Reinforcement Learning

Notes

Jingye Wang

✉ wangjy5@shanghaitech.edu.cn

Spring 2020

Contents

7 Model-Free Control

In the last section, we introduce Monte-Carlo (MC) and Temporal Difference (TD) methods to evaluate the value function of a policy. With the idea of generalized policy iteration (GPI) we mentioned in section 5.3.2, we now consider how the value function can be used in control, that is, to find the optimal policy.

For an MDP with large state space, the way we sample episodes matters a lot. We define *behavior policy* that determines which action to take and *target policy* that determines the best policy we have so far, then based on the two concepts, we have two learning form:

- **On-policy** learning: the target policy and the behavior policy are the same, which means the action we will take follows the optimal policy we find;
- **Off-policy** learning: the target policy and the behavior policy are different, which means the action we will take is independent to the optimal policy we find.

On-policy may not ensure the enough exploration of the state space as when we update the policy greedily we may discard those states with great potential. Compared with on-policy learning, off-policy learning is more powerful and general, though it is often of greater variance and is slower to converge.

7.1 On Policy Monte-Carlo Control

Like MC evaluation we mentioned in section 6.1, MC control also requires fully episodes to improve the policy. It is natural to alternate between evaluation and improvement on an episode-by-episode basis. The following algorithms are based on the implementation of MC, and the differences lie in how they generate samples.

Exploring Starts: *To obtain diverse episodes, a naive ideal is initializing each episode randomly:*

Random starts usually are not common in reality. In the iteration part, we can leverage $\varepsilon-$*Greedy Exploration* to avoid the unlikely assumption of exploring starts.

$\varepsilon-$**Greedy Exploration**: *For a state with m actions, there is non-zero probability to try them:*

$$\pi(a|s) = \begin{cases} \frac{\varepsilon}{m} + 1 - \varepsilon, & a = \arg\max_{a' \in \mathcal{A}} Q(s, a') \\ \frac{\varepsilon}{m}, & \text{otherwise} \end{cases}$$

which means we will choose the greedy action with probability $1 - \varepsilon$ and choose an action at random with probability ε.

Policy improvement theorem: *For any ε-greedy policy π, the ε-greedy policy π' with respect to q_π is an improvement, i.e., $v'_\pi(s) \geq v_\pi(s)$ for all $s \in \mathcal{S}$.*

Algorithm 13 Monte-Carlo Control

```
1: initialize $\pi(s) \in \mathcal{A}(s)$(arbitrarily), for all $s \in \mathcal{S}$; initialize $return(s) \leftarrow$ an empty list for all $s \in \mathcal{S}$;
2: for true do:
   # variants for this alg. can be start with $s_0 \in \mathcal{S}, a_0 \in \mathcal{A}(s_0)$ randomly, $\varepsilon-$greedy, etc.
3:     Generate a complete episode $\tau = (s_0, a_0, r_1, ..., s_{T-1}, a_{T-1}, r_T)$;
4:     $G \leftarrow 0$;
5:     for $t = T-1, T-2, ...0$ do:
6:         $G \leftarrow \gamma G + r_{t+1}$;
7:         if $s_t, a_t$ appears in $(s_0, a_0, s_1, a_1, ..., s_{t-1}, a_{t-1})$ then:
8:             Append $G$ to $return(s_t, a_t)$;
9:             $q(s_t, a_t) \leftarrow$average($return(s_t, a_t)$);
10:            $\pi(s_t) \leftarrow \arg\max_{a'} Q(s_t, a')$;
11:        end if
12:    end for
13: end for
```

Proof:

$$\begin{aligned}
q_\pi\left(s, \pi'(s)\right) &= \sum_{a\in\mathcal{A}} \pi'(a|s) q_\pi(s,a) \\
&= \frac{\varepsilon}{m} \sum_{a\in\mathcal{A}} q_\pi(s,a) + (1-\varepsilon) \max_a q_\pi(s,a) \\
&\geq \frac{\varepsilon}{m} \sum_{a\in\mathcal{A}} q_\pi(s,a) + (1-\varepsilon) \sum_{a\in\mathcal{A}} \frac{\pi(a|s) - \frac{\varepsilon}{m}}{1-\varepsilon} q_\pi(s,a) \\
&= \sum_{a\in\mathcal{A}} \pi(a|s) q_\pi(s,a) \\
&= v_\pi(s).
\end{aligned} \tag{1}$$

□

(The above proof is given in the Chapter 5 of the book [1]. After expanding the sum, however, it can be shown the inequality should be equality. Hmm...)

For ε-greedy, when we explore, we choose actions at random without regard to their estimated values, which may waste a chance on the action that we can sure it has a bad performance. To focus more on those states with higher value, we can refer to *Boltzmann exploration*.

Boltzmann Exploration: *Boltzmann exploration also known as Gibbs sampling and soft-max, it chooses an action based on its estimated value:*

$$\pi(a|s) = \frac{e^{\beta \hat{Q}(s,a)}}{\sum_{a'} e^{\beta \hat{Q}(s,a')}},$$

where $\hat{Q}(s,a)$ is the estimation of the value of being in state s and taking action a, β is a tunable parameter.

7.2 On Policy Temporal-Difference Control: Sarsa

As we mentioned before, Temporal-Difference(TD) learning has several advantages over Monte-Carlo (MC) such as lower variance, online, incomplete sequences. The difference between TD control and TD learning is similar to that between MC control and MC learning. What we need to do is applying TD to $Q(S, A)$ and updating the policy every time-step.

Sarsa: *The name Sarsa is inspired by the exploitation on the quintuple* $s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}$. *Consider transitions from state-action pair to state-action pair, and learn the values of state-action pairs:*

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

Algorithm 14 TD Sarsa

```
1: initialize $\pi(s) \in \mathcal{A}(s)$(arbitrarily), for all $s \in \mathcal{S}$; initialize $Q(s,a) \in \mathcal{R}$(arbitrarily) for all possible
   pairs in $\mathcal{S} \times \mathcal{A}$; initialize $Q(terminal, \cdot) = 0$;
2: for true do:
   # variants for this alg. can be start with $s_0 \in \mathcal{S}, a_0 \in \mathcal{A}(s_0)$ randomly, $\varepsilon-$greedy, etc.
3:     Generate a start state $s$;
4:     Choose action $a \leftarrow \pi(s)$;
5:     while $s$ is not terminal do:
6:         $r, s' \leftarrow environment(s, a)$;
7:         $a' \leftarrow \pi(s')$
8:         $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s', a') - Q(s,a)]$;
9:         Update $\pi$ according to $Q(s,a)$;
10:        $s \leftarrow s'$;
11:        $a \leftarrow a'$;
12:    end while
13: end for
```

There are also n-step Sarsa and Sarsa(λ) which can be derived from TD methods.

n-**step Sarsa**: *Define the* n*-step Q-return as*

$$q_t^{(n)} = r_{t+1} + \gamma r_{t+2} + ... + \gamma^{n-1} r_{t+n} + \gamma^n Q(s_{t+n}, a_{t+n}),$$

then n*-step Sarsa updates* $Q(s,a)$ *towards the* n*-step Q-return*

$$Q(s_t, a_t) \leftarrow Q(a_t, t_t) + \alpha(q_t^{(n)} - Q(s_t, a_t)).$$

Like what we mentioned in n-step TD learning, when $n \to \infty$, Sarsa$\to$MC.

7.3 Off-Policy Temporal-Difference Control: Q-Learning

The development of Q-learning is a big breakout in the early days of Reinforcement Learning.

For the target policy, Q-learning updates it in a *greedy* way:

$$\pi(s_{t+1}) = \arg\max_{a'} Q(s_{t+1}, a').$$

For the behavior policy, it will be updated in an ε-*greedy* way on $Q(s,a)$.

The state-action value will be updated as

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s,a)].$$

Algorithm 15 Q-Learning

```
1: initialize $\pi(s) \in \mathcal{A}(s)$(arbitrarily), for all $s \in \mathcal{S}$; initialize $Q(s,a) \in \mathcal{R}$(arbitrarily) for all possible
   pairs in $\mathcal{S} \times \mathcal{A}$; initialize $Q(terminal, \cdot) = 0$;
2: for true do:
   # variants for this alg. can be start with $s_0 \in \mathcal{S}, a_0 \in \mathcal{A}(s_0)$ randomly, $\varepsilon-$greedy, etc.
3:     Generate a start state $s$;
4:     Choose action $a \leftarrow \pi(s)$;
5:     while $s$ is not terminal do:
6:         $r, s' \leftarrow environment(s,a)$;
7:         $a' \leftarrow \arg\max_{a'} Q(s', a')$
8:         $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$;
9:         Update $\pi$ according to $Q(s,a)$;
10:        $s \leftarrow s'$;
11:        $a \leftarrow a'$;
12:    end while
13: end for
```

One should notice that different with TD Sarsa in line 7, the action Q-learning uses to update Q is not the same as the one it truly takes.