Model-Free TD Control: Q-Learning and SARSA

Terminology in Today's Topic

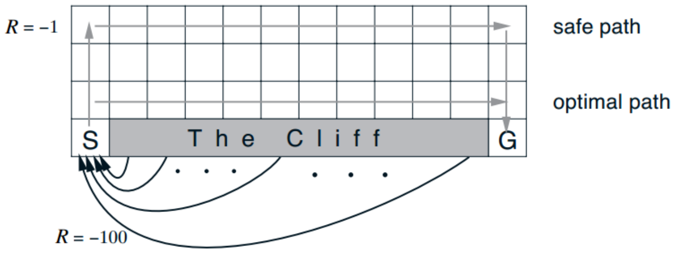
- **model-free**: different from the DP, in Q-Learning, we do all the things within an unknown MDP, just what the model-free means.
- **prediction & control**: model-free prediction means *estimating* the value function of an unknown MDP, Model-free control means *optimizing* the value function of an unknown MDP.
- **on-policy & off-policy**: on-policy learning is to learn about policy ππ from experience sample from ππ, off-policy learning is to learn about policy ππ *from other policy* μμ's experience, e.g. learn from a coach
- **TD-learning**: temporal-difference learning is a combination of Monte Carlo(MC) ideas and dynamic programming(DP) ideas. Like MC methods, TD methods can learn directly from raw experience without a model of the environment's dynamics. Like DP, TD methods update estimates based in part on other learned estimates, without waiting for a final outcome like MC.

Now it's time for introducing Q-Learning algorithm, which is an off-policy TD control algorithm, one of the early breakthroughs in model-free RL, defined by

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a_t) - Q(s_t, a_t) \right]$$

Next, we will implement Q-learning in cliff-walking task.

Cliff Walking



This is a standard undiscounted, episodic task, with start and goal states, and the usual actions causing movement up, down, right, and left. Reward is −1 on all transitions except those into the region marked "The Cliff." Stepping into this region incurs a reward of −100 and sends the agent instantly back to the start.

```
class Env():
def cliff_walk():
```

Q-Learning Algorithm

First, Recall the ϵ-greedy action selection:

- with probability ϵ, choose an action at random
- with probability $1-\epsilon$, choose the greedy action

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm 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 $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose A from S using policy derived from Q (e.g., ϵ-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$
    until S is terminal

```
class Q_table():
def take_action(self, x, y, num_episode):
def max_q(self, x, y):
def update(self, a, s0, s1, r, is_terminated):
```

SARSA

modify the code to SARSA, unlike Q-learning which is on-policy algorithm, and see the different conduct of the agent when algorithm converges.

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm 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 $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
    Initialize S
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Loop for each step of episode:
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal