# Introduction

In this homework, we are going to train a Deep Reinforcement learning network using gym environment. The game that we used in this homework is cart pole. In the first part, we compare the effects of two different exploration profiles, Soft\_max and Epsilon\_greedy policies, in the learning curve of the RL agent.

…

In the second part of this homework, we trained the RL agent for the same problem, *CartPole*, but with a different *observation space.* In this part, the observation space will be the screen pixels of the environment.

# Model

The network that is used for studying the exploration profile is defined with named *DQN* which is a very simple fully connected neural network with two layers with 128 neurons in each layer. Because the observation space has four elements, which are shown in Figure 1, the model has input layer with four neurons

## Softmax Action Selection

This selection method is defined in a function named *choose\_action\_softmax. In the s*oftmax method, the network output is turned into the probability and with that probability, the action is selected not just by selecting the index of the maximum element. This is done by the formula shown in formula 1.

## C:\Users\user\Desktop\softmaxformula.png

## (1)

In this formula, is called temperature. Choosing a high temperature causes the actions to be nearly equal. In the case the temperature is zero it becomes greedy action selection.

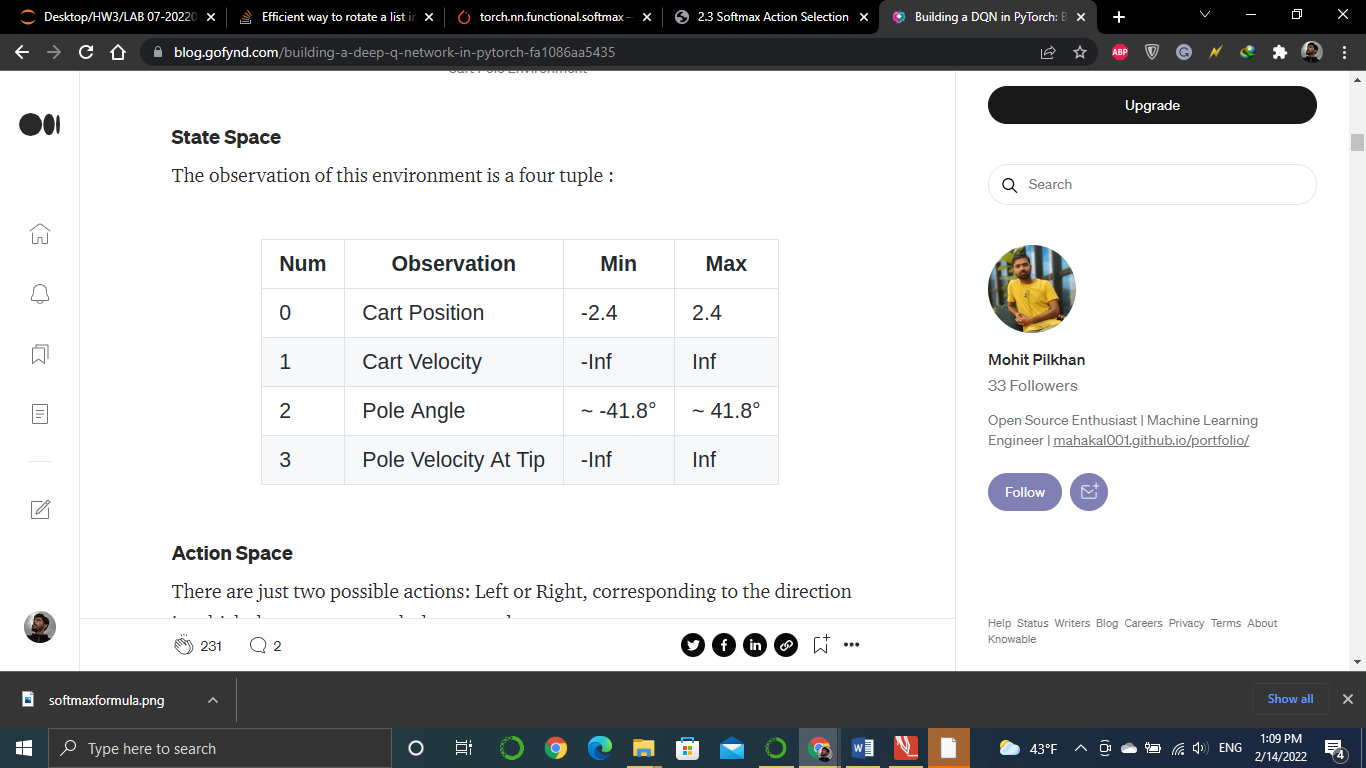
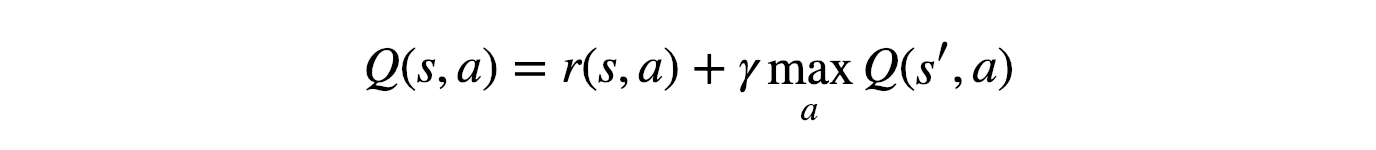


Figure 1. Observation space of cart-pole

## greedy action selection

In greedy model selection, with probability the action is the best possible action and with probability the action is randomly chosen from other possible actions (in this case other possible action is only one). This method is defined in a function named *choose\_action\_epsilon\_greedy.*

## Training loop

 The training is done in a *for* loop. In this loop first state is the initial state and then an action is chosen using one of the two above selection methods and then the current state, next state ( the state that is achieved by the chosen action), reward, and Q values are saved in the memory. Updating the network weights starts after there are 1000 samples in memory. During the update process which is done in the function named *update\_step,* q-values corresponding the actions taken are calculated by the policy network and then q-values of the next sates are calculated using the target network, and according to formula 2 the expected q-values are computed and by comparing the *q-values* and *expected q-values* the loss can be calculated.

(2)

## Using screen pixels for observation space

In this part, we changed the observation space and used screen pixels for the training process. The code for this part is written in a different Jupyter notebook named *HW3\_part2\_cartpole\_screen*. First thing that comes to mind for training a neural network on images is that the best way is to use a convolutional neural network (CNN). Therefore, I defined a CNN with three convolutional layers. The architecture used is shown in Table 1. By using the command *env.render(mode='rgb\_array').transpose((2, 0, 1))* the screen is returned. After that using the function

Table 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Number of filters | stride | Kernel size | Activation function |
| Layer 1 | 16 | 2 | 5 | Relu |
| Layer 2 |  | 2 | 5 | Relu |
| Layer 3 |  | 2 | 5 | Relu |

…

# Results

## softmax and greedy selection methods

In Figure 2, the learning curve for the RL agent trained over 2000 epochs with a two-layer fully connected network is shown. On the localhost, it took ***38 minutes*** to train the network. The Figure shows that after about 1350 epochs reached the maximum score. In Figure 3, the learning curve for the same network with the greedy selection method is shown. The epsilon in this training was set to 0.1 and it took approximately 30 minutes to complete the training. In Figure 4, both of the learning curves are shown in the same Figure and it can be seen that the learning curve for the Softmax selection method Works better and is more robust for our problem.

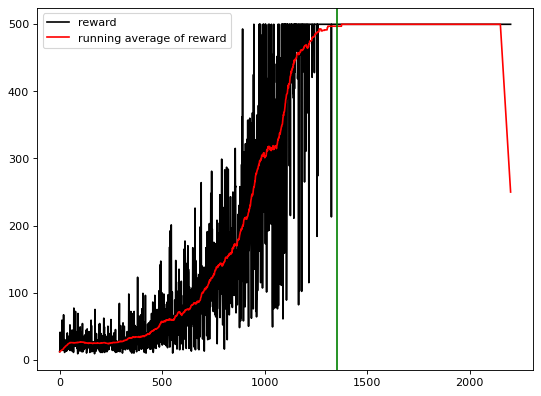


Figure 2. The learning curve of the network using the Softmax selection method

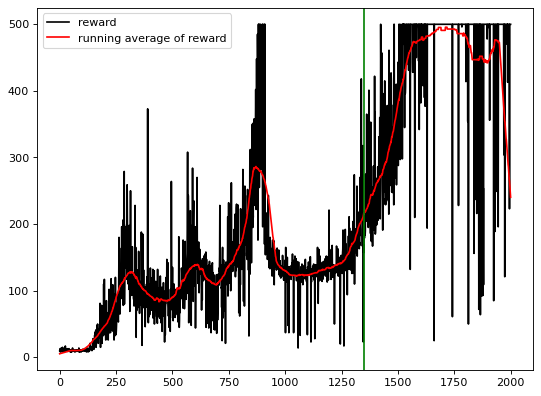


Figure 3

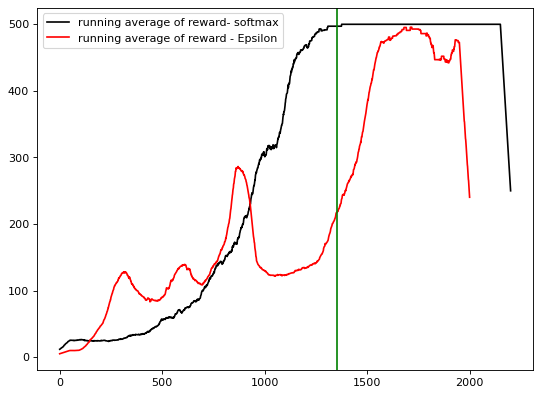


Figure 4

## Network with 3 fully connected layers

One of the hyperparameters that we can explore its impact on the learning curve is the depth of the network. In this part, we defined a network with 3 fully connected layers in which the first and second layers have 128 neurons and the third layer has 68 neurons. The learning curve is shown in Figure 5. Compared to what was achieved in previous part, the learning process for this process was faster and it could achieve the apex in fewer epochs.

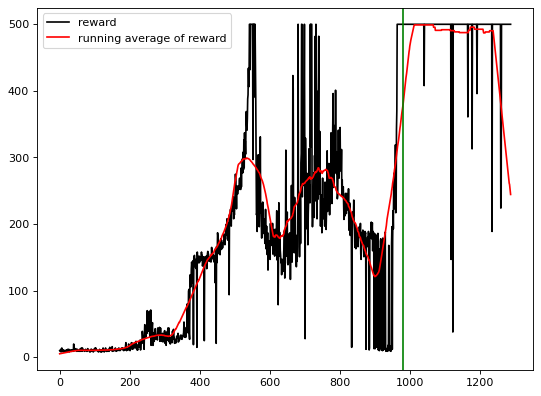


Figure 5

## Reward function

In the previous part, the reward that was used was only the reward given by the environment but here we added another term to the reward to see it compact on the training process. In the cart pole game, the best output is that the cart pole stays in middle and does not go away too much from the middle so proportional to the cart location we added a negative term to the reward function. Learning curve for this method is shown in Figure 6.

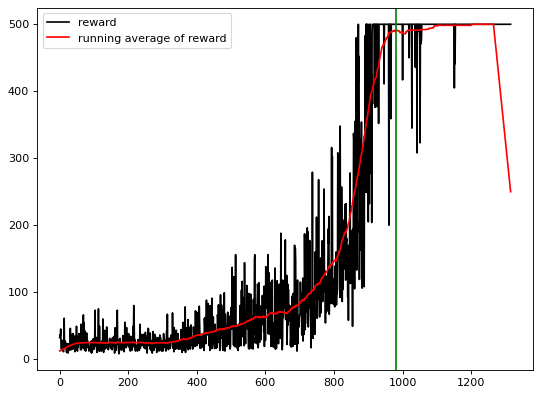


Figure 6

Finally, I trained the network with 3 fully connected layers using the new reward function described in the previous part. Learning curve for this training process is shown in the figure 7. As it can be seen, the training process is faster than all previous ones. Less than ten minutes and in only 970 epochs it reached the maximum score.

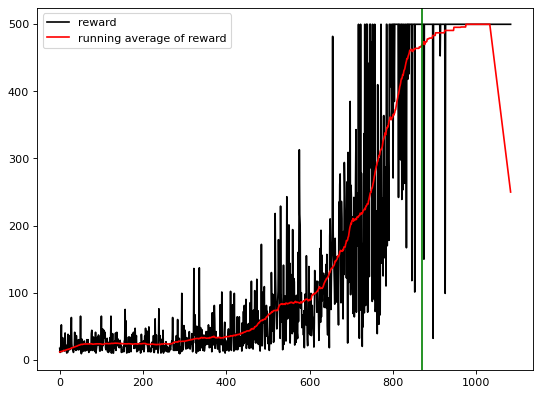


Figure 7

## Screen pixels as observation space

Until here the observation space used for training the network was the four elements shown in figure 1. Here we changed the observation space and instead of receiving the information given by the environment we just used the screen pixels of the environment.