Report of homework 5: reinforcement learning using numpy

Clara Eminente

February 5, 2021

1. Introduction

In this report the basic features of a simple reinforcement learning paradigm are explored. In this particular case, the model is composed of an agent moving in a *10x10* grid.
In the first part of this assignment the code made available during the sixth laboratory session is expanded with some adjustments to make training and testing more versatile. Afterwards, the learning process is analyzed in a very simple environment (an empty maze) to highlight some emerging patterns and characteristics. In the the second part different learning schemes are compared. In the third and last section a less simple maze is considered and the different learning methods are compared in a more challenging context, highlighting the differences between them and the role of some parameters.
The analysis is contained in the jupyter notebook *Eminente_H5.ipynb*. The script *training.py* performs quick training and testing over the maze presented in the last section.

1.1. Reinforcement Learning

Reinforcement Learning (RL) is a learning paradigm in machine learning in which an agent learns to maximize some reward function by exploring a given environment; *agent* and *environment* are the basic ingredients of the paradigm.
An RL agent interacts with its environment in discrete time steps. At each time t, the agent receives the current state s_t and reward r_t. It then chooses an action a_t from the set of available actions, which is subsequently sent to the environment. The environment moves to a new state s_{t+1} and the reward r_{t+1} associated with the transition (s_t, a_t, s_{t+1}) is determined. The task of the algorithm is to learn the transition probabilities associated with each pair $\{s_t, a_t\}$. So, starting from a given initialization of such probabilities, the agent gradually adjusts their value in order to maximize the obtained reward. In this context, another fundamental ingredient of the algorithm is the $Q - Table$, which approximates such learned transition probabilities.

2. Architecture changes

As mentioned in the introduction, no major changes were made on the available code.
A function that trains the agent for *episodes* episodes, each one lasting *episode_length* steps, was implemented to make learning more versatile. The function gives the The function outputs the reward per step obtained at each episode, i.e. the values obtained dividing the reward obtained at the end of each episode by the number of steps in the episode.
A testing function was also implemented: the function basically moves the agent in the environment without updating the *Q-Table* and always taking the *greediest* action, i.e. the one associated with the highest expected reward. The function outputs the path followed by the agent.
The function *plot_moves* prints the maze associated to the environment and the path obtained through testing. Optionally, two more features can be plotted: an heat map of the maximum expected reward for each state and the actions associated to them.
Some changes were made to the *environment.py* library. The basic environment is composed of two main ingredients: a *10x10* grid (i.e. there are *100* available states) and the *4* boundaries that enclose the empty maze. Moreover, the environment is initialized with a starting state and a goal state.
The main additions to the available environment are *walls* and *swamps*. A *wall* is a state in the grid associated with *-1* reward (same as the boundaries); a *swamp* is a state associated to a reward equal to *swamp_reward* whose default value is *-0.5*. In the next section these two ingredients will be added to environment to observe how the agent reacts to their presence.

3. Results

3.1. Empty grid

In this section the results obtained using the already available parameters and empty grid are discussed. The parameters of the model are those in Table 1.

episodes	2000	goal	[0,3]
episode length	50	discount	0.9
x_dim	10	softmax	False
y_dim	10	sarsa	False
start	[7,8]	alpha (lr)	0.25

Table 1: *Parameters for the empty grid training*

The decay of the "*greediness*" parameter, ϵ, is linear, from *0.8* to *0.001*.

Figure 1 shows the output of the testing process. The first thing that is possible to notice is that the agent successfully learns the path to the target. The heat map shows that getting closer to the target the expected reward for each state increases, which is an ulterior hint of the fact that the learning worked properly. The numbers plotted on each state show the best action associated to the current state. Remember that 0 is stay, 1 is up, 2 is down, 3 is right, 4 is left. We notice that the agent not only successfully learns the path but it also recognizes all its possible alternatives.

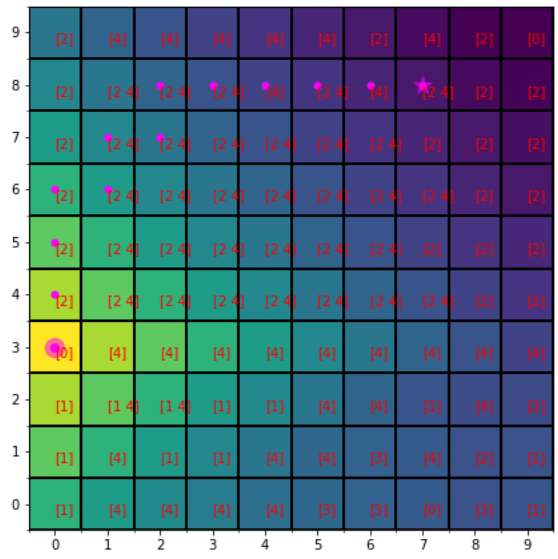


Figure 1: *Plot of the initial training*

In this context it is important to highlight the role of the initial state in the learning process. The training function gives the possibility to initialize the state randomly. It was observed that doing so, if not enough episodes are given (at least 3500) most states are not explored "properly". In these cases it can happen that during testing the learner fails to reach the target, especially for initial states very far from the goal. This is somehow expected, as changing the initial state forces the learner to not only learn a single path and its close variations but the whole environment. If not enough time is provided, some zones of the environment can remain poorly explored. From now on the initial state will be fixed during training and testing, mainly to be able to successfully train the agent with *2000* episodes.

3.2. Comparing methods

It was decided to train the agent with different combinations of algorithms (*off-policy=Q-Learning*, *on-policy=Sarsa*), different policies (ϵ greedy and softmax) and ϵ decays (linear and exponential). The other parameters are as in Table 1 and the empty grid of the previous section was used.
The performance of each combination was computed as the reward per step for each episode, averaged over *20* repetitions of the training. This value is referred to as *mean reward per step* (*MRPS*). This was done as the training process is stochastic and rewards at the end of each episode can be very different from run to run.
For this configuration the highest expected reward per step is ~ 0.78 as the minimum number of steps to get to the goal is 11 and thus the reward per step in the best case is $\frac{(50-11)}{50}$.

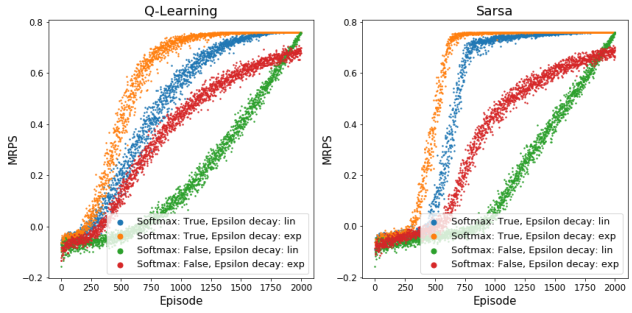


Figure 2: *Different combinations of architectures*

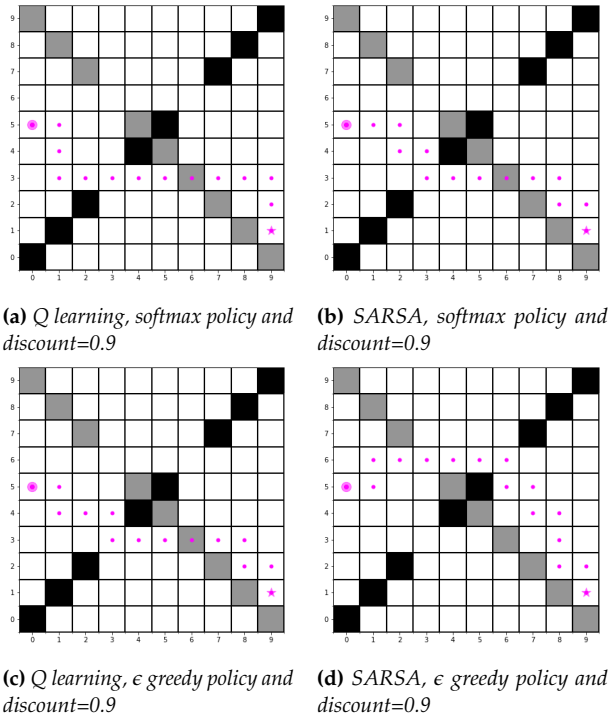
Figure 2 shows the performances of the aforementioned combinations. The first thing it is possible to notice is that, compared to other methods, the combination ϵ greedy and linear decay has a very slow trend. However, for this method and all combinations involving *softmax* policy the learning is successfull and eventually the agent learns the path associated with the best reward per episode. Moreover, the scatter plots tend to narrow as the end of the learning approaces, meaning that the $Q-Table$ is actually converging to one path (or many equivalent ones).
With this choice of parameters the worst performing combination is ϵ greedy with exponential decay: as it is possible to observe, the curve does not narrow enough and the final *MRPS* is lower compared to the others. This can be due to the fact that the combination of ϵ greedy and exponential decay makes the agent too conservative in its choices. Following an ϵ greedy policy the $Q-value$ of an action given a state is just used to check which action is

the best one, whereas the agent considers equivalent whichever action is not the best one. The exponential decay makes the agent more prone to chose the action associated with the highest $Q-value$ very early in the training. As a consequence the learner might become very conservative very early in the training. Its poor explorability might be the cause of the trend in Figure 2.

3.3. A simple maze

In order to get more insight on the behaviour of the agent with different parameters and learning methods a simple maze was implemented. It basically consists of a wall running all across one diagonal and a swamp running across the other diagonal, with some exceptions. The starting point is [9,1] and goal is [0,5]. With this configuration the shortest path passing through a swamp has highest reward per step ~ 0.75. The same quantity associated with the shortest path without swamps is 0.72.
The idea of this simple maze is to understand the choices of the agent when it comes to avoid a path that is instantly disadvantageous (passes through a swamp) but will lead to a slightly higher reward in the long run. The difference in the long term reward is small so that no neat preference can be given.
The parameters of the training are the same as before.
In the following only the performances obtained with the exponential decay for ϵ will be shown since not much difference was found using the linear decay.

Figure 3a and 3b show the behaviour of the agent with fixed discount and softmax policy. Both algorithms learn the fastest and, eventually, more rewarding path (or its equivalent versions).
It is interesting to notice something that was theorized when considering Figure 2 and that can be now observed and expanded: Figures 3c and 3d show the results obtained with same parameters but ϵ greedy policy. The agent trained with $Q-Learning$ is more willing to take risks even in this case and learns the same path as before. The agent trained using $SARSA$ is more conservative and sacrifices the long term reward for more immediate "safety". This is due to the fact that *Q-learning* is an *off-policy* algorithm: after having taken a certain action the agent assumes that it will act optimally in the future steps, as a consequence it is more willing to take risks immediately. On the contrary $SARSA$ uses the same policy for decision making and update of the $Q-value$. As mentioned before the



(a) *Q learning, softmax policy and discount=0.9*

(b) *SARSA, softmax policy and discount=0.9*

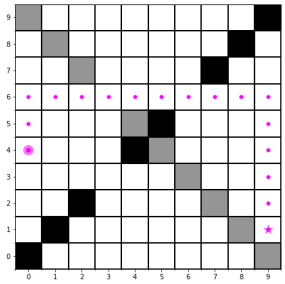
(c) *Q learning, ϵ greedy policy and discount=0.9*

(d) *SARSA, ϵ greedy policy and discount=0.9*

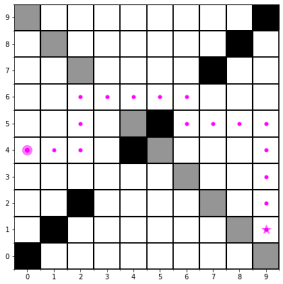
Figure 3: *Comparison of different combinations of parameters with exponential decay for ϵ*

combination of ϵ greedy policy and exponential decay makes the learner a bit more careful and this is emphasized by $SARSA$.
The agent trained through $Q-Learning$ and both agents trained using *softmax* become more conservative when the discount is lowered to 0.8 (proof can be found in the appendix).
In the appendix are reported the results obtained keeping the discount fixed to 0.9 but making the reward associated to passing through a swamp higher (in module) . The results are exactly the same as those in Figure 3.

Appendix

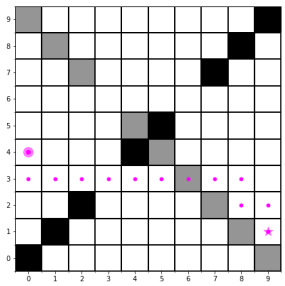


(a) *Q learning*

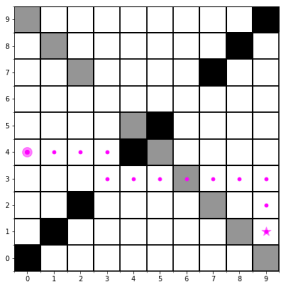


(b) *SARSA*

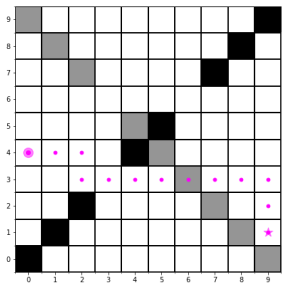
Figure 4: *Q-learning and SARSA with discount=0.8 and softmax policy*



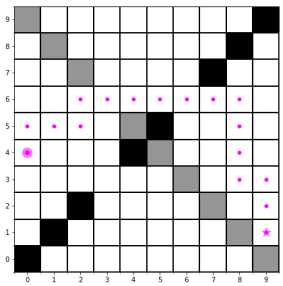
(a) *Q learning, softmax policy*



(b) *SARSA, softmax policy*



(c) *Q learning, ϵ greedy policy*



(d) *SARSA, ϵ greedy policy*

Figure 5: *Comparison of different combinations of parameters with exponential decay for ϵ, discount=0.9 and swamp_reward=-0.6*