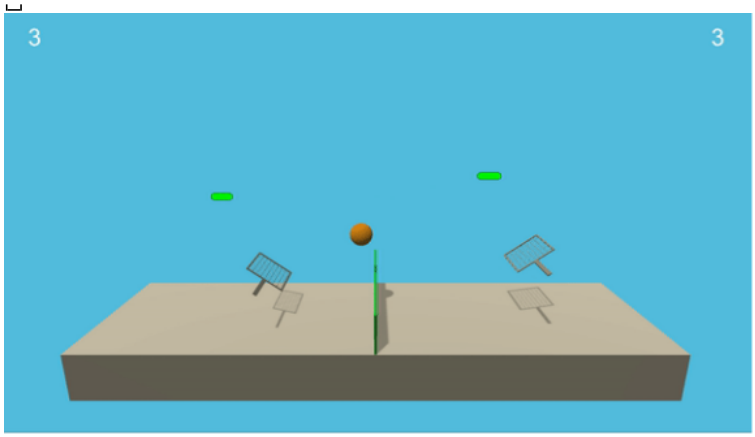
Project 3 – Collaboration and Competition

Jasdeep Sidhu

This report is also available in the form of a medium article by accessing the link here (medium link).

Introduction

This is the final project of Udacity's deep reinforcement learning program where we work with the tennis 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 is 24 dimensional consisting 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.

Learning Algorithm

I will start of this section by providing a reader with some background on why in certain situations multi-agent deep-reinfrocement learning(DRL) is better suited than a single agent DRL. In the next section, multi-agent Deep Deterministic Policy Gradients(MADDPG) will be introduced. It is important to remember that MADDPG structure builds on top of DDPG which in turn is an enhancement of deep Q-learning algorithm(DQN). I have discussed both DDPGand deep Q-learning algorithms in some detail in previous projects.This project assumes prior knowledge of deep Q-Networks(DQN) covered here {see DQN} and Actor-Critic methods in particular Deep Deterministic Policy Gradients(DDPG), covered here {see DDPG}. The learning algorithm used for implementing this project is multi agent DDPG. Finally, in this section, I will discuss the network architecture with hyperparameters implemented for this algorithm.

Why do we need multi-agent DRL?

In order to better model the real world situations, we need multi-agent systems which learn to perform complex tasks while simultaneously collaborating and/or competing with other agents. The more complex the task, more likely that multi-agents are needed. Just to give a simple example, a

task which requires a robot to lift a heavy equipment can be handled much better by two agents where one agent is programmed to lift the load on the left side, while the other focuses on the right side.

One of the advantages of a multi agent apprach is this additional factor of learning via cooperation or competiton. Instead of one agent trying to master several tasks, the tasks can be divided among agents and one can agent can utilize the knowledge aquired by the other. This ensures solving much more complex tasks modeling real world situations such as robotic or simulating a self driving car. However, multi agent DRL comes with its own set of challenges. For example, from the perspective of a single agent, the environment is now no longer stationary as was always the case for a single agent deep RL problem.

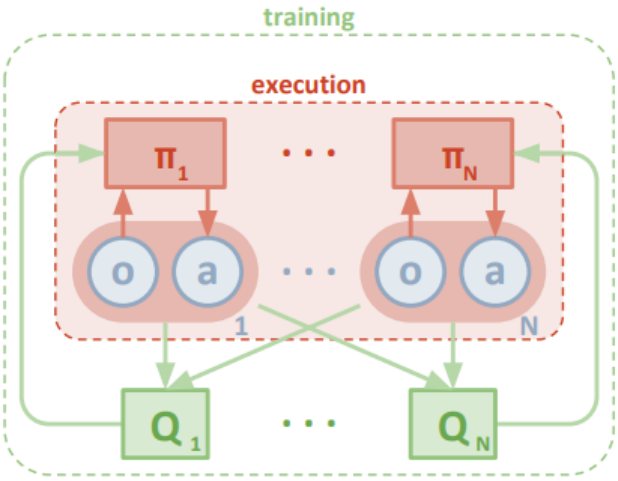
Several methods have been implemented to navigate around the problem of non-stationary nature of the environment. One such apprach that we will use in this project i(tennis environment) is to use MADDPG. The environment for the project requires the training of two separate agents, and the agents need to collaborate under certain situations (like don't let the ball hit the ground) and compete under other situations (like gather as many points as possible). Just doing a simple extension of single agent RL by independently training the two agents does not work very well because the agents are independently updating their policies as learning progresses. And this causes the environment to appear non-stationary from the viewpoint of any one agent. While we can have non-stationary Markov processes, the convergence guarantees offered by many DRL algorithms such as Q-learning requires stationary environments.

Multi Agent Deep Deterministic Policy Gradients

Recall that the Deep Deterministic Policy Gradient (DDPG) lies under the class of Actor Critic Methods but is a bit different than the vanilla Actor-Critic algorithm. The actor produces a deterministic policy instead of the usual stochastic policy and the critic evaluates the deterministic policy. The critic is updated using the TD-error and the actor is trained using the deter-

ministic policy gradient algorithm. Infact, it could be seen as approximate Deep-Q Network(DQN) method. The reason for this is the critic in DDPG is used to approximate the maximizer over the Q-values of the next state, and not as a learned baseline, as have seen so far. One of the limitation of the DQN agent is that it is not straightforward to use in continuous action spaces. For example, how do you get the value of a continuous action with DQN architecture? This is the problem DDPG sovles.

Now coming back to MADDPG, in the previous section I mentioned the non-stationary environment problem that arises in multi-agent algorithms. MADDPG attempts to solve this problem by using centralized critic and decentralized actor apprach, as seen in the fig. below.



The figure has been taken from the Open-AI paper of Wu et. al. on MADDPG(see MADDPG paper). The key idea in this paper is that if all the actions of each agent are known then the environment is no longer non-stationary even if the policites of each agent are changing.

Network Architecture The network achitecture for the actor is comprised of two fully connected hidden layers of 256 units with ReLU activations. In-order to help speed up learning and avoid getting stuck in local minimum, batch normalization was introduced to each hidden layer for both actor and critic networks. The hyperbolic tan activation was used on the out-

put layer for the actor network as it ensures that every entry in the action vector is a number between -1 and 1. Recall that the state space was 24-dimensional and the output (action space) is 2-dimensional for the actor. In case of critic, the output(action-space) is one-dimensional corresponding to a single agent. In case of critic, I have used hidden units to be 128 units each corresponding to a shared critic network. Rest of the implementation for the critic is same as in the DDPG case that is we add the state space and action space for the fully connected second layer, and then apply relu activation. Adam was used as an optimizer for both actor and critic networks.

Hyperparameters

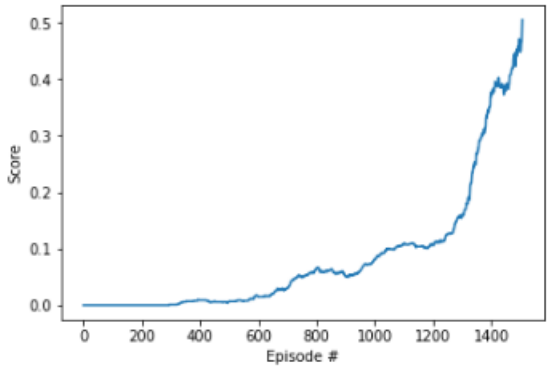
Hyperparameters	Values
Batch size	256
Replay buffer size	1e5
Discount factor(GAMMA)	0.999
update interval	4
tau	1e-3
learning rate(Actor)	1e-4
learning rate (Critic)	3e-4
Number of episodes	3000
Maximum number of time-steps per episode	4000
Weight decay	0
Noise hyperparameter:{mu}	0
Noise hyperparameter:{theta}	0.15
Noise hyperparameter:{sigma}	0.1

Results

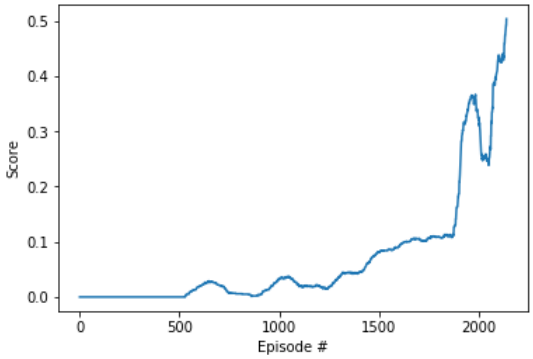
The problem was solved using MADDPG algorithm where the average reward of +0.5 obtained over atleast 100 episodes was achieved in 1505 episodes. The result depends significantly on the fine tuning of the hyper-

parameters. For example, if the number of time steps are too high/low, learning rate or the seed is too large or small, the system can fall into a local minima where the score may start to decrease after a certain number of episodes.Batch normalization layer was added to the network architecture to help improve training. Below I plot the best result I obtained first with the hyperparameters listed above.

The plot of the rewards across episodes is shown below:



In an earlier version with slightly different set of hyperparameters, I had found the training to be slower with oscillatory patterns in particular a dip as seen from around episodes ~1800 to 2050. For this version the problem was solved in 2138 episodes.



I had made two modifications that helped in improving the training. Firstly, I changed the discount factor gamma from 0.99 to 0.999. This would mean that the discount factor now is a bit closer to 1 and that way the terms in the later time-steps per episode are slightly smaller in value as compared to earlier time-steps. Secondly, from looking at the oscillator

patterns, I figured that learning rate could be too small. So, I slightly increased the learning rate and in particular I gave the critic a slightly higher learning rate than the actor so that the critic learns a bit faster than the actor as the actor uses information given by the critic to change its policy. The value of the final hayperparameters can be seen in the hyperparameters section.

Future Improvements

One implementation that can be made to this algorithm is incorporating prioritized replay. Recall, that any variant of DDPG algorithm uses a Replay buffer, the purpose of which is to avoid harmful correlations. Prioritized replay can help further reduce the correlations so that there is less bias in the agent's training while it learns the optimal policy. Another avenue of future research is to use Multi Agent Proximal Policy Optimization (MAPPO) to see how it compares to MADDPG. For the purposes of this nano-degree I have concentrated on DDPG and its variants only but MAPPO could be an interesting option.