Reinforcement Learning - Mid Semester Course Project

Jonathan Erell (ID. 207044447), Etay Todress ID. 316517689)

January 5, 2023

1 Introduction

Reinforcement learning is a popular approach for solving problems in which an agent learns to make decisions that maximize a reward signal. Maze navigation is a classic problem in artificial intelligence, and has been studied extensively in the context of search algorithms. However, RL offers an alternative approach that allows an agent to learn through trial and error, rather than relying on pre-defined rules. This makes RL particularly well-suited to tasks that may be difficult to specify explicitly, or that may change over time. Additionally, RL has the potential to handle partial observability, noise, and stochasticity in a natural way, which are all features of real-world environments. In this paper, we test the performance of several RL methods on the maze navigation task, and compare their results with traditional search algorithms.

2 Solution

2.1 General approach

We approached the algorithms by first understanding their pseudo code and mathematics, then implementing them in code. We then experimented with hyperparameters to understand their effect on the algorithms' performance. This approach allowed us to gain a thorough understanding of the algorithms and effectively optimize them.

2.2 Design

We implemented four reinforcement learning algorithms and provided visualizations to aid in understanding their performance. Each algorithm was implemented as a class with its own functions and the option to choose epsilon soft exploration method. The MDP algorithm trained in a few seconds, while the other algorithms took about a minute. We encountered some technical challenges in ensuring the accuracy of the implementation. The loss functions and

architectures varied depending on the algorithm. We also provided a dataframe with the best hyperparameters for each algorithm.

In addition to the primary algorithms, we also provided several visualizations to aid in understanding the performance of the algorithms. These included graphs showing the number of steps as a function of episodes and the sum of rewards as a function of episodes, as well as a heatmap illustrating the states visited most frequently during training. These visualizations can be helpful in understanding the behavior of the algorithms and identifying areas for further optimization.

3 Experimental results

Our experimental settings involved testing the hyperparameters of each of the four algorithms. For the MDP algorithm, we tested various values for gamma, the initialization of the V values, the initialization of the rewards values, the initialization of the final reward value, and delta. For the SARSA and Q-Learning algorithms, we tested different values for gamma, alpha, epsilon, and the use of hot start or epsilon greedy methods. Finally, for the MC algorithm, we tested different values for gamma, epsilon, and the use of hot start or epsilon greedy methods. Through these experiments, we were able to understand the effect of each hyperparameter on the performance of the algorithms and identify the optimal settings for each.

3.1 Markov Decision Process

Our experimental results showed that the model was not sensitive to the initialization of the V values or the rewards values. All tested values resulted in a converged model with any given gamma. We also observed better results for models with lower delta and higher initialized final reward. These findings suggest that the model is relatively robust to the initialization of these parameters and that the choice of gamma and delta may be more important in determining the performance of the model.

gamma	init_V_value	init_rewards	init_final_reward	delta	AVG_rewards	AVG_steps
0.8	0	0	5	0.001	0.9716	8.1
0.9	-1	default	100	0.010	0.9708	8.3
0.9	1	default	5	0.001	0.9708	8.3
0.9	1	0	100	0.010	0.9708	8.3
1.0	-1	-0.01	1	0.010	0.9708	8.3

Figure 1: Hyper-parameters for MDP

3.2 Q - Learning

Our experiments showed that the model was relatively insensitive to the values of the parameters gamma and alpha, as well as the initialized reward values. However, the model was much more sensitive to the "greedy epsilon" method, achieving better results for higher epsilon values and the best results for models that used the greedy epsilon method. The model was observed to converge at around 200 episodes to an optimal path, as shown by the convergence graph (Figure 4). Overall, the results suggest that the model is particularly sensitive to the greedy epsilon method, and that higher epsilon values may be preferred in this case.

gamma	alpha	epsilon	Greedy Epsilon	Hot Start	Init Maze Reward	Final State Reward	AVG_rewards	AVG_steps
0.9	0.1	0.10	True	False	-0.0001	default	0.9861	32.19
0.9	0.1	0.10	True	False	default	default	0.9860	32.43
0.9	0.1	0.05	True	False	default	5	0.9860	32.57
0.9	0.1	0.05	False	False	default	default	0.9860	32.57
0.8	0.1	0.05	False	False	default	default	0.9860	32.61

Figure 2: Hyper-parameters for Q - Learning

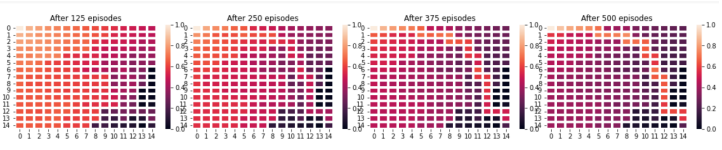


Figure 3: Most Visited States / No. of Episodes for Q - Learning

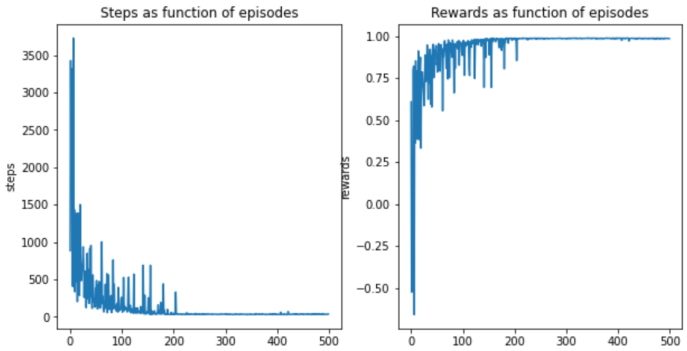


Figure 4: Total Rewards and No. of steps / No. of Episodes for Q - Learning

3.3 SARSA

According to our findings, the SARSA algorithm appears to be less "exploration-wise" than some other algorithms. In particular, it seems to be less reactive to greedy epsilon methods, but more reactive to the initialized reward values for the normal states and the final one. The results also suggest that lower gamma values may be preferred for the SARSA algorithm, as they resulted in better performance. This may be because lower gamma values give less weight to "further" steps in the episode, which may be related to explorations. Overall, these results suggest that the SARSA algorithm may be more sensitive to the initialization of reward values and the value of gamma, and less sensitive to the use of greedy epsilon methods.

gamma	alpha	epsilon	Greedy Epsilon	Hot Start	Init Maze Reward	Final State Reward	AVG_rewards	AVG_steps
0.8	0.1	0.15	False	False	default	5	0.9863	31.73
0.8	0.2	0.05	True	False	-0.0001	5	0.9862	31.98
0.8	0.2	0.05	False	False	default	5	0.9861	32.17
0.9	0.1	0.05	False	False	-0.0001	default	0.9861	32.26
0.9	0.2	0.05	True	False	-0.0001	default	0.9861	32.26

Figure 5: Hyper-parameters for SARSA

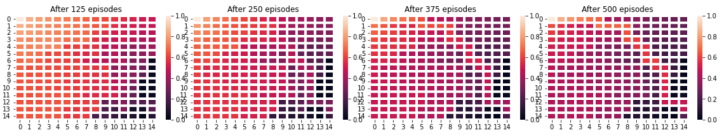


Figure 6: Most Visited States / No. of Episodes for SARSA

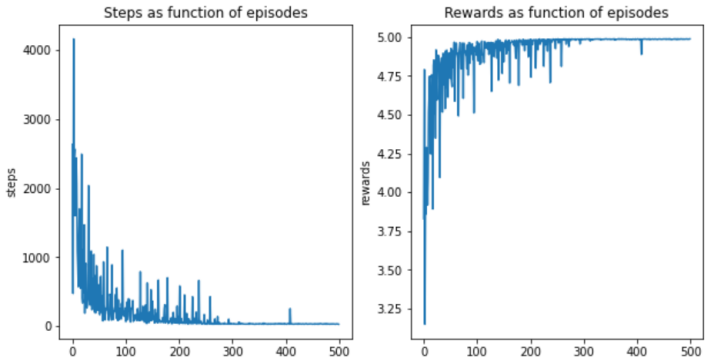


Figure 7: Total Rewards and No. of steps / No. of Episodes for SARSA

3.4 Monte Carlo

The Monte Carlo algorithm appears to achieve better results when exploring and when the reward value of the final state is initialized as a larger number. This is consistent with the nature of the Monte Carlo algorithm, which learns its policy by first exploring till reaching the final state and then using the resulting experience to learn. This can be seen in the heatmap (Figure 8) and convergence graphs (Figure 9), which show that the model initially walks around a bit before taking some time to converge, in contrast to other algorithms which converge more slowly over time. Overall, these results suggest that the Monte Carlo algorithm may be particularly well-suited to tasks that involve exploration and can benefit from initializing the reward value of the final state as a larger number.

gamma	epsilon	Greedy Epsilon	Hot Start	Init Maze Rewards	Init Final Reward	AVG_rewards	AVG_steps
0.90	0.05	True	False	default	5	0.9858	32.91
0.90	0.15	False	False	default	default	0.9846	35.62
0.90	0.15	False	False	default	5	0.9846	35.73
0.99	0.10	True	False	-0.0001	5	0.9845	35.80
0.90	0.10	False	False	-0.0001	5	0.9841	36.68

Figure 8: Hyper-parameters for MC

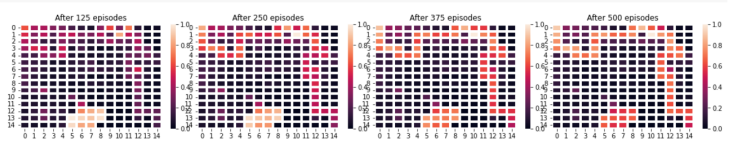


Figure 9: Most Visited States / No. of Episodes for MC

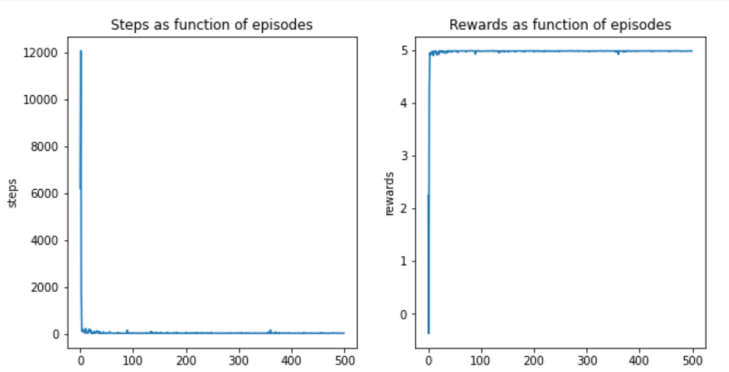


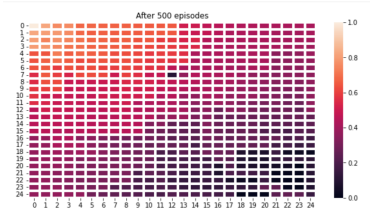
Figure 10: Total Rewards and No. of steps / No. of Episodes for MC

3.5 25x25 Maze

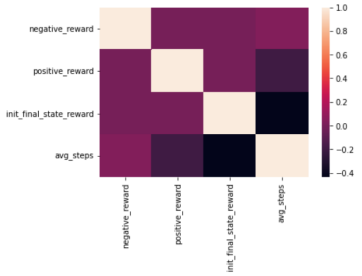
For the last mission, we chose the SARSA algorithm because it tends to learn less through exploration, which was important given the large size of the environment and the restriction of 500 episodes. The SARSA algorithm was observed to learn the environment well, with the model starting to converge, and tends to do so with around 100 more episodes (Figure 11.d). We also found that there is a correlation between the number of steps required to solve the maze and the initialized value of the final state (Figure 11.c), with larger values resulting in fewer average steps. Using our best parameters, we were able to solve the maze in around 172 steps on average across 100 runs. Overall, these results suggest that the SARSA algorithm is well-suited to this task and can effectively learn the environment with a limited number of episodes.

negative_reward	negative_state	positive_reward	positive_state	init_maze_reward	init_final_state_reward	avg_steps
-0.00020	(3, 7)	-0.0009	(12, 7)	default	1.0	170.81
-0.00025	(13, 24)	-0.0001	(15, 16)	default	1.0	174.48
-0.00020	(5, 20)	-0.0009	(12, 7)	default	2.0	176.50
-0.00030	(3, 7)	-0.0009	(12, 7)	default	1.0	176.60
-0.00020	(3, 7)	-0.0005	(12, 7)	default	1.0	178.11

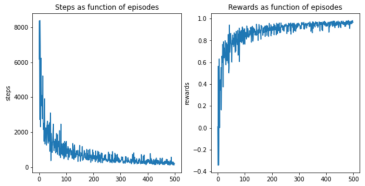
(a) Hyper-parameters for custom SARSA



(b) Most Visited States / No. of Episodes for custom SARSA



(c) Correlation between parameters for custom SARSA



(d) Total Rewards and No. of steps / No. of Episodes for custom SARSA

Figure 11: Visualization for both the learning progress, and parameters correlation

4 Discussion

In this experiment, we implemented four algorithms: MDP, SARSA, Q learning, and Monte Carlo. We found that the MDP algorithm converged the fastest and was relatively insensitive to its parameters, while the other algorithms were more sensitive to their parameters and could potentially diverge with small changes to certain parameters. We also observed differences in the approaches taken by the three latter algorithms, with the Monte Carlo algorithm starting slowly but converging the fastest and tending to explore, and the Q learning algorithm exploring a lot but converging slowly. Both the Monte Carlo and Q learning algorithms achieved good results, solving the maze in an average of around 32 steps. However, the SARSA algorithm outperformed both of these algorithms, achieving an average of 31 steps to solve the maze. Overall, these results suggest that the SARSA algorithm may be particularly well-suited to missions with less time for exploration, especially in large environments.

Note - your project will be evaluated for aspects, including the technique you selected, the rational of the experiments you decided to run, the insights you learned from this process and more. Remember, for the purpose of this course, the process that you demonstrate is very important.

5 Code

Link to google colab's notebook:
Code's Link