Introduction

This is my report of project 3: Collaboration and Competition. In this project, I trained two agents to control rackets bouncing 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, agents must get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents).

Algorithm

I adapted DDPG to simultaneously train both agents through self-play in order to solve the problem. In this case, each agent used the same actor network to select actions, and the experiences were added to a shared replay buffer. The basic structure of the DDPG algorithm is shown below:

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Hyperparameters:

buffer size: 1000000

batch size: 1024

γ (discounting rate): 0.99

τ (soft update): 0.001

learning rate for actor: 0.0001

learning rate for critic: 0.001

number of time steps that model learns: 20

number of times that model updates: 10

noise parameters: 0.2 (σ), 0.15 (θ), 1.0 (ϵ -init), 1e-6 (ϵ -decay), 0.1 (ϵ -min)

Architecture of the actor network:

input layer (#24) -> hidden layer (#128) -> batch normalization -> hidden layer (#64) -> output layer (#2)

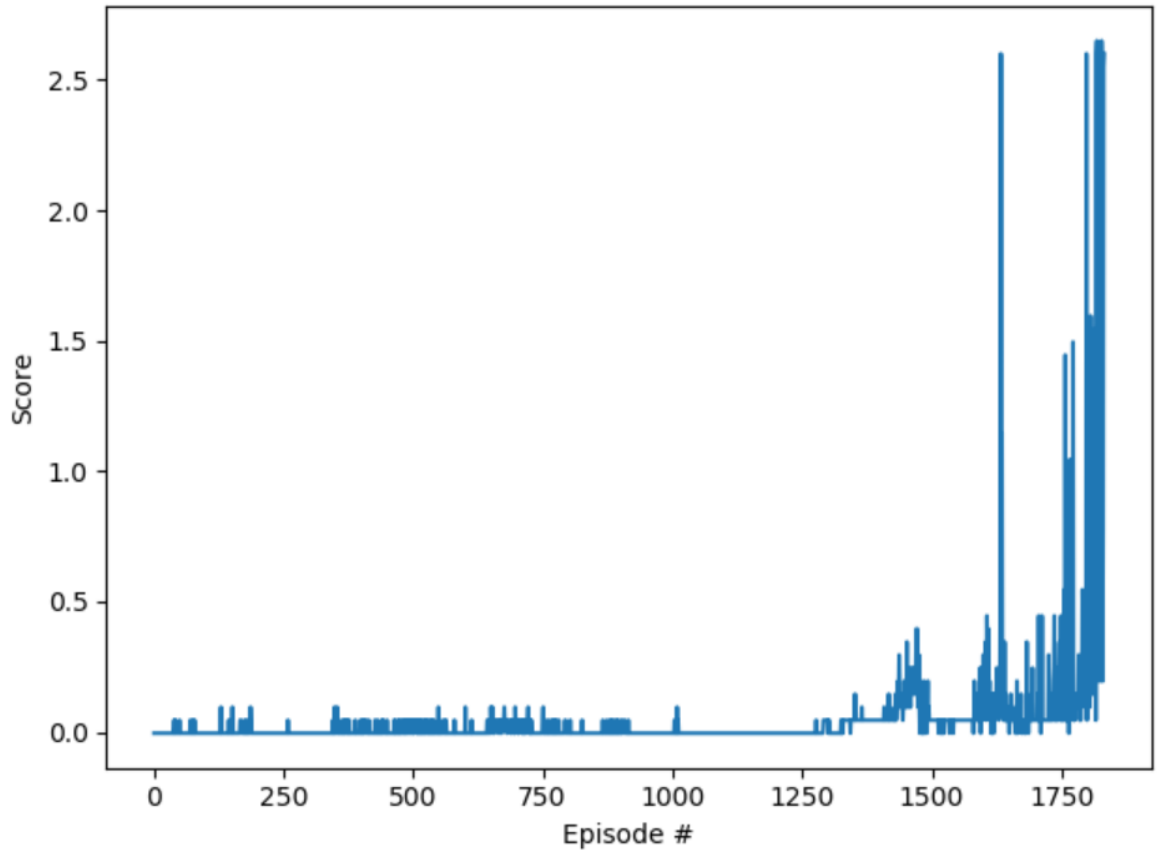
All fully connected layers use ReLU for activation except for the output layer which uses tanh for activation.

Architecture of the critic network:

input layer (#24) -> batch normalization -> hidden layer (#128) -> batch normalization -> concatenate actions -> hidden layer (#64) -> -> batch normalization -> output layer (#1)

All fully connected layers use Leaky-ReLU for activation except for the output layer which doesn't use an activation.

Result



The problem was solved in 1947 episodes, with average score of 0.51.

Future work

Tune hyperparameters to achieve faster learning and implement MADDPG.