Lab 6 - Reinforcement Learning

Vidaich Francesco

1 Introduction

The goal of this assignment is understand the basic concepts of Reinforcement Learning and explore how it is possible to exploit them in order to solve problems where an agent has to reach one goal position in a 10x10 grid under different conditions.

2 Original Environment

The already provided `Environment` class was an empty 10x10 grid where at each turn he agent could make one out of five actions: go towards one of the four directions (up, down left, right) or staying still. After choosing the move, the method `move()` of the `Environment` class evaluates it by computing the correspondent `reward` and `next_state`. The `reward` is equal to zero in most cases, it is equal to 1 if the agent is on the goal state and chooses to stay there and it is equal to -1 when it tries to go through a boundary of the grid.
The `Agent` class was also provided, but two new methods were added to it: `visualize_values()` plots the estimated values that the agent learnt during training (computed by taking the maximum Q-value from that state) and `play_episode_gif()` makes the agent play one episode and builds an animation of it.

2.1 First Training

All the code needed to start training the agent was already available. We chose to rewrite the training process in the `train_agent()` function in the *training.py* file, but all the default settings were kept to the original values. That means that the result showed in Figure 1 and in this animation are obtained after training the agent for 2000 episodes of 50 moves each using the Q-learning algorithm, the ϵ-greedy policy, discount factor $\gamma = 0.9$ and ϵ values that decreases linearly from 0.8 to 0.001.

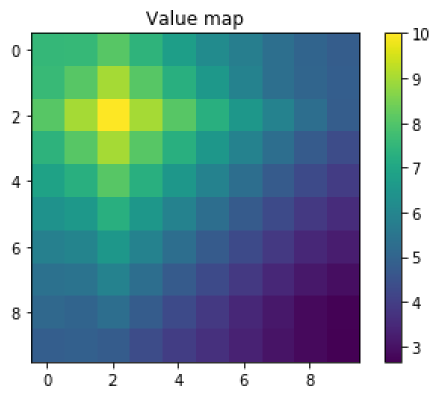

Figure 1: *Estimated values of each state of the system where the goal was* $[2, 2]$

The value of each state increases as the agent gets close to the goal, so the distribution is correct, but it is possible to check that also its magnitude makes sense given the specific discount factor $\gamma = 0.9$. For example, consider the estimated value of the goal state: once that the agent arrives there by following the optimal policy, it should not leave the state for the rest of the episode. This means that the value of the goal state is

$$V_{\text{goal}} = \sum_{i=0}^{N_{\text{episodes}}} \gamma^i R_{\text{goal}} = \sum_{i=0}^{N_{\text{episodes}}} \gamma^i \sim \frac{1}{1-\gamma}$$

where the last expression is true in the approximation of $N_{\text{episodes}} >> 1$, and it is equal to 10 when using a $\gamma = 0.9$, which is the same result obtained through training.
We also tried to start each episode from a fixed position and the obtained value map was equal to the one shown in Figure 1, which means that the exploration is working properly.

2.2 Different methods comparison

In this section the agent will be trained and tested using different RL algorithms (Q-learning and SARSA), different policies (ϵ-greedy and Softmax) and different decays for ϵ (linear and exponential). In order to evaluate the quality of each agent, the starting point was fixed to $[7, 7]$. By doing so, the agent can always arrive to the goal in 10 steps and spend the other 40 in the goal position if it follows the optimal policy, so the maximum mean reward per step that it can obtain is 0.8. The `compare_algorithms_and_policies()` function defined in *training.py* initialize and train an agent for each combination of algorithm and policy using the same number of episodes and the same decay rule for ϵ. The function then plots the evolution of the average reward per step computed over 100 episodes of each training procedure.

2.2.1 Linear decay of ϵ

In Figure 2 are shown the results obtained by training for 1500 episodes and decreasing ϵ linearly, always from 0.8 to 0.001.

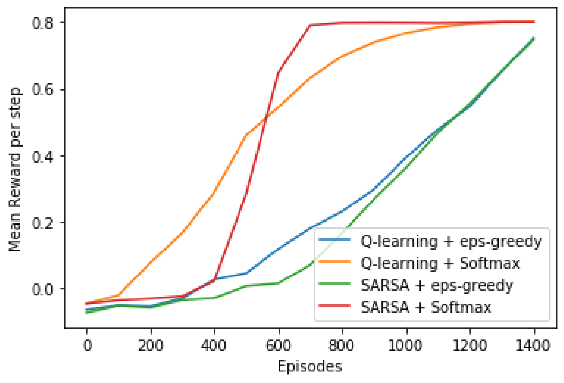

Figure 2: *Performance of agents trained with different algorithm and policies linear decay of* ϵ

2.2.2 Exponential decay of ϵ

The values of the exponential decay can be computed iteratively in the following way:

$$\epsilon_{i+1} = \eta \epsilon_i \quad \text{where} \quad \eta < 1$$

Now we have to find the value of the decay factor η. We want that after N_{episodes} the value is decreased from ϵ_0 to ϵ_{end}, so it must holds that

$$\epsilon_{\text{end}} = \eta^{N_{\text{episodes}}} \epsilon_0 \implies \eta = \left(\frac{\epsilon_{\text{end}}}{\epsilon_0} \right)^{\frac{1}{N_{\text{episodes}}}}$$

Using the same extremes of the linear decay ($\epsilon_0 = 0.8$ and $\epsilon_{\text{end}} = 0.001$), we obtained $\eta = 0.87486$ and the performances shown in Figure 3.

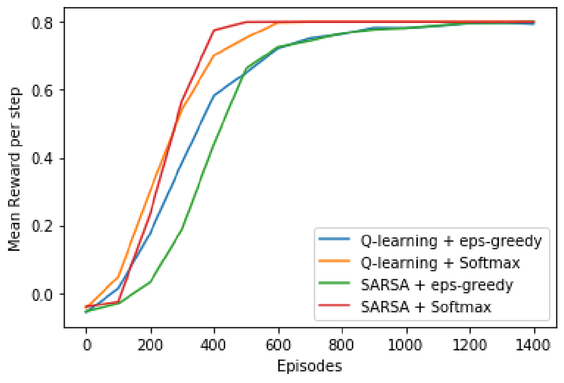

Figure 3: *Performance of agents trained with different algorithm and policies exponential decay of* ϵ

It is clear that the exponential decay of ϵ makes each combination of algorithms and policies converge faster to the optimal Q-values. This is particularly true for the Q-learning algorithm, which kept very low values of *Mean reward per step* in the previous case because of the slower linear decrease of ϵ. Indeed, increasing N_{episodes} did not help if the training was performed with the linear decrease of ϵ because in that case its value had even a slower decrease, so the *Mean reward per step* scaled with N_{episodes}. For these reasons, from the next section only the exponential decay will be used.

3 The Maze

In this section we will present a maze that the agent will have to solve. It will be contained in the same 10x10 grid that made the environment in the previous section and consists as many special positions (called `walls` in the code) that becomes inaccessible to the agent (or are accessible but give the agent a penalization if it tries to do so). The maze is shown in Figure 4, where black cells represent the walls, grey ones are the start (on the bottom right) and the goal (on the top left) states.

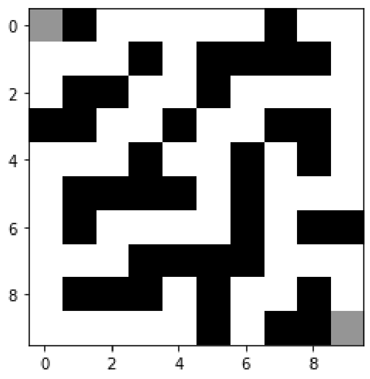

Figure 4: *Maze structure*

3.1 Insurmountable walls

If the walls are considered as boundaries, when the agent tries to go through them, it receives a negative reward of -1 and its position is not updated. Under these condition, the maze can be solved in 42 moves, so the length of each episode was extended to 168 (which is equal to four times the min number of steps) and the best obtainable *Mean reward per step* becomes 0.75. The `compare_algorithms_and_policies()` function was used again and its results are shown in Figure 5.

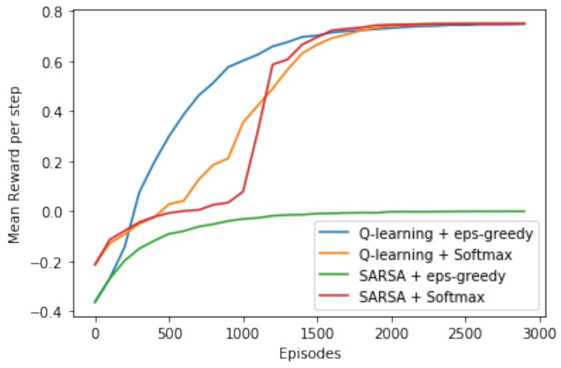

Figure 5: *Performance of agents trained with different algorithm and policies for the maze*

Here we see that, for the agent trained with SARSA and ϵ-greedy policy, the *Mean reward per step* converges to 0. This means that it was not able to reach the goal and obtain a positive reward.
Using any of the other combinations of algorithms and policies resulted in a value map equal to the one shown in Figure 6.

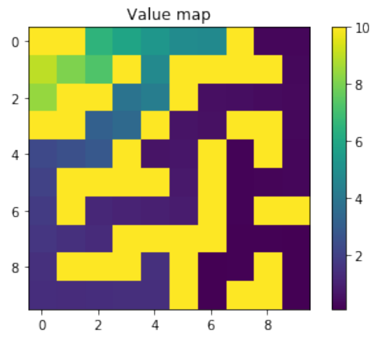

Figure 6: *Estimated values of the maze states*

The value of each position increase as the agents gets close to the goal. Moreover, the cells corresponding the walls have the maximum value, but this does not mean that they are the best states to stay. This is due to the initialization of every value of the Q-table, which are equal to be $\frac{1}{1-\gamma}$, and since the agent can't reach those positions, their value is never updated.

3.2 Surmountable walls

Now the agent will be able to go through walls, receiving a negative reward (called `wall_penal`) if it decides to do so. Even in these condition, the agent trained using SARSA with the ϵ-greedy policy was not able to reach the goal, but for many values of `wall_penal` the other agents continued following the maze. The agent's behaviour was study using `wall_penal` as a parameter and the results are shown in Figure 7.

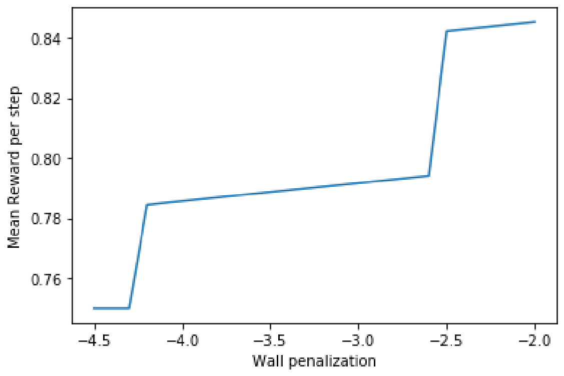

Figure 7: *Mean rewards of different trajectories inducted from different wall penalizations*

The agents seems very conservative and start going through walls from `wall_penal = -4.3`, even if it is the best strategy also for lower values of `wall_penal`. This is due to the discount rate γ being too small, which makes that choice less probable since the reward is many steps away. For example, the first breakthrough consists on passing through the wall in $[3, 0]$, which is close to the goal state. The best strategy (at least for any `wall_penal` higher than -20) is discovered only at `wall_penal = 2.5` because it consists on going through the wall in $[0, 7]$, in $[1, 6]$, or in $[2, 5]$ (same final result), which are relatively far from the goal, so the discounted return suggest them as disadvantageous moves.
By increasing the discount rate γ, both strategies are chosen for lower values of `wall_penal`.
The episodes corresponding to each strategies are:

- Insurmountable walls
- Surmountable walls - First trajectory
- Surmountable walls - Best trajectory

4 Conclusions

After fixing the initial state in each test to reliably compare the performances, the exponential decay for ϵ makes the agent converge faster to the optimal Q-values. In the maze, we saw how the discount rate affects heavily the trajectory, but further studies should be required to understand why the agent trained with SARSA and was not able to complete the maze.