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. Full code is in here: taxi_grid_RL.py

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

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

```python
        - q_table: The Q-table containing Q-values for each state-action pair.
        - epsilon: The exploration rate (probability of choosing a random action).
    Returns:
        - action: The selected action (either exploratory or exploitative).
    """
    if random.uniform(0, 1) < epsilon:
        # choose a random action
        action = random.choice(range(len(q_table[state])))
    else:
        # choose the action with the highest Q-value for the current state
        action = np.argmax(q_table[state])

    return action
```

SARSA_training

```python
def SARSA_training(env, q_table, epsilon, gamma, alpha, episodes):
    # SARSA training loop
    for episode in range(episodes):
        # Reset the environment and get the initial state
        state_info = env.reset()
        state = state_info[0] if isinstance(state_info, tuple) else state_info
        done = False # done means whether current episode is finished or not

        # Choose initial action using epsilon-greedy policy
        action = choose_action(state, q_table, epsilon)

        # Loop until this episode is done
        while not done:
            # Perform the action and get the next state, reward, and done flag
            step_info = env.step(action)
            '''
                step_info: (107, -1, False, False, {'prob': 1.0, 'action_mask': array([1, 1, 1, 0, 0, 0], dtype=int8)})
                107-> state number, -1 -> reward, False-> episode is not finished.
            '''
            next_state = step_info[0]
            reward = step_info[1]
            done = step_info[2]

```

```
            # Choose the next action using epsilon-greedy policy.
            next_action = choose_action(next_state, q_table, epsilon)

            # Update Q-value[state, action] using the Bellman equation for SARSA.
            q_table[state, action] = q_table[state, action] + alpha * (
                reward + gamma * q_table[next_state, next_action] - q_table[state, action]
            )

            # Move to the next state and action
            state = next_state
            action = next_action

    print("SARSA training completed.")
```

Qlearning_training

```
def Qlearning_training(env, q_table, epsilon, gamma, alpha, episodes):
    # Q-learning training loop
    for episode in range(episodes):
        # Reset the environment and get the initial state
        state_info = env.reset()
        state = state_info[0] if isinstance(state_info, tuple) else state_info
        done = False # done means whether current episode is finished or not

        # Loop until this episode is done
        while not done:
            # Choose an action using epsilon-greedy policy.
            action = choose_action(state, q_table, epsilon)

            # Perform the action and get the next state, reward, and done flag
            step_info = env.step(action)
            next_state = step_info[0] if isinstance(step_info, tuple) else step_info
            reward = step_info[1]
            done = step_info[2]

            # Update Q-value[state, action] using the Bellman equation for Q-learning
            q_table[state, action] = q_table[state, action] + alpha * (
                reward + gamma * np.max(q_table[next_state]) - q_table[state, action]
```

```
            )

            # Update the state only.
            state = next_state

    print("Q-learning based Training completed.")
```