Course EE5904 Neural Networks Part II

Project 2: Q-Learning for World Grid Navigation

Lecturer: **Assoc. Prof. Peter C. Y. Chen**

Author: XU YIMIAN
Student ID: A0295779Y
Email: e1349917@u.nus.edu

National University of Singapore

Apr, 2025

Contents

List of Tables

List of Figures

1 Q-Learning Algorithm Design

The learning process follows the standard Q-learning framework with ε-greedy exploration, tailored for a 10×10 grid world.

1.1 Episode and Trial Setup

Each learning run consists of N episodes. In each episode, the agent starts from state $s_1 = 1$ and moves step-by-step using an ε-greedy policy until it reaches the goal state $s = 100$. The Q-table is shared across episodes and updated incrementally. Initialization steps include:

- γ (discount factor),
- $\varepsilon_k = 1 - \frac{1}{k}$ (exploration probability at time step k),
- $\alpha_k = \varepsilon_k$ (learning rate),
- $k = 1$, initial timestep,
- $s_k = 1$, initial state,
- The Q-table is initialized to zero only once at the beginning of the run.

1.2 Action Selection and Q-Update

At each step k, the agent selects an action a_k based on an ε_k-greedy strategy:

$$a_k = \begin{cases} \text{argmax}_{\hat{a}}\, Q(s_k, \hat{a}) & \text{with probability } 1 - \varepsilon_k \\ \text{random from available actions} & \text{with probability } \varepsilon_k \end{cases}$$

The agent moves to the next state s_{k+1}, receives reward r_k, and updates the Q-value:

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha_k \left[r_k + \gamma \cdot \max_a Q_k(s_{k+1}, a) - Q_k(s_k, a_k) \right]$$

1.3 Termination Conditions

An episode terminates under the following conditions:

- The agent reaches the goal state $s = 100$;

- The learning rate α_k drops below 0.005 (optional early stopping).

After each episode, the learned Q-table is retained and passed to the next episode.

2 Task 1

2.1 Overview and Experiment Setup

The Q-learning agent was initialized with a zero-valued Q-table and allowed to interact with the grid environment for up to 3000 episodes per run, with a total of 10 independent runs. In each episode, the agent began from the initial state $s = 1$ and aimed to reach the goal state $s = 100$, learning from rewards and updating the Q-table using the standard Bellman update formula.

To ensure training efficiency and avoid excessive computation, multiple convergence and early termination criteria were implemented in each Q-learning run:

- A maximum of 3000 episodes was set for each run. Within each episode, the agent interacted with the environment until it reached the terminal state ($s = 100$), the learning rate ε_k dropped below 0.05, or the maximum number of **steps** exceeded 10000.
- As mentioned above, to prevent infinite loops in each episode, a **step counter** was decremented at every step. If it reached zero, the episode was forcefully terminated and flagged as "Max steps: early stopping."
- After each episode, convergence of the Q-table was monitored by comparing the element-wise difference between two successive tables:

  $$\max|\delta Q| = \max|Q^{(t+1)}(s,a) - Q^{(t)}(s,a)|$$

  If this quantity remained below 10^{-4} for 300 consecutive episodes, the run was terminated early, as the Q-values were deemed to have converged.

This convergence check provides a reliable and computationally efficient way to detect stabilization in value updates, helping reduce unnecessary episodes once policy learning plateaus.

The ε_k strategies tested include both fast-decaying and slow-decaying schedules, such as:

$$\varepsilon_k \in \left\{ \frac{1}{k}, \frac{100}{100+k}, \frac{1+\log k}{k}, \frac{1+5\log k}{k}, \frac{200}{200+k}, \frac{300}{300+k}, \frac{500}{500+k} \right\}$$

with discount factors $\gamma \in \{0.5, 0.9\}$ in the initial evaluation and $\gamma \in \{0.7, 0.8, 0.9\}$ for the grid search.

2.2 Results and Comparative Analysis

Figure 1 and Figure 2 below show examples of optimal policy arrows and the best execution trajectory learned under the setting $(\gamma = 0.9, \varepsilon_k = \frac{100}{100+k})$. The learned policy encourages steady progression to the goal state $(10, 10)$ along the bottom-right diagonal, reflecting an efficient navigation strategy.

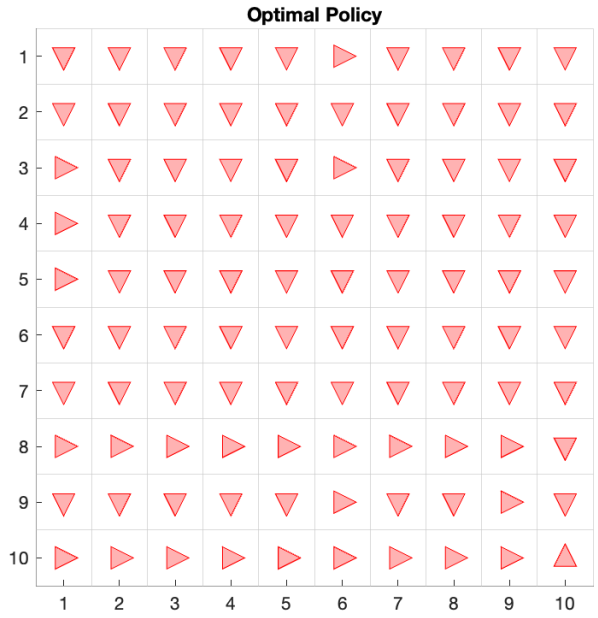

Figure 1: Optimal policy learned under $(\gamma = 0.9, \varepsilon_k = \frac{100}{100+k})$

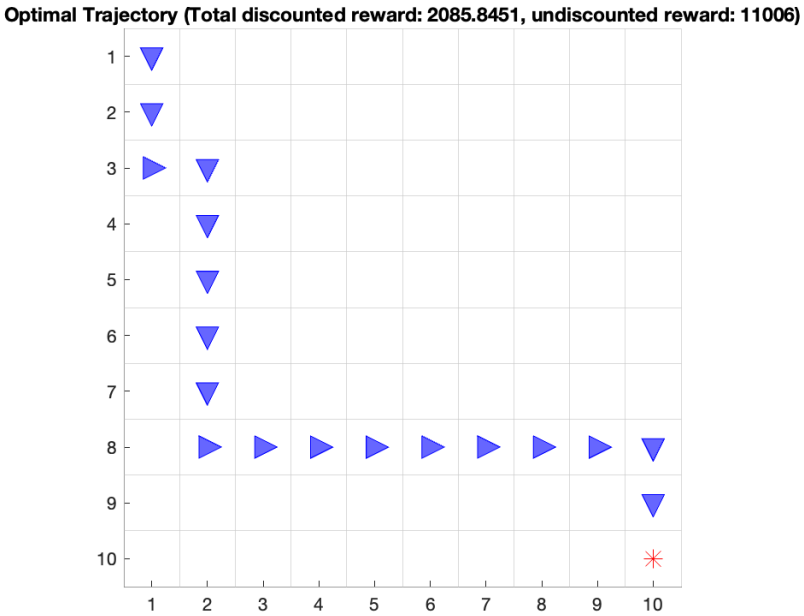

Figure 2: Execution trajectory under optimal policy in Task 1 (total discounted reward: `2085.8451`)

Table 1 summarizes the performance across all tested parameter pairs, highlighting success rate (out of 10) and average execution time. Notably, all slow-decay schedules ($\varepsilon_k = \frac{100}{100+k}, \frac{1+5\log k}{k}$, etc.) achieved perfect success rates across multiple discount factors, while fast-decay schedules (e.g., $\varepsilon_k = \frac{1}{k}$) frequently failed to converge.

Table 1: Q-learning success rates and runtimes under different parameter settings

ε_k, α_k	No. of goal-reached runs		Execution time (sec.)	
	γ=0.5	γ=0.9	γ=0.5	γ=0.9
$\frac{1}{k}$	0	0	-	-
$\frac{100}{100+k}$	0	10	-	0.2981
$\frac{1+log(k)}{k}$	0	0	-	-
$\frac{1+5log(k)}{k}$	0	10	-	0.4298

2.3 Comments on Results

The comparative analysis reveals several insights:

- **Impact of discount factor:** A low $\gamma = 0.5$ led to zero successful runs across all ε_k schedules. This underscores that excessive preference for immediate reward prevents the agent from considering long-term gains necessary to reach distant goals.
- **Importance of exploration rate decay:** Aggressive decay strategies such as $\varepsilon_k = \frac{1}{k}$ and $\frac{1+\log k}{k}$ diminish exploration too early, causing premature convergence to suboptimal policies. Conversely, slower decay schedules such as $\varepsilon_k = \frac{100}{100+k}$ and $\frac{500}{500+k}$ allow for more sustained exploration, resulting in higher success rates and smoother policy paths.
- **Parameter selection rationale:** Among all combinations, $(\gamma = 0.9, \varepsilon_k = \frac{100}{100+k})$ emerged as a strong candidate for Task 2. It achieved full success across all 10 runs with a balance of runtime efficiency and trajectory quality.

These findings support the notion that both γ and ε_k must be tuned jointly to balance exploration, exploitation, and temporal credit assignment in grid-based reinforcement learning problems.

2.4 Transition to Task 2

To further refine the exploration strategy in Q-learning, I analyzed the behavior of several ε_k decay functions with respect to the number of iteration steps k. In the first stage of experiments, four classic forms were considered:

$$\varepsilon_k \in \left\{ \frac{1}{k}, \frac{100}{100+k}, \frac{1+\log k}{k}, \frac{1+5\log k}{k} \right\}$$

As visualized in Figure 3, these strategies exhibit varying decay speeds. For instance, $\varepsilon_k = \frac{1}{k}$ drops below 0.05 as early as $k = 20$, while $\varepsilon_k = \frac{100}{100+k}$ maintains exploration until $k \approx 1900$. The logarithmic schedules decay moderately, offering a middle ground between early and prolonged exploration.

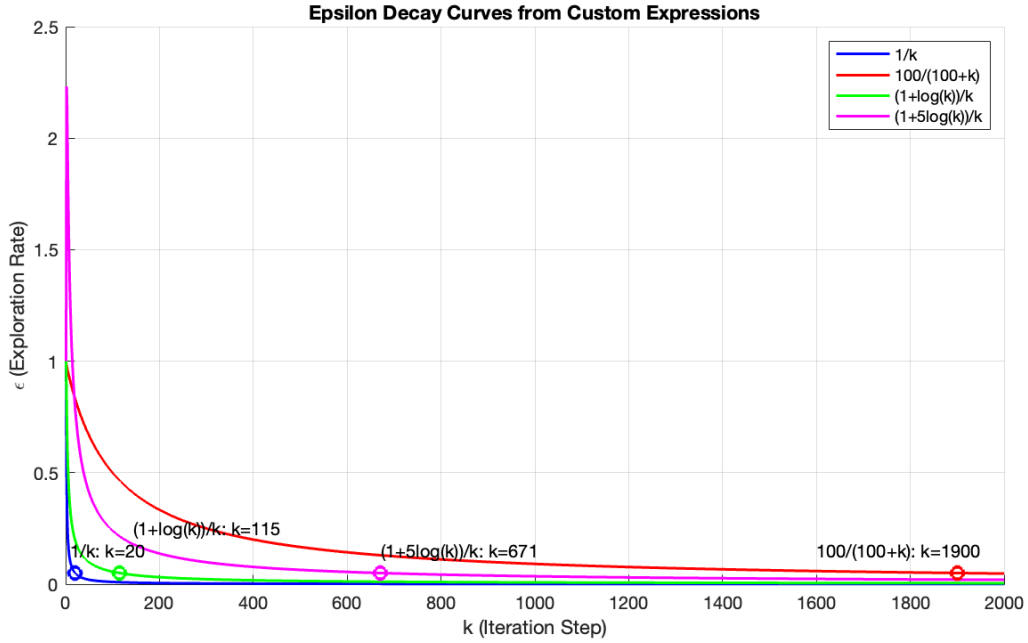

Figure 3: Decay curves of classic exploration schedules

To ensure robustness in more complex or sparse-reward environments, I extended the candidate pool with slower-decaying schedules:

$$\varepsilon_k \in \left\{ \frac{200}{200+k}, \frac{300}{300+k}, \frac{500}{500+k} \right\}$$

These maintain higher exploration levels for longer durations, as shown in Figure 4. For example, $\varepsilon_k = \frac{500}{500+k}$ remains above 0.05 until $k = 9500$.

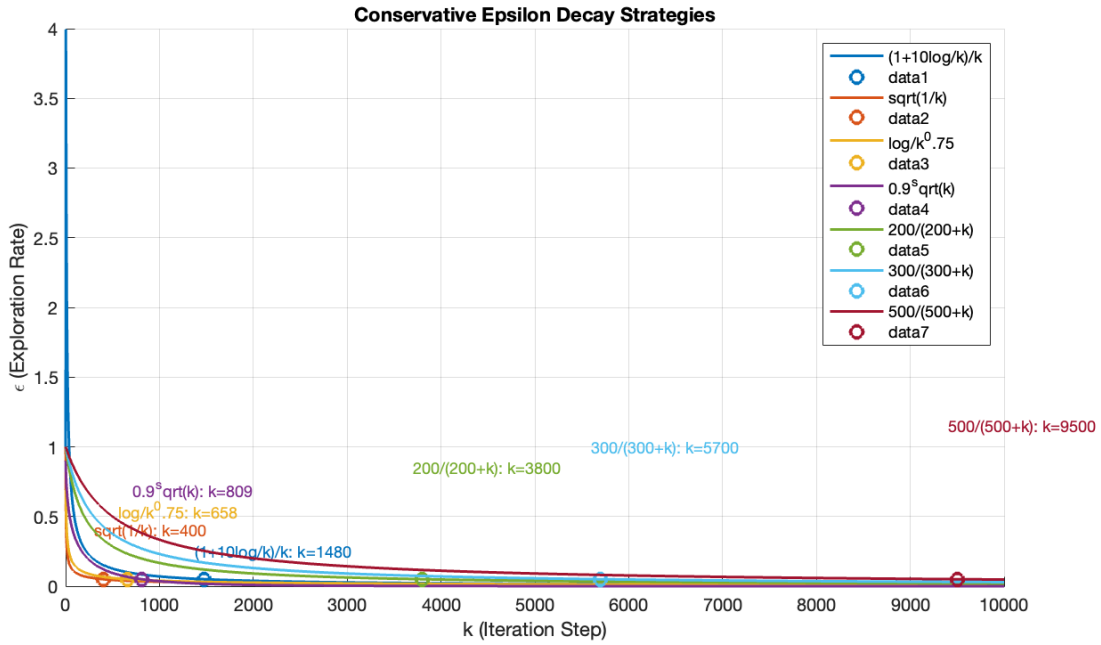

Figure 4: Additional conservative exploration schedules considered for grid search

Based on this analysis, I performed a grid search using the following five ε_k schedules and three discount factors:

$$\varepsilon_k \in \left\{ \frac{100}{100+k}, \frac{1+5\log k}{k}, \frac{200}{200+k}, \frac{300}{300+k}, \frac{500}{500+k} \right\}, \quad \gamma \in \{0.7,\ 0.8,\ 0.9\}$$

Table 2: Grid search results: Q-learning success rates and runtimes under extended parameter settings

ε_k, α_k	No. of goal-reached runs			Execution time (sec.)		
	γ=0.7	γ=0.8	γ=0.9	γ=0.7	γ=0.8	γ=0.9
$\frac{100}{100+k}$	10	10	10	0.1256	0.1528	0.3252
$\frac{1+5\log k}{k}$	10	10	10	0.1763	0.2636	0.4838
$\frac{200}{200+k}$	10	10	10	0.0948	0.1048	0.2625
$\frac{300}{300+k}$	10	10	10	0.0824	0.1042	0.2075
$\frac{500}{500+k}$	10	10	10	**0.0758**	0.1084	0.1267

Final Parameter Selection

All combinations achieved a perfect success rate, indicating that all ε_k schedules allowed sufficient exploration. Therefore, runtime becomes the dominant selection criterion. Among them, the pair $(\gamma = 0.7,\ \varepsilon_k = \frac{500}{500+k})$ recorded the lowest average execution time (0.0758 seconds), making it the most efficient and reliable configuration for Task 2.

3 Task 2

3.1 Results and Policy Evaluation

Based on the results of the grid search conducted in Task 1, I selected the parameter pair $\gamma = 0.7$ and $\varepsilon_k = \frac{500}{500+k}$ for Task 2. I use *task1.mat* to test the validity of the MATLAB script `RL_main.m`. As expected, the agent successfully reached the goal in all 10 runs, confirming the robustness of the selected parameters. The average runtime across successful runs was approximately 0.0645 seconds. Among the 10 executions, the best-performing trajectory was extracted for visualization and evaluation.

Figure 5 shows the optimal policy learned by the agent using red triangle arrows, while Figure 6 illustrates the execution path of the agent under the learned policy. The agent is seen to follow a direct and efficient trajectory toward the goal, demonstrating consistent forward and downward movements, aligning with optimal navigation logic in the grid.

The cumulative discounted reward obtained along this path was 185.4288, indicating efficient exploitation of positive rewards and successful avoidance of negative traps. The optimal path was saved as a column vector *qevalstates* for future evaluation or deployment.

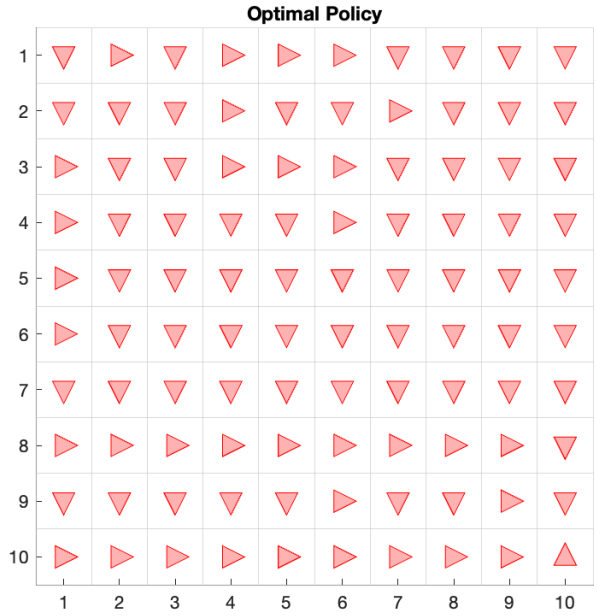

Figure 5: Visualization of the learned optimal policy (gamma = 0.7, $\varepsilon_k = \frac{500}{500+k}$)

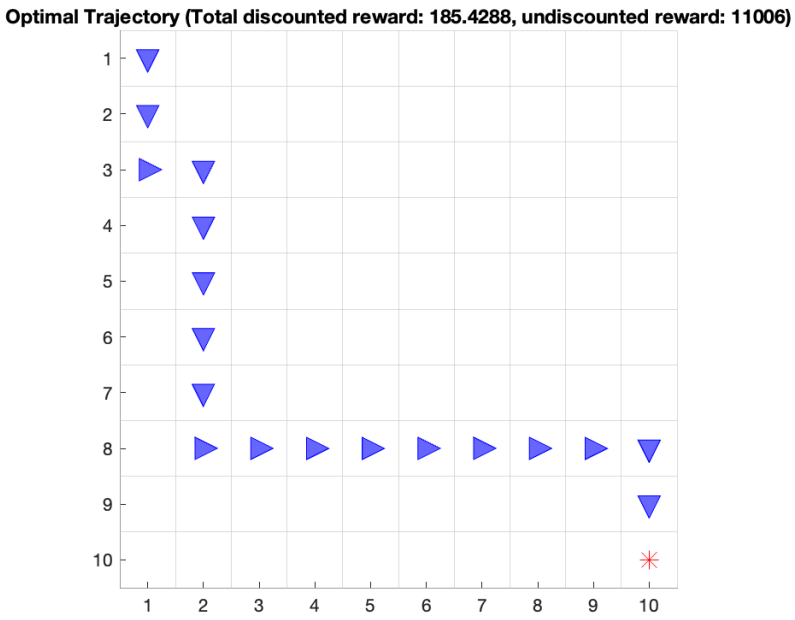

Figure 6: Execution trajectory of the optimal policy (total discounted reward: `185.4288`)

MATLAB command line output is shown as follows:

```
==== Final Q-learning Result ====
Gamma = 0.7000
Epsilon Type = 500/(500+k)
Successful Runs = 10 / 10
Average Time = 0.0645 seconds
Total Discounted Reward = 185.4288
Visualization and optimal policy saved to Task2/
Total Reward: 185.4288
qevalstates:
1
2
3
13
14
15
16
17
18
28
38
48
58
68
78
88
98
99
```

100

3.2 Summary

The Task 2 implementation confirms that the Q-learning algorithm, when configured with $\varepsilon_k = \frac{500}{500+k}$ and $\gamma = 0.7$, can reliably produce an optimal policy within limited runtime and achieve a high total reward. While the final evaluation was conducted on the known environment from `task1.mat`, the algorithm structure was designed to be compatible with the assessment conditions specified for `qeval.mat`.

Specifically, the output policy was stored as a column vector `qevalstates`, and visualizations of the optimal path and total discounted reward were generated and saved. These results suggest that the MATLAB implementation is robust and ready to handle external evaluation environments such as the one defined in `qeval.mat`, once it becomes available.

Moving forward, testing the same algorithm on the hidden reward matrix `qevalreward` would provide a more comprehensive validation of its generalization ability.