1.2 Reinforcement learning

Reinforcement learning (RL) is a type of machine learning that focuses on training agents to make decisions in an environment by maximizing a reward signal. The agent receives feedback in the form of rewards or penalties for its actions, and uses this feedback to improve its decision-making over time.

In RL, the agent interacts with an environment, where it selects actions to perform based on its current state. The environment then transitions to a new state and the agent receives a reward or penalty for its action. The agent's goal is to learn a policy, which is a mapping from states to actions, that maximizes the expected cumulative reward over time. One of the key components of RL is the use of a value function, which estimates the expected cumulative reward for a given state or state-action pair. The agent can use this value function to guide its decision-making, by selecting actions that lead to states with high estimated values. A popular algorithm for RL is Q-learning, which is a type of model-free, off-policy RL algorithm. Q-learning uses a Q-table to store the estimated values for all state-action pairs. At each time step, the agent selects the action that maximizes the Q-value for its current state. The Q-values are then updated using the Bellman equation, which expresses the Q-value for a state-action pair in terms of the Q-values for the next state.

Here's an example of Q-learning implemented in Python:

import gym

import numpy as np

# Define the Q-table and the learning parameters

q\_table = np.zeros((num\_states, num\_actions))

alpha = 0.1

gamma = 0.9

# Define the environment

env = gym.make("FrozenLake-v0")

# Define the training loop

for episode in range(num\_episodes):

    # Reset the environment and get the initial state

    state = env.reset()

    done = False

    while not done:

        # Select an action based on the current state

        action = np.argmax(q\_table[state,:] + np.random.randn(1,num\_actions)\*(1./(episode+1)))

        # Perform the action and observe the next state and reward

        next\_state, reward, done, info = env.step(action)

        # Update the Q-value for the current state-action pair

        q\_table[state, action] = q\_table[state, action] + alpha\*(reward + gamma\*np.max(q\_table[next\_state,:]) - q\_table[state, action])

        # Update the current state

        state = next\_state

In this example, we are using the FrozenLake-v0 environment from the OpenAI Gym library, which is a simple grid-based game where the agent must navigate to a goal while avoiding holes. The Q-table is initialized with all zeros, and the learning parameters are set to alpha = 0.1 and gamma = 0.9. The agent uses an epsilon-greedy strategy to select actions, where it selects the action with the highest Q-value with probability (1-epsilon) and a random action with probability epsilon. The Q-values are updated using the Bellman equation at each time step.