Reinforcement Learning Course Work 1

Jamie Shing Him Ho

October 31, 2023

Contents

1 Dynamic Programming **1**
1.1 Q1.1 1
1.2 Q1.2 1
1.3 Q1.3 1

2 Monte-Carlo Reinforcement Learning **2**
2.1 Q2.1 2
2.2 Q2.2 3
2.3 Q2.3 3

3 Temporal Difference Reinforcement Learning **4**
3.1 Q3.1 4
3.2 Q3.2 4
3.3 Q3.3 4

1 Dynamic Programming

1.1 Q1.1

- I decided to use policy iteration for solving the Maze environment using Dynamic Programming. After running a series of tests, I found that policy iteration was the most suitable approach for our problem because it takes fewer epochs. Policy iteration took around 1 second and 4 epochs, whereas value iteration takes around 2 seconds and 25 epochs. In the code, we use a fixed threshold of epsilon equal to 1e-6 as the convergence criterion for policy evaluation to get a small positive value, ensuring that the value function converges to an optimal solution when it is a smaller positive value than this threshold.

Synchronous backups are used instead of asynchronous backups because the maze only has a small state space, which consists of only 98 possible states, 120 total blocks but 32 states belong to the wall and targets. Synchronous backups are suitable for our problem, there is no need to use asynchronous backups in this context, as it is already sufficient for finding the optimal policy with known prior information of state space.

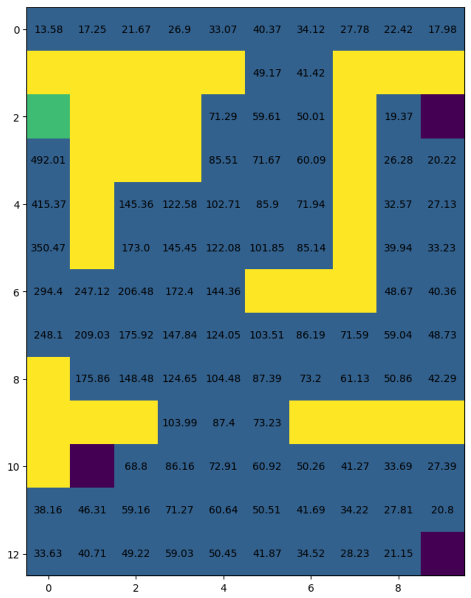
The assumptions made are: firstly, The probability p is set as p = 0.8 + 0.02 $\times$ (9 - y), the discount factor γ is set as γ = 0.8 + 0.02 $\times$ y, the reward state is Ri with i = z mod 4, where y=3, z=0.

1.2 Q1.2

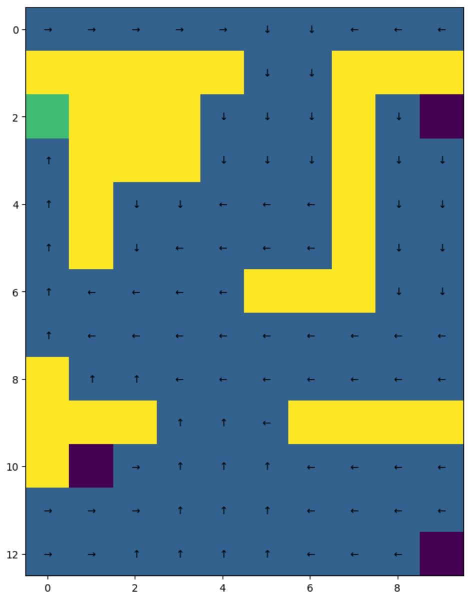
-

1.3 Q1.3

- By varying the p and γ, which is the changing probability of moving in the desired direction p, where the undesired direction has the probability of 1-p/3, and changing the discount factor, we can observe different changes in the optimal value function and the optimal policy.



(a) DP Agent Optimal Value Function



(b) DP Agent Optimal Policy

Figure 1: DP Optimal Policy and Optimal Value Function

In Figure 2a, when p is very low, p¡0.25, the agent has a higher chance of moving in undesired directions. it is moving backward on the graph from the target to the starting point. The optimal policy diverges, causing the agent to move away from the target state. It moves randomly, and the undesired direction is larger than the probability of moving to the desired direction.

In Figure 2b, when p=0.25, all the available actions have equal probabilities, the probability to the target equals undesired directions, and the agent explores the maze randomly, as there is no preference for the action to make progress toward the target.

In Figure 2c, when p is relatively high, p¿0.25, the agent has a higher chance to move to the desired direction. The optimal policy is more likely to guide the agent toward the target, as it has a higher possibility than the undesired direction, which takes the steps that maximize Q(s, a).

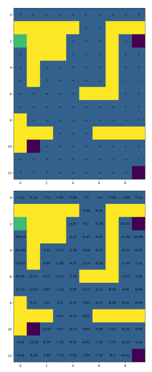
Varying γ, in Figure 2d, the γ¡0.5 makes the agent focus on the immediate rewards rather than the future rewards. This can be seen as the value far away from the goal will have similar values close to -1.33, but the values closer to the goal grow exponentially to 466. This leads to the agent finding it harder to find the optimal policy from the starting points which are far away from the target, because the values in that area are all -1.33.

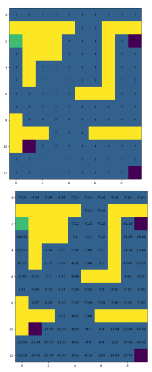
In Figure 2e, the γ¿ 0.5, the agent focuses on the future rewards rather than the immediate rewards. The optimal policy can be found at the starting point which is far away from the target, as the rewards influence from the target state back to the distant starting state. For the value function, it assigns higher values to the states at a greater distance, as the impact of future rewards is not as heavily discounted, compared to the γ¡0.5 dropping exponentially.

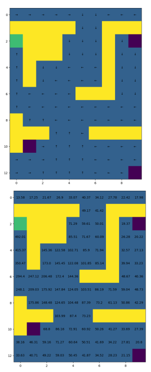
2 Monte-Carlo Reinforcement Learning

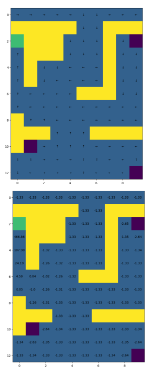
2.1 Q2.1

- The Monte Carlo method we used has several distinctive characteristics and assumptions. 1. The agent employs the every-visit method, considering every visit state-action pair, this enables the agent to learn more effectively than the first visit because of its learning effectiveness, where the agent has limited prior knowledge of the environment. 2. It is in a Model-Free Environment as we are restricted from accessing the transition dynamics (T) and reward function (R) of the environment, but we are allowed to use env.reset() and env.step(), which do not rely on the model environment. 3. We use online updates for incremental learning, the agent updates its policy after each episode.


(a) p=0.10 γ=0.86


(b) p=0.25 γ=0.86


(c) p=0.92 γ=0.86


(d) p=0.92 γ=0.25

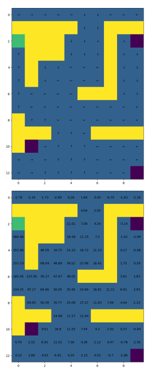

(e) p=0.92 γ=0.75

Figure 2: Policy and Value Function for varying p and γ

This allows the agent to learn from new data immediately, so the agent's policy continually improves over time based on its interactions with the environment. We are not using batch, the agent learns from its experiences incrementally, rather that the batch mode updates occur after collecting multiple episodes. 4. The agent employs a decaying ϵ-greedy policy. Initially, the policy is more exploratory with a high ϵ to discover the environment's dynamics and rewards. Then as the agent accumulates the result of the exploration, ϵ decay, prioritizing the exploitation of the learned information, where decay = epsilon/episodes. This method balances between exploration and exploitation. Implementing an online averaging method, where the Q-values are updated using the running average of returns. Q-values with a learning rate depend on the number of times a state-action pair has been encountered. 5. Assuming the starting episodes from various states have equal probabilities, using 500 episodes, which is large enough to support the convergence. The dynamics of the environment are assumed to be stationary because the rewards do not depend on time, therefore this eliminates the need to forget old episodes. 6. We do not use off-policy methods, because the maze only has 98 possible states, and this method learns after the episodes after the last non-greedy action, which will slow down the learning process.

2.2 Q2.2

-

2.3 Q2.3

- The MC agent learning curve of the mean across 25 runs has the shape of the logarithmic curve, with a larger standard deviation when the number of episodes is small, and a very small standard deviation when the number of episodes reaches 250. When the number of episodes is small, it has limited experience, small amounts of data, and knowledge about the environment. This results in a higher level of uncertainty, where each run may lead to different outcomes, so the standard deviation of the rewards is high. As the number of episodes increases, it gathers more experience, which refines its policy through interactions with the environment. The logarithmic shape indicates that the agent significantly improves its policy and converges toward the optimal strategy. As the number of episodes increases, the stability increases, resulting in smaller variations in the total non-discounted rewards across the 25 runs. The rate of learning slows down as the agent gains more experience and gradually reaches the optimal solution.

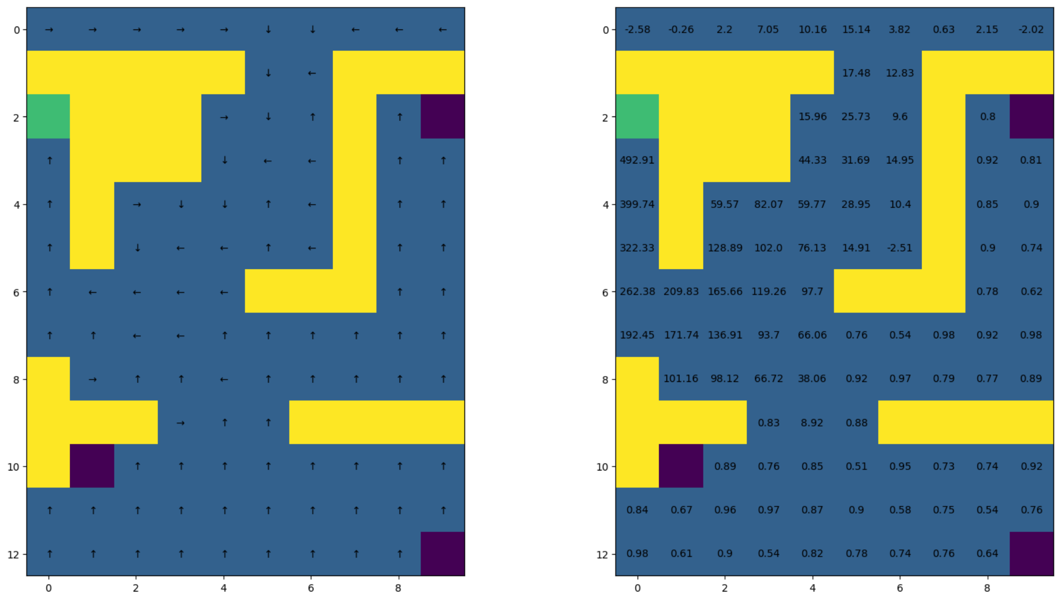

(a) MC Agent Optimal Policy (b) MC Agent Optimal Value Function

Figure 3: MC Learning Optimal Policy and Optimal Value Function

3 Temporal Difference Reinforcement Learning

3.1 Q3.1

- We use the State-Action-Reward-State-Action (SARSA) and on policy learning, it updates the Q-values based on the state transitions and rewards experienced during the episodes. The Q-values are initialized and set to zeros, and the policy is initialized as a uniform random policy. The policy is updated using the epsilon-greedy approach based on the Q-values. ϵ-greedy policy without decay is used because we choose to use a small constant epsilon. The choices of alpha learning rate and epsilon greedy policy are set to 0.3 and 0.1 respectively. The reason behind this is that the lower fixed alpha value results in more stable learning and avoids overshooting optimal value. Also, the number of episodes is set to 500, or else it would take a very long time to converge. From Figure 8b, we can see that when alpha is set to 0.3, it converges quickly with low overshooting. The epsilon value is chosen to be at 0.1 because we would expect to have more exploitation than exploration, so it will choose actions that it currently believes to be the optimal value and learn the environment, we can see that in Figure 7. These values work well as the maze size is limited, it only has 98 state spaces, and the target space is not very far away.

3.2 Q3.2

-

3.3 Q3.3

- Varying the ϵ and α impacts our learning curve greatly. The ϵ-greedy policy has the equation: π(s, a) = $\epsilon/|A(s)|$, ϵ can vary from the value 0 to 1, it tells us about how greedy are our policy. When ϵ = 0, it is purely an exploitation of the current policy, it constantly chooses the action that is believed to be optimal, based on the current Q-value estimates. It does not explore new actions even if the current actions may be incorrect. When ϵ = 1, it is purely exploration, which randomly selects actions without exploiting the learned Q-values, all actions are likely to be chosen. It is highly unlikely to converge. The learning rate alpha compares the difference between Q(s, a) and reward + γ(max Q(s', a)). When alpha is small, which is close to equal to Q(s, a), it updates its value estimate slowly, having less weight on new information and less sensitivity to the rapid change of the environment, but it is less likely to

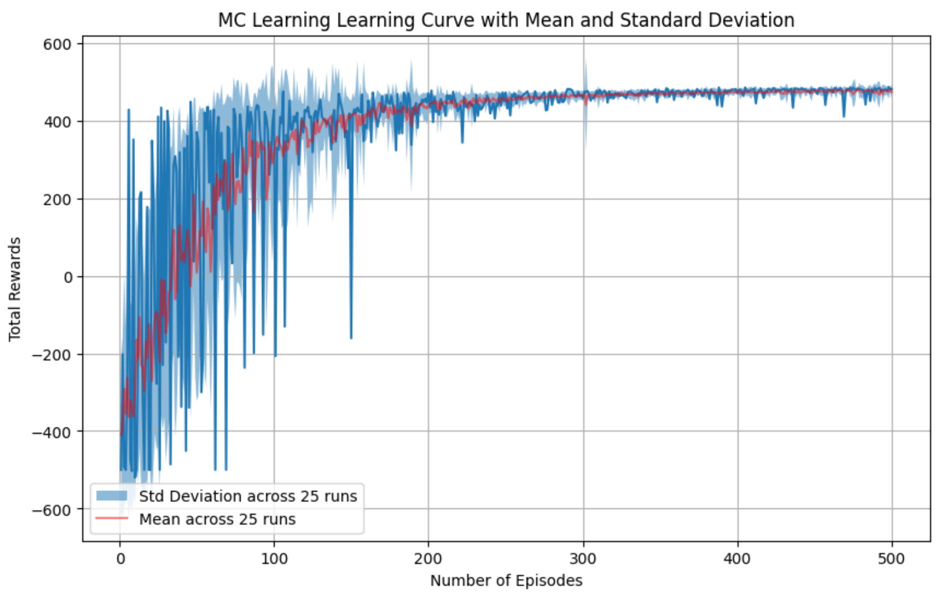
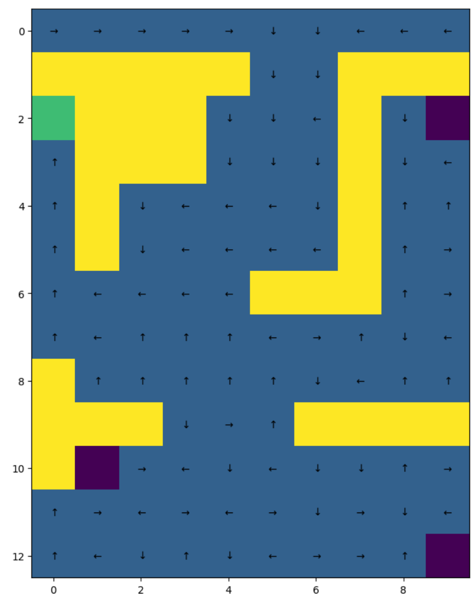


Figure 4: MC Learning Learning Curve: the total non-discounted rewards against the number of episodes with 25 loops.

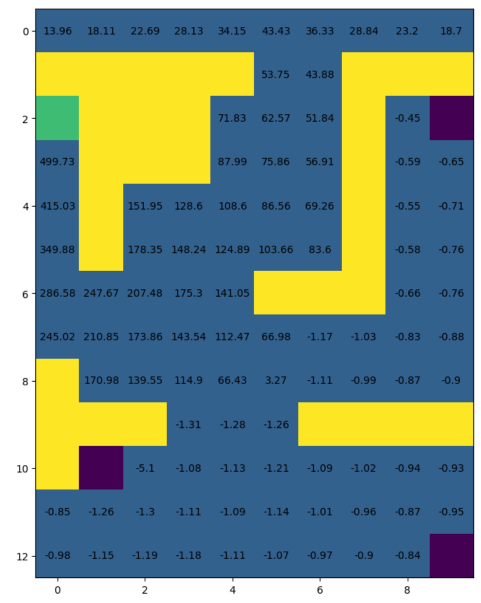
result in overshooting. This leads to slower convergence and requires more episodes. When alpha=0, it does not learn. When alpha is large, where becomes Q(s, a) = γ(max Q(s', a)) when alpha=1, it updates its value estimate quickly, updating greedily, reaching convergence very quickly. But this might lead to overshooting the optimal value and even divergence in a noisy or complex environment.

From Figure 7a, when alpha=0.3 and epsilon=0.1, it has a similar graph as the one in Q2.3, having the logarithmic shape that the agent significantly improves its policy at the start and converges toward the optimal strategy as the number of episodes increases. Compared to Figure 7b, where alpha=1 and epsilon=0.1, it has a faster convergence as we can see a steeper slope when the number of epsilon is small.

Figure 7c and 7d with alpha=1 and epsilon=1, and alpha=0.1 and epsilon=1 respectively, we can see that it is in a random walking stage and the standard deviation varies significantly across all episodes. The total non-discounted sum of reward remains negative around -200 to -400, indicating that it does not converge as the agent is randomly exploring without exploiting the learned Q-values. From Figure 8a, we can see that epsilon starts moving significantly, having both positive and negative total non-discounted sum of reward values, from epsilon = 0.7 to 1.0. From Figure 8b, we can see that alpha=1.0 has significant overshooting at the start, having both positive and negative total non-discounted sum of reward values as well.

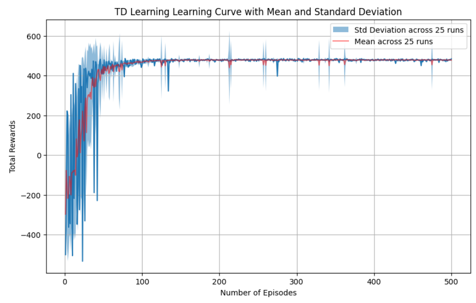


(a) TD Agent Optimal Policy

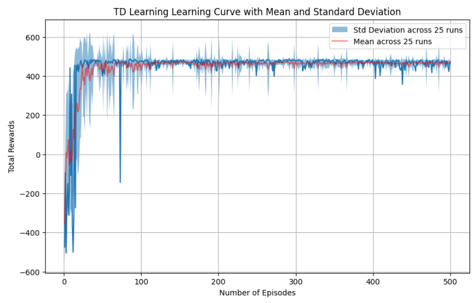


(b) TD Agent Optimal Value Function

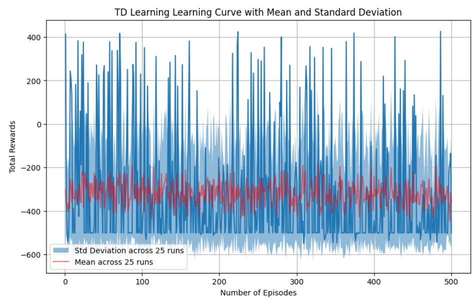
Figure 5: TD Learning Optimal Policy and Optimal Value Function



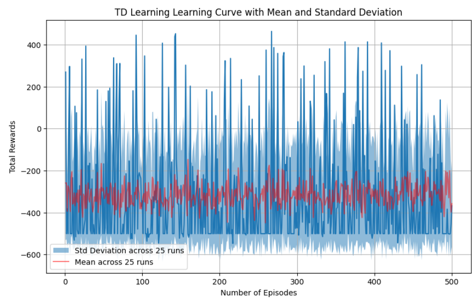
(a) TD Learning Learning Curve α=0.3,ϵ=0.1: the total non-discounted rewards against the number of episodes with 25 loops.



(b) TD Learning Learning Curve α=1,ϵ=0.1: the total non-discounted rewards against the number of episodes with 25 loops.

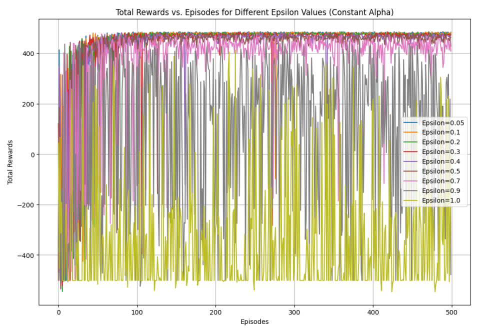


(a) TD Learning Learning Curve α=1,ϵ=1: the total non-discounted rewards against the number of episodes with 25 loops.

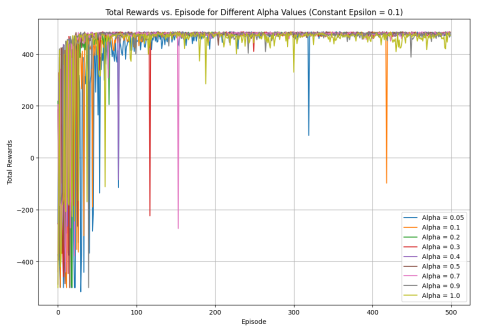


(b) TD Learning Learning Curve α=0.1,ϵ=1: the total non-discounted rewards against the number of episodes with 25 loops.

Figure 7: Learning Curves for hyper-parameter α and ϵ



(a) Total Rewards vs. Episodes for Different Epsilon Values (Constant Alpha=0.3)



(b) Total Rewards vs. Episode for Different Alpha Values (Constant Epsilon = 0.1)

Figure 8: hyper-parameter of constant α or ϵ