Chapter 7 Temporal-Difference Methods

TD Learning for State Values, Sarsa, n-step Sarsa, Q-learning, Online/Offline, On-Policy/Off-Policy

1. What is the relationship between TD Learning for State Values, Sarsa, n-step Sarsa, Q-learning?

TD Learning for State Values & Sarsa

2. Derive the formula of TD for state values.
3. Why is v_k* the TD target?
4. The formula of Sarsa.
5. Explain the process of using Sarsa and ε-greedy to find optimal policy.
6. What is Expected Sarsa?
7. Explain the converge condition of TD for state values, Sarsa and Q-learning.
8. Compare TD with MC.

n-step Sarsa

9. What is the relationship between MC, Sarsa and n-step Sarsa?
10. What happens when n increases?

Q-learning

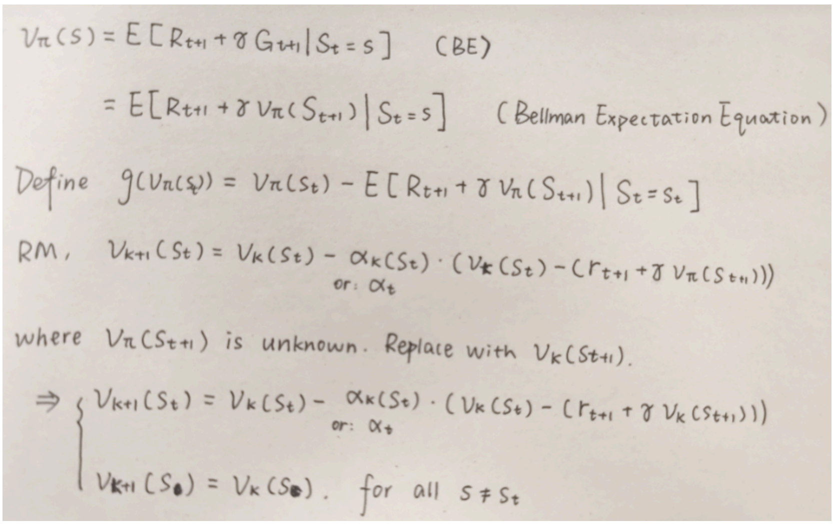
11. The expression of Q-learning
12. The porcess of Q-learning(on-policy and off-policy). Why the policy update methods are difference?

Online/Offline & On-Policy/Off-Policy

13. What is online/offline?
14. What is on-policy/off-policy?
15. What is the relationship between online/offline and on-policy/off-policy?

16. Explain the unified viewpoint of MC, Sarsa, n-step Sarsa, Q-learning.

1. TD for state values can only estimate state values used for policy evaluation, but can not find an optimal policy. Sarsa can estimate action values and can be combined with policy improvement steps to find optimal policy. n-step Sarsa is the generalization of Sarsa and MC. Q-learning directly solve BOE to obtain optimal policy.
2. Generate expeirence examples (s_0, r_1, s_1, r_2,..., s_t, r_{t+1}, s_{t+1}, ...) following given policy π. Initially guess $v_0(s)$ for every state. To estimate state values:



$$v_\pi(s) = E[R_{t+1} + \gamma G_{t+1} | S_t = s] \quad \text{(BE)}$$

$$= E[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \quad \text{(Bellman Expectation Equation)}$$

Define $g(v_\pi(s_t)) = v_\pi(s_t) - E[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s_t]$

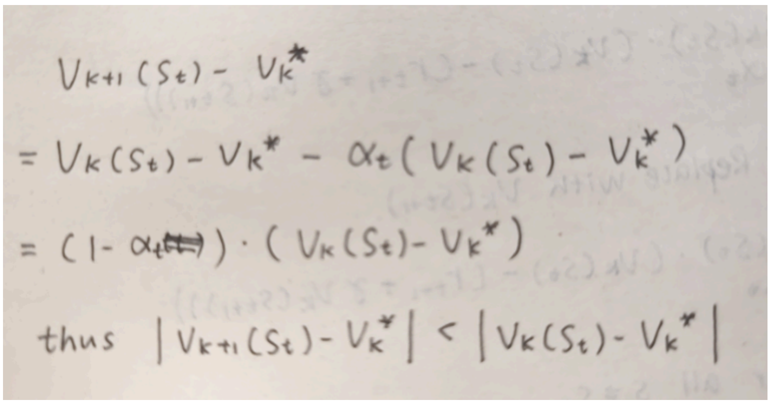
RM, $v_{k+1}(s_t) = v_k(s_t) - \underset{\text{or: } \alpha_t}{\alpha_k(s_t)} \cdot (v_k(s_t) - (r_{t+1} + \gamma v_\pi(s_{t+1})))$

where $v_\pi(s_{t+1})$ is unknown. Replace with $v_k(s_{t+1})$.

$$\Rightarrow \begin{cases} v_{k+1}(s_t) = v_k(s_t) - \underset{\text{or: } \alpha_t}{\alpha_k(s_t)} \cdot (v_k(s_t) - (r_{t+1} + \gamma v_k(s_{t+1}))) \\ v_{k+1}(s) = v_k(s). \quad \text{for all } s \neq s_t \end{cases}$$

where v_{k+1} is new estimate, v_k is current estimate, $(r_{t+1} + \gamma v_k(s_{t+1}))$ is TD target v_k^*, $(v_k(s_t) - (r_{t+1} + \gamma v_k(s_{t+1})))$ is TD error δ_k.

3.



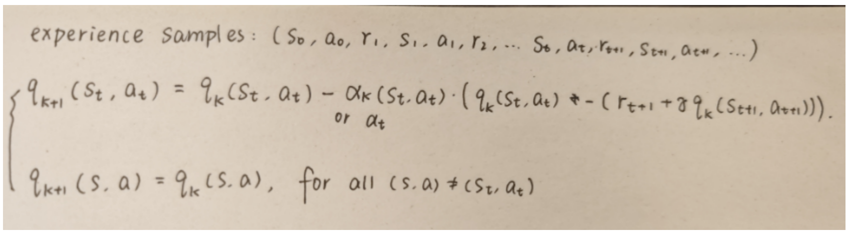
$$v_{k+1}(s_t) - v_k^*$$

$$= v_k(s_t) - v_k^* - \alpha_t(v_k(s_t) - v_k^*)$$

$$= (1 - \alpha_t) \cdot (v_k(s_t) - v_k^*)$$

thus $|v_{k+1}(s_t) - v_k^*| < |v_k(s_t) - v_k^*|$

4. Based on the derivation of TD for State Values, Sarsa only needs few changes.



experience samples: $(s_0, a_0, r_1, s_1, a_1, r_2, \dots s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, \dots)$

$$\begin{cases} q_{k+1}(s_t, a_t) = q_k(s_t, a_t) - \underset{\text{or } \alpha_t}{\alpha_k(s_t, a_t)} \cdot (q_k(s_t, a_t) - (r_{t+1} + \gamma q_k(s_{t+1}, a_{t+1}))). \\ q_{k+1}(s, a) = q_k(s, a), \quad \text{for all } (s, a) \neq (s_t, a_t) \end{cases}$$

Here, s_{t+1} and r_{t+1} is generated by interacting with the environment. a_{t+1} is generated by current policy.

5.

Algorithm 7.1: Optimal policy learning by Sarsa

Initialization: $\alpha_t(s,a) = \alpha > 0$ for all (s,a) and all t. $\epsilon \in (0,1)$. Initial $q_0(s,a)$ for all (s,a). Initial ϵ-greedy policy π_0 derived from q_0.
Goal: Learn an optimal policy that can lead the agent to the target state from an initial state s_0.

```
For each episode, do
    Generate $a_0$ at $s_0$ following $\pi_0(s_0)$
    If $s_t$ $(t = 0, 1, 2, \dots)$ is not the target state, do
        Collect an experience sample $(r_{t+1}, s_{t+1}, a_{t+1})$ given $(s_t, a_t)$: generate $r_{t+1}, s_{t+1}$
        by interacting with the environment; generate $a_{t+1}$ following $\pi_t(s_{t+1})$.
        Update q-value for $(s_t, a_t)$:
            $q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)\Big[q_t(s_t, a_t) - (r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1}))\Big]$
        Update policy for $s_t$:
            $\pi_{t+1}(a|s_t) = 1 - \frac{\epsilon}{|\mathcal{A}(s_t)|}(|\mathcal{A}(s_t)| - 1)$ if $a = \arg\max_a q_{t+1}(s_t, a)$
            $\pi_{t+1}(a|s_t) = \frac{\epsilon}{|\mathcal{A}(s_t)|}$ otherwise
        $s_t \leftarrow s_{t+1}$, $a_t \leftarrow a_{t+1}$
```

6. Expected sarsa replaces $q_k(s_{t+1}, a_{t+1})$ with $E[q_k(s_{t+1}, A)] = \Sigma_a \pi_t(a|s_{t+1}) q_k(s_{t+1}, a) = v_t(s_{t+1})$. It can reduce the estimate variance of sarsa for deleting the random variable a_{t+1}.

7. The learning rate should decrease but cannot decrease very fast.

$$\sum_{t=0}^{\infty} \alpha_t = \infty$$

$$\sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

In the practice of **on-policy** methods, we often set learning rate a small constant because the policy is keep changing and a decaying learning rate may be too small to effectively evaluate the policy. Even a constant learning rate may result in fluctuation, if the constant is small enough, then the flucuation can be neglectable.

8.

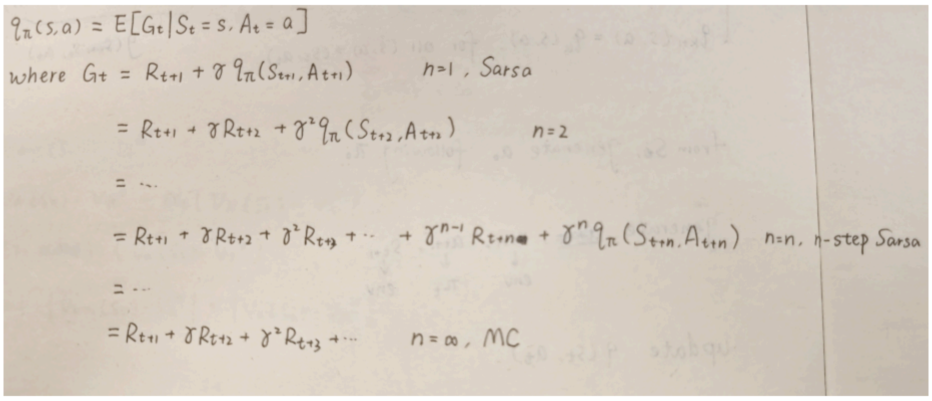
(1) TD is incremental. It can update state value/action value immediately after receiving an experience example. MC is non-incremental. It must wait until the whole episode is collected because MC need to calculate the discounted rate of the episode.

(2) TD can solve continuing tasks because it is incremental. MC can only solve finite episodes or infinite episode but terminate after finite steps.

(3) TD bootstraps. It estimates state values/action values relying on previous estimate. Thus TD need an initial guess. MC is not bootstrapping because it directly calculate state values/action values without initial guess.

(4) TD has a lower estimation variance than MC because it needs fewer random variables. Take calculating action value $q_\pi(s_t, a_t)$ as an example, Sarsa only needs R_{t+1}, S_{t+1}, A_{t+1} but MC needs R_{t+1}, R_{t+2}, R_{t+3}, … .

9. The difference is how to calculate $q_\pi(s_t, a_t)$. MC and Sarsa are two extreme situation of n-step Sarsa.



$$q_\pi(s,a) = E[G_t | S_t = s, A_t = a]$$

where $G_t = R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})$ $n=1$, Sarsa

$$= R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, A_{t+2}) \qquad n=2$$

$$= \cdots$$

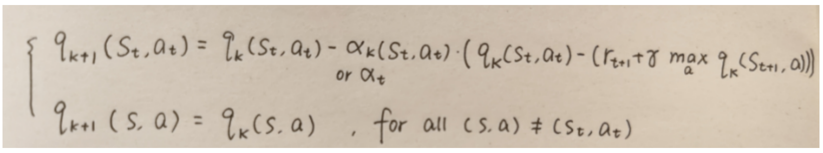
$$= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n q_\pi(S_{t+n}, A_{t+n}) \qquad n=n, \text{ n-step Sarsa}$$

$$= \cdots$$

$$= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \qquad n=\infty, \text{ MC}$$

10. First, each time updating action value needs a longer time to wait for collecting data (r_{t+1}, r_{t+2}, r_{t+3}, ..., r_{t+n}, s_{t+n}, a_{t+n}). Second, the variance increases but bias decreases.

11.



$$\begin{cases} q_{k+1}(s_t, a_t) = q_k(s_t, a_t) - \underset{\text{or } \alpha_t}{\alpha_k(s_t, a_t)} \cdot \left(q_k(s_t, a_t) - (r_{t+1} + \gamma \max_a q_k(s_{t+1}, a))\right) \\ q_{k+1}(s, a) = q_k(s, a) \quad , \text{ for all } (s, a) \neq (s_t, a_t) \end{cases}$$

12.

Algorithm 7.2: Optimal policy learning via Q-learning (on-policy version)

Initialization: $\alpha_t(s,a) = \alpha > 0$ for all (s,a) and all t. $\epsilon \in (0,1)$. Initial $q_0(s,a)$ for all (s,a). Initial ϵ-greedy policy π_0 derived from q_0.
Goal: Learn an optimal path that can lead the agent to the target state from an initial state s_0.

```
For each episode, do
    If $s_t$ $(t = 0, 1, 2, \ldots)$ is not the target state, do
        Collect the experience sample $(a_t, r_{t+1}, s_{t+1})$ given $s_t$: generate $a_t$ following $\pi_t(s_t)$; generate $r_{t+1}, s_{t+1}$ by interacting with the environment.
        Update q-value for $(s_t, a_t)$:
            $q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)\Big[q_t(s_t, a_t) - (r_{t+1} + \gamma \max_a q_t(s_{t+1}, a))\Big]$
        Update policy for $s_t$:
            $\pi_{t+1}(a|s_t) = 1 - \frac{\epsilon}{|\mathcal{A}(s_t)|}(|\mathcal{A}(s_t)| - 1)$ if $a = \arg\max_a q_{t+1}(s_t, a)$
            $\pi_{t+1}(a|s_t) = \frac{\epsilon}{|\mathcal{A}(s_t)|}$ otherwise
```

Algorithm 7.3: Optimal policy learning via Q-learning (off-policy version)

Initialization: Initial guess $q_0(s,a)$ for all (s,a). Behavior policy $\pi_b(a|s)$ for all (s,a). $\alpha_t(s,a) = \alpha > 0$ for all (s,a) and all t.
Goal: Learn an optimal target policy π_T for all states from the experience samples generated by π_b.

```
For each episode $\{s_0, a_0, r_1, s_1, a_1, r_2, \ldots\}$ generated by $\pi_b$, do
    For each step $t = 0, 1, 2, \ldots$ of the episode, do
        Update q-value for $(s_t, a_t)$:
            $q_{t+1}(s_t, a_t) = q_t(s_t, a_t) - \alpha_t(s_t, a_t)\Big[q_t(s_t, a_t) - (r_{t+1} + \gamma \max_a q_t(s_{t+1}, a))\Big]$
        Update target policy for $s_t$:
            $\pi_{T,t+1}(a|s_t) = 1$ if $a = \arg\max_a q_{t+1}(s_t, a)$
            $\pi_{T,t+1}(a|s_t) = 0$ otherwise
```

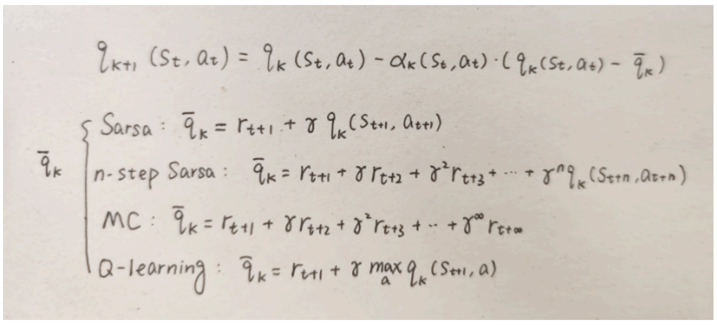
For on-policy method, the policy is used in both exporation and update. Thus we need ε-greedy policy for exploring. For off-policy method, π_T only focus on update without exploring. Thus we just need greedy policy.

13. Online is updating values and policy with interacting with the environment. Offline is learning from generated experience samples following a given policy.
14. The policy which we generate experience samples from is called behavior policy π_b. The policy which we aim to update to find the optimal policy is called target policy π_T. For on-policy methods, these two policies are the same. For off-policy methods, these two policies are different.

Sarsa and MC are on-policy, Q-learning can either be on-policy or off-policy.

The reason why Q-learning can be off-policy is that it directly solve BOE. Thus the update of q value is not related to the policy.

15. If a method is off-policy, then it can use both online and offline and both are same. If a method is on-policy, then it can only use online because the policy needs to be updated in each step.
16.



$$q_{k+1}(s_t, a_t) = q_k(s_t, a_t) - \alpha_k(s_t, a_t)\cdot(q_k(s_t, a_t) - \bar{q}_k)$$

$$\bar{q}_k \begin{cases} \text{Sarsa}: \bar{q}_k = r_{t+1} + \gamma q_k(s_{t+1}, a_{t+1}) \\ n\text{-step Sarsa}: \bar{q}_k = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^n q_k(s_{t+n}, a_{t+n}) \\ \text{MC}: \bar{q}_k = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^\infty r_{t+\infty} \\ \text{Q-learning}: \bar{q}_k = r_{t+1} + \gamma \max_a q_k(s_{t+1}, a) \end{cases}$$

Algorithm	**Equation to be solved**
Sarsa	BE: $q_\pi(s,a) = \mathbb{E}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right]$
n-step Sarsa	BE: $q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^n q_\pi(S_{t+n}, A_{t+n}) \mid S_t = s, A_t = a]$
Q-learning	BOE: $q(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_a q(S_{t+1}, a) \middle\vert S_t = s, A_t = a\right]$
Monte Carlo	BE: $q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s, A_t = a]$