Policy Gradients

Lecture 4

Peter Henderson

COS 435 / ECE 433

Thanks to helpful slides/notes by Ben Van Roy, Emma Brunskill, Ben Eysenbach, and Csaba Szepesvári.

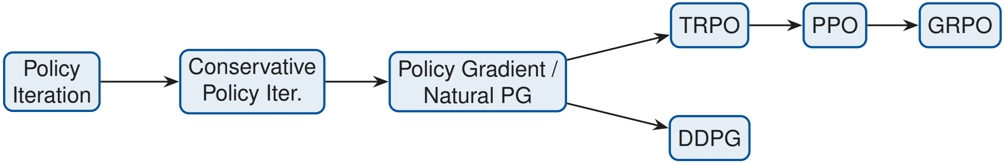
Project Deadlines

1a. **Project proposal:** 11:59pm on March 20, 2025, posted publicly on Ed.
    1-page PDF submitted on Ed using provided LaTeX template.
1b. **1:1 meeting with instructors:** 11:59pm on Mar 31 (late: April 7, 2025).
    10 min conversation with a TA during office hours. Don't wait until the last minute!
1c. **Peer feedback on proposal:** 11:59pm on Mar 27, 2025, submitted on Ed.
    Each group member independently posts $\sim$1 paragraph on another group's proposal ($<$ 3 comments).

2. **One experimental figure:** 11:59pm on April 24, 2025, added to shared slide deck.
   High-quality figure with title, axis labels, legend, caption explaining what it depicts and what to learn.
3. **Student presentations:** 3.5 min, last week of class.
   Prepared in shared slide deck. Randomly assigned to a slot the last week of class.
4. **Final written report and code:** May 5 at 10:30pm, 8 pages (excl. references)

Roadmap for This Lecture and Next

Goal: Give you a history and tools to see how modern policy gradient methods came to be. Understand what policy gradient methods are.



Lecture 5 will focus on understanding **advantage functions** and **baselines**, and their role in **bias-variance trade-offs**.

Review: Q-Values vs. Policies

What Is a Value Function?

A **value function** maps states (or state-action pairs) to *real numbers*.

Example: 3-state chain $\rightarrow$ goal on the right. Actions: $\{$Left, Right$\}$.

$Q^\pi(s,a)$	Left	Right	$V^\pi(s)$
s_1 (far)	1.2	$\mathbf{4.8}$	3.0
s_2 (near)	3.1	$\mathbf{7.5}$	5.3
s_3 (goal)	0.0	0.0	0.0

$Q^\pi(s_2, \text{Right}) = 7.5$ means: *"In s_2, take Right, then follow π — expected return is 7.5."*

State value: $V^\pi(s) = \sum_a \pi(a|s)\, Q^\pi(s,a)$ (weighted average over actions)

Advantage: $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$ (how much *better* than average?)

What Is a Policy?

A **policy** $\pi(a \mid s)$ is a *probability distribution* over actions, given a state.

Same 3-state chain example:

$\pi(a \mid s)$	$P(\text{Left})$	$P(\text{Right})$
s_1	0.1	0.9
s_2	0.2	0.8
s_3	0.5	0.5

Each row sums to 1. $\pi(\text{Right} \mid s_1) = 0.9$ means: *"In s_1, go Right 90% of the time."*

Q-values are real numbers that tell you *"how good is this action?"* while a policy gives probabilities that tell you *"what should I do?"* Value-based RL learns Q, then derives π via $\arg\max$. Policy gradient methods learn π **directly**.

When Do We Need to Learn Policies Directly? A couple of examples.

1. **Stochastic optimal policies:**
2. **Continuous action spaces:**

Review: The RL Objective

Review: The RL Objective (1/3)

We want to find a policy π_θ that maximizes the expected discounted return:

$$\max_{\pi_\theta} \mathbb{E}_{s_0 \sim p(s_0),\, a_t \sim \pi_\theta(\cdot|s_t),\, s_{t+1} \sim p(\cdot|s_t, a_t)} \left[\sum_t \gamma^t r(s_t, a_t) \right]$$

Three distributions interact inside this expectation:

$$\underbrace{s_0 \sim p(s_0)}_{\text{initial state distribution}}, \qquad \underbrace{a_t \sim \pi_\theta(\cdot \mid s_t)}_{\text{policy chooses action}}, \qquad \underbrace{s_{t+1} \sim p(\cdot \mid s_t, a_t)}_{\text{environment transitions}}$$

This is sometimes called the **policy optimization** or **policy search** problem.

Review: The RL Objective (2/3) — Trajectories

For simplicity, let $\tau = (s_0, a_0, s_1, a_1, \ldots)$ denote a **trajectory** — the full sequence of states and actions.

The distribution over trajectories under policy π_θ factors as:

$$\pi_\theta(\tau) = \underbrace{p(s_0)}_{\text{initial state}} \prod_{t=0}^{T} \underbrace{p(s_{t+1} \mid s_t, a_t)}_{\text{dynamics}} \underbrace{\pi_\theta(a_t \mid s_t)}_{\text{policy}}$$

Notice that the **dynamics** $p(s_{t+1} \mid s_t, a_t)$ do not depend on θ — only the policy terms do. This will be important when we take gradients.

Review: The RL Objective (3/3) — Compact Form

Define the **discounted return** of a trajectory:

$$R(\tau) = \sum_{t=0}^{T} \gamma^t r(s_t, a_t)$$

Then our objective has the compact form:

$$\max_\theta \; \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[R(\tau) \right]$$

$R(\tau)$ is the return of a single trajectory. Its expectation over trajectories gives $V(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$, which is what we want to maximize.

The Policy Gradient

Gradient Ascent on the RL Objective

Now, let’s think about doing **gradient ascent** on the RL objective.

Gradients are derivatives, and they point towards increasing values of the function. If we know the gradient $\nabla_\theta f(\theta)$, we could optimize by iterating: $\theta \leftarrow \theta + \eta \nabla_\theta f(\theta)$.

Gradient ascent is a **first-order method** because it uses the gradients (first-derivatives).

To apply gradient ascent to the RL problem, we need to compute:

$$\nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[R(\tau)]$$

Supervised Learning vs. RL: A Key Difference

Before going further, note an important difference from supervised learning.

In **supervised learning**, you sample data from a fixed dataset (e.g., $(x, y) \sim \mathcal{D}$) and then maximize the likelihood:

$$\nabla_\theta \mathbb{E}_{(x,y)\sim\mathcal{D}}[\log p_\theta(y \mid x)]$$

The key difference is that in **supervised learning**, the expectation is over a *fixed* distribution (dataset).

In **reinforcement learning**, the distribution we're taking the expectation over depends on θ! When we change the policy, we change the data distribution itself. This means we'll have to be a bit careful when computing this derivative.

A Common Mistake

Many students attempt to compute the derivative of the policy gradient using the following, which is incorrect:

$$\nabla_\theta \mathbb{E}_{\pi_\theta(\tau)}[R(\tau)] = \mathbb{E}_{\pi_\theta(\tau)}[\nabla_\theta R(\tau)] \overset{?}{=} 0 \qquad \text{(Wrong!)}$$

The mistake here is that we **cannot push the gradient operator inside the expectation** because the expectation's distribution depends on θ, the variable that we are trying to take the gradient of.

The correct approach: Write out the expectation as an integral, which will make clear the dependence on θ.

Deriving the Policy Gradient (Step 1)

We want to compute:

$$\nabla_\theta \mathbb{E}_{\pi_\theta(\tau)}[R(\tau)]$$

Step 1: Write out the expectation as an integral to make the dependence on θ explicit:

$$\nabla_\theta \mathbb{E}_{\pi_\theta(\tau)}[R(\tau)] = \nabla_\theta \int R(\tau)\, \pi_\theta(\tau)\, d\tau$$

Step 2: Push the gradient inside the integral.
This is allowed because the *bounds of integration* don't depend on θ — only the integrand does:

$$= \int R(\tau) \nabla_\theta \pi_\theta(\tau)\, d\tau$$

Deriving the Policy Gradient (Step 2: The Log-Derivative Trick)

We have: $\nabla_\theta \mathbb{E}_{\pi_\theta(\tau)}[R(\tau)] = \int R(\tau)\, \nabla_\theta \pi_\theta(\tau)\, d\tau$

Step 3: Apply the **log-derivative trick.**
Remember from calculus:
$\nabla_\theta \log \pi_\theta(\tau) = \frac{1}{\pi_\theta(\tau)} \nabla_\theta \pi_\theta(\tau) \Rightarrow \nabla_\theta \pi_\theta(\tau) = \pi_\theta(\tau)\, \nabla_\theta \log \pi_\theta(\tau)$. Substituting:

$$= \int R(\tau)\, \pi_\theta(\tau)\, \nabla_\theta \log \pi_\theta(\tau)\, d\tau$$

Why the Log-Derivative Trick Matters

The gradient is now expressed as an *expectation* under π_θ:

$$\boxed{\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[R(\tau)] = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[R(\tau)\,\nabla_\theta \log \pi_\theta(\tau)\right]}$$

Which can be approximated with Monte Carlo samples:

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] \approx \frac{1}{K}\sum_{k=1}^{K} R(\tau_k)\,\nabla_\theta \log \pi_\theta(\tau_k)$$

where $\tau_1, \ldots, \tau_K$ are trajectories sampled from the current policy π_θ.

The Policy Gradient Theorem

Without the log-derivative trick, $\int R(\tau)\nabla_\theta \pi_\theta(\tau)\, d\tau$ would not be an expectation we could sample from.

If we naively tried to approximate the integral with samples:

$$\int R(\tau)\nabla_\theta \pi_\theta(\tau)\, d\tau \approx \frac{1}{K}\sum_{k=1}^{K} R(\tau_k)\nabla_\theta \pi_\theta(\tau_k) \quad \text{(Incorrect.)}$$

where $\tau_k \sim \pi_\theta(\tau)$. This is not a valid Monte Carlo expectation estimate. Note the missing $\pi_\theta(\tau)$ in the integrand as compared to the approximatable version:

$$\int R(\tau)\, \pi_\theta(\tau)\, \nabla_\theta \log \pi_\theta(\tau)\, d\tau$$

Breaking Up the Trajectory: Log-Probability

Our next step is to express the gradient in terms of **individual actions**, rather than entire trajectories.

The log-probability of a trajectory factors as:

$$\log \pi_\theta(\tau) = \log p(s_0) + \sum_{t=0}^{T} \Big(\log p(s_{t+1} \mid s_t, a_t) + \log \pi_\theta(a_t \mid s_t) \Big)$$

Notice: $p(s_0)$ and $p(s_{t+1} \mid s_t, a_t)$ are the **dynamics** — they don't depend on θ.

Breaking Up the Trajectory: Taking the Gradient

Taking the gradient w.r.t. θ, the dynamics terms vanish:

$$\nabla_\theta \log \pi_\theta(\tau) = \cancelto{0}{\nabla_\theta \log p(s_0)} + \sum_{t=0}^{T} \Big(\cancelto{0}{\nabla_\theta \log p(s_{t+1} \mid s_t, a_t)} + \nabla_\theta \log \pi_\theta(a_t \mid s_t) \Big)$$

So we are left with:

$$\nabla_\theta \log \pi_\theta(\tau) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t)$$

The dynamics model **cancels out.** We can compute the policy gradient **without knowing the dynamics** — we only need the policy $\pi_\theta(a_t \mid s_t)$.

Derivation (1/7): Putting these altogether

We want $\nabla_\theta J(\theta)$ where $J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$. Start by pushing the gradient inside:

$$\nabla_\theta J(\theta) = \int \underbrace{\nabla_\theta \pi_\theta(\tau)}_{\text{gradient of trajectory prob.}} \quad \underbrace{R(\tau)}_{\text{return of trajectory } \tau} \quad d\tau$$

Multiply and divide by $\pi_\theta(\tau)$ to prepare the log-derivative trick:

$$= \int \pi_\theta(\tau) \left(\frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} \right) R(\tau)\, d\tau$$

Explainer: The integrand is unchanged; we are just rewriting it so the next step turns the integral into an expectation.

Derivation (2/7): Log-Derivative Trick $\Rightarrow$ Expectation

Use $\nabla_\theta \log \pi_\theta(\tau) = \frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)}$, so the gradient becomes an expectation:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[\underbrace{\nabla_\theta \log \pi_\theta(\tau)}_{\text{score / log-prob gradient}}\ \underbrace{R(\tau)}_{\text{return weight}}\right]$$

Explainer: We can now estimate this with Monte Carlo: sample trajectories $\tau \sim \pi_\theta$, compute $R(\tau)\nabla_\theta \log \pi_\theta(\tau)$ for each, and average.

Derivation (3/7): Expand Trajectory Probability

Write the probability of a trajectory as a product over time:

$$\pi_\theta(\tau) = \underbrace{p(s_0)}_{\text{initial state}} \prod_{t=0}^{T} \underbrace{p(s_{t+1} \mid s_t, a_t)}_{\text{dynamics}} \underbrace{\pi_\theta(a_t \mid s_t)}_{\text{policy}}$$

So $\log \pi_\theta(\tau) = \log p(s_0) + \sum_{t=0}^{T} \big(\log p(s_{t+1} \mid s_t, a_t) + \log \pi_\theta(a_t \mid s_t)\big)$, and:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\nabla_\theta \log \left(p(s_0) \prod_{t=0}^{T} p(s_{t+1} \mid s_t, a_t)\pi_\theta(a_t \mid s_t)\right) \cdot R(\tau)\right]$$

Derivation (4/7): Only Policy Terms Depend on θ

Taking ∇_θ of the log: $p(s_0)$ and $p(s_{t+1} \mid s_t, a_t)$ do not depend on θ, so their gradients are zero. Only the policy terms remain:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\underbrace{\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right)}_{\text{sum of log-policy gradients over all actions in } \tau} \underbrace{R(\tau)}_{\text{return}}\right]$$

Explainer: The gradient of the transition terms goes to zero because they have no dependence on θ; we only need the policy $\pi_\theta(a_t \mid s_t)$ to compute this.

Derivation (5/7): Expand Return and Separate Time Indices

Write the return as a discounted sum and use a separate index t' for rewards to avoid confusion with the time index t in the log-policy sum:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\underbrace{\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t)}_{\text{log-policy sum over } t} \underbrace{\sum_{t'=0}^{T} \gamma^{t'} r(s_{t'}, a_{t'})}_{\substack{\text{discounted return} \\ R(\tau)\text{, sum over } t'}}\right]$$

Explainer: We use t for the gradient terms and t' for the reward terms so the two summations are clearly distinct.

Derivation (6/7): Past Rewards $\to$ Zero; Rewards-to-Go

Move the sum over t outside the expectation and split the return into *past* ($t' < t$) and *future* ($t' \geq t$) rewards:

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T} \mathbb{E}_{\tau\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(\underbrace{\sum_{t'=0}^{t-1} \gamma^{t'} r(s_{t'}, a_{t'})}_{\substack{\text{past rewards}\\ \Rightarrow \mathbf{0}}} + \underbrace{\sum_{t'=t}^{T} \gamma^{t'} r(s_{t'}, a_{t'})}_{\text{rewards-to-go } G_t}\right)\right]$$

Derivation (7/7): Past Rewards $\to$ Zero; Rewards-to-Go

Past rewards do not depend on the action a_t (or θ at time t), so we can reduce variance by removing them (for a more thorough derivation of why see OpenAI Spinning Up RL). Dropping that term:

$$\boxed{\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \underbrace{\left(\sum_{t'=t}^{T} \gamma^{t'} r(s_{t'}, a_{t'})\right)}_{G_t \text{ (rewards-to-go)}} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right]}$$

The REINFORCE Algorithm

REINFORCE: A Complete RL Algorithm

The simplest complete RL algorithm based on the policy gradient is **REINFORCE** (Williams, 1992):

1. Initialize policy $\pi_\theta(a \mid s)$
2. Collect trajectories τ using π_θ
3. Compute the policy gradient estimate $\hat{\nabla}_\theta$
4. Update: $\theta \leftarrow \theta + \eta \hat{\nabla}_\theta$
5. Go back to step 2

You'll implement this on your next homework. It works fairly well for simple problems, but has high variance.

Some intuition: Reward-Weighted Imitation Learning

One way to interpret the policy gradient is as doing **weighted imitation learning**.

You can cast a supervised learning objective in the RL framework as “imitation learning” or behavioral cloning (it’s just learning doing supervied learning of your policy from demonstrations).

$$\max_\theta \mathbb{E}_{(s,a)\sim\mathcal{D}}[\log \pi_\theta(a \mid s)] \quad \Rightarrow \quad \nabla_\theta = \mathbb{E}_{(s,a)\sim\mathcal{D}}[\nabla_\theta \log \pi_\theta(a \mid s)]$$

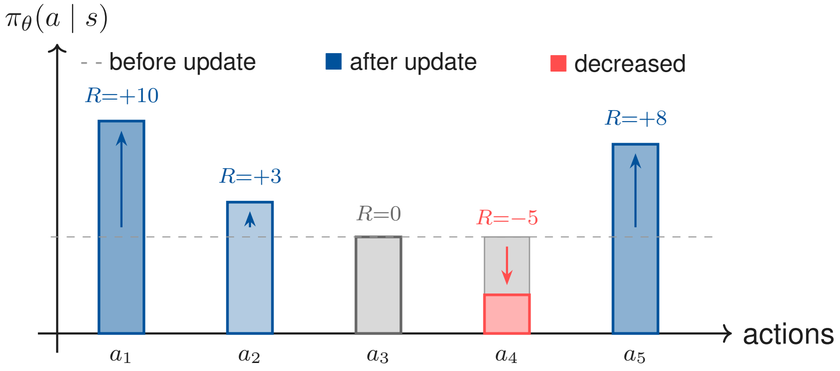
The policy gradient looks like a **weighted version of behavioral cloning**:

$$\nabla_\theta J(\theta) \approx \frac{1}{K} \sum_{\tau_k} R(\tau_k) \nabla_\theta \log \pi_\theta(\tau_k)$$

Key difference: we sample data from *our own policy* and then do the reweighting. The “predictions” are actions from our policy, the “labels” are the actions actually taken, weighted by their reward.

Visualizing the Policy Gradient Update

The policy gradient pushes up the probability of actions *in proportion to their return*:



A note on on off-policy sampling

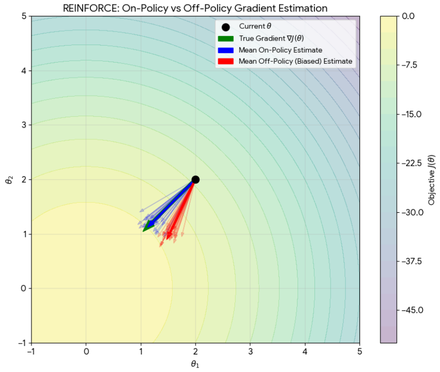
REINFORCE is on policy. Recall our goal is to estimate via Monte Carlo:

$$\boxed{\nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[R(\tau)] = \mathbb{E}_{\tau \sim \pi_\theta(\tau)}\left[R(\tau)\, \nabla_\theta \log \pi_\theta(\tau)\right]}$$

$$\nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)] \approx \frac{1}{K}\sum_{k=1}^{K} R(\tau_k)\, \nabla_\theta \log \pi_\theta(\tau_k)$$

A note on on off-policy sampling

If we take off policy samples, our distribution is shifted and we're no longer estimating the correct gradient.



Break 20 minutes

From REINFORCE to Trust Region Methods

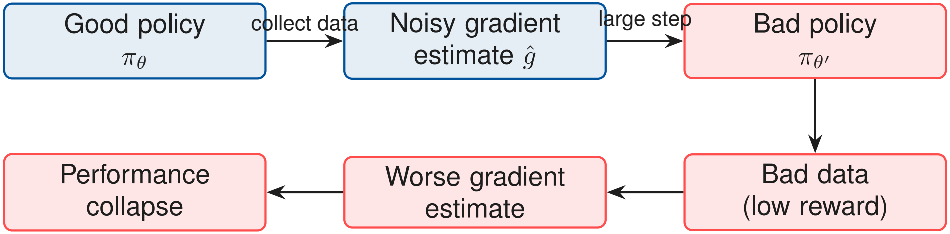
The Problem with REINFORCE

Before the break we derived REINFORCE: collect trajectories, compute $\nabla_\theta J \approx \frac{1}{K}\sum_k R(\tau_k)\nabla_\theta \log \pi_\theta(\tau_k)$, take a gradient step.

This works, but it is **fragile**. The gradient estimate has very high variance (it depends on entire trajectories), so any single batch of data can produce a wildly inaccurate gradient. If we take a step that is too large, the policy can change drastically and performance collapses.

The fundamental issue: the gradient tells us a direction that *locally* improves the objective. But we have no guarantee about what happens after a *finite-sized* step. In supervised learning this is usually fine because the data distribution is fixed. In RL, a bad policy *collects bad data*, which makes the next update even worse.

Why Large Steps Are Dangerous in RL



In supervised learning, taking a bad gradient step is annoying but recoverable — the training data doesn't change. In RL, a bad step changes the data distribution, which can create a **negative feedback loop**. We need a way to take *the largest step we can* while *guaranteeing* the policy does not get worse.

What Would We Want?

Recall that **policy iteration** from Lecture 2 does provide a monotonic improvement guarantee: $V^{\pi_{k+1}}(s) \geq V^{\pi_k}(s)$ for all s. But it makes a drastic (greedy) change to the policy at each step, which is also destabilizing with function approximation.

Ideally, we want both:

$$\underbrace{\text{guaranteed improvement}}_{\text{like policy iteration}} \qquad + \qquad \underbrace{\text{small, controlled steps}}_{\text{like gradient methods}}$$

This is exactly the problem that Kakade & Langford (2002) and Schulman et al. (2015) solved. The rest of this lecture develops their solution step by step.

Notation for This Section

We follow the notation from Schulman et al. (2015). Write $\eta(\pi)$ for the **expected discounted return** of a policy π:

$$\eta(\pi) = \mathbb{E}_{\substack{s_0 \sim \rho_0 \\ a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim P(\cdot|s_t, a_t)}} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]$$

Note on Notation

I know that changing notation is a little annoying, but I'm purposefully showing that people will use different notation for discounted return. In this case, we are following the TRPO paper notation.

This is the same objective we have been maximizing (previously written $J(\theta)$).

The Performance Difference Identity (1/2)

The following identity expresses the return of a new policy $\tilde{\pi}$ in terms of the advantage over the old policy π (Kakade & Langford, 2002):

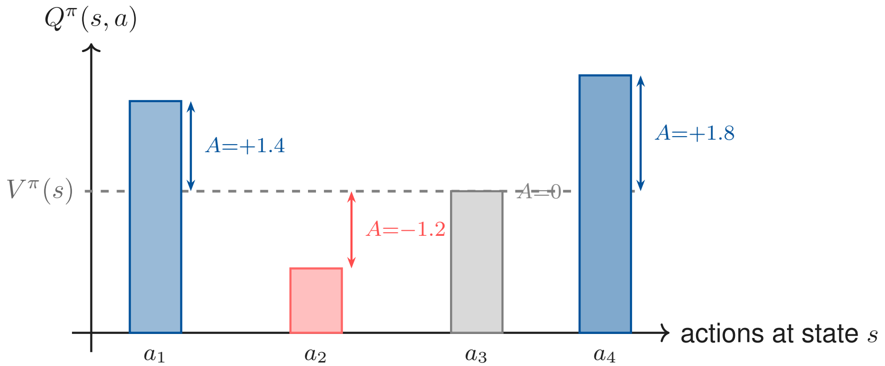
$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{s_0, a_0, \ldots \sim \tilde{\pi}} \left[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t) \right] \tag{1}$$

In words: the return of the new policy equals the return of the old policy, plus the expected cumulative advantage of the new policy's actions, evaluated using the *old* policy's value function.

If the new policy consistently chooses actions with positive advantage (i.e., actions that are better than what π would do on average), then $\eta(\tilde{\pi}) > \eta(\pi)$.

Intuition: What Are Advantages?

At a state s, $V^\pi(s)$ is the average return under π. The advantage $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$ measures how much *better* (or worse) action a is than this average:



The Performance Difference Identity (2/3)

Define the **(unnormalized) discounted visitation frequency**:

$$\rho_\pi(s) = P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + \cdots$$

where $s_0 \sim \rho_0$ and the actions are chosen according to π. We can rewrite (1) with a sum over states instead of timesteps:

$$\begin{aligned}
\eta(\tilde{\pi}) &= \eta(\pi) + \sum_{t=0}^{\infty} \sum_{s} P(s_t = s \mid \tilde{\pi}) \sum_{a} \tilde{\pi}(a \mid s)\, \gamma^t A_\pi(s,a) \\
&= \eta(\pi) + \sum_{s} \sum_{t=0}^{\infty} \gamma^t P(s_t = s \mid \tilde{\pi}) \sum_{a} \tilde{\pi}(a \mid s)\, A_\pi(s,a) \\
&= \eta(\pi) + \sum_{s} \rho_{\tilde{\pi}}(s) \sum_{a} \tilde{\pi}(a \mid s)\, A_\pi(s,a). \qquad (2)
\end{aligned}$$

The Performance Difference Identity (3/3)

This equation implies that any policy update $\pi \to \tilde{\pi}$ that has a **nonnegative expected advantage at every state** s, i.e. $\sum_a \tilde{\pi}(a \mid s) A_\pi(s, a) \geq 0$, is guaranteed to *increase* the policy performance η, or leave it constant when the expected advantage is zero everywhere.

This implies the classic result that the update performed by **exact policy iteration**, which uses the deterministic policy $\tilde{\pi}(s) = \arg\max_a A_\pi(s, a)$, improves η.

The Local Approximation L_π

The expression $\sum_s \rho_{\tilde{\pi}}(s) \sum_a \tilde{\pi}(a|s) A_\pi(s,a)$ is hard to optimize because $\rho_{\tilde{\pi}}$ depends on $\tilde{\pi}$ — the very thing we are searching for.

Replace $\rho_{\tilde{\pi}}$ with ρ_π (the *current* policy's visitation) to obtain a **local approximation**:

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_\pi(s) \sum_a \tilde{\pi}(a \mid s)\, A_\pi(s,a)$$

For a parameterized policy π_θ, L matches η to **first order** around the current parameters (Kakade & Langford, 2002):

$$L_{\pi_{\theta_0}}(\pi_{\theta_0}) = \eta(\pi_{\theta_0}), \qquad \nabla_\theta L_{\pi_{\theta_0}}(\pi_\theta)\big|_{\theta=\theta_0} = \nabla_\theta \eta(\pi_\theta)\big|_{\theta=\theta_0}$$

First-Order Matching: What It Means

The fact that L and η agree to first order means: a sufficiently small step that improves L will also improve η.

But “sufficiently small” is not quantitative. If we take a large step, L might increase a lot while η actually decreases — because L ignores how the new policy changes the *state distribution*.

We need a bound on the gap $|\eta(\tilde{\pi}) - L_\pi(\tilde{\pi})|$. This is exactly what Kakade & Langford provided for a special class of policies.

Conservative Policy Iteration (Kakade & Langford, 2002)

Rather than replacing the policy entirely, CPI **mixes** the old and new:

$$\pi_{\text{new}}(a \mid s) = (1-\alpha)\,\pi_{\text{old}}(a \mid s) + \alpha\,\pi'(a \mid s)$$

where $\pi' = \arg\max_{\pi'} L_{\pi_{\text{old}}}(\pi')$ and $\alpha \in [0,1]$ controls how aggressive the update is. For this mixture policy, Kakade & Langford proved:

$$\underbrace{\eta(\pi_{\text{new}})}_{\text{true performance}} \geq \underbrace{L_{\pi_{\text{old}}}(\pi_{\text{new}})}_{\text{surrogate } L} - \underbrace{\frac{2\varepsilon\gamma}{(1-\gamma)^2}\alpha^2}_{\text{penalty to approximation w/}\alpha^2}$$

where $\varepsilon = \max_s |\mathbb{E}_{a\sim\pi'}[A_\pi(s,a)]|$ (max expected advantage).
The penalty grows as α^2 while L improves linearly in α. So for small enough $\alpha > 0$, improvement is guaranteed.

Limitations of the CPI Bound

There are two problems with the CPI bound:

First, it only applies to **mixture policies** $\pi_{\text{new}} = (1-\alpha)\pi + \alpha\pi'$. In practice, we use neural network policies — not mixtures. We need a result for *arbitrary* policy updates.

Second, the bound is very conservative. With $\gamma = 0.99$ and $\varepsilon = 1$, the constant is $C = 2 \cdot 0.99/(0.01)^2 = 19{,}800$. Even a tiny step $\alpha = 0.01$ incurs a penalty of 1.98, requiring a substantial surrogate improvement to guarantee progress.

Schulman et al. (2015) addressed both of these issues.

Trust Region Policy Optimization

Extending to General Stochastic Policies (Schulman et al., 2015)

The key theoretical contribution of TRPO is extending the CPI bound from mixture policies to *all* stochastic policies. The idea is to replace the mixing coefficient α with a distance measure between the old and new policies.

The **total variation divergence** between two distributions p and q is:

$$D_{\mathsf{TV}}(p\|q) = \frac{1}{2}\sum_i |p_i - q_i|$$

Define $D_{\mathsf{TV}}^{\max}(\pi, \tilde{\pi}) = \max_s D_{\mathsf{TV}}(\pi(\cdot|s)\|\tilde{\pi}(\cdot|s))$.

The TRPO Improvement Theorem

Let $\alpha = D_{\text{TV}}^{\max}(\pi_{\text{old}}, \pi_{\text{new}})$. Then:

$$\eta(\pi_{\text{new}}) \geq L_{\pi_{\text{old}}}(\pi_{\text{new}}) - \frac{4\varepsilon\gamma}{(1-\gamma)^2}\alpha^2, \quad \varepsilon = \max_{s,a}|A_\pi(s,a)|$$

This looks like the CPI bound, but α is now the **total variation distance** between the old and new policies rather than the mixing coefficient.

The proof extends Kakade & Langford's argument using a **coupling argument**: if two distributions have TV distance $\leq \alpha$, their random variables can be coupled so they agree with probability $\geq 1 - \alpha$.

From TV Divergence to KL Divergence

Total variation is hard to work with computationally. But can use the known property:

$$D_{\mathsf{TV}}(p\|q)^2 \leq D_{\mathsf{KL}}(p\|q)$$

From TV Divergence to KL Divergence

Substituting into the bound gives:

$$\eta(\pi_{\text{new}}) \geq L_{\pi_{\text{old}}}(\pi_{\text{new}}) - \frac{4\varepsilon\gamma}{(1-\gamma)^2}\alpha^2, \quad \varepsilon = \max_{s,a}|A_\pi(s,a)|$$

$$\eta(\tilde{\pi}) \geq L_\pi(\tilde{\pi}) - C \cdot \max_s D_{\text{KL}}(\pi(\cdot|s)\|\tilde{\pi}(\cdot|s)), \quad C = \frac{4\varepsilon\gamma}{(1-\gamma)^2}$$

This is the form used by TRPO. The KL divergence is much easier to estimate and differentiate than the TV divergence, and it arises naturally in the Fisher information matrix.

Guaranteed Monotonic Improvement

Define $M_i(\pi) = L_{\pi_i}(\pi) - C \cdot D_{\text{KL}}^{\max}(\pi_i, \pi)$. From the bound:

$$\eta(\pi_{i+1}) \geq M_i(\pi_{i+1}) \qquad \text{and} \qquad \eta(\pi_i) = M_i(\pi_i)$$

Therefore:

$$\eta(\pi_{i+1}) - \eta(\pi_i) \geq M_i(\pi_{i+1}) - M_i(\pi_i)$$

By choosing $\pi_{i+1} = \arg\max_\pi M_i(\pi)$, we guarantee $M_i(\pi_{i+1}) \geq M_i(\pi_i)$, which gives:

$$\eta(\pi_0) \leq \eta(\pi_1) \leq \eta(\pi_2) \leq \cdots$$

This is a **minorization-maximization (MM)** algorithm, the same family as EM. At each step, M_i is a lower bound on η that touches it at π_i.

From Theory to Practice: Three Approximations

The theory says: at each step, solve $\max_\pi \left[L_{\pi_i}(\pi) - C \cdot D_{\mathsf{KL}}^{\max}(\pi_i, \pi)\right]$. This is impractical for three reasons, so TRPO makes three approximations.

Approximation 1: The penalty coefficient $C = 4\varepsilon\gamma/(1-\gamma)^2$ is enormous, leading to tiny steps. Replace the **penalty** with a **hard constraint** (trust region):

$$\max_\theta \; L_{\theta_{\mathsf{old}}}(\theta) \qquad \text{s.t.} \quad D_{\mathsf{KL}}^{\max}(\theta_{\mathsf{old}}, \theta) \le \delta$$

Empirically, choosing the penalty coefficient is fragile; a hard constraint with δ is more robust and allows much larger steps.

From Theory to Practice (continued)

Approximation 2: The constraint $\max_s D_{\text{KL}}$ requires checking every state. Replace it with the **average** KL:

$$\max_{\theta} \; L_{\theta_{\text{old}}}(\theta) \qquad \text{s.t.} \quad \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}}\left[D_{\text{KL}}(\pi_{\theta_{\text{old}}}(\cdot|s)\|\pi_\theta(\cdot|s))\right] \le \delta$$

This can be estimated from samples. Experiments in the paper show that the average KL constraint performs similarly to the max KL constraint.

Approximation 3: Replace the sum over states $\sum_s \rho_{\theta_{\text{old}}}(s)[\cdots]$ with a sample average, and use **importance sampling** to handle the action distribution.

The Importance-Sampled Surrogate

After these approximations, the surrogate objective becomes:

$$L_{\theta_{\text{old}}}(\theta) = \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}},\, a\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A^{\pi_{\theta_{\text{old}}}}(s,a)\right]$$

The ratio $\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$ is an **importance weight**: it corrects for the fact that we are sampling actions from $\pi_{\theta_{\text{old}}}$ but evaluating π_θ.

This expression can be computed entirely from data collected by the old policy — we do not need to run the new policy to evaluate it. This is what makes the optimization tractable.

Solving the Constrained Problem

TRPO solves $\max_\theta L_{\theta_{\text{old}}}(\theta)$ subject to $\bar{D}_{\text{KL}}(\theta_{\text{old}}, \theta) \leq \delta$ using two ideas.

Linearize L around θ_{old} and make a quadratic approximation to the KL constraint. The resulting subproblem is:

$$\max_\theta \; g^T(\theta - \theta_{\text{old}}) \qquad \text{s.t.} \quad \tfrac{1}{2}(\theta - \theta_{\text{old}})^T F(\theta - \theta_{\text{old}}) \leq \delta$$

where $g = \nabla_\theta L|_{\theta_{\text{old}}}$ and F is the **Fisher information matrix.** The solution is $\theta_{\text{new}} = \theta_{\text{old}} + \sqrt{2\delta / g^T F^{-1} g}\; F^{-1} g$.

Since F is huge, TRPO uses **conjugate gradient** to compute $F^{-1}g$ without forming F, followed by a **line search** along this direction to ensure the true KL constraint is satisfied.

PPO: Simplifying TRPO (Schulman et al., 2017)

TRPO works well but the constrained optimization (conjugate gradient + line search) is complex. PPO achieves a similar effect with a much simpler approach.

Define the **importance ratio**: $\rho_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$. The TRPO surrogate is $L(\theta) = \mathbb{E}[r_t(\theta)A_t]$.

If ρ_t is close to 1, the old and new policies agree on action a_t. If the *new* policy takes it much more often, ρ_t can be very large, and the update overshoots.

PPO's solution: **clip** the ratio to $[1-\epsilon, 1+\epsilon]$ where $\epsilon \approx 0.2$.

PPO: The Clipped Surrogate Objective

PPO clips the importance ratio directly:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}\left[\min\left(\rho_t(\theta)\, A_t,\ \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)\, A_t\right)\right]$$

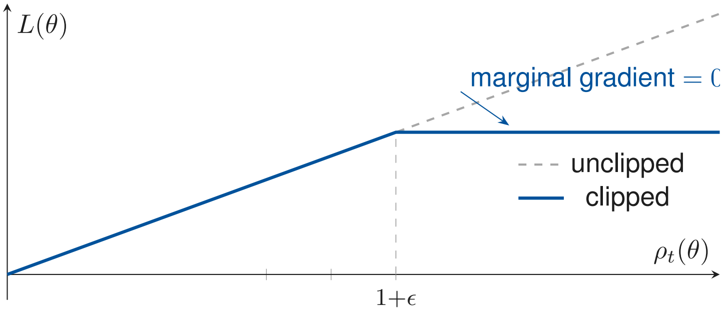
The $\min$ selects whichever term is more conservative.

When $A_t > 0$ (good action), we want to increase $\pi_\theta(a_t|s_t)$, but the clip caps r_t at $1+\epsilon$, preventing the probability from increasing too much. When $A_t < 0$ (bad action), the unclipped $r_t A_t$ can give an arbitrarily strong penalty, and the $\min$ keeps it — no artificial floor on how much a bad action's probability can drop.

PPO is the workhorse of modern RL: used in RLHF for ChatGPT, robotics, game playing, etc.

Visualizing the PPO Clip ($A_t > 0$)

When an action has positive advantage, we want to increase its probability. The clipped objective prevents overshooting:



Once $\rho_t > 1+\epsilon$, the objective flattens and the marginal added gradient vanishes. The policy cannot increase this action's probability further, preventing the kind of overshooting that destabilizes REINFORCE.

A note on on off-policy sampling

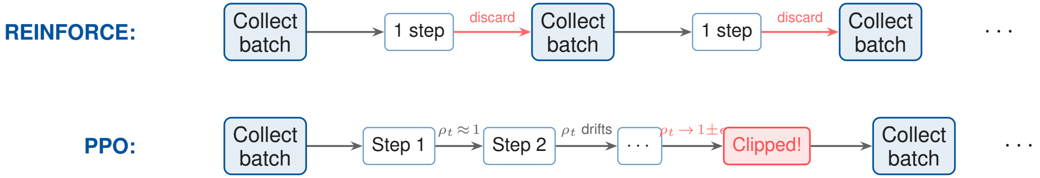
PPO is actually not quite fully on-policy, which allows us to take multiple gradient steps. Recall that in REINFORCE, vanilla policy gradient we're trying to approximate:

$$\boxed{\nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[R(\tau)] = \mathbb{E}_{\tau \sim \pi_\theta(\tau)}\left[R(\tau)\, \nabla_\theta \log \pi_\theta(\tau)\right]}$$

$$\nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)] \approx \frac{1}{K} \sum_{k=1}^{K} R(\tau_k)\, \nabla_\theta \log \pi_\theta(\tau_k)$$

A note on on off-policy sampling

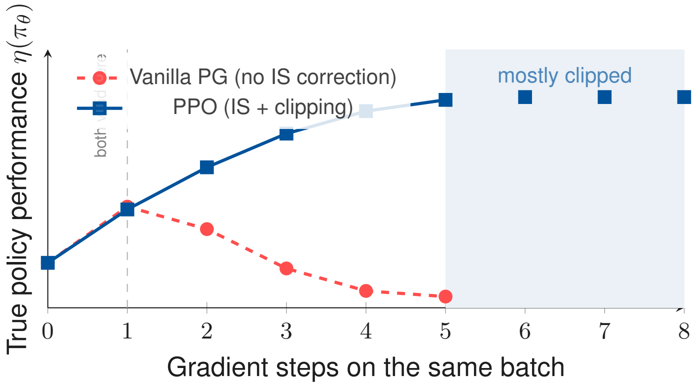
PPO's importance ratio $\rho_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ corrects for the fact that batch data comes from $\pi_{\theta_{\text{old}}}$, not π_θ. Combined with clipping, this allows **multiple gradient steps per batch**:



As θ moves away from θ_{old}, the ratios ρ_t drift from 1. Once they hit $1 \pm \epsilon$, the clip zeroes out the gradient for those samples. When most samples are clipped, the batch is "used up" — time for fresh data. PPO typically takes $K = 3$–10 steps per batch

REINFORCE vs. PPO: Multiple Steps on the Same Batch

What happens if we keep taking gradient steps on the same batch of trajectories?

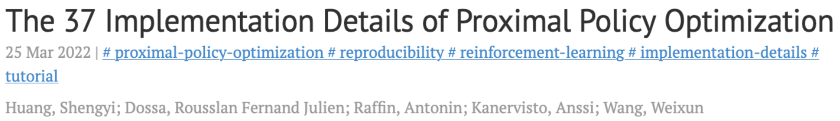


Many tunable hyperparameters

Without importance weighting, reusing off-policy data gives a **biased** gradient that can hurt performance. PPO's ratio ρ_t corrects the bias; the clip stops optimization when the batch is exhausted. This also can make PPO more **sample efficient**, but requires more tuning. Each batch yields multiple useful gradient steps, but we need to make sure we're taking a well-calbirated step size. PPO infamously has many tunable hyperparameters.

Encourage you to read

https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/


The 37 Implementation Details of Proximal Policy Optimization

25 Mar 2022 | # proximal-policy-optimization # reproducibility # reinforcement-learning # implementation-details # tutorial

Huang, Shengyi; Dossa, Rousslan Fernand Julien; Raffin, Antonin; Kanervisto, Anssi; Wang, Weixun

The Action Space, Continuous Control versus Discrete Actions

Discrete Actions: Categorical Policy

For discrete action spaces (e.g., Atari, board games), the network outputs a logit per action. We sample from this categorical distribution:

```
class DiscretePolicy(nn.Module):
    def __init__(self, obs_dim, num_actions):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(obs_dim, 64), nn.Tanh(),
            nn.Linear(64, 64),      nn.Tanh(),
            nn.Linear(64, num_actions),  # one logit per action
        )
    def forward(self, obs):
        logits = self.net(obs)
        return Categorical(logits=logits)  # softmax inside

# Usage
dist = policy(obs)          # Categorical distribution
action = dist.sample()      # e.g., tensor(2) meaning "go right"
log_prob = dist.log_prob(action)
```

The log-probability $\log \pi_\theta(a \mid s)$ comes directly from the softmax over logits.

Continuous Actions: Gaussian Policy

For continuous actions (e.g., robotics, MuJoCo), the network outputs the **mean** of a Gaussian. The standard deviation is a separate learnable parameter:

```python
class ContinuousPolicy(nn.Module):
    def __init__(self, obs_dim, act_dim):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(obs_dim, 64), nn.Tanh(),
            nn.Linear(64, 64),      nn.Tanh(),
            nn.Linear(64, act_dim),  # output = mean of Gaussian
        )
        self.log_std = nn.Parameter(torch.zeros(act_dim))
    def forward(self, obs):
        mean = self.net(obs)
        std = self.log_std.exp()
        return Normal(mean, std)  # diagonal Gaussian

# Usage
dist = policy(obs)           # Normal distribution
action = dist.sample()       # e.g., tensor([0.23, -1.07])
log_prob = dist.log_prob(action).sum(-1)  # sum over action dims
```

Preview: Lecture 5 — Advantage Estimation and Baselines

Next time: More tools to stabilize and scale policy gradient methods, all the way to LLMs.