ELEC6901J Deep Reinforcement Learning
Homework 2

April 3, 2025

YIN, Tianci
20587470

---

1. TD Learning (10 pts)

Answer: The TD Learning update is given by

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right]$$

Since we only have one observed transition, given by:

$$B, \text{east}, r = -2, C,$$

only the value estimate of state B will be updated. The update is given by:

$$\begin{aligned} V(B) &\leftarrow V(B) + \alpha \left[R_{t+1} + \gamma V(C) - V(B)\right] \\ &= 2 + 0.5\left[-2 + 1 \cdot 8 - 2\right] \\ &= 2 + 0.5\left[-2 + 8 - 2\right] \\ &= 4 \end{aligned}$$

The resulting values estimates are:

$$\begin{aligned} \hat{V}^{\pi}(A) &= 1 \\ \hat{V}^{\pi}(B) &= 4 \\ \hat{V}^{\pi}(C) &= 8 \\ \hat{V}^{\pi}(D) &= 10 \\ \hat{V}^{\pi}(E) &= 10 \end{aligned}$$

2. A Simple MDP (20 pts)

Answer:

(a)

Q2

- S_0
  - > first time in traj.1, t=0
    $G_0 = 0 + \gamma 0 + \gamma^2 0 + \gamma^3 10 = 7.29$, where $\gamma = 0.9$
  - > first time in traj.2, t=0
    $G_0 = 0 + \gamma 0 + \gamma^2 1 + \gamma^3 0 + \gamma^4 0 + \gamma^5 0 + \gamma^6 10 = 6.1244$
  - > first time in traj.3, t=0
    $G_0 = 0 + \gamma 1 + \gamma^2 0 + \gamma^3 0 + \gamma^4 0 + \gamma^5 0 + \gamma^6 0 + \gamma^7 0 + \gamma^8 10 = 5.2047$

  $V(S_0) = \overline{G_0} = \frac{18.619}{3} = 6.21$

- S_1
  - > first time in traj.1, t=1
    $G_1 = 0 + \gamma 0 + \gamma^2 10 = 8.1$
  - > first time in traj.2, t=2
    $G_2 = 1 + \gamma 0 + \gamma^2 0 + \gamma^3 0 + \gamma^4 10 = 7.561$
  - > first time in traj.3, t=1
    $G_1 = 1 + \gamma 0 + \gamma^2 0 + \gamma^3 0 + \gamma^4 0 + \gamma^5 0 + \gamma^6 0 + \gamma^7 10 = 5.783$

  $V(S_1) = \frac{8.1 + 7.561 + 5.783}{3} = 7.15$

- S_2
  - > first time in traj.1, t=2
    $G_2 = 0 + \gamma 10 = 9$
  - > first time in traj.2, t=5
    $G_5 = 0 + \gamma 10 = 9$
  - > first time in traj.3, t=5
    $G_5 = 0 + \gamma 0 + \gamma^2 0 + \gamma^3 10 = 7.29$

  $V(S_2) = \frac{9 + 9 + 7.29}{3} = 8.43$

(b)

b) SARSA $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$,
where $\alpha = 0.5$ $\gamma = 0.9$

seq.1

$Q(S_1, a_0) \leftarrow Q(S_1, a_0) + 0.5 [1 + 0.9 \cdot Q(S_0, a_1) - Q(S_1, a_0)]$

$\leftarrow 0 + 0.5 [1 + 0.9 \cdot 0 - 0]$

$\leftarrow 0.5$

$Q(S_1, a_0) = 0.5$

seq.2

$Q(S_2, a_1) \leftarrow Q(S_2, a_1) + 0.5 [0 + 0.9 \cdot Q(S_3, a_1) - Q(S_2, a_1)]$

$\leftarrow 0 + 0.5 [0 + 0.9 \cdot 0 - 0]$

$\leftarrow 0$

$Q(S_2, a_1) = 0$

seq.3

$Q(S_2, a_1) \leftarrow Q(S_2, a_1) + 0.5 [0 + 0.9 \cdot Q(S_1, a_0) - Q(S_2, a_1)]$

$\leftarrow 0 + 0.5 [0 + 0.9 \cdot 0.5 - 0]$

$\leftarrow 0.225$

$Q(S_2, a_1) = 0.225$

updated values

$Q(S_1, a_0) = 0.5$

$Q(S_2, a_1) = 0.225$

(c)

c) Q-learning $Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha\left[R_{t+1} + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$,
where $\alpha = 0.5$ $\gamma = 0.9$

seq. 1

$Q(s_1,a_0) \leftarrow Q(s_1,a_0) + 0.5\left[1 + 0.9 \max\left[Q(s_0,a_0), Q(s_0,a_1)\right] - Q(s_1,a_0)\right]$

$\leftarrow 0 + 0.5[1 + 0.9 \cdot 0 - 0]$

$\leftarrow 0.5$

$Q(s_1,a_0) = 0.5$

seq. 2

$Q(s_2,a_1) \leftarrow Q(s_2,a_1) + 0.5\left[0 + 0.9 \cdot \max\left[Q(s_3,a_0), Q(s_3,a_1)\right] - Q(s_2,a_1)\right]$

$\leftarrow 0 + 0.5[0 + 0.9 \cdot 0 - 0]$

$\leftarrow 0$

$Q(s_2,a_1) = 0$

seq. 3

$Q(s_2,a_1) \leftarrow Q(s_2,a_1) + 0.5\left[0 + 0.9 \cdot \max\left[Q(s_1,a_0), Q(s_1,a_1)\right] - Q(s_2,a_1)\right]$

$\leftarrow 0 + 0.5[0 + 0.9 \cdot 0.5 - 0]$

$\leftarrow 0.225$

$Q(s_2,a_1) = 0.225$

updated value

$Q(s_1,a_0) = 0.5$

$Q(s_2,a_1) = 0.225$

(d) With the Q-learning update being:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a')\right]$$

When $\alpha = 1$, the equation becomes:

$$Q(s,a) \leftarrow (1-1)Q(s,a) + 1[r + \gamma \max_{a'} Q(s',a')]$$
$$Q(s,a) \leftarrow 0 + [r + \gamma \max_{a'} Q(s',a')]$$
$$Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s',a')$$

In this case the updated $Q(s,a)$ no longer depends on the previous value of $Q(s,a)$, and is only dependent on the immediate reward and the maximum value of the next state. Evidently, no averaging is done, and instead simply the value from the sample will be used. So statement A is correct.

When $\alpha = 0$, the equation becomes:

$$Q(s,a) \leftarrow (1-0)Q(s,a) + 0[r + \gamma \max_{a'} Q(s',a')]$$
$$Q(s,a) \leftarrow Q(s,a) + 0$$
$$Q(s,a) \leftarrow Q(s,a)$$

In this case, the value of $Q(s,a)$ will not be updated at all, and will remain the same. And evidently, the sample will not influence the update. statement B is correct.

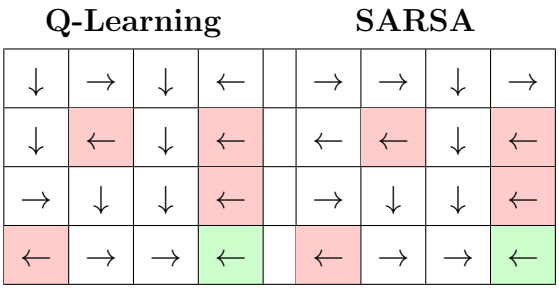
Consequently, statement C is incorrect.

3. Frozen Lake MDP (60 pts)

Answer:

(a) view 'td.py'

(b) The converged policies for Q-Learning and SARSA are as follows:



Notably, SARSA converged to a policy, where although there is a clear optimal path to the goal, other potential paths seem to have counterintuitive optimal actions mapped to them. This is likely due to the fact that SARSA is on-policy and continues to iterate over the policy, such that it may get entrenched in a first successful path. Other paths were not yet explored enough to converge to more optimal policies.

Interestingly, when $gamma = 1$, the Q-learning policy sometimes fails to converge to a working policy at all. This possibly happens when without discounting, the algorithm had enough time for the reward at the terminal state to be propagated back to the start state (via the max function in the update rule), such that all states' value estimates tend to 1.

(c) One such example of a real-world MDP with discrete state and action space would be a game of Texas Hold'em Poker (from the perspective of a single player = single agent), though only partially observable (POMDP).

To define the MDP, we need to define the state space, and action space.

- Finite set of cards in hand (2 cards)
- Finite set of cards on the table (5 cards)
- Discrete number of chips in the pot, player's stack, and opponent's stack
- Discrete sequence of betting rounds and actions (fold, call, raise)

The action space has the discrete dimensions of:

- Fold, Check, Call, (All-in)

- Discrete ranges of Raise/Bet sizes

Though we could stop here with a model-free RL problem, poker also has quantfiable rewards, which are the net amount of chips won or lost per hand (episode). The transition dynamics is a stochastic process, as cards are drawn randomly from the deck, and the opponent's policies are complex.

(d) Since Monte Carlo methods are based on averaging the **actual** discounted total rewards from the sampled episode, the algorithm is largely unbiased. However, since the algorithm relies on the entire sampled episode, the compounding effect of the transition dynamics and the corresponding reward variability (especially in long episodes) introduces large variances.

TD(0) methods on the other hand involve bootstrapping, where the estimated value of the next state is used to update the current state's values estimate. As there are no guarantees that the estimated value of the next state is correct, this introduces a bias in the update. However, since updates in TD(0) are only based on the current state and the next state, the variance of the update is much lower than that of Monte Carlo methods.

For environments with highly noisy rewards, Monte Carlo methods are even more sensitive to the sampled episodes, and the variance of the updates will be very high. As such, TD(0) methods are preferred in this case.

(e) In REINFORCE (a policy gradient method), we use gradient ascent to maximize the expected return of the policy. This involves calculating derivatives. By taking the log of the policy, we turn the product of the probabilities into a sum, which is easier to differentiate. This is possible since the log function is monotonic, and maximizing $\log f(x)$ is equivalent to maximizing $f(x)$.