Vidyavardhini's College of Engineering and Technology
Department of Artificial Intelligence & Data Science

Experiment No.1
Implementing a simple grid-world environment and training an agent using basic Q-learning
Date of Performance:
Date of Submission:

EXPERIMENT 1

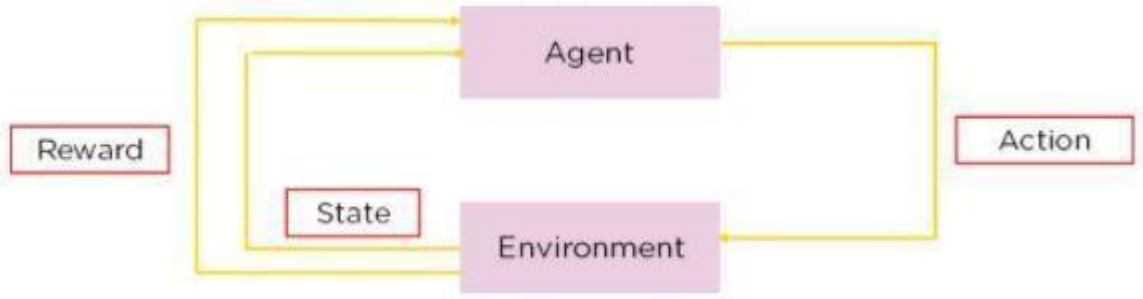
Aim: Implementing a simple grid-world environment and training an agent using basic Q-learning

Objective: The objective of Q-learning is to understand how Q-values are updated based on state-action pairs and how the agent learns to navigate the environment to maximize cumulative rewards.

Theory:

Q-Learning

Q-Learning is a Reinforcement learning policy that will find the next best action, given a current state. It chooses this action at random and aims to maximize the reward.



Q-learning is a model-free, off-policy reinforcement learning that will find the best course of action, given the current state of the agent. Depending on where the agent is in the environment, it will decide the next action to be taken.

The objective of the model is to find the best course of action given its current state. To do this, it may come up with rules of its own or it may operate outside the policy given to it to follow. This means that there is no actual need for a policy, hence we call it off-policy.

Model-free means that the agent uses predictions of the environment's expected response to move forward. It does not use the reward system to learn, but rather, trial and error.

An example of Q-learning is an Advertisement recommendation system. In a normal ad recommendation system, the ads you get are based on your previous purchases or websites you may have visited. If you've bought a TV, you will get recommended TVs of different brands.

Using Q-learning, we can optimize the ad recommendation system to recommend products that are frequently bought together. The reward will be if the user clicks on the suggested product.

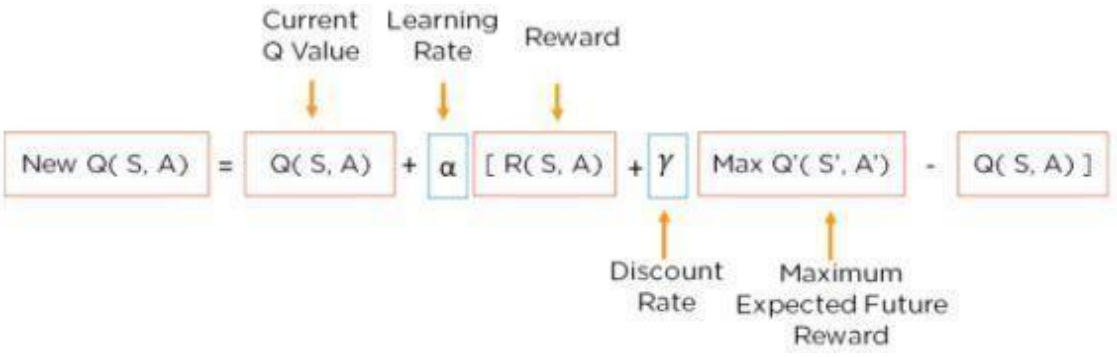
Important Terms in Q-Learning

1. States: The State, S, represents the current position of an agent in an environment.
2. Action: The Action, A, is the step taken by the agent when it is in a particular state.
3. Rewards: For every action, the agent will get a positive or negative reward.
4. Episodes: When an agent ends up in a terminating state and can't take a new action.
5. Q-Values: Used to determine how good an Action, A, taken at a particular state, S, is. Q (A, S).
6. Temporal Difference: A formula used to find the Q-Value by using the value of current state and action and previous state and action.

What Is The Bellman Equation?

The Bellman Equation is used to determine the value of a particular state and deduce how good it is to be in/take that state. The optimal state will give us the highest optimal value.

The equation is given below. It uses the current state, and the reward associated with that state, along with the maximum expected reward and a discount rate, which determines its importance to the current state, to find the next state of our agent. The learning rate determines how fast or slow, the model will be learning.



How to Make a Q-Table?

While running our algorithm, we will come across various solutions and the agent will take multiple paths. How do we find out the best among them? This is done by tabulating our findings in a table called a Q-Table.

A Q-Table helps us to find the best action for each state in the environment. We use the Bellman Equation at each state to get the expected future state and reward and save it in a table to compare with other states.

Lets us create a q-table for an agent that has to learn to run, fetch and sit on command. The steps taken to construct a q-table are :

Step 1: Create an initial Q-Table with all values initialized to 0

When we initially start, the values of all states and rewards will be 0. Consider the Q-Table shown below which shows a dog simulator learning to perform actions :

Action	Fetching	Sitting	Running
Start	0	0	0
Idle	0	0	0
Wrong Action	0	0	0
Correct Action	0	0	0
End	0	0	0

Step 2: Choose an action and perform it. Update values in the table

This is the starting point. We have performed no other action as of yet. Let us say that we want the agent to sit initially, which it does. The table will change to:

Action	Fetching	Sitting	Running
Start	0	1	0
Idle	0	0	0
Wrong Action	0	0	0
Correct Action	0	0	0
End	0	0	0

Step 3: Get the value of the reward and calculate the value Q-Value using Bellman Equation

For the action performed, we need to calculate the value of the actual reward and the Q(S, A) value

Action	Fetching	Sitting	Running
Start	0	1	0
Idle	0	0	0
Wrong Action	0	0	0
Correct Action	0	34	0
End	0	0	0

Step 4: Continue the same until the table is filled or an episode ends

The agent continues taking actions and for each action, the reward and Q-value are calculated and it updates the table.

Code:

```
import random

class QLearningAgent:
    def __init__(self, actions, alpha=0.1, gamma=0.9, epsilon=0.1):
        self.Q = {}
        self.actions = actions
        self.alpha = alpha  # learning rate
        self.gamma = gamma  # discount factor
        self.epsilon = epsilon  # exploration rate
    def get_Q_value(self, state, action):
        return self.Q.get((state, action), 0.0)
```

```
    def choose_action(self, state):
        if random.random() < self.epsilon:
            return random.choice(self.actions)
        else:
            q_values = [self.get_Q_value(state, action) for action in
self.actions]
            max_q_value = max(q_values)
            best_actions = [action for action, q_value in zip(self.actions,
q_values) if q_value == max_q_value]
            return random.choice(best_actions)
    def update_Q_value(self, state, action, reward, next_state):
        q_value = self.get_Q_value(state, action)
        best_next_action = max([self.get_Q_value(next_state, next_action) for
next_action in self.actions])
        new_q_value = q_value + self.alpha * (reward + self.gamma *
best_next_action - q_value)
        self.Q[(state, action)] = new_q_value
def train_agent(env, agent, episodes, max_steps):
    for _ in range(episodes):
        state = env.start
        for _ in range(max_steps):
            action = agent.choose_action(state)
            reward = env.move(action)
            next_state = env.get_state()
            agent.update_Q_value(state, action, reward, next_state)
            if next_state == env.goal:
                break
            state = next_state
        env.reset()
# Define the grid-world environment
width = 5
height = 5
start = (0, 0)
goal = (4, 4)
obstacles = [(1, 1), (2, 2), (3, 3)]
actions = ["up", "down", "left", "right"]
# Create the environment and agent
env = GridWorld(width, height, start, goal, obstacles)
agent = QLearningAgent(actions)
# Train the agent
train_agent(env, agent, episodes=1000, max_steps=100)
# Test the trained agent
```

```
state = start
while state != goal:
    action = agent.choose_action(state)
    print("Current state:", state, "Action:", action)
    env.move(action)
    state = env.get_state()
print("Goal reached!")
```

Output:

```
Current state: (0, 0) Action: down
Current state: (0, 1) Action: down
Current state: (0, 2) Action: right
Current state: (1, 2) Action: down
Current state: (1, 3) Action: down
Current state: (1, 4) Action: right
Current state: (2, 4) Action: right
Current state: (3, 4) Action: right
Goal reached!
```

Conclusion:

1. Explain Simple grid world Environment
   In the realm of reinforcement learning, the simple grid world environment serves as a foundational playground for understanding and experimenting with various algorithms. This environment typically consists of a grid of cells, each representing a state, where an agent can move around.

2. Explain Bellman Equation
   The Bellman equation is a fundamental principle in reinforcement learning that expresses the relationship between the value of a state and the values of its neighboring states. It mathematically formalizes the notion of optimality in terms of maximizing cumulative rewards over time.