Neural Networks and Deep Learning

# Homework 3, Reinforcement Learning

Daniele Foscarin

10 February 2021

## 1 Introduction

In this project we focused on the implementation of reinforcement learning techniques. We organized the code in three notebooks, corresponding to three tasks:

- Improving CartPole training, we serch a solution to improve the proposed code in the Lab 7 by trying many exploration profiles and tuning the hyperparameters, be developed an exploration profile that showed to greatly reduce the time needed for the model to stabilize in the winning score.
- CartPole training from pixels, We implement the training of the CartPole using the information provided by the rendered frames in every steps of the game. We implemented a function that reads a sequence of frames and returns a low dimensional state. This design choice, together with the use of the same useful techniques we introduced in the previous trainings, and a more complex DQN architecture led us to winning the game.
- Training of the Acrobot environment, we adapt the successful techniques found before in order to train a new environment provided by OpenAI.

In the following section we describe the strategy used in every part, and in the last section we present the results obtained at the end of the trainings, and we discuss their quality.

## 2 Method

### 2.1 Improving the CartPole training

In order to train the base CartPole enviroment we performed a revision on the strategies and hyperparameters used during the lab 7.
First of all we needed to define an objective that we want to reach in order to consider a training finished. This environment is defined in such a way that if the cart reach 500 frames without the pole falling the game is considered won and the episode stops. But since the training scores tend to be very noisy, it is not sufficient to train until the first time the game is won, instead we need to train until the model is able to control the cart consistently winning the game many times. So we define a stopping condition where we obtain the score 500 in ten consecutive episodes.

At this point we can look at the number of episodes that have been necessary to achieve such condition, as the qualitative metric for the model training.
The DQN model architecture proposed in the Lab 7 gave us good results, also compared to other more complex architectures with different layer size, so we kept the model with three fully connected layers followed by the *tanh* activation function. The number of neurons in every layer is 128. Similarly, the gamma parameter for the long term reward, the learning rate and the batch size have been kept the same as the proposed implementation, as well as the SGD optimizer and the loss function. We noticed, instead, that we could greatly improve the training speed by changing the exploration profile and the policy used for the choice of the action from the output of the model. We experimented with many exploration profiles used with the softmax policy and the greedy policy. At the end we settled with the epsilon-greedy policy using a profile for the epsilon parameters that we called epsilon-skipping. It consists in alternating an exploratory episode with a high epsilon value, and a exploitative episode with epsilon set to zero. The model trained with this strategy showed to greatly reduce the number of episode necessary for reaching the stopping condition. We used epsilon equal to 0.1 for the exploratory episodes.
We also changed the reward penalties, with a linear penalty based on the position of the cart, and also a penalty on the angle of the pole with respect to the cart. This showed to speed up the training as well, by penalizing the states where the pole is distant from the vertical position.

## 2.2 CartPole training from pixels

For this task we focused on the reconstruction of a low dimensional state from the image information contained in the frame rendered by the environment. The idea behind this choice is that all the information that are useful for the training are the position of the objects (so the cart and the pole) and the derivatives of the position, so the velocity, acceleration and so on. By feeding to a (convolutional) network the frame or a list of subsequent frames, we provide those information together with not useful information about the cart and pole shape and other graphical elements. So the model would need to learn how to extract the useful motion information from the images and then use them to control the cart. But if we implement a function that extract that information from the frames, we are using our prior knowledge about the problem and its physical properties so we don't need the model to learn them from scratch. Ideally the results obtained with this approach cannot be worse than the result obtained with a model trained on the entire images, and we save a lot of time and computational power.
Another problem that we must consider while implementing the training from the images, is that even if we perfectly reconstruct the motion information from the frame, we only have access to the spatially quantized data provided by the pixels. In fact the original state provided by the environments has a level of precision that is only bounded by the numerical precision of the variables it returns, while the image provides a maximum quantity of information depending on its resolution. To make an example, the position of the cart provided by the environment state is described as a double precision number in the range [-2.4, 2.4] so the possible values it can assume are in the order of $10^{14}$, while the cart positions that we can retrieve from the images are 600, since the resolution of the frame is $400 \times 600$. This means that if we start from the pixels, the training will be more difficult than using the provided states, even if we implement the best reconstruction of the state from the image.
We define the function *state_from_image* that takes in input 3 consecutive frames and return a six dimensional observation state. The returned values are

1. cart position, with values in the range [-3,3],
2. cart velocity,
3. cart acceleration,
4. pole position relatively to the cart position,
5. pole velocity,
6. pole acceleration.

This function has been implemented using only numpy methods and not other computer vision libraries with the intention of making it as fast as possible.

For this task we implement a different model for the Deep Q network, called Dueling Deep Q Network[1]. So the model contains two fully connected layers with ReLU activation that provides the *features*, then other two fully connected layers provides the *values* and the *advantages* starting from the feature. The model return a single value that is the summation of the *values* and the difference from the *advantages* and the *advantages* mean. This architecture should facilitate the training of complex reinforcement learning problems[2], and in fact it increased the performances of our training, compared against the simpler architecture used in the previous problem.
As before the model showed better result with the epsilon-greedy policy and we let it train using the epsilon skipping policy, with epsilon equal to 0.1 in the exploratory episodes. We apply linear penalties on the cart position and on the pole position during the training.

## 2.3 Training of the Acrobot-v1 environment

For the last part we implemented the training of another classical control environment provided by OpenAI. The Acrobot-v1 environment presents a double pendulum that can be swung applying a torque on the joints, with the aim of makeing the second arm surpass a horizontal line positioned above the the system. The state has 6 dimension and it consists of the sine and cosine of the two rotational joint angles and the joint angular velocities.
This game dynamic is the inverse of the CartPole, indeed the Acrobot keeps swinging until it reaches the target horizontal line or for a maximum of 500 frames. The scores are negative, and the best result is obtained with a lower number of frames in an episode. There is a lower bound on the performances (score equal to -500 that is 500 frames without reaching the objective) and there is no higher bound other than the trivial 0 frames. But we need a higher bound condition in order to evaluate the performance in our training framework, so we define a satisfying performance if the Acrobot reach the objective in less than 100 frames (that means a score greater than -100). We want to stop the training when the moving average of the score evaluated in 10 episodes is stabilized over the -100 score.
As before we see that a Dueling Deep Q network provides more stable results, using this time a smaller layer size of 64 neurons. The epsilon-greedy policy with the epsilon skipping profile let us obtain a faster training. We use epsilon equal to 0.06 in the exploratory episodes. Due to the

[1]implementation based on this Towards Data Science article: `https://towardsdatascience.com/dueling-deep-q-networks-81ffab672751`

[2]Dueling Network Architectures for Deep Reinforcement Learning, Ziyu Wang, `https://arxiv.org/abs/1511.06581`

different nature of the problem, we do not apply penalties during the training based on the state values.

## 3 Results

In the improved training of the CartPole environment we reach the stopping condition before 500 episodes, as we can see in Figure 1. The model starts to learn quickly after 400 episodes and the score stabilizes on the winning score after 470 episodes. It is a considerable improvement over the training proposed in the Lab 7 that took circa 900 episodes to stabilize in the winning score. The final test executed using epsilon zero on ten episodes shows the winning score of 500 in every episode, confirming the stability and quality of the final result. Looking at the videos incorporated in the notebook after the final test we can see that the cart is doing a very good job keeping the pole very stable in the vertical position.

The most challenging task that is the CartPole pixel based training is solved after a long training of 2000 episodes. The training remains very noisy until the end but we assist at the frequent occurrence of the winning score after circa 1200 episodes and the moving average of the score evaluated on 100 episodes tends to increase in the second half of the training. After 2000 episodes we do not encounter the stopping condition (ten consequent winning scores) but the final test performed with 10 episodes with epsilon zero shows the winning score in every episode, so we consider the problem solved and the game won.
We need to notice also that the epsilon skipping policy increase the noise in the training scores, since the exploratory episodes generally obtain a worse score than the exploitative episodes. So we can plot them separately, and in Figure 3 we can see separately the moving average of the exploratory and exploitative episodes. We notice that the exploitative episodes settles near the winning scores after circa 1900 episodes.
If we observe the video generated after the final test we can see that the cart is able to keep the pole balanced for the time necessary to win, but it is much less stable than the cart that we observe in the results of the first part of the project. This is probably a consequence of the loss of information during the pixel quantization, that prevent the model from successfully control the pole movements that are smaller than a single pixel.

The Acrobot training can be finished faster than the CartPole one, in Figure 4 we can see that our condition for a satisfying result is accomplished after 140 episodes. After this the final test, performed with ten episodes with epsilon zero, gives 9 results equal or higher than -100, with the best one at -76, and only one slightly worse result with -114. So the average of the final test results is 92.6 and we consider it a good result. In the videos of the final test we can see that the model is able to swing the arms with the right timing necessary to quickly raise them over the target line.

## 4 Appendix: Figures

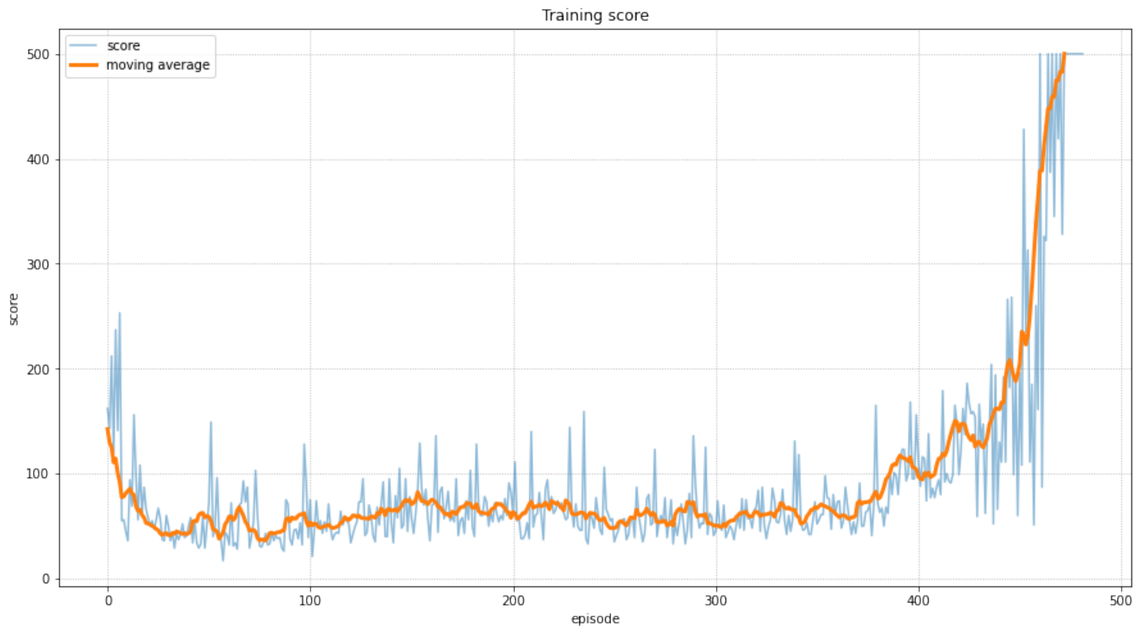

Figure 1: Training score over the episodes of the improved CartPole training.

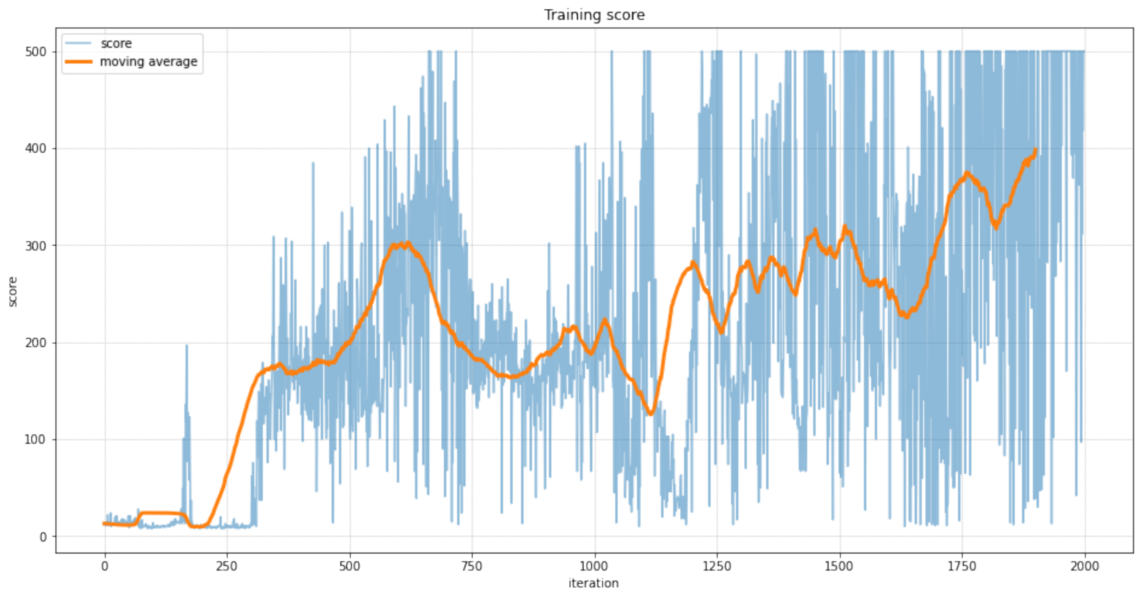

Figure 2: Training score over the episodes of the pixel based CartPole training.

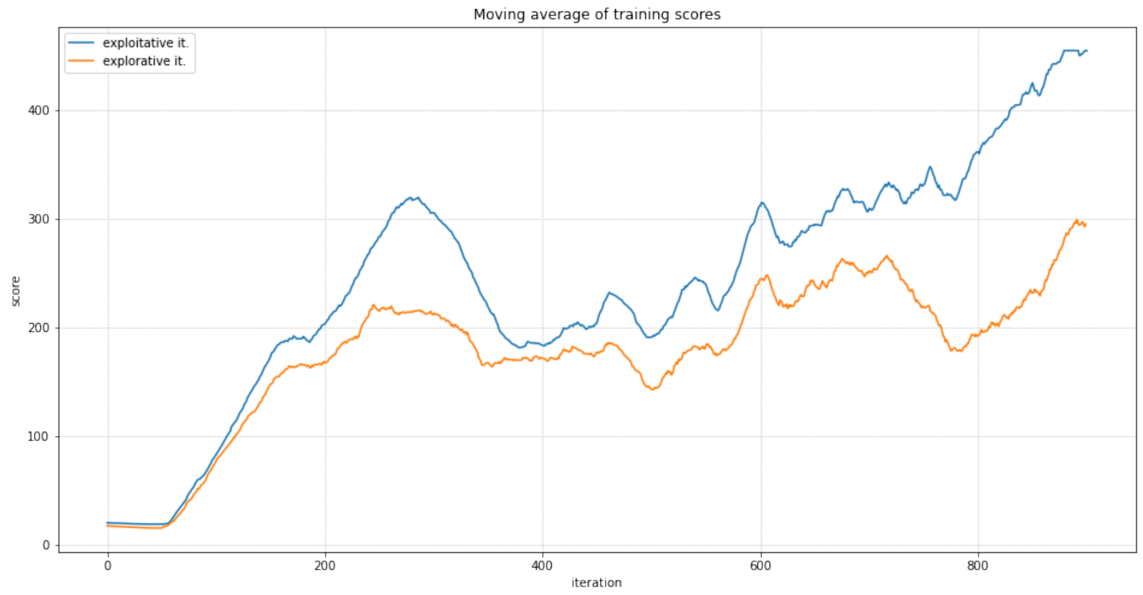

Figure 3: Moving averages of the explorative and exploitative episodes during the pixel based CartPole training.

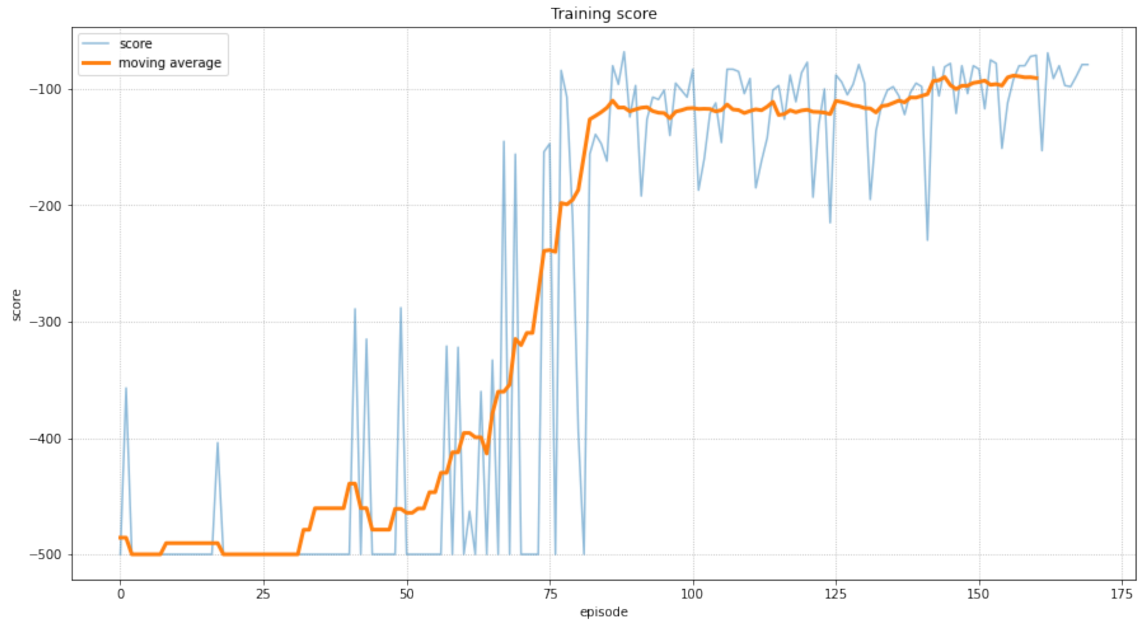

Figure 4: Training score over the episodes of the Acrobot training.