Q Learning

The goal of reinforcement learning is to compute an optimal policy π^*. As previously explained, the optimal policy can be easily computed from $Q^*(s,a)$. The following algorithm can be used to compute $Q^*(s,a)$.

0. Start with $\hat{Q}$ as an estimate for Q^*. For example, for all states s and actions a set $\hat{Q}(s,a) = 0$.

1. Repeat:

1.1 Select a state s and an action a.

1.2 Let $r = r(s,a)$ be the reward.

1.3 Let $s' = \delta(s,a)$ be the state one moves to from s if action a is taken.

1.4 Update:

$$\hat{Q}(s,a) = r + \gamma \max_{a'} \hat{Q}(s',a')$$

The method of selecting a state and an action at 1.1 is arbitrary. The algorithm is guaranteed to converge to the right Q^* if each (s,a) pair is visited infinitely many times.

Convergence

Let t be the iteration number. In terms of t the iteration in 1.4 can be written as:

$$\hat{Q}_{t+1}(s,a) = r + \gamma \max_{a'} \hat{Q}_t(s',a') \qquad (*)$$

Consider the following error terms:

$$e_t(s,a) = |\hat{Q}_t(s,a) - Q^*(s,a)|$$
$$e_t(s) = \max_a \ e_t(s,a)$$
$$e_t = \max_s \ e_t(s) \qquad = \max_{s,a} |\hat{Q}_t(s,a) - Q^*(s,a)|$$

At iteration t the algorithm selects s, a, computes r, s', and updates $\hat{Q}(s,a)$. The key technical lemma here states that:

$$\boxed{e_{t+1}(s,a) \leq \gamma \ e_t}$$

Proof:

$$\begin{aligned}
e_{t+1}(s,a) &= |\hat{Q}_{t+1}(s,a) - Q^*(s,a)| \\
&= |(r + \gamma(\max_{a'} \hat{Q}_t(s',a')) - (r + \gamma(\max_{a'} Q^*(s',a'))| \\
&= \gamma |\max_{a'} \hat{Q}_t(s',a') - \max_{a'} Q^*(s',a')| \\
&\leq \gamma \ \max_{a'} |\hat{Q}_t(s',a') - Q^*(s',a')| \\
&= \gamma \ \max_{a'} \ e_t(s',a') \\
&\leq \gamma \ e_t
\end{aligned}$$

The inequality between the third and the fourth line is:

$$|\max_a f(a) - \max_a g(a)| \leq \max_a |f(a) - g(a)|$$

It follows from the triangle inequality in the l_∞ norm, and can also be easily proved directly.

The value of $e_t(s,a)$ is not guaranteed to decrease as a function of t, and may even increase. But as shown below the value of e_t cannot increase. It is guaranteed to decrease by a factor of γ in a cycle where all (s,a) pairs are visited. The first observation is that for *all* pairs s, a:

$$e_{t+j}(s,a) \leq e_t \quad \text{for } j \geq 0$$

For the proof it is enough to show that it holds for $j = 1$, and then apply induction. The case $j = 1$ follows from the lemma. Now consider the relation between the error at iteration t and the error at iteration $t + \Delta$. We can make the following observations:

1. $e_{t+j} \leq e_t$ for $j = 0, 1, \ldots, \Delta$.

2. $e_{t+\Delta}(s,a) \leq \gamma\ e_t$ for all (s,a) that were visited in the Δ iterations.

3. $e_{t+\Delta} \leq \gamma\ e_t$ if all (s,a) were visited in the Δ iterations.

Conclusion: $\hat{Q}(s,a)$ approaches $Q^*(s,a)$ if each (s,a) is visited infinitely many times.

Another conclusion: An optimal policy always exists.

A practical Q learning algorithm

The online algorithm Q learning algorithm is interesting, but in most cases it is too slow to be practical. A related practical algorithm can be obtained by updating all $\hat{Q}$ values simultaneously.

Recall that the pair (s,a) corresponds to an edge, and let m be the number of edges. (It is also the number of actions.) Define the four m-vectors $\overrightarrow{Q}_t$, $\overrightarrow{Q}_{t+1}$, $\overrightarrow{R}$, $\overrightarrow{P}_{t+1}$ as follows:

$$\overrightarrow{Q}_t = \begin{pmatrix} \hat{Q}_t(s_1,a_1) \\ \vdots \\ \hat{Q}_t(s_m,a_m) \end{pmatrix}, \overrightarrow{Q}_{t+1} = \begin{pmatrix} \hat{Q}_{t+1}(s_1,a_1) \\ \vdots \\ \hat{Q}_{t+1}(s_m,a_m) \end{pmatrix}, \overrightarrow{R} = \begin{pmatrix} r(s_1,a_1) \\ \vdots \\ r(s_m,a_m) \end{pmatrix}, \overrightarrow{P}_{t+1} = \begin{pmatrix} \hat{P}_{t+1}(s_1,a_1) \\ \vdots \\ \hat{P}_{t+1}(s_m,a_m) \end{pmatrix}$$

where:

$$\hat{P}_{t+1}(s_i' a_i) = \max_{a'} \hat{Q}_t(s_i', a'), \quad s_i' = \delta(s_i, a_i)$$

Here we view $\hat{Q}$ as a vector, where each coordinate is $\hat{Q}(s,a)$. ($\hat{Q}$ has a value for each edge.) Let R be the vector corresponding to all the rewards. (It is of the same size as $\hat{Q}$.) Then the update of all $\hat{Q}$ values at once can be written as:

$$\overrightarrow{Q}_{t+1} = \overrightarrow{R} + \gamma \overrightarrow{P}_{t+1} \qquad (**)$$

This update rule is typically too aggressive. Instead, the practical rule is the following:

$$\overrightarrow{Q}_{t+1} = \overrightarrow{Q}_t + \alpha(\overrightarrow{R} + \gamma \overrightarrow{P}_{t+1} - \overrightarrow{Q}_t) \qquad (***)$$

Here α is the "learning rate" parameter that should satisfy $0 < \alpha \leq 1$.

The main practical problem with this algorithm is that m may be too big. Modern implementations use deep learning to represent $\overrightarrow{Q}$ compactly.