Project 2: Continuous Control

Learning Algorithm

I decided to try the Deterministic Policy Gradient (DDPG) method.

You have the ddpg_agent.py file that contains the learn and acting part. And a model.py file that contains the 2 actor and critic Deep Neural Networks, use to evaluate the Q function and the Value function.

I have also a classic replay buffer that is coded inside the ddpg_agent.py file.

Hyperparameters for the DDPG:

All hyperparameters are manage from this cell, and are passed an argument dictionary to the relevant function.

```
from ddpg_agent import Agent

demo = False

hyperparameters = {
        'ddpg' :    {   'n_episodes':                   120,
                        'max_t':                        1000,
                        'num_agents':                   20,
                        'add_noise':                    True
                    },
        'agent' :   {   'state_size':                   33,
                        'action_size':                  4,
                        'random_seed':                  2710,
                        'GPU':                          True,
                        'LEARN_EVERY':                  100,            # How many steps between learning
                        'LEARN_REPEAT':                 100,            # Number of time we sample the buffer
                        'BUFFER_SIZE':                  int(1e6),       # replay buffer size
                        'BATCH_SIZE':                   256,            # minibatch size
                        'GAMMA':                        0.99,           # discount factor
                        'TAU':                          1e-3,           # for soft update of target parameters
                        'Actor_network':                [256, 128],     # Hidden layer in the FC Actor network
                        'LR_ACTOR':                     1e-4,           # learning rate of the actor
                        'Critic_network':               [256, 128],     # Hidden layer in the FC Critic network
                        'LR_CRITIC':                    1e-4,           # learning rate of the critic
                        'WEIGHT_DECAY':                 0               # L2 weight decay
                    },
        }

agent = Agent(**hyperparameters['agent'])
```

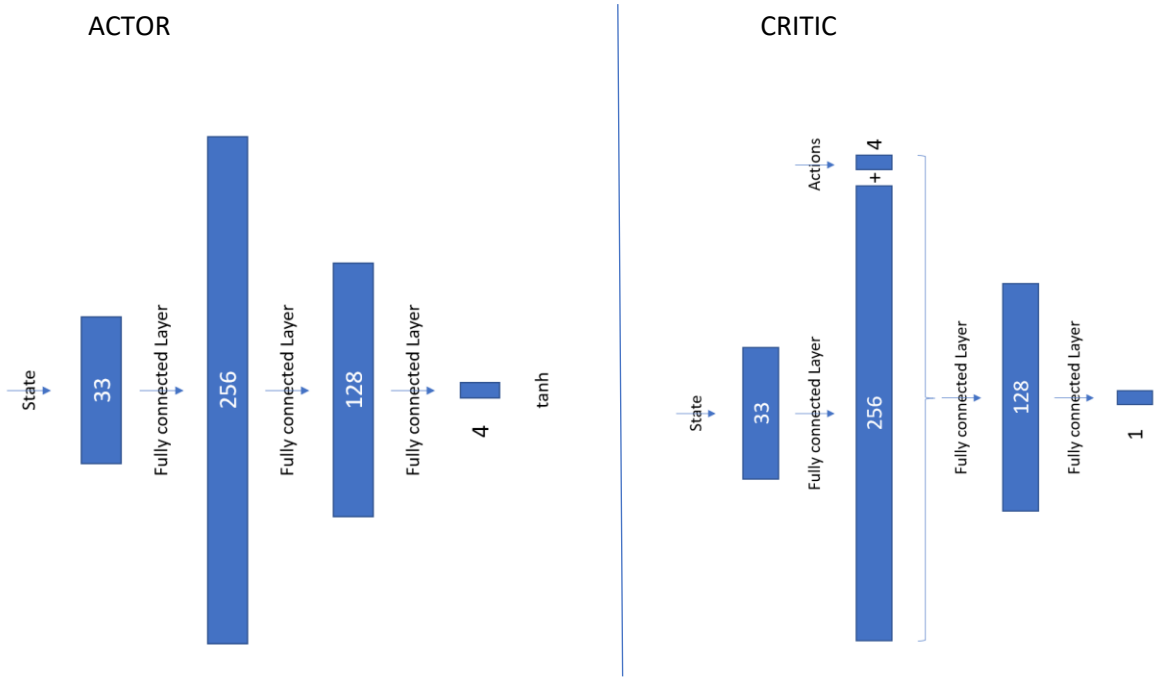
After some long trial and error, I have settled on those parameters:

- GPU active for performance.
- Learn every 100 steps
- Sample the buffer 100 times for learning
- Replay buffer at 1 million
- Batch size for learning at 256
- Gamma at 0.99, very standard, just not to loop.
- Learning Rate 0.0001 for both actor and critic
- Tau at 0.001 as the rate to move from current to target network.

I have trained a single pair of neural networks with 12 agents to maximise performance. The Noise factor and the initial setting of each arm made the learning different for each agent.

For the Neural Network

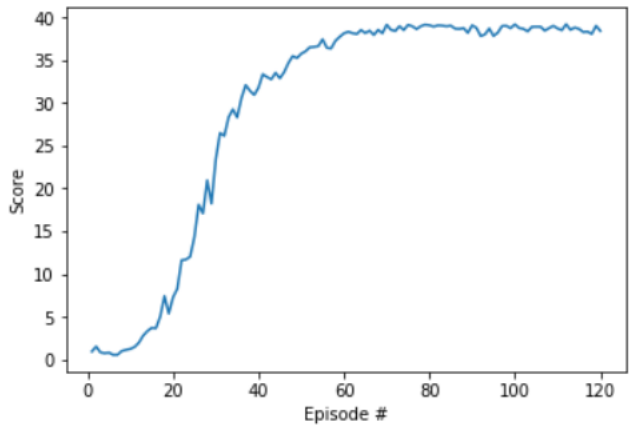
I have used a simple structure for this problem,



Plot of Rewards

When using the “Continuous Control.ipynb” you have the following result in a plots. That shows the score over the last 100 episodes per episode of training.

```
Training Mode
Episode   10 in 27.99s. Score:  1.26    Average Score:  0.95    LrA: 1.0e-04    LrC: 1.0e-04
Episode   20 in 31.89s. Score:  7.29    Average Score:  2.58    LrA: 1.0e-04    LrC: 1.0e-04
Episode   30 in 36.47s. Score: 23.40    Average Score:  6.91    LrA: 1.0e-04    LrC: 1.0e-04
Episode   40 in 41.41s. Score: 31.79    Average Score: 12.57    LrA: 1.0e-04    LrC: 1.0e-04
Episode   50 in 46.05s. Score: 35.75    Average Score: 16.86    LrA: 1.0e-04    LrC: 1.0e-04
Episode   60 in 47.16s. Score: 38.13    Average Score: 20.20    LrA: 1.0e-04    LrC: 1.0e-04
Episode   70 in 46.89s. Score: 39.15    Average Score: 22.80    LrA: 1.0e-04    LrC: 1.0e-04
Episode   80 in 46.97s. Score: 39.08    Average Score: 24.80    LrA: 1.0e-04    LrC: 1.0e-04
Episode   90 in 46.95s. Score: 39.08    Average Score: 26.36    LrA: 1.0e-04    LrC: 1.0e-04
Episode  100 in 46.92s. Score: 39.18    Average Score: 27.58    LrA: 1.0e-04    LrC: 1.0e-04
Problem solved in 107 episodes, score = 30.226804324378254LrA: 1.0e-04  LrC: 1.0e-04
Episode  110 in 47.02s. Score: 38.71    Average Score: 31.36    LrA: 1.0e-04    LrC: 1.0e-04
Episode  120 in 47.12s. Score: 38.42    Average Score: 34.79    LrA: 1.0e-04    LrC: 1.0e-04
```



The average score over 100 episodes has reached 30 on episodes 107. That is when the average reached 30, which means that the network was reaching 30 points much earlier than that. That is extra fast, but you have to keep in mind that I am getting data from 12 agents at each episodes.

That is still super-fast, compare to the crawler Project.

Ideas for Future Work

On the continuous control problem:

- Try other technic like PPO A3C and D4PG
- Try the prioritized Replay buffer to see if it works better than the Navigation project.
- Try to make the network more generalizing, I see in some configuration tha the arm has trouble following the target.
- Change the rewards to give more point in the middle of the target so that there is less risk of going out of the target.
- Experiment more on the Noise function and its benefits.

I have spent some time on the code to make it nice, it is more now the fine tuning of parameters the next and perpetual challenge.

Crawler Project

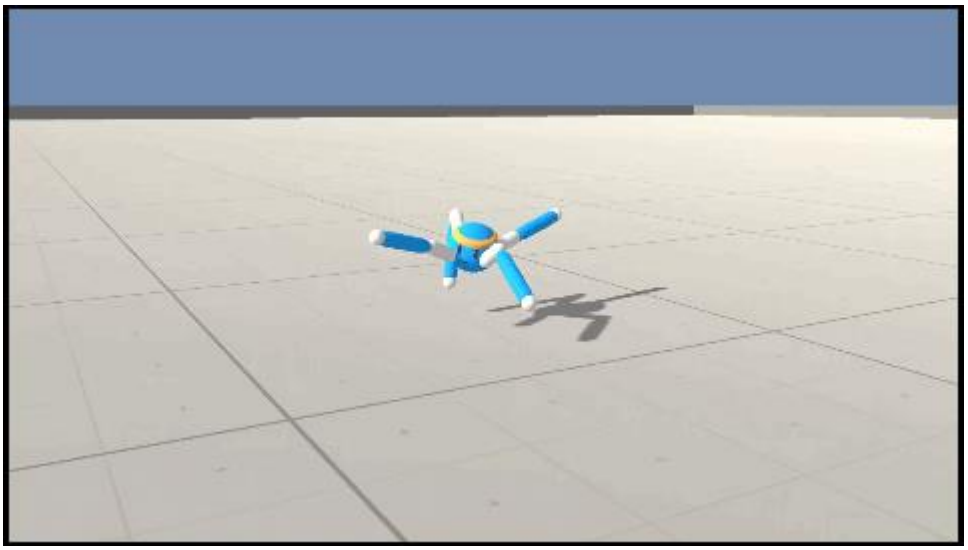
As this is optional, I am not going to follow the dictated structure.

This one tool me a lot of effort to get convergence, plenty of time my score would crash or stagnate after some time. I have tried complex network due to the state dimensions, but in fact a more compact fully connected network gave me the best results.

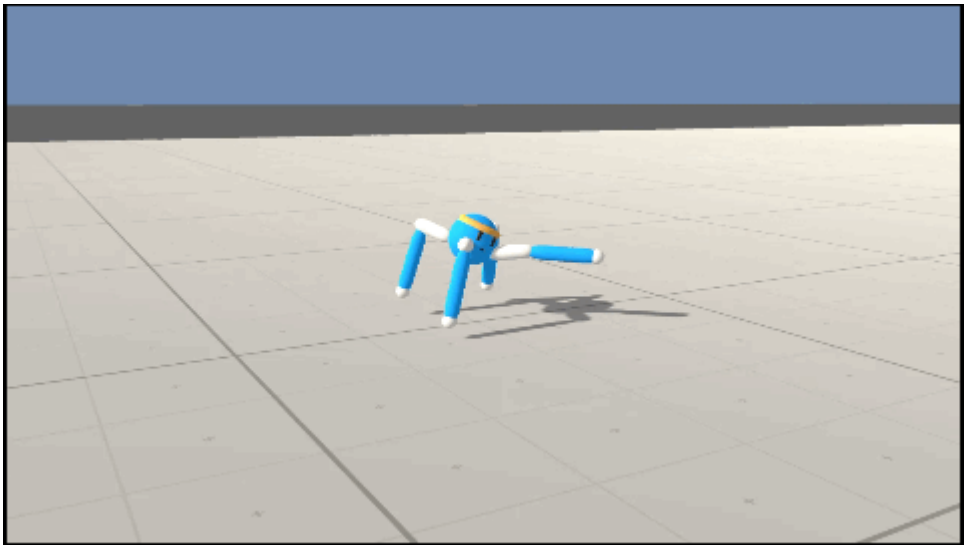
The network was composed of fully connected layer 512, 256 and 128 weights

My highest score over 100 episodes is **1138 points**, I was quite proud.

This is a video of the result



But I have also had strange solution proposed by the network, like a Ninja style way of walking.



The real complexity is also on the training time. More than 6000 episodes was needed to obtain this.