Report of coursework 1

Contents

Introduction

In this coursework, three methods of implementing reinforcement learning were applied to an agent. The goal is to find the optimal policy for the agent to move to a designated absorbing state regardless of which state to be the initial state.

Question 0: defining the environment

Two functions, get_CID() and get_login() were filled with CID number and short login for auto-marking purpose. The second last digits in CID were used to define transition probability and discount factor for defining the environment.The transition probability, p was calculated based on the following function $p = 0.8 + 0.02 * (9 - 8)$, which is $p = 0.82$ defining the chance that the agent moves to the desired state. The discount factor was calculated based on the function $\gamma = 0.8 + 0.02 * 8$, which is $\gamma = 0.96$. In the maze, to simulate each step would have a cost, all states were initially set with value $r = -1$.

Question 1: Dynamic programming

Question 1.1: Method and parameters

To solve the maze problem with a fully defined model for the environment, a value iteration algorithm was applied to the agent to solve for optimal solutions for the path to reward state regardless of where it initially started. There are several reasons for choosing a value iteration algorithm. The most significant reason is that the value iteration algorithm would not require policy evaluation sweep through all the iterations compared with a policy iteration algorithm (Richard S. Sutton & Andrew G. Barto, 2018). In a policy iteration algorithm, the optimal policy cannot be acquired until the value function converges, protracting the computation process. With a value iteration algorithm, the computational approach can be truncated to accelerate the solving process. Another reason for choosing the value iteration algorithm is that it would reduce the coding complexity because value iteration directly turns the Bellman optimality equation into an update rule for the value function. By implementing two algorithms, the time required for the code with a value iteration algorithm to specify optimal value is shorter than that of the policy iteration algorithm.

Values of transition probability, discount factor (γ) and reward state had been explained in the previous text. Firstly, the threshold used to determine the convergence was selected as 0.0001, which is significantly small to consider the value function is converged.

Question 1.2: Result of optimal policy and value functions

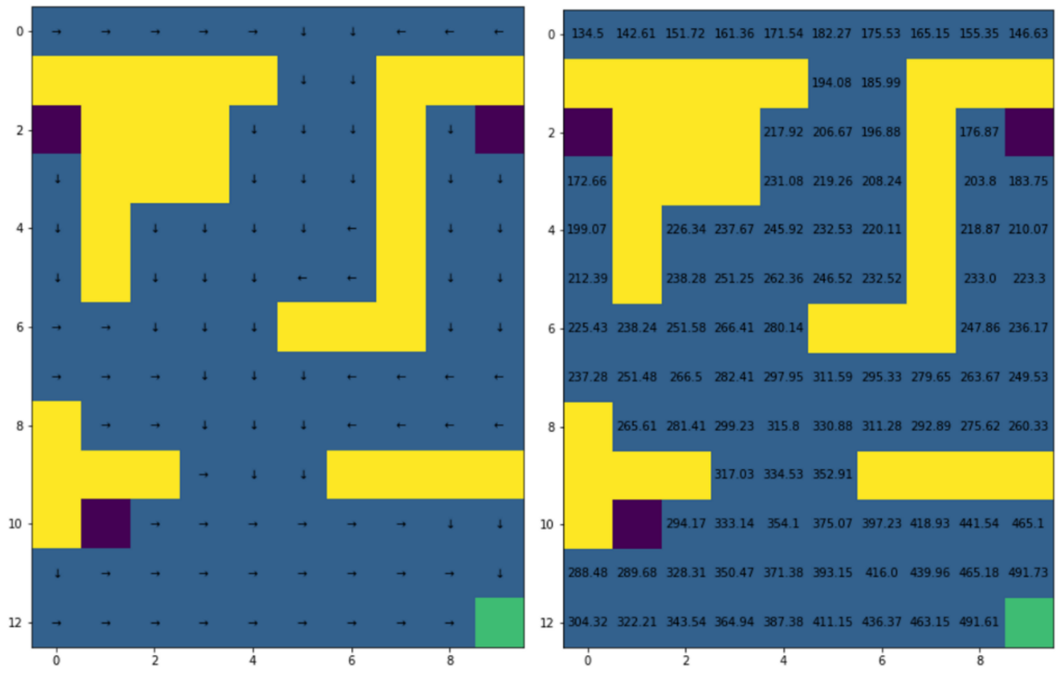

Figure 1. The resulted policy and value function on grid acquired computed by a value iteration algorithm [Author's original figure]

Question 1.3: The influence of γ and p

The parameter γ is representing the discount factor, which regulates the weight of values in successor states. The parameter p is used for denoting the probability of transition to the next state under a given policy. The optimal solution would be calculated through the logic of the equation, $\pi(s) = argmax_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$, where the relationships among these two parameters and policy are specified (Richard S. Sutton & Andrew G. Barto, 2018).

Influence of γ

The value γ tells the agent how far should be taken into consideration. With a low gamma value, the final value equations would mislead the final policy, which results in those states can lead the agent in a wrong direction. The reason is that the discount factor would weigh the value of successor states exponentially. Once the state is far away from reward state, the influence of the value in reward state would be significantly reduced and resulted in infinitely slight changes in the value function (the change can be neglected if it is smaller than threshold) through each epoch. As the value of γ increased, the states with long distance from the reward state would start to result to point towards reward state.

Influence of p

The value p stands for the probability of transitioning to the desired state. If the value of p is less than 0.25, it means that there is less chance for the agent to move towards the desired direction. In this way, the policy tends to move the agent to the state with the smallest state value, which is the reverse direction to the reward state. If $p = 0.25$, it means that the action would result in a completely random direction since all transitions are equally distributed. The graphical result is that all policies are moving the agent to the north. If p is larger than 0.25, it means that the agent is more likely to move towards the desired direction. Thus, the policies at all states are moving the agent closer to the reward state.

Question 2: Monte-Carlo Reinforcement Learning

Question 2.1: Method, parameters, and assumption

On-policy first-visit Monte-Carlo algorithm was adapted for solving the problem. The reason for choosing this algorithm is because that there were no precedent examples, and it can reduce coding complexity. To update the policy, ε-greedy policy was adapted to balance the selection between exploring and currently known optimal policy. Moreover, the maze was assumed to be stationary.

The ε for determining the policy was selected as 0.7. The reason for choosing ε in this value is that the agent would relatively more tend to explore the maze instead of finding optimal value only based on current data. In this way, theoretically, a smaller number of episodes would be required to confirm value function for all states. The number of episodes was selected to be 12000. This number was confirmed by running the program many times.

Question 2.2: Result of optimal policy and value functions

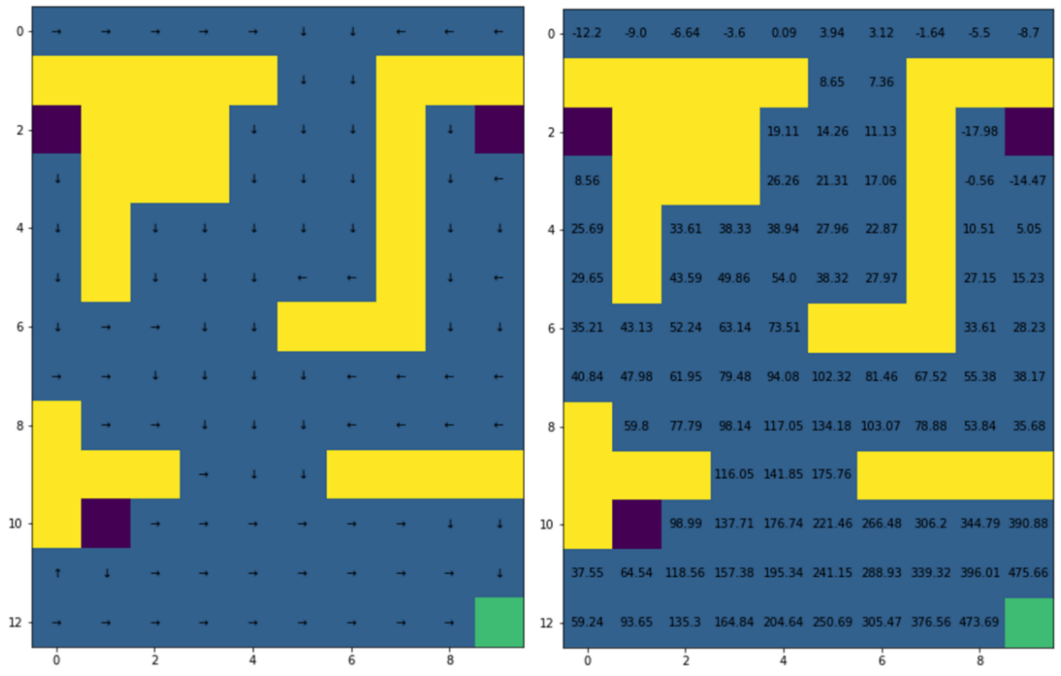

Figure 2. The resulted policy and value functions on grid acquired computed by a Monte-Carlo algorithm [Author's original figure]

Question 2.3: Variability in results and solution

Because both the initial action value matrix and the initially started location were generated randomly, the learning result with a low number of episodes may significantly differ from the optimal policy acquired from dynamic programming (which can be considered as the real optimal policy). By repeating the learning program several times, it was discovered that with the training amount of 12000 times, the final policy would be almost the same as a result acquired from dynamic programming. Moreover, by replicating the process 12000 times, the number of steps required from the arbitrary state to the reward state are the same.

Question 2.4: Learning curve of the agent

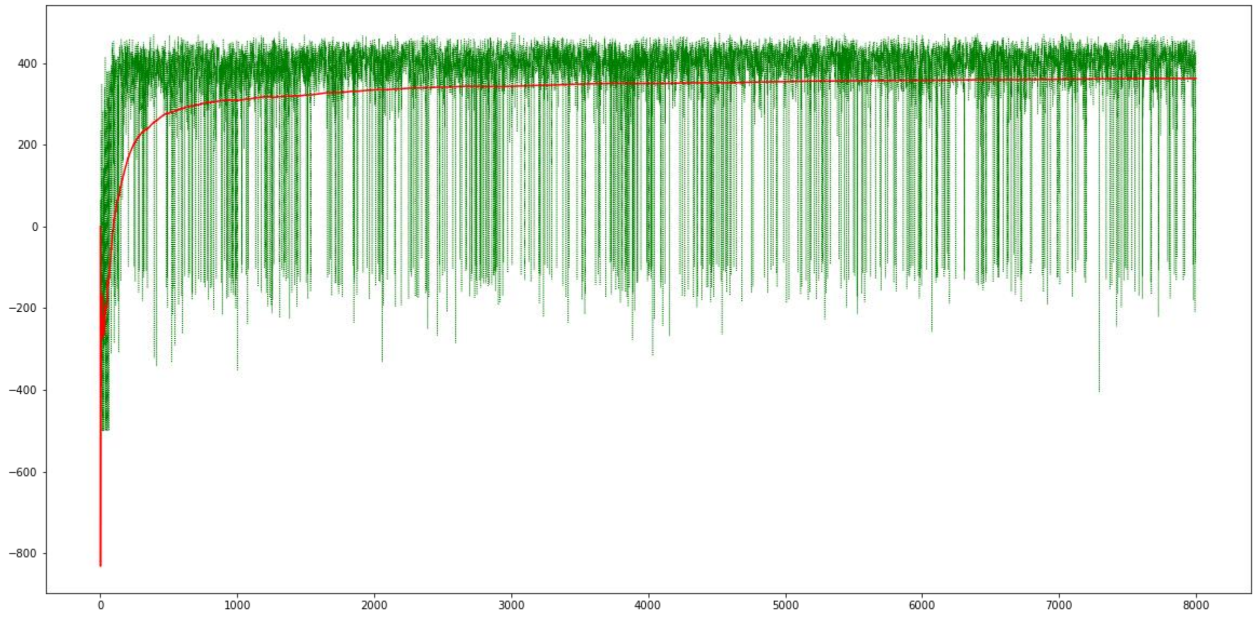

Figure 3. The learning curve of the agent with Monte-Carlo algorithm, where red line is representing the averaged total reward and the green line is representing the total reward of each episode [Author’s Original figure]

Question 2.5: Exploration parameter ε

In this part, exploration parameter was selected as 0.7, which is relatively balancing between exploring the grid world and determining the optimal policy. With a small exploration parameter, according to the formula of ε-greedy policy presented below, the agent would tend to find optimal solution with calculated action value functions (Richard S. Sutton & Andrew G. Barto, 2018).

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|A(S_t)|} & (the\ possibility\ to\ finding\ the\ optimal\ policy) \\ \frac{\varepsilon}{|A(S_t)|} & (the\ possibility\ of\ exploring\ the\ environment) \end{cases}$$

By experimenting the exploration parameter in 0.1, 0.3, 0.5, and 0.7, it was discovered that the total reward would converge faster with a small exploration parameter in 4000 episodes. However, by comparing the resulted policy with the reference policy from dynamic programming, the policies acquired with a low ε value are not optimal in some states.

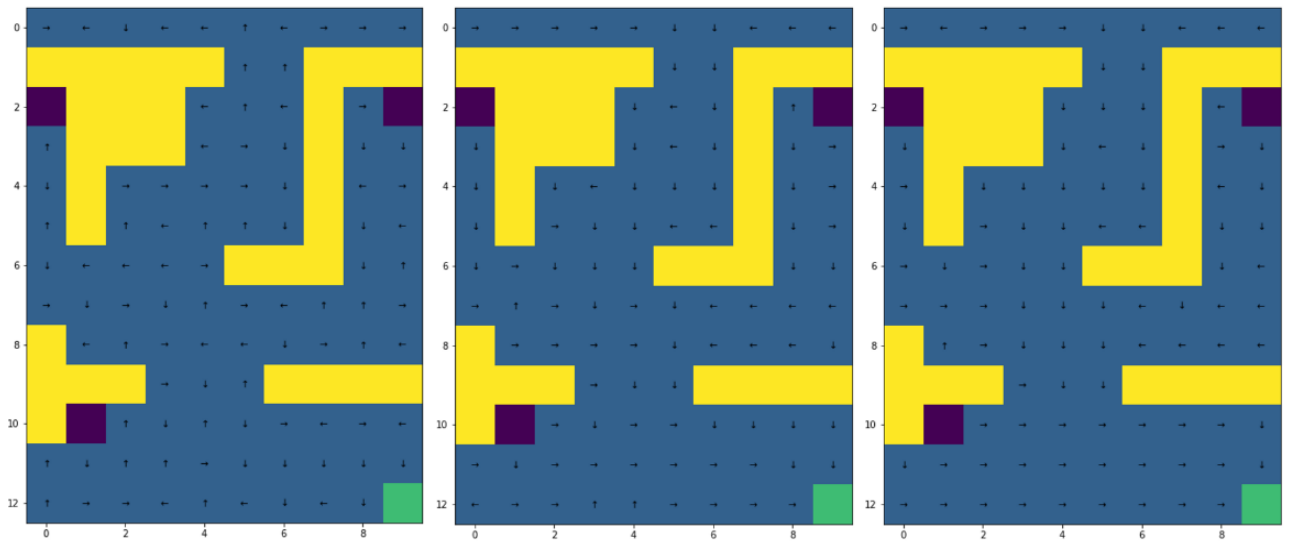

Figure 4. The policy acquired with $\varepsilon = 0.1, 0.3, 0.5$ (from left to right), there are several errors in the policies showed in these figures.

Question 3: Temporal Difference Reinforcement Learning

Question 3.1: Method, parameters, and assumption

The method designed is SARSA (on-policy TD control) algorithm. This algorithm is a straightforward method to implement temporal difference learning. The TD control method is highly suitable for the object with Markov properties since the concept of state and action was induced. While determining the policy, ε-greedy policy was used to trade off between exploration and exploitation.

As for parameters, the exploration parameter, ε was chosen to be 0.7, which was tested to a decent value for solving the problem. The learning rate, α, is scaling the step size to compensate the error between the outcome value of an action and the expected outcome value of the action. α should be a small value to prevent oscillating around the convergence. Therefore, based on the principle that convergence is the priority, the learning rate was selected as 0.05. The environment was assumed to be a Markov state space.

Question 3.2: Result of optimal policy and value functions

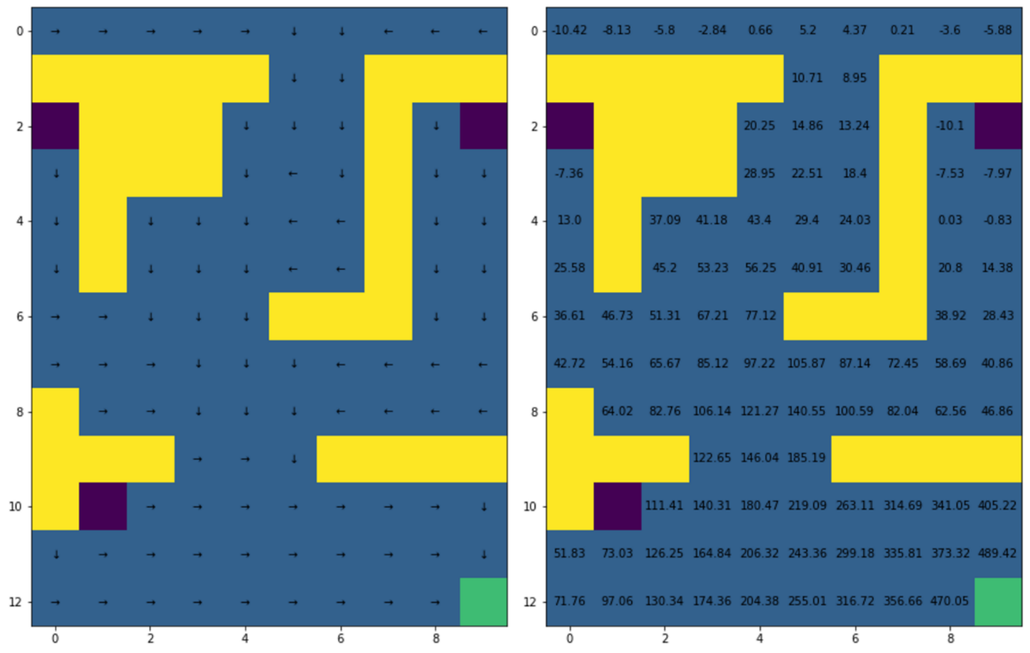

Figure 5. The resulted policy and value functions on grid acquired computed by a Temporal Difference algorithm [Author's original figure]

Question 3.3: Learning curve of the agent

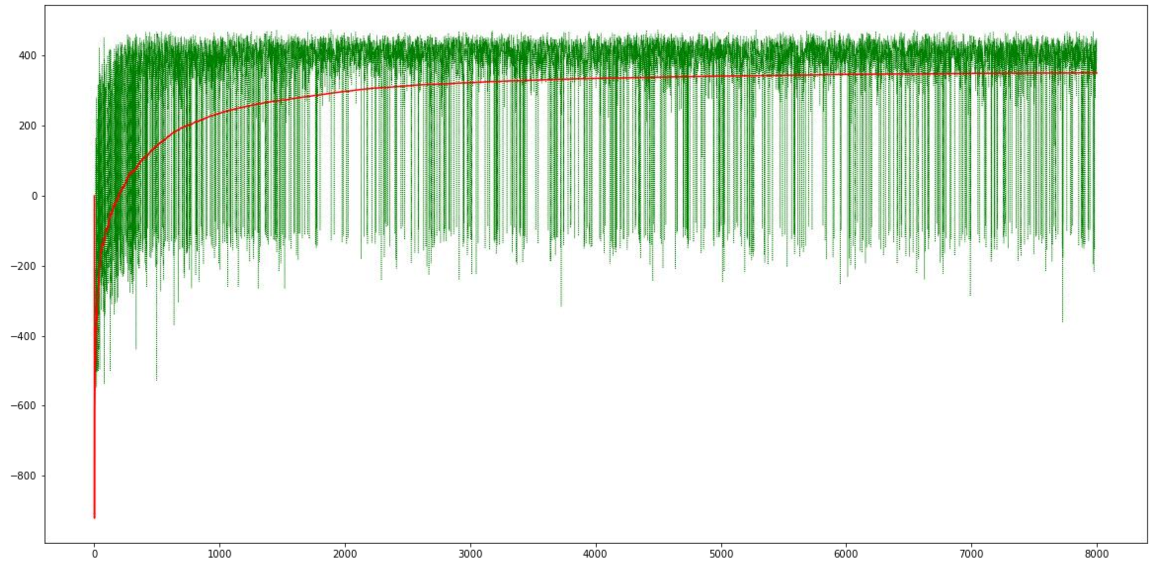

Figure 6. The learning curve of the agent with temporal difference algorithm, where red line is representing the averaged total reward and the green line is representing the total reward of each episode [Author's Original figure]

Question 3.4: Learning rate α and exploration parameter ε

The learning rate is regulation the step size of compensating errors between the acquired values and expected values in different state-action pairs. To reach the convergence of the value functions, learning rate α should be a relatively small value to avoid oscillation. According to multiple tests with learning rate in an increasing order, the number of episodes that takes to reach convergence would tend to reduce while the accuracies of the policies would also reduce. In this way, a training strategy with small α and many learning episodes was selected.

The exploration parameter ε, according to the equation presented in question 2.5, would influence the episode required to reach the convergence status. According to the tests conducted, the number of replications required for convergence would decrease with the value of ε decreases. However, with a long ε, the agent would be biased on exploiting optimal policies while exploration can be neglected, which would lead to false optimal policy.

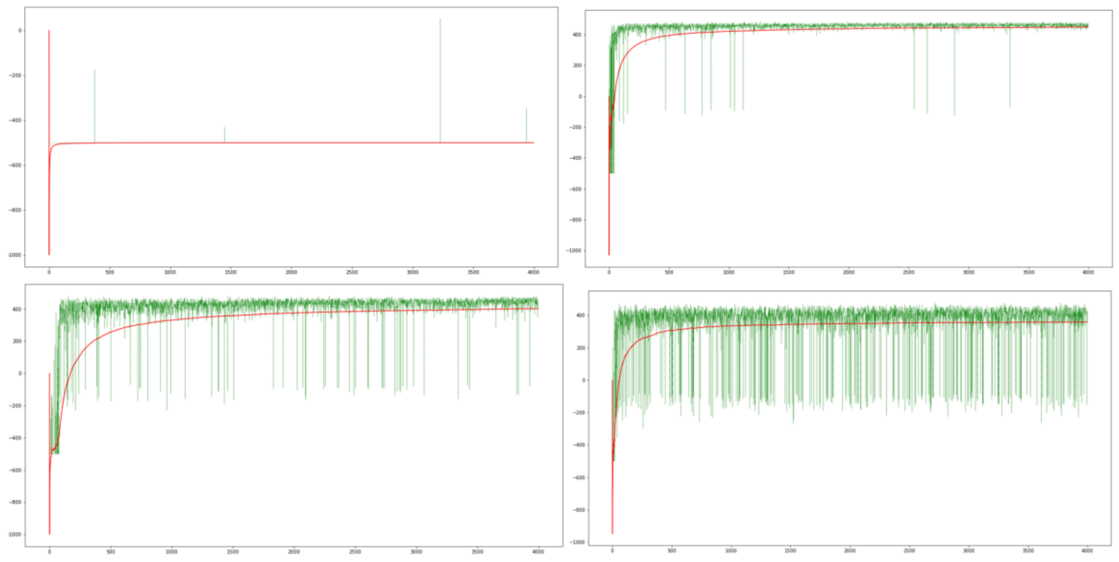

Figure 7. The learning curve with $\varepsilon = 0.1$ (top left), $\varepsilon = 0.3$ (top right), $\varepsilon = 0.5$ (bottom left), and $\varepsilon = 0.7$ (bottom right) [Author's Original figure]

Question 4: Comparison of Learners

Question 4.1: Value function estimation error

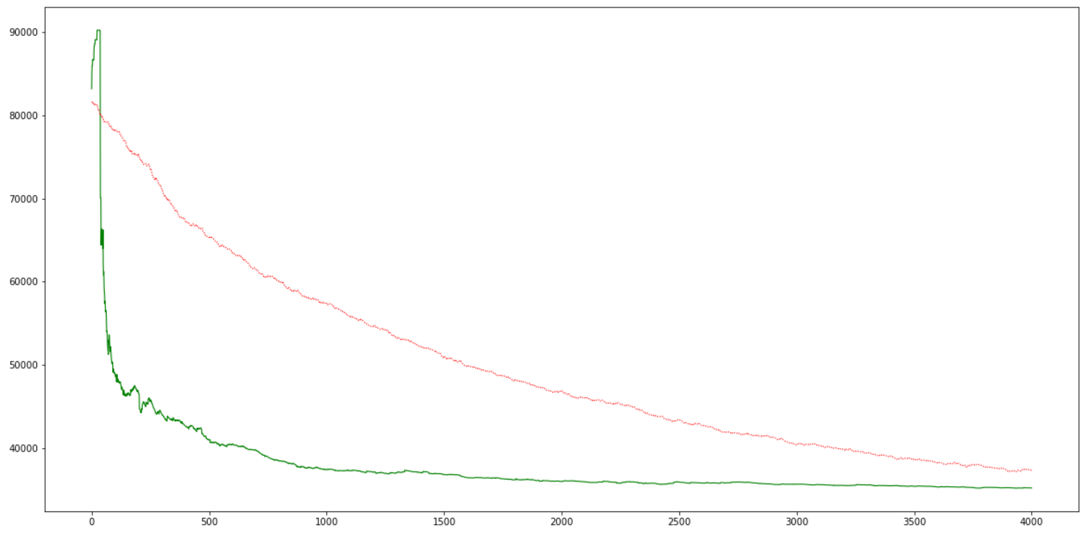

Figure 8. The estimation error (in mean squared error) for MC (green) and TD (red) learners referencing with the state value acquired from DP. The X axis is representing the number of episode and the Y axis is the estimation error [Author's Original figure]

Question 4.2: Analysis of two approaches refer to estimation error

With the plot presented above, it can be discovered that the solution of the MC learning algorithm converges faster than that of the TD learning algorithm. Firstly, the main reason for slowing down the convergence speed of TD is that the learning rate was set to a small value, which reduced the step, compensating the error between the actual value and expected value. Secondly, the estimation error of the MC algorithm initially changed significantly since the world has not been fully explored. Therefore, since the value of ε was set to a relatively large number, the agent would tend to conduct exploration instead of exploitation of optimal policy. Once the world was completely explored, the agent then would have a more significant chance to optimize its policy, which caused the rapid decrease in estimation error.

Question 4.3: Scatter plot

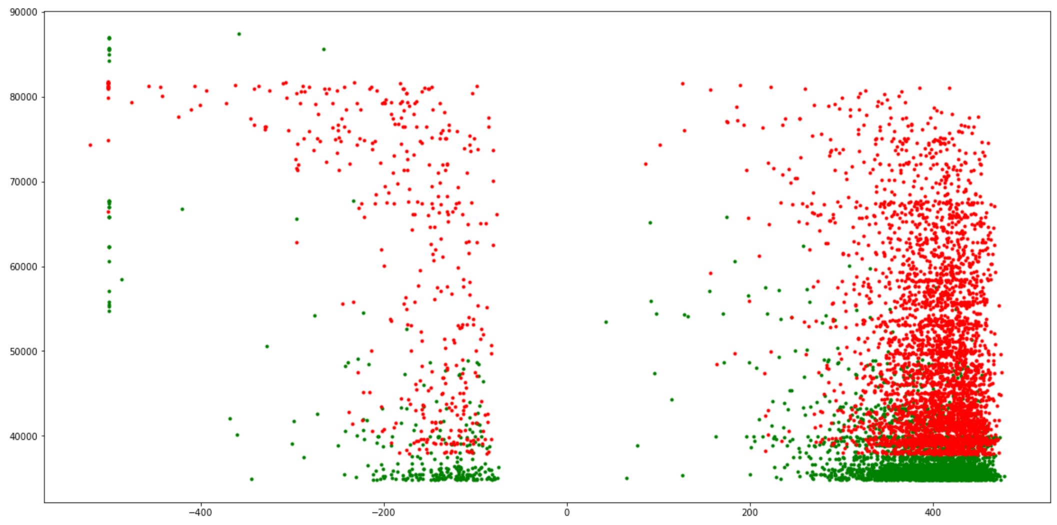

Figure 9. Scatter plot for estimation error (Y axis) against total non-discounted sum of reward for certain episode. The red dots are representing data from TD learner and the green dots are representing data from MC learner. [Author's Original figure]

Question 4.4: Explanation and Conclusion from scatter plot

The points in the plot above represent the relationship between the reward accumulated in an episode and the corresponding estimation error under different learning algorithms. A good value function estimation is low in estimation error, which is low on the Y-axis value in the plot above. Another characteristic of the plot is that most of the points are distributed at the right bottom corner, which are the episodes low in estimation errors and high in accumulated rewards. Therefore, a conclusion can be drawn that with the estimation error reduced (meaning that the value function is optimizing), the total rewards would be increased, which means that the agent can solve the problem with more optimized policies.

Reference

Richard S. Sutton & Andrew G. Barto (2018) *Reinforcement Learning An Introduction*. Second edition. Cambridge, Massachusetts, The MIT Press.