Section 3: Policy Gradients and Actor-Critic

Author: Seohong Park

1 Policy Gradient and REINFORCE

The reinforcement learning objective is to learn a θ^* that maximizes the objective function:

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}\big[r(\tau)\big] \tag{1}$$

where each rollout τ is of length H, distributed as follows:

$$p_\theta(\tau) = p(s_0, a_0, \dots, s_{H-1}, a_{H-1}) = p(s_0)\,\pi_\theta(a_0 \mid s_0) \prod_{t=1}^{H-1} p(s_t \mid s_{t-1}, a_{t-1})\,\pi_\theta(a_t \mid s_t), \tag{2}$$

and

$$r(\tau) = r(s_0, a_0, \dots, s_{H-1}, a_{H-1}) = \sum_{t=0}^{H-1} r(s_t, a_t). \tag{3}$$

One straightforward approach to optimizing this objective is to directly take the gradient of $J(\theta)$ with respect to θ and perform gradient ascent. That is, we iteratively update θ using the following gradient:

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)}\big[r(\tau)\big]. \tag{4}$$

But how do we compute this gradient in practice? Well, one might think that we can simply approximate the expectation with trajectory samples, as usual:

$$\nabla_\theta J(\theta) \overset{?}{\approx} \nabla_\theta \frac{1}{N} \sum_{i=1}^{N} r(\tau^i), \tag{5}$$

where $\tau^1, \dots, \tau^N$ are sampled from $p_\theta(\tau)$. Unfortunately, this does **not** work. Do you see why? (**Hint:** Note that $\frac{1}{N}\sum_{i=1}^{N} r(\tau^i)$ is a constant with respect to θ, so its gradient is 0.) The main issue is that the distribution over trajectories $p_\theta(\tau)$ also depends on θ, so we need to take that into account when taking the gradient.

The key idea behind policy gradients is to rewrite $\nabla_\theta J(\theta)$ as *some* expectation over trajectories τ (not the *derivative* of an expectation as in Equation (4)!), so that we can then do the usual sample-based approximation. That is, we want to fill in the question mark in the following equation:

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)}\big[r(\tau)\big] = \mathbb{E}_{\tau \sim p_\theta(\tau)}\Big[\boxed{\quad ? \quad}\Big]. \tag{6}$$

If we can fill in the box, then we can compute the right-hand side using samples of τ from $p_\theta(\tau)$ as usual, which doesn't suffer from the above issue because we don't have to take the gradient of an expectation.

The remaining task is to simply figure out what goes in the box. Let's first rewrite both sides as integrals:

$$\begin{aligned}
\text{(LHS)} &= \nabla_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)}\big[r(\tau)\big] && (7)\\
&= \nabla_\theta \int p_\theta(\tau)\, r(\tau)\, d\tau && (8)\\
&= \int \nabla_\theta p_\theta(\tau)\, r(\tau)\, d\tau, && (9)\\
\text{(RHS)} &= \mathbb{E}_{\tau \sim p_\theta(\tau)}\Big[\boxed{\quad ? \quad}\Big] && (10)\\
&= \int p_\theta(\tau) \boxed{\quad ? \quad} d\tau, && (11)
\end{aligned}$$

where we swapped the order of integration and differentiation in Equation (9) (which is valid under mild regularity conditions that we won't worry about in practice). Comparing Equation (9) and Equation (11), we see that we can fill in the box with

$$\boxed{\quad ? \quad} = \frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} r(\tau) = \nabla_\theta \log p_\theta(\tau)\, r(\tau). \tag{12}$$

Using this result, we can approximate the $\nabla_\theta J(\theta)$ as follows:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{\tau\sim p_\theta(\tau)}\big[r(\tau)\big] &(13)\\
&= \mathbb{E}_{\tau\sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau)\, r(\tau)] &(14)\\
&\approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log p_\theta(\tau^i)\, r(\tau^i) &(15)\\
&= \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i)\right)\left(\sum_{t=0}^{H-1} r(s_t^i, a_t^i)\right), &(16)
\end{aligned}$$

where $\tau^1, \ldots, \tau^N$ are sampled from $p_\theta(\tau)$. This is the vanilla policy gradient estimator, which serves as the basis for many policy gradient methods. The step of rewriting the policy gradient as an expectation (Equation (13) to Equation (14)) is often called the REINFORCE trick.[1]

2 Variance Reduction

While Equation (16) gives us a mathematically correct way to compute an estimate of the policy gradient using samples, this estimator often has high variance, which can lead to slow learning and poor performance in practice. So in practice, we use various variance reduction techniques to improve the performance of policy gradient methods.

2.1 Reward-To-Go

One way to reduce the variance is to note that the policy cannot affect rewards in the past, so we can ignore those rewards when computing the policy gradient. That is, we can replace the sum of rewards in Equation (16) with the sum of rewards starting from time step t:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i)\left(\sum_{t'=t}^{H-1} r(s_{t'}^i, a_{t'}^i)\right). \tag{17}$$

The term $\sum_{t'=t}^{H-1} r(s_{t'}^i, a_{t'}^i)$ is often called the "reward-to-go". This reduces variance simply because the reward-to-go consists of fewer terms than the full trajectory reward.

This reward-to-go formulation is unbiased (i.e., mathematically correct). That is, we have the following equality:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\left(\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right)\left(\sum_{t'=0}^{H-1} r(s_{t'}, a_{t'})\right)\right] &(18)\\
&= \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(\sum_{t'=t}^{H-1} r(s_{t'}, a_{t'})\right)\right]. &(19)
\end{aligned}$$

Here's a proof for the above equality. First, by comparing Equation (18) and Equation (19), note that it suffices to show that for any $t' < t$,

$$\mathbb{E}_{\tau\sim p_\theta(\tau)}[\nabla_\theta \log \pi_\theta(a_t \mid s_t) r(s_{t'}, a_{t'})] = 0. \tag{20}$$

[1]REINFORCE is a cursed acronym for "REward Increment = Nonnegative Factor times Offset Reinforcement times Characteristic Eligibility."

Let's define $\tau_{i,j} = (s_i, a_i, s_{i+1}, a_{i+1}, \ldots, s_j, a_j)$ for any $0 \le i \le j \le H-1$. Using this notation, we can rewrite the left-hand side as

$$\begin{aligned}
&\mathbb{E}_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log \pi_\theta(a_t \mid s_t) r(s_{t'}, a_{t'})] && (21)\\
&= \mathbb{E}_{\tau_{0:t} \sim p_\theta(\tau_{0:t})}[\nabla_\theta \log \pi_\theta(a_t \mid s_t) r(s_{t'}, a_{t'})] && (22)\\
&= \mathbb{E}_{\tau_{0:t'} \sim p_\theta(\tau_{0:t'})}\left[\mathbb{E}_{\tau_{t'+1:t} \sim p_\theta(\tau_{t'+1:t} \mid \tau_{0:t'})}[\nabla_\theta \log \pi_\theta(a_t \mid s_t) r(s_{t'}, a_{t'})]\right] && (23)\\
&= \mathbb{E}_{\tau_{0:t'} \sim p_\theta(\tau_{0:t'})}\left[r(s_{t'}, a_{t'}) \cdot \mathbb{E}_{\tau_{t'+1:t} \sim p_\theta(\tau_{t'+1:t} \mid \tau_{0:t'})}[\nabla_\theta \log \pi_\theta(a_t \mid s_t)]\right] && (24)\\
&= \mathbb{E}_{\tau_{0:t'} \sim p_\theta(\tau_{0:t'})}\left[r(s_{t'}, a_{t'}) \cdot \mathbb{E}_{s_t \sim p_\theta(s_t \mid \tau_{0:t'})}\left[\mathbb{E}_{a_t \sim \pi_\theta(a_t \mid s_t)}[\nabla_\theta \log \pi_\theta(a_t \mid s_t)]\right]\right] && (25)\\
&= \mathbb{E}_{\tau_{0:t'} \sim p_\theta(\tau_{0:t'})}\left[r(s_{t'}, a_{t'}) \cdot \mathbb{E}_{s_t \sim p_\theta(s_t \mid \tau_{0:t'})}\left[\int \pi_\theta(a_t \mid s_t)\, \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, da_t\right]\right] && (26)\\
&= \mathbb{E}_{\tau_{0:t'} \sim p_\theta(\tau_{0:t'})}\left[r(s_{t'}, a_{t'}) \cdot \mathbb{E}_{s_t \sim p_\theta(s_t \mid \tau_{0:t'})}\left[\int \nabla_\theta \pi_\theta(a_t \mid s_t)\, da_t\right]\right] && (27)\\
&= \mathbb{E}_{\tau_{0:t'} \sim p_\theta(\tau_{0:t'})}\left[r(s_{t'}, a_{t'}) \cdot \mathbb{E}_{s_t \sim p_\theta(s_t \mid \tau_{0:t'})}\left[\nabla_\theta \int \pi_\theta(a_t \mid s_t)\, da_t\right]\right] && (28)\\
&= \mathbb{E}_{\tau_{0:t'} \sim p_\theta(\tau_{0:t'})}\left[r(s_{t'}, a_{t'}) \cdot \mathbb{E}_{s_t \sim p_\theta(s_t \mid \tau_{0:t'})}[\nabla_\theta 1]\right] && (29)\\
&= 0. && (30)
\end{aligned}$$

This completes the proof.

2.2 Baseline

We can further reduce the variance of the policy gradient by introducing a baseline, which is a constant b that we subtract from the reward-to-go. That is, we replace the reward-to-go in Equation (19) with the reward-to-go minus a baseline:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}\left[\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \left(\sum_{t'=t}^{H-1} r(s_{t'}, a_{t'}) - b\right)\right]. \tag{31}$$

This leaves the policy gradient unbiased because for any t,

$$\begin{aligned}
&\mathbb{E}_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log \pi_\theta(a_t \mid s_t) b] && (32)\\
&= b \cdot \mathbb{E}_{s_t \sim p_\theta(s_t)}\left[\mathbb{E}_{a_t \sim \pi_\theta(a_t \mid s_t)}[\nabla_\theta \log \pi_\theta(a_t \mid s_t)]\right] && (33)\\
&= 0, && (34)
\end{aligned}$$

as we have shown in the proof for the reward-to-go formulation.

In fact, we can even use any *state-dependent* baseline $b(s_t)$ instead of a constant baseline:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}\left[\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \left(\sum_{t'=t}^{H-1} r(s_{t'}, a_{t'}) - b(s_t)\right)\right]. \tag{35}$$

This is still unbiased no matter how we choose $b(s_t)$. At this point, you should be able to prove this yourself using the same technique as before, and we encourage you to try writing down the proof on your own.

The next question is, what should we actually use as the baseline $b(s_t)$? In practice, we typically use a learned *value function* $\hat{V}^\pi_\varphi(s_t)$ as the baseline, where φ is the parameters of a neural network. This value function is trained to approximate the true value function $V^\pi(s_t)$, which is the expected return from state s_t under the current policy π_θ:

$$\hat{V}^\pi_\varphi(s_t) \approx V^\pi(s_t) = \mathbb{E}_{\tau \sim p_\theta(\tau)}\left[\sum_{t'=t}^{H-1} r(s_{t'}, a_{t'}) \mid s_t\right]. \tag{36}$$

With this choice of baseline, the policy gradient estimator becomes

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(\sum_{t'=t}^{H-1} r(s_{t'}, a_{t'}) - \hat{V}_\varphi^\pi(s_t)\right)\right]. \tag{37}$$

Intuitively, this value function baseline makes the policy gradient estimator more stable by *centering* the reward-to-go around its expected value, which can help reduce variance. Also, note that even when the value function is not accurate, the policy gradient estimator in Equation (37) is still unbiased (because we're allowed to subtract *any* function $b(s_t)$!).

Then, is it the *optimal* choice of baseline that minimizes the variance of the policy gradient estimator? The answer is no. In fact, it is possible to analytically write down the optimal baseline that minimizes the variance of the policy gradient estimator, which turns out to be a *weighted* average of the reward-to-go by some magnitude of the gradient $\nabla_\theta \log \pi_\theta(a_t \mid s_t)$ (if you're interested, see Page 19 of Lecture 5 from the Fall 2023 version of the course for details). However, this optimal baseline is difficult to approximate in practice, so we typically use the value function baseline as a practical choice for variance reduction.

2.3 Discounting

Another common technique for variance reduction is to introduce a discount factor $\gamma \in [0, 1]$ to the rewards. By multiplying the rewards by γ^t for time step t, we can reduce the variance of the policy gradient estimator by downweighting rewards that are further into the future, which are more uncertain and thus have higher variance. Specifically, we modify the original objective $J(\theta)$ to a new objective $J^\gamma(\theta)$ that incorporates discounting:

$$J^\gamma(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=0}^{H-1} \gamma^t r(s_t, a_t)\right]. \tag{38}$$

Then, as before, the (reward-to-go) policy gradient estimator becomes

$$\nabla_\theta J^\gamma(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(\sum_{t'=t}^{H-1} \gamma^{t'} r(s_{t'}, a_{t'})\right)\right]. \tag{39}$$

Note that introducing discounting changes the objective that we're optimizing, so it is technically biased (i.e., $J^\gamma(\theta) \neq J(\theta)$). However, this has several practical benefits. First, it reduces the variance. Second, in many cases, we actually *do* prefer immediate rewards to future rewards, so this change in the objective is often rather desirable in practice. Third, it allows us to handle the infinite-horizon case ($H = \infty$) without worrying about divergence of the returns.

In practice, we typically apply discounting to the reward-to-go formulation, which gives us another variant of the policy gradient estimator:

$$\nabla_\theta J^\gamma_{\text{rtg}}(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(\sum_{t'=t}^{H-1} \gamma^{t'-t} r(s_{t'}, a_{t'})\right)\right]. \tag{40}$$

Note that the discounting in Equation (39) is applied to the full trajectory reward, while the discounting in Equation (40) is applied to the reward-to-go by replacing $\gamma^{t'}$ with $\gamma^{t'-t}$. This modification changes the objective again. Between these two variants (Equation (39) and Equation (40)), we almost always use the reward-to-go discounting (Equation (40)) in practice. Do you see why? (**Hint:** The first variant makes the policy care less about *states* visited later in the trajectory because the gradient itself ($\nabla_\theta \log \pi_\theta(a_t \mid s_t)$) is downweighted by γ^t, whereas the second variant doesn't downweight the gradient, but only downweights the rewards that are further into the future. What do you think is more desirable in practice?)

With discounting, we also need to modify the value function to be discounted as well:

$$V^\pi(s_t) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t'=t}^{H-1} \gamma^{t'-t} r(s_{t'}, a_{t'}) \mid s_t\right]. \tag{41}$$

The baselined policy gradient estimator then becomes:

$$\nabla_\theta J^\gamma_{\text{rtg}}(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(\sum_{t'=t}^{H-1} \gamma^{t'-t} r(s_{t'}, a_{t'}) - \hat{V}^\pi_\varphi(s_t)\right)\right]. \tag{42}$$

From now on, we will use this discounted reward-to-go formulation with a value function baseline as our default policy gradient estimator, and we will drop the superscript γ and the subscript rtg in $J^\gamma_{\text{rtg}}(\theta)$ in the rest of the notes for simplicity.

3 Actor-Critic

While Equation (42) gives us a nice policy gradient estimator with variance reduction techniques, it may still have high variance in practice, due to the use of single-sample Monte Carlo returns (i.e., $\sum_{t'=t}^{H-1} \gamma^{t'-t} r(s_{t'}, a_{t'})$). To further reduce variance, we note that the policy gradient formula in Equation (42) can be rewritten as

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(\sum_{t'=t}^{H-1} \gamma^{t'-t} r(s_{t'}, a_{t'}) - \hat{V}^\pi_\varphi(s_t)\right)\right] &(43)\\
&= \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(r(s_t, a_t) + \gamma V^\pi(s_{t+1}) - \hat{V}^\pi_\varphi(s_t)\right)\right], &(44)
\end{aligned}$$

using the value function V^π defined in Equation (41). The proof is left as an exercise. (**Hint:** This can be done by splitting the expectation into nested expectations over $\tau_{0:t}$ and $\tau_{t+1:H-1}$ as in the proof for the reward-to-go formulation, and then using the definition of the value function to simplify the inner expectation.) Intuitively, the latter replaces the single-sample Monte Carlo return $\sum_{t'=t}^{H-1} \gamma^{t'-t} r(s_{t'}, a_{t'})$ with the expected return $r(s_t, a_t)+\gamma V^\pi(s_{t+1})$. This significantly reduces variance because the expected return already aggregates over all possible future trajectories.

In practice, we don't have access to the true value function $V^\pi(s_{t+1})$, so we replace it with the learned value function $\hat{V}^\pi_\varphi(s_{t+1})$. This gives us the following *actor-critic*[2] policy gradient estimator:

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(r(s_t, a_t) + \gamma \hat{V}^\pi_\varphi(s_{t+1}) - \hat{V}^\pi_\varphi(s_t)\right)\right]. \tag{45}$$

Note that this estimator is now *biased* due to modeling errors in $\hat{V}^\pi_\varphi(s_{t+1})$ (unlike Equation (44), which is unbiased because it uses the true value function $V^\pi(s_{t+1})$). However, it can perform much better than the unbiased Monte Carlo estimator in Equation (43) in practice, due to the significant reduction in variance. This is the first example of a *bias-variance tradeoff* in reinforcement learning.

[2]An "actor" means a policy (π), and a "critic" means a value function (V or Q). Sometimes, a "critic" refers only to a Q function in the literature, but here we use it to refer to value functions in general (including both V and Q). "Actor-critic" is a general term for methods that use a value function to help train a policy (so technically Equation (37) is already an actor-critic method, since it uses a value function as a baseline). These terms are a bit unfortunate, but we have no choice but to go along with them.

Quick exercise: Among the following four variants of Equation (45), which ones are unbiased and which ones are biased? The only difference between these four variants is where the hats are placed.

$$(1)\ \nabla_\theta J(\theta) \approx \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(r(s_t, a_t) + \gamma V_\varphi^\pi(s_{t+1}) - V_\varphi^\pi(s_t)\right)\right] \tag{46}$$

$$(2)\ \nabla_\theta J(\theta) \approx \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(r(s_t, a_t) + \gamma V_\varphi^\pi(s_{t+1}) - \hat{V}_\varphi^\pi(s_t)\right)\right] \tag{47}$$

$$(3)\ \nabla_\theta J(\theta) \approx \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(r(s_t, a_t) + \gamma \hat{V}_\varphi^\pi(s_{t+1}) - V_\varphi^\pi(s_t)\right)\right] \tag{48}$$

$$(4)\ \nabla_\theta J(\theta) \approx \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(r(s_t, a_t) + \gamma \hat{V}_\varphi^\pi(s_{t+1}) - \hat{V}_\varphi^\pi(s_t)\right)\right] \tag{49}$$

Now, let's talk about value function estimation. Our goal is to train $\hat{V}_\varphi^\pi$ to approximate the true value function V^π. The most straightforward way to do this is to minimize the mean squared error between $\hat{V}_\varphi^\pi(s_t)$ and the Monte Carlo return, as we discussed before:

$$\mathcal{L}(\varphi) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=0}^{H-1}\left(\hat{V}_\varphi^\pi(s_t) - \sum_{t'=t}^{H-1}\gamma^{t'-t} r(s_{t'}, a_{t'})\right)^2\right]. \tag{50}$$

While this is a valid way to train the value function, it can also suffer from the high variance of the Monte Carlo returns, just like the Monte Carlo policy gradient estimator in Equation (43). To reduce variance, we can apply the same idea to value function estimation as well, by replacing the Monte Carlo return with the *bootstrapped* return $r(s_t, a_t) + \gamma\hat{V}_\varphi^\pi(s_{t+1})$:

$$\mathcal{L}(\varphi) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=0}^{H-1}\left(\hat{V}_\varphi^\pi(s_t) - \left[r(s_t, a_t) + \gamma\hat{V}_\varphi^\pi(s_{t+1})\right]_\times\right)^2\right], \tag{51}$$

where the subscript $\times$ indicates the "stop-gradient" operator, which means that we treat $r(s_t, a_t) + \gamma\hat{V}_\varphi^\pi(s_{t+1})$ as a constant when computing the gradient with respect to φ. Equation (51) may seem a bit circular at first glance, since we're using $\hat{V}_\varphi^\pi$ to train itself ("bootstrapping"). However, under certain assumptions, we have a theoretical guarantee that minimizing this loss eventually leads to the true value function V^π with enough training iterations, even when we start from a random initialization of $\hat{V}_\varphi^\pi$. This is known as the temporal difference (TD) learning, and we will talk about it in more detail in the next couple of lectures. (**Aside:** You may also wonder why we have to put the stop-gradient operator in Equation (51). There is a subtle reason we won't go into here, but the short answer is that this becomes biased in stochastic environments without the stop-gradient operator. We may discuss this in more detail when we study TD learning.)

Our discussion so far can be summarized in the following actor-critic algorithm:

1. **Data collection:** Collect a batch of trajectories $\tau^1, \ldots, \tau^N$ by executing the current policy π_θ in the environment.
2. **Policy ("actor") update:** Update the policy parameters θ using the policy gradient estimator in Equation (45).
3. **Value ("critic") update:** Update the value function parameters φ by minimizing the loss in Equation (50) or Equation (51).
4. **Repeat:** Repeat the above steps until convergence.

3.1 n-Step Returns

So far, we've discussed two ways to estimate the expected return from a state s_t: the Monte Carlo return

$$\hat{R}^{\pi}_{\mathrm{MC},t} = \sum_{t'=t}^{H-1} \gamma^{t'-t} r(s_{t'}, a_{t'}), \tag{52}$$

and the (1-step) bootstrapped return

$$\hat{R}^{\pi}_{\mathrm{C},t} = r(s_t, a_t) + \gamma \hat{V}^{\pi}_{\varphi}(s_{t+1}). \tag{53}$$

The former is unbiased but has high variance, while the latter has lower variance but is biased due to modeling errors in $\hat{V}^{\pi}_{\varphi}$. Can we find a middle ground between these two extremes?

A natural way to interpolate between these two is to use *n-step returns* defined as follows:

$$\hat{R}^{\pi}_{n,t} = \sum_{t'=t}^{t+n-1} \gamma^{t'-t} r(s_{t'}, a_{t'}) + \gamma^{n} \hat{V}^{\pi}_{\varphi}(s_{t+n}). \tag{54}$$

where n is a hyperparameter that controls the bias-variance tradeoff. Note that $n = 1$ recovers the 1-step bootstrapped return, while $n = H - t$ recovers the Monte Carlo return (under the assumption that $\hat{V}^{\pi}_{\varphi}(s_H) = 0$ for convenience). In practice, by tuning n, we can often find a sweet spot that achieves a better bias-variance tradeoff than either of the two extremes.

3.2 Generalized Advantage Estimation

What if we combine the n-step returns from *all possible values* of n, instead of just picking one? This is the idea behind generalized advantage estimation (GAE), which is another widely used technique to control the bias-variance tradeoff in policy gradient methods.

Before explaining GAE, let's first define the Monte Carlo, bootstrapped, and n-step *advantage* estimates as follows:

$$\hat{A}^{\pi}_{\mathrm{MC}}(s_t, a_t) = \hat{R}^{\pi}_{\mathrm{MC},t} - \hat{V}^{\pi}_{\varphi}(s_t) = \left(\sum_{t'=t}^{H-1} \gamma^{t'-t} r(s_{t'}, a_{t'}) \right) - \hat{V}^{\pi}_{\varphi}(s_t), \tag{55}$$

$$\hat{A}^{\pi}_{\mathrm{C}}(s_t, a_t) = \hat{R}^{\pi}_{\mathrm{C},t} - \hat{V}^{\pi}_{\varphi}(s_t) = r(s_t, a_t) + \gamma \hat{V}^{\pi}_{\varphi}(s_{t+1}) - \hat{V}^{\pi}_{\varphi}(s_t), \tag{56}$$

$$\hat{A}^{\pi}_{n}(s_t, a_t) = \hat{R}^{\pi}_{n,t} - \hat{V}^{\pi}_{\varphi}(s_t) = \sum_{t'=t}^{t+n-1} \gamma^{t'-t} r(s_{t'}, a_{t'}) + \gamma^{n} \hat{V}^{\pi}_{\varphi}(s_{t+n}) - \hat{V}^{\pi}_{\varphi}(s_t). \tag{57}$$

Recall that the advantage function $A^{\pi}(s_t, a_t)$ is defined as $Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$, and we replace $Q^{\pi}(s_t, a_t)$ with the various return estimates to get the corresponding advantage estimates. Note also that knowing advantage estimates is sufficient for policy gradient estimation, because we typically subtract the baseline $\hat{V}^{\pi}_{\varphi}(s_t)$ from the reward-to-go (see Equations (43) and (44)).

GAE combines all the n-step advantage estimates $\hat{A}^{\pi}_{n}(s_t, a_t)$ for $n = 1, \ldots, H - t$ as an exponentially weighted average with a weighting factor $\lambda \in [0, 1]$:

$$\hat{A}^{\pi}_{\mathrm{GAE}}(s_t, a_t) = \frac{1 - \lambda}{1 - \lambda^{H-t}} \sum_{n=1}^{H-t} \lambda^{n-1} \hat{A}^{\pi}_{n}(s_t, a_t), \tag{58}$$

where $\frac{1-\lambda}{1-\lambda^{H-t}} = \frac{1}{1+\lambda+\cdots+\lambda^{H-t-1}}$ is a normalizing constant that ensures the weights sum to 1. The λ parameter in GAE plays a similar role to n in n-step returns. Do you see what happens when $\lambda \approx 0$ and when $\lambda \approx 1$, and how this relates to the bias-variance tradeoff?

One apparent issue with GAE is that it seems to require computing all the n-step advantage estimates $\hat{A}^{\pi}_{n}(s_t, a_t)$ for $n = 1, \ldots, H - t$, which can be computationally expensive. However, it turns out that we can compute the

GAE advantage estimates efficiently using a recursive formula. To see this, let's first assume that we are in the infinite-horizon case ($H = \infty$) for simplicity. Then, one can show that GAE can be equivalently written as

$$\hat{A}^{\pi}_{\text{GAE}}(s_t, a_t) = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} \hat{A}^{\pi}_{n}(s_t, a_t) \tag{59}$$

$$= \sum_{t'=t}^{\infty} (\gamma\lambda)^{t'-t}\, \hat{A}^{\pi}_{\text{C}}(s_{t'}, a_{t'}), \tag{60}$$

where we omit the derivation here (it is a bit tedious but not difficult, and you can find the details in the original GAE paper: https://arxiv.org/pdf/1506.02438.pdf).

In the finite-horizon case, we can similarly show that

$$\hat{A}^{\pi}_{\text{GAE}}(s_t, a_t) = \sum_{t'=t}^{H-1} (\gamma\lambda)^{t'-t}\, \hat{A}^{\pi}_{\text{C}}(s_{t'}, a_{t'}). \tag{61}$$

This enables us to compute the GAE advantage estimates efficiently using the following recursive formula:

$$\hat{A}^{\pi}_{\text{GAE}}(s_t, a_t) = \hat{A}^{\pi}_{\text{C}}(s_t, a_t) + \gamma\lambda\, \hat{A}^{\pi}_{\text{GAE}}(s_{t+1}, a_{t+1}). \tag{62}$$

This only requires computing the 1-step bootstrapped advantage estimates $\hat{A}^{\pi}_{\text{C}}(s_t, a_t)$ for all t, and then we can compute the GAE advantage estimates in a single backward loop over the trajectory.

GAE is not necessarily better or worse than n-step returns in general. GAE and n-step returns are just different ways to control the bias-variance tradeoff in policy gradient estimation, and they both have a hyperparameter (λ for GAE and n for n-step returns) that needs to be tuned in practice. In policy gradient methods, GAE is often more popular than n-step returns because λ is typically easier to tune than n and is often more robust across different tasks. On the other hand, in off-policy RL algorithms (which we will cover in the next couple of lectures), n-step returns are more popular than GAE because off-policy bias can be better controlled with n-step returns than with GAE.