ML-WEEK-12

Understand Q-Learning using python/scikit-learn

1. Understand the working of Q-Learning algorithm (refer to the material shared).
2. Implement Q-Learning in python (assuming a Grid/gymnasium/Equivalent)
3. Understand how to learn the q-function using Neural Networks.

****

Some Theory:

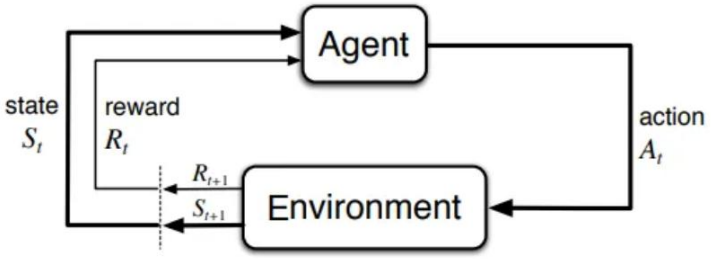
Reinforcement Learning (RL) is a type of machine learning. It trains an agent to make decisions by interacting with an environment. RL include states, actions, rewards, policies, and the Markov Decision Process (MDP).

Key Concepts in Reinforcement Learning

Reinforcement Learning (RL) involves several core ideas that shape how machines learn from experience and make decisions:

1. **Agent**: It's the decision-maker, that interacts with its environment.
2. **Environment**: The external system with which the agent interacts.
3. **State**: A representation of the current situation of the environment.
4. **Action**: Choices that the agent can take in a given state.
5. **Reward**: Immediate feedback the agent gets after taking an action in a state.
6. **Policy**: A set of rules the agent follows to decide its actions based on states.
7. **Value Function**: Estimates the expected long-term reward from a specific state under a policy.

Markov Decision Process



Markov Decision Process

A Markov Decision Process (MDP) is a mathematical framework. MDPs give a structured way to describe the environment in reinforcement learning.

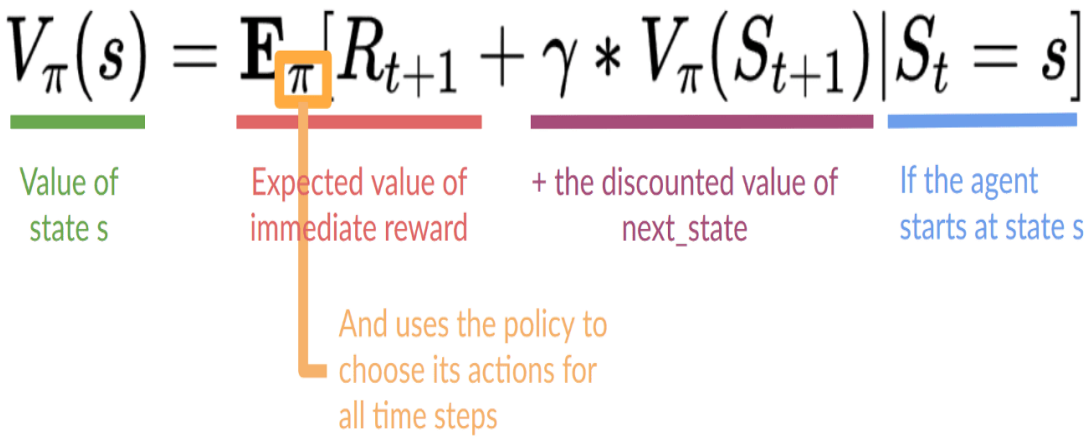
An MDP is defined by the tuple (S,A,T,R,γ). The components of the tuple are described below.

- **States**: A set of all possible states in the environment.
- **Actions (A)**: A set of all possible actions the agent can take.
- **Transition Model (T)**: The probability of transitioning from one state to another.
- **Reward Function (R)**: The immediate reward received after transitioning from one state to another.
- **Discount Factor (γ)**: A factor between 0 and 1 that represents the importance of future rewards.

Bellman Equation

The Bellman equation calculates the value of being in a state or taking an action based on the expected future rewards.

It breaks down the expected total reward. The first part is the immediate reward received. The second part is the discounted value of future rewards. This equation helps agents make decisions to maximize their long-term benefits.



Bellman Equation

Steps of Reinforcement Learning

1. **Define the Environment**: Specify the states, actions, transition rules, and rewards.
2. **Initialize Policies and Value Functions**: Set up initial strategies for decision-making and value estimations.
3. **Observe the Initial State**: Gather information about the initial conditions of the environment.
4. **Choose an Action**: Decide on an action based on current strategies.
5. **Observe the Outcome**: Receive feedback in the form of a new state and reward from the environment.

6. **Update Strategies**: Adjust decision-making policies and value estimations based on the received feedback.

Reinforcement Learning Algorithms

There are several algorithms used in reinforcement learning.

1. **Q-Learning**: A model-free algorithm that learns the value of actions in a state-action space.
2. **Deep Q-Network (DQN)**: An extension of Q-Learning using deep neural networks to handle large state spaces.
3. **Policy Gradient Methods**: Directly optimize the policy by adjusting the policy parameters using gradient ascent.
4. **Actor-Critic Methods**: Combine value-based and policy-based methods. The actor updates the policy, and the critic evaluates the action.

Q-Learning Algorithm

Q-Learning is a key algorithm in reinforcement learning. It is a model-free method. This means that it doesn't need a model of the environment. Q-Learning learns actions by directly interacting with the environment. Its main goal is to find the best action-selection policy that maximizes cumulative reward.

Key Concepts

- **Q-Value**: The Q-value, denoted as Q(s,a), represents the expected cumulative reward of taking a specific action in a specific state and following the policy thereafter.
- **Q-Table**: A table where each cell Q(s,a) corresponds to the Q-value for a state-action pair. This table is continually updated as the agent learns from its experiences.
- **Learning Rate (α)**: A factor that determines how much new information should overwrite old information It lies between 0 and 1.
- **Discount Factor (γ)**: A factor that reduces the value of future rewards. It also lies between 0 and 1.

Implementation of Q-Learning with Python

Import required libraries

Import the necessary libraries. '**gym**' is used to create and interact with the environment. Furthermore, 'numpy' is used for numerical operations.

```
import gymnasium as gym
import numpy as np
```

Initialize the Environment and Q-Table

Create the FrozenLake environment and initialize the Q-table with zeros.

```
env = gym.make("FrozenLake-v1", is_slippery=False)
Q = np.zeros((env.observation_space.n, env.action_space.n))
```

Define Hyperparameters

Define the hyperparameters for the Q-Learning algorithm.

```
learning_rate = 0.8
discount_factor = 0.95
epsilon = 0.1
episodes = 10000
max_steps = 100
```

Implementing Q-Learning

Implement the Q-Learning algorithm on the above setup.

```
for episode in range(episodes):
    state = env.reset()
    done = False

    for _ in range(max_steps):
        # Choose action (epsilon-greedy strategy)
        if np.random.uniform(0, 1) < epsilon:
            action = env.action_space.sample()
        else:
            action = np.argmax(Q[state, :])

        # Perform action and observe the outcome
        next_state, reward, done, _ = env.step(action)
```

```
        # Update Q-value using the Bellman equation
        Q [state, action] = Q [state, action] + learning_rate * (reward + discount_factor *
np.max(Q [next_state,:]) - Q [state, action])

        # Transition to next state
        state = next_state

        # If the episode is finished, break the loop
        if done:
            break
```

Evaluate the Trained Agent

Calculate the total reward collected as the agent interacts with the environment.

```
state = env.reset()
done = False
total_reward = 0

while not done:
    action = np.argmax(Q[state, :])
    next_state, reward, done, _ = env.step(action)
    total_reward += reward
    state = next_state
    env.render()
```

****

Code snippet:

```
import numpy as np

# 1. Setup: 4x4 GridWorld, Q-table initialization, hyperparameters
n_states, n_actions = 16, 4
Q_table = np.zeros((n_states, n_actions))
lr, gamma, epsilon = 0.8, 0.95, 0.2
epochs = 1000

def get_next_state(state, action):
    # Simplified grid movement
    row, col = divmod(state, 4)
    if action == 0: col = max(0, col - 1) # Left
    elif action == 1: col = min(3, col + 1) # Right
    elif action == 2: row = max(0, row - 1) # Up
```

```
    elif action == 3: row = min(3, row + 1) # Down
    return row * 4 + col

# 2. Q-Learning Training Loop
for _ in range(epochs):
    state = np.random.randint(0, n_states)
    while state != 15: # Goal state is 15
        if np.random.rand() < epsilon:
            action = np.random.randint(0, n_actions)
        else:
            action = np.argmax(Q_table[state])

        next_state = get_next_state(state, action)
        reward = 1 if next_state == 15 else 0

        # Q-update: Q(s,a) = Q(s,a) + alpha * (reward + gamma *
max(Q(s',a')) - Q(s,a))
        Q_table[state, action] += lr * (reward + gamma *
np.max(Q_table[next_state]) - Q_table[state, action])
        state = next_state

print("Learned Q-Table:\n", Q_table)
```
****