Reinforcement learning with tabular methods

Nicola Dainese

I. INTRODUCTION

In this project I tested two different reinforcement learning (RL) algorithms from the tabular methods class: SARSA and Q-learning.

They are very similar in a sense, because they both estimate the so-called Q-values Q(s,a), that are the expected returns from an episode being in the state s and taking the action a, and they do it with a similar update rule:

$$\Delta Q(s,a) = r(s,a) + \gamma Q(s', a_{\pi(s')})$$

where:

- r(s,a) is the reward received for taking action a in state s.
- γ is the discount factor $\in [0,1]$, that expresses the preference for a present return over a future one.
- $a_{\pi(s')}$ is the action chosen following the policy π in the state s' (the one in which we arrive from s taking action a used in the update).
- π in the case of Q-learning is the greedy policy $argmax_a Q(s', a)$, whereas for SARSA is same policy π that we are learning (concretely ϵ-greedy or softmax). That is why SARSA is called an On-Policy algorithm, whereas Q-learning is an Off-Policy algorithm.

The policies used for exploration in Q-learning and for SARSA in general where ϵ-greedy and softmax.

II. HYPER-PARAMETERS TUNING

In this section I briefly describe the changes that I made to the original code in order to make the agents more performing.

First of all I consider a discount factor of $\delta = 1 - \frac{1}{T}$, where T is the length of an episode. This is equivalent to consider future payoffs without discount in T-1 out of T cases and equal to 0 in just 1 out of T cases (that is the case when the episode ends). In general, for finite-horizon episodes, discount factors smaller than 1 usually impact negatively the learning process and there doesn't seem to be any real advantage in using them, so at least this is a statistical trick to take into account the possibility of the episode ending before we collect all the return that we expect from the next state.

Then I introduced a negative reward as a baseline for actions. This can be thought as a punishment for an agent that decides to stay still and it would be useful especially in cases where the episode ends when the agent reaches the goal (because there is a just a single positive reward in the entire episode, so to maximize the reward it has to reach the goal as soon as possible). However since the task is very simple and the episode lasts always the same number of turns, this change doesn't really influence the outcome of the learning nor the speed of it.

The most influential change was to implement an exponential decaying schedule both for the exploration factor ϵ and the learning rate α, in order to train the agent faster and get a stabler result.

III. SARSA VS Q-LEARNING

In this section I present the tests made on the Sandbox environment, a 10×10 grid without anything in it, where the goal of the agent is to reach a fixed location in the shortest amount of time and then to stay there until the episode ends.

In Figure 1 we can see the training curves of three different agents trained with the Q-learning algorithm but with different parameters. The basic version is an ϵ-greedy policy with linear decreasing in ϵ from 0.8 to 0.01 and learning rate fixed to 0.25. The softmax version has the same linear decrease in ϵ and same learning rate, but uses the softmax policy:

$$a_{softmax(s)} \sim \frac{e^{Q(s,a)/\epsilon}}{\sum_{a'} e^{Q(s,a')/\epsilon}}$$

Where $\sim$ is used to indicate that the action is sampled from that distribution. Notice that in the limit of ϵ that goes to 0, this is equivalent to the greedy policy.

The final implementation is the one that uses softmax policy, the discount factor described in the previous section (concretely it was 0.98 instead of 0.90) and the exponential decaying schedule for α and ϵ. In particular α started from 1 and decreased until 0.1 whereas ϵ started from 0.8 and decreased until $8 \cdot 10^{-3}$ (two order of magnitude from start to end).

It is evident that softmax increases the speed of convergence to the optimal policy and so do the other changes introduced, with the final implementation converging almost 1000 episodes earlier.

Similar trends can be seen for the SARSA algorithm in Figure 2, where in particular the final implementation converges almost immediately.

For completeness I also reported the 6 curves in a single graph (Figure 3) in order to make a comparison between the two algorithms. It seems that with the softmax policy SARSA converges faster than Q-learning.

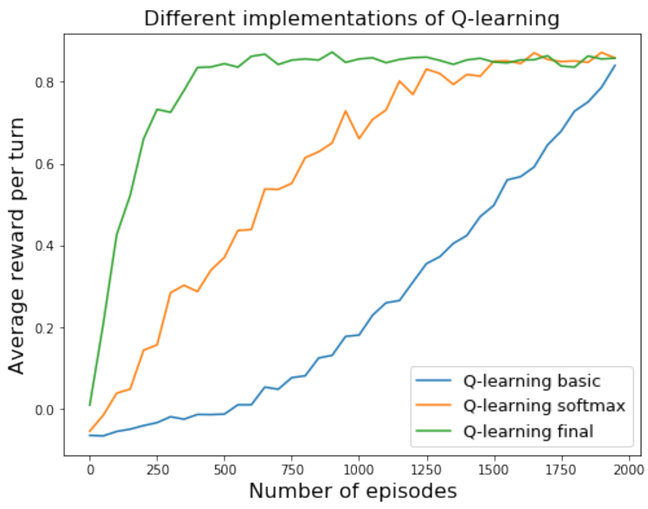


FIG. 1: Three different implementations of Q-learning algorithm. The basic one uses ϵ-greedy policy, linear decrease in ϵ and constant α. The softmax version is identical except for the policy, that is of course a softmax instead of an ϵ-greedy. The final version is discussed in this section in detail. All the curves represent averages in windows of 50 episodes (thus there are just 40 points instead of 2000); this was done in order to cut down the noise and make the curves more readable.

IV. DIFFERENT ENVIRONMENTS AND VISUALIZATION

In this section I describe the other two environments that I tested and some techniques to visualize what agent learns.

I call the first environment "Seaside" because I wanted to model a task where an agent has to go from one point to another (as before), but this time it starts on the sand and the goal is in the sea. The idea is that movement on sand is faster than on water, so I gave a higher cost (negative reward) when moving on water instead that on sand. The final trajectory of the agent should be something similar to the path that light follows in diffraction phenomena, e.g. less time/space in the mean with more resistance, more in the one with less resistance.

A visualization of the environment is provided in Figure 4(a) and in the notebook is presented an animated version (gif) of an agent that has been successfully trained on that environment, where is possible to see how the agent moves concretely. Instead if we want a static visualization of the policy learned, we can turn to the value maps. A value map is a useful and simple way to visualize what an agent has learned. The idea is to take the Q-values table and for each state to choose the best action (optimal/greedy policy); what we obtain is the expected return from the episode for being in a given state, also known as value of a state. In Figure 4(b) is possible to see the value map of the Seaside environment. Notice that for the goal cell, that value is equal to the reward of a single turn for staying in that cell multiplied by the number of turns.

The other environment that I implemented is called Bridge environment and the idea is to have two cliffs on the sides of the map and the void in between; there are however 2 passages (bridges) and the agent has to learn to pass through one of them in order to get to the goal. The void is implemented as follows: if the agent tries to go there it receives a strong penalty (-10 w.r.t. -10 of the border of the map) and restarts from the initial point. From this follows that a void state can never be entered and so its real Q-value is not defined. That is the reason why it is represented as a high-value region: that is just the initialization value of all Q-values. Also in this case in the notebook is possible to find the gif of a trained agent solving this task successfully.

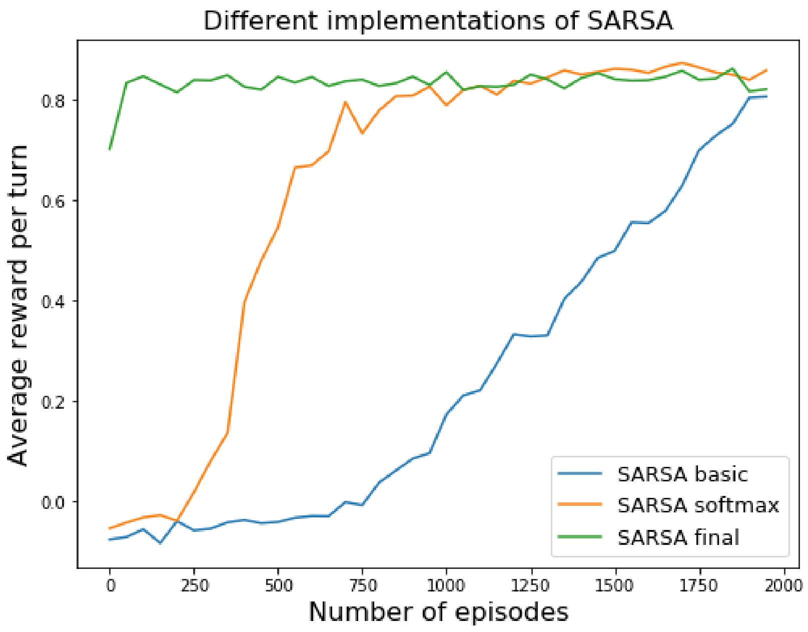


FIG. 2: Three different implementations of SARSA algorithm. For more details read caption of Figure 1.

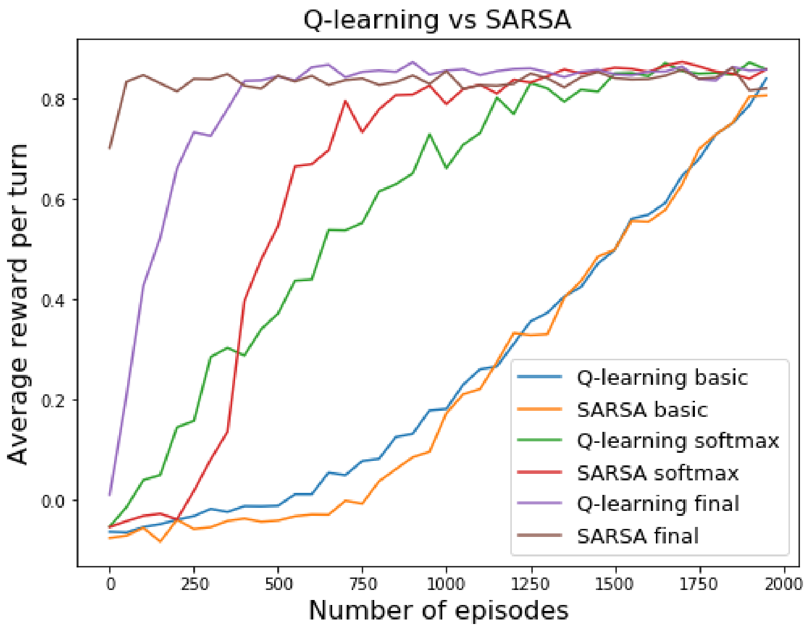
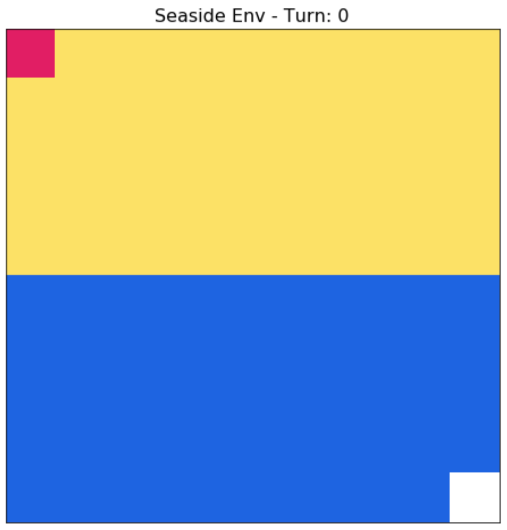
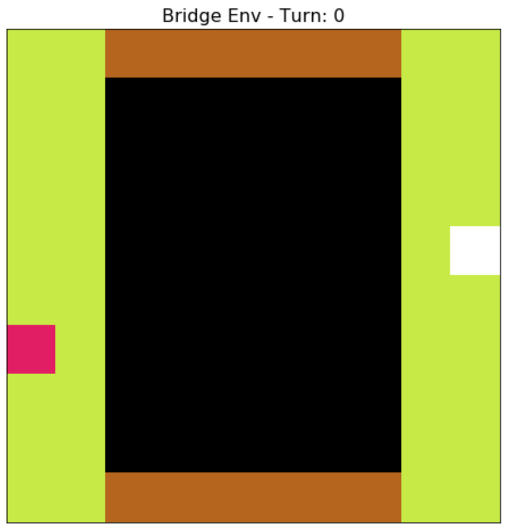


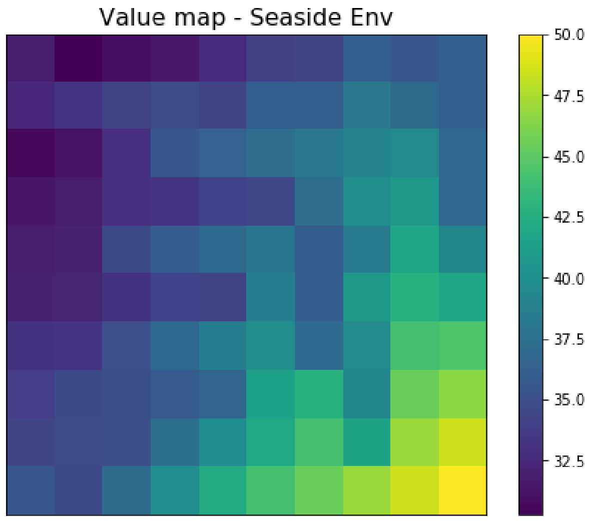
FIG. 3: Comparison between Q-learning and SARSA algorithms. For more details read caption of Figure 1.


Seaside Env - Turn: 0

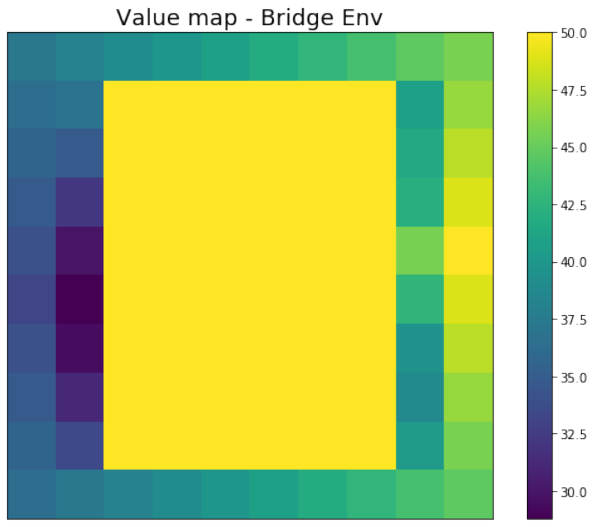
(a)


Bridge Env - Turn: 0

(b)


Value map - Seaside Env
50.0
47.5
45.0
42.5
40.0
37.5
35.0
32.5

(c)


Value map - Bridge Env
50.0
47.5
45.0
42.5
40.0
37.5
35.0
32.5
30.0

(d)