Reinforcement Learning Coursework 1 Report

Andy Fu

CID: 01737344

1 Dynamic Programming

1.1 Design, Parameters and Justification

Since value iteration is considered a particular case of policy iteration, the policy evaluation part of the algorithm is only repeated once ($max\_iter = 1$) for each policy improvement. I have implemented the policy iteration algorithm for this question, but both algorithms could be tested by simply changing the parameter max_iter.

Besides max_iter, ϵ-convergence of value function is also used as a stopping condition for the policy evaluation. ϵ-convergence stops iteration early when the update on the value function is too small, hence enhancing the efficiency of the algorithm.

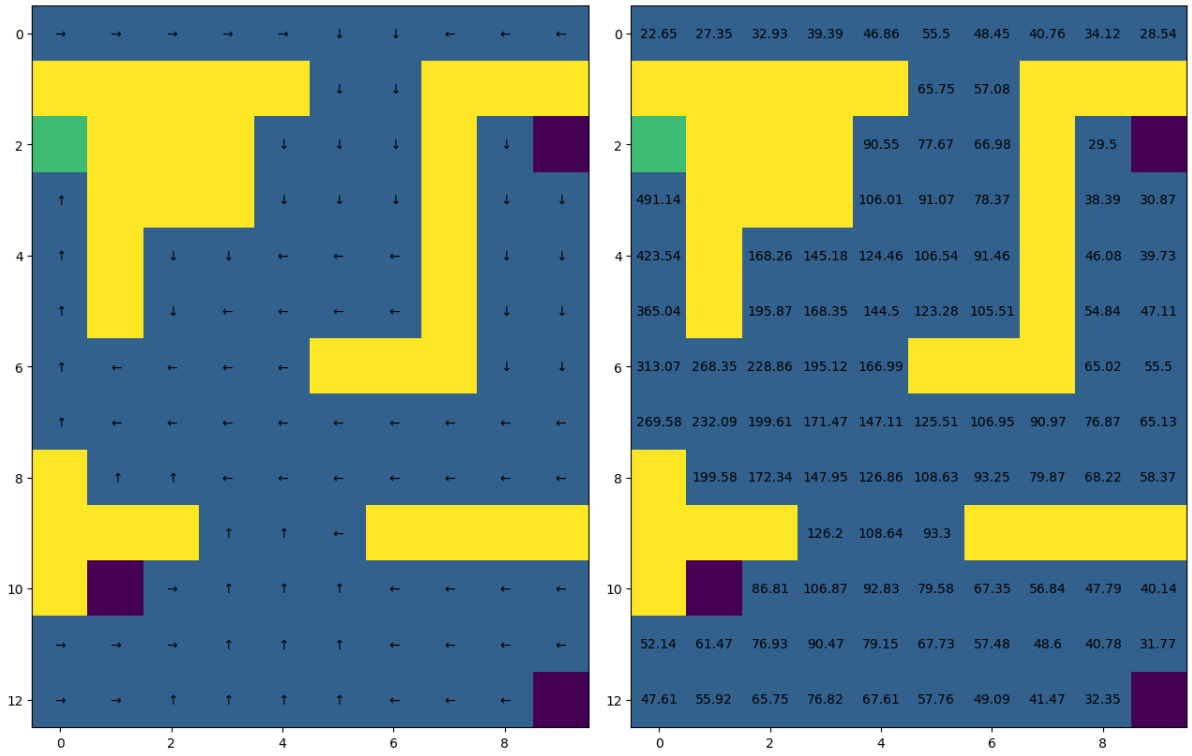
I have added an extra if condition to execute value iteration: When $max\_iter = 1$ and the value function converges, the policy will only be updated once, even if it is unstable ($policy\_stable$ is False).

In this question, we assume the world can be perfectly modelled by a finite and static Markov decision process (MDP): there are a finite number of states and a finite number of actions; the transition and reward functions are known; the current state of the agent only depends on the previous state and the previous action; the environment will never change. These assumptions are essential for solving the agent problem with dynamic programming, although they are often wrong in real-world situations.

I have tested value iteration with $\epsilon = 0.001$ and policy iteration with $\epsilon = 0.001$, $max\_iter = 1000$. Both settings yielded the optimal policy. For a simple environment like Maze, dynamic programming algorithms would converge quickly. Therefore, small ϵ and max_iter are suitable.

1.2 Graphical Representation

Figure 1 demonstrates the optimal policy and value function of the Maze environment. Both value iteration with $\epsilon = 0.001$ and policy iteration with $\epsilon = 0.001$, $max\_iter = 1000$ yield the same result.



(a) Optimal Policy: DP

(b) Optimal Value Function: DP

Figure 1: Graphical Representation of Results of DP

1.3 Probability of Success and Discount Factor

When the probability of success p is smaller than 0.25, the agent will be least likely to move toward the intended direction. Therefore, the policy will always intend to move towards a neighbouring state with the lowest value. The probability of the agent moving to the neighbouring state with the highest value is the same as moving to the other two neighbouring states. The agent takes longer to reach a rewarded terminal state. Therefore, the value of each state is closer to the infinite sum of discounted action rewards ($\sum_{i=0}^{\infty} -\gamma^i$).

When the probability of success p is 0.25, the agent's state is independent of the previous action. The agent will learn no policy after the random initialisation. However, the value can still be learned, as the agent has a higher chance of reaching a nearer terminal state.

When the probability of success is slightly greater than 0.25, the agent tends to behave randomly when away from terminal states. The agent is expected to walk aimlessly with more steps before reaching a terminal state, and the discounted reward of the terminal state becomes negligible. The agent cannot learn a policy efficiently with sparse rewards. The value of each state is also closer to $\sum_{i=0}^{\infty} -\gamma^i$ compared to when the probability of success is larger.

Similarly, a small discount factor γ will make the agent "short-sighted"; the final policy has random behaviours when the agent is away from all terminal states. Moreover, the state value will become closer to the immediate action reward (-1).

In contrast, a significant discount factor (i.e., 0.98) may mitigate the problem of sparse rewards, as the optimal policy could be found when $\gamma = 0.98$ even if the probability of success is small

($p = 0.26$). The value of all states will become closer to the highest final reward, as the final reward will be discounted less.

2 Monte Carlo Reinforcement Learning

2.1 Design, Parameters and Justification

I have chosen the on-policy every-visit Monte Carlo (MC) reinforcement learning algorithm with ϵ-greedy policy for this question. The off-policy algorithm was not used due to its greater variance[1].

For this question, we assume the environment can be modelled by an unknown MDP with finite states and actions; each state can be reached, and the agent can perform each action; the current state of the agent only depends on the previous state and the previous action; the environment does not change throughout each episode.

Let (s, a) be a state-action pair such that s is a state next to a terminal state; the success of action a will lead the agent to the terminal state. The reward of acting a at state s can be either -1, -50, or 500. Since the agent will no longer move after reaching a terminal state, the reward of the "first but not the last visit" of (s, a) will always be -1, and the action value $Q(s, a)$ will be less affected by the final reward. The every-visit algorithm is implemented to mitigate this problem, although it is slower to converge.

When ϵ is too small, the policy is more greedy (less exploration). Once the agent learns a path to the desired terminal state (with $reward = 500$), it will be improbable to explore states near an unvisited terminal state, leading to ineffective learning. As a result, the agent policy may move randomly near some undesired terminal states.

When ϵ is too large, the agent's behaviour is less guided by the action value function in each episode (less exploitation). The agent will take longer on average to reach the terminal state, making the discounted terminal reward negligible. The agent will learn ineffectively when it is far from all terminal states. As a result, the agent policy may move randomly when the agent is far from all terminal states.

To balance exploration and exploitation, $\epsilon = 0.7$ is used. I chose $max\_episode = 1000$ for a low computational cost.

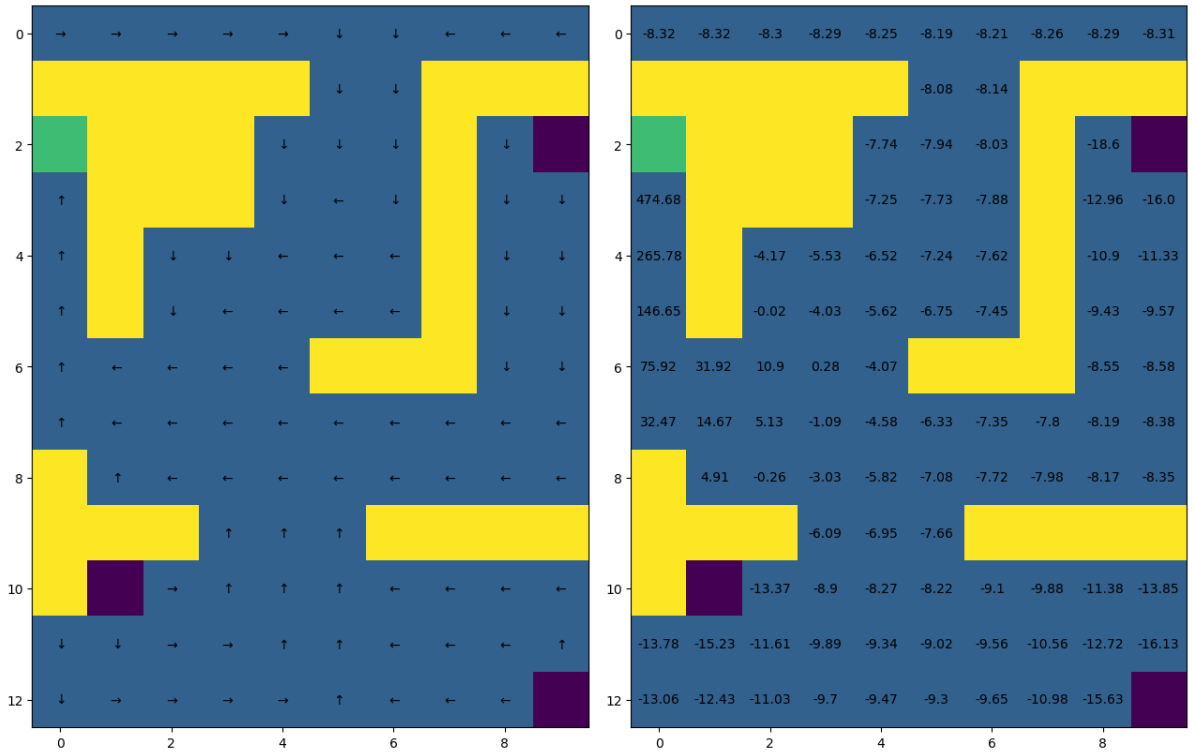
2.2 Graphical Representation

Figure 2 demonstrates the estimated value and resulting policy of the Monte Carlo algorithm. The algorithm is non-deterministic; results of multiple runs may differ.

2.3 Learning Curve

Figure 3 shows the mean and standard deviation of the total non-discounted sum of rewards against the number of episodes. The learning is repeated 25 times, and the learning curve is smoothed to improve readability.

The standard deviation is low at early episodes as the total rewards in all 25 runs are low at the



(a) Resulting Policy: MC

(b) Estimated Value Function: MC

Figure 2: Graphical Representation of Results of MC

beginning. The average reward then increased sharply, while not all experimental runs started to converge at the same episode. Thus, the standard deviation increased. All experimental runs converged after around the first 100 episodes, and the average total reward was high, and the variance of total reward was low in later episodes.

3 Temporal Difference Reinforcement Learning

3.1 Design, Parameters and Justification

Temporal Difference (TD) algorithms usually have lower variance than MC algorithms; thus, I have used on-policy MC and off-policy TD (Q-learning) in this coursework. As demonstrated by the Cliff walking example in lecture 3.2, Q-learning may outperform SARSA.

Similar to the previous section, we assume a MDP with finite states and actions can model the environment. Each state can be reached, and the agent can perform each action; the current state of the agent only depends on the previous state and the previous action; the environment does not change throughout each episode. Moreover, we make an assumption of coverage: for every action taken by the target policy, the probability of a behaviour policy choosing such action should be greater than 0. Thus, a ϵ-greedy soft behaviour policy and a deterministic greedy target policy are necessary. Q-learning is a powerful algorithm for the Maze environment, and it can find a good policy with most ϵ and learning rate α values. Since the action value is not directly computed using rewards of the ϵ-greedy behaviour policy in Q-learning, a larger ϵ is suitable as it encourages

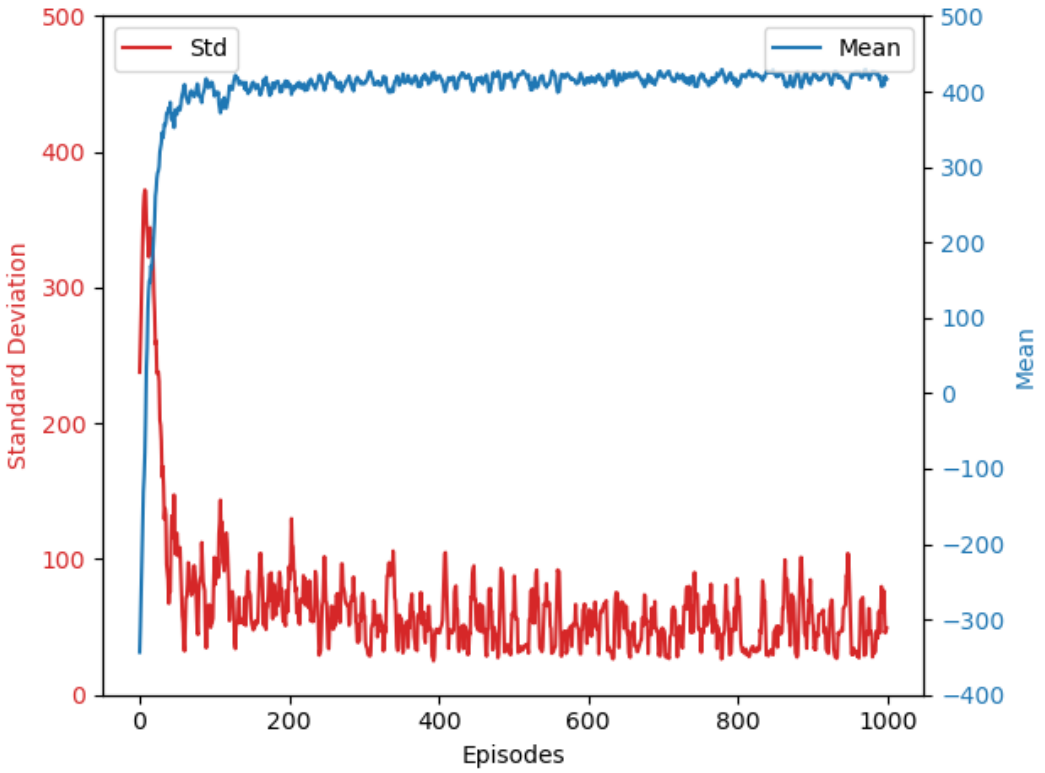
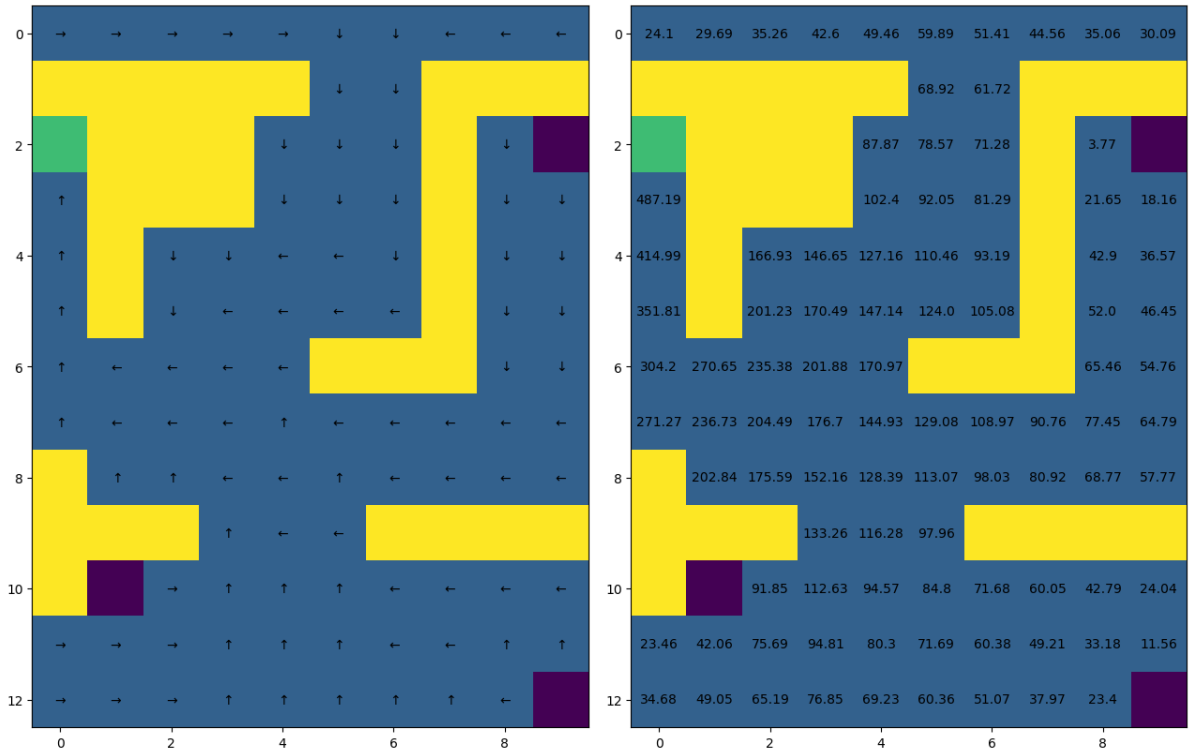


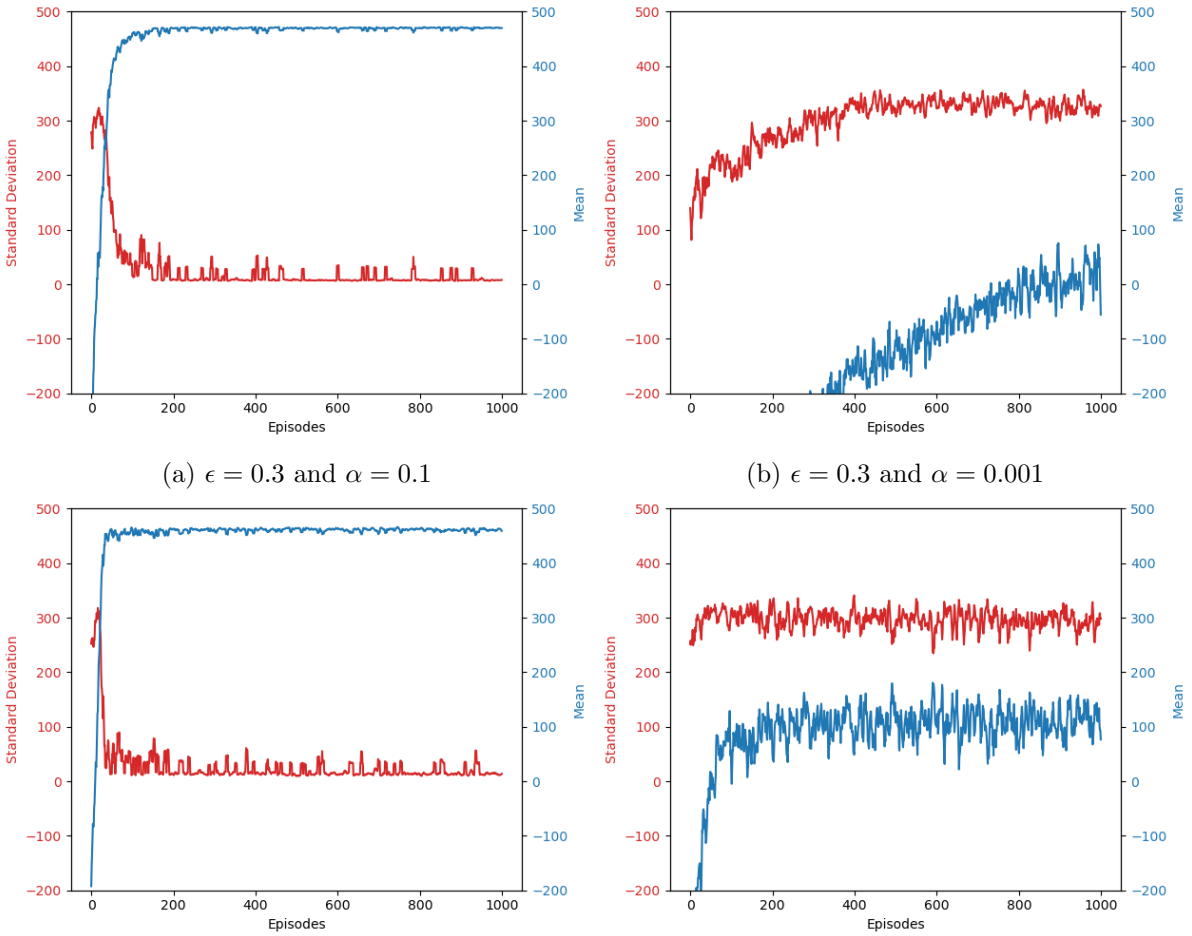
Figure 3: Smoothed Learning Curve of MC



(a) Resulting Policy: TD

(b) Estimated Value Function: TD

Figure 4: Graphical Representation of Results of TD



(a) $\epsilon = 0.3$ and $\alpha = 0.1$

(b) $\epsilon = 0.3$ and $\alpha = 0.001$

(c) $\epsilon = 0.3$ and $\alpha = 0.9$

(d) $\epsilon = 0.9$ and $\alpha = 0.1$

Figure 5: Learning Curves of TD with Different Settings

exploration and even sampling. I set $\epsilon = 0.9$, $\alpha = 0.1$, and each run involves 1000 episodes.

3.2 Graphical Representation

Figure 3 demonstrates the estimated value and resulting policy of the Q-learning algorithm. The algorithm is non-deterministic; results of multiple runs may differ.

3.3 Parameters and Learning Curve

With a large exploration parameter ϵ, the agent will likely explore and ignore the attraction of positive reward. It is more likely to reach states with negative rewards and take longer before reaching the terminal state with a positive reward. The expected total reward in each episode is inversely correlated to the exploration parameter ϵ. Moreover, larger ϵ introduces more randomness, so the variance of the total reward rises as ϵ gets larger.

The learning rate α decides the pace of updating the action value function. With a large α, the average total reward converges quickly but may overshoot, causing a slightly higher variance

in later episodes. With a tiny α, the algorithm may not converge before the limited number of episodes.

As shown in Figure 5, when the exploration parameter ϵ is fixed to 0.3, the average reward converges earlier with $\alpha = 0.9$ but much later with $\alpha = 0.001$. The variance of later episodes is slightly higher when $\alpha = 0.9$ compared to when $\alpha = 0.1$. When the learning rate α is fixed to 0.1, the expected total reward drops when ϵ is larger, but the variance of the total reward is larger when $\epsilon = 0.9$ compared to when $\epsilon = 0.3$.

References

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.