Reinforcement learning

Lorenzo Mancini 2019098

January 18, 2022

Abstract

In this homework we present two applications of Reinforcement Learning. We investigate the ablity of the model to learn by maximizing its cumulative reward in two different Gym environments: the CartPole and the LunarLander.

1 Introduction

In Reinforcement Learning the aim is to make an agent able to choose among some possible actions in order to maximize its cumulative reward. The main ingredients for RL are: the agent, a set of states $S = \{s_1, ..., s_n\}$ and a set $A = \{a_1, ..., a_n\}$ of action per state. The agent can move from state to state by performing some transitions according to a certain policy. When an action is executed, the agent receives a reward. As stated before, the main goal is to maximize the cumulative reward, given by:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ..., \tag{1}$$

where the discount factor $\gamma \in [0, 1]$ can be tuned in order to give more or less importance to future rewards.

1.1 Q-learning

We use the so called Q-learning to make the agent learn how to link a value Q to a particular state-action:

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t|s, a], \tag{2}$$

where $\pi(s)$ is the so called policy. The cumulative reward is maximized by the Bellman equation:

$$Q\left(s_t, a_t\right) \leftarrow Q\left(s_t, a_t\right) + \alpha\left(r_t + \gamma \max_a Q\left(s_{t+1}, a\right) - Q\left(s_t, a_t\right)\right) \tag{3}$$

The agent explores the enviornment through the above-mentioned policy. We're going to consider two exploration policies, namely:

- ϵ-greedy policy, in which the agent selects a non-optimal action with probability ϵ and selects the optimal one with probability $1 - \epsilon$;
- softmax, in which the anget selects the action from a softmax distribution (tuned with a temperature).

We will use a neural network in order to approximate the Q-values. Moreover, in order to avoid instability, we'll use two copies of the network: the first (online network) is updated every step, whereas the second (target networ) is updated only after a certain number of steps and it's used for action selection. Thus we want to minimize:

$$\mathcal{L}\left[\underset{\text{net}}{\text{online}}\left(s_t, a_t\right); r_t + \gamma \max_{a'} \underset{net}{\text{target}}\left(s_{t+1}, a'\right)\right], \tag{4}$$

where $\mathcal{L}$ is the loss function (see later). Finally, we also make use of replay memory, that is we store in a memory buffer previous learning episodes and then we randomly pick them in order to make the model able to generalize better.

1.2 Gym

We're going to apply Reinforcements methods to two different tasks developed by OpenAI in the Gym package.

- the **CartPole** enviornment;
- the **Lunar Lander** enviornment.

1.3 Softmax polocy

For our tasks we're going to consider an exponentially decreasing exploration profile using a softmax policy:

$$T_i = T_0 \exp\left(-\frac{i\beta \log(T_0)}{N}\right),$$

where $\beta = 6$, N is the number of iterations and $i = 1, 2, ..., N$.

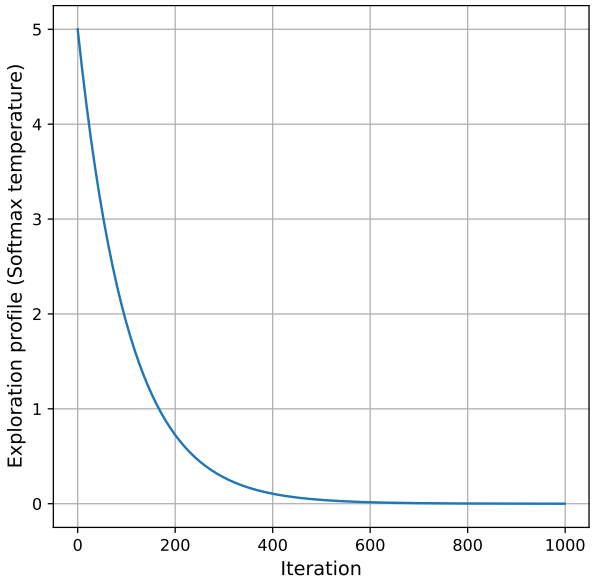


Figure 1: Plot of the exponentially decreasing exploration profile.

2 CartPole

Here, a pole is linked by a joint to a cart and the latter is allowed to move horizontally in a frictionless track. The aim is to let the Pole remain upright as long as possible. We're allowed to move the track left or right by applying a discrete force. If the pole overtakes 15 degrees from the vertical or if the Cart moves more than 2.4 units from the center. We use a network composed of an input layer of 4 neurons an hidden layer of 128 neurons and an output layer of 2 neurons. The activation function and the optimizer considered are respectively the hyperbolic tangent and Adam optimizer.

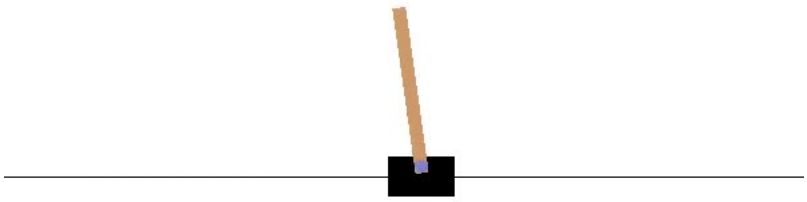


Figure 2: Screenshot of the CartPole enviornment.

We perform an initial training with the following hyperparameters:

- $\gamma = 0.97$, the discount rate;
- learning rate equal to 0.001;
- batch size of 128;
- SGD optimizer.

As loss function we use the `SmoothL1Loss` (Huber loss).

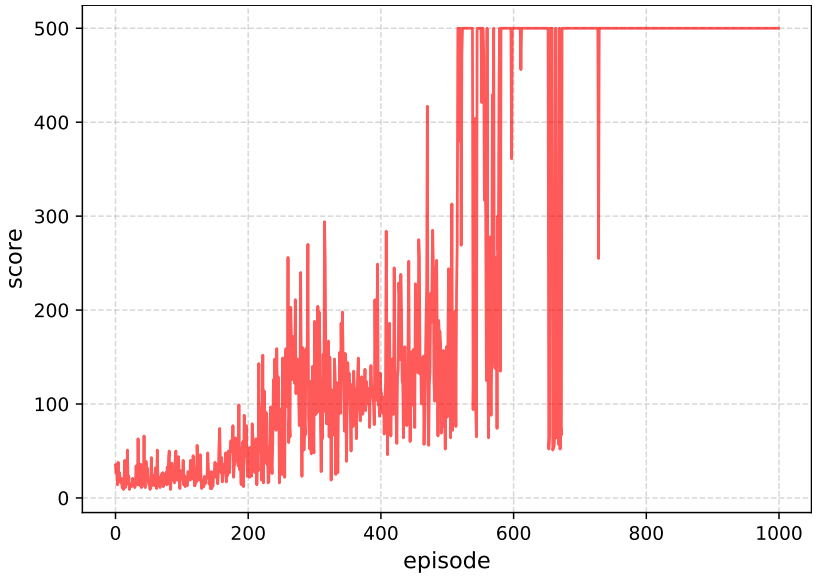


Figure 3: Score results for the initial training.

Now we want to investigate if something changes if we modify a little bit some hyperparameters. In particular, now we try give more weight to future rewards and we use a different optimizer with a different initial learning rate:

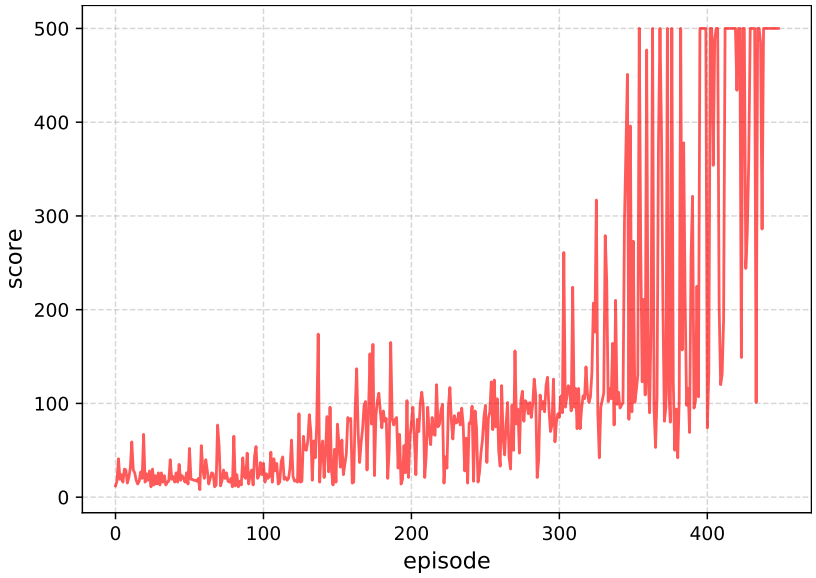
- $\gamma = 0.99$, the discount rate;
- learning rate equal to 0.0001;
- batch size of 64;

- Adam optimizer;

Furthermore, we add a penalty if the block moves far from the center, as we want it to stay as close as possible to te center and not going out of the screen:

$$reward- = |blockposition|. \tag{5}$$

Finally, we add a stopping condition that breaks the training if the network reaches the maximum score for at least 10 epochs.



It is clear that with those new hyperparameters it takes less episodes to reach the maximum score.

3 LunarLander

In this second task, as the name suggests, we want to navigate a lander to its landing pad. From the documentation we can find the following:
Landing pad is always at coordinates (0,0). Coordinates are the first two numbers in state vector. Reward for moving from the top of the screen to landing pad and zero speed is about 100..140 points. If lander moves away from landing pad it loses reward back. Episode finishes if the lander crashes or comes to rest, receiving additional -100 or +100 points. Each leg ground contact is +10. Firing main engine is -0.3 points each frame. Solved is 200 points. Landing outside landing pad is possible. Fuel is infinite, so an agent can learn to fly and then land on its first attempt. Four discrete actions available: do nothing, fire left orientation engine, fire main engine, fire right orientation engine.

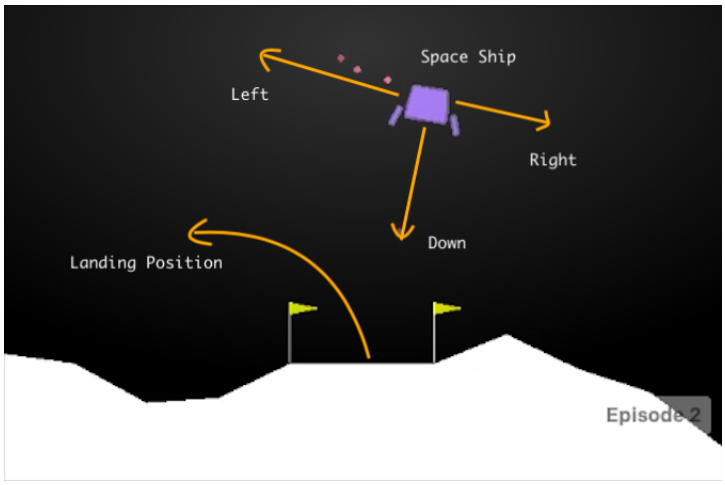


Figure 4: Screenshot of the LunarLander enviornment with corresponding actions.

The structure is the following:

- the input layer with 8 neurons;
- two hidden layers with 128 neurons each;
- output layer with 4 neurons;

3.1 Results

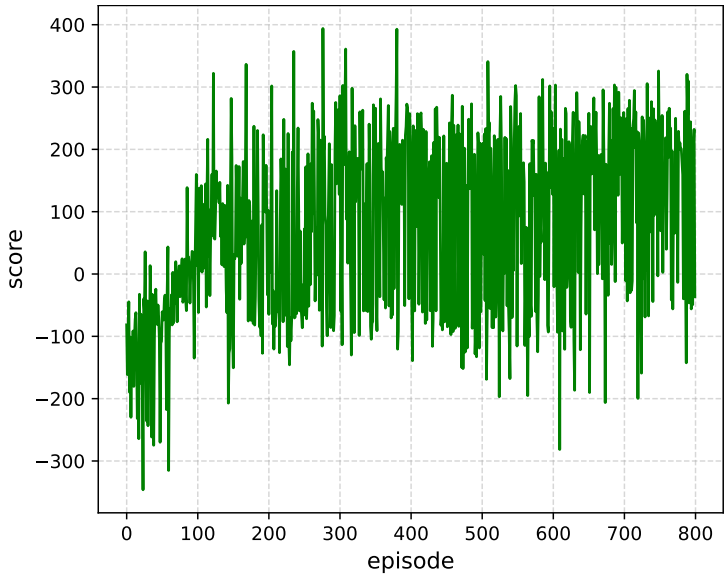


Figure 5: Scores obtained with the first set of hyperparameters.

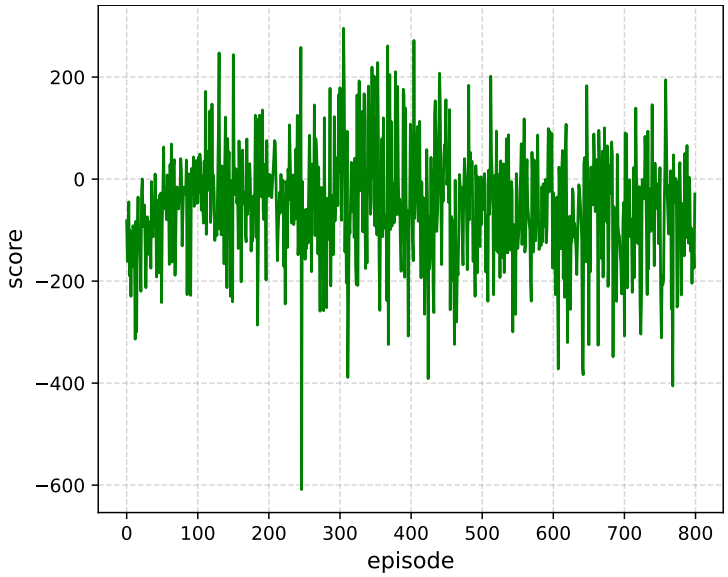


Figure 6: Scores obtained with the second set of hyperparameters.

Here, the training results much more difficult than before. Moreover, with the second combination of hyperparameters, the scores are quite lower than the ones obtained with the first training