CS294-40 Learning for Robotics and Control Lecture 17 - 10/23/2008

TD, Q-learning and Sarsa

Lecturer: Pieter Abbeel *Scribe: Zhang Yan*

Lecture outline

Note: Ch 7 & 8 in Sutton & Barto book

- TD (Temporal difference) learning
- Q-learning
- Sarsa (State Action Reward State Action)

1 TD

Consider the following conditions:

- w/o having a transition model
- w/o having a reward function

Still, $MDP(\mathcal{S}, \mathcal{A}, \mathcal{T}, R, \gamma)$, but here $\mathcal{T}$ and R are unknown.

TD

Policy evaluation:

$$V_\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t R(s_t) | s_0 = s, \pi]$$

In our standard dynamic programming approach, we compute $V_\pi(s)$ by iterating:

$$\forall s: \quad V(s) \leftarrow R(s) + \gamma \sum_{s'} P(s'|s,a) V(s')$$

Now we consider a stochastic version thereof:

$$\text{pick some state } s: \quad \text{sample } \; s' \sim P(s'|s, \pi(s))$$
$$V_\pi(s) \leftarrow \alpha(R(s) + \gamma V'_\pi(s)) + (1-\alpha) V_\pi(s)$$

where α is step size. A possible step size: $\alpha_k = \frac{1}{k}$ for the k'th time we perform an update.
Convergence: the stochastic version of policy evaluation converges to the true value fuction under certain assumptions. The following assumptions are sufficient conditions:

- every state visited infinitely often.
- α satisfies $\sum_{k=0}^{\infty} \alpha_k = \infty$; $\sum_{k=0}^{\infty} \alpha_k^2 < \infty$.

A standard way of proving this uses the supermartingale convergence theorem. We will not cover the proof in class.

In practice, a reason to use TD for policy evaluation could be that we do not have the transition model available. The samples are then generated by executing the policy, and performing the stochastic value function updates according to the states being visited.

TD only considers policy *evaluation.* If we are interested in finding a (near-)optimal policy, we can use TD as the policy evaluation step in *policy iteration:*

Iterate:

- Run TD to perform policy evaluation, which gives us $V_\pi(s)$, $\forall s$.
- Pick a new policy π as follows: $\pi(s) = \arg\max_a \, [R(s) + \gamma \sum_{s'} P(s'|s,a)V(s')]$

2 Q-learning

Recall the Q function:

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a)V(s')$$

We can run dynamic programming (value iteration) by performing the Bellman back-ups in terms of the Q function as follows:

$$\begin{aligned}&\text{Iterate:}\\ &\forall s,a : Q(s,a) \leftarrow R(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')\end{aligned}$$

Similar to the case of TD learning, we can run a stochastic version instead:

$$\begin{aligned}&\text{For } k = 0,1,2,\ldots\\ &\text{Sample some } s,a : Q(s,a) \leftarrow (1-\alpha_k)Q(s,a) + \alpha_k(R(s) + \gamma \sum_{s'} \max_{a'} Q(s',a'))\end{aligned}$$

The stochastic version does not require a priori knowledge of the transition probability distribution P or the reward function R; it suffices to have access to a trace. The stochastic version can be shown to converge to the true Q function under the same assumptions as for TD, namely:

- every state visited infinitely often.
- α satisfies $\sum_{k=0}^{\infty} \alpha_k = \infty$; $\sum_{k=0}^{\infty} \alpha_k^2 < \infty$.

In practice, one can ensure that every (reachable) state is visited infinitely often by using an ϵ greedy policy:

$$\begin{aligned}&\text{with probability } \epsilon : \text{choose an action at random}\\ &\text{with probability } 1-\epsilon : a = \arg\max_a Q(s,a)\end{aligned}$$

Another popular way to choose the actions:

$$\text{pick action } a \text{ with probability} \frac{\exp(-Q(s,a)/T)}{\sum_b \exp(-Q(s,b)/T)}$$

Here T is the "temperature," and it determines how greedily the actions are being chosen. A natural choice is to have the temperature decrease over time.

Note: the temperature T, and the factor α need not be the same for all states: e.g., one could have these variables depend on how often the current state s has been visited.

```
Q LEARNING
Initialize $Q(s,a)$ arbitrary ($\forall s, a$).
• For $t = 0, 1, 2, \ldots$
        Choose the action $a_t$ for the current state $s_t$. (E.g., by using an $\epsilon$ greedy policy.)
        Take action $a_t$, observe $R(s_t)$, $s_{t+1}$
        $Q(s_t, a_t) \leftarrow (1 - \alpha_t) Q(s_t, a_t) + \alpha_t [R(s) + \gamma \max_a Q(s_{t+1}, a)]$        $(*)$
```

3 Sarsa - State action reward state action

Sarsa is almost identical to Q-learning. The only difference is in the Q-function update: (*) becomes:

$$Q(s_t, a_t) \leftarrow (1 - \alpha_k) Q(s_t, a_t) + \alpha_k [R(s) + \gamma Q(s_{t+1}, a_{t+1})]$$

Here a_{t+1} is the action the agent ends up taking (which is not necessarily $\arg\max_a Q(s_{t+1}, a)$, as the policy includes randomness).

4 Discussion

TD, Q-learning, and Sarsa have the following properties:

- No need to store a model
- No need even to store reward function

Looking forward

1. Incorporate function approximation
2. $TD(\lambda)$, $Q(\lambda)$, $Sarsa(\lambda)$