Chapter 13

Policy Gradient Methods

In this chapter we consider something new. So far in this book almost all the methods have learned the values of actions and then selected actions based on their estimated action values[1]; their policies would not even exist without the action-value estimates. In this chapter we consider methods that instead learn a *parameterized policy* that can select actions without consulting a value function. A value function may still be used to *learn* the policy parameter, but is not required for action selection. We use the notation $\boldsymbol{\theta} \in \mathbb{R}^d$ for the policy's parameter vector. Thus we write $\pi(a|s,\boldsymbol{\theta}) = \Pr\{A_t{=}a \mid S_t{=}s, \boldsymbol{\theta}_t{=}\boldsymbol{\theta}\}$ for the probability that action a is taken at time t given that the agent is in state s at time t with parameter $\boldsymbol{\theta}$. If a method uses a learned value function as well, then the value function's weight vector is denoted $\mathbf{w} \in \mathbb{R}^m$, as in $\hat{v}(s,\mathbf{w})$.

In this chapter we consider methods for learning the policy parameter based on the gradient of some performance measure $J(\boldsymbol{\theta})$ with respect to the policy parameter. These methods seek to *maximize* performance, so their updates approximate gradient *ascent* in J:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\widehat{\nabla J(\boldsymbol{\theta}_t)}, \tag{13.1}$$

where $\widehat{\nabla J(\boldsymbol{\theta}_t)}$ is a stochastic estimate whose expectation approximates the gradient of the performance measure with respect to its argument $\boldsymbol{\theta}_t$. All methods that follow this general schema we call *policy gradient methods*, whether or not they also learn an approximate value function. Methods that learn approximations to both policy and value functions are often called *actor–critic methods*, where 'actor' is a reference to the learned policy, and 'critic' refers to the learned value function, usually a state-value function. First we treat the episodic case, in which performance is defined as the value of the start state under the parameterized policy, before going on to consider the continuing case, in which performance is defined as the average reward rate, as in Section 10.3. In the end we are able to express the algorithms for both cases in very similar terms.

[1]The lone exception is the gradient bandit algorithms of Section 2.8. In fact, that section goes through many of the same steps, in the single-state bandit case, as we go through here for full MDPs. Reviewing that section would be good preparation for fully understanding this chapter.

13.1 Policy Approximation and its Advantages

In policy gradient methods, the policy can be parameterized in any way, as long as $\pi(a|s,\boldsymbol{\theta})$ is differentiable with respect to its parameters, that is, as long as $\nabla_{\boldsymbol{\theta}}\pi(a|s,\boldsymbol{\theta})$ exists and is always finite. In practice, to ensure exploration we generally require that the policy never becomes deterministic (i.e., that $\pi(a|s,\boldsymbol{\theta}) \in (0,1)\ \forall s,a,\boldsymbol{\theta}$. In this section we introduce the most common parameterization for discrete action spaces and point out the advantages it offers over action-value methods. Policy-based methods also offer useful ways of dealing with continuous action spaces, as we describe later in Section 13.7.

If the action space is discrete and not too large, then a natural kind of parameterization is to form parameterized numerical preferences $h(s,a,\boldsymbol{\theta}) \in \mathbb{R}$ for each state–action pair. The most preferred actions in each state are given the highest probability of being selected, for example, according to an exponential softmax distribution:

$$\pi(a|s,\boldsymbol{\theta}) = \frac{\exp(h(s,a,\boldsymbol{\theta}))}{\sum_b \exp(h(s,b,\boldsymbol{\theta})}, \tag{13.2}$$

where $\exp(x) = e^x$, where $e \approx 2.71828$ is the base of the natural logarithm. Note that the denominator here is just what is required so that the action probabilities in each state to sum to one. The preferences themselves can be parameterized arbitrarily. For example, they might be computed by a deep neural network, where $\boldsymbol{\theta}$ is the vector of all the connection weights of the network (as in the AlphaGo system described in Section 16.7). Or the preferences could simply be linear in features,

$$h(s,a,\boldsymbol{\theta}) = \boldsymbol{\theta}^\top \mathbf{x}(s,a), \tag{13.3}$$

using feature vectors $\mathbf{x}(s,a) \in \mathbb{R}^d$ constructed by any of the methods described in Chapter 9.

An immediate advantage of selecting actions according to the softmax in action preferences (13.2) is that the approximate policy can approach determinism, whereas with ε-greedy action selection over action values there is always an ε probability of selecting a random action. Of course, one could select according to a softmax over action values, but this alone would not approach determinism. Instead, the action-value estimates would converge to their corresponding true values, which would differ by a finite amount, translating to specific probabilities other than 0 and 1. If the softmax included a temperature parameter, then the temperature could be reduced over time to approach determinism, but in practice it would be difficult to choose the reduction schedule, or even the initial temperature, without more knowledge of the true action values than we would like to assume. Action preferences are different because they do not approach specific values; instead they are driven to produce the optimal stochastic policy. If the optimal policy is deterministic, then the preferences of the optimal actions will be driven infinitely higher than all suboptimal actions (if permited by the parameterization).

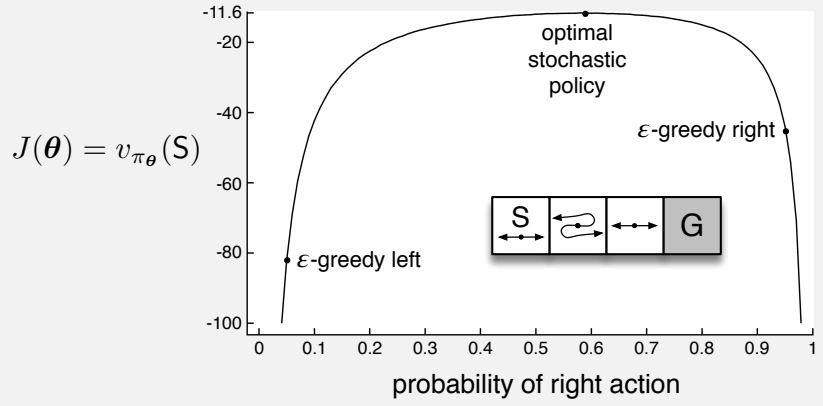
Perhaps the simplest advantage that policy parameterization may have over action-value parameterization is that the policy may be a simpler function to approximate.

Problems vary in the complexity of their policies and action-value functions. For some, the action-value function is simpler and thus easier to approximate. For others, the policy is simpler. In the latter case a policy-based method will typically be faster to learn and yield a superior asymptotic policy (as seems to be the case with Tetris; see Şimşek, Algórta, and Kothiyal, 2016).

In problems with significant function approximation, the best approximate policy may be stochastic. For example, in card games with imperfect information the optimal play is often to do two different things with specific probabilities, such as when bluffing in Poker. Action-value methods have no natural way of finding stochastic optimal policies, whereas policy approximating methods can, as shown in Example 13.1. This is a third significant advantage of policy-based methods.

Example 13.1 Short corridor with switched actions

Consider the small corridor gridworld shown inset in the graph below. The reward is -1 per step, as usual. In each of the three nonterminal states there are only two actions, right and left. These actions have their usual consequences in the first and third states, but in the second state they are reversed, so that right moves to the left and left moves to the right. The problem is difficult because all the states appear identical under the function approximation. In particular, we define $\mathbf{x}(s, \text{right}) = [1, 0]^\top$ and $\mathbf{x}(s, \text{left}) = [0, 1]^\top$, for all s. An action-value method with ε-greedy action selection is forced to choose between just two policies: choosing right with high probability $1 - \varepsilon/2$ on all steps or choosing left with the same high probability on all time steps. If $\varepsilon = 0.1$, then these two policies achieve a value (at the start state) of less than -44 and -82, respectively, as shown in the graph. A method can do significantly better if it can learn a specific probability with which to select right. The best probability is about 0.59, which achieves a value of about -11.6.



Exercise 13.1 Use your knowledge of the gridworld and its dynamics to determine an *exact* symbolic expression for the optimal probability of selecting the right action in Example 13.1. □

Finally, we note that the choice of policy parameterization is sometimes a good way of injecting prior knowledge about the desired form of the policy into the reinforcement learning system.

13.2 The Policy Gradient Theorem

In addition to the practical advantages of policy parameterization over ε-greedy action selection, there is also an important theoretical advantage. With continuous policy parameterization, the action probabilities changes smoothly as a function of the learned parameter, whereas in ε-greedy selection the action probabilities may change dramatically for an arbitrarily small change in the estimated action values, if that change results in a different action having the maximal value. Because of this, stronger convergence guarantees are available for policy-gradient methods than for action-value methods. In particular, it is the continuity of the parameterized policy that enables policy-gradient methods that approximate gradient ascent (13.1).

The episodic and continuing cases define the performance measure, $J(\boldsymbol{\theta})$, differently and thus have to be treated separately to some extent. Nevertheless, we will try to present both cases uniformly, and we develop a notation so that the major theoretical results can be decribed with a single set of equations.

In this section we treat the episodic case, for which we define the performance measure as the value of the start state of the episode. We can simplify the notation without losing any meaningful generality by assuming that every episode starts in some particular (non-random) state s_0. Then, in the episodic case we define performance as

$$J(\boldsymbol{\theta}) \doteq v_{\pi_{\boldsymbol{\theta}}}(s_0), \tag{13.4}$$

where $v_{\pi_{\boldsymbol{\theta}}}$ is the true value function for $\pi_{\boldsymbol{\theta}}$, the policy determined by $\boldsymbol{\theta}$.

With function approximation, it may seem challenging to change the policy parameter in a way that ensures improvement. The problem is that performance depends on both the action selections and the distribution of states in which those selections are made, and that both of these are affected by the policy parametre. Given a state, the effect of the policy parameter on the actions, and thus on reward, can be computed in a relatively straightforward way from knowledge of the parameterization. But the effect of the policy on the state distribution is completely a function of the environment and is typically unknown. How can we estimate the performance gradient with respect to the policy parameter, when the gradient depends on the unknown effect of changing the policy on the state distribution?

Fortunately, there is an excellent theoretical answer to this challenge in the form of the *policy gradient theorem*, which provides us an analytic expression for the gradient of performance with respect to the policy parameter (which is what we need to approximate for gradient ascent (13.1)) that does *not* involve the derivative

Proof of the Policy Gradient Theorem (episodic case)

With just elementary calculus and re-arranging terms we can prove the policy gradient theorem from first principles. To keep the notation simple, we leave it implicit in all cases that π is a function of $\boldsymbol{\theta}$, and all gradients are also implicitly with respect to $\boldsymbol{\theta}$. First note that the gradient of the state-value function can be written in terms of the action-value function as

$$
\begin{aligned}
\nabla v_\pi(s) &= \nabla \left[\sum_a \pi(a|s) q_\pi(s,a) \right], \quad \forall s \in \mathcal{S} && \text{(Exercise 3.11)} \\
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \nabla q_\pi(s,a) \Big] && \text{(product rule)} \\
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \nabla \sum_{s',r} p(s',r|s,a) \big(r + \gamma v_\pi(s') \big) \Big] \\
& && \text{(Exercise 3.12 and Equation 3.8)} \\
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \sum_{s'} \gamma p(s'|s,a) \nabla v_\pi(s') \Big] \\
& && \text{(Eq. 3.10)} \\
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \sum_{s'} \gamma p(s'|s,a) && \text{(unrolling)} \\
&\qquad \sum_{a'} \big[\nabla \pi(a'|s') q_\pi(s',a') + \pi(a'|s') \sum_{s''} \gamma p(s''|s',a') \nabla v_\pi(s'') \big] \Big] \\
&= \sum_{x \in \mathcal{S}} \sum_{k=0}^{\infty} \gamma^k \Pr(s \to x, k, \pi) \sum_a \nabla \pi(a|x) q_\pi(x,a),
\end{aligned}
$$

after repeated unrolling, where $\Pr(s \to x, k, \pi)$ is the probability of transitioning from state s to state x in k steps under policy π. It is then immediate that

$$
\begin{aligned}
\nabla J(\boldsymbol{\theta}) &= \nabla v_\pi(s_0) \\
&= \sum_s \sum_{k=0}^{\infty} \gamma^k \Pr(s_0 \to s, k, \pi) \sum_a \nabla \pi(a|s) q_\pi(s,a) \\
&= \sum_s \mu_\pi(s) \sum_a \nabla \pi(a|s) q_\pi(s,a). && \text{Q.E.D.}
\end{aligned}
$$

of the state distribution. The policy gradient theorem is that

$$\nabla J(\boldsymbol{\theta}) = \sum_s \mu_\pi(s) \sum_a q_\pi(s,a) \nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta}), \tag{13.5}$$

where the gradients in all cases are the column vectors of partial derivatives with respect to the components of $\boldsymbol{\theta}$, and π denotes the policy corresponding to parameter vector $\boldsymbol{\theta}$. The notion of the distribution μ_π here should be clear from what transpired in Chapters 9 and 10. That is, in the episodic case, $\mu_\pi(s)$ is defined to be the expected number of time steps t on which $S_t = s$ in a randomly generated episode starting in s_0 and following π and the dynamics of the MDP. The policy gradient theorem is proved for the episodic case in the box.

13.3 REINFORCE: Monte Carlo Policy Gradient

We are now ready for our first policy-gradient learning algorithm. Recall our overall strategy of stochastic gradient ascent (13.1), for which we need a way of obtaining samples whose expectation is equal to the performance gradient. The policy gradient theorem gives us an exact expression for this gradient; all we need is some way of sampling whose expectation equals or approximates this expression. Notice that the right-hand side of the policy gradient theorem is a sum over states weighted by how often the states occurs under the target policy π, weighted again by γ times how many steps it takes to get to those states; if we just follow π we will encounter states in these proportions, which we can then weight by γ^t to preserve the expected value. Thus

$$\begin{aligned}
\nabla J(\boldsymbol{\theta}) &= \sum_s \mu_\pi(s) \sum_a q_\pi(s,a) \nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta}), \qquad (13.5)\\
&= \mathbb{E}_\pi \left[\gamma^t \sum_a q_\pi(S_t,a) \nabla_{\boldsymbol{\theta}} \pi(a|S_t,\boldsymbol{\theta}) \right].
\end{aligned}$$

This is good progress, and we would like to carry it further and handle the action in the same way (replacing a with the sample action A_t). The remaining part of the expectation above is a sum over actions; if only each term was weighted by the probability of selecting the actions, that is, according to $\pi(a|S_t,\boldsymbol{\theta})$. So let us make it that way, multiplying and dividing by this probability. Continuing from the previous equation, this gives us

$$\begin{aligned}
\nabla J(\boldsymbol{\theta}) &= \mathbb{E}_\pi \left[\gamma^t \sum_a \pi(a|S_t,\boldsymbol{\theta}) q_\pi(S_t,a) \frac{\nabla_{\boldsymbol{\theta}} \pi(a|S_t,\boldsymbol{\theta})}{\pi(a|S_t,\boldsymbol{\theta})} \right]\\
&= \mathbb{E}_\pi \left[\gamma^t q_\pi(S_t,A_t) \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})} \right] \quad \text{(replacing } a \text{ by the sample } A_t \sim \pi)\\
&= \mathbb{E}_\pi \left[\gamma^t G_t \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})} \right] \qquad \text{(because } \mathbb{E}_\pi[G_t|S_t,A_t] = q_\pi(S_t,A_t))
\end{aligned}$$

which is exactly what we want, a quantity that we can sample on each time step whose expectation is equal to the gradient. Using this sample to instantiate our generic stochastic gradient ascent algorithm (13.1), we obtain the update

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \gamma^t G_t \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})}. \tag{13.6}$$

We call this algorithm REINFORCE (after Williams, 1992). Its update has an intuitive appeal. Each increment is proportional to the product of a return G_t and a vector, the gradient of the probability of taking the action actually taken, divided by the probability of taking that action. The vector is the direction in parameter space that most increases the probability of repeating the action A_t on future visits to state S_t. The update increases the parameter vector in this direction proportional to the return, and inversely proportional to the action probability. The former makes sense because it causes the parameter to move most in the directions that favor actions that yield the highest return. The latter makes sense because otherwise actions that are selected frequently are at an advantage (the updates will be more often in their direction) and might win out even if they do not yield the highest return.

Note that REINFORCE uses the complete return from time t, which includes all future rewards up until the end of the episode. In this sense REINFORCE is a Monte Carlo algorithm and is well defined only for the episodic case with all updates made in retrospect after the episode is completed (like the Monte Carlo algorithms in Chapter 5). This is shown explicitly in the boxed pseudocode below.

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta}), \forall a \in \mathcal{A}, s \in \mathcal{S}, \boldsymbol{\theta} \in \mathbb{R}^d$
Initialize policy parameter $\boldsymbol{\theta}$
Repeat forever:
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  For each step of the episode $t = 0, \ldots, T-1$:
    $G \leftarrow$ return from step t
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t, \boldsymbol{\theta})$

The vector $\frac{\nabla_{\boldsymbol{\theta}} \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})}$ in the REINFORCE update is the only place the policy parameterization appears in the algorithm. This vector has been given several names and notations in the literature; we will refer to it simply as the *eligibility vector*. The eligibility vector is often written in the compact form $\nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t, \boldsymbol{\theta})$, using the identity $\nabla \log x = \frac{\nabla x}{x}$. This form is used in all the boxed pseudocode in this chapter. In earlier examples in this chapter we considered exponential softmax policies (13.2) with linear action preferences (13.3). For this parameterization, the eligibility vector is

$$\nabla_{\boldsymbol{\theta}} \log \pi(a|s, \boldsymbol{\theta}) = \mathbf{x}(s, a) - \sum_b \pi(b|s, \boldsymbol{\theta}) \mathbf{x}(s, b). \tag{13.7}$$

As a stochastic gradient method, REINFORCE has good theoretical convergence properties. By construction, the expected update over an episode is in the same direction as the performance gradient.[2] This assures an improvement in expected performance for sufficiently small α, and convergence to a local optimum under standard stochastic approximation conditions for decreasing α. However, as a Monte Carlo method REINFORCE may be of high variance and thus slow to learn.

Exercise 13.2 Prove (13.7) using the definitions and elementary calculus. □

13.4 REINFORCE with Baseline

The policy gradient theorem (13.5) can be generalized to include a comparison of the action value to an arbitrary *baseline* $b(s)$:

$$\nabla J(\boldsymbol{\theta}) = \sum_s \mu_\pi(s) \sum_a \Big(q_\pi(s,a) - b(s)\Big) \nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta}). \tag{13.8}$$

The baseline can be any function, even a random variable, as long as it does not vary with a; the equation remains true, because the the subtracted quantity is zero:

$$\sum_a b(s) \nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta}) = b(s) \nabla_{\boldsymbol{\theta}} \sum_a \pi(a|s,\boldsymbol{\theta}) = b(s) \nabla_{\boldsymbol{\theta}} 1 = 0 \qquad \forall s \in \mathcal{S}.$$

However, after we convert the policy gradient theorem to an expectation and an update rule, using the same steps as in the previous section, then the baseline can have a significant effect on the *variance* of the update rule.

The update rule that we end up with is a new version of REINFORCE that includes a general baseline:

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \gamma^t \Big(G_t - b(S_t)\Big) \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})}. \tag{13.9}$$

As the baseline could be uniformly zero, this update is a strict generalization of REINFORCE. In general, the baseline leaves the expected value of the update unchanged, but it can have a large effect on its variance. For example, we saw in Section 2.8 that an analogous baseline can significantly reduce the variance (and thus speed the learning) of gradient bandit algorithms. In the bandit algorithms the baseline was just a number (the average of the rewards seen so far), but for MDPs the baseline should vary with state. In some states all actions have high values and we need a high baseline to differentiate the higher valued actions from the less highly valued ones; in other states all actions will have low values and a low baseline is appropriate.

[2] Technically, this is only true if each episode's updates are done *off-line*, meaning they are accumulated on the side during the episode and only used to change $\boldsymbol{\theta}$ by their sum at the episode's end. However, this would probably be a worse algorithm in practice, and its desireable theoretical properties would probably be shared by the algorithm as given (although this has not been proved).

One natural choice for the baseline is an estimate of the state value, $\hat{v}(S_t,\mathbf{w})$, where $\mathbf{w} \in \mathbb{R}^m$ is a weight vector learned by one of the methods presented in previous chapters. Because REINFORCE is a Monte Carlo method for learning the policy parameter, $\boldsymbol{\theta}$, it seems natural to also use a Monte Carlo method to learn the state-value weights, $\mathbf{w}$. A complete pseudocode algorithm for REINFORCE with baseline is given in the box using such a learned state-value function as the baseline.

REINFORCE with Baseline (episodic)

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta}), \forall a \in \mathcal{A}, s \in \mathcal{S}, \boldsymbol{\theta} \in \mathbb{R}^d$
Input: a differentiable state-value parameterization $\hat{v}(s,\mathbf{w}), \forall s \in \mathcal{S}, \mathbf{w} \in \mathbb{R}^m$
Parameters: step sizes $\alpha > 0$, $\beta > 0$

Initialize policy parameter $\boldsymbol{\theta}$ and state-value weights $\mathbf{w}$
Repeat forever:
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  For each step of the episode $t = 0, \ldots, T-1$:
    $G_t \leftarrow$ return from step t
    $\delta \leftarrow G_t - \hat{v}(S_t,\mathbf{w})$
    $\mathbf{w} \leftarrow \mathbf{w} + \beta\,\delta\,\nabla_{\mathbf{w}}\hat{v}(S_t,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t\,\delta\,\nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t,\boldsymbol{\theta})$

This algorithm has two step sizes, α and β. The step size for values (here β) is relatively easy; in the linear case we have rules of thumb for setting it, such as $\beta = 0.1/\mathbb{E}\big[\|\mathbf{x}_t\|_\mu^2\big]$. For action values though it is much less clear. It depends on the range of variation of the rewards and on the policy parameterization.

13.5 Actor–Critic Methods

Although the REINFORCE-with-baseline method learns both a policy and a state-value function, we do not consider it to be an actor–critic method because its state-value function is used only as a baseline, not as a critic. That is, it is not used for bootstrapping (updating a state from the estimated values of subsequent states), but only as a baseline for the state being updated. This is a useful distinction, for only through bootstrapping do we introduce bias and an asymptotic dependence on the quality of the function approximation. As we have seen, the bias introduced through bootstrapping and reliance on the state representation is often on balance beneficial because it reduces variance and accelerates learning. REINFORCE with baseline is unbiased and will converge asymptotically to a local minimum, but like all Monte Carlo methods it tends to be slow to learn (of high variance) and inconvenient to implement online or for continuing problems. As we have seen earlier in this book, with temporal-difference methods we can eliminate these inconveniences, and through multi-step methods we can flexibly choose the degree of bootstrapping. In order to gain these advantages in the case of policy gradient methods we use actor–

critic methods with a true bootstrapping critic.

First consider one-step actor–critic methods, the analog of the TD methods introduced in Chapter 6 such as TD(0), Sarsa(0), and Q-learning. The main appeal of one-step methods is that they are fully online and incremental, yet avoid the complexities of eligibility traces. They are a special case of the eligibility trace methods, and not as general, but easier to understand. One-step actor–critic methods replace the full return of REINFORCE (13.9) with the one-step return (and use a learned state-value function as the baseline) as follow:

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha\gamma^t\Big(G_{t:t+1} - \hat{v}(S_t,\mathbf{w})\Big)\frac{\nabla_{\boldsymbol{\theta}}\pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})} \tag{13.10}$$

$$= \boldsymbol{\theta}_t + \alpha\gamma^t\Big(R_{t+1} + \gamma\hat{v}(S_{t+1},\mathbf{w}) - \hat{v}(S_t,\mathbf{w})\Big)\frac{\nabla_{\boldsymbol{\theta}}\pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})} \tag{13.11}$$

$$= \boldsymbol{\theta}_t + \alpha\gamma^t\delta_t\frac{\nabla_{\boldsymbol{\theta}}\pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})}. \tag{13.12}$$

The natural state-value-function learning method to pair with this is semi-gradient TD(0). Pseudocode for the complete algorithm is given in the box below. Note that it is now a fully online, incremental algorithm, with states, actions, and rewards processed as they occur and then never revisited.

One-step Actor–Critic (episodic)

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta}), \forall a \in \mathcal{A}, s \in \mathcal{S}, \boldsymbol{\theta} \in \mathbb{R}^d$
Input: a differentiable state-value parameterization $\hat{v}(s,\mathbf{w}), \forall s \in \mathcal{S}, \mathbf{w} \in \mathbb{R}^m$
Parameters: step sizes $\alpha > 0$, $\beta > 0$

Initialize policy parameter $\boldsymbol{\theta}$ and state-value weights $\mathbf{w}$
Repeat forever:
  Initialize S (first state of episode)
  $I \leftarrow 1$
  While S is not terminal:
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \beta\delta\nabla_{\mathbf{w}}\hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha I\delta\nabla_{\boldsymbol{\theta}}\log\pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

The generalizations to the forward view of multi-step methods and then to a λ-return algorithm are straightforward. The one-step return in (13.10) is merely replaced by $G^{\lambda}_{t:t+k}$ and G^{λ}_t respectively. The backward views are also straightforward, using separate eligibility traces for the actor and critic, each after the patterns in Chapter 12. Pseudocode for the complete algorithm is given in the box on the next page.

Actor–Critic with Eligibility Traces (episodic)

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta}), \forall a \in \mathcal{A}, s \in \mathcal{S}, \boldsymbol{\theta} \in \mathbb{R}^d$
Input: a differentiable state-value parameterization $\hat{v}(s,\mathbf{w}), \forall s \in \mathcal{S}, \mathbf{w} \in \mathbb{R}^m$
Parameters: step sizes $\alpha > 0$, $\beta > 0$

Initialize policy parameter $\boldsymbol{\theta}$ and state-value weights $\mathbf{w}$
Repeat forever (for each episode):
  Initialize S (first state of episode)
  $\mathbf{e}^{\boldsymbol{\theta}} \leftarrow \mathbf{0}$ (n-component eligibility trace vector)
  $\mathbf{e}^{\mathbf{w}} \leftarrow \mathbf{0}$ (m-component eligibility trace vector)
  $I \leftarrow 1$
  While S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{e}^{\mathbf{w}} \leftarrow \gamma\lambda^{\mathbf{w}}\mathbf{e}^{\mathbf{w}} + I\nabla_{\mathbf{w}}\hat{v}(S,\mathbf{w})$
    $\mathbf{e}^{\boldsymbol{\theta}} \leftarrow \gamma\lambda^{\boldsymbol{\theta}}\mathbf{e}^{\boldsymbol{\theta}} + I\nabla_{\boldsymbol{\theta}} \log \pi(A|S,\boldsymbol{\theta})$
    $\mathbf{w} \leftarrow \mathbf{w} + \beta\delta\mathbf{e}^{\mathbf{w}}$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\delta\mathbf{e}^{\boldsymbol{\theta}}$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

13.6 Policy Gradient for Continuing Problems

As discussed in Section 10.3, for continuing problems without episode boundaries we need to define performance in terms of the average rate of reward per time step:

$$\begin{aligned} J(\boldsymbol{\theta}) \doteq r(\pi) &\doteq \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}[R_t \mid A_{0:t-1} \sim \pi] \\ &= \lim_{t\to\infty} \mathbb{E}[R_t \mid A_{0:t-1} \sim \pi], \\ &= \sum_s \mu_\pi(s) \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) r, \end{aligned} \tag{10.6}$$

where μ_π is the steady-state distribution under π, $\mu_\pi(s) \doteq \lim_{t\to\infty} \Pr\{S_t = s | A_{0:t} \sim \pi\}$, which is assumed to exist and to be independent of S_0 (an ergodicity assumption). Remember that this is the special distribution under which, if you select actions according to π, you remain in the same distribution:

$$\sum_s \mu_\pi(s) \sum_a \pi(a|s,\boldsymbol{\theta}) p(s'|s,a) = \mu_\pi(s'). \tag{10.7}$$

We also define values, $v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s]$ and $q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$, with respect to the differential return:

$$G_t \doteq R_{t+1} - \eta(\pi) + R_{t+2} - \eta(\pi) + R_{t+3} - \eta(\pi) + \cdots. \tag{10.8}$$

With these alternate definitions, and $\gamma \doteq 1$, the policy gradient theorem as given for the episodic case (13.5) remains true for the continuing case. A proof is given in the box on the next page. The forward and backward view equations also remain the same. Complete pseudocode for the backward view is given in the box below.

Actor–Critic with Eligibility Traces (continuing)

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta}), \forall a \in \mathcal{A}, s \in \mathcal{S}, \boldsymbol{\theta} \in \mathbb{R}^d$
Input: a differentiable state-value parameterization $\hat{v}(s,\mathbf{w}), \forall s \in \mathcal{S}, \mathbf{w} \in \mathbb{R}^m$
Parameters: step sizes $\alpha > 0$, $\beta > 0$, $\eta > 0$

$\mathbf{e}^{\boldsymbol{\theta}} \leftarrow \mathbf{0}$ (n-component eligibility trace vector)
$\mathbf{e}^{\mathbf{w}} \leftarrow \mathbf{0}$ (m-component eligibility trace vector)
Initialize $\bar{R} \in \mathbb{R}$ (e.g., to 0)
Initialize policy parameter $\boldsymbol{\theta}$ and state-value weights $\mathbf{w}$ (e.g., to $\mathbf{0}$)
Initialize $S \in \mathcal{S}$ (e.g., to s_0)
Repeat forever:
  $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
  Take action A, observe S', R
  $\delta \leftarrow R - \bar{R} + \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
  $\bar{R} \leftarrow \bar{R} + \eta\delta$
  $\mathbf{e}^{\mathbf{w}} \leftarrow \lambda^{\mathbf{w}}\mathbf{e}^{\mathbf{w}} + \nabla_{\mathbf{w}}\hat{v}(S,\mathbf{w})$
  $\mathbf{e}^{\boldsymbol{\theta}} \leftarrow \lambda^{\boldsymbol{\theta}}\mathbf{e}^{\boldsymbol{\theta}} + \nabla_{\boldsymbol{\theta}} \log \pi(A|S,\boldsymbol{\theta})$
  $\mathbf{w} \leftarrow \mathbf{w} + \beta\delta\mathbf{e}^{\mathbf{w}}$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\delta\mathbf{e}^{\boldsymbol{\theta}}$
  $S \leftarrow S'$

Proof of the Policy Gradient Theorem (continuing case)

The proof of the policy gradient theorem for the continuing case begins similarly to the episodic case. Again we leave it implicit in all cases that π is a function of $\boldsymbol{\theta}$ and that the gradients are with respect to $\boldsymbol{\theta}$. Recall that in the continuing case $J(\boldsymbol{\theta}) = r(\pi)$ (10.6) and that v_π and q_π denote values with respect to the differential return (10.8). The gradient of the state-value function can be written as

$$\begin{aligned}
\nabla v_\pi(s) &= \nabla \left[\sum_a \pi(a|s) q_\pi(s,a) \right], \quad \forall s \in \mathcal{S} && \text{(Exercise 3.15)} \\
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \nabla q_\pi(s,a) \Big] && \text{(product rule)} \\
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \nabla \sum_{s',r} p(s',r|s,a) \big(r - r(\boldsymbol{\theta}) + \gamma v_\pi(s') \big) \Big] \\
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \big[-\nabla r(\boldsymbol{\theta}) + \sum_{s'} \gamma p(s'|s,a) \nabla v_\pi(s') \big] \Big].
\end{aligned}$$

After re-arranging terms, we obtain

$$\nabla r(\boldsymbol{\theta}) = \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \sum_{s'} \gamma p(s'|s,a) \nabla v_\pi(s') \Big] - \nabla v_\pi(s), \ \ \forall s \in \mathcal{S}.$$

Notice that the left-hand side can be written $\nabla J(\boldsymbol{\theta})$ and that it does not depend on s. Thus the right-hand side does not depend on s either, and we can safely sum it over all $s \in \mathcal{S}$, weighted by $\mu_\pi(s)$, without changing it (because $\sum_s \mu_\pi(s) = 1$). Thus

$$\begin{aligned}
\nabla J(\boldsymbol{\theta}) &= \sum_s \mu_\pi(s) \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \sum_{s'} \gamma p(s'|s,a) \nabla v_\pi(s') \Big] - \nabla v_\pi(s) \\
&= \sum_s \mu_\pi(s) \sum_a \nabla \pi(a|s) q_\pi(s,a) \\
&\quad + \mu_\pi(s) \sum_a \pi(a|s) \sum_{s'} \gamma p(s'|s,a) \nabla v_\pi(s') - \mu_\pi(s) \sum_a \nabla v_\pi(s) \\
&= \sum_s \mu_\pi(s) \sum_a \nabla \pi(a|s) q_\pi(s,a) \\
&\quad + \sum_{s'} \underbrace{\sum_s \mu_\pi(s) \sum_a \pi(a|s) p(s'|s,a)}_{\mu_\pi(s') \ \text{(10.7)}} \nabla v_\pi(s') - \sum_s \mu_\pi(s) \nabla v_\pi(s) \\
&= \sum_s \mu_\pi(s) \sum_a \nabla \pi(a|s) q_\pi(s,a) + \sum_{s'} \mu_\pi(s') \nabla v_\pi(s') - \sum_s \mu_\pi(s) \nabla v_\pi(s) \\
&= \sum_s \mu_\pi(s) \sum_a \nabla \pi(a|s) q_\pi(s,a). \qquad \text{Q.E.D.}
\end{aligned}$$

13.7 Policy Parameterization for Continuous Actions

Policy-based methods offer practical ways of dealing with large actions spaces, even continuous spaces with an infinite number of actions. Instead of computing learned probabilities for each of the many actions, we instead compute learned the statistics of the probability distribution. For example, the action set might be the real numbers, with actions chosen from a normal (Gaussian) distribution.

The conventional probability density function for the normal distribution is written

$$p(x) \doteq \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \tag{13.13}$$

where μ and σ here are the mean and standard deviation of the normal distribution, and of course π here is just the number $\pi \approx 3.14159$. The probability density function for several different means and standard deviations is shown in Figure 13.1. The value $p(x)$ is the *density* of the probability at x, not the probability. It can be greater than 1; it is the total area under $p(x)$ that must sum to 1. In general, one can take the integral under $p(x)$ for any range of x values to get the probability of x falling within that range.

To produce a policy parameterization, we can define the policy as the normal probability density over a real-valued scalar action, with mean and standard deviation give by parametric function approximators. That is, we define

$$\pi(a|s,\boldsymbol{\theta}) \doteq \frac{1}{\sigma(s,\boldsymbol{\theta})\sqrt{2\pi}} \exp\left(-\frac{(a-\mu(s,\boldsymbol{\theta}))^2}{2\sigma(s,\boldsymbol{\theta})^2}\right). \tag{13.14}$$

To complete the example we need only give a form for the approximators for the mean and standard-deviation functions. For this we divide the policy's parameter vector into two parts, $\boldsymbol{\theta} = [\boldsymbol{\theta}^\mu, \boldsymbol{\theta}^\sigma]^\top$, one part to be used for the approximation of the mean and one part for the approximation of the standard deviation. The mean

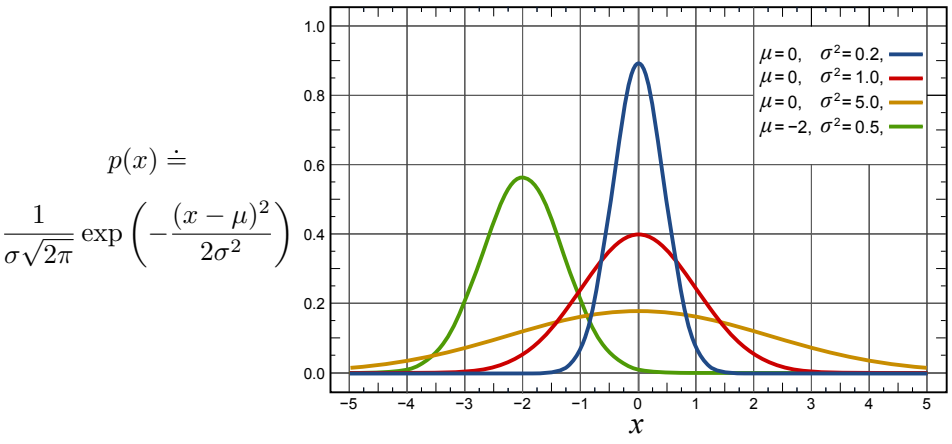


Figure 13.1: The probability density function of the normal distribution for different means and variances.

can be approximated as a linear function. The standard deviation must always be positive and is better approximated as the exponential of a linear function. Thus

$$\mu(s, \mathbf{w}) \doteq \boldsymbol{\theta}^{\mu\top}\mathbf{x}(s) \quad \text{and} \quad \sigma(s, \boldsymbol{\theta}) \doteq \exp\Big(\boldsymbol{\theta}^{\sigma\top}\mathbf{x}(s)\Big), \tag{13.15}$$

where $\mathbf{x}(s)$ is a state feature vector constructed perhaps by one of the methods described in Chapter 9. With these definitions, all the algorithms described in the rest of this chapter can be applied to learn to select real-valued actions.

Exercise 13.3 A *Bernoulli-logistic unit* is a stochastic neuron-like unit used in some artificial neural networks (see Section 9.6). Its input at time t is a feature vector $\mathbf{x}(S_t)$; its output, A_t, is a random variable having two values, 0 and 1, with $\Pr\{A_t = 1\} = P_t$ and $\Pr\{A_t = 0\} = 1 - P_t$ (the Bernoulli distribution). Let $h(s, 0, \boldsymbol{\theta})$ and $h(s, 1, \boldsymbol{\theta})$ be the preferences in state s for for the unit's two actions given policy parameter $\boldsymbol{\theta}$. Assume that the difference between the preferences is given by a weighted sum of the unit's input vector, that is, assume that $h(s, 1, \boldsymbol{\theta}) - h(s, 0, \boldsymbol{\theta}) = \boldsymbol{\theta}^\top\mathbf{x}(s)$, where $\boldsymbol{\theta}$ is the unit's weight vector.
(a) Show that if the exponential softmax distribution (13.2) is used to convert preferences to policies, then $P_t = \pi(1|S_t, \boldsymbol{\theta}_t) = 1/(1 + \exp(-\boldsymbol{\theta}_t^\top\mathbf{x}(S_t)))$ (the logistic function).
(b) What is the Monte-Carlo REINFORCE update of $\boldsymbol{\theta}_t$ to $\boldsymbol{\theta}_{t+1}$ upon receipt of return G_t?
(c) Express the eligibility $\nabla_{\boldsymbol{\theta}} \log \pi(a|s, \boldsymbol{\theta})$ for a Bernoulli-logistic unit, in terms of a, $\mathbf{x}(s)$, and $\pi(a|s, \boldsymbol{\theta})$ by calculating the gradient. Hint: separately for each action compute the derivative of the log first with respect to $p = \pi(1|s, \boldsymbol{\theta})$, combine the two results into one expression that depends on a and p, and then use the chain rule, noting that the derivative of the logistic function $f(x)$ is $f(x)(1 - f(x))$. □

13.8 Summary

Prior to this chapter, this book has focused on *action-value methods*—meaning methods that learn action values and then use them to determine action selections. In this chapter, on the other hand, we have considered methods that learn a parameterized policy that enables actions to be taken without consulting action-value estimates—though action-value estimates may still be learned and used to update the policy parameter. In particular, we have considered *policy-gradient methods*—meaning methods that update the policy parameter on each step in the direction of an estimate of performance with respect to the policy parameter.

Methods that learn and store a policy parameter have many advantages. They can learn specific probabilities for their actions. They can learn appropriate levels of exploration and approach determinism asymptotically. They can naturally handle continuous state spaces. All these things are easy for policy-based methods, but awkward or impossible for ε-greedy methods and for action-value methods in general. In addition, on some problems the policy is just simpler to represent parametrically than the value function; these are more suited to parameterized policy methods.

Parameterized policy methods also have an important theoretical advantage over action-value methods in the form of the *policy gradient theorem*, which gives an exact formula for how performance is affected by the policy parameter that does not involve derivatives of the state distribution. This theorem provides a theoretical foundation for all policy gradient methods.

The *REINFORCE* method follows directly from the policy gradient theorem. Adding a state-value function as a *baseline* reduces REINFORCE's variance without introducing bias. Using the state-value function for bootstrapping results introduces bias, but is often desirable for the same reason that bootstrapping TD methods are often superior to Monte Carlo methods (substantially reduced variance). The state-value function assigns credit to—critizes—the policy's action selections, and accordingly the former is termed the *critic* and the latter the *actor*, and these overall methods are sometimes termed *actor–critic* methods.

Overall, policy-gradient methods provide a significantly different set of proclivities, strengths, and weaknesses than action-value methods. Today they are less well understood, but a subject of excitement and ongoing research.

Bibliographical and Historical Remarks

Methods that we now see as related to policy gradients were actually some of the earliest to be studied in reinforcement learning (Witten, 1977; Barto, Sutton, and Anderson, 1983; Sutton, 1984; Williams, 1987, 1992) and in predecessor fields (Phansalkar and Thathachar, 1995). They were largely supplanted in the 1990s by the action-value methods that are the focus of the other chapters of this book. In recent years, however, extensive attention has returned to actor–critic methods and to policy-gradient methods in general. Among the further developments beyond what we cover here are natural-gradient methods (Amari, 1998; Kakade, 2002, Peters, Vijayakumar and Schaal, 2005; Peters and Schall, 2008; Park, Kim and Kang, 2005; Bhatnagar, Sutton, Ghavamzadeh and Lee, 2009; see Grondman, Busoniu, Lopes and Babuska, 2012), and deterministic policy gradient (Silver et al., 2014). Major applications include acrobatic helicopter autopilots and AlphaGo (see Section 16.7).

Our presentation in this chapter is based primarily on that by Sutton, McAllester, Singh, and Mansour (2000), who introduced the term "policy gradient methods". A useful overview is provided by Bhatnagar et al. (2003). One of the earliest related works is by Aleksandrov, Sysoyev, and Shemeneva (1968).

13.1 Example 13.1 was implemented by Eric Graves.

13.2 The policy gradient theorem was first obtained by Marbach and Tsitsiklis (1998, 2001) and then independently by Sutton et al. (2000). A similar expression was obtained by Cao and Chen (1997). Other early results are due to Konda and Tsitsiklis (2000, 2003) and Baxter and Bartlett (2000).

13.3 REINFORCE is due to Williams (1987, 1992). The use of a power of the

discount factor in the update is due to Thomas (2014).

Phansalkar and Thathachar (1995) proved both local and global convergence theorems for modified versions of REINFORCE algorithms.

13.4 The baseline was introduced in Williams's (1987, 1992) original work. Greensmith, Bartlett, and Baxter (2004) analyzed an arguable better baseline (see Dick, 2015).

13.5 Actor–critic methods were among the earliest to be investigated in reinforcement learning (Witten, 1977; Barto, Sutton, and Anderson, 1983; Sutton, 1984). The algorithms presented here and in Section 13.6 are based on the work of Degris, White, and Sutton (2012), who also introduced the study of off-policy policy-gradient methods.

13.7 The first to show how continuous actions could be handled this way appears to have been Williams (1987, 1992).