Islamic University of Technology

Report on Lab 06

(CSE 4618 Artificial Intelligence Lab)

Name: **Md Mahmudul Islam Mahin**
Student ID: **210042140**

Department: CSE
Program: BSc in SWE

Date: 8^{th} September 2025

Contents

1 Introduction

This report details the implementation of a Q-learning agent for the reinforcement learning tasks outlined in Lab 6. The objective was to develop a Q-learning algorithm to learn optimal policies through interaction with environments like Gridworld, Crawler, and Pacman by implementing key methods in qlearningAgents.py, including computeValueFromQValues, computeActionFromQValues, getAction, and update. This report discusses the problem, solution, findings, challenges, and hyperparameter exploration.

2 Problem Analysis

The task required implementing a Q-learning agent that learns an optimal policy without prior knowledge of the environment's Markov Decision Process (MDP). Unlike value iteration, Q-learning is a model-free approach that updates Q-values based on state-action-reward transitions. The challenge was to handle exploration versus exploitation, ties in action selection, and unseen state-action pairs with a Q-value of 0, which could be optimal if known actions had negative Qvalues. The implementation needed to support diverse environments (Gridworld, Crawler, Pacman) with varying state and action spaces.

3 Solution Explanation

The Q-learning agent was implemented in qlearningAgents.py with the following approach:

Initialization:

Used util.Counter to store Q-values, initialized to 0 for unseen state-action pairs.

getQValue:

Retrieves the Q-value for a state-action pair, returning 0.0 for unseen pairs.

computeValueFromQValues:

Computes the maximum Q-value over legal actions, returning 0.0 for terminal states.

computeActionFromQValues:

Selects the action with the highest Q-value, breaking ties randomly using random.choice.

getAction:

Implements epsilon-greedy action selection using util.flipCoin.

update:

Updates Q-values using the formula:

$$Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot (r + \gamma \cdot \max_{a'} Q(s',a'))$$

Key decisions included using util.Counter for efficient storage and random tie-breaking to avoid biases. The complexity is O(—A—) per update, where —A— is the number of legal actions. Here is the update method:

```
def update(self, state, action, nextState, reward):
    oldQ = self.getQValue(state, action)
    nextValue = self.computeValueFromQValues(nextState)
    self.qValues[(state, action)] = (1 - self.alpha) * oldQ + self.alpha * (reward
     + self.discount * nextValue)
```

4 Findings and Insights

In Gridworld, the agent learned the optimal policy over multiple episodes, with Q-values converging when guided along the optimal path for four episodes with noise disabled. The epsilon-greedy strategy led to early exploration of suboptimal actions, which improved long-term learning by discovering better paths, emphasizing the exploration-exploitation trade-off.

5 Challenges Faced

Ensuring random tie-breaking in computeActionFromQValues was challenging; initial deterministic selection caused biases, resolved by using random.choice. Debugging Q-value updates to match autograder expectations required testing with –noise 0.0.

6 Hyperparameter Exploration

Performance varied with hyperparameters:

- **alpha (learning rate):** High alpha (e.g., 0.5) caused unstable Q-values, while alpha=0.2 ensured stable but slower convergence.
- **gamma (discount factor):** gamma=0.8 balanced immediate and future rewards, while gamma=0.9 favored long-term rewards, risking overfitting.

7 GitHub Repository

I will be uploading the lab tasks in the following repository: CSE 4618: Artificial Intelligence