Reinforcement Learning - Assignment 3

Thalagalage Don Lahiru, Nazanin Baramaki

1^{st} of Aug 2025

You can find the source code on GitHub: View Source Code on GitHub

1 Environment Setup

Grid-world environment of 5 x 5 as below:

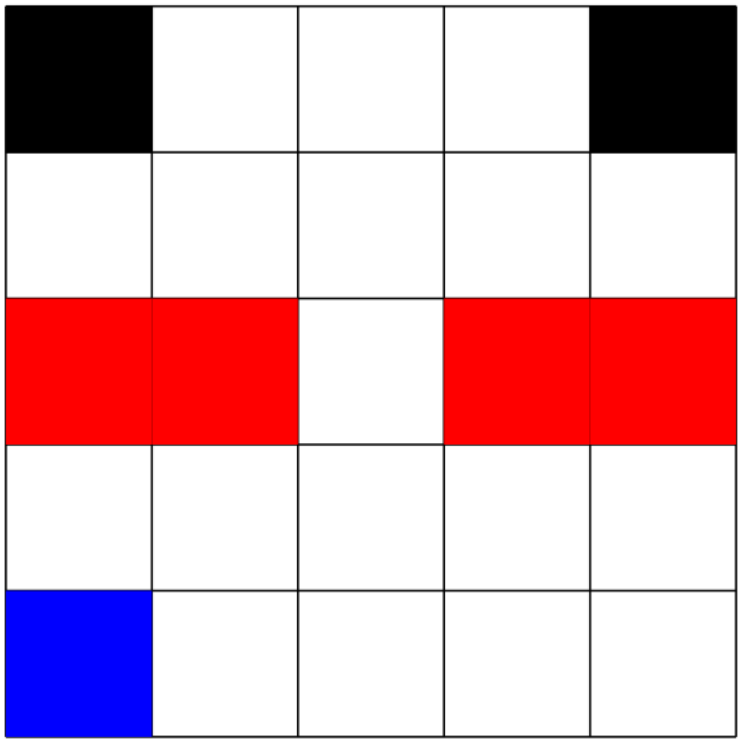

Figure 1: Grid World

Each of the 25 cells of the grid-world represents a possible state of the world. Grid-world is defined in the 'grid_world.py' file as a multi-dimensional array, where W, B, R and T represent the white, blue, red, and black grid (terminal), respectively.

An agent in the grid_world can take a step up, down, left or right, which is represented by $\uparrow$, $\downarrow$, $\leftarrow$ and $\rightarrow$ respectively.

Given a current action pair (s, a), a tuple is returned (p, $s^{'}$, r, t), which is the result of the model that used to model the dynamics of the environment, given an action space and a map layout is defined. p represents the probability of transitioning from state s to an new state $s\prime$ after taking action a. t is used to indicate whether the next state $s\prime$ is a terminal state.

Each episode begins in the blue cell. It works as follows: At the top of each episode, the agent initializes this state to this blue grid and tries to figure out the best path to the goal. The agent has the freedom to migrate through blue and white cells. Moving outside the grid is also rewarded by -1. The cost of entering a red cell is -20, and the agent gets restarted at the blue start point. The end of an episode is marked by getting close to a black cell.

2 Algorithms

2.1 Sarsa

Sarsa algorithm is implemented according to the pseudocode given in the book as below.

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

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)
  - $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
  - $S \leftarrow S'; \quad A \leftarrow A'$
- until S is terminal

2.2 Q-learning

Similarly, this algorithm can be implemented according to the pseudocode given in the book.

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

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'
  - $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
  - $S \leftarrow S'$
- until S is terminal

3 Results

Exploring grid parameters consists of a discount factor $\boldsymbol{\gamma = 0.99}$, ϵ-greedy action selection with $\boldsymbol{\epsilon = 0.2}$, learning rate $\boldsymbol{\alpha = 0.5}$ and a policy with **equiprobable moves**. The results of the two exploration algorithms are shown in the following figures. Grey states are **terminal** states, red states are **high penalty** states, and blue state is the **start** state for the agent.

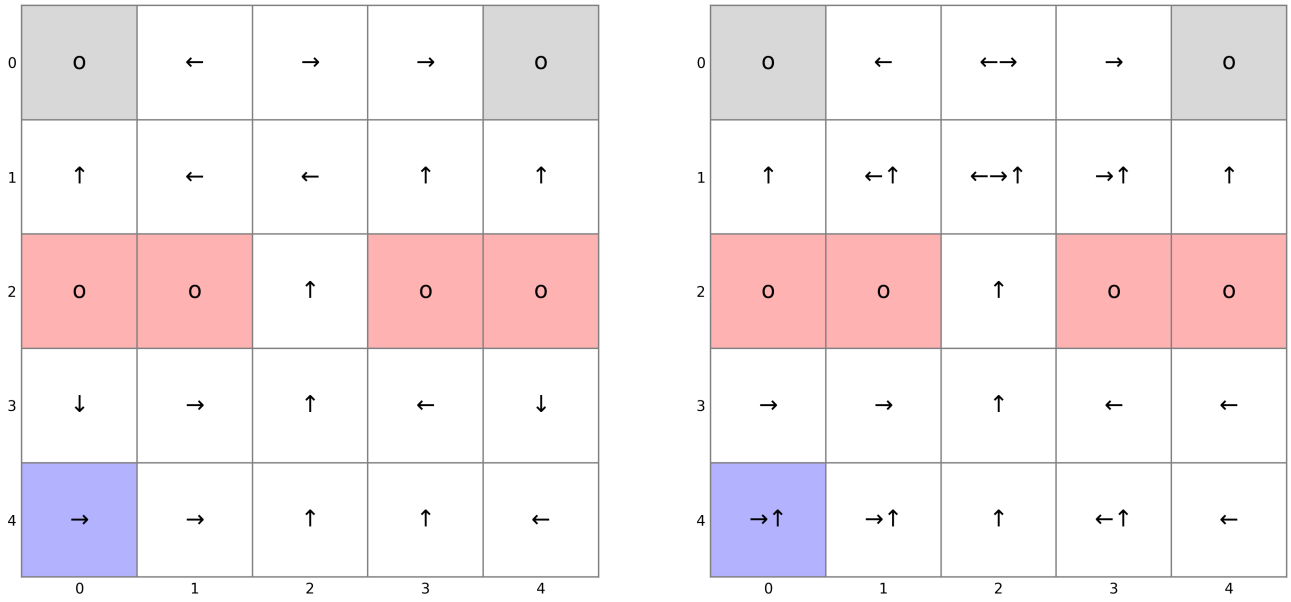

Figure 2: (Left) Sarsa; (Right) Q-learning Optimal Policy.

According to the optimal policy of the algorithms in Figure 2, we can observe that the **Q-learning algorithm performs better than the Sarsa algorithm**. The Q-learning algorithm, for the majority of the time, gives the correct choice of action in each state. Sarsa is less optimal at avoiding the penalty region compared to the Q-learning algorithm, **except in the bottom right state**.

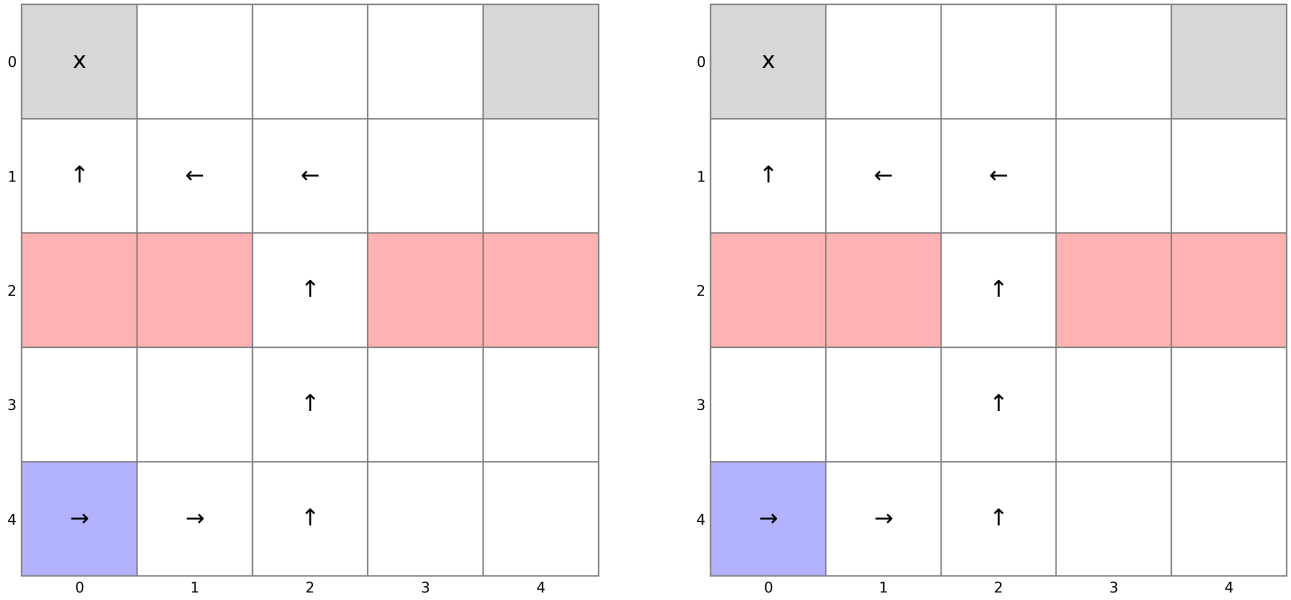

Figure 3: (Left) Sarsa; (Right) Q-learning Trajectory Under Optimal Policy.

Trajectories given by both algorithms are **generally similar**, as illustrated in Figure 3; both successfully avoid red zones and move to the terminal states with a minimum number of steps. We observe that the **cause** for that is the **symmetric nature of the grid-world**. As long as the agent does not try to move out, move back towards the starting point, or move to the red grid, any trajectory can be one of the best choices.

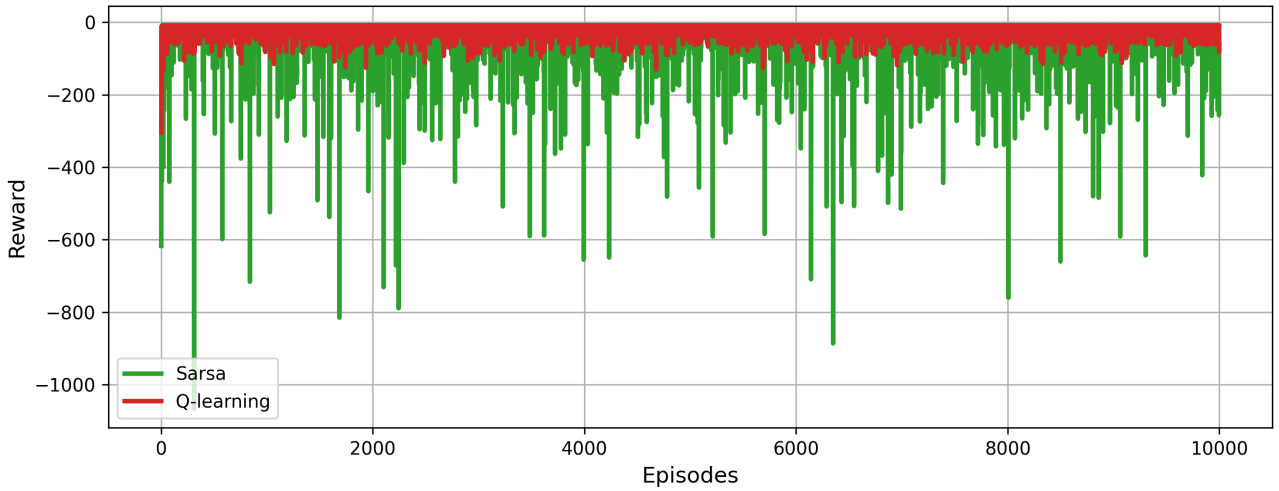

Figure 4: Sum of rewards

Figure 4 illustrates the sum of rewards over each episode for the Sarsa and Q-learning algorithms. It can be clearly observed that the sum of rewards of Q-learning is much higher than that of Sarsa. From these results, we can come to the conclusion that the Q-learning algorithm is less likely to move to the high penalty states and move rather quickly towards the terminal states than the Sarsa algorithm.