**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering**

**Department of Aritificial Intelligence**

Program: B.Tech/MBA Tech AI Semester: VII

**Course: Reinforcement Learning**

**List of Experiments**

**Faculty:** Dr. Ami Munshi W.E.F. July 2024

|  |  |  |  |
| --- | --- | --- | --- |
| **Exp No.** | **Title** | **Prerequisite\*** | **CO#** |
| 1 | **Introduction to RL terminologies and Elements using OpenAI Gym**   1. Explore various environments in Open AI Gym 2. Relate RL terminologies and elements with the examples in Gym Environment 3. Implement any Gym Environment from classic control 4. Customize the above gym environment parameters 5. Demonstrate how to interpret state transitions, rewards, and termination conditions with reference to the above example | Python programming  RL terminologies | 1 |
| 2. | **Implementation of Multi-Arm Bandit (MAB) Problem**   1. To implement MAB problem using pure exploitation algorithm. 2. To implement MAB problem using pure exploration algorithm 3. To implement MAB problem using Fixed Exploration followed by Exploitation 4. To implement MAB problem using Greedy algorithm 5. To apply Upper Confidence Bound(UCB) in the implementation done in part d 6. To analyse the algorithms and compare them in terms of rewards, regrets and complexity | Concept of Multi-Arm Bandit Problem in RL | 1,2 |
| 3. | **Understand and apply the concepts of Markov Decision Processes (MDPs) by modeling a real-world decision-making scenario**   1. Model decision making process 2. Implement value iteration algorithm 3. Identify the optimal policy by analysing the results | Concept of Markov Property, Markov Chain, Markov Decision Process | 2,3 |
| 4 | **Dynamic Programming- Policy Iteration Algorithm for MDP**   1. To find the optimal policy that maximizes the expected cumulative reward for a given MDP by iteratively improving an initial policy until convergence 2. To analyse the impact of discount factors for myopic and farsighted agents | Policy Iteration |  |
| 5 | **Dynamic Programming- Value Iteration Algorithm for MDP**   1. To compute the optimal policy by iteratively improving the value function for each state and selecting the action that maximizes the expected return 2. To analyse the impact of discount factors for myopic and farsighted agents 3. To compare value iteration and policy iteration algorithms in terms of time and computational complexity | Value Iteration |  |
| 6 | **Monte-Carlo Prediction**   1. To implement First Visit Monte Carlo Prediction Algorithm 2. To implement Every Visit Monte Carlo Prediction Algorithm 3. To compare First Visit Algorithm and Every Visit Algorithm on the basis of following points    * Time complexity    * Value Table for each Episode |  |  |
| 7 | **Temporal Difference Prediction**   1. To implement Temporal Difference (0) in a complex grid world environment 2. To implement Temporal Difference (1) in a complex grid world environment 3. To compare Temporal Difference Prediction Algorithm with Monte-Carlo Prediction Algorithm |  |  |
| 8 | **Q Learning and SARSA**   1. To implement Q Learning Algorithm in a complex grid world environment 2. To implement SARSA Algorithm in a complex grid world environment 3. To compare Q learning and SARA Algorithm | On policy and Off Policy |  |
| 9 |  |  |  |
| 10 |  |  |  |
| 11 |  |  |  |

\* Students are expected to be ready with the prerequisite before attending the lab

**Experiment No.06=7**

PART A

(PART A: TO BE REFFERED BY STUDENTS)

**A.1 Aim:**  **Q Learning and SARSA**

1. To implement Q Learning Algorithm in a complex grid world environment
2. To implement SARSA Algorithm in a complex grid world environment
3. To compare Q learning and SARA Algorithm

**A.2 Prerequisite:**

Concept of on-policy and off-policy predicition

**A.3 Learning Outcome:**

After completing this experiment you will be able to-

* Appreciate the need for Q learning and SARSA algorithms

**A.4 Theory:**

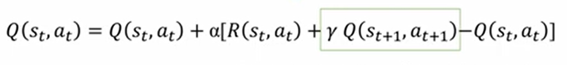
**A.4.1 Q Learning**

**Update Rule for Q learning**



**A.4.2 SARSA**

**Update Rule for SARSA**



**A.4.3 Problem Description**

The agent is placed in a 5x5 grid, where it needs to navigate from a start state to a goal state while avoiding obstacles and negative rewards. The agent receives a reward of +10 for reaching the goal state, a penalty of -1 for every move, and a penalty of -10 if it hits a wall. The agent can move in four directions: up, down, left, and right..

#### **Grid World Setup**:

* The grid has 25 states represented as S={S0,S1,...,S24}
* The robot can move **up**, **down**, **left**, or **right**, but if it tries to move outside the grid boundaries, it remains in the same state.
* The robot receives the following rewards:
  + **+10** for reaching the goal state G
  + **-10** for hitting an obstacle.
  + **-1** for each step to discourage long paths.

The **goal state** is S4, and the **start state** is S20​. Obstacles are placed in several states, and the robot must learn to avoid them.

**Grid Layout**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | **S0** | **S1** | **S2** | **S3** | **S4** | | **S5** | **X** | **S7** | **S8** | **S9** | | **S10** | **X** | **S12** | **X** | **S14** | | **S15** | **S16** | **X** | **S18** | **S19** | | **S20** | **S21** | **S22** | **S23** | **S24** | | S0​: Start state.  S24 ​: Goal state.  **X**: Obstacle states that give a reward of **-10**.  Other states have a step penalty of **-1**. |

**A.5 Task to be completed:**

**Implement Q-Learning Algorithm**:

* Implement Q-learning with a simple tabular approach.
* Train the agent in the environment using Q-learning.
* Tune hyperparameters such as learning rate, discount factor, and exploration strategy (e.g., ε-greedy).
* Store and update the Q-table during training.

**Implement SARSA Algorithm**:

* Implement the SARSA algorithm (State-Action-Reward-State-Action).
* Train the agent using SARSA.
* Use similar hyperparameters for comparison with Q-learning.
* Store and update the Q-table for SARSA during training.

**References**

1. Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction.
2. Dr Saeed Saeedvand <https://youtu.be/CFHYKlPz-Ps?feature=shared>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**PART B**

(PART B: TO BE COMPLETED BY STUDENTS)

**(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Portal/MS Teams assignment link at the end of the practical)**

|  |  |
| --- | --- |
| Roll No. C052 | Name: Drumil Kotecha |
| Program : BTI | Division: B |
| Batch: B2 | Date of Experiment: 28/03/2025 |
| Date of Submission: 28/03/2025 | Grade : |

**B.1 Tasks given in PART A to be completed here**

*(****Students must write the answers of the task(s) given in the PART A /Students must copy the code, output screenshots here based on the task(s) given in section A.5)***

# %%

import numpy as np

import random

import matplotlib.pyplot as plt

# -------------

# ENVIRONMENT SETUP

# -------------

# Grid configuration

n\_states = 25

grid\_rows = 5

grid\_cols = 5

start\_state = 20

goal\_state = 4

# Obstacles (yield -10 reward when entered)

obstacles = [5, 6, 12, 15]

# Define actions: 0=Up, 1=Right, 2=Down, 3=Left

actions = [0, 1, 2, 3]

# %%

def state\_to\_rc(state):

    """Convert state index to (row, col) position."""

    return state // grid\_cols, state % grid\_cols

def rc\_to\_state(row, col):

    """Convert (row, col) to state index."""

    return row \* grid\_cols + col

def step(state, action):

    """

    Take an action in the environment.

    Returns:

        next\_state: the resulting state after taking the action.

        reward: reward received for taking the action.

        done: boolean indicating if the episode ended.

    """

    r, c = state\_to\_rc(state)

    # Determine intended next position

    if action == 0:  # Up

        r\_next, c\_next = r - 1, c

    elif action == 1:  # Right

        r\_next, c\_next = r, c + 1

    elif action == 2:  # Down

        r\_next, c\_next = r + 1, c

    else:  # Left

        r\_next, c\_next = r, c - 1

    # Check for boundary hit

    if r\_next < 0 or r\_next >= grid\_rows or c\_next < 0 or c\_next >= grid\_cols:

        return state, -10, False  # Hit wall, stay in same state

    next\_state = rc\_to\_state(r\_next, c\_next)

    # Check for obstacles

    if next\_state in obstacles:

        return next\_state, -10, False

    # Check for goal state

    if next\_state == goal\_state:

        return next\_state, 10, True

    # Valid move with a step cost

    return next\_state, -1, False

def reset():

    """Reset the environment to the start state."""

    return start\_state

# %%

def q\_learning(num\_episodes=1000, alpha=0.1, gamma=0.99, epsilon=0.1):

    """

    Q-learning with tabular Q-table.

    Returns:

        Q: Learned Q-table.

        rewards\_list: List of cumulative rewards per episode.

    """

    Q = np.zeros((n\_states, len(actions)))

    rewards\_list = []

    for episode in range(num\_episodes):

        s = reset()

        done = False

        episode\_reward = 0

        while not done:

            # ε-greedy action selection

            if random.random() < epsilon:

                a = random.choice(actions)

            else:

                a = np.argmax(Q[s, :])

            s\_next, r, done = step(s, a)

            episode\_reward += r

            # Q-learning update (off-policy)

            Q[s, a] += alpha \* (r + gamma \* np.max(Q[s\_next, :]) - Q[s, a])

            s = s\_next

        rewards\_list.append(episode\_reward)

    return Q, rewards\_list

# %%

def sarsa(num\_episodes=1000, alpha=0.1, gamma=0.99, epsilon=0.1):

    """

    SARSA algorithm with a tabular Q-table.

    Returns:

        Q: Learned Q-table.

        rewards\_list: List of cumulative rewards per episode.

    """

    Q = np.zeros((n\_states, len(actions)))

    rewards\_list = []

    for episode in range(num\_episodes):

        s = reset()

        # Choose initial action using ε-greedy policy

        if random.random() < epsilon:

            a = random.choice(actions)

        else:

            a = np.argmax(Q[s, :])

        done = False

        episode\_reward = 0

        while not done:

            s\_next, r, done = step(s, a)

            episode\_reward += r

            if not done:

                if random.random() < epsilon:

                    a\_next = random.choice(actions)

                else:

                    a\_next = np.argmax(Q[s\_next, :])

                # SARSA update (on-policy)

                Q[s, a] += alpha \* (r + gamma \* Q[s\_next, a\_next] - Q[s, a])

                s, a = s\_next, a\_next

            else:

                # Terminal update

                Q[s, a] += alpha \* (r - Q[s, a])

        rewards\_list.append(episode\_reward)

    return Q, rewards\_list

# %%

episodes = 2000

alpha = 0.1

gamma = 0.99

epsilon = 0.1

# Train the agent using Q-learning

Q\_qlearning, rewards\_qlearning = q\_learning(num\_episodes=episodes, alpha=alpha, gamma=gamma, epsilon=epsilon)

# Train the agent using SARSA

Q\_sarsa, rewards\_sarsa = sarsa(num\_episodes=episodes, alpha=alpha, gamma=gamma, epsilon=epsilon)

# %%

plt.figure(figsize=(10, 6))

plt.plot(range(episodes), rewards\_qlearning, label="Q-learning", alpha=0.7)

plt.plot(range(episodes), rewards\_sarsa, label="SARSA", alpha=0.7)

plt.xlabel("Episode")

plt.ylabel("Cumulative Reward")

plt.title("Learning Curve Comparison: Q-learning vs SARSA")

plt.legend()

plt.grid(True)

plt.show()

# %%

def print\_q\_table(Q):

    for i in range(n\_states):

        print(f"State {i}: {Q[i, :]}")

# %%

print\_q\_table(Q\_qlearning)

# %%

print\_q\_table(Q\_qlearning)

# %%

# %%

**B.2 Observations and Learning:**

*(****Students must write the observations and learning based on their understanding built about the subject matter and inferences drawn)***

* Both Q-learning and SARSA successfully learn policies to reach the goal while avoiding obstacles.
* SARSA’s on-policy updates yield smoother, more cautious learning, adapting steadily to penalties.
* Q-learning’s off-policy updates quickly explore higher reward paths, though it may risk overshooting near obstacles.
* Overall, both methods improve cumulative rewards over episodes, each with its own trade-offs in convergence and stability.

**B.3 Conclusion:**

*(****Students must write the conclusive statements as per the actual attainment of individual outcomes listed above and learning/observation noted in section B.2)***

## Q-learning and SARSA both, effectively learned navigation policies in the 5x5 grid environment by balancing exploration and exploitation to avoid obstacles and reach the goal. Q-learning's off-policy updates allowed for rapid convergence, albeit with occasional overshooting near obstacles, while SARSA's on-policy strategy provided more cautious and stable learning under risky conditions.