Q-Learning and Deep Q-Network for Maze Navigation

Lucas Yeykelis
University of Miami, Miami, Florida, USA
lucasyeykelis@miami.edu

Mingzhe Chen
University of Miami, Miami, Florida, USA
mingzhe.chen@miami.edu

***Abstract*—This report presents the implementation of Q-Learning and Deep Q-Network (DQN) algorithms for maze navigation. We implement epsilon-greedy action selection, Q-table TD updates, DQN network architecture, and expected Q-value calculation. Experimental results show both algorithms successfully navigate the maze, with Q-Learning demonstrating faster convergence for this small-scale problem. Hyperparameter analysis reveals that the discount factor is the most critical for performance.**

I. Action Selection Mechanism

To balance exploration and exploitation, we implemented the epsilon-greedy strategy. At each time step, the agent selects a random action with probability ϵ (exploration), or the action with the highest Q-value with probability $1-\epsilon$ (exploitation):

$$a_t = \begin{cases} \text{random action from } \mathcal{A} & \text{with probability } \epsilon \\ \arg\max_a Q(s_t, a) & \text{with probability } 1-\epsilon \end{cases} \quad (1)$$

The implementation mirrors the action selection provided in `QNetwork.step()`. The code already shows exponential epsilon decay ($\epsilon_{t+1} = \max(0.01, 0.995 \times \epsilon_t)$) to shift from exploration-heavy early training to exploitation-focused later training.

The function `np.random.random()` generates a uniform random number in $[0, 1)$. If this value is less than ϵ, it selects a random action; otherwise, it selects the action with the maximum Q-value using `argmax`.

II. Algorithm Design

A. Q-Learning: Q-Table Update Rule

We implemented the Q-table update in `QTable.update_q_table()` using the temporal difference (TD) learning rule. Given the transition (s, a, r, s'), we update:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s,a) \right] \quad (2)$$

The framework provides the TD target calculation (`predict` and `target`). We completed the update rule:

```
self.q_table[state, action] =
predict + self.learning_rate*(target - predict)
```

This incrementally adjusts the Q-value toward the TD target, where $\alpha = 0.1$ controls the step size and $\gamma = 0.99$ weights future rewards.

B. Deep Q-Network (DQN)

DQN uses a neural network to approximate the Q-function, $Q(s, a; \theta) \approx Q^*(s, a)$, enabling generalization to larger state spaces where tabular methods not sufficient.

1) Network Architecture (Task III): We implemented the neural network in `QNetwork.__init__()` as a fully connected network:

```
self.network = nn.Sequential(
  nn.Linear(config.MAZE_SIZE**2, 64),
  nn.ReLU(),
  nn.Linear(64, 64),
  nn.ReLU(),
  nn.Linear(64, len(self.actions))
)
```

The architecture maps one-hot encoded states (25 inputs for a 5×5 grid) through two hidden layers with ReLU activations to 4 output Q-values (one per action).

2) Expected Q-Value Calculation (Task IV): We implemented the TD target calculation in `QNetwork.learn()`:

```
expected_q_values = reward_mem +
self.discount_factor * next_q_values * (1-done_flags)
```

This computes $Q_{\text{expected}} = r + \gamma \max_{a'} Q_{\text{target}}(s', a') \times (1 - \text{done})$. The $(1 - \text{done})$ term ensures terminal states receive only the immediate reward with no future value.

3) Framework Components: The template provides experience replay (storing transitions in a buffer and sampling mini-batches) and a target network updated periodically to stabilize training.

C. Key Differences Between Q-Learning and DQN

The main difference is that Q-Learning stores exact Q-values in a table, while DQN uses a neural network like a function approximator to estimate Q-values. This leads to several practical differences:

TABLE I
COMPARISON OF Q-LEARNING AND DQN

Aspect	Q-Learning	DQN
Q-function	Explicit lookup table	Function approximator (neural network)
Scalability	Not scalable to large state spaces	Handles large/continuous states
Memory	Must compute Q(s,a) for every state-action pair	Fixed (network parameters)
Update	Direct table assignment with TD	Gradient descent on loss
Data usage	Online (each transition used once)	Experience replay (random batches)
Target value	Same Q-table	Separate target network

Experience replay (DQN) breaks the correlation between consecutive training samples by randomly sampling from stored transitions. The target network provides stable Q-value targets during training, since the main network's weights change with each update.

III. EXPERIMENTAL RESULTS

A. Q-Learning Results

Figure 1 shows the learning curve for the Q-Learning agent over 500 episodes with default settings.

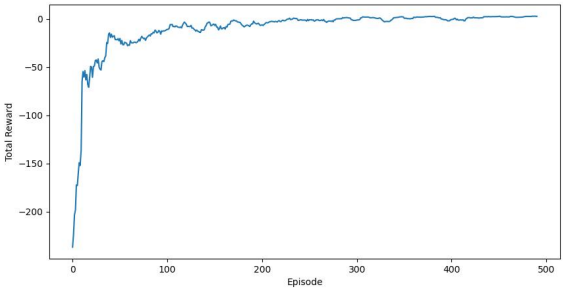

Fig. 1. Q-Learning training curve showing total reward per episode. The agent shows rapid initial improvement from approximately -230 to near-zero rewards within 100 episodes, with stable positive performance after episode 200.

The agent starts with rewards around -230 (random exploration), improves rapidly in the first 100 episodes, and converges to stable positive rewards (∼+2.5 to +3.0) by episode 200.

B. DQN Results

Figure 2 shows the learning curve for the DQN agent over 500 episodes with default settings.

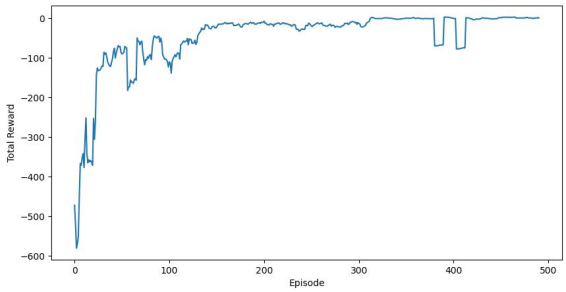

Fig. 2. DQN training curve showing total reward per episode. The agent shows more unstable learning compared to Q-Learning, with performance dips around episodes 80-100 and 400-420.

DQN starts with lower rewards (∼-580) due to random network initialization and exhibits more volatility throughout training with periodic dips. The agent converges to positive rewards by episode 300, although it is less consistent than Q-Learning.

C. Comparison of Q-Learning and DQN

TABLE II
PERFORMANCE COMPARISON

Metric	Q-Learning	DQN
Initial reward	∼-230	∼-580
Episodes to positive reward	∼150	∼200
Stable convergence episode	∼200	∼300
Final avg. reward (last 20 ep.)	∼2.5	∼1.5
Learning stability	High	Moderate

For this small-scale problem, Q-Learning outperforms DQN in convergence speed, stability, and final performance. This is expected since tabular Q-Learning is well-suited for small, discrete state spaces (like this maze simulation), while DQN's neural network approximation introduces complexity which helps for larger state spaces.

D. Successful Navigation

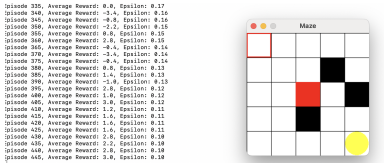

Fig. 3. Trained agent successfully navigating the maze. The red square represents the agent, black squares are walls, and the yellow circle is the goal/terminal.

IV. HYPERPARAMETER ANALYSIS

This section presents the analysis of how different hyperparameter configurations affect learning performance.

A. Discount Factor (γ)

The discount factor determines how much the agent values future rewards relative to immediate rewards.

TABLE III
EFFECT OF DISCOUNT FACTOR ON Q-LEARNING PERFORMANCE

γ	Convergence Ep.	Observation
0.90	$\sim$200	Very unstable, multiple deep dips to -400/-600 throughout training
0.95	$\sim$170	Smoother learning curve, fewer performance drops
0.99	$\sim$130	Good balance (baseline)

Analysis: A high discount factor ($\gamma = 0.99$) is important for maze navigation because the agent must plan multiple steps ahead to reach the goal. Lower values cause the agent to prioritize avoiding immediate negative step rewards (-1 per step) over reaching the distant positive goal reward (+10). With $\gamma = 0.90$, future rewards are discounted more heavily: a reward 8 steps away is worth only $0.9^8 \approx 0.43$ of its face value, compared to $0.99^8 \approx 0.92$ with $\gamma = 0.99$. This explains the slower convergence and lower final rewards observed with lower discount factors.

B. Initial Exploration Rate (ϵ)

The initial epsilon value determines how much the agent explores at the beginning of training.

TABLE IV
EFFECT OF INITIAL EPSILON ON Q-LEARNING PERFORMANCE

ϵ_0	Convergence Ep.	Observation
0.50	$\sim$240	Very unstable, frequent dips to -300/-400 throughout
0.90	$\sim$200	Balanced exploration (baseline)
1.00	$\sim$230	Extreme dip to -1200 at ep. 100, delayed convergence

Analysis: Starting with $\epsilon = 0.9$ provides the best balance between exploration and exploitation. With $\epsilon = 0.50$, the agent exploits too early before adequately exploring the state space, leading to highly unstable learning with frequent performance drops as the agent gets stuck in suboptimal paths. With $\epsilon = 1.0$ pure random exploration fails to reinforce successful trajectories, resulting in poorly calibrated Q-values. When the agent begins exploiting these unreliable estimates (around episode 100), it confidently follows these less optimal paths, causing the extreme performance dip (to -1200) before eventually recovering. The baseline $\epsilon = 0.9$ avoids both extremes, enabling stable convergence.

C. Learning Rate (α) - DQN

The learning rate controls how quickly the neural network updates its weights.

TABLE V
EFFECT OF LEARNING RATE ON DQN PERFORMANCE

Learning Rate	Convergence Ep.	Observation
10^{-3}	$\sim$160	Fast convergence, late small dip at ep. 430
10^{-4}	$\sim$150	Stable (baseline)
10^{-5}	$\sim$220	Slower learning, small dip at ep. 370

Analysis: The learning rate of 10^{-4} provides the best convergence for DQN, reaching stable performance fastest ($\sim$150 episodes) with no significant instability. With 10^{-3}, the network updates more aggressively but actually converges slower ($\sim$160 episodes) and exhibits a dip around episode 430, suggesting the large weight updates cause the network to occasionally overshoot and unlearn good policies. With 10^{-5}, updates are too conservative, resulting in the slowest convergence ($\sim$220 episodes) and slower recovery from random perturbations when they occur. The baseline 10^{-4} offers the best balance of learning speed and stability.

D. Batch Size - DQN

The batch size determines how many experiences are sampled from the replay buffer for each training update.

TABLE VI
EFFECT OF BATCH SIZE ON DQN PERFORMANCE

Batch Size	Convergence Ep.	Observation
32	$\sim$140	Fast convergence, stable afterwards
64	$\sim$150	Balanced (baseline)
128	$\sim$140	Smooth, very stable learning

Analysis: All three batch sizes performed similarly, converging within $\sim$140-150 episodes. Batch sizes 32 and 128 converged slightly faster ($\sim$140 episodes) than the baseline 64 ($\sim$150 episodes), but the difference is minimal. This suggests that for this small maze problem, batch size is not a critical hyperparameter. The similar performance across all configurations indicates that the default batch size of 64 is a reasonable choice, although smaller or larger values work equally well.

E. Summary of Hyperparameter Impact

Based on our experiments, the relative importance of hyperparameters for this maze navigation task is:

1) **Discount Factor (γ)**: Important - must be high (0.99) for long-horizon planning
2) **Learning Rate (α)**: Important - affects convergence speed and stability
3) **Initial Epsilon (ϵ)**: Moderate - affects early exploration quality
4) **Batch Size**: Low - minimal impact for this project

V. Conclusion

This report demonstrated the implementation of Q-Learning and DQN algorithms for maze navigation. Q-Learning outperformed DQN for this small-scale problem (25 states) in convergence speed, stability, and final performance, which is expected since tabular methods are well-suited for discrete, small state spaces. The discount factor ($\gamma = 0.99$) proved most critical, as the agent must plan multiple steps ahead to reach the goal. Both algorithms successfully learned to navigate the maze using the epsilon-greedy exploration strategy.