Neural Networks and Deep Learning 2021
Homework 3: Deep Reinforcement Learning

Matteo Grandin

January 2022

1 Introduction

The goal of this homework is to use deep reinforcement learning techniques, and more specifically Deep Q-learning, to solve problems from the 'gym' environment framework. Three different problems will be tackled in this homework:

- Solving the Cart-Pole-v1 environment using state dynamics variables: position, velocity, angle and angular velocity.
- Solving the Cart-Pole-v1 environment, but using only the image pixels to control the agent.
- Solving the MountainCar-v0 environment, using state dynamics variables: position and velocity.

2 Cart-Pole-v1

The Cart-Pole-v1 environment consist in a cart free to move along a line, with a pole attached to it, which can freely rotate around. The objective is to keep the pole balanced against gravity for more time steps as possible, in particular, as written in the documentation, the environment is considered solved when the average return is greater than or equal to 195.0 over 100 consecutive trials, however this report will focus on performance in a more qualitative way, in order to understand the dependencies on the hyperparamters and the exploration profile. The environment has 4 observable states: cart position, cart velocity, pole angle and pole angular velocity. The agent has only 2 possible actions: pushing the cart to the left or to the right.

The problem has been addressed using a simple fully connected network for the policy network and the target network. In order to manage the balance between exploration and exploitation an exploration profile has been used, adapted for both epsilon greedy policies and softmax with temperature policies. The exploration profile is characterized by an exponential decay in the number of episodes, both the initial values and the speed of convergence can be tuned to achieve the best performance.

The training process requires a lot of parameters tuning in order to reduce the number of episodes required to solve the environment. Two possible parameters combinations have been found: an 'aggressive' combination based on a very high learning rate and a more stable, slower combination. Stochastic gradient descent (SGD) without momentum has been used, to deal with the extremely dynamic setting of reinforcement learning. Huber loss has been used to train the networks, since it's a trade off between L2 and L1 losses.

2.1 Results

The aggressive combination (lr $= 0.12$, $\gamma = 0.9$, target net update steps $= 8$, exploration profile convergence speed $= 0.8$) perfectly solves the environment (maximum score of 500 in 50 episodes) after training for only 120 episodes, thanks to a very aggressive learning rate and an exploration profile focused on exploration. It's important to notice that this combination of parameters doesn't work if it's trained for more epochs as the learning becomes too unstable and the model starts overfitting on the replay memory buffer.

The more balanced combination (lr $= 3 \times 10^{-2}$, $\gamma = 0.95$, target net update steps $=$ 5, exploration profile convergence speed $= 6$) achieves perfect score on 50 episodes after training for 500 epochs.

The exploration profile greatly influence the training of the agent: if it converges too fast or starts from a temperature (or ϵ) value too low, the optimizer gets stuck in a local minimum and the scores don't improve. On the other hand, if there is too much exploration the network can't train on more advanced situations because a suboptimal action prevent the agent from reaching a particular state. In the case of the Cart-Pole-v1, for example, before trying to keep the cart in the center, the agent needs to be able to control the pole balance very well. During testing it has also been noticed that some exploration can help also in later stages of the training, in order to learn more advanced behaviors.

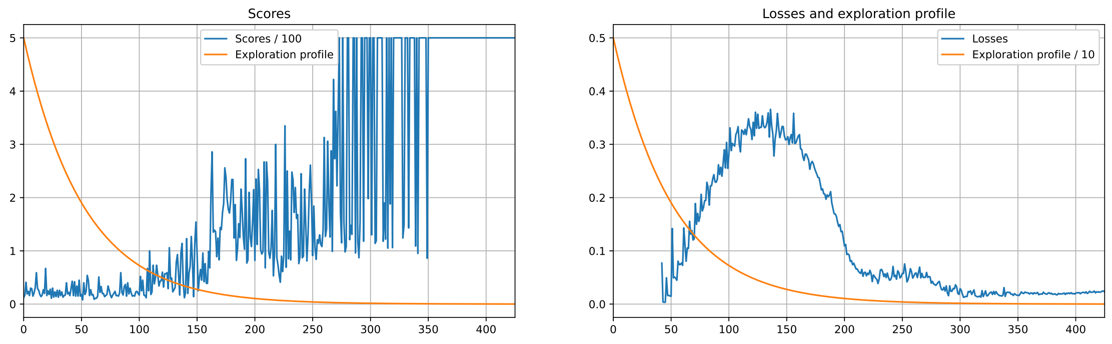

Figure 1: CartPole-v1: Scores and Losses relation with the Exploration Profile

3 Cart-Pole-v1 with Pixels

The problem of solving Cart-Pole-v1 using only the pixels from the rendering of the environment is much more difficult than the previous one. The problem has been addressed using a deep convolutional neural network composed of a convolutional section and a linear section. The environment rendering is preprocessed before being fed to the network: the image is cropped and centered on the cart, then it's converted to grayscale and brightness levels are modified. In order to capture the movement, 2 consecutive frames are subtracted. Some small markings are drawn to give a reference point of the center of the original frame (otherwise it would be impossible to estimate the distance from the center). The image is finally downsampled to 32 by 32 pixels, to keep the algorithm computational and memory efficient. In order to speed up training, several tricks and change of implementations were used to increase performance and speed, such as moving the heavy lifting section of the software and the replay memory buffer on the GPU.

This first naive architecture is extremely difficult to train and requires a lot of epochs to achieve unsatisfactory results. In order to improve on the design the network is finetuned, during training, by outputting, in addition to the action value function, also the predicted state dynamics (position, velocity, etc..) which is then evaluated against the true dynamics. This will allow to drastically improve the performance. The reward function also needs to be tweaked in order to focus the training on the more basic task of keeping the pole upright.

3.1 Results

The first version of the design is incapable of solving the problem. The finetuned version of the design instead, achieves much better results. After 600 epochs of training, the model achieves an average score of more than 400. Looking at the videos of the tests, however, it's possible to see that the control is very unstable, this could be due to a number of reasons: the network is not capable of correctly predicting the state dynamics, the image is compressed too much or the preprocessing is modifying the image in ways that generates instability in the network output. Due to the huge amount of resources spent to tackle this problem this issues remain unsolved, with more time different strategies could be tried, for example it would be possible to feed the network a sequence of images or a combinations of difference of frames and normal frames.

4 MountainCar-v0

In the MountainCar-v0 environment a car is placed at the lowest point in a valley and the objective is to make it reach a flag on the top of the hill to the right before the time runs out. The observation space is 2-dimensional: only horizontal position and velocity are observable. The action space is composed of 3 actions: accelerating left or right and not accelerating. The difficulty of the problem lies in the fact that the car is

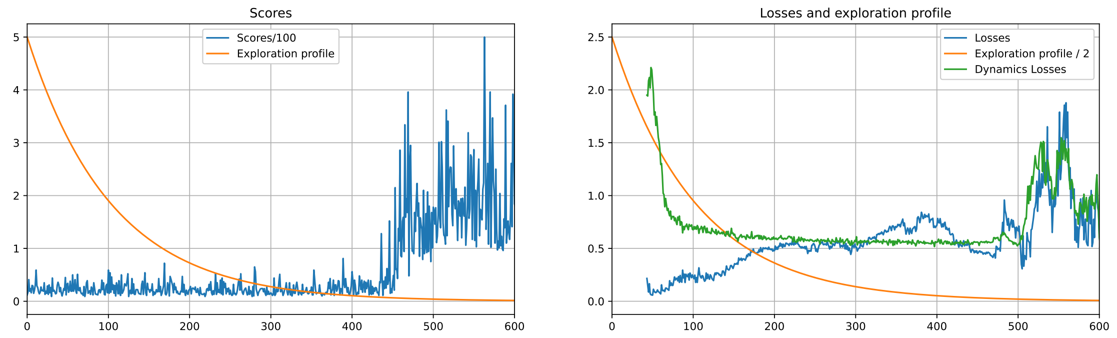

Figure 2: CartPole-v1 with pixels: Scores, losses and dynamics losses against the exploration profile.

incapable of directly reaching the top of the hill, but needs to swing back and forth in order to gain momentum. This specific fact makes this problem extremely different from the previous ones. In the Cart-Pole-v1 environment random actions were useful in the training process because there is a almost direct causality relation between the action and the reward; in this environment is practically impossible to reach the flag in time performing random actions. For this reason the key in solving this problem is a defining a good reward function to accelerate the training process. Several attempts were made to find an effective reward: checkpoints were used, linear and quadratic rewards based on the position, potential fields that gives more reward based on how far the car is from the bottom, and others. The one that turned out to be effective is slightly convoluted: the main term is related to velocity, the speed record of the car is hold in memory and every time in the episode that the car beats its previous maximum velocity it gets a fixed reward and one proportional to the difference in speed from the previous record. There is also a minor term related to the maximum position reached that behaves in the same way.

A simple fully connected network has been used paired with Huber loss, for this particular problem Adam optimizer was used, paired with a batch size of 4096, the underline idea behind this choice is to allow the network to learn more broadly since the task requires a lot of planning.

5 Results

After training for less than 300 episodes, the model is capable of perfectly solving the environment and achieves perfect score in all the 50 episodes of testing. If we check the videos (available in the notebook) we see that the agent has developed a very effective and secure strategy, however this is not the optimal strategy in term of solving the environment. The model doesn't converge to the optimal strategy because it's trying to maximize the designed reward but not the environment objective which is arriving at the flag as fast as possible. With more time and resources different strategies could be implemented to solve this issue: for example the designed reward could progressively fade as the score improves leaving only the environment reward, or could be completely removed after a predefined amount of epochs.

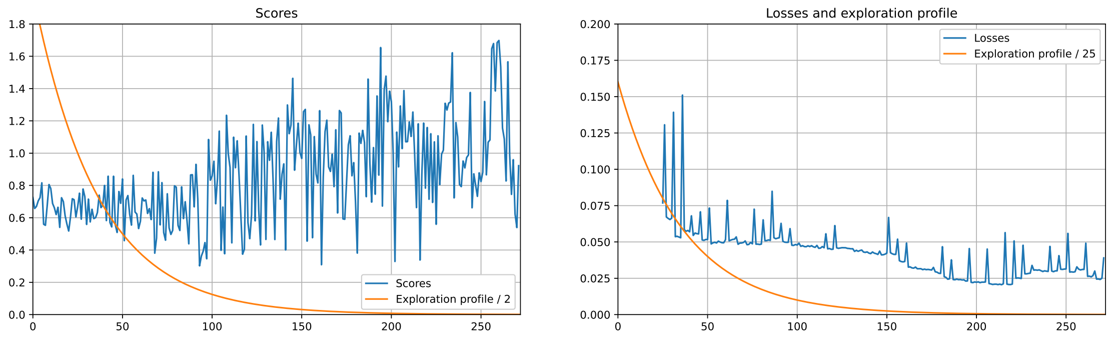

Figure 3: MountainCar-v0: Scores and losses against the exploration profile.