IMPERIAL COLLEGE LONDON

Coursework 1
Reinforcement Learning

Livia Soro - CID: 01549029
November 8, 2021

Question 1: Dynamic Programming

Throughout this report we used $p_{test} = 0.94$ and $\gamma_{test} = 0.84$. The absorbing state with reward 500 is R_1.

1. The value function and optimal policy were computed using the value iteration algorithm, after comparing its performance against the policy iteration algorithm.

(a) **Choice of algorithm:** In *policy iteration*, we obtain a sequence of monotonically improving policies and value functions. One drawback of this method is that each iteration involves a full policy evaluation, which in addition can require multiple sweeps through the state set. This makes the algorithm computationally expensive, thus requiring 8.78s to converge and 194 epochs, for the parameters set. In *value iteration*, the policy evaluation step is stopped after one sweep of the value function. This is equivalent to turning the Bellman Optimality Equation into an update rule, thus by the contraction mapping theorem, convergence to the optimal value function and policy is still guaranteed. In this case the algorithm converged in 0.99s and 22 epochs, for the same set parameters. Hence, value iteration was chosen to solve the problem.

(b) **Threshold:** The threshold value reflects a trade-off between computational cost and accuracy. Lower thresholds yield a more accurate value function but converge in a higher number of epochs. Higher thresholds converge faster but yield a less accurate result. A range of values between 0.00001 and 0.5 were tested. We found that 0.01 showed the best balance between speed of convergence and accuracy of result.

(c) **Deterministic policy:** The implementation assumed a deterministic action for each state. However, while the value function converges to a unique solution, there could be multiple optimal policies. This means that for a given state, multiple actions could be equivalently optimal.

2. Graphical representation of the optimal policy and optimal value function:

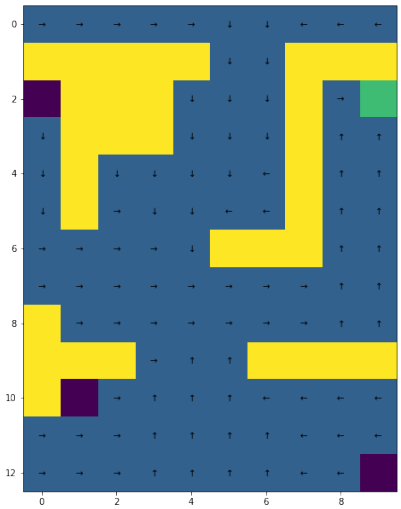

Figure 1: Optimal Policy

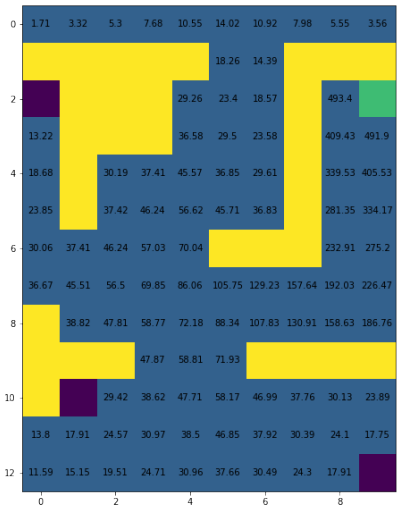

Figure 2: Optimal Value Function

3. Exploring the influence of γ and p:

(a) **Discount factor:** When computing the value of the current state, the discount factor $\gamma \in [0,1]$ is the factor that controls the weight of future rewards.

For $\gamma < 0.5$, evaluation is myopic, the value of each state will resemble the rewards assigned to states in close proximity. This was observed in the grid, where the values of the states would tend toward the immediate reward values (-1, -50, +500).

Inversely, when $\gamma > 0.5$, evaluation is increasingly far-sighted. This means further future rewards get propagated back more, as seen in Fig. 2. High values of γ, such as $\gamma_{test} = 0.84$, ensure the agent is informed early on about the direction of the goal state. Note however that having $\gamma = 1$ can be harmful, as rewards would decay linearly instead of exponentially when propagating back in the trace. Given the magnitude of the cost of each step (-1) compared to the goal reward (500), an exponential decay is key to obtain a clearly identifiable value function.

(b) **Probability of success:** A chosen action has a probability p to succeed and lead to the expected direction.

For $p < 0.25$, the agent has a higher probability of doing any other action rather than the chosen one. The optimal policy will converge to one where actions lead away from the goal state and towards the absorbing states with negative reward.

For $p = 0.25$, the probability of doing the chosen action is exactly the same as that of doing any other action. The optimal policy will converge to a random policy.

For $p > 0.25$, the agent will most likely do the chosen action. The optimal policy will hence reflect the agent's choices, as seen for $p_{test} = 0.94$ in Fig. 1.

Question 2: Monte-Carlo Reinforcement Learning

1. Monte Carlo (MC) methods learn directly from experience. MC policy evaluation uses the *empirical mean* return computed from sample traces instead of the *expected* return, which is used when the environment dynamics are known.

(a) **MC evaluation:** Two types of policy evaluation steps were investigated. In the *first-visit* MC, the total discounted return for each state-action pair is updated only for the first time the pair is encountered. In *every-visit* MC, every time the state-action pair is encountered is accounted for. First-visit MC was chosen for policy evaluation as more variability was observed with every-visit MC. This could be due to the values of the states being more sensitive to the rewards of the absorbing states. For example, if a state is visited four times, the absorbing state's reward will be accounted for four times in the update.

(b) **MC Control:** ϵ-greedy policy improvement with Greedy in the Limit with Infinite Exploration (GLIE) was used. ϵ-greedy policies have $\pi(s,a) > 0 \forall s \in S, \forall a \in A$ which ensures that the agent also does exploration (in contrast with greedy policies where the agent only does exploitation). With GLIE, all state-action pairs are explored infinitely many times in the limit, but the exploration parameter is progressively reduced so that the policy converges to a greedy policy. We chose to reduce ϵ exponentially, $\epsilon = e^{\frac{-n_{trace}}{500}}$, as it allowed the agent to do sufficient exploration initially leading to better performance in the long term.

2. Graphical representation of the optimal policy and optimal value function:

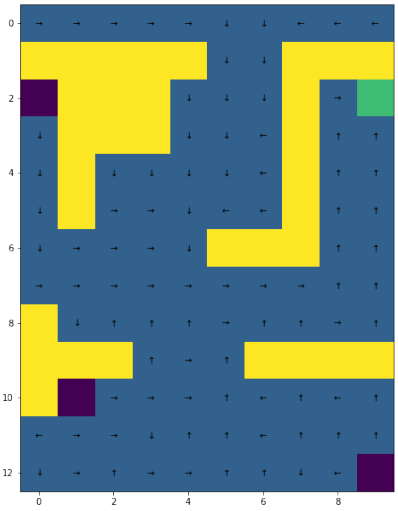

Figure 3: Optimal Policy

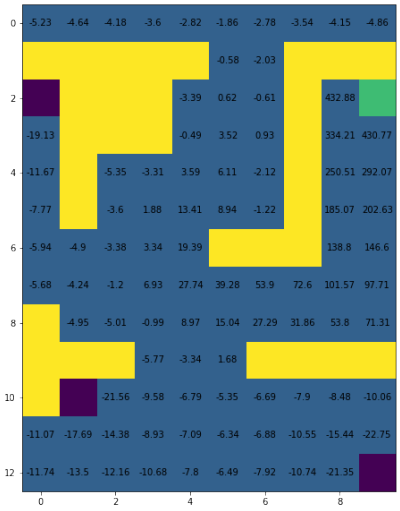

Figure 4: Optimal Value Function

3. A first source of variability is the random initialisation of the state-action value function Q. Secondly, the starting state is chosen randomly in each episode. Thirdly, we introduce randomness with the ϵ, which controls the exploration-exploitation trade-off when choosing an action. Lastly, the chosen action will fail with probability 1-p.

The variance of the mean of a quantity of interest over independent runs is the variance for a single run over the number of runs. To assess a "sufficient" number of replications, MC agent was run ten times for 2000 episodes each. The variance of the total non-discounted sum of reward of the final episode was about 16. Hence, 20 runs would be sufficient to get a variance of the performance of the MC-agent below 1.

4. Learning curve of the MC-agent:

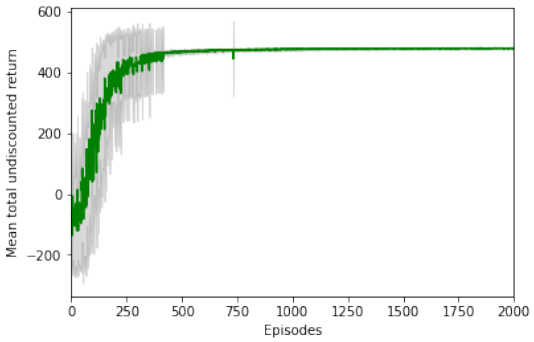

Figure 5: Learning curve of the MC-agent as the mean of the total non-discounted sum of rewards across 20 runs (green) with its corresponding standard deviation (grey).

5. Exploring the influence of α and ϵ:

(a) **Learning rate:** α controls to how extent newly acquired information overrides old information, namely the Q update magnitude. As shown in Fig. 6, bigger α values (0.2, 0.5) lead to unstable learning curves and high standard deviations across

episodes. Small α values (0.001) lead to slow convergence, meaning the agent is not learning enough from updates. For stationary problems α can be replaced by $\frac{1}{N(s,a)}$, where N(s,a) is a counter for each state-action pair. In our case, this performs well.

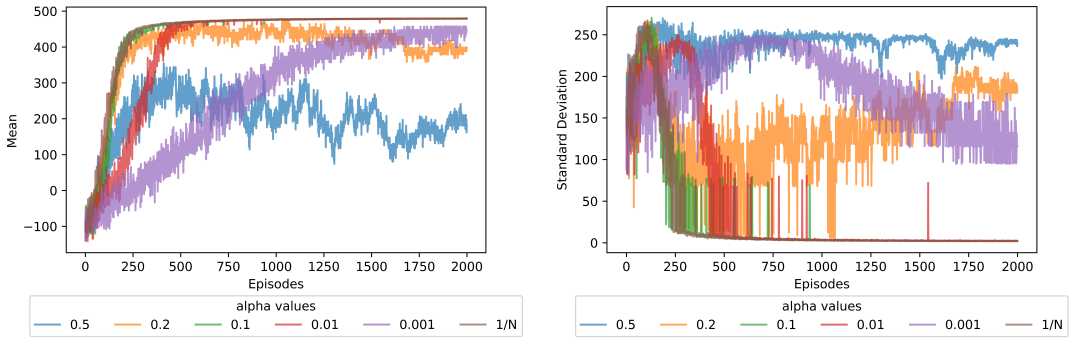

Figure 6: Left: Learning curves of the MC-agent with different values of α. **Right:** Corresponding standard deviation

(b) **Exploration parameter:** ϵ controls the exploration-exploitation trade-off when choosing an action. Firstly, fixed ϵ values across episodes were investigated. As shown in Fig. 7, low ϵ values (0.1, 0.2) converge slowly. This is because the agent privileges exploitation without having enough knowledge on the environment. For higher fixed ϵ values (0.3, 0.4), convergence is faster. However, as the agent maintains high exploration throughout the run, variability spikes frequently even in later episodes (when the goal is not reached). For this reason, we explored algorithms where $\epsilon = e^{-\frac{\tau}{\lambda}}$ decays exponentially, with τ the episode number. Faster decay rates ($\lambda = 100, 250$) focus almost exclusively on exploitation after only limited exploration. The decay with $\lambda = 500$ had the best balance between speed of convergence and variability.

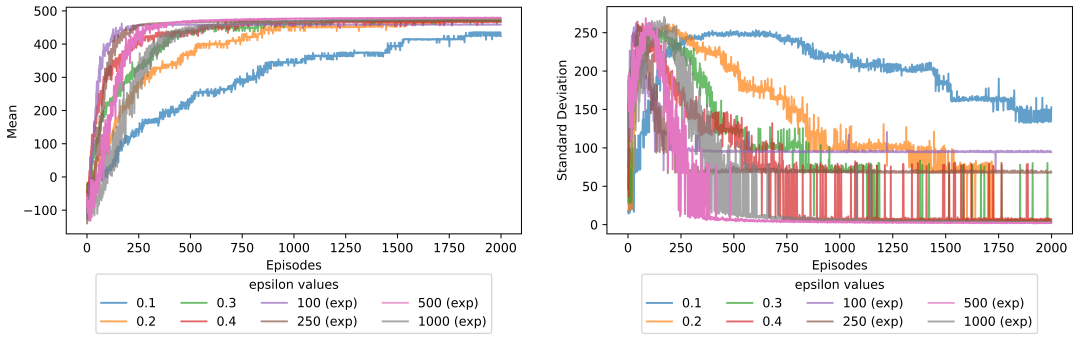

Figure 7: Left: Learning curves of the MC-agent with different values of ϵ **Right:** Corresponding standard deviation

Question 3: Temporal Difference Reinforcement Learning

1. Temporal Difference (TD) methods also learn from experience.

(a) **TD evaluation:** TD learns from incomplete episodes by bootstrapping. This means that instead of updating the state-value function towards the *actual* return, the updates are done towards the *estimated* return, also known as the TD target.

(b) **TD control** GLIE policy improvement with exponential decay of ϵ was kept. The performances of on-policy (SARSA) and off-policy (Q-learning) TD algorithms were compared. On-policy methods attempt to evaluate or improve the policy that is used to make decisions, whereas off-policy methods evaluate or improve a policy different from that used to generate the data[1]. In this case, the two algorithms performed very similarly both in term of speed of convergence and of variability. This might be because there are no negative absorbing states on the path to the goal state, which would penalise SARSA. More specifically, SARSA accounts for possible penalties from exploratory steps, hence it might avoid "dangerous paths" at first, whereas Q-learning disregards these penalties by updating Q towards the greedy action. As performances were comparable, SARSA will be taken into consideration for the following questions.

2. Graphical representation of the optimal policy and optimal value function:

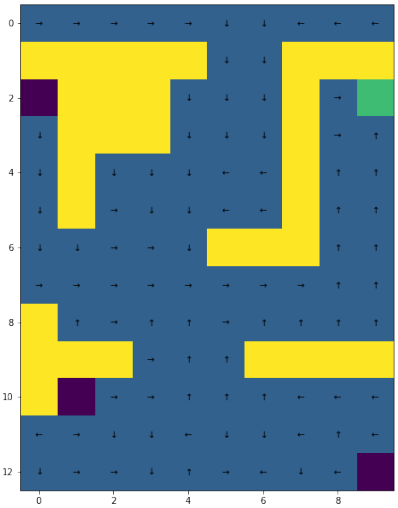

Figure 8: Optimal Policy

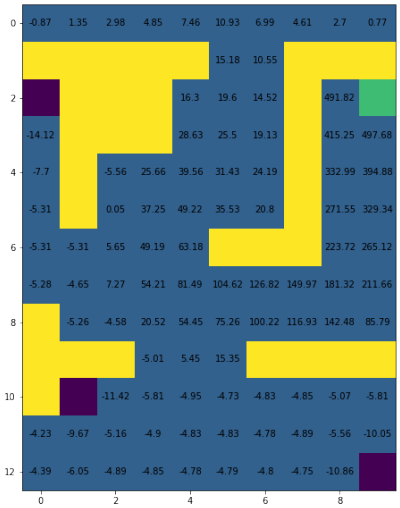

Figure 9: Optimal Value Function

3. Learning curve of the TD-agent:

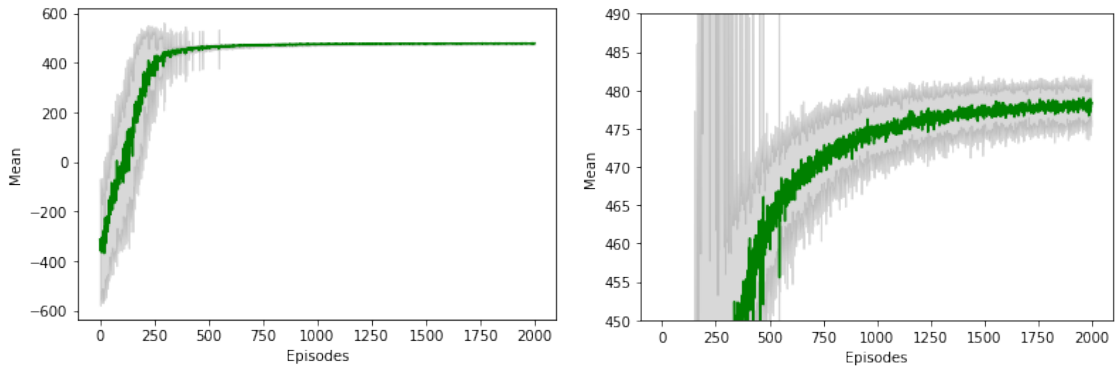

Figure 10: Left: Learning curve of the TD-agent as the mean of the total non-discounted sum of rewards across 20 runs (green) with its corresponding standard deviation (grey) **Right:** Zoom of the learning curve in the range [450, 490]

[1]Sutton, R., Bach, F. and Barto, A., 2018. Reinforcement Learning. Massachusetts: MIT Press Ltd.

4. Exploring the influence of α and ϵ:

(a) **Learning rate:** The impact of α can be explained in the same manner as in Q2.5(a). Note that here higher α values perform better than in MC. This could be because TD has lower variance, so the quality of the Q update is better, meaning that bigger step sizes do not compromise the learning rate. The value that was used was $\alpha = 0.1$.

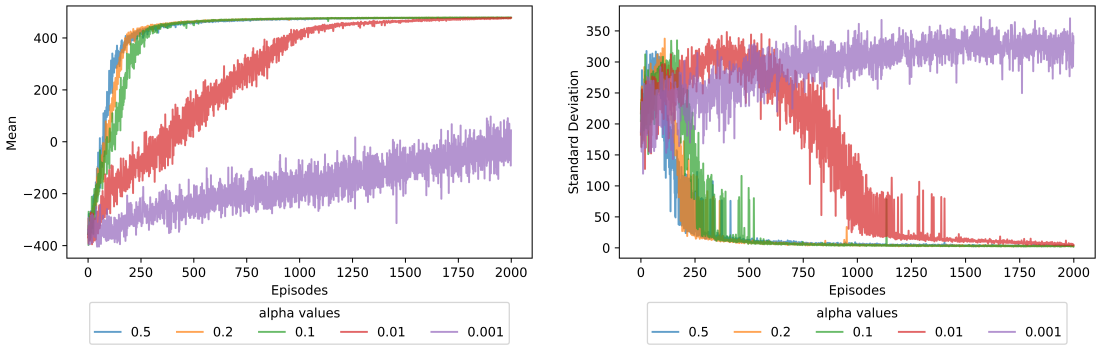

Figure 11: Left: Learning curves of the TD-agent with different values of α **Right:** Corresponding standard deviation

(b) **Exploration parameter:** The impact of ϵ can be explained in the same manner as in Q2.5(b). Note that in this case, all the learning curves converge reasonably quickly. The value that was used was the one where ϵ decays exponentially with $\lambda = 500$, where λ is previously defined.

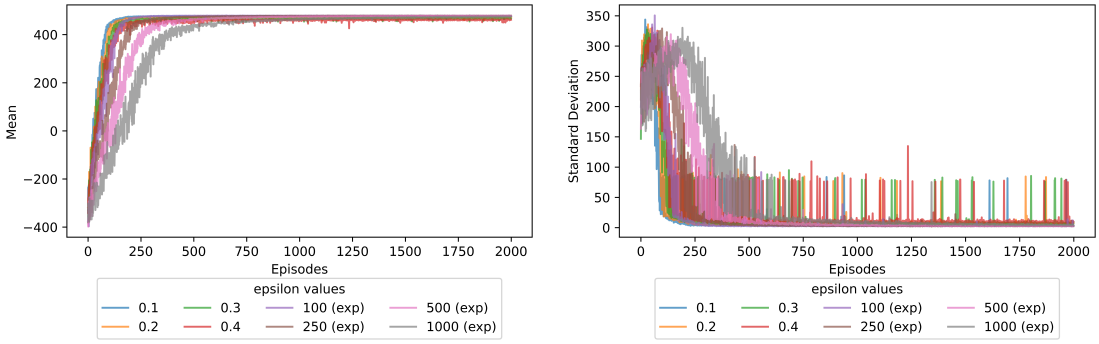

Figure 12: Left: Learning curves of the TD-agent with different values of ϵ **Right:** Corresponding standard deviation

Question 4: Comparison of Learners

1. Mean square error between the optimal value function vector computed through Dynamic Programming and the optimal value function vector computed through the MC and TD learners:

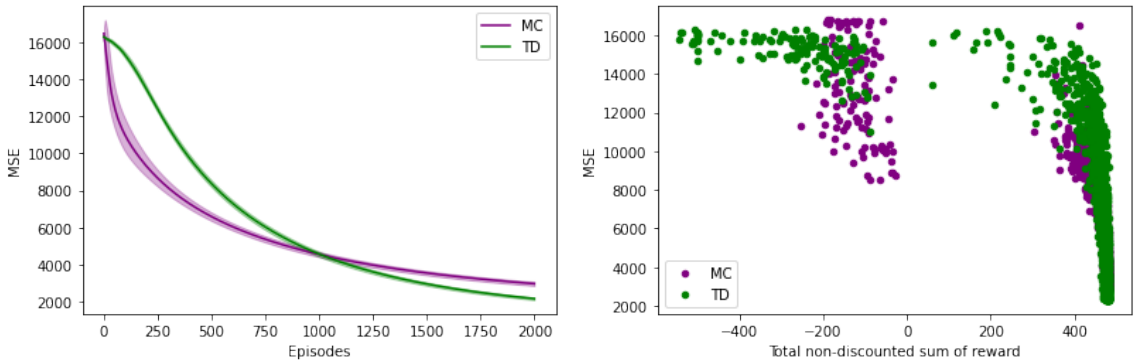

Figure 13: Left: Mean Square Error between the optimal value function vector computed through Dynamic Programming and the optimal value function vector computed through the MC and TD learners. **Right:** Scatter plot of the MSE of single episodes against the total non-discounted sum of rewards in a single run

2. In general, the MSE decreases as the number of episodes increases, indicating that the agents' optimal value function is converging to the optimal value function computed through DP (Fig. 13, left). The MSE corresponding to the MC-agent is seen to decrease faster for the initial episodes. This could be due to the fact that MC is not very sensitive to the initial value of Q, whereas TD is more sensitive as it bootstraps on the randomly initialised value function. TD performs better in later episodes because it exploits the Markov property, making it better in Markov environments. Since the MC target - the return - depends on every reward, state-transition and policy from the start state to the terminal state, we expect MC's variance to be higher than that of TD, where variance only depends on a single step at a time. This is indeed observed.

3. The scatter plot of the MSE of single episodes against the total non-discounted sum of rewards in a single run is shown in Fig. 13 (right).

4. Fig. 13 shows the function estimation error of the two agents against the total non-discounted sum of rewards. It characterises how having a good return relates to having a good value function estimate and vice-versa. We observe that having a good value function estimate is not necessary to obtain a good return, as there are episodes with high return and very high MSE. We could predict this as the learning curves of the agents (Fig. 5 and Fig. 10) converge faster than the value function (Fig. 13, left) - meaning the goal state is found before the value of each state is updated enough times to resemble the DP values. The main difference between the MC and TD scatter plots is that some relatively late MC episodes - which we identify as having a lower MSE - are found to have rewards lower than 0, meaning they have not reached the goal state. This could be because the MC-agent does not bootstrap and because the agent is still doing some amount of exploration in later episodes.