Batch: C2 Roll No.: 16010121221

Experiment / assignment / tutorial No.__7___

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

TITLE : 7 To implement Q-learning approach

AIM: To understand that Q-learning is a model-free, off-policy reinforcement learning that will find the best course of action.

Expected OUTCOME of Experiment: (Mention CO/CO's attained here): CO3

CO3 Apply different temporal difference learning policies

Books/ Journals/ Websites referred:

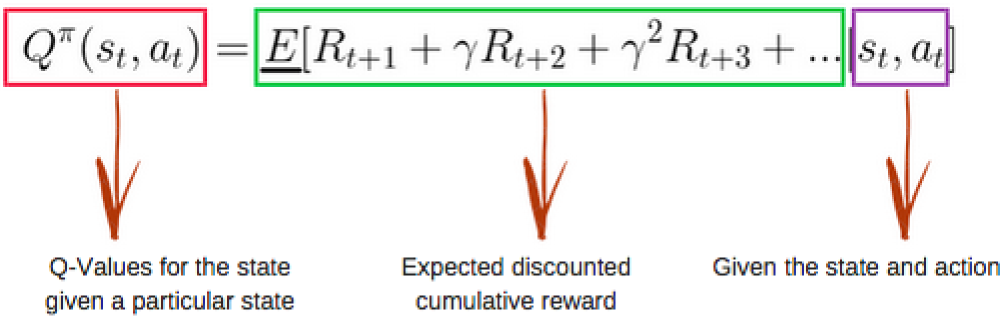
Richard S. Sutton and Andrew G. Barto, "*Reinforcement Learning:An Introduction",* The MIT Press,Second Edition, 2018

Pre Lab/ Prior Concepts:

In Q learning, we consider policy learning of action-values Q(s; a). No importance sampling is required. Next action is chosen using a behaviour policy.

Q-Learning is a basic form of Reinforcement Learning which uses Q-values (also called action values) to iteratively improve the behavior of the learning agent.

The q learning update rule is given as;



Chosen Problem Statement: Frozen Lake Problem

The **Frozen Lake** problem is a popular reinforcement learning problem where the agent (the learner) must navigate a frozen lake, represented as a grid, to reach the goal while avoiding holes (which represent falling through the ice). The environment is stochastic, meaning there is a chance the agent will slip and move in a direction different from the one intended. The objective is for the agent to learn a policy to maximize the cumulative reward by reaching the goal safely.

Explain following concepts w.r.t. chosen problem statement:

Policy:

A policy in reinforcement learning defines the behavior of the agent at each state. It tells the agent what action to take based on the current state. In the context of the Frozen Lake problem, a policy specifies the direction (up, down, left, right) the agent should move from any given tile on the frozen lake.

- In Q-learning: The policy is implicitly defined by the Q-values, where the agent chooses the action with the highest Q-value in each state.

In the Frozen Lake problem, a good policy would be one where the agent successfully reaches the goal while avoiding slipping into holes as frequently as possible.

Reward function:

A reward function provides feedback to the agent based on its actions. In the Frozen Lake problem:

- The agent receives a reward of +1 for reaching the goal.
- The reward is 0 for any other state, including stepping on the frozen lake tiles or falling into a hole.
- In Q-learning: The agent updates its Q-values based on the rewards received for the actions taken, adjusting its behavior to maximize future rewards.

Value function:

The value function in reinforcement learning estimates the total amount of reward the agent can expect to accumulate starting from a particular state (or state-action pair). There are two types:

- State value function (V(s)): Measures the expected reward from being in state `s` and following a policy.
- Action value function (Q(s,a)): Measures the expected reward from taking action `a` in state `s` and following a policy thereafter.
- In Q-learning: The agent estimates the Q-value for each state-action pair iteratively based on the rewards and the maximum future Q-values. The Q-values are updated using the following rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

Here, `α` is the learning rate, `γ` is the discount factor, `r` is the immediate reward, and `max Q(s', a')` represents the maximum Q-value for the next state.

Model of the environment:

In reinforcement learning, the model of the environment refers to a representation of how the environment behaves, including the state transitions and the rewards for those transitions.

- In Q-learning: It is a model-free method, meaning it does not require any knowledge about the environment's dynamics (i.e., how states transition or the exact reward structure). The agent learns purely from interactions with the environment, updating Q-values without needing to model state transitions.

In the Frozen Lake problem, Q-learning is advantageous as it does not rely on a precise model of how the agent will slip or transition between tiles. In contrast, Dynamic Programming requires this knowledge to compute the optimal policy.

Implementation:

```
import gym

import numpy as np



# Initialize the Frozen Lake environment

env = gym.make('FrozenLake-v1', is_slippery=True)    # set
is_slippery=False for a deterministic environment



# Define the Q-table dimensions (states x actions)

action_space_size = env.action_space.n   # Number of possible
actions

state_space_size = env.observation_space.n  # Number of states



# Create Q-table and initialize it with zeros
```

```
q_table = np.zeros((state_space_size, action_space_size))


# Hyperparameters

learning_rate = 0.1  # Alpha

discount_rate = 0.99  # Gamma

exploration_rate  =  1.0    #  Epsilon  for  exploration-exploitation
trade-off

max_exploration_rate = 1.0

min_exploration_rate = 0.01

exploration_decay_rate = 0.001


# Training parameters

num_episodes = 10000

max_steps_per_episode = 100


# List to hold the rewards for each episode

rewards_all_episodes = []


# Q-learning algorithm

for episode in range(num_episodes):

    state = env.reset()[0]  # Get initial state

    done = False

    rewards_current_episode = 0
```

```
    for step in range(max_steps_per_episode):
        # Exploration-exploitation trade-off
        exploration_rate_threshold = np.random.uniform(0, 1)
        if exploration_rate_threshold > exploration_rate:
            action = np.argmax(q_table[state, :])   # Exploit (choose action with highest Q-value)
        else:
            action = env.action_space.sample()   # Explore (choose random action)

        # Take action and observe the result
        new_state, reward, done, _, _ = env.step(action)

        # Update Q-table using the Q-learning formula
        q_table[state, action] = q_table[state, action] + learning_rate * (reward + discount_rate * np.max(q_table[new_state, :]) - q_table[state, action])

        # Transition to the next state
        state = new_state

        # Accumulate the rewards
        rewards_current_episode += reward
```

```
        # End the episode if done
        if done:
            break

    # Exploration rate decay
    exploration_rate = min_exploration_rate + (max_exploration_rate
- min_exploration_rate) * np.exp(-exploration_decay_rate * episode)

    # Append the reward for the current episode
    rewards_all_episodes.append(rewards_current_episode)

# Print the Q-table after training
print("Q-table after training:")
print(q_table)

# Calculate and print average reward per 1000 episodes
rewards_per_thousand_episodes                                      =
np.split(np.array(rewards_all_episodes), num_episodes / 1000)
count = 1000

print("\n********Average reward per thousand episodes********\n")
for r in rewards_per_thousand_episodes:
```

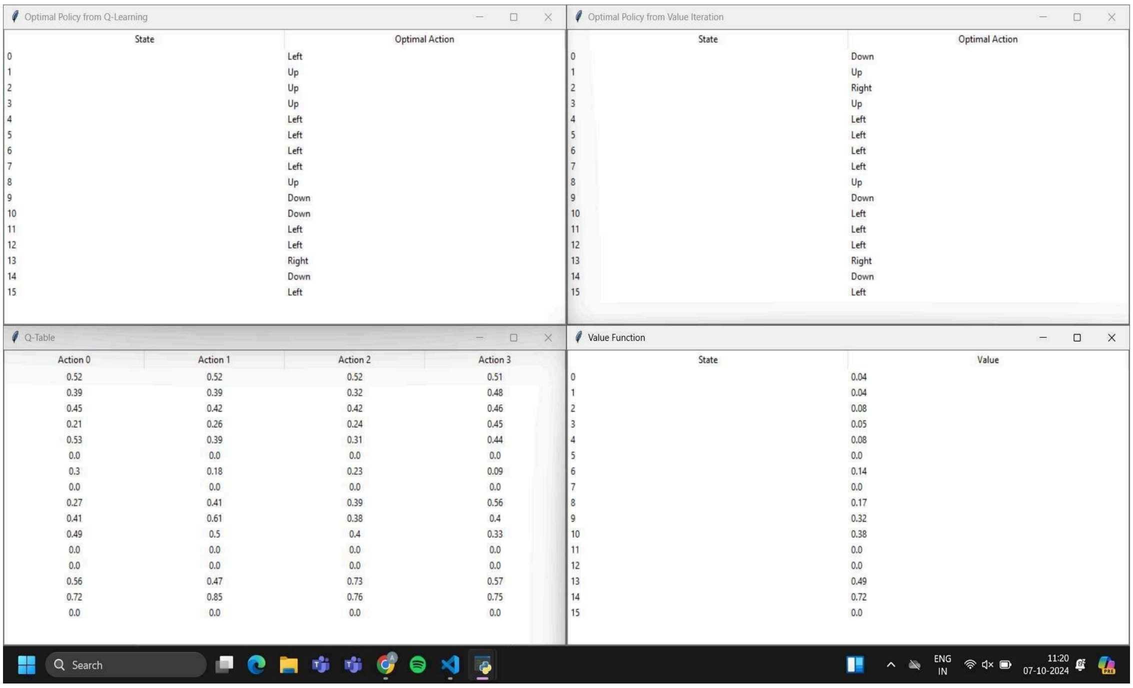
```
    print(count, ": ", str(sum(r / 1000)))
    count += 1000


# Watch the trained agent in action
for episode in range(3):
    state = env.reset()[0]
    done = False
    print("Episode", episode + 1)
    for step in range(max_steps_per_episode):
        env.render()  # Render the environment


           action = np.argmax(q_table[state, :])   # Take the best
action
        new_state, reward, done, _, _ = env.step(action)


        if done:
            break
        state = new_state


env.close()
```



State	Optimal Action
0	Left
1	Up
2	Up
3	Up
4	Left
5	Left
6	Left
7	Left
8	Up
9	Down
10	Down
11	Left
12	Left
13	Right
14	Down
15	Left

State	Optimal Action
0	Down
1	Up
2	Right
3	Up
4	Left
5	Left
6	Left
7	Left
8	Up
9	Down
10	Left
11	Left
12	Left
13	Right
14	Down
15	Left

Action 0	Action 1	Action 2	Action 3
0.52	0.52	0.52	0.51
0.39	0.39	0.32	0.48
0.45	0.42	0.42	0.46
0.21	0.26	0.24	0.45
0.53	0.39	0.31	0.44
0.0	0.0	0.0	0.0
0.3	0.18	0.23	0.09
0.0	0.0	0.0	0.0
0.27	0.41	0.39	0.56
0.41	0.61	0.38	0.4
0.49	0.5	0.4	0.33
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.56	0.47	0.73	0.57
0.72	0.85	0.76	0.75
0.0	0.0	0.0	0.0

State	Value
0	0.04
1	0.04
2	0.08
3	0.05
4	0.08
5	0.0
6	0.14
7	0.0
8	0.17
9	0.32
10	0.38
11	0.0
12	0.0
13	0.49
14	0.72
15	0.0

Conclusion:

Q-learning is a **model-free**, off-policy reinforcement learning algorithm that iteratively updates Q-values to find the optimal policy. It is suitable for environments like Frozen Lake, where the agent learns to act optimally even without a model of the environment's dynamics. Q-learning relies on exploration (through ε-greedy strategies) and exploitation (choosing the best-known action).

Post Lab Descriptive Questions:

1. Differentiate between Q learning and SARSA.

Q-learning and SARSA (State-Action-Reward-State-Action) are two popular reinforcement learning (RL) algorithms. Both are model-free and based on temporal difference (TD) learning, but they differ in how they update the action-value function, which affects their behavior.

Key Differences:

1. **Off-Policy vs. On-Policy**:

  - **Q-learning** is off-policy, meaning it learns from a **greedy policy** (best possible actions), even if the agent is exploring during training.
  - **SARSA** is on-policy, meaning it learns from the **current policy** the agent follows, which could include exploration.
2. **Action Selection in Next State**:
  - In **Q-learning**, the agent selects the **maximum Q-value** action in the next state (greedy approach), regardless of the action the agent actually takes.
  - In **SARSA**, the agent updates based on the **actual action** it chooses in the next state, which could be an exploratory action.
3. **Risk and Stability**:
  - **Q-learning** tends to favor more **risky, optimal actions** because it always updates using the maximum Q-value for the next state.
  - **SARSA** is more **conservative**, updating based on what the agent actually does, making it more stable in environments where randomness or exploration is critical.

Q-learning Update Rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

- The update is based on the **best action** a′a'a′ (greedy) in the next state s′s's′.

SARSA Update Rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma Q(s',a') - Q(s,a) \right]$$

- The update is based on the **actual action** a′a'a′ that the agent took in the next state s′s's′.

Date: 07-10-2024 **Signature of faculty in-charge**