Exercise 3

Ludovica Mazzucco
Department of Control and Computer Engineering
Politecnico di Torino
Italy
s315093@studenti.polito.it

Abstract: The aim of this report is to present the performance obtained appling the Q-learning to the Cart-pole agent from the Gym Environment.

1 Introduction

Reinforcement Learning approach is based on making the agent learn through the interaction with the environment the right behaviour to follow, i.e. the policy, in order to maximize its expected return. Since in general it is not provided with prior knowledge about which action is better to take in a certain state nor the reward it will get from it, it has to collect through its experience as much information as possible to construct the estimates of these values it could then exploit in the future. More precisely, any RL algorithm is focused on drawing the so called $state - value\ function$ $v_\pi(s)$ expliciting the expected return the agent will get when it finds itself in that state s, plus the $action - value\ function$ $q_\pi(s, a)$, indicating in this case the expected return when the agent is in state s, decides to take an action a and finally follows the policy. One could think of this overall decision-making process as a random walk in a MDP, where the agent's degrees of freedom are the action it can perform first, obbeying to the probabilities given by the policy $\pi(a|s)$, and then the successor states s' it can end up in obtaining a certain reward r under the transition probability $p(s', r|S_t = s, A_t = a)$. Rather than just computing the value for each state or state-action pair of the forementioned value functions, the agent would aim at learning the optimal ones $v_*(s)$ and $q_*(s, a)$, that are the functions it would get when the optimal policy π_* is followed.
Fortunately, the Optimal Bellman Equations provides us with a nice closed formula to find them just solving a system of non-linear equations giving a unique solution, therefore it is relatively easy to draw from them the optimal policy if for every state the corresponding optimal value function is known, because it is nothing but the one giving non-zero probabilities to those actions maximizing the function in that state starting from the ones of all the possible successors. Besides, if the computation exploiting the $v_*(s)$ would take one step more, using the $q_*(s, a)$ would be quite effortless since it gives us directly the knowledge of the returns when deciding to perform the candidate best action.
The latter method is basically the idea evoked by Dynamic Programming (DP) solutions to progressively learn the optimal policy in the two phases of policy evaluation ($prediction$) and policy improvement ($control$). Instead of solving a unique system of equations, this approach just exploits the Bellman Equations as update rule (the classic ones for policy iteration while the ones for optimality for value iteration) in the first phase and then greedily improve the undergoing policy selecting for each state the best action according to $q(s, a)$ instead of the one suggested by the running policy, then again evaluate the brand-new policy to generate a value function coherent with it and regenerate a greedy policy with respect to the latter until the two processes converge in a single point since the policy cannot be further improved, hence it is the optimal one as well as the corresponding value function. DP rationale is easy to implement and has demonstrated to be successful once its hypothesis are verified, but actually it is not practical given that it requires a not negligible computational expense and, furthermore, it expects the dynamics model to be known.
At this point other kinds of Reinforcement Learning algorithms provide for the latter aspect, like Monte Carlo (MC) and Temporal Difference (TD). Both of them exploit the idea of approximizing

the state-value (or action-value) function with samples of returns experienced by the agent along its trajectory in the MDP on subsequent -State, Action, Reward- triplets and use them as target in the update of the estimate. Conversely, they differ in the sense that the first needs necessarily to complete the episode and reach a terminal state before obtaining the estimate, while the second is based on bootstrapping and each step updates the current value function. Indeed Temporal Difference is the RL algorithm exploited in this assignment in order to train the Cart-Pole agent from the Gym Environment with the aim to test its performance.

1.1 Overview on Q-Learning

Q-Learning falls under the Temporal Difference different typologies of algorithms for Reinforcement Learning, meaning that, as suggested in the previous paragraph:

- Requires no a-priori information about the agent dynamics;
- Is based on bootstrapping;
- Updates the value functions estimates at each step without waiting until the end of the episode;
- Develops through samples of the state and action sets.

Moreover, the critic characteristic of this method is that it is ”off-policy”, consequently it exploits not just one but two different policies, the $behaviour$ policy and the $target$ policy. On one hand, the target policy is the one the agent has to learn and it is generally a greedy policy, but in order to maintain a little bit of exploration and fully be able to perfectly approximate the action-value function for any state-action couple, the behaviour policy is used instead for the ”motion” of the agent.
All in all, the resulting update formula used by Q-learning is the following:

$$Q(s,a) = Q(s,a) + \alpha(R_{t+1} + \gamma \max_{\forall a \in A(s')} Q(s',a) - Q(s,a)) \tag{1}$$

This expression indicates that from a state s the agent performs an action a stated by the behaviour policy, it receives then the corresponding reward from the environment and ends up in the next state s’. Hence, those quantities are exploited to calculate the update of the Q function with an approximation of q_* as target policy. Finally, the parameter α is the step-size and γ is the discounting factor.

1.2 Close-up on the Behaviour Policy: the ϵ-greedy policy

We have just underlined the fact that off-policy algorithms work with different policies for behaviour and for learning, we have also specified that generally the target policy is greedy, so what about the behaviour policy?
Rationally, it should be by definition a policy enforcing the agent’s exploration in the environment in order to better improve the candidate optimal policy and to define the action-value function for all the possible state-action couples. A wise move, which is also the choice taken for this simulation, is to consider an ϵ-greedy policy, that explores with probability ϵ and selects otherwise the greedy action.

2 Experimental Results

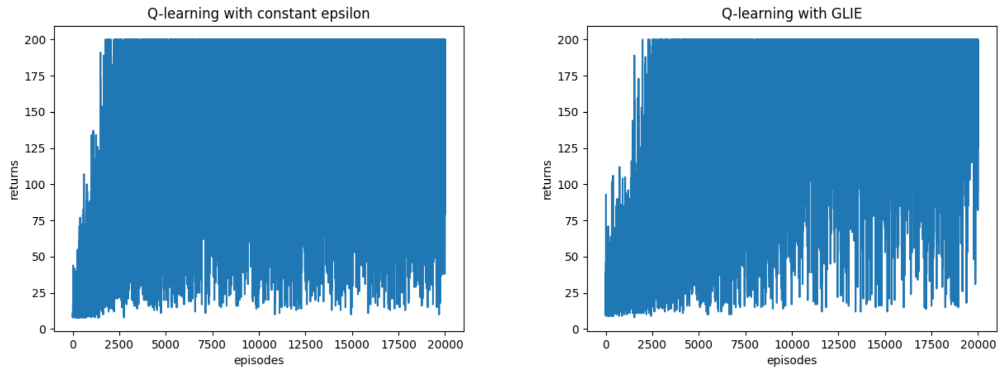
After having introduced the whole topic of Q-learning and the rationale behind it, let’s now discuss the performance obtained in the special case of the Cart-pole under different points of view.

2.1 The Influence of GLIE Approach

As disclosed in section [1.2], for the purpose of this assignment it was chosen for the agent behaviour an ϵ-greedy policy in order to balance exploration and exploitation. Furthermore, two options has been tested for the ϵ shape, one consists in maintaining it constant (precisely with value 0.2) and the other in letting it change over time and, in particular, decrease with respect to the passing episodes, to focus the exploration during the initial training phase and gradually prefer the greedy actions. We refer to this approach as GLIE, Greedy in Limit with Infinite Exploration. The expression of ϵ at the k-th episode is thus gived by:

$$\epsilon_k = \frac{b}{b+k}$$

with b properly chosen to obtain $\epsilon_{20000} = 0.1$, in this case $b = 2222$.
To better visualize the different impact on the agent learning process of the constant or GLIE parameter, two simulations have been run and the following plots show the corresponding returns obtained, which are directly proportional to the episode length.



(a) $\epsilon = 0.2$ (b) ϵ under GLIE

Figure 1: Q-learning performance difference with respect to the expression chosen for ϵ

What can be noticed in those pictures is that the simulation with a constant ϵ (left) gives as output a plot more "solid" compared to the other, especially from episode 12500 on. This reflects the fact that keeping a fixed probability to explore makes the agent try and fail more, generating episodes with early termination, while with GLIE it gradually tends to choose more the greedy action opposed to the random one. In fact, the lines in the right plot are condensed in the upper part where episodes last more on average. Besides, both of the plots highlight the agent tendency to explore and consequently fail more often during the beginning, where the two ϵ have similar effect to the learning process.

2.2 Visualizing the State-Value Function

Considering the case with a variable ϵ, the learned Q function should have been converged to the optimal one as assured by the Q-learning algorithm. The optimal policy then is drawn very easily, or well just taking into account for every state the action for which the correpsonding entry in the Q matrix is greater than the other (remind that in the cart-pole environment there are just two discrete actions possible 0 or 1). Moreover, this action-value function can be exploited to recover for any state the state-value function through the expression below:

$$V(s) = \max_{a \in A(s)} Q(s,a)$$

In this way we are enabled to immediately get the expected return from a given state. In order to better understand its content, it is possible to represent the matrix as an image through the heatmap, built compressing the fourth and third dimensions (the angular and linear velocities, respectively) averaging their values and after plotting it by means of the position and angle dimensions. What results from this process is shown here:

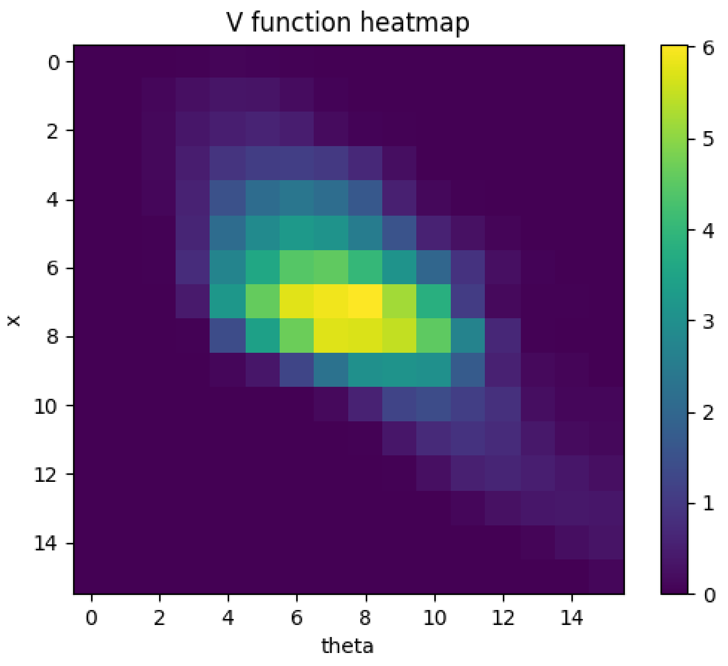
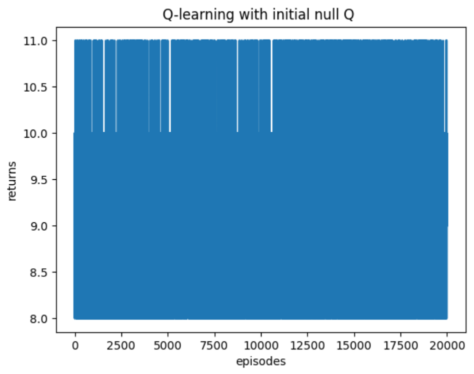


Figure 2: Heatmap corresponding to the state value function

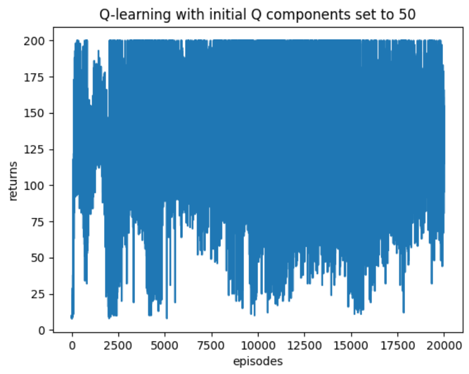
The heatmap associates different intensities of colors to different magnitudes of matrix elements, as the colorbar on the right edge suggests, thus this can be exploited for an interpretation of what the figure is reporting. An immediate observation is that the inner contours of the square are dark-colored, meaning that the corresponding entries in the matrix are close to zero, while the centre is perfectly yellow, shading to green along the diagonal. Hence, this zone aggregates the greatest elements in the multidimensional array. Now if we think at those elements as returns we will recognise that the central part of the matrix collects the states from which the highest rewards have been experienced, indeed those are the configurations in which either x and θ are close to zero, in other words, where the cart is in the centre of the screen with the pole at equilibrium (note that the numbers along the figure axes are just counting the cells, in practice $x \in [-2.4, 2.4]$ and $\theta \in [-0.3, 0.3]$). Conversely, the contour represents the forbidden regions in the state space where the agent has never received any reward, due to the fact that they correspond to critic situations with the cart in complete instability. Following this line of reasoning, before the start of the training the heatmap would be all dark, since we have considered in this case null initial conditions and the agent has not experienced anything about the environment yet, only after the first trials its knowledge would be enriched and as a consequence the heatmap would start to show zones with a lighter blue still around the centre of the figure which by the end of the training will look like the one above.

2.3 The Importance of Initial Conditions

As we have claimed in the previous paragraphs, the choice of an ϵ-greedy policy is aimed at allowing the agent explore configurations in which it would not end up in if it strictly followed the greedy policy but they are potential ways to get better rewards. Moreover, without a minimum exploration there would be state-action pairs for which the value function won't be defined (i.e the contour of the heatmap in 2) given that, unlike Dynamic Programming, it is approximated from samples of trajectories in the MDP and not directly from all the states in the state space. If ϵ is set, then we have seem that this trade-off between exploration and exploitation is well managed but, if it is cleared an we cannot rely anymore on it , we can still force the agent to explore tuning the initial conditions. In fact, if they are set to values strongly greater then any return it would get, then any action will result in an unsatisfactory performance since, recalling the formula in Figure 1, the error between the target and the initial estimate would be negative. As a result the agent is encouraged to explore despite the fact ϵ does not play any role. This assertion is demonstrated by the figures below corresponding to simulations with $\epsilon = 0$ and Q initially set to all zeros in one case and to all 50 in the other.



(a) initial Q set to zero



(b) initial Q set to 50

Figure 3: Comparison between returns when $\epsilon = 0$ and Q is set to different initial conditons

As it is easy to notice, the plot on the left corresponding to an initial null Q testifies a poor performance, with the agent hardly reaching 11 time steps alive given that, as it was said, there is no exploration at all. The setting with the 50-valued Q performs definitely better. Nonetheless, this trick becomes useless in the case of a non-stationary dynamics because the effect of the initial condition vanishes with the increasing episodes, hence focusing the exploratory part at the beginning of the training, but nothing assures that the model changes in time and another exploration phase would be needed.

2.4 Extension to Continuous State and Action Spaces

The cart-pole example taken into account in this assignment to test the Q-learning algorithm is intrinsically continuous in the state space, for this reason it has been discretized and the tabular version of Q-learning is used. Anyway, when it comes to continuous state spaces, methods like the one followed here are not feasible anymore since the agent would never be able to experience the whole state space which has infinite cardinality, let alone have enough memory and computational resources available. Nevertheless, function approximation could help to generalize quite well the original set and provide for parametrized value functions with parameters properly tuned with Gradient Descent or a way other then this is through Artificial Neural Networks. Unfortunately, there can be cases in which function approximation lead to divergence when applied to off-policy algorithms like Q-learning and this results from the fact that updates are not done accordingly to the behaviour and target policies, but, inspite this, when using an ϵ-greedy policy as behaviour this problem can be minimized since the two are approximatively the same. So, Q-learning can bear continuous state space but with all the due caution. In what regards the continuous action space, this represents a real problem when coming to the calculation of the TD target, in which the maximum over all the possible actions has to be drown; if this set is very large, say it has infinite cardinality, it would be very unlikely to match the exact value.