Tennis project report

1- Model

For this project, we used the Deep Deterministic Policy Gradient (**DDPG**) model and modified it to become applicable for multi agent problems. In fact, we gave the model the states of the two agents and we expected from it the actions of both agents. Both agents are rewarded with the same reward. We will explain this reward in the Learning part.

a) Architecture

The model architecture is based on actor and critic methodology.

- **Actor** : the actor contains 3 fully connected linear layers with the following dimensions
    - (48, 512) with 48 = 2 * state size
    - (512, 256)
    - (256, 4) with 4 = 2 * action size
- **Critic** : the critic also contains 3 fully connected linear layers with the following dimensions
    - (48, 512) with 48 = 2 * state size
    - (516, 256) with 516 = 512 + 2* action size
    - (256, 1)

b) Learning

We made the **DDPG** model learn once at the end of each episode. In fact, we add the episode to the memory and learn 200 times from sampled data from the memory. We also changed the rewards as the rewards from the environment are quite vague. For instance, if the agent failed to return the ball it means it failed from the start of the episode (or from the step that the opposing agent returned the ball) to the end of the episode. We can also apply the same logic for a good bounce too. This is why we used the current rewards

- **Fail reward:** 0, -0.01, .., -0.01*num_steps (with num_steps is the number of steps from beginning the failure)
- **Success reward:** number_of_bounces + 0.1*num_steps, .., number_of_bounces + 0.1, number_of_bounces (with number_of_bounces the number of rebound between the agents and num_steps the number of steps from the beginning of the episode to the last rebound)

This reward has in it an emphasis on encouraging the agents to reproduce the long sequences of rebounds. For sure, both agents are rewarded with the same rewards as we only have one ddpg agent for them. The agents are not punished heavily on their failure because the architecture we chose punish the two agents for dropping the ball while it can be one agent's fault. This is why punishing heavily could have drawbacks on the learning especially in the start. For the same reason, we added a made our memory a selective one where all the successful steps are saved but only 50% of the failed rewards are saved. This accelerates the learning for the first steps too as the agent is more likely to fail in the first steps.

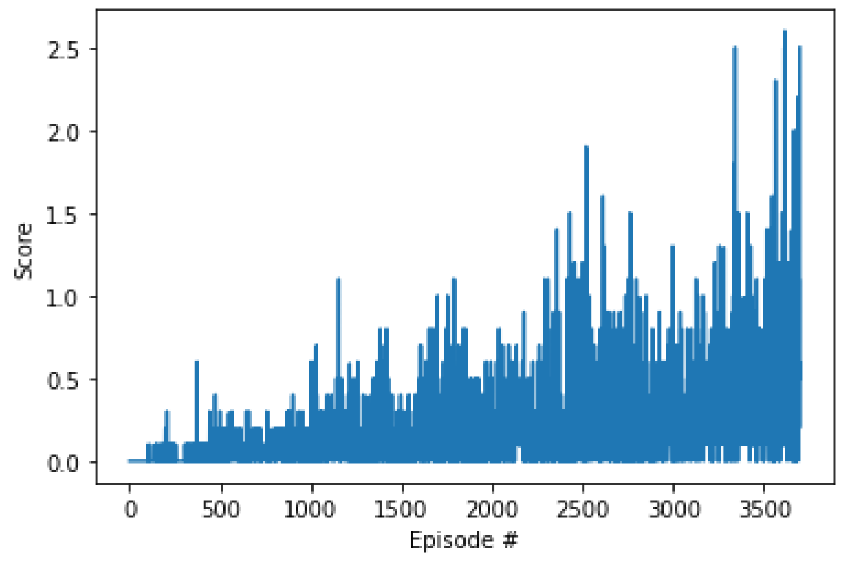
c) Hyperparameters

```
BUFFER_SIZE = int(1e6)  # replay buffer size

BATCH_SIZE = 512        # minibatch size

GAMMA = 0.99            # discount factor

TAU = 1e-3              # for soft update of target parameters

LR_ACTOR = 1e-4         # learning rate of the actor

LR_CRITIC = 3e-4        # learning rate of the critic

WEIGHT_DECAY = 0        # L2 weight decay
```

2- Training

```
Episode 100    Average Score: 0.00
Episode 200    Average Score: 0.02
Episode 300    Average Score: 0.02
Episode 400    Average Score: 0.06
Episode 500    Average Score: 0.08
Episode 600    Average Score: 0.09
Episode 700    Average Score: 0.09
Episode 800    Average Score: 0.09
Episode 900    Average Score: 0.09
Episode 1000   Average Score: 0.09
Episode 1100   Average Score: 0.13
Episode 1200   Average Score: 0.16
Episode 1300   Average Score: 0.14
Episode 1400   Average Score: 0.14
Episode 1500   Average Score: 0.13
Episode 1600   Average Score: 0.11
Episode 1700   Average Score: 0.21
Episode 1800   Average Score: 0.18
Episode 1900   Average Score: 0.16
Episode 2000   Average Score: 0.17
Episode 2100   Average Score: 0.17
Episode 2200   Average Score: 0.18
Episode 2300   Average Score: 0.19
Episode 2400   Average Score: 0.22
Episode 2500   Average Score: 0.27
Episode 2600   Average Score: 0.26
Episode 2700   Average Score: 0.27
Episode 2800   Average Score: 0.26
Episode 2900   Average Score: 0.23
Episode 3000   Average Score: 0.23
Episode 3100   Average Score: 0.25
Episode 3200   Average Score: 0.27
Episode 3300   Average Score: 0.31
Episode 3400   Average Score: 0.40
Episode 3500   Average Score: 0.32
Episode 3600   Average Score: 0.36
Episode 3700   Average Score: 0.51
```

Environment solved in 3700 episodes!
Average Score: 0.51



3- Next steps

To improve the results we have we can try the following steps:

- Run the model for a longer duration. The model didn’t show a sign of declining yet so training it more should improve its performance. It could take days though!
- Try to apply the same ddpg model for both actors separately instead of having the model predict them at the same time. This could have a better result on learning how to shoot the ball but won’t improve the cooperation between them.
- Try out other actor critique models and compare the results.
- Implement Prioritized Experience Replay using special data structure Sum Tree.