12 Policy Gradient Optimization

We can use estimates of the policy gradient to drive the search of the parameter space toward an optimal policy. The previous chapter outlined methods for estimating this gradient. This chapter explains how to use these estimates to guide the optimization. We begin with gradient ascent, which simply takes steps in the direction of the gradient at each iteration. Determining the step size is a major challenge. Large steps can lead to faster progress to the optimum, but they can overshoot. The natural policy gradient modifies the direction of the gradient to better handle variable levels of sensitivity across parameter components. We conclude with the trust region method, which starts in exactly the same way as the natural gradient method to obtain a candidate policy. It then searches along the line segment in policy space connecting the original policy to this candidate to find a better policy.

12.1 Gradient Ascent Update

We can use *gradient ascent* (reviewed in appendix A.11) to find a policy parameterized by θ that maximizes the expected utility $U(\theta)$. Gradient ascent is a type of *iterated ascent* method, which involves taking steps in the parameter space at each iteration in an attempt to improve the quality of the associated policy. All the methods discussed in this chapter are iterated ascent methods, but they differ in how they take steps. The gradient ascent method discussed in this section takes steps in the direction of $\nabla U(\theta)$, which may be estimated using one of the methods discussed in the previous chapter. The update of θ is

$$\theta \leftarrow \theta + \alpha \nabla U(\theta) \tag{12.1}$$

where the step length is equal to a step factor $\alpha > 0$ times the magnitude of the gradient.

Algorithm 12.1 implements a method that takes such a step. This method can be called for either a fixed number of iterations or until θ or $U(\theta)$ converges. Gradient ascent, as well as the other algorithms discussed in this chapter, is not guaranteed to converge to the optimal policy. However, there are techniques to encourage convergence to a *locally optimal* policy, in which taking an infinitesimally small step in parameter space cannot result in a better policy. One approach is to decay the step factor with each step.[1]

[1] This approach, as well as many others, are covered in detail by M. J. Kochenderfer and T. A. Wheeler, *Algorithms for Optimization*. MIT Press, 2019.

```
struct PolicyGradientUpdate
    ∇U # policy gradient estimate
    α  # step factor
end

function update(M::PolicyGradientUpdate, θ)
    return θ + M.α * M.∇U(θ)
end
```

Algorithm 12.1. The gradient ascent method for policy optimization. It takes a step from a point `θ` in the direction of the gradient `∇U` with step factor `α`. We can use one of the methods in the previous chapter to compute `∇U`.

Very large gradients tend to overshoot the optimum and may occur due to a variety of reasons. Rewards for some problems, such as for the 2048 problem (appendix F.2), can vary by orders of magnitude. One approach for keeping the gradients manageable is to use *gradient scaling*, which limits the magnitude of a gradient estimate before using it to update the policy parameterization. Gradients are commonly limited to having an L_2-norm of 1. Another approach is *gradient clipping*, which conducts elementwise clamping of the gradient before using it to update the policy. Clipping commonly limits the entries to lie between ± 1. Both techniques are implemented in algorithm 12.2.

```
scale_gradient(∇, L2_max) = min(L2_max/norm(∇), 1)*∇
clip_gradient(∇, a, b) = clamp.(∇, a, b)
```

Algorithm 12.2. Methods for gradient scaling and clipping. Gradient scaling limits the magnitude of the provided gradient vector `∇` to `L2_max`. Gradient clipping provides elementwise clamping of the provided gradient vector `∇` to between `a` and `b`.

Scaling and clipping differ in how they affect the final gradient direction, as demonstrated in figure 12.1. Scaling will leave the direction unaffected, whereas clipping affects each component individually. Whether this difference is advantageous depends on the problem. For example, if a single component dominates the gradient vector, scaling will zero out the other components.

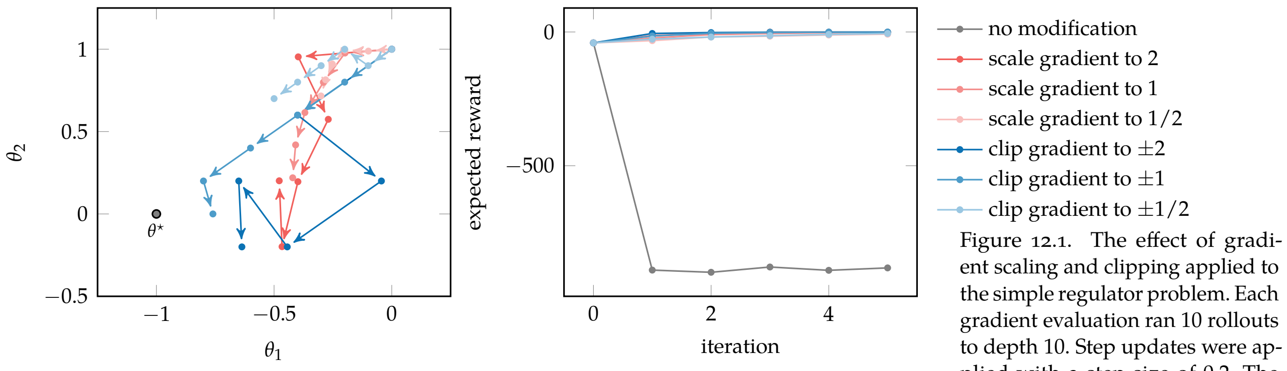

Figure 12.1. The effect of gradient scaling and clipping applied to the simple regulator problem. Each gradient evaluation ran 10 rollouts to depth 10. Step updates were applied with a step size of 0.2. The optimal policy parameterization is shown in black.

12.2 *Restricted Gradient Update*

The remaining algorithms in this chapter attempt to optimize an approximation of the objective function $U(\theta)$, subject to a constraint that the policy parameters at the next step θ' are not too far from θ at the current step. The constraint takes the form $g(\theta, \theta') \leq \epsilon$, where $\epsilon > 0$ is a free parameter in the algorithm. The methods differ in their approximation of $U(\theta)$ and the form of g. This section describes a simple *restricted step* method.

We use the first-order Taylor approximation (appendix A.12) obtained from our gradient estimate at θ to approximate U:

$$U(\theta') \approx U(\theta) + \nabla U(\theta)^\top (\theta' - \theta) \tag{12.2}$$

For the constraint, we use

$$g(\theta, \theta') = \frac{1}{2}(\theta' - \theta)^\top \mathbf{I}(\theta' - \theta) = \frac{1}{2}\|\theta' - \theta\|_2^2 \tag{12.3}$$

We can view this constraint as limiting the step length to no more than $\sqrt{2\epsilon}$. In other words, the feasible region in our optimization is a ball of radius $\sqrt{2\epsilon}$ centered at θ.

The optimization problem is, then,

$$\begin{aligned} \underset{\theta'}{\text{maximize}} \quad & U(\theta) + \nabla U(\theta)^\top (\theta' - \theta) \\ \text{subject to} \quad & \frac{1}{2}(\theta' - \theta)^\top \mathbf{I}(\theta' - \theta) \leq \epsilon \end{aligned} \tag{12.4}$$

We can drop $U(\theta)$ from the objective since it does not depend on θ'. In addition, we can change the inequality to an equality in the constraint because the linear objective forces the optimal solution to be on the boundary of the feasible region. These changes result in an equivalent optimization problem:

$$\begin{aligned} \underset{\theta'}{\text{maximize}} \quad & \nabla U(\theta)^\top(\theta' - \theta) \\ \text{subject to} \quad & \frac{1}{2}(\theta' - \theta)^\top \mathbf{I}(\theta' - \theta) = \epsilon \end{aligned} \tag{12.5}$$

This optimization problem can be solved analytically:

$$\theta' = \theta + \mathbf{u}\sqrt{\frac{2\epsilon}{\mathbf{u}^\top\mathbf{u}}} = \theta + \sqrt{2\epsilon}\frac{\mathbf{u}}{\|\mathbf{u}\|} \tag{12.6}$$

where the unnormalized search direction $\mathbf{u}$ is simply $\nabla U(\theta)$. Of course, we do not know $\nabla U(\theta)$ exactly, but we can use any of the methods described in the previous chapter to estimate it. Algorithm 12.3 provides an implementation.

```
struct RestrictedPolicyUpdate
    𝒫     # problem
    b     # initial state distribution
    d     # depth
    m     # number of samples
    ∇logπ # gradient of log likelihood
    π     # policy
    ϵ     # divergence bound
end

function update(M::RestrictedPolicyUpdate, θ)
    𝒫, b, d, m, ∇logπ, π, γ = M.𝒫, M.b, M.d, M.m, M.∇logπ, M.π, M.𝒫.γ
    πθ(s) = π(θ, s)
    R(τ) = sum(r*γ^(k-1) for (k, (s,a,r)) in enumerate(τ))
    τs = [simulate(𝒫, rand(b), πθ, d) for i in 1:m]
    ∇log(τ) = sum(∇logπ(θ, a, s) for (s,a) in τ)
    ∇U(τ) = ∇log(τ)*R(τ)
    u = mean(∇U(τ) for τ in τs)
    return θ + u*sqrt(2*M.ϵ/dot(u,u))
end
```

Algorithm 12.3. The update function for the restricted policy gradient method at `θ` for a problem `𝒫` with initial state distribution `b`. The gradient is estimated from an initial state distribution `b` to depth `d` with `m` simulations of parameterized policy `π(θ, s)` with log policy gradient `∇logπ`.

12.3 Natural Gradient Update

The *natural gradient* method[2] is a variation of the restricted step method discussed in the previous section to better handle situations when some components of the parameter space are more sensitive than others. *Sensitivity* in this context refers to how much the utility of a policy varies with respect to small changes in one of the parameters. The sensitivity in gradient methods is largely determined by the choice of scaling of the policy parameters. The natural policy gradient method makes the search direction $\mathbf{u}$ invariant to parameter scaling. Figure 12.2 illustrates the differences between the true gradient and the natural gradient.

[2] S. Amari, "Natural Gradient Works Efficiently in Learning," *Neural Computation*, vol. 10, no. 2, pp. 251–276, 1998.

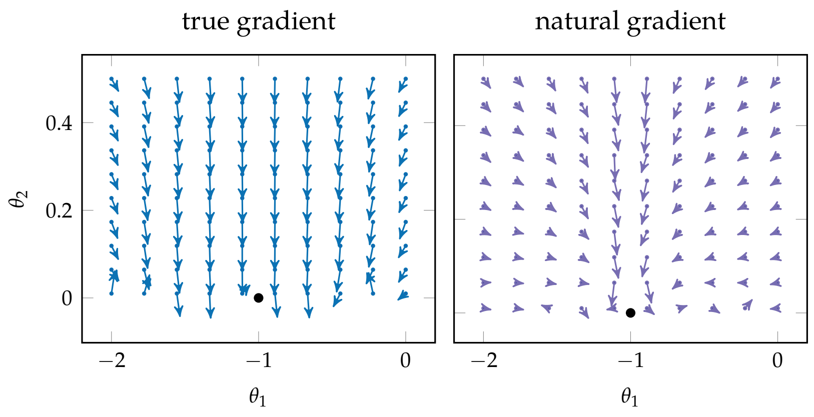

Figure 12.2. A comparison of the true gradient and the natural gradient on the simple regulator problem (see appendix F.5). The true gradient generally points strongly in the negative θ_2 direction, whereas the natural gradient generally points toward the optimum (black dot) at $[-1, 0]$. A similar figure is presented in J. Peters and S. Schaal, "Reinforcement Learning of Motor Skills with Policy Gradients," *Neural Networks*, vol. 21, no. 4, pp. 682–697, 2008.

The natural policy gradient method uses the same first-order approximation of the objective as in the previous section. The constraint, however, is different. The intuition is that we want to restrict changes in $\boldsymbol{\theta}$ that result in large changes in the distribution over trajectories. A way to measure how much a distribution changes is to use the *Kullback-Leibler divergence*, or KL divergence (appendix A.10). We could impose the constraint

$$g(\boldsymbol{\theta}, \boldsymbol{\theta}') = D_{\mathrm{KL}}\big(p(\cdot \mid \boldsymbol{\theta}) \,||\, p(\cdot \mid \boldsymbol{\theta}')\big) \le \epsilon \tag{12.7}$$

but instead we will use a second-order Taylor approximation:

$$g(\boldsymbol{\theta}, \boldsymbol{\theta}') = \frac{1}{2}(\boldsymbol{\theta}' - \boldsymbol{\theta})^\top \mathbf{F}_{\boldsymbol{\theta}} (\boldsymbol{\theta}' - \boldsymbol{\theta}) \le \epsilon \tag{12.8}$$

where the *Fisher information matrix* has the following form:

$$
\begin{aligned}
\mathbf{F}_{\theta} &= \int p(\tau \mid \theta) \nabla \log p(\tau \mid \theta) \nabla \log p(\tau \mid \theta)^{\top} \, \mathrm{d}\tau && (12.9) \\
&= \mathbb{E}_{\tau}\left[\nabla \log p(\tau \mid \theta) \nabla \log p(\tau \mid \theta)^{\top}\right] && (12.10)
\end{aligned}
$$

The resulting optimization problem is

$$
\begin{aligned}
\underset{\theta'}{\text{maximize}} \quad & \nabla U(\theta)^{\top}(\theta' - \theta) \\
\text{subject to} \quad & \frac{1}{2}(\theta' - \theta)^{\top} \mathbf{F}_{\theta} (\theta' - \theta) = \epsilon
\end{aligned} \tag{12.11}
$$

which looks identical to equation (12.5) except that instead of the identity matrix $\mathbf{I}$, we have the Fisher matrix $\mathbf{F}_{\theta}$. This difference results in an ellipsoid feasible set. Figure 12.3 shows an example in two dimensions.

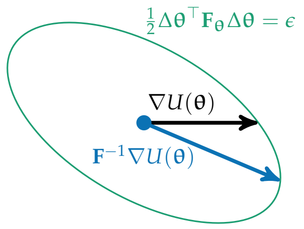
$\frac{1}{2}\Delta\theta^{\top}\mathbf{F}_{\theta}\Delta\theta = \epsilon$
$\nabla U(\theta)$
$\mathbf{F}^{-1}\nabla U(\theta)$

Figure 12.3. The natural policy gradient places a constraint on the approximated Kullback-Leibler divergence. This constraint takes the form of an ellipse. The ellipse may be elongated in certain directions, allowing larger steps if the gradient is rotated.

This optimization problem can be solved analytically and has the same form as the update in the previous section:

$$
\theta' = \theta + \mathbf{u}\sqrt{\frac{2\epsilon}{\nabla U(\theta)^{\top}\mathbf{u}}} \tag{12.12}
$$

except that we now have[3]

$$
\mathbf{u} = \mathbf{F}_{\theta}^{-1} \nabla U(\theta) \tag{12.13}
$$

We can use sampled trajectories to estimate $\mathbf{F}_{\theta}$ and $\nabla U(\theta)$. Algorithm 12.4 provides an implementation.

[3] This computation can be done using conjugate gradient descent, which reduces computation when the dimension of θ is large. S. M. Kakade, "A Natural Policy Gradient," in *Advances in Neural Information Processing Systems* (*NIPS*), 2001.

12.4 *Trust Region Update*

This section discusses a method for searching within the *trust region*, defined by the elliptical feasible region from the previous section. This category of approach is referred to as *trust region policy optimization* (*TRPO*).[4] It works by computing the next evaluation point θ' that would be taken by the natural policy gradient and then conducting a *line search* along the line segment connecting θ to θ'. A key property of this line search phase is that evaluations of the approximate objective and constraint do not require any additional rollout simulations.

[4] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust Region Policy Optimization," in *International Conference on Machine Learning* (*ICML*), 2015.

```
struct NaturalPolicyUpdate
    𝒫     # problem
    b     # initial state distribution
    d     # depth
    m     # number of samples
    ∇logπ # gradient of log likelihood
    π     # policy
    ϵ     # divergence bound
end

function natural_update(θ, ∇f, F, ϵ, τs)
    ∇fθ = mean(∇f(τ) for τ in τs)
    u = mean(F(τ) for τ in τs) \ ∇fθ
    return θ + u*sqrt(2ϵ/dot(∇fθ,u))
end

function update(M::NaturalPolicyUpdate, θ)
    𝒫, b, d, m, ∇logπ, π, γ = M.𝒫, M.b, M.d, M.m, M.∇logπ, M.π, M.𝒫.γ
    πθ(s) = π(θ, s)
    R(τ) = sum(r*γ^(k-1) for (k, (s,a,r)) in enumerate(τ))
    ∇log(τ) = sum(∇logπ(θ, a, s) for (s,a) in τ)
    ∇U(τ) = ∇log(τ)*R(τ)
    F(τ) = ∇log(τ)*∇log(τ)'
    τs = [simulate(𝒫, rand(b), πθ, d) for i in 1:m]
    return natural_update(θ, ∇U, F, M.ϵ, τs)
end
```

Algorithm 12.4. The update function for the natural policy gradient, given policy `π(θ, s)`, for an MDP `𝒫` with initial state distribution `b`. The natural gradient with respect to the parameter vector `θ` is estimated from `m` rollouts to depth `d` using the log policy gradients `∇logπ`. The `natural_update` helper method conducts an update according to equation (12.12), given an objective gradient `∇f(τ)` and a Fisher matrix `F(τ)` for a list of trajectories.

During the line search phase, we no longer use a first-order approximation. Instead, we use an approximation derived from an equality involving the advantage function[5]

$$U(\theta') = U(\theta) + \mathop{\mathbb{E}}_{\tau \sim \pi_{\theta'}}\left[\sum_{k=1}^{d} A_\theta(s^{(k)}, a^{(k)})\right] \tag{12.14}$$

[5] A variation of this equality is proven in lemma 6.1 of S. M. Kakade and J. Langford, "Approximately Optimal Approximate Reinforcement Learning," in *International Conference on Machine Learning (ICML)*, 2002.

Another way to write this is to use $b_{\gamma,\theta}$, which is the *discounted visitation distribution* of state s under policy π_θ, where

$$b_{\gamma,\theta}(s) \propto P(s^{(1)} = s) + \gamma P(s^{(2)} = s) + \gamma^2 P(s^{(3)} = s) + \cdots \tag{12.15}$$

Using the discounted visitation distribution, the objective becomes

$$U(\theta') = U(\theta) + \mathop{\mathbb{E}}_{s \sim b_{\gamma,\theta'}}\left[\mathop{\mathbb{E}}_{a \sim \pi_{\theta'}(\cdot \mid s)}[A_\theta(s,a)]\right] \tag{12.16}$$

We would like to pull our samples from our policy parameterized by θ instead of θ' so that we do not have to run more simulations during the line search. The samples associated with the inner expectation can be replaced with samples from our original policy so long as we appropriately weight the advantage:[6]

[6] This weighting comes from *importance sampling*, which is reviewed in appendix A.14.

$$U(\theta') = U(\theta) + \mathop{\mathbb{E}}_{s \sim b_{\gamma,\theta'}}\left[\mathop{\mathbb{E}}_{a \sim \pi_{\theta}(\cdot \mid s)}\left[\frac{\pi_{\theta'}(a \mid s)}{\pi_\theta(a \mid s)} A_\theta(s,a)\right]\right] \tag{12.17}$$

The next step involves replacing the state distribution with $b_{\gamma,\theta}$. The quality of the approximation degrades as θ' gets further from θ, but it is hypothesized that it is acceptable within the trust region. Since $U(\theta)$ does not depend on θ', we can drop it from the objective. We can also drop the state value function from the advantage function, leaving us with the action value function. What remains is referred to as the *surrogate objective*:

$$f(\theta, \theta') = \mathop{\mathbb{E}}_{s \sim b_{\gamma,\theta}}\left[\mathop{\mathbb{E}}_{a \sim \pi_{\theta}(\cdot \mid s)}\left[\frac{\pi_{\theta'}(a \mid s)}{\pi_\theta(a \mid s)} Q_\theta(s,a)\right]\right] \tag{12.18}$$

This equation can be estimated from the same set of trajectories that was used to estimate the natural gradient update. We can estimate $Q_\theta(s,a)$ using the reward-to-go in the sampled trajectories.[7]

[7] Algorithm 12.5 instead uses $\sum_{\ell=k} r^{(\ell)} \gamma^{\ell-1}$, which effectively discounts the reward-to-go by γ^{k-1}. This discount is needed to weight each sample's contribution to match the discounted visitation distribution. The surrogate constraint is similarly discounted.

The *surrogate constraint* in the line search is given by

$$g(\theta, \theta') = \mathop{\mathbb{E}}_{s \sim b_{\gamma,\theta}}[D_{\mathrm{KL}}(\pi_\theta(\cdot \mid s) \mid\mid \pi_{\theta'}(\cdot \mid s))] \le \epsilon \tag{12.19}$$

Line search involves iteratively evaluating our surrogate objective f and surrogate constraint g for different points in the policy space. We begin with the θ' obtained from the same process as the natural gradient update. We then iteratively apply

$$\theta' \leftarrow \theta + \alpha(\theta' - \theta) \tag{12.20}$$

until we have an improvement in our objective with $f(\theta, \theta') > f(\theta, \theta)$ and our constraint is met with $g(\theta, \theta') \leq \epsilon$. The step factor $0 < \alpha < 1$ shrinks the distance between θ and θ' at each iteration, with α typically set to 0.5.

Algorithm 12.5 provides an implementation of this approach. Figure 12.4 illustrates the relationship between the feasible regions associated with the natural gradient and the line search. Figure 12.5 demonstrates the approach on a regulator problem, and example 12.1 shows an update for a simple problem.

12.5 *Clamped Surrogate Objective*

We can avoid detrimental policy updates from overly optimistic estimates of the trust region surrogate objective by *clamping*.[8] The surrogate objective from equation (12.18), after exchanging the action value advantage, is

$$\mathop{\mathbb{E}}_{s \sim b_{\gamma,\theta}}\left[\mathop{\mathbb{E}}_{a \sim \pi_\theta(\cdot \mid s)}\left[\frac{\pi_{\theta'}(a \mid s)}{\pi_\theta(a \mid s)} A_\theta(s,a)\right]\right] \tag{12.21}$$

[8] Clamping is a key idea in what is known as *proximal policy optimization* (*PPO*) as discussed by J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," 2017. arXiv: 1707.06347v2.

The probability ratio $\pi_{\theta'}(a \mid s)/\pi_\theta(a \mid s)$ can be overly optimistic. A pessimistic lower bound on the objective can significantly improve performance:

$$\mathop{\mathbb{E}}_{s \sim b_{\gamma,\theta}}\left[\mathop{\mathbb{E}}_{a \sim \pi_\theta(\cdot \mid s)}\left[\min\left(\frac{\pi_{\theta'}(a \mid s)}{\pi_\theta(a \mid s)} A_\theta(s,a), \operatorname{clamp}\left(\frac{\pi_{\theta'}(a \mid s)}{\pi_\theta(a \mid s)}, 1-\epsilon, 1+\epsilon\right) A_\theta(s,a)\right)\right]\right] \tag{12.22}$$

where ϵ is a small positive value[9] and $\operatorname{clamp}(x, a, b)$ forces x to be between a and b. By definition, $\operatorname{clamp}(x, a, b) = \min\{\max\{x, a\}, b\}$.

[9] While this ϵ does not directly act as a threshold on divergence, as it did in previous algorithms, its role is similar. A typical value is 0.2.

Clamping the probability ratio alone does not produce a lower bound; we must also take the minimum of the clamped and original objectives. The lower bound is shown in figure 12.6, together with the original and clamped objectives. The end result of the lower bound is that the change in probability ratio is ignored when it would cause the objective to improve significantly. Using the lower bound thus prevents large, often detrimental, updates in these situations and removes the need for the trust region surrogate constraint equation (12.19). Without the

```
struct TrustRegionUpdate
    𝒫     # problem
    b     # initial state distribution
    d     # depth
    m     # number of samples
    π     # policy π(s)
    p     # policy likelihood p(θ, a, s)
    ∇logπ # log likelihood gradient
    KL    # KL divergence KL(θ, θ′, s)
    ϵ     # divergence bound
    α     # line search reduction factor (e.g., 0.5)
end

function surrogate_objective(M::TrustRegionUpdate, θ, θ′, τs)
    d, p, γ = M.d, M.p, M.𝒫.γ
    R(τ, j) = sum(r*γ^(k-1) for (k,(s,a,r)) in zip(j:d, τ[j:end]))
    w(a,s) = p(θ′,a,s) / p(θ,a,s)
    f(τ) = mean(w(a,s)*R(τ,k) for (k,(s,a,r)) in enumerate(τ))
    return mean(f(τ) for τ in τs)
end

function surrogate_constraint(M::TrustRegionUpdate, θ, θ′, τs)
    γ = M.𝒫.γ
    KL(τ) = mean(M.KL(θ, θ′, s)*γ^(k-1) for (k,(s,a,r)) in enumerate(τ))
    return mean(KL(τ) for τ in τs)
end

function linesearch(M::TrustRegionUpdate, f, g, θ, θ′)
    fθ = f(θ)
    while g(θ′) > M.ϵ || f(θ′) ≤ fθ
        θ′ = θ + M.α*(θ′ - θ)
    end
    return θ′
end

function update(M::TrustRegionUpdate, θ)
    𝒫, b, d, m, ∇logπ, π, γ = M.𝒫, M.b, M.d, M.m, M.∇logπ, M.π, M.𝒫.γ
    πθ(s) = π(θ, s)
    R(τ) = sum(r*γ^(k-1) for (k, (s,a,r)) in enumerate(τ))
    ∇log(τ) = sum(∇logπ(θ, a, s) for (s,a) in τ)
    ∇U(τ) = ∇log(τ)*R(τ)
    F(τ) = ∇log(τ)*∇log(τ)'
    τs = [simulate(𝒫, rand(b), πθ, d) for i in 1:m]
    θ′ = natural_update(θ, ∇U, F, M.ϵ, τs)
    f(θ′) = surrogate_objective(M, θ, θ′, τs)
    g(θ′) = surrogate_constraint(M, θ, θ′, τs)
    return linesearch(M, f, g, θ, θ′)
end
```

Algorithm 12.5. The update procedure for trust region policy optimization, which augments the natural gradient with a line search. It generates `m` trajectories using policy `π` in problem `𝒫` with initial state distribution `b` and depth `d`. To obtain the starting point of the line search, we need the gradient of the log-probability of the policy generating a particular action from the current state, which we denote as `∇logπ`. For the surrogate objective, we need the probability function `p`, which gives the probability that our policy generates a particular action from the current state. For the surrogate constraint, we need the divergence between the action distributions generated by π_θ and $\pi_{\theta'}$. At each step of the line search, we shrink the distance between the considered point `θ′` and `θ` while maintaining the search direction.

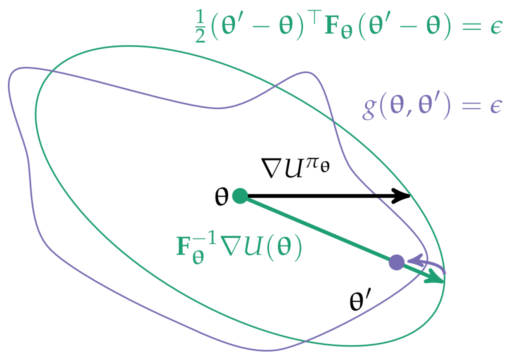
$\frac{1}{2}(\theta' - \theta)^\top \mathbf{F}_\theta(\theta' - \theta) = \epsilon$
$g(\theta, \theta') = \epsilon$
∇U^{π_θ}
θ
$\mathbf{F}_\theta^{-1}\nabla U(\theta)$
θ'

Figure 12.4. Trust region policy optimization searches within the elliptical constraint generated by a second-order approximation of the Kullback-Leibler divergence. After computing the natural policy gradient ascent direction, a line search is conducted to ensure that the updated policy improves the policy reward and adheres to the divergence constraint. The line search starts from the estimated maximum step size and reduces the step size along the ascent direction until a satisfactory point is found.

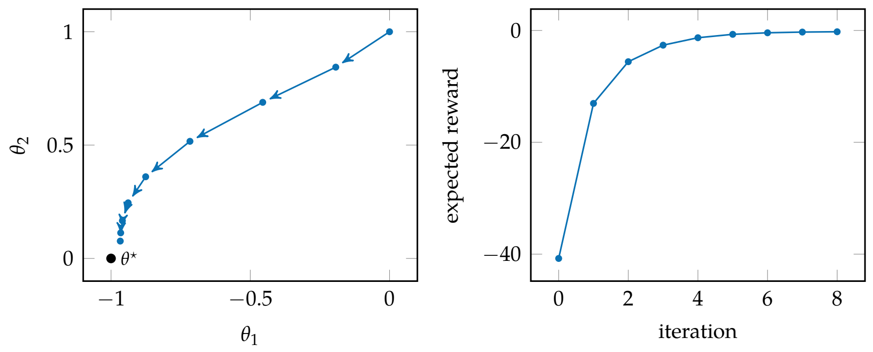

Figure 12.5. Trust region policy optimization applied to the simple regulator problem with rollouts to depth 10 with $\epsilon = 1$ and $c = 2$. The optimal policy parameterization is shown in black.

Example 12.1. An example of one iteration of trust region policy optimization.

Consider applying TRPO to the Gaussian policy $\mathcal{N}(\theta_1, \theta_2^2)$ from example 11.3 to the single-state MDP from example 11.1 with $\gamma = 1$. Recall that the gradient of the log policy likelihood is

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

Suppose that we run two rollouts with $\boldsymbol{\theta} = [0, 1]$ (this problem only has one state):

$$\begin{aligned}
\tau_1 &= \{(a = r = -0.532), (a = r = 0.597), (a = r = 1.947)\} \\
\tau_2 &= \{(a = r = -0.263), (a = r = -2.212), (a = r = 2.364)\}
\end{aligned}$$

The estimated Fisher information matrix is

$$\begin{aligned}
\mathbf{F}_{\boldsymbol{\theta}} &= \frac{1}{2}\left(\nabla \log p(\tau^{(1)}) \nabla \log p(\tau^{(1)})^\top + \nabla \log p(\tau^{(2)}) \nabla \log p(\tau^{(2)})^\top\right) \\
&= \frac{1}{2}\left(\begin{bmatrix} 4.048 & 2.878 \\ 2.878 & 2.046 \end{bmatrix} + \begin{bmatrix} 0.012 & -0.838 \\ -0.838 & 57.012 \end{bmatrix}\right) = \begin{bmatrix} 2.030 & 1.020 \\ 1.019 & 29.529 \end{bmatrix}
\end{aligned}$$

The objective function gradient is $[2.030, 1.020]$. The resulting descent direction $\mathbf{u}$ is $[1, 0]$. Setting $\epsilon = 0.1$, we compute our updated parameterization vector and obtain $\boldsymbol{\theta}' = [0.314, 1]$.

The surrogate objective function value at $\boldsymbol{\theta}$ is 1.485. Line search begins at $\boldsymbol{\theta}'$, where the surrogate objective function value is 2.110 and the constraint yields 0.049. This satisfies our constraint (as $0.049 < \epsilon$), so we return the new parameterization.

constraint, we can also eliminate line search and use standard gradient ascent methods.

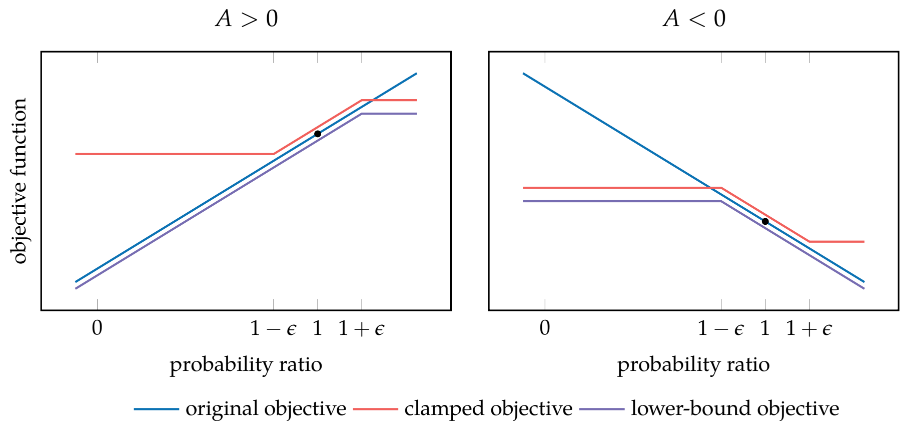

Figure 12.6. A visualization of the lower-bound objective for positive and negative advantages compared to the original objective and the clamped objective. The black point shows the baseline around which the optimization is performed, $\pi_{\theta'}(a \mid s)/\pi_\theta(a \mid s) = 1$. The three line plots in each axis are vertically separated for clarity.

The gradient of the unclamped objective equation (12.21) with action values is

$$\nabla_{\theta'} f(\theta, \theta') = \mathop{\mathbb{E}}_{s \sim b_{\gamma,\theta}}\left[\mathop{\mathbb{E}}_{a \sim \pi_\theta(\cdot \mid s)}\left[\frac{\nabla_{\theta'}\pi_{\theta'}(a \mid s)}{\pi_\theta(a \mid s)} Q_\theta(s,a)\right]\right] \tag{12.23}$$

where $Q_\theta(s,a)$ can be estimated from reward-to-go. The gradient of the lower-bound objective equation (12.22) (with clamping), is the same, except there is no contribution from experience tuples for which the objective is actively clamped. That is, if either the reward-to-go is positive and the probability ratio is greater than $1+\epsilon$, or if the reward-to-go is negative and the probability ratio is less than $1-\epsilon$, the gradient contribution is zero.

Like TRPO, the gradient can be computed for a parameterization θ' from experience generated from θ. Hence, several gradient updates can be run in a row using the same set of sampled trajectories. Algorithm 12.6 provides an implementation of this.

The clamped surrogate objective is compared to several other surrogate objectives in figure 12.7, which includes a line plot for the effective objective for TRPO:

$$\mathop{\mathbb{E}}_{\substack{s \sim b_{\gamma,\theta} \\ a \sim \pi_\theta(\cdot \mid s)}}\left[\frac{\pi_{\theta'}(a \mid s)}{\pi_\theta(a \mid s)} A_\theta(s,a) - \beta D_{\mathrm{KL}}(\pi_\theta(\cdot \mid s) \mid\mid \pi_{\theta'}(\cdot \mid s))\right] \tag{12.24}$$

```
struct ClampedSurrogateUpdate
    𝒫     # problem
    b     # initial state distribution
    d     # depth
    m     # number of trajectories
    π     # policy
    p     # policy likelihood
    ∇π    # policy likelihood gradient
    ϵ     # divergence bound
    α     # step size
    k_max # number of iterations per update
end

function clamped_gradient(M::ClampedSurrogateUpdate, θ, θ′, τs)
    d, p, ∇π, ϵ, γ = M.d, M.p, M.∇π, M.ϵ, M.𝒫.γ
    R(τ, j) = sum(r*γ^(k-1) for (k,(s,a,r)) in zip(j:d, τ[j:end]))
    ∇f(a,s,r_togo) = begin
        P = p(θ, a,s)
        w = p(θ′,a,s) / P
        if (r_togo > 0 && w > 1+ϵ) || (r_togo < 0 && w < 1-ϵ)
            return zeros(length(θ))
        end
        return ∇π(θ′, a, s) * r_togo / P
    end
    ∇f(τ) = mean(∇f(a,s,R(τ,k)) for (k,(s,a,r)) in enumerate(τ))
    return mean(∇f(τ) for τ in τs)
end

function update(M::ClampedSurrogateUpdate, θ)
    𝒫, b, d, m, π, α, k_max= M.𝒫, M.b, M.d, M.m, M.π, M.α, M.k_max
    πθ(s) = π(θ, s)
    τs = [simulate(𝒫, rand(b), πθ, d) for i in 1:m]
    θ′ = copy(θ)
    for k in 1:k_max
        θ′ += α*clamped_gradient(M, θ, θ′, τs)
    end
    return θ′
end
```

Algorithm 12.6. An implementation of clamped surrogate policy optimization, which returns a new policy parameterization for policy `π(s)` of an MDP `𝒫` with initial state distribution `b`. This implementation samples `m` trajectories to depth `d`, and then uses them to estimate the policy gradient in `k_max` subsequent updates. The policy gradient using the clamped objective is constructed using the policy gradients `∇p` with clamping parameter `ϵ`.

which is the trust region policy objective where the constraint is implemented as a penalty for some coefficient β. TRPO typically uses a hard constraint rather than a penalty because it is difficult to choose a value of β that performs well within a single problem, let alone across multiple problems.

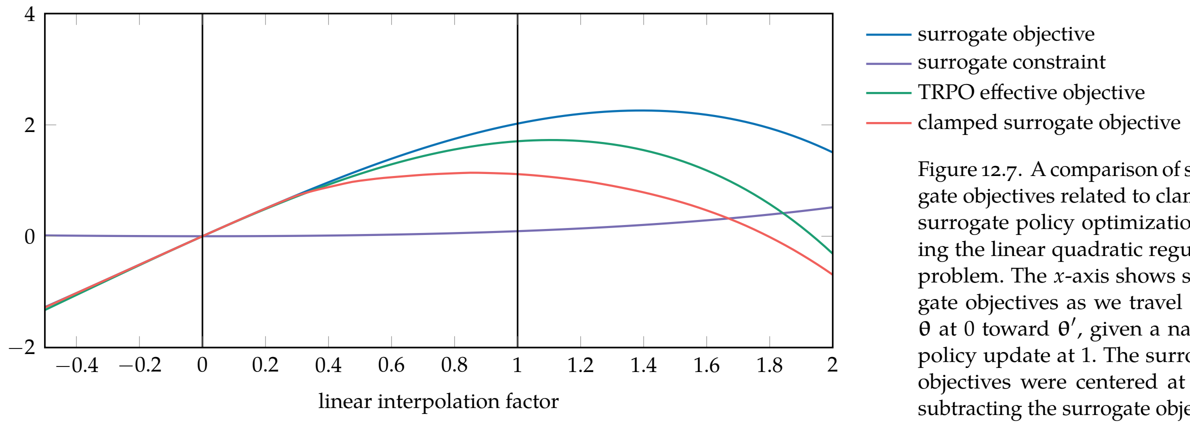

Figure 12.7. A comparison of surrogate objectives related to clamped surrogate policy optimization using the linear quadratic regulator problem. The x-axis shows surrogate objectives as we travel from θ at 0 toward θ', given a natural policy update at 1. The surrogate objectives were centered at 0 by subtracting the surrogate objective function value for θ. We see that the clamped surrogate objective behaves very similarly to the effective TRPO objective without needing a constraint. Note that ϵ and β can be adjusted for both algorithms, which would affect where the maximum is in each case.

12.6 Summary

- The gradient ascent algorithm can use the gradient estimates obtained from the methods discussed in the previous chapter to iteratively improve our policy.
- Gradient ascent can be made more robust by scaling, clipping, or forcing the size of the improvement steps to be uniform.
- The natural gradient approach uses a first-order approximation of the objective function with a constraint on the divergence between the trajectory distribution at each step, approximated using an estimate of the Fisher information matrix.
- Trust region policy optimization involves augmenting the natural gradient method with a line search to further improve the policy without additional trajectory simulations.
- We can use a pessimistic lower bound of the TRPO objective to obtain a clamped surrogate objective that performs similarly without the need for line search.

12.7 *Exercises*

Exercise 12.1. TRPO starts its line search from a new parameterization given by a natural policy gradient update. However, TRPO conducts the line search using a different objective than the natural policy gradient. Show that the gradient of the surrogate objective equation (12.18) used in TRPO is actually the same as the reward-to-go policy gradient equation (11.26).

Solution: The gradient of TRPO's surrogate objective is

$$\nabla_{\theta'} U_{\text{TRPO}} = \mathop{\mathbb{E}}_{s\sim b_{\gamma,\theta}}\left[\mathop{\mathbb{E}}_{a\sim\pi_\theta(\cdot|s)}\left[\frac{\nabla_{\theta'}\pi_{\theta'}(a \mid s)}{\pi_\theta(a \mid s)} Q_\theta(s,a)\right]\right]$$

When conducting the initial natural policy gradient update, the search direction is evaluated at $\theta' = \theta$. Furthermore, the action value is approximated with the reward-to-go:

$$\nabla_{\theta'} U_{\text{TRPO}} = \mathop{\mathbb{E}}_{s\sim b_{\gamma,\theta}}\left[\mathop{\mathbb{E}}_{a\sim\pi_\theta(\cdot|s)}\left[\frac{\nabla_{\theta}\pi_{\theta}(a \mid s)}{\pi_\theta(a \mid s)} r_{\text{to-go}}\right]\right]$$

Recall that the derivative of $\log f(x)$ is $f'(x)/f(x)$. It thus follows that

$$\nabla_{\theta'} U_{\text{TRPO}} = \mathop{\mathbb{E}}_{s\sim b_{\gamma,\theta}}\left[\mathop{\mathbb{E}}_{a\sim\pi_\theta(\cdot|s)}\left[\nabla_\theta \log \pi_\theta(a \mid s) r_{\text{to-go}}\right]\right]$$

which takes the same form as the reward-to-go policy gradient equation (11.26).

Exercise 12.2. Perform the calculations of example 12.1. First, compute the inverse of the Fisher information matrix $\mathbf{F}_\theta^{-1}$, compute $\mathbf{u}$, and compute the updated parameters θ'.

Solution: We start by computing the inverse of the Fisher information matrix:

$$\mathbf{F}_\theta^{-1} \approx \frac{1}{0.341(29.529) - 0.332(0.332)}\begin{bmatrix} 29.529 & -0.332 \\ -0.332 & 0.341 \end{bmatrix} \approx \begin{bmatrix} 0.501 & -0.017 \\ -0.017 & 0.034 \end{bmatrix}$$

Now, we update $\mathbf{u}$ as follows:

$$\mathbf{u} = \mathbf{F}_\theta^{-1}\nabla U(\theta) \approx \begin{bmatrix} 0.501 & -0.017 \\ -0.017 & 0.034 \end{bmatrix}\begin{bmatrix} 2.030 \\ 1.020 \end{bmatrix} \approx \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Finally, we estimate the updated parameters $\boldsymbol{\theta}$:

$$\begin{aligned}
\boldsymbol{\theta}' &= \boldsymbol{\theta} + \mathbf{u}\sqrt{\frac{2\epsilon}{\nabla U(\boldsymbol{\theta})^\top \mathbf{u}}} \\
&\approx \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \sqrt{\frac{2(0.1)}{\begin{bmatrix} 2.030 & 1.020 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}}} \\
&\approx \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \sqrt{\frac{0.2}{2.030}} \\
&\approx \begin{bmatrix} 0.314 \\ 1 \end{bmatrix}
\end{aligned}$$

Exercise 12.3. Suppose we have the parameterized policies $\pi_{\boldsymbol{\theta}}$ and $\pi_{\boldsymbol{\theta}'}$ given in the following table:

	a_1	a_2	a_3	a_4
$\pi_{\boldsymbol{\theta}}(a \mid s_1)$	0.1	0.2	0.3	0.4
$\pi_{\boldsymbol{\theta}'}(a \mid s_1)$	0.4	0.3	0.2	0.1
$\pi_{\boldsymbol{\theta}}(a \mid s_2)$	0.1	0.1	0.6	0.2
$\pi_{\boldsymbol{\theta}'}(a \mid s_2)$	0.1	0.1	0.5	0.3

Given that we sample the following five states, s_1, s_2, s_1, s_1, s_2, approximate $\mathbb{E}_s\left[D_{\mathrm{KL}}(\pi_{\boldsymbol{\theta}}(\cdot \mid s) \mid\mid \pi_{\boldsymbol{\theta}'}(\cdot \mid s))\right]$ using the definition

$$D_{\mathrm{KL}}(P \mid\mid Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

Solution: First, we compute the KL divergence for a state sample s_1:

$$D_{\mathrm{KL}}(\pi_{\boldsymbol{\theta}}(\cdot \mid s_1) \mid\mid \pi_{\boldsymbol{\theta}'}(\cdot \mid s_1)) = 0.1 \log\left(\tfrac{0.1}{0.4}\right) + 0.2 \log\left(\tfrac{0.2}{0.3}\right) + 0.3 \log\left(\tfrac{0.3}{0.3}\right) + 0.4 \log\left(\tfrac{0.4}{0.1}\right) \approx 0.456$$

Now, we compute the KL divergence for a state sample s_2:

$$D_{\mathrm{KL}}(\pi_{\boldsymbol{\theta}}(\cdot \mid s_2) \mid\mid \pi_{\boldsymbol{\theta}'}(\cdot \mid s_2)) = 0.1 \log\left(\tfrac{0.1}{0.1}\right) + 0.1 \log\left(\tfrac{0.1}{0.1}\right) + 0.6 \log\left(\tfrac{0.6}{0.5}\right) + 0.2 \log\left(\tfrac{0.2}{0.3}\right) \approx 0.0283$$

Finally, we compute the approximation of the expectation, which is the average KL divergence of the parameterized policies over the n state samples:

$$\begin{aligned}
\mathbb{E}_s[D_{\mathrm{KL}}(\pi_{\boldsymbol{\theta}}(\cdot \mid s) \mid\mid \pi_{\boldsymbol{\theta}'}(\cdot \mid s))] &\approx \frac{1}{n}\sum_{i=1}^{n} D_{\mathrm{KL}}\left(\pi_{\boldsymbol{\theta}}(\cdot \mid s^{(i)}) \,\middle|\middle|\, \pi_{\boldsymbol{\theta}'}(\cdot \mid s^{(i)})\right) \\
&\approx \frac{1}{5}(0.456 + 0.0283 + 0.456 + 0.456 + 0.0283) \\
&\approx 0.285
\end{aligned}$$