EE5904/ME5404 Neural Networks AY2024/2025 Semester 2

Q-Learning for World Grid Navigation

XU NUO

A0313771H

e1499166@u.nus.edu

April 17, 2025

1. Project Description

The project focuses on implementing the Q-Learning algorithm to train a robot to traverse on a 10×10 grid world, moving from the top-left cell (start state) to the bottom-right cell (goal state). It can take one of four deterministic actions: move up, right, down or left, and receives a reward based on the current state-action pair. The robot is to reach the goal state by maximizing the total reward of the trip.

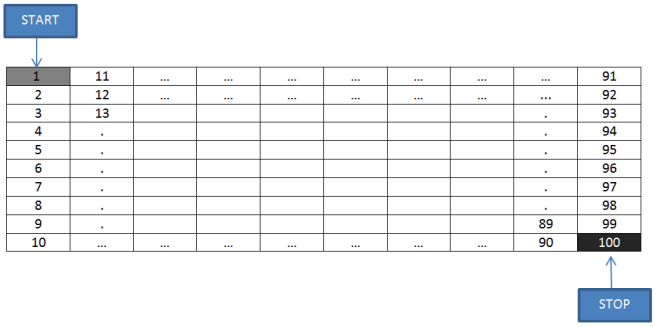

Fig. 1. Illustration of the grid with start state and goal state

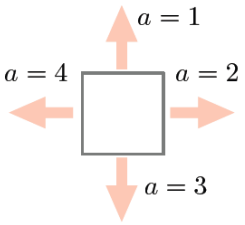

Fig. 2. Possible actions of the robot

Task 1 requires the implementation of Q-Learning with ε-greedy exploration, where the exploration rate ε_k and learning rate α_k both follow one of four decaying functions of the episode index k, as specified:

- $\varepsilon_k = \alpha_k = \frac{1}{k}$
- $\varepsilon_k = \alpha_k = \frac{100}{100+k}$
- $\varepsilon_k = \alpha_k = \frac{1+\log(k)}{k}$
- $\varepsilon_k = \alpha_k = \frac{1+5\log(k)}{k}$

For each ε_k setting and discount rate $\gamma = \{0.5, 0.9\}$, the algorithm is to be executed 10 times, and the maximum number of episodes in each run is set to 3000. The number of goal-reaching runs and average execution time are recorded. The optimal policy should be visualized in three ways as:

- A column vector representing the optimal policy
- A 10×10 grid of directional arrows showing the optimal policy
- A 10×10 grid showing the optimal path and its corresponding reward

Task 2 requires the implementation of Q-Learning using our own ε_k decay function and tuning the relevant parameters, which will be used to find an optimal policy using a reward function not provided.

2. Project Implementation

2.1 Q-Learning Algorithm

Given a set of states $S = \{s_1, s_2, \dots\}$ and a set of actions $A = \{a_1, a_2, \dots\}$, if the robot takes an action a to move from state s to state s', it will receive a corresponding reward r after completing the move. Then, the reward robot receives at time t, when it takes action a_t and moves from state s_t to state s_{t+1}, can be expressed as:

$$r_{t+1} = \rho(s_t, a_t, s_{t+1})$$

where ρ is a matrix of size $|S| \times |A| \times |S|$, also known as the reward function.

If the robot starts from initial state and reaches s after t times steps, the total reward from the current state onward is defined as:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

Where k is the index of time steps after t, with $k = 0$ being 1^{st} step, γ is called the discount rate, with $0 \leq \gamma \leq 1$. R_t determines present value of future rewards, the further away a reward is from the current state, the lower its value. A small γ forces the robot to pay more attention to immediate rewards, responds more quickly, but may behave greedily and ignore long-term outcomes. A large γ forces the robot to focus more on long-term rewards, becomes more foresighted, and is better to make plans for future actions.

A policy $\pi(s) = a, s \in S, a \in A$ is a rule that decides the action the robot is to take at any given state, and each π is associated with a set of $Q^{\pi}(s, a)$ values:

$$Q^{\pi}: S \times A \to \mathbb{R}$$

$$Q^{\pi}(s, a) = \mathbb{E}^{\pi}[R_t | s_t = s] = R_t | s_t = s \text{ (deterministic transition)}$$

where $Q^{\pi}(s, a)$ is defined as the expected return from taking action a at state s at time step t, and thereafter following policy π.

An optimal policy is one that maximizes values of Q-function over all possible (s, a) pairs with respect to the given task. To solve this problem, Bellman Equation is required:

$$Q^{\pi}(s, a) = \mathbb{E}^{\pi}[r_{t+1}] + \mathbb{E}^{\pi}\left[\gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s\right]$$

$$or \quad Q^{\pi}(s, a) = \mathbb{E}^{\pi}\left[r_{t+1} + \gamma Q^{\pi}\left(s_{t+1}, \pi(s_{t+1})\right) \middle| s_t = s, a_t = a\right]$$

It consists of two parts: the immediate reward received after executing the action, and

the discounted sum of all future rewards. Among all possible policies, the optimal Q-function is defined as the maximum expected return achievable by taking action a at state s and thereafter following the policy π:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

2.2 ε-greedy Exploration

A policy is called a greedy policy if at each s, a that yields the largest value for the Q-function is selected. A greedy policy is optimal if it satisfies:

$$\pi^*(s) \in \arg\max_{a} Q^*(s,a)$$

Exploration and exploitation are both necessary during learning. Exploration requires the robot to try action other than currently known best action:

$$a_{k+1} \neq \max_{a'} Q_k(s_{k+1}, a')$$

Exploitation requires the robot to use greedy policy to select currently known best action:

$$a_{k+1} = \max_{a'} Q_k(s_{k+1}, a')$$

The method to balance exploration-exploitation trade-off is called ϵ-greedy exploration. The way it works can be expressed as follow:

$$a_k = \begin{cases} a \in \arg\max_{\hat{a}} Q_k(s_k, \hat{a}) & \textbf{with probability } \mathbf{1-\varepsilon_k} \\ \text{an action uniformly randomly selected from all other actions available at state } s_k & \textbf{with probability } \boldsymbol{\varepsilon_k} \end{cases}$$

ε is kept small so that the robot will focus on task at hand most of the time, and will explore only occasionally, and ε typically reduces as learning continues.

2.3 Implementation Process

In this section, I will describe the implementation details of the Q-Learning algorithm applies to Task 1.

The world where the robot navigate is a 10×10 grid. The robot can take one of four actions at each state: moving up, right, down and left, which are numbered accordingly (1,2,3,4). The Q-table is initialized as a 100×4 zero matrix, representing the estimated value of taking each action at each state:

$$Q(s,a) \leftarrow 0, \ \ \forall s \in S, a \in A$$

The training process is organized into 3 main loops:

- Outer loop ($run = 1{\sim}10$): repeated executions to evaluate stability.
- Trial loop ($trial = 1{\sim}3000$): each trial represents a full episode starting from the initial state. Q-table will be updated along the way.
- Step loop ($steps \leq 1000$): the robot navigates through the grid until it reaches the goal or exceeds the maximum allowed steps. If the number of steps the robot navigates in the grid is not limited, the robot may fall into a local optimization dead loop, causing the program to be unable to continue executing under certain parameter settings.

In the implementation of Q-Learning, the program first follows the ϵ-greedy exploration algorithm. It compares a randomly generated number with the current trial's corresponding ε value. If the random number is less than ε, the agent continues to explore the surrounding environment randomly; otherwise, it exploits the current best-known action based on the greedy policy. Subsequently, the Q-table is updated according to the iterative update rule. The general mathematical form is given as:

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha_k \left(r_{k+1} + \gamma \max_{a'} Q_k(s_{k+1}, a') - Q_k(s_k, a_k) \right)$$

To improve training efficiency, I introduced two early stopping criteria. The first one is based on the convergence of the exploration rate. If $\varepsilon_k < \varepsilon_{\text{max\_trial}} \times 1.1$, it means that the robot almost no longer conducts random exploration and there is little point in continuing training. The second one is based on the difference between the Q-tables. If the maximum difference between the flattened Q-tables across two successive iterations is less than the threshold 0.05, it indicates that the Q-table has converged and the training can be terminated.

After each run, the learned Q-table is used to derive a greedy policy. This policy is then evaluated by simulating its execution from the initial state. If the robot reaches the goal state, the path is recorded and its total accumulated reward is calculated. If this reward is higher than any previously recorded one, it is saved as the best policy.

To ensure robust policy learning, I compare the cumulative rewards across 10 independent runs under each combination of ε and γ values. The best policy per setting is selected by tracking the maximum reward. Further, the overall best policy across all settings is identified by comparing these best-per-setting rewards and recorded for final evaluation and visualization.

3. Comments on Task 1

To meet the requirements for result analysis and visualization, for each combination of exploration rate and discount rate, both the locally optimal policy, the overall best

policy and path were extracted, visualized and saved in the folder ***task1_clip***. The following table summarizes the main output files, their contents and the corresponding variables used in the program.

Filename	Description	Source Variable
optimal_policy_epsilonX_gammaY.mat	The optimal policy (as a vector of actions) for a specific setting of epsilon and gamma	best_policy
Path_epsilonX_gammaY.png	The path trajectory taken under the best policy for the current setting	best_policy_actions
Policy_epsilonX_gammaY.png	The complete optimal policy extracted from the Q-table	best_Q
best_overall_policy.mat	The globally optimal policy with the highest reward across all parameter settings	global_best_policy
best_overall_path.mat	The best trajectory of actions taken along the globally optimal path	global_states_trans

TABLE 1. Main Output Files

ε_k, α_k	No. of goal-reached runs		Execution time (sec.)	
	$\gamma = 0.5$	$\gamma = 0.9$	$\gamma = 0.5$	$\gamma = 0.9$
$\frac{1}{k}$	0	0	N.A.	N.A.
$\frac{100}{100+k}$	0	10	N.A.	0.00342
$\frac{1+\log(\mathrm{k})}{\mathrm{k}}$	0	3	N.A.	0.01916
$\frac{1+5\log(k)}{k}$	0	0	N.A.	N.A.

TABLE 2. Parameter Values and Performance of Q-Learning

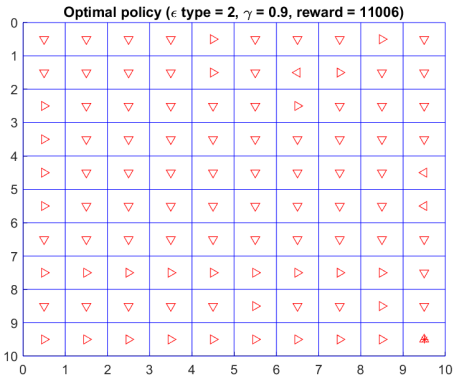

Fig. 1. Optimal Policy

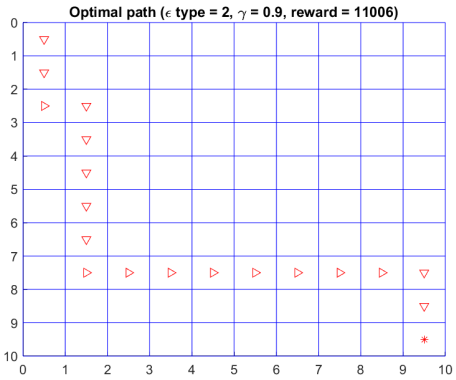

Fig. 2. Optimal Path

The results are summarized in Table 2, which shows the number of successful goal-reaching runs and the average execution time under each configuration. Figure 1 and Figure 2 illustrate the global optimal policy and the corresponding optimal path, which are obtained when the exploration rate and learning rate are set to $\frac{100}{100+k}$, and the discount rate γ is set to 0.9. Following this path, the robot achieves a total reward of 11006.

Impact of Discount Rate γ:

For all parameter settings of ε_k/α_k, the robot failed to reach the goal even once when $\gamma = 0.5$. This aligns with the nature of the discount rate. $\gamma = 0.9$ makes the robot more foresighted and encourages consideration of long-term rewards, which is crucial for this task that requires many steps to reach the goal state. In contrast, $\gamma = 0.5$ heavily discounts future rewards, leading to myopic decisions and premature convergence to suboptimal behavior.

Impact of ε_k/α_k:

The $\frac{1}{k}$, $\frac{1+\log(k)}{k}$ and $\frac{1+5\log(k)}{k}$ strategies decay too aggressively, causing the robot to stop exploring early and get trapped in local loops or suboptimal paths. This results in 0 or few successful runs for both γ values. The best-performing strategy is $\frac{100}{100+k}$, which decays gradually and balances exploration and exploitation effectively. It achieves 10 out of 10 successes when $\gamma = 0.9$.

Execution Time:

From the analysis of the two ε functions that successfully led the robot to the goal state, it can be observed that $\varepsilon_k = \frac{100}{100+k}$ decreases slowly with increasing k, and maintains relatively high values. This means that the robot can explore more under this setting, making it easier to discover the optimal path, and thus the execution time

is relatively short. On the other hand, $\varepsilon_k = \frac{1+\log(k)}{k}$ decreases faster with k, resulting in a smaller epsilon value. With this setting, the robot behaves more greedily, which increases the risk of falling into local optima and makes it harder to find the best path, leading to a significantly longer execution time.

4. Design on Task 2

From the results of Task 1, it can be observed that a relatively high discount rate and an ε function that decays slowly while maintaining relatively large values are particularly beneficial for solving this type of task. Therefore, for Task 2, I designed a new epsilon function:

$$\varepsilon_k = e^{-0.001k}$$

and fixed the discount rate at $\gamma = 0.9$. After testing this model on a dummy version of qeval.mat, it successfully completed all 10 runs within an extremely short execution time and produced the optimal policy and path. To avoid confusion with the official test version of qeval.mat, I stored the dummy qeval.mat as well as the corresponding outputs in the folder ***task2_clip*** for reference. The comparison of different ε_k functions is plotted as Figure 3 below.

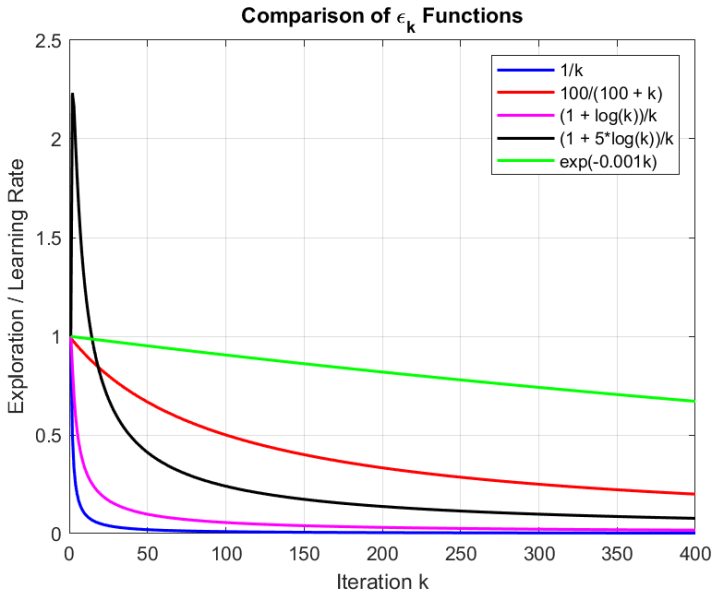

Fig. 3. Comparison of Different ε Functions