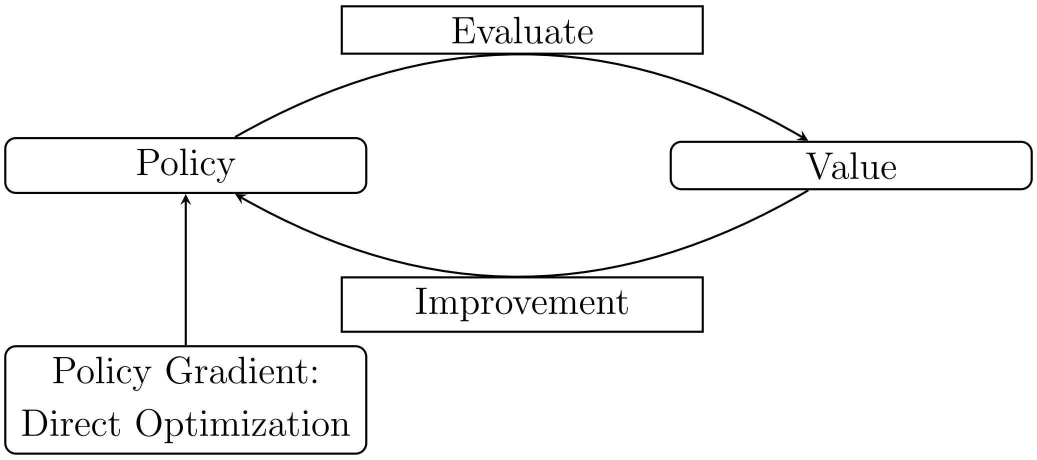
17-740 Spring 2025
Algorithmic Foundations of Interactive Learning

Lecture 12: Policy Gradient and Actor-Critic Method

Lecturer : Steven Wu Scribe: Nuoya Xiong, Anupam Nayak, Lujing Zhang,Vansh K.

1.1 Policy Gradient



In reinforcement learning, many algorithms, such as policy iteration, revolve around two key objects: policies and value functions. Policy iteration, for instance, alternates between *policy evaluation* and *policy improvement*. In contrast, policy gradient methods primarily focus on directly optimizing policy parameters. While we will momentarily step away from the interplay between value functions and policies, the second part of this lecture will reintroduce value functions as a tool for variance reduction.

To recep policy gradient methods improve $J(\pi_\theta)$ by performing policy gradient ascent update:

$$\theta = \theta + \eta \nabla_\theta J(\pi_\theta).$$

A few key points to consider are that $J(\pi_\theta)$ is almost always non-convex and must be estimated using sampled data. Methods that directly optimize the policy are particularly useful in scenarios where maintaining an explicit value function is unnecessary or impractical. This is often the case in applications to language models.

1.2 Notation

The policy parameterized by θ is represented as $\pi_\theta(a|s) = \pi(a|s,\theta)$. P_θ represents the distribution induced by π_θ over trajectories

$$\tau = (s_0, a_0, r_0, \cdots, s_{H-1}, a_{H-1}, r_{H-1}, S_H).$$

The objective function is defined by

$$J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta} \underbrace{\left[\sum_{h=0}^{H-1} r_h\right]}_{R(\tau)}.$$

1.3 Policy Gradient Theorem

The REINFORCE algorithm computes the gradient by

$$(\text{REINFORCE}) : \nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim P_\theta}\left[R(\tau)\cdot\left(\sum_{h=0}^{H-1}\nabla_\theta \log(\pi(a_h|s_h))\right)\right].$$

Observations:

$$\begin{aligned}
&\sum_{h=0}^{H-1}\nabla\log\pi_\theta(a_h|s_h)\cdot\left(\sum_{h=0}^{H-1} r_h\right)\\
&\quad=\sum_{h=0}^{H-1}\nabla\log\pi_\theta(a_h|s_h)\cdot\left(\sum_{h=0}^{H-1} r_h\right)\\
&\quad=\sum_{h=0}^{H-1}\nabla\log\pi_\theta(a_h|s_h)\cdot\left(\underbrace{\sum_{h'=0}^{h-1} r_{h'}}_{A}+\underbrace{\sum_{h'=h}^{H-1} r_{h'}}_{B}\right).
\end{aligned}$$

The term A is unaffected by a_h, and the term b in expectation is $Q_h^{\pi_\theta}(s_h, a_h)$. When we take the expectation outside, the term A would be zero (as we will show later). Hence, we can derive the Q-version of REINFORCE as

$$(\text{Q-Version}) : \nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim P_\theta}\left[\sum_{h=0}^{H-1} Q_h^{\pi_\theta}(s_h,a_h)\nabla_\theta\log(\pi(a_h|s_h))\right].$$

Now if $b(s)$ is unaffected by the action a, we can derive

$$\begin{aligned}\mathbb{E}_{a\sim\pi(\cdot|s)}\left[b(s)\cdot\nabla_\theta\log\pi(a|s)\right] &= \sum_a b(s)\cdot\pi(a|s)\cdot\frac{1}{\pi(a|s)}\nabla_\theta(\pi(a|s)\\ &= b(s)\sum_a \nabla_\theta\pi_\theta(a|s)\\ &= b(s)\nabla_\theta\sum_\theta \pi_\theta(a|s)\\ &= 0.\end{aligned}$$

The last equation is because $\sum_a \pi(a|s) = 1$. Hence, we can subtract a baseline $b_h(s) = V_h(s_h)$ that is independent with a_h, and get the A-version of REINFORCE as

$$\text{(A-Version)} : \nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim P_\theta}\left[\sum_{h=0}^{H-1} A_h^{\pi_\theta}(s_h,a_h)\nabla_\theta\log(\pi(a_h|s_h))\right],$$

where $A_h^{\pi_\theta}(s_h,a_h) = Q_h^{\pi_\theta}(s_h,a_h) - V_h^{\pi_\theta}(s_h)$.

1.4 Estimation of Policy Gradient

In practice, we cannot usually get the exact full gradient. Hence, we need to derive an estimator of the policy gradient.

1.4.1 Monte-Carlo Estimation

Using Monte-Carlo estimation, we first roll out N trajectories

$$\{\tau_i = (s_0^{(i)}, a_0^{(i)}, r_0^{(i)}, \cdots, s_{H-1}^{(i)}, a_{H-1}^{(i)}, r_{H-1}^{(i)}, s_H^{(i)})\}_{i\in[N]}$$

following the policy π_θ. Then, the gradient can be estimated by

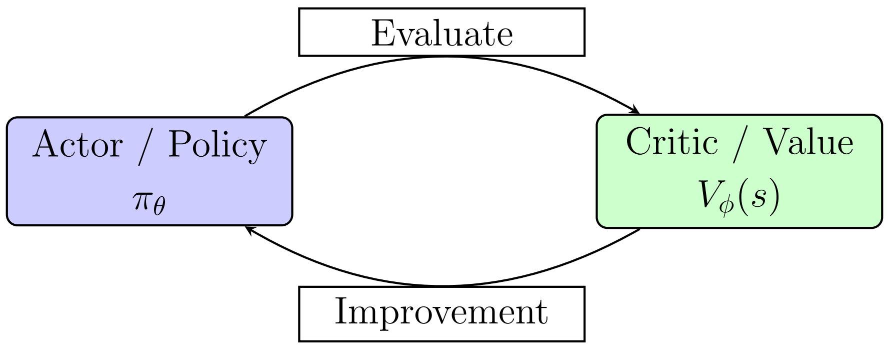
$$\nabla_\theta J(\pi_\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_h \nabla_\theta\log\pi_\theta(a_h^{(i)}|s_h^{(i)})\cdot R(\tau_i).$$

You can also get the Q function by $\hat{Q}_h^{(i)} = \sum_{h'=h}^{H-1} r_{h'}^{(i)}$ and estimate the gradient by

$$\nabla_\theta J(\pi_\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_h \nabla_\theta\log\pi_\theta(a_h^{(i)}|s_h^{(i)})\cdot \hat{Q}_h^{(i)}.$$

However, the Monte-Carlo sampling method often results in a gradient estimate with high variance. We will now introduce methods that reduce variance by introducing learned value functions V_ϕ as a baseline.

1.4.2 Actor-Critic Method



We define the actor/policy as π_θ, and the critic/value as $V_\phi(s)$. In practice, the actor and the critic are both modeled by neural networks.

When we have the critic/value function, we can estimate the policy gradient as

$$\nabla_\theta J(\theta) \approx \sum_{h=0}^{H-1} \nabla_\theta \log \pi(a_h|s_h) \cdot \left(\underbrace{r_h + V_\phi(s_{h+1}) - V_\phi(s_h)}_{\text{Temporal Difference Error}} \right),$$

where the **T**emporal **D**ifference (TD) error is an estimate of $A_h^{\pi_\theta}(s_h, a_h)$.

Note that the TD error is also an estimate of the *Bellman error*: $V(s) - \mathbb{E}_{a\sim\pi, s'\sim P}\left[r(s,a) + \gamma V(s')\right]$. This motivates an optimization objective for the *critic*, who tries to minimize the square loss given by

$$L(\phi) = \mathbb{E}\left[(r_h + V_\phi(s_{h+1}) - V_\phi(s_h))^2\right].$$

The critic will optimize this objective using the *semi-gradient method* which treats $V_\phi(s_{h+1})$ as a constant, ignoring its dependency when computing the gradient.

The full pseudocode is shown in the following algorithm:

Algorithm 1 Actor-Critic Method

```
1: **Input:** Learning rate $\eta_\phi, \eta_\theta$.
2: **for** episode $= 1, 2, \cdots, N$ **do**
3:     **for** $h = 1, 2, \cdots, H-1$ **do**
4:         **Critic Update:**
5:         $\delta_h = r_h + V_\phi(s_{h+1}) - V_\phi(s_h)$
6:         $\phi \leftarrow \phi + \eta_\phi \cdot \delta_h \nabla_\phi V_\phi(s_h)$.
7:         **Actor Update:**
8:         $\theta \leftarrow \theta + \eta_\theta \cdot \delta_h \nabla_\theta \log \pi_\theta(a_h|s_h)$.
9:     **end for**
10: **end for**
```

Suppose we execute the policy π_θ to get a trjaectory:

$$\pi_\theta \to s_0, a_0, r_0, \cdots, r_1, \cdots, r_2, \cdots, r_{H-1}, S_H.$$

At each step $h \in [H-1]$, you can receive a new reward r_h and get the (empirical) value function $V_\phi(s_h)$, and get the one-step advantage function $r_{h-1} + V_\phi(s_h) - V_\phi(s_{h-1})$.

One alternative of this one-step advantage function is called the **Multi-step** advantage function:

$$A^{(n)}(s_h, a_h) = r_h + \cdots + r_{h+n-1} + V_\phi(s_{h+n}) - V_\phi(s_h).$$

This gives us some form of interpolation between the Monte-Carlo estimation and the TD(0).

1.4.3 Generalized Advantage Estimation

We provide a more generalized advantage estimation $\hat{A}_h(\lambda)$ as

$$\hat{A}_h(\lambda) = \delta_h + \lambda\delta_{h+1} + \cdots + \lambda^{H-1-h}\delta_{H-1}.$$

When $\lambda = 0$, $\hat{A}_h(0)$ is the TD error, corresponding to the Actor-Critic Method.

When $\lambda = 1$, $\hat{A}_h(1)$ is the Monte-Carlo Estimation Approach.

With $\lambda = 0$, the estimate is a one-step TD error—highly biased due to its short-term focus but with low variance from stable updates. At $\lambda = 1$, it becomes a Monte Carlo estimator, reducing bias by considering full returns but increasing variance from accumulated stochastic noise. Hence, $\hat{A}_h(\lambda)$ is a trade-off between the high-bias estimate TD error and the high-variance Monte-Carlo estimator.