Collaborate and Complete

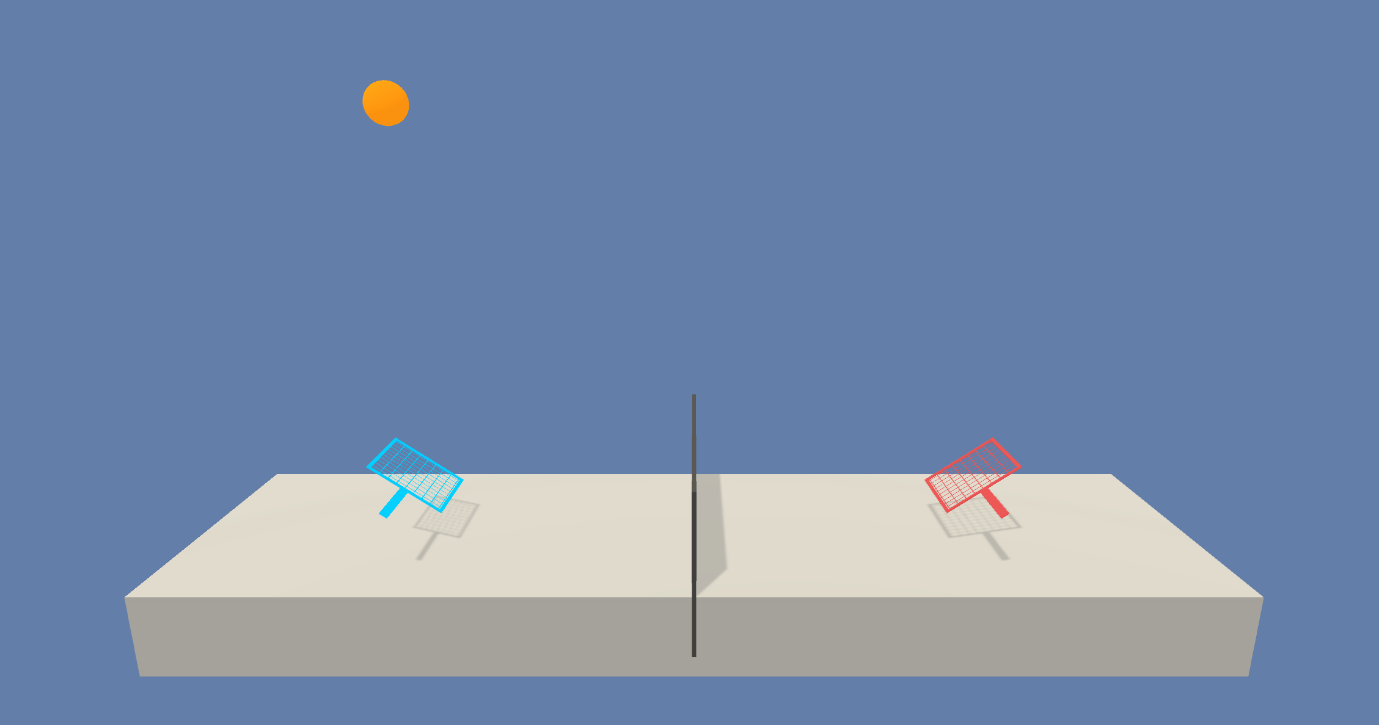
Project Report

# Introduction:

This project involves solving an environment in which two agent (Tennis Rackets) which has freedom of motion in a pitch environment, the rackets have to ensure they receive and strike the ball so that both of the rackets play a rally as long as possible and hence this is a collaboration task. My choice of algorithm for this task is to use MADDPG (Multi Agent Deep Deterministic Policy Gradient) Algorithm which is an off-policy actor-critic algorithm, and is well suited for tasks like these.

# Environment:

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.

The task is episodic, and in order to solve the environment, your 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 episodes) of those scores is at least +0.5.

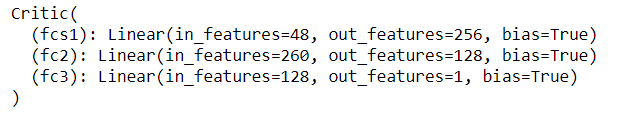
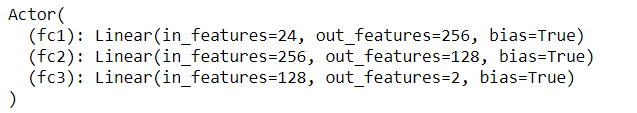
# Implementation:

1. Tennis\_Udacity\_Workspace.ipynb
2. Buffer.py
3. Checkpoint.pth (2 such files)
4. Ddpg.py
5. Maddpg.py
6. Model.py
7. Ounoise.py

Tennis\_Udacity\_Workspace.ipynb is the notebook where training happens. . We initially setup the environment instance and do the basic analysis of it, lie the number of agents (2), actions size (2 for each), state vector size (24 for each) etc. The Actor is required to take in input of 24 dim vector and supposed to give out the most appropriate action to maximize the reward. Critic takes in Full action space (Concatenation of individual observation space) and combined actions as input. Critic is what distinguishes this model from a regular DDPG. This implementation converts an actual non stationary environment to a stationary one.

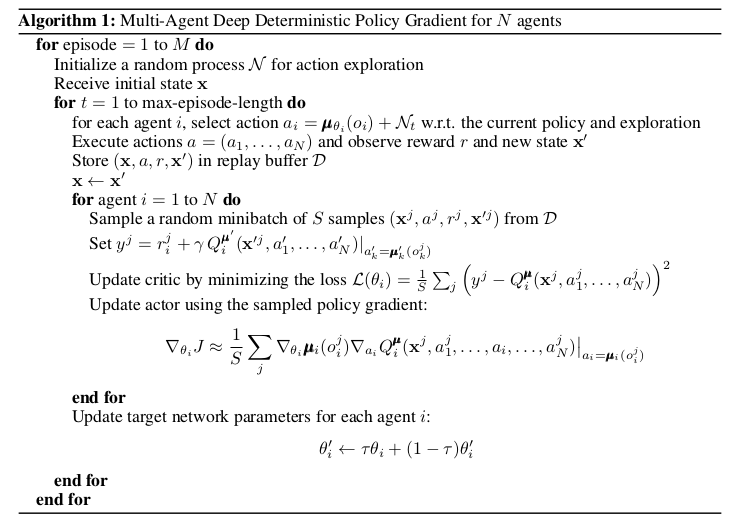
The training loop consists of n\_episodes of the environment. We collect the episodes in a buffer and use it to train the agent. This training happens until the episode is done without stopping in between. And if the average score crosses 1, we save the weights and break the loop.

Model.py consists of two nn.modules for both actor and critic networks. The architectures are given below. Actor has an output activation function of Tanh to get the appropriate range for action space.



# Learning Algorithm:

In MADDPG, each agent’s critic is trained using the observations and actions from all the agents, whereas each agent’s actor is trained using just its own observations. This allows the agents to be effectively trained without requiring other agents’ observations during inference (because the actor is only dependent on its own observations). Here is the gist of the MADDPG algorithm:

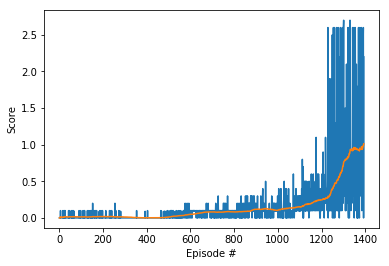


Optimiser : Adam

Loss Function : MSE Loss

|  |  |
| --- | --- |
| Parameter | Values |
| Buffer Size | 1e5 |
| Batch Size | 256 |
| Gamma | 0.99 |
| Tau | 1e-3 |
| LR Actor | 1e-4 |
| LR Critic | 3e-4 |
| Noise start (Epsilon) | 1 |
| Noise End | 0.1 |
| Noise Reduction | 0.999 |
| Num learning per step | 3 |

# Training:



# Ideas for Future work:

1. Try using other algorithms like A3C and PPO.
2. Explore with weight decay aswell
3. Try adding dropouts to the networks.