DEEP REINFORCEMENT LEARNING PROJECT 3 – COLLABORATION AND COMPETITION

The goal of this project is to train two agents to 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. Furthermore, 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).

LEARNING ALGORITHM

I used an approach inspired by my solution to Project 2 (link). Specifically, I reused the Deep Deterministic Policy Gradient (DDPG) algorithm because of its excellent performance on continuous control problems [1]. Recall, the DDPG algorithm involves an actor and critic neural networks. The actor determines the next action to take given the environment's state as input while the critic predicts the value ($Q(s, a)$) of the action taken in that state. The actor is trained using a sampled policy gradient (which is derived from the Q-values produced by the critic) while the critic is trained to minimize the one-step TD-Error (which is derived from the state, action, reward, next state produced by the actor interacting with the environment).

However, unlike the classic DDPG algorithm, my solution involved the two agents sharing the actor network (which one can think of as implementing self-play) but each agent maintained its own critic network. Furthermore, the agents shared a replay buffer so that each agent could learn from the experience the other.

ACTOR-CRITIC NEURAL NETWORK ARCHITECTURE

As in Project 2, I started from Udacity's DDPG implementation [2] for the OpenAI Gym's Pendulum environment and evaluated multiple hyperparameter settings until I solved the environment. However, I reused the learning rates for the actor and critic networks that I had arrived at in Project 2.

Shared Actor Neural Network

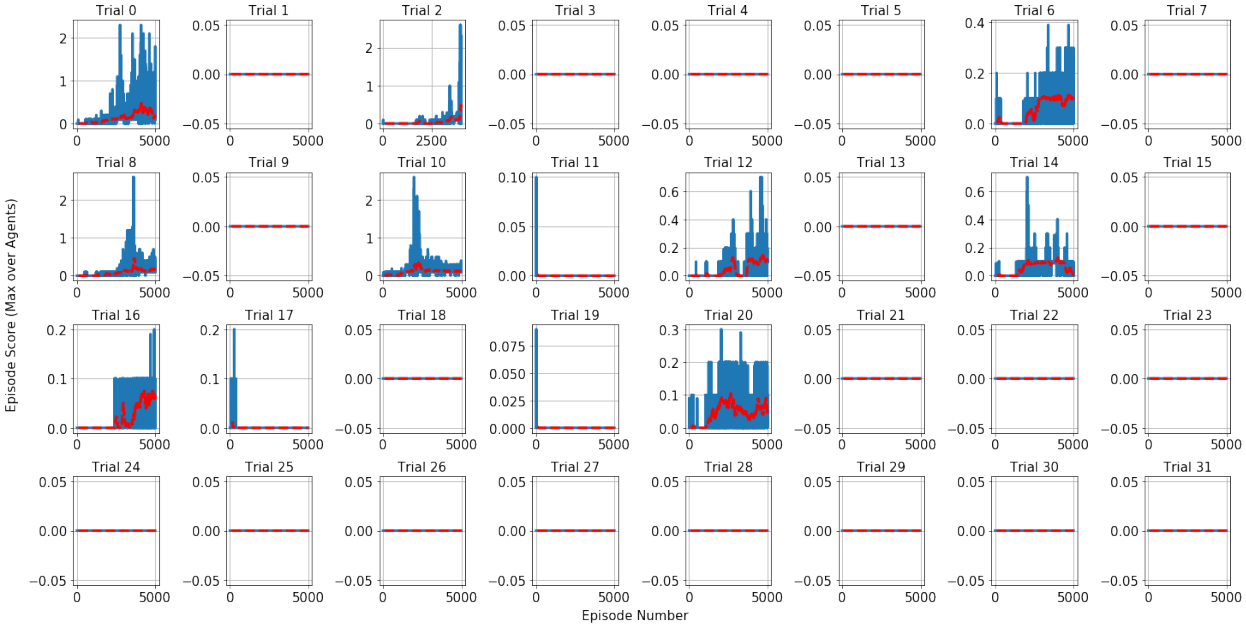
The actor (shared between both agents) used three fully-connected layers. The first layer had 32 hidden nodes, a ReLU activation function, and batch normalization. The second layer had 64 hidden nodes and ReLU activation function. The third layer had 2 hidden nodes (corresponding to the action-space size) and a Tanh activation function (to produce actions between -1 and 1). The actor was trained with a learning rate of 2.6e-3 and batch size of 128.

Critic Neural Network

The critic (one for each agent) used three fully-connected layers. The first layer had 32 hidden nodes, a ReLU activation function, and batch normalization. The second layer had 64 hidden node and ReLU activation function. The third layer had 1 hidden node and a no activation function (to produce a single q-value). The critic was trained with a learning rate of 1.9e-4 and batch size of 128.

DETERMINING NEURAL NETWORK HYPEPARAMETERS

I conducted 32 trials that involved setting the number of nodes in the first and second fully-connected layers of the actor/critic networks to values in the set [32, 64, 128, 512] as well as whether each network should use batch normalization after the first fully-connected layer (see project IPython notebook cell titled "Hyperparameter sweep"). The plot below shows the scores of each trial for 5000 episodes unless a trial terminated early due to a success. The blue line is the raw score and the red line is the mean score (maxed over agents) for the previous 100 episodes. *All trials failed with the exception of Trial 2 which solved the environment with a score 0.5156 after 3994 episodes. Trial 2 involved 32 hidden nodes in the first layer of the actor/critic; 64 hidden nodes in the second layer of the actor/critic; and the use of batch normalization after the first layer of both the actor and critic.*



TRAINING TRICKS

Similar to Project 2, a few techniques were employed to help the 2 agents learn.

Sharing Experiences

A shared replay buffer (of size 5e6) accumulated the experiences (state, actions, reward, next state) of all the agents. From this replay buffer all agents sampled mini-batches during training.

Sharing Actor Networks

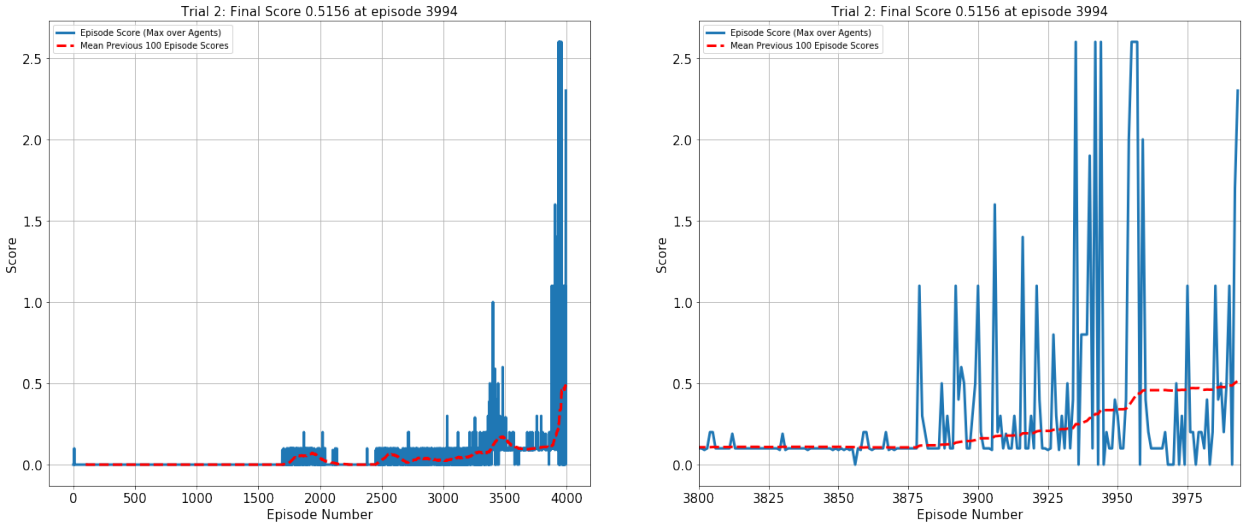
The agents shared the Actor network which allowed the implementation of self-play a technique that has proven very effective in algorithms such as AlphaZero.

Decrease Frequency of Learning

As noted in the description of Project 2, it is also beneficial to decrease the number of updates made to the network. So, rather than update the network at every timestep, agents should interact with the environment for a while (e.g. 20 timesteps) and then take a few learning steps (e.g. 10 steps).

RESULTS

The plot below is a larger view of the results of the winning trial. The blue line is the raw score and the red line is the mean score over the previous 100 episodes. Trial 2 solved the environment with a score 0.5156 after 3994 episodes. The plot in the left panel shows that the agents required approximately 1500 steps before they began to accumulate a positive score. The plot in the right panel illustrates agent performance for episodes 3800-3994 and demonstrates how the mean score (max over agents for previous 100 episodes) crosses the 0.5 threshold.



FUTURE WORK

Future work on this project would testing other multi-agent reinforcement learning algorithms. Specifically, I would be interested in testing a "meta-agent" learning algorithm where the algorithm would use a single actor/critic network pair to specify the actions of both agents. Moreover, I'd also be interested in testing algorithms specifically designed for multi-agent problems such MADDPG [3].

REFERENCES

[1] Lillicarp et al. *Continuous Control with Deep Reinforcement Learning*. arXiv:1509.02971v5 (link)

[2] Udacity Deep Reinforcement Learning Github (link)

[3] Lowe et al. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments (link)