29.11.2024

BITS F464: Machine Learning (1st Sem 2024-25)

Introduction to Reinforcement Learning(RL)

Chittaranjan Hota, Sr. Professor
Dept. of Computer Sc. and Information Systems
hota@hyderabad.bits-pilani.ac.in

What is Reinforcement Learning?

- Agent tries to maximize the cumulative reward from the environment by performing a set of actions.

Image source: https://highlandcanine.com/

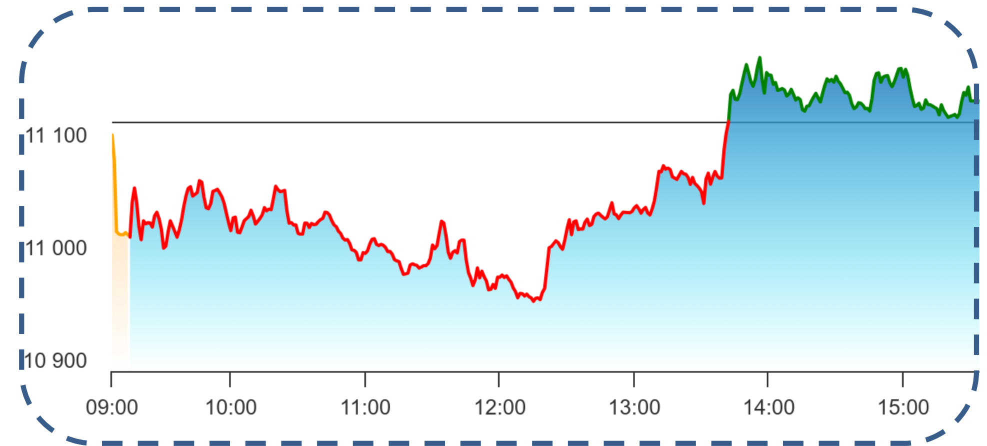
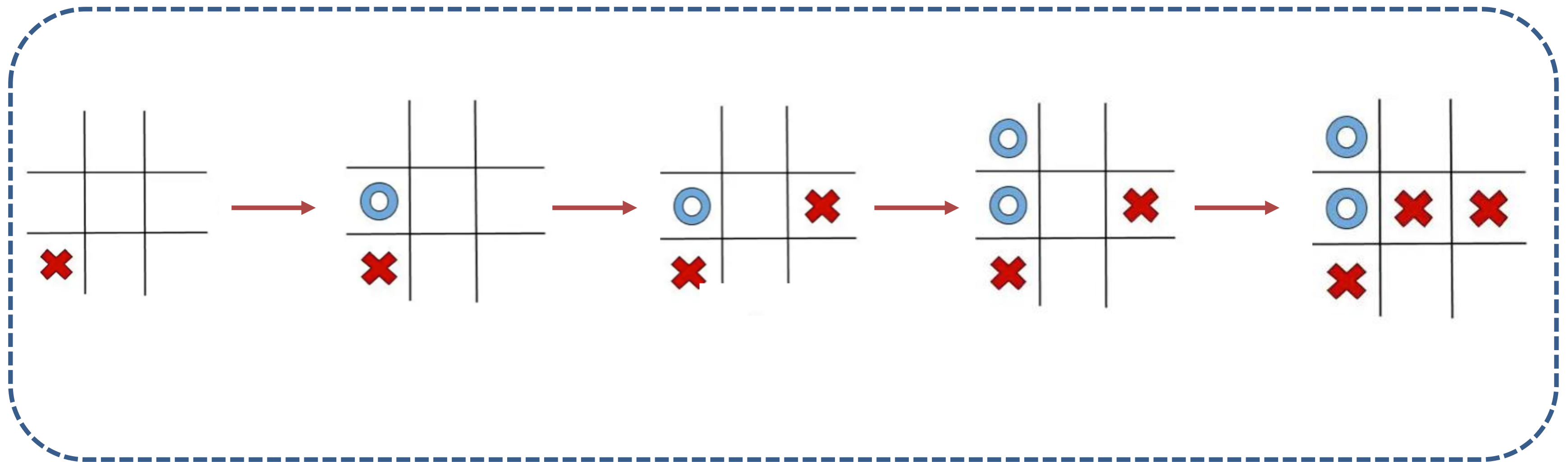


Image source: UltraTech Cement Stock, 27th Nov 2024 from www.nseindia.com

Applications: Gaming, Robotics, Autonomous vehicles, Personalized treatment etc.

Formal Modelling: Markov Decision Process

Markov: The future state can be determined only from the present state that encapsulates all the necessary information from the past.



What should the player 'O' do here to avoid a loss?

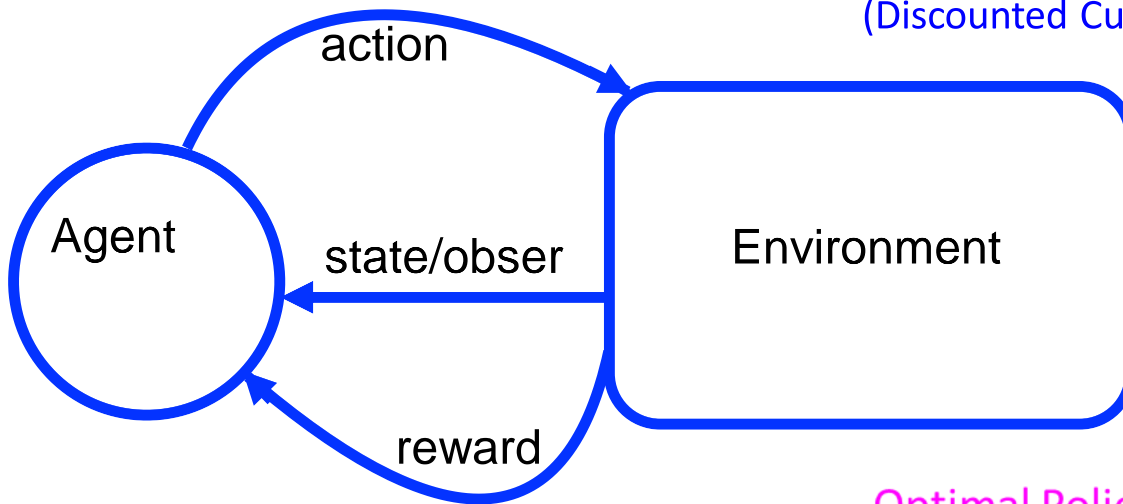
MDP Continued…

An MDP is defined as a tuple (S,A,P,R,γ):

$$V^{\pi}(s_t) = r_t + \gamma\, r_{t+1} + \gamma^2\, r_{t+2} + \cdots$$

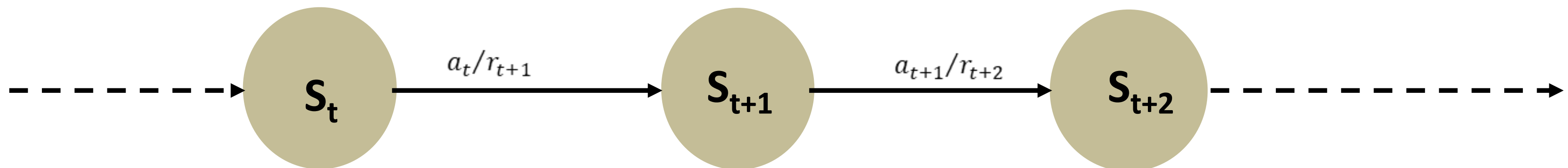
(Discounted Cumulative Reward)

$$\equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$



γ: Discount factor controlling future rewards.

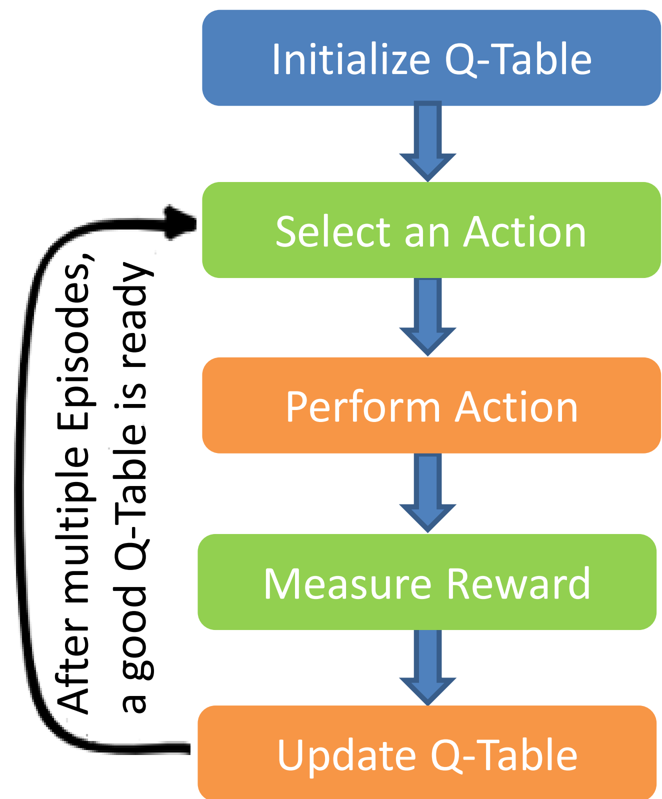
Optimal Policy $(\pi^*) = \arg\max_{\pi} V^{\pi}(s), (\forall s)$



Learning Goal: Learn a Policy, $\pi(s_t) = a_t$

Q-Learning Algorithm

- Q-learning is a model-free reinforcement learning (RL) algorithm used to learn the optimal policy for a Markov Decision Process (MDP)



Step 1: Initialize $Q(s, a)$ for all states and actions to 0.

Step 2: For each Episode:

Start at initial state s_0.

While not at s_{goal}:

a. Choose an action 'a' using an $\epsilon - greedy$ policy:

- with prob ϵ, select a random action (exploration).
- else, select $\arg\max_a Q(s, a)$ (exploitation).

b. Execute *a*, Observe reward *R*, and next state *S'*.

c. Update *Q(s, a)* using the formula:

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

d. Set $s = s'$.

An Example of Q-Learning

- Initializing the environment: States: $\{s_0, s_1, s_2\}$, Actions: $\{a_0, a_1\}$, Rewards: $R(s_0, a_0) = -1$, $R(s_0, a_1) = +2$, $R(s_1, a_0) = +3$, $R(s_1, a_1) = +1$, $R(s_2, \text{any action}) = 0$ (terminal state).
- Transitions: $T(s_0, a_0) \rightarrow s_1$, $T(s_0,a_1) \rightarrow s_2$ (goal), $T(s_1,a_0) \rightarrow s_2$, $T(s_1, a_1) \rightarrow s_0$
- Parameters: $\alpha = 0.5$, $\gamma = 0.9$, Initial Q-values ($Q(s, a) = 0$ for all s, a).
- Episode 1:
  - current state: s_0, action chosen: a_0 (randomly using exploration), reward: $R(s_0, a_0) = -1$, next state: s_1.
  - Update $Q(s_0,a_0)$ using Bellman's equation:

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

  - $Q(s_0, a_0) \leftarrow 0 + 0.5 [-1 + 0.9 * \max Q(s_1, a') - 0]$
  - $Q(s_0, a_0) \leftarrow 0.5 * [-1 + 0] = -0.5$ (Since, $Q(s_1, a') = 0$ initially (no knowledge of s_1).

Ex. Continued…

Updated Q-values after 3 Episodes

State	Action(a_0)	Action(a_1)
s_0	-0.5	1.0
s_1	1.5	0.0
s_2	0.0	0.0

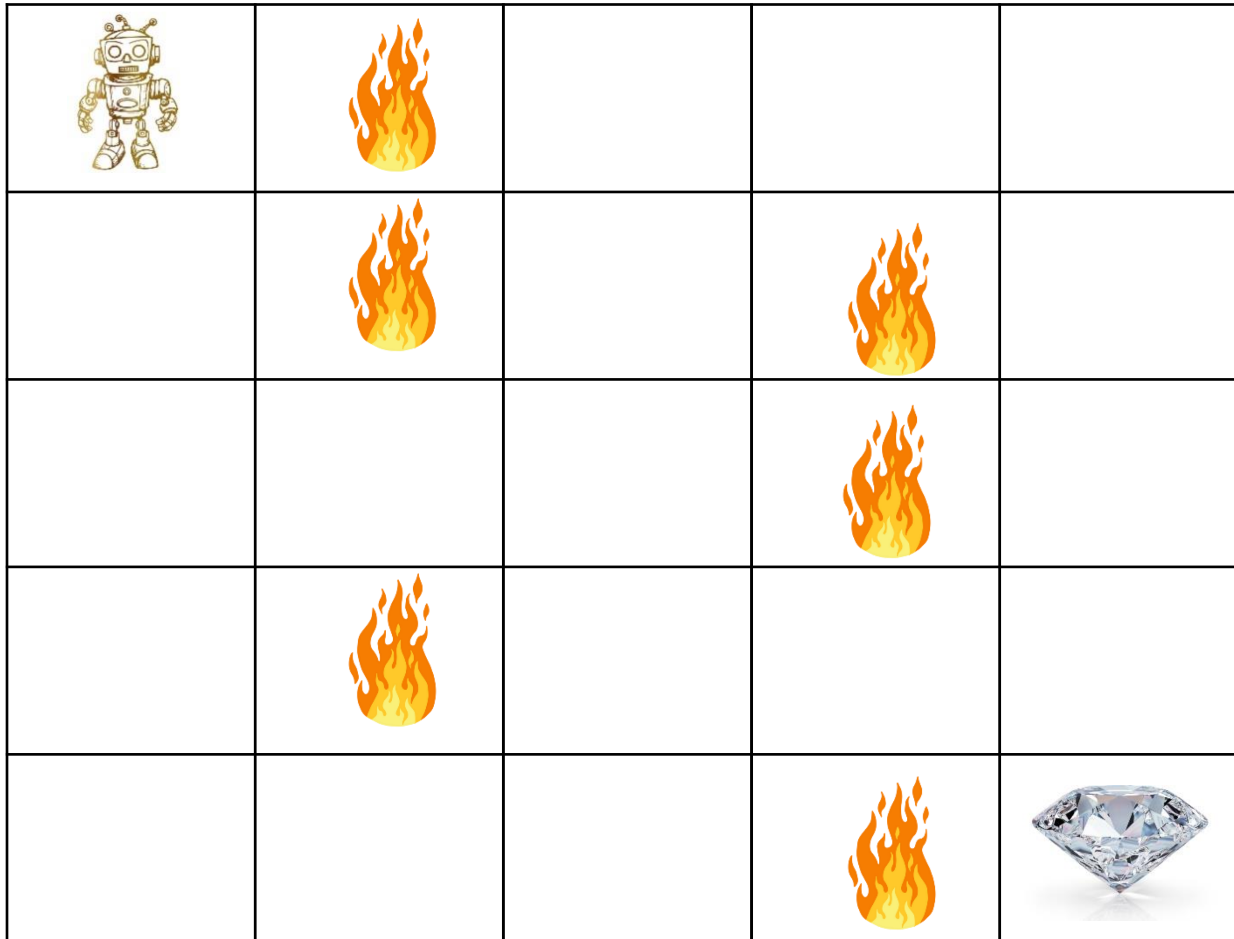
- Episode 2: From s_1
  - current state: s_1, action chosen: a_0, reward: $R(s_1, a_0) = +3$, next state: s_2.
  - Update $Q(s_0,a_0)$ using Bellman's equation:

  $Q(s_1,a_0) \leftarrow Q(s_1,a_0) + \alpha[R + \gamma \max_{a'} Q(s_2, a') - Q(s_1, a_0)]$

  - $Q(s_1, a_0) \leftarrow 0 + 0.5 [3 + 0.9 * 0 - 0] = 1.5$
- Episode 3: Back to s_0(different action)
  - current state: s_0, action chosen: a_1, reward: $R(s_0, a_1) = +2$, next state: s_2.
  - Update $Q(s_0,a_1)$ using Bellman's equation:
  - $Q(s_0,a_1) \leftarrow Q(s_0,a_1) + \alpha[R + \gamma \max_{a'} Q(s_2, a') - Q(s_0, a_1)]$
  - $Q(s_0, a_1) \leftarrow 0 + 0.5 [2 + 0.9 * 0 - 0] = 1.0$

- Alternatively, you may use an ANN to learn Q-values: Deep Q-Learning (DQN)

Optimal Solution using Q-Learning: Maze



```
import numpy as np
import matplotlib.pyplot as plt

# Maze parameters
maze = [
   [0, 1, 0, 0, 0],
   [0, 1, 0, 1, 0],
   [0, 0, 0, 1, 0],
   [0, 1, 0, 0, 0],
   [0, 0, 0, 1, 2]
#'2'is the diamond (goal state)
]

maze = np.array(maze)
```

Python Code

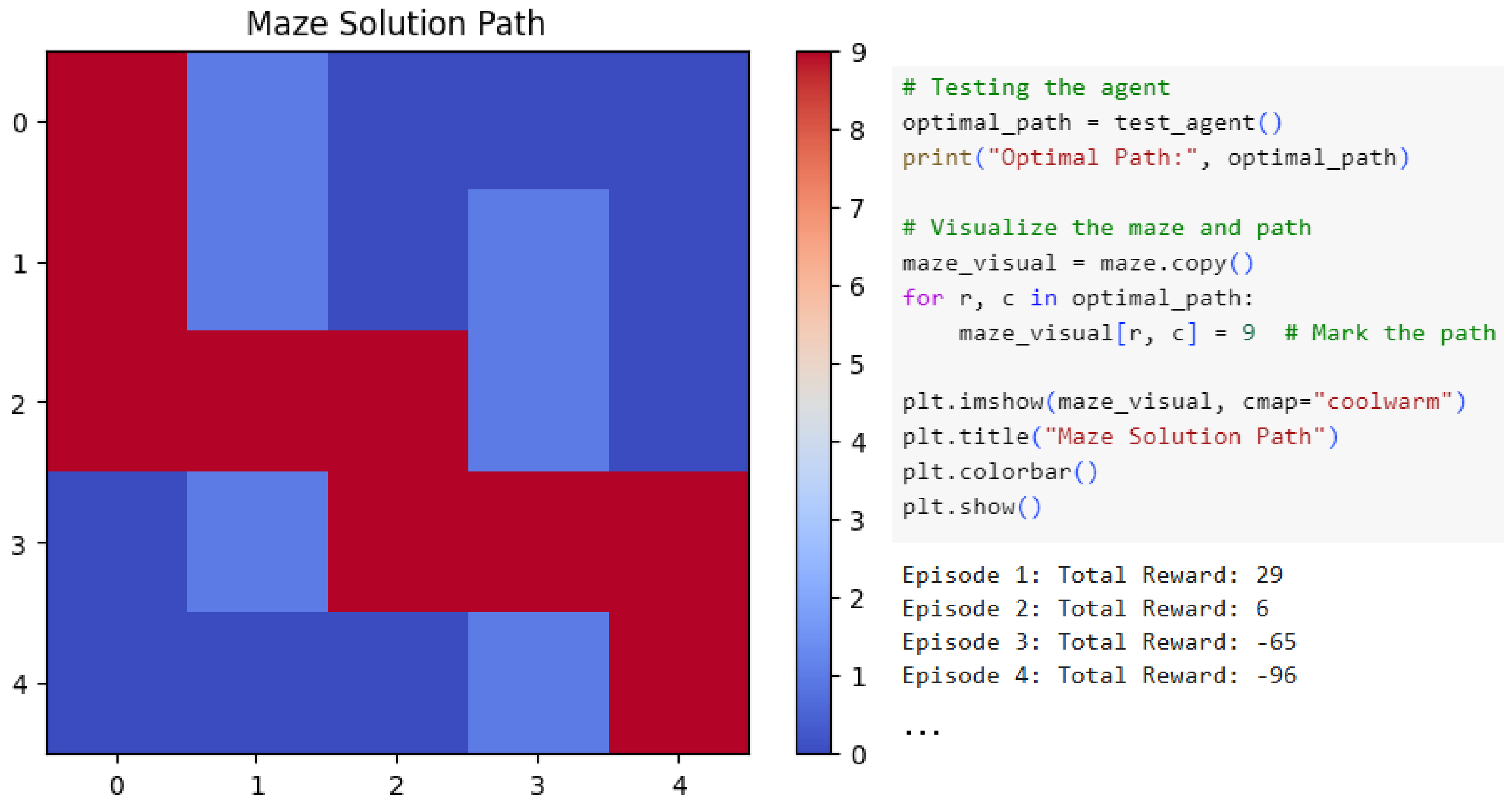
```python
# Actions: up, down, left, right
actions = ["up", "down", "left", "right"]
action_dict = {"up": (-1, 0), "down": (1, 0), "left": (0, -1),
                                            "right": (0, 1)}

# Q-Learning parameters
q_table = np.zeros((maze.size, len(actions)))
epsilon = 0.9  # Exploration rate
alpha = 0.1  # Learning rate
gamma = 0.9  # Discount factor
```

```python
# Convert (row, col) to state index
def state_index(state):
    return state[0] * maze.shape[1] + state[1]

# Check if a move is valid
def valid_move(state, action):
    rows, cols = maze.shape
    next_state = (state[0] + action[0], state[1] + action[1])
    if 0 <= next_state[0] < rows and 0 <= next_state[1] < cols:
        return maze[next_state] != 1  # Check for walls
    return False

# Get reward for a state
def get_reward(state):
    if maze[state] == 2:
        return 100  # Reaching the diamond
    return -1  # Default penalty for each step
```

```python
# Choose an action using epsilon-greedy policy
def choose_action(state):
    if np.random.rand() < epsilon:
        return np.random.choice(len(actions))  # Explore
    return np.argmax(q_table[state_index(state)])  # Exploit

# Train Q-Learning agent
def train_agent(episodes):
    for episode in range(episodes):
        state = (0, 0)  # Start at top-left corner
        total_reward = 0

        while True:
            action_idx = choose_action(state)
            action = action_dict[actions[action_idx]]

            if valid_move(state, action):
                next_state = (state[0] + action[0], state[1] +
                                                            action[1])
            else:
                next_state = state  # Stay in the same state if move
                                                            is invalid

            reward = get_reward(next_state)
            total_reward += reward
```

```python
# Update Q-table
current_q = q_table[state_index(state), action_idx]
max_next_q = np.max(q_table[state_index(next_state)])
q_table[state_index(state), action_idx] = current_q +
        alpha * (reward + gamma * max_next_q - current_q)

state = next_state
if maze[state] == 2:  # Goal reached
    break

print(f"Episode {episode + 1}:Total Reward:{total_reward}")
```

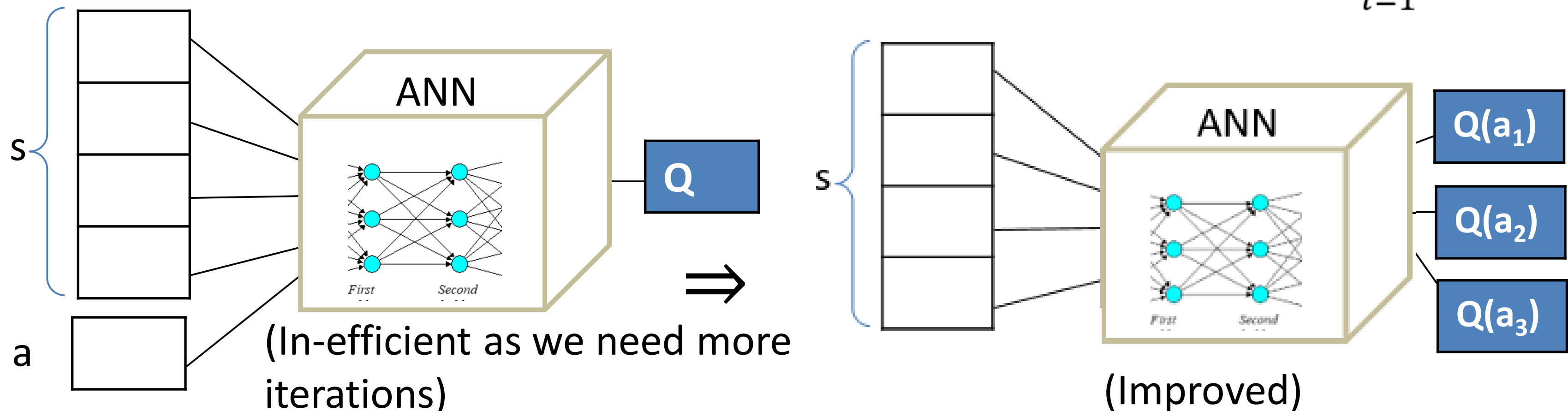
Continued...




```
Optimal Path: [(0, 0), (1, 0), (2, 0), (2, 1), (2, 2), (3, 2), (3, 3), (3, 4), (4, 4)]
```

Deep Q-Learning (DQN) for RL

- When the number of states and actions become very large, how do you scale?
- Solution: Combine Q-Learning and Deep Learning → Deep Q-Networks (DQN)
- Goal: Approximate a function: $Q(s,a;\ \theta)$, where θ represents the trainable weights of the network
- $Q(s,a) = r(s,a) + \gamma \max Q(s',a)$ Bellman's equation
- $\text{Cost} = \{Q(s,a;\ \theta) - [r(s,a)+\gamma \max Q(s',a;\ \theta)]\}^2$ $\Leftrightarrow$ $\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{Y}_i)^2$



Thank You!

Good luck for Comprehensive Exams!