Reinforcement Learning - Coursework 1

Lorenzo Stigliano

November 14, 2022

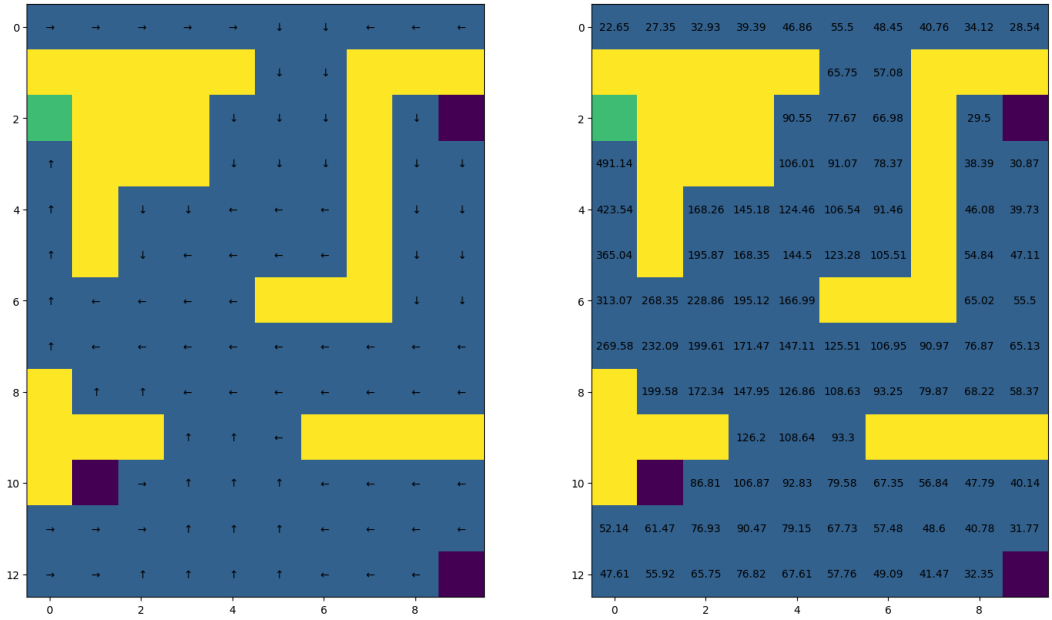
Contents

1 Dynamic Programming

1.1 - We decided to use **value iteration**. After running a series of tests to see the average number of epochs taken and the average time taken over 10 runs. Value iteration took 0.12 seconds and 25 epochs while policy iteration took 1.24 seconds and 275 epochs to converge to an optimal policy.

We also need to consider what threshold value to use evaluating the value function, in this case a fixed threshold of 0.0001. Sufficient for confidence in convergence. Finally, I needed to consider whether to use synchronous vs asynchronous DP methods. Due to our state space being small, we only have 98 possible states, I decided to use synchronous backups since we would need to keep track of 2*number of states for our problem. No need to use asynchronous backups. However, it is important to note that if we didn't have prior knowledge of the state space or the space was big then asynchronous backups would have been beneficial.

1.2 - See Figure 1.



(a) Optimal policy (b) Optimal value function

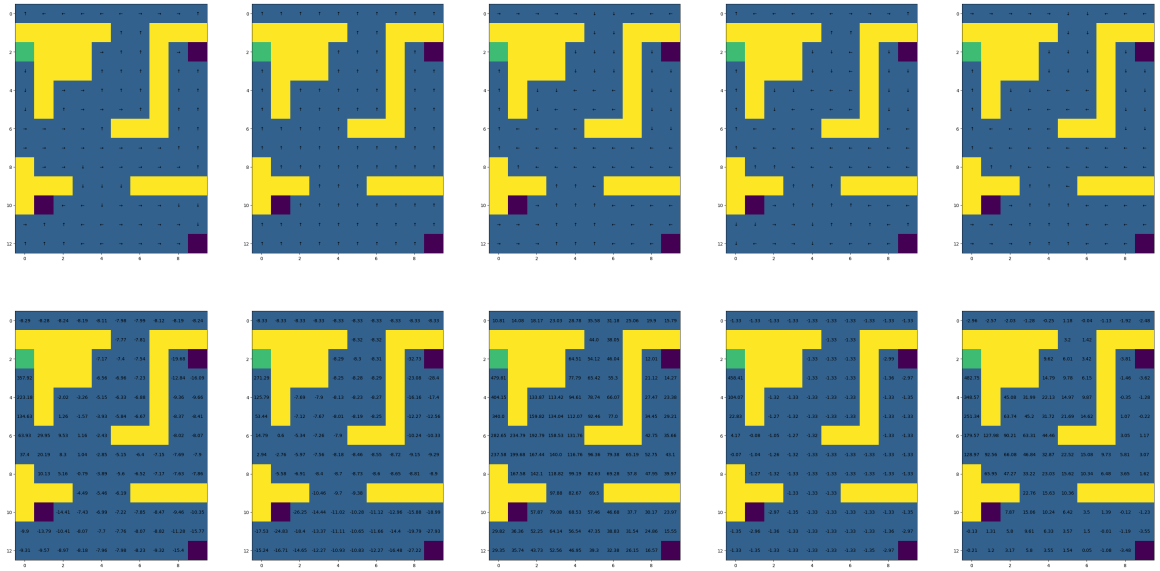
Figure 1: Optimal policy and optimal value function for value iteration.

1.3 - Lets look at the effects of changing p while keeping γ constant. We have probability p of moving in the desired direction and $\frac{1-p}{3}$ of moving in the other directions. If $p < 0.25$, Figure 2 (a),then our agent will have a higher chance of moving in the directions it doesn't want to move in, as a result the optimal policy will be divergent and move away from the goal state. If $p = 0.25$, Figure 2 (b),then our agent will explore in a random walk, since all the actions that it can take will have equal probability of being chosen. So our the agent will have difficulties moving towards the goal. If $p > 0.25$, Figure 2 (c), our agent has more chance of taking a step towards the direction that maximises $Q(s, a)$ as a result we see that for all of our initial starting points the agent will move towards the goal. Furthermore, $V(s)$ will in behave as expected, as $p \to 1$ the values of the greedy actions will be propagated back to the initial states.

Looking at γ, the discount factor which determines present value of future rewards, while we keep p constant. If γ is close to 0 then our agent is **short-sighted** that is to say it only cares about immediate rewards close to it. Figure 2 (d) for states that are far away from the goal state their values are very similar, all close to -1.3, as we move closer to the goal state the values grow exponentially since we are closer so the rewards are propagated back. The optimal

policy, the agent will have trouble finding an action to take that moves it towards the goal when it is very far away, as seen in the bottom of the state space, however as we move towards the goal the policy moves towards the goal.

Similarly if γ is close to 1 then our agent is **long-sighted** as a result takes into account future rewards more strongly. Looking at Figure 2 (e) the reward from the goal state is propagated back and so the the optimal policy can be found from states which are further away from the goal state, as a result the value function will have higher values for states that are further away since we are not discounting them as much.



(a) $p = 0.10$, $\gamma = 0.88$ (b) $p = 0.25$, $\gamma = 0.88$ (c) $p = 0.80$, $\gamma = 0.88$ (d) $p = 0.90$, $\gamma = 0.25$ (e) $p = 0.90$, $\gamma = 0.75$

Figure 2: Policy and value functions for varying γ and p.

2 Monte-Carlo Reinforcement Learning

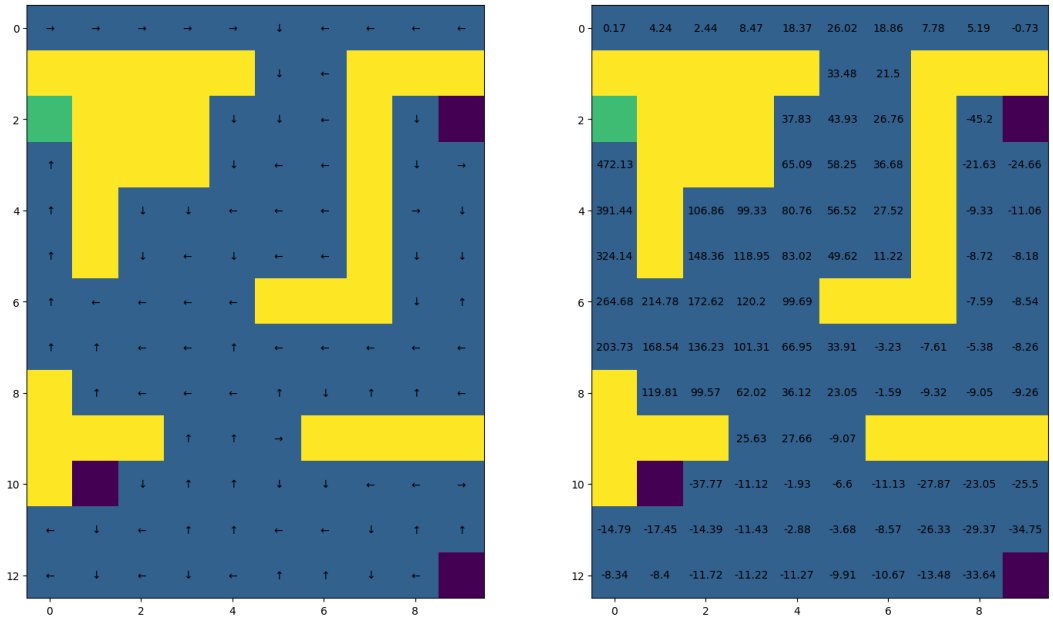
2.1 - The Monte-Carlo (MC) method we implemented has the following properties. We use an **every-visit** method this is because both first and every-visit methods converge to $V(s)$ as the number of visits (or first visits) to s goes to infinity [3]. The MC method uses **online** updates, this allows us to update the agents knowledge as soon as we experience new data, creating a better policy after each episode. However, it is important to note that using batch process reduces the amount noise of per episode. Since if our agent learns from a bad episode and we update online, this introduces variance. Furthermore, we assume the dynamics of the environment are **stationary** since the rewards do not depend on time, so there is no need to *forget old episodes*. So we do not use a fixed α learning rate. Instead $\alpha = \frac{1}{count(s,a)}$ which results in the sample-average method, which is guarantees $V(a)$, $Q(s,a)$ to converge by the law of large numbers [3]. (γ given by environment since specification allowed us to use it.)

The two assumptions that guarantee the convergence of MC methods - episodes have exploring starts and policy evaluation can be done with an infinite number of episodes. To take care of the second assumption we have ensured the number of episodes, in this case 5000 episodes, is large enough to ensure convergence, as seen in Figure 5 (a). To satisfy the exploring states assumption I use an **ϵ-greedy policy**. Using this policy it allows us to ensure that all states can be visited an arbitrary number of times. We implemented a decaying ϵ to ensure *GLIE*. is updated using after each episode $\epsilon = \epsilon(1 - \frac{n}{\#episodes})$ where n is the current episode. We know that when n = number of episodes then $\epsilon = 0$. As a result, we are ensured that algorithm converges [2]. Furthermore, this increase performance over a fixed ϵ for this problem.

Furthermore, we use an **on-policy** MC method. Since we are using an **ϵ-greedy policy** we can have the benefits of a greedy policy without having to drop the assumption of exploring starts. Furthermore, since the state space is small and discrete there isn't a need to use an off-policy method since the goal is find a way to reach a certain state. Another reason I have decided not to use off-policy MC is because they only learn from the tails of episodes after the last non greedy action [3]. In turn, this could slow learning.

We chose ϵ by running tests on a range of different parameters Figure 5 (a), we use $\epsilon = 0.4$ which allows for a good balance between exploitation and exploration, while having rapid convergence to the highest total non-discounted sum of rewards within 5000 episodes. It also offers more stability and exploration than $\epsilon = 0.2$.

2.2 - See Figure 3. **2.4** - See Figure 4.



(a) Optimal policy

(b) Optimal value function

Figure 3: Optimal policy and optimal value function for MC method.

2.3 - Variability in MC methods arises in the creation of episodes and how we update $V(s)$. They come from the environment and the ϵ-greedy policy. At each state is $\frac{1-p}{3}$ probability that the agent takes another different action then the desired one. Thus, there is a chance of creating different paths for the same action at the same state. Furthermore, since we use a ϵ-greedy policy, there is a chance of taking a non greedy action with probability $\frac{\epsilon}{|A(s)|}$ resulting in, again, different paths. Looking at the backup equations $V(s) \leftarrow V(s) + \alpha(G_t - V(s))$ we know that $G_t = \sum_{k=t}^{T-1} \gamma^{k-t} R_{k+1}$, this is where most of the variance is introduced, since this will be different for a range of episodes.

I will make the **assumption** that the total non-discounted sum of rewards at the end of a run are identically and independent normally distributed random variables. The margin error is given by $E = z_{\alpha/2}\frac{\sigma}{\sqrt{n}}$ therefore the number of replications is given by $n = (z_{\alpha/2}\frac{\sigma}{E})^2$. $z_{\alpha/2}$ is z critical value, we use a 95% confidence ($z_{0.025} = 1.96$). We would like to have a margin error of ± 2.5 so $E = 2.5$. To estimate σ, the population standard deviation, we estimated σ on the total non-discounted sum of rewards at the end of 10 runs, $\sigma \approx 11.65$. So, n (number of replications) $= (z_{\alpha/2}\frac{\sigma}{E})^2 = (1.96 * \frac{8.90}{2.5})^2 \approx 25$.

2.5 - The ϵ dictates how greedy our policy is, this means that all non-greedy actions are given the minimal probability of selection namely $\frac{\epsilon}{|A(s)|}$. If $\epsilon = 0$ then our agent will follow a deterministic greedy policy, as we increase ϵ we increase exploration while decreasing exploitation.

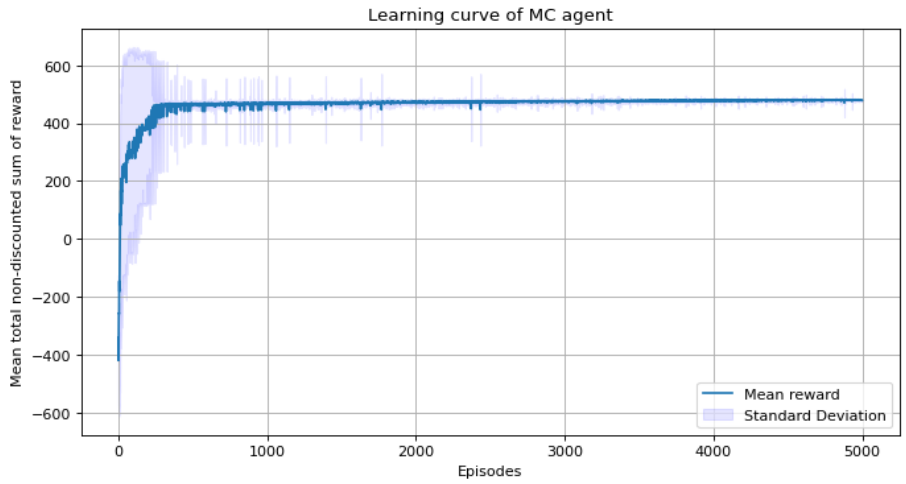
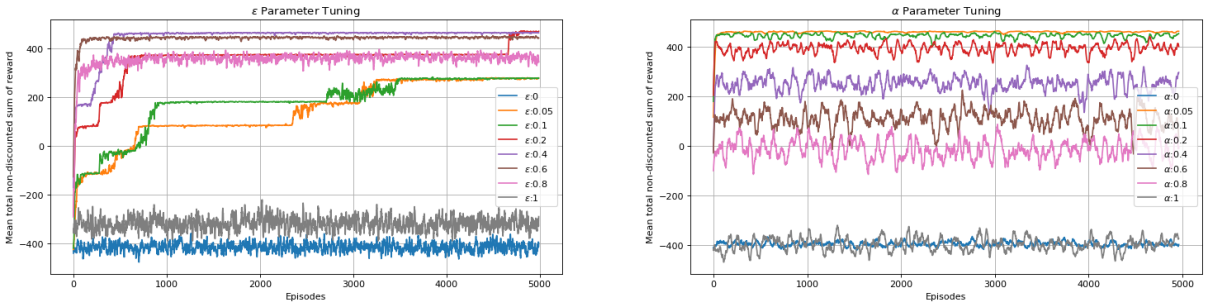


Figure 4: Total non-discounted sum of reward against the number of episodes for MC method described in 2.1 for number of replications = 25.



(a) ϵ tuning while keeping α fixed.

(b) α tuning while keeping ϵ fixed (smoothed)

Figure 5: Learning curves used for hyper-parameter tuning for α and ϵ over 5000 iterations of the MC method described in 2.1.

Ideally, we would like to find a balance between the two. As a result, we should expect to see that as ϵ increases our model should be able to find the goal state since it is more exploratory and in turn have higher total non-discounted sum of rewards. Looking at Figure 5 (a) we can see that as when $\epsilon = 0$ the reward is around -500 this means that the agent struggles to explore the space and so is ultimately stuck in a sub-optimal greedy policy. As ϵ increases we can see that average reward increase since we can explore the state space allowing it to converge onto a maximal reward for this problem. However, when $\epsilon > 0.5$ the agent is very exploratory thus struggles to find a solution as seen by the grey and pink lines.

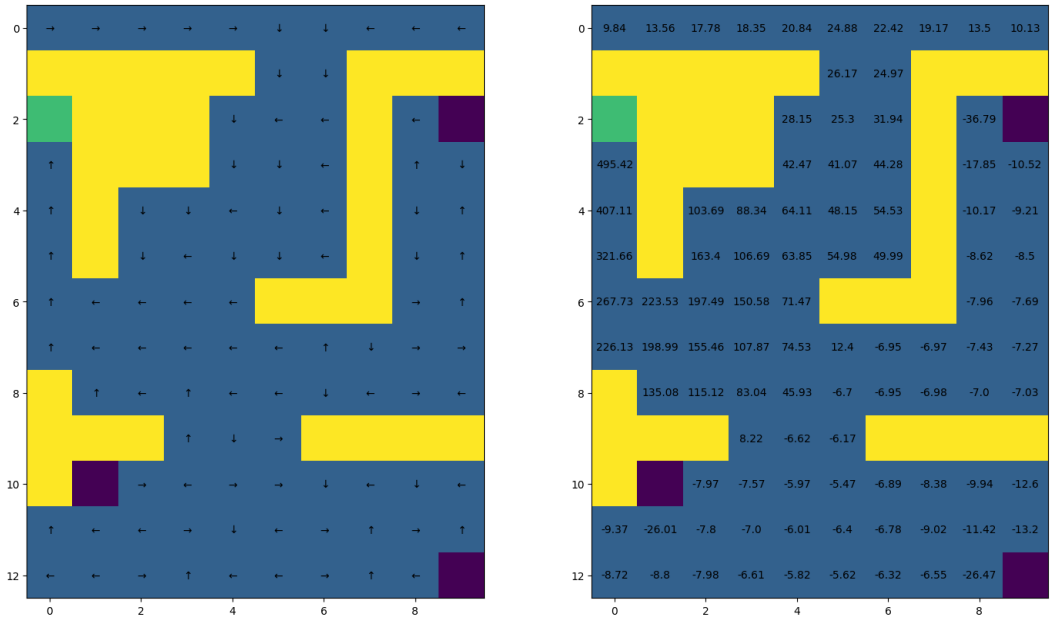
Let's explore the learning rate α, we have modified our algorithm such that we use the following update rule $Q(s_t, a_t) = Q(s_t, a_t) + \alpha(G_t - Q(s_t, a_t))$ instead of the empirical mean. Notice that the update rule can be written as $\alpha G_t + (1 - \alpha)Q(s_t, a_t)$, where G_t is the expected discounted return. So α finds the balance between G_t and $Q(s_t, a_t)$, if $\alpha = 0$ there is no learning since $Q(s_t, a_t) = Q(s_t, a_t) + 0(G_t - Q(s_t, a_t)) = Q(s_t, a_t)$. As α is increases then G_t has more weight, thus learning from the episode, thus learning from the underlying policy which made it. We are prone to variance since we learn online, suppose that the first episodes moves in an unwanted direction the algorithm will struggle to converge. From Figure 5 (b) the blue and grey lies at the bottom, when $\alpha = 0$ and 1 the agent does not learn well. We can see that the maximal amount of reward was found for small values of α (0.05 and 0.1) and then gradually decreases as α increases. This means that the agent tries to use a higher proportion of G_t which gives new information from the episode, while remembering a small proportion of $Q(s, a)$.

3 Temporal Difference Reinforcement Learning

3.1 - We had to choose between SARSA and Q-Learning methods. We decided to use **Q-Learning**. In Q-learning (**off-policy** method) considers an behaviour policy and a deterministic target policy for our problem. As a result, we benefit from both an exploratory behaviour policy and exploitation in the target policy which will hopefully help find the solution to our problem quicker. We use an **ϵ-greedy policy without decay** for our behaviour policy. The reason for this is, as we shall see, we use a small ϵ, as a result there is no need to decay it. Furthermore, we also used a **fixed learning rate** α we know from [1] that Q-learning converges if the learning rate α_t has Robbins-Munro conditions. However, this holds true if we do an infinite number of episodes and there is **no guarantee** on the time taken. As a result, since we cannot satisfy the condition of infinite number of episodes and we have no time guarantees, we have decided to fix α. Furthermore, when implementing α_t satisfying the Robbins-Munro conditions, the performance over a fixed number of iterations decreased. We use a greedy deterministic policy, $\pi(s, a) = max_a Q(s, a)$, as the target policy. (γ given by environment since specification allowed us to use it.)

In order to tune the values of ϵ and α we use the results from Figure 8. We can see that when $0.05 \leq \epsilon \leq 0.2$ there is fast convergence to the greatest reward, we use 0.2 since it will allow for more exploration (we do not use $\epsilon = 0$ because behaviour policy needs to be ϵ-greedy). For α we use 0.4 even though 0.6 has faster performance, we chose 0.4 since it will give more weight to $Q(s, a)$, learning from previous episodes. Furthermore, we use 1000 iterations, since we can see that after 200 iterations the algorithm has converged.

3.2 - See Figure 6. **3.3** - See Figure 7.



(a) Optimal policy (b) Optimal value function

Figure 6: Optimal policy and optimal value function for TD method.

3.4 - Similarly to 2.5, ϵ dictates how greedy our policy is, all non-greedy actions are given the minimal probability of selection namely $\frac{\epsilon}{|A(s)|}$. If $\epsilon = 0$ then we have a deterministic target policy increasing ϵ increases exploration, when $\epsilon = 1$ the policy is a random walk. From Figure 8 we can see that the mean reward does not increase and in fact stuck at below -400. For our agent when $0.05 \leq \epsilon \leq 0.2$ we have the fast convergence, this follows from the theory since we are using a greedy policy which in turn reduces exploration and increase exploitation. We can

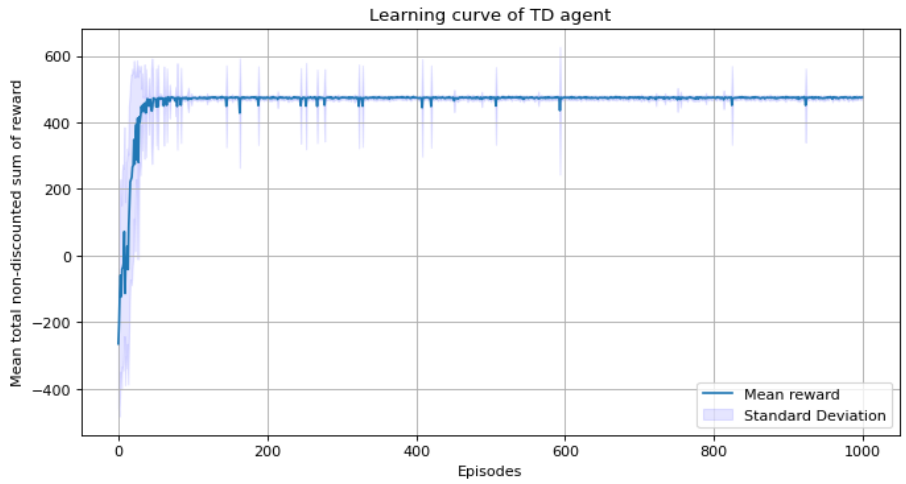
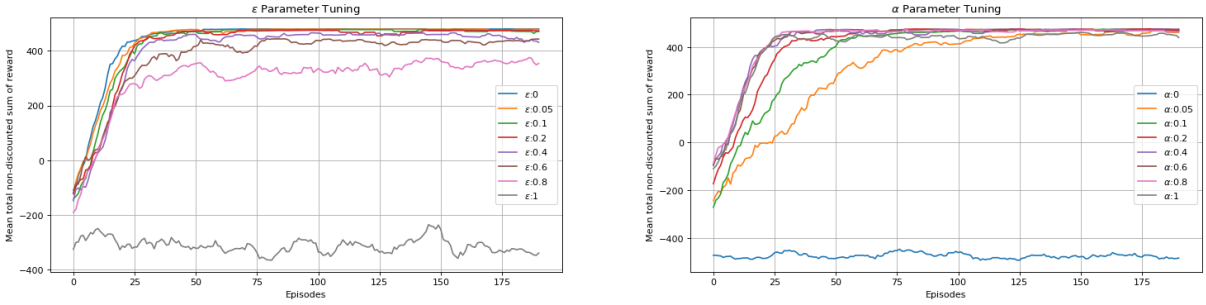


Figure 7: Total non-discounted sum of reward against the number of episodes for TD method described in 3.1 for number of replications = 25.

see that as $\epsilon > 0.2$ the agent struggles to converge to a high mean and the convergence rate decrease, since we are exploring the space.

Likewise, α finds the balance between $Q(S, A)$ and $R + \gamma(max_a Q(S', a))$. So if $\alpha = 0$ then the update rule will be $Q(S, A) = Q(S, A)$ so agent does not learn as seen in Figure 8 (b) blue line. If $\alpha = 1$ then the update rule will be $Q(S, A) = R + \gamma(max_a Q(S', a))$ as a result, the update will be greedy, so when we create the new episode with our behaviour policy we will move towards the optimal goal as seen in Figure 8 (b) grey line. For this environment this seems to work well since the state space is small and the goal state is close to the initial positions, however, if the space was larger and the goal state was further away using a small learning rate would be detrimental, since we would like to learn from past experience.



(a) ϵ tuning while keeping α fixed.

(b) α tuning while keeping ϵ fixed.

Figure 8: Learning curves used for hyper-parameter tuning for α and ϵ for 1000 iterations of TD method described in 3.1.

4 Comparison of Learners

4.1 - See Figure 9. **4.3** - See Figure 10.

4.2 - From Figure 9 we can see that the rate of decay at which of the TD method is greater than that of the MC method. One reason for this that TD methods exploit the Markov property. Since our environment is Markovian we would expect TD to be more efficient and thus explaining the very steep learning curve. Furthermore, since we are using a *GLIE* ϵ for MC we know it should converge after an infinite number of episodes, but since we are only doing 5000 episodes, it does not converge to the optimal $V(S)$, but does find an optimal policy from

any of the starting states. These two reason explain why TD out preforms MC in a limited number of iterations, in the limit we would expect them to converge (if α_t has Robbins-Munro conditions [1])

Furthermore, we can see that MC has much larger variance than the TD method, which is consistent with theory. Looking at the back up for MC - $V(s) \leftarrow V(s) + \alpha(G_t - V(s))$ we know that $G_t = \sum_{k=t}^{T-1} \gamma^{k-t} R_{k+1}$, this is where most of the variance is introduced, since $V(s)$ depends on many rewards. In contrast the back up for TD is given by: $V(s) \leftarrow V(s) + \alpha(R_t - V(s') + V(s))$. As a result, less variance is introduced since it only depends on the immediate reward, R_t. It is worth noting that in theory, TD is a biased estimator of $V(s)$ while MC is unbiased, however, for TD as the number of episodes increases in converges to $V(s)$. This does not seem to pose a problem in our environment.

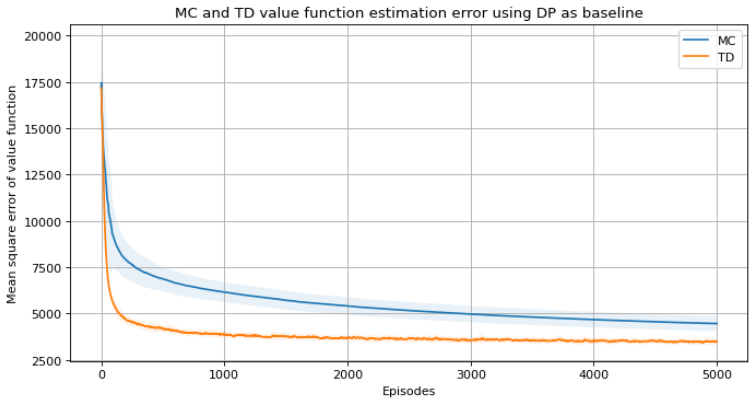


Figure 9: Comparison of the estimation error for $V(s)$ for TD and MC learners with DP as the baseline.

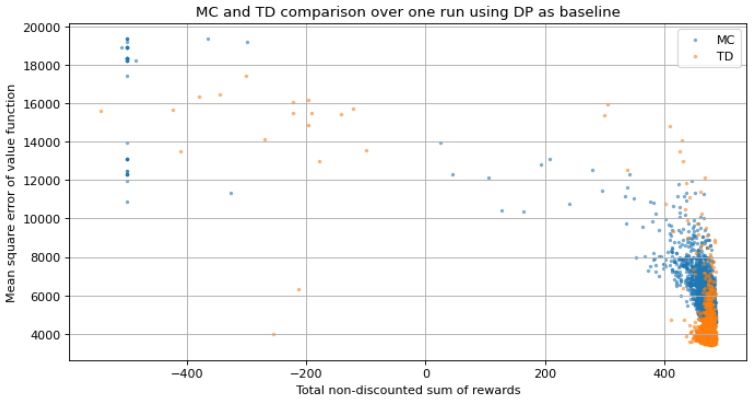


Figure 10: Comparison over one episode between reward and value function estimation error.

4.4 - This plot shows us trade-off between finding an optimal value function a good reward for our problem. We can see that both the MC and TD converge to an good reward but on a range of different values ranging from 3000 to 8000, showing the variance of these methods. This suggests that TD finds a better approximation to $V(s)$ than MC while finding an optimal policy as good as the one found by MC. MC has worse estimates for states that are not in the optimal policy path and as a result a larger variance on $V(s)$ since the points are more spread. If we look at backup for MC, $V(s)$ is updated with $G_t = \sum_{k=t}^{T-1} \gamma^{k-t} R_{k+1}$ if all states visited after the current one have negative reward $V(s)$ will take on a very negative value, which is good for the agent such to not to visit this state anymore. But to find an optimal policy, the value of $V(s)$ will be poor estimation. In contrast, TD backup uses $R_t - V(s')$ and so only depends on the current reward and the next states value-function, making it more robust to bad episodes.

Notice that the value function error is not 0 since our agents have been tuned to find a policy from any of the initial states to reach the goal using greedy and ϵ-greedy policies. Once they reach the goal they fail to explore the rest of the space, states with negative rewards and surroundings are not visited frequently, thus resulting in a poor estimation of $V(s)$. This raises the question on finding a good error function to compare the methods on the same problem, depending on what is the *goal* of our agent is. If we wanted to reduce the current function error then we would increase ϵ to encourage exploration, thus approximating $V(s)$ better. In turn, different error functions can be used to compare different agent behavior.

References

[1] E. Even-Dar and Y. Mansour, *Convergence of optimistic and incremental q-learning*, Advances in neural information processing systems, 14 (2001).

[2] D. Silver, *Lecture 5: Model-free control*, UCL, Computer Sci. Dep. Reinf Learn. Lect., (2015), pp. 101–140.

[3] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, MIT press, 2018.