Reinforcement learning

The basic setup is as follows. At each time step $t = 0, 1, 2, \ldots$, the agent (i) perceives a state $s_t \in S$, (ii) takes an action $a_t \in A$, (iii) receives a reward $r_t = r(s_t, a_t)$, and (iv) moves to a new state $s_{t+1} = \delta(s_t, a_t)$. The agent's task is to learn a policy $\pi : S \to A$ that maps each state to the "best" action to take in that state. We define "best" in terms of the discounted cumulative reward

$$V^\pi(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots ,$$

where $\gamma \in [0, 1)$ is the discount factor. (Other possible reward functions include finite horizon reward $\sum_{i=0}^{h} r_{t+i}$ for horizon h, or average reward $\lim_{h\to\infty} \frac{1}{h} \sum_{i=0}^{h} r_{t+i}$.) Given such a reward function, the optimal policy is

$$\begin{aligned} \pi^* &= \arg \max_{\pi:S\to A} V^\pi(s) \quad \forall s \in S \quad \text{(Mitchell's formulation)} \\ &= \arg \max_{\pi:S\to A} (\min_{s\in S} V^\pi(s)). \end{aligned}$$

The difficulty with the above setup is that typically the agent knows neither the reward function r nor the transition function δ. However, if the agent is currently in state s, the agent can discover the value of $r(a, s)$ and $\delta(a, s)$ for any single action a by simply taking the action – and this information can be used to approximate the optimal policy π^* even without complete knowledge of r and δ, as follows. We first define the Q function as

$$Q(s, a) = r(s, a) + \gamma V^{\pi^*}(\delta(s, a)).$$

Note that for any state s, the optimal policy $\pi^*(s) = \arg\max_a Q(s, a)$, and the optimal reward is $V^{\pi^*}(s) = \max_a Q(s, a)$. Substituting for the optimal reward in the Q function, we get the recurrence

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

By approximately solving this recurrence for Q, we also get an approximation to the optimal policy as $\pi^*(s) = \arg\max_a Q(s, a)$.

We can approximate Q as follows (Q learning algorithm):

Initialize $\hat{Q}$ to the all zeros matrix.
Observe current state s_0.
For $t = 0, 1, 2, \ldots$,
. Select action a_t and execute it.
. Receive reward r_t.
. Observe new state s_{t+1}.
. Update $\hat{Q}(s_t, a_t) = r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a')$.

Under certain conditions (deterministic MDP, bounded reward, and states visited infinitely often) $\hat{Q}$ can be shown to converge to Q.

The Q learning algorithm above does not specify how to select action a_t. One possibility is to select action a in state s according to

$$P(a \mid s) = \frac{k^{\hat{Q}(s,a)}}{\sum_{a \in A} k^{\hat{Q}(s,a_j)}},$$

where $k > 0$ is a constant that controls the tradeoff between exploration and exploitation: actions with large $\hat{Q}$ values are increasingly favored as k becomes large. The value of k can start small and be increased over time so as to initially favor exploration and later favor exploitation.

If the environment is not deterministic (i.e., is a nondeterministic MDP), then we change the Q learning setup as follows. We define the objective as the expected value (over the distribution of outcomes) of discounted cumulative reward:

$$V^{\pi}(s_t) = E\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i}\right].$$

We define the optimal policy π^* and its reward V^{π^*} as before. We define the Q function as the expected value of the deterministic Q function:

$$\begin{aligned} Q(s,a) &= E[r(s,a) + \gamma V^{\pi^*}(\delta(s,a))] \\ &= E[r(s,a) + \gamma \sum_{s'} P(s'|s,a) V^{\pi^*}(s'). \end{aligned}$$

In this nondeterministic environment, the previous Q learning algorithm does not converge in general. To ensure convergence, we modify the $\hat{Q}$ update to a decaying weighted average of the current $\hat{Q}$ value and the new estimate:

. ...
. Copy $\hat{Q}_t = \hat{Q}_{t-1}$.
. Update $\hat{Q}_t(s,a) = (1-\alpha_t)\hat{Q}_{t-1}(s_t,a_t) + \alpha_t[r + \gamma \max_{a'} \hat{Q}_{t-1}(s_{t+1},a')]$.

Here the weight $\alpha_t = 1/(1 + visits_t(s,a))$ and $visits_t(s,a)$ is the total number of times this state-action pair has been visited up to and including the tth iteration.

In the Q learning algorithm's approximation to Q, we look at the current time step's (state,action) pair's true reward, and combine that with our estimate of the reward after that:

$$Q^{(1)}(s_t,a_t) = r_t + \gamma \max_{a'} \hat{Q}(s_{t+1},a').$$

Instead, we can look at more than one time step ahead in order to approximate Q. If we look two steps ahead, we get

$$Q^{(2)}(s_t, a_t) = r_t + \gamma r_{t+1} + \gamma^2 \max_{a'} \hat{Q}(s_{t+2}, a'),$$

or in general if we look n steps ahead, we get

$$Q^{(n)}(s_t, a_t) = r_t + \gamma r_{t+1} + \cdots + \gamma^{(n-1)} r_{t+n-1} + \gamma^n \max_{a'} \hat{Q}(s_{t+n}, a'),$$

One can combine these, obtaining the TD(λ) update:

$$\begin{aligned} Q^\lambda(s_t, a_t) &= (1-\lambda)\left[Q^{(1)}(s_t, a_t) + \lambda Q^{(2)}(s_t, a_t) + \lambda^2 Q^{(3)}(s_t, a_t) + \cdots\right] \\ &= r_t + \gamma\left[(1-\lambda)\max_{a'} \hat{Q}(s_t, a_t) + \lambda Q^\lambda(s_{t+1}, a_{t+1})\right]. \end{aligned}$$

Another variant is to represent the $\hat{Q}$ table using a function approximator, instead of storing it explicitly. For example, (1) one can train a neural network with backpropogation, where the (s, a) pairs are given as input and the output is the target values of $\hat{Q}$ given in the algorithms listed above. Alternatively, (2) one can train a separate neural network for each action a, where the state s is input and the output is the target value of $\hat{Q}$ for the (s, a) as before. Or (3) one can use linear regression to approximate the map from (s, a) to $\hat{Q}(s, a)$.

This kind of representation can be good because it reduces memory requirements and can allow one to generalize across states (?), but it can be bad because it can introduce additional error that may prevent convergence.

In SARSA, we use

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$