Project 3: Collaboration and Competition

The main aim of this project is to train two agents to control rackets to bounce a ball over a net. The goal is for the agents to control rackets to bounce a ball over a net and keep the ball in play for as many time steps as possible; hence, if an agent hits the ball over the net, it receives a reward of +0.1. However, if the agent lets a ball hit the ground or hits the ball out of bounds, it receives a reward of -0.01.

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.

For this project, an environment built in Unity is used.

Learning Algorithm

To solve the environment, a Deep Deterministic Policy Gradients (DDPG) algorithm is used. The DDPG code considers only a single agent, and with each step, the agent adds it experience to the replay buffer, and the actor and critic networks are updated, using a sample from the replay buffer.

Some of the parameters used include:

- `BUFFER_SIZE = int(2e5)`: This is the replay buffer size
- `BATCH_SIZE = 256`: This is the size of each minibatch
- `GAMMA = 0.99`: This is the discount factor
- `TAU = 1e-3`: This is used for soft update of target parameters
- `LR_ACTOR = 2e-4`: This is the learning rate of the actor
- `LR_CRITIC = 3e-3`: This is the learning rate of the critic
- `LEARN_EVERY = 1`: This is the learning timestep interval
- `LEARN_NUM = 1`: This is the number of learning passes

Model Architecture

To solve the environment, three fully connected layers is used for the actor-critic network. Furthermore, a batch normalization layer is also used.

For the actor network:

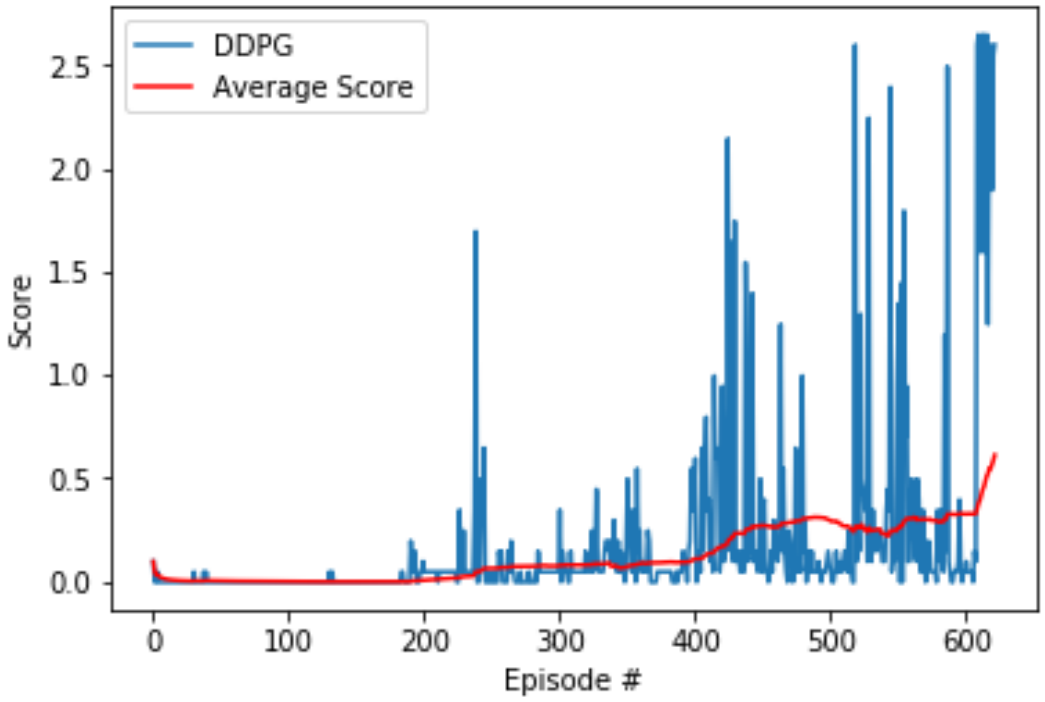
```
self.bn1 = nn.BatchNorm1d(state_size)
self.fc1 = nn.Linear(state_size, fc1_units)
self.bn2 = nn.BatchNorm1d(fc1_units)
self.fc2 = nn.Linear(fc1_units, fc2_units)
self.bn3 = nn.BatchNorm1d(fc2_units)
self.fc3 = nn.Linear(fc2_units, action_size)
x = self.bn1(state)
x = F.relu(self.bn2(self.fc1(x)))
x = F.relu(self.bn3(self.fc2(x)))
return F.tanh(self.fc3(x))
```

For the critic network:

```
self.bn1 = nn.BatchNorm1d(state_size)
self.fcs1 = nn.Linear(state_size, fcs1_units)
self.bn2 = nn.BatchNorm1d(fcs1_units)
self.fc2 = nn.Linear(fcs1_units+action_size, fc2_units)
self.bn3 = nn.BatchNorm1d(fc2_units)
self.fc3 = nn.Linear(fc2_units, 1)
x = self.bn1(state)
x = self.fcs1(x)
xs = F.relu(self.bn2(x))
x = torch.cat((xs, action), dim=1)
x = self.fc2(x)
x = self.bn3(x)
x = F.relu(x)
return self.fc3(x)
```

Result

The agents are trained until the environment is solved. To achieve this, the agents obtained an average score of at least +0.50 over the last 100 episodes.



The environment was solved in 522 episodes with an average score of +0.61.

Ideas for Future Work

For future work, I intend to implement PPO and Prioritized Experience Replay.