MAZE NAVIGATION WITH REINFORCEMENT LEARNING

ABSTRACT

Our paper examines the performance and behaviour of multiple reinforcement learning methods (SARSA, Q-learning, Deep Q-Network), in a maze navigation setting. The mazes are generated using a Kruskal-based algorithm, which give it the ability to fine-tune the maze's complexity by adding extra passages. We then observed the behavioural patterns of Q-learning agents and SARSA agents in these mazes. It was shown that Q-learning agents outperformed SARSA agents in almost all different configurations and their differences are particularly significant in the initial phase with all 7x7 and 9x9 mazes and with 5x5 mazes with complexity level of 2. We have also proved that SARSA agents with learning rate $\alpha = 0.3$, discount factor $\gamma = 0.95$ and exploration rates $\epsilon = 0.1$ provided the best results in the initial exploration phases although still underperforming Q-learning agent. We have consequently concluded that the superior results from Q-learning agent originated from its risk taking nature. Finally we extended our research with preliminary experiments using DQN agents in mazes of different sizes and demonstrated the importance of exploration decay rate in the DQN agent's performances.

1 INTRODUCTION

Reinforcement learning is an approach to machine learning in which a learning agent learns how to map the states of its environment to actions to maximize a reward signal (1). Not being told which actions to take, agent takes actions independently by interacting with the environment and learns through the consequences of its actions and based on the reward it obtains. During the same time, the environment provides certain rewards based on the actions the agent has taken, which in turn affects the actions the learning agent will take in the next round of interaction. There are a variety of algorithms used for reinforcement learning, among these q-learning is one of the most widely used methods. Q-learning algorithms are a type of reinforcement learning algorithm that uses a q-function that measures the expected reward assuming that the agent makes the optimal decisions(2). These algorithms are often used because they are easy to understand, have great versatility because they do not require a model of their environment, and can be adapted to solve larger or more complex problems (3).

Amongst different tasks that could be solved by reinforcement learning, navigation and path finding problems have always been of great research interest. Such problems are often redefined as various maze navigation problems: in (4)Zhang et al. proposed a curiosity method which can reduce repeated exploration of areas, allowing for an increased convergence rate compared to an q-learning algorithm without curiosity; in (5) Wang et al. presented an incremental learning algorithm for changing environments that detects and subsequently explores the changed parts of the environment to allow greater adaptability and efficiency in dynamic environments; in (6) Mannucci et al. proposed a maze exploration strategy that provides an agent with a minimum level of performance by means of sequential tasks and off-line learning to prevent collisions with the environment. Maze navigation also presents challenges similar to many other reinforcement learning tasks such as exploration versus exploitation dilemma and sequential decision-making, where each action has an effect on subsequent states. Reducing path finding problems in complicated real life scenarios also allows us to re-create the challenges while keeping them easy to interpret and directly observe which strategies the algorithms employ.

In this paper, our main research focus is to examine the differences between Q-learning agent and the SARSA (State–action–reward–state–action) agent with varying maze sizes and complexities. As a bonus, we would also like to experiment with combining Q-learning and neural network (Deep-Q-Network) and briefly investigate the effect of different hyperparameters on the DQN agent's performances in different mazes. The detailed research questions are summarized in the next section.

2 Research Questions

- How do different Q-value update rules (SARSA and Q learning) affect the agent's behaviour in maze navigation?
- How does the maze complexity affect the choice of hyperparameters for each agent?
- Bonus: How does the hyperparameter affect a deep-Q-network agent's performance in mazes of different sizes?

3 Experimental Methods

The conducted experiments can be divided into two different parts. First, two temporal-difference (TD) reinforcement learning methods were implemented: an on-policy (SARSA) algorithm and an off-policy (Q-learning) algorithm. We then extended our research to deep reinforcement learning experimenting the behaviour of the Deep-Q-Network (DQN) agents. During the experiments, the learning of the agents was achieved by their interactions with a separate maze environment. First we would like to introduce the maze environment to lay out the foundation of the experimental configurations.

3.1 Maze Environment

3.1.1 Maze Configuration

First we manually constructed two mazes with size of 5×5 and 9×9) (shown in **Figure1**) for prototyping purposes. There was no specification on complexity, the aim was to observe agents' behaviour in different maze settings. We then implemented a maze generation algorithm based on Kruskal's algorithm for environments with more controlled complexity. As maze generation is not the main focus of our research, the detail of the implementations will not be explained. However, we would like to briefly introduce the mechanism of our maze generation algorithm and how the complexity has been defined in our research.

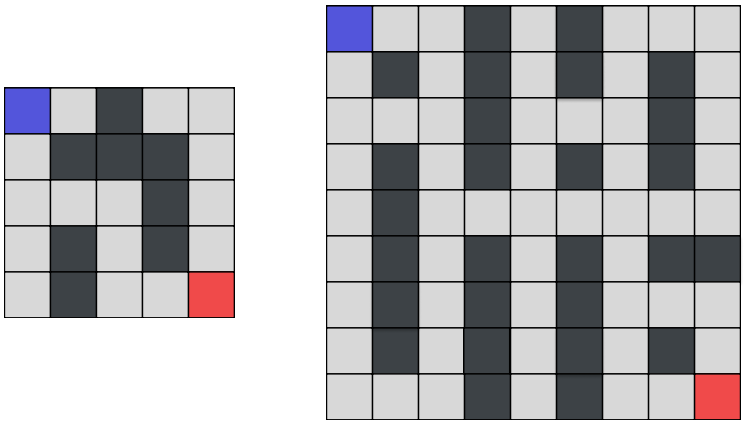

Figure 1: Starter 5x5 and 9x9 maze configuration for prototyping.

Kruskal's algorithm(7), initially designed as a greedy algorithm for finding a minimum spanning tree (MST) in a graph where the MST must connect all the nodes with the least possible amount of

total edge weights while avoiding cycles. To generate a maze in a similar fashion, one could assign the nodes as the cells of the maze, the edges as the potential walls between adjacent cells while treating the edge weight equally. Kruskal's algorithm guarantees the generation of a 'perfect' maze that allows only one solution for the maze problem. One could then adjust the complexity of the maze by adding extra passages that allows alternative routes (8). In our experiments we considered that the less the amount of extra passages that one maze possessed, the higher the complexity of the maze, as the lack of extra passages potentially increases the amount of dead ends, especially for the larger mazes. However one could argue that mazes with extra passages could actually be more difficult for the agents to learn as the paths found by the agents in perfect mazes are always the optimal paths as long as they could reach the goal. The complexity problem was essentially translated into a competition between loops and dead ends. Therefore as a supplementary question for our first two research questions, we would also like to examine how our simple definition of complexity reflect on the agents' learning processes.

3.1.2 Environment Agent

To enable the reward mechanism between the agents and the maze environment, a separate environment agent was implemented. First, our generated mazes were converted to a matrix where the walls are denoted by 1 and the rest is denoted by 0. The environment agent first read different maze configurations and then provided a set of reward mechanisms guiding the agent's behaviour in the maze environment:

- **Step penalty**: for each action taken that does not lead to the goal state, there is a default step penalty of -0.01;
- **Goal Reaching Reward**: +100 would be rewarded if the agent reaches the goal state;
- **Collision Penalty**: If the move leads the agent into a wall even if the position was not updated a penalty of -5 would be induced;
- **(Optional) Proximity reward**: For some of the experimental settings, a proximity reward was given for getting closer to the goal. The value of proximity reward is calculated by scaling the differences in Euclidean distance before and after the step by 0.1.

3.2 Reinforcement Learning Agents

Temporal difference (TD) learning falls into the big family of tabular solution method for tackling reinforcement learning problems(1). The behaviour of the TD learning agents are based on the principles of Markov Decision Process (MDP) denoted by *S* the state space, *A* the action space, *A* the reward function and *P* the state transition function (5). The objective of the agents is to obtain an estimation of the optimal policy governing the state-action pairs that maximizes the total reward. In our case, to reach the goal state with least possible penalties along the way. The temporal difference denotes the difference between the current estimation value and a target estimation that the current estimation should move towards.

TD algorithms combine the ideas of Monte Carlo methods and dynamic programming, learn directly from raw experience without the requirement of a model for their environment, update their estimates based on the experience without a final outcome through bootstrapping, finally save the state-action estimation pairs in a tabular format(1). Both Q-learning and SARSA agents are part of the TD learning methods with slight variances in their approach in finding the optimal policy.

3.2.1 Q-learning

Our simple Q-learning agent is created based on the off-policy TD control algorithm developed by Watkins in 1989 (9). With no prior knowledge of the environment, Q-learning agents could achieve optimal policies from delayed rewards. The learned action-value function Q that directly approximates the optimal action-function value function is defined as:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \text{max}_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)] \quad (1)$$

where $Q(s, a)$ represents the current Q value for state s and action a, α is the learning rate that defines how much new information is take to update the old values, r denoting the immediate

reward after taking action a in state s, the significance of future reward is controlled by γ the discount factor and finally $\max_{a'} Q(S_{t+1}, a')$ finds the action that maximise the Q value for the next state hence the best estimated future reward. The Q values for all the states are maintained in a Q-table with dimensions of $|S| \times |A|$ where S is the set of all states and A is the set of all actions. The Q-table is initialised with all 0 values for each entry and is updated with the learnt Q-value every time an state transition takes place. The implementation of the Q-learning Q value updating rules is described in **Algorithm 1**.

Algorithm 1 Q-Learning Q Update

Require: Current state s, action a, reward r, next state s', learning rate α, discount factor γ

$q_{\text{current}} \leftarrow Q(s, a)$

$q'_{\max} \leftarrow \max_{a'} Q(s', a')$ ▷ Best future value estimate

$Q(s, a) \leftarrow q_{\text{current}} + \alpha[r + \gamma q'_{\max} - q_{\text{current}}]$ ▷ TD update

The agent then balances exploration with exploitation through an Epsilon-greedy action selection controlled by ϵ exploration rate (see **Algorithm 2**). For each state S the agent would choose a random action with probability of ϵ or choose an action with $\text{argmax}_a Q(s, a)$ with a probability of $1 - \epsilon$. For our experiments, the actions are discrete and have a fixed size of 4 (up, down, left and right), each cell in the maze represents a distinct state. Therefore for a maze size $n \times n$, the returned Q table would be of a size of $4 \times n \times n$. Q-learning is thus depicted as an 'off-policy' method as the TD target value uses the maximum Q-value of the next time step regardless of the actual policy of the agent.

Algorithm 2 ϵ-Greedy Action Selection

Require: Current state s, Exploration rate ϵ, Q-value table $Q(s, a)$

Generate random $\beta \sim \text{Uniform}(0, 1)$

if $\beta < \epsilon$ **then**

  $a \leftarrow \text{Random}(0, 1, 2, 3)$ ▷ Exploration phase

else

  $a^* \leftarrow \text{argmax}_a Q(s, a)$ ▷ Exploitation phase

  $a \leftarrow a^*$

end if

return a

3.2.2 SARSA

SARSA (state-action-reward-state-action) agents (10) learn and act in a similar manner as Q-learning agents with a slight difference in the Q value update defined as:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(S_t, A_t)] \tag{2}$$

The symbolic definitions are the same as what were depicted in Equation 1. It is termed as an 'on-policy' method as the action of the next time step will be chosen when updating the current state-action value, while in Q-learning a_{t+1} is kept for estimation and will not be taken immediately. The reward update policy of the SARSA agent is thus achieved in a similar pattern as **Algorithm 1** with the action chosen directly based on the known policy shown in **Algorithm 3**. A ϵ-Greedy selection like **2**was also applied with the chosen action defined by the policy if the exploitation phase was selected.

Algorithm 3 SARSA-Update

Require: Current state s_t, Current action a_t , Observed reward r_t, Next state s_{t+1}, Next action a_{t+1} (from policy), Learning rate α, Discount factor γ

$q_{\text{current}} \leftarrow Q(s_t, a_t)$ ▷ Current Q-value

$q_{\text{next}} \leftarrow Q(s_{t+1}, a_{t+1})$ ▷ Next Q-value from policy

$\delta \leftarrow r_t + \gamma q_{\text{next}} - q_{\text{current}}$ ▷ TD Error

$Q(s_t, a_t) \leftarrow q_{\text{current}} + \alpha\delta$ ▷ On-policy update

For the next steps we tested out the deep reinforcement learning algorithms inspired by the research conducted by Deepmind in (11). Although the simplicity of our maze environments might not be the best playground for the deep rl networks due to its lack of intermediate rewards, by experimenting with different hyperparameters and comparing the behaviours of Deep Q Network , we wish to construct the fundamental groundwork for future researches utilising deep reinforcement learning with sequential training and dynamic awards.

3.2.3 DEEP-Q-NETWORK

DQN extends Q-learning by using a neural network to approximate the optimal action-value function $Q*(s, a)$. While such approach theoretically converges to $Q*(s, a)$, it struggles in more complicated environments due to the need to store and update Q-values for all state-action pairs. To address this, DQN replaces the Q-table with a deep neural network that approximates $Q(s, a; \theta)$, allowing it to generalize across states. The network is trained by minimizing the loss function using stochastic gradient descent:

$$L_i(\theta) = E_{s,a\sim\rho(\cdot)}\left[(y_i - Q(s, a; \theta))^2\right]$$

where $L_i\theta$ is the loss at step i, ρ_i is the distribution over states and actions in the replay buffer or behavirous policy and y_i is target Q-value computed using a separate target network:

$$y_i = r + \gamma \max_{a'} Q(s', a'; \theta^-)$$

The basic Q-learning algorithm described above is still inefficient in that experiences obtained by trial-and-error are utilized to adjust the networks only once and then thrown away. Therefore the DQN developed in (11) made use of experience replay that recorded the states, actions and rewards received and then was accessed by taking batches out of the memory pool and training the network based on that. This technique preserved the valuable strategies performed many cycles ago but then forgotten due to incompatible learning rate. The batch size is kept constant to avoid instability in the training.

Each iteration of DQN follows a structured process. First, the agent selects an action with the same epsilon-greedy policy used in the TD methods, then executes the action, observes the reward and next state, and stores this experience in the replay buffer. Periodically, a minibatch of experiences is sampled from the buffer, and the network's weights are updated using gradient descent. A separate target network, which is only updated periodically, prevents the Q-values from fluctuating too rapidly. Through this iterative process, DQN is able to learn optimal policies efficiently, even in environments with high-dimensional state spaces. Additionally ϵ decay was also implemented by multiplying the ϵ value with a decay factor smaller than 1 after each episode of exploration to balance out between exploration and exploitation in the later navigation process.

3.3 EXPERIMENTAL PROCEDURE

3.3.1 TD METHODS

To answer our first research question, we first tested out the Q-learning and SARSA agents on single mazes with fixed complexity (extra passages) with maze size 5×5, 7×7 and 9×9, visualize their learned optimal paths and Q-values. We also examined the effect of providing a proximity reward in the maze environment for both of the agents for the given settings.

Then for each maze size, we then generated 100 maze for each complexity level defined by the allowed extra passages in the solution. The number of complexity levels are dependent on the maze size, as the mazes are generated using Kruskal's algorithm, the maximum amount of extra passages that could define a different configuration each time is dependent on the size of maze itself. For each complexity level in each maze size, we then conducted experiments on the generated mazes with the combinations of learning rate $\alpha \in \{0.1, 0.2, 0.3\}$, discount factor $\gamma \in \{0.85, 0.90, 0.95\}$ and exploration rates $\epsilon \in \{0.1, 0.2, 0.3\}$.

Finally for each agent, we compare their performances by visualising their performance heatmaps for different hyperparameter combinations and hyperparameter sensitivity across complexities.

3.3.2 Neural Network Methods

Unfortunately due to time and computational power limitation, we could not conduct a thorough research on the generalisation capabilities of the deep RL agent. However hyperparameter searches for DQN agent on fixed mazes with sizes of 5×5 and 9×9 were conducted and compared. The results will be briefly discussed in the next section and detailed plots will be provided in the appendix.

4 Results and Discussion

4.1 SARSA vs. Q-learning

Starting with learning rate of 0.1, discount factor of 0.9 and epsilon of 0.1, both agents converged rather quickly in 5×5 maze and 9×9 maze in our prototype, however a turn of strategy has been noticed in the 9×9 maze. As depicted in (1), Q-learning agents iterates through the Q-values for the optimal policy while SARSA tends to learn the longer but safer path by taking actions into account. It was then illustrated by a 'Cliff Walking' problem in which an environment with a single cliff was setup. With -1 reward given for all the empty space and -100 reward given for hitting the cliff, it was seen that SARSA agent would take the safer but longer path while Q-learning agent would walk next to the cliff for the most optimal path. Such behaviour was also observed in our experiments (shown as **Figure2**). The Q-values and arrows are plotted for all the non-negative and non-zero Q-values saved in the Q-table. Although both agents found the same optimal policy in the end, it could be seen that around one of the single blocks in the environment, Q-learning agent evaluated the Q values around the block and navigated its path towards the goal while SARSA agents extended further to the direction with more empty space. However, the risk taking behaviour of Q-learning agent could potentially lead the agent to walk around in loops more often aiming to find the most optimal policy when there are multiple paths available.

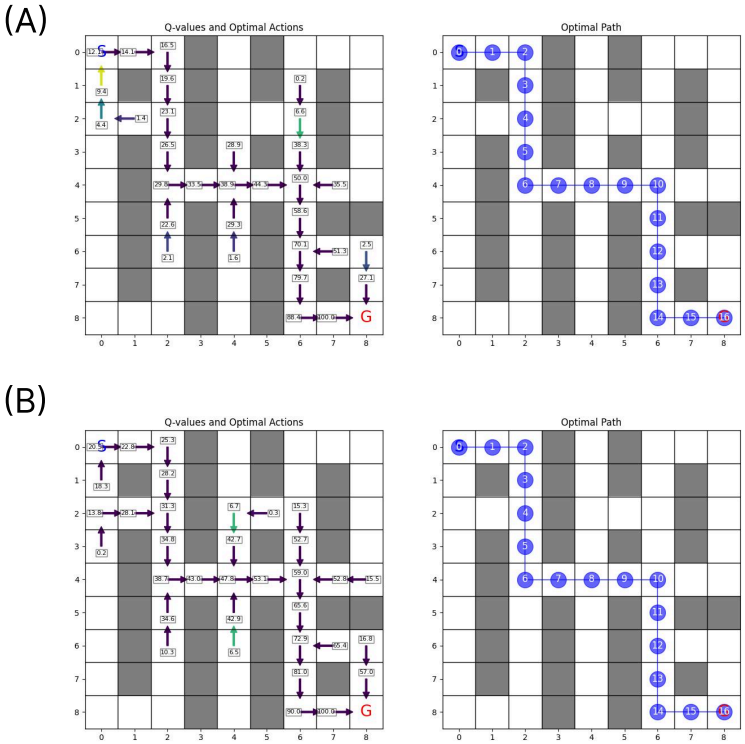

Figure 2: Illustration of 'Cliff Walking' behaviour in maze environments where the final Q values and learnt optimal policy for (A) SARSA agent and (B) Q learning agent are shown.

To examine the effect of such behaviour, we then compared the result of both agents with incremental complexity settings in the maze environments. It could be seen in **Figure 3** that the Q-learning agent's risk taking behaviour did not bring it to a disastrous exploration loop in mazes with lower complexity hence more alternative paths, however SARSA agents, on the other hand, sometimes struggles to converge in the beginning even when exploring mazes with lower complexity.

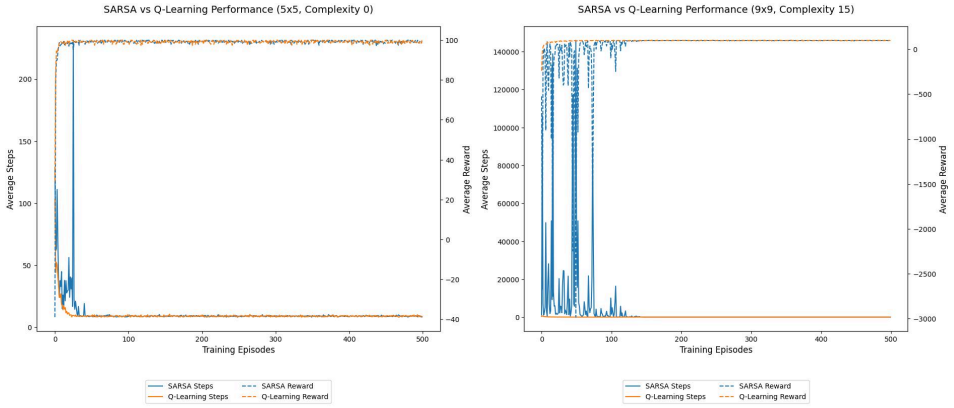

Figure 3: Average steps and reward history for SARSA and Q-learning agents in 100 5x5 mazes with complexity 0 (left) and 100 9x9 mazes with complexity 15 (Right)

To obtain a further understanding of the behaviour of both agents. One 9×9 maze was tested out with a proximity award as introduced in the previous experimental set up. To our surprise, with the presence of a proximity award, the Q-learning agent did not manage to converge at all while the initial instability of the SARSA agent was greatly stabilised (shown in **Figure 4**).

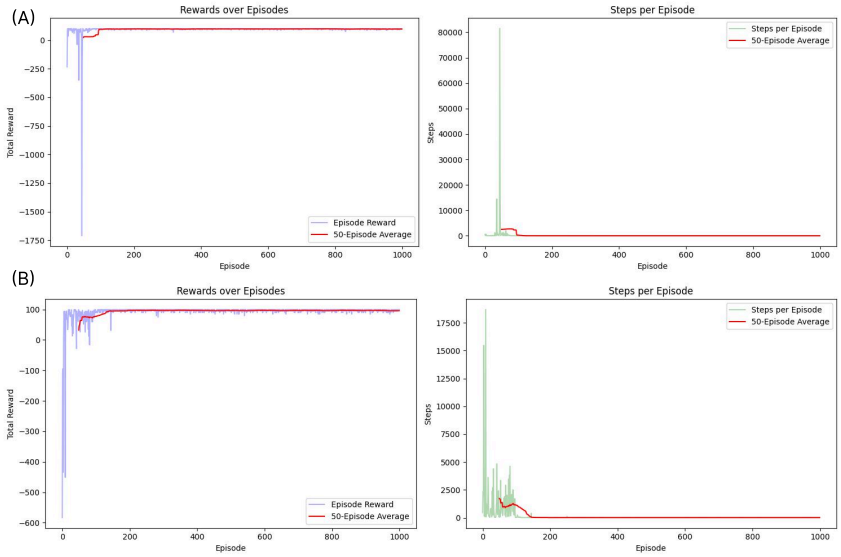

Figure 4: Reward and step history for SARSA agent (A) with proximity reward (B) without proximity reward

Finally before moving on to examining the significance of hyperparameters, we conducted a thorough statistical testing on both agents' average reward history in the first 100 episodes of explorations in 100 mazes per complexity level in maze sizes of 5x5, 7x7 and 9x9. For our generated

mazes, the maximum extra passages that could provide variation in maze configuration was 3, 8 and 16 for 5x5, 7x7 and 16x16 respectively. The complexity level is then determined by the differences between the maximum extra passages and the amount of extra passages the current maze possesses. The the p-values obtained from Mann-Whitney U test(12) provided the decisions on the validity of significant differences between two agents. We then computed the 95% confidence interval using the t-distribution. The results are shown in **Figure 5**) with the shaded area representing 98% confidence intervals and the * depicting the sets where the difference between the reward histories of the two agents are considered significant. The first 100 episodes were chosen as both of the agents tend to converge eventually to an optimal result, therefore we would like to testify our hypothesis from previous observations that the SARSA agents took longer and more steps to explore the maze in the beginning.

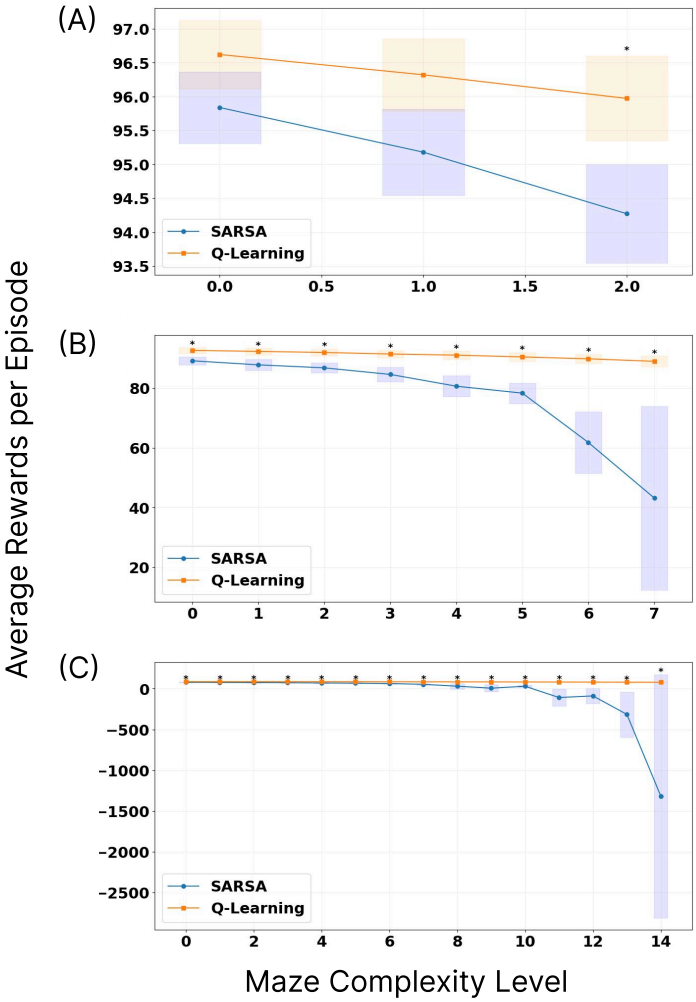

Figure 5: Average steps and reward history for SARSA and Q-learning agents in 100 different mazes per complexity level with sizes of (A) 5x5, (B) 7x7 and (C) 9x9

Finally we would like to investigate how each hyperparameter learning rate α, discount factor γ and exploration rate ϵ reacts to increasing complexity. We computed the average reward for the last 100 episodes (when convergence has most likely occurred) of each maze run with the same experimental settings as before while varying the hyperparameters specified in the experimental procedure. We then plotted the sensitivity results by selecting the same two hyperparaters and varying the third one. The axis was scaled between 0-100 rewards for clear comparison. It could be seen that for the α, γ and ϵ remained quite stable with increasing complexities, the slight decay

observed was most likely due to the longer optimal paths with the increasing complexity. It could also be seen that while the change of α and γ did not result in significant changes in the average reward, a higher exploration rate would lead to lower rewards even during the final episodes when the convergence has achieved. On the other hand the SARSA agent was significantly more sensitive to the change of hyperparameters and maze complexity. The best performing hyperparameters for the SARSA agent turned out to be $\alpha = 0.3$, $\gamma = 0.95$ and $\epsilon = 0.1$. However even with the most optimal hyperparameters, it could be seen that SARSA agent still underperformed Q-learning agent in mazes with higher complexity.

Finally we would like to evaluate our definition of complexity. Although we had suspicions in the beginning that limiting the amount of alternative paths would ultimately result in a more challenging task for the agents to learn and adapt, we had observed that the agents indeed had more difficulty to navigate in a more constraining environment especially for the SARSA agent. However such criteria could alter in mazes generated by a different algorithm or mazes with varying start and goal positions.

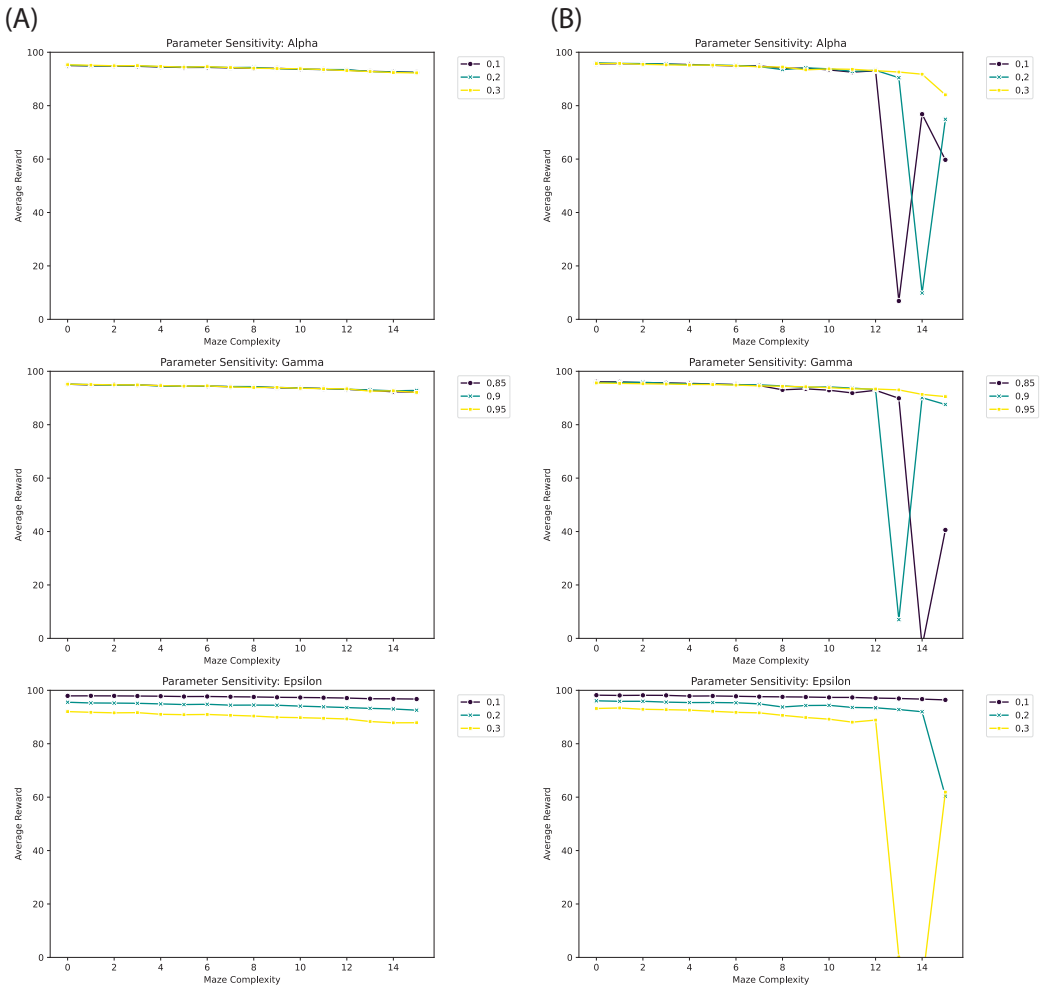

Figure 6: Sensitivity of hyperparameters α, γ and ϵ in 9x9 mazes with increasing complexity for (A) Q-learing and (B) SARSA agents

4.2 DQN HYPERPARAMETER SEARCH

As explained in the experimental procedure, we examined the hyperparameter importance for the DQN agent in mazes of two different sizes. The complexity of the mazes were chosen as the same as the ones depicted in **Figure 3** (5x5 with 0 complexity and 9x9 with 15 complexity). The hyperparameter search plots for each maze could be found in the appendix as this is not part of our main research question. We have concluded that for a simple discrete environment like our maze environments a lower ϵ decay factor of 0.9 (faster decay) was the most optimal regardless of the maze sizes and with ϵ decay factor of 0.9995 the agents would most definitely fail to converge. However, when experimenting with sequential training using only a few episodes from each maze with increasing complexity, the ϵ decay tuning became more difficult as the agents would struggle to gather any new information from the new mazes if the decay factor was set too low. Once again these are the preliminary results we obtained from multiple trials of errors, it is of our personal interests to dive deeper in the development and enhancement of DQN agents. For future updates please refer to our codebase.

5 CONCLUSION AND FUTURE WORKS

In this paper, we first defined the complexity of a maze generated by Kruskal's algorithm according to the amount of extra passages in the maze environment. We then generated multiple difficulty levels for mazes of different sizes and complexities. It was shown that Q-learning agents outperformed SARSA agents in almost all different configurations and their differences are particularly significant in more complex mazes. The significance of exploration rate amongst all the other hyperparameters were shown for both agents and the best hyperparameter combinations were found for SARSA agents and Q-learning agents. We then concluded our research with a hyperparameter search on the DQN agent in different maze sizes indicating the significance of the exploration decay factor.

Our paper hence provided several starting points for future researches including the initial ideas that we gathered such as sequential training or tailored learning curriculum for DQN agent to explore its potentials in more complicated and dynamic environment, implementation of Double DQN agent to investigate its advancement compared to DQN agent in navigation task.

REFERENCES

[1] R. S. Sutton, A. G. Barto, *et al.*, *Reinforcement learning: An introduction*, vol. 1. MIT press Cambridge, 1998.

[2] J. Clifton and E. Laber, "Q-learning: Theory and applications," *Annual Review of Statistics and Its Application*, vol. 7, pp. 279–301, Mar 2020.

[3] B. Jang, M. Kim, G. Harerimana, and J. W. Kim, "Q-learning algorithms: A comprehensive classification and applications," *IEEE Access*, vol. 7, pp. 133653–133667, 2019.

[4] X. Zhang, Y. Liu, D. Hu, and L. Liu, "A maze robot autonomous navigation method based on curiosity and reinforcement learning," 2021. Available: `https://iwaciii2021.bit.edu.cn/docs/2021-12/fcf78b20c59d46e0a8bdd3e3df41d883.pdf`.

[5] Z. Wang, C. Chen, H.-X. Li, D. Dong, and T.-J. Tarn, "Incremental reinforcement learning with prioritized sweeping for dynamic environments," *IEEE/ASME Transactions on Mechatronics*, vol. 24, no. 2, pp. 621–632, 2019.

[6] T. Mannucci and E.-J. van Kampen, "A hierarchical maze navigation algorithm with reinforcement learning and mapping," in *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8, Dec 2016.

[7] J. Kleinberg and E. Tardos, *Algorithm design*. Pearson Education India, 2006.

[8] P. Gabrovšek, "Analysis of maze generating algorithms," *IPSI Transactions on Internet Research*, vol. 15, no. 1, pp. 23–30, 2019.

[9] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, 1992.

[10] D. Zhao, H. Wang, K. Shao, and Y. Zhu, "Deep reinforcement learning with experience replay based on sarsa," in *2016 IEEE symposium series on computational intelligence (SSCI)*, pp. 1–6, IEEE, 2016.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[12] T. P. Hettmansperger and J. W. McKean, *Robust nonparametric statistical methods*. CRC press, 2010.

[13] Tensorflow, "Better performance with tf.function." `https://www.tensorflow.org/guide/function`, 2024. Accessed: 16-03-2025.

6 APPENDIX

6.1 PERFORMANCE BENCHMARKING PYTORCH VS. KERAS

Although the adoption of Keras was suggested for deep learning projects, major performance bottleneck was observed during our experiments. To address such issue, we conducted a standard performance benchmarking, training a deep Q-network agent interacting with the starter 5x5 maze environment shown in Figure 1 using same hyperparameters summarized in Table1, the hyperparameters chosen here were for benchmarking only and were not optimised. With GPU utilised for both frameworks, the whole training process was completed in 1.96 minutes using PyTorch, while Keras took staggering 12.72 minutes even with GPU memory growth and threading enabled. Upon consulting web forums and respective documentations, we had come to a conclusion that the slow training speed could be related to the default eager execution in Tensorflow 2. It is worth noticing that one could improve the performance of Tensorflow/Keras framework with careful tailoring of `@tf.function` (13) to compile functions into graphs for rapid execution of functions that need to be called repetitively (the replay method in our case for example). However the focus of our research is not on the analysis of different deep learning frameworks, therefore PyTorch was chosen for our experiments simply due to its efficiency.

Hyperparameters	Values
learning rate	0.1
gamma	0.99
epsilon	1
epsilon decay	0.9995
minimum epsilon	0.01
batch size	128
replay buffer	20,000
hidden layers	(32, 32)
target network update frequency	100
loss	Huber
optimiser	Adam
number of episodes	500
max steps per episode	500

Table 1: Choice of hyperparameters for performance benchmarking.

6.1.1 Deep Q Network Hyperparameter Search

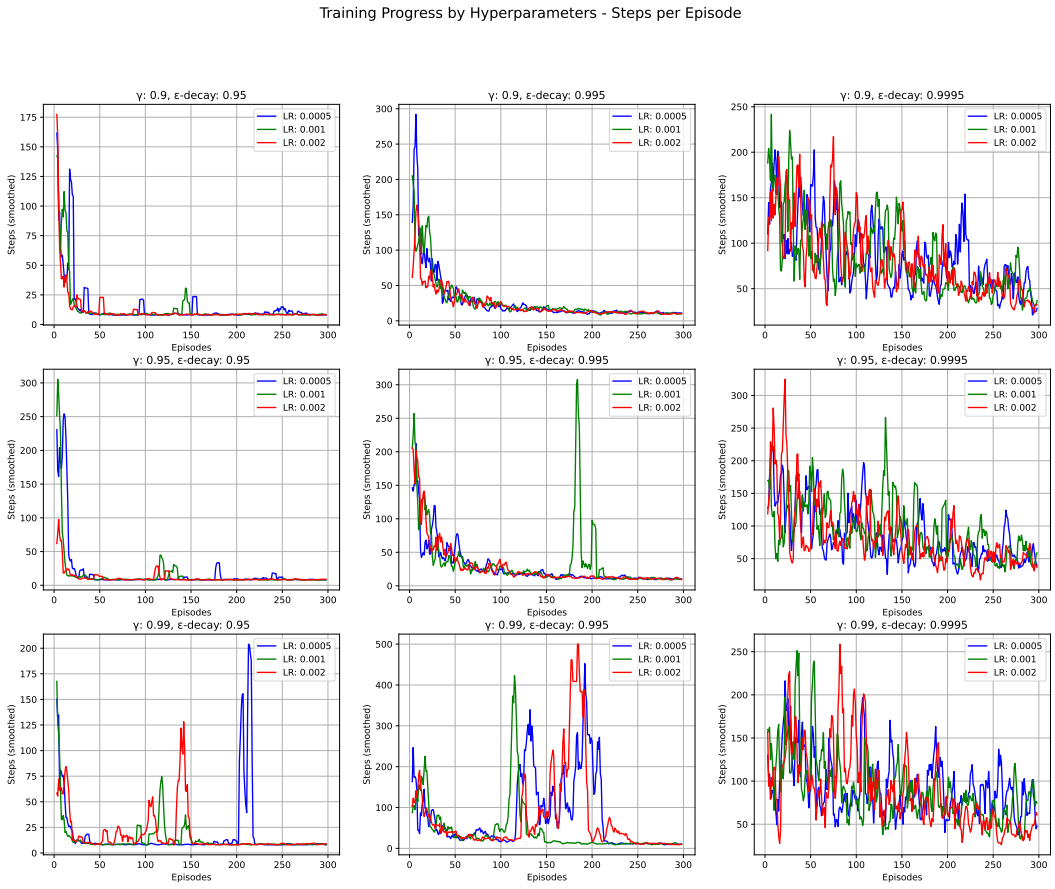

Figure 7: Step history for DQN agents with different learning rate, discount factor and epsilon decay in 5x5 maze environment

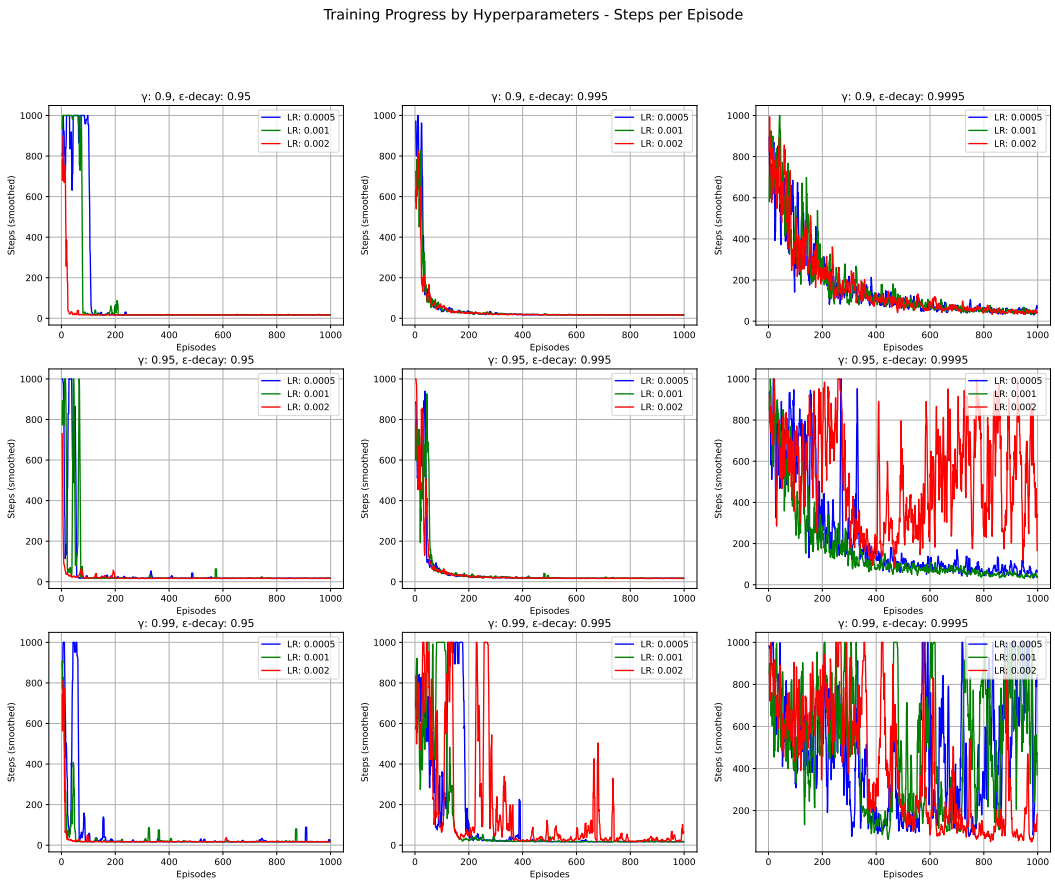

Figure 8: Step history for DQN agents with different learning rate, discount factor and epsilon decay in 9x9 maze environment