NNDL Homework 3.
Deep Reinforcement Learing

Enrico Convento – ID: 2023572

February 12, 2022

1 Introduction

In this homework, Reinforcement Learning were dealt using *OpenAI's gym* environment. The homework is composed by three main part, in the first part it is being considered the *Cart-Pole* environment when high level knowledge of the environment is available, the second part deal with the same environment but in this case only dealing with raw pixels and in the third part the *Acrobot* environment is being solved. The report is organized in two main sections, in Sec.2 the methods used have been introduced, then in Sec.3 the results obtained have been shown and described.

2 Methods

Cart-Pole As said in the introduction the first environment used is the *Cart-pole*, the associated structure is simple, it consists of a cart with a pole attached that moves on a rail, and it could only move right or left. The aim of the game is to keep the pole standing for a number of episodes greater than 500, the game ends when the pole falls ($|\theta| > 12\,\mathrm{deg}$) or the cart goes out of the screen($|x| > 2.4$).

2.1 Cart-pole with state information

The first approach considers high level information associated to the state of the cart, which is composed of:

x	cart position
$\dot{x}$	cart velocity
θ	pole angle
$\dot{\theta}$	pole angular velocity

In this case the state space has dimension 4 while the action space has dimension 2, since the agent can only choose to move left or right. In order to implement the DQN algorithm a fully connected network is being used, it is composed by 2 hidden layers in turn made up of 128 neurons, the input has the same dimension as the state space while the output has the same dimension of the action space. The algorithm is being trained using an exponential decay exploration profile shown in the Fig.6. Two versions are being implemented, the vanilla version in which the agent receives a reward of $+1$ at each step of the episode until failure and a more complex one in which the reward is being modified as follows *reward* = *reward* - $\alpha|x|$, where α is a weight factor. This kind of reward suggest to the agent that it needs to keep the pole close to the center of the screen. In order to get some improvement the second version of the algorithm was subjected to a study of the parameters that are involved in the training. The parameters have been tuned to try to get faster convergence, this is done randomly by changing the following parameters:

γ	0.9 to 0.99 (0.1 step)
lr	0.001 to 0.1 (log scale)
batch size	32, 64, 128, 256
k	5, 10, 15, 20

where γ is the discount factor, *lr* is associated to the learning rate and *k* is the number of episodes to wait before updating the target network. The best set of parameter is then used to train the agent in order to perform a test with zero temperature over 10 trials in order to observe if the training is being effective.

2.2 Cart-pole from raw pixels

In this section will be exposed the methods used to solve the previous environment from raw information. Solving the environment starting from raw pixels is more complex w.r.t. what has been done in the previous case, it presents several difficulties that must be faced with more complex methods. First the state space in this case is huge, the raw image is an immense mine of information, however the computational effort to deal with it is much greater than in the previous case, so a trade-off must be taken into consideration. The various frames are composed of three RGB channels of 400x600 size, which are difficult to handle and also saving them in the replay memory has some unwanted consequences. The image pre-processing is scheduled in the following list:

- Crop along x-axis between 100 and 500
- Crop along y-axis between 160 and 320
- Gray scale conversion
- Negative transform
- Down sampling the y-axis, i.e. keeping only one row of pixels every 6 of them, the x-axis not down sampled

In order to sum up what's were being done, the cropping and the color transformations are used to keep only strong informative part of the image, the down sampling is done only along the y-axis as it is the axis that contains less information. In this case, two convolutional networks are used to deal with the images, in particular they are both composed of three convolutional layers followed by two linear ones, the associated parameters have been manually tuned. The input image has then size 4x27x400, the initial 4 is used because those networks are fed with 4 consecutive frames in order to get information about the motion of the cart on the rail. In this case two classes have been created:

- class *Agent*, it is used to handle the agent-environment interaction
- class *Training*, it is used to perform the training

The main difference with respect to before is associated to the fact that the agent is fed with images, as in the standard algorithm we fed the policy net with the state (image) and compute then the associated q_value in order to finally get an action that will be applied. After executing the action chosen, the reward associated is being observed and modified, and then all the necessary updates are done. When it is said that the reward is modified, it means that the reward obtained is handled by the function *modifier*, which is a function of class *Training* used to adapt the reward to the problem. In particular it is modified as follows :

$$reward = reward + w_{position} * |position|^{position\_exp} + w_{angle} * |angle|^{angle\_exp}$$

The algorithm is then trained for 1000 epochs exploiting an exponential exploration decay as previously for the first solution. The reward function used is

$$reward = reward - 1.4 * |position|^2 - 7 * |angle|$$

The agent take also a -10 penalties if the pole falls. As said the training is managed from the previously introduced class, the training is basically the same of before but in this case is also being added another reward modification, basically if the agent is able to beat consistently the game a reward is given to him. More version of this have been implemented in order to improve the results, in particular two more versions with regularization have been implemented.

2.3 Acrobot

Lastly, it was chosen to deal with another environment, in particular the Acrobot environment was chosen because the state space is larger than the Cart-Pole, but it however does not have the same complexity of dealing with images. The environment is composed of a double pendulum composed of two bars and the joint between them is motorized, substantially the agent can apply three actions corresponding to + 1, -1 and 0 which correspond to applying torque in one or in the opposite direction or not. The state space is parameterized by $[cos(\theta_1), sin(\theta_1), cos(\theta_2), sin(\theta_2), \dot{\theta}_1, \dot{\theta}_2]$ where θ_1 represents the angle formed by the first bar w.r.t. in the rest position, while θ_2 is π minus the angle formed between your bars.

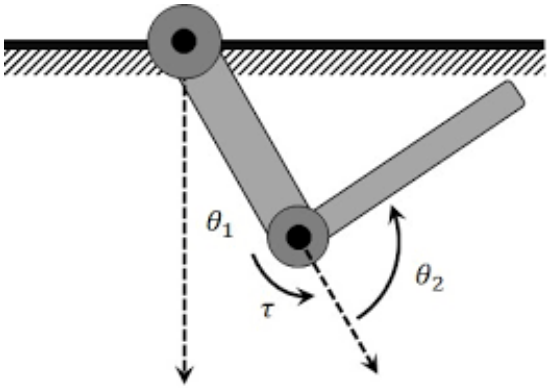


Figure 1: Acrobot representation

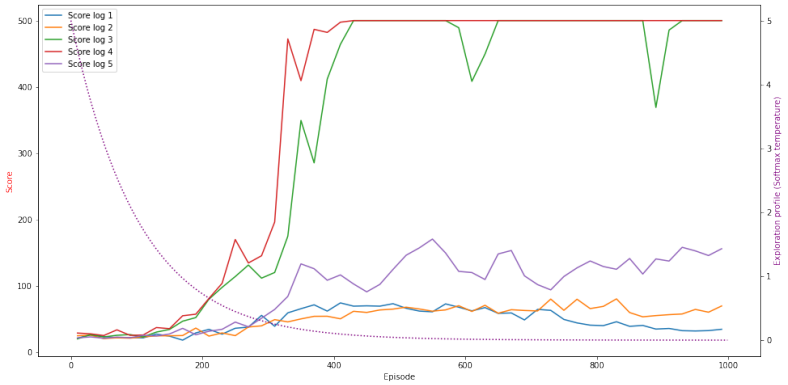
The object of the game is to obtain an overall height of the pendulum $h > 1$ for more than 500 steps. In each time step, it receives -1 as punishment if the goal is not reached, each episode is composed of 500-time steps, so the worst cumulative reward is -500. In the vanilla version, which uses only the reward of -1 for each step until the game is terminated, a simple fully connected network made up of two layers was used. A second version has been implemented, in which the reward changed from $r = -1$ to $r = w_h(1 + h)^{e_h}$ in particular $w_h = 0.5$ and $e_h = 1$. Also a further term was added, namely *'end_state_penalty'* $= 10$ has been added to reward of the agent when he manages to resolve the environment. In both cases, an exponential temperature decay has been used.

3 Results

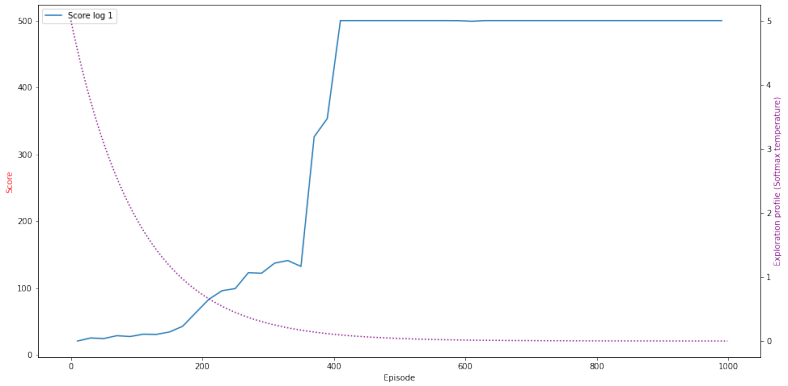
3.1 Cart-pole with state information

Using the high-level information provided make this task really easy to be solved, in the vanilla version in which the agent receives a reward of $+1$ at each step of the episode until failure, the agent is not able to solve the game, the associated training is slow and strongly dependent on the initialization of the net, the results can be observed in Fig.7. In order to get some improvement, the reward has to be modified as said above, in this case the agent is able to solve the environment consistently, the results are shown in Fig.7. The study associated to the parameters' exploration is composed of 5 trial which returns interesting results. The scores obtained during this study can be observed in Fig.2(a).

Comparing the results obtained from Fig.2 and Fig.3 it can be observed that the best results are obtained considering the third and the fourth set of parameters, which share two important aspects, both have a high learning rate, this is certainly linked to the speed with which the agent learns while the other is the number of steps after which the network is updated, i.e. a value equal to 5 is used. The best set of parameter it has the following setup gamma = 0.97, learning rate = 0.0599484, batch_size = 32 and update the target net after 5 steps. The associated network is then re-trained in order to better observe the results, and it leads to what can be observed in Fig.2(b), basically the



(a) *Optimization study results*



(b) *Best net result*

Figure 2: Cart-pole scores

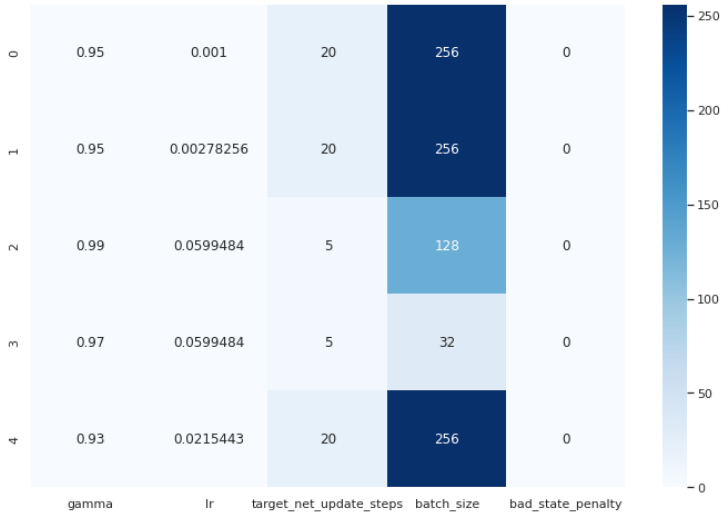
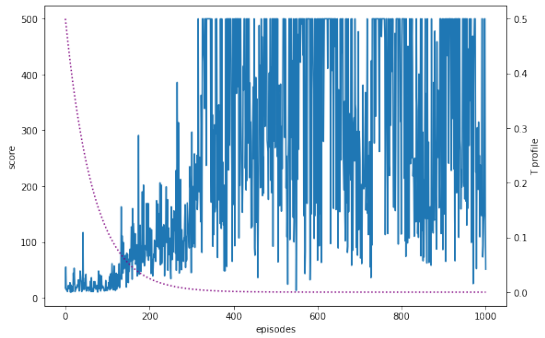
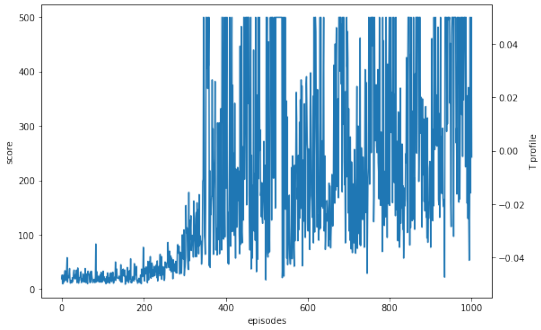


Figure 3: Parameters

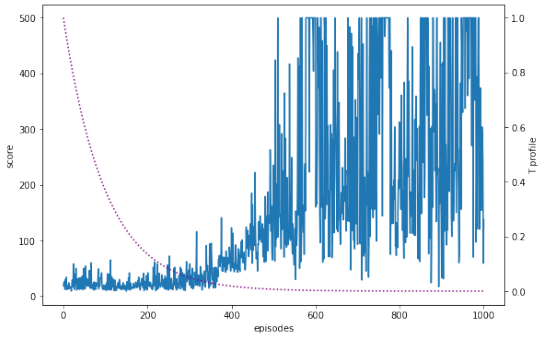
trained agent requires approximately 300 episodes to be able to beat the environment consistently. The plotted score reported in the figures are averaged using a time window of length 20 because the original signals have a big variance, so exploiting this trick is useful in order to make the plot more clear. The final test done at zero temperature highlights the fact that the training was effective, as the agent is able to obtain the maximum score on the 10 attempts given to him.



(a) *Version 1: no regularization*



(b) *Version 2: weight decay*



(c) *Version 3: weight decay + dropout*

Figure 4:

3.2 Cart-pole from raw pixels

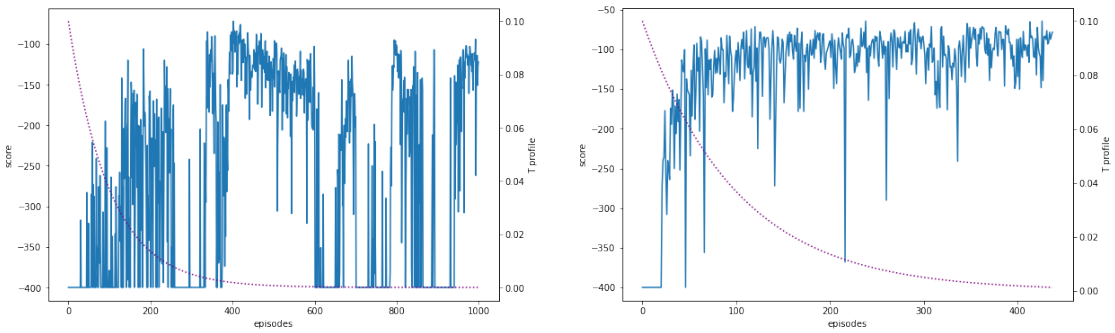
Analyzing the results it is observed that the agent generally begins to win after 300 steps, however the behavior is not the best in particular it is not at all stable but on the contrary high variance results are obtained, this can be observed in Fig.4(a), in Sec.B of App. are reported the averaged results over a 20 episodes window, from that is quite evident that the agent is not able to consistently beat the environment but it only provides very high variance results, looking that at the averaged results the means seem to settle around 300 score. One of the causes associated with this is that the crop of the images may have been excessive since the crop performed remove all the superior part of the cart which should help to get information about the angle, it could be interesting to feed the network with more information about the superior part of the frames in order to provide more informative data, however, the computing power is limited. Another cause could be linked to the fact that the frames provided are very similar to each other, and this could lead to overfitting. The algorithm was tested at zero temperature, however this causes very poor results, the average score is approximately 250. As said and as it can be observed in Fig.4(a) the results present high variance, this is a sign of overfitting, because of that two more version have been implemented, both make use of regularization to improve the training. The two version exploit the following regularization techniques:

- weight decay
- weight decay and drop out

The results obtained are interesting and can be observed in Fig.4, in particular it can be observed that more regularization is introduced more the high variance part of the training is moved far in the training. More precisely, in the second version the strong oscillations starts after 300/400 episodes while regarding the third version they start after 500/600 episodes. This confirms the fact that the overfitting is one of the cause that leads to the high variance phenomenon observed in all the training. The results in terms of testing are not good at all but the second version lead to an average score of approximately 300, which is a small but good improvement w.r.t. to the first version while the third

version, although the training looks better, leads to an average score of 250. Notice that as in the high-information version, in order to solve the environment the agent require some modification on reward function, in particular associated to the cart state. The use of this information introduces a bit of cheating, a solution to this using only raw pixel would be to estimate the position of the state in some way. A version in which the auxiliary state was not used was also tested, however in this case no improvement can be obtained at all.

3.3 Acrobot



(a) *Vanilla version scores* (b) *Modified reward version score*

Figure 5: Acrobot scores

In this section will be analyzed the two-way of solving the *Acrobot* environment exploited. The simpler version is able to solve the environment, but not consistently, the results are very noisy, and it can be observed the reward used is not enough infromative to get better results, the results can be observed in Fig.5(a). The second version that add the information regarding the high shows good improvements. Regarding this case, the reward function has been modified as introduced above since the agent is not only penalized as in the first case, but he is also rewarded based on the high he can reach. The results associated to this second version can be observed in Fig.5(b). From the results we can notice that the score value settles around -100 in approximately 100 steps and in this case the variance is not as huge as in the previous case. The algorithm stops after 500 epochs because the agent is able to beat the environment 20 times consecutively.

After the training the agent is being test on 100 episodes with zero temperature(i.e. it always act greedy) and the mean score is equal to -86.04.

A Cart-pole with state information

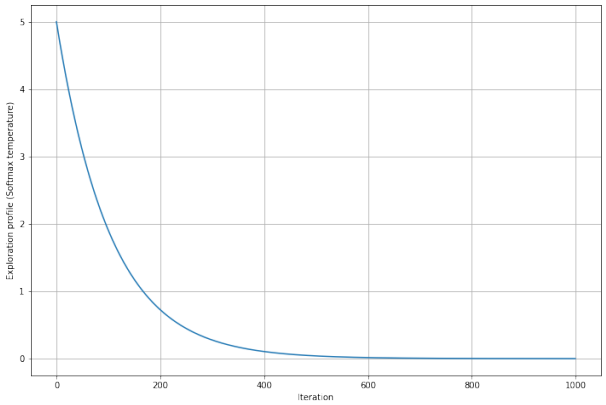
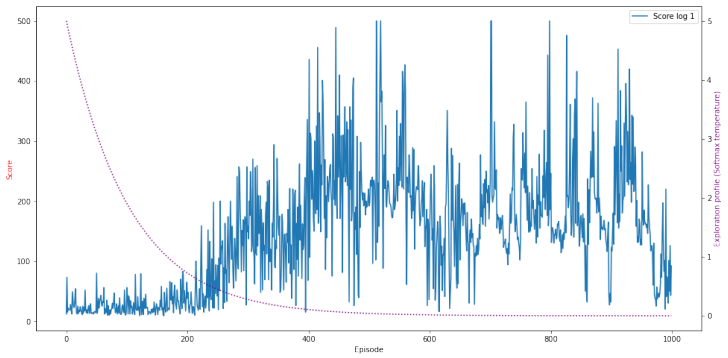
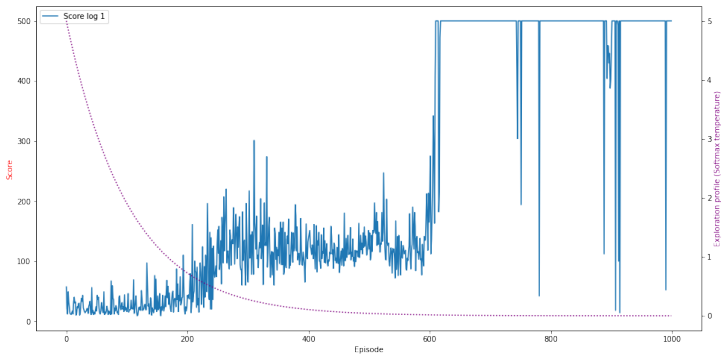


Figure 6: Exponential exploration profile



(a) *Vanilla version scores*



(b) *Modified reward version score*

Figure 7:

B Cart-pole from raw pixels

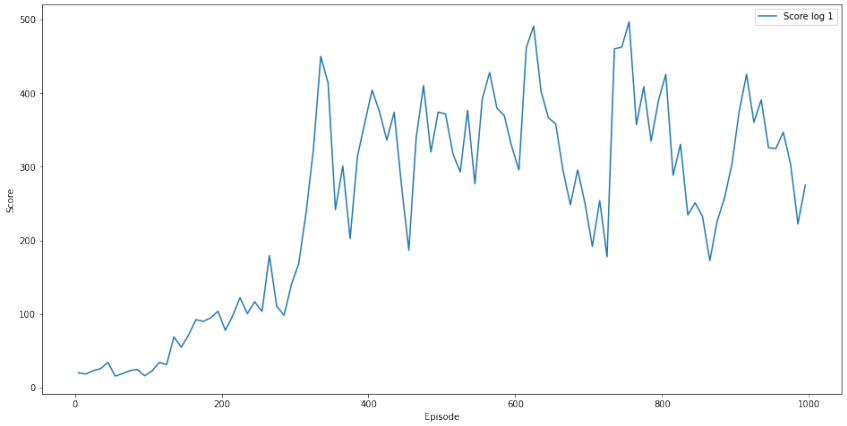


Figure 8: Raw information average results