Report for Project 3 – Collab-compete

The project, like project 2, was solved using a DDPG agent in Pytorch. Two similar agents play tennis with each other. Both agents have two possible actions – move, jump. The goal is to keep the ball in play as long as possible – Hence, the collaborative nature of this project.

The architecture is very similar to project 2. I only changed the hyper parameters as the agent was struggling to learn anything without having a higher learning rate.

Observations

- Initially I followed the instructions in the exercise as closely as possible. I implemented a two- agent with DDPG, double DQN first. I fixed a couple of bugs in model.py and then changed the code to work for current project.
- The major change was in the scoring function as described in the project details.
- Training the agent resulted in no improvements. The agent would return 0 reward consistently and would somehow latch to either going left or right and jumping up.
- I decided to change some hyper parameters as I thought that the agent is very slow to learn.
- I changed the learning rate to 1e-4 for both actor and critic agents. As there are only two agents, I also increased the buffer size significantly.
- The other major hyper parameter change was increasing the value of TAU significantly from 0.001 to 0.1. This helped in learning from local agents significantly faster as initially the agents were not learning at all.
- The initial episodes usually result in no improvements in score. But once the agent gets to about 0.1 average score, the agent learns faster.
- Like previous project, the initial episodes are critical. If the initial few episodes results in a 0 core, chances are the agent will never improve.

Learning algorithm:

ddpg_agent.py describes the agent.

I used 2 actor (local) networks and 2 critic (target) networks – one each for local and target. I also used a replay buffer and noise model – OUNoise for the agent. I used Adam optimizer. A soft update is performed using tau as the control variable. The number is relatively bigger than previous project, so it updates the target by a bigger number each iteration.

Hyperparameters:

```
BUFFER_SIZE = 2**20   # replay buffer size
BATCH_SIZE = 128      # 128 minibatch size
GAMMA = 0.99          # discount factor
TAU = 1e-1            # for soft update of target parameters
LR_ACTOR = 1e-4       # learning rate of the actor
LR_CRITIC = 1e-4      #  learning rate of the critic
WEIGHT_DECAY = 0      # L2 weight decay
UPDATE_EVERY = 1
```

Model architecture:
The model is contained in model.py. It is exactly same as the model used in project 2.

I used 2 hidden layers network for Actor and 1 hidden layer network for Critc.
For Actor network, I used ReLU for activations and 2 sets of batch normalizations before 1st and 2nd linear transformations and ReLU activations.

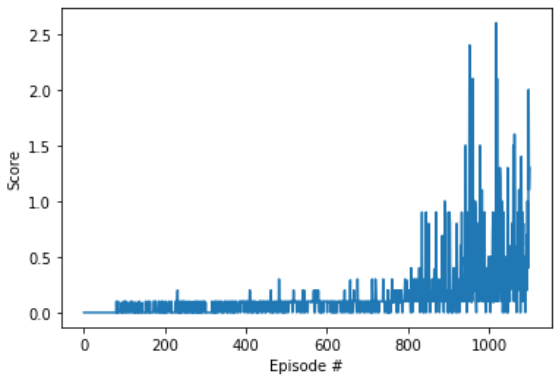
For Critic network, only one batch normalization after the first hidden layer followed by linear transformation and ReLU activation worked well for me.

Tennis.ipnyb contains the python notebook to load unity environment and calls to train the agent.
The walkthrough of the code is in readme file.

Results

```
Episode 100    Average Score: 0.0110  , Average Score: 0.01
Episode 200    Average Score: 0.0309  , Average Score: 0.03
Episode 300    Average Score: 0.0400  , Average Score: 0.04
Episode 400    Average Score: 0.0510  , Average Score: 0.05
Episode 500    Average Score: 0.0710  , Average Score: 0.07
Episode 600    Average Score: 0.0910  , Average Score: 0.09
Episode 700    Average Score: 0.0810  , Average Score: 0.08
Episode 800    Average Score: 0.1119  , Average Score: 0.11
Episode 900    Average Score: 0.2090  , Average Score: 0.20
Episode 1000   Average Score: 0.3920  , Average Score: 0.39
Episode 1100   ,local max score 1.30  , Average Score: 0.50
```

The plot of rewards per episode is shown below:



Future work:

We can try other algorithms such as SAC, A3C, A2C, PPO etc. to improve the results. More effort can be put in fine tuning the hyper parameters – learning rate for actor, critic; number of agents, epochs, iterations, weight_decay etc to improve the results based on current DDPG implementation.
Another thing I want to try is to change the score function. Due to the max function, score of 0 (when the agent completely fails) may not help in guiding the agent to get to a optimum fats enough. So adding an initial small negative value might help.