13 Actor-Critic Methods

The previous chapter discussed ways to improve a parameterized policy through gradient information estimated from rollouts. This chapter introduces *actor-critic methods*, which use an estimate of a value function to help direct the optimization. The actor, in this context, is the policy, and the critic is the value function. Both are trained in parallel. We will discuss several methods that differ in whether they approximate the value function, advantage function, or action value function. Most focus on stochastic policies, but we will also discuss one method that supports deterministic policies that output continuous actions. Finally, we will discuss a way to incorporate an online method for generating more informative trajectories for training the actor and critic.

13.1 Actor-Critic

In actor-critic methods, we have an actor represented by a policy $\pi_{\boldsymbol{\theta}}$, parameterized by $\boldsymbol{\theta}$ with the help of a critic that provides an estimate of the value function $U_{\boldsymbol{\phi}}(s)$, $Q_{\boldsymbol{\phi}}(s,a)$, or $A_{\boldsymbol{\phi}}(s,a)$ parameterized by $\boldsymbol{\phi}$. We will start this chapter with a simple actor-critic approach in which the optimization of $\pi_{\boldsymbol{\theta}}$ is done through gradient ascent, with the gradient of our objective being the same as in equation (11.44):

$$\nabla U(\boldsymbol{\theta}) = \mathbb{E}_\tau\left[\sum_{k=1}^{d} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a^{(k)} \mid s^{(k)})\gamma^{k-1} A_{\boldsymbol{\theta}}\left(s^{(k)}, a^{(k)}\right)\right] \tag{13.1}$$

The advantage when following a policy parameterized by $\boldsymbol{\theta}$ can be estimated using a set of observed transitions from s to s' with reward r:

$$A_{\boldsymbol{\theta}}(s,a) = \mathbb{E}_{r,s'}\left[r + \gamma U^{\pi_{\boldsymbol{\theta}}}(s') - U^{\pi_{\boldsymbol{\theta}}}(s)\right] \tag{13.2}$$

The $r + \gamma U^{\pi_\theta}(s') - U^{\pi_\theta}(s)$ inside the expectation is referred to as the *temporal difference residual*.

The critic allows us to estimate the true value function U^{π_θ} when following π_θ, resulting in the following gradient for the actor:

$$\nabla U(\theta) \approx \mathbb{E}_\tau\left[\sum_{k=1}^{d} \nabla_\theta \log \pi_\theta(a^{(k)} \mid s^{(k)})\gamma^{k-1}\left(r^{(k)} + \gamma U_\phi(s^{(k+1)}) - U_\phi(s^{(k)})\right)\right] \tag{13.3}$$

This expectation can be estimated through rollout trajectories, as done in chapter 11.

The critic is also updated through gradient optimization. We want to find a ϕ that minimizes our loss function:

$$\ell(\phi) = \frac{1}{2}\,\mathbb{E}_s\left[\left(U_\phi(s) - U^{\pi_\theta}(s)\right)^2\right] \tag{13.4}$$

To minimize this objective, we can take steps in the opposite direction of the gradient:

$$\nabla \ell(\phi) = \mathbb{E}_s\left[\left(U_\phi(s) - U^{\pi_\theta}(s)\right)\nabla_\phi U_\phi(s)\right] \tag{13.5}$$

Of course, we do not know U^{π_θ} exactly, but it can be estimated using the reward-to-go along rollout trajectories, resulting in

$$\nabla \ell(\phi) = \mathbb{E}_\tau\left[\sum_{k=1}^{d}\left(U_\phi(s^{(k)}) - r_{\text{to-go}}^{(k)}\right)\nabla_\phi U_\phi(s^{(k)})\right] \tag{13.6}$$

where $r_{\text{to-go}}^{(k)}$ is the reward-to-go at step k in a particular trajectory τ.

Algorithm 13.1 shows how to estimate $\nabla U(\theta)$ and $\nabla \ell(\phi)$ from rollouts. With each iteration, we step θ in the direction of $\nabla U(\theta)$ to maximize utility, and we step ϕ in the opposite direction of $\nabla \ell(\phi)$ to minimize our loss. This approach can become unstable due to the dependency between the estimation of θ and ϕ, but this approach has worked well for a variety of problems. It is a common practice to update the policy more frequently than the value function to improve stability. The implementations in this chapter can easily be adapted to update the value function only for a subset of the iterations that the policy is updated.

```
struct ActorCritic
    𝒫     # problem
    b     # initial state distribution
    d     # depth
    m     # number of samples
    ∇logπ # gradient of log likelihood ∇logπ(θ,a,s)
    U     # parameterized value function U(ϕ, s)
    ∇U    # gradient of value function ∇U(ϕ,s)
end

function gradient(M::ActorCritic, π, θ, ϕ)
    𝒫, b, d, m, ∇logπ = M.𝒫, M.b, M.d, M.m, M.∇logπ
    U, ∇U, γ = M.U, M.∇U, M.𝒫.γ
    πθ(s) = π(θ, s)
    R(τ,j) = sum(r*γ^(k-1) for (k,(s,a,r)) in enumerate(τ[j:end]))
    A(τ,j) = τ[j][3] + γ*U(ϕ,τ[j+1][1]) - U(ϕ,τ[j][1])
    ∇Uθ(τ) = sum(∇logπ(θ,a,s)*A(τ,j)*γ^(j-1) for (j, (s,a,r))
                in enumerate(τ[1:end-1]))
    ∇ℓϕ(τ) = sum((U(ϕ,s) - R(τ,j))*∇U(ϕ,s) for (j, (s,a,r))
                in enumerate(τ))
    trajs = [simulate(𝒫, rand(b), πθ, d) for i in 1:m]
    return mean(∇Uθ(τ) for τ in trajs), mean(∇ℓϕ(τ) for τ in trajs)
end
```

Algorithm 13.1. A basic actor-critic method for computing both a policy gradient and a value function gradient for an MDP `𝒫` with initial state distribution `b`. The policy `π` is parameterized by `θ` and has a log-gradient `∇logπ`. The value function `U` is parameterized by `ϕ` and the gradient of its objective function is `∇U`. This method runs `m` rollouts to depth `d`. The results are used to update `θ` and `ϕ`. The policy parameterization is updated in the direction of `∇θ` to maximize the expected value, whereas the value function parameterization is updated in the negative direction of `∇ϕ` to minimize the value loss.

13.2 *Generalized Advantage Estimation*

Generalized advantage estimation (algorithm 13.2) is an actor-critic method that uses a more general version of the advantage estimate shown in equation (13.2) that allows us to balance between bias and variance.[1] Approximation with the temporal difference residual has low variance, but it introduces bias due to a potentially inaccurate U_ϕ used to approximate U^{π_θ}. An alternative is to replace $r + \gamma U^{\pi_\theta}(s')$ with the sequence of rollout rewards $r_1, \ldots, r_d$:

$$\begin{aligned} A_\theta(s,a) &= \mathbb{E}_{r_1,\ldots,r_d}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots + \gamma^{d-1} r_d - U^{\pi_\theta}(s)\right] && (13.7) \\ &= \mathbb{E}_{r_1,\ldots,r_d}\left[-U^{\pi_\theta}(s) + \sum_{\ell=1}^{d} \gamma^{\ell-1} r_\ell\right] && (13.8) \end{aligned}$$

We can obtain an unbiased estimate of this expectation through rollout trajectories, as done in the policy gradient estimation methods (chapter 11). However, the estimate is high variance, meaning that we need many samples to arrive at an accurate estimate.

[1] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation," in *International Conference on Learning Representations (ICLR)*, 2016. arXiv: 1506.02438v6.

The approach taken by generalized advantage estimation is to balance between these two extremes of using temporal difference residuals and full rollouts. We define $\hat{A}^{(k)}$ to be the advantage estimate obtained from k steps of a rollout and the utility associated with the resulting state s':

$$\hat{A}^{(k)}(s,a) = \mathbb{E}_{r_1,\ldots,r_k,s'}\left[r_1 + \gamma r_2 + \cdots + \gamma^{k-1} r_k + \gamma^k U^{\pi_\theta}(s') - U^{\pi_\theta}(s)\right] \quad (13.9)$$

$$= \mathbb{E}_{r_1,\ldots,r_k,s'}\left[-U^{\pi_\theta}(s) + \gamma^k U^{\pi_\theta}(s') + \sum_{\ell=1}^{k} \gamma^{\ell-1} r_\ell\right] \quad (13.10)$$

An alternative way to write $\hat{A}^{(k)}$ is in terms of an expectation over temporal difference residuals. We can define

$$\delta_t = r_t + \gamma U(s_{t+1}) - U(s_t) \quad (13.11)$$

where s_t, r_t, and s_{t+1} are the state, reward, and subsequent state along a sampled trajectory and U is our value function estimate. Then,

$$\hat{A}^{(k)}(s,a) = \mathbb{E}\left[\sum_{\ell=1}^{k} \gamma^{\ell-1} \delta_\ell\right] \quad (13.12)$$

Instead of committing to a particular value for k, generalized advantage estimation introduces a parameter $\lambda \in [0,1]$ that provides an *exponentially weighted average* of $\hat{A}^{(k)}$ for k ranging from 1 to d:[2]

[2] The exponentially weighted average of a series $x_1, x_2, \ldots$ is $(1-\lambda)(x_1 + \lambda x_2 + \lambda^2 x_3 + \cdots)$.

$$\hat{A}^{\text{GAE}}(s,a)\mid_{d=1} = \hat{A}^{(1)} \quad (13.13)$$

$$\hat{A}^{\text{GAE}}(s,a)\mid_{d=2} = (1-\lambda)\hat{A}^{(1)} + \lambda\hat{A}^{(2)} \quad (13.14)$$

$$\hat{A}^{\text{GAE}}(s,a)\mid_{d=3} = (1-\lambda)\hat{A}^{(1)} + \lambda\Big((1-\lambda)\hat{A}^{(2)} + \lambda\hat{A}^{(3)}\Big) \quad (13.15)$$

$$= (1-\lambda)\hat{A}^{(1)} + \lambda(1-\lambda)\hat{A}^{(2)} + \lambda^2\hat{A}^{(3)} \quad (13.16)$$

$$\vdots$$

$$\hat{A}^{\text{GAE}}(s,a) = (1-\lambda)\Big(\hat{A}^{(1)} + \lambda\hat{A}^{(2)} + \lambda^2\hat{A}^{(3)} + \cdots + \lambda^{d-2}\hat{A}^{(d-1)}\Big) + \lambda^{d-1}\hat{A}^{(d)} \quad (13.17)$$

For an infinite horizon, the generalized advantage estimate simplifies to

$$\begin{aligned}
\hat{A}^{\text{GAE}}(s,a) &= (1-\lambda)\left(\hat{A}^{(1)} + \lambda\hat{A}^{(2)} + \lambda^2\hat{A}^{(3)} + \cdots\right) &\quad (13.18)\\
&= (1-\lambda)\left(\delta_1\left(1+\lambda+\lambda^2+\cdots\right) + \gamma\delta_2\left(\lambda+\lambda^2+\cdots\right) + \gamma^2\delta_3\left(\lambda^2+\cdots\right) + \cdots\right) &\quad (13.19)\\
&= (1-\lambda)\left(\delta_1\frac{1}{1-\lambda} + \gamma\delta_2\frac{\lambda}{1-\lambda} + \gamma^2\delta_3\frac{\lambda^2}{1-\lambda} + \cdots\right) &\quad (13.20)\\
&= \mathbb{E}\left[\sum_{k=1}^{\infty}(\gamma\lambda)^{k-1}\delta_k\right] &\quad (13.21)
\end{aligned}$$

We can tune parameter λ to balance between bias and variance. If $\lambda = 0$, then we have the high-bias, low-variance estimate for the temporal difference residual from the previous section. If $\lambda = 1$, we have the unbiased full rollout with increased variance. Figure 13.1 demonstrates the algorithm with different values for λ.

```julia
struct GeneralizedAdvantageEstimation
    𝒫     # problem
    b     # initial state distribution
    d     # depth
    m     # number of samples
    ∇logπ # gradient of log likelihood ∇logπ(θ,a,s)
    U     # parameterized value function U(ϕ, s)
    ∇U    # gradient of value function ∇U(ϕ,s)
    λ     # weight ∈ [0,1]
end

function gradient(M::GeneralizedAdvantageEstimation, π, θ, ϕ)
    𝒫, b, d, m, ∇logπ = M.𝒫, M.b, M.d, M.m, M.∇logπ
    U, ∇U, γ, λ = M.U, M.∇U, M.𝒫.γ, M.λ
    πθ(s) = π(θ, s)
    R(τ,j) = sum(r*γ^(k-1) for (k,(s,a,r)) in enumerate(τ[j:end]))
    δ(τ,j) = τ[j][3] + γ*U(ϕ,τ[j+1][1]) - U(ϕ,τ[j][1])
    A(τ,j) = sum((γ*λ)^(ℓ-1)*δ(τ, j+ℓ-1) for ℓ in 1:d-j)
    ∇Uθ(τ) = sum(∇logπ(θ,a,s)*A(τ,j)*γ^(j-1)
                for (j, (s,a,r)) in enumerate(τ[1:end-1]))
    ∇ℓϕ(τ) = sum((U(ϕ,s) - R(τ,j))*∇U(ϕ,s)
                for (j, (s,a,r)) in enumerate(τ))
    trajs = [simulate(𝒫, rand(b), πθ, d) for i in 1:m]
    return mean(∇Uθ(τ) for τ in trajs), mean(∇ℓϕ(τ) for τ in trajs)
end
```

Algorithm 13.2. Generalized advantage estimation for computing both a policy gradient and a value function gradient for an MDP `𝒫` with initial state distribution `b`. The policy is parameterized by `θ` and has a log-gradient `∇logπ`. The value function `U` is parameterized by `ϕ` and has gradient `∇U`. This method runs `m` rollouts to depth `d`. The generalized advantage is computed with exponential weighting `λ` using equation (13.21) with a finite horizon. The implementation here is a simplified version of what was presented in the original paper, which included aspects of trust regions when taking steps.

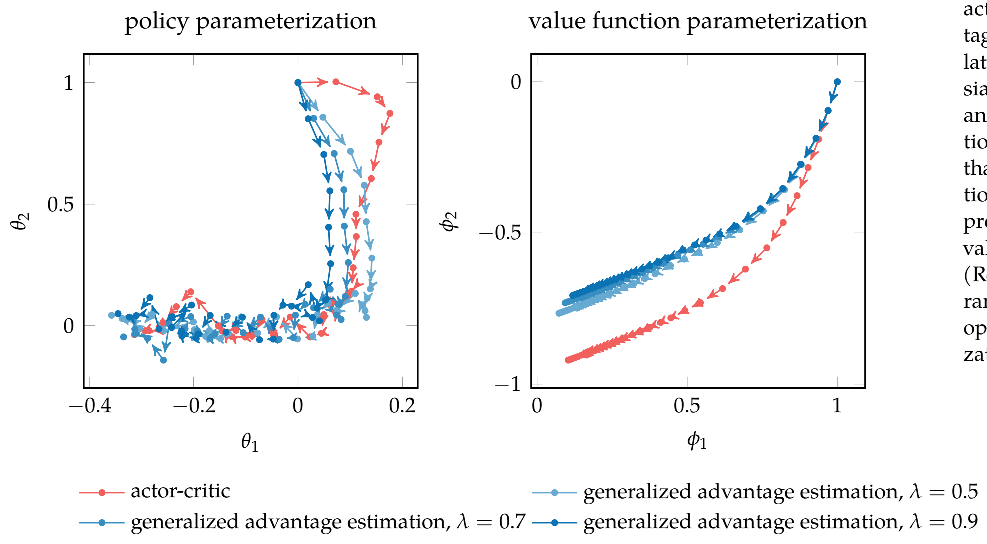


Figure 13.1. A comparison of basic actor-critic to generalized advantage estimation on the simple regulator problem with $\gamma = 0.9$, a Gaussian policy $\pi_\theta(s) = \mathcal{N}(\theta_1 s, \theta_2^2)$, and an approximate value function $U_\phi(s) = \phi_1 s + \phi_2 s^2$. We find that generalized advantage estimation is more efficiently able to approach well-performing policy and value function parameterizations. (Recall that the optimal policy parameterization is $[-1, 0]$ and the optimal value function parameterization is near $[0, -0.7]$.)

13.3 *Deterministic Policy Gradient*

The *deterministic policy gradient* approach[3] involves optimizing a deterministic policy $\pi_\theta(s)$ that produces continuous actions with the help of a critic in the form of a parameterized action value function $Q_\phi(s, a)$. As with the actor-critic methods discussed so far, we define a loss function with respect to the parameterization ϕ:

$$\ell(\phi) = \frac{1}{2} \mathop{\mathbb{E}}_{s,a,r,s'} \left[\left(r + \gamma Q_\phi(s', \pi_\theta(s')) - Q_\phi(s, a) \right)^2 \right] \tag{13.22}$$

where the expectation is over the experience tuples generated by rollouts of π_θ. This loss function attempts to minimize the residual of Q_ϕ, similar to how the actor-critic method in the first section tried to minimize the residual of U_ϕ.

Similar to the other methods, we update ϕ by taking a step in the opposite direction of the gradient:

$$\nabla \ell(\phi) = \mathop{\mathbb{E}}_{s,a,r,s'} \left[\left(r + \gamma Q_\phi(s', \pi_\theta(s')) - Q_\phi(s, a) \right) \left(\gamma \nabla_\phi Q_\phi(s', \pi_\theta(s')) - \nabla_\phi Q_\phi(s, a) \right) \right] \tag{13.23}$$

We thus need a differentiable parameterized action value function from which we can compute $\nabla_\phi Q_\phi(s, a)$, such as a neural network.

[3] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms," in *International Conference on Machine Learning (ICML)*, 2014.

For the actor, we want to find a value of $\boldsymbol{\theta}$ that maximizes

$$U(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{s \sim b_{\gamma,\boldsymbol{\theta}}} \left[Q_{\boldsymbol{\phi}}(s, \pi_{\boldsymbol{\theta}}(s)) \right] \tag{13.24}$$

where the expectation is over the states from the discounted visitation frequency when following $\pi_{\boldsymbol{\theta}}$. Again, we can use gradient ascent to optimize $\boldsymbol{\theta}$ with the gradient given by

$$\begin{aligned} \nabla U(\boldsymbol{\theta}) &= \mathbb{E}_s \left[\nabla_{\boldsymbol{\theta}} Q_{\boldsymbol{\phi}}(s, \pi_{\boldsymbol{\theta}}(s)) \right] & (13.25) \\ &= \mathbb{E}_s \left[\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(s) \nabla_a Q_{\boldsymbol{\phi}}(s, a)|_{a=\pi_{\boldsymbol{\theta}}(s)} \right] & (13.26) \end{aligned}$$

Here, $\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(s)$ is a Jacobian matrix whose ith column is the gradient with respect to the ith action dimension of the policy under parameterization $\boldsymbol{\theta}$. An example for this term is given in example 13.1. The gradient $\nabla_a Q_{\boldsymbol{\phi}}(s, a)|_{a=\pi_{\boldsymbol{\theta}}(s)}$ is a vector that indicates how much our estimated action value changes as we perturb the action given by our policy at state s. In addition to the Jacobian, we need to supply this gradient to use this method.

Example 13.1. An example of the Jacobian in the deterministic policy gradient.

Consider the following deterministic policy for a two-dimensional action space and a one-dimensional state space:

$$\pi_{\boldsymbol{\theta}}(s) = \begin{bmatrix} \theta_1 + \theta_2 s + \theta_3 s^2 \\ \theta_1 + \sin(\theta_4 s) + \cos(\theta_5 s) \end{bmatrix}$$

The matrix $\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(s)$ then takes the following form:

$$\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(s) = \begin{bmatrix} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(s)\,|_{a_1} & \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(s)\,|_{a_2} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ s & 0 \\ s^2 & 0 \\ 0 & \cos(\theta_4 s) s \\ 0 & -\sin(\theta_5 s) s \end{bmatrix}$$

As with the other actor-critic methods, we perform gradient descent on $\ell(\boldsymbol{\phi})$ and gradient ascent on $U(\boldsymbol{\theta})$. For this approach to work in practice, a few additional techniques are needed. One is to generate experiences from a stochastic policy to allow better exploration. It is often adequate to simply add zero-mean

Gaussian noise to actions generated by our deterministic policy $\pi_{\boldsymbol\theta}$, as done in algorithm 13.3. To encourage stability when learning $\boldsymbol\theta$ and $\boldsymbol\phi$, we can use experience replay.[4]

An example of this method and the effect of σ on performance is given in example 13.2.

[4] We will discuss experience replay in section 17.7 in the context of reinforcement learning. Other techniques for stabilizing learning include using *target parameterizations*, described in the context of neural representations by T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous Control with Deep Reinforcement Learning," in *International Conference on Learning Representations (ICLR)*, 2016. arXiv: 1509.02971v6.

```
struct DeterministicPolicyGradient
    𝒫     # problem
    b     # initial state distribution
    d     # depth
    m     # number of samples
    ∇π    # gradient of deterministic policy π(θ, s)
    Q     # parameterized value function Q(ϕ,s,a)
    ∇Qϕ   # gradient of value function with respect to ϕ
    ∇Qa   # gradient of value function with respect to a
    σ     # policy noise
end

function gradient(M::DeterministicPolicyGradient, π, θ, ϕ)
    𝒫, b, d, m, ∇π = M.𝒫, M.b, M.d, M.m, M.∇π
    Q, ∇Qϕ, ∇Qa, σ, γ = M.Q, M.∇Qϕ, M.∇Qa, M.σ, M.𝒫.γ
    π_rand(s) = π(θ, s) + σ*randn()*I
    ∇Uθ(τ) = sum(∇π(θ,s)*∇Qa(ϕ,s,π(θ,s))*γ^(j-1) for (j,(s,a,r))
                in enumerate(τ))
    ∇ℓϕ(τ,j) = begin
        s, a, r = τ[j]
        s′ = τ[j+1][1]
        a′ = π(θ,s′)
        δ = r + γ*Q(ϕ,s′,a′) - Q(ϕ,s,a)
        return δ*(γ*∇Qϕ(ϕ,s′,a′) - ∇Qϕ(ϕ,s,a))
    end
    ∇ℓϕ(τ) = sum(∇ℓϕ(τ,j) for j in 1:length(τ)-1)
    trajs = [simulate(𝒫, rand(b), π_rand, d) for i in 1:m]
    return mean(∇Uθ(τ) for τ in trajs), mean(∇ℓϕ(τ) for τ in trajs)
end
```

Algorithm 13.3. The deterministic policy gradient method for computing a policy gradient `∇θ` for a deterministic policy `π` and a value function gradient `∇ϕ` for a continuous action MDP `𝒫` with initial state distribution `b`. The policy is parameterized by `θ` and has a gradient `∇π` that produces a matrix where each column is the gradient with respect to that continuous action component. The value function `Q` is parameterized by `ϕ` and has a gradient `∇Qϕ` with respect to the parameterization and gradient `∇Qa` with respect to the action. This method runs `m` rollouts to depth `d`, and performs exploration using 0-mean Gaussian noise with standard deviation `σ`.

13.4 *Actor-Critic with Monte Carlo Tree Search*

We can extend concepts from online planning (chapter 9) to the actor-critic setting in which we improve a parameterized policy $\pi_{\boldsymbol\theta}(a \mid s)$ and a parameterized value function $U_{\boldsymbol\phi}(s)$.[5] This section discusses the application of Monte Carlo tree search (section 9.6) to learning a stochastic policy with a discrete action space. We use our parameterized policy and value function to guide Monte Carlo tree search,

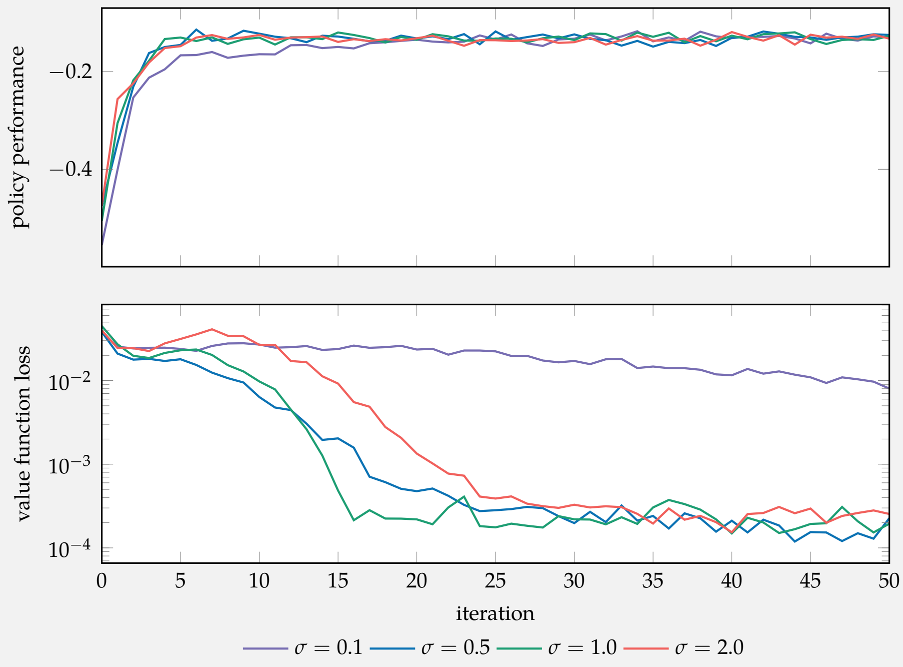
[5] Deterministic policy gradient used $Q_{\boldsymbol\phi}$, but this approach uses $U_{\boldsymbol\phi}$ like the other actor-critic methods discussed in this chapter.

Example 13.2. An application of the deterministic policy gradient method to the simple regulator problem and an exploration of the impact of the policy stochasticity parameter σ.

Consider applying the deterministic policy gradient algorithm to the simple regulator problem. Suppose we use a simple parameterized deterministic policy $\pi_{\boldsymbol{\theta}}(s) = \theta_1$ and the parameterized state-action value function:

$$Q_{\boldsymbol{\phi}}(s, a) = \phi_1 + \phi_2 s + \phi_3 s^2 + \phi_4 (s + a)^2$$

Here, we plot a progression of the deterministic policy gradient algorithm starting with $\boldsymbol{\theta} = [0]$ and $\boldsymbol{\phi} = [0, 1, 0, -1]$ for different values of σ. Each iteration was run with five rollouts to depth 10 with $\gamma = 0.9$.



For this simple problem, the policy quickly converges to optimality almost regardless of σ. However, if σ is either too small or too large, the value function takes longer to improve. In the case of very small values of σ, our policy conducts insufficient exploration from which to effectively learn the value function. For larger values of σ, we explore more, but we also tend to make poor move choices more frequently.

and we use the results from Monte Carlo tree search to refine our parameterized policy and value function. As with the other actor critic methods, we apply gradient-based optimization of θ and ϕ.[6]

As we perform Monte Carlo tree search, we want to direct our exploration to some extent by our parameterized policy $\pi_\theta(a \mid s)$. One approach is to use an action that maximizes the *probabilistic upper confidence bound*:

$$a = \arg\max_a Q(s,a) + c\pi_\theta(a \mid s)\frac{\sqrt{N(s)}}{1+N(s,a)} \tag{13.27}$$

where $Q(s,a)$ is the action value estimated through the tree search, $N(s,a)$ is the visit count as discussed in section 9.6, and $N(s) = \sum_a N(s,a)$.[7]

After running tree search, we can use the statistics that we collect to obtain $\pi_{\text{MCTS}}(a \mid s)$. One way to define this is in terms of the counts:[8]

$$\pi_{\text{MCTS}}(a \mid s) \propto N(s,a)^\eta \tag{13.28}$$

where $\eta \geq 0$ is a hyperparameter that controls the greediness of the policy. If $\eta = 0$, then π_{MCTS} will generate actions at random. As $\eta \to \infty$, it will select the action that was selected the most from that state.

In our optimization of θ, we want our model π_θ to match what we obtain through Monte Carlo tree search. One loss function that we can define is the expected cross entropy of $\pi_\theta(\cdot \mid s)$ relative to $\pi_{\text{MCTS}}(\cdot \mid s)$:

$$\ell(\theta) = -\mathbb{E}_s\left[\sum_a \pi_{\text{MCTS}}(a \mid s)\log \pi_\theta(a \mid s)\right] \tag{13.29}$$

where the expectation is over states experienced during the tree exploration. The gradient is

$$\nabla \ell(\theta) = -\mathbb{E}_s\left[\sum_a \frac{\pi_{\text{MCTS}}(a \mid s)}{\pi_\theta(a \mid s)}\nabla_\theta \pi_\theta(a \mid s)\right] \tag{13.30}$$

To learn ϕ, we define a loss function in terms of a value function generated during the tree search:

$$U_{\text{MCTS}}(s) = \max_a Q(s,a) \tag{13.31}$$

which is defined at least at the states that we explore during tree search. The loss function aims to make U_ϕ agree with the estimates from the tree search:

$$\ell(\phi) = \frac{1}{2}\mathbb{E}_s\left[\left(U_\phi(s) - U_{\text{MCTS}}(s)\right)^2\right] \tag{13.32}$$

[6] This general approach was introduced by D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al., "Mastering the Game of Go Without Human Knowledge," *Nature*, vol. 550, pp. 354–359, 2017. The discussion here loosely follows their *AlphaGo Zero* algorithm, but instead of trying to solve the game of Go, we are trying to solve a general MDP. Both the fact that Alpha Zero plays as both Go players and that games tend to have a winner and a loser allow the original method to reinforce the winning behavior and punish the losing behavior. The generalized MDP formulation will tend to suffer from sparse rewards when applied to similar problems.

[7] There are some notable differences from the upper confidence bound presented in equation (9.1); for example, there is no logarithm in equation (13.27) and we add 1 to the denominator to follow the form used by AlphaGo Zero.

[8] In algorithm 9.5, we select the greedy action with respect to Q. Other strategies are surveyed by C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012. The approach suggested here follows AlphaGo Zero.

The gradient is

$$\nabla \ell(\boldsymbol{\phi}) = \mathbb{E}_s\left[\left(U_{\boldsymbol{\phi}}(s) - U_{\text{MCTS}}(s)\right)\nabla_{\boldsymbol{\phi}} U_{\boldsymbol{\phi}}(s)\right] \tag{13.33}$$

Like the actor-critic method in the first section, we need to be able to compute the gradient of our parameterized value function.

After performing some number of Monte Carlo tree search simulations, we update $\boldsymbol{\theta}$ by stepping in the direction opposite to $\nabla \ell(\boldsymbol{\theta})$ and $\boldsymbol{\phi}$ by stepping in the direction opposite to $\nabla \ell(\boldsymbol{\phi})$.[9]

[9] The AlphaGo Zero implementation uses a single neural network to represent both the value function and the policy instead of independent parameterizations as discussed in this section. The gradient used to update the network parameters is a mixture of equations (13.30) and (13.33). This enhancement significantly reduces evaluation time and feature learning time.

13.5 Summary

- In actor-critic methods, an actor attempts to optimize a parameterized policy with the help of a critic that provides a parameterized estimate of the value function.
- Generally, actor-critic methods use gradient-based optimization to learn the parameters of both the policy and value function approximation.
- The basic actor-critic method uses a policy gradient for the actor and minimizes the squared temporal difference residual for the critic.
- The generalized advantage estimate attempts to reduce the variance of its policy gradient at the expense of some bias by accumulating temporal difference residuals across multiple time steps.
- The deterministic policy gradient can be applied to problems with continuous action spaces and uses a deterministic policy actor and an action value critic.
- Online methods, such as Monte Carlo tree search, can be used to direct the optimization of the policy and value function estimate.

13.6 Exercises

Exercise 13.1. Would the actor-critic method with Monte Carlo tree search, as presented in section 13.4, be a good method for solving the cart-pole problem (appendix F.3)?

Solution: The Monte Carlo tree search expands a tree based on visited states. The cart-pole problem has a continuous state space, leading to a search tree with an infinite branching factor. Use of this algorithm would require adjusting the problem, such as discretizing the state space.

Exercise 13.2. In the following expressions of advantage functions, determine which ones are correct and explain what they are referring to:

$$
\begin{array}{rl}
\text{(a)} & \mathbb{E}_{r,s'}\left[r + \gamma U^{\pi_\theta}(s) - U^{\pi_\theta}(s')\right] \\
\text{(b)} & \mathbb{E}_{r,s'}\left[r + \gamma U^{\pi_\theta}(s') - U^{\pi_\theta}(s)\right] \\
\text{(c)} & \mathbb{E}_{r_{1:d},s'}\left[-U^{\pi_\theta}(s) + \gamma^k U^{\pi_\theta}(s') + \sum_{\ell=1}^{k} \gamma^{l-1} r_l\right] \\
\text{(d)} & \mathbb{E}_{r_{1:d},s'}\left[-U^{\pi_\theta}(s) + \gamma U^{\pi_\theta}(s') + \sum_{\ell=1}^{k} \gamma^{l-1} r_l\right] \\
\text{(e)} & \mathbb{E}\left[-U^{\pi_\theta}(s) + \sum_{\ell=1}^{d} \gamma^{l-1} r_l\right] \\
\text{(f)} & \mathbb{E}\left[-\gamma U^{\pi_\theta}(s') + \sum_{\ell=1}^{d+1} \gamma^{l-1} r_l\right] \\
\text{(g)} & \mathbb{E}\left[\sum_{\ell=1}^{k} \gamma^{l-1} \delta_{l-1}\right] \\
\text{(h)} & \mathbb{E}\left[\sum_{\ell=1}^{k} \gamma^{l-1} \delta_{l}\right] \\
\text{(i)} & \mathbb{E}\left[\sum_{k=1}^{\infty} (\gamma\lambda)^{k-1} \delta_{k}\right] \\
\text{(j)} & \mathbb{E}\left[\sum_{k=1}^{\infty} (\lambda)^{k-1} \delta_{k}\right]
\end{array}
$$

Solution: The following table lists the correct expressions:

(b)	Advantage with temporal difference residual
(c)	Advantage estimate after k-step rollouts
(e)	Advantage with the sequence of rollout rewards
(h)	Advantage estimate with temporal difference residuals
(i)	Generalized advantage estimate

Exercise 13.3. What are the benefits of using a temporal difference residual over a sequence of rollout rewards and vice versa?

Solution: Approximation using a temporal difference residual is more computationally efficient than using a sequence of rollouts. Temporal difference residual approximation has low variance but high bias due to using the critic value function U_ϕ as an approximator of the true value function U^{π_θ}. On the other hand, rollout approximation has high variance but is unbiased. Obtaining an accurate estimate using a temporal difference residual approximation typically requires far fewer samples than when using a rollout approximation, at the cost of introducing bias into our estimate.

Exercise 13.4. Consider the action value function given in example 13.2, $Q_{\boldsymbol{\phi}}(s, a) = \phi_1 + \phi_2 s + \phi_3 s^2 + \phi_4 (s + a)^2$. Calculate the gradients required for the deterministic policy gradient approach.

Solution: We need to calculate two gradients. For the actor, we need to compute $\nabla_{\boldsymbol{\phi}} Q_{\boldsymbol{\phi}}(s, a)$, while for the critic, we need to compute $\nabla_a Q_{\boldsymbol{\phi}}(s, a)$.

$$\begin{aligned} \nabla_{\boldsymbol{\phi}} Q(s, a) &= \left[1, s, s^2, (s + a)^2\right] \\ \nabla_a Q(s, a) &= 2\phi_4 (s + a) \end{aligned}$$