Temporal-Difference Learning

Bolivar Enrique Solarte Pardo

ID: 106061858

April 8, 2019

1. Introduction

The present report refers to homework 2 for EE5510 System Theory course, 2019 spring semester. The objective for this assignment is to implement the on-policy method SARSA and the off-policy method Q-learning. In the following sections, detail about their implementation, python code, and analysis over their performance is presented.

2. Implementation details

The implementation of this assignment considers the follow premises:

1. The environment is the classic Cliff-walking as shown in figure 1
2. Each simulation considers 500 episodes for both algorithms.

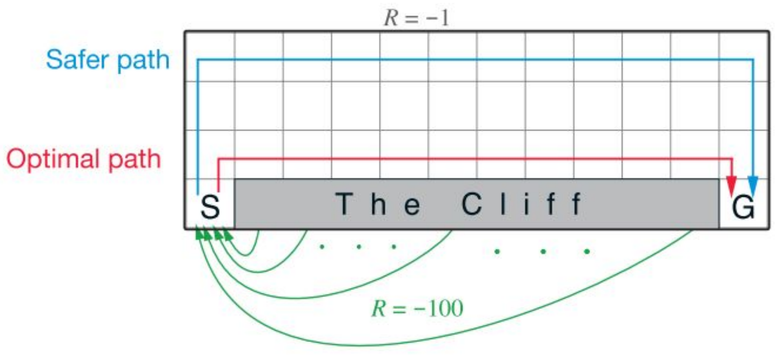

Figure 1. Cliff-walking environment ***[Sutton et. al. 2018]***

2.1 Q-Learning Implementation

The follow Q-learning implementation is based on equation (1). Such equation is evaluated by threefold: first, given a state **S** the action probabilities associated with this state are evaluated by a greedy-epsilon policy, then one action is chosen randomly, lastly, the corresponded reward and next state are evaluated (***see line 7 to 100***). Second, the reached reward is updated (line 13). Third, the target and the delta error are evaluated in lines 18 and 19. Finally, the equation is totally calculated in line 20.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]. \qquad (1)$$

```
      # start training
    for i_episode in range(num_episodes):
        # Reset the environment and pick the first action
        state = env.reset()

        for t in itertools.count():
            # Take a from the environment
            action_probs = policy(state)
            action = np.random.choice(np.arange(len(action_probs)), p=action_probs)
            next_state, reward, done, _ = env.step(action)

            # Update rewards
            episode_rewards[i_episode] += reward
            episode_lengths[i_episode] = t

            # Dynamic Update
            b_next_action_idx = np.argmax(Q[next_state])
            td_target = reward + discount_factor * Q[next_state][b_next_action_idx]
            td_delta = td_target - Q[state][action]
            Q[state][action] += alpha * td_delta

            if done:
                break

            state = next_state

    return Q, episode_rewards, episode_lengths
```

2.1 SARSA Implementation

Same as the previous section, SARSA implementation comprises by threefold: (1) take an action given a state-environment and evaluate the next state. **St+1**; (2) update rewards; and finally evaluation of the temporal difference equation for the SARSA algorithm, which is described by the equation (2).

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]. \quad (2)$$

```
# start training SARSA
    for i_episode in range(num_episodes):
        # Reset the environment and pick the first action
        state = env.reset()
        action_probs = policy(state)
        action = np.random.choice(np.arange(len(action_probs)), p=action_probs)

        for t in itertools.count():
            # Take action from state S and the env environment
            next_state, reward, done, _ = env.step(action)

            # Next action
            n_act_probs = policy(next_state)
            n_act_idx = np.random.choice(np.arange(len(n_act_probs)), p=n_act_probs)

            # Update rewards
            episode_rewards[i_episode] += reward
            episode_lengths[i_episode] = t

            # TD Update
            td_target = reward + discount_factor * Q[next_state][n_act_idx]
            td_delta = td_target - Q[state][action]
            Q[state][action] += alpha * td_delta

            if done:
                break

            action = n_act_idx
            state = next_state

    return Q, episode_rewards, episode_lengths
```

3. Experiments and Analysis

Figure 2 shows the performance of both methods, SARSA, and Q-Learning regarding the sum of rewards for each episode. In the same way, in figure 3 the episode length for each episode simulation is presented for both before mentioned methods. Lastly, the rendered trajectory for both methods is presented in figure 4.

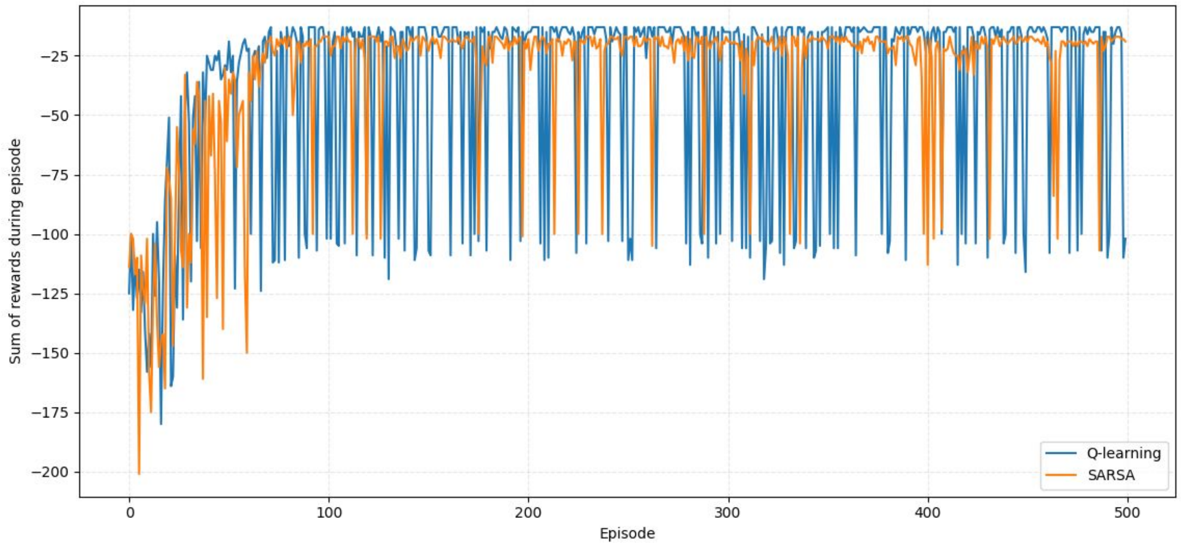

Figure 2. Sum of Rewards during episodes

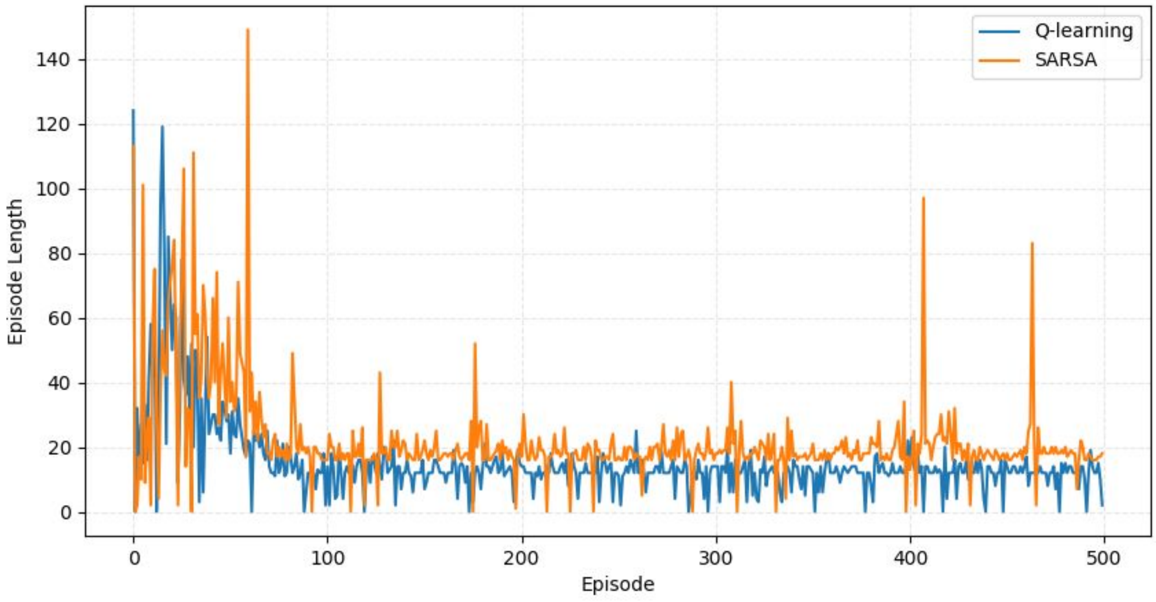

Figure 3.Episode length

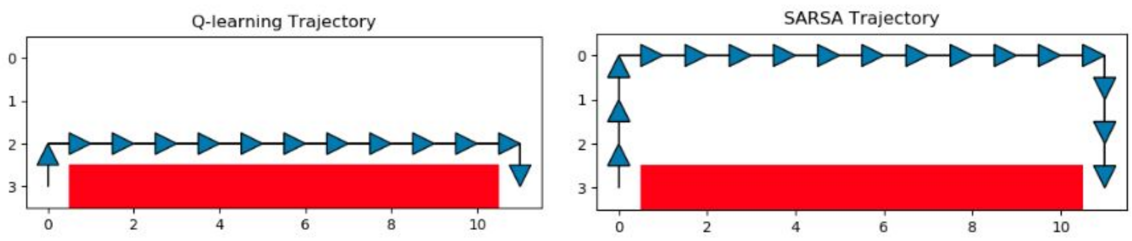

Figure 4. Rendered environment

3.1 Plot the episode length (time steps taken per episode) v.s. episode. What do you observe?

Considering the figure 3. Which shows the length episodes for each model (Sarsa and Q-Learning) we can conclude that the SARSA algorithm presents a episode length larger than Q-learning.

3.2 Render and show the trajectory of each method. What do you observe?

Figure 4 shows the rendered trajectories for both methods above described. From this trajectories, we can corroborate that the SARSA method draws a trajectory longer than Q-Learning. This trajectory is more conservative.

3.3 Observe the reward curve of each algorithm. We can observe that the reward curve of SRASA is more stable than Q-learning (less severe drop to -100). Please explain

Figure 2 shows the reached rewards for both models. Nevertheless, the plotting corresponding to Q-Learning reaches the highest sum of rewards, it also drops drastically for some episodes. We can explain easily this behavior if we consider the figure 4. In this figure, we can appreciate that the Q-Learning algorithm describes a trajectory closer of the cliff than SARSA. Thus, Q-Learning has more chances to drop into the cliff which means dropping rewards in figure 2.

3.4 Why is Q-learning considered an off-policy control method? How about SARSA?

While Q-learning is considering off-policy because it directly approximates the optimal action-value function Q (Q max[*]) independent of the policy being followed. SARSA method is considered on-policy beacuse it estimates the action-value function Q(s, a) from the states and actions being selected. This main difference can be appreciated in their respective equations (1), (2).

3.5 Vary the TD learning rate? what happens

The figures 5 and 6 show the rendered trajectory and performance plotting for both methods using different learning rates values. In figure 6, the botton-right figure shows how increasing the learning rate increases the length of the episodes. At the top-right figure, we can appreciate the same direct relation but decreasing the learning value. In the same figure 6, on the left side, we can appreciate how increasing the learning rate also increases the dropping rewards.

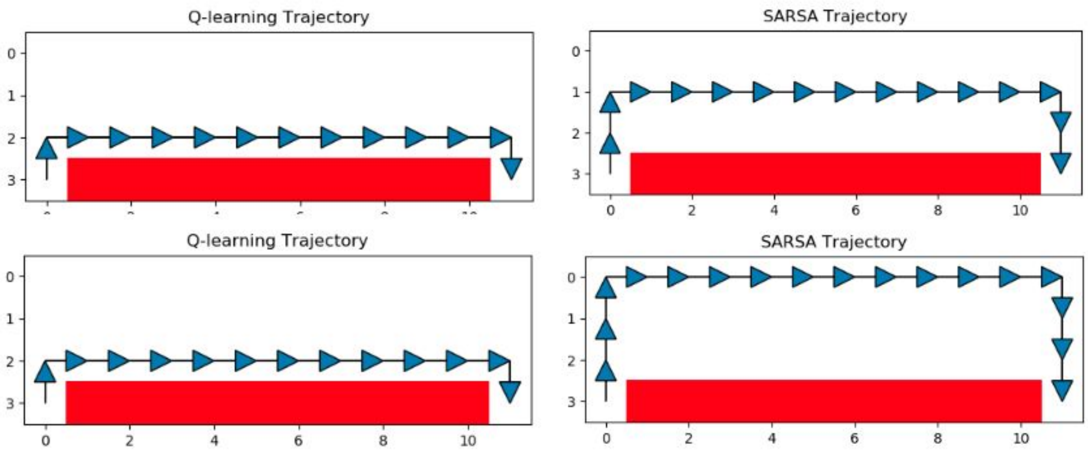

Figure 5. The rendered environment with different learning rates. First row, learning rate at 0.1. Second row, learning rate at 0.7.

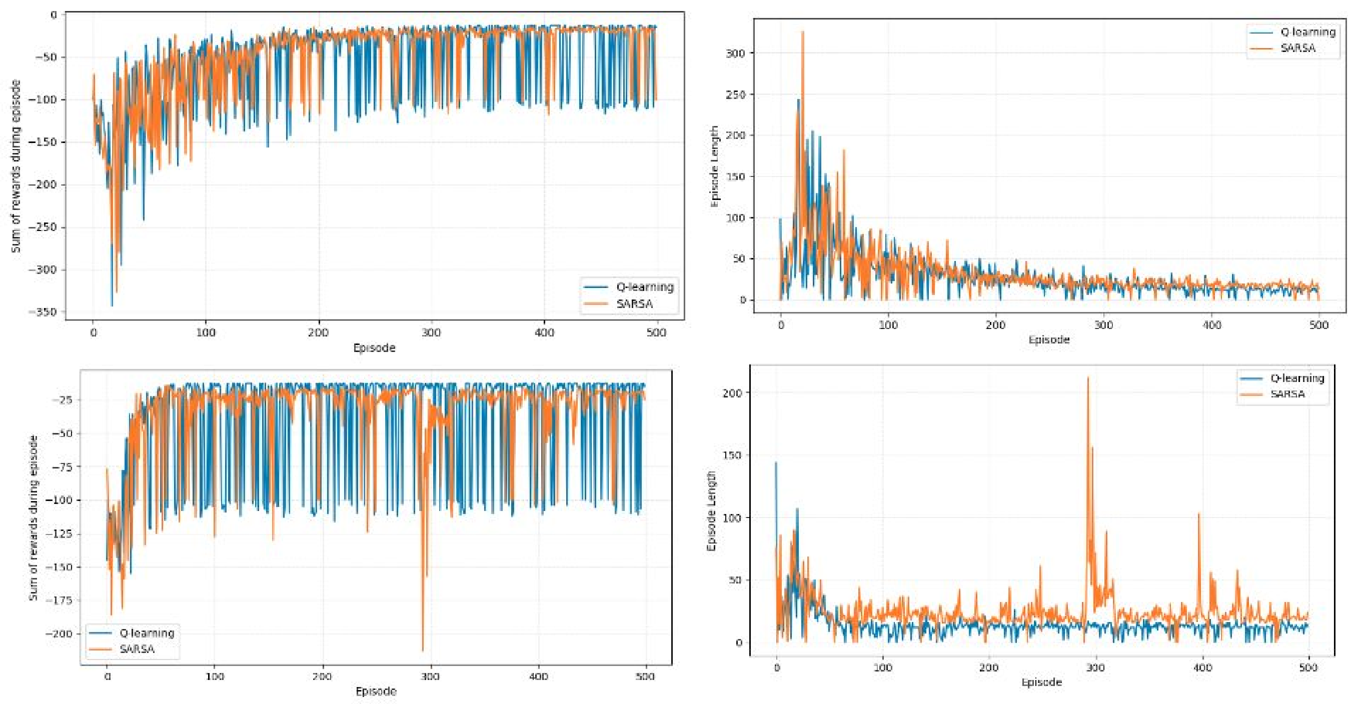

Figure 6. Plotting performance for SARSA and Q-Learning methods with different learning rates. First row, learning rate at 0.1. Second row, learning rate at 0.7.