3.3 Returns and Episodes

So far we have discussed the objective of learning informally. We have said that the agent's goal is to maximize the cumulative reward it receives in the long run. How might this be defined formallfy? If the sequence of rewards received after time step t is denoted Rt+1, Rt+2, Rt+3, . . ., then what precise aspect of this sequence do we wish to maximize? In general, we seek to maximize the expected return, where the return, denoted Gt, is defined as some specific function of the reward sequence. In the simplest case the return is the sum of the rewards: Gt .= Rt+1 + Rt+2 + Rt+3 + · · · + RT , (3.7) where T is a final time step. This approach makes sense in applications in which there is a natural notion of final time step, that is, when the agent–environment interaction breaks naturally into subsequences, which we call episodes,7 such as plays of a game, trips through a maze, or any sort of repeated interaction. Each episode ends in a special state called the terminal state, followed by a reset to a standard starting state or to a sample from a standard distribution of starting states. Even if you think of episodes as ending in different ways, such as winning and losing a game, the next episode begins independently of how the previous one ended. Thus the episodes can all be considered to end in the same terminal state, with different rewards for the different outcomes. Tasks with episodes of this kind are called episodic tasks. In episodic tasks we sometimes need to distinguish the set of all nonterminal states, denoted S, from the set of all states plus the terminal state, denoted S+. The time of termination, T, is a random variable that normally varies from episode to episode. On the other hand, in many cases the agent–environment interaction does not break naturally into identifiable episodes, but goes on continually without limit. For example, this would be the natural way to formulate an on-going process-control task, or an application to a robot with a long life span. We call these continuing tasks. The return formulation (3.7) is problematic for continuing tasks because the final time step would be T = ∞, and the return, which is what we are trying to maximize, could itself easily 5Better places for imparting this kind of prior knowledge are the initial policy or initial value function, or in influences on these. 6Section 17.4 delves further into the issue of designing effective reward signals. 7Episodes are sometimes called "trials" in the literature. be infinite. (For example, suppose the agent receives a reward of +1 at each time step.) Thus, in this book we usually use a definition of return that is slightly more complex conceptually but much simpler mathematically. The additional concept that we need is that of discounting. According to this approach, the agent tries to select actions so that the sum of the discounted rewards it receives over the future is maximized. In particular, it chooses At to maximize the expected discounted return: Gt .= Rt+1 + γRt+2 + γ2Rt+3 + · · · = ∞ X k=0 γkRt+k+1, (3.8) where γ is a parameter, 0 ≤γ ≤1, called the discount rate. The discount rate determines the present value of future rewards: a reward received k time steps in the future is worth only γk−1 times what it would be worth if it were received immediately. If γ < 1, the infinite sum in (3.8) has a finite value as long as the reward sequence {Rk} is bounded. If γ = 0, the agent is "myopic" in being concerned only with maximizing immediate rewards: its objective in this case is to learn how to choose At so as to maximize only Rt+1. If each of the agent's actions happened to influence only the immediate reward, not future rewards as well, then a myopic agent could maximize (3.8) by separately maximizing each immediate reward. But in general, acting to maximize immediate reward can reduce access to future rewards so that the return is reduced. As γ approaches 1, the return objective takes future rewards into account more strongly; the agent becomes more farsighted. Returns at successive time steps are related to each other in a way that is important for the theory and algorithms of reinforcement learning: Gt .= Rt+1 + γRt+2 + γ2Rt+3 + γ3Rt+4 + · · · = Rt+1 + γ Rt+2 + γRt+3 + γ2Rt+4 + · · · = Rt+1 + γGt+1 (3.9) Note that this works for all time steps t < T, even if termination occurs at t + 1, if we define GT = 0. This often makes it easy to compute returns from reward sequences. Note that although the return (3.8) is a sum of an infinite number of terms, it is still finite if the reward is

nonzero and constant—if γ < 1. For example, if the reward is a constant +1, then the return is Gt = ∞ X k=0 γk = 1 1 −γ . (3.10) Exercise 3.5 The equations in Section 3.1 are for the continuing case and need to be modified (very slightly) to apply to episodic tasks. Show that you know the modifications needed by giving the modified version of (3.3). □ Example 3.4: Pole-Balancing The objective in this task is to apply forces to a cart moving along a track so as to keep a pole hinged to the cart from falling over: A failure is said to occur if the pole falls past a given an- gle from vertical or if the cart runs off the track. The pole is reset to vertical after each failure. This task could be treated as episodic, where the natural episodes are the repeated attempts to balance the pole. The reward in this case could be +1 for every time step on which failure did not occur, so that the return at each time would be the number of steps until failure. In this case, successful balancing forever would mean a return of infinity. Alternatively, we could treat pole-balancing as a con- tinuing task, using discounting. In this case the reward would be −1 on each failure and zero at all other times. The return at each time would then be related to −γK, where K is the number of time steps before failure. In either case, the return is maximized by keeping the pole balanced for as long as possible. Exercise 3.6 Suppose you treated pole-balancing as an episodic task but also used dis- counting, with all rewards zero except for −1 upon failure. What then would the return be at each time? How does this return differ from that in the discounted, continuing formulation of this task? □ Exercise 3.7 Imagine that you are designing a robot to run a maze. You decide to give it a reward of +1 for escaping from the maze and a reward of zero at all other times. The task seems to break down naturally into episodes—the successive runs through the maze—so you decide to treat it as an episodic task, where the goal is to maximize expected total reward (3.7). After running the learning agent for a while, you find that it is showing no improvement in escaping from the maze. What is going wrong? Have you effectively communicated to the agent what you want it to achieve? □ Exercise 3.8 Suppose γ = 0.5 and the following sequence of rewards is received R1 = −1, R2 = 2, R3 = 6, R4 = 3, and R5 = 2, with T = 5. What are G0, G1, . . ., G5? Hint: Work backwards. □ Exercise 3.9 Suppose γ = 0.9 and the reward sequence is R1 = 2 followed by an infinite sequence of 7s. What are G1 and G0? □ Exercise 3.10 Prove (3.10). □

3.4 Unified Notation for Episodic and Continuing Tasks

In the preceding section we described two kinds of reinforcement learning tasks, one in which the agent–environment interaction naturally breaks down into a sequence of separate episodes (episodic tasks), and one in which it does not (continuing tasks). The former case is mathematically easier because each action affects only the finite number of rewards subsequently received during the episode. In this book we consider sometimes one kind of problem and sometimes the other, but often both. It is therefore useful to establish one notation that enables us to talk precisely about both cases simultaneously. To be precise about episodic tasks requires some additional notation. Rather than one long sequence of time steps, we need to consider a series of episodes, each of which consists of a finite sequence of time steps. We number the time steps of each episode starting anew from zero. Therefore, we have to refer not just to St, the state representation at time t, but to St,i, the state representation at time t of episode i (and similarly for At,i, Rt,i, πt,i, Ti, etc.). However, it turns out that when we discuss episodic tasks we almost never have to distinguish between different episodes. We are almost always considering a particular single episode, or stating something that is true for all episodes. Accordingly, in practice we almost always abuse notation slightly by dropping the explicit reference to episode number. That is, we write St to refer to St,i, and so on. We need one other convention to obtain a single notation that covers both episodic and continuing tasks. We have defined the return as a sum over a finite number of terms in one case (3.7) and as a sum over an infinite number of terms in the other (3.8). These can be unified by considering episode

termination to be the entering of a special absorbing state that transitions only to itself and that generates only rewards of zero. For example, consider the state transition diagram: R1 = +1 S0 S1 R2 = +1 S2 R3 = +1 R4 = 0 R5 = 0 . . . Here the solid square represents the special absorbing state corresponding to the end of an episode. Starting from S0, we get the reward sequence +1, +1, +1, 0, 0, 0, ... Summing these, we get the same return whether we sum over the first T rewards (here T = 3) or over the full infinite sequence. This remains true even if we introduce discounting. Thus, we can define the return, in general, according to (3.8), using the convention of omitting episode numbers when they are not needed, and including the possibility that γ = 1 if the sum remains defined (e.g., because all episodes terminate). Alternatively, we can also write the return as Gt .= T X k=t+1 γk−t−1Rk, (3.11) including the possibility that T = ∞or γ = 1 (but not both). We use these conventions throughout the rest of the book to simplify notation and to express the close paral- lels between episodic and continuing tasks. (Later, in Chapter 10, we will introduce a formulation that is both continuing and undiscounted.)

3.5 Policies and Value Functions

Almost all reinforcement learning algorithms involve estimating value functions—functions of states (or of state–action pairs) that estimate how good it is for the agent to be in a given state (or how good it is to perform a given action in a given state). The notion of "how good" here is defined in terms of future rewards that can be expected, or, to be precise, in terms of expected return. Of course the rewards the agent can expect to receive in the future depend on what actions it will take. Accordingly, value functions are defined with respect to particular ways of acting, called policies. Formally, a policy is a mapping from states to probabilities of selecting each possible action. If the agent is following policy π at time t, then π(a|s) is the probability that At = a if St = s. Like p, π is an ordinary function; the "|" in the middle of π(a|s) merely reminds that it defines a probability distribution over a ∈A(s) for each s ∈S. Reinforcement learning methods specify how the agent's policy is changed as a result of its experience. Exercise 3.11 If the current state is St, and actions are selected according to stochastic policy π, then what is the expectation of Rt+1 in terms of π and the four-argument function p (3.2)? □ The value function of a state s under a policy π, denoted vπ(s), is the expected return when starting in s and following π thereafter. For MDPs, we can define vπ formally by vπ(s) .= Eπ[Gt | St =s] = Eπ " ∞ X k=0 γkRt+k+1 St =s # , for all s ∈S, (3.12) where Eπ[·] denotes the expected value of a random variable given that the agent follows policy π, and t is any time step. Note that the value of the terminal state, if any, is always zero. We call the function vπ the state-value function for policy π. Similarly, we define the value of taking action a in state s under a policy π, denoted qπ(s, a), as the expected return starting from s, taking the action a, and thereafter following policy π: qπ(s, a) .= Eπ[Gt | St =s, At = a] = Eπ " ∞ X k=0 γkRt+k+1 St =s, At =a # . (3.13) We call qπ the action-value function for policy π. The value functions vπ and qπ can be estimated from experience. For example, if an agent follows policy π and maintains an average, for each state encountered, of the actual returns that have followed that state, then the average will converge to the state's value, vπ(s), as the number of times that state is encountered approaches infinity. If separate averages are kept for each action taken in each state, then these averages will similarly converge to the action values, qπ(s, a). We call estimation methods of this kind Monte Carlo methods because they involve averaging over many random samples of 3.5. Unified Notation for Episodic and Continuing Tasks 59 actual returns. These kinds of methods are presented in Chapter 5. Of course, if there are very many states, then it may not be practical to keep separate averages for each state individually. Instead, the agent would have to maintain vπ and qπ as parameter- ized functions (with fewer parameters than states) and adjust the parameters to better match the observed returns. This can also produce accurate estimates, although much depends on the nature of the parameterized function approximator. These possibilities are

discussed in Part II of the book. A fundamental property of value functions used throughout reinforcement learning and dynamic programming is that they satisfy recursive relationships similar to that which we have already established for the return (3.9). For any policy π and any state s, the following consistency condition holds between the value of s and the value of its possible successor states: vπ(s) .= Eπ[Gt | St =s] = Eπ[Rt+1 + γGt+1 | St =s] (by (3.9)) = X a π(a|s) X s′ X r p(s′, r|s, a) h r + γEπ[Gt+1|St+1 =s′] i = X a π(a|s) X s′,r p(s′, r|s, a) h r + γvπ(s′) i , for all s ∈S, (3.14) where it is implicit that the actions, a, are taken from the set A(s), that the next states, s′, are taken from the set S (or from S+ in the case of an episodic problem), and that the rewards, r, are taken from the set R. Note also how in the last equation we have merged the two sums, one over all the values of s′ and the other over all the values of r, into one sum over all the possible values of both. We use this kind of merged sum often to simplify formulas. Note how the final expression can be read easily as an expected value. It is really a sum over all values of the three variables, a, s′, and r. For each triple, we compute its probability, π(a|s)p(s′, r|s, a), weight the quantity in brackets by that probability, then sum over all possibilities to get an expected value. ↑ s s0 ↑ r p a Backup diagram for vπ Equation (3.14) is the Bellman equation for vπ. It expresses a relationship between the value of a state and the values of its successor states. Think of looking ahead from a state to its possible successor states, as suggested by the diagram to the right. Each open circle represents a state and each solid circle represents a state–action pair. Starting from state s, the root node at the top, the agent could take any of some set of actions— three are shown in the diagram—based on its policy π. From each of these, the environment could respond with one of several next states, s′ (two are shown in the figure), along with a reward, r, depending on its dynamics given by the function p. The Bellman equation (3.14) averages over all the possibilities, weighting each by its probability of occurring. It states that the value of the start state must equal the (discounted) value of the expected next state, plus the reward expected along the way. The value function vπ is the unique solution to its Bellman equation. We show in subsequent chapters how this Bellman equation forms the basis of a number of ways to compute, approximate, and learn vπ. We call diagrams like that above backup diagrams because they diagram relationships that form the basis of the update or backup operations that are at the heart of reinforcement learning methods. These operations transfer value information back to a state (or a state–action pair) from its successor states (or state–action pairs). We use backup diagrams throughout the book to provide graphical summaries of the algorithms we discuss. (Note that, unlike transition graphs, the state nodes of backup diagrams do not necessarily represent distinct states; for example, a state might be its own successor.) Example 3.5: Gridworld Figure 3.2 (left) shows a rectangular gridworld represen- tation of a simple finite MDP. The cells of the grid correspond to the states of the environment. At each cell, four actions are possible: north, south, east, and west, which deterministically cause the agent to move one cell in the respective direction on the grid. Actions that would take the agent offthe grid leave its location unchanged, but also result in a reward of −1. Other actions result in a reward of 0, except those that move the agent out of the special states A and B. From state A, all four actions yield a reward of +10 and take the agent to A′. From state B, all actions yield a reward of +5 and take the agent to B′. 3.3 8.8 4.4 5.3 1.5 1.5 3.0 2.3 1.9 0.5 0.1 0.7 0.7 0.4 -0.4 -1.0 -0.4 -0.4 -0.6 -1.2 -1.9 -1.3 -1.2 -1.4 -2.0 A B A' B' +10 +5 Actions Figure 3.2: Gridworld example: exceptional reward dynamics (left) and state-value function for the equiprobable random policy (right). Suppose the agent selects all four actions with equal probability in all states. Figure 3.2 (right) shows the value function, vπ, for this policy, for the discounted reward case with γ = 0.9. This value function was computed by solving the system of linear equations (3.14). Notice the negative values near the lower edge; these are the result of the high probability of hitting the edge of the grid there under the random policy. State A is the best state to be in under this policy, but its expected return is less than 10, its immediate reward, because from A the agent is taken to A′, from which it is likely to run into the edge of the

grid. State B, on the other hand, is valued more than 5, its immediate reward, because from B the agent is taken to B′, which has a positive value. From B′ the expected penalty (negative reward) for possibly running into an edge is more than compensated for by the expected gain for possibly stumbling onto A or B. Exercise 3.12 The Bellman equation (3.14) must hold for each state for the value function vπ shown in Figure 3.2 (right) of Example 3.5. Show numerically that this equation holds for the center state, valued at +0.7, with respect to its four neighboring states, valued at +2.3, +0.4, −0.4, and +0.7. (These numbers are accurate only to one decimal place.) 3.5. Unified Notation for Episodic and Continuing Tasks 61 □ r s0 s, a a0 ↑ p q↑backup diagram Exercise 3.13 What is the Bellman equation for action values, that is, for qπ? It must give the action value qπ(s, a) in terms of the action values, qπ(s′, a′), of possible successors to the state–action pair (s, a). Hint: the backup diagram to the right corresponds to this equation. Show the sequence of equations analogous to (3.14), but for action values. □ Q*(s,driver) V putt sand green !1 s a n d !2 !2 !3 !4 !1 !5 !6 !4 !3 !3 !2 !4 sand green !1 s a n d !2 !3 !2 0 0 !" !" vputt q←(s, driver) Figure 3.3: A golf example: the state-value function for putting (upper) and the optimal action-value function for using the driver (lower). Example 3.6: Golf To formulate playing a hole of golf as a reinforcement learning task, we count a penalty (nega- tive reward) of −1 for each stroke until we hit the ball into the hole. The state is the location of the ball. The value of a state is the negative of the number of strokes to the hole from that location. Our actions are how we aim and swing at the ball, of course, and which club we se- lect. Let us take the former as given and consider just the choice of club, which we assume is either a putter or a driver. The upper part of Figure 3.3 shows a possi- ble state-value function, vputt(s), for the policy that always uses the putter. The terminal state in-the-hole has a value of 0. From anywhere on the green we as- sume we can make a putt; these states have value −1. Offthe green we can- not reach the hole by putting, and the value is greater. If we can reach the green from a state by putting, then that state must have value one less than the green's value, that is, −2. For simplicity, let us assume we can putt very precisely and deter- ministically, but with a limited range. This gives us the sharp contour line labeled −2 in the figure; all locations between that line and the green require exactly two strokes to complete the hole. Similarly, any location within putting range of the −2 contour line must have a value of −3, and so on to get all the contour lines shown in the figure. Putting doesn't get us out of sand traps, so they have a value of −∞. Overall, it takes us six strokes to get from the tee to the hole by putting. Exercise 3.14 In the gridworld example, rewards are positive for goals, negative for running into the edge of the world, and zero the rest of the time. Are the signs of these rewards important, or only the intervals between them? Prove, using (3.8), that adding a constant c to all the rewards adds a constant, vc, to the values of all states, and thus does not affect the relative values of any states under any policies. What is vc in terms of c and γ? □ Exercise 3.15 Now consider adding a constant c to all the rewards in an episodic task, such as maze running. Would this have any effect, or would it leave the task unchanged as in the continuing task above? Why or why not? Give an example. □ Exercise 3.16 The value of a state depends on the values of the actions possible in that state and on how likely each action is to be taken under the current policy. We can think of this in terms of a small backup diagram rooted at the state and considering each possible action: s taken with probability ↑(a|s) v↑(s) q↑(s, a) a1 a2 a3 Give the equation corresponding to this intuition and diagram for the value at the root node, vπ(s), in terms of the value at the expected leaf node, qπ(s, a), given St = s. This equation should include an expectation conditioned on following the policy, π. Then give a second equation in which the expected value is written out explicitly in terms of π(a|s) such that no expected value notation appears in the equation. □ Exercise 3.17 The value of an action, qπ(s, a), depends on the expected next reward and the expected sum of the remaining rewards. Again we can think of this in terms of a small backup diagram, this one rooted at an action (state–action pair) and branching to the possible next states: s, a q↑(s, a) s0 3 s0 2 s0 1 r1 r2 r3 expected rewards v↑(s0) Give the equation corresponding to this intuition and diagram for the

action value, $q_\pi(s, a)$, in terms of the expected next reward, R_{t+1}, and the expected next state value, $v_\pi(S_{t+1})$, given that $S_t = s$ and $A_t = a$. This equation should include an expectation but not one conditioned on following the policy. Then give a second equation, writing out the expected value explicitly in terms of $p(s', r|s, a)$ defined by (3.2), such that no expected value notation appears in the equation.

4.1 Policy Evaluation (Prediction)

First we consider how to compute the state-value function vπ for an arbitrary policy π. This is called policy evaluation in the DP literature. We also refer to it as the prediction problem. Recall from Chapter 3 that, for all s ∈S, vπ(s) .= Eπ[Gt | St =s] = Eπ[Rt+1 + γGt+1 | St =s] (from (3.9)) = Eπ[Rt+1 + γvπ(St+1) | St =s] (4.3) = X a π(a|s) X s′,r p(s′, r|s, a) h r + γvπ(s′) i , (4.4) where π(a|s) is the probability of taking action a in state s under policy π, and the ex- pectations are subscripted by π to indicate that they are conditional on π being followed. The existence and uniqueness of vπ are guaranteed as long as either γ < 1 or eventual termination is guaranteed from all states under the policy π. If the environment's dynamics are completely known, then (4.4) is a system of |S| si- multaneous linear equations in |S| unknowns (the vπ(s), s ∈S). In principle, its solution is a straightforward, if tedious, computation. For our purposes, iterative solution meth- ods are most suitable. Consider a sequence of approximate value functions v0, v1, v2, . . ., each mapping S+ to R (the real numbers). The initial approximation, v0, is chosen arbi- trarily (except that the terminal state, if any, must be given value 0), and each successive approximation is obtained by using the Bellman equation for vπ (4.4) as an update rule: vk+1(s) .= Eπ[Rt+1 + γvk(St+1) | St =s] = X a π(a|s) X s′,r p(s′, r|s, a) h r + γvk(s′) i , (4.5) for all s ∈S. Clearly, vk = vπ is a fixed point for this update rule because the Bellman equation for vπ assures us of equality in this case. Indeed, the sequence {vk} can be shown in general to converge to vπ as k →∞under the same conditions that guarantee the existence of vπ. This algorithm is called iterative policy evaluation. To produce each successive approximation, vk+1 from vk, iterative policy evaluation applies the same operation to each state s: it replaces the old value of s with a new value obtained from the old values of the successor states of s, and the expected immediate rewards, along all the one-step transitions possible under the policy being evaluated. We call this kind of operation an expected update. Each iteration of iterative policy evaluation updates the value of every state once to produce the new approximate value function vk+1. There are several different kinds of expected updates, depending on whether a state (as here) or a state–action pair is being updated, and depending on the precise way the estimated values of the successor states are combined. All the updates done in DP algorithms are called expected updates because they are based on an expectation over all possible next states rather than on a sample next state. The nature of an update can be expressed in an equation, as above, or in a backup diagram like those introduced in Chapter 3. For example, the backup diagram corresponding to the expected update used in iterative policy evaluation is shown on page 59. To write a sequential computer program to implement iterative policy evaluation as given by (4.5) you would have to use two arrays, one for the old values, vk(s), and one for the new values, vk+1(s). With two arrays, the new values can be computed one by one from the old values without the old values being changed. Of course it is easier to use one array and update the values "in place," that is, with each new value immediately overwriting the old one. Then, depending on the order in which the states are updated, sometimes new values are used instead of old ones on the right-hand side of (4.5). This in-place algorithm also converges to vπ; in fact, it usually converges faster than the two-array version, as you might expect, because it uses new data as soon as they are available. We think of the updates as being done in a sweep through the state space. For the in-place algorithm, the order in which states have their values updated during the sweep has a significant influence on the rate of convergence. We usually have the in-place version in mind when we think of DP algorithms. A complete in-place version of iterative policy evaluation is shown in pseudocode in the box below. Note how it handles termination. Formally, iterative policy evaluation converges only in the limit, but in practice it must be halted short of this. The pseu- docode tests the quantity maxs∈S |vk+1(s) −vk(s)| after each sweep and stops when it is sufficiently small. Iterative Policy Evaluation, for estimating V ≈vπ Input π, the policy to be evaluated Algorithm parameter: a small threshold θ > 0 determining accuracy of estimation

Initialize V (s), for all s ∈S+, arbitrarily except that V (terminal) = 0 Loop: ∆←0 Loop for each s ∈S: v ←V (s) V (s) ←P a π(a|s) P s′,r p(s′, r|s, a) \x02 r + γV (s′) \x03 ∆←max(∆, |v −V (s)|) until ∆< θ Example 4.1 Consider the 4×4 gridworld shown below. actions r = !1 on all transitions 1 2 3 4 5 6 7 8 9 10 11 12 13 14 Rt = −1 The nonterminal states are S = {1, 2, . . . , 14}. There are four actions possible in each state, A = {up, down, right, left}, which deterministically cause the corresponding state transitions, except that actions that would take the agent offthe grid in fact leave the state unchanged. Thus, for instance, p(6, −1|5, right) = 1, p(7, −1|7, right) = 1, and p(10, r|5, right) = 0 for all r ∈R. This is an undiscounted, episodic task. The reward is −1 on all transitions until the terminal state is reached. The terminal state is shaded in the figure (although it is shown in two places, it is formally one state). The expected reward function is thus r(s, a, s′) = −1 for all states s, s′ and actions a. Suppose the agent follows the equiprobable random policy (all actions equally likely). The left side of Figure 4.1 shows the sequence of value functions {vk} computed by iterative policy evaluation. The final estimate is in fact vπ, which in this case gives for each state the negation of the expected number of steps from that state until termination. Exercise 4.1 In Example 4.1, if π is the equiprobable random policy, what is qπ(11, down)? What is qπ(7, down)? □ Exercise 4.2 In Example 4.1, suppose a new state 15 is added to the gridworld just below state 13, and its actions, left, up, right, and down, take the agent to states 12, 13, 14, and 15, respectively. Assume that the transitions from the original states are unchanged. What, then, is vπ(15) for the equiprobable random policy? Now suppose the dynamics of state 13 are also changed, such that action down from state 13 takes the agent to the new state 15. What is vπ(15) for the equiprobable random policy in this case? □ Exercise 4.3 What are the equations analogous to (4.3), (4.4), and (4.5) for the action- value function qπ and its successive approximation by a sequence of functions q0, q1, q2, . . . ? □

4.2 Policy Improvement

Our reason for computing the value function for a policy is to help find better policies. Suppose we have determined the value function vπ for an arbitrary deterministic policy π. For some state s we would like to know whether or not we should change the policy to deterministically choose an action a6= π(s). We know how good it is to follow the current policy from s—that is vπ(s)—but would it be better or worse to change to the new policy? One way to answer this question is to consider selecting a in s and thereafter 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.7 -2.0 -2.0 -1.7 -2.0 -2.0 -2.0 -2.0 -2.0 -2.0 -1.7 -2.0 -2.0 -1.7 -2.4 -2.9 -3.0 -2.4 -2.9 -3.0 -2.9 -2.9 -3.0 -2.9 -2.4 -3.0 -2.9 -2.4 -6.1 -8.4 -9.0 -6.1 -7.7 -8.4 -8.4 -8.4 -8.4 -7.7 -6.1 -9.0 -8.4 -6.1 -14. -20. -22. -14. -18. -20. -20. -20. -20. -18. -14. -22. -20. -14. Vk for the Random Policy Greedy Policy w.r.t. Vk k = 0 k = 1 k = 2 k = 10 k = ! k = 3 optimal policy random policy 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 vk for the random policy vk greedy policy w.r.t. vk Figure 4.1: Convergence of iterative policy evaluation on a small gridworld. The left column is the sequence of approximations of the state-value function for the random policy (all actions equal). The right column is the sequence of greedy policies corresponding to the value function estimates (arrows are shown for all actions achieving the maximum). The last policy is guar- anteed only to be an improvement over the random policy, but in this case it, and all policies after the third iteration, are optimal. following the existing policy, π. The value of this way of behaving is qπ(s, a) .= E[Rt+1 + γvπ(St+1) | St =s, At =a] (4.6) = X s′,r p(s′, r|s, a) h r + γvπ(s′) i . The key criterion is whether this is greater than or less than vπ(s). If it is greater—that is, if it is better to select a once in s and thereafter follow π than it would be to follow π all the time—then one would expect it to be better still to select a every time s is encountered, and that the new policy would in fact be a better one overall. That this is true is a special case of a general result called the policy improvement theorem. Let π and π′ be any pair of deterministic policies such that, for all s ∈S, qπ(s, π′(s)) ≥vπ(s). (4.7) Then the policy π′ must

be as good as, or better than, π. That is, it must obtain greater or equal expected return from all states s ∈S: vπ′(s) ≥vπ(s). (4.8) Moreover, if there is strict inequality of (4.7) at any state, then there must be strict inequality of (4.8) at at least one state. This result applies in particular to the two policies that we considered in the previous paragraph, an original deterministic policy, π, and a changed policy, π′, that is identical to π except that π′(s) = a /= π(s). Obviously, (4.7) holds at all states other than s. Thus, if qπ(s, a) > vπ(s), then the changed policy is indeed better than π. The idea behind the proof of the policy improvement theorem is easy to understand. Starting from (4.7), we keep expanding the qπ side with (4.6) and reapplying (4.7) until we get vπ′(s): vπ(s) ≤qπ(s, π′(s)) = E[Rt+1 + γvπ(St+1) | St =s, At =π′(s)] (by (4.6)) = Eπ′[Rt+1 + γvπ(St+1) | St =s] ≤Eπ′[Rt+1 + γqπ(St+1, π′(St+1)) | St =s] = Eπ′[Rt+1 + γEπ′[Rt+2 + γvπ(St+2)|St+1] | St =s] = Eπ′\x02 Rt+1 + γRt+2 + γ2vπ(St+2) St =s \x03 ≤Eπ′\x02 Rt+1 + γRt+2 + γ2Rt+3 + γ3vπ(St+3) St =s \x03 ... ≤Eπ′\x02 Rt+1 + γRt+2 + γ2Rt+3 + γ3Rt+4 + · · · St =s \x03 = vπ′(s). So far we have seen how, given a policy and its value function, we can easily evaluate a change in the policy at a single state to a particular action. It is a natural extension to consider changes at all states and to all possible actions, selecting at each state the action that appears best according to qπ(s, a). In other words, to consider the new greedy policy, π′, given by π′(s) .= argmax a qπ(s, a) = argmax a E[Rt+1 + γvπ(St+1) | St =s, At =a] (4.9) = argmax a X s′,r p(s′, r|s, a) h r + γvπ(s′) i , where argmaxa denotes the value of a at which the expression that follows is maximized (with ties broken arbitrarily). The greedy policy takes the action that looks best in the short term—after one step of lookahead—according to vπ. By construction, the greedy policy meets the conditions of the policy improvement theorem (4.7), so we know that it is as good as, or better than, the original policy. The process of making a new policy that improves on an original policy, by making it greedy with respect to the value function of the original policy, is called policy improvement. Suppose the new greedy policy, π′, is as good as, but not better than, the old policy π. Then vπ = vπ′, and from (4.9) it follows that for all s ∈S: vπ′(s) = max a E[Rt+1 + γvπ′(St+1) | St =s, At =a] = max a X s′,r p(s′, r|s, a) h r + γvπ′(s′) i . But this is the same as the Bellman optimality equation (4.1), and therefore, vπ′ must be v∗, and both π and π′ must be optimal policies. Policy improvement thus must give us a strictly better policy except when the original policy is already optimal. So far in this section we have considered the special case of deterministic policies. In the general case, a stochastic policy π specifies probabilities, π(a|s), for taking each action, a, in each state, s. We will not go through the details, but in fact all the ideas of this section extend easily to stochastic policies. In particular, the policy improvement theorem carries through as stated for the stochastic case. In addition, if there are ties in policy improvement steps such as (4.9)—that is, if there are several actions at which the maximum is achieved—then in the stochastic case we need not select a single action from among them. Instead, each maximizing action can be given a portion of the probability of being selected in the new greedy policy. Any apportioning scheme is allowed as long as all submaximal actions are given zero probability. The last row of Figure 4.1 shows an example of policy improvement for stochastic policies. Here the original policy, π, is the equiprobable random policy, and the new policy, π′, is greedy with respect to vπ. The value function vπ is shown in the bottom- left diagram and the set of possible π′ is shown in the bottom-right diagram. The states with multiple arrows in the π′ diagram are those in which several actions achieve the maximum in (4.9); any apportionment of probability among these actions is permitted. The value function of any such policy, vπ′(s), can be seen by inspection to be either −1, −2, or −3 at all states, s ∈S, whereas vπ(s) is at most −14. Thus, vπ′(s) ≥vπ(s), for all s ∈S, illustrating policy improvement. Although in this case the new policy π′ happens to be optimal, in general only an improvement is guaranteed.