ID2223 - Final project

Filip Finfando, Javier de la Rúa Martínez

10th January 2020

1 Problem Description

CartPole is a simple classical control problem commonly used in reinforcement learning experiments. Basically, it is a 2D game in which a pole is attached to a cart which moves along a friction-less track. The pole is standing perpendicularly over the cart, whose goal is to balance the pole and prevent it from falling by moving left or right. By these means, the agent controls the cart by applying a force of 1 unit to the left or right. Every timestep that the pole remains upright, the agent receives a reward of +1. Each time the agent plays the game until it wins or losses is called episode. Each episode ends when one of different conditions are met. The following figure represents an observation of the environment.

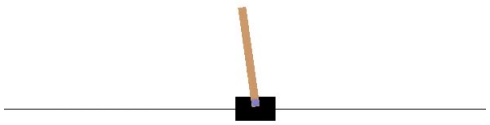

Figure 1: CartPole 2D game representation

1.1 Environment

The CartPole problem is represented by two environments in Gym [1] (i.e versions 0 and 1) of OpenAI, differing in the conditions to end an episode. Both environments ends the episode when the angle of the pole exceeds 15 degrees from the vertical or the cart go out of the board. By contrast, while the version 0 also ends an episode after a maximum of 200 steps or a maximum reward of 195, the version 1 extends those numbers to 500 of steps and a reward of 475.
The environment provides a discrete action space, from which the agent have to choose the next action to perform, and the observation space from which the environment returns an observation after a selected action is applied. Together with the next observation, the environment returns a reward associated with the action and informs about the possible end of the episode. With these information, the agent have to choose the next action from the action space and so on.
By default, the observations returned by the environment are vectors of 4 values: cart position, cart velocity, pole angle and pole velocity at tip. Instead, we try to solve this problem using convolutional neural networks which receives extracts of the images of the environment to learn representations and predict the best action to take.
In order to interact with the environment easily, we implemented a wrapper called *CartPoleEnv* with utilities such as extracting these smaller representations of the images or deciding when to keep track of the frames to generate videos of the episodes.

2 Agent and CNN

We implemented a class called *CNNAgent* that acts as the agent that interacts with the environment. This class contains several methods for deciding the new action, memorizing the transitions of the environment and learning patterns to optimize the reward of these transitions. In order to choose the next action to take, the agent uses a convolutional neural network to interpret the representation of the state of the environment (i.e screen). The architecture of the CNN is shown in the Figure 2. Relu activation functions is used for the convolutional layers.

```
[CNN] H: 40 - W: 60 - O: 2
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 16, 18, 28]           1,216
       BatchNorm2d-2          [-1, 16, 18, 28]              32
            Conv2d-3           [-1, 32, 7, 12]          12,832
       BatchNorm2d-4           [-1, 32, 7, 12]              64
            Conv2d-5            [-1, 32, 2, 4]          25,632
       BatchNorm2d-6            [-1, 32, 2, 4]              64
            Linear-7                  [-1, 2]             514
================================================================
Total params: 40,354
Trainable params: 40,354
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.03
Forward/backward pass size (MB): 0.17
Params size (MB): 0.15
Estimated Total Size (MB): 0.35
----------------------------------------------------------------
```

Figure 2: CNN architecture

For the training of the CNN, the agent keep track of all the transitions which are composed of: previous observation, action taken, reward obtained and resulted observation. After each move during an episode, the model is trained according to the following equation.

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$

3 Experiments

After dealing with several issues mentioned in the corresponding section, we implemented the solution in a Jupyter notebook and run it in Google Colab. We tried either different general parameters for the experiment and different hyperparameters for the agent and CNN. The ones that provided better results are the following.

- Episodes: 100
- Memory capacity: 20000 transitions
- Batch size: 128
- Epsilon: 0.9
- Minimum epsilon: 0.01
- Epsilon decay: 225
- Gamma: 0.999
- Learning rate: 0.0005
- Kernel size: 5
- Stride: 2

4 Results

After playing 100 episodes, the agent managed to win 2 games, keeping improving with each episode but noticeably unstable as it gets better in the game. The following figures shows the rewards obtained by the agent and the timesteps associated with each episode.

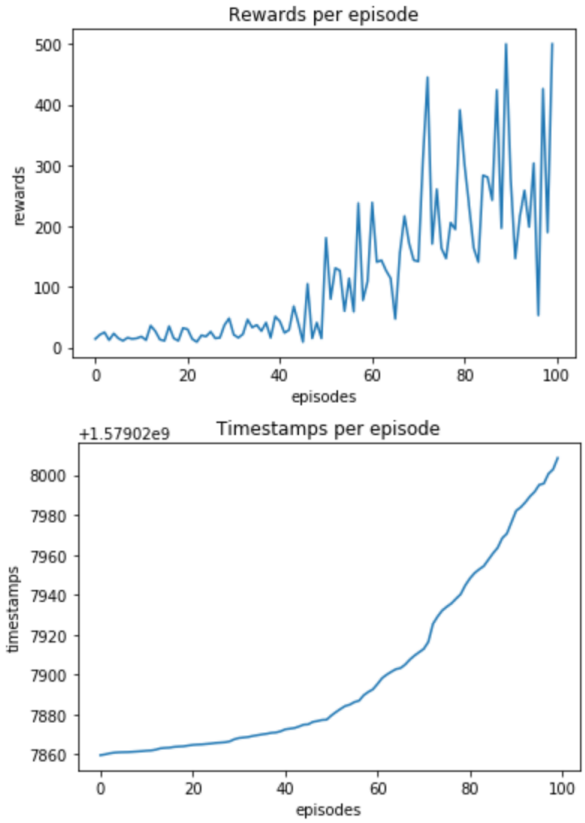

Figure 3: Rewards and timesteps per episode obtained by the agent during the experiment

The jupyter notebook with the implementation and a video of a winning game can be found in the folder of the project.

5 Issues

In our experience it was not straightforward to set up the environment for tackling a reinforcement learning problem with DQN. We have spent vast amount of time setting up VM instances on Google Cloud Platform trying to find a fit between NVIDIA CUDA version, CUDNN version and tensorflow version or pytorch version. Since rendering is required for the environment to provide the observations (i.e screens) and remotely accessed servers normally lack of screen, one of the main issues we were facing is that NVIDIA drivers had to be installed without OpenGL libraries in order to make the rendering of the game's screen work. We tried different approaches such as attaching virtual screens or trying to mislead the program to believe there is a real screen available. At last we have found a working solution in a Google Colab environment.

6 Future work

In the future we would like to understand better how to tackle the challenge of CartPole environment e.g. trying different DNN architectures or restarting the epsilon every n-th episode to push agent more towards exploration. Next, we will consider to try different environments.

References

[1] OpenAI. *Gym*. URL: https://gym.openai.com/.