DPRL Assignment 4: Q-learning

Zhiyan Yao
Vrije Universiteit Amsterdam
z.yao2@student.vu.nl

Bo-Chian Chen
Vrije Universiteit Amsterdam
b.chen4@student.vu.nl

I. Problem Definition

Given a 3×3 mini maze and terminate reward 1 at top right, starting from bottom left, the first task involves using tabular and ϵ-greedy Q-learning to find optimal Q-values for the maze, indicating expected cumulative rewards. In addition, it entails calculating state-action pairs with a focus on fine-tuning crucial hyperparameters (α and ϵ) in the ϵ-greedy Q-learning algorithm. The aim is to optimize learning and convergence toward the goal state.

For the second task, solve the maze problem with function approximator for discount factor $\gamma = 0.9$ and 5×5 grid instead of 3×3.

II. State and action space

A. The state space $\mathcal{X}$

State can be encoded from 0 to 9, indicating the current position of agent, shown in the table below, and the initial state is always 6 and terminate state is 2.

0	1	2
3	4	5
6	7	8

B. The action space $\mathcal{A}$

Action can be encoded from 0 to 3, indicating move up, right, down and left.

III. Tabular Q-learning

We recursively get the discounted reward from the next state. And the selection of next state is based on the formula:

$$Q\left(x\right) = \max\left(\gamma * Q(\text{approachable state})\right)$$

where the approachable state is defined as x±3 or x±1.

A. 0ptimal Q-values

By executing the script "tabular_Q_learning.py," one can observe the dynamic updates to the Q-table throughout the learning process. Additionally, the corresponding policies employed at each iteration can be scrutinized. This iterative exploration and refinement ultimately yield the final Q-table and the corresponding optimal policy.

B. Evaluation

It's the right trajectory, which is verified by manual examination.

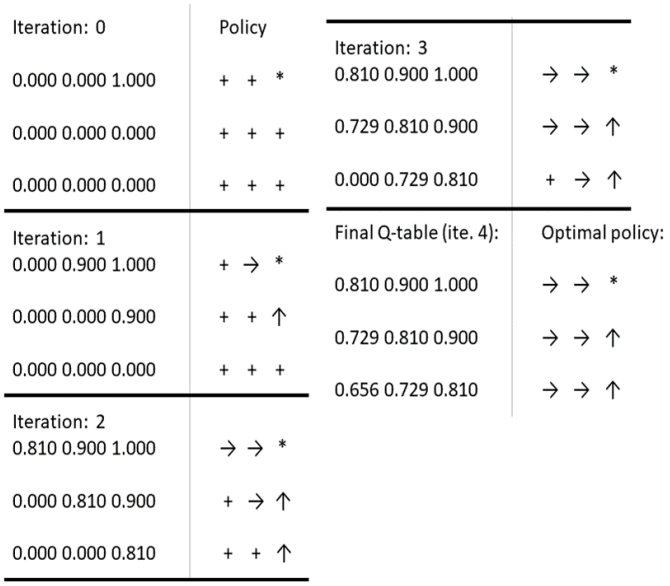

Fig. 1. Tabular Q-learning iteration

IV. ϵ-greedy Q-learning

A. Implementation

First, this implementation begins by initializing a Q-table with arbitrary values, setting all values to 0 in this case. The sole goal state is designated with a reward of 1. Second, the strategy for selecting the next state follows an ϵ-greedy approach. With a probability ϵ, a random approachable state is chosen; otherwise, the state with the highest Q-value is selected. Third, executing ϵ-greedy Q-learning involves specifying a pair of hyperparameters, namely the learning rate (α) and the exploration rate (ϵ). Consequently, to comprehensively explore the performance of the algorithm, various pairs of (α, ϵ) are employed. The resulting relationships between these hyperparameters and the average steps taken to reach the goal state are systematically visualized through plotting in Fig. 2. This analysis provides insights into the algorithm's behavior under different exploration-exploitation strategies.

B. Evaluation

Using ϵ-greedy Q-learning, we get the same optimal policy as tabular one. Besides, from Fig. 2, we can see the influence of learning rate variations is subtle, yet a slight decrease in average steps is notable with higher rates. Notably, a smaller

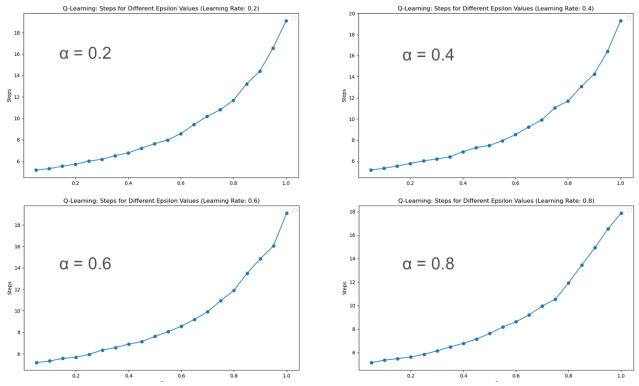

Fig. 2. α - ϵ relation graph (x-axis: ϵ, y-axis: average steps)

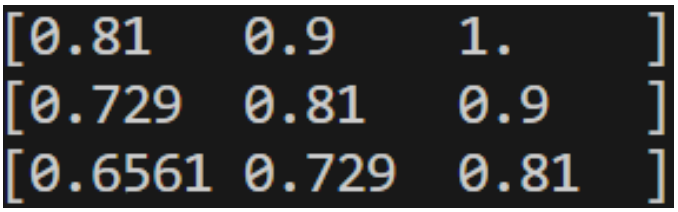

Fig. 3. optimal Q-values from ϵ-greedy Q-learning iteration

ϵ is preferred. Hence, the recommendation is to employ a higher learning rate and a lower exploration rate for optimal performance in this environment.

V. Q-Learning with function approximator

We use a simple neural network as function approximator to represent value function. Let θ denotes parameters of neural network, x denotes current state, and a denotes action, then we have

$$Q(x,a) \approx Q(x,a;\theta) \tag{1}$$

The parametere θ are updated with gradient desent:

$$\theta := \theta - \alpha \frac{d}{d\theta}(Q(x,a;\theta) - Y_k^Q)^2 \tag{2}$$

with

$$Y_k^Q = reward + \gamma \underset{a' \in \mathcal{A}}{max} Q(x', a'; \theta_k) \tag{3}$$

where α is learning rate of the optimizer, γ is discount factor, in this case $\gamma = 0.9$, x' is possible next state from current state x and $\mathcal{A}$ is action space.

A. State and action space

State space is extend to 5×5 from 3×3, shown in the table below, where initial state is 20 and terminate state is 4. And action space still stays the same.

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

B. Neural network architecture

We use a simple multi-layer perceptron(MLP) of 4 dense layers for function approximation because the problem is relatively easy and thus does not require complicated architectures. The input layer consists of 2 inputs, which are state and action, and then two hidden layers with 50 and 20 neurons respectively, in the end the output layer of a single neuron for predicting Q value. The detailed architecture is shown in figure 4.

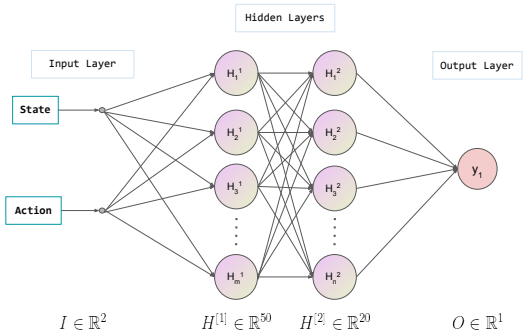

Fig. 4. Neural network architecture

C. Implementation

Firstly, adding several samples $< x, a, Y >$ to empty dataset by randomly choosing actions and getting reward, where Y is derived by equation 4. After gathering enough data, start from the initial state, choose action based on ϵ-greedy policy and get corresponding reward, and then still add current sample to dataset and run one epoch of training. If the current state if terminate state then terminate the current loop and proceed to the next episode. After hundreds of epiodes, optimal Q values are derived.

D. Optimal Q-values

After convergence, we can get the optimal Q-values by

$$Q(x) = \underset{a \in \mathcal{A}}{max} Q(x, a; \theta_k) \tag{4}$$

The optimal Q-values table is shown in the table below,

0.670	0.881	0.705	0.882	0.996
0.662	0.605	0.824	0.553	0.214
0.203	0.158	0.142	0.138	0.140
0.143	0.128	0.120	0.118	0.116
0.114	0.114	0.114	0.114	0.113

where the red trajectory is the optimal policy derived by

$$policy(x) = \underset{x'}{argmax}\{Q(x')\}$$

E. Hyper-parameters and convergence

1) Learning rate α: We studied the effect of optimizer learning rate in equation 2 on convergence by running the algorithm under 3 different learning rate: 0.1, 0.01 and 0.001. Different loss curves are shown in figure 5, where blue line represents learning rate 0.1, orange line represents learning rate 0.01 and green line represents learning rate 0.001. From the plot we could see that with larger learning rate, loss decreases faster in the early stage of training, but with unstable fluctuations, which is obvious for the blue line; with smaller learning rate, loss decreases slower but smoother, which is the case for the green line. When it comes to choosing learning rate, it depends on what matters, speed of stability. In general, we prefer something in the middle, in our case, learning rate = 0.01 gains the best performance.

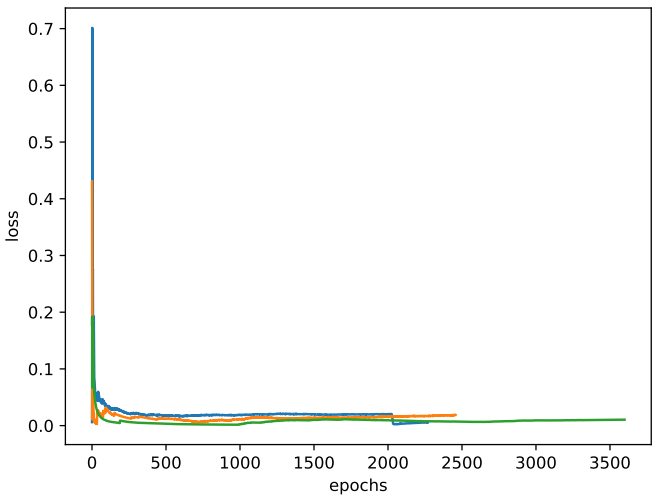

Fig. 5. loss curves of different learning rates