ESE 6510 Note 3: Bias-Variance Trade Off

Jefferson Ng, Antonio Loquercio

February 14, 2026

1 Bias–Variance Trade-off in Policy Gradients

So far, we have seen that policy gradient methods update the policy using gradient estimates of the form

$$g = \mathbb{E}\left[\sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, \Psi_t\right],$$

where Ψ_t is a scalar signal that measures how good an action was.

Different choices of Ψ_t lead to different estimators of the policy gradients. These choices trade off variance and bias of the policy gradient estimator. High-variance gradients lead to unstable learning, whereas biased gradients may converge faster but to suboptimal (or unintended) solutions.

The goal of *advantage estimation* is therefore not to change the objective, but to construct Ψ_t in a way that appropriately balances bias and variance for the task at hand.

Different valid choices for Ψ_t

All of the following choices produce valid policy gradient estimators:

- $\Psi_t = \sum_{k=t}^{T-1} r_k$ (Monte Carlo return)
- $\Psi_t = \sum_{k=t}^{T-1} r_k - B(s_t)$ (Baseline)
- $\Psi_t = Q^\pi(s_t, a_t)$ (State–action value)
- $\Psi_t = A^\pi(s_t, a_t)$ (Advantage)
- $\Psi_t = r_t + V^\pi(s_{t+1}) - V^\pi(s_t)$ (TD residual)

These choices differ only in how the return is estimated, not in the objective being optimized.

1.1 Monte Carlo Returns, Q^π, and Advantage

Recall that in earlier sections, policy gradient updates were written using returns of the form $R_t - V^\pi(s_t)$. This quantity compares the observed return from time t against the policy's average performance at state s_t. We now refine this idea by introducing a more precise object: the state–action value function.

Formally, the state–action value function is defined as

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi \left[\sum_{k=t}^{T-1} r_k \;\middle|\; s_t, a_t \right],$$

which represents the expected total future reward when action a_t is taken in state s_t and the policy π is followed thereafter. In practice, we do not have access to this expectation. Instead, from a single trajectory we can compute the empirical return

$$\hat{Q}_t = \sum_{k=t}^{T-1} r_k,$$

which is a *singe sample* Monte Carlo estimate of $Q^\pi(s_t, a_t)$.

Monte Carlo targets: unbiased but high variance

The Monte Carlo return $\hat{Q}_t$ is an unbiased estimator of the true state–action value:

$$\mathbb{E}_{\tau\sim\pi}[\hat{Q}_t] = Q^\pi(s_t, a_t).$$

However, it depends on the entire future trajectory. Randomness in future actions, state transitions, and rewards can cause large fluctuations in $\hat{Q}_t$, resulting in high variance when used directly in gradient estimates.

Using Q^π, we can now define the *advantage function*:

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t).$$

The advantage measures how much better or worse a particular action performed relative to the policy's *average behavior* at that state. Recall that $V^\pi(s_t)$ is defined as the expected return obtained by sampling actions from the current policy at state s_t. In other words, it represents what the policy typically achieves when it reaches that state.

By subtracting $V^\pi(s_t)$, we are not comparing an action against the best possible outcome, but against what the policy itself usually does. This makes the learning signal well-aligned with the current policy: actions are reinforced only if they outperform the policy's own expectation, and discouraged otherwise.

1.2 Actor–Critic Algorithm

Any policy gradient algorithm that employs a learned value function as a baseline is classified as an *actor–critic* method. In this setting, the policy π_θ is referred to as the *actor*, since it selects actions, while the value function V_w serves as the *critic*, providing an estimate of expected return used to guide and stabilize the policy update. The defining characteristic of actor–critic algorithms is therefore the simultaneous learning of a policy and a value function.

Algorithm 1 Vanilla Policy Gradient with Learned Baseline (Actor–Critic)

```
1: for iteration $i = 1, 2, \ldots$ do
2:     Run policy $\pi_\theta(a_t \mid s_t)$ and collect $N$ trajectories
3:     for each timestep $t$ in each trajectory do
4:         Compute the single sample Monte Carlo return:
               $$\hat{Q}_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$$
5:         Compute advantage estimate:
               $$\hat{A}_t = \hat{Q}_t - V_w(s_t)$$
6:     end for
7:     Compute policy gradient estimate:
           $$\hat{g} = \frac{1}{N} \sum_n \sum_t \left[\nabla_\theta \log \pi_\theta(a_t^n \mid s_t^n)\, \hat{A}_t^n\right]$$
8:     Update actor parameters:
           $$\theta \leftarrow \theta + \alpha\, \hat{g}$$
9:     Update critic parameters using gradient descent using the observed returns:
           $$L(w) = \frac{1}{2N} \sum_n \sum_t |V_w(s_t^n) - \hat{Q}_t^n|^2$$
           $$w \leftarrow \mathrm{SGD}(L(w))$$
10: end for
```

1.3 Temporal Difference Learning, Bootstrapping, and the Discount Factor

In the previous section, we defined the advantage using Monte Carlo estimates of $Q^\pi(s_t, a_t)$, which depend on the entire future trajectory. While this estimator is unbiased, it can suffer from very high variance, especially in long horizon or stochastic environments. To further reduce variance, we can stop summing rewards early and instead use the critic's current estimate for the remainder of the trajectory. This idea, known as *bootstrapping*, leverages the following identity

$$Q^\pi(s_t, a_t) = \mathbb{E}_{a_t \sim \pi}[r(s_t, a_t)] + V^\pi(s_t). \tag{1}$$

If we have a (good) estimate of $V^\pi(s_t)$, we can compute an estimate of $Q^\pi(s_t, a_t)$ by using a single-sample estimator of only the first expectation

$$\hat{Q}_t = r_t + V_w(s_{t+1}) \tag{2}$$

. While single sample Monte Carlo estimates of the state-action function are unbiased, they are high variance. Bootstrapping from the value function increases bias if the critic is not perfect (which never is), but significantly reduce variance by avoiding dependence on long-term future outcomes. The bias introduced by bootstrapping can be particularly problematic if the system is highly chaotic and critic estimates are very off.

We can generalize the previous idea to n-step estimators that interpolate between single-sample Monte Carlo and one-step bootstrapping:

$$\hat{Q}_t^{(n)} = \sum_{k=t}^{t+n-1} r_k + V_w(s_{t+n}).$$

The horizon n controls how much empirical returns are used: larger n relies more on sampled rewards, while smaller n relies more heavily on the critic. This is effectively a mechanism to trade off between variance and bias.

Temporal-Difference (TD) Error

Using the bootstrapped return estimators introduced above, we can write the advantage as:

$$\hat{A}_t = r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t) \;=\; \delta_t.$$

The quantity δ_t is called the *temporal-difference (TD) error*. It measures the discrepancy between the current value estimate $V^\pi(s_t)$ and a one-step bootstrapped target $r_t + \gamma V^\pi(s_{t+1})$.
More generally, when using n-step bootstrapped returns, the corresponding error is referred to as the *n-step TD error* (TD(n)).

1.4 Discount Factors

Another method to trade-off bias and variance of the policy gradient uses *discount factors*. A discount factor $\gamma \in [0, 1]$ controls how future rewards are weighted:

$$\hat{Q}_{t,\gamma}^{(n)} = \sum_{k=t}^{t+n-1} \gamma^{k-t-1} r^k + \gamma^t V_w(s_{t+n}) \tag{3}$$

Introducing γ serves two purposes: it prioritizes near-term rewards and ensures that bootstrapped estimates remain well-behaved over long horizons.

Effective horizon induced by γ

Although trajectories may be long, discounting introduces an *effective planning horizon* of approximately

$$H_{\text{eff}} = 1 + \sum_k \gamma^k \approx \frac{1}{1-\gamma}.$$

Rewards far beyond this horizon contribute negligibly to the return, which helps control variance and stabilize learning.

In usual robotics-related RL tasks, γ is chosen close to 1 (e.g., 0.99) to encourage long-term planning. Smaller values place more emphasis on immediate rewards and can reduce variance, but may lead to short-sighted policies.

1.5 Generalized Advantage Estimation

Viewed through the lens of advantage estimation, temporal-difference methods replace the Monte Carlo estimate of $Q^\pi(s_t, a_t)$ with a biased but lower-variance approximation. This trade-off between bias and variance motivates more flexible estimators, which we will unify next using Generalized Advantage Estimation (GAE). GAE unifies many advantage estimators by exponentially weighting multi-step TD residuals,

$$\hat{A}_t^{\text{GAE}(\gamma,\lambda)} = \frac{1}{\sum_l w_l} \sum_l w_l \delta_{t+\ell} \quad w_l = \lambda^{l-1},$$

With a little algebra, one can show that

$$\hat{A}_t^{\text{GAE}(\gamma,\lambda)} = \sum_{\ell=0}^{\infty} (\gamma\lambda)^\ell \, \delta_{t+\ell}.$$

Expanding the first few terms makes the role of λ very explicit:

$$\hat{A}_t^{\text{GAE}(\gamma,\lambda)} = \delta_t + \gamma\lambda\, \delta_{t+1} + (\gamma\lambda)^2 \, \delta_{t+2} + (\gamma\lambda)^3 \, \delta_{t+3} + \cdots .$$

Special cases: $\lambda = 0$ and $\lambda = 1$

$\lambda = 0$ **(one-step TD advantage).** All higher-order terms vanish, so

$$\hat{A}_t^{\mathrm{GAE}(\gamma,0)} = \delta_t = r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t).$$

$\lambda = 1$ **(Monte Carlo-style advantage).** Now all TD residuals are included:

$$\hat{A}_t^{\mathrm{GAE}(\gamma,1)} = \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + \cdots .$$

Substituting $\delta_{t+\ell}$ and collecting terms yields the familiar form

$$\hat{A}_t^{\mathrm{GAE}(\gamma,1)} = \left(\sum_{k=t}^{T-1} \gamma^{k-t} r_k\right) - V^\pi(s_t),$$

i.e., the (discounted) Monte Carlo return minus the value baseline.

What does λ control?

The parameter $\lambda \in [0,1]$ interpolates between TD and Monte Carlo:

- $\lambda = 0$: pure TD(0), low variance, higher bias
- $\lambda = 1$: Monte Carlo advantage, unbiased, high variance

Intermediate λ values mix multiple TD residuals, giving a smooth bias–variance trade-off that can be tuned to the task.