an action a is convenient is a state s w.r.t. the average utility of the actions.

Value functions are useful for evaluating the quality of a given policy or in order to define policies themselves. In fact, having an estimation of the value function for every state (and possibly action) one can build a policy following a greedy scheme. At each step we may choose the state-action pair that maximize the action-value function, or an action that brings to the maximally valuable state among those that are reachable from the current state.

1.3 Reinforcement Learning

The most basic way to solve MDPs is *dynamic programming*, which simply consists in solving the Bellman equations presented in the previous chapter through a fixed-point iteration scheme. *Value iteration* and *Policy Iteration* are the most popular dynamic programming algorithms [96], which provide the basic structure for most of the value-based reinforcement learning methods. However, these algorithms suffer of two major problems. First, they require knowledge of the MDP transition kernel and of the reward function, which is not available in many real world problems. Second, the update involves an optimization with respect to all possible actions, which can be carried out efficiently only in the case of finite (small) action spaces. Reinforcement Learning (RL) is a subfield of Machine Learning which aims at solving large-scale problems where the dynamics of the MDP are not know, and thus dynamic programming algorithms can not be employed.

1.3.1 Problem formulation

RL is a branch of machine learning that aims at building learning agents capable 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 state-action pairs are the most profitable by trial-and-error. Therefore, RL algorithms can be seen as computational methods to solve MDPs by directly interacting with the environment, for which a model may or may not be available. The RL problem can be formulated as a stochastic optimization problem aiming at finding the optimal policy π^*:

$$\pi^* = \arg\max_{\pi} J(\pi) \tag{1.15}$$

Remarkably, it is guaranteed that every MDP admits an optimal policy when the space of possible policies is unconstrained [83].

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 usually less frequent and can be very sparse. In supervised learning, the agent is provided with examples of the correct or expected behaviour by a knowledgeable external supervisor and the agent's 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 receives a numerical reward that only represents a partial feedback about the goodness of actions taken. A feedback system of this sort is said to be *evaluative* rather than *instructive* and it makes it much more difficult for the agent to learn how to behave in uncharted territory. Evidently, the exploration-exploitation trade-off is a substantial problem in RL as much as for bandits. An agent must exploit what is known of the environment and the reward function in order to obtain rewards, but it also needs to explore to select better actions in the future. On top of this, since rewards might be delayed in time, it will be difficult for the agent to understand which actions are mostly responsible for the outcome received. This represents another characteristic problem of reinforcement learning known as *assignement problem* [96]. As we will see, RL algorithms are based on two key ideas: the first is to use samples to derive an approximate representation of the unknown dynamics of the controlled system. The second idea is to use function approximation methods to estimate value functions and policies in high-dimensional state and action spaces.

1.3.2 Taxonomy

Along the years, many algorithms have been proposed to solve the RL problem as presented above. The solutions proposed often display similar approaches that can be used to outline a rough taxonomy of the RL algorithms. In our taxonomy we consider the algorithms along four dimensions: *model requirements*, *policy-based sampling strategy*, *solution search*, *sample usage*. The first dimension refers to the requirements of the algorithm w.r.t. the model of the MDP. On the two sides of this dimension we have:

- *Model-based* algorithms, which require an explicit approximation of the model of the MDP. Not every model-based algorithm is equally demanding

on this matter: some of them might need information about every element of the MDP (transition model, reward model, initial state distribution, etc.) while others might require approximations only for a fraction of them, e.g., the reward model only. Examples: DYNA [95], contextual policy-search[57].

- *Model-free* algorithms, on the other hand, do not require any explicit approximation of the model. Model-free algorithms usually have low computational demands but require lots of data in order to get good results; they are well-suited for problems in which computation is costly but sampling is cheap. Examples: Q-learning [104], SARSA [86], REINFORCE [107].

The *policy-based sampling strategy* dimension refers to the relation between the policy that is being used to interact with the environment (*behavioural policy*) and the policy that is being learned by the agent (*target policy*);

- in *off-policy* algorithms there is a clear distinction between the behavioural policy, the one which collects samples and explore the environment, and target policy, which is learned independently. This approach is often useful to safely improve exploration but it is usually more cumbersome to analyse theoretically. Examples: Q-learning [104], off-policy actor-critic [35].
- in *on-policy* algorithms there is no distinction between target and behavioural policy. Examples: SARSA [86], TD$\{\lambda\}$ [94].

The *solution search-space* dimension refers to the space where the optimal solution is searched:

- in direct *policy search* methods the optimal solution is searched in the space of policies, e.g., the space in which their parameters lie. These methods are well-suited for large state or action spaces, produce smooth changes in the policy during the learning process, and allow to introduce prior expert knowledge in the policy; unfortunately, they often suffer from a high variance and have optimization issues. Examples: REINFORCE [107], G(PO)MDP [13].
- *Value-based* methods rely instead on the computation of the value function to learn the optimal policy indirectly. Value-based techniques have good

convergence properties under small state and action spaces and turn out very suitable when there is insufficient or none prior knowledge about the problem. However, changes to the underlying policy during the learning process can be unstable, and the convergence properties disappear when large (e.g., continuous) state and action spaces demand the use of function approximation. Examples: value-iteration [96], FQI [6].

Differently from the other dimensions which are populated by sharp dichotomies, policy and value-based methods can be combined, often with very successful results. For example, *actor-critic* methods [42] look for the optimal policy in the policy parameter space while leveraging on the predictive power of value-functions in order to guide the parameters update.
The fourth dimension of the taxonomy is the *sample usage frequency*, which refers to the degree of interleaving between learning and experience. Along this dimension, we distinguish:

- *continuing* algorithms, which update the policy (directly or indirectly through the value function) every time new information is available, possibly at each time step;
- *episodic* algorithms, which divides experience into finite segments, called episodes, and update the policy in between each episode;
- *batch* algorithms, which divide learning and experience in two distinct phases. Typically, these are off-policy methods in which a set of behavioural policies collects the data and the target policy is updated upon completion of the batch.

A finer classification of this dimension is presented in [60], where the authors remark that the distinctions between the categories does not depend on the actual formulation: almost any algorithm can fall in any of these three categories depending on the implementation.

1.4 Policy Search

As mentioned in the introduction, this work focuses on improving the exploration task in continuous state-action domains, possibly characterized by some noise in the state as it happens for control tasks with noisy sensors. In such conditions,

value-based methods present numerous difficulties. First, they have no guarantees of convergence to an optimal policy due to the need of function approximation for large, continuous MDPs; second, value based methods are highly sensitive to small perturbations in the state, which makes them useless in noisy control tasks. Hence, direct Policy Search (PS) methods represent the best alternative in such settings. Theoretical guarantees are often available [70], environment noise has less impactful effects, and prior domain knowledge can be exploited to design ad-hoc policies for specific tasks. Moreover, approximate value functions can still be used to guide the search for the optimal policy, such as in actor-critic methods. Indeed, PS has been successfully applied to complex control tasks, as reported in [36]: from robotic arms playing baseball to simulated table tennis, from dart throwing to pan-cake flipping.

1.4.1 Overview

Given a predetermined class of approximating policies $\hat{\Pi}$, PS aims at finding the policy whose performance $J(\pi)$ is as close as possible to the performance of the optimal policy $J(\pi^*) = J^*$:

$$\hat{\pi} = \arg\min_{\pi \in \hat{\Pi}} \|J^* - J(\pi)\|_p . \tag{1.16}$$

When $p = 2$, this optimization problem can be interpreted as finding the policy $\hat{\pi} \in \hat{\Pi}$ whose orthogonal projection in the space of Markovian stationary policies Π coincides with the optimal policy π^*.
A Variety of technique have been proposed in literature to solve the problem defined by Equation (1.16). As well as for value-based methods, we can distinguish between model-free, which learn policies directly based on sampled trajectories, and model-based approaches, which use the sampled trajectories to first build a model of the state dynamics. A part from this macro distinction, a further classification can be based on different policy update strategies, as depicted in Figure (1.2). The policy updates in both model-free and model-based PS are based on either policy gradients Policy Gradient (PG), expectation maximization-based updates, or information-theoretic insights (Inf.Th.). In the followings, we will focus on model-free PS with PG updates.

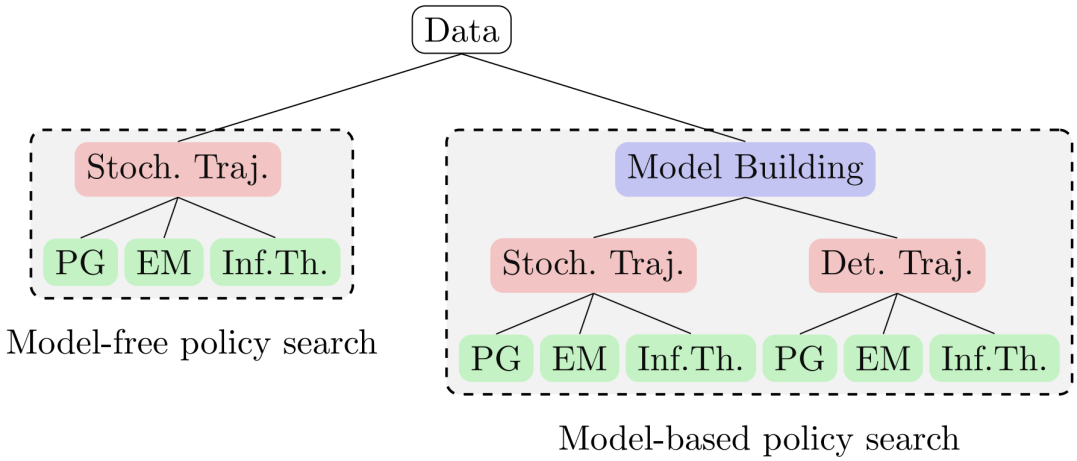

Figure 1.2: A taxonomy of PS methods [36]

1.4.2 Policy gradient

In the context of model-free PG, the agent's behaviour is modelled as a differentiable parametric policy $\pi_{\boldsymbol{\theta}} : \mathcal{S} \to \Delta(\mathcal{A})$, independent of time (*stationary*), such that the current action is drawn as $a_h \sim \pi_{\boldsymbol{\theta}}(\cdot|s_h) = \pi(\cdot|s_h, \boldsymbol{\theta})$, where $\boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^m$ are the policy parameters. The most simple parametrization one could think of is the linear one, where the policy only depends linearly on the policy parameters and the action is drawn deterministically:

$$a = \pi_{\boldsymbol{\theta}}(s) = \boldsymbol{\theta}^T \phi(s), \tag{1.17}$$

where $\phi(s)$ can be any differentiable approximation function. In stochastic formulations, typically, a zero-mean Gaussian noise vector is added to $\pi_{\boldsymbol{\theta}}(s)$, so that the policy becomes:

$$\pi_{\boldsymbol{\theta}}(a|s) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\left(\frac{a - \boldsymbol{\theta}^T\phi(s)}{\sigma}\right)^2\right). \tag{1.18}$$

In some cases, the covariance matrix is learnable too, and the parameters set becomes $\{\mu_{\boldsymbol{\theta}} = \boldsymbol{\theta}^T\phi(s); \Sigma\}$. In most cases, Σ is diagonal, thus easing the complexity of the learning task along with the theoretical analysis. Hence, the Problem (1.16) translates into finding the policy optimal parameters:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta} \in \Theta} J(\boldsymbol{\theta}). \tag{1.19}$$

This new optimization problem can be solved by resorting to gradient ascent on the policy parameters, which is guaranteed to converge to a local optimum at least:

$$\boldsymbol{\theta}^{t+1} \leftarrow \boldsymbol{\theta}^t + \alpha \boldsymbol{G}^{-1}(\boldsymbol{\theta}^t) \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^t), \tag{1.20}$$

where $\alpha \geqslant 0$ is the *learning rate* or *step size*, and $\boldsymbol{G}(\boldsymbol{\theta})$ is a positive definite matrix. The quantity $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ denotes the *policy gradient* PG which gives the name to this method. Its expression is derived in [97].

Theorem 1.4.1. Policy Gradient Theorem *Given a MDP $\mathcal{M}$ and a Markovian stationary stochastic parametric policy $\pi_{\boldsymbol{\theta}}$ differentiable w.r.t. to $\boldsymbol{\theta}$ for all state-action pairs $(s, a) \in (\mathcal{S}, \mathcal{A})$, the gradient of the policy performance is given by:*

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \int_{\mathcal{S}} d^{\pi_{\boldsymbol{\theta}}}(s) \int_{\mathcal{A}} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a|s) Q^{\pi_{\boldsymbol{\theta}}}(s, a) da ds$$

$\pi_{\boldsymbol{\theta}}(s)$ is the stationary state distribution induced by policy $\pi_{\boldsymbol{\theta}}$, as defined in Equation (1.7). By applying a trivial differentiation rule, $\nabla \log f = \nabla f / f$, we can rewrite Equation (1.21) in a way that will turn out extremely useful:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \int_{\mathcal{S}} d^{\pi_{\boldsymbol{\theta}}}(s) \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) Q^{\pi_{\boldsymbol{\theta}}}(s, a) da ds \\ &= \mathop{\mathbb{E}}_{s \sim d^{\pi_{\boldsymbol{\theta}}}, a \sim \pi_{\boldsymbol{\theta}}(\cdot|s)} \left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) Q^{\pi_{\boldsymbol{\theta}}}(s, a) \right]. \end{aligned} \tag{1.21}$$

This rephrasing is know as REINFORCE trick [107] and gives the name to an algorithm which is a milestone of RL. In practical applications it can be inconvenient or impossible to compute the Q-function. One solution is to leverage simple function approximators belonging to the class of *compatible basis functions*, i.e., functions of the form $f_{\boldsymbol{\omega}} = \boldsymbol{\omega}^T \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s)$. Such approximators can replace $Q^{\pi_{\boldsymbol{\theta}}}(s, a)$ in Equation (1.21) for the calculation of the policy gradient as proven in [97].
A second solution that prevent from calculating $Q^{\pi_{\boldsymbol{\theta}}}(s, a)$ explicitly is to resort to a trajectory-based reformulation of Equation (1.21). By rephrasing the expected performance under policy $\pi_{\boldsymbol{\theta}}$ defined in Equation (1.6) as:

$$J(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\tau \sim p_{\boldsymbol{\theta}}} [\mathcal{R}(\tau)], \tag{1.22}$$

where $p_{\boldsymbol{\theta}}$ is the distribution over trajectories $\tau \in \mathcal{T}$ induced by the policy $\pi_{\boldsymbol{\theta}}$, and $\mathcal{R}(\tau) = \sum_{h=0}^{H-1} \gamma^h r_{h+1}$, we can write:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \int_{\mathcal{T}} \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\tau) \mathcal{R}(\tau) d\tau \tag{1.23}$$

$$= \mathop{\mathbb{E}}_{\tau \sim p_{\boldsymbol{\theta}}} \left[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) \mathcal{R}(\tau) \right] \tag{1.24}$$

This formulation of the PG turns out to be extremely practical. Not only because it prevents from the calculation of the Q-function, but also because it does not require any knowledge about the transition kernel of the MDP. In fact, $\log p_{\boldsymbol{\theta}}(\tau)$ can be easily derived from the sheer knowledge of the trajectory and definition of the policy:

$$\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) = \nabla_{\boldsymbol{\theta}} \log \left(\mu(s_{\tau,0}) \prod_{t=0}^{H-1} \pi_{\boldsymbol{\theta}}(a_{\tau,h}|s_{\tau,h}) P(a_{\tau,h+1}|s_{\tau,h+1}) \right) \tag{1.25}$$

$$= \sum_{h=0}^{H-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_{\tau,h}|s_{\tau,h}), \ \forall \tau \in \mathcal{T} \tag{1.26}$$

At this point, the careful reader might have noticed that we have not yet discussed the meaning of the $\boldsymbol{G}(\boldsymbol{\theta})$ in the parameters update Equation (1.20). The choice of this matrix represent the direction we want to take when performing a gradient update. The two most common direction choices in the RL literature are *steepest gradient ascent* and *natural gradient ascent*. In the first case, it is sufficient to set $\boldsymbol{G}(\boldsymbol{\theta}) = \boldsymbol{I}$. However, this option might be ineffective in many cases [5]:

- when there are large plateaus where the gradient is very small and does not point in the direction of the global optimum;
- when the performance function (or, in general, the loss function to optimize) is multimodal;
- when the gradient is not isotropic in magnitude w.r.t. any direction away from its maximum,creating troubles in the step size tuning.

In such cases, the *natural gradient* as been empirically proven to have faster convergence and to avoid premature convergence to local maxima. Natural

Algorithm 1.2 Generic policy gradient algorithm

1: **Input**: initial policy parameters $\boldsymbol{\theta}^0$, learning rate α
2: **Initialize** $t = 0$
3: **while** not converged **do**
4: Estimate $\boldsymbol{G}(\boldsymbol{\theta}^t)$ with an estimator $\widehat{\boldsymbol{G}}(\boldsymbol{\theta}^t)$
5: Estimate $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^t)$ with an estimator $\widehat{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^t)$
6: Update the policy parameters $\boldsymbol{\theta}^{t+1} \leftarrow \boldsymbol{\theta}^t + \alpha \widehat{\boldsymbol{G}}^{-1}(\boldsymbol{\theta}^t) \widehat{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^t)$
7: $t = t + 1$
8: **end while**
9: **return** an approximation of the optimal policy parameters $\boldsymbol{\theta}^*$

gradient considers the parameter space as a Riemann manifold equipped with its own norm $\|\boldsymbol{\theta}\|^2_{\boldsymbol{G}(\boldsymbol{\theta})} = \boldsymbol{\theta}^T \boldsymbol{G}(\boldsymbol{\theta}) \boldsymbol{\theta}$, which replaces the Euclidean norm, $\|\boldsymbol{\theta}\|^2_{\boldsymbol{I}} = \boldsymbol{\theta}^T \boldsymbol{\theta}$. Such parametric space is a Riemann manifold whose points are probability measures defined on a common probability space. Since $\boldsymbol{G}$ represents the local Riemann metric tensor, in this context it is given by the *Fisher Information Matrix*, i.e., $\boldsymbol{G}(\boldsymbol{\theta}) = \boldsymbol{F}(\boldsymbol{\theta})$:

$$\boldsymbol{F}(\boldsymbol{\theta} = \mathop{\mathbb{E}}_{\tau \sim p_{\boldsymbol{\theta}}} \left[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau)^T \right]. \tag{1.27}$$

For more details on the advantages of the natural gradient over the steepest ascent gradient we refer to [5, 4, 81].
We are now ready to present the general setup for policy gradient methods, showed in Algorithm (1.2).

1.4.3 Policy gradient estimation

As shown in the generic policy gradient algorithm, at each iteration an estimate of the policy gradient is required. Indeed, finding a good estimator is one of the main challenges of policy gradient methods. Since *RL* aims at solving MDPs whose model is unknown to the agent, the sole mean to estimate the policy gradient is by leveraging on the experience collected, i.e., the samples. Among several techniques proposed in the literature over the last years, the most prominent approaches to estimate the policy gradient from samples are *Finite-Difference* and *Likelihood Ratio* methods [41].

Finite differences

Finite difference methods aim at approximating the policy gradient as a quotient of finite increments. The idea is to perform multiple perturbations $\Delta\boldsymbol{\theta}_1, \Delta\boldsymbol{\theta}_2, \ldots, \Delta\boldsymbol{\theta}_N$ of the policy parameters. Then, with each set of perturbed parameters collect a number of sample trajectories in order to calculate $\Delta\hat{J}_i = J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_i) - J(\boldsymbol{\theta})$. At this point, the policy gradient can be estimated by regression as:

$$\hat{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = (\boldsymbol{\Delta\Theta}^T \boldsymbol{\Delta\Theta})^{-1} \boldsymbol{\Delta\hat{J}}, \tag{1.28}$$

where $\boldsymbol{\Delta\Theta} = [\Delta\boldsymbol{\theta}_1, \Delta\boldsymbol{\theta}_2, \ldots, \Delta\boldsymbol{\theta}_N]$ and $\boldsymbol{\Delta\hat{J}} = [\Delta\hat{J}_1, \Delta\hat{J}_2, \ldots, \Delta\hat{J}_N]$. This method is easy to implement and very efficient when applied to deterministic tasks or pseudo-random number simulations. However, it becomes useless in real control tasks, where a large number of trajectories is required and noise slows down convergence. Moreover, the choice of the perturbation of the parameters is a very difficult problem which may cause instability. For these reasons, the likelihood-ratio method is largely preferred in real control tasks.

Likelihood ratio

Likelihood ratio methods were among the first policy search methods introduced in the early 1990s by Williams [107], and include the famous REINFORCE algorithm. These methods build upon the policy gradient formulation given by Equation (1.24), which, in the same way of Equation (1.21), has been rewritten by applying the so called *REINFORCE trick*, also called *likelihood ratio trick*. As we have seen in Equation (1.26), the policy gradient (Equation (1.24)) can be approximated by using a sum over the sampled trajectories. In a similar way, we can estimate $\mathcal{R}(\tau)$ in a Monte-Carlo fashion. However, such estimates suffer of very large variance. The variance can be reduced by introducing a baseline b for the trajectory reward $\mathcal{R}(\tau)$, i.e.,

$$\hat{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, b) = \mathop{\mathbb{E}}_{\tau \sim p_{\boldsymbol{\theta}}} \left[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) \left(\mathcal{R}(\tau) - b\right)\right] \tag{1.29}$$

Remarkably, adding the baseline keep the estimator unbiased. the baseline can be chosen arbitrarily in order to minimize the variance. The variance-minimizing baseline is usually referred to as *optimal baseline*. Clearly, the optimal baseline depends on the specific likelihood gradient estimation adopted.

1.4.4 Algorithms

The most popular sample-based estimate of the gradient is undoubtedly REINFORCE [107], which uses the total return directly:

$$\hat{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, b)_{RF} = \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{h=0}^{H-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_{\tau_i,h} | s_{\tau_i,h}) \right) \left(\sum_{h=0}^{H-1} \gamma^h r_{\tau_i}^{h+1} - b \right), \quad (1.30)$$

based on N independent trajectories. The variance of such estimator is represented by a $m \times m$ covariance matrix, with m equals the number of parameters of the policy. We can obtain an optimal baseline in two ways. Either we minimize each element of the diagonal, obtaining a component-wise baseline made of m elements [81]. Or we can minimize the trace of the covariance matrix, obtaining a scalar baseline[109]:

$$b_{RF}^* = \frac{\mathop{\mathbb{E}}_{\tau \sim p_{\boldsymbol{\theta}}} \left[\left\| \sum_{h=0}^{H-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_{\tau,h} | s_{\tau,h}) \right\|_2^2 \mathcal{R}(\tau) \right]}{\mathop{\mathbb{E}}_{\tau \sim p_{\boldsymbol{\theta}}} \left[\left\| \sum_{h=0}^{H-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_{\tau,h} | s_{\tau,h}) \right\|_2^2 \right]} \quad (1.31)$$

$$(b_{RF}^*)_j = \frac{\mathop{\mathbb{E}}_{\tau \sim p_{\boldsymbol{\theta}}} \left[\left(\sum_{h=0}^{H-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_{\tau,h} | s_{\tau,h}) \right)^2 \mathcal{R}(\tau) \right]}{\mathop{\mathbb{E}}_{\tau \sim p_{\boldsymbol{\theta}}} \left[\left(\sum_{h=0}^{H-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_{\tau,h} | s_{\tau,h}) \right)^2 \right]}, \; j = 1, 2, ..., m. \quad (1.32)$$

As for the REINFORCE estimator (1.30), these optimal baselines are estimated simply by replacing the expectation with the empirical average. Algorithm (1.3) shows the full REINFORCE procedure with component-based optimal baseline estimation. For sake of clarity, we underline that this algorithm, as well as the others that we discuss in this section, is meant to carry out step (5) of the generic policy gradient algorithm, i.e., Algorithm (1.2).

Another solution tackling the high-variance problem of REINFORCE is to leverage on the intuitive observation that future actions do not depend on past rewards (unless policy changes take place continuously along the trajectory). Hence, we can derive two other well-know gradient estimators, Gradient of the average reward in Partially Observable MDPs (GPOMDP) [13] and PGT [97]:

Algorithm 1.3 Episodic REINFORCE with component-wise optimal baseline.

Input: policy parametrization $\boldsymbol{\theta}$
Initialize: $n = 0$
while not converged **do**
  collect a trajectory τ_n
  for every gradient component $j = 1, 2, \dots, m$ **do**
    Estimate the optimal baseline

$$b_j^n = \frac{\sum_{i=1}^{n} \left(\sum_{h=0}^{H-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_{\tau_i,h}|s_{\tau_i,h}) \right)^2 \left(\sum_{h=0}^{H-1} \gamma^h r_{\tau_i}^{h+1} \right)}{\sum_{i=1}^{n} \left(\sum_{h=0}^{H-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_{\tau_i,h}|s_{\tau_i,h}) \right)^2}$$

    Estimate the gradient element

$$\hat{\nabla}_{\boldsymbol{\theta}_j} J(\boldsymbol{\theta})^n = \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{h=0}^{H-1} \nabla_{\boldsymbol{\theta}_j} \log \pi_{\boldsymbol{\theta}}(a_{\tau_i,h}|s_{\tau_i,h}) \right) \left(\sum_{h=0}^{H-1} \gamma^h r_{\tau_i}^{h+1} - b_j^n \right)$$

  end for
  $n = n + 1$
end while
return gradient estimate $\hat{\nabla}_{\boldsymbol{\theta}_j} J(\boldsymbol{\theta})^N$

$$\hat{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, b)_{GPOMDP} = \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{h=0}^{H-1} \left(\sum_{h'=0}^{h} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_{\tau_i,h'}|s_{\tau_i,h'}) \right) \left(\gamma^h r_{\tau_i}^{h+1} - b_h \right) \right), \tag{1.33}$$

$$\hat{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, b)_{PGT} = \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{h=0}^{H-1} \gamma^h \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_{\tau_i,h}|s_{\tau_i,h}) \left(\sum_{h'=0}^{H-1} \gamma^{h'-h} r_{\tau_i}^{h'+1} - b_h \right) \right). \tag{1.34}$$

In [81] the authors show that these two estimators are equal, despite their apparent difference. In the same paper, the authors show how to derive the optimal baseline for GPOMDP, which, differently from the REINFORCE case, can be time dependent. In the case b = 0 and under mild assumptions, this gradient estimation has been proven to suffer from less variance than REINFORCE [109]. However, large variance remains a common trait of these Monte Carlo PG techniques. In fact, being trajectories generated by sampling at each time step

according to a stochastic policy $\pi_{\boldsymbol{\theta}}$, the estimators inherit the variance of the policy. To address this issue, in [89] 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. This allows the use of a deterministic controller $\pi_{\boldsymbol{\theta}} : \boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^m$ for sampling the trajectories, i.e., $\pi_{\boldsymbol{\theta}}(a|s) = \delta(a - \upsilon_{\boldsymbol{\theta}}(s))$, where $\upsilon_{\boldsymbol{\theta}}$ is a deterministic function of the state s, e.g., [90]. The policy parameters are sampled from a distribution $\nu_{\boldsymbol{\xi}} \in \delta(\Theta)$, called *hyper-policy*, where $\boldsymbol{\xi} \in \Xi \subseteq \mathbb{R}^d$ are the hyper-policy parameters, or *hyper-parameters*. Thus, in episodic PGPE it is sufficient to sample the parameters $\boldsymbol{\theta} \sim \nu_{\boldsymbol{\xi}}$ once at the beginning of the episode and then generate an entire trajectory following the deterministic policy $\pi_{\boldsymbol{\theta}}$, with a consequent reduction of the gradient estimate variance. As an additional benefit, the parameter gradient is estimated by direct parameter perturbations, without having to back-propagate any derivatives, which allows to use non-differentiable controllers. For what concern the hyper-policy parametrization choice, the most typical one is Gaussian with diagonal covariance matrix, as it happens in classical PG. Although the policy is deterministic, the exploration is guaranteed by the stochasticity of the hyper-policy. The hyper-parameters $\boldsymbol{\xi}$ are updated by following the gradient ascent direction of the gradient of the expected reward, which can be rewritten as:

$$J(\boldsymbol{\xi}) = \int_{\Theta} \int_{\mathcal{T}} \nu_{\boldsymbol{\xi}}(\boldsymbol{\theta}) p_{\boldsymbol{\theta}}(\tau) \mathcal{R}(\tau) d\tau d\boldsymbol{\theta}, \tag{1.35}$$

where $\nu_{\boldsymbol{\xi}}(\boldsymbol{\theta}) p_{\boldsymbol{\theta}}(\tau) = p_{\boldsymbol{\xi}}(\boldsymbol{\theta}, \tau)$ because τ is conditionally independent from $\boldsymbol{\xi}$ given $\boldsymbol{\theta}$. Applying the likelihood ratio technique, we obtain:

$$\nabla_{\boldsymbol{\xi}} J(\boldsymbol{\xi}) = \mathop{\mathbb{E}}_{\boldsymbol{\theta} \sim \nu_{\boldsymbol{\xi}}, \tau \sim p_{\boldsymbol{\theta}}} [\nabla_{\boldsymbol{\xi}} \log \nu_{\boldsymbol{\xi}}(\boldsymbol{\theta}) \mathcal{R}(\tau)] \tag{1.36}$$

In order to further reduce the estimate variance, we can adopt an estimator with baseline and compute the optimal one similarly to the REINFORCE case. The episodic PGPE algorithm is reported in Algorithm 1.4.

Algorithm 1.4 Episodic PGPE

Input: initial hyper-parameters $\boldsymbol{\xi}^0$, learning rate α
Initialize $t = 0$
while not converged **do**
    for $n = 1, 2, \ldots, N$ **do**
        Sample controller parameters $\boldsymbol{\theta}^{(n)} \sim \nu_{\boldsymbol{\xi}_t}$
        Sample trajectory $\tau \sim p_{\boldsymbol{\theta}}$ under policy $\pi_{\boldsymbol{\theta}}$
    end for
    Estimate optimal baseline b
    Estimate the hyper-policy gradient:

$$\hat{\nabla}_{\boldsymbol{\xi}} J(\boldsymbol{\xi}_t) = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\xi}} \log \nu_{\boldsymbol{\xi}}(\boldsymbol{\theta}^{(i)})(\mathcal{R}(\tau^{(i)}) - b)$$

    Update the hyper-parameters $\boldsymbol{\xi}_{t+1} = \boldsymbol{\xi}_t + \alpha \hat{\nabla}_{\boldsymbol{\xi}} J(\boldsymbol{\xi}_t)$
    $t = t + 1$
end while
return an approximation of the optimal policy parameters $\boldsymbol{\theta}^* \sim \nu_{\boldsymbol{\xi}*}$

Chapter 2

Exploration Techniques

In this chapter we present the state of the art of MABs and RL techniques that focus on the exploration challenge arising in these learning frameworks. While sketching a sort of taxonomy of the exploration techniques, we describe some of the most emblematic algorithms for each category. The exploration problem is the major goal of our research project, which has been inspired by many of the algorithms described in this chapter.
Following the taxonomy outlined by Sebatian Thrun in 1992 for the RL framework [99], we divide both MABs and RL exploration techniques in two families of exploration schemes: *undirected* and *directed* exploration. While the former family is closely related to random walk exploration, directed exploration techniques memorize exploration-specific knowledge which is used for guiding the exploration search. In many finite deterministic domains, any learning technique based on undirected exploration is inefficient in terms of learning time, i.e. learning time is expected to scale exponentially with the size of the state space [106]. For all these domains, reinforcement learning using a directed technique can always be performed in polynomial time, demonstrating the important role of exploration in reinforcement learning [99]. However, an important remark is that we do not address the problem of pure exploration, in which there is no price to be paid for exploration and the only objective is to find the reward-maximizing actions. Instead, we exploration in the more general context of learning described in the previous chapter (1.1.1), where the agent face the exploration-exploitation dilemma all along its journey.

2.1 Exploration in Multi Armed Bandits

2.1.1 Undirected Exploration

The most uninformed and basic way of exploring an unknown environment is to generate actions randomly with uniform probability. This method is often applied if exploration costs do not matter during learning. Sometimes tasks are divided into two phases, namely an exploration and an exploitation phase, and costs are not considered during the exploration phase. This is the case of the *explore-then-commit* algorithm (Algorithm 1.1) presented in the previous chapter, which explores uniformly the finite number of arms m before starting the exploitation phase, after mK steps. Another common algorithm with undirected uniform exploration is the ϵ-greedy algorithm [62]. One can think of it as an extension of the *explore-then-commit* algorithm which allows to continue exploring with probability ϵ even during the performance phase (i.e., the exploitation phase). The idea is, after a pure exploration phase of mK steps as for the $explore - then - commit$ algorithm, to exploit the estimated best arm $x_t = \arg\max \widehat{\mu}_i(t)$ with probability $(1 - \epsilon_t)$ at each time step, and explore uniformly the remaining set of arms with probability ϵ_t at each step.
Another approach to undirected exploration which is more effective when costs are relevant during learning is *non-uniform* exploration utilizing the current utility estimates to influence action-selection. The higher the expected utility by selecting action i, the more likely i gets selected. This ensures that the learning system explores and exploits simultaneously, as it happens for the ϵ-greedy algorithm, but in an intuitively more effective way since actions are not chosen uniformly during exploration. An example of utility-driven non-uniform undirected exploration is the Boltzmann-distributed exploration [24]. By defining the average reward of an arm as in Equation 1.3:

$$\widehat{\mu}_i(t) = \frac{1}{T_i(t)} \sum_{s=1}^{t} \mathbb{1}_{\{x_s=i\}} r_s, \tag{2.1}$$

the probability for an action i to get selected is a non-linear function of $\widehat{\mu}_i$:

$$Pr(i) = \frac{e^{\widehat{\mu}_i \tau^{-1}}}{\sum_{a \in \mathcal{X}} e^{\widehat{\mu}_a \tau^{-1}}}. \tag{2.2}$$

Here τ is a gain factor, often called *temperature*, which determines the amount of randomness in the action selection procedure. With $\tau \to 0$ pure exploitation is

approached and with $\tau \to \infty$ the resulting distribution approached the uniform distribution.
In the rest of this section, we present some techniques of *directed exploration*, which memorizes knowledge about the learning process itself and utilizes this knowledge for directing the exploration.

2.1.2 Count-based exploration and Upper Confidence Bound

Count-based exploration memorizes knowledge about the number of times $T_i(t)$ action i has been visited up to time-step t, for all $i \in \mathcal{X}$. Strategies based on counting usually evaluate actions by a linear combination of an exploitation term and an exploration term, called the *exploration bonus*, which is a (usually inverse) function of $T_i(t)$. Hence, action x_t is selected in a deterministic fashion as:

$$x_t = \arg\max_{x \in \mathcal{X}} \{\widehat{\mu}_x + bonus(T_x(t-1))\}. \tag{2.3}$$

Such strategy is at the core of the celebrated Upper Confidence Bound (UCB) algorithm [58, 2, 9] that overcomes all of the limitations of strategies based on exploration followed by commitment. The algorithm has many different forms, depending on the distributional assumptions on the noise. Here, we assume the noise is 1-subgaussian, i.e.,:

Definition 2.1.1. Subgaussianity *A random variable X is σ-subgaussian if, for all $\lambda \in \mathbb{R}$, it holds that $\mathbb{E}\exp(\lambda X) \leqslant \exp(\lambda^2\sigma^2/2)$.*

Coupling this property with the application of the *Markov's inequality*, we can prove that:

Theorem 2.1.1. *[62] If X is σ-subgaussian, then, for any $\epsilon \geqslant 0$,*

$$Pr(X \geqslant \epsilon) \leqslant \exp\left(-\frac{\epsilon^2}{2\sigma^2}\right)$$

From this theorem, we can derive a key corollary which gives us precious information about the concentration of the sample mean around the true mean of independent, σ-subgaussian random variables:

Corollary 2.1.1. *Assume that $X_i - \mu$, for $i = 1, 2, \ldots, n$ are independent, σ-subgaussian random variables. Then, for any $\epsilon \geqslant 0$, with probability at least $1 - \delta$,*

$\delta \in [0, 1]$*:*

$$\mu \leqslant \widehat{\mu} + \sqrt{\frac{2\sigma^2 \log(1/\delta)}{n}} \quad \textit{and} \quad \mu \geqslant \widehat{\mu} - \sqrt{\frac{2\sigma^2 \log(1/\delta)}{n}},$$

where $\widehat{\mu} = \frac{1}{n} \sum_{t=1}^{n} X_t$.

When considering its options in round t the learner has observed $T_i(t-1)$ samples from arm i and received rewards from that arm with an empirical mean of $\widehat{\mu}_i$. Then an *optimist* candidate for the unknown mean of the ith arm is:

$$UCB_i(t-1, \delta) = \widehat{\mu}_i(t-1) + \sqrt{\frac{2\log(1/\delta)}{T_i(t-1)}}. \tag{2.4}$$

Great care is required when comparing (2.1.1) and (2.4) because in the former the number of samples is the constant n, but in the latter it is a random variable $T_i(t-1)$. A part from this technicality, the intuition remains that δ is approximately an upper bound on the probability of the event that the above quantity is an underestimate of the true mean. In fact, by Corollary (2.1.1), for any $\delta \in [0, 1]$:

$$Pr\left(\mu \geqslant \widehat{\mu} + \sqrt{\frac{2\sigma^2 \log(1/\delta)}{n}}\right) \leqslant \delta. \tag{2.5}$$

After having picked all actions once, the UCB policy simply picks at each time step t the action i whose $UCB_i(t-1, \delta)$ index is maximum, as shown in Algorithm (2.1).

The value of $1 - \delta$ is called the confidence level and different choices lead to different algorithms, each with their pros and cons, and sometimes different analysis. For example, Auer et al. [9], present an algorithm originating in [2] called *UCB1* which achieves expected logarithmic regret uniformly over time, for all reward distributions, with no prior knowledge of the reward distribution. *UCB1* takes $\delta = 1/t$, where t is the current time step, resulting in:

$$UCB1_i(t, \delta) = \widehat{\mu}_i(t) + \sqrt{\frac{2\log(t)}{T_i(t)}} \tag{2.6}$$

Algorithm 2.1 UCB(δ) algorithm

Input: K arms and δ
Choose each action once
for $t = 1, 2, \ldots, T$ **do**
    if $t \leqslant K$ **then**
        $x_t = t$
    else
        $x_t = \arg\max_i UCB_i(t-1, \delta)$
    end if
end for

However, rather than considering 1-subgaussian environments, Auer et al. [9] considers bandits where the payoffs are confined to the $[0, 1]$ interval, which are ensured to be $1/2$-subgaussian.

2.1.3 Optimism in the Face of Uncertainty

The class of UCB algorithms is of particular relevance because represents a successful implementation of the principle of Optimism in the Face of Uncertainty (OFU), which is applicable to various exploration problems not only finite-armed stochastic bandits. The OFU principle states that one should choose their actions as if the environment is as nice as plausibly possible.
To illustrate the intuition imagine of being in the same situation depicted in (1.1.1), i.e.,, a standard lunch break from work. You can choose between your good old-fashioned favourite restaurant or sampling a restaurant that you never visited before. Taking an optimistic view of the unknown restaurant leads to exploration because without data it *could* be amazing. Then, after trying the new option a few times you can update your statistics about each choice and make a more informed decision. On the other hand, taking a pessimistic view of the new option discourages exploration and you may suffer significant regret if the local options are delicious. The UCB strategy assigns to each arm an upper confidence bound that with high probability is an overestimate of the unknown mean reward. The intuitive reason why this leads to sublinear regret is simple. Assuming the upper confidence bound assigned to the optimal arm is indeed an overestimate, then another arm can only be played if its upper confidence bound is larger than that of the optimal arm, which in turn is larger than the mean of the optimal arm. And yet this cannot happen too often because the additional

data provided by playing a suboptimal arm means that the upper confidence bound for this arm will eventually fall below that of the optimal arm.
The next algorithm that we discuss will be useful to better understand the OFU principle and its exploration power, with application to a more complex environment.

2.1.4 Hierarchical Optimistic Optimization

In 2011 Bubeck et al. [22] proposed a novel optimistic arm selection strategy whose regret improves upon previous results on continuum and Lipschitz bandits (e.g.,, [56, 54]), extending and generalizing the environment class of application to arbitrary topological spaces. In particular, the setting is that of $\mathcal{X}$-armed bandits, introduced in (1.1.3). This algorithm is particularly interesting to us for two reasons:

(i) it applies the OFU principle to continuous action spaces, as we wish to do in the context of RL Policy Search;

(ii) it leverages the correlation between arms (expressed as a dissimilarity function) to explore the environment more effectively.

The Hierarchical Optimistic Optimization (HOO) strategy assumes that the decision maker is able to cover the space of arms in a recursive manner, successively refining the regions in the covering such that the diameters of these sets shrink at a known geometric rate when measured with the dissimilarity metric. In particular, the authors define a tree of coverings as:

Definition 2.1.2. *(*Tree of coverings*) A tree of coverings is a family of measurable subsets* $(\mathcal{P}_{(h,i)})_{1\leqslant i\leqslant 2^h, h\geqslant 0}$ *of* $\mathcal{X}$ *such that for all fixed integer* $h \geqslant 0$*, the covering* $\bigcup_{1\leqslant i\leqslant 2^h} P_{h,i} = \mathcal{X}$ *holds. Moreover, the elements of the covering are obtained recursively: each subset* $P_{h,i}$ *is covered by the two subsets* $P_{h+1,2i-1}$ *and* $P_{h+1,2i}$*.*

Remark 2.1.1. A typical choice for the coverings in a cubic domain is to let the domains be hyper-rectangles. They can be obtained, e.g., in a dyadic manner, by splitting at each step hyper-rectangles in the middle along their longest side, in an axis parallel manner; if all sides are equal, we split them along the first axis.

The number of visits of a node (h, i) up to time-step T is given by the number of visits of every node belonging to $\mathcal{C}(h, i)$, i.e., the set of the node (h, i) and its descendants:

$$T_{h,i}(t) = \sum_{l=1}^{t} \mathbb{1}_{\{(h_l,i_l)\in\mathcal{C}(h,i)\}}, \tag{2.7}$$

where (h_l, i_l) are the coordinates of the node selected at time step l. Then, the empirical average of the rewards received for the time-points when the path followed the algorithm has gone through (h, i) is:

$$\widehat{\mu}_{h,i}(t) = \frac{1}{T_{h,i}(t)} \sum_{l=1}^{t} r_l \mathbb{1}_{\{(h_l,i_l)\in\mathcal{C}(h,i)\}}. \tag{2.8}$$

Thus, similarly to the $UCB(\delta)$ algorithm, an upper confidence bound can be defined as:

$$U_{h,i}(t) = \widehat{\mu}_{h,i}(t) + \sqrt{\frac{2\log t}{T_{h,i}(t)}} + \nu_1 \rho^h, \tag{2.9}$$

where $\rho \in (0, 1)$ and $\nu_1 \in \mathbb{R}_0^+$ are parameters of the algorithm. Along with the nodes the algorithm stores what the authors call B-values:

$$B_{h,i}(t) = \min\{U_{h,i}(t), \max\{B_{h+1,2i-1}(t), B_{h+1,2i}(t)\}\} \tag{2.10}$$

Finally, let us denote by $\mathcal{T}$ the infinite tree of coverings, by $\mathcal{T}_t$ the set of nodes of the tree that have been picked in previous rounds and by $\mathcal{S}_t$ the nodes which are not in $\mathcal{T}_t$. Now, for a node (h, i) in $\mathcal{S}_t$, we define its B-value to be $B_{h,i}(t) = +\infty$. We now have everything we need to present the full HOO strategy, i.e., Algorithm (2.2).

The definition and use of B-values, that put in relationship each node (subset of $X space$) with all its children (subset of the subset of $\mathcal{X}$), shows how the algorithm relies on the correlation between the reward of an arm and that of its neighbours. Indeed, the proof of the regret achieved by HOO relies on:

Algorithm 2.2 HOO algorithm

Input: the infinite tree of coverings $\mathcal{T}$, $\rho \in (0,1)$ and $\nu_1 \in \mathbb{R}_0^+$
Initialize: $t = 0$, $\mathcal{T}_0 = \varnothing$, $\mathcal{S}_0 = \mathcal{T}$, $B_{h,i}(t) = +\infty \ \forall (h,i) \in \mathcal{S}_0$
Choose the root node $(h_0, i_0) = (0,1)$ and update $t = t+1$
for $t = 1, 2, \ldots, T$ **do**
    Choose a node according to the deterministic rule:

$$(h_t, i_t) = \arg \max_{(h,i) \in \mathcal{S}_t} B_{h,i}(t) \tag{2.11}$$

    Choose, possibly at random, an arm $x_t \in \mathcal{P}_{(h_t,i_t)}$ and collect reward r_t
    Update $B_{h,i}(t)$ for (h_t, i_t) and all its parent nodes
    $\mathcal{T}_{t+1} = \mathcal{T}_t \cup (h_t, i_t)$, $\mathcal{S}_{t+1} = \mathcal{S}_t / (h_t, i_t)$
end for

Assumption 2.1.1. The mean-payoff function μ is weakly Lipschitz w.r.t. a dissimilarity metric ℓ, i.e., $\forall x_1, x_2 \in \mathcal{X}$,

$$\mu^* - \mu(x_2) \leqslant \mu^* - \mu(x_1) + \max\{\mu^* - \mu(x_1), \ell(x_1, x_2)\}. \tag{2.12}$$

Note that weak Lipschitzness is satisfied whenever μ is 1-Lipschitz, i.e., $\forall x_1, x_2 \in \mathcal{X}$, $|\mu(x_1) - \mu(x_2)| \leqslant \ell(x_1, x_2)$. On the other hand, weak Lipschitzness implies local (one-sided) 1-Lipschitzness at any maxima. Indeed, at an optimal arm x^*, Equation 2.12 rewrites to $\mu(x^*) - \mu(x_2) \leqslant \ell(x^*, x_2)$.
By carefully choosing the tree of coverings and the algorithm parameters, the authors show that the regret of the proposed approach is bounded by $\widetilde{\mathcal{O}}(\sqrt{T}) = \mathcal{O}(\log \sqrt{T})$. For more details, we remand to [22].

2.1.5 Posterior Sampling

Description To conclude with: Why is Posterior Sampling Better than Optimism for Reinforcement Learning? [74]

2.2 Exploration in Reinforcement Learning

2.2.1 Undirected Exploration

One may ask why you would want to randomize? The best answer is simplicity: The policy only depends on the exploration parameter ϵ rather than some com-

plicated schedule. This can be useful in complicated settings like reinforcement learning where many instances of the same algorithm are acting simultaneously and communication is costly. In Chapter 11 we'll see the role of randomization when the bandit itself is allowed to react to the actions of the learner in a malicious way. The history of ϵ-greedy is unclear, but it is a popular and widely used and known algorithm in reinforcement learning [Sutton and Barto, 1998]. Auer et al. [2002a] analyze the regret of ϵ-greedy with slowly decreasing ϵ (see Exercise 6.7). There are other kinds of randomized exploration as well, including Thompson sampling [1933] and Boltzmann exploration analyzed recently by Cesa-Bianchi et al. [2017].

Algorithm:

- Soft Q-learning [43]

2.2.2 Counter-based exploration and OFU

Description.
Algorithms:

- GPUCB [92]
- PixelCNN algorithm from Count-based exploration with neural density models [76]. This paper builds upon Unifying Count-Based Exploration and Intrinsic Motivation [14].

2.2.3 Value Difference and Recency-based exploration

Brief description, citing:

- Value-Difference based Exploration: AdaptiveControl between epsilon-Greedy and Softmax [100]
- Efficient Exploration In Reinforcement Learning [99] refers to recency-based exploraiton.

2.2.4 Intrinsic Motivation

Description, including: Unifying Count-Based Exploration and Intrinsic Motivation [14], talks about the connection between intrinsic motivation and counter-based exploration

Algorithms:

- Vime: Variational information maximizing exploration [46]
- Diversity-Inducing Policy Gradient[64], similarly to us, uses an exploration bonus based on diversity between distributions

2.2.5 Thompson Sampling

Description To conclude with: Why is Posterior Sampling Better than Optimism for Reinforcement Learning? [74]

Bibliography

[1] Abbasi-Yadkori, Y., Pál, D., and Szepesvári, C. Improved algorithms for linear stochastic bandits. In *Advances in Neural Information Processing Systems* (2011), pp. 2312–2320.

[2] Agrawal, R. The continuum-armed bandit problem. *SIAM Journal on Control and Optimization 33*, 6 (1995), 1926–1951.

[3] Agrawal, R. Sample mean based index policies by o (log n) regret for the multi-armed bandit problem. *Advances in Applied Probability 27*, 4 (1995), 1054–1078.

[4] Amari, S.-I. Natural gradient works efficiently in learning. *Neural computation 10*, 2 (1998), 251–276.

[5] Amari, S.-I., and Douglas, S. C. Why natural gradient? In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'98 (Cat. No. 98CH36181)* (1998), vol. 2, IEEE, pp. 1213–1216.

[6] Antos, A., Szepesvári, C., and Munos, R. Fitted q-iteration in continuous action-space mdps. In *Advances in neural information processing systems* (2008), pp. 9–16.

[7] Atan, O., Tekin, C., and Schaar, M. Global multi-armed bandits with hölder continuity. In *Artificial Intelligence and Statistics* (2015), pp. 28–36.

[8] Auer, P. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research 3*, Nov (2002), 397–422.

[9] Auer, P., Cesa-Bianchi, N., and Fischer, P. Finite-time analysis of the multiarmed bandit problem. *Machine learning 47*, 2-3 (2002), 235–256.

[10] Auer, P., Ortner, R., and Szepesvári, C. Improved rates for the stochastic continuum-armed bandit problem. In *International Conference on Computational Learning Theory* (2007), Springer, pp. 454–468.

[11] Bach, F. R., and Blei, D. M., Eds. *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015* (2015), vol. 37 of *JMLR Workshop and Conference Proceedings*, JMLR.org.

[12] Baird III, L. C. Advantage updating. Tech. rep., WRIGHT LAB WRIGHT-PATTERSON AFB OH, 1993.

[13] Baxter, J., and Bartlett, P. L. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research 15* (2001), 319–350.

[14] Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems* (2016), pp. 1471–1479.

[15] Bengio, S., Wallach, H. M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., Eds. *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada* (2018).

[16] Boucheron, S., Lugosi, G., and Massart, P. *Concentration inequalities: A nonasymptotic theory of independence*. Oxford university press, 2013.

[17] Brafman, R. I., and Tennenholtz, M. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research 3*, Oct (2002), 213–231.

[18] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016.

[19] Bubeck, S., Cesa-Bianchi, N., et al. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends® in Machine Learning 5*, 1 (2012), 1–122.

[20] Bubeck, S., Cesa-Bianchi, N., and Lugosi, G. Bandits with heavy tail. *IEEE Transactions on Information Theory 59*, 11 (2013), 7711–7717.

[21] Bubeck, S., and Munos, R. Open loop optimistic planning. In *COLT 2010 - The 23rd Conference on Learning Theory, Haifa, Israel, June 27-29, 2010* (2010), pp. 477–489.

[22] Bubeck, S., Munos, R., Stoltz, G., and Szepesvári, C. X-armed bandits. *Journal of Machine Learning Research 12*, May (2011), 1655–1695.

[23] Bubeck, S., Stoltz, G., Szepesvári, C., and Munos, R. Online optimization in x-armed bandits. In *Advances in Neural Information Processing Systems* (2009), pp. 201–208.

[24] Cesa-Bianchi, N., Gentile, C., Lugosi, G., and Neu, G. Boltzmann exploration done right. In *Advances in Neural Information Processing Systems* (2017), pp. 6284–6293.

[25] Cesa-Bianchi, N., and Lugosi, G. Combinatorial bandits. *Journal of Computer and System Sciences 78*, 5 (2012), 1404–1422.

[26] Chapelle, O., and Li, L. An empirical evaluation of thompson sampling. In *Advances in neural information processing systems* (2011), pp. 2249–2257.

[27] Chen, W., Wang, Y., Yuan, Y., and Wang, Q. Combinatorial multi-armed bandit and its extension to probabilistically triggered arms. *The Journal of Machine Learning Research 17*, 1 (2016), 1746–1778.

[28] Chentanez, N., Barto, A. G., and Singh, S. P. Intrinsically motivated reinforcement learning. In *Advances in neural information processing systems* (2005), pp. 1281–1288.

[29] Choshen, L., Fox, L., and Loewenstein, Y. Dora the explorer: Directed outreaching reinforcement action-selection. *arXiv preprint arXiv:1804.04012* (2018).

[30] Chowdhury, S. R., and Gopalan, A. Online learning in kernelized markov decision processes. *CoRR abs/1805.08052* (2018).

[31] Cochran, W. G. *Sampling techniques*. John Wiley & Sons, 2007.

[32] Cortes, C., Mansour, Y., and Mohri, M. Learning bounds for importance weighting. In *Advances in Neural Information Processing Systems 23*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, Eds. Curran Associates, Inc., 2010, pp. 442–450.

[33] Dann, C., and Brunskill, E. Sample complexity of episodic fixed-horizon reinforcement learning. In *Advances in Neural Information Processing Systems* (2015), pp. 2818–2826.

[34] Dann, C., Lattimore, T., and Brunskill, E. Unifying pac and regret: Uniform pac bounds for episodic reinforcement learning. In *Advances in Neural Information Processing Systems* (2017), pp. 5713–5723.

[35] Degris, T., White, M., and Sutton, R. S. Off-policy actor-critic. *arXiv preprint arXiv:1205.4839* (2012).

[36] Deisenroth, M. P., Neumann, G., Peters, J., et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics 2*, 1–2 (2013), 1–142.

[37] Dorato, P., Abdallah, C. T., Cerone, V., and Jacobson, D. H. *Linear-quadratic control: an introduction*. Prentice Hall Englewood Cliffs, NJ, 1995.

[38] Dy, J. G., and Krause, A., Eds. *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018* (2018), vol. 80 of *JMLR Workshop and Conference Proceedings*, JMLR.org.

[39] Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., and Kavukcuoglu, K. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018* (2018), pp. 1406–1415.

[40] Gil, M., Alajaji, F., and Linder, T. Rényi divergence measures for commonly used univariate continuous distributions. *Information Sciences 249* (2013), 124–131.

[41] Glynn, P. W. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM 33*, 10 (1990), 75–84.

[42] Grondman, I., Busoniu, L., Lopes, G. A., and Babuska, R. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 42*, 6 (2012), 1291–1307.

[43] Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. Reinforcement learning with deep energy-based policies. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017* (2017), pp. 1352–1361.

[44] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018* (2018), pp. 1856–1865.

[45] Hershey, J. R., and Olsen, P. A. Approximating the kullback leibler divergence between gaussian mixture models. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on* (2007), vol. 4, IEEE, pp. IV–317.

[46] Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems* (2016), pp. 1109–1117.

[47] Ionides, E. L. Truncated importance sampling. *Journal of Computational and Graphical Statistics 17*, 2 (2008), 295–311.

[48] Jaksch, T., Ortner, R., and Auer, P. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research 11*, Apr (2010), 1563–1600.

[49] Jin, C., Allen-Zhu, Z., Bubeck, S., and Jordan, M. I. Is q-learning provably efficient? In *Advances in Neural Information Processing Systems* (2018), pp. 4868–4878.

[50] Kalai, A. T., and Mohri, M., Eds. *COLT 2010 - The 23rd Conference on Learning Theory, Haifa, Israel, June 27-29, 2010* (2010), Omnipress.

[51] Kallus, N. Instrument-armed bandits. In *Algorithmic Learning Theory* (2018), pp. 529–546.

[52] Kaufmann, E., Korda, N., and Munos, R. Thompson sampling: An asymptotically optimal finite-time analysis. In *International Conference on Algorithmic Learning Theory* (2012), Springer, pp. 199–213.

[53] Kearns, M., and Singh, S. Near-optimal reinforcement learning in polynomial time. *Machine learning 49*, 2-3 (2002), 209–232.

[54] Kleinberg, R., Slivkins, A., and Upfal, E. Multi-armed bandits in metric spaces. In *Proceedings of the fortieth annual ACM symposium on Theory of computing* (2008), ACM, pp. 681–690.

[55] Kleinberg, R., Slivkins, A., and Upfal, E. Bandits and experts in metric spaces. *arXiv preprint arXiv:1312.1277* (2013).

[56] Kleinberg, R. D. Nearly tight bounds for the continuum-armed bandit problem. In *Advances in Neural Information Processing Systems* (2005), pp. 697–704.

[57] Kupcsik, A. G., Deisenroth, M. P., Peters, J., and Neumann, G. Data-efficient generalization of robot skills with contextual policy search. In *Twenty-Seventh AAAI Conference on Artificial Intelligence* (2013).

[58] Lai, T. L., and Robbins, H. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics 6*, 1 (1985), 4–22.

[59] Lakshmanan, K., Ortner, R., and Ryabko, D. Improved regret bounds for undiscounted continuous reinforcement learning. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015* (2015), pp. 524–532.

[60] Lange, S., Gabel, T., and Riedmiller, M. Batch reinforcement learning. In *Reinforcement learning*. Springer, 2012, pp. 45–73.

[61] Lattimore, T., and Hutter, M. Near-optimal pac bounds for discounted mdps. *Theoretical Computer Science 558* (2014), 125–143.

[62] Lattimore, T., and Szepesvári, C. *Bandit Algorithms*. Cambridge University Press (preprint), 2019.

[63] Magureanu, S., Combes, R., and Proutiere, A. Lipschitz bandits: Regret lower bounds and optimal algorithms. *arXiv preprint arXiv:1405.4758* (2014).

[64] Masood, M. A., and Doshi-Velez, F. Diversity-inducing policy gradient: Using mmd to find a set of policies that are diverse in terms of state-visitation.

[65] McCluskey, L., Williams, B. C., Silva, J. R., and Bonet, B., Eds. *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012* (2012), AAAI.

[66] Medina, A. M., and Yang, S. No-regret algorithms for heavy-tailed linear bandits. In *Proceedings of The 33rd International Conference on Machine Learning* (New York, New York, USA, 20–22 Jun 2016), M. F. Balcan and K. Q. Weinberger, Eds., vol. 48 of *Proceedings of Machine Learning Research*, PMLR, pp. 1642–1650.

[67] Mersereau, A. J., Rusmevichientong, P., and Tsitsiklis, J. N. A structured multiarmed bandit problem and the greedy policy. *IEEE Transactions on Automatic Control 54*, 12 (2009), 2787–2802.

[68] Metelli, A. M., Papini, M., Faccio, F., and Restelli, M. Policy optimization via importance sampling. In *Advances in Neural Information Processing Systems* (2018), pp. 5447–5459.

[69] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016* (2016), pp. 1928–1937.

[70] Moré, J. J., and Thuente, D. J. Line search algorithms with guaranteed sufficient decrease. *ACM Transactions on Mathematical Software (TOMS) 20*, 3 (1994), 286–307.

[71] Ok, J., Proutière, A., and Tranos, D. Exploration in structured reinforcement learning. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.* (2018), pp. 8888–8896.

[72] Ortner, R., and Ryabko, D. Online regret bounds for undiscounted continuous reinforcement learning. In *Proceedings of the 25th International Conference on Neural Information Processing Systems-Volume 2* (2012), Curran Associates Inc., pp. 1763–1771.

[73] Osband, I., Russo, D., and Roy, B. V. (more) efficient reinforcement learning via posterior sampling. In *Advances in Neural Information*

Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States. (2013), pp. 3003–3011.

[74] Osband, I., and Van Roy, B. Why is posterior sampling better than optimism for reinforcement learning? In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (2017), JMLR. org, pp. 2701–2710.

[75] Osband, I., Van Roy, B., and Wen, Z. Generalization and exploration via randomized value functions. In *International Conference on Machine Learning* (2016), pp. 2377–2386.

[76] Ostrovski, G., Bellemare, M. G., van den Oord, A., and Munos, R. Count-based exploration with neural density models. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (2017), JMLR. org, pp. 2721–2730.

[77] Owen, A., and Zhou, Y. Safe and Effective Importance Sampling. *Journal of the American Statistical Association* (Mar. 2000), 135–143.

[78] Owen, A. B. *Monte Carlo theory, methods and examples.* 2013.

[79] Owen, A. B. *Monte Carlo theory, methods and examples.* 2013.

[80] Pandey, S., Chakrabarti, D., and Agarwal, D. Multi-armed bandit problems with dependent arms. In *Proceedings of the 24th international conference on Machine learning* (2007), ACM, pp. 721–728.

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

[82] Precup, D., and Teh, Y. W., Eds. *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017* (2017), vol. 70 of *Proceedings of Machine Learning Research*, PMLR.

[83] Puterman, M. L. *Markov decision processes: discrete stochastic dynamic programming.* John Wiley & Sons, 2014.

[84] Rényi, A. On measures of entropy and information. Tech. rep., Hungarian Academy of Sciences Budapest Hungary, 1961.

[85] Robbins, H. Some aspects of the sequential design of experiments. In *Herbert Robbins Selected Papers*. Springer, 1985, pp. 169–177.

[86] Rummery, G. A., and Niranjan, M. *On-line Q-learning using connectionist systems*, vol. 37. University of Cambridge, Department of Engineering Cambridge, England, 1994.

[87] Saritaç, A. Ö., and Tekin, C. Combinatorial multi-armed bandit problem with probabilistically triggered arms: A case with bounded regret. In *Signal and Information Processing (GlobalSIP), 2017 IEEE Global Conference on* (2017), IEEE, pp. 111–115.

[88] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International Conference on Machine Learning* (2015), pp. 1889–1897.

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

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

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

[92] Srinivas, N., Krause, A., Kakade, S., and Seeger, M. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)* (Haifa, Israel, June 2010), J. Fürnkranz and T. Joachims, Eds., Omnipress, pp. 1015–1022.

[93] Strehl, A. L., Li, L., and Littman, M. L. Reinforcement learning in finite mdps: Pac analysis. *Journal of Machine Learning Research 10*, Nov (2009), 2413–2444.

[94] Sutton, R. S. Learning to predict by the methods of temporal differences. *Machine learning 3*, 1 (1988), 9–44.

[95] Sutton, R. S. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin 2*, 4 (1991), 160–163.

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

[97] Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems* (2000), pp. 1057–1063.

[98] Thompson, W. R. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika 25*, 3/4 (1933), 285–294.

[99] Thrun, S. B. Efficient exploration in reinforcement learning.

[100] Tokic, M., and Palm, G. Value-difference based exploration: adaptive control between epsilon-greedy and softmax. In *Annual Conference on Artificial Intelligence* (2011), Springer, pp. 335–346.

[101] Vakili, S., Liu, K., and Zhao, Q. Deterministic sequencing of exploration and exploitation for multi-armed bandit problems. *arXiv preprint arXiv:1106.6104* (2011).

[102] Veach, E., and Guibas, L. J. Optimally combining sampling techniques for Monte Carlo rendering. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques - SIGGRAPH '95* (1995), ACM Press, pp. 419–428.

[103] Wang, Z., Zhou, R., and Shen, C. Regional multi-armed bandits. In *International Conference on Artificial Intelligence and Statistics* (2018), pp. 510–518.

[104] Watkins, C. J. C. H. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

[105] Weinstein, A., and Littman, M. L. Bandit-based planning and learning in continuous-action markov decision processes. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012* (2012).

[106] Whitehead, S. D., and Ballard, D. H. *A study of cooperative mechanisms for faster reinforcement learning*. University of Rochester, Department of Computer Science Rochester, NY, 1991.

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

[108] Yu, X., Shao, H., Lyu, M. R., and King, I. Pure exploration of multi-armed bandits with heavy-tailed payoffs.

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