Solutions for week 5
Reinforcement Learning: Q-value and SARSA

Exercise 1: SARSA algorithm

In the lecture, we introduced the SARSA (state-action-reward-state-action) algorithm, which (for discount factor $\gamma = 1$) is defined by the update rule

$$\Delta Q(s,a) = \eta\left[r - \big(Q(s,a) - Q(s',a')\big)\right] , \tag{1}$$

where s' and a' are the state and action subsequent to s and a. In this exercise, we apply a greedy policy, i.e., at each time step, the action chosen is the one with maximal expected reward, i.e.,

$$a_t^* = \arg\max_a Q_t(s,a) \,. \tag{2}$$

If the available actions have the same Q-value, we take both actions with probability 0.5.

Consider a rat navigating in a 1-armed maze (=linear track). The rat is initially placed at the upper end of the maze (state s), with a food reward at the other end. This can be modeled as a one-dimensional sequence of states with a unique reward ($r = 1$) as the goal is reached. For each state, the possible actions are going up or going down (Figure 1). When the goal is reached, the trial is over, and the rat is picked up by the experimentalist and placed back in the initial position s and the exploration starts again.

a. Suppose we discretize the linear track by 6 states, $s_1, \ldots, s_6$ where s_1 is the initial state and s_6 is the goal state. Initialize all the Q-values at zero. How do the Q-values develop as the rat walks down the maze in the first trial?

b. Calculate the Q-values after 3 complete trials. How many Q-values are non-zero? How many trials do we need so that information about the reward has arrived in the state just 'below' the starting state?

c. What happens to the learning speed if the number of states increases from 6 to 12? How many Q-values are non-zero after 3 trials? How many trial do we need so that information about the reward has arrived in the state just 'below' the starting state?

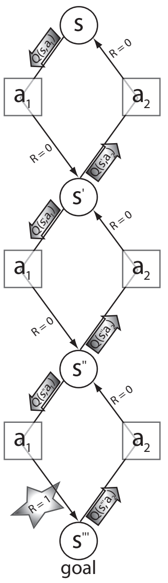


Figure 1: A linear maze.

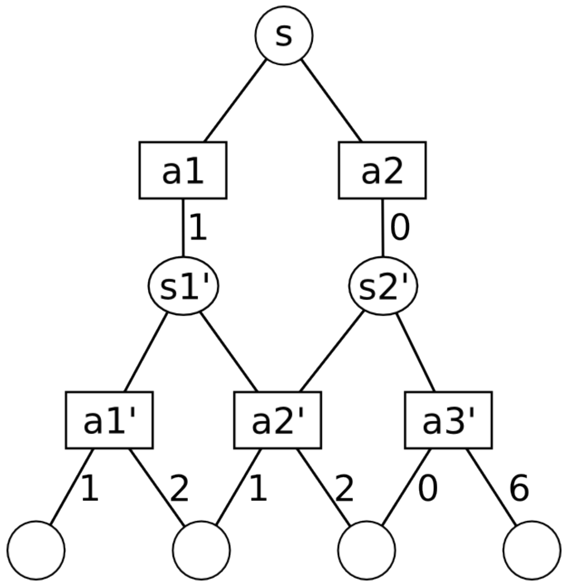


Figure 2: A tree–like environment.

Solution:

a. In the first trial, since all Q's are zero, the term $(Q(s,a) - Q(s',a'))$ is always zero. Learning only occurs when there is a reward ie, the first time action a_1 is taken from state s_5. The learning is then

$$\Delta Q(s_5,a_1) = \eta\,[r - (Q(s_5,a_1) - Q(s_6,a_2))] = \eta\,, \tag{3}$$

so that now all Q are zero except for $Q(s_5,a_1) = \eta$.

b. In the second trial, the first time $\Delta Q(s,a)$ is not zero is when the agent takes action a_1 from state s_4, and we have

$$\Delta Q(s_4,a_1) = \eta\,[r - (Q(s_4,a_1) - Q(s_5,a_1))] = \eta(0 - (0 - \eta)) = \eta^2\,. \tag{4}$$

Next, from state s_5, the agent chooses the action with the highest Q value, a_1, and the weight update is

$$\Delta Q(s_5,a_1) = \eta\,[r - (Q(s_5,a_1) - Q(s_6,a_2))] = \eta(1 - (\eta - 0)) = \eta - \eta^2\,. \tag{5}$$

So at the end of the second trial, the non-zero Qs are:

$$Q(s_4,a_1) = \eta^2 \qquad \text{and} \qquad Q(s_5,a_1) = 2\eta - \eta^2\,.$$

In the third trial, the first Q update happens for $Q(s_3,a_1)$

$$\Delta Q(s_3,a_1) = \eta\,[r - (Q(s_3,a_1) - Q(s_4,a_1))] = \eta(0 - (0 - \eta^2)) = \eta^3\,. \tag{6}$$

The subsequent updates are

$$\begin{aligned}\Delta Q(s_4,a_1) &= \eta\,[r - (Q(s_4,a_1) - Q(s_5,a_1))] = \eta(0 - (\eta^2 - 2\eta + \eta^2)) = 2(\eta^2 - \eta^3)\\ \Delta Q(s_5,a_1) &= \eta\,[r - (Q(s_5,a_1) - Q(s_6,a_2))] = \eta(1 - (2\eta - \eta^2 - 0)) = \eta - 2\eta^2 + \eta^3\,.\end{aligned}$$

So after three trials, the Qs are:

$$Q(s_3,a_1) = \eta^3\,, \qquad Q(s_4,a_1) = 3\eta^2 - 2\eta^3 \qquad \text{and} \qquad Q(s_5,a_1) = 3\eta - 3\eta^2 + \eta^3\,.$$

Note that terms for all the Qs converge towards 1 (the reward after). The higher η is, the faster the convergence, with convergence in 1 step in the extreme case $\eta = 1$.

We need one more trial until $Q(s_2,a_1)$ becomes non-zero, i.e. in total 4 trials.

c. Also with 12 states only 3 Q-values are non-zero after 3 trials. It takes 10 trials until the reward has arrived just 'below' the starting state.

Exercise 2: Bellman equation

Use the Bellman equation to calculate $Q(s,a1)$ and $Q(s,a2)$ for the environment shown in Figure 2. Consider two different policies:

- Total exploration: All actions are chosen with equal probability.
- Greedy exploitation: The agent always chooses the best action.

Note that the rewards/next states are stochastic for the actions $a1'$, $a2'$ and $a3'$. Assume that the probabilities for the outcome of these actions are all equal, and the discount factor γ is 1.

Solution:

Total exploration: Since we have a directed graph without loops, the Bellman equation is solved via dynamic programming, starting at the bottom of the tree We start by computing the state-action values for states s'_1 and s'_2:

$$\begin{aligned}
Q(s'_1, a'_1) &= \frac{1}{2}(1+2) = \frac{3}{2}\,, \\
Q(s'_1, a'_2) &= \frac{1}{2}(1+2) = \frac{3}{2}\,, \\
Q(s'_2, a'_2) &= \frac{1}{2}(1+2) = \frac{3}{2} \qquad \text{and} \\
Q(s'_2, a'_3) &= \frac{1}{2}(0+6) = 3.
\end{aligned}$$

We can now compute the state-action values for state s:

$$\begin{aligned}
Q(s, a_1) &= 1 + \frac{1}{2}(Q(s'_1, a'_1) + Q(s'_1, a'_2)) = \frac{5}{2} \qquad \text{and} \\
Q(s, a_2) &= 0 + \frac{1}{2}(Q(s'_2, a'_2) + Q(s'_2, a'_3)) = \frac{9}{4}\,.
\end{aligned}$$

Greedy exploitation: In that case, the state-action values for the s'_1 and s'_2 are unchanged, but those for s reflect the fact that we now take the best action:

$$\begin{aligned}
Q(s, a_1) &= 1 + Q(s'_1, a'_1) = \frac{5}{2} \qquad \text{and} \\
Q(s, a_2) &= 0 + Q(s'_2, a'_3) = 3\,.
\end{aligned}$$

Notice that now the "best" action in state s is a_2, whereas it was a_1 for the total exploration policy.

Exercise 3: Expected SARSA and a new variant of TD learning

We have seen the algorithm 'Expected Sarsa' and 'TD-algorithm in the narrow sense'. Suppose you invent a new algorithm that keeps tables of both Q-values and V-values. To do so consider the following:

a. Rewrite the updated step of 'Expected Sarsa' using both Q-values and V-values in the update rule.

b. Write down the full pseudo-algorithm.

   Hint: (i) keep track of the sequence of update steps of Q-values and V-values in your new algorithm. (ii) At which point in the sequence of online steps through the graph can you update $Q(s, a)$ and how far do you have to look backward?

c. Sketch the 'back-up-diagram' of your algorithm and compare it to that of SARSA. What are the costs and benefits of your new algorithm compared to SARSA?

Solution:

a. The update rule is given by

$$
\begin{aligned}
Q(s_t, a_t) &\leftarrow (1-\eta)\cdot Q(s_t, a_t) + \eta \cdot \Big(r_t + \gamma V(s_{t+1})\Big) \\
V(s_t) &\leftarrow (1-\eta)\cdot V(s_t) + \eta \cdot \Big(r_t + \gamma \sum_{a'} \pi(a'|s_{t+1}) Q(s_{t+1}, a')\Big)
\end{aligned}
$$

b. Pseudo-code:

```
• Initialize $Q(s,a)$ and $V(s)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$ with $V(term) = Q(term, .) = 0$
• Repeat (for each episode):
    – Initialize $s$
    – Choose $a$ from $s$ using policy derived from $Q$
    – Repeat (for each step of episode):
        ∗ Take action $a$, observe $r$ and $s'$
        ∗ $Q(s,a) \leftarrow (1-\eta)\cdot Q(s,a) + \eta \cdot \Big(r + \gamma V(s')\Big)$
        ∗ $V(s) \leftarrow (1-\eta)\cdot V(s) + \eta \cdot \Big(r + \gamma \sum_{a'} \pi(a'|s') Q(s', a')\Big)$
        ∗ Choose $a'$ from $s'$ using policy derived from $Q$
        ∗ $s \leftarrow s'$ and $a \leftarrow a'$
    – until $s$ is terminal.
```

c. Cost: You need to keep (and update) two tables instead of a single one.

Benefit: you can update the Q-value of (s,a) immediately at state s' (without playing round with potential next actions) because you already updated the V-value last time you visited the state s'. Hence the back-up diagram is shorter than for SARSA (you only need to keep (s, a, r, s') in the buffer whereas for SARSA you need to keep s, a, r, s', a').

Exercise 4: Monte Carlo versus expected SARSA[3]

We compare two algorithms: The Batch-Monte-Carlo algorithm (around slide 47), and the Online-Expected-SARSA (around slide 19). The aim of the exercise is to understand why bootstrap algorithms (i.e., TD algorithms) perform, under some conditions, more efficiently than a naive estimation of Q-Values via Monte-Carlo algorithms. We set the discount factor to $\gamma = 1$ and run 5 episodes in a given environment:

Each episode starts in one of the states 1 or 2 or 3 with action a_1. In state 4 there is a choice between actions a_2 and a_3 which are taken with equal probability $\pi = 0.5$. The transition sequence and total return for each of the 5 episodes is given in the figure above. All rewards are deterministic and only depend on the transition (s, a, s').

a. Calculate the Q-values in state 1, 2, 3, and 4 using Batch-Monte-Carlo (i.e., average total returns from each starting state).

b. Calculate the Q-values in state 1, 2, 3, and 4 using Online-Expected SARSA. For a given Q-value $Q(s,a)$, use $\eta_1 = 1$ the FIRST TIME you update this value and $\eta \in [0, 0.5]$ for all later update steps.

Hint: You can neglect terms of order η^2.

[3]The result of Exercise 4 will be used in the second lecture of this week.

5 episodes, first action is always a1.

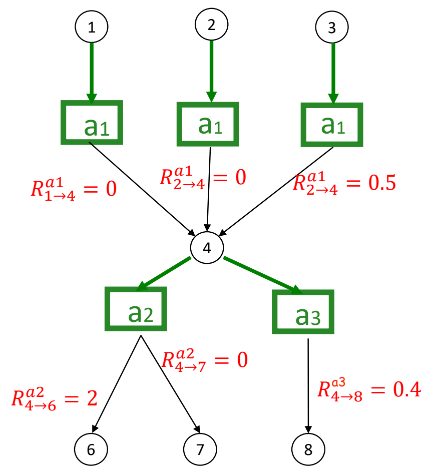
Episode 1: States 1-4-7 with action a2, Return=0

Episode 2: States 1-4-8 with action a3, Return=0.4

Episode 3: States 2-4-6 with action a2, Return=2

Episode 4: States 2-4-8 with action a3, Return=0.4

Episode 5: States 3-4-7 with action a2, Return=0.5



c. You can choose the initial state for episode 6. Which initial state looks best in case a (Batch-Monte-Carlo)? Which initial state looks best in case b (Online-Expected SARSA)?

Where does the difference come from? Which solution is closer to the exact Q-values under the $\pi = 0.5$ policy in state 4? Why?

d. What happens qualitatively if the order of episodes 1 and 2 were switched? What would be the value of $Q(s = 1, a_1)$ in this case? How would the other Q-values change?

Solution:

a. Batch-Monte-Carlo:

- $Q(1, a_1) = \frac{1}{2}(0 + 0.4) = 0.2$
- $Q(2, a_1) = \frac{1}{2}(2 + 0.4) = 1.2$
- $Q(3, a_1) = 0.5$
- $Q(4, a_2) = \frac{1}{3}(0 + 2 + 0) = 2/3$
- $Q(4, a_3) = \frac{1}{2}(0.4 + 0.4) = 0.4$

b. Online Expected SARSA:

Episode 1:

- $Q(1, a_1) \leftarrow 0 \cdot Q(1, a_1) + 1 \cdot \left(0 + \frac{Q(4,a_2)+Q(4,a_3)}{2}\right) = 0$
- $Q(4, a_2) \leftarrow 0 \cdot Q(4, a_2) + 1 \cdot 0 = 0$

Episode 2:

- $Q(1, a_1) \leftarrow (1 - \eta) \cdot Q(1, a_1) + \eta \cdot \left(0 + \frac{Q(4,a_2)+Q(4,a_3)}{2}\right) = 0$
- $Q(4, a_3) \leftarrow 0 \cdot Q(4, a_3) + 1 \cdot 0.4 = 0.4$

Episode 3:

- $Q(2, a_1) \leftarrow 0 \cdot Q(2, a_1) + 1 \cdot \left(0 + \frac{Q(4,a_2)+Q(4,a_3)}{2}\right) = 0.2$
- $Q(4, a_2) \leftarrow (1 - \eta) \cdot Q(4, a_2) + \eta \cdot 2 = 2\eta$

Episode 4:

- $Q(2, a_1) \leftarrow (1 - \eta) \cdot Q(2, a_1) + \eta \cdot \left(0 + \frac{Q(4,a_2)+Q(4,a_3)}{2}\right) = 0.2 + \eta^2 \approx 0.2$
- $Q(4, a_3) \leftarrow (1 - \eta) \cdot Q(4, a_3) + \eta \cdot 0.4 = 0.4$

Episode 5:

- $Q(3,a_1) \leftarrow 0 \cdot Q(3,a_1) + 1 \cdot \left(0.5 + \frac{Q(4,a_2)+Q(4,a_3)}{2}\right) = 0.7 + \eta$
- $Q(4,a_2) \leftarrow (1-\eta) \cdot Q(4,a_2) + \eta \cdot 0 = 2\eta - 2\eta^2 \approx 2\eta.$

As a result:

$$Q(1,a_1) = 0; \quad Q(2,a_1) = 0.2; \quad Q(3,a_1) = 0.7 + \eta; \quad Q(4,a_2) = 2\eta; \quad Q(4,a_3) = 0.4$$

c. With Batch MC state 2 looks best, but with online expected SARSA state 3 looks best.

Also expected SARSA is closer to the exact solution for Q-values:

$$Q(1,a_1) = 0.7; \quad Q(2,a_1) = 0.7; \quad Q(3,a_1) = 1.2; \quad Q(4,a_2) = 1; \quad Q(4,a_3) = 0.4$$

Note that with a choice $\eta = 0.5$ or $\eta = 0.3$, the difference between the Expected SARSA and the excat solution is really small.

The reason is that for the update of Q-values for states 2 and 3, SARSA uses ALL previous trials that involve state 4, and hence uses better statistics for 4 than the Batch MC algorithm.

d. then the return of 0.4 of the new first episode would influence the Q-value of state 4 used to update the Q-value of state 1 in the second episode. The other Q-values would not change Hence, we would have

$$Q(1,a_1) = 0.2\eta; \quad Q(2,a_1) = 0.2; \quad Q(3,a_1) = 0.7 + \eta; \quad Q(4,a_2) = 2\eta; \quad Q(4,a_3) = 0.4$$

Conclusion: TD-algorithms make better use of the data since return information from intermediate states is shared!

Exercise 5: Computer exercises: Environment 1 (part 1)

Download the Jupyter notebook of the 1st computer exercise, setup your Python environment, and complete ‘Exercise 0: One step horizon’.