CS 554 – Introduction to Machine Learning and Artificial Neural Networks
Assignment-5: Q-Learning
Assignment Report

Osman Furkan Kınlı – S002969

1. Introduction

In this assignment, we implement Q-Learning algorithm which is a well-studied Reinforcement Learning algorithm. The main idea behind Q-Learning is to observe the movements of the agent in an environment to maximize the reward that the agent can gain by exploiting and exploring the environment iteratively. The agent uses the experience during the iterations to maximize its final reward, and record it to a table, namely Q-Table. Initially, this table contains all zeros, so it leads to have random moves in the environment for the agent. As finding the final reward during iterations, the randomness in the movements give the place to more reliable movements. The formula of updating the Q-Table for each iteration as follows:

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$

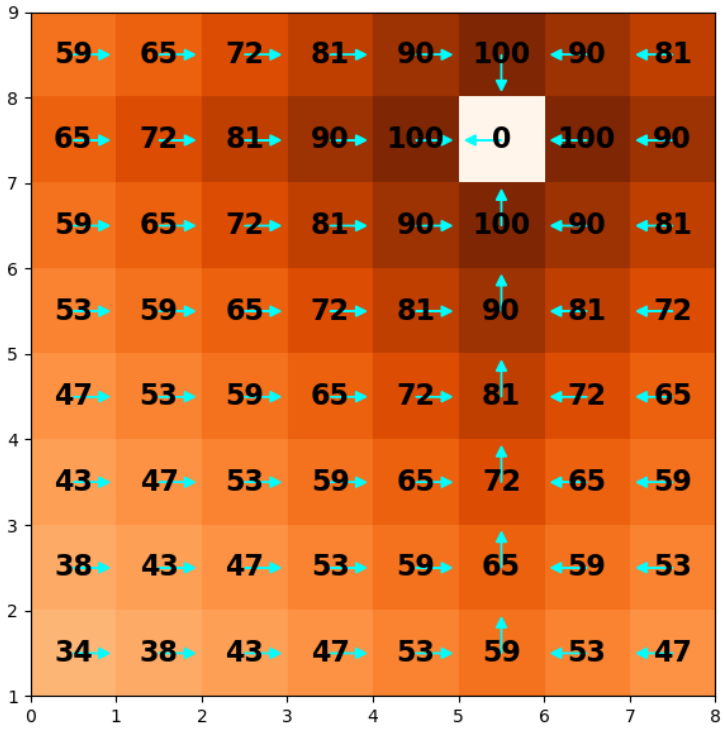
where α is learning rate, which decides how each experience in each iteration changes the movements, γ is discount factor, which determines how much new experience changes older experiences. Finally, the agent can find the goal state with the help of the whole process.

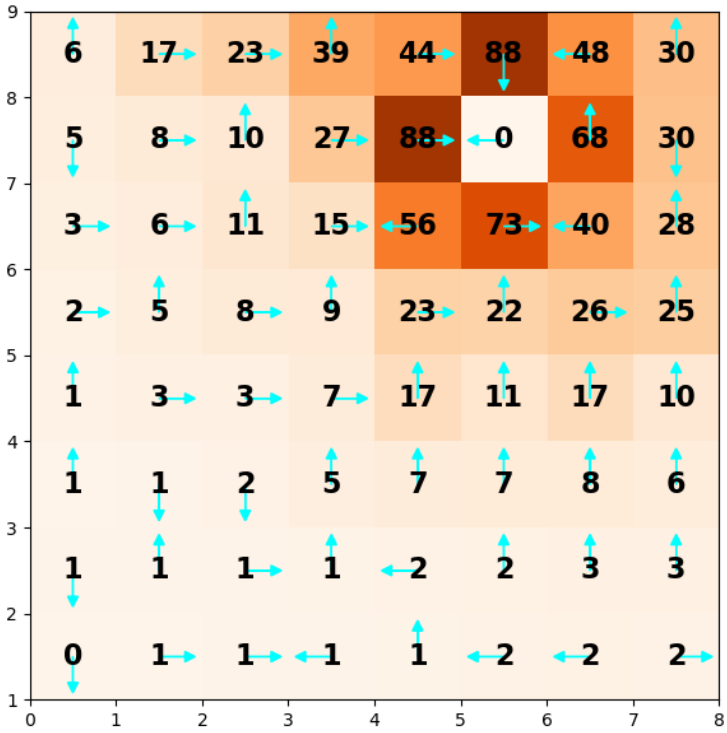
2. Environment

Consider 8x8 grid table for our environment, and an agent can move in four dimensions, north, south, east and west. The agent has an initial start point where it is the lowest left corner in the grid table. The goal state gives an immediate reward of 100 to the agent when it is reached, while the other states has no reward. The discount factor is 0.9, the learning rate 0.5. Once learning is completed, the agent can find the goal state, and knows how to reach it in the most efficient way.

Language: Python 3.6
Plotting library: Matplotlib
Grid table is implemented on NumPy

3. Results


59 65 72 81 90 100 90 81
65 72 81 90 100 0 100 90
59 65 72 81 90 100 90 81
53 59 65 72 81 90 81 72
47 53 59 65 72 81 72 65
43 47 53 59 65 72 65 59
38 43 47 53 59 65 59 53
34 38 43 47 53 59 53 47


6 17 23 39 44 88 48 30
5 8 10 27 88 0 68 30
3 6 11 15 56 73 40 28
2 5 8 9 23 22 26 25
1 3 3 7 17 11 17 10
1 1 2 5 7 7 8 6
1 1 1 1 2 2 3 3
0 1 1 1 1 2 2 2

4. Code

```
import numpy as np
from matplotlib import pyplot as plt

grid_size = 8
Qs = np.zeros(shape=(grid_size, grid_size, 4))
rewards = np.zeros(shape=(grid_size, grid_size))

initial_state = (7, 0)  # given
goal_state = (1, 5)  # given

rewards[goal_state] = 100

discount_factor = 0.9
lr = 0.5

actions = [0, 1, 2, 3]
action_indices = [(0, -1), (0, 1), (-1, 0), (1, 0)]
NUM_EPISODE = 10000


def init():
    global Qs
    Qs = np.zeros(shape=(grid_size, grid_size, 4))


def epsilon_greedy(s, eps=0.9):
    val = np.random.uniform()
    if val < eps:
        return np.random.randint(low=0, high=len(actions))
    return np.argmax(Qs[s])


def is_valid_action(s, a):
    return (a == 0 and s[1] > 0) or \
           (a == 1 and s[1] < grid_size - 1) or \
           (a == 2 and s[0] > 0) or \
           (a == 3 and s[0] < grid_size - 1)


def take_action(s, a, det=True):
    if det is False:
        val = np.random.uniform()
        if val >= 0.5:
            if a == 0 or a == 1:
                if val <= 0.75:
                    a = 2
                else:
                    a = 3
            elif a == 2 or a == 3:
                if val <= 0.75:
                    a = 0
                else:
```

```
            a = 1

    if is_valid_action(s, a):
        a_i = action_indices[a]
        s = (s[0] + a_i[0], s[1] + a_i[1])

    return s, rewards[s]


def run(det=True):
    for _ in range(NUM_EPISODE):
        s = initial_state
        a = epsilon_greedy(s)
        while s != goal_state:
            s_prime, r = take_action(s, a, det=det)
            if det:
                if s != s_prime:
                    Qs[s][a] = Qs[s][a] + lr * (r + discount_factor * np.max(Qs[s_prime]) - Qs[s][a])
                a = epsilon_greedy(s_prime)
                s = s_prime
            else:
                a_prime = epsilon_greedy(s_prime)
                if s != s_prime:
                    Qs[s][a] = Qs[s][a] + lr * (r + discount_factor * Qs[s_prime][a_prime] - Qs[s][a])
                s = s_prime
                a = a_prime


def visualize():
    max_vals = np.max(Qs, axis=-1)
    max_actions = np.argmax(Qs, axis=-1)

    plt.figure(figsize=(6, 6))
    plt.imshow(max_vals, cmap='Oranges', interpolation='nearest', vmin=0, vmax=100)

    ax = plt.gca()
    ax.set_xticks(np.arange(9) - .5)
    ax.set_yticks(np.arange(9) - .5)
    ax.set_xticklabels(range(9))
    ax.set_yticklabels(range(9, -1, -1))

    action_dict = {0: (-1, 0), 1: (1, 0), 2: (0, 1), 3: (0, -1)}

    for y in range(grid_size-1, -1, -1):
        for x in range(grid_size-1, -1, -1):
            best_action = max_actions[y, x]
            best_value = max_vals[y, x]

            plt.text(x, y, str(int(best_value)), color='black', size=16, verticalalignment='center',
horizontalalignment='center',
                     fontweight='bold')

            u, v = action_dict[best_action]
            plt.arrow(x, y, u * .3, -v * .3, color='cyan', head_width=0.12, head_length=0.12)
    plt.show()
```

```
run()
visualize()

init()

run(det=False)
visualize()
```