2 Evaluative Feedback

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 trial-and-error 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. Evaluative feedback is the basis of methods for function optimization, including evolutionary methods. 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. There are also interesting cases between these two in which evaluation and instruction blend together.

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 *non-associative* 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 will enable us to see most clearly how evaluative feedback differs from, and yet can be combined with, instructive feedback.

The particular non-associative, evaluative-feedback problem that we explore is a simple version of the **n** -armed bandit problem. We use this problem to introduce a number of basic learning algorithms that we extend in later chapters to apply to the full reinforcement learning problem. We end this chapter by taking 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.

Next Up Previous

Next: 2.1 An n-armed Bandit **Up:** Contents **Previous:** 1 Introduction

---

Richard Sutton
Sat May 31 12:02:11 EDT 1997

2.1 An n-armed Bandit Problem

Consider the following learning problem. You are faced repeatedly with a choice among **n** different options, or actions. After each choice you receive a numerical reward chosen from a stationary probability distribution dependent on the action you selected. Your objective is to maximize the expected total reward over some time period, for example, over 1000 action selections. Each action selection is called a *play*.

This is the original form of the ***n*** *-armed bandit problem*, so named by analogy to a slot machine, or ``one-armed bandit,'' except with **n** 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 plays you are to maximize your winnings by concentrating your plays on the best levers. Another analogy is that of a doctor choosing between experimental treatments for a series of seriously ill patients. Each play is a treatment selection, and each reward is the survival or well being of the patient. Today the term ``**n** -armed bandit problem'' is often 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 **n** -armed bandit problem, each action has an expected or mean reward given that that action is selected; let us call this the *value* of that action. If you knew the value of each action, then it would be trivial to solve the **n** -armed bandit problem: you would simply always select the action with highest value. We assume that you do not know the action values with certainty, although you may have estimates.

If you maintain estimates of the action values, then at any time there is at least one action whose estimated value is greatest. We call this a *greedy* action. If you select a greedy action, we say that you are *exploiting* your current knowledge of the value of the actions. If instead you select one of the non-greedy actions, then we say you are *exploring* because this enables you to improve your estimate of the non-greedy action's value. Exploitation is the right thing to do to maximize expected reward on the one play, but exploration may produce greater total reward in the long run. For example, suppose the 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 is probably actually better than the greedy action, only you don't know which. If you have many plays yet to make, then it may be better to explore the non-greedy 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*. Because it is not possible to both explore and 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 exploit or explore depends in a complex way on the precise values of the estimates, uncertainties, and the number of remaining plays. There are many sophisticated methods for balancing exploration and exploitation for particular mathematical formulations of the **n** -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 exploitation and exploration in a sophisticated way; we worry only about balancing them at all. In this chapter we present several simple balancing methods for the **n** -armed bandit problem and show that they work much better than methods that always exploit. In addition, we point out that supervised learning methods (or rather the methods closest to supervised learning methods when adapted to this problem) perform poorly on this problem because they do not balance exploitation and exploration at all. The need to balance exploration and exploitation is a distinctive challenge that arises in reinforcement learning; the simplicity of the **n** -armed bandit problem enables us to show this in a particularly clear form.

Next Up Previous
Next: 2.2 Action-Value Methods **Up:** 2 Evaluative Feedback **Previous:** 2 Evaluative Feedback

2.2 Action-Value Methods

We begin by looking more closely at some simple algorithms for estimating the value of actions and for using the estimates to make action-selection decisions. In this chapter, we denote the true (actual) value of action **a** as $Q^*(a)$, and the estimated value after **t** plays as $Q_t(a)$. Recall that the true value of an action is the mean reward given that the action is selected. One natural way to estimate this is by averaging the rewards actually received when the action was selected. In other words, if, after **t** decisions, action **a** has been chosen k_a times, yielding rewards $r_1, r_2, \ldots, r_{k_a}$, then its value is estimated to be

$$Q_t(a) = \frac{r_1 + r_2 + \cdots + r_{k_a}}{k_a}. \qquad (2.1)$$

If $k_a = 0$, then we define $Q_t(a)$ instead as some default value, e.g., $Q_0(a) = 0$. As $k_a \rightarrow \infty$, 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 a simple average of the sample of relevant rewards. Of course this is just one way to estimate action values, not necessarily the best one. Nevertheless, for now let us stay with this simple estimation algorithm and turn to the question of how the estimates might be used to select actions.

The simplest action selection rule is to select the action (or one of the actions) with highest estimated action value, i.e., to select on the **t**th play one of the greedy actions, a_t^*, for which $Q_{t-1}(a_t^*) = \max_a Q_{t-1}(a)$. This method 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 ϵ , instead select an action at random, uniformly, independently of the action-value estimates. We call methods using this near-greedy action selection rule ϵ *-greedy* methods. An advantage of these methods is that in the limit as the number of plays increases, every action will be sampled an infinite number of times, guaranteeing that $k_a \rightarrow \infty$ for all **a**, and 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 $1 - \epsilon$, i.e., to near certainty. These are just asymptotic guarantees, however, and say little about the practical effectiveness of the methods.

To roughly assess the relative effectiveness of the greedy and ϵ -greedy methods, we compared them numerically on a suite of test problems. This is a set of 2000 randomly generated **n** -armed bandit tasks with **n=10**. For each action, **a**, the rewards were selected from a normal (Gaussian) probability distribution with mean $Q^*(a)$ and variance **1**. The 2000 **n** -armed bandit tasks were generated by re-selecting the $Q^*(a)$ 2000 times, each according to a normal distribution with mean **0** and variance **1**. Averaging over tasks, we can plot the performance and behavior of various methods as they improve with experience over 1000 plays, as in Figure 2.1. We call this suite of test tasks the *10-armed testbed*.

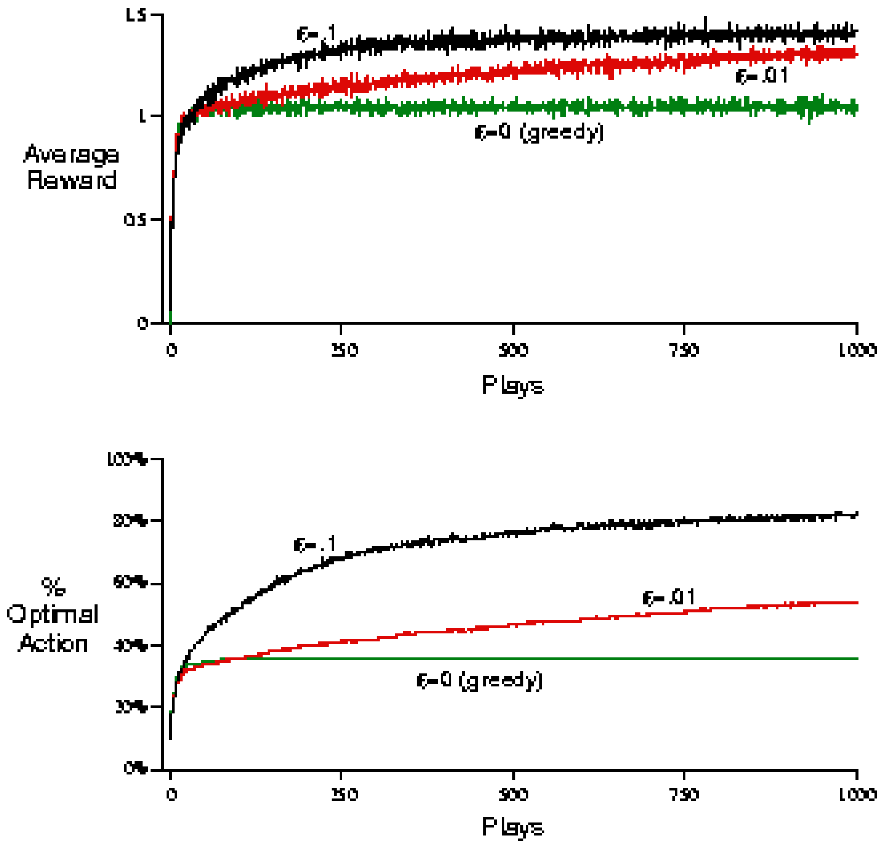


Figure 2.1: Average performance of ϵ -greedy action-value methods on 10-armed testbed.

These data are averages over 2000 tasks. All methods used sample averages as their action-value estimates.

Figure 2.1 compares a greedy method with two ϵ -greedy methods ($\epsilon = .01$ and $\epsilon = .1$), as described above, on the 10-armed testbed. Both 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 improves slightly faster than the other methods at the very beginning, but then levels off at a lower level. It achieves a reward-per-step of only about 1 compared to the best possible of about 1.55 on this testbed. The greedy method performs significantly worse in the long run because it often gets stuck performing suboptimal actions. The lower graph shows that the greedy method found the optimal action only in 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 ϵ -greedy methods eventually perform better because they continue to explore, and continue to improve their chances of recognizing the optimal action. The $\epsilon = .1$ method explores more, and usually finds the optimal action earlier, but never selects it more than 91% of the time. The $\epsilon = .01$ method improves more slowly, but eventually performs better than the $\epsilon = .1$ method by both performance measures. It is also possible to reduce ϵ over time to try to get the best of both high and low values.

The advantage of ϵ -greedy over greedy methods depends on the problem. 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 ϵ -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 problem were nonstationary, that is, that 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 non-greedy actions has not changed to become better than the greedy one. As we will see in the next few chapters, effective nonstationarity is the case most commonly encountered in reinforcement learning. Even if the underlying problem is stationary and deterministic, the learner faces a set of bandit-like decision problems each of which changes over time due to the learning process itself. Reinforcement learning problems have a strong requirement for some sort of balance between exploration and exploitation.

Exercise 2.1

In the comparison shown in Figure 2.1, which method will perform best in the long run, in terms of cumulative reward and cumulative probability of selecting the best action? How much better will it be?

Richard Sutton
Sat May 31 12:02:11 EDT 1997

2.3 Softmax Action Selection

Although ϵ -greedy action-selection is an effective and popular means of balancing exploration and exploitation in reinforcement learning, one drawback is that when it explores it chooses equally among all actions. This means that it is just as likely to choose the worst appearing action as it is to choose the next-to-best. In tasks where the worst actions are very bad, this may be unsatisfactory. The obvious solution is to vary the action probabilities as a graded function of estimated value. The greedy action is still given the highest selection probability, but all the others are ranked and weighted according to their value estimates. These are called *softmax* action selection rules. The most common softmax method uses a Gibbs, or Boltzmann, distribution. It chooses action **a** on the **t**th play with probability

$$\frac{e^{Q_{t-1}(a)/\tau}}{\sum_b e^{Q_{t-1}(b)/\tau}}, \qquad (2.2)$$

where τ is a positive parameter called the *temperature*. High temperatures cause the actions to be all (nearly) equi-probable. Low temperatures cause a greater difference in selection probability for actions that differ in their value estimates. In the limit as $\tau \rightarrow 0$, softmax action selection becomes the same as greedy action selection. Of course, the softmax effect can be produced in a large number of ways other than by a Gibbs distribution. For example, one could simply add a random number from a long-tailed distribution to each $Q_{t-1}(a)$, and then pick the action whose sum was largest.

Whether softmax action selection or ϵ -greedy action selection is better is unclear and may depend on the task and on human factors. Both methods have only one parameter that must be set. Most people find it easier to set the ϵ \ parameter with confidence; setting τ requires knowledge of the likely action values, and of powers of **e**. We know of no careful comparative studies of these two simple action-selection rules.

Exercise 2.2 (programming)

How does the softmax action selection method (using the Gibbs distribution) fare on the 10-armed testbed? Implement the method and run it at several temperatures to produce graphs similar to those in Figure 2.1. To verify your code, first implement the ϵ -greedy methods and reproduce some specific aspect of the results in Figure 2.1.

Exercise 2.3*

Show that in the case of two actions, the softmax operation using the Gibbs distribution becomes the logistic, or sigmoid, function commonly used in artificial neural networks. What effect does the temperature parameter τ have on the function?

Next Up Previous

Next: 2.4 Evaluation versus Instruction **Up:** 2 Evaluative Feedback **Previous:** 2.2 Action-Value Methods

---

Richard Sutton
Sat May 31 12:02:11 EDT 1997

2.4 Evaluation versus Instruction

The **n** -armed bandit problem we considered above is a case in which the feedback is purely evaluative. The reward received after each action gives some information about how good the action was, but it says nothing at all about whether the action was correct or incorrect, that is, whether it was a best action or not. Here, correctness is a relative property of actions that can only be determined by trying them all and comparing their rewards. In this sense the problem is inherently one requiring explicit search among the alternative actions. You have to perform some form of the generate-and-test method whereby you try actions, observe the outcomes, and selectively retain those that are the most effective. This is learning by selection, in contrast to learning by instruction, and all reinforcement learning methods have to use it in one form or another.

This contrasts sharply with supervised-learning, where the feedback from the environment directly indicates what the correct action should have been. In this case there is no need to search: whatever action you try, you will be told what the right one would have been. There is no need to try a variety of actions; the instructive ``feedback'' is typically *independent* of the action selected (so is not really feedback at all). It might still be necessary to search in the parameter space of the supervised learning system (e.g., the weight-space of a neural network), but searching in the space of actions is not required.

Of course, supervised learning is usually applied to problems that are much more complex in some ways than the **n** -armed bandit. In supervised learning there is not one situation in which action is taken, but a large set of different situations, each of which must be responded to correctly. The main problem facing a supervised learning system is to construct a mapping from the situations to actions that mimics the correct actions specified by the environment and that generalizes correctly to new situations. A supervised learning system cannot be said to learn to control its environment because it follows, rather than influences, the instructive information it receives. Instead of trying to make its environment behave in a certain way, it tries to make *itself* behave as instructed by its environment.

Focusing on the special case of a single situation that is encountered repeatedly helps make plain the distinction between evaluation and instruction. Suppose there are 100 possible actions and you select action number 32. Evaluative feedback would give you a score, say 0.9, for that action whereas instructive training information would say what other action, say action number 67, would actually have been correct. The latter is clearly much more informative training information. Even if instructional information is noisy, it is still more informative than evaluative feedback. It is always true that a *single* instruction can be used to advantage to direct changes in the action selection rule, whereas evaluative feedback must be compared with that of other actions before any inferences can be made about action selection.

The difference between evaluative feedback and instructive information remains significant even if there are only two actions and two possible rewards. For these *binary bandit* tasks, let us call the two rewards *success* and *failure*. If you received success, then you might reasonably infer that whatever action you selected was correct, and if you received failure, then you might infer that whatever action you did *not* select was correct. You could then keep a tally of how often each action was (inferred to be) correct and select the action that was correct most often. Let us call this the *supervised* algorithm because it corresponds most closely to what supervised learning methods might do in the case of only a single input pattern. If the rewards are deterministic, then the inferences of the supervised algorithm are all correct and it performs excellently. If the rewards are stochastic, then the picture is more complicated.

In the stochastic case, a particular binary bandit task is defined by two numbers, the probabilities of success for each possible action. The space of all possible tasks is thus a unit square, as shown in Figure 2.2. The upper-left and lower-right quadrants correspond to relatively easy tasks for which the supervised algorithm would work well. For these, the probability of success for the better action is greater than a half and the probability of success for the poorer action is less than a half. For these tasks, the action inferred to be correct as described above will actually be the correct action more than half the time.

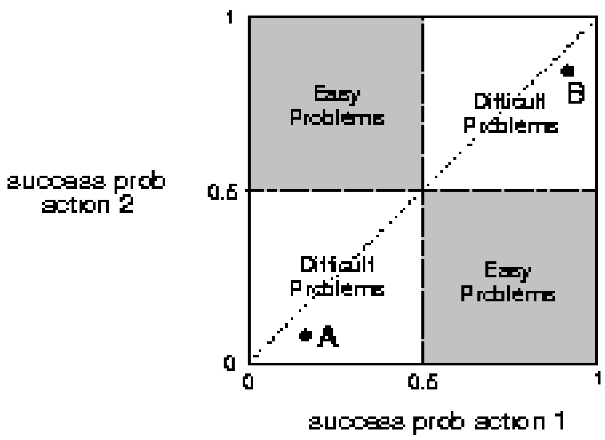
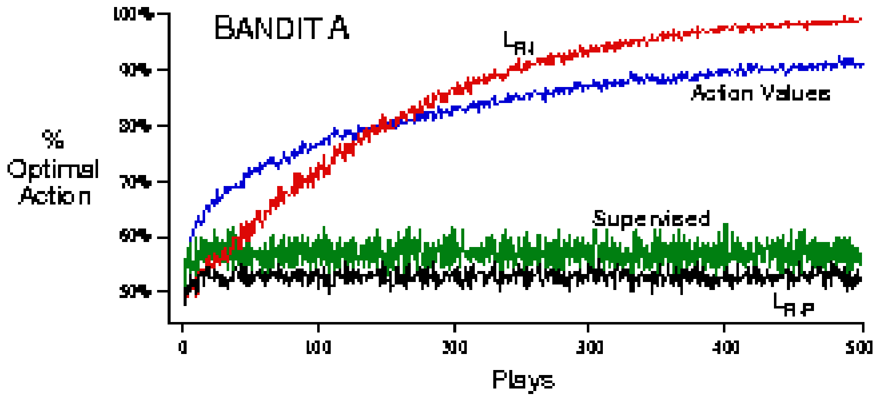


Figure 2.2: The easy and difficult regions in the space of all binary bandit tasks.

However, binary bandit tasks in the other two quadrants of Figure 2.2 are more difficult and cannot be solved effectively by the supervised algorithm. For example, consider a task with success probabilities .1 and .2, corresponding to point A in the lower-left difficult quadrant of Figure 2.2. Because both actions produce failure at least 80% of the time, any method that takes failure as an indication that the other action was correct will oscillate between the two actions, never settling on the better one. Now consider a task with success probabilities .8 and .9, corresponding to point B in the upper-right difficult quadrant of Figure 2.2. In this case both actions produce success almost all the time. Any method that takes success as an indication of correctness can easily become stuck selecting the wrong action.

Figure 2.3 shows the average behavior of the supervised algorithm and several other algorithms on the binary bandit tasks corresponding to points A and B. For comparison, also shown is the behavior of an ϵ -greedy action-value method ($\epsilon = 0.1$) as described in Section 2.2. In both tasks, the supervised algorithm learns to select the better action only slightly more than half the time.



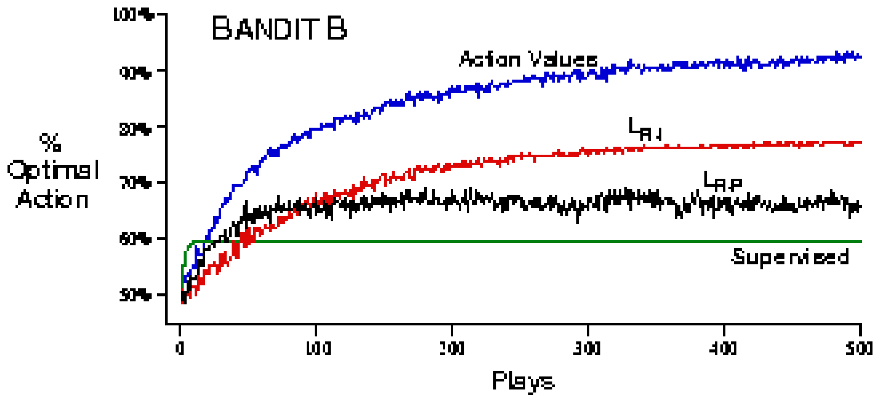


Figure 2.3: Performance of selected algorithms on the binary bandit tasks corresponding to points A and B in Figure 2.2. These data are averages over 2000 runs.

The graphs in Figure 2.3 also show the average behavior of two other algorithms, known as L_{R-I} and L_{R-P} . These are classical methods from the field of *learning automata* that follow a logic similar to that of the supervised algorithm. Both methods are stochastic, updating the probabilities of selecting each action, denoted $\pi_t(1)$ and $\pi_t(2)$. The L_{R-P} method infers the correct action just as the supervised algorithm does, and then adjusts its probabilities as follows. If the action inferred to be correct on play **t** was d_t, then $\pi_t(d_t)$ is incremented a fraction, α, of the way from its current value towards one:

$$\pi_{t+1}(d_t) = \pi_t(d_t) + \alpha(1 - \pi_t(d_t)). \qquad (2.3)$$

The probability of the other action is adjusted inversely, so that the two probabilities sum to one. For the results shown in Figure 2.3, $\alpha = 0.1$. The idea of L_{R-P} is very similar to that of the supervised algorithm, only it is stochastic. Rather than committing totally to the action inferred to be best, L_{R-P} gradually increases its probability.

The name L_{R-P} stands for ``linear, reward-penalty," meaning that the update (2.3) is linear in the probabilities and that the update is performed on both success (reward) plays and failure (penalty) plays. The name $L_{R-I}$ stands for ``linear, reward-inaction." This algorithm is identical to L_{R-P} except that it only updates its probabilities upon success plays; failure plays are ignored entirely. The results in Figure 2.3 show that L_{R-P} performs little, if any, better than the supervised algorithm on the binary bandit tasks corresponding to points A and B in Figure 2.2. L_{R-I} eventually performs very well on the A task, but not on the B task, and learns slowly in both cases.

Binary bandit problems are an instructive special case blending aspects of supervised and reinforcement learning problems. Because the rewards are binary, it is possible to infer something about the correct action given just a single reward. On some instances of such problems, these inferences are quite reasonable and lead to effective algorithms. In other instances, however, such inferences are less appropriate and lead to poor behavior. In bandit problems with non-binary rewards, such as the 10-armed test bandit introduced in Section 2.2, it is not at all clear how the ideas behind these inferences could be applied to produce effective algorithms. All of these are very simple problems, but already we see the need for capabilities beyond those of supervised learning methods.

Exercise 2.4

Consider a simplified supervised learning problem in which there is only one situation (input pattern) and two actions. One action, say **a**, is correct and the other, **b**, is incorrect. The instruction signal is noisy: it instructs the wrong action with probability **p**; that is, with probability **p** it says that **b** is correct. You can think of this as a binary bandit problem if you treat agreeing with the (possibly wrong) instruction signal as *success*, and disagreeing with it as *failure*. Discuss the resulting class of binary bandit problems. Is anything special about these problems? How does the supervised algorithm perform on this type of problem?

Next Up Previous

Next: 2.5 Incremental Implementation **Up:** 2 Evaluative Feedback **Previous:** 2.3 Softmax Action Selection

---

Richard Sutton
Sat May 31 12:02:11 EDT 1997

2.5 Incremental Implementation

The action-value methods we have discussed so far all estimate their action values as the sample averages of observed rewards. The obvious implementation is to maintain, for each action, **a**, a record of all the rewards that have followed the selection of that action. Then, whenever estimate of the action value is needed, it can be computed according to (2.1), which we repeat here:

$$Q_t(a) = \frac{r_1 + r_2 + \cdots + r_{k_a}}{k_a}, \qquad (2.1)$$

where $r_1, \ldots, r_{k_a}$ are all the rewards received following all selections of action **a** up until play **t**. A problem with this straightforward implementation is that its memory and computational requirements grow over time without bound. That is, each additional reward following a selection of action **a** requires more memory to store it and results in more computation being required to compute $Q_t(a)$.

As you might suspect, this is not really necessary. It is easy to devise incremental update formulas for computing averages with small, constant computation required to process each new reward. For some action, let Q_k denote the average of its first **k** rewards. Given this average and a $(k+1)$st reward, r_{k+1}, the average of all **k+1** rewards can be computed by:

$$\begin{aligned}
Q_{k+1} &= \frac{1}{k+1}\sum_{i=1}^{k+1} r_i \\
&= \frac{1}{k+1}\left[r_{k+1} + \sum_{i=1}^{n} r_i\right] \\
&= \frac{1}{k+1}\left[r_{k+1} + kQ_k + Q_k - Q_k\right] \\
&= \frac{1}{k+1}\left[r_{k+1} + (k+1)Q_k - Q_k\right] \\
&= Q_k + \frac{1}{k+1}\left[r_{k+1} - Q_k\right], \qquad (2.4)
\end{aligned}$$

which holds even for **k=0**, obtaining $Q_1 = r_1$ for arbitrary Q_0. This implementation requires memory only for Q_k and **k**, and only the small computation (2.4) for each new reward.

The update rule (2.4) is of a form that occurs frequently throughout this book. The general form is:

$$NewEstimate \leftarrow OldEstimate + StepSize\left[Target - OldEstimate\right].$$

The expression $\left[Target - OldEstimate\right]$ 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 $(k+1)$st reward.

Note that the step size used in the incremental method described above changes from time step to time step. In processing the **k**th reward for action **a**, that method uses a step-size of $\frac{1}{k}$. In this book we denote the step size by the symbol α, or, more generally, by $\alpha_k(a)$. For example, the above incremental implementation of the sample average method is described by the equation $\alpha_k(a) = \frac{1}{k_a}$. Accordingly, we sometimes use the informal shorthand $\alpha = 1/k$ to refer to this case, leaving the action dependence implicit.

Exercise 2.5

Give pseudo-code for a complete algorithm for the **n** -armed bandit problem. Use greedy action selection and incremental computation of action values with $\alpha = 1/k$ step size. Assume a function $bandit(a)$ that takes an action and returns a reward. Use arrays and variables; do not subscript anything by the time index **t**. Indicate how the action values are initialized and updated after each reward.

Next Up Previous
Next: 2.6 Tracking a Nonstationary **Up:** 2 Evaluative Feedback **Previous:** 2.4 Evaluation versus Instruction

---

Richard Sutton
Sat May 31 12:02:11 EDT 1997

2.6 Tracking a Nonstationary Problem

The averaging methods discussed so far are appropriate in a stationary environment, but not if the bandit is changing over time. As we noted earlier, we often encounter reinforcement learning problems that are effectively nonstationary. In such cases it makes sense to weight recent rewards more heavily than long past ones. One of the most popular ways of doing this is to use a fixed step size. For example, the incremental update rule (2.4) for updating an average Q_k of the k past rewards is modified to be

$$Q_{k+1} = Q_k + \alpha\left[r_{k+1} - Q_k\right] \tag{2.5}$$

where the step size, α, $0 < \alpha \leq 1$, is constant. This results in Q_k being a weighted average of past rewards and the initial estimate Q_0:

$$\begin{aligned}
Q_k &= Q_{k-1} + \alpha\left[r_k - Q_{k-1}\right] \\
&= \alpha r_k + (1-\alpha)Q_{k-1} \\
&= \alpha r_k + (1-\alpha)\alpha r_{k-1} + (1-\alpha)^2 Q_{k-2} \\
&= \alpha r_k + (1-\alpha)\alpha r_{k-1} + (1-\alpha)^2 \alpha r_{k-2} + \\
&\qquad \cdots + (1-\alpha)^{k-1}\alpha r_1 + (1-\alpha)^k Q_0 \\
&= (1-\alpha)^k Q_0 + \sum_{i=1}^{k} \alpha(1-\alpha)^{k-i} r_i \qquad (2.6)
\end{aligned}$$

We call this a weighted average because the sum of the weights is

$(1-\alpha)^k + \sum_{i=1}^{k} \alpha(1-\alpha)^{k-i} = 1$, as you can check yourself. Note that the weight, $\alpha(1-\alpha)^{k-i}$, given to the reward, r_i, depends on how many rewards ago, **k**-**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 $1-\alpha$. Accordingly, this is sometimes called an *exponential, recency-weighted average*.

Sometimes it is convenient to vary the step size from step to step. Let $\alpha_k(a)$ denote the step size used to process the reward received after the **k**th selection of action **a**. As we have noted, the choice $\alpha_k(a) = 1/k$ 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 $\{\alpha_k(a)\}$. A classic result in stochastic approximation theory gives us the conditions required to assure convergence with probability one:

$$\sum_{k=1}^{\infty} \alpha_k(a) = \infty \qquad \text{and} \qquad \sum_{k=1}^{\infty} \alpha_k^2(a) < \infty. \qquad (2.7)$$

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, $\alpha_k(a) = 1/k$, but not for the case of constant step size, $\alpha_k(a) = \alpha$. 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 norm in reinforcement learning. In addition, step-size sequences that meet the conditions (2.7) often converge very slowly or need considerable tuning in order to obtain a satisfactory convergence rate. Although step-size sequences that meet these convergence conditions are often used in theoretical work, they are seldom used in applications and empirical research.

Exercise 2.6

If the step sizes, $\alpha_k(a)$, are not constant, then the estimate $Q_k(a)$ 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?

Exercise 2.7 (programming)

Design and conduct an experiment to demonstrate the difficulties that sample-average methods have on 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. Prepare plots like Figure 2.1 for an action-value method using sample averages, incrementally computed by $\alpha = 1/k$, and another action-value method using a constant step size, $\alpha = 0.1$. Use $\epsilon = .1$ and, if necessary, runs longer than 1000 plays.

Richard Sutton
Sat May 31 12:02:11 EDT 1997

Next Up Previous
Next: 2.8 Reinforcement Comparison **Up:** 2 Evaluative Feedback **Previous:** 2.6 Tracking a Nonstationary

2.7 Optimistic Initial Values

All the methods we have discussed so far are dependent to some extent on the initial action-value estimates, $Q_0(a)$. 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 α, 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 whole 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 of encouraging 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.

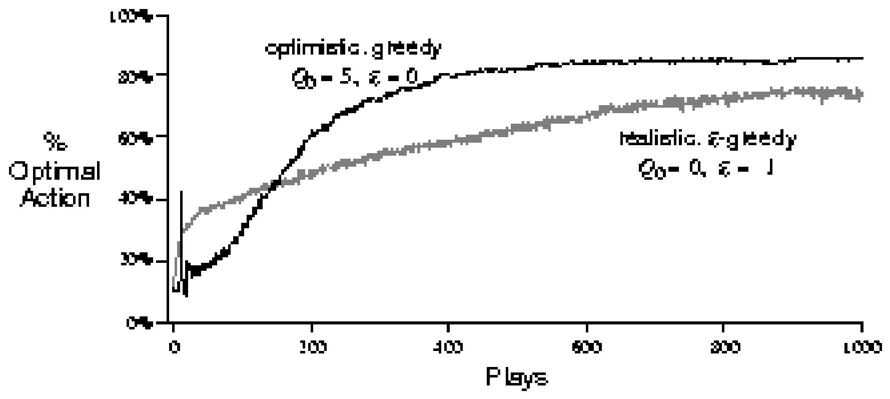


Figure 2.4: The effect of optimistic initial action-values estimates on the 10-armed

testbed. Both algorithms used constant step size, $\alpha = .1$.

Figure 2.4 shows the performance on the 10-armed bandit testbed of a greedy method using $Q_0(a) = +5$, for all **a**. For comparison, also shown is an ϵ -greedy method with $Q_0(a) = 0$. Both methods used a constant step size, $\alpha = .1$. 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 temporary. If the task changes, creating a renewed need for exploration, this method cannot help. Indeed, any method that focuses on the initial state in any special way is unlikely to help with the general non-stationary 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 with equal weights all subsequent rewards. Nevertheless, all of these methods are very simple, and one or some simple combination of them is often adequate in practice. In the rest of this book we make frequent use of many of the exploration techniques that we explore here within the simpler non-associative setting of the **n** -armed bandit problem.

Exercise 2.8

The results shown in Figure 2.4 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? What might make this method perform particularly better or worse, on average, on particular early plays?

Next Up Previous

Next: 2.8 Reinforcement Comparison **Up:** 2 Evaluative Feedback **Previous:** 2.6 Tracking a Nonstationary

Richard Sutton
Sat May 31 12:02:11 EDT 1997

2.8 Reinforcement Comparison

A central intuition underlying reinforcement learning is that actions followed by large rewards should be made more likely to recur, whereas actions followed by small rewards should be made less likely to recur. But how is the learner to know what constitutes a large or a small reward? If an action is taken and the environment returns a reward of 5, is that large or small? To make such a judgment one must compare the reward with some standard or reference level, called the *reference reward*. A natural choice for the reference reward is an average of previously received rewards. In other words, a reward is interpreted as large if it is higher than average, and small if it is lower than average. Learning methods based on this idea are called *reinforcement comparison* methods. These methods are sometimes more effective than action-value methods. They are also the precursors to actor-critic methods, an important class of methods, which we present later, for solving the full reinforcement learning problem.

Reinforcement-comparison methods typically do not maintain estimates of action values but only of an overall reward level. In order to pick among the actions, they maintain a separate measure of their preference for each action. Let us denote the preference for action **a** on play **t** by $p_t(a)$. The preferences might be used to determine action-selection probabilities according to a softmax relationship, e.g.:

$$\pi_t(a) = Pr\left\{a_t = a\right\} = \frac{e^{p_{t-1}(a)}}{\sum_b e^{p_{t-1}(b)}}, \qquad (2.8)$$

where $\pi_t(a)$ denotes the probability of selecting action **a** on the **t**th play. The reinforcement comparison idea is used in updating the action preferences. After each play, the preference for the action selected on that play, a_t, is incremented by the difference between the reward, r_t, and the reference reward, $\bar{r}_t$:

$$p_{t+1}(a_t) = p_t(a_t) + \beta(r_t - \bar{r}_t), \qquad (2.9)$$

where β is a positive step-size parameter. This equation implements the idea that high rewards should increase the probability of re-selecting the action taken and low rewards should decrease its probability.

The reference reward is an incremental average of *all* recently received rewards, whichever actions were taken. After the update (2.9), the reference reward is updated:

$$\bar{r}_{t+1} = \bar{r}_t + \alpha(r_t - \bar{r}_t), \tag{2.10}$$

where α, $0 < \alpha \leq 1$, is a step size as usual. The initial value of the reference reward, $\bar{r}_0$, can be set either optimistically, to encourage exploration, or according to prior knowledge. The initial values of the action preferences can simply all be set to zero. Constant α is a good choice here because the distribution of rewards is changing over time as action selection improves. We see here the first case in which the learning problem is effectively nonstationary even though the underlying problem is stationary.

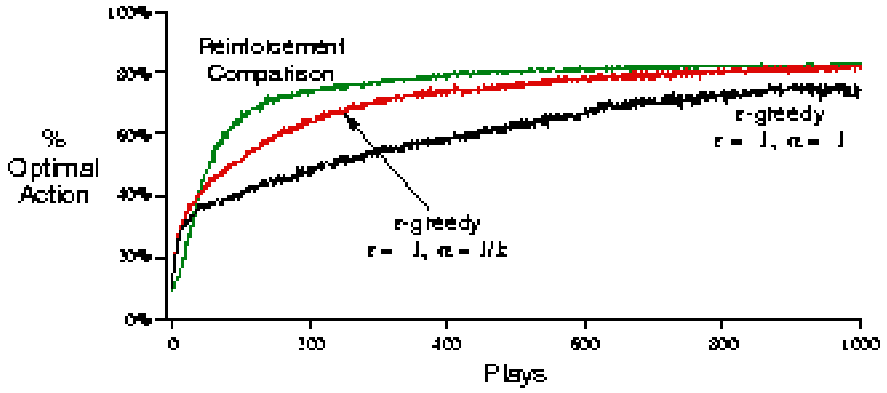


Figure 2.5: Reinforcement comparison methods versus action-value methods on the 10-armed testbed.

Reinforcement-comparison methods can be very effective, sometimes performing even better than action-value methods. Figure 2.5 shows the performance of the above algorithm ($\alpha = .1$) on the 10-armed testbed. The performances of ϵ -greedy ($\epsilon = 0.1$) action-value methods with $\alpha = 0.1$ and $\alpha = 1/k$ are also shown for comparison.

Exercise 2.9

The softmax action-selection rule given for reinforcement-comparison methods (2.8) lacks the temperature parameter, τ, used in the earlier softmax equation (2.2). Why do you think this was done? Has any important flexibility been lost here by omitting τ?

Exercise 2.10

The reinforcement-comparison methods described here have two step-size parameters, α and β. Could we, in general, reduce this to just one parameter by choosing $\alpha = \beta$? What would be lost by doing this?

Exercise 2.11 (programming)

Suppose the initial reference reward, $\bar{r}$, is far too low. Whatever action is selected first will then probably increase in its probability of selection. Thus it is likely to be selected again, and increased in probability again. In this way an early action that is no better than any other could crowd out all other actions for a long time. To counteract this effect, it is common to add a factor of $(1 - \pi_t(a_t))$ to the increment in (2.9). Design and implement an experiment to determine whether or not this really improves the performance of the algorithm.

Next Up Previous
Next: 2.9 Pursuit Methods **Up:** 2 Evaluative Feedback **Previous:** 2.7 Optimistic Initial Values

---

Richard Sutton
Sat May 31 12:02:11 EDT 1997

2.9 Pursuit Methods

Another class of effective learning methods for the **n** -armed bandit problem are *pursuit* methods. Pursuit methods maintain both action-value estimates *and* action preferences, with the preferences continually ``pursuing'' the action that is greedy according to the current action-value estimates. In the simplest pursuit method, the action preferences, $p_t(a)$, are the probabilities with which each action, **a**, is selected on play **t**.

Just before selecting the action, the probabilities are updated so as to make the greedy action more likely to be selected. Suppose $a_t^* = \arg\max_a Q_{t-1}(a)$ is the greedy action (or a random sample from the greedy actions if there are more than one) just prior to selecting action a_t. Then the probability of selecting $a_t = a_t^*$ is incremented a fraction, β, of the way toward one:

$$\pi_t(a_t^*) = \pi_{t-1}(a_t^*) + \beta(1 - \pi_{t-1}(a_t^*)), \qquad (2.11)$$

while the probabilities of selecting the other actions are decremented toward zero:

$$\pi_t(a) = \pi_{t-1}(a) + \beta(0 - \pi_{t-1}(a)), \qquad \text{for all } a \neq a_t^*. \qquad (2.12)$$

After a_t is taken and a new reward observed, the action value, $Q_t(a_t)$, is updated in one of the ways discussed in the preceding sections, e.g., to be a sample average of the observed rewards, using (2.1).

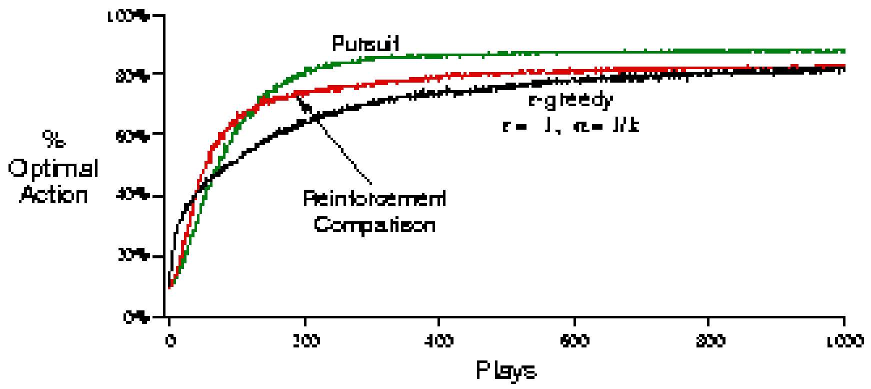


Figure 2.6: Performance of the pursuit method vis-a-vis action-value and reinforcement-comparison methods on the 10-armed testbed.

Figure 2.6 shows the performance of the pursuit algorithm described above when the action values were estimated using sample averages (incrementally computed using $\alpha = 1/k$). In these results, the initial action probabilities were $\pi_0(a) = 1/n$, for all **a** and the parameter β was .01. For comparison, we also show the performance of an ϵ -greedy method ($\epsilon = .1$) with action values also estimated using sample averages. The performance of the reinforcement-comparison algorithm from the previous section is also shown. Although the pursuit algorithm performs the best of these three on this task at these parameter settings, the ordering could well be different in other cases. All three of these methods appear to have their uses and advantages.

Exercise 2.12

An ϵ -greedy method always selects a random action on a fraction, ϵ , of the time steps. How about the pursuit algorithm? Will it eventually select the optimal action with probability approaching certainty?

Exercise 2.13

For many of the problems we will encounter later in this book it is not feasible to directly update action probabilities. To use pursuit methods in these cases it is necessary to modify them to use action preferences that are not probabilities, but which determine action probabilities according to a softmax relationship such as the Gibbs distribution (2.8). How can the pursuit algorithm described above be modified to be used in this way? Specify a complete algorithm, including the equations for action-values, preferences, and probabilities at each play.

Exercise 2.14 **(programming)**

How well does the algorithm you proposed in

Exercise 2.13

perform? Design and run an experiment assessing the performance of your method. Discuss the role of parameter value settings in your experiment.

Exercise 2.15

The pursuit algorithm described above is suited only for stationary environments because the action probabilities converge, albeit slowly, to certainty. How could you combine the pursuit idea with the ϵ-greedy idea to obtain a method with performance close to that of the pursuit algorithm, but which always continues to explore to some small degree?

Richard Sutton
Sat May 31 12:02:11 EDT 1997

2.10 Associative Search

So far in this chapter we have considered only non-associative 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 non-associative tasks extend to the associative setting.

As an example, suppose there are several different **n** -armed bandit tasks, and that on each play you confront one of these chosen at random. Thus, the bandit task changes randomly from play to play. This would appear to you as a single, nonstationary **n** -armed bandit task, whose true action values change randomly from play to play. 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, e.g., if red, play arm 1; if green, play 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 in the form of *search* for the best actions and *association* of these actions with the situations in which they are best. Associative search tasks are intermediate between the **n** -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 the **n** -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 2.16

Suppose you face a binary bandit task whose true action values change randomly from play to play. Specifically, suppose that for any play the true values of actions **1** and **2** are respectively .1 and .2 with probability .5 (case A), and .9 and .8 with probability .5 (case B). If you are not able to tell which case you face at any play, what is the best expectation of success you can achieve and how should you behave to achieve it? Now suppose that on each play you are told if 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?

Next Up Previous

Next: 2.11 Conclusion **Up:** 2 Evaluative Feedback **Previous:** 2.9 Pursuit Methods

Richard Sutton
Sat May 31 12:02:11 EDT 1997

2.11 Conclusion

We have presented in this chapter some simple ways of balancing exploration and exploitation. The ϵ -greedy methods simply choose randomly a small fraction of the time, the softmax methods grade their action probabilities according to the current action-value estimates, and the pursuit methods just keep taking steps toward the current greedy action. Are these simple methods really the best we can do in terms of practically useful algorithms? So far, the answer appears to be ``yes''. 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. We conclude this chapter with a brief look at some of the current ideas that, while not as yet practically useful, may point the way toward better solutions.

One promising idea is to use estimates of the uncertainty of the action-value estimates to direct and encourage exploration. For example, suppose there are two actions estimated to have values slightly less than that of the greedy action, but which differ greatly in their degree of uncertainty. One estimate is very certain; perhaps that action has been tried many times and many rewards have been observed. The uncertainty for this action value is so low that its true value is very unlikely to be higher than the value of the greedy action. The other action is known less well, and the estimate of its value is very uncertain. The true value of this action could easily be better than that of the greedy action. Obviously, it makes more sense to explore the second action than the first.

This line of thought leads to *interval estimation* methods. These methods estimate for each action the $(1-\epsilon)$ confidence interval of the action value. That is, rather than learning that the action value is approximately 10, they learn that it is between 9 and 11 with 95% confidence. The action selected is then the action whose confidence interval has the highest upper limit. This encourages exploration of actions that are uncertain *and* have a chance of ultimately being the best action. In some cases one can obtain guarantees that the optimal action has been found with confidence equal to the confidence factor (e.g., the 95%). Interval estimation methods are problematic in practice because of the complexity of the statistical methods used to estimate the confidence intervals. Moreover, the underlying statistical assumptions required by these methods are often not satisfied. Nevertheless, the idea of using confidence intervals, or some other measure of uncertainty, to encourage exploration of particular actions is sound and appealing.

There is also a well-known algorithm for computing the Bayes optimal way to balance exploration and exploitation. This method is computationally intractable when done exactly, but there may be efficient ways to approximate it. In this optimal method we assume that we know the distribution of problem instances, i.e., the probability of each possible set of true action values. Given any action selection, we can then compute the probability of each possible immediate reward and the resultant posterior probability distribution over action values. This evolving distribution becomes the *information state* of the problem. Given a horizon, say 1000 plays, one can consider all possible actions, all possible resulting rewards, all possible next actions, all next rewards, and so on for all 1000 plays. 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 are only two actions and two rewards the tree will have 2^{2000} leaves. This approach effectively turns the bandit problem into an instance of the full reinforcement learning problem. In the end, we may be able to use reinforcement learning methods to approximate this optimal solution. But that is a topic for current research and beyond the scope of this introductory book.

The classical solution to balancing exploration and exploitation in $\mathbf{n}$ -armed bandit problems is to compute special functions called *Gittins indices*. These provide an optimal solution to a certain kind of bandit problem more general than that considered here, but which assumes the prior distribution of possible problems is known. Unfortunately, this method does not appear to generalize either in its theory or computational tractability to the full reinforcement learning problem that we consider in the rest of the book.

Richard Sutton
Sat May 31 12:02:11 EDT 1997

2.12 Bibliographical and Historical Remarks

2.1

Bandit problems have been studied in statistics, engineering, and psychology. In statistics, bandit problems they fall under the heading of the ``sequential design of experiments," having been 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 focussed on them. In psychology, bandit problems played roles in statistical learning theory (e.g., Bush and Mosteller, 1958; 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, 1976)). 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 exploitation and new information.

2.2

Action-value methods for our n-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 ϵ-greedy methods may also have been Watkins (1989, p. 187), but the idea is so simple that some earlier use seems likely.

2.3

The term *softmax* for the action selection rule (2.2) is due to Bridle (1990). This rule appears to have been first proposed by Luce (1959). The parameter τ is called temperature in simulated annealing algorithms (Kirkpatrick, Gelatt and Vecchi, 1983).

2.4

The main argument and results in this section were first presented by Sutton (1984). Further analysis of the relationship between evaluation and instruction has been presented by Barto (1985, 1991, 1992), and Barto and Anandan (1985). The unit-square representation of a binary bandit task used in Figure 2.2 has been called a contingency space in experimental psychology (e.g., Staddon, 1983).

Narendra and Thathachar (1989) provide a comprehensive treatment of modern learning automata theory and its applications. They also discuss similar algorithms from the statistical learning theory of psychology. Other methods based on converting reinforcement-learning experience into target actions were developed by Widrow, Gupta, and Maitra (1973) and by Gällmo and Asplund (1995).

2.5 and 2 .6

This material falls under the general heading of stochastic iterative algorithms, which is well covered by Bertsekas and Tsitsiklis (1996).

2.8

Reinforcement comparison methods were extensively developed by Sutton (1984) and further refined by Williams (1986, 1992), Kaelbling (1993), and Dayan (1991). These authors analyzed many variations of the idea including various eligibility terms that may significantly improve performance. Perhaps the earliest use of reinforcement comparison was by Barto, Sutton, and Brouwer (1981).

2.9

The pursuit algorithm is due to Thathachar and Sastry (1986).

2.10

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). 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 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 situations.

2.11

Interval estimation methods are due to Lai (1987) and Kaelbling (1993). 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 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). The Gittins index approach is due to Gittins and Jones (1974). Duff (1996) showed how it is possible to learn Gittins indices for bandit problems through reinforcement learning.

Next Up Previous

Next: 3 The Reinforcement Learning Problem **Up:** 2 Evaluative Feedback **Previous:** 2.11 Conclusion

Richard Sutton
Sat May 31 12:02:11 EDT 1997

...selection.

The difference between instruction and evaluation can be clarified by contrasting two types of function optimization algorithms. One type is used when information about the *gradient* of the function being minimized (or maximized) is directly available. The gradient instructs the algorithm as to how it should move in the search space. The errors used by many supervised learning algorithms are gradients (or approximate gradients). The other type of optimization algorithm uses only function values, corresponding to evaluative information, and has to actively probe the function at additional points in the search space in order to decide where to go next. Classical examples of these types of algorithms are respectively the Robbins-Monro and the Kiefer-Wolfowitz stochastic approximation algorithms (see, e.g., Kashyap, Blaydon, and Fu, 1970).

...probability.

This is actually a considerable simplification of these learning automata algorithms. For example, they are defined as well for **n>2** and often use a different step-size parameter (70#70) on success and on failure. Nevertheless, the limitations identified in this section still apply.

Richard Sutton
Sat May 31 12:02:11 EDT 1997