Project 3 –Collaboration and competition

Introduction

As part of Collaboration and competition project, I have leveraged Multi Agent Deep Deterministic Policy Gradient to train a pair of agents so that they could play tennis game. Here I have trained two agents in parallel reason being game of tennis requires two player to collaborate and compete. This project required long training time and I had to go through several attempts to find right set of hyper parameters to achieve threshold value of the agent to succeed.

Network Architecture

Since I have already hands on experience with DDPG, so I am using it's another variant to train multiple agents and have used Multi Agent Deep Deterministic Policy Gradient.
Each agent has its own actor-critic model implementation. Both the agents use a common replay memory to store the experiences and learn from them to increase the efficiency and create synchronization at the same time.

Both the agents are based on same approach as DDPG

As part of the Deep Deterministic Policy Gradient approach we have built for this project. In this environment, two agents control rackets to bounce a ball over a net. If an agent hits the ball over the net, it receives a reward of +0.1. If an agent lets a ball hit the ground or hits the ball out of bounds, it receives a reward of -0.01. Thus, the goal of each agent is to keep the ball in play.
The observation space consists of 8 variables corresponding to the position and velocity of the ball and racket. Each agent receives its own, local observation. Two continuous actions are available, corresponding to movement toward (or away from) the net, and jumping.

Solving the environment
The task is episodic, and in order to solve the environment, the agents must get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents). Specifically,

After each episode, we add up the rewards that each agent received (without discounting), to get a score for each agent. This yields 2 (potentially different) scores. We then take the maximum of these 2 scores.
This yields a single score for each episode

The environment is considered solved, when the average (over 100 consecutive episodes) of those scores is at least +0.5.
Thus, our final policy has the following parameters:

1. Input Layer - 8 input variables
2. Hidden Layer 1 - 256 units
3. Hidden Layer 2 - 256 units
4. Output Layer - Tanh layer
Both the Hidden layer uses Actor-critic approach

Under the Hood

As part of this implementation, complete code / python notebook can be found at location : https://github.com/divya-bhanot/Project3_collaboration-and-competition

Hyper parameters used

Hyper Parameters are parameters which are given by the users to train a model. Similarly, for DDPG we defined several hyper-parameter to ensure we get optimized results.
Though it took me some iterations to come up with right set of values, following are the final values which helped me get optimum threshold results and their significance:

Buffer Size -> This hyper-parameter define how many values of state, action, etc. we could store in memory. For our model, we have defined this to be 10000

Batch Size -> Number of examples to be picked up when the agent updates the learning curve i.e. for learning updates. I have taken that as 256.

Gamma -> The discount factor which we have used is multiplied by future rewards in order to dampen these rewards' effect on the agent's choice of action. We have used discount value of 1 literally no discount.

LR_Actor -> This rate denotes how the learning curve for Actor optimizes the results in each run and minimize the loss function. The value for current implementation is taken as 2e-4.

LR_CRITIC -> This rate denotes how the learning curve for Critic optimizes the results in each run and minimize the loss function. The value for current implementation is taken as 2e-4.

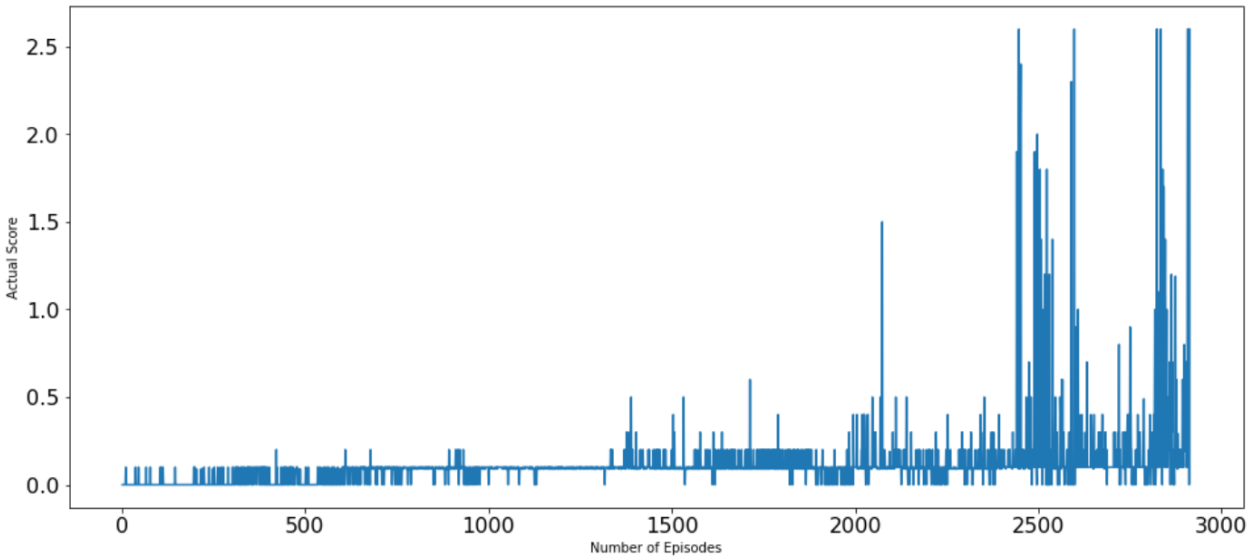
Tau -> Tanh activation factor, which steers the action values corresponding to different actions like position, rotation, velocity, and angular velocities of the arm of the agent.

WEIGHT_DECAY -> Multiplicative factor, which helps while training the agent.

time_steps -> Kept as 30000, intial amount to start the training of agent

Output

As we can see from the resulting plot that multi-agents were trained in around 2800 episodes as they reached the average score of 0.5 after iterating through 2800 episodes. While training the agents, I have found that the training performance by using the GPU instead of CPU is much better and less time consuming. While training the agent with same hyper parameters and using CPU the average score of 0.5 was not achieved even after iterating 4400 episodes.



Future Changes

Here are list of changes which we could do

1. ***Hyper parameter Tuning***: Though I did hyper parameter tuning with different values to achieve threshold results. I could have used more different parameters to achieve similar results/thresholds faster.

2. ***Automated ML Hyper parameter Tuning***: This is a new field of research and
I have been closely following how different companies such as Google who are working on automated ML hyper parameter Tuning. One of the technique is using Bayesian optimization to help in automated ML Hyper parameter Tuning
(We can read more about it here https://towardsdatascience.com/automated-machine-learning-hyperparameter-tuning-in-python-dfda59b72f8a).

I am fascinated by this field, because of the 5 tribes of ML (as mentioned famously by Pedro Domingo's in his book "The Master Algorithm" (https://medium.com/42ai/the-5-tribes-of-the-ml-world-670ebce96b4c)
I could see two tribes i.e. Connectionist (of Deep Learning) and Bayesians coming together to solve this fascinating problem of Automation of Hyper parameter tuning.