70028 - Reinforcement Learning Coursework 1, November 2024

Belfiore Asia, CID: 02129867

Q1) DYNAMIC PROGRAMMING

a) I implemented DP Policy Iteration with parameters: $\gamma = 0.92$ and $p = 0.86$ (both automatically calculated from personal CID). I set the initial action from any start state to be 1 (i.e. E, going east) although the same results arose when choosing both random action choice and initial action equal to any other one (0-N,2-S,3-W). I set the delta threshold for policy evaluation to 0.0001 as it yielded the best results out of all the tested values (0.01, 0.001, 0.0001).

b) The DP agent with parameters described in Q1.a), I achieved the following optimal policy and value functions (Figure 1b):

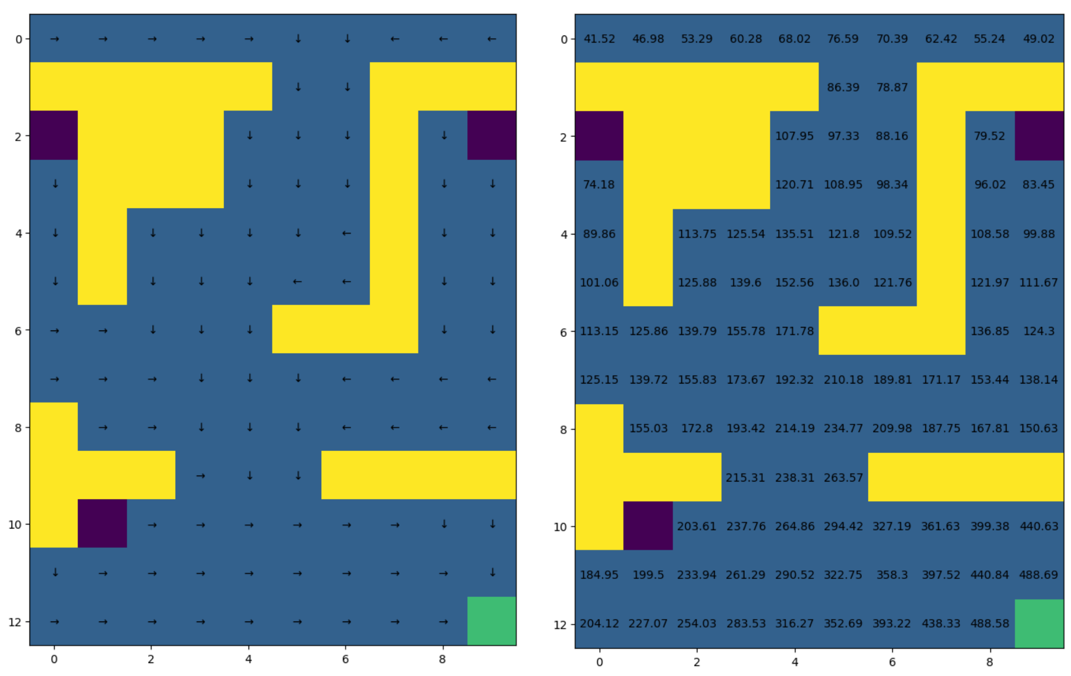


Figure 1b: DP Policy (Left) and Values (Right) with $\gamma = 0.92$ and $p = 0.86$

c) Setting the discount factor $\gamma = 2$ theoretically means that the agent cares about future rewards exponentially more than immediate ones. More precisely, it means that any reward collected at time t, r_{t+1}, is valued twice as much as the reward collected at the previous time step $t-1$, i.e. $r_{t+1} = 2r_t$. The total reward gained from the start to some time step t is:

$$R_t = \sum_{k=0}^{t} 2^k r_{k+1}$$

Since future rewards are much more valued, both the policy and value function will be affected as V is the expected return collected from a given state. Note that this could result in a never-ending loop (or an unfeasibly long search) since any value $\gamma \geq 1$ is not assured to converge.

d) To model such function for any state s_t and terminal states s_T and goal state s_G we can set:

- Discount factor $\gamma = 1$, to ensure the model is far-sighted and cares heavily about future rewards. Since we are concerned with '*eventually* reaching the terminal state' (and not how long it takes to get there), we can set γ to 1 even if it may lead to infinite state transitions.
- Reward for any state s_i that never reaches the final state equal to 0, $r_{s_i} = 0$. This (neutral) value would allow the agent to move around any states with reward = 0 and find any path that eventually leads to the final state. This means that we do not care how many steps the agent takes as long as it finds the goal state (i.e. we do not penalise taking more steps to get to the goal).
- Reward for reaching goal state s_G equal to 1, as the positive reward incentivises the agent to always take the action that leads to the goal state.
- Reward for reaching other terminal states $s_T \neq s_G$ equal to -1, as the negative reward incentivises the agent to avoid non-goal terminal states, since ending up in these states means that the maze has failed (terminated the maze without reaching the goal state).

$$R_{s_t} = \begin{cases} 1 & \text{if } s_t = s_G \text{ terminal goal state} \\ 0 & \text{if } s_t \text{ nonterminal state} \\ -1 & \text{if } s_t = s_T \neq s_G \text{ terminal non-goal state} \end{cases}$$

Q2) MONTE CARLO LEARNING

a) I used an On-Policy First-Visit MC Model with ε-greedy policy update and epsilon decay (ϵ-decay). The $\gamma = 0.92$ and $p = 0.86$ parameters are both automatically calculated from personal CID (see Q1a). I set $\epsilon = 0.4$ as the epsilon parameter for the greedy policy as it gave the best agent performance (quicker convergence) out of all the values tested, including 0.1, 0.2, and 0.4 (Figure 2ci).

I implemented epsilon decay (ϵ-decay) as it allowed the agent to better estimate the values with less variance and set the ϵ-decay rate to 0.9995 as it yielded overall the best performance compared to the other tested values (0.995 and 0.99995).

5000 (on-policy) episodes were initially generated, each with a maximum of 500 steps (as set in the problem description). The agent's learning stabilised at around 300 episodes (Figure 2cii) and was able to learn to solve the maze from any starting state sufficiently well. However, pushing the number of episodes to 5000 resulted in an increased accuracy in state values for more remote regions of the maze but increased the average running time as well (from 3 to 10+ seconds), and although the values improved, the final policy did not improve significantly. Thus to accomodate for shorter run times the number of episodes was kept to 1000.

b) The MC agent with final parameters described in Q2.a), I achieved the following optimal policy and value functions:

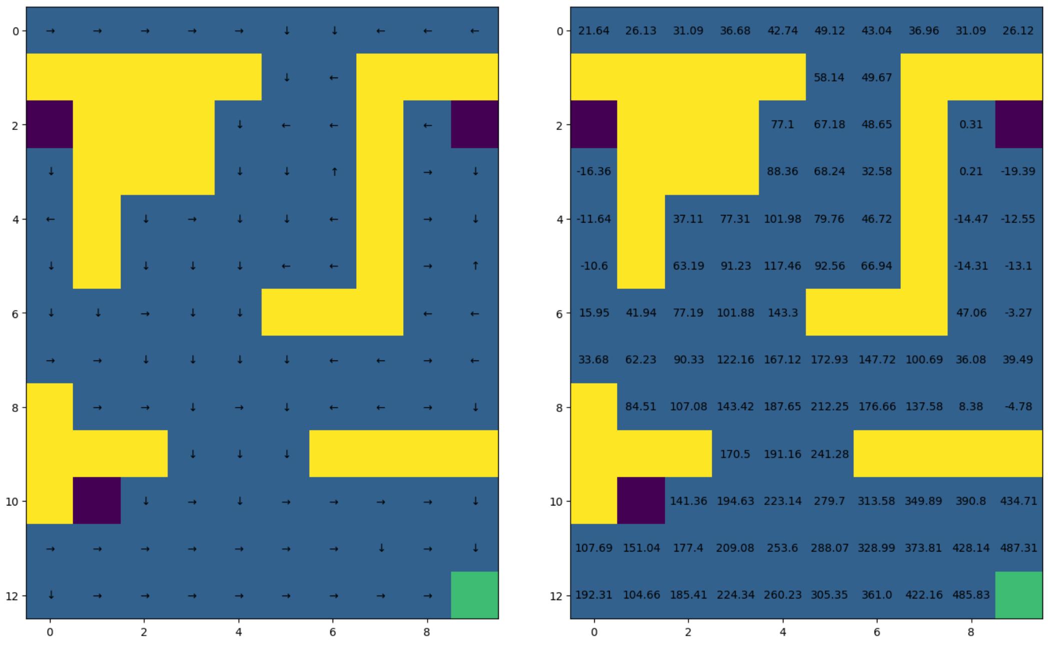
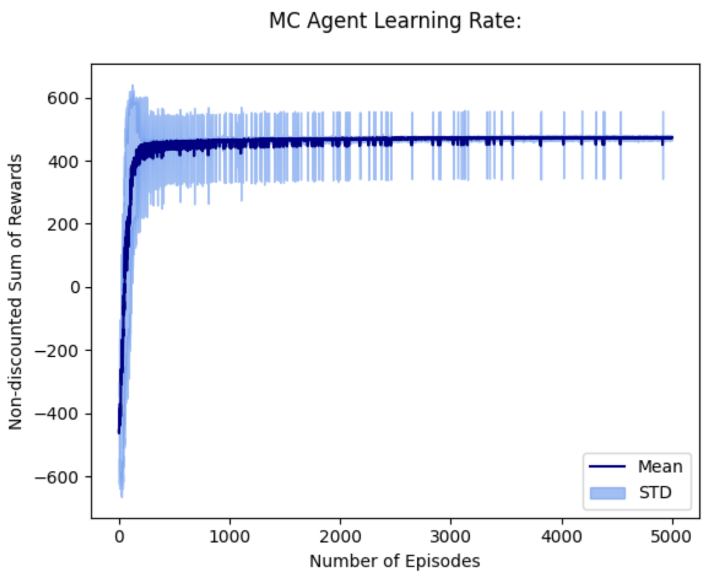


Figure 2b: MC Policy (Left) and Values (Right) with $\gamma = 0.92$, $p = 0.86$, $\epsilon = 0.4$ and ϵ-decay $= 0.9995$

c) The MC agent with final parameters described in Q2.a), I achieved the following learning rate across 25 training runs:



MC with $\epsilon = 0.4$ and ϵ-decay $= 0.9995$ across 25 training runs

The following plots show the learning rates of the MC agent with different parameters. For the same minimal amount of episodes (1000), setting $\epsilon = 0.4$ resulted in a lower variance and quicker convergence, while lower values of epsilon (eg. $\epsilon = 0.2$) resulted in a higher deviation and slower convergence (Figure 2ci).

Increasing the number of episodes from 1000 to 5000 had no further impact on the learning rate and variance (Figure 2cii), so the episodes were capped to 1000 for quicker run times although there was a tangible improvement in the value function for more remote states.

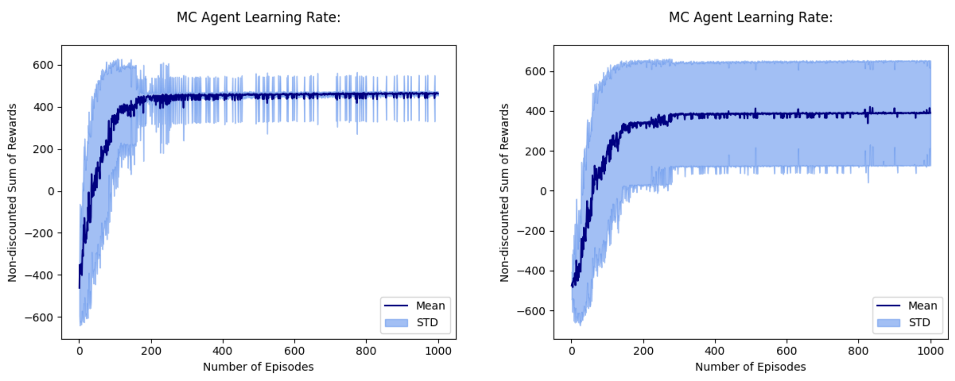


Figure 2ci: MC Learning rate with $\epsilon = 0.4$ (Left) and $\epsilon = 0.2$ (Right) over 25 runs

Lastly, ϵ-decay was implemented as it reduced the standard deviation of the learning rate and led to a quickly converging policy and better overall values (Figure 2cii).

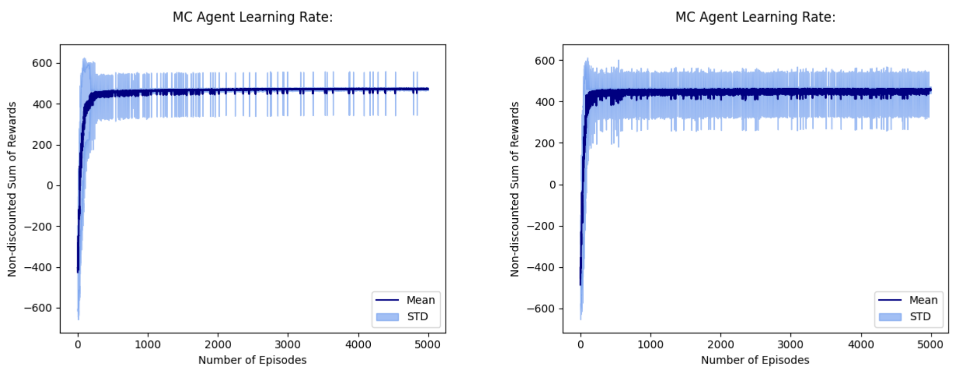


Figure 2cii: MC Learning rate for 5000 episodes with (Left) and without (Right) ϵ-decay $= 0.9995$ over 25 runs

d) Limiting the amount of steps that an agent can take before an episode is considered 'over' can help the agent understand 'how long' it should take to find the optimal path. Suppose we expose the agent to a set of maze walks $s_0, a_0, r_1, s_1, \ldots, s_k$ (from any starting state s_0):

- If (some) of these don't end in a terminal state before *(eg.)* 500 steps, the agent will never collect the highest (500 for goal state) or lowest (-50 for other terminal states) rewards. This signals to the agent that it needs to search for a better path from the same start state s_0 that *does* reach a terminal state, since it has been exposed to other (non-truncated) episodes that do reach a terminal state within the step limit. In this case the agent can more quickly learn the optimal policy that allows it to reach the terminal states (from any start state) in less that 500 steps and thus the Value function will converge faster to the expected optimal value.
- If all episodes end in a terminal state, then the agent is only exposed to trajectories that end either in failure (reach terminal state that is not the goal state) or sucess (reach goal state). i.e. the agent always finds its way to the terminal state, even if it takes longer routes. In this second case, it will take more time for the agent to find the optimal route because it will spend time (mainly in the begininning) exploring longer trajectories that the agent in the first case will never experience (i.e. with 500+ steps). The value function will *still* converge to the

optimal value, but it would take longer than in the first scenario and the agent will need to learn from more episodes than in the first case.

Q3) TEMPORAL DIFFERENCE LEARNING

a) I implemented Q-Learning with ε-greedy policy update and epsilon decay (ϵ-decay). The $\gamma = 0.92$ and $p = 0.86$ parameters are both automatically calculated from personal CID (see Q1a). I set $\epsilon = 0.2$ as the epsilon parameter for the greedy policy as it gave the best agent performance out of all the values tested (0.1, 0.2, 0.4). As for MC, I set the ϵ-decay rate to 0.9995 as it yielded the best result compared to all the tested values. I set the learning rate $\alpha = 0.2$ as it led to an overall better performance when compared to learning rates of 0.01, 0.2 and 0.4. I initially set the total generated episodes to 5000, and the agent's learning seemed to stabilise well before 1000 episodes. The training runs with 5000 episodes yielded value functions that did not significantly enhanced the approximation of the optimal values (found through TD) and drastically slowed down run times (from 16 to over 60 seconds). Thus the episodes were capped to 1000.

b) The TD agent with parameters described in Q2.a), I achieved the following optimal policy and value functions:

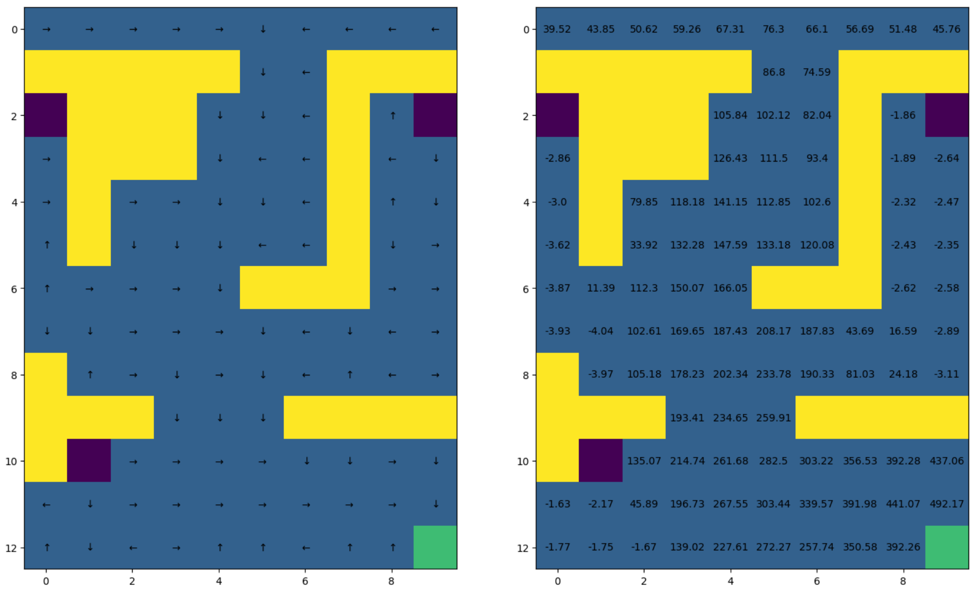


Figure 2b: TD Q-Learning Policy (Left) and Values (Right) for $\alpha = 0.2$, $\epsilon = 0.2$ and ϵ-decay $= 0.9995$

c) The MC agent with final parameters described in Q2.a), I achieved a learning rate across 25 training runs shown in Figure 3c.

The other plots show the learning rates of the TD agent with different parameters. Setting the learning rate $\alpha = 0.2$ led to a quicker conversion of the values with less fluctuations when compared to other values (Figure 3ci, Next Page) for the same number of episodes. Increasing the number of episodes from 1000 to 5000 had no tangible impact on the learning rate and variance (Figure 2cii, Next Page), so the episodes were capped to 1000 for quicker run times. Lastly, the ϵ-greedy policy was set to $\epsilon = 0.2$ as it led to a quicker convergence and less model instability (Figure 2cii, Next Page).

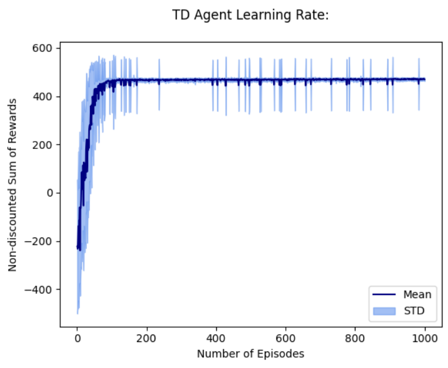


Figure 3c: TD with
$\epsilon = 0.2,\ \alpha = 0.2$

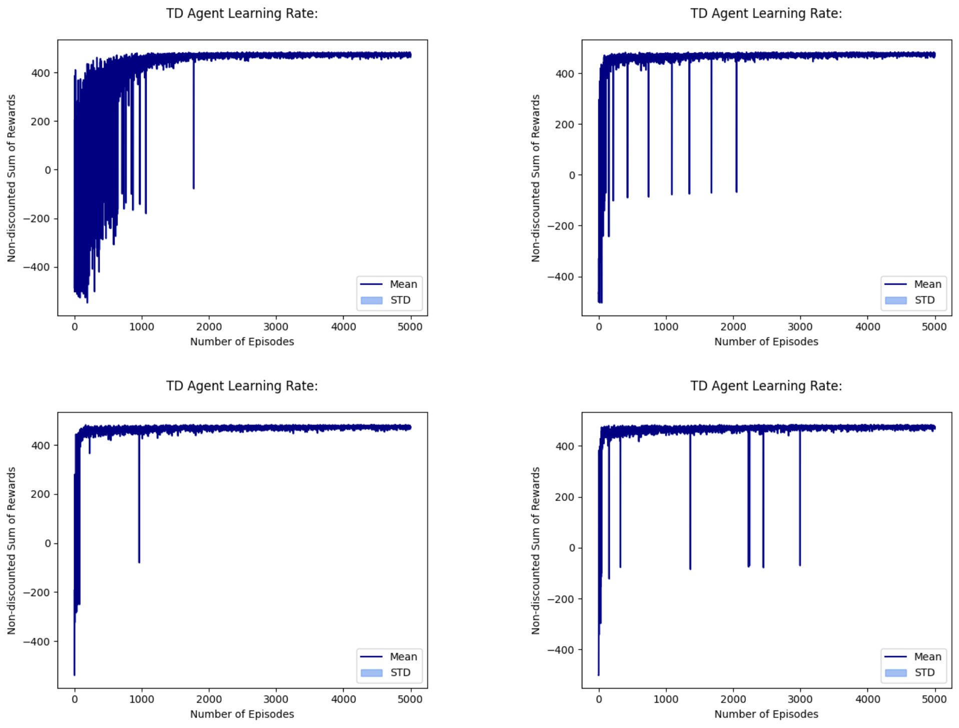


Figure 3ci: TD Learning for different values of α: 0.01 (Top-Left), 0.1 (Top-RIght), 0.2 (Bottom-Left) and 0.3 (Bottom-Right) over 25 runs

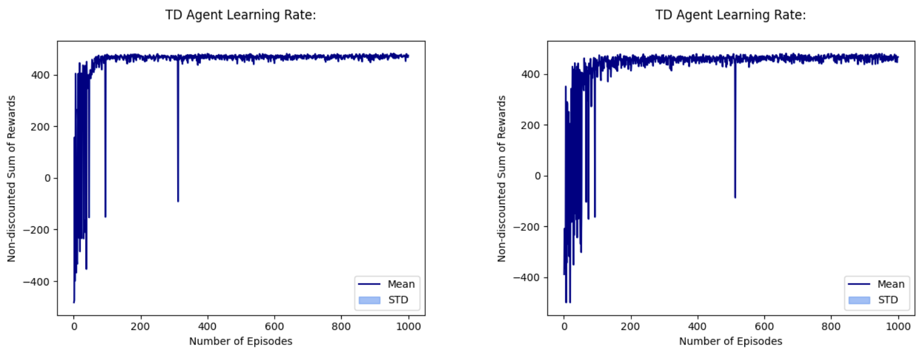


Figure 3cii: TD learning for $\epsilon = 0.2$ (Left) and $\epsilon = 0.4$ (Right) over 25 runs

d) Since an on-policy agent uses only one policy π for both exploring the maze and deciding what the best action to take is from any state, the policy being GLIE is a necessary requirement since π needs to find a balance between exploring different routes (i.e. choosing non-optimal actions mainly in the beginning) and finding the best route (i.e. always choosing the optimal action). With off-policy methods instead, the two actions are separated:

- The behaviour policy π_B is concerned with exploration in order to visit every state in the maze as many times as possible. This policy doesn't necessarily need to be greedy, since it is concerned with exploring and should take non-optimal actions.
- The target policy π_T is concerned with finding the best action at all times in order to to find the optimal route that maximises the return and thus *should* be greedy.

In this case, the Value function will then still converge if neither policy is GLIE *per se*, as long as the overall process is. That is, if the target policy is always greedy and the behaviour policy is allowed to explore the environment infinitely. Note that this setup, however, could slow down learning (when compared to both the behavioural and target policy being GLIE) if the behavioural policy "explores too much", i.e. if often or most times takes non-optimal actions.