IN4320 Machine Learning — Assignment 3

Francisca Mestre - 4930142

March 2019

1

Implementation of the scenario. For the implementation of the scenario the transition probability matrices $P(a)$ and the matrices of rewards $R(a)$ for each of the four actions possible were specified. The transitions are deterministic in each direction, therefore the element of $P(a)$ corresponding to the transition from state s to s' when taking action a is 1. Similarly for the reward matrices, the element corresponding to the transition ss' is 1 if s'=G and 0 otherwise.

Implementation of Q-iteration. The Q-iteration was implemented using the algorithm in the Lecture Slides [1].

The Q-iteration requires 12 *iterations until convergence*, with $\Delta < 0.0001$.

The *value function for the goal state G is zero* because the episode ends when the robot reaches G, and the robot does not move from G to any other state.

Figure 1 shows the value function V^* obtained with our implementation of Q-iteration with $\gamma = 0.9$. The *optimal policy* is to move to the state with higher function value than the one we are currently on; there exists more than one, in this example, and the arrows on the figure show one of them.

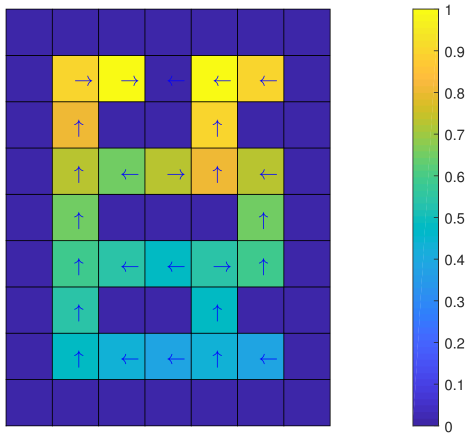
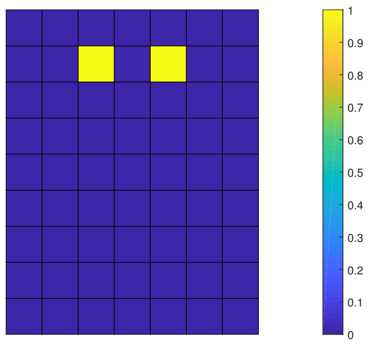


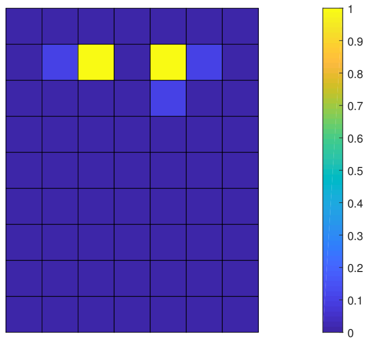
Figure 1: Value function V* with $\gamma = 0.9$ and optimal policy

2

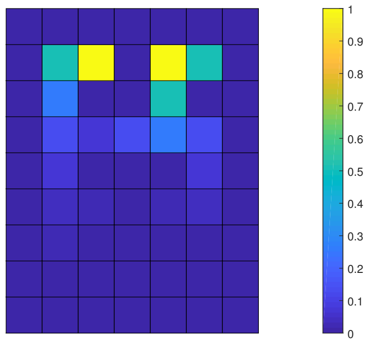
Figure 2 shows the value function V^* obtained for different values of the discount $\gamma \in \{0, 0.1, 0.5, 1\}$ for a fixed number of iterations. The discount factor $\gamma \in [0,1]$ is linked to the current values of future rewards [2]. Looking at the extremes, $\gamma = 0$ would mean that the future rewards don't matter at all to where we currently are at. On the opposite end, $\gamma = 1$ would mean that the values of future rewards mean as much in the future as they do now, i.e., they will spread to all possible states (not obstacles). This is very well illustrated by Figure 2. In Figure 2a we can see that for $\gamma = 0$ only the states that directly link to the goal state G have a function value of 1. In Figure 2d we see that for $\gamma = 1$ the reward of reaching state G is uniformly spread to all states where the agent might be in, since all of these states have a function value 1. Figures 2b and 2c represent in-between cases: for $\gamma = 0.1$ the reward of reaching state G is propagated to only to states very close to the goal; for $\gamma = 0.5$ we see the reward starting to spread to states that are farther from the goal. The infinite discount return has a finite value if the reward sequence is bounded [2]. As our reward sequence is 0 everywhere except when reaching the goal G then the reward 1 will simply spread to the entire maze while no other reward is accumulated in the meantime.



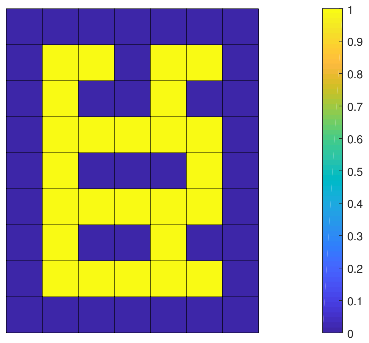
(a) Value function V^* with $\gamma = 0$



(b) Value function V^* with $\gamma = 0.1$



(c) Value function V^* with $\gamma = 0.5$



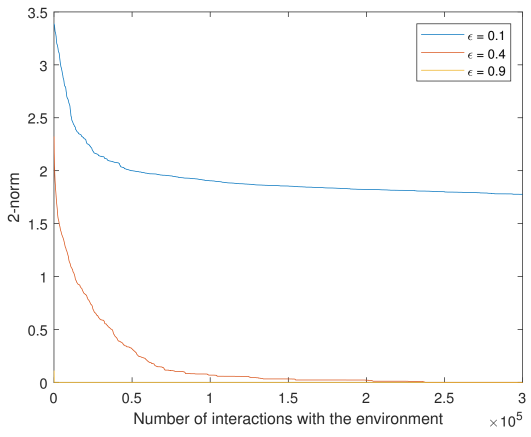
(d) Value function V^* with $\gamma = 1$

Figure 2: Value function V^*

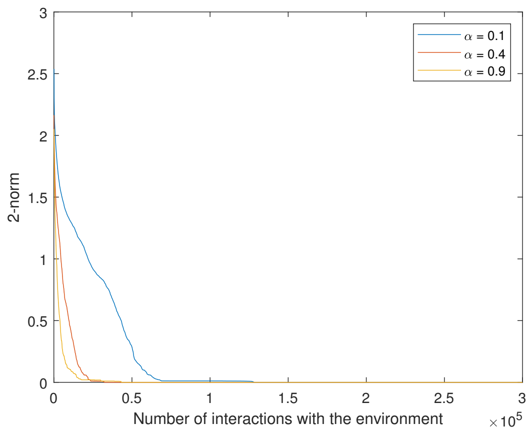
3

Implementation of Q-learning. The implementation of the Q-learning algorithm [1], [2] was initialized with value function equal to 0 for all states of the environment

We tested several values of the exploration ε and the learning rate α and computed the difference between the value function approximated by the Q-learning and the one derived from Q-iteration. Figure 3 shows the 2-norm of that difference against the number of interactions between the robot and the environment. In Figure 3a we fixed the value of α at 0.5 and in Figure 3b we fixed the value of ε at 0.5.



(a) α fixed to 0.5



(b) ε fixed to 0.5

Figure 3: Value function V^*

As we can observe from Figure 3a if ε is very small and the agent tries to use a lot of exploitation the approximated value function will need a very long number of interactions to converge to the real value function, since there is no information available on the system to exploit. As for very large values of the exploration the system converges much faster, however we don not wish that the agent mostly takes random values. We want to find some equilibrium between the randomness of the agent's actions and the actions he takes based on knowledge. In Figure 3b we can see how the convergence is when α changes. Note that the value of ε used is a bit too high. However the curves for $\alpha = 0.4$ and $\alpha = 0.9$ are not so different. One could argue that taking $\varepsilon = 0.4$ and $\alpha = 0.7$ would be a good combination to use in Q-learning.

4

In order to speed up Q-learning two methods were implemented: the first consists of using eligibility traces through Watkins's Q(λ) algorithm [3]; the second consists of decaying ε-greedy [4]. Figure 4 shows the evolution of the mean and variance of the difference with the number of interactions with the environment.

For the Q-learning we set $\alpha = 0.7$ and $\varepsilon = 0.4$; this method takes more than 300000 interactions to converge to the real value function; looking at Figure 3 we can better under-

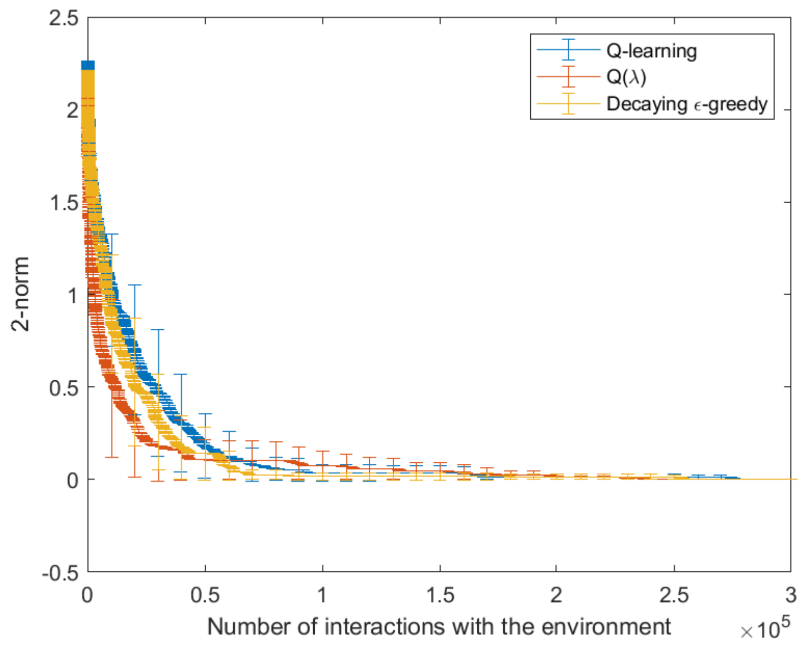


Figure 4: 2-norm of the difference between the approximated value function and the true value functions and its variance

stand why. We can also verify that Q(λ) with the same parameters $\alpha = 0.7$ and $\varepsilon = 0.4$ does in fact speed up the Q-learning algorithm significantly in the early stages. Further into the simulation, however, Q-learning and the decaying ε catch up with Q(λ). As for the Decaying ε-greedy method we started with a high exploration $\varepsilon = 0.9$ such that in the beginning the agent takes mostly random actions in order to learn about the environment faster. Then we start decreasing the exploration so that the agent will use more and more the information he now has available. This allows for the robot to find the true function value and its optimal policy faster.

References

[1] J. Kober. *Reinforcement learning, lecture slides, Machine Learning IN4320, Delft University of Technology, delivered 3 May 2019.* 2018.

[2] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning in a Nutshell.*

[3] Stack overflow. How to understand watkins's q() learning algorithm in suttonbarto's rl book? https://stackoverflow.com/questions/40862578/how-to-understand-watkinss-q%CE%BB-learning-algorithm-in-suttonbartos-rl-book, 2016. [Online; accessed 15 May 2019].

[4] Stack overflow. Q learning - epsilon greedy update. https://stackoverflow.com/questions/48583396/q-learning-epsilon-greedy-update, 2018. [Online; accessed 16 May 2019].