Neural Networks and Deep Learning :

Homework 3: Reinforcement Learning

Prepared by : Norhen Abdennadher

Prof : Dr. Alberto Testolin

Academic year 2021-2022

I. Overview:

This homework is about learning how to implement and test neural network models for solving reinforcement learning problems. The basic tasks for the homework will require to implement some extensions to the code that we have seen in the Lab, and we will continue to work on DQN networks. The basic tasks consist of:

- Trying different hyperparameters or to tweak the reward to speed-up the learning convergence and study the impact of the exploration profile on learning curve.
- Learn to control the CartPole environment using directly the screen pixels, rather than the compact state representation by changing the observation space.
- Train a deep RL agent on a different Gym environment.

II. Hyperparameter tune and exploration profile:

Let's define the work environment used in this section. We will work on the Cartpole environment: Cartpole known also as an Inverted Pendulum is a pendulum with a center of gravity above its pivot point. It's unstable but can be controlled by moving the pivot point under the center of mass. The goal is to keep the cartpole balanced by applying appropriate forces to a pivot point. So, the main task of the agent is to decide between two actions - moving the cart left or right - so that the pole attached to it stays upright.

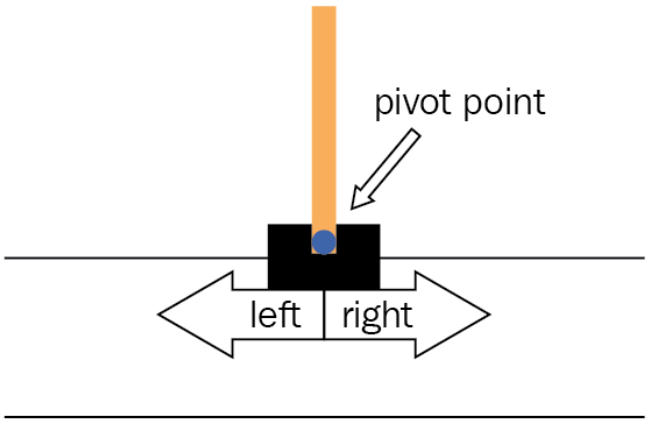

Figure 1: Cartpole illustration

As the agent observes the current state of the environment and chooses an action, the environment transitions to a new state, and returns a reward that indicates the consequences of the action. In this task, rewards are +1 for every incremental timestep and the environment terminates if the pole falls over too far, or the cart moves more than 2.4 units away from center. This means better performing scenarios will run for longer duration, accumulating larger return.

The CartPole task is designed so that the inputs to the agent are 4 real values representing the environment state which are:

- Cart Position
- Cart Velocity
- Pole Angle
- Pole Angular Velocity

The different possible actions to apply are:

- Push cart to the left 0.
- Push cart to the right 1.

Reward:

The default reward is 1 for every step taken, including the termination step. We slightly modified it so that the cart doesn't go out of the screen. We applied a linear penalty when the cart is far from the center of screen.

$$\text{Reward} = \text{reward} - \text{weight} * |\text{observation}[0]|$$

Starting State:

All observations are assigned a uniform random value between [-0.05,0.05].

Here is the diagram that illustrates the overall resulting data flow.

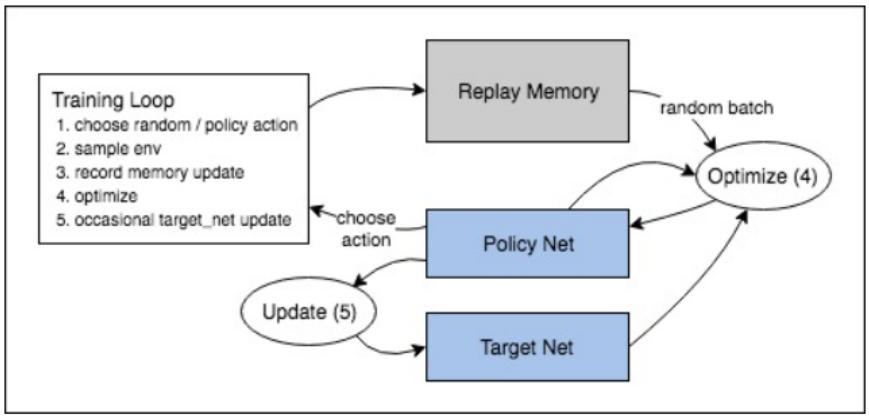

Figure2: System architecture

Actions are chosen either randomly or based on a policy, getting the next step sample from the gym environment. We recorded the results in the replay memory and also run optimization step on every iteration. Optimization picks a random batch from the replay memory to do training of the new policy. "Older" target network is also used in optimization to compute the expected Q values; it is updated occasionally to keep it current.

In this work, in order to choose an action, we will use a Softmax distribution (with temperature t) of the Q-values. The Highest the temperature, the more the distribution will converge to a random uniform distribution. At zero temperature, the policy will always choose the action with the highest Q-value. We will also perform experience replay.

The policy network takes a state as input and provides Q-value for each possible action. The policy network architecture consists of 3 hidden linear layers with Tanh activation function and a linear output layer with action space dimension outputs.

The exploration profile values are as follows:

- Initial value = 8
- Number of iterations= 1000
- Exp_decay = np.exp(-np.log(initial_value) / num_iterations * 8)

And the hyperparameters:

- Gamma = 0.95
- Target net update steps = 8
- Batch size = 256
- Learning rate = 1e-1
-

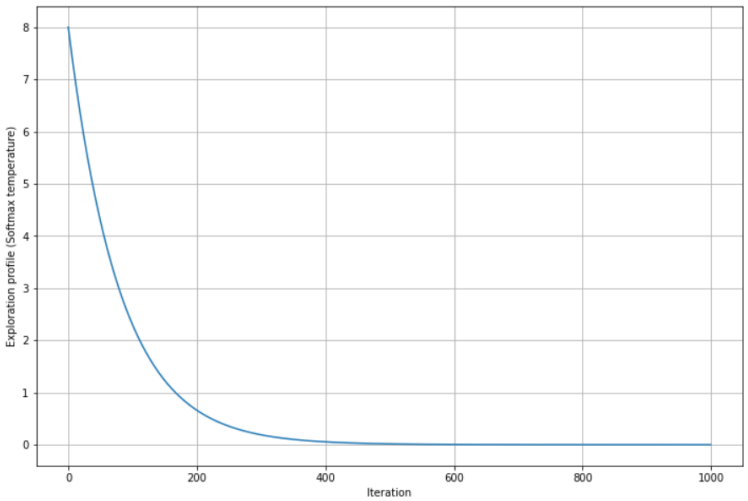

Figure3: Exploration profile

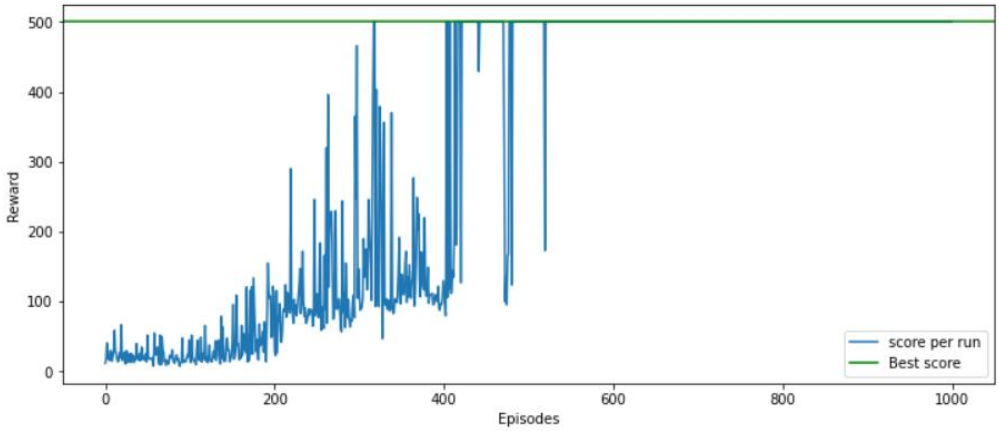

Figure4: Reward as a function of episodes with 500 as best score

III. Control the CartPole environment using the screen pixels:

This part is about to learn how to control the CartPole environment using directly the screen pixels, rather than the compact state representation. For this to happen, we need to define a new convolutional neural network (CNN) to interpret from the display the state of the Cartpole.

It takes in the difference between the current and previous screen patches. It has two outputs, representing Q(s,{left}) and Q(s,{right}) (where s is the input to the network). In effect, the network is trying to predict the expected return of taking each action given the current input.

The network consists of:

- 3 Convolutional layers of kernel 5 followed by batch normalization and a ReLU activation function.
- One output linear layer.

Input extraction:

We defined functions get_cart_location(screen_width) and get_screen() for extracting and processing rendered images from the environment. It uses the torchvision package, which makes it easy to compose image transforms.

Training the network:

To train the network, we reset the environment and initialize the state Tensor at the beginning. Then, we sample an action, execute it, observe the next screen and the reward (always 1), and optimize our model once. When the episode ends (our model fails), we restart the loop.
The results obtained with this second approach are not really reliable as the first approach, the model isn't stable.

IV. Train a deep RL agent on a mountain car GYM environment:

This part consists of exploring a new environment provided by gym called Mountain car.

A Mountain car system is a one-dimensional track, positioned between two "mountains". The goal is to drive up the mountain on the right; however, the car's engine is not strong enough to scale the mountain in a single pass. Therefore, the only way to succeed is to drive back and forth to build up momentum.

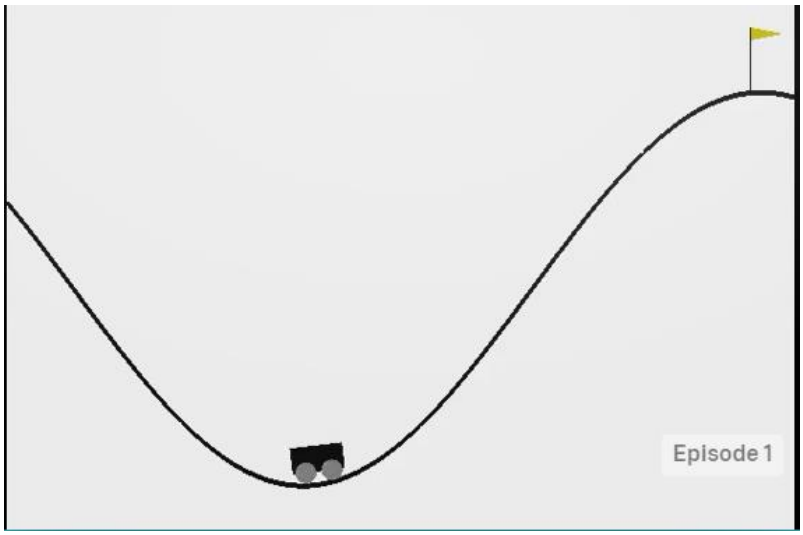

Figure 5: Mountain car system

The car's state, at any point in time, is given by a vector containing its horizonal position and velocity. The car commences each episode stationary, at the bottom of the valley between the hills (at position approximately -0.5), and the episode ends when either the car reaches the flag (position > 0.5) or after 200 moves.

We can see that the first element of the state vector (representing the cart's position) can take on any value in the range -1.2 to 0.6, while the second element (representing the cart's velocity) can take on any value in the range -0.07 to 0.07.

The following are the environment variables:

Observation:

0- Car position between -1.2 and 0.6

1- Car velocity between -0.07 and 0.07

Actions:

0- Accelerate to the left.

1- Don't accelerate.

2- Accelerate to the right.

We kept the same implemented architecture of the policy network in the first gym environment(CartPole) and we can find below the hyperparameters for the network:

- gamma = 0.97
- replay_memory_capacity = 10000
- lr = 1e-2
- target_net_update_steps = 10
- batch_size = 128
- min_samples_for_training = 1000

System reward:

The default reward is 0 if the agent reached the flag (position = 0.5) and -1 if the position of the agent is less than 0.5. We slightly modified it so that the car can reach the top of the mountain by increasing the reward if the car position greater than 0.
If observation [0] > 0:
Reward = reward + weight * (observation [0] + 1)
Else:
reward= reward - pos_weight * (np.abs(state[0])+2)

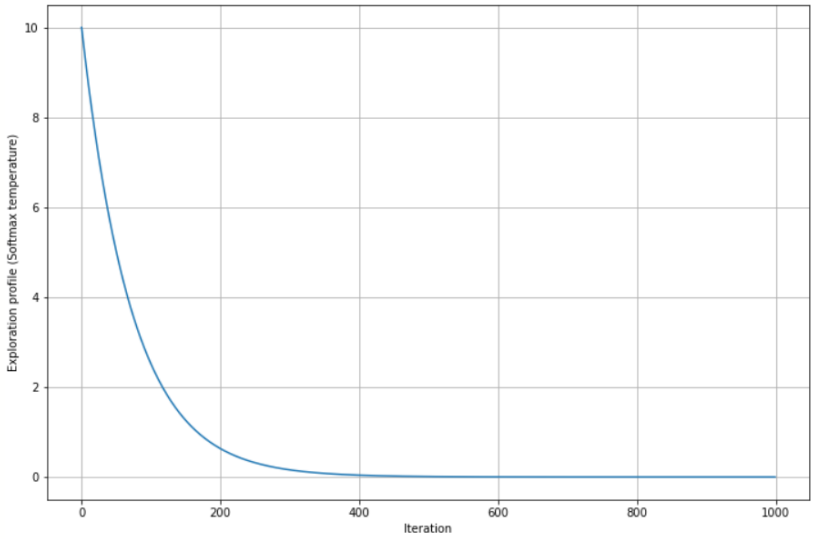

Figure6: Exploration profile

```
Updating target network...
EPISODE: 981 - FINAL SCORE: 0.4929231128368925 - Temperature: 1.3182567385564235e-05
EPISODE: 982 - FINAL SCORE: 0.4896693712607187 - Temperature: 1.3001695780333064e-05
EPISODE: 983 - FINAL SCORE: -0.3864437713961768 - Temperature: 1.2823305826560376e-05
EPISODE: 984 - FINAL SCORE: 0.4994072093735944 - Temperature: 1.264736347471167e-05
EPISODE: 985 - FINAL SCORE: 0.4911793592590972 - Temperature: 1.2473835142429584e-05
EPISODE: 986 - FINAL SCORE: -0.7398727669497315 - Temperature: 1.2302687708123968e-05
EPISODE: 987 - FINAL SCORE: 0.493013120676151 66 - Temperature: 1.2133888504649924e-05
EPISODE: 988 - FINAL SCORE: 0.4908801553940776 - Temperature: 1.1967405313072585e-05
EPISODE: 989 - FINAL SCORE: 0.4999346149016904 - Temperature: 1.1803206356517444e-05
EPISODE: 990 - FINAL SCORE: 0.490743403325888 95 - Temperature: 1.164126029410506e-05
Updating target network...
EPISODE: 991 - FINAL SCORE: 0.497344792873363 64 - Temperature: 1.1481536214968971e-05
EPISODE: 992 - FINAL SCORE: 0.4969021102226429 7 - Temperature: 1.1324003632355711e-05
EPISODE: 993 - FINAL SCORE: 0.4936371610407132 6 - Temperature: 1.1168632477805752e-05
EPISODE: 994 - FINAL SCORE: 0.4942119213662103 4 - Temperature: 1.101539309541429e-05
EPISODE: 995 - FINAL SCORE: 0.4997253812737824 - Temperature: 1.0864256236170792e-05
EPISODE: 996 - FINAL SCORE: -0.485126971078626 - Temperature: 1.0715193052376199e-05
EPISODE: 997 - FINAL SCORE: 0.4913234621140228 - Temperature: 1.0568175092136717e-05
EPISODE: 998 - FINAL SCORE: -0.9038274620114425 - Temperature: 1.0423174293933173e-05
EPISODE: 999 - FINAL SCORE: 0.4958153786873214 5 - Temperature: 1.0280162981264865e-05
EPISODE: 1000 - FINAL SCORE: 0.26006311580767955 - Temperature: 1.0139113857366923e-05
```

Figure7: Final episodes scores results

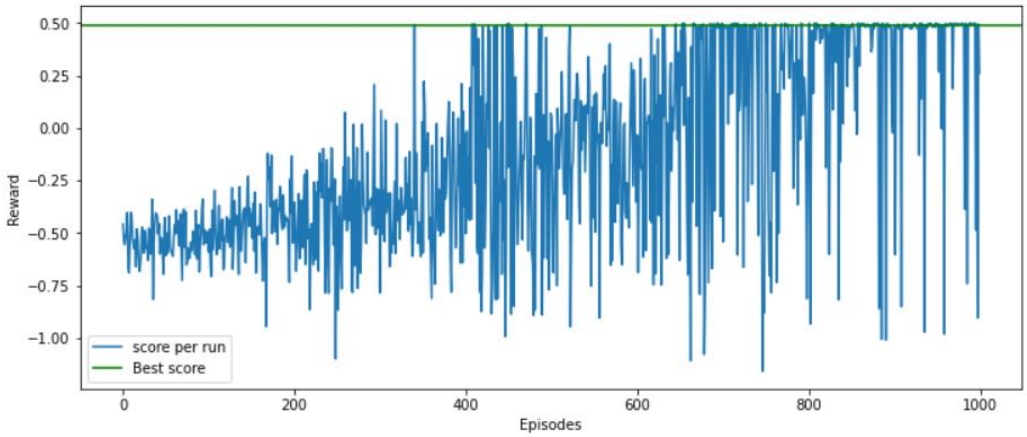

Figure8: Reward visualization

As we can see the car took around 600 episodes to be more stable. And when we tested our model we noticed that the car was able to reach the top of the mountain in most of the cases.