C5: Temporal Difference Learning

▼ Table of Content

Temporal Difference Learning

- In Dynamic Programming, we need to know the model of the environment to solve MDP using Bellman's Equation to find the Expected Return for the optimal policy.
- In MC Control, we can find the optimal policy through optimizing both the value function and policy function without the need to know the model of the environment. However, MC Control requires the task to be episodic and sometimes it might be too long of an episode for the reward to be properly tallied to make an estimate of the value function.
- TD Learning gets the best of both world from DP and MC Control whereby:
  - No need to know the exact model of the environment (MC Control)

- Able to make estimate immediately based on the previously learned estimate which is called bootstrapping. (DP)

Temporal Difference Prediction

- In TD prediction, we tries to predict/estimate the value function (state value) just like in Monte Carlo Prediction.
  - In MC prediction, we estimate value function by simply taking the mean return
  - In TD prediction, we update the state value using the weighted difference of current state value $r + \gamma V(s')$, against the previous reward $V(s)$, using learning rate, α.
  - TD Update Rule

$$V(s) = V(s) + \alpha[r + \gamma V(s') - V(s)]$$

- When we continue the iteration the TD Error $r + \gamma V(s') - V(s)$ will eventually approach to 0 once the value function converged.

TD-Prediction Algorithm

1. Initialize $V(S)$ to 0 or some values
2. Then, we begin episode and for each step in an episode, we perform action A in state S and receive R and move to next state (S')
3. Then, we update the value of the previous state using the TD update rule
4. Repeat step 3 and 4 until we reach the terminal state.

Temporal Difference Control

- In TD control, we control/optimize the value function with following two algorithms:
- In control algorithms, the state value does not matter that much as it is just a reflection to the policy chosen (Refer to C4).

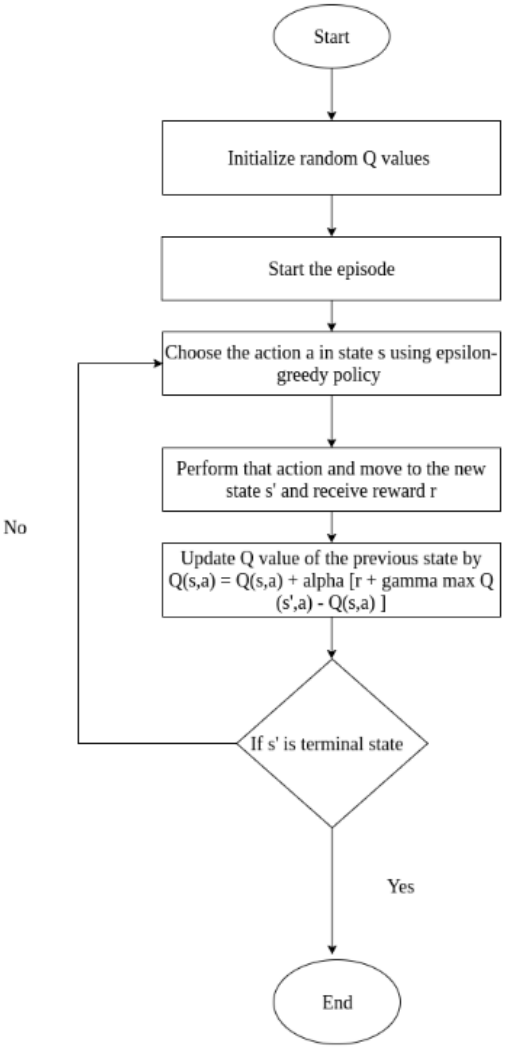
Off Policy Learning : Q-Learning

- In Q-Learning, our concern is optimizing and estimating state-action value pair (the effect of performing and action A in state S.)
- Similar to TD Update Rule, Q-value is updated as follow

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma\ argmax\ Q(s',a') - Q(s,a)]$$

Q-Learning Algorithm

1. Initialize the Q function to some arbitrary values
2. Take an action from a state using epsilon-greedy policy ($\epsilon > 0$) and traverse to the next state
3. Update Q value of previous state using the update rule above
4. Repeat step 2 and 3 till we reach terminal state
    - While choosing what action to take, we perform epsilon-greedy policy
    - While updating the Q-value, we don't perform the epsilon-greedy policy and simply select action that has maximum value



On Policy Learning : SARSA

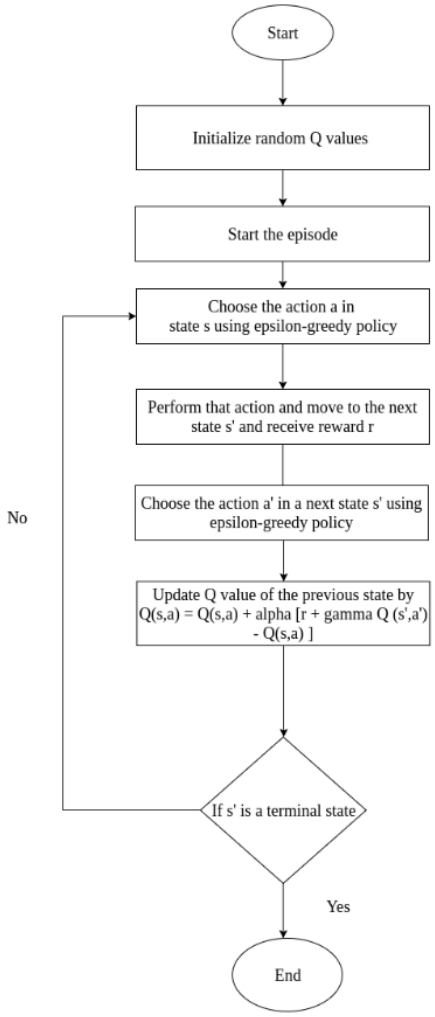
- State-Action-Reward-State-Action is an on-policy TD algorithm

- In SARSA, Q-value is updated always using epsilon-greedy policy in both choosing action phase and updating Q-value phase.
- The Update Rule for SARSA is as follows:

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma\, Q(s', a' < selected\ with\ \epsilon >)\ - Q(s,a)]$$

SARSA Algorithm

1. Initialize Q-values to some arbitrary values
2. Selection action based on epsilon-greedy policy ($\epsilon < 0$) and move from one state to another
3. Update Q-value of previous state with update rule above where a' is chosen by epsilon-greedy policy ($\epsilon > 0$)



Q-Learning vs SARSA

```
|            | SARSA | Q-learning |
|:-----------:|:-----:|:----------:|
```

```
 | Choosing A' |   π   |      π     |
 | Updating Q  |   π   |      μ     |

μ : Select Action with Maximum Q(s',a')
π : Select Action with Epsilon-Greedy Algorithm
```

Off-policy vs On-policy

1. Given that Q-learning is using different policies for choosing next action A' and updating Q. In other words, it is trying to evaluate π while following another policy μ, so it's an off-policy algorithm.
2. In contrast, SARSA uses π all the time, hence it is an on-policy algorithm.

Epsilon-Greedy or Not

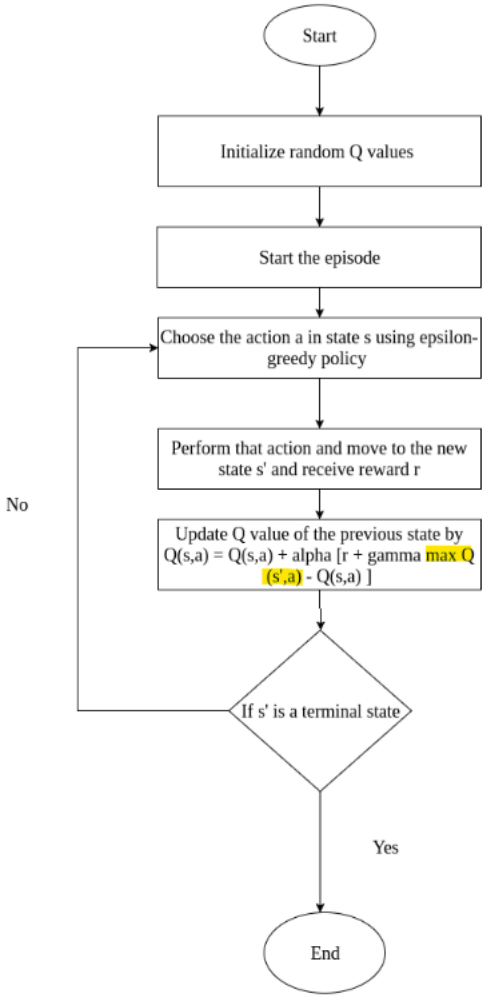
1. The most important difference between the two is how Q is updated after each action. SARSA uses the Q' following a ε-greedy policy exactly, as A' is drawn from it. In contrast, Q-learning uses the maximum Q' over all possible actions for the next step. This makes it look like following a greedy policy with ε=0, i.e. NO exploration in this part.
2. However, when actually taking an action, Q-learning still uses the action taken from a ε-greedy policy. This is why "Choose A ..." is inside the repeat loop.
3. Following the loop logic in Q-learning, A' is still from the ε-greedy policy.

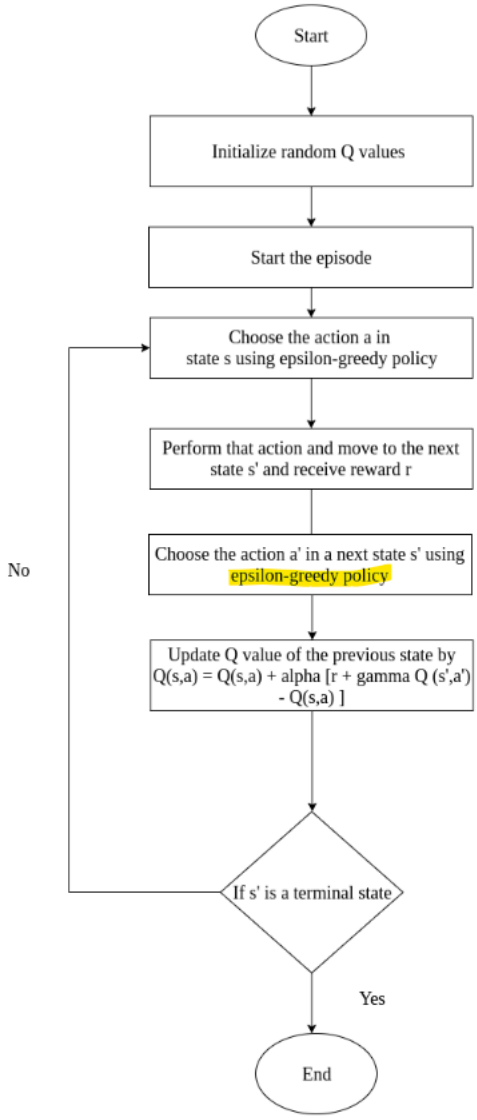
Choosing of Q-learning vs SARSA

> It all depends on environment and the goal.
>
> 1. Q-learning can find a better goal (provided the epsilon does not converge) than SARSA as it does not penalize the action-value for doing epsilon-exploration.
>
> 2. SARSA is more conservative (as explore large actions that could penalize the q-table), and tend to avoid dangerous path even tho there could be positive/optimal result given afterwards. As such, in

> an environment with alot of danger zone (negative reward), SARSA might find it hard to find the most optimal policy.

1. Q-learning directly learns the optimal policy, whilst SARSA learns a near-optimal policy whilst exploring. If you want to learn an optimal policy using SARSA, then you will need to decide on a strategy to decay $\epsilon\epsilon$ in $\epsilon\epsilon$greedy action choice, which may become a fiddly hyperparameter to tune.
2. Q-learning (and off-policy learning in general) has higher per-sample variance than SARSA, and may suffer from problems converging as a result. This turns up as a problem when training neural networks via Q-learning.
3. SARSA will approach convergence *allowing* for possible penalties from exploratory moves, whilst Q-learning will ignore them. That makes SARSA more conservative - if there is risk of a large negative reward close to the optimal path, Q-learning will tend to trigger that reward whilst exploring, whilst SARSA will tend to avoid a dangerous optimal path and only slowly learn to use it when the exploration parameters are reduced. The classic toy problem that demonstrates this effect is called cliff walking.





Taxi Problem

Fetch a passenger from its original location (Blue) to its destination (red).

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
(Action)
```

Actions

- 0 : South
- 1 : North
- 2 : East
- 3 : West
- 4 : Pickup passenger
- 5 : Drop off passenger

Rewards

- -1 per time step
- +20 if passenger is delivered (pickup + drop off)
- -10 if illegal pickup and drop-off

Cliff Walking