Deep reinforcement learning

Delfina Iriarte (1231682)

[1]University of Padova, Physics of Data

Abstract. *In this work, deep reinforcement learning techniques are applied using the* `Cartpole` *and the environment in* PYTORCH. *In particular, Q-learning model is used which is a model-free reinforcement learning that has the goal to find an optimal policy, in the sense of maximizing the expected value of the total reward. Furthermore, the concepts of replay memory and target network are introduced.*

1. Introduction

Reinforcement learning tasks is about training an agent which interacts with its environment. The agent arrives at different scenarios known as states by performing actions leading to rewards. The goal of the agent is to interact with the emulator by selecting actions in a way that maximises future rewards. Q-Learning is a method of finding the optimal policies: the agent learns to associate a value to each state-action pair, which takes into account both the immediate reward for that action and the future rewards.

Deep reinforcement learning consists of algorithms that are based on function approximation i.e there is some function that approximates the Q values for actions. The Q-values founded for the next state can be used as an estimate of future values. In particular, the optimal action-value function obeys an important identity known as the Bellman equation. In practice, this basic approach is totally impractical and instead the update step can be consider as:

$$Q(s_t, a_t) \rightarrow Q(s_t, a_t) + \lambda \left(r_t + \gamma \max_a Q(s_{t+1}, a_t) - Q(s_t, a_t) \right) \tag{1}$$

where s_t, a_t are the state and the action at time t, λ is the learning rate and γ the discount factor that takes into account how much future rewards matter.

Given the Q-values a policy is then needed. Many action selection techniques can be used for exploration such as epsilon greedy or softmax. Notice that exploration is crucial in the initial phases so that the agent find out as much as it can from the environment. If the agent is too greedy during the first episodes, it might get stuck in a local minima.

In this work, examples of reinforcement learning are implemented in PYTORCH. In particular, the environment studied are the `Cartpole-v1` and the `LunarLanding-v1` from the OPENAI GYM framework.

2. Methods

In this section, the environments, with its correspondent set of states, actions and rewards, are introduced. Furthermore, the network architecture consider for each of them are presented.

2.1. Cartpole

The Cart-Pole environment consists of a cart that moves along the horizontal axis and a pole that is anchored on the cart. At every step the observations are: position, velocity, angle and angular velocity. The set of actions of the cart are only two: move to the left or move to the right. In this environment, the reward is given as long as the pole is balance in which the agent gets $+1$. An episode is over as soon as the pole falls beyond a certain angle or the cart strays too far off to the left or right. An example of the rendered environment is shown in Figure 1.

Figure 1: Rendered environment of the Cartpole

The agent acts in the environment and explore the world. All the experience of the agent are stored in the **replay memory** inside a buffer space and the model is train by sampling out the batches of experiences randomly or prioritized experiences. By sampling from it randomly, the transitions that build up a batch are decorrelated. It has been shown that this greatly stabilizes and improves the DQN training procedure. Furthermore, **target network** is used to compute $Q(st + 1, at)$ for added stability. The target network has its weights kept frozen most of the time, but is updated with the policy network's weights every so often.

As an action selection, the **softmax with decreasing temperature** is used. Nevertheless, when the temperature is set to zero, the epsilon greedy exportation is applied which is a common choice in this models. In particular, an exponential decay for the exploration profile is used with an adjust parameter called `init_value`. The stochastic gradient descent (SGD) but without momentum is used as optimizer of the network. Finally, it should be also mentioned that the reward also has a linear penalty when the cart is far from the center since it is training to escape from the screen.

The network architecture of the model is shown in Figure 2. It takes as input the four state space dimension, then it pass through two fully connected layers. As activation function the `tanh` function has been used.

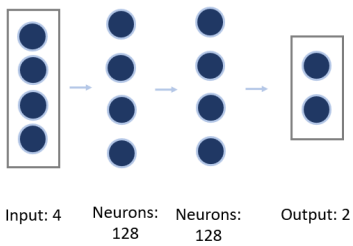

Figure 2: Model architecture for the cartpole environment

In order to speed up the convergence of the model, several exploration profiles are study with different starting temperature. Finally, the model performance is also compared to one obtained by a random agent.

2.2. Cartpole using screen pixels

Deep reinforcement learning (DRL) has been leading to state-of-the-art performance to learn control policies for a wide range of applications. The CartPole task (Section 2.1) is designed so that the inputs

to the agent are 4 real values representing the environment state (position, velocity, etc.). However, neural networks can solve the task purely by looking at the scene. Strictly speaking, states are present as the difference between the current screen patch and the previous one. This will allow the agent to take the velocity of the pole into account from one image. Unfortunately this does slow down the training, because we have to render all the frames. Therefore, as **input the screen** it is provided. Notice that the image is quite large $(800 \times 600 \times 3)$, therefore the following manipulation are done:

1. Convert the image in grayscale considering only one channel.
2. Crop the image removing empty undesired screen.
3. Rescale up to 1x85x200

The network model is a convolutional neural network as shown in Figure 3. As activation function the ReLu has been used for the convolutional layer and tanh for the fully connected ones.

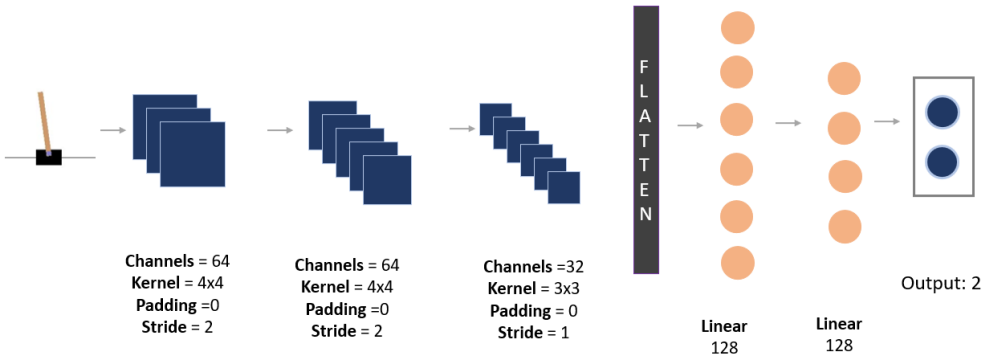

Figure 3: Model architecture for the cartpole environment using screen as input.

2.3. Landing A Rocket

In this section, Q-learning agent is tasked to learn the task of landing a spacecraft on the lunar surface given by the environment called `LunarLanding-v1`. A picture of the rendered environment is shown in Figure 4. The landing pad is always at coordinates (0,0). The states of the systems are the following:

1. `s[0]`: horizontal coordinate
2. `s[1]`: vertical coordinate
3. `s[2]`: horizontal speed
4. `s[3]`: vertical speed
5. `s[4]`: angle
6. `s[5]`: angular speed
7. `s[6]`: 1 if first leg has contact, else 0
8. `s[7]`: 1 if second leg has contact, else 0

whereas the the action available are the following discrete ones:

1. do nothing
2. fire left orientation engine
3. fire main engine
4. fire right orientation engine

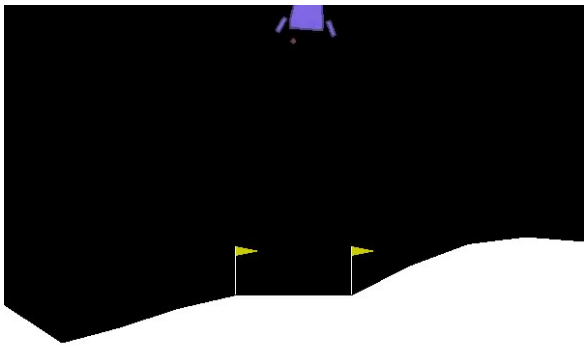

Figure 4: Rendered environment for the lunar landing environment.

Reward for moving from the top of the screen to the landing pad and zero speed is about 100-140 points. If the lander moves away from the landing pad it loses reward. The episode finishes if the lander crashes or comes to rest, receiving an additional -100 or +100 points. Each leg with ground contact is +10 points. Firing the main engine is -0.3 points each frame. Firing the side engine is -0.03 points each frame. Solved is 200 points.

Again, a random agent will be train in order to make comparison with the deep Q agent. The network architecture for this model is similar to the one of Section 2.1, but in this occasion is composed by 3 linear layers as shown in Figure 5. As activation function the tanh function is used. Additionally, a linear bonus for going downward is consider in order to avoid the space to fly indefinitely.

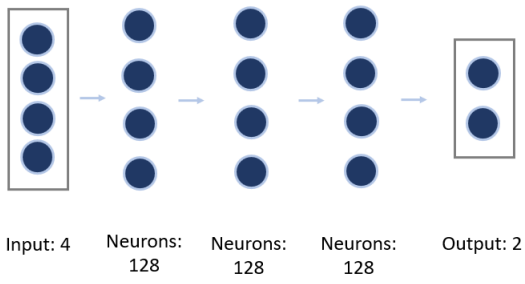

Figure 5: Model architecture for the lunar landing environment.

3. Results

In this section, the main results are present and discuss.

3.1. Cartpole

Figure 6 presents the training score per iteration for a random agent and its frequency. As it can be observed, it's impossible to solve the environment using this approach. Even tough sometimes is lucky enough to get a reward of almost 75, it can be noticed that the agent is not learning from their experience. In fact, the average performance is of ~ 20.

Figure 6: Training score and frequency of the rewards for the Cartpole environment when training a random agent.

The training score and its frequency for several hyperparameters are shown in Figure 9 and Figure 10. It can be observed that the trend line is positive, since the performances increases over time and indeed the model is able to learn. The accuracy of the model highly depend on the hyperparameter choosen. In particular, the best parameter corresponds to the model which wins first which ocurrs in 436 number of episodes. The best parameters obtained were: `'gamma'`: 0.94, `'lr'`: 0.062, `'target_net_update_steps'`: 5,`'batch_size'`: 128, `'bad_state_penalty'`: 0 and `init_time`: 25. Finally, it should stress out that the training score oscillates a lot.

3.2. Cartpole using screen pixels

Figure 7 shows the evolution of the reward of the cartpole. It can be observed that the model is not able to win the game, the average test score is around 102. In fact, it is resemble to a random agent in the sense that there is not any positive trend and the algorithm is not learning.

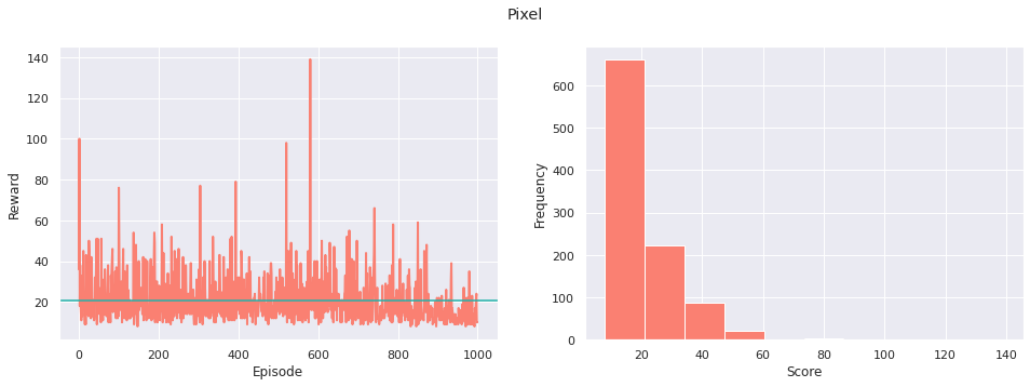

Figure 7: Training score and frequency of the rewards for the CartPole with screen input when training a random agent

3.3. Landing A Rocket

As in Section 3.1, the random agent for the model is not able to achieve high performance as shown in Figure 8, for the same reasoning as before. In fact, the rewards are on average negative.

Figure 8: Training score and frequency of the rewards for the lunar landing environment when training a random agent

Figure 11 presents the training score of the several parameters try in the model. As it can be observed, the environment is solved in almost all the cases. In particular for `'gamma'`: 0.98, `'lr'`: 0.010, `'target_net_update_steps'`: 10,`'batch_size'`: 128, `'bad_state_penalty'`: 0 and `init_time`: 9 it can be observed that the score with more frequency correspond to 200 which is the solved game.

4. Conclusion

In conclusion, deep reinforcement learning seems a challenging and promising method in order to solve ”games” by learning from the environment. Despite the efforts, most of the games were successfully solved by tuning the hyper parameters. Unfortunately, one of the main drawback for the model is the RAM consuming, specially when dealing with pixel images.

5. Appendix

Figure 9: Training score and frequency for the several hyperparametes for the Carpole environment

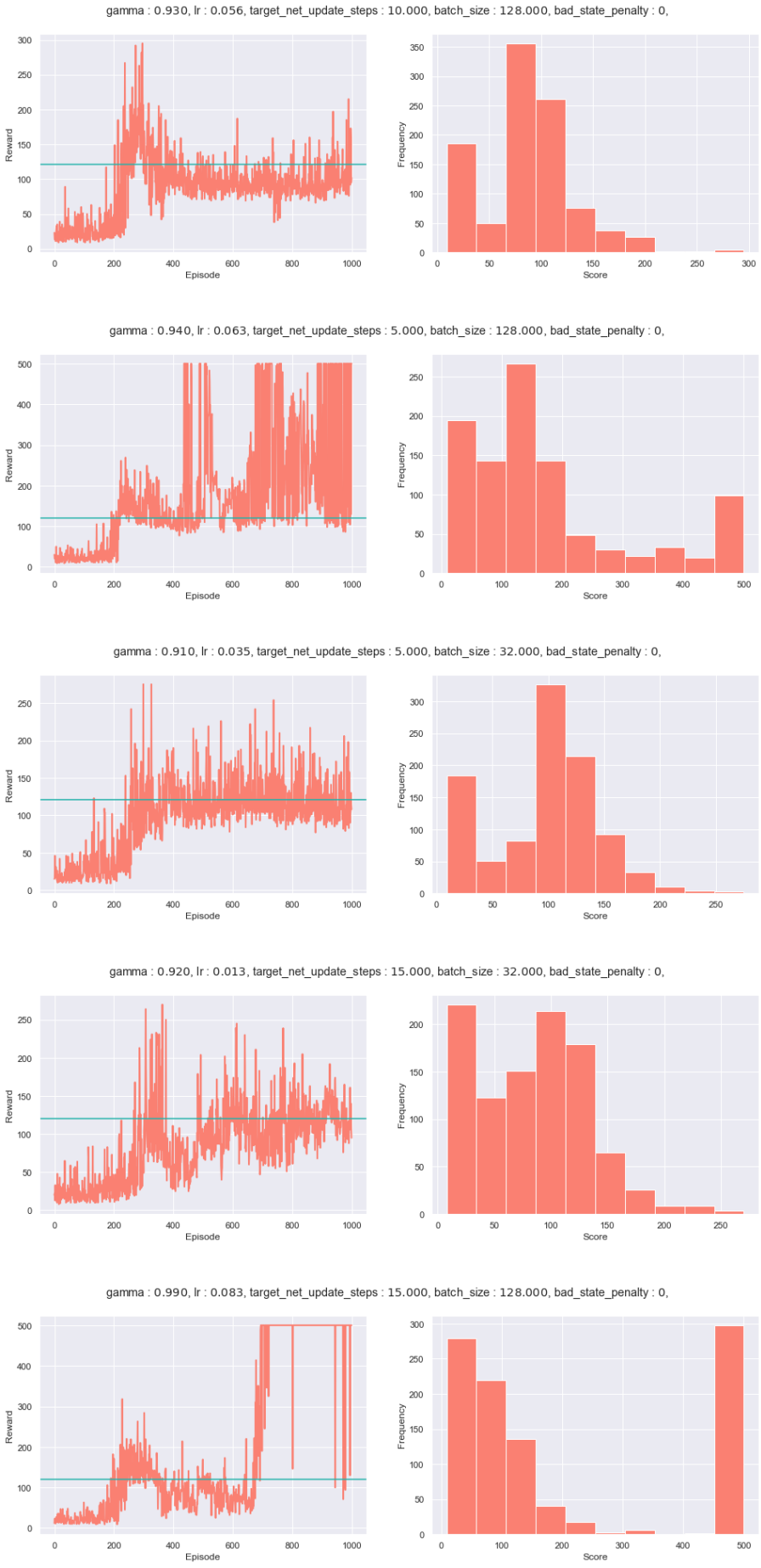

Figure 10: Training score and frequency for the several hyperparametes for the Carpole environment.

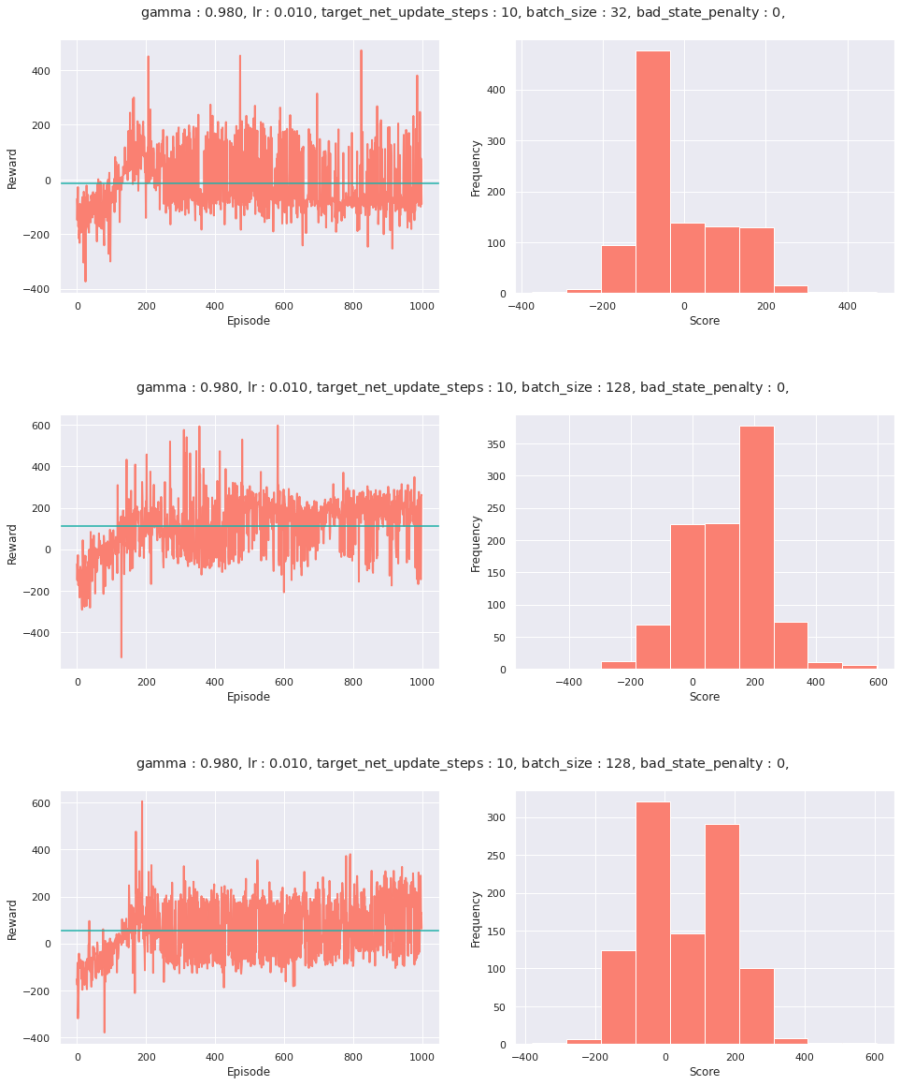

Figure 11: Training score and frequency for the several hyperparametes for the LunarLanding environment.