Introduction

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.

Multi-agent Deep Deterministic Policy Gradients (MADDPG)

MADDPG is an actor-critic approach. Derived from DDPG, it manages to solve tasks with multiple agents. As an off-policy algorithm, MADDPG utilizes four neural networks (a local actor, a target actor, a local critic and a target critic). Their playing experiences (state, action, action_other_agent, reward, next state, next_state_other_agent) at each step are stored.

For each training step, the agents learned from a random sample from the stored experience. The actor tries to estimate the optimal policy by using the estimated state-action values. The critic tries to estimate the optimal q-value function and learns by using a normal q-learning approach. Through giving the actor and critic access to the action of the other player, the learning process gets stabilized without additional information.

For the details of DDPG, please refer to the second project

Architecture

The network architecture is comprised of two fully connected hidden 128 units layers with leaky_relu activations, respectively. In order to help speed up learning and avoid getting stuck in a local minimum, batch normalization was introduced to each hidden layer. The hyperbolic tan activation was used on the output layer for the actor-network as it ensures that every entry in the action vector is a number between -1 and 1. Adam was used as an optimizer for both actor and critic networks.

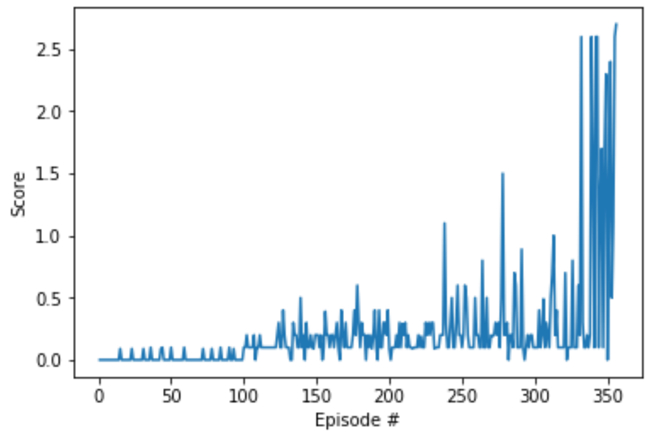
```
---Actor---
Actor(
  (bn1): BatchNorm1d(24, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc1): Linear(in_features=24, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=128, bias=True)
  (fc3): Linear(in_features=128, out_features=2, bias=True))
---Critic---
Critic(
  (bn1): BatchNorm1d(24, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fcs1): Linear(in_features=24, out_features=128, bias=True)
  (fc2): Linear(in_features=132, out_features=128, bias=True)
  (fc3): Linear(in_features=128, out_features=1, bias=True))
```

Hyperparameters

The various hyperparameters used are as follows:

```
# hyperparameters

BUFFER_SIZE = int(1e5)  # replay buffer size
BATCH_SIZE = 128        # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR_ACTOR = 1e-4         # learning rate of the actor
LR_CRITIC = 1e-3        # learning rate of the critic
LEARN_EVERY = 1         # learn every LEARN_EVERY steps
LEARN_NB = 1            # how often to execute learn-function every LEARN_EVERY steps
```



Plot of average scores each episode.

The environment is considered solved in 256 episodes

Future works

Some ideas to improve the agent's performance in the future:

- Tune up hyperparameters to improve computation time and scores
- Explore prioritized replay to improve performance
- Try algorithm PPO which was used to solve Dota2
- Try solving Soccer environment using a similar implementation