hw1-report

Federico Gambassi

March 2021

1 Introduction

In this report we are going to see the application of DQN framework on two Atari games, namely **Ms. Pacman** and **Lunar Lander**.

2 Question 1

The first task is to train the agent to play Ms. Pacman using a basic Deep QLearning algorithm which makes use of an experience replay stack in order to decorrelate batch elements, and a target network, which keeps fixed the regression objective for a number of steps, helping in reduce noise during updates. The default value of the most relevant parameters are

- episode length = 200
- batch size = 32
- replay buffer size = 1e6
- $\gamma = 0.99$
- lr = 1e-3
- ϵ = 1e-4

As we can see in Fig.1, the average return stabilizes around 1500 after 1 million steps.

The algorithm has been first tested on Lunar Lander (for faster debugging) for 500k steps, with default value parameters:

- episode length = 200
- batch size = 32
- replay buffer size = 50k
- $\gamma = 1$

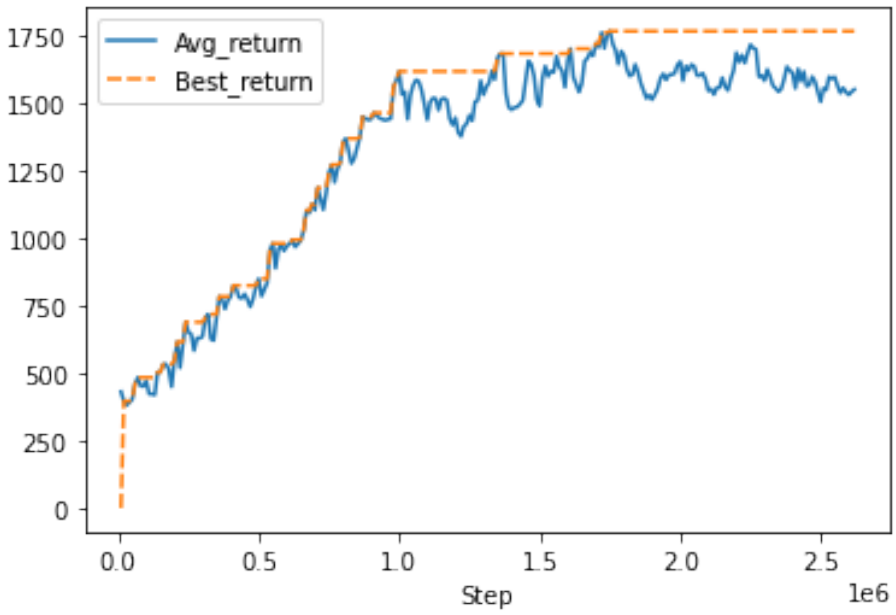


Figure 1: Ms. Pacman

- lr = 1e-3

As we can see in Fig.2, the average return reaches 100 towards the middle training phase, meaning that the spaceship lands correctly. Even for Ms. Pacman things go well, since the average return oscillates around 1500, which is a good result meaning that the agent is playing correct strategies.

3 Question 2

A drawback of QLearning is that Q-function seems to systematically think it's going to get larger rewards than it actually gets. This has an intuitive explanation: the problem is that in the target we take the maximum of the Q-function w.r.t. all feasible actions; we know that for two random variables X_1, X_2 it holds $E[\max(X_1, X_2)] \geq \max(E[X_1], E[X_2])$. So if our Q-function is noisy this can lead to overestimation of the actual function.
So, if somehow we manage to decorrelate the noise in the action selection from the noise in the evaluation step, this problem can go away. A possible solution is **Double QLearning**, in which two networks are used:

- $Q_{\phi_A}(s,a) \leftarrow r + \gamma Q_{\phi_B}(s', _{a'} Q_{\phi_A}(s', a'))$
- $Q_{\phi_B}(s,a) \leftarrow r + \gamma Q_{\phi_A}(s', _{a'} Q_{\phi_B}(s', a'))$

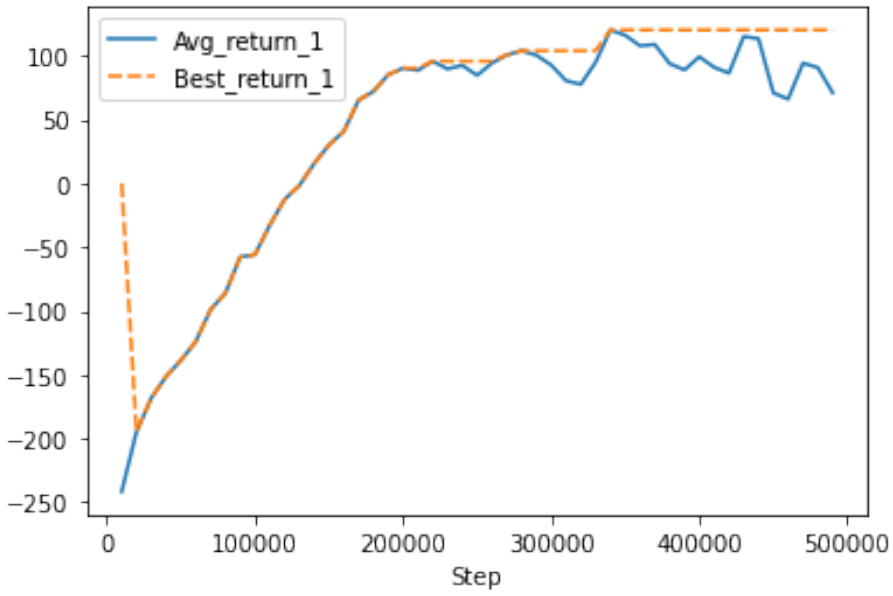


Figure 2: Lunar Lander

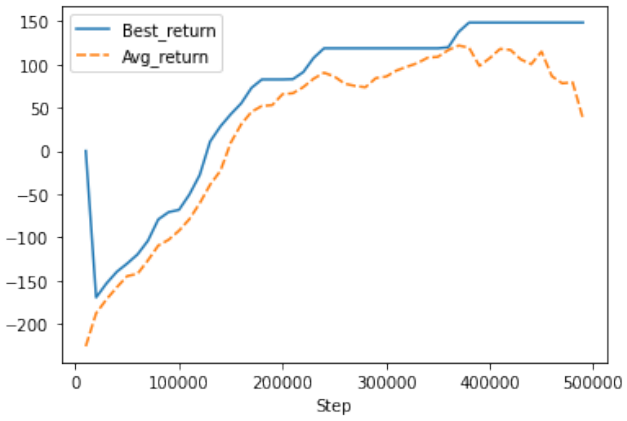
; in our case, the two networks are the target and the evaluation networks.
To compare the difference with and without using the double estimator, I have run both versions (vanilla and non-vanilla) with three different seeds each and then averaged the results, which are compared in Fig.3a-b :

We observe that results are more or less similar, except for the final part in which the double estimator version remains stable and doesn't deteriorate.

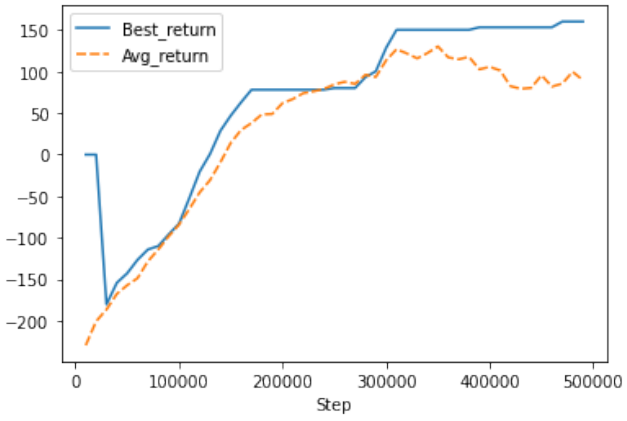
4 Question 3

Finally, I wanted to see the effect of learning rate tuning on Lunar Lander (the vanilla implementation). In Figs. 4,5,6 I have plotted the results with the nonoptimal learning rates, while in Fig. 7 we have a comparison of the average return for every lr setting. As we can see, either a very low value (lr=1e-3) or an intermediate one (lr=1e-2) works well; on the contrary, a too large lr prevent the agent to learn good actions, given that the best return stays constant below -250, and this is something I could expect.

Figure 3



(a) Vanilla-DQN: average over 3 seeds



(b) Double DQN: average over 3 seeds

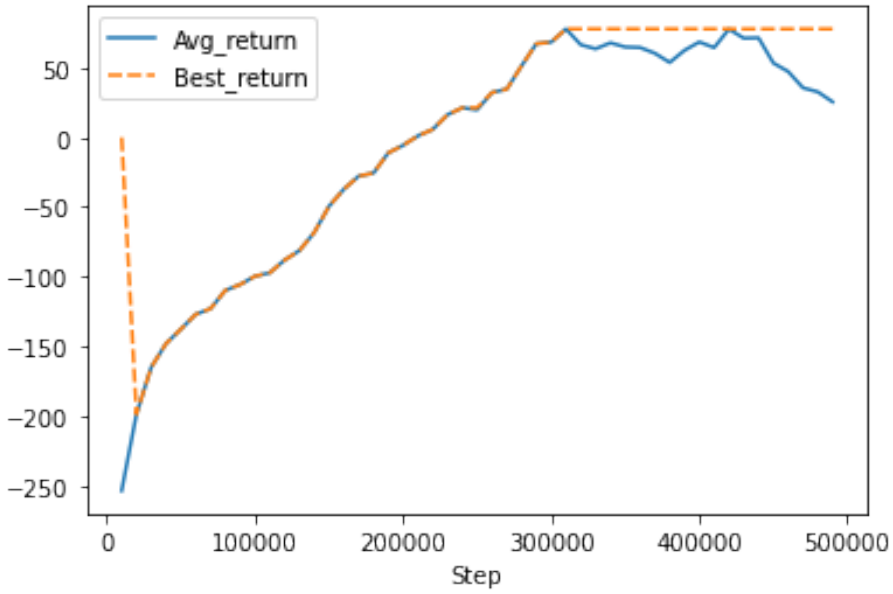


Figure 4: Lunar Lander - lr = 0.01

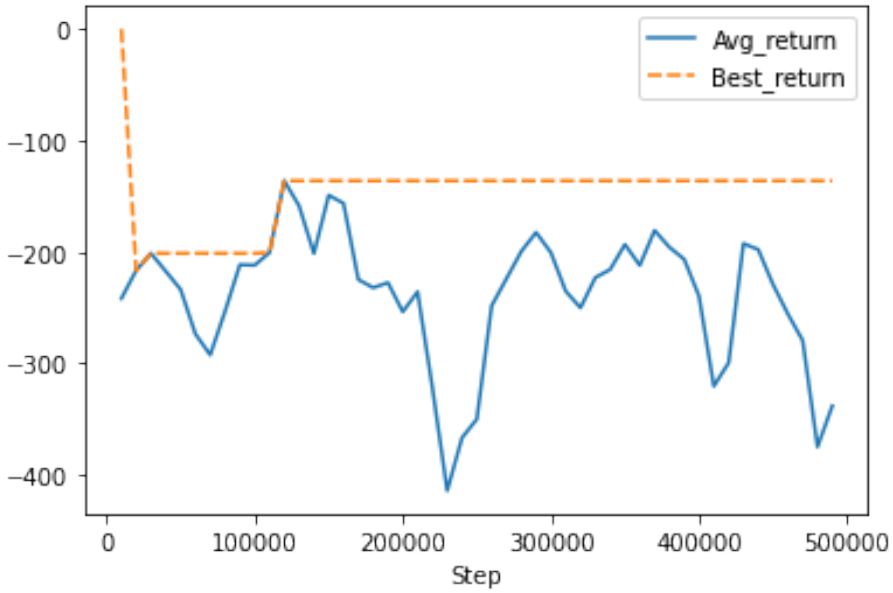


Figure 5: Lunar Lander - lr = 0.25

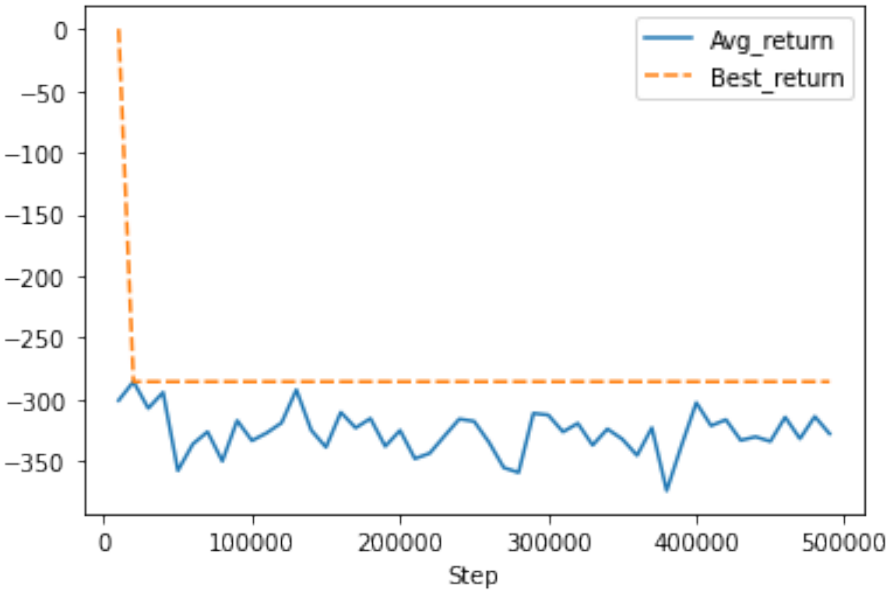


Figure 6: Lunar Lander - lr = 10

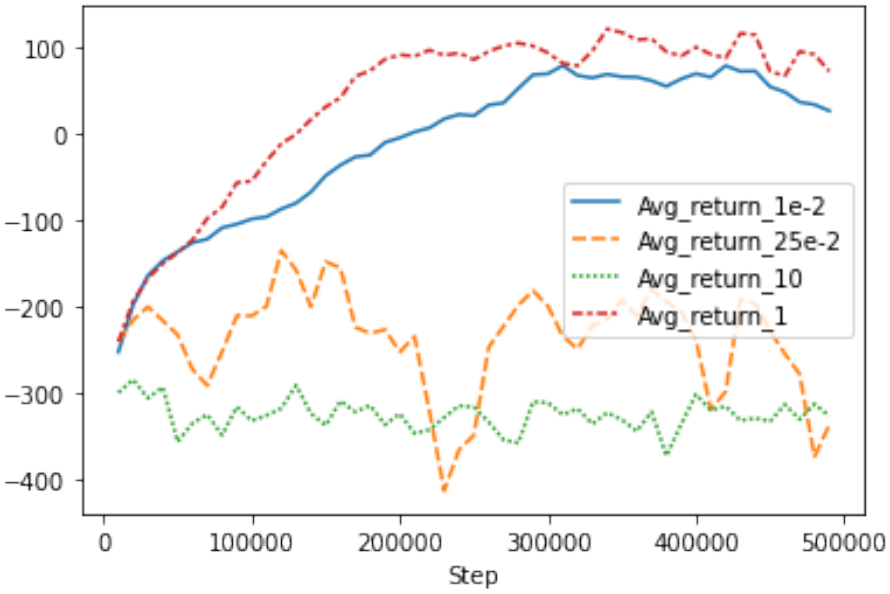


Figure 7: Lunar Lander - comparison