Report Project 3: Collaborate and Compete

1. Introduction:

In project 2, we trained an agent, Reacher robot, to reach a ball which is moving in a continuous 3D space. In this project, we take train 2 agents to collaborate with each other. The task here is to make sure that the max score of the 2 agents' is greater than 0.5 for 100 consecutive episodes. The agents receive a reward of 0.1 if they are able to hit the ball over the net. And the final score is the max score of both agents. In a sense, this is more of a collaborative environment, in that the agents' want to perform well together. The competitive nature in this environment comes from the negative rewards, in the reward structure. When an agent lets the ball hit the ground, it is awarded a negative reward of 0.01. This motivates the agent to compete with other agent, so as to not let the ball hit the ground.

As in the last project, we use DDPG to solve this task. However, we use it with a few changes to augment DDPG to train multiple agents, in this case 2 agents. Below are the changes to DDPG.

2. Changes to DDPG Algorithm:

The new, modified, algorithm works under the following constraints [1]:

- Learned policies can use only local information at execution time
- Don't assume differentiable model of environment dynamics and
- There is no communication between agents.

So, to build such a system a framework of centralized training with decentralized execution is adapted. To enable this an extension to actor critic is performed, where the critic is expanded to have extra info about policies of other agents [1].

The gradient of the expected return for agent i, $J(\theta_i)$:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{s\sim p^{\boldsymbol{\mu}}, a_i\sim \boldsymbol{\pi}_i}[\nabla_{\theta_i} \log \boldsymbol{\pi}_i(a_i|o_i) Q_i^{\boldsymbol{\pi}}(\mathbf{x}, a_1, ..., a_N)].$$

Where Q_i^{π} is the centralized action-value function which takes as input the actions of all the agents along with some state information. Note that this Q is leant separately and can have arbitrary values for different agents [1].

The gradient of the policies wrt parameters θ_i can be written as:

$$\nabla_{\theta_i} J(\boldsymbol{\mu}_i) = \mathbb{E}_{\mathbf{x}, a\sim \mathcal{D}}[\nabla_{\theta_i} \boldsymbol{\mu}_i(a_i|o_i) \nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}, a_1, ..., a_N)|_{a_i=\boldsymbol{\mu}_i(o_i)}],$$

And finally, note that the experienced replay buffer stores the tuples of all the environments.

2.1 Actor Neural Network Architecture:

The input layer takes the state observation, 24 units. There are 2 hidden layers, 1st hidden layer has 128 units and 2nd hidden layer has 128 units. And the output layer represents the continuous

action vector, with 2 units. Leaky ReLu activation was used after each hidden layer. At the final layer normalization was performed, followed by tanh activation.

2.2 Critic Neural Network Architecture:

The critic neural network is used to get the Q value of the corresponding actions that the actor neural network determines. Hence, the critic neural network takes both the agent's state observations and actions to estimate the Q value. The critic neural network has 2 hidden layers, 1st with 128 units and 2nd with 128 units. Note that the input layer takes the 2 agents' state observations. The 2nd hidden layer takes the outputs of the 1st hidden layer and concatenates the action vectors of the agents' and takes the concatenated vector as input and outputs 128 units. Note that LeakyReLu activation is used after both the hidden layers. Note that the output layer doesn't need activation since the critic NN is used to estimate the Q value.

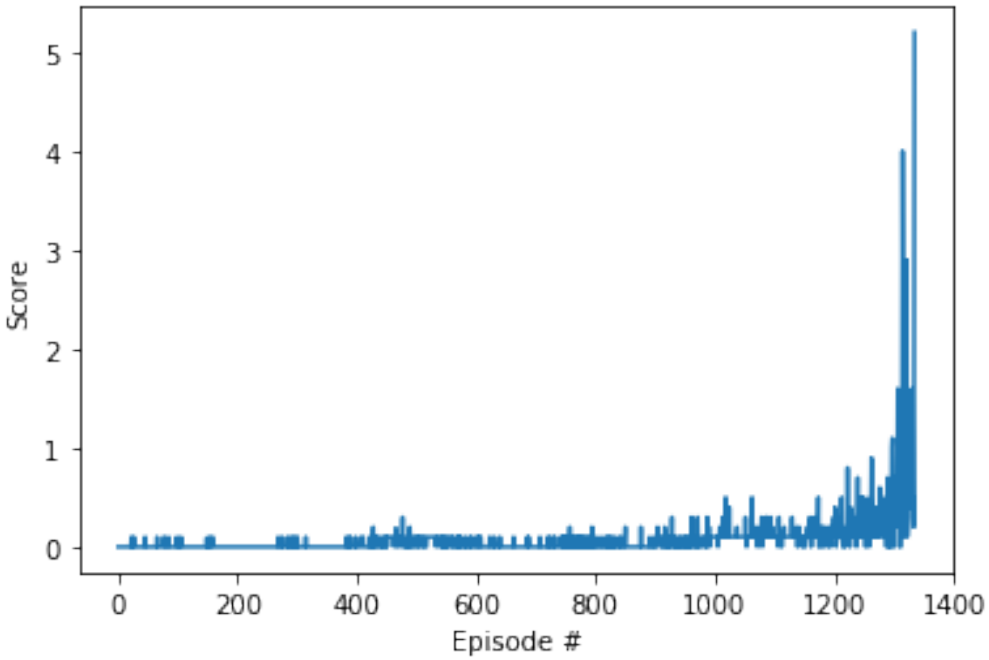
2.3 Hyperparameters:

The following were the optimal tuned hyperparameter settings:

Hyperparameter	**Value**
Replay Buffer Size	1e5
Batch Size	128
Gamma	0.99
Tau (Soft Update rate)	1e-3
Actor Learning Rate	1e-4
Critic Learning Rate	1e-3
UPDATE_NETWORK_FREQUENCY	3
OUNoise Mu	0
OUNoise Theta	0.15
OUNoise Sigma	0.2

3. Results:

The environment was solved in 1334 episodes. Following is the plot of the rewards:



4. Ideas for future work:

- Use other techniques like Proximal Policy Optimization (PPO) and see if we can make valid modifications to them for Multi Agent RL problems.
- Express the problem as a Policy optimization problem. And see if we can represent the multi-agents policy gradients' as an optimization problem. Genetic algorithms might be a good tool to solve such kinds of optimization problems.

5. References:

1 – Multi Agent Actor Critic for Mixed Cooperative-Competitive Environments, Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbel, Igor Mordatch https://papers.nips.cc/paper/7217-multi-agent-actor-critic-for-mixed-cooperative-competitive-environments.pdf

2 – Reference for the code https://github.com/udacity/deep-reinforcement-learning/tree/master/ddpg-pendulum