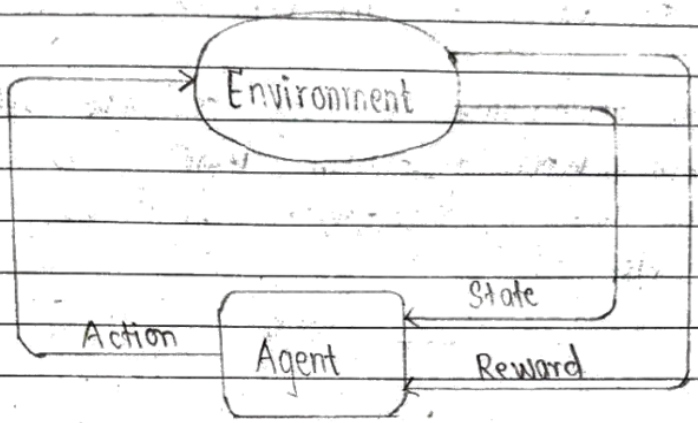
10. Reinforcement Learning

1. Introduction



- Reinforcement learning is a feedback-base machine learning technique where an agent learns to which actions to perform by looking at the environment and the results of actions.

- For each correct action, the agent gets positive feedback, and for each incorrect action, the agent gets negative feedback or penalty.

- The agent interacts with the environment and identifies the possible actions he can perform.

- The primary goal of an agent in reinforcement learning is to perform actions by looking at the environment and get the maximum positive rewards.

- In Reinforcement learning, the agent learns automatically using feedbacks without any labeled data, unlike supervised learning.

- Since there is no labeled data, so the agent is bound to learn by its experience only.
- Reinforcement learning is used to solve specific type of problem where decision making is sequential, and the goal is long-term, such as game playing, robotics, etc.

2. Types of Reinforcement Learning

There are two types of reinforcement learning:

1. Positive reinforcement learning
2. Negative reinforcement learning

1. Positive Reinforcement Learning

- Positive reinforcement involves adding a rewarding stimulus after a desired behaviour is performed, which increases the likelihood of behaviour being repeated.
- It is recurrence of behaviour due to positive rewards.
- Rewards increase strength and frequency of a specific behaviour.

2. Negative Reinforcement Learning:

- Negative reinforcement involves removing an aversive stimulus after a desired behaviour is performed, which also increases the likelihood of that behaviour being repeated.
- In negative reinforcement learning, negative rewards are used as a deterrent to weaken the behaviour and to avoid it.
- Reward decreases strength and the frequency of a specific behaviour.

8. Use Cases of Reinforcement Learning

Reinforcement learning (RL) is a powerful technique used in various real-world applications where decision making is crucial.

Here are some areas where RL is commonly applied:

1. Robotics

RL is extensively used in robotics for tasks such as:

- **Industrial Automation:** Robots learn to perform complex tasks like assembly, welding and painting by optimizing their actions to maximize efficiency and

accuracy.

- **Autonomous Navigation:** Robots learn to navigate through environments, avoiding obstacles and reaching targets efficiently.

2. Autonomous Vehicles:

Self driving cars use RL to make real-time decisions, such as:

- **Path Planing:** Determing the optimal route to a destination while avoiding obstacles and following traffic rules.
- **Control Systems:** Managing acceleration, braking, and steering to ensure safe and efficient driving.

3. Games Playing

RL has successfully applied to various games, including

- **Video Games:** Agents learn to play games like chess, go and atari games at superhuman levels by optimizing their strategies through trial and error.
- **Board Games:** RL algorithms have been used to develop AI that can compete and defeat human champions.

4. Health Care

In health care RL is used for:

- **Personalized Treatment Plans:** Developing customized treatment strategies for patients based on their unique medical histories and responses to treatments.
- **Robotics Surgery:** Assisting surgeons by optimizing the movements of surgical robots to improve precision and outcomes.

5. Finance

RL is applied in finance for:

- **Algorithmic Trading:** Developing trading strategies that maximize returns by learning from market data and trends.
- **Portfolio Management:** Optimizing the allocation of assets in a portfolio to balance risk and return.

6. Natural Language Processing (NLP)

In NLP, RL is used for:

- **Dialogue Systems:** Training chatbots and virtual assistants to provide more accurate and contextually relevent responses.

- **Machine Translation:** Improving the accuracy of translating text from one language to another by learning from feedback.

7. Education:

RL is used to create adaptive learning systems such as:

- **Personalized Learning:** Tailor educational content and pacing to individual students' needs and learning styles.
- **Intelligent Tutoring Systems:** Provide customized feedback and support to students based on their performance.

4. Q-Table

- A Q-table is a matrix used in reinforcement learning to store the expected future rewards for each state-action pair.
- Each row represents a state and each column represents an action.
- The values in the table indicates the quality (Q-value) of taking a particular action in particular state.

Structure of a Q-Table.

State	Action: Up	Action: Down	Action: Left	Action: Right
0	0.5	0.2	0.1	0.4
1	0.3	0.6	0.2	0.5
2	0.4	0.3	0.7	0.2
3	0.6	0.1	0.3	0.8

- **Rows:** Represents different states of the environment.
- **Columns:** Represents the possible action the agent can take.
- **Values:** Represent the expected future rewards (Q-values) for taking a specific action in a specific state.

Updation in Q-table

- The Q-values are updated using Q-learning algorithm.
- Q-learning is a model-free, value-based reinforcement learning algorithm used to find the optimal action-selection policy for any given finite Markov decision process (MDP).
- Q-learning helps an agent learn to maximize the total reward over time through repeated interactions

with the environment, even when the model of that environment is not known.

↳ Mathematically, the Q-value is updated using Bellman equation also called Q-learning formula:

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

where,

- $Q(s,a)$ is the current Q-value for state (s) and action (a).
- α is the learning rate, which determines how much new information overrides the old information.
- r is the reward received after taking action (a).
- γ is the discount factor, which determines the importance of future rewards.
- $\max_{a'} Q(s',a')$ is the maximum value of Q-value for the next state (s')

↳ Steps in Q-learning:

1. Initialize the Q-table with zeros.
2. Observe the current state (s).
3. Choose an action (a) using an exploration - exploration strategy (e.g. epsilon-greedy)
4. Perform the action (a) and observe the reward (r) and next state (s').
5. Update the Q-value using the Q-learning formula.
6. Repeat the process until the Q-table converges or

for a fixed number of episodes.

Working Practically

```
import numpy as np

# Define the grid world
grid = [
    ['S', 'O', 'O', 'O', 'G'],
    ['O', 'X', 'O', 'X', 'O'],
    ['O', 'X', 'O', 'X', 'O'],
    ['O', 'O', 'O', 'X', 'O'],
    ['O', 'X', 'O', 'O', 'O']
]

# Define the state and action space
num_states = 25    # 5x5 grid
num_actions = 4    # Up, Down, Left, Right

# Initialize Q-table with zeros
Q = np.zeros((num_states, num_actions))

# Parameters
alpha = 0.8    # Learning rate
gamma = 0.95   # Discount factor
epsilon = 0.1  # Exploration rate

# Define the reward function
def get_reward(state):
    row, col = divmod(state, 5)
```

```
if grid[row][col] == 'G':
    return 1
elif grid[row][col] == 'X':
    return -1
else:
    return 0

# Define the next state function
def get_next_state(state, action):
    row, col = divmod(state, 5)
    if action == 0 and row > 0:  # Up
        row -= 1
    elif action == 1 and row < 4:  # Down
        row += 1
    elif action == 2 and col > 0:  # Left
        col -= 1
    elif action == 3 and col < 4:  # Right
        col += 1
    return row * 5 + col

# Training the agent
for episode in range(1000):
    state = 0  # start position
    done = False

    while not done:
        if np.random.uniform(0, 1) < epsilon:
            action = np.random.choice(num_actions)
            # Explore
        else:
            action = np.argmax(Q[state, :])
```

#Exploit

```
next_state = get_next_state(state, action)
reward = get_reward(next_state)
Q[state, action] = Q[state, action] + alpha *
                   (reward + gamma * np.max(
                   Q[next_state, :]) -
                   Q[state, action])

state = next_state

if grid[divmod(state, 5)[0]][divmod(
        state, 5)[1]] == 'G' or grid[
        divmod(state, 5)[0]][divmod(
        state, 5)[1]] == 'x':
    done = True

print("Trained Q-table:")
print(Q)
```