Q-Learning with Numpy (Reinforcement Learning)

Summery

The article applies **Q-learning** to the game of **Connect Four**, aiming to find an optimal strategy that maximizes a player's chance of winning. The environment is the Connect Four board, a 6×7 grid where each state represents the current placement of pieces, and the action space is dropping a disc into one of the available columns. The Q-learning algorithm updates its **Q-table** using the rule:

Q(s,a)←Q(s,a)+α[r+γmax⁡a′Q(s′,a′)−Q(s,a)]Q(s,a) \leftarrow Q(s,a) + \alpha \big[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \big]Q(s,a)←Q(s,a)+α[r+γa′maxQ(s′,a′)−Q(s,a)]

where α\alphaα is the learning rate, γ\gammaγ is the discount factor, rrr is the reward for a move, s′s's′ is the resulting board state, and a′a'a′ is the next action. By iteratively playing games against itself or another agent, the Q-learning agent gradually learns which moves are most likely to lead to victory.

Short Report: Results and Observations

A Q-learning agent was trained to play Connect Four using a dictionary-based Q-table and an ε-greedy action selection strategy. After 5,000 training episodes, the agent showed basic gameplay behavior but struggled to consistently make strong strategic moves due to the large state space and sparse rewards. Learning was slow, and performance against a human player was inconsistent.

This experiment demonstrates how Q-learning works in practice while also highlighting its limitations when applied to complex environments like Connect Four without advanced techniques such as self-play optimization or function approximation.