11 Policy Gradient Estimation

The previous chapter discussed several ways to go about directly optimizing the parameters of a policy to maximize expected utility. In many applications, it is often useful to use the gradient of the utility with respect to the policy parameters to guide the optimization process. This chapter discusses several approaches to estimating this gradient from trajectory rollouts.[1] A major challenge with this approach is the variance of the estimate due to the stochastic nature of the trajectories arising from both the environment and our exploration of it. The next chapter will discuss how to use these algorithms to estimate gradients for the purpose of policy optimization.

[1] An additional resource on this topic is M. C. Fu, "Gradient Estimation," in *Simulation*, S. G. Henderson and B. L. Nelson, eds., Elsevier, 2006, pp. 575–616.

11.1 Finite Difference

Finite difference methods estimate the gradient of a function from small changes in its evaluation. Recall that the derivative of a univariate function f is

$$\frac{\mathrm{d}f}{\mathrm{d}x}(x) = \lim_{\delta \to 0} \frac{f(x+\delta) - f(x)}{\delta} \tag{11.1}$$

The derivative at x can be approximated by a sufficiently small step $\delta > 0$:

$$\frac{\mathrm{d}f}{\mathrm{d}x}(x) \approx \frac{f(x+\delta) - f(x)}{\delta} \tag{11.2}$$

This approximation is illustrated in figure 11.1.

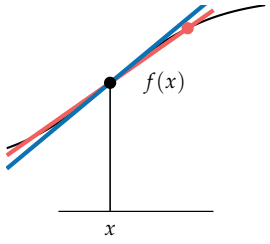


Figure 11.1. The finite difference method approximates the derivative of $f(x)$ using an evaluation of a point near x. The finite-difference approximation, in red, is not a perfect match for the true derivative, in blue.

The gradient of a multivariate function f with an input of length n is

$$\nabla f(\mathbf{x}) = \left[\frac{\partial f}{\partial x_1}(\mathbf{x}), \ldots, \frac{\partial f}{\partial x_n}(\mathbf{x})\right] \tag{11.3}$$

Finite differences can be applied to each dimension to estimate the gradient.

In the context of policy optimization, we want to estimate the gradient of the utility expected from following a policy parameterized by θ:

$$\nabla U(\theta) = \left[\frac{\partial U}{\partial \theta_1}(\theta), \ldots, \frac{\partial U}{\partial \theta_n}(\theta)\right] \tag{11.4}$$

$$\approx \left[\frac{U(\theta + \delta \mathbf{e}^{(1)}) - U(\theta)}{\delta}, \ldots, \frac{U(\theta + \delta \mathbf{e}^{(n)}) - U(\theta)}{\delta}\right] \tag{11.5}$$

where $\mathbf{e}^{(i)}$ is the ith *standard basis* vector, consisting of zeros except for the ith component, which is set to 1.

As discussed in section 10.1, we need to simulate policy rollouts to estimate $U(\theta)$. We can use algorithm 11.1 to generate trajectories. From these trajectories, we can compute their return and estimate the utility associated with the policy. Algorithm 11.2 implements the gradient estimate in equation (11.5) by simulating m rollouts for each component and averaging the returns.

```
function simulate(𝒫::MDP, s, π, d)
    τ = []
    for i = 1:d
        a = π(s)
        s′, r = 𝒫.TR(s,a)
        push!(τ, (s,a,r))
        s = s′
    end
    return τ
end
```

Algorithm 11.1. A method for generating a trajectory associated with problem `𝒫` starting in state `s` and executing policy `π` to depth `d`. It creates a vector `τ` containing state-action-reward tuples.

A major challenge in arriving at accurate estimates of the policy gradient is the fact that the variance of the trajectory rewards can be quite high. One approach to reduce the resulting variance in the gradient estimate is to have each rollout share the same random generator seeds.[2] This approach can be helpful, for example, in cases where one rollout happens to hit a low-probability transition early on. Other rollouts will have the same tendency due to the shared random generator, and their rewards will tend to be biased in the same way.

Policy representations have a significant effect on the policy gradient. Example 11.1 demonstrates the sensitivity of the policy gradient to the policy parameterization. Finite differences for policy optimization can perform poorly when the parameters differ in scale.

[2] This random seed sharing is used in the PEGASUS algorithm. A. Y. Ng and M. Jordan, "A Policy Search Method for Large MDPs and POMDPs," in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2000.

```
struct FiniteDifferenceGradient
    𝒫 # problem
    b # initial state distribution
    d # depth
    m # number of samples
    δ # step size
end

function gradient(M::FiniteDifferenceGradient, π, θ)
    𝒫, b, d, m, δ, γ, n = M.𝒫, M.b, M.d, M.m, M.δ, M.𝒫.γ, length(θ)
    Δθ(i) = [i == k ? δ : 0.0 for k in 1:n]
    R(τ) = sum(r*γ^(k-1) for (k, (s,a,r)) in enumerate(τ))
    U(θ) = mean(R(simulate(𝒫, rand(b), s->π(θ, s), d)) for i in 1:m)
    ΔU = [U(θ + Δθ(i)) - U(θ) for i in 1:n]
    return ΔU ./ δ
end
```

Algorithm 11.2. A method for estimating a policy gradient using finite differences for a problem $\mathcal{P}$, a parameterized policy $\pi(\theta, s)$, and a policy parameterization vector θ. Utility estimates are made from m rollouts to depth d. The step size is given by δ.

Consider a single-state, single-step MDP with a one-dimensional continuous action space and a reward function $R(s, a) = a$. In this case, larger actions produce higher rewards.

Suppose we have a stochastic policy π_θ that samples its action according to a uniform distribution between θ_1 and θ_2 for $\theta_2 > \theta_1$. The expected value is

$$U(\theta) = \mathbb{E}[a] = \int_{\theta_1}^{\theta_2} a \frac{1}{\theta_2 - \theta_1} \, \mathrm{d}a = \frac{\theta_1 + \theta_2}{2}$$

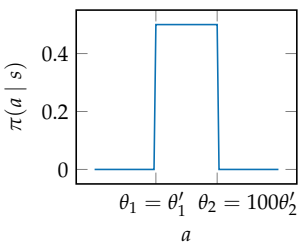
The policy gradient is

$$\nabla U(\theta) = [1/2, 1/2]$$

The policy could be reparameterized to draw actions from a uniform distribution between θ_1' and $100\theta_2'$, for $100\theta_2' > \theta_1'$. Now the expected reward is $(\theta_1' + 100\theta_2')/2$ and the policy gradient is $[1/2, 50]$.

The two parameterizations can represent the same policies, but they have very different gradients. Finding a suitable perturbation scalar for the second policy is much more difficult because the parameters vary widely in scale.

Example 11.1. An example of how policy parameterization has a significant impact on the policy gradient.



11.2 Regression Gradient

Instead of estimating the gradient at θ by taking a fixed step along each coordinate axis, as done in the previous section, we can use *linear regression*[3] to estimate the gradient from the results of random perturbations from θ. These perturbations are stored in a matrix as follows:[4]

$$\Delta\mathbf{\Theta} = \begin{bmatrix} (\Delta\theta^{(1)})^\top \\ \vdots \\ (\Delta\theta^{(m)})^\top \end{bmatrix} \tag{11.6}$$

More policy parameter perturbations will tend to produce better gradient estimates.[5]

For each of these perturbations, we perform a rollout and estimate the change in utility:[6]

$$\Delta\mathbf{U} = \left[U(\theta + \Delta\theta^{(1)}) - U(\theta), \ldots, U(\theta + \Delta\theta^{(m)}) - U(\theta)\right] \tag{11.7}$$

The policy gradient estimate using linear regression is then[7]

$$\nabla U(\theta) \approx \Delta\mathbf{\Theta}^{+}\Delta\mathbf{U} \tag{11.8}$$

Algorithm 11.3 provides an implementation of this approach in which the perturbations are drawn uniformly from a hypersphere with radius δ. Example 11.2 demonstrates this approach with a simple function.

[3] Linear regression is covered in section 8.6.

[4] This general approach is sometimes referred to as *simultaneous perturbation stochastic approximation* by J. C. Spall, *Introduction to Stochastic Search and Optimization*. Wiley, 2003. The general connection to linear regression is provided by J. Peters and S. Schaal, "Reinforcement Learning of Motor Skills with Policy Gradients," *Neural Networks*, vol. 21, no. 4, pp. 682–697, 2008.

[5] A recommended rule of thumb is to use about twice as many perturbations as the number of parameters.

[6] This equation shows the *forward difference*. Other finite-difference formulations, such as the central difference, can also be used.

[7] As discussed in section 8.6, $\mathbf{X}^{+}$ denotes the pseudoinverse of $\mathbf{X}$.

11.3 Likelihood Ratio

The *likelihood ratio* approach[8] to gradient estimation uses an analytical form of $\nabla\pi_\theta$ to improve our estimate of $\nabla U(\theta)$. Recall from equation (10.2) that

$$U(\theta) = \int p_\theta(\tau) R(\tau)\, \mathrm{d}\tau \tag{11.9}$$

[8] P. W. Glynn, "Likelihood Ratio Gradient Estimation for Stochastic Systems," *Communications of the ACM*, vol. 33, no. 10, pp. 75–84, 1990.

```
struct RegressionGradient
    𝒫 # problem
    b # initial state distribution
    d # depth
    m # number of samples
    δ # step size
end

function gradient(M::RegressionGradient, π, θ)
    𝒫, b, d, m, δ, γ = M.𝒫, M.b, M.d, M.m, M.δ, M.𝒫.γ
    Δθ = [δ.*normalize(randn(length(θ)), 2) for i = 1:m]
    R(τ) = sum(r*γ^(k-1) for (k, (s,a,r)) in enumerate(τ))
    U(θ) = R(simulate(𝒫, rand(b), s->π(θ,s), d))
    ΔU = [U(θ + Δθ) - U(θ) for Δθ in Δθ]
    return pinv(reduce(hcat, Δθ)') * ΔU
end
```

Algorithm 11.3. A method for estimating a policy gradient using finite differences for an MDP `𝒫`, a stochastic parameterized policy `π(θ, s)`, and a policy parameterization vector `θ`. Policy variation vectors are generated by normalizing normally distributed samples and scaling by a perturbation scalar `δ`. A total of `m` parameter perturbations are generated, and each is evaluated in a rollout from an initial state drawn from `b` to depth `d` and compared to the original policy parameterization.

Hence,

$$\nabla U(\theta) = \nabla_\theta \int p_\theta(\tau) R(\tau) \, \mathrm{d}\tau \tag{11.10}$$

$$= \int \nabla_\theta p_\theta(\tau) R(\tau) \, \mathrm{d}\tau \tag{11.11}$$

$$= \int p_\theta(\tau) \frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} R(\tau) \, \mathrm{d}\tau \tag{11.12}$$

$$= \mathbb{E}_\tau \left[\frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} R(\tau) \right] \tag{11.13}$$

The name for this method comes from this trajectory likelihood ratio. This likelihood ratio can be seen as a weight in likelihood weighted sampling (section 3.7) over trajectory rewards.

Applying the log derivative trick,[9] we have

$$\nabla U(\theta) = \mathbb{E}_\tau[\nabla_\theta \log p_\theta(\tau) R(\tau)] \tag{11.14}$$

We can estimate this expectation using trajectory rollouts. For each trajectory τ, we need to compute the product $\nabla_\theta \log p_\theta(\tau) R(\tau)$. Recall that $R(\tau)$ is the return associated with trajectory τ. If we have a stochastic policy,[10] the gradient $\nabla_\theta \log p_\theta(\tau)$ is

$$\nabla_\theta \log p_\theta(\tau) = \sum_{k=1}^{d} \nabla_\theta \log \pi_\theta(a^{(k)} \mid s^{(k)}) \tag{11.15}$$

[9] The log derivative trick was introduced in section 10.5. It uses the following equality:

$$\nabla_\theta \log p_\theta(\tau) = \nabla_\theta p_\theta(\tau) / p_\theta(\tau)$$

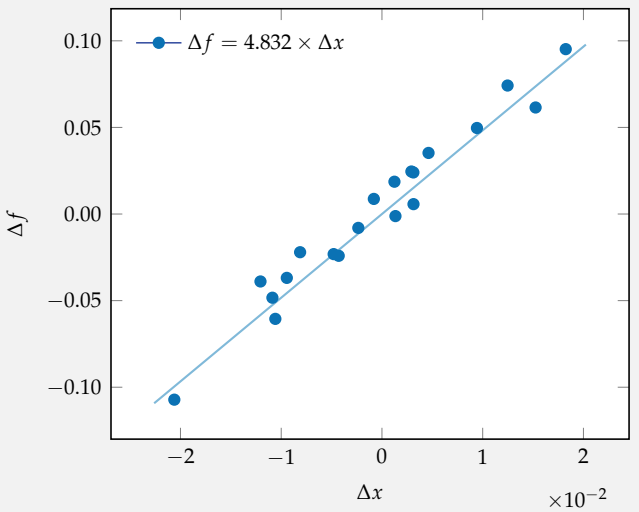
[10] We use $\pi_\theta(a \mid s)$ to represent the probability (either density or mass) the policy π_θ assigns to taking action a from state s.

Example 11.2. Using the regression gradient method on a one-dimensional function.

We would like to apply the regression gradient to estimate the gradient of a simple, one-dimensional function $f(x) = x^2$, evaluated at $x_0 = 2$ from $m = 20$ samples. To imitate the stochasticity inherent in policy evaluation, we add noise to the function evaluations. We generate a set of disturbances ΔX, sampled from $\mathcal{N}(0, \delta^2)$, and evaluate $f(x_0 + \Delta x) - f(x_0)$ for each disturbance Δx in ΔX. We can then estimate the one-dimensional gradient (or derivative) $\Delta X^+ \Delta F$ with this code:

```
f(x) = x^2 + 1e-2*randn()
m = 20
δ = 1e-2
ΔX = [δ.*randn() for i = 1:m]
x0 = 2.0
ΔF = [f(x0 + Δx) - f(x0) for Δx in ΔX]
pinv(ΔX) * ΔF
```

The samples and linear regression are shown here. The slope of the regression line is close to the exact solution of 4:



because $p_\theta(\tau)$ takes the form

$$p_\theta(\tau) = p(s^{(1)}) \prod_{k=1}^{d} T(s^{(k+1)} \mid s^{(k)}, a^{(k)}) \pi_\theta(a^{(k)} \mid s^{(k)}) \tag{11.16}$$

where $s^{(k)}$ and $a^{(k)}$ are the kth state and action, respectively, in trajectory τ. Algorithm 11.4 provides an implementation in which m trajectories are sampled to arrive at a gradient estimate. Example 11.3 illustrates the process.

If we have a deterministic policy, the gradient requires computing:[11]

$$\nabla_\theta \log p_\theta(\tau) = \nabla_\theta \log \left[p(s^{(1)}) \prod_{k=1}^{d} T(s^{(k+1)} \mid s^{(k)}, \pi_\theta(s^{(k)})) \right] \tag{11.17}$$

$$= \sum_{k=1}^{d} \nabla_\theta \pi_\theta(s^{(k)}) \frac{\partial}{\partial a^{(k)}} \log T(s^{(k+1)} \mid s^{(k)}, a^{(k)}) \tag{11.18}$$

Equations (11.17) and (11.18) require knowing the transition likelihood, which is in contrast with equation (11.15) for stochastic policies.

[11] Many problems have vector-valued actions $\mathbf{a} \in \mathbb{R}^n$. In this case, $\nabla_\theta \pi_\theta(s^{(k)})$ is replaced with a Jacobian matrix whose jth column is the gradient with respect to the jth action component, and the $\frac{\partial}{\partial a^{(k)}} \log T(s^{(k+1)} \mid s^{(k)}, a^{(k)})$ is replaced with an action gradient.

```
struct LikelihoodRatioGradient
    𝒫 # problem
    b # initial state distribution
    d # depth
    m # number of samples
    ∇logπ # gradient of log likelihood
end

function gradient(M::LikelihoodRatioGradient, π, θ)
    𝒫, b, d, m, ∇logπ, γ = M.𝒫, M.b, M.d, M.m, M.∇logπ, M.𝒫.γ
    πθ(s) = π(θ, s)
    R(τ) = sum(r*γ^(k-1) for (k, (s,a,r)) in enumerate(τ))
    ∇U(τ) = sum(∇logπ(θ, a, s) for (s,a) in τ)*R(τ)
    return mean(∇U(simulate(𝒫, rand(b), πθ, d)) for i in 1:m)
end
```

Algorithm 11.4. A method for estimating a policy gradient of a policy `π(s)` for an MDP `𝒫` with initial state distribution `b` using the likelihood ratio trick. The gradient with respect to the parameterization vector `θ` is estimated from `m` rollouts to depth `d` using the log policy gradients `∇logπ`.

11.4 Reward-to-Go

The likelihood ratio policy gradient method is unbiased but has high variance. Example 11.4 reviews bias and variance. The variance generally increases significantly with rollout depth due to the correlation between actions, states, and

Example 11.3. Applying the likelihood ratio trick to estimate a policy gradient in a simple problem.

Consider the single-step, single-state problem from example 11.1. Suppose we have a stochastic policy π_θ that samples its action according to a Gaussian distribution $\mathcal{N}(\theta_1, \theta_2^2)$, where θ_2^2 is the variance.

$$\begin{aligned}\log \pi_\theta(a \mid s) &= \log\left(\frac{1}{\sqrt{2\pi\theta_2^2}} \exp\left(-\frac{(a-\theta_1)^2}{2\theta_2^2}\right)\right) \\ &= -\frac{(a-\theta_1)^2}{2\theta_2^2} - \frac{1}{2}\log\left(2\pi\theta_2^2\right)\end{aligned}$$

The gradient of the log policy likelihood is

$$\begin{aligned}\frac{\partial}{\partial\theta_1}\log \pi_\theta(a \mid s) &= \frac{a-\theta_1}{\theta_2^2} \\ \frac{\partial}{\partial\theta_2}\log \pi_\theta(a \mid s) &= \frac{(a-\theta_1)^2 - \theta_2^2}{\theta_2^3}\end{aligned}$$

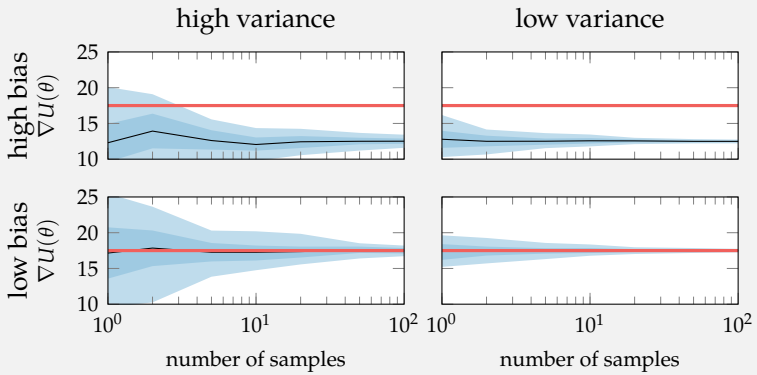
Suppose we run three rollouts with $\theta = [0, 1]$, taking actions $\{0.5, -1, 0.7\}$ and receiving the same rewards ($R(s, a) = a$). The estimated policy gradient is

$$\begin{aligned}\nabla U(\theta) &\approx \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \log p_\theta(\tau^{(i)}) R(\tau^{(i)}) \\ &= \frac{1}{3}\left(\begin{bmatrix}0.5\\-0.75\end{bmatrix} 0.5 + \begin{bmatrix}-1.0\\0.0\end{bmatrix}(-1) + \begin{bmatrix}0.7\\-0.51\end{bmatrix} 0.7\right) \\ &= [0.58, -0.244]\end{aligned}$$

Example 11.4. An empirical demonstration of bias and variance when estimating $\nabla U(\theta)$.

When estimating a quantity of interest from a collection of simulations, we generally want to use a scheme that has both low *bias* and low *variance*. In this chapter, we want to estimate $\nabla U(\theta)$. Generally, with more simulation samples, we can arrive at a better estimate. Some methods can lead to bias, where—even with infinitely many samples—it does not lead to an accurate estimate. Sometimes methods with nonzero bias may still be attractive if they also have low variance, meaning that they require fewer samples to converge.

Here are plots of the estimates from four notional methods for estimating $\nabla U(\theta)$. The true value is 17.5, as indicated by the red lines. We ran 100 simulations 100 times for each method. The variance decreases as the number of samples increases. The blue regions indicate the 5 % to 95 % and 25 % to 75 % empirical quantiles of the estimates.



rewards across time steps. The *reward-to-go* approach attempts to reduce the variance in the estimate.

To derive this approach, we begin by expanding equation (11.14):

$$\nabla U(\theta) = \mathbb{E}_\tau\left[\left(\sum_{k=1}^{d} \nabla_\theta \log \pi_\theta(a^{(k)} \mid s^{(k)})\right)\left(\sum_{k=1}^{d} r^{(k)}\gamma^{k-1}\right)\right] \tag{11.19}$$

Let $f^{(k)}$ replace $\nabla_\theta \log \pi_\theta(a^{(k)} \mid s^{(k)})$ for convenience. We then expand as follows:

$$\nabla U(\theta) = \mathbb{E}_\tau\left[\left(\sum_{k=1}^{d} f^{(k)}\right)\left(\sum_{k=1}^{d} r^{(k)}\gamma^{k-1}\right)\right] \tag{11.20}$$

$$= \mathbb{E}_\tau\left[\left(f^{(1)} + f^{(2)} + f^{(3)} + \cdots + f^{(d)}\right)\left(r^{(1)} + r^{(2)}\gamma + r^{(3)}\gamma^2 + \cdots + r^{(d)}\gamma^{d-1}\right)\right] \tag{11.21}$$

$$= \mathbb{E}_\tau\begin{bmatrix} f^{(1)}r^{(1)} + f^{(1)}r^{(2)}\gamma + f^{(1)}r^{(3)}\gamma^2 + \cdots + f^{(1)}r^{(d)}\gamma^{d-1} \\ +f^{(2)}r^{(1)} + f^{(2)}r^{(2)}\gamma + f^{(2)}r^{(3)}\gamma^2 + \cdots + f^{(2)}r^{(d)}\gamma^{d-1} \\ +f^{(3)}r^{(1)} + f^{(3)}r^{(2)}\gamma + f^{(3)}r^{(3)}\gamma^2 + \cdots + f^{(3)}r^{(d)}\gamma^{d-1} \\ \vdots \\ +f^{(d)}r^{(1)} + f^{(d)}r^{(2)}\gamma + f^{(d)}r^{(3)}\gamma^2 + \cdots + f^{(d)}r^{(d)}\gamma^{d-1} \end{bmatrix} \tag{11.22}$$

The first reward, $r^{(1)}$, is affected only by the first action. Thus, its contribution to the policy gradient should not depend on subsequent time steps. We can remove other such causality-violating terms as follows:[12]

[12] The term $\sum_{\ell=k}^{d} r^{(\ell)}\gamma^{\ell-k}$ is often called the *reward-to-go* from step k.

$$\nabla U(\theta) = \mathbb{E}_\tau\begin{bmatrix} f^{(1)}r^{(1)} + f^{(1)}r^{(2)}\gamma + f^{(1)}r^{(3)}\gamma^2 + \cdots + f^{(1)}r^{(d)}\gamma^{d-1} \\ + f^{(2)}r^{(2)}\gamma + f^{(2)}r^{(3)}\gamma^2 + \cdots + f^{(2)}r^{(d)}\gamma^{d-1} \\ + f^{(3)}r^{(3)}\gamma^2 + \cdots + f^{(3)}r^{(d)}\gamma^{d-1} \\ \vdots \\ + f^{(d)}r^{(d)}\gamma^{d-1} \end{bmatrix} \tag{11.23}$$

$$= \mathbb{E}_\tau\left[\sum_{k=1}^{d} \nabla_\theta \log \pi_\theta(a^{(k)} \mid s^{(k)})\left(\sum_{\ell=k}^{d} r^{(\ell)}\gamma^{\ell-1}\right)\right] \tag{11.24}$$

$$= \mathbb{E}_\tau\left[\sum_{k=1}^{d} \nabla_\theta \log \pi_\theta(a^{(k)} \mid s^{(k)})\left(\gamma^{k-1}\sum_{\ell=k}^{d} r^{(\ell)}\gamma^{\ell-k}\right)\right] \tag{11.25}$$

$$= \mathbb{E}_\tau\left[\sum_{k=1}^{d} \nabla_\theta \log \pi_\theta(a^{(k)} \mid s^{(k)})\gamma^{k-1} r^{(k)}_{\text{to-go}}\right] \tag{11.26}$$

Algorithm 11.5 provides an implementation of this.

Notice that the reward-to-go for a state-action pair (s, a) under a policy parameterized by θ is really an approximation of the state-action value from that state, $Q_\theta(s, a)$. The action value function, if known, can be used to obtain the policy gradient:

$$\nabla U(\theta) = \mathbb{E}_\tau\left[\sum_{k=1}^{d} \nabla_\theta \log \pi_\theta(a^{(k)} \mid s^{(k)}) \gamma^{k-1} Q_\theta\left(s^{(k)}, a^{(k)}\right)\right] \tag{11.27}$$

```
struct RewardToGoGradient
    𝒫 # problem
    b # initial state distribution
    d # depth
    m # number of samples
    ∇logπ # gradient of log likelihood
end

function gradient(M::RewardToGoGradient, π, θ)
    𝒫, b, d, m, ∇logπ, γ = M.𝒫, M.b, M.d, M.m, M.∇logπ, M.𝒫.γ
    πθ(s) = π(θ, s)
    R(τ, j) = sum(r*γ^(k-1) for (k,(s,a,r)) in zip(j:d, τ[j:end]))
    ∇U(τ) = sum(∇logπ(θ, a, s)*R(τ,j) for (j, (s,a,r)) in enumerate(τ))
    return mean(∇U(simulate(𝒫, rand(b), πθ, d)) for i in 1:m)
end
```

Algorithm 11.5. A method that uses reward-to-go for estimating a policy gradient of a policy `π(s)` for an MDP `𝒫` with initial state distribution `b`. The gradient with respect to the parameterization vector `θ` is estimated from `m` rollouts to depth `d` using the log policy gradient `∇logπ`.

11.5 Baseline Subtraction

We can further build on the approach presented in the previous section by subtracting a *baseline* value from the reward-to-go[13] to reduce the variance of the gradient estimate. This subtraction does not bias the gradient.

[13] We could also subtract a baseline from a state-action value.

We now subtract a baseline $r_{\text{base}}(s^{(k)})$:

$$\nabla U(\theta) = \mathbb{E}_\tau\left[\sum_{k=1}^{d} \nabla_\theta \log \pi_\theta(a^{(k)} \mid s^{(k)}) \gamma^{k-1}\left(r_{\text{to-go}}^{(k)} - r_{\text{base}}(s^{(k)})\right)\right] \tag{11.28}$$

To show that baseline subtraction does not bias the gradient, we first expand:

$$\nabla U(\theta) = \mathbb{E}_\tau\left[\sum_{k=1}^{d} \nabla_\theta \log \pi_\theta(a^{(k)} \mid s^{(k)}) \gamma^{k-1} r_{\text{to-go}}^{(k)} - \sum_{k=1}^{d} \nabla_\theta \log \pi_\theta(a^{(k)} \mid s^{(k)}) \gamma^{k-1} r_{\text{base}}(s^{(k)})\right] \tag{11.29}$$

The *linearity of expectation* states that $\mathbb{E}[a + b] = \mathbb{E}[a] + \mathbb{E}[b]$, so it is sufficient to prove that equation (11.29) is equivalent to equation (11.26), if for each step k, the expected associated baseline term is $\mathbf{0}$:

$$\mathbb{E}_\tau\left[\nabla_\theta \log \pi_\theta(a^{(k)} \mid s^{(k)})\gamma^{k-1} r_{\text{base}}(s^{(k)})\right] = \mathbf{0} \tag{11.30}$$

We begin by converting the expectation into nested expectations, as illustrated in figure 11.2:

$$\mathbb{E}_\tau\left[\nabla_\theta \log \pi_\theta(a^{(k)} \mid s^{(k)})\gamma^{k-1} r_{\text{base}}(s^{(k)})\right] = \mathbb{E}_{\tau_{1:k}}\left[\mathbb{E}_{\tau_{k+1:d}}\left[\nabla_\theta \log \pi_\theta(a^{(k)} \mid s^{(k)})\gamma^{k-1} r_{\text{base}}(s^{(k)})\right]\right] \tag{11.31}$$

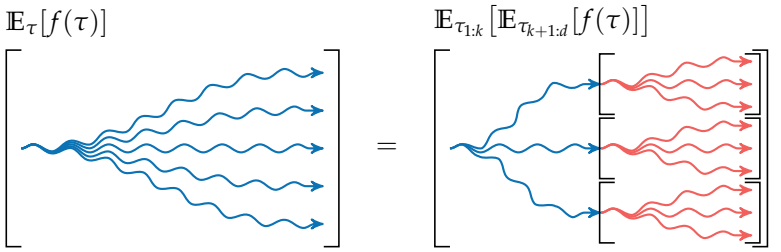


Figure 11.2. The expectation of a function of trajectories sampled from a policy can be viewed as an expectation over a nested expectation of subtrajectories. For a mathematical derivation, see exercise 11.4.

We continue with our derivation, using the same log derivative trick from section 11.3:

$$\begin{aligned}
&\mathbb{E}_{\tau_{1:k}}\left[\mathbb{E}_{\tau_{k+1:d}}\left[\nabla_\theta \log \pi_\theta(a^{(k)} \mid s^{(k)})\gamma^{k-1} r_{\text{base}}(s^{(k)})\right]\right] \\
&\quad = \mathbb{E}_{\tau_{1:k}}\left[\gamma^{k-1} r_{\text{base}}(s^{(k)})\ \mathbb{E}_{\tau_{k+1:d}}\left[\nabla_\theta \log \pi_\theta(a^{(k)} \mid s^{(k)})\right]\right] && (11.32)\\
&\quad = \mathbb{E}_{\tau_{1:k}}\left[\gamma^{k-1} r_{\text{base}}(s^{(k)})\ \mathbb{E}_{a^{(k)}}\left[\nabla_\theta \log \pi_\theta(a^{(k)} \mid s^{(k)})\right]\right] && (11.33)\\
&\quad = \mathbb{E}_{\tau_{1:k}}\left[\gamma^{k-1} r_{\text{base}}(s^{(k)}) \int \nabla_\theta \log \pi_\theta(a^{(k)} \mid s^{(k)}) \pi_\theta(a^{(k)} \mid s^{(k)})\, \mathrm{d}a^{(k)}\right] && (11.34)\\
&\quad = \mathbb{E}_{\tau_{1:k}}\left[\gamma^{k-1} r_{\text{base}}(s^{(k)}) \int \frac{\nabla_\theta \pi_\theta(a^{(k)} \mid s^{(k)})}{\pi_\theta(a^{(k)} \mid s^{(k)})} \pi_\theta(a^{(k)} \mid s^{(k)})\, \mathrm{d}a^{(k)}\right] && (11.35)\\
&\quad = \mathbb{E}_{\tau_{1:k}}\left[\gamma^{k-1} r_{\text{base}}(s^{(k)})\, \nabla_\theta \int \pi_\theta(a^{(k)} \mid s^{(k)})\, \mathrm{d}a^{(k)}\right] && (11.36)\\
&\quad = \mathbb{E}_{\tau_{1:k}}\left[\gamma^{k-1} r_{\text{base}}(s^{(k)})\, \nabla_\theta 1\right] && (11.37)\\
&\quad = \mathbb{E}_{\tau_{1:k}}\left[\gamma^{k-1} r_{\text{base}}(s^{(k)})\, \mathbf{0}\right] && (11.38)
\end{aligned}$$

Therefore, subtracting a term $r_{\text{base}}(s^{(k)})$ does not bias the estimate. This derivation assumed continuous state and action spaces. The same result applies to discrete spaces.

We can choose a different $r_{\text{base}}(s)$ for every component of the gradient, and we will select them to minimize the variance. For simplicity, we will drop the dependence on s and treat each baseline component as constant.[14] For compactness in writing the equations in our derivation, we define

$$\ell_i(a, s, k) = \gamma^{k-1} \frac{\partial}{\partial \theta_i} \log \pi_\theta(a \mid s) \tag{11.39}$$

[14] Some methods approximate a state-dependent baseline using $r_{\text{base}}(s^{(k)}) = \boldsymbol{\phi}(s^{(k)})^\top \mathbf{w}$. Selecting appropriate baseline functions tends to be difficult. J. Peters and S. Schaal, "Reinforcement Learning of Motor Skills with Policy Gradients," *Neural Networks*, vol. 21, no. 4, pp. 682–697, 2008.

The variance of the ith component of our gradient estimate in equation (11.28) is

$$\mathop{\mathbb{E}}_{a,s,r_{\text{to-go}},k}\left[\left(\ell_i(a, s, k)\left(r_{\text{to-go}} - r_{\text{base},i}\right)\right)^2\right] - \mathop{\mathbb{E}}_{a,s,r_{\text{to-go}},k}\left[\ell_i(a, s, k)\left(r_{\text{to-go}} - r_{\text{base},i}\right)\right]^2 \tag{11.40}$$

where the expectation is over the $(a, s, r_{\text{to-go}})$ tuples in our trajectory samples, and k is each tuple's depth.

We have just shown that the second term is zero. Hence, we can focus on choosing $r_{\text{base},i}$ to minimize the first term by taking the derivative with respect to the baseline and setting it to zero:

$$\begin{aligned}
&\frac{\partial}{\partial r_{\text{base},i}} \mathop{\mathbb{E}}_{a,s,r_{\text{to-go}},k}\left[\left(\ell_i(a, s, k)\left(r_{\text{to-go}} - r_{\text{base},i}\right)\right)^2\right] \\
&\quad = \frac{\partial}{\partial r_{\text{base},i}}\left(\mathop{\mathbb{E}}_{a,s,r_{\text{to-go}},k}\left[\ell_i(a, s, k)^2 r_{\text{to-go}}^2\right] - 2\mathop{\mathbb{E}}_{a,s,r_{\text{to-go}},k}\left[\ell_i(a, s, k)^2 r_{\text{to-go}} r_{\text{base},i}\right] + r_{\text{base},i}^2 \mathop{\mathbb{E}}_{a,s,k}\left[\ell_i(a, s, k)^2\right]\right) && (11.41) \\
&\quad = -2\mathop{\mathbb{E}}_{a,s,r_{\text{to-go}},k}\left[\ell_i(a, s, k)^2 r_{\text{to-go}}\right] + 2 r_{\text{base},i} \mathop{\mathbb{E}}_{a,s,k}\left[\ell_i(a, s, k)^2\right] = 0 && (11.42)
\end{aligned}$$

Solving for $r_{\text{base},i}$ yields the baseline component that minimizes the variance:

$$r_{\text{base},i} = \frac{\mathbb{E}_{a,s,r_{\text{to-go}},k}\left[\ell_i(a, s, k)^2 r_{\text{to-go}}\right]}{\mathbb{E}_{a,s,k}\left[\ell_i(a, s, k)^2\right]} \tag{11.43}$$

```
struct BaselineSubtractionGradient
    𝒫 # problem
    b # initial state distribution
    d # depth
    m # number of samples
    ∇logπ # gradient of log likelihood
end

function gradient(M::BaselineSubtractionGradient, π, θ)
    𝒫, b, d, m, ∇logπ, γ = M.𝒫, M.b, M.d, M.m, M.∇logπ, M.𝒫.γ
    πθ(s) = π(θ, s)
    ℓ(a, s, k) = ∇logπ(θ, a, s)*γ^(k-1)
    R(τ, k) = sum(r*γ^(j-1) for (j,(s,a,r)) in enumerate(τ[k:end]))
    numer(τ) = sum(ℓ(a,s,k).^2*R(τ,k) for (k,(s,a,r)) in enumerate(τ))
    denom(τ) = sum(ℓ(a,s,k).^2 for (k,(s,a)) in enumerate(τ))
    base(τ) = numer(τ) ./ denom(τ)
    trajs = [simulate(𝒫, rand(b), πθ, d) for i in 1:m]
    rbase = mean(base(τ) for τ in trajs)
    ∇U(τ) = sum(ℓ(a,s,k).*(R(τ,k).-rbase) for (k,(s,a,r)) in enumerate(τ))
    return mean(∇U(τ) for τ in trajs)
end
```

Algorithm 11.6. Likelihood ratio gradient estimation with reward-to-go and baseline subtraction for an MDP $\mathcal{P}$, policy π, and initial state distribution b. The gradient with respect to the parameterization vector θ is estimated from m rollouts to depth d using the log policy gradients $\nabla\log\pi$.

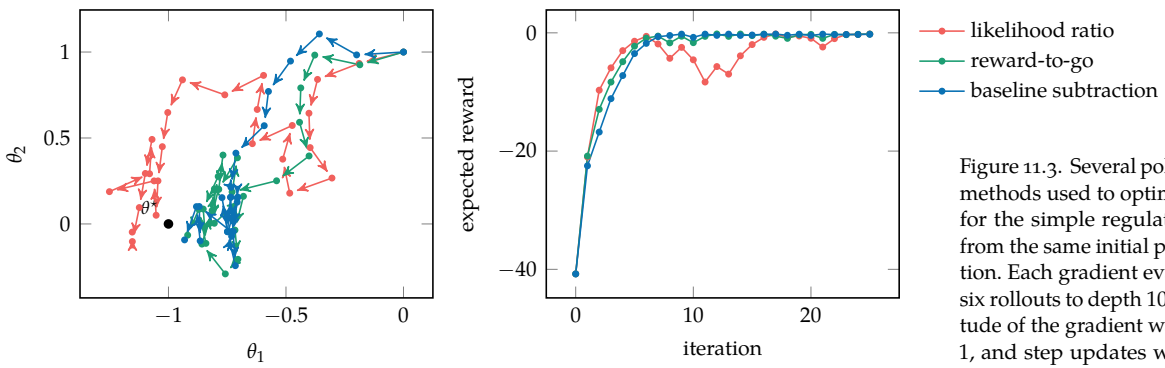


Figure 11.3. Several policy gradient methods used to optimize policies for the simple regulator problem from the same initial parameterization. Each gradient evaluation ran six rollouts to depth 10. The magnitude of the gradient was limited to 1, and step updates were applied with step size 0.2. The optimal policy parameterization is shown in black.

It is common to use likelihood ratio policy gradient estimation with this baseline subtraction (algorithm 11.6).[15]. Figure 11.3 compares the methods discussed here.

[15] This combination is used in the class of algorithms called *REINFORCE* as introduced by R. J. Williams, "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning," *Machine Learning*, vol. 8, pp. 229–256, 1992.

Qualitatively, when considering the gradient contribution of state-action pairs, what we really care about is the relative value of one action over another. If all actions in a particular state produce the same high value, there is no real signal in the gradient, and baseline subtraction can zero that out. We want to identify the actions that produce a higher value than others, regardless of the mean value across actions.

An alternative to the action value is the *advantage*, $A(s, a) = Q(s, a) - U(s)$. Using the state value function in baseline subtraction produces the advantage. The policy gradient using the advantage is unbiased and typically has much lower variance. The gradient computation takes the following form:

$$\nabla U(\theta) = \mathbb{E}_\tau \left[\sum_{k=1}^{d} \nabla_\theta \log \pi_\theta(a^{(k)} \mid s^{(k)}) \gamma^{k-1} A_\theta\left(s^{(k)}, a^{(k)}\right) \right] \tag{11.44}$$

As with the state and action value functions, the advantage function is typically unknown. Other methods, covered in chapter 13, are needed to approximate it.

11.6 Summary

- A gradient can be estimated using finite differences.
- Linear regression can also be used to provide more robust estimates of the policy gradient.
- The likelihood ratio can be used to derive a form of the policy gradient that does not depend on the transition model for stochastic policies.
- The variance of the policy gradient can be significantly reduced using the reward-to-go and baseline subtraction.

11.7 Exercises

Exercise 11.1. If we estimate the expected discounted return of a given parameterized policy π_θ defined by an n-dimensional vector of parameters θ using m rollouts, how many total rollouts do we need to perform to compute the policy gradient using a finite difference approach?

Solution: In order to estimate the policy gradient using a finite difference approach, we need to estimate the utility of the policy given the current parameter vector $U(\theta)$, as well as all n variations of the current parameter vector $U(\theta + \delta \mathbf{e}^{(i)})$ for $i = 1:n$. Since we estimate each of these using m rollouts, we need to perform a total of $m(n+1)$ rollouts.

Exercise 11.2. Suppose we have a robotic arm with which we are able to run experiments manipulating a wide variety of objects. We would like to use the likelihood ratio policy gradient or one of its extensions to train a policy that is efficient at picking up and moving these objects. Would it be more straightforward to use a deterministic or a stochastic policy, and why?

Solution: The likelihood ratio policy gradient requires an explicit representation of the transition likelihood when used with deterministic policies. Specifying an accurate explicit transition model for a real-world robotic arm manipulation task would be challenging. Computing the policy gradient for a stochastic policy does not require having an explicit representation of the transition likelihood, making the use of a stochastic policy more straightforward.

Exercise 11.3. Consider policy gradients of the form

$$\nabla_\theta U(\theta) = \mathbb{E}_\tau \left[\sum_{k=1}^{d} \gamma^{k-1} y \, \nabla_\theta \log \pi_\theta(a^{(k)} \mid s^{(k)}) \right]$$

Which of the following values of y result in a valid policy gradient? Explain why.

(a) $\gamma^{1-k} \sum_{\ell=1}^{\infty} r^{(\ell)} \gamma^{\ell-1}$

(b) $\sum_{\ell=k}^{\infty} r^{(\ell)} \gamma^{\ell-k}$

(c) $\left(\sum_{\ell=k}^{\infty} r^{(\ell)} \gamma^{\ell-k} \right) - r_{\text{base}}(s^{(k)})$

(d) $U(s^{(k)})$

(e) $Q(s^{(k)}, a^{(k)})$

(f) $A(s^{(k)}, a^{(k)})$

(g) $r^{(k)} + \gamma U(s^{(k+1)}) - U(s^{(k)})$

Solution:

(a) $\sum_{\ell=1}^{\infty} r^{(\ell)}$ results in the total discounted reward, as

$$\gamma^{k-1}\gamma^{1-k}\sum_{\ell=1}^{\infty} r^{(\ell)}\gamma^{\ell-1} = \sum_{\ell=1}^{\infty} r^{(\ell)}\gamma^{\ell-1}$$

and produces a valid policy gradient, as given in equation (11.19).

(b) $\sum_{\ell=k}^{\infty} r^{(\ell)}\gamma^{\ell-k}$ is the reward-to-go and produces a valid policy gradient, as given in equation (11.26).

(c) $\left(\sum_{\ell=k}^{\infty} r^{(\ell)}\right) - r_{\text{base}}(s^{(k)})$ is the baseline subtracted reward-to-go and produces a valid policy gradient, as given in equation (11.28).

(d) $U(s^{(k)})$ is the state value function and does not produce a valid policy gradient.

(e) $Q(s^{(k)}, a^{(k)})$ is the state-action value function and produces a valid policy gradient, as given in equation (11.27).

(f) $A(s^{(k)}, a^{(k)})$ is the advantage function and produces a valid policy gradient, as given in equation (11.44).

(g) $r^{(k)} + \gamma U(s^{(k+1)}) - U(s^{(k)})$ is the temporal difference residual (to be discussed further in chapter 13) and produces a valid policy gradient because it is an unbiased approximation of the advantage function.

Exercise 11.4. Show that $\mathbb{E}_{\tau\sim\pi}[f(\tau)] = \mathbb{E}_{\tau_{1:k}\sim\pi}[\mathbb{E}_{\tau_{k:d}\sim\pi}[f(\tau)]]$ for step k.

Solution: The nested expectations can be proven by writing the expectation in integral form and then converting back:

$$\begin{aligned}
&\mathbb{E}_{\tau\sim\pi}[f(\tau)] = \\
&= \int p(\tau) f(\tau)\, \mathrm{d}\tau \\
&= \int \left(p(s^{(1)}) \prod_{k=1}^{d} p(s^{(k+1)} \mid s^{(k)}, a^{(k)}) \pi(a^{(k)} \mid s^{(k)}) \right) f(\tau)\, \mathrm{d}\tau \\
&= \int\int\int\int \cdots \int \left(p(s^{(1)}) \prod_{k=1}^{d} p(s^{(k+1)} \mid s^{(k)}, a^{(k)}) \pi(a^{(k)} \mid s^{(k)}) \right) f(\tau)\, \mathrm{d}s^{(d)} \cdots \mathrm{d}a^{(2)}\, \mathrm{d}s^{(2)}\, \mathrm{d}a^{(1)}\, \mathrm{d}s^{(1)} \\
&= \mathop{\mathbb{E}}_{\tau_{1:k}\sim\pi}\left[\int\int\int\int \cdots \int \left(\prod_{q=k}^{d} p(s^{(q+1)} \mid s^{(q)}, a^{(q)}) \pi(a^{(q)} \mid s^{(q)}) \right) f(\tau)\, \mathrm{d}s^{(d)} \cdots \mathrm{d}a^{(k+1)}\, \mathrm{d}s^{(k+1)}\, \mathrm{d}a^{(k)}\, \mathrm{d}s^{(k)} \right] \\
&= \mathbb{E}_{\tau_{1:k}\sim\pi}[\mathbb{E}_{\tau_{k:d}\sim\pi}[f(\tau)]]
\end{aligned}$$

Exercise 11.5. Our implementation of the regression gradient (algorithm 11.3) fits a linear mapping from perturbations to the difference in returns, $U(\theta + \Delta\theta^{(i)}) - U(\theta)$. We evaluate $U(\theta + \Delta\theta^{(i)})$ and $U(\theta)$ for each of the m perturbations, thus reevaluating $U(\theta)$ a total of m times. How might we reallocate the samples in a more effective manner?

Solution: One approach is to evaluate $U(\theta)$ once and use the same value for each perturbation, thereby conducting only $m + 1$ evaluations. Having an accurate estimate of $U(\theta)$ is particularly important for an accurate regression gradient estimate. An alternative is to still compute $U(\theta)$ once, but use m rollouts, thus preserving the total number of rollouts per iteration. This approach uses the same amount of computation as algorithm 11.3, but it can produce a more reliable gradient estimate.