11.3.3 Q-learning

In Q-learning and related algorithms, an agent tries to learn the optimal policy from its history of interaction with the environment. A history of an agent is a sequence of state-action-rewards:

$$\langle s_0,a_0,r_1,s_1,a_1,r_2,s_2,a_2,r_3,s_3,a_3,r_4,s_4 \ldots \rangle,$$

which means that the agent was in state s_0 and did action a_0, which resulted in it receiving reward r_1 and being in state s_1; then it did action a_1, received reward r_2, and ended up in state s_2; then it did action a_2, received reward r_3, and ended up in state s_3; and so on.

We treat this history of interaction as a sequence of experiences, where an experience is a tuple

$$\langle s,a,r,s' \rangle,$$

which means that the agent was in state s, it did action a, it received reward r, and it went into state s'. These experiences will be the data from which the agent can learn what to do. As in decision-theoretic planning, the aim is for the agent to maximize its value, which is usually the discounted reward.

Recall that $Q^*(s,a)$, where a is an action and s is a state, is the expected value (cumulative discounted reward) of doing a in state s and then following the optimal policy.

Q-learning uses temporal differences to estimate the value of $Q^*(s,a)$. In Q-learning, the agent maintains a table of Q[S,A], where S is the set of states and A is the set of actions. Q[s,a] represents its current estimate of $Q^*(s,a)$.

An experience $\langle s,a,r,s' \rangle$ provides one data point for the value of Q(s,a). The data point is that the agent received the future value of $r+ \gamma V(s')$, where $V(s') = \max_{a'} Q(s',a')$; this is the actual current reward plus the discounted estimated future value. This new data point is called a return. The agent can use the temporal difference equation (11.3.2) to update its estimate for Q(s,a):

$$Q[s,a] \leftarrow Q[s,a] + \alpha(r+ \gamma \max_{a'} Q[s',a'] - Q[s,a])$$

or, equivalently,

$$Q[s,a] \leftarrow (1-\alpha)\, Q[s,a] + \alpha(r+ \gamma \max_{a'} Q[s',a']).$$

Figure 11.10 shows the Q-learning controller. This assumes that α is fixed; if α is varying, there will be a different count for each state-action pair and the algorithm would also have to keep track of this count.

```
controller Q-learning(S,A,γ,α)
2:      Inputs
3:          S is a set of states
4:          A is a set of actions
5:          γ the discount
6:          α is the step size
7:      Local
8:          real array Q[S,A]
9:          previous state s
10:         previous action a
11:     initialize Q[S,A] arbitrarily
12:     observe current state s
13:     repeat
14:         select and carry out an action a
15:         observe reward r and state s'
16:         Q[s,a] ←Q[s,a] + α(r+ γmax_a' Q[s',a'] - Q[s,a])
17:         s ←s' until termination
```

Figure 11.10: Q-learning controller

Q-learning learns an optimal policy no matter which policy the agent is actually following (i.e., which action a it selects for any state s) as long as there is no bound on the number of times it tries an action in any state (i.e., it does not always do the same subset of actions in a state). Because it learns an optimal policy no matter which policy it is carrying out, it is called an off-policy method.

Example 11.9: Consider the domain Example 11.7, shown in Figure 11.8. Here is a sequence of □s,a,r,s'□ experiences, and the update, where γ=0.9 and α=0.2, and all of the Q-values are initialized to 0 (to two decimal points):

s	a	r	s'	Update
s_0	upC	-1	s_2	$Q[s_0,upC]=-0.2$
s_2	up	0	s_4	$Q[s_2,up]=0$
s_4	left	10	s_0	$Q[s_4,left]=2.0$
s_0	upC	-1	s_2	$Q[s_0,upC]=-0.36$
s_2	up	0	s_4	$Q[s_2,up]=0.36$
s_4	left	10	s_0	$Q[s_4,left]=3.6$
s_0	up	0	s_2	$Q[s_0,upC]=0.06$
s_2	up	-100	s_2	$Q[s_2,up]=-19.65$
s_2	up	0	s_4	$Q[s_2,up]=-15.07$
s_4	left	10	s_0	$Q[s_4,left]=4.89$

Notice how the reward of -100 is averaged in with the other rewards. After the experience of receiving the -100 reward, $Q[s_2,up]$ gets the value

$0.8\times0.36 + 0.2\times(-100+0.9\times0.36) = -19.65$

At the next step, the same action is carried out with a different outcome, and $Q[s_2,up]$ gets the value

$0.8\times-19.65 + 0.2\times(0+0.9\times3.6) = -15.07$

After more experiences going up from s_2 and not receiving the reward of -100, the large negative reward will eventually be averaged in with the positive rewards and eventually have less influence on the value of $Q[s_2,up]$, until going up in state s_2 once again receives a reward of -100.

It is instructive to consider how using α_k to average the rewards works when the earlier estimates are much worse than more recent estimates. The following example shows the effect of a sequence of deterministic actions. Note that when an action is deterministic we can use $\alpha=1$.

Example 11.10: Consider the domain Example 11.7, shown in Figure 11.8. Suppose that the agent has the experience

$\square s_0,\text{right},0,s_1,\text{upC},-1,s_3,\text{upC},-1,s_5,\text{left},0,s_4,\text{left},10,s_0\square$

and repeats this sequence of actions a number of times. (Note that a real Q-learning agent would not keep repeating the same actions, particularly when some of them look bad, but we will assume this to let us understand how Q-learning works.)

This is a trace of Q-learning described in Example 11.10.

(a) Q-learning for a deterministic sequence of actions with a separate α_k-value for each state-action pair, $\alpha_k=1/k$.

Iteration	$Q[s_0,\text{right}]$	$Q[s_1,\text{upC}]$	$Q[s_3,\text{upC}]$	$Q[s_5,\text{left}]$	$Q[s_4,\text{left}]$
1	0	-1	-1	0	10
2	0	-1	-1	4.5	10
3	0	-1	0.35	6.0	10
4	0	-0.92	1.36	6.75	10
10	0.03	0.51	4	8.1	10
100	2.54	4.12	6.82	9.5	11.34
1000	4.63	5.93	8.46	11.3	13.4
10,000	6.08	7.39	9.97	12.83	14.9
100,000	7.27	8.58	11.16	14.02	16.08
1,000,000	8.21	9.52	12.1	14.96	17.02
10,000,000	8.96	10.27	12.85	15.71	17.77
∞	11.85	13.16	15.74	18.6	20.66

(b) Q-learning for a deterministic sequence of actions with $\alpha=1$:

Iteration	$Q[s_0,\text{right}]$	$Q[s_1,\text{upC}]$	$Q[s_3,\text{upC}]$	$Q[s_5,\text{left}]$	$Q[s_4,\text{left}]$
1	0	-1	-1	0	10
2	0	-1	-1	9	10
3	0	-1	7.1	9	10
4	0	5.39	7.1	9	10
5	4.85	5.39	7.1	9	14.37
6	4.85	5.39	7.1	12.93	14.37
10	7.72	8.57	10.64	15.25	16.94
20	10.41	12.22	14.69	17.43	19.37
30	11.55	12.83	15.37	18.35	20.39
40	11.74	13.09	15.66	18.51	20.57
∞	11.85	13.16	15.74	18.6	20.66

(c) Q-values after full exploration and convergence:

Iteration	$Q[s_0,\text{right}]$	$Q[s_1,\text{upC}]$	$Q[s_3,\text{upC}]$	$Q[s_5,\text{left}]$	$Q[s_4,\text{left}]$
∞	19.5	21.14	24.08	27.87	30.97

Figure 11.11: Updates for a particular run of Q-learning

Figure 11.11 shows how the Q-values are updated though a repeated execution of this action sequence. In both of these tables, the Q-values are initialized to 0.

a. In the top run there is a separate α_k-value for each state-action pair. Notice how, in iteration 1, only the immediate rewards are updated. In iteration 2, there is a one-step backup from the positive rewards. Note that the -1 is not backed up because another action is available that has a Q-value of 0. In the third iteration, there is a two-step backup. $Q[s_3,\text{upC}]$ is updated because of the reward of 10, two steps ahead; its value is the average of its experiences: $(-1 + -1 + (-1+0.9\times6))/3$.
b. The second run is where $\alpha=1$; thus, it only takes into account the current estimate. Again, the reward is backed up one step in each iteration. In the third iteration, $Q[s_3,\text{upC}]$ is updated because of the reward of 10 two steps ahead, but with $\alpha=1$, the algorithm ignores its previous estimates and uses its new experience, $-1+0.9\times0.9$. Having $\alpha=1$ converges much faster than when $\alpha_k=1/k$, but $\alpha=1$ only converges when the actions are deterministic because $\alpha=1$ implicitly assumes that the last reward and resulting state are representative of future ones.
c. If the algorithm is run allowing the agent to explore, as is normal, some of the Q-values after convergence are shown in part (c). Note that, because there are stochastic actions, α cannot be 1 for the algorithm to converge. Note that the Q-values are larger than for the deterministic sequence of actions because these actions do not form an optimal policy.

The final policy after convergence is to do up in state s_0, upC in state s_2, up in states s_1 and s_3, and

left in states s_4 and s_5.

You can run the applet for this example that is available on the book web site. Try different initializations, and try varying α.