Lecture 3c: Policy Optimization

Qi Zhang

Last Updated: October 2025

1 The Policy Optimization Objective

Value-based methods such as Q-learning maintain (only) value estimates as the RL agent's internal state and extract policies to finally recommend from those value estimates. As the ultimate goal of RL is to find value-maximizing policies, in this note we consider a more direct approach, *policy optimization*, that, as its name suggests, directly maintains and optimizes a policy from experiences.

Formally, consider discounted-reward MDP $M = (\mathcal{S}, \mathcal{A}, P, R, \gamma, d_0)$ where $d_0 \in \Delta(\mathcal{S})$ is the initial state distribution, and let (in-episode) timesteps be 0-based indexed as $h = 0, 1, \dots$ and $r_h := R(s_h, a_h)$ be the reward at timestep h. For policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, its performance is quantified as $J(\pi)$, the expected discounted cumulative reward starting from an initial state $s_0 \sim d_0$ and follow policy π:

$$J(\pi) := \mathbb{E}\left[\sum_{h=0}^{\infty} \gamma^h r_h \;\middle|\; s_0 \sim d_0, a_h \sim \pi(s_h), s_{h+1} \sim P(s_h, a_h)\right] = \mathbb{E}_{s_0 \sim d_0}\left[V_\pi(s_0)\right].$$

The policy optimization problem is therefore $\max_{\pi \in \Pi} J(\pi)$, where Π is a class of policies being searched over. Policy optimization can be performed in the planning setting (i.e., MDP M is fully known) or in the RL setting where the agent interacts with the MDP without requiring prior knowledge of (P, R).

2 Policy Gradient Methods: An Introduction

For the rest of this note, we focus on the case where the policy class is parameterized as $\Pi = \{\pi_\theta : \theta \in \Theta\}$, where parameter space $\Theta \subseteq \mathbb{R}^n$ is a set of continuous real vectors. Therefore, identifying a good policy in $\pi_\theta \in \Pi$ is equivalent to identifying a good parameter θ. We assume that $\pi_\theta(a|s)$ is differentiable with respect to θ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$ and let $\nabla_\theta \pi_\theta(a|s)$ denote the derivative.

A good example to keep in mind is the *neural policy* where a neural network is parameterized by θ, takes as input state s, and outputs $\pi_\theta(a|s)$ for all $a \in \mathcal{A}$, and therefore the differentiation of $\pi_\theta(a|s)$ can be achieved efficiently by backpropagation.

To search over such Π, we here consider gradient ascent methods that update parameter θ iteratively. For simplicity, we consider the unconstrained case where θ always remains in the valid set Θ after the update (e.g., $\Theta = \mathbb{R}^n$). For example, the standard gradient ascent update is $\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta)$ where scalar $\alpha > 0$ determines the size of the update in the direction of the *policy gradient*:

$$\nabla_\theta J(\pi_\theta) = \left[\tfrac{\partial J(\pi_\theta)}{\partial \theta_1}, \dots, \tfrac{\partial J(\pi_\theta)}{\partial \theta_n}\right] \in \mathbb{R}^n, \quad \text{where } \tfrac{\partial J(\pi_\theta)}{\partial \theta_i} = \lim_{\Delta\theta_i \to 0} \frac{J(\pi_{\theta + \Delta\theta_i}) - J(\pi_\theta)}{\Delta\theta_i}. \tag{1}$$

In principle, $\nabla_\theta J(\pi_\theta)$ can be numerically evaluated as in (1) by evaluating the policies before and after the n small perturbations (), but it is impractical when n is large. It turns out policy gradient $\nabla_\theta J(\pi_\theta)$ can be expressed in closed-form and there are even more than one such closed-form expression.

2.1 REINFORCE

We begin with the simplest policy gradient form called REINFORCE, which applies to the case where any episode always terminates after a finite number of H transitions. We denote the trajectory of an episode and its cumulative discounted reward as

$$\tau = (s_0, a_0, r_0, \dots, s_{H-1}, a_{H-1}, r_{H-1}, s_H), \quad R(\tau) = \textstyle\sum_{h=0}^{H-1} \gamma^h r_h.$$

The probability of obtaining a particular trajectory τ when following policy π is

$$P^\pi(\tau) = d_0(s_0)\pi(a_0|s_0)P(s_1 \mid s_0, a_0) \cdots \pi(a_{H-1}|s_{H-1})P(s_H \mid s_{H-1}, a_{H-1})$$

For notation conciseness, we will drop the subscript θ in ∇_θ and π_θ. The policy gradient can then be computed as

$$\begin{aligned}
\nabla J(\pi) &= \nabla \mathbb{E}_{\tau\sim\pi}[R(\tau)] \\
&= \nabla \left(\sum_\tau R(\tau)P^\pi(\tau)\right) \\
&= \sum_\tau R(\tau)\nabla P^\pi(\tau) \\
&= \sum_\tau R(\tau)P^\pi(\tau)\nabla \log P^\pi(\tau) \qquad \textbf{(Homework 3's 2a)} \\
&= \sum_\tau R(\tau)P^\pi(\tau)\nabla \log \Big(d_0(s_0)\pi(a_0|s_0)P(s_1 \mid s_0, a_0)\cdots\pi(a_{H-1}|s_{H-1})\Big) \\
&= \sum_\tau R(\tau)P^\pi(\tau)\nabla \Big(\log d_0(s_0) + \textstyle\sum_{h=0}^{H-1}\log\pi(a_h|s_h) + \sum_{h=0}^{H-1}\log P(s_{h+1} \mid s_h, a_h)\Big) \\
&= \sum_\tau R(\tau)P^\pi(\tau)\nabla \Big(\textstyle\sum_{h=0}^{H-1}\log\pi(a_h|s_h)\Big) \qquad \textbf{(Homework 3's 2b)} \qquad (2) \\
&= \mathbb{E}_{\tau\sim\pi}\Big[\underbrace{R(\tau)\textstyle\sum_{h=0}^{H-1}\nabla\log\pi(a_h|s_h)}_{\widehat{g}^{\text{ REINFORCE-1}}(\tau)}\Big]. \qquad (3)
\end{aligned}$$

We now discuss how to utilize the derivations above:

- In the planning setting where MDP M is fully known, we can compute $P^\pi(\tau)$ for any trajectory τ. Therefore, we can enumerate all possible trajectories to compute $\nabla J(\pi)$ exactly via (2).
- In the RL setting, (P, R) is unknown and therefore we cannot compute $P^\pi(\tau)$ or (2). Instead, the estimator $\widehat{g}^{\text{ REINFORCE-1}}(\tau)$ as defined in (3) is an unbiased estimator of $\nabla J(\pi)$ and we can update the parameter in a stochastic gradient ascent fashion, e.g. $\theta \leftarrow \theta + \alpha\widehat{g}(\tau)$, with trajectory τ sampled from current policy $\pi = \pi_\theta$. The resulting RL algorithm, REINFORCE, is outlined in Algorithm 1.

2.2 The action-value expression

Although the gradient estimator of (3) is unbiased, it is often of high variance, which will then hurt the performance of the stochastic gradient ascent. We next derive alternative policy gradient estimators that usually are of lower variances. A thorough discussion of variance reduction techniques for policy gradient is beyond the scope of this note.

Algorithm 1 REINFORCE

```
1: learner initializes parameter $\theta$ for policy $\pi_\theta$;
2: repeat over episodes:
3:     learner finishes current episode $\tau$ by following $\pi_\theta$ until termination;
4:     learner updates parameter $\theta$ with REINFORCE estimator $\widehat{g}(\tau)$ (cf. (3) (6));
5: until some stopping criterion is met
6: learner outputs policy $\widehat{\pi} = \pi_\theta$ or $\widehat{\pi}(s) = \arg\max_{a \in \mathcal{A}} \pi_\theta(a|s)$;
```

One reason that estimator (3) is of high variance is due to the episodic reward $R(\tau)$: its summation of rewards over many steps is usually of high variance. This issue can be alleviated by turning to an alternative expression of policy gradient that essentially replace the episodic reward with the action-values:

$$\nabla J(\pi) = \mathbb{E}_{(s,a)\sim d^\pi}\left[\nabla \log \pi(a|s) \cdot Q^\pi(s,a)\right] \tag{4}$$

Here, d^π is the (unnormalized) state-action *occupancy measure* that accumulates the discounted expectation of visitation of state-action pairs over time:

$$d^\pi(s,a) := \sum_{h=0}^{\infty} \gamma^t d_h^\pi(s,a) \text{ with } d_h^\pi(s,a) := \Pr(s_h = s, a_h = a \mid s_0 \sim d_0, \pi).$$

Here, $d^\pi(s,a)$ is not a probability measure (i.e., $\sum_{(s,a)} d^\pi(s,a) \neq 1$), but we still interpret the expectation as $\mathbb{E}_{(s,a)\sim d^\pi}[\cdot] = \sum_{s,a} d^\pi(s,a)(\cdot)$.

Again, in the planning setting, d^π and Q^π can be exactly computed, and therefore policy gradient $\nabla J(\pi)$ can be exactly computed via (4). To see how (4) yields a gradient estimator for the RL setting, note the following fact: for any function $f(s,a)$ on state-action pairs,

$$\mathbb{E}_{(s,a)\sim d^\pi}[f(s,a)] = \mathbb{E}_{\tau\sim\pi}\left[\sum_{h=0}^{\infty} \gamma^h f(s_h,a_h)\right]. \tag{5}$$

We omit a proof of (5), but it is fairly intuitive: d^π accumulates discounted visitation of state-action pair (s,a) by following π, so it is equivalent to using π to sample trajectories and averaging over timesteps. From this, we can obtain another policy gradient estimator:

$$\begin{aligned}
\nabla J(\pi) &= \mathbb{E}_{(s,a)\sim d^\pi}\left[\nabla \log \pi(a|s) \cdot Q^\pi(s,a)\right] \\
&= \mathbb{E}_{\tau\sim\pi}\left[\textstyle\sum_{h=0}^{H} \gamma^h \nabla \log \pi(a_h|s_h) \cdot Q^\pi(s_h,a_h)\right] && (\tau \text{ terminates after } H \text{ transitions}) \\
&= \mathbb{E}_{\tau\sim\pi}\Big[\underbrace{\textstyle\sum_{h=0}^{H} \gamma^h \nabla \log \pi(a_h|s_h) \cdot \sum_{h'=h}^{H} \gamma^{h'-h} r_{h'}}_{\widehat{g}^{\text{REINFORCE-2}}(\tau)}\Big] && (6)
\end{aligned}$$

Roughly speaking, the last equality above is due to $\mathbb{E}_{\tau\sim\pi}[Q^\pi(s_h,a_h)] = \mathbb{E}_{\tau\sim\pi}[\sum_{h'=h}^{H} \gamma^{h'-h} r_{h'}]$, and a more rigorous argument can be made via law of total expectation (conditioned on particular realizations of (s_h,a_h)). The resulting estimator $\widehat{g}^{\text{REINFORCE-2}}(\tau)$ in (6) can then be used as the gradient estimator in Algorithm 1.

3 Policy Optimization in a Trust Region, Proximal Policy Optimization

All methods in Section 2, often referred to as *vanilla policy gradient* methods, adopt the standard gradient ascent approach that updates the parameter with a single step aligned with the policy gradient, with different methods varying in how the gradient estimation is performed. In this section, we introduce another family of methods, *trust region* policy optimization methods, that update the current policy parameter with multiple gradient steps. We will describe in detail one such method, Proximal Policy Optimization (PPO), which is easy to implement and are in many cases more efficient and stable than vanilla policy gradient methods.

3.1 Policy optimization in a trust region.

The key idea behind the trust region methods is to form a surrogate objective that well approximates the true objective to $J(\pi_\theta)$ within a local region, i.e., trust region, where the deviation from π_θ is sufficiently small. The technical tool to form such a surrogate objective is the *Performance Difference Lemma* that quantifies the difference in the values of two policies:

Lemma 1 (Performance Difference Lemma)**.** *For any two policies π, π',*

$$J(\pi') - J(\pi) = \mathbb{E}_{(s,a)\sim d^{\pi'}}\left[A^\pi(s,a)\right]$$

where $A^\pi(s,a) := Q^\pi(s,a) - V^\pi(s)$ is called the advantage of policy π.

To use Lemma 1 to optimize $J(\pi_\theta)$, consider setting $\pi = \pi_{\theta_{\text{old}}}$ and $\pi' = \pi_\theta$, where θ_{old} is the parameter of the current policy from which we have sampled trajectories ($\tau \sim \pi_{\theta_{\text{old}}}$) and θ is the candidate policy parameter we are optimizing over. We proceed as

$$\begin{aligned}
J(\pi_\theta) - J(\pi_{\theta_{\text{old}}}) &= \mathbb{E}_{(s,a)\sim d^{\pi_\theta}}\left[A^{\pi_{\theta_{\text{old}}}}(s,a)\right] \\
&= \mathbb{E}_{s\sim d^{\pi_\theta},\, a\sim\pi_\theta(s)}\left[A^{\pi_{\theta_{\text{old}}}}(s,a)\right] \quad (d^{\pi_\theta} \text{ is abused as the marginal state distribution}) \\
&= \mathbb{E}_{s\sim d^{\pi_\theta},\, a\sim\pi_{\theta_{\text{old}}}(s)}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A^{\pi_{\theta_{\text{old}}}}(s,a)\right] \qquad \textbf{(Homework 3's 2c)} \\
&\approx \mathbb{E}_{s\sim d^{\pi_{\theta_{\text{old}}}},\, a\sim\pi_{\theta_{\text{old}}}(s)}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A^{\pi_{\theta_{\text{old}}}}(s,a)\right] \\
&= \mathbb{E}_{(s,a)\sim d^{\pi_{\theta_{\text{old}}}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A^{\pi_{\theta_{\text{old}}}}(s,a)\right] =: L(\theta)
\end{aligned}$$

The above derivation is useful because:

- Since $\pi_{\theta_{\text{old}}}$ is fixed, $\max_\theta J(\theta)$ is equivalent to $\max_\theta(J(\pi_\theta) - J(\pi_{\theta_{\text{old}}}))$, i.e., maximizing the improvement of π_θ over $\pi_{\theta_{\text{old}}}$, which is exactly what is being quantified.
- According to the first equality, the exact improvement can be estimated by trajectories sampled from π_θ (i.e., $\mathbb{E}_{d^{\pi_\theta}}$) but we only have trajectories from $\pi_{\theta_{\text{old}}}$. To solve this issue, the derivation replaces d^{π_θ} with $d^{\pi_{\theta_{\text{old}}}}$, using the importance sampling trick (the third equality) and also paying the approximation error ($\approx$). In the RL setting, we can approximate $\mathbb{E}_{(s,a)\sim d^{\pi_{\theta_{\text{old}}}}}$ via trajectories sampled from the policy, $\tau \sim \pi_{\theta_{\text{old}}}$, as justified in Equation (5). These trajectories can also be used to estimate advantages $A^{\pi_{\text{old}}}$.

- The approximation, $L(\theta)$, is the surrogate objective we will be optimizing, which is a good approximation when π_θ is not too different from $\pi_{\theta_{\text{old}}}$ (so d^{π_θ} is not too different from $d^{\pi_{\theta_{\text{old}}}}$):

$$\max_\theta \; L(\theta) \quad \text{subject to: } \pi_\theta \text{ is not too different from } \pi_{\theta_{\text{old}}} \tag{7}$$

- We can perform a full-scale optimization of (7), e.g., via multiple gradient ascent steps. After the optimization, we will set θ_{old} to the solution and iteratively proceed to the next round.

3.2 PPO

PPO is an easy-to-implement method for the optimization problem of (7). Instead of a hard-coded trust region constraint, PPO performs the unconstrained optimization of a clipped version of $L(\theta)$: letting $r_\theta(a|s) := \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$

$$\max_\theta \; \mathbb{E}_{(s,a)\sim d^{\pi_{\theta_{\text{old}}}}}\left[\min\left\{\underbrace{r_\theta(a|s)A^{\pi_{\theta_{\text{old}}}}(s,a)}_{\text{same as in } L(\theta)},\; \underbrace{\text{clip}_{1-\epsilon}^{1+\epsilon}\left(r_\theta(a|s)\right)A^{\pi_{\theta_{\text{old}}}}(s,a)}_{\text{same, but with } r_\theta \text{ clipped}}\right\}\right].$$

The clipping implicitly discourages changing π_θ too much from $\pi_{\theta_{\text{old}}}$. Figure 1 illustrates what would happen respectively for advantage being positive vs negative and the policy ratio being clipped vs not clipped.

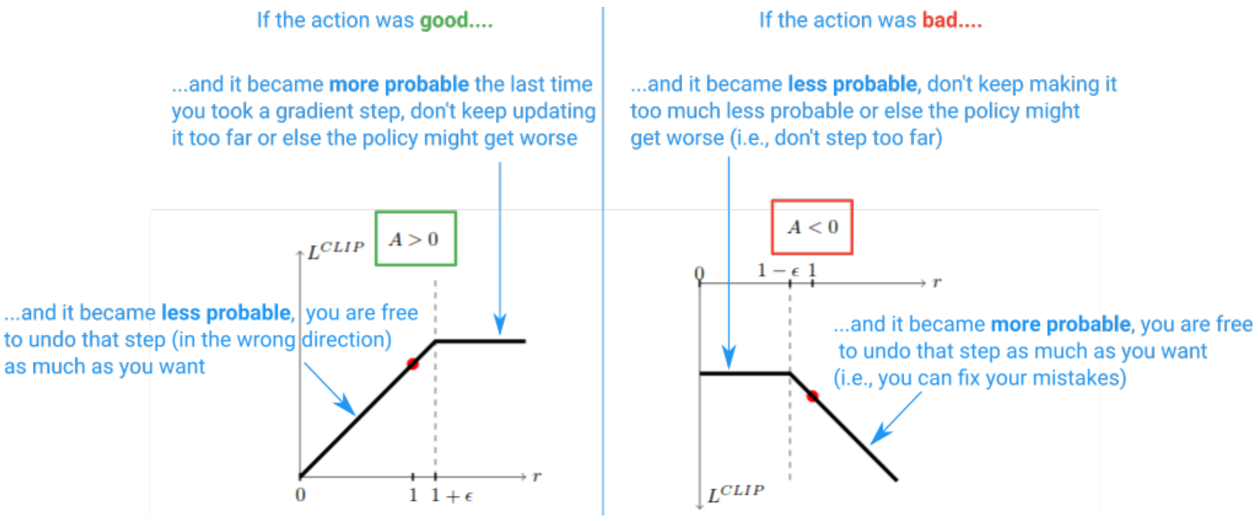

Figure 1: Plot showing one term for a certain (s, a)-pair of the PPO objective as a function of the policy ratio $r_\theta(a|s)$. Credit to this post.

Implementation details. We provide a pseudocode of PPO in Algorithm 2, which should be useful for you to implement it in hw3.ipynb. The PPO version you will implement in hw3.ipynb is significantly simplified from a standard version, particularly on 1) how the policy is parameterized and how 2) the advantages are estimated (line 5). We recommend a standard PPO implementation (like this one) for regular use cases.

Algorithm 2 Proximal Policy Optimization (PPO) with clipped objective

```
1: learner initializes parameter $\theta$ for policy $\pi_\theta$;
2: $\theta_{\text{old}} \leftarrow \theta$;
3: **repeat**
4:     Follow $\pi_{\theta_{\text{old}}}$ to collect and store multiple transitions $\{(s_t, a_t, r_t, s_{t+1})\}$;
5:     Estimates advantages $\widehat{A}_t \approx A^{\pi_{\theta_{\text{old}}}}(s_t, a_t)$;        ▷ Can involve training and using a *critic*
6:     Optimize $\theta$ to maximize
           $$\mathbb{E}_t \left[ \min \left\{ r_t(\theta)\widehat{A}_t,\ \text{clip}_{1-\epsilon}^{1+\epsilon}\left(r_t(\theta)\right)\widehat{A}_t \right\} \right]$$
       where $r_t(\theta) := \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ and $\mathbb{E}_t$ denotes sampling from the stored transitions;
7:     Update parameter $\theta_{\text{old}} \leftarrow \theta$;
8: **until** some stopping criterion is met
9: learner outputs policy $\widehat{\pi} = \pi_\theta$ or $\widehat{\pi}(s) = \arg\max_{a \in \mathcal{A}} \pi_\theta(a|s)$ with $\theta = \theta_{\text{old}}$;
```