

CSE584 Homework #2: Experimenting with SARSA and Q-Learning to Train an Agent in the Taxi-v3 Environment

Wahid Uz Zaman
wzz5219@psu.edu

Abstract

For this homework assignment, I worked with reinforcement learning algorithms, specifically SARSA and Q-learning, to train an agent to navigate the Taxi-v3 environment from the OpenAI Gym library. This environment simulates a taxi driver who picks up and drops off passengers at specific locations on a grid. This environment has predefined states, actions, and rewards that make my work easier as I didn't have to design the states, actions, and rewards manually. In Taxi-v3, there are 500 different states and 6 possible actions that represent how the taxi can move. The environment also has a `reset()` method that resets the environment randomly. I use this at the start of each episode, adding variety to both training and testing. The agent learns to make better decisions by interacting with the environment. Both SARSA and Q-learning use the epsilon-greedy policy for choosing actions and rely on a Q-table during training to learn optimal action-value pairs. I chose to run 10,000 training episodes for both SARSA and Q-learning to give the agent plenty of opportunities to learn. The code is designed to let users pick which algorithm they want to use for training as a command-line argument. After the training is complete, I evaluate the agent's performance by running test episodes and showing how it behaves in the environment.

RL Algorithm Implementation

I used $\alpha = 0.1$ as the learning rate, $\gamma = 0.9$ as the discount factor, and $\epsilon = 0.1$ as the exploration rate in both algorithms.

First, I will present my *choose_action* method, followed by the implementation of the SARSA and Q-learning algorithms.

choose_action

```
1 def choose_action(state, q_table, epsilon):
2     """
3     Choose an action based on epsilon-greedy policy.
4     Parameters:
5         - state: The current state of the agent.
```

```

6         - q_table: The Q-table containing Q-values for each state
          -action pair.
7         - epsilon: The exploration rate (probability of choosing
          a random action).
8     Returns:
9         - action: The selected action (either exploratory or
          exploitative).
10    """
11    if random.uniform(0, 1) < epsilon:
12        # choose a random action
13        action = random.choice(range(len(q_table[state])))
14    else:
15        # choose the action with the highest Q-value for the
          current state
16        action = np.argmax(q_table[state])
17
18    return action

```

SARSA_training

```

1 def SARSA_training(env, q_table, epsilon, gamma, alpha, episodes)
  :
2     # SARSA training loop
3     for episode in range(episodes):
4         # Reset the environment and get the initial state
5         state_info = env.reset()
6         state = state_info[0] if isinstance(state_info, tuple)
          else state_info
7         done = False # done means whether current episode is
          finished or not
8
9         # Choose initial action using epsilon-greedy policy
10        action = choose_action(state, q_table, epsilon)
11
12        # Loop until this episode is done
13        while not done:
14            # Perform the action and get the next state, reward,
              and done flag
15            step_info = env.step(action)
16            '''
17            step_info: (107, -1, False, False, {'prob': 1.0,
              'action_mask': array([1, 1, 1, 0, 0, 0], dtype
              =int8)})
18            107-> state number, -1 -> reward, False-> episode
              is not finished.
19            '''
20            next_state = step_info[0]
21            reward = step_info[1]
22            done = step_info[2]
23

```

```

24         # Choose the next action using epsilon-greedy policy.
25         next_action = choose_action(next_state, q_table,
26                                     epsilon)
27
28         # Update Q-value[state, action] using the Bellman
29         # equation for SARSA.
30         q_table[state, action] = q_table[state, action] +
31             alpha * (
32                 reward + gamma * q_table[next_state, next_action]
33                 - q_table[state, action]
34             )
35
36         # Move to the next state and action
37         state = next_state
38         action = next_action
39
40     print("SARSA training completed.")

```

Qlearning_training

```

1 def Qlearning_training(env, q_table, epsilon, gamma, alpha,
2     episodes):
3     # Q-learning training loop
4     for episode in range(episodes):
5         # Reset the environment and get the initial state
6         state_info = env.reset()
7         state = state_info[0] if isinstance(state_info, tuple)
8         else state_info
9         done = False # done means whether current episode is
10            finished or not
11
12        # Loop until this episode is done
13        while not done:
14            # Choose an action using epsilon-greedy policy.
15            action = choose_action(state, q_table, epsilon)
16
17            # Perform the action and get the next state, reward,
18            # and done flag
19            step_info = env.step(action)
20            next_state = step_info[0] if isinstance(step_info,
21            tuple) else step_info
22            reward = step_info[1]
23            done = step_info[2]
24
25            # Update Q-value[state, action] using the Bellman
26            # equation for Q-learning
27            q_table[state, action] = q_table[state, action] +
28                alpha * (
29                    reward + gamma * np.max(q_table[next_state]) -
30                    q_table[state, action]

```

```
23         )
24
25         # Update the state only.
26         state = next_state
27
28     print("Q-learning based Training completed.")
```