Machine Learning **Spring 2021**

Lecture 15: Reinforcement Learning

Lecturer: Liwei Wang *Scribe: Mingwei Yang, Hongyu Yan*

15.1 Monte Carlo Decision Process(MDP)

Monte Carlo methods are ways of solving the reinforcement learning problem based on averaging sample returns. Here we define Monte Carlo methods only for episodic tasks. For all $s \in S$, there are two Monte Carlo (MC) methods, *first-visit MC* method and *every-visit MC* method.

First-visit MC: Estimate the value of a state as the average of the returns that have followed the first visits to the state, where a first visit is the first time during a trial that the state is visited.
Every-visit MC: Estimate the value of a state as the average of the returns that have followed all visits to the state.

It can be trivially implied that the *first-visit MC* method is a sound method, since the samples are of independently identically distributed estimate of $v^\pi(s)$ with finite variance. By the law of large numbers, the sequence of averages of these estimates converges to their expected value. Each average is itself an unbiased estimate, and the standard deviation of its error falls as $\frac{1}{\sqrt{N}}$, where N is the number of returns averaged. *Every-visit MC* is less straightforward. Surprisingly, however, the *every-visit MC* method is also a sound method.

Lemma 15.1 *Suppose $v_1, v_2 \cdots$ are random variables and have common mean. Let N take positive integers. Suppose $\mathbb{E}[v_j | N \geq j] = \mathbb{E}[v_1]$. Then*

$$\frac{\mathbb{E}\left[\sum_{j=1}^N v_j\right]}{\mathbb{E}[N]} = \mathbb{E}[v_1]$$

Proof:

$$\mathop{\mathbb{E}}_{N,v}\left[\sum_{j=1}^N v_j\right] = \sum_{n=1}^\infty \Pr[N=n]\mathbb{E}\left[\sum_{j=1}^n v_j | N=n\right] \tag{15.1}$$

$$= \sum_{n=1}^\infty \sum_{j=1}^n \Pr[N=n]\mathbb{E}[v_j|N=n] \tag{15.2}$$

$$= \sum_{j=1}^\infty \Pr[N \geq j]\mathbb{E}[v_j|N=j] \tag{15.3}$$

$$= \mathbb{E}[N] \cdot \mathbb{E}[v_j] \tag{15.4}$$

■

Theorem 15.2 *The* Every-visit MC *method is a sound method for policy evaluation.*

Proof: For trajectories $k = 1, 2 \cdots K$, let $v_{k,j}$ be the return of state s of k-th trajectory for the j-th visit. Then the *Every-visit MC* method has the estimation

$$estimation = \frac{\sum_{k=1}^{K} \sum_j v_{k,j}}{\sum_{k=1}^{K} N(k)}$$

where $N(k) = \#$ visits of s in trajectory k.
Let's define E as the limit of estimation, e.g.,

$$E := \lim_{K\to\infty} \frac{\sum_{k=1}^{K} \sum_j v_{k,j}}{\sum_{k=1}^{K} N(K)} = \lim_{K\to\infty} \frac{\frac{1}{m(K)} \sum_{k=1}^{K} \sum_j v_{k,j}}{\frac{1}{m(K)} \sum_{k=1}^{K} N(k)} = \frac{\lim_{K\to\infty} \frac{1}{m(K)} \sum_{k=1}^{K} \sum_j v_{k,j}}{\lim_{K\to\infty} \frac{1}{m(K)} \sum_{k=1}^{K} N(k)}$$

where $m(K) = \#$ trajectories in which s is visited.
To prove the method is a sound method, it is sufficient to prove

$$E = v^{\pi}(s)$$

Let N be the (random) number of visits of s in an trajectory, v_j be the (random) return of s for the j-th visit. Clearly, $v_1, v_2 \cdots$ have common mean. According to **lemma 15.1**

$$E = \frac{\mathbb{E}\left[\sum_{j=1}^{N} v_j | N \geq 1\right]}{\mathbb{E}[N|N \geq 1]} = \mathbb{E}[v_1|N \geq 1] = \mathbb{E}[v_1] = v^{\pi}(s)$$

■

Suppose our state sequence in a trajectory is $S_1, S_2, \cdots, S_t \cdots$. We need to check if S_t is not in the sequence $S_1, S_2, \cdots, S_{t-1}$ in *First-visit MC* method while we don't need to check it in *Every-visit MC* method. In practice, *Every-visit MC* is more data efficient.

15.2 Temporal-Difference Learning (TD-Learning)

15.2.1 TD(0) Algorithm

Here we present TD(0) algorithm, for evaluating a policy in the case where the environment model is unknown. The algorithm is based on Bellman's equations giving the value of a policy π:

$$\begin{aligned} v^{\pi}(s) &= \mathbb{E}[R(s,a)] + \gamma \sum_{s'} \Pr[s'|s, \pi(s)] v^{\pi}(s') \\ &= \mathbb{E}\left[R(s, \pi(s)) + \gamma \cdot v^{\pi}(s') | s\right] \end{aligned}$$

Suppose during the sampling process, we sample a new state s_{t+1} and get reward R_t when taking a action according to π in state s_t. The TD(0) algorithm updates the policy values according to the following:

$$\begin{aligned} v(s_t) &\leftarrow (1 - \alpha_t) v(s_t) + \alpha_t \left[R_t + \gamma \cdot v(s_{t+1})\right] \\ &= v(s_t) + \alpha_t \left[R_t + \gamma \cdot v(s_{t+1}) - v(s_t)\right] \end{aligned}$$

where $\alpha_t \in (0, 1)$. The term $[R_t + \gamma \cdot v(s_{t+1}) - v(s_t)]$ is called the temporal difference of v values, which justifies the name of the algorithm.

Theorem 15.3 *Assume that $\sum_{t=0}^{\infty} \alpha_t = \infty$, and $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$. Then, the TD(0) algorithm converges to $v(s)$.*

15.2.2 TD(λ) Algorithm

The idea of TD(λ) consists of using multiple steps ahead, where TD(0) only uses one step ahead. Thus, for $n > 1$ steps, we would have the update

$$v(s_t) \leftarrow v(s_t) + \alpha_t(G_t^{(n)} - v(s_t))$$

where $G_t^{(n)}$ is defined by

$$G_t^{(n)} = R_t + \gamma R_{t+1} + \cdots + \gamma^{n-1} R_{t+n-1} + \gamma^n v(s_{t+n})$$

Comparing to TD(0), the normal defined TD(n) uses more information in each iteration. However it has longer delays, e.g., the current state would be updated only after n steps. Moreover, we could only update state once a time. But TD(λ) beautifully solves those problems by using G_t^{λ} instead of $G_t^{(n)}$, where

$$G_t^{\lambda} = (1-\lambda) \sum_{n=0}^{\infty} \lambda^n G_t^{(n)}, \quad \lambda \in [0,1]$$

is based on the geometric distribution over all $G_t^{(n)}$. Thus, the main update becomes

$$v(s_t) \leftarrow v(s_t) + \alpha_t(G_t^{\lambda} - v(s_t))$$

and we get the main steps of TD(λ) algorithm:

1. For all $s \in S$, $E_0(s) = 0$, $E_t(s) = \gamma\lambda E_{t-1}(s) + I[s_t = s]$.
2. $\delta_t \leftarrow R_t + \gamma \cdot v(s_{t+1}) - v(s_t)$.
3. For all $s \in S$, $v(s) \leftarrow v(s) + \alpha_t \cdot \delta_t E_t(s)$.

15.3 Q-learning

To find the optimal policy in the unknown model, e.g., the transition and reward probabilities are unknown, we can't directly apply iteration based on the Bellman Optimal Equation. Note that the optimal policy or policy value can be straightforwardly derived from state-action value function Q via: $\pi^*(s) = \arg\max_{a \in A} Q(s,a)$ and $v^*(s) = \max_{a \in A} Q(s,a)$.

The Q-learning algorithm is based on the equations giving the optimal state-action function Q^*:

$$\begin{aligned} Q^*(s,a) &= \mathbb{E}[R(s,a)] + \gamma \sum_{s' \in S} \Pr[s'|s,a] v^*(s') \\ &= \mathbb{E}_{s'}\left[R(s,a) + \gamma \max_{a \in A} Q^*(s',a)\right] \end{aligned}$$

Similar to TD-learning, we update $Q(s, a)$ with real experience as follows:

$$
\begin{aligned}
Q(s_t, a_t) &\leftarrow (1-\alpha_t)Q(s_t, a_t) + \alpha_t \left[R(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a')\right] \\
&= Q(s_t, a_t) + \alpha_t \left[R(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\right]
\end{aligned}
$$

It's remain to specify how to choose actions. Note that random selection is not a good idea, since it doesn't update the current best Q_t frequently, especially when there are a lot of actions. On the other hand, a bad initial value can invalidate the "Choose only the current $\arg\max_a Q_t(s, a)$" method, which can be viewed as keeping doing exploitation but ignoring exploration.

15.3.1 Action Selection Policy

To select actions, a standard choice in reinforcement learning is the so called *ϵ-greedy policy*, which consists of selecting with probability $(1-\epsilon)$ the greedy action from state s, that is, $\arg\max_{a\in A} Q_t(s, a)$, and with probability ϵ a random action from s, for some $\epsilon \in (0, 1)$. Thus method ensures that if enough trials are done, each action will be tried an infinite number of times, thus ensuring optimal action are discoverd.

We generalize the different types of the action selection policies into on-policy learning and off-policy learning.

15.4 On-Policy and Off-Policy Learning

Reinforcement learning algorithms include two components: a *learning policy*, which determines the action to take, and an *update rule*, which defines the new estimate of the optimal value function.

For an *off-policy learning algorithm*, the update rule does not necessarily depend on the learning policy. Q-learning is an off-policy algorithm since its update rule is based on the max operator and the comparison of all possible actions a', yet its learning policy is based on the ϵ-greedy policy.

On-Policy learning algorithms are the algorithms that evaluate and improve the same policy which is being used to select actions. Some examples of On-Policy algorithms are Policy Iteration, Value Iteration and Monte Carlo for On-policy.

References

[1] Mehryar Mohri and Afshin Rostamizadeh and Ameet Talwalkar. The Foundation of Machine Learning.

[2] Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction.