Direct utility estimation

A simple method for **direct utility estimation** was invented in the late 1950s in the area of **adaptive control theory** by Widrow and Hoff (1960). The idea is that the utility of a state is the expected total reward from that state onward (called the expected **reward-to-go**), and each trial provides a *sample* of this quantity for each state visited

For example, the first trial in the set of three given earlier provides a sample total reward of 0.72 for state (1,1), two samples of 0.76 and 0.84 for (1,2), two samples of 0.80 and 0.88 for (1,3), and so on.

Thus, at the end of each sequence, the algorithm calculates the observed reward-to-go for each state and updates the estimated utility for that state accordingly, just by keeping a running average for each state in a table.

It is clear that direct utility estimation is just an instance of supervised learning where each example has the state as input and the observed reward-to-go as output

This means that we have reduced reinforcement learning to a standard inductive learning problem

Unfortunately, it misses a very important source of information, namely, the fact that the utilities of states are not independent! *The utility of each state equals its own reward plus the expected utility of its successor states*.

$$U^\pi(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) U^\pi(s') \ .$$

TRIALS

$$(1,1)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (2,3)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (4,3)_{+1}$$
$$(1,1)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (2,3)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (3,2)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (4,3)_{+1}$$
$$(1,1)_{-.04} \rightsquigarrow (2,1)_{-.04} \rightsquigarrow (3,1)_{-.04} \rightsquigarrow (3,2)_{-.04} \rightsquigarrow (4,2)_{-1} \ .$$

By ignoring the connections between states, direct utility estimation misses opportunities for learning. For example, the second of the three trials given earlier reaches the state (3,2), which has not previously been visited. The next transition reaches (3,3), which is known from the first trial to have a high utility.

The Bellman equation suggests immediately that (3,2) is also likely to have a high utility, because it leads to (3,3), but direct utility estimation 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.

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 a dynamic programming method.

For a passive learning agent, this means plugging the learned transition model $P(s' \mid s, \pi(s))$ and the observed rewards $R(s)$ into the Bellman equations (21.2) to calculate the utilities of the states.

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) U^{\pi}(s') . \qquad (21.2)$$

Alternatively, we can adopt the approach of **modified policy iteration ,** 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 should converge quite quickly.

function PASSIVE-ADP-AGENT(*percept*) **returns** an action
  inputs: *percept*, a percept indicating the current state s' and reward signal r'
  persistent: π, a fixed policy
    mdp, an MDP with model P, rewards R, discount γ
    U, a table of utilities, initially empty
    N_{sa}, a table of frequencies for state–action pairs, initially zero
    $N_{s'|sa}$, a table of outcome frequencies given state–action pairs, initially zero
    s, a, the previous state and action, initially null

  if s' is new **then** $U[s'] \leftarrow r'$; $R[s'] \leftarrow r'$
  if s is not null **then**
    increment $N_{sa}[s, a]$ and $N_{s'|sa}[s', s, a]$
    for each t such that $N_{s'|sa}[t, s, a]$ is nonzero **do**
      $P(t \mid s, a) \leftarrow N_{s'|sa}[t, s, a] \,/\, N_{sa}[s, a]$
  $U \leftarrow$ POLICY-EVALUATION(π, U, *mdp*)
  if s'.TERMINAL? **then** $s, a \leftarrow$ null **else** $s, a \leftarrow s', \pi[s']$
  return a

Figure 21.2 A passive reinforcement learning agent based on adaptive dynamic programming. The POLICY-EVALUATION function solves the fixed-policy Bellman equations, as described on page 657.

The process of learning the model itself is easy, because the environment is fully observable. This means that we have a supervised learning task where the input is a state–action pair and the output is the resulting state. In the simplest case, we can represent the transition model as a table of probabilities. We keep track of how often each action outcome occurs and estimate the transition probability $P(s' \mid s, a)$ from the frequency with which s' is reached when executing a in s.

For example, in the three trials given Right is executed three times in (1,3) and two out of three times the resulting state is (2,3), so $P((2, 3) \mid (1, 3), \text{Right})$ is estimated to be 2/3.

Figure 21.2 is using maximum-likelihood estimation to learn the transition model;

For example, a taxi agent that didn't know about how traffic lights might ignore a red light once or twice without no ill effects and then formulate a policy to ignore red lights from then on.

There are two mathematical approaches

The first approach, **Bayesian reinforcement learning**, assumes a prior probability $\mathrm{P}(\mathrm{h})$ for each hypothesis h about what the true model is; the posterior probability $\mathrm{P}(\mathrm{h} \mid \mathbf{e})$ is obtained in the usual way by Bayes' rule given the observations to date.

Let u_h^π be the expected utility, averaged over all possible start states, obtained by executing policy π in model h. Then we have

$$\pi^* = \operatorname*{argmax}_\pi \sum_h P(h \mid \mathbf{e}) u_h^\pi \ .$$

however, then finding an optimal policy becomes considerably more difficult, because the agent must consider the effects of future observations on its beliefs about the transition model.

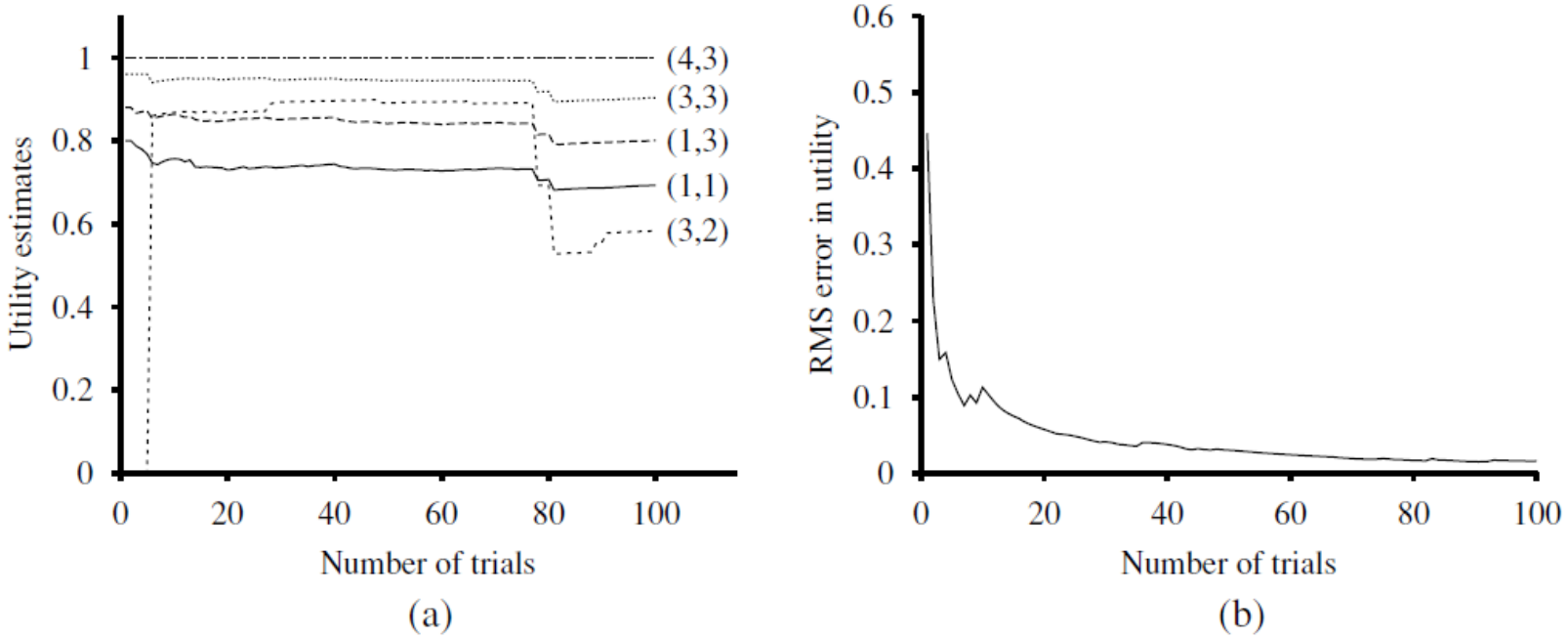

Figure 21.3 The passive ADP learning curves for the 4×3 world, given the optimal policy shown in Figure 21.1. (a) The utility estimates for a selected subset of states, as a function of the number of trials. Notice the large changes occurring around the 78th trial—this is the first time that the agent falls into the -1 terminal state at (4,2). (b) The root-mean-square error (see Appendix A) in the estimate for $U(1,1)$, averaged over 20 runs of 100 trials each.

The second approach, derived from **robust control theory**, allows for a *set* of possible models H and defines an optimal robust policy as one that gives the best outcome in the *worstcase* over H:

$$\pi^* = \operatorname*{argmax}_{\pi} \min_{h} u_h^{\pi} .$$

Often, the set H will be the set of models that exceed some likelihood threshold on P(h | **e**), so the robust and Bayesian approaches are related.

Temporal-difference learning

Solving the underlying MDP 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 .Suppose that, as a result of the first trial, the utility estimates are $U^{\pi}(1,3)=0.84$ and $U^{\pi}(2,3)=0.92$. Now, if this transition 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 0.88. Thus, its current estimate of 0.84 might be a little low and should be increased.

More generally, when a transition occurs from state s to state s', we apply the following update to U π (s):

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s)) \ .$$

Here, α is the **learning rate** parameter. Because this update rule uses the difference in utilities between successive states, it is often called the **temporal-difference**, or TD, equation.

All temporal-difference methods work by adjusting the utility estimates towards the ideal equilibrium that holds locally when the utility estimates are correct. In the case of passive learning, the equilibrium is given by Equation (21.2). Now Equation (21.3) does in fact cause the agent to reach the equilibrium given by Equation (21.2), but there is some subtlety involved.

function PASSIVE-TD-AGENT($percept$) **returns** an action
  inputs: $percept$, a percept indicating the current state s' and reward signal r'
  persistent: π, a fixed policy
    U, a table of utilities, initially empty
    N_s, a table of frequencies for states, initially zero
    s, a, r, the previous state, action, and reward, initially null

  if s' is new **then** $U[s'] \leftarrow r'$
  if s is not null **then**
    increment $N_s[s]$
    $U[s] \leftarrow U[s] + \alpha(N_s[s])(r + \gamma U[s'] - U[s])$
  if s'.TERMINAL? **then** $s, a, r \leftarrow$ null **else** $s, a, r \leftarrow s', \pi[s'], r'$
  return a

Figure 21.4 A passive reinforcement learning agent that learns utility estimates using temporal differences. The step-size function $\alpha(n)$ is chosen to ensure convergence, as described in the text.

if we change α from a fixed parameter to a function that decreases as the number of times a state has been visited increases, then $U^{\pi}(s)$ itself will converge to the correct value. This gives us the agent program shown in Figure 21.4.

Figure 21.5 illustrates the performance of the passive TD agent on the 4×3 world. It does not learn quite as fast as the ADP agent and shows much higher variability, but it is much simpler and requires much less computation per observation.

Notice that *TD does not need a transition model to perform its updates*.

One can also use a heuristic to rank the possible adjustments so as to carry PRIORITIZED out only the most significant ones. The **prioritized sweeping** heuristic prefers to make adjustments to states whose *likely* successors have just undergone a *large* adjustment in their own utility estimates.