learns nothing until the end of the trial. More broadly, we can view direct utility estimation as searching for U in a hypothesis space that is much larger than it needs to be, in that it includes many functions that violate the Bellman equations. For this reason, the algorithm often converges very slowly.

22.2.2 Adaptive dynamic programming

Adaptive dynamic programming

An **adaptive dynamic programming** (or ADP) agent takes advantage of the constraints among the utilities of states by learning the transition model that connects them and solving the corresponding Markov decision process using dynamic programming. For a passive learning agent, this means plugging the learned transition model $P(s' \mid s, \pi(s))$ and the observed rewards $R(s, \pi(s), s')$ into Equation (22.2) to calculate the utilities of the states. As we remarked in our discussion of policy iteration in Chapter 17, these Bellman equations are linear when the policy π is fixed, so they can be solved using any linear algebra package.

Alternatively, we can adopt the approach of **modified policy iteration** (see page 578), using a simplified value iteration process to update the utility estimates after each change to the learned model. Because the model usually changes only slightly with each observation, the value iteration process can use the previous utility estimates as initial values and typically converge very quickly.

Learning the transition model is easy, because the environment is fully observable. This means that we have a supervised learning task where the input for each training example is a state–action pair, (s,a), and the output is the resulting state, s'. The transition model $P(s' \mid s,a)$ is represented as a table and it is estimated directly from the counts that are accumulated in $N_{s'|sa}$. The counts record how often state s' is reached when executing a in s. For example, in the three trials given on page 792, *Right* is executed four times in (3,3) and the resulting state is (3,2) twice and (4,3) twice, so $P((3,2) \mid (3,3), \mathit{Right})$ and $P((4,3) \mid (3,3), \mathit{Right})$ are both estimated to be $\frac{1}{2}$.

The full agent program for a passive ADP agent is shown in Figure 22.2. Its performance on the 4×3 world is shown in Figure 22.3. In terms of how quickly its value estimates improve, the ADP agent is limited only by its ability to learn the transition model. In this sense, it provides a standard against which to measure any other reinforcement learning algorithms. It is, however, intractable for large state spaces. In backgammon, for example, it would involve solving roughly 10^{20} equations in 10^{20} unknowns.

22.2.3 Temporal-difference learning

Solving the underlying MDP as in the preceding section is not the only way to bring the Bellman equations to bear on the learning problem. Another way is to use the observed transitions to adjust the utilities of the observed states so that they agree with the constraint equations. Consider, for example, the transition from (1,3) to (2,3) in the second trial on page 792. Suppose that as a result of the first trial, the utility estimates are $U^\pi(1,3) = 0.88$ and $U^\pi(2,3) = 0.96$. Now, if this transition from (1,3) to (2,3) occurred all the time, we would expect the utilities to obey the equation

$$U^\pi(1,3) = -0.04 + U^\pi(2,3),$$

so $U^\pi(1,3)$ would be 0.92. Thus, its current estimate of 0.88 might be a little low and should be increased. More generally, when a transition occurs from state s to state s' via action $\pi(s)$,