Project 3: Collaboration Competition

To Solve this problem, the DDGP algorithm was again used using python3 and pytorch. The project consists of three main python:

1) model.py
   Which holds the structure used for the target and local networks for both Actor and Critic. The Actor network consisted of 2 fully connected layers with the inner layers having 400 and 300 neurons respectively. It takes a state as an input of size 24 and outputs the best action for this state of size 2.
   The Critic Network consisted of 2 fully connected layers with the inner layers having 400 and 300 neurons respectively. It takes a state as input in the input layer and the action is inserted with the first inner layer and outputs the Q value for the (S,A) pair.

2) ddgp_agent.py
   Has the implementation of the DDGP agent including the act, step and learn function of the agent along with the Replay_buffer class and the Noise class.

3) main.py
   Has the implementation of the DDGP algorithm where an agent is created and trained.

The chosen hyperparameters are:

```
BUFFER_SIZE = int(1e5)  # replay buffer size
BATCH_SIZE = 128  # minibatch size
GAMMA = 0.99  # discount factor
TAU = 1e-2  # for soft update of target parameters
LR_ACTOR = 1e-4  # learning rate of the actor
LR_CRITIC = 1e-3  # learning rate of the critic
WEIGHT_DECAY = 0  # L2 weight decay
NOISE_DECAY = 0.99  # Noise decay
```

First I initialized the Actor and Critic Networks, the unity environment and the agent. Then I called the `train_agents`() function which trains the agent for at least 2000 episodes until the agents successfully learn the environment by achieving 0.5 or more average points on the last 100 episodes.

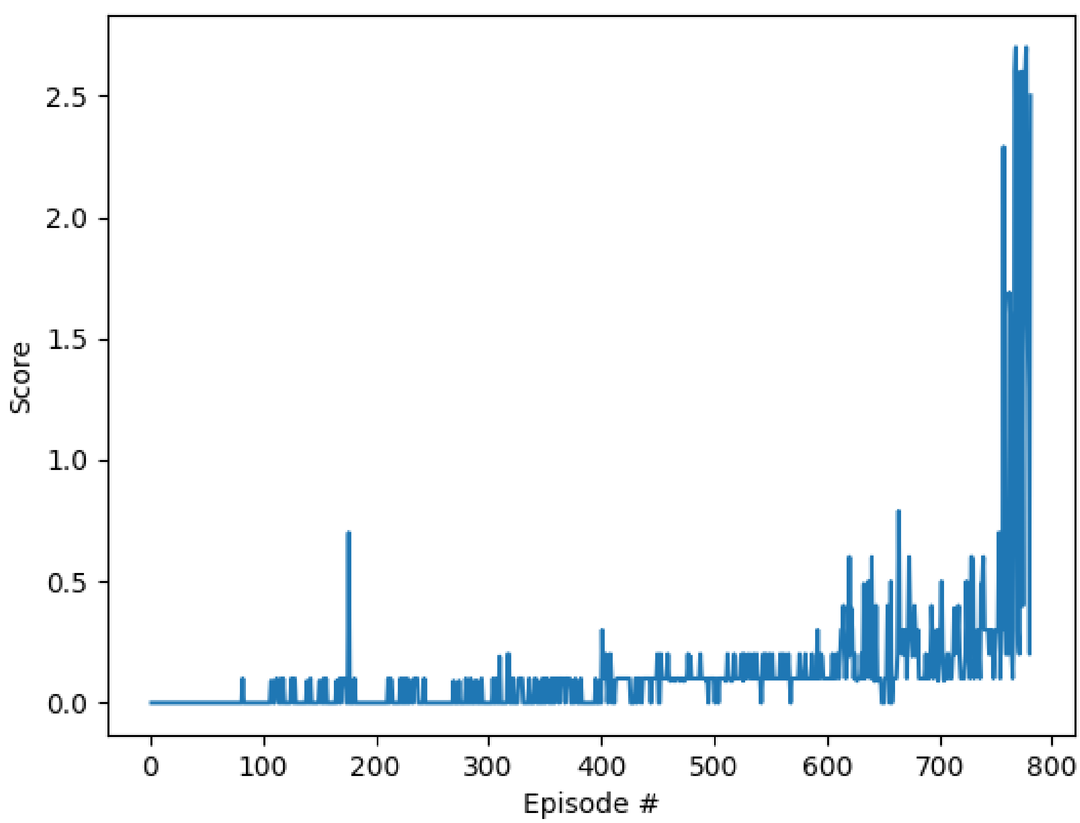
In each episode the environment is reset and an initial state is observed and lasts until the environment is done. In each time step the best action is selected from the local Actor network, then the next state and reward are observed based on the chosen action. This new experience is then saved in the replay buffer by calling the step() function where a learning step is taken if

we have enough experiences. This step function is called twice, once with the experience tuple of the first agent and a second time with the experience tuple of the second agent.
In the learning step, we sample 128 random experiences from the replay buffer and use them to update the local Critic and Actor network. The critic is updated by using the target actor to get the best action for the next_states, then the Q_target is computed from the target critic using these (next_state,action) pairs. The actor on the other hand is updated by computing the mean Q values of the current states and their corresponding best action, then using it to perform a gradient ascent. At the end of each learning step, a soft update is made for the target actor and critic networks.

<u>Tips that helped stabilise the learning process:</u>

1. This time making the LR of the Actor less than the Critic was more beneficial.
2. Removing max time steps per episode and only breaking when the env is done.
3. Increasing the number of nodes in the FC layers in both the Actor and Critic Networks.
4. Increasing the tau a little from 1e-3 to 1e-2.

<u>Results:</u>



As shown in the figure the agent was able to achieve an average score of 0.5 in 681 episodes.

Future work:

I intend to attempt solving this environment using MADDGP.