Chapter 10

On-policy Control with Approximation

In this chapter we return to the control problem, now with parametric approximation of the action-value function $\hat{q}(s, a, \mathbf{w}) \approx q_*(s, a)$, where $\mathbf{w} \in \mathbb{R}^d$ is a finite-dimensional weight vector. We continue to restrict attention to the on-policy case, leaving off-policy methods to Chapter 11. The present chapter features the semi-gradient Sarsa algorithm, the natural extension of semi-gradient TD(0) (last chapter) to action values and to on-policy control. In the episodic case, the extension is straightforward, but in the continuing case we have to take a few steps backward and re-examine how we have used discounting to define an optimal policy. Surprisingly, once we have genuine function approximation we have to give up discounting and switch to a new "average-reward" formulation of the control problem, with new "differential" value functions.

Starting first in the episodic case, we extend the function approximation ideas presented in the last chapter from state values to action values. Then we extend them to control following the general pattern of on-policy GPI, using ε-greedy for action selection. We show results for n-step linear Sarsa on the Mountain Car problem. Then we turn to the continuing case and repeat the development of these ideas for the average-reward case with differential values.

10.1 Episodic Semi-gradient Control

The extension of the semi-gradient prediction methods of Chapter 9 to action values is straightforward. In this case it is the approximate action-value function, $\hat{q} \approx q_\pi$, that is represented as a parameterized functional form with weight vector $\mathbf{w}$. Whereas before we considered random training examples of the form $S_t \mapsto U_t$, now we consider examples of the form $S_t, A_t \mapsto U_t$. The update target U_t can be any approximation of $q_\pi(S_t, A_t)$, including the usual backed-up values such as the full Monte Carlo return (G_t) or any of the n-step Sarsa returns (7.4). The general gradient-descent update for action-value

prediction is

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\Big[U_t - \hat{q}(S_t, A_t, \mathbf{w}_t)\Big]\nabla\hat{q}(S_t, A_t, \mathbf{w}_t). \tag{10.1}$$

For example, the update for the one-step Sarsa method is

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\Big[R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t)\Big]\nabla\hat{q}(S_t, A_t, \mathbf{w}_t). \tag{10.2}$$

We call this method *episodic semi-gradient one-step Sarsa*. For a constant policy, this method converges in the same way that TD(0) does, with the same kind of error bound (9.14).

To form control methods, we need to couple such action-value prediction methods with techniques for policy improvement and action selection. Suitable techniques applicable to continuous actions, or to actions from large discrete sets, are a topic of ongoing research with as yet no clear resolution. On the other hand, if the action set is discrete and not too large, then we can use the techniques already developed in previous chapters. That is, for each possible action a available in the next state S_{t+1}, we can compute $\hat{q}(S_{t+1}, a, \mathbf{w}_t)$ and then find the greedy action $A^*_{t+1} = \arg\max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t)$. Policy improvement is then done (in the on-policy case treated in this chapter) by changing the estimation policy to a soft approximation of the greedy policy such as the ε-greedy policy. Actions are selected according to this same policy. Pseudocode for the complete algorithm is given in the box.

Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    If S' is terminal:
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
      Go to next episode
    Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε-greedy)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
    $S \leftarrow S'$
    $A \leftarrow A'$

Example 10.1: Mountain Car Task Consider the task of driving an underpowered car up a steep mountain road, as suggested by the diagram in the upper left of Figure 10.1. The difficulty is that gravity is stronger than the car's engine, and even at full throttle the car cannot accelerate up the steep slope. The only solution is to first move away from the goal and up the opposite slope on the left. Then, by applying full throttle the car

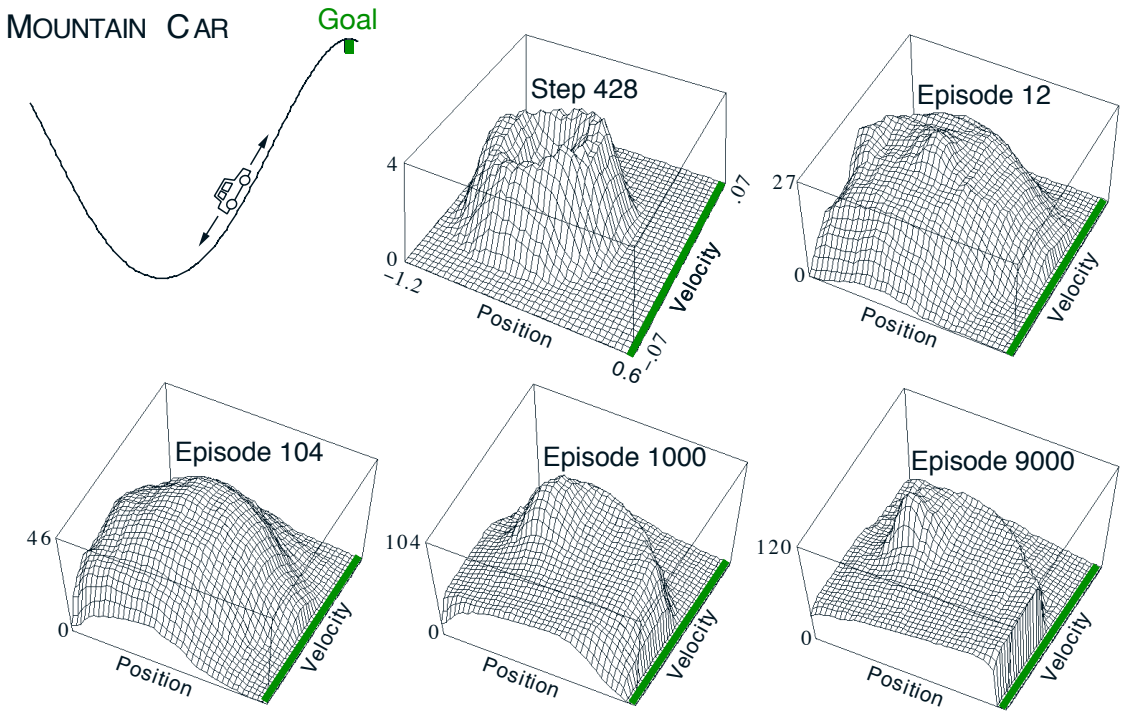

Figure 10.1: The Mountain Car task (upper left panel) and the cost-to-go function ($-\max_a \hat{q}(s, a, \mathbf{w})$) learned during one run.

can build up enough inertia to carry it up the steep slope even though it is slowing down the whole way. This is a simple example of a continuous control task where things have to get worse in a sense (farther from the goal) before they can get better. Many control methodologies have great difficulties with tasks of this kind unless explicitly aided by a human designer.

The reward in this problem is -1 on all time steps until the car moves past its goal position at the top of the mountain, which ends the episode. There are three possible actions: full throttle forward ($+1$), full throttle reverse (-1), and zero throttle (0). The car moves according to a simplified physics. Its position, x_t, and velocity, $\dot{x}_t$, are updated by

$$x_{t+1} \doteq \mathit{bound}\big[x_t + \dot{x}_{t+1}\big]$$

$$\dot{x}_{t+1} \doteq \mathit{bound}\big[\dot{x}_t + 0.001 A_t - 0.0025\cos(3x_t)\big],$$

where the *bound* operation enforces $-1.2 \leq x_{t+1} \leq 0.5$ and $-0.07 \leq \dot{x}_{t+1} \leq 0.07$. In addition, when x_{t+1} reached the left bound, $\dot{x}_{t+1}$ was reset to zero. When it reached the right bound, the goal was reached and the episode was terminated. Each episode started from a random position $x_t \in [-0.6, -0.4)$ and zero velocity. To convert the two continuous state variables to binary features, we used grid-tilings as in Figure 9.9. We used 8 tilings, with each tile covering 1/8th of the bounded distance in each dimension,

and asymmetrical offsets as described in Section 9.5.4.[1] The feature vectors $\mathbf{x}(s,a)$ created by tile coding were then combined linearly with the parameter vector to approximate the action-value function:

$$\hat{q}(s,a,\mathbf{w}) \doteq \mathbf{w}^\top \mathbf{x}(s,a) = \sum_{i=1}^{d} w_i \cdot x_i(s,a), \tag{10.3}$$

for each pair of state, s, and action, a.

Figure 10.1 shows what typically happens while learning to solve this task with this form of function approximation.[2] Shown is the negative of the value function (the *cost-to-go* function) learned on a single run. The initial action values were all zero, which was optimistic (all true values are negative in this task), causing extensive exploration to occur even though the exploration parameter, ε, was 0. This can be seen in the middle-top panel of the figure, labeled "Step 428". At this time not even one episode had been completed, but the car has oscillated back and forth in the valley, following circular trajectories in state space. All the states visited frequently are valued worse than unexplored states, because the actual rewards have been worse than what was (unrealistically) expected. This continually drives the agent away from wherever it has been, to explore new states, until a solution is found.

Figure 10.2 shows several learning curves for semi-gradient Sarsa on this problem, with various step sizes.

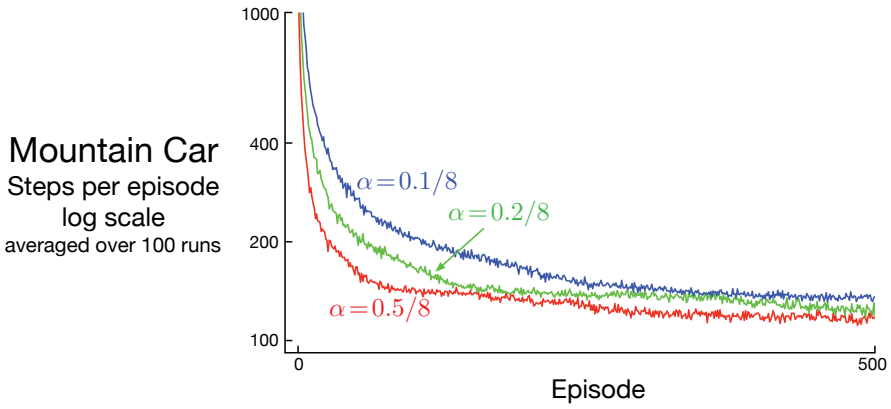

Figure 10.2: Mountain Car learning curves for the semi-gradient Sarsa method with tile-coding function approximation and ε-greedy action selection. ■

[1] In particular, we used the tile-coding software, available at `http://incompleteideas.net/tiles/tiles3.html`, with `iht=IHT(4096)` and `tiles(iht,8,[8*x/(0.5+1.2),8*xdot/(0.07+0.07)],[A])` to get the indices of the ones in the feature vector for state (`x`, `xdot`) and action `A`.

[2] This data is actually from the "semi-gradient Sarsa(λ)" algorithm that we will not meet until Chapter 12, but semi-gradient Sarsa would behave similarly.

10.2 Semi-gradient n-step Sarsa

We can obtain an n-step version of episodic semi-gradient Sarsa by using an n-step return as the update target in the semi-gradient Sarsa update equation (10.1). The n-step return immediately generalizes from its tabular form (7.4) to a function approximation form:

$$G_{t:t+n} \doteq R_{t+1}+\gamma R_{t+2}+\cdots+\gamma^{n-1}R_{t+n}+\gamma^n \hat{q}(S_{t+n}, A_{t+n}, \mathbf{w}_{t+n-1}), \quad t+n < T, \tag{10.4}$$

with $G_{t:t+n} \doteq G_t$ if $t+n \geq T$, as usual. The n-step update equation is

$$\mathbf{w}_{t+n} \doteq \mathbf{w}_{t+n-1} + \alpha \left[G_{t:t+n} - \hat{q}(S_t, A_t, \mathbf{w}_{t+n-1})\right] \nabla \hat{q}(S_t, A_t, \mathbf{w}_{t+n-1}), \qquad 0 \leq t < T. \tag{10.5}$$

Complete pseudocode is given in the box below.

Episodic semi-gradient n-step Sarsa for estimating $\hat{q} \approx q_*$ or q_π

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Input: a policy π (if estimating q_π)
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$, a positive integer n
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
All store and access operations (S_t, A_t, and R_t) can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim \pi(\cdot|S_0)$ or ε-greedy wrt $\hat{q}(S_0, \cdot, \mathbf{w})$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
    If $t < T$, then:
      Take action A_t
      Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
      If S_{t+1} is terminal, then:
        $T \leftarrow t+1$
      else:
        Select and store $A_{t+1} \sim \pi(\cdot|S_{t+1})$ or ε-greedy wrt $\hat{q}(S_{t+1}, \cdot, \mathbf{w})$
    $\tau \leftarrow t-n+1$ (τ is the time whose estimate is being updated)
    If $\tau \geq 0$:
      $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
      If $\tau+n < T$, then $G \leftarrow G + \gamma^n \hat{q}(S_{\tau+n}, A_{\tau+n}, \mathbf{w})$ ($G_{\tau:\tau+n}$)
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[G - \hat{q}(S_\tau, A_\tau, \mathbf{w})\right] \nabla \hat{q}(S_\tau, A_\tau, \mathbf{w})$
  Until $\tau = T-1$

As we have seen before, performance is best if an intermediate level of bootstrapping is used, corresponding to an n larger than 1. Figure 10.3 shows how this algorithm tends to learn faster and obtain a better asymptotic performance at $n=8$ than at $n=1$ on the Mountain Car task. Figure 10.4 shows the results of a more detailed study of the effect of the parameters α and n on the rate of learning on this task.

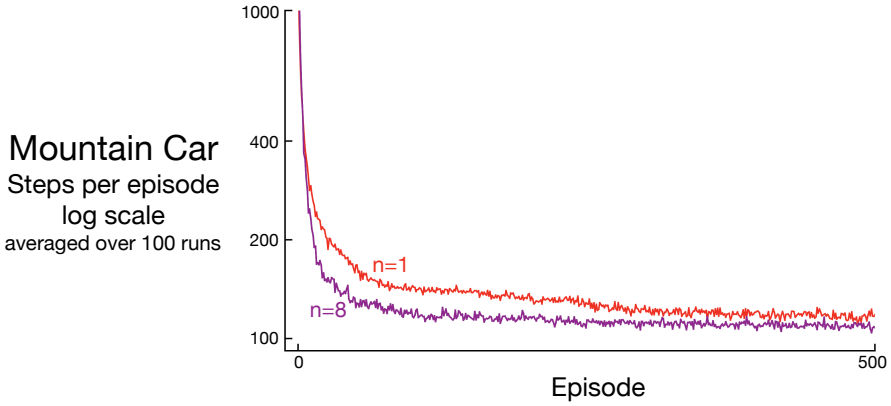

Figure 10.3: Performance of one-step vs 8-step semi-gradient Sarsa on the Mountain Car task. Good step sizes were used: $\alpha = 0.5/8$ for $n = 1$ and $\alpha = 0.3/8$ for $n = 8$.

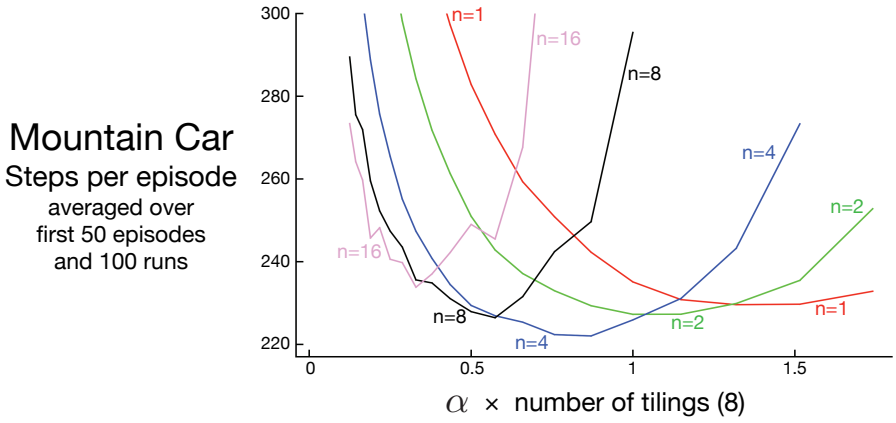

Figure 10.4: Effect of the α and n on early performance of n-step semi-gradient Sarsa and tile-coding function approximation on the Mountain Car task. As usual, an intermediate level of bootstrapping ($n = 4$) performed best. These results are for selected α values, on a log scale, and then connected by straight lines. The standard errors ranged from 0.5 (less than the line width) for $n = 1$ to about 4 for $n = 16$, so the main effects are all statistically significant.

Exercise 10.1 We have not explicitly considered or given pseudocode for any Monte Carlo methods in this chapter. What would they be like? Why is it reasonable not to give pseudocode for them? How would they perform on the Mountain Car task? □

Exercise 10.2 Give pseudocode for semi-gradient one-step *Expected* Sarsa for control. □

Exercise 10.3 Why do the results shown in Figure 10.4 have higher standard errors at large n than at small n? □

10.3 Average Reward: A New Problem Setting for Continuing Tasks

We now introduce a third classical setting—alongside the episodic and discounted settings—for formulating the goal in Markov decision problems (MDPs). Like the discounted setting, the *average reward* setting applies to continuing problems, problems for which the interaction between agent and environment goes on and on forever without termination or start states. Unlike that setting, however, there is no discounting—the agent cares just as much about delayed rewards as it does about immediate reward. The average-reward setting is one of the major settings commonly considered in the classical theory of dynamic programming and less-commonly in reinforcement learning. As we discuss in the next section, the discounted setting is problematic with function approximation, and thus the average-reward setting is needed to replace it.

In the average-reward setting, the quality of a policy π is defined as the average rate of reward, or simply *average reward*, while following that policy, which we denote as $r(\pi)$:

$$\begin{aligned} r(\pi) &\doteq \lim_{h\to\infty} \frac{1}{h} \sum_{t=1}^{h} \mathbb{E}[R_t \mid S_0, A_{0:t-1} \sim \pi] && (10.6)\\ &= \lim_{t\to\infty} \mathbb{E}[R_t \mid S_0, A_{0:t-1} \sim \pi], && (10.7)\\ &= \sum_s \mu_\pi(s) \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) r, \end{aligned}$$

where the expectations are conditioned on the initial state, S_0, and on the subsequent actions, $A_0, A_1, \ldots, A_{t-1}$, being taken according to π. The second and third equations hold if the steady-state distribution, $\mu_\pi(s) \doteq \lim_{t\to\infty} \Pr\{S_t = s \mid A_{0:t-1} \sim \pi\}$, exists and is independent of S_0, in other words, if the MDP is *ergodic*. In an ergodic MDP, the starting state and any early decision made by the agent can have only a temporary effect; in the long run the expectation of being in a state depends only on the policy and the MDP transition probabilities. Ergodicity is sufficient but not necessary to guarantee the existence of the limit in (10.6).

There are subtle distinctions that can be drawn between different kinds of optimality in the undiscounted continuing case. Nevertheless, for most practical purposes it may be adequate simply to order policies according to their average reward per time step, in other words, according to their $r(\pi)$. This quantity is essentially the average reward under π, as suggested by (10.7), or the *reward rate*. In particular, we consider all policies that attain the maximal value of $r(\pi)$ to be optimal.

Note that the steady state distribution μ_π is the special distribution under which, if you select actions according to π, you remain in the same distribution. That is, for which

$$\sum_s \mu_\pi(s) \sum_a \pi(a|s) p(s' \,|\, s, a) = \mu_\pi(s'). \tag{10.8}$$

In the average-reward setting, returns are defined in terms of differences between rewards and the average reward:

$$G_t \doteq R_{t+1} - r(\pi) + R_{t+2} - r(\pi) + R_{t+3} - r(\pi) + \cdots. \tag{10.9}$$

This is known as the *differential* return, and the corresponding value functions are known as *differential* value functions. Differential value functions are defined in terms of the new return just as conventional value functions were defined in terms of the discounted return; thus we will use the same notation, $v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s]$ and $q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$ (similarly for v_* and q_*), for differential value functions. Differential value functions also have Bellman equations, just slightly different from those we have seen earlier. We simply remove all γs and replace all rewards by the difference between the reward and the true average reward:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{r,s'} p(s', r\,|\,s, a)\Big[r - r(\pi) + v_\pi(s')\Big],$$

$$q_\pi(s,a) = \sum_{r,s'} p(s', r\,|\,s, a)\Big[r - r(\pi) + \sum_{a'} \pi(a'|s') q_\pi(s', a')\Big],$$

$$v_*(s) = \max_a \sum_{r,s'} p(s', r\,|\,s, a)\Big[r - \max_\pi r(\pi) + v_*(s')\Big], \text{ and}$$

$$q_*(s,a) = \sum_{r,s'} p(s', r\,|\,s, a)\Big[r - \max_\pi r(\pi) + \max_{a'} q_*(s', a')\Big]$$

(cf. (3.14), Exercise 3.17, (3.19), and (3.20)).

There is also a differential form of the two TD errors:

$$\delta_t \doteq R_{t+1} - \bar{R}_t + \hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t), \tag{10.10}$$

and

$$\delta_t \doteq R_{t+1} - \bar{R}_t + \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t), \tag{10.11}$$

where $\bar{R}_t$ is an estimate at time t of the average reward $r(\pi)$. With these alternate definitions, most of our algorithms and many theoretical results carry through to the average-reward setting without change.

For example, an average reward version of semi-gradient Sarsa could be defined just as in (10.2) except with the differential version of the TD error. That is, by

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \delta_t \nabla \hat{q}(S_t, A_t, \mathbf{w}_t), \tag{10.12}$$

with δ_t given by (10.11). Pseudocode for a complete algorithm is given in the box on the next page. One limitation of this algorithm is that it does not converge to the differential values but to the differential values plus an arbitrary offset. Notice that the Bellman equations and TD errors given above are unaffected if all the values are shifted by the same amount. Thus, the offset may not matter in practice. How this algorithm could be changed to eliminate the offset is an interesting question for future research.

Exercise 10.4 Give pseudocode for a differential version of semi-gradient Q-learning. □

Exercise 10.5 What equations are needed (beyond 10.10) to specify the differential version of TD(0)? □

Differential semi-gradient Sarsa for estimating $\hat{q} \approx q_*$

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameters: step sizes $\alpha, \beta > 0$, small $\varepsilon > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Initialize average reward estimate $\bar{R} \in \mathbb{R}$ arbitrarily (e.g., $\bar{R} = 0$)

Initialize state S, and action A
Loop for each step:
  Take action A, observe R, S'
  Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε-greedy)
  $\delta \leftarrow R - \bar{R} + \hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})$
  $\bar{R} \leftarrow \bar{R} + \beta\delta$
  $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\nabla\hat{q}(S, A, \mathbf{w})$
  $S \leftarrow S'$
  $A \leftarrow A'$

Exercise 10.6 Suppose there is an MDP that under any policy produces the deterministic sequence of rewards $+1, 0, +1, 0, +1, 0, \ldots$ going on forever. Technically, this violates ergodicity; there is no stationary limiting distribution μ_π and the limit (10.7) does not exist. Nevertheless, the average reward (10.6) is well defined. What is it? Now consider two states in this MDP. From A, the reward sequence is exactly as described above, starting with a $+1$, whereas, from B, the reward sequence starts with a 0 and then continues with $+1, 0, +1, 0, \ldots$. We would like to compute the differential values of A and B. Unfortunately, the differential return (10.9) is not well defined when starting from these states as the implicit limit does not exist. To repair this, one could alternatively define the differential value of a state as

$$v_\pi(s) \doteq \lim_{\gamma \to 1} \lim_{h \to \infty} \sum_{t=0}^{h} \gamma^t \Big(\mathbb{E}_\pi[R_{t+1} | S_0 = s] - r(\pi) \Big). \tag{10.13}$$

Under this definition, what are the differential values of states A and B? □

Exercise 10.7 Consider a Markov reward process consisting of a ring of three states A, B, and C, with state transitions going deterministically around the ring. A reward of $+1$ is received upon arrival in A and otherwise the reward is 0. What are the differential values of the three states, using (10.13)? □

Exercise 10.8 The pseudocode in the box on page 251 updates $\bar{R}_t$ using δ_t as an error rather than simply $R_{t+1} - \bar{R}_t$. Both errors work, but using δ_t is better. To see why, consider the ring MRP of three states from Exercise 10.7. The estimate of the average reward should tend towards its true value of $\frac{1}{3}$. Suppose it was already there and was held stuck there. What would the sequence of $R_{t+1} - \bar{R}_t$ errors be? What would the sequence of δ_t errors be (using Equation 10.10)? Which error sequence would produce a more stable estimate of the average reward if the estimate were allowed to change in response to the errors? Why? □

Example 10.2: An Access-Control Queuing Task This is a decision task involving access control to a set of 10 servers. Customers of four different priorities arrive at a single queue. If given access to a server, the customers pay a reward of 1, 2, 4, or 8 to the server, depending on their priority, with higher priority customers paying more. In each time step, the customer at the head of the queue is either accepted (assigned to one of the servers) or rejected (removed from the queue, with a reward of zero). In either case, on the next time step the next customer in the queue is considered. The queue never empties, and the priorities of the customers in the queue are uniformly randomly distributed. Of course a customer cannot be served if there is no free server; the customer is always rejected in this case. Each busy server becomes free with probability $p = 0.06$ on each time step. Although we have just described them for definiteness, let us assume the statistics of arrivals and departures are unknown. The task is to decide on each step whether to accept or reject the next customer, on the basis of his priority and the number of free servers, so as to maximize long-term reward without discounting.

In this example we consider a tabular solution to this problem. Although there is no generalization between states, we can still consider it in the general function approximation setting as this setting generalizes the tabular setting. Thus we have a differential action-value estimate for each pair of state (number of free servers and priority of the customer at the head of the queue) and action (accept or reject). Figure 10.5 shows the solution found by differential semi-gradient Sarsa with parameters $\alpha = 0.01$, $\beta = 0.01$, and $\varepsilon = 0.1$. The initial action values and $\bar{R}$ were zero.

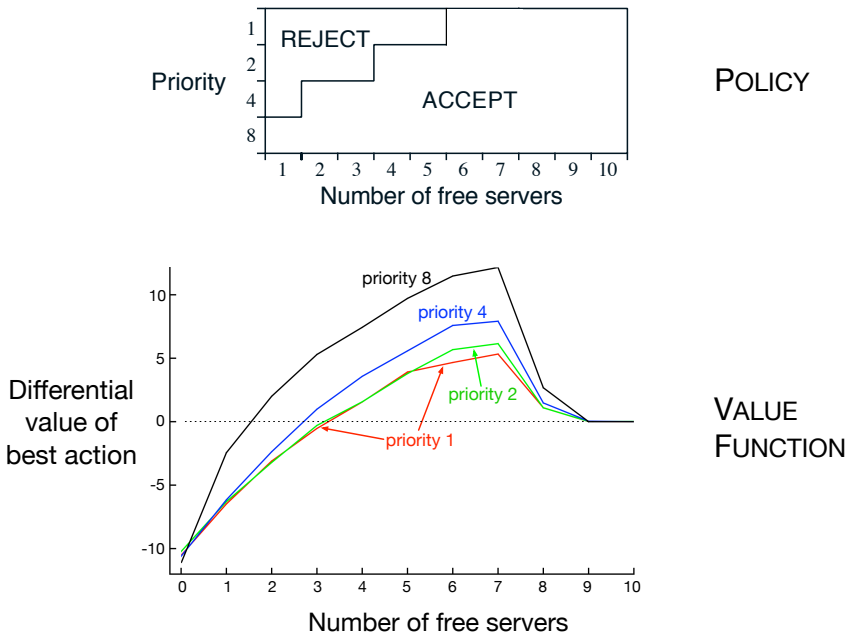

Figure 10.5: The policy and value function found by differential semi-gradient one-step Sarsa on the access-control queuing task after 2 million steps. The drop on the right of the graph is probably due to insufficient data; many of these states were never experienced. The value learned for $\bar{R}$ was about 2.31. (Note that priority 1 here is the lowest priority.) ■

10.4 Deprecating the Discounted Setting

The continuing, discounted problem formulation has been very useful in the tabular case, in which the returns from each state can be separately identified and averaged. But in the approximate case it is questionable whether one should ever use this problem formulation.

To see why, consider an infinite sequence of returns with no beginning or end, and no clearly identified states. The states might be represented only by feature vectors, which may do little to distinguish the states from each other. As a special case, all of the feature vectors may be the same. Thus one really has only the reward sequence (and the actions), and performance has to be assessed purely from these. How could it be done? One way is by averaging the rewards over a long interval—this is the idea of the average-reward setting. How could discounting be used? Well, for each time step we could measure the discounted return. Some returns would be small and some big, so again we would have to average them over a sufficiently large time interval. In the continuing setting there are no starts and ends, and no special time steps, so there is nothing else that could be done. However, if you do this, it turns out that the average of the discounted returns is proportional to the average reward. In fact, for policy π, the average of the discounted returns is always $r(\pi)/(1-\gamma)$, that is, it is essentially the average reward, $r(\pi)$. In particular, the *ordering* of all policies in the average discounted return setting would be exactly the same as in the average-reward setting. The discount rate γ thus has no effect on the problem formulation. It could in fact be *zero* and the ranking would be unchanged.

This surprising fact is proven in the box on the next page, but the basic idea can be seen via a symmetry argument. Each time step is exactly the same as every other. With discounting, every reward will appear exactly once in each position in some return. The tth reward will appear undiscounted in the $t-1$st return, discounted once in the $t-2$nd return, and discounted 999 times in the $t-1000$th return. The weight on the tth reward is thus $1+\gamma+\gamma^2+\gamma^3+\cdots = 1/(1-\gamma)$. Because all states are the same, they are all weighted by this, and thus the average of the returns will be this times the average reward, or $r(\pi)/(1-\gamma)$.

This example and the more general argument in the box show that if we optimized discounted value over the on-policy distribution, then the effect would be identical to optimizing *undiscounted* average reward; the actual value of γ would have no effect. This strongly suggests that discounting has no role to play in the definition of the control problem with function approximation. One can nevertheless go ahead and use discounting in solution methods. The discounting parameter γ changes from a problem parameter to a solution method parameter! Unfortunately, discounting algorithms with function approximation do not optimize discounted value over the on-policy distribution, and thus are not guaranteed to optimize average reward.

The root cause of the difficulties with the discounted control setting is that with function approximation we have lost the policy improvement theorem (Section 4.2). It is no longer true that if we change the policy to improve the discounted value of one state then we are guaranteed to have improved the overall policy in any useful sense. That guarantee was key to the theory of our reinforcement learning control methods. With

The Futility of Discounting in Continuing Problems

Perhaps discounting can be saved by choosing an objective that sums discounted values over the distribution with which states occur under the policy:

$$
\begin{aligned}
J(\pi) &= \sum_s \mu_\pi(s) v_\pi^\gamma(s) && \text{(where } v_\pi^\gamma \text{ is the discounted value function)} \\
&= \sum_s \mu_\pi(s) \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \,|\, s, a) \left[r + \gamma v_\pi^\gamma(s') \right] && \text{(Bellman Eq.)} \\
&= r(\pi) + \sum_s \mu_\pi(s) \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \,|\, s, a) \gamma v_\pi^\gamma(s') && \text{(from (10.7))} \\
&= r(\pi) + \gamma \sum_{s'} v_\pi^\gamma(s') \sum_s \mu_\pi(s) \sum_a \pi(a|s) p(s' \,|\, s, a) && \text{(from (3.4))} \\
&= r(\pi) + \gamma \sum_{s'} v_\pi^\gamma(s') \mu_\pi(s') && \text{(from (10.8))} \\
&= r(\pi) + \gamma J(\pi) \\
&= r(\pi) + \gamma r(\pi) + \gamma^2 J(\pi) \\
&= r(\pi) + \gamma r(\pi) + \gamma^2 r(\pi) + \gamma^3 r(\pi) + \cdots \\
&= \frac{1}{1-\gamma} r(\pi).
\end{aligned}
$$

The proposed discounted objective orders policies identically to the undiscounted (average reward) objective. The discount rate γ does not influence the ordering!

function approximation we have lost it!

In fact, the lack of a policy improvement theorem is also a theoretical lacuna for the total-episodic and average-reward settings. Once we introduce function approximation we can no longer guarantee improvement for any setting. In Chapter 13 we introduce an alternative class of reinforcement learning algorithms based on parameterized policies, and there we have a theoretical guarantee called the "policy-gradient theorem" which plays a similar role as the policy improvement theorem. But for methods that learn action values we seem to be currently without a local improvement guarantee (possibly the approach taken by Perkins and Precup (2003) may provide a part of the answer). We do know that ε-greedification may sometimes result in an inferior policy, as policies may chatter among good policies rather than converge (Gordon, 1996a). This is an area with multiple open theoretical questions.

10.5 Differential Semi-gradient n-step Sarsa

In order to generalize to *n-step* bootstrapping, we need an n-step version of the TD error. We begin by generalizing the n-step return (7.4) to its differential form, with function approximation:

$$G_{t:t+n} \doteq R_{t+1} - \bar{R}_{t+n-1} + \cdots + R_{t+n} - \bar{R}_{t+n-1} + \hat{q}(S_{t+n}, A_{t+n}, \mathbf{w}_{t+n-1}), \quad (10.14)$$

where $\bar{R}$ is an estimate of $r(\pi)$, $n \geq 1$, and $t + n < T$. If $t + n \geq T$, then we define $G_{t:t+n} \doteq G_t$ as usual. The n-step TD error is then

$$\delta_t \doteq G_{t:t+n} - \hat{q}(S_t, A_t, \mathbf{w}), \quad (10.15)$$

after which we can apply our usual semi-gradient Sarsa update (10.12). Pseudocode for the complete algorithm is given in the box.

Differential semi-gradient n-step Sarsa for estimating $\hat{q} \approx q_\pi$ or q_*

Input: a differentiable function $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$, a policy π
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Initialize average-reward estimate $\bar{R} \in \mathbb{R}$ arbitrarily (e.g., $\bar{R} = 0$)
Algorithm parameters: step sizes $\alpha, \beta > 0$, small $\varepsilon > 0$, a positive integer n
All store and access operations (S_t, A_t, and R_t) can take their index mod $n + 1$

Initialize and store S_0 and A_0
Loop for each step, $t = 0, 1, 2, \ldots$:
  Take action A_t
  Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$, or ε-greedy wrt $\hat{q}(S_{t+1}, \cdot, \mathbf{w})$
  $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
  If $\tau \geq 0$:
    $\delta \leftarrow \sum_{i=\tau+1}^{\tau+n} (R_i - \bar{R}) + \hat{q}(S_{\tau+n}, A_{\tau+n}, \mathbf{w}) - \hat{q}(S_\tau, A_\tau, \mathbf{w})$
    $\bar{R} \leftarrow \bar{R} + \beta\delta$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\nabla\hat{q}(S_\tau, A_\tau, \mathbf{w})$

Exercise 10.9 In the differential semi-gradient n-step Sarsa algorithm, the step-size parameter on the average reward, β, needs to be quite small so that $\bar{R}$ becomes a good long-term estimate of the average reward. Unfortunately, $\bar{R}$ will then be biased by its initial value for many steps, which may make learning inefficient. Alternatively, one could use a sample average of the observed rewards for $\bar{R}$. That would initially adapt rapidly but in the long run would also adapt slowly. As the policy slowly changed, $\bar{R}$ would also change; the potential for such long-term nonstationarity makes sample-average methods ill-suited. In fact, the step-size parameter on the average reward is a perfect place to use the unbiased constant-step-size trick from Exercise 2.7. Describe the specific changes needed to the boxed algorithm for differential semi-gradient n-step Sarsa to use this trick. □

10.6 Summary

In this chapter we have extended the ideas of parameterized function approximation and semi-gradient descent, introduced in the previous chapter, to control. The extension is immediate for the episodic case, but for the continuing case we have to introduce a whole new problem formulation based on maximizing the *average reward setting* per time step. Surprisingly, the discounted formulation cannot be carried over to control in the presence of approximations. In the approximate case most policies cannot be represented by a value function. The arbitrary policies that remain need to be ranked, and the scalar average reward $r(\pi)$ provides an effective way to do this.

The average reward formulation involves new *differential* versions of value functions, Bellman equations, and TD errors, but all of these parallel the old ones, and the conceptual changes are small. There is also a new parallel set of differential algorithms for the average-reward case.

Bibliographical and Historical Remarks

10.1 Semi-gradient Sarsa with function approximation was first explored by Rummery and Niranjan (1994). Linear semi-gradient Sarsa with ε-greedy action selection does not converge in the usual sense, but does enter a bounded region near the best solution (Gordon, 1996a, 2001). Precup and Perkins (2003) showed convergence in a differentiable action selection setting. See also Perkins and Pendrith (2002) and Melo, Meyn, and Ribeiro (2008). The mountain–car example is based on a similar task studied by Moore (1990), but the exact form used here is from Sutton (1996).

10.2 Episodic n-step semi-gradient Sarsa is based on the forward Sarsa(λ) algorithm of van Seijen (2016). The empirical results shown here are new to the second edition of this text.

10.3 The average-reward formulation has been described for dynamic programming (e.g., Puterman, 1994) and from the point of view of reinforcement learning (Mahadevan, 1996; Tadepalli and Ok, 1994; Bertsekas and Tsitsiklis, 1996; Tsitsiklis and Van Roy, 1999). The algorithm described here is the on-policy analog of the "R-learning" algorithm introduced by Schwartz (1993). The name R-learning was probably meant to be the alphabetic successor to Q-learning, but we prefer to think of it as a reference to the learning of differential or *relative* values. The access-control queuing example was suggested by the work of Carlström and Nordström (1997).

10.4 The recognition of the limitations of discounting as a formulation of the reinforcement learning problem with function approximation became apparent to the authors shortly after the publication of the first edition of this text. Singh, Jaakkola, and Jordan (1994) may have been the first to observe it in print.