Reinforcement Learning

Policy Gradient (Finite Horizon)

Lecturer: *Matteo Papini* version May 24, 2026

Consider a (stationary) finite-horizon MDP with state space $\mathcal{S}$, **continuous action space** $\mathcal{A} \subseteq \mathbb{R}^{d_{\mathcal{A}}}$, transition kernel $\{p(\cdot|s,a) : s \in \mathcal{S}, a \in \mathcal{A}\}$, reward function $r : \mathcal{S} \times \mathcal{A} \to [-1,1]$, starting-state distribution p_0, and horizon $T \in \mathbb{N}$. We will consider stationary (stochastic Markovian) policies $\pi : \mathcal{S} \to \Delta_{\mathcal{A}}$ for simplicity, although the optimal policy for a finite-horizon MDP is typically non-stationary. Also, assume $r_T(s) = 0$ for $s \in \mathcal{S}$ (no terminal rewards).

While value-function approximation allows dealing with large and continuous *state* spaces, continuous actions require some sort of **policy approximation**. The simplest form of policy approximation is *action discretization*: divide $\mathcal{A}$ into a finite number K of bins and define a new, finite action space $\mathcal{A}' = [K]$ by defining a (state-dependent) representative action for each bin. This approach clearly suffers from the curse of dimensionality. However, the important thing to notice here is that discretizing the action space restricts the space of available (stochastic Markovian stationary) policies.

In general, consider a policy space or *policy class* $\Pi \subseteq \Delta_{\mathcal{A}}^{\mathcal{S}}$, where $\Delta_{\mathcal{A}}^{\mathcal{S}}$ denotes the space of all mappings from states to action distributions. Let $J : \Pi \to \mathbb{R}$ denote the expected return or *performance function*:

$$J(\pi) = \mathbb{E}\left[\sum_{t=0}^{T-1} r(S_t, A_t)\right],$$

where $S_0 \sim p_0$, $A_t \sim \pi(\cdot|S_t)$, and $S_{t+1} \sim p(\cdot|S_t, A_t)$ for $t = 0, \dots, T-1$. Let $\pi^* \in \arg\max_{\pi \in \Pi} J(\pi)$ denote the *best-in-class policy*, which may differ from the optimal policy of the MDP when Π is a proper subset of the space of all policies. Reinforcement learning algorithms that aim to find the best-in-class policy for a given policy space are called **policy search** or policy optimization algorithms. Typical policy spaces are parametric:

$$\Pi_\Theta = \left\{\pi_{\boldsymbol{\theta}} \in \Delta_{\mathcal{A}}^{\mathcal{S}} | \boldsymbol{\theta} \in \Theta\right\},$$

where $\Theta \subseteq \mathbb{R}^d$ is the parameter space, and each parameter vector $\boldsymbol{\theta} \in \Theta$ corresponds to a policy. A common example is that of policies parametrized by neural networks, where parameters are network weights. The mapping from parameter vectors to policies is not necessarily injective nor surjective. Since we will focus on parametric policies, we redefine the performance functions as $J(\boldsymbol{\theta}) = J(\pi_{\boldsymbol{\theta}})$. This is not just an abbreviation, as we are interested in $J : \Theta \to \mathbb{R}$ specifically as a function of parameters. Policy optimization reduces to the problem of finding the *optimal parameters* $\boldsymbol{\theta}^* \in \arg\max_{\boldsymbol{\theta} \in \Theta} J(\boldsymbol{\theta})$. Unfortunately, J is a nonconvex function in general. Nonconvexity can arise both from the structure of the policy optimization problem itself and from the parametrization. For simplicity, we consider $\Theta = \mathbb{R}^d$ in the following (unconstrained optimization).

Example 1 (Gaussian policies) *Let $\mathcal{A} = \mathbb{R}$ (scalar continuous actions) and let $\mu_{\boldsymbol{\theta}} : \mathcal{S} \to \mathbb{R}$ be a family of functions parametrized by $\boldsymbol{\theta} \in \mathbb{R}^d$. We can define a class of stochastic policies for*

this action space by specifying the probability density function (pdf) for each state and parameter vectors:

$$\pi_{\boldsymbol{\theta}}(a|s) = \mathcal{N}(a; \mu_{\boldsymbol{\theta}}(s), \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(a - \mu_{\boldsymbol{\theta}}(s))^2\right),$$

where $\mathcal{N}$ denotes the pdf of the Gaussian distribution and $\sigma > 0$ is a (fixed) standard deviation. We call $\mu_{\boldsymbol{\theta}}$ the mean-action function. We shall see some concrete examples in the practical sessions. Gaussian policies can be easily generalized to multi-dimensional actions and learnable covariance matrices.

Example 2 (Softmax Policies) *Although parametric policies find their natural application in continuous action spaces, we can easily define a parametric policy class over a finite action $\mathcal{A}$ by specifying the probability mass function for each state and parameter vector:*

$$\pi_{\boldsymbol{\theta}}(a|s) = \text{Softmax}(\text{q}_{\boldsymbol{\theta}}(\text{s}, \text{a})) = \frac{\exp(q_{\boldsymbol{\theta}}(s,a))}{\sum_{a' \in \mathcal{A}} \exp(q_{\boldsymbol{\theta}}(s,a'))},$$

where $q_{\boldsymbol{\theta}} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, called the logit function, assigns a preference to each action for the given state.

Policy Gradient (PG) algorithms aim to solve the policy optimization problem by (stochastic) *gradient ascent* over policy parameters. Consider a class Π_Θ of stochastic (fully mixed) *differentiable* (with respect to parameters) policies. In particular, assume the following exists for all $s, a, \boldsymbol{\theta}$:

$$\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) \in \mathbb{R}^d.$$

The latter is called **score function** or eligibility vector. For example, the score function for the Gaussian policy, assuming the action-mean function is differentiable, is

$$\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) = \frac{a - \mu_{\boldsymbol{\theta}}(s)}{\sigma^2} \nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(s),$$

and the score function of Softmax policies with differentiable logits is:

$$\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) = \nabla_{\boldsymbol{\theta}} q_{\boldsymbol{\theta}}(s,a) - \mathbb{E}_{A \sim \pi_{\boldsymbol{\theta}}(\cdot|s)} \left[\nabla_{\boldsymbol{\theta}} q_{\boldsymbol{\theta}}(s, A)\right].$$

Under mild regularity assumptions, we can define the gradient of the performance function, $\nabla_{\boldsymbol{\theta}} J$, often called *policy gradient*. A simple policy gradient scheme is the following: starting from some initial parameters $\boldsymbol{\theta}_0$, iterate the following parameter update:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_k),$$

for $k = 0, 1, \ldots$, where $\alpha > 0$ is a *step size* or learning rate. Under mild regularity assumptions and a sufficiently small step size, this can be shown to converge, in the sense that $\|\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_k)\| \to 0$ as $k \to \infty$. The result might be a local maximum (or even a minimum, or a saddle point). Anyway, computing the gradient of the performance function requires full knowledge of the MDP. In the following, we will see how to estimate the policy gradient from on-policy data in a model-free fashion.

REINFORCE. Denote by $\tau = (S_0, A_0, \ldots, S_{T-1}, A_{T-1}, S_T)$ a *trajectory*, that is the sequence of states and actions observed during an episode. Every policy π induces a distribution over trajectories, whose pdf can be written as follows:

$$p_\pi(\tau) = p_0(S_0)\pi(A_0|S_0)p(S_1|S_0, A_0)\cdots p(S_T|S_{T-1}, A_{T-1}) = p_0(S_0)\prod_{t=0}^{T-1}\pi(A_t|S_t)p(S_{t+1}|S_t, A_t).$$

For parametric policies, we abbreviate $p_{\pi_{\boldsymbol{\theta}}}(\tau)$ as $p_{\boldsymbol{\theta}}(\tau)$. The *return* of a trajectory is defined as:

$$R(\tau) = \sum_{t=0}^{T-1} r(S_t, A_t).$$

It is easy to see that the performance function (or expected return) can be written as follows:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\tau\sim p_{\boldsymbol{\theta}}}\left[R(\tau)\right].$$

Theorem 3 (Finite-Horizon Policy Gradient Theorem)

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\tau\sim p_{\boldsymbol{\theta}}}\left[\left(\sum_{t=0}^{T-1}\nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(A_t|S_t)\right)R(\tau)\right].$$

Proof. We prove this and other results from this chapter for the case in which $\mathcal{S}$ and $\mathcal{A}$ are both finite. Let $\mathcal{T}$ denote the set of all possible trajectories of length T, which is also finite. The results generalize to continuous spaces under some regularity assumptions. Here and in the following, we omit the gradient subscript when clear from context.

$$\begin{aligned}
\nabla J(\boldsymbol{\theta}) &= \nabla\mathbb{E}_{\tau\sim p_{\boldsymbol{\theta}}}\left[R(\tau)\right] \\
&= \nabla\sum_{\tau\in\mathcal{T}} p_{\boldsymbol{\theta}}(\tau)R(\tau) \\
&= \sum_{\tau\in\mathcal{T}}\nabla p_{\boldsymbol{\theta}}(\tau)R(\tau) \\
&= \sum_{\tau\in\mathcal{T}} p_{\boldsymbol{\theta}}(\tau)\nabla\log p_{\boldsymbol{\theta}}(\tau)R(\tau) \\
&= \mathbb{E}_{\tau\sim p_{\boldsymbol{\theta}}}\left[\nabla\log p_{\boldsymbol{\theta}}(\tau)R(\tau)\right],
\end{aligned}$$

where in the fourth equality we used the *log trick* or likelihood-ratio trick: since $\nabla_x\log f(x) = \nabla f(x)/x$, $\nabla_x f(x) = x\nabla_x\log f(x)$ whenever $\nabla_x\log f(x)$ is defined. The purpose of this was to rewrite the policy gradient as an expectation under the same trajectory distribution as the performance function. Finally:

$$\begin{aligned}
\nabla\log p_{\boldsymbol{\theta}}(\tau) &= \nabla\log\left(p_0(S_0)\prod_{t=1}^{T-1}\pi_{\boldsymbol{\theta}}(A_t|S_t)p(S_{t+1}|S_t, A_t)\right) \\
&= \nabla\left(\log p_0(S_0) + \sum_{t=0}^{T-1}\log\pi_{\boldsymbol{\theta}}(A_t|S_t) + \sum_{t=0}^{T-1}\log p(S_{t+1}|S_t, A_t)\right) \\
&= \sum_{t=0}^{T-1}\nabla\log\pi_{\boldsymbol{\theta}}(A_t|S_t),
\end{aligned}$$

since all the terms that do not depend on policy parameters vanish. □

This expression for the policy gradient allows us to immediately define an unbiased *Monte Carlo* gradient estimator, called the **REINFORCE estimator**. Given a trajectory $\tau = (S_0, A_0, \dots, S_T)$ sampled from $p_{\boldsymbol{\theta}}$:

$$g(\boldsymbol{\theta}; \tau) = \left(\sum_{t=0}^{T-1} \nabla \log \pi_{\boldsymbol{\theta}}(A_t|S_t) \right) R(\tau).$$

This estimator has three important properties: (i) it is an unbiased estimate of the policy gradient, that is, $\mathbb{E}_{\tau \sim p_{\boldsymbol{\theta}}}[g(\boldsymbol{\theta}; \tau)] = \nabla J(\boldsymbol{\theta})$ when $\tau \sim p_{\boldsymbol{\theta}}$, as is obvious from the previous theorem; (ii) it can be computed entirely from the observed states, actions, and rewards, without prior knowledge of the MDP; (iii) the data are naturally obtained by the interaction of the agent with the environment, as playing $\pi_{\boldsymbol{\theta}}$ produces trajectories distributed according to $p_{\boldsymbol{\theta}}$. It is a common practice to aggregate multiple trajectories from the same policy to reduce the *variance* of the estimate:

$$\widehat{\nabla} J(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} g(\boldsymbol{\theta}; \tau_i),$$

where $\tau^1, \dots \tau^n$ are n i.i.d. trajectories from $p_{\boldsymbol{\theta}}$ and n is called *batch size*. These can be obtained from n independent episodes of interaction, which may be simulated in parallel. The averaged estimator is clearly also unbiased, but its variance is smaller by a factor $1/n$. The **REINFORCE algorithm** performs *stochastic* gradient ascent using the REINFORCE estimator. Variants of REINFORCE constitute a family of algorithms known as **actor-only policy gradient algorithms**, where the "actor" is the policy. The name refers to the fact that these algorithms only employ policy approximation, without any value-function approximation, while methods employing both value-function and policy approximation are called "actor-critic" (covered in the next lectures).

Algorithm 1 REINFORCE

Input: Initial policy parameters $\boldsymbol{\theta}_0$, step size α, batch size N.

1: **for** $k = 1, 2, \dots$ **do**
2: Collect a batch of n independent trajectories $\tau^1, \dots, \tau^n$ with $\pi_{\boldsymbol{\theta}_k}$
3: Compute

$$\widehat{\nabla} J(\boldsymbol{\theta}_k) = \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{t=0}^{T-1} \nabla \log \pi_{\boldsymbol{\theta}}(A_t^i|S_t^i) \right) R(\tau^i)$$

4: Update policy parameters as $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k + \alpha \widehat{\nabla} J(\boldsymbol{\theta}_k)$
5: **end for**

Common heuristics from supervised learning, such as the ADAM optimizer, can be employed in practice, especially when policies are parametrized by deep neural networks. The training loop can be stopped after a fixed number K of iterations, or when the norm of the stochastic gradient falls below some small threshold. REINFORCE can be shown to converge *on average* using diminishing step sizes.

Pros and cons of policy gradient algorithms. Compared to other families of RL algorithms, the main advantages of policy gradient algorithms are listed below. We refer to methods relying

primarily on value-function approximation (such as DQN) as **value-based** as opposed to **policy-based** methods such as actor-only *and* actor-critic policy gradient algorithms.

1. Naturally support continuous actions.
2. Well-understood (local) convergence guarantees even if function approximation is involved.
3. Robustness to noise: compared to value-based methods, actions (action distributions) change less abruptly when the states or the parameters are perturbed.[1] This typically yields more stable learning.
4. Naturally support stochastic policies, which are necessary in partially observable and strategic environments, and provide a minimal amount of exploration.
5. Suffer less from violations of the Markov assumption since it is not exploited directly (at least by actor-only algorithms).
6. Easy to incorporate domain knowledge: the policy space can be designed to only include behaviors relevant to the application (e.g., controllers with a small number of tunable parameters).
7. Safety: the policy space can be designed to exclude unsafe behaviors.

Some disadvantages are:

1. Variance: gradient estimators tend to have high variance, which makes the convergence slow and requires large amounts of simulation data. We will see some variance-reduction techniques in the following.
2. Bias: the policy space may not include good policies if it is not designed carefully.
3. Only convergence to local optima is guaranteed in general. Heuristics from nonconvex optimization such as random restarts can be employed. Global convergence can be ensured in some special cases (e.g., Linear Quadratic Regulator).
4. Do not naturally support deterministic policies. This can be a problem for safety-critical applications. Deterministic variants exist.

Variance Reduction (Basic Techniques). We now cover some basic techniques to reduce the variance of the REINFORCE estimator while keeping it unbiased. We will use the fact that scores are zero-mean conditional on the state:

$$\mathbb{E}\left[\nabla \log \pi_{\boldsymbol{\theta}}(A|s)|S\right] = \sum_{a\in\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|S)\nabla \log \pi_{\boldsymbol{\theta}}(a|S) = \sum_{a\in\mathcal{A}} \nabla \pi_{\boldsymbol{\theta}}(a|S) = \nabla \sum_{a\in\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|S) = \nabla 1 = \mathbf{0},$$

[1] Consider a (deterministic) greedy policy $\pi_{\boldsymbol{\theta}}(s) = \arg\max Q_{\boldsymbol{\theta}}(s,a)$, where $Q_{\boldsymbol{\theta}}$ is some parametric approximation of Q^*. If we perturb the state or the parameters even slightly, the selected action might be completely different.

when $A \sim \pi_{\boldsymbol{\theta}}(\cdot|S)$. This implies that $\mathbb{E}[\nabla \log p_{\boldsymbol{\theta}}(\tau)] = \mathbf{0}$ when $\tau \sim p_{\boldsymbol{\theta}}$, since:

$$\begin{aligned}\mathbb{E}\left[\nabla \log p_{\boldsymbol{\theta}}(\tau)\right] &= \mathbb{E}\left[\sum_{t=0}^{T-1} \nabla \log \pi_{\boldsymbol{\theta}}(A_t|S_t)\right] \\ &= \sum_{t=0}^{T-1} \mathbb{E}\left[\nabla \log \pi_{\boldsymbol{\theta}}(A_t|S_t)\right] \\ &= \sum_{t=0}^{T-1} \mathbb{E}\left[\mathbb{E}\left[\nabla \log \pi_{\boldsymbol{\theta}}(A_t|S_t)|S_t\right]\right] = \mathbf{0},\end{aligned}$$

where the third equality is by the *tower rule* or "law of conditional expectation": $E[X] = E[E[X|Y]]$ for any random variables X, Y (not necessarily independent).

Besides using larger batch sizes, adding a **baseline** to the REINFORCE estimator is the simplest way to reduce the variance without introducing any bias. Let $b \in \mathbb{R}$ be any constant (independent of the batch of trajectories used to estimate the gradient). Consider the following **REINFORCE estimator with baseline**:

$$\widehat{\nabla} J(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{t=0}^{T-1} \nabla \log \pi_{\boldsymbol{\theta}}(A_t^i|S_t^i) \right) \left(R(\tau^i) - b\right).$$

Theorem 4 *The REINFORCE estimator with baseline is unbiased.*

Proof.

$$\begin{aligned}\mathbb{E}\left[\widehat{\nabla} J(\boldsymbol{\theta})\right] &= \mathbb{E}\left[\frac{1}{n} \sum_{i=1}^{n} \nabla \log p_{\boldsymbol{\theta}}(\tau^i) \left(R(\tau^i) - b\right)\right] \\ &= \mathbb{E}_{\tau \sim p_{\boldsymbol{\theta}}}\left[\nabla \log p_{\boldsymbol{\theta}}(\tau) \left(R(\tau) - b\right)\right] \\ &= \mathbb{E}_{\tau \sim p_{\boldsymbol{\theta}}}\left[\nabla \log p_{\boldsymbol{\theta}}(\tau) R(\tau)\right] - b \mathbb{E}_{\tau \sim p_{\boldsymbol{\theta}}}\left[\nabla \log p_{\boldsymbol{\theta}}(\tau)\right] \\ &= \nabla J(\boldsymbol{\theta}) - \mathbf{0}.\end{aligned}$$

□

A common choice of baseline is the *average baseline*, which tends to reduce variance by centering the returns:

$$b = \frac{1}{n} \sum_{i=1}^{n} R(\tau^i).$$

In principle, this baseline should be computed from a separate, independent batch of trajectories. However, in practice, it is usually computed on the same batch of trajectories used for the gradient estimator. Common practice suggests that the bias is negligible if compared to the speedup granted by variance reduction and does not justify the cost of collecting extra trajectories.

A more subtle but still simple way to reduce variance is to exploit the temporal structure of the RL problem. Note that, in REINFORCE, $R(\tau)$ could be an arbitrary (possibly non-additive) function

of the trajectory, such as $R(\tau) = \max_{t=0,\dots T-1}\{r(S_t, A_t)\}$. Indeed, REINFORCE can be used for non-Markovian processes and applications other than RL. On the downside, when applied to policy optimization in MDPs, it does not exploit the causal structure of trajectories at all. The **GPOMDP estimator**, on the contrary, exploits the Markov assumption to remove some terms that do not contribute to the expectation, but may be a source of variance:

$$\widehat{\nabla} J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}\sum_{t=0}^{T-1} \nabla \log \pi_{\boldsymbol{\theta}}(A_t^i|S_t^i) \sum_{h=t}^{T-1} r(S_h^i, A_h^i) = \frac{1}{n}\sum_{i=1}^{n}\sum_{t=0}^{T-1} r(S_t^i, A_t^i) \sum_{h=0}^{t} \nabla \log \pi_{\boldsymbol{\theta}}(A_h^i|S_h^i).$$

That the two expressions are equivalent can be checked by changing the order of summation. We will refer to them as the *first* and *second form* of GPOMDP.

Theorem 5 *The GPOMDP estimator is unbiased.*

Proof. By the finite-horizon policy gradient theorem:

$$\begin{aligned}
\nabla J(\boldsymbol{\theta}) &= \mathbb{E}\left[\left(\sum_{t=0}^{T-1} r(S_t, A_t)\right)\left(\sum_{t=0}^{T-1} \nabla \log \pi_{\boldsymbol{\theta}}(A_t|S_t)\right)\right] \\
&= \mathbb{E}\left[\sum_{t=0}^{T-1}\left(r(S_t, A_t)\sum_{h=0}^{T-1} \nabla \log \pi_{\boldsymbol{\theta}}(A_h|S_h)\right)\right] \\
&= \mathbb{E}\left[\sum_{t=0}^{T-1}\left(r(S_t, A_t)\sum_{h=0}^{t} \nabla \log \pi_{\boldsymbol{\theta}}(A_h|S_h)\right)\right] + \mathbb{E}\left[\sum_{t=0}^{T-1}\left(r(S_t, A_t)\sum_{h=t+1}^{T-1} \nabla \log \pi_{\boldsymbol{\theta}}(A_h|S_h)\right)\right].
\end{aligned}$$

If we can show that the second term is zero, then the first term shows that GPOMDP (better seen in the second form) is an unbiased Monte Carlo estimator of the policy gradient. Indeed:

$$\begin{aligned}
&\mathbb{E}\left[\sum_{t=0}^{T-1}\left(r(S_t, A_t)\sum_{h=t+1}^{T-1} \nabla \log \pi_{\boldsymbol{\theta}}(A_h|S_h)\right)\right] \\
&\quad= \mathbb{E}\left[\sum_{t=0}^{T-1}\sum_{h=t+1}^{T-1} \mathbb{E}\left[(r(S_t, A_t)\nabla \log \pi_{\boldsymbol{\theta}}(A_h|S_h))|S_0, A_0, \dots S_h\right]\right] \\
&\quad= \mathbb{E}\left[\sum_{t=0}^{T-1} r(S_t, A_t)\sum_{h=t+1}^{T-1} \mathbb{E}\left[(\nabla \log \pi_{\boldsymbol{\theta}}(A_h|S_h))|S_0, A_0, \dots S_h\right]\right] \\
&\quad= \mathbb{E}\left[\sum_{t=0}^{T-1} r(S_t, A_t)\sum_{h=t+1}^{T-1} \mathbb{E}\left[(\nabla \log \pi_{\boldsymbol{\theta}}(A_h|S_h))|S_h\right]\right] \\
&\quad= \mathbf{0}.
\end{aligned}$$

We have used: in the first equality, linearity and the tower rule; in the second one, the fact that $r(S_t, A_t)$ is entirely determined by $S_0, \dots, S_h$, since $h > t$, and can be moved out of the conditional expectation; in the third one, the Markov property; in the last one, the fact that the score function is zero-mean conditional on the state. □

Baselines can be added also to the GPOMDP estimator to further reduce variance, and can be made time-dependent:

$$\widehat{\nabla} J(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \sum_{t=0}^{T-1} \left(r(S_t^i, A_t^i) - b_t \right) \sum_{h=0}^{t} \nabla \log \pi_{\boldsymbol{\theta}}(A_h^i | S_h^i).$$

In this case the average baseline is computed on rewards rather than returns:

$$b_t = \frac{1}{n} \sum_{i=1}^{n} r(S_t^i, A_t^i).$$

Natural Policy Gradient.* Parametric policy classes make policy updates *indirect*: we modify parameters to obtain an intended behavior. In most cases, the Euclidean metric in parameter space (standard distance in $\mathbb{R}^d$) does not reflect the entity of corresponding changes in policy space. For small displacements $\boldsymbol{h} \in \mathbb{R}^{\boldsymbol{d}}$, the **Fisher metric** provides distances in parameter space that are more representative of the difference between the corresponding policies:

$$d_F(\boldsymbol{\theta}, \boldsymbol{\theta} + \boldsymbol{h}) = \|\boldsymbol{h}\|_{F(\boldsymbol{\theta})} = \sqrt{\boldsymbol{h}^\top F(\boldsymbol{\theta}) \boldsymbol{h}},$$

where $F(\boldsymbol{\theta}) \in \mathbb{R}^{d \times d}$ is the (trajectory-wise) **Fisher information matrix** of policy $\pi_{\boldsymbol{\theta}}$:

$$F(\boldsymbol{\theta}) = \mathbb{E}_{\tau \sim p_{\boldsymbol{\theta}}} \left[\nabla \log p_{\boldsymbol{\theta}}(\tau) \nabla \log p_{\boldsymbol{\theta}}(\tau)^\top \right],$$

a symmetric positive semi-definite matrix. Note that the Fisher metric is *local*: the distance between two points depends on where these points are located in parameter space. That is why, in most cases, it is only meaningful for small parameter displacements.

The (squared) Fisher metric provides an approximation of the *statistical distance* between the trajectory distributions induced by the corresponding policies (proof omitted):

$$D_{\mathrm{KL}}(p_{\boldsymbol{\theta}+\boldsymbol{h}} \| p_{\boldsymbol{\theta}}) = \frac{1}{2} d_F(\boldsymbol{\theta}, \boldsymbol{\theta} + \boldsymbol{h})^2 + o(\|\boldsymbol{h}\|^3),$$

where D_{KL} is the **Kullback-Leibler divergence** or *relative entropy* of two distributions:

$$D_{\mathrm{KL}}(p \| q) = \sum_{\tau \in \mathcal{T}} p(\tau) \log \frac{p(\tau)}{q(\tau)}.$$

This shows that the Fisher metric is (approximately) invariant to reparametrization: displacements under different but equivalent (that is, inducing the same behavior) parametrizations are measured in the same way. We can make this precise for affine parameter transformations:

Theorem 6 *Consider a parametric policy class $\{\pi_{\boldsymbol{\theta}} \,|\, \boldsymbol{\theta} \in \mathbb{R}^d\}$ and a change of coordinates $\boldsymbol{\theta} = A\boldsymbol{\omega} + \boldsymbol{b}$, where $A \in \mathbb{R}^d$ is an invertible matrix and $\boldsymbol{b} \in \mathbb{R}^d$. Then, if $\boldsymbol{\theta'} = A\boldsymbol{\omega'} + \boldsymbol{b}$:*

$$d_F(\boldsymbol{\omega}, \boldsymbol{\omega'}) = d_F(\boldsymbol{\theta}, \boldsymbol{\theta'}).$$

PROOF. First, notice that $p_{\boldsymbol{\theta}} = p_{\boldsymbol{\omega}}$ since $\boldsymbol{\theta}$ and ω correspond to the same policy, hence induce the same trajectory distribution. By the chain rule:

$$\begin{aligned}\nabla_{\boldsymbol{\omega}} \log p_{\boldsymbol{\omega}}(\tau) &= \nabla_{\boldsymbol{\omega}} \log p_{\boldsymbol{\theta}}(\tau) \\ &= \left(\frac{\partial \boldsymbol{\theta}}{\partial \boldsymbol{\omega}}\right)^\top \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) \\ &= A^\top \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau).\end{aligned}$$

Then:

$$\begin{aligned}F(\boldsymbol{\omega}) &= \mathbb{E}_{\tau \sim p_{\boldsymbol{\omega}}}\left[\nabla_{\boldsymbol{\omega}} \log p_{\boldsymbol{\omega}}(\tau) \nabla_{\boldsymbol{\omega}} \log p_{\boldsymbol{\omega}}(\tau)^\top\right] \\ &= \mathbb{E}_{\tau \sim p_{\boldsymbol{\theta}}}\left[\nabla_{\boldsymbol{\omega}} \log p_{\boldsymbol{\omega}}(\tau) \nabla_{\boldsymbol{\omega}} \log p_{\boldsymbol{\omega}}(\tau)^\top\right] \\ &= \mathbb{E}_{\tau \sim p_{\boldsymbol{\theta}}}\left[A^\top \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau)^\top A\right] \\ &= A^\top F(\boldsymbol{\theta}) A.\end{aligned}$$

Finally, since $\boldsymbol{\omega}' - \boldsymbol{\omega} = A^{-1}(\boldsymbol{\theta}' - \boldsymbol{b}) - A^{-1}(\boldsymbol{\theta} - \boldsymbol{b}) = A^{-1}(\boldsymbol{\theta}' - \boldsymbol{\theta})$:

$$\begin{aligned}d_F(\boldsymbol{\omega}, \boldsymbol{\omega}') &= \sqrt{(\boldsymbol{\omega}' - \boldsymbol{\omega})^\top F(\boldsymbol{\omega})(\boldsymbol{\omega}' - \boldsymbol{\omega})} \\ &= \sqrt{(\boldsymbol{\omega}' - \boldsymbol{\omega})^\top A^\top F(\boldsymbol{\theta}) A (\boldsymbol{\omega}' - \boldsymbol{\omega})} \\ &= \sqrt{(AA^{-1}(\boldsymbol{\theta}' - \boldsymbol{\theta}))^\top F(\boldsymbol{\theta}) AA^{-1}(\boldsymbol{\theta}' - \boldsymbol{\theta})} \\ &= \sqrt{(\boldsymbol{\theta}' - \boldsymbol{\theta})^\top F(\boldsymbol{\theta})(\boldsymbol{\theta}' - \boldsymbol{\theta})} \\ &= d_F(\boldsymbol{\theta}, \boldsymbol{\theta}').\end{aligned}$$

□

Note that the two parametrizations define exactly the same policy space, so it is *natural* to measure distances in the same way under the two parametrizations. In contrast, the Euclidean metric would give:

$$\|\boldsymbol{\omega} - \boldsymbol{\omega}'\| = \|A^{-1}(\boldsymbol{\theta}' - \boldsymbol{\theta})\| = \sqrt{(\boldsymbol{\theta}' - \boldsymbol{\theta})^\top A^{-\top} A^{-1}(\boldsymbol{\theta}' - \boldsymbol{\theta})} = \|\boldsymbol{\theta}' - \boldsymbol{\theta}\|_{(AA^\top)^{-1}},$$

which is *not* the same as $\|\boldsymbol{\theta}' - \boldsymbol{\theta}\|$ unless A is an orthogonal matrix ($AA^\top = A^\top A = I$), that is, unless the change of coordinates is an *isometry*.

Recall that the gradient gives the *steepest ascent* direction according to the Euclidean metric. The **natural gradient** gives the steepest ascent direction according to the Fisher metric (proof omitted), and is obtained by *preconditioning* the gradient with the inverse Fisher information matrix:

$$\widetilde{\nabla} J(\boldsymbol{\theta}) = F(\boldsymbol{\theta})^{-1} \nabla J(\boldsymbol{\theta}).$$

An (exact) **natural policy gradient** update is then:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha \widetilde{\nabla} J(\boldsymbol{\theta}_k).$$

It has two important properties, which are easily obtained from the properties of the Fisher metric we established above:

1. It allows to control the statistical distance between the updated policy and the original one. Namely, if $\boldsymbol{h} = \alpha\widetilde{\nabla} J(\boldsymbol{\theta})$ and $\alpha = \epsilon/\|\widetilde{\nabla} J(\boldsymbol{\theta})\|_{F(\boldsymbol{\theta})}$,

$$D_{\mathrm{KL}}(p_{\boldsymbol{\theta}} \| p_{\boldsymbol{\theta}+\boldsymbol{h}}) = \frac{\epsilon^2}{2} + o(\|\boldsymbol{h}\|^3).$$

   This is used to make sure the behavior of the agent does not change abruptly when updating the policy parameters with a small step size.

2. Invariance to parametrization. Consider a policy class $\{\pi_{\boldsymbol{\theta}} \,|\, \boldsymbol{\theta} \in \mathbb{R}^d\}$ and an invertible change of coordinates $\boldsymbol{\omega} = A\boldsymbol{\theta} + \boldsymbol{b}$. Let $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha\widetilde{\nabla} J(\boldsymbol{\theta}_k)$ and $\boldsymbol{\omega}_{k+1} = \boldsymbol{\omega}_k + \alpha\widetilde{\nabla} J(\boldsymbol{\omega}_k)$. If $p_{\boldsymbol{\theta}_k} = p_{\boldsymbol{\omega}_k}$, then $p_{\boldsymbol{\theta}_{k+1}} = p_{\boldsymbol{\omega}_{k+1}}$, hence also $J(\boldsymbol{\theta}_{k+1}) = J(\boldsymbol{\omega}_{k+1})$. The learning process is not affected by an affine reparametrization.

Practical implementations of natural policy gradient incur statistical and computational challenges: the Fisher information matrix must also be estimated from data, and inverting it may be computationally expensive for policies with many parameters.