Experiment No. 8

Title - Implement Reinforcement Learning Algorithm

Theory -

Reinforcement Learning (RL) is a type of machine learning paradigm where an agent learns to make decisions by interacting with an environment. The agent receives feedback in the form of rewards or punishments, and its objective is to learn a strategy or policy that maximizes the cumulative reward over time. RL is inspired by the way animals and humans learn from trial and error to achieve goals.

1. **Agent:**

- The learner or decision-maker that interacts with the environment.
- It makes decisions based on its current understanding and the feedback it receives.

2. **Environment:**

- The external system with which the agent interacts.
- It provides feedback to the agent based on the actions taken, and its state may change as a result of the agent's actions.

3. **State (S):**

- A representation of the current situation or configuration of the environment.
- The state is crucial as the agent's decisions are based on it, and it influences the subsequent state and reward.

4. **Action (A):**

- The set of possible moves or decisions that the agent can make in a given state.
- The agent selects actions based on its policy, which is its strategy for making decisions.

5. **Policy (π):**

- A mapping from states to actions, representing the agent's strategy.
- The policy can be deterministic or stochastic, depending on whether it prescribes a single action or a distribution over actions for a given state.

6. **Reward (R):**

- A numerical signal that the environment provides to the agent after it takes an action in a particular state.
- The goal of the agent is to maximize the cumulative reward over time.

7. **Trajectory or Episode:**

- A sequence of states, actions, and rewards that occurs during the interaction between the agent and the environment.
- An episode typically starts from an initial state, consists of a series of actions, and ends when a termination condition is met.

8. **Value Function (V or Q):**

- The value of a state (V) or a state-action pair (Q) is an estimate of the expected cumulative future reward that the agent can obtain from that state (or state-action pair) onwards.
- The value function guides the agent in evaluating the desirability of different states or actions.

9. **Exploration vs. Exploitation:**

- Exploration involves trying new actions to discover their effects and improve the agent's understanding of the environment.
- Exploitation involves choosing actions that the agent believes will lead to high immediate rewards based on its current knowledge.

10. **Learning Algorithm:**

- The algorithm that the agent uses to update its policy or value function based on the observed rewards and experiences.

- Common RL algorithms include Q-learning, SARSA, and deep reinforcement learning methods like Deep Q Networks (DQN) and Policy Gradient methods.

Q - Learning

Q-learning is a fundamental reinforcement learning algorithm that aims to find an optimal action-selection policy for a given finite Markov decision process (MDP). It was introduced by Chris Watkins in 1989. Q-learning belongs to the class of model-free reinforcement learning algorithms, meaning that it doesn't require knowledge about the underlying dynamics of the environment.

1. Markov Decision Process (MDP):

- Q-learning is applicable to problems modelled as MDPs.
- An MDP is defined by a tuple (S, A, P, R), where:
  - S is the set of states.
  - A is the set of actions.
    - P is the state transition probability function, indicating the probability of transitioning from one state to another given an action.
    - R is the reward function, specifying the immediate reward received after taking an action in a particular state.

2. Q-Table:

Q-learning maintains a Q-table, which is a matrix where each entry Q(s, a) represents the expected cumulative future reward of taking action a in state s and following the optimal policy thereafter. Initially, the Q-table is often initialized arbitrarily.

3. Initialization:

Initialize the Q-table with small random values.

4. Exploration vs. Exploitation:

During each time step, the agent decides whether to explore new actions or exploit the current knowledge. Exploration is typically achieved using strategies such as epsilon-greedy, where the agent chooses a random action with probability epsilon, and otherwise, it selects the action with the highest Q-value.

5. Update Rule:

The Q-value for a state-action pair is updated using the Q-learning update rule:

$$\mathbf{Q}(\mathbf{s}, \mathbf{a}) \leftarrow (\mathbf{1} - \alpha) \cdot \mathbf{Q}(\mathbf{s}, \mathbf{a}) + \alpha \cdot [\mathbf{r} + \gamma \cdot \max_{\mathbf{a}'} \mathbf{Q}(\mathbf{s}', \mathbf{a}')]$$

where:

- $\mathbf{Q}(\mathbf{s}, \mathbf{a})$is the current Q-value for state $\mathbf{s}$ and action $\mathbf{a}$.
- $\mathbf{r}$ is the immediate reward obtained after taking action $\mathbf{a}$ in state $\mathbf{s}$.
- $\mathbf{s}'$ is the next state.
- α is the learning rate ($0 < \alpha \leq 1$), determining the influence of new information.
- γ is the discount factor ($0 \leq \gamma \leq 1$), accounting for the importance of future rewards.

6. Termination:

The agent continues interacting with the environment and updating the Q-table until a termination condition is met (e.g., a predefined number of episodes or a convergence criterion).

7. Policy Extraction:

The learned Q-values can be used to derive the optimal policy. The optimal policy at each state is the action with the highest Q-value.

Implementation :

1. Defining functions

```
import numpy as np
from tqdm import tqdm

# Define your QLearning class
class QLearning:
    def __init__(self, state_space, action_space, learning_rate, discount_factor):
        self.state_space = state_space
        self.action_space = action_space
        self.learning_rate = learning_rate
        self.discount_factor = discount_factor
        self.q_table = np.zeros((state_space, action_space))

    def get_action(self, state, epsilon):
        if np.random.rand() < epsilon:
            return np.random.choice(self.action_space)  # Random action (exploration)
        else:
            return np.argmax(self.q_table[state])  # Exploitation using learned Q-values

    def update_q_table(self, state, action, reward, next_state):
        old_value = self.q_table[state, action]
        next_max = np.max(self.q_table[next_state])
                new_value = (1 - self.learning_rate) * old_value + self.learning_rate * (reward +
self.discount_factor * next_max)
        self.q_table[state, action] = new_value
```

2. Implementing algorithm

```
# Define your specific state space, action space, learning rate, and discount factor
state_space = 9  # number of states
action_space = 4  # number of actions
learning_rate = 0.2
discount_factor = 0.99

# Create an instance of the QLearning class
ql = QLearning(state_space, action_space, learning_rate, discount_factor)

# Simulate a number of episodes using tqdm for progress visualization
num_episodes = 10000
```

```
epsilon = 0.5  # Initial exploration rate
min_epsilon = 0.01  # Minimum exploration rate
decay_rate = 0.001  # Exploration rate decay
rewards = np.random.randn(state_space)  # Define random rewards for each state

for episode in tqdm(range(num_episodes)):
    state = np.random.randint(0, state_space)
    done = False

    while not done:
        action = ql.get_action(state, epsilon)

        # Simulate a transition - here, just a random transition for demonstration
        next_state = np.random.randint(0, state_space)
        reward = rewards[next_state]  # Use the defined rewards

        ql.update_q_table(state, action, reward, next_state)
        state = next_state

        if state == state_space - 1:
            done = True

    # Decay exploration rate epsilon
    epsilon = min_epsilon + (1 - min_epsilon) * np.exp(-decay_rate * episode)
print("Final Q-table:")
print(ql.q_table)
```

Output (Actual results may vary)

```
100%|████████████████| 10000/10000 [00:00<00:00, 18544.14it/s]
Final Q-table:
[[26.59612402 26.62478417 26.598401   27.02019464]
 [26.54215703 26.5325388  26.88811188 26.53009904]
 [26.53513699 26.4886326  26.52049472 27.32266776]
 [26.6036865  26.56117206 26.80018192 26.66500551]
 [27.08214962 26.55504005 26.55245073 26.4486053 ]
 [26.53188503 26.71258365 26.54135352 26.82801822]
 [26.59497875 26.54551248 26.92728842 26.57705163]
 [26.5434918  26.92533249 26.47621987 26.50384238]
 [24.22957191 25.01815374 26.83266818 21.58941471]]
```