Reinforcement Learning

Sunmook Choi `felixchoi@korea.ac.kr`

August 28, 2023

1 Introduction

We have learned a method, dynamic programming, to solve Markov decision processes. Markov decision processes provide a formal framework for modeling decision-making problems in which the optimal decision depends on the current state of the environment. Dynamic programming approximates optimal value functions and policies in iterative ways to find optimal decisions.

However, a huge assumption is required to use dynamic programming method. That is, the dynamics $p(s', r|s, a)$ must be known in advance. Since the dynamics define markov decision processes (MDPs) (a model of the environment), dynamic programming is called *model-based* programming. In most cases, however, the dynamics of an environment is not known. Therefore, to solve MDPs without the knowledge of the underlying MDPs, we instead use a method of 'learning', which is also called *model-free* learning. In this section, we will figure out reinforcement learning as a type of model-free learning.

Reinforcement learning (RL) is a method to solve MDPs without being known the complete information of an environment. Recall that we say MDPs are solved if an optimal value function is found. In dynamic programming method, an optimal value function v_* is estimated by V^* using iterative algorithms based on Bellman equations. With an optimal value function, an optimal policy is found by the following equation:

$$\pi^*(s) \approx \arg\max_a Q^*(s,a) \quad \text{and} \quad Q^*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma V^*(s')] \tag{1.1}$$

However, being unknown the dynamics $p(s', r|s, a)$ of an environment changes the whole algorithm. First, the exact Bellman equations cannot be implemented in such situations with incomplete information. Second, an optimal policy cannot be found from estimated $V^*(s)$ because it also requires the dynamics. Therefore, we should estimate optimal value functions or optimal policies in different perspectives.

As we do not know the complete information of an environment, RL methods only require simulated experiences (samples or observations) from an environment to learn optimal policies. In the following methods, Monte Carlo method and Temporal Difference methods, an optimal action-value function is estimated from the samples. Such methods are under a principle of generalized policy iteration (GPI), which alternatively proceeds policy evaluation and policy improvement repeatedly.

Exploration and Exploitation Exploration and exploitation are two conflicting goals in reinforcement learning. Exploitation means taking the action that is currently believed to be the best based on the current knowledge. We have been discussed about exploitation by using greedy policy. On the other hand, exploration means taking a non-greedy action to gain more information about the environment. In other words, exploitation is the act of making the best decision based on current information, and exploration is the act of gathering more information to make better decisions in the future. A good balance between exploration and exploitation is required to find the optimal policy and value function. One of the most popular exploration strategies is the ϵ-greedy strategy, which selects the greedy action with probability $1-\epsilon$ and selects a random action with probability ϵ for small $\epsilon \in (0,1)$.

2 Monte Carlo Method

Monte Carlo method is a way of solving a given MDP, that is, finding the optimal action-value function of an MDP, based on averaging sample returns. To ensure the availability of well-defined returns, we define Monte Carlo methods only for episodic tasks. In the policy evaluation phase, the method samples experiences from an environment, and it averages returns to estimate the action-value function with the current policy. In the policy improvement phase, the method updates the policy based on the so far estimated action-value function.

One thing we should notice is that, in the policy evaluation phase, we try to estimate $Q^\pi(s,a)$ corresponding to a fixed policy π. It is necessary to estimate the action-value function $Q^\pi(s,a)$ instead of the state-value function $V^\pi(s)$ because we improve our policy π into π' as below:

$$\pi'(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)[r+\gamma V^\pi(s')] = \arg\max_a Q^\pi(s,a) \tag{2.1}$$

When we are using RL, we are in a situation without knowing the dynamics of given environment. Therefore, we cannot improve the policy by estimating state-value function $V^\pi(s)$. For this reason, action-value functions are estimated in various manners.

2.1 Policy Evaluation (Monte Carlo Prediction)

Recall the definition of the action-value function:

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \tag{2.2}$$

An obvious way to estimate $q_\pi(s,a)$ is to average the returns observed after visits to that state-action pair (s,a), following the idea of Monte Carlo methods. In particular, we will try to estimate $q_\pi(s,a)$ given a set of episodes obtained by following π and passing through (s,a). Each occurrence of state-action pair (s,a) in an episode is called a *visit* to (s,a). Possibly, the pair (s,a) may be visited more than one times in the same episode; let us call the first time it is visited in an episode the *first visit* to (s,a). The *first-visit MC method* estimates $q_\pi(s,a)$ as the average of the returns following first visits to (s,a), whereas the *every-visit MC method* averages the returns following all visits to (s,a). Practically, we can estimate $q_\pi(s,a)$ by following the steps below.

At the first (or every) time step t that the state-action pair (s,a) was visited in an episode,

1. increment count: $n(s,a) \leftarrow n(s,a) + 1$
2. increment total return: $S(s,a) \leftarrow S(s,a) + G_t$

By accumulating the count $n(s,a)$ and total return $S(s,a)$ for all episodes experienced, the action-value $q_\pi(s,a)$ is estimated by the mean return $Q(s,a) = \frac{S(s,a)}{n(s,a)}$.

Incremental Monte Carlo Updates Equivalently, we can use the moving average trick. That is, given a sequence $x_1, x_2, \dots,$ the partial mean μ_k can be computed incrementally:

$$\mu_k = \frac{1}{k}\sum_{i=1}^{k} x_i = \frac{1}{k}\Big(\sum_{i=1}^{k-1} x_i + x_j\Big) = \frac{1}{k}\Big((k-1)\mu_{k-1} + x_k\Big) = \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1}).$$

With this idea, we can estimate $q_\pi(s,a)$ by updating $Q(s,a)$ incrementally after an episode. For each episode $S_0, A_0, R_1, S_1, \dots S_{T-1}, A_{T-1}, R_T$ and for each state-action pair (S_t, A_t) with return G_t in the episode, we update $Q(s,a)$ as below:

$$\begin{aligned} n(S_t, A_t) &\leftarrow n(S_t, A_t) + 1 \\ Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \frac{1}{n(S_t, A_t)}\big[G_t - Q(S_t, A_t)\big] \end{aligned}$$

Constant-α Monte Carlo Updates In practice, instead of tracking the count parameter $n(s,a)$ for each pair, we may use a constant 'step-size' α instead of $\frac{1}{n(S_t,A_t)}$:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[G_t - Q(S_t, A_t)].$$

The use of constant α would get a benefit during training. Along the learning process, the Q-table gets 'smarter', so recent samples should be more important than old ones. Because of the term $(1-\alpha)Q(S_t, A_t)$, old episodes will be exponentially forgotten.

2.2 Policy Improvement (Monte Carlo Control)

In Chapter 2, we have shown that the maximization of the action-value function over actions actually provides better policy using the policy improvement theorem. In this section, we define an ϵ-greedy policy, and we will show that it is better than the older one.

Algorithm 1 Monte Carlo method

```
Initialize $Q(s,a)$ arbitrarily for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, and $Q(terminal, \cdot) = 0$.
           $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
           $\pi \leftarrow$ arbitrarily $\epsilon$-soft policy (non-empty probabilities)
 1: for each episode do
 2:     Generate an episode following (fixed) $\pi$: $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$
 3:     $G \leftarrow 0$
 4:     for $t = T-1, T-2, \dots, 1, 0$ (each step in episode) do
 5:         $G \leftarrow \gamma G + R_{t+1}$
 6:         if $(S_t, A_t)$ appears in $(S_0, A_0), \cdots, (S_{t-1}, A_{t-1})$ then      ▷ First-visit MC Prediction
 7:             break
 8:         else
 9:             Append $G$ to $Returns(S_t, A_t)$
10:             $Q(S_t, A_t) \leftarrow$ average($Returns(S_t, A_t)$)      ▷ average: incremental or constant-$\alpha$
11:         end if
12:     end for
13:     for each $S_t$ in the episode do      ▷ $\epsilon$-greedy MC control
14:         $\pi(a|S_t) \leftarrow \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(S_t)|} & \text{if } a = \arg\max_a Q(S_t, a) \\ \frac{\epsilon}{|\mathcal{A}(S_t)|} & \text{otherwise} \end{cases}$
15:     end for
16: end for
```

Definition 1 (ϵ-greedy policy). *For an action-value function q_π, the ϵ-greedy policy π' with respect to q_π is defined as below:*

$$\pi'(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}|} & \textit{if} \quad a = \arg\max_{a'} q_\pi(s, a'), \\ \frac{\epsilon}{|\mathcal{A}|} & \textit{otherwise} \end{cases} . \tag{2.3}$$

Proposition 2. *For any policy π, the ϵ-greedy policy π' with respect to q_π is always improved, i.e.,*

$$v_{\pi'}(s) \geq v_\pi(s)$$

for all $s \in \mathcal{S}$.

Proof. To show that $\pi' \geq \pi$, it suffices to show that $q_\pi(s, \pi') \geq v_\pi(s)$ for all $s \in \mathcal{S}$ by the policy improvement theorem in Chapter 2. Recall that $q_\pi(s,\pi') = \sum_a \pi'(a|s) q_\pi(s,a)$. Let $m = |\mathcal{A}|$.

$$\begin{aligned}
q_\pi(s,\pi'(s)) &= \sum_{a\in\mathcal{A}} \pi'(a|s) q_\pi(s,a) \\
&= \frac{\epsilon}{m}\sum_{a\in\mathcal{A}} q_\pi(s,a) + (1-\epsilon)\max_{a'} q_\pi(s,a') \\
&\geq \frac{\epsilon}{m}\sum_{a\in\mathcal{A}} q_\pi(s,a) + (1-\epsilon)\sum_{a\in\mathcal{A}} \frac{\pi(a|s)-\epsilon/m}{1-\epsilon} q_\pi(s,a') \qquad (\because \textstyle\sum_a \frac{\pi(a|s)-\epsilon/m}{1-\epsilon} = 1) \\
&= \sum_{a\in\mathcal{A}} \pi(a|s) q_\pi(s,a) \\
&= v_\pi(s)
\end{aligned}$$

This completes the proof. □

3 Temporal Difference Method

Recall the update equation of constant-α Monte Carlo (MC) policy evaluation:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha\big[\,G_t - Q(S_t,A_t)\,\big]$$

The MC target is the return G_t, which could be computed after an episode is finished, that is, MC methods updates $Q(s,a)$ for the pair (s,a) that occurs in the trajectory of an episode. In particular, MC methods must wait until the end of the episode to update the table, which might be considered inefficient.

Temporal difference (TD) methods also update Q-table by obtaining samples from the environment. However, unlike MC methods, TD methods estimate the action-value function by using the estimate of the value function at the next time step to update the estimate of the value function at the current time step. This *bootstrapping method* allows us to update Q-table at each time step, which makes TD method more efficient. We describe two types of TD method, Sarsa and Q-learning. They both use bootstrapping, but the underlying theory of estimating Q-table is slightly different.

3.1 Sarsa

We first state the update equation of $Q(s,a)$:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha\big[\,R_{t+1} + \gamma\, Q(S_{t+1},A_{t+1}) - Q(S_t,A_t)\,\big]$$

This update equation is indeed eligible to estimate $Q(s,a)$ because of Bellman expectation equation:

$$q_\pi(s,a) = \mathbb{E}_\pi\big[R_{t+1} + \gamma\, q_\pi(S_{t+1},A_{t+1}) \,|\, S_t = s, A_t = a\,\big]$$

However, because $q_\pi(S_{t+1},A_{t+1})$ is not known, we use the current estimate $Q(S_{t+1},A_{t+1})$ instead. When its update is based on an existing estimate, we say that it is a *bootstrapping method*. Of course, the bootstrapping method makes the TD target biased. The algorithm of Sarsa is as follows:

Algorithm 2 Sarsa

```
Parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$.
 1: for each episode do
 2:     Initialize $S$
 3:     Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
 4:     for each step in episode do
 5:         Take action $A$, observe $R$, $S'$
 6:         Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
 7:         $Q(S,A) \leftarrow Q(S,A) + \alpha\,[\,R + \gamma\, Q(S',A') - Q(S,A)\,]$
 8:         $S \leftarrow S'$; $A \leftarrow A'$
 9:         if $S$ is terminal then
10:             break
11:         end if
12:     end for
13: end for
```

Notice that the update rule uses every element of 5-tuple $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, that make up a transition from one state-action pair to the next. This 5-tuple give rise to the name "SARSA" for the algorithm. Notice that the action A_{t+1} in the target $R_{t+1} + \gamma\, Q(S_{t+1}, A_{t+1})$ is chosen by the same policy π which also chose the action A_t (and we call it *on-policy method*).

The quantity $R_{t+1} + \gamma\, Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$ is a sort of error, measuring the difference between the estimated value $Q(S_t, A_t)$ and the better estimate $R_{t+1} + \gamma\, Q(S_{t+1}, A_{t+1})$. This quantity is called the TD error, and it arises in various forms throughout reinforcement learning.

Remark 1. *The convergence properties of the Sarsa algorithm depend on the nature of the policy's dependence on Q. For example, one could use ϵ-greedy or ϵ-soft policies. Sarsa converges with probability 1 to an optimal policy and action-value function as long as all state-action pairs are visited an infinite number of times and the policy converges in the limit to the greedy policy (which can be arrange, for example, with ϵ-greedy policies by setting $\epsilon = 1/t$). (page 129, [1])*

3.2 Q-Learning

One of the early breakthroughs in reinforcement learning was the development of an off-policy TD control algorithm known as Q-learning (Watkins, 1989), defined by

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[\,R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\,\Big].$$

The update equation can be explained by Bellman optimality equation:

$$q_*(s,a) = \mathbb{E}\big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a\big]$$

In this case, the Q-table directly approximates the optimal action-value function q_* independent of the policy being followed. Similar to Sarsa algorithm, since we have no access to the true optimal action-value function $q_*(S_{t+1}, a)$, the update equation uses the current estimate $Q(S_{t+1}, a)$ (bootstrapping), and hence, the TD target is again biased. The algorithm of Q-learning is as follows:

Algorithm 3 Q-learning

```
Parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$.
1: for each episode do
2:     Initialize $S$
3:     for each step in episode do
4:         Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
5:         Take action $A$, observe $R$, $S'$
6:         $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
7:         $S \leftarrow S'$
8:         if $S$ is terminal then
9:             break
10:        end if
11:    end for
12: end for
```

In Q-learning, the TD target is $R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$, where the chosen action a is the one that maximizes Q in the next state. The action a is different from the action that would be chosen by the policy that chooses the current action A_t (*off-policy method*). We describe the difference of on-policy and off-policy in the next subsection.

3.3 On-Policy v.s. Off-Policy

3.4 Sarsa vs. Q-learning

3.5 Expected Sarsa

Expected Sarsa like Q-learning algorithm that uses the expected value over next state-action pairs instead of the maximization.

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha\big[R_{t+1} + \gamma \mathbb{E}_\pi[Q(S_{t+1}, A_{t+1}) \,|\, S_{t+1}] - Q(S_t, A_t)\big] \\ &\leftarrow Q(S_t, A_t) + \alpha\big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)\big] \end{aligned}$$

Given the next state, S_{t+1}, this algorithm moves *deterministically* in the same direction as Sarsa moves in expectation. Expected Sarsa costs more than the original Sarsa, but it eliminates the variance due to the random selection of A_{t+1}.

3.6 Maximization Bias and Double Learning

References

[1] Andrew Barto and Richard S. Sutton, *Reinforcement Learning: An Introduction* (2nd ed.), The MIT Press, 2018.