

## Chapter 2

# Multi-armed Bandits

The most important feature distinguishing reinforcement learning from other types of learning is that it uses training information that *evaluates* the actions taken rather than *instructs* by giving correct actions. This is what creates the need for active exploration, for an explicit search for good behavior. Purely evaluative feedback indicates how good the action taken was, but not whether it was the best or the worst action possible. Purely instructive feedback, on the other hand, indicates the correct action to take, independently of the action actually taken. This kind of feedback is the basis of supervised learning, which includes large parts of pattern classification, artificial neural networks, and system identification. In their pure forms, these two kinds of feedback are quite distinct: evaluative feedback depends entirely on the action taken, whereas instructive feedback is independent of the action taken.

In this chapter we study the evaluative aspect of reinforcement learning in a simplified setting, one that does not involve learning to act in more than one situation. This *nonassociative* setting is the one in which most prior work involving evaluative feedback has been done, and it avoids much of the complexity of the full reinforcement learning problem. Studying this case enables us to see most clearly how evaluative feedback differs from, and yet can be combined with, instructive feedback.

The particular nonassociative, evaluative feedback problem that we explore is a simple version of the ***k*-armed bandit** problem. We use this problem to introduce a number of basic learning methods which we extend in later chapters to apply to the full reinforcement learning problem. At the end of this chapter, we take a step closer to the full reinforcement learning problem by discussing what happens when the bandit problem becomes associative, that is, when actions are taken in more than one situation.

### 2.1 A *k*-armed Bandit Problem

Consider the following learning problem. You are faced repeatedly with a choice among  $k$  different options, or actions. After each choice you receive a numerical reward chosen from a stationary probability distribution that depends on the action you selected. Your

objective is to maximize the expected total reward over some time period, for example, over 1000 action selections, or *time steps*.

This is the original form of the *k-armed bandit problem*, so named by analogy to a slot machine, or “one-armed bandit,” except that it has  $k$  levers instead of one. Each action selection is like a play of one of the slot machine’s levers, and the rewards are the payoffs for hitting the jackpot. Through repeated action selections you are to maximize your winnings by concentrating your actions on the best levers. Another analogy is that of a doctor choosing between experimental treatments for a series of seriously ill patients. Each action is the selection of a treatment, and each reward is the survival or well-being of the patient. Today the term “bandit problem” is sometimes used for a generalization of the problem described above, but in this book we use it to refer just to this simple case.

In our  $k$ -armed bandit problem, each of the  $k$  actions has an expected or mean reward given that that action is selected; let us call this the *value* of that action. We denote the action selected on time step  $t$  as  $A_t$ , and the corresponding reward as  $R_t$ . The value then of an arbitrary action  $a$ , denoted  $q_*(a)$ , is the expected reward given that  $a$  is selected:

$$q_*(a) \doteq \mathbb{E}[R_t \mid A_t = a].$$

If you knew the value of each action, then it would be trivial to solve the  $k$ -armed bandit problem: you would always select the action with highest value. We assume that you do not know the action values with certainty, although you may have estimates. We denote the estimated value of action  $a$  at time step  $t$  as  $Q_t(a)$ . We would like  $Q_t(a)$  to be close to  $q_*(a)$ .

If you maintain estimates of the action values, then at any time step there is at least one action whose estimated value is greatest. We call these the *greedy* actions. When you select one of these actions, we say that you are *exploiting* your current knowledge of the values of the actions. If instead you select one of the nongreedy actions, then we say you are *exploring*, because this enables you to improve your estimate of the nongreedy action’s value. **Exploitation** is the right thing to do to maximize the expected reward on the one step, but **exploration** may produce the greater total reward in the long run. For example, suppose a greedy action’s value is known with certainty, while several other actions are estimated to be nearly as good but with substantial uncertainty. The uncertainty is such that at least one of these other actions probably is actually better than the greedy action, but you don’t know which one. If you have many time steps ahead on which to make action selections, then it may be better to explore the nongreedy actions and discover which of them are better than the greedy action. Reward is lower in the short run, during exploration, but higher in the long run because after you have discovered the better actions, you can exploit *them* many times. Because it is not possible both to explore and to exploit with any single action selection, one often refers to the “conflict” between exploration and exploitation.

In any specific case, whether it is better to **explore or exploit depends in** a complex way on the precise values of the estimates, uncertainties, and the number of remaining steps. There are many sophisticated methods for balancing exploration and exploitation for particular mathematical formulations of the  $k$ -armed bandit and related problems.

However, most of these methods make strong assumptions about stationarity and prior knowledge that are either violated or impossible to verify in applications and in the full reinforcement learning problem that we consider in subsequent chapters. The guarantees of optimality or bounded loss for these methods are of little comfort when the assumptions of their theory do not apply.

In this book we do not worry about balancing exploration and exploitation in a sophisticated way; we worry only about balancing them at all. In this chapter we present several simple balancing methods for the  $k$ -armed bandit problem and show that they work much better than methods that always exploit. The need to balance exploration and exploitation is a distinctive challenge that arises in reinforcement learning; the simplicity of our version of the  $k$ -armed bandit problem enables us to show this in a particularly clear form.

## 2.2 Action-value Methods

We begin by looking more closely at methods for estimating the values of actions and for using the estimates to make action selection decisions, which we collectively call *action-value methods*. Recall that the true value of an action is the mean reward when that action is selected. One natural way to estimate this is by averaging the rewards actually received:

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}, \quad * R_0=0 \quad (2.1)$$

where  $\mathbb{1}_{\text{predicate}}$  denotes the random variable that is 1 if *predicate* is true and 0 if it is not. If the denominator is zero, then we instead define  $Q_t(a)$  as some default value, such as 0. As the denominator goes to infinity, by the law of large numbers,  $Q_t(a)$  converges to  $q_*(a)$ . We call this the *sample-average* method for estimating action values because each estimate is an average of the sample of relevant rewards. Of course this is just one way to estimate action values, and not necessarily the best one. Nevertheless, for now let us stay with this simple estimation method and turn to the question of how the estimates might be used to select actions.

The simplest action selection rule is to select one of the actions with the highest estimated value, that is, one of the greedy actions as defined in the previous section. If there is more than one greedy action, then a selection is made among them in some arbitrary way, perhaps randomly. We write this *greedy* action selection method as

$$A_t \doteq \arg\max_a Q_t(a), \quad (2.2)$$

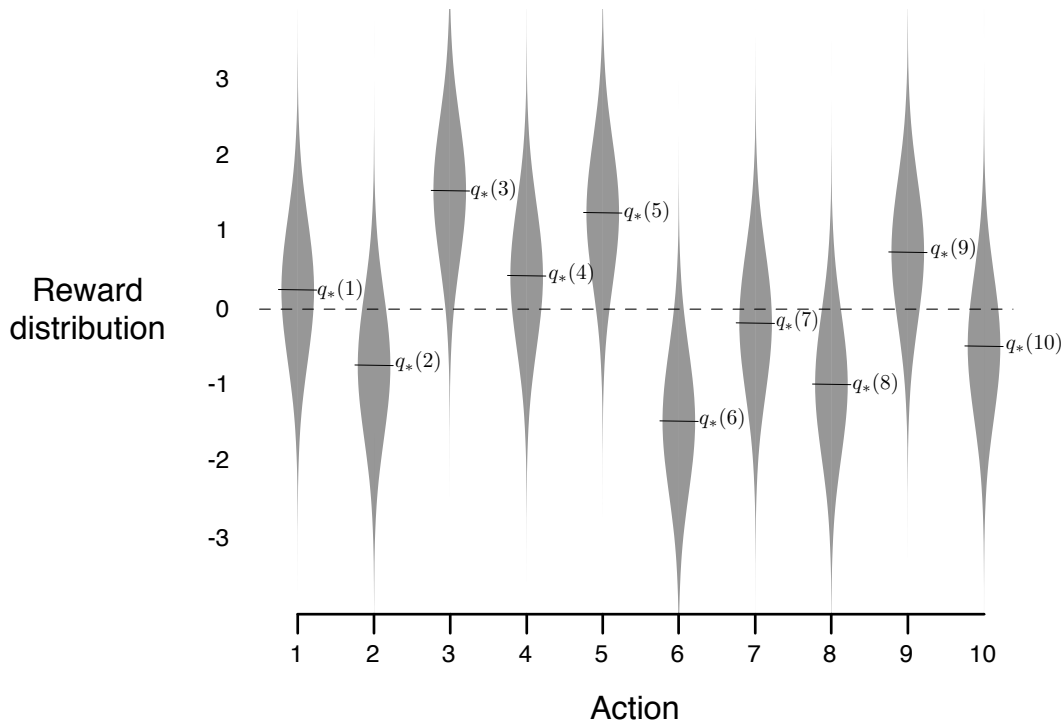
where  $\arg\max_a$  denotes the action  $a$  for which the expression that follows is maximized (again, with ties broken arbitrarily). Greedy action selection always exploits current knowledge to maximize immediate reward; it spends no time at all sampling apparently inferior actions to see if they might really be better. A simple alternative is to behave greedily most of the time, but every once in a while, say with small probability  $\varepsilon$ , instead

select randomly from among all the actions with equal probability, independently of the action-value estimates. We call methods using this near-greedy action selection rule  $\varepsilon$ -greedy methods. An advantage of these methods is that, in the limit as the number of steps increases, every action will be sampled an infinite number of times, thus ensuring that all the  $Q_t(a)$  converge to  $q_*(a)$ . This of course implies that the probability of selecting the optimal action converges to greater than  $1 - \varepsilon$ , that is, to near certainty. These are just asymptotic guarantees, however, and say little about the practical effectiveness of the methods.

*Exercise 2.1* In  $\varepsilon$ -greedy action selection, for the case of two actions and  $\varepsilon = 0.5$ , what is the probability that the greedy action is selected?  $\square$

## 2.3 The 10-armed Testbed

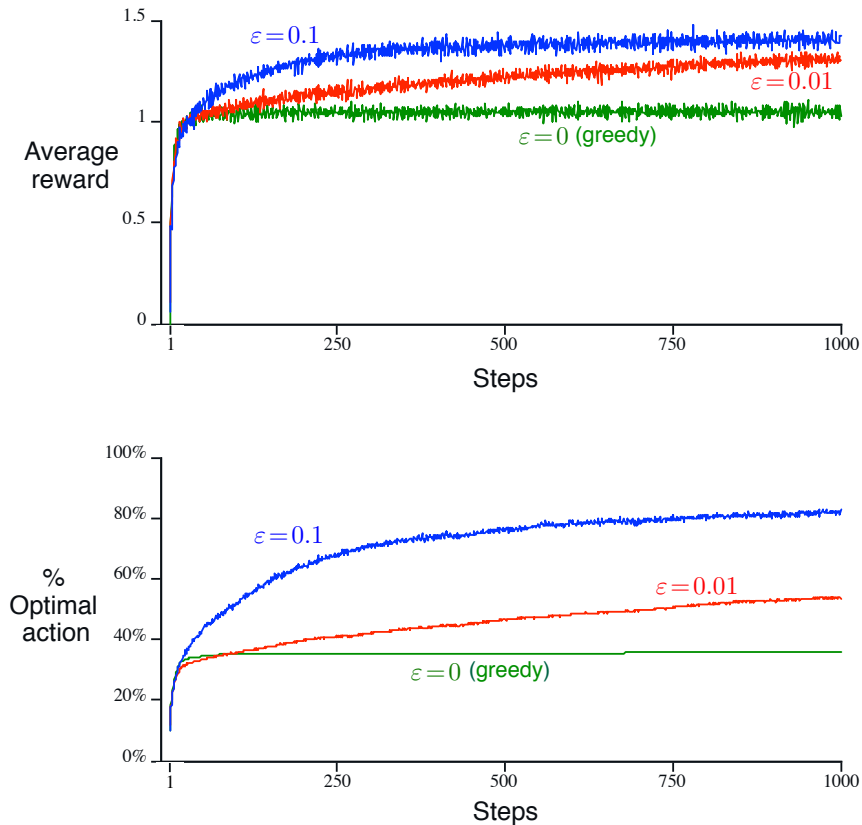
To roughly assess the relative effectiveness of the greedy and  $\varepsilon$ -greedy action-value methods, we compared them numerically on a suite of test problems. This was a set of 2000 randomly generated  $k$ -armed bandit problems with  $k = 10$ . For each bandit problem, such as the one shown in Figure 2.1, the action values,  $q_*(a)$ ,  $a = 1, \dots, 10$ ,



**Figure 2.1:** An example bandit problem from the 10-armed testbed. The true value  $q_*(a)$  of each of the ten actions was selected according to a normal distribution with mean zero and unit variance, and then the actual rewards were selected according to a mean  $q_*(a)$  unit variance normal distribution, as suggested by these gray distributions.

were selected according to a normal (Gaussian) distribution with mean 0 and variance 1. Then, when a learning method applied to that problem selected action  $A_t$  at time step  $t$ , the actual reward,  $R_t$ , was selected from a normal distribution with mean  $q_*(A_t)$  and variance 1. These distributions are shown in gray in Figure 2.1. We call this suite of test tasks the **10-armed testbed**. For any learning method, we can measure its performance and behavior as it improves with experience over 1000 time steps when applied to one of the bandit problems. This makes up one *run*. Repeating this for 2000 independent runs, each with a different bandit problem, we obtained measures of the learning algorithm's average behavior.

Figure 2.2 compares a greedy method with two  $\varepsilon$ -greedy methods ( $\varepsilon=0.01$  and  $\varepsilon=0.1$ ), as described above, on the 10-armed testbed. All the methods formed their action-value estimates using the sample-average technique. The upper graph shows the increase in expected reward with experience. The greedy method improved slightly faster than the other methods at the very beginning, but then leveled off at a lower level. It achieved a reward-per-step of only about 1, compared with the best possible of about 1.55 on this testbed. The greedy method performed significantly worse in the long run because it



**Figure 2.2:** Average performance of  $\varepsilon$ -greedy action-value methods on the 10-armed testbed. These data are averages over 2000 runs with different bandit problems. All methods used sample averages as their action-value estimates.

often got stuck performing suboptimal actions. The lower graph shows that the greedy method found the optimal action in only approximately one-third of the tasks. In the other two-thirds, its initial samples of the optimal action were disappointing, and it never returned to it. The  $\varepsilon$ -greedy methods eventually performed better because they continued to explore and to improve their chances of recognizing the optimal action. The  $\varepsilon = 0.1$  method explored more, and usually found the optimal action earlier, but it never selected that action more than 91% of the time. The  $\varepsilon = 0.01$  method improved more slowly, but eventually would perform better than the  $\varepsilon = 0.1$  method on both performance measures shown in the figure. It is also possible to reduce  $\varepsilon$  over time to try to get the best of both high and low values.

The advantage of  $\varepsilon$ -greedy over greedy methods depends on the task. For example, suppose the reward variance had been larger, say 10 instead of 1. With noisier rewards it takes more exploration to find the optimal action, and  $\varepsilon$ -greedy methods should fare even better relative to the greedy method. On the other hand, if the reward variances were zero, then the greedy method would know the true value of each action after trying it once. In this case the greedy method might actually perform best because it would soon find the optimal action and then never explore. But even in the deterministic case there is a large advantage to exploring if we weaken some of the other assumptions. For example, suppose the bandit task were nonstationary, that is, the true values of the actions changed over time. In this case exploration is needed even in the deterministic case to make sure one of the nongreedy actions has not changed to become better than the greedy one. As we shall see in the next few chapters, nonstationarity is the case most commonly encountered in reinforcement learning. Even if the underlying task is stationary and deterministic, the learner faces a set of banditlike decision tasks each of which changes over time as learning proceeds and the agent's decision-making policy changes. Reinforcement learning requires a balance between exploration and exploitation.

*Exercise 2.2: Bandit example* Consider a  $k$ -armed bandit problem with  $k = 4$  actions, denoted 1, 2, 3, and 4. Consider applying to this problem a bandit algorithm using  $\varepsilon$ -greedy action selection, sample-average action-value estimates, and initial estimates of  $Q_1(a) = 0$ , for all  $a$ . Suppose the initial sequence of actions and rewards is  $A_1 = 1$ ,  $R_1 = -1$ ,  $A_2 = 2$ ,  $R_2 = 1$ ,  $A_3 = 2$ ,  $R_3 = -2$ ,  $A_4 = 2$ ,  $R_4 = 2$ ,  $A_5 = 3$ ,  $R_5 = 0$ . On some of these time steps the  $\varepsilon$  case may have occurred, causing an action to be selected at random. On which time steps did this definitely occur? On which time steps could this possibly have occurred?  $\square$

*Exercise 2.3* In the comparison shown in Figure 2.2, which method will perform best in the long run in terms of cumulative reward and probability of selecting the best action? How much better will it be? Express your answer quantitatively.  $\square$

## 2.4 Incremental Implementation

The action-value methods we have discussed so far all estimate action values as sample averages of observed rewards. We now turn to the question of how these averages can be computed in a computationally efficient manner, in particular, with constant memory

and constant per-time-step computation.

To simplify notation we concentrate on a single action. Let  $R_i$  now denote the reward received after the  $i$ th selection of *this action*, and let  $Q_n$  denote the estimate of its action value after it has been selected  $n - 1$  times, which we can now write simply as

$$Q_n \doteq \frac{R_1 + R_2 + \cdots + R_{n-1}}{n - 1}.$$

The obvious implementation would be to maintain a record of all the rewards and then perform this computation whenever the estimated value was needed. However, if this is done, then the memory and computational requirements would grow over time as more rewards are seen. Each additional reward would require additional memory to store it and additional computation to compute the sum in the numerator.

As you might suspect, this is not really necessary. It is easy to devise incremental formulas for updating averages with small, constant computation required to process each new reward. Given  $Q_n$  and the  $n$ th reward,  $R_n$ , the new average of all  $n$  rewards can be computed by

$$\begin{aligned} Q_{n+1} &= \frac{1}{n} \sum_{i=1}^n R_i \\ &= \frac{1}{n} \left( R_n + \sum_{i=1}^{n-1} R_i \right) \\ &= \frac{1}{n} \left( R_n + (n-1) \frac{1}{n-1} \sum_{i=1}^{n-1} R_i \right) \\ &= \frac{1}{n} (R_n + (n-1)Q_n) \\ &= \frac{1}{n} (R_n + nQ_n - Q_n) \\ &= Q_n + \frac{1}{n} [R_n - Q_n], \end{aligned} \tag{2.3}$$

which holds even for  $n = 1$ , obtaining  $Q_2 = R_1$  for arbitrary  $Q_1$ . This implementation requires memory only for  $Q_n$  and  $n$ , and only the small computation (2.3) for each new reward.

This update rule (2.3) is of a form that occurs frequently throughout this book. **The general form** is

$$NewEstimate \leftarrow OldEstimate + StepSize [Target - OldEstimate]. \tag{2.4}$$

The expression  $[Target - OldEstimate]$  is an *error* in the estimate. It is reduced by taking a step toward the “Target.” The target is presumed to indicate a desirable direction in which to move, though it may be noisy. In the case above, for example, the target is the  $n$ th reward.

Note that the step-size parameter (*StepSize*) used in the incremental method (2.3) changes from time step to time step. In processing the  $n$ th reward for action  $a$ , the

method uses the step-size parameter  $\frac{1}{n}$ . In this book we denote the step-size parameter by  $\alpha$  or, more generally, by  $\alpha_t(a)$ .

Pseudocode for a complete bandit algorithm using incrementally computed sample averages and  $\varepsilon$ -greedy action selection is shown in the box below. The function *bandit*(*a*) is assumed to take an action and return a corresponding reward.

#### A simple bandit algorithm

Initialize, for  $a = 1$  to  $k$ :

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Loop forever:

$A \leftarrow \begin{cases} \operatorname{argmax}_a Q(a) & \text{with probability } 1 - \varepsilon \quad (\text{breaking ties randomly}) \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$

$R \leftarrow \text{bandit}(A)$

$N(A) \leftarrow N(A) + 1$

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)} [R - Q(A)]$

## 2.5 Tracking a Nonstationary Problem

The averaging methods discussed so far are appropriate for stationary bandit problems, that is, for bandit problems in which the reward probabilities do not change over time. As noted earlier, we often encounter reinforcement learning problems that are effectively nonstationary. In such cases **it makes sense to give more weight to recent rewards than to long-past rewards.** One of the most popular ways of doing this is to **use a constant step-size parameter.** For example, the incremental update rule (2.3) for updating an average  $Q_n$  of the  $n - 1$  past rewards is modified to be

$$Q_{n+1} \doteq Q_n + \alpha [R_n - Q_n], \quad (2.5)$$

where the step-size parameter  $\alpha \in (0, 1]$  is constant. This results in  $Q_{n+1}$  being a weighted average of past rewards and the initial estimate  $Q_1$ :

$$\begin{aligned} Q_{n+1} &= Q_n + \alpha [R_n - Q_n] \\ &= \alpha R_n + (1 - \alpha) Q_n \\ &= \alpha R_n + (1 - \alpha) [\alpha R_{n-1} + (1 - \alpha) Q_{n-1}] \\ &= \alpha R_n + (1 - \alpha) \alpha R_{n-1} + (1 - \alpha)^2 Q_{n-1} \\ &= \alpha R_n + (1 - \alpha) \alpha R_{n-1} + (1 - \alpha)^2 \alpha R_{n-2} + \\ &\quad \cdots + (1 - \alpha)^{n-1} \alpha R_1 + (1 - \alpha)^n Q_1 \\ &= (1 - \alpha)^n Q_1 + \sum_{i=1}^n \alpha (1 - \alpha)^{n-i} R_i. \end{aligned} \quad (2.6)$$



We call this a weighted average because the sum of the weights is  $(1 - \alpha)^n + \sum_{i=1}^n \alpha(1 - \alpha)^{n-i} = 1$ , as you can check for yourself. Note that the weight,  $\alpha(1 - \alpha)^{n-i}$ , given to the reward  $R_i$  depends on how many rewards ago,  $n - i$ , it was observed. The quantity  $1 - \alpha$  is less than 1, and thus the weight given to  $R_i$  decreases as the number of intervening rewards increases. In fact, the weight decays exponentially according to the exponent on  $1 - \alpha$ . (If  $1 - \alpha = 0$ , then all the weight goes on the very last reward,  $R_n$ , because of the convention that  $0^0 = 1$ .) Accordingly, this is sometimes called an **exponential recency-weighted average**.

Sometimes it is convenient to vary the step-size parameter from step to step. Let  $\alpha_n(a)$  denote the step-size parameter used to process the reward received after the  $n$ th selection of action  $a$ . As we have noted, the choice  $\alpha_n(a) = \frac{1}{n}$  results in the sample-average method, which is guaranteed to converge to the true action values by the law of large numbers. But of course convergence is not guaranteed for all choices of the sequence  $\{\alpha_n(a)\}$ . A well-known result in stochastic approximation theory gives us the conditions required to assure convergence with probability 1:

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty \quad \text{and} \quad \sum_{n=1}^{\infty} \alpha_n^2(a) < \infty. \quad (2.7)$$

The first condition is required to guarantee that the steps are large enough to eventually overcome any initial conditions or random fluctuations. The second condition guarantees that eventually the steps become small enough to assure convergence.

Note that both convergence conditions are met for the sample-average case,  $\alpha_n(a) = \frac{1}{n}$ , but not for the case of constant step-size parameter,  $\alpha_n(a) = \alpha$ . In the latter case, the second condition is not met, indicating that the estimates never completely converge but continue to vary in response to the most recently received rewards. As we mentioned above, this is actually desirable in a nonstationary environment, and problems that are effectively nonstationary are the most common in reinforcement learning. In addition, sequences of step-size parameters that meet the conditions (2.7) often converge very slowly or need considerable tuning in order to obtain a satisfactory convergence rate. Although sequences of step-size parameters that meet these convergence conditions are often used in theoretical work, they are seldom used in applications and empirical research.

*Exercise 2.4* If the step-size parameters,  $\alpha_n$ , are not constant, then the estimate  $Q_n$  is a weighted average of previously received rewards with a weighting different from that given by (2.6). What is the weighting on each prior reward for the general case, analogous to (2.6), in terms of the sequence of step-size parameters?  $\square$

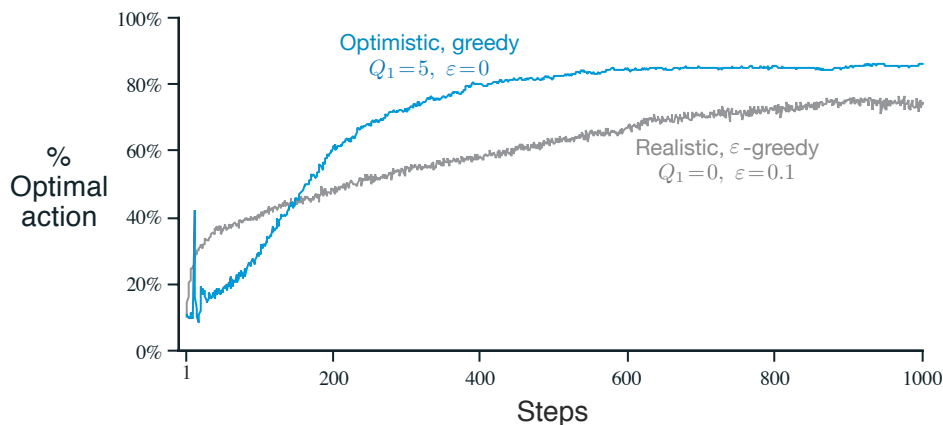
*Exercise 2.5 (programming)* Design and conduct an experiment to demonstrate the difficulties that sample-average methods have for nonstationary problems. Use a modified version of the 10-armed testbed in which all the  $q_*(a)$  start out equal and then take independent random walks (say by adding a normally distributed increment with mean zero and standard deviation 0.01 to all the  $q_*(a)$  on each step). Prepare plots like Figure 2.2 for an action-value method using sample averages, incrementally computed, and another action-value method using a constant step-size parameter,  $\alpha = 0.1$ . Use  $\varepsilon = 0.1$  and longer runs, say of 10,000 steps.  $\square$

## 2.6 Optimistic Initial Values

All the methods we have discussed so far are dependent to some extent on the initial action-value estimates,  $Q_1(a)$ . In the language of statistics, these methods are *biased* by their initial estimates. For the sample-average methods, the bias disappears once all actions have been selected at least once, but for methods with constant  $\alpha$ , the bias is permanent, though decreasing over time as given by (2.6). In practice, this kind of bias is usually not a problem and can sometimes be very helpful. The downside is that the initial estimates become, in effect, a set of parameters that must be picked by the user, if only to set them all to zero. The upside is that they provide an easy way to supply some prior knowledge about what level of rewards can be expected.

Initial action values can also be used as a simple way to encourage exploration. Suppose that instead of setting the initial action values to zero, as we did in the 10-armed testbed, we set them all to +5. Recall that the  $q_*(a)$  in this problem are selected from a normal distribution with mean 0 and variance 1. An initial estimate of +5 is thus wildly optimistic. But this optimism encourages action-value methods to explore. Whichever actions are initially selected, the reward is less than the starting estimates; the learner switches to other actions, being “disappointed” with the rewards it is receiving. The result is that all actions are tried several times before the value estimates converge. The system does a fair amount of exploration even if greedy actions are selected all the time.

Figure 2.3 shows the performance on the 10-armed bandit testbed of a greedy method using  $Q_1(a) = +5$ , for all  $a$ . For comparison, also shown is an  $\varepsilon$ -greedy method with  $Q_1(a) = 0$ . Initially, the optimistic method performs worse because it explores more, but eventually it performs better because its exploration decreases with time. We call this technique for encouraging exploration *optimistic initial values*. We regard it as a simple trick that can be quite effective on stationary problems, but it is far from being a generally useful approach to encouraging exploration. For example, it is not well suited to nonstationary problems because its drive for exploration is inherently



**Figure 2.3:** The effect of optimistic initial action-value estimates on the 10-armed testbed. Both methods used a constant step-size parameter,  $\alpha = 0.1$ .

temporary. If the task changes, creating a renewed need for exploration, this method cannot help. Indeed, any method that focuses on the initial conditions in any special way is unlikely to help with the general nonstationary case. The beginning of time occurs only once, and thus we should not focus on it too much. This criticism applies as well to the sample-average methods, which also treat the beginning of time as a special event, averaging all subsequent rewards with equal weights. Nevertheless, all of these methods are very simple, and one of them—or some simple combination of them—is often adequate in practice. In the rest of this book we make frequent use of several of these simple exploration techniques.

*Exercise 2.6: Mysterious Spikes* The results shown in Figure 2.3 should be quite reliable because they are averages over 2000 individual, randomly chosen 10-armed bandit tasks. Why, then, are there oscillations and spikes in the early part of the curve for the optimistic method? In other words, what might make this method perform particularly better or worse, on average, on particular early steps?  $\square$

*Exercise 2.7: Unbiased Constant-Step-Size Trick* In most of this chapter we have used sample averages to estimate action values because sample averages do not produce the initial bias that constant step sizes do (see the analysis leading to (2.6)). However, sample averages are not a completely satisfactory solution because they may perform poorly on nonstationary problems. Is it possible to avoid the bias of constant step sizes while retaining their advantages on nonstationary problems? One way is to use a step size of

$$\beta_n \doteq \alpha / \bar{o}_n, \quad (2.8)$$

to process the  $n$ th reward for a particular action, where  $\alpha > 0$  is a conventional constant step size, and  $\bar{o}_n$  is a trace of one that starts at 0:

$$\bar{o}_n \doteq \bar{o}_{n-1} + \alpha(1 - \bar{o}_{n-1}), \quad \text{for } n \geq 0, \quad \text{with } \bar{o}_0 \doteq 0. \quad (2.9)$$

Carry out an analysis like that in (2.6) to show that  $Q_n$  is an exponential recency-weighted average *without initial bias*.  $\square$

## 2.7 Upper-Confidence-Bound Action Selection

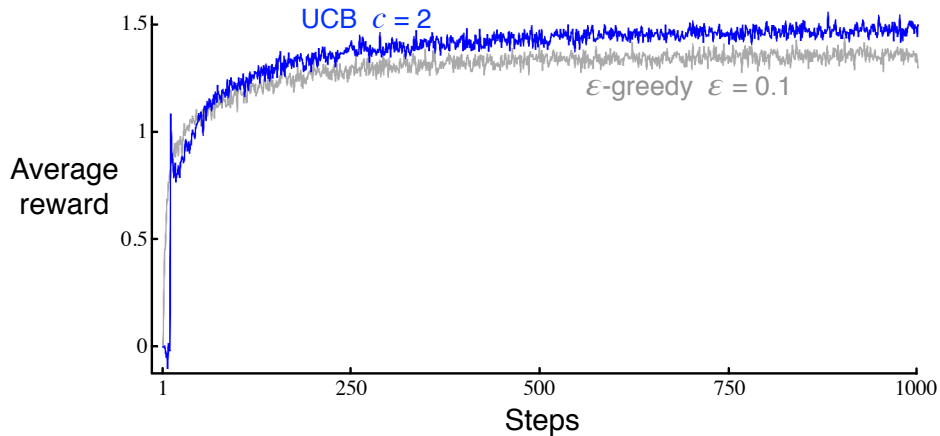
Exploration is needed because there is always uncertainty about the accuracy of the action-value estimates. The greedy actions are those that look best at present, but some of the other actions may actually be better.  $\varepsilon$ -greedy action selection forces the non-greedy actions to be tried, but indiscriminately, with no preference for those that are nearly greedy or particularly uncertain. It would be better to select among the non-greedy actions according to their potential for actually being optimal, taking into account both how close their estimates are to being maximal and the uncertainties in those estimates. One effective way of doing this is to select actions according to

$$A_t \doteq \operatorname{argmax}_a \left[ Q_t(a) + c \sqrt{\frac{\ln t}{N_t(a)}} \right], \quad (2.10)$$

where  $\ln t$  denotes the natural logarithm of  $t$  (the number that  $e \approx 2.71828$  would have to be raised to in order to equal  $t$ ),  $N_t(a)$  denotes the number of times that action  $a$  has been selected prior to time  $t$  (the denominator in (2.1)), and the number  $c > 0$  controls the degree of exploration. If  $N_t(a) = 0$ , then  $a$  is considered to be a maximizing action.

The idea of this *upper confidence bound* (UCB) action selection is that the square-root term is a measure of the uncertainty or variance in the estimate of  $a$ 's value. The quantity being max'ed over is thus a sort of upper bound on the possible true value of action  $a$ , with  $c$  determining the confidence level. Each time  $a$  is selected the uncertainty is presumably reduced:  $N_t(a)$  increments, and, as it appears in the denominator, the uncertainty term decreases. On the other hand, each time an action other than  $a$  is selected,  $t$  increases but  $N_t(a)$  does not; because  $t$  appears in the numerator, the uncertainty estimate increases. The use of the natural logarithm means that the increases get smaller over time, but are unbounded; all actions will eventually be selected, but actions with lower value estimates, or that have already been selected frequently, will be selected with decreasing frequency over time.

Results with UCB on the 10-armed testbed are shown in Figure 2.4. UCB often performs well, as shown here, but is more difficult than  $\varepsilon$ -greedy to extend beyond bandits to the more general reinforcement learning settings considered in the rest of this book. One difficulty is in dealing with nonstationary problems; methods more complex than those presented in Section 2.5 would be needed. Another difficulty is dealing with large state spaces, particularly when using function approximation as developed in Part II of this book. In these more advanced settings the idea of UCB action selection is usually not practical.



**Figure 2.4:** Average performance of UCB action selection on the 10-armed testbed. As shown, UCB generally performs better than  $\varepsilon$ -greedy action selection, except in the first  $k$  steps, when it selects randomly among the as-yet-untried actions.

*Exercise 2.8: UCB Spikes* In Figure 2.4 the UCB algorithm shows a distinct spike in performance on the 11th step. Why is this? Note that for your answer to be fully satisfactory it must explain both why the reward increases on the 11th step and why it decreases on the subsequent steps. Hint: If  $c = 1$ , then the spike is less prominent.  $\square$

## 2.8 Gradient Bandit Algorithms

So far in this chapter we have considered methods that estimate action values and use those estimates to select actions. This is often a good approach, but it is not the only one possible. In this section we consider learning a numerical *preference* for each action  $a$ , which we denote  $H_t(a)$ . The larger the preference, the more often that action is taken, but the preference has no interpretation in terms of reward. Only the relative preference of one action over another is important; if we add 1000 to all the action preferences there is no effect on the action probabilities, which are determined according to a *soft-max distribution* (i.e., Gibbs or Boltzmann distribution) as follows:

$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^k e^{H_t(b)}} \doteq \pi_t(a), \quad (2.11)$$

where here we have also introduced a useful new notation,  $\pi_t(a)$ , for the probability of taking action  $a$  at time  $t$ . Initially all action preferences are the same (e.g.,  $H_1(a) = 0$ , for all  $a$ ) so that all actions have an equal probability of being selected.

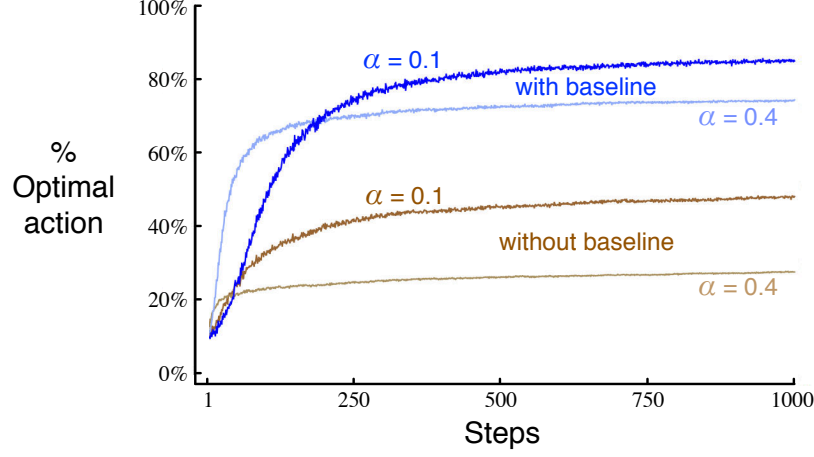
*Exercise 2.9* Show that in the case of two actions, the soft-max distribution is the same as that given by the logistic, or sigmoid, function often used in statistics and artificial neural networks.  $\square$

There is a natural learning algorithm for soft-max action preferences based on the idea of stochastic gradient ascent. On each step, after selecting action  $A_t$  and receiving the reward  $R_t$ , the action preferences are updated by:

$$\begin{aligned} H_{t+1}(A_t) &\doteq H_t(A_t) + \alpha(R_t - \bar{R}_t)(1 - \pi_t(A_t)), & \text{and} \\ H_{t+1}(a) &\doteq H_t(a) - \alpha(R_t - \bar{R}_t)\pi_t(a), & \text{for all } a \neq A_t, \end{aligned} \quad (2.12)$$

where  $\alpha > 0$  is a step-size parameter, and  $\bar{R}_t \in \mathbb{R}$  is the average of all the rewards up to but not including time  $t$ , which can be computed incrementally as described in Section 2.4 (or Section 2.5 if the problem is nonstationary). The  $\bar{R}_t$  term serves as a baseline with which the reward is compared. If the reward is higher than the baseline, then the probability of taking  $A_t$  in the future is increased, and if the reward is below baseline, then the probability is decreased. The non-selected actions move in the opposite direction.

Figure 2.5 shows results with the gradient bandit algorithm on a variant of the 10-armed testbed in which the true expected rewards were selected according to a normal distribution with a mean of +4 instead of zero (and with unit variance as before). This shifting up of all the rewards has absolutely no effect on the gradient bandit algorithm because of the reward baseline term, which instantaneously adapts to the new level. But if the baseline were omitted (that is, if  $\bar{R}_t$  was taken to be constant zero in (2.12)), then performance would be significantly degraded, as shown in the figure.



**Figure 2.5:** Average performance of the gradient bandit algorithm with and without a reward baseline on the 10-armed testbed when the  $q_*(a)$  are chosen to be near +4 rather than near zero.

#### The Bandit Gradient Algorithm as Stochastic Gradient Ascent

One can gain a deeper insight into the gradient bandit algorithm by understanding it as a stochastic approximation to gradient ascent. In exact *gradient ascent*, each action preference  $H_t(a)$  would be incremented proportional to the increment's effect on performance:

$$H_{t+1}(a) \doteq H_t(a) + \alpha \frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)}, \quad (2.13)$$

where the measure of performance here is the expected reward:

$$\mathbb{E}[R_t] = \sum_x \pi_t(x) q_*(x),$$

and the measure of the increment's effect is the *partial derivative* of this performance measure with respect to the action preference. Of course, it is not possible to implement gradient ascent exactly in our case because by assumption we do not know the  $q_*(x)$ , but in fact the updates of our algorithm (2.12) are equal to (2.13) in expected value, making the algorithm an instance of *stochastic gradient ascent*. The calculations showing this require only beginning calculus, but take several

steps. First we take a closer look at the exact performance gradient:

$$\begin{aligned}\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} &= \frac{\partial}{\partial H_t(a)} \left[ \sum_x \pi_t(x) q_*(x) \right] \\ &= \sum_x q_*(x) \frac{\partial \pi_t(x)}{\partial H_t(a)} \\ &= \sum_x (q_*(x) - B_t) \frac{\partial \pi_t(x)}{\partial H_t(a)},\end{aligned}$$

where  $B_t$ , called the *baseline*, can be any scalar that does not depend on  $x$ . We can include a baseline here without changing the equality because the gradient sums to zero over all the actions,  $\sum_x \frac{\partial \pi_t(x)}{\partial H_t(a)} = 0$ ; as  $H_t(a)$  is changed, some actions' probabilities go up and some go down, but the sum of the changes must be zero because the sum of the probabilities is always one.

Next we multiply each term of the sum by  $\pi_t(x)/\pi_t(x)$ :

$$\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} = \sum_x \pi_t(x) (q_*(x) - B_t) \frac{\partial \pi_t(x)}{\partial H_t(a)} / \pi_t(x).$$

The equation is now in the form of an expectation, summing over all possible values  $x$  of the random variable  $A_t$ , then multiplying by the probability of taking those values. Thus:

$$\begin{aligned}&= \mathbb{E} \left[ (q_*(A_t) - B_t) \frac{\partial \pi_t(A_t)}{\partial H_t(a)} / \pi_t(A_t) \right] \\ &= \mathbb{E} \left[ (R_t - \bar{R}_t) \frac{\partial \pi_t(A_t)}{\partial H_t(a)} / \pi_t(A_t) \right],\end{aligned}$$

where here we have chosen the baseline  $B_t = \bar{R}_t$  and substituted  $R_t$  for  $q_*(A_t)$ , which is permitted because  $\mathbb{E}[R_t|A_t] = q_*(A_t)$ . Shortly we will establish that  $\frac{\partial \pi_t(x)}{\partial H_t(a)} = \pi_t(x)(\mathbb{1}_{a=x} - \pi_t(a))$ , where  $\mathbb{1}_{a=x}$  is defined to be 1 if  $a = x$ , else 0. Assuming that for now, we have

$$\begin{aligned}&= \mathbb{E} \left[ (R_t - \bar{R}_t) \pi_t(A_t) (\mathbb{1}_{a=A_t} - \pi_t(a)) / \pi_t(A_t) \right] \\ &= \mathbb{E} \left[ (R_t - \bar{R}_t) (\mathbb{1}_{a=A_t} - \pi_t(a)) \right].\end{aligned}$$

Recall that our plan has been to write the performance gradient as an expectation of something that we can sample on each step, as we have just done, and then update on each step proportional to the sample. Substituting a sample of the expectation above for the performance gradient in (2.13) yields:

$$H_{t+1}(a) = H_t(a) + \alpha (R_t - \bar{R}_t) (\mathbb{1}_{a=A_t} - \pi_t(a)), \quad \text{for all } a,$$

which you may recognize as being equivalent to our original algorithm (2.12).

Thus it remains only to show that  $\frac{\partial \pi_t(x)}{\partial H_t(a)} = \pi_t(x)(\mathbb{1}_{a=x} - \pi_t(a))$ , as we assumed. Recall the standard quotient rule for derivatives:

$$\frac{\partial}{\partial x} \left[ \frac{f(x)}{g(x)} \right] = \frac{\frac{\partial f(x)}{\partial x} g(x) - f(x) \frac{\partial g(x)}{\partial x}}{g(x)^2}.$$

Using this, we can write

$$\begin{aligned} \frac{\partial \pi_t(x)}{\partial H_t(a)} &= \frac{\partial}{\partial H_t(a)} \pi_t(x) \\ &= \frac{\partial}{\partial H_t(a)} \left[ \frac{e^{H_t(x)}}{\sum_{y=1}^k e^{H_t(y)}} \right] \\ &= \frac{\frac{\partial e^{H_t(x)}}{\partial H_t(a)} \sum_{y=1}^k e^{H_t(y)} - e^{H_t(x)} \frac{\partial \sum_{y=1}^k e^{H_t(y)}}{\partial H_t(a)}}{\left( \sum_{y=1}^k e^{H_t(y)} \right)^2} \quad (\text{by the quotient rule}) \\ &= \frac{\mathbb{1}_{a=x} e^{H_t(x)} \sum_{y=1}^k e^{H_t(y)} - e^{H_t(x)} e^{H_t(a)}}{\left( \sum_{y=1}^k e^{H_t(y)} \right)^2} \quad (\text{because } \frac{\partial e^x}{\partial x} = e^x) \\ &= \frac{\mathbb{1}_{a=x} e^{H_t(x)}}{\sum_{y=1}^k e^{H_t(y)}} - \frac{e^{H_t(x)} e^{H_t(a)}}{\left( \sum_{y=1}^k e^{H_t(y)} \right)^2} \\ &= \mathbb{1}_{a=x} \pi_t(x) - \pi_t(x) \pi_t(a) \\ &= \pi_t(x)(\mathbb{1}_{a=x} - \pi_t(a)). \end{aligned} \quad \text{Q.E.D.}$$

We have just shown that the expected update of the gradient bandit algorithm is equal to the gradient of expected reward, and thus that the algorithm is an instance of stochastic gradient ascent. This assures us that the algorithm has robust convergence properties.

Note that we did not require any properties of the reward baseline other than that it does not depend on the selected action. For example, we could have set it to zero, or to 1000, and the algorithm would still be an instance of stochastic gradient ascent. The choice of the baseline does not affect the expected update of the algorithm, but it does affect the variance of the update and thus the rate of convergence (as shown, e.g., in Figure 2.5). Choosing it as the average of the rewards may not be the very best, but it is simple and works well in practice.



## 2.9 Associative Search (Contextual Bandits)

So far in this chapter we have considered only nonassociative tasks, that is, tasks in which there is no need to associate different actions with different situations. In these tasks the learner either tries to find a single best action when the task is stationary, or tries to track the best action as it changes over time when the task is nonstationary. However, in a general reinforcement learning task there is more than one situation, and the goal is to learn a policy: a mapping from situations to the actions that are best in those situations. To set the stage for the full problem, we briefly discuss the simplest way in which nonassociative tasks extend to the associative setting.

As an example, suppose there are several different  $k$ -armed bandit tasks, and that on each step you confront one of these chosen at random. Thus, the bandit task changes randomly from step to step. This would appear to you as a single, nonstationary  $k$ -armed bandit task whose true action values change randomly from step to step. You could try using one of the methods described in this chapter that can handle nonstationarity, but unless the true action values change slowly, these methods will not work very well. Now suppose, however, that when a bandit task is selected for you, you are given some distinctive clue about its identity (but not its action values). Maybe you are facing an actual slot machine that changes the color of its display as it changes its action values. Now you can learn a policy associating each task, signaled by the color you see, with the best action to take when facing that task—for instance, if red, select arm 1; if green, select arm 2. With the right policy you can usually do much better than you could in the absence of any information distinguishing one bandit task from another.

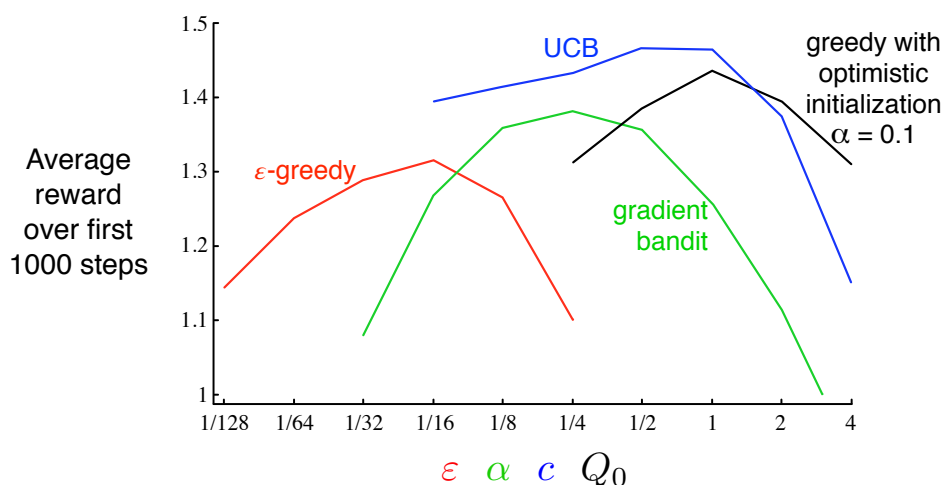
This is an example of an *associative search* task, so called because it involves both trial-and-error learning to *search* for the best actions, and *association* of these actions with the situations in which they are best. Associative search tasks are often now called *contextual bandits* in the literature. Associative search tasks are intermediate between the  $k$ -armed bandit problem and the full reinforcement learning problem. They are like the full reinforcement learning problem in that they involve learning a policy, but like our version of the  $k$ -armed bandit problem in that each action affects only the immediate reward. If actions are allowed to affect the *next situation* as well as the reward, then we have the full reinforcement learning problem. We present this problem in the next chapter and consider its ramifications throughout the rest of the book.

*Exercise 2.10* Suppose you face a 2-armed bandit task whose true action values change randomly from time step to time step. Specifically, suppose that, for any time step, the true values of actions 1 and 2 are respectively 0.1 and 0.2 with probability 0.5 (case A), and 0.9 and 0.8 with probability 0.5 (case B). If you are not able to tell which case you face at any step, what is the best expectation of success you can achieve and how should you behave to achieve it? Now suppose that on each step you are told whether you are facing case A or case B (although you still don't know the true action values). This is an associative search task. What is the best expectation of success you can achieve in this task, and how should you behave to achieve it?  $\square$

## 2.10 Summary

We have presented in this chapter several simple ways of balancing exploration and exploitation. The  $\epsilon$ -greedy methods choose randomly a small fraction of the time, whereas UCB methods choose deterministically but achieve exploration by subtly favoring at each step the actions that have so far received fewer samples. Gradient bandit algorithms estimate not action values, but action preferences, and favor the more preferred actions in a graded, probabilistic manner using a soft-max distribution. The simple expedient of initializing estimates optimistically causes even greedy methods to explore significantly.

It is natural to ask which of these methods is best. Although this is a difficult question to answer in general, we can certainly run them all on the 10-armed testbed that we have used throughout this chapter and compare their performances. A complication is that they all have a parameter; to get a meaningful comparison we have to consider their performance as a function of their parameter. Our graphs so far have shown the course of learning over time for each algorithm and parameter setting, to produce a *learning curve* for that algorithm and parameter setting. If we plotted learning curves for all algorithms and all parameter settings, then the graph would be too complex and crowded to make clear comparisons. Instead we summarize a complete learning curve by its average value over the 1000 steps; this value is proportional to the area under the learning curve. Figure 2.6 shows this measure for the various bandit algorithms from this chapter, each as a function of its own parameter shown on a single scale on the x-axis. This kind of graph is called a *parameter study*. Note that the parameter values are varied by factors of two and presented on a log scale. Note also the characteristic inverted-U shapes of each algorithm's performance; all the algorithms perform best at an intermediate value of their parameter, neither too large nor too small. In assessing



**Figure 2.6:** A parameter study of the various bandit algorithms presented in this chapter. Each point is the average reward obtained over 1000 steps with a particular algorithm at a particular setting of its parameter.

a method, we should attend not just to how well it does at its best parameter setting, but also to how sensitive it is to its parameter value. All of these algorithms are fairly insensitive, performing well over a range of parameter values varying by about an order of magnitude. Overall, on this problem, UCB seems to perform best.

Despite their simplicity, in our opinion the methods presented in this chapter can fairly be considered the state of the art. There are more sophisticated methods, but their complexity and assumptions make them impractical for the full reinforcement learning problem that is our real focus. Starting in Chapter 5 we present learning methods for solving the full reinforcement learning problem that use in part the simple methods explored in this chapter.

Although the simple methods explored in this chapter may be the best we can do at present, they are far from a fully satisfactory solution to the problem of balancing exploration and exploitation.

One well-studied approach to balancing exploration and exploitation in  $k$ -armed bandit problems is to compute a special kind of action value called a *Gittins index*. In certain important special cases, this computation is tractable and leads directly to optimal solutions, although it does require complete knowledge of the prior distribution of possible problems, which we generally assume is not available. In addition, neither the theory nor the computational tractability of this approach appear to generalize to the full reinforcement learning problem that we consider in the rest of the book.

The Gittins-index approach is an instance of *Bayesian* methods, which assume a known initial distribution over the action values and then update the distribution exactly after each step (assuming that the true action values are stationary). In general, the update computations can be very complex, but for certain special distributions (called *conjugate priors*) they are easy. One possibility is to then select actions at each step according to their posterior probability of being the best action. This method, sometimes called *posterior sampling* or *Thompson sampling*, often performs similarly to the best of the distribution-free methods we have presented in this chapter.

In the Bayesian setting it is even conceivable to compute the *optimal* balance between exploration and exploitation. One can compute for any possible action the probability of each possible immediate reward and the resultant posterior distributions over action values. This evolving distribution becomes the *information state* of the problem. Given a horizon, say of 1000 steps, one can consider all possible actions, all possible resulting rewards, all possible next actions, all next rewards, and so on for all 1000 steps. Given the assumptions, the rewards and probabilities of each possible chain of events can be determined, and one need only pick the best. But the tree of possibilities grows extremely rapidly; even if there were only two actions and two rewards, the tree would have  $2^{2000}$  leaves. It is generally not feasible to perform this immense computation exactly, but perhaps it could be approximated efficiently. This approach would effectively turn the bandit problem into an instance of the full reinforcement learning problem. In the end, we may be able to use approximate reinforcement learning methods such as those presented in Part II of this book to approach this optimal solution. But that is a topic for research and beyond the scope of this introductory book.

*Exercise 2.11 (programming)* Make a figure analogous to Figure 2.6 for the nonstationary case outlined in Exercise 2.5. Include the constant-step-size  $\varepsilon$ -greedy algorithm with  $\alpha=0.1$ . Use runs of 200,000 steps and, as a performance measure for each algorithm and parameter setting, use the average reward over the last 100,000 steps.  $\square$

## Bibliographical and Historical Remarks

- 2.1** Bandit problems have been studied in statistics, engineering, and psychology. In statistics, bandit problems fall under the heading “sequential design of experiments,” introduced by Thompson (1933, 1934) and Robbins (1952), and studied by Bellman (1956). Berry and Fristedt (1985) provide an extensive treatment of bandit problems from the perspective of statistics. Narendra and Thathachar (1989) treat bandit problems from the engineering perspective, providing a good discussion of the various theoretical traditions that have focused on them. In psychology, bandit problems have played roles in statistical learning theory (e.g., Bush and Mosteller, 1955; Estes, 1950).

The term *greedy* is often used in the heuristic search literature (e.g., Pearl, 1984). The conflict between exploration and exploitation is known in control engineering as the conflict between identification (or estimation) and control (e.g., Witten, 1976b). Feldbaum (1965) called it the *dual control* problem, referring to the need to solve the two problems of identification and control simultaneously when trying to control a system under uncertainty. In discussing aspects of genetic algorithms, Holland (1975) emphasized the importance of this conflict, referring to it as the conflict between the need to exploit and the need for new information.

- 2.2** Action-value methods for our  $k$ -armed bandit problem were first proposed by Thathachar and Sastry (1985). These are often called *estimator algorithms* in the learning automata literature. The term *action value* is due to Watkins (1989). The first to use  $\varepsilon$ -greedy methods may also have been Watkins (1989, p. 187), but the idea is so simple that some earlier use seems likely.
- 2.4–5** This material falls under the general heading of stochastic iterative algorithms, which is well covered by Bertsekas and Tsitsiklis (1996).
- 2.6** Optimistic initialization was used in reinforcement learning by Sutton (1996).
- 2.7** Early work on using estimates of the upper confidence bound to select actions was done by Lai and Robbins (1985), Kaelbling (1993b), and Agrawal (1995). The UCB algorithm we present here is called UCB1 in the literature and was first developed by Auer, Cesa-Bianchi and Fischer (2002).
- 2.8** Gradient bandit algorithms are a special case of the gradient-based reinforcement learning algorithms introduced by Williams (1992), and that later developed into the actor–critic and policy-gradient algorithms that we treat later in this book. Our development here was influenced by that by Balaraman Ravindran (personal

communication). Further discussion of the choice of baseline is provided there and by Greensmith, Bartlett, and Baxter (2002, 2004) and Dick (2015). Early systematic studies of algorithms like this were done by Sutton (1984).

The term *soft-max* for the action selection rule (2.11) is due to Bridle (1990). This rule appears to have been first proposed by Luce (1959).

- 2.9** The term *associative search* and the corresponding problem were introduced by Barto, Sutton, and Brouwer (1981). The term *associative reinforcement learning* has also been used for associative search (Barto and Anandan, 1985), but we prefer to reserve that term as a synonym for the full reinforcement learning problem (as in Sutton, 1984). (And, as we noted, the modern literature also uses the term “contextual bandits” for this problem.) We note that Thorndike’s Law of Effect (quoted in Chapter 1) describes associative search by referring to the formation of associative links between situations (states) and actions. According to the terminology of operant, or instrumental, conditioning (e.g., Skinner, 1938), a discriminative stimulus is a stimulus that signals the presence of a particular reinforcement contingency. In our terms, different discriminative stimuli correspond to different states.

- 2.10** Bellman (1956) was the first to show how dynamic programming could be used to compute the optimal balance between exploration and exploitation within a Bayesian formulation of the problem. The Gittins index approach is due to Gittins and Jones (1974). Duff (1995) showed how it is possible to learn Gittins indices for bandit problems through reinforcement learning. The survey by Kumar (1985) provides a good discussion of Bayesian and non-Bayesian approaches to these problems. The term *information state* comes from the literature on partially observable MDPs; see, e.g., Lovejoy (1991).

Other theoretical research focuses on the efficiency of exploration, usually expressed as how quickly an algorithm can approach an optimal decision-making policy. One way to formalize exploration efficiency is by adapting to reinforcement learning the notion of *sample complexity* for a supervised learning algorithm, which is the number of training examples the algorithm needs to attain a desired degree of accuracy in learning the target function. A definition of the sample complexity of exploration for a reinforcement learning algorithm is the number of time steps in which the algorithm does not select near-optimal actions (Kakade, 2003). Li (2012) discusses this and several other approaches in a survey of theoretical approaches to exploration efficiency in reinforcement learning. A thorough modern treatment of Thompson sampling is provided by Russo, Van Roy, Kazerouni, Osband, and Wen (2018).