Project 3: Collaboration and Competition

Learning Algorithm

I decided to try the Multi Agent Deep Deterministic Policy Gradient (MADDPG) method.

You have the maddpg_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, and a new reward buffer that is coded inside the tools.py file.

Hyperparameters for the DDPG:

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

```
hyperparameters = {
          'algo' :  {   'algo':                          'ma_ddpg',
                        'n_episodes':                    10000,
                        'max_t':                         2000,
                        'num_agent':                     2,
                        'add_noise':                     True
                    },
  'agent_maddpg' :  {   'state_size':                    24,
                        'action_size':                   2,
                        'nb_agent':                      2,
                        'random_seed':                   2710,
                        'GPU':                           True,
                        'LEARN_EVERY':                   100,          # How many steps between learning
                        'LEARN_REPEAT':                  20,           # Number of time we sample the buffer
                        'BUFFER_SIZE':                   int(1e6),     # replay buffer size
                        'BATCH_SIZE':                    2048,         # minibatch size
                        'GAMMA':                         0.99,         # discount factor
                        'TAU':                           1e-3,         # for soft update of target parameters
                        'Actor_network':                 [400, 300],   # Hidden layer in the fully connected Actor
                        'LR_ACTOR':                      1e-3,         # learning rate of the actor
                        'Critic_network':                [400, 300],   # Hidden layer in the fully connected Critic
                        'LR_CRITIC':                     1e-3,         # learning rate of the critic
                        'WEIGHT_DECAY':                  0,            # L2 weight decay
                        'clip_gradient':                 False
                    },
```

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

- GPU active for performance.
- Learn every 100 steps
- Sample the buffer 20 times for learning
- Replay buffer at 1 million
- Batch size for learning at 2048
- Gamma at 0.99, very standard, just not to loop.
- Learning Rate 0.001 for both actor and critic, with a scheduler to ratio it by 10
- Tau at 0.001 as the rate to move from current to target network.

I have trained a pair of agents to maximise performance by collaborating to a common objective, making to game last to score more.

This was by far the project I had more issue to make converge. I have tried many hyperparameters settings and network size.

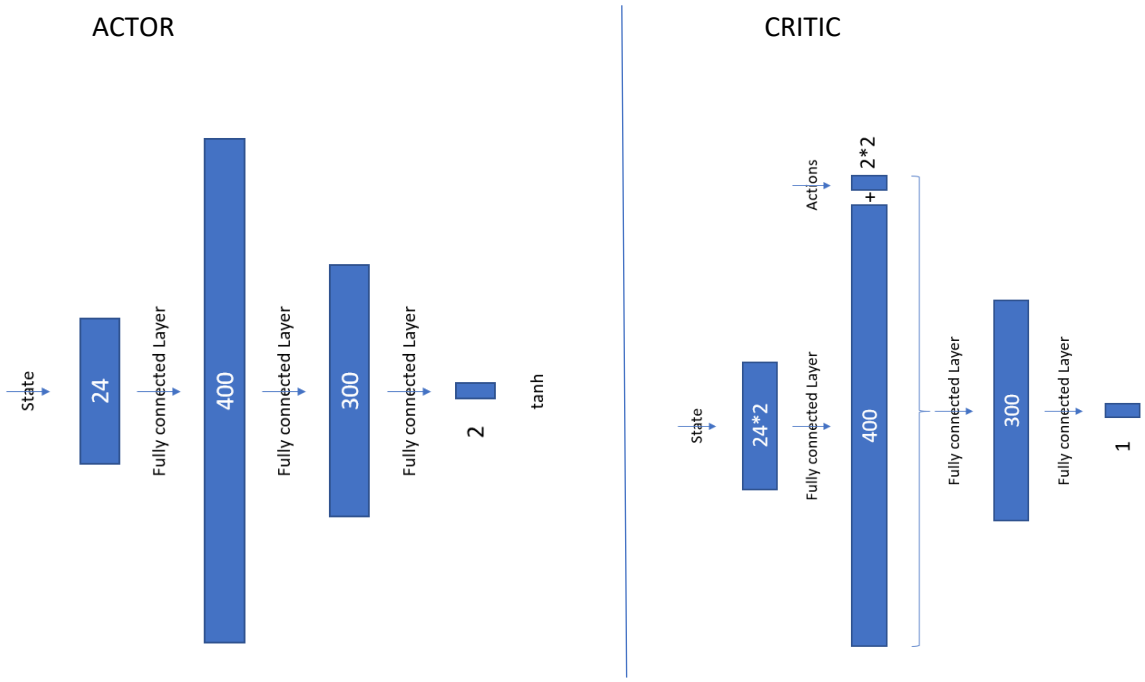
To make it work, I had to add a few elements:

- Adding a variable Sigma parameter (the ratio of effect of the normal distribution of noise) in the OUNoise exploration parameters, starting at 0.6 to reduce to 0.2 and 0.1

- As the tennis shot and the reward (crossing the net) are not at the same time, I had the idea of adding a reward buffer to propagate 10 steps in the past the rewards obtained at a given step. Using the Gamma value to algorithm go toward the rewards. The concept is that the new states are added to the buffer, if there is a reward it is added to the experience in the buffer and the last experience is added to the real main replay buffer. That helps the algorithm propagate the reward in the Q value network more efficiently.

For the Neural Network

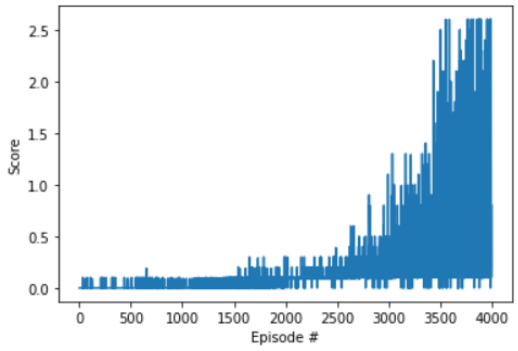
I have used a simple structure for this problem,



Plot of Rewards

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

```
Training Mode
Episode  200 in  0.05s. Score:  0.000   Average Score:  0.002   LrA: 1.0e-03    Sigma: 6.0e-01
Episode  400 in  0.05s. Score:  0.000   Average Score:  0.005   LrA: 1.0e-03    Sigma: 6.0e-01
Episode  600 in  0.11s. Score:  0.100   Average Score:  0.013   LrA: 1.0e-03    Sigma: 6.0e-01
Episode  800 in  0.05s. Score:  0.000   Average Score:  0.010   LrA: 1.0e-03    Sigma: 6.0e-01
Episode 1000 in  0.05s. Score:  0.000   Average Score:  0.012   LrA: 1.0e-03    Sigma: 2.0e-01
Episode 1200 in  0.10s. Score:  0.090   Average Score:  0.019   LrA: 1.0e-03    Sigma: 2.0e-01
Episode 1400 in  9.97s. Score:  0.090   Average Score:  0.023   LrA: 1.0e-03    Sigma: 2.0e-01
Episode 1600 in  0.05s. Score:  0.000   Average Score:  0.050   LrA: 1.0e-03    Sigma: 2.0e-01
Episode 1800 in  0.15s. Score:  0.100   Average Score:  0.071   LrA: 1.0e-03    Sigma: 2.0e-01
Episode 2000 in  9.84s. Score:  0.100   Average Score:  0.093   LrA: 1.0e-03    Sigma: 1.0e-01
Episode 2200 in  0.10s. Score:  0.090   Average Score:  0.095   LrA: 1.0e-03    Sigma: 1.0e-01
Episode 2400 in  0.10s. Score:  0.090   Average Score:  0.104   LrA: 1.0e-03    Sigma: 1.0e-01
Episode 2600 in  0.10s. Score:  0.090   Average Score:  0.128   LrA: 1.0e-03    Sigma: 1.0e-01
Episode 2800 in 10.30s. Score:  0.290   Average Score:  0.181   LrA: 1.0e-04    Sigma: 1.0e-01
Episode 3000 in 10.03s. Score:  0.300   Average Score:  0.213   LrA: 1.0e-04    Sigma: 1.0e-01
Episode 3200 in 30.22s. Score:  0.600   Average Score:  0.268   LrA: 1.0e-04    Sigma: 1.0e-01
Episode 3400 in  0.16s. Score:  0.100   Average Score:  0.408   LrA: 1.0e-04    Sigma: 1.0e-01
Problem solved in 3588 episodes, score = 0.5030000075325369
Episode 3600 in 19.73s. Score:  0.300   Average Score:  0.513   LrA: 1.0e-05    Sigma: 1.0e-01
Episode 3800 in 102.45s.        Score:  2.600   Average Score:  0.795   LrA: 1.0e-05    Sigma: 1.0e-01
Episode 4000 in 31.22s. Score:  0.800   Average Score:  0.717   LrA: 1.0e-05    Sigma: 1.0e-01
```



The average score over 100 episodes has reached 0.503 on episodes 3588. We should expect a very high raise of the score after that, has the task is not cumulatively complex, it is just a matter of being constant in sending the ball back, so at one stage the limit is going to be the maximum step allowed by episode.

Ideas for Future Work

On the continuous control problem:

- Try comparing two DDPG agents and MADDPG.
- Try the prioritized Replay buffer to see if it works better than the Navigation project.
- Try to make the network more generalizing, the agent is not understanding that the ball is unreachable on the other side.
- Experiment a system where the same agent could play both side by training on a canonical version of the state, so the agent would have a collaborate with his own self.
- 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.