Collaboration & Competition

The purpose of the collaboration & competition project was to train multiple agents to control their respective racket & bounce a ball over a net with the goal of keeping a ball in play for as many time steps as possible during a tennis game. A multi-agent DDPG approach was used. The trained agents successfully reached the goal of achieving an average of 0.5 points on the environment after training for ~3800 episodes. Agents were trained using seed 0 for agents & environment & evaluated for 100 episodes using seed 5 for agents & environment respectively. An example of agent performance during training & evaluation are provided below (95% confidence interval shown in light red).

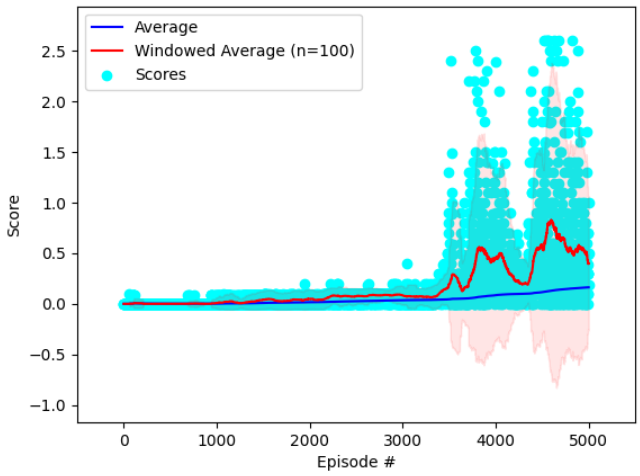

Figure 1 – MADDPG Training

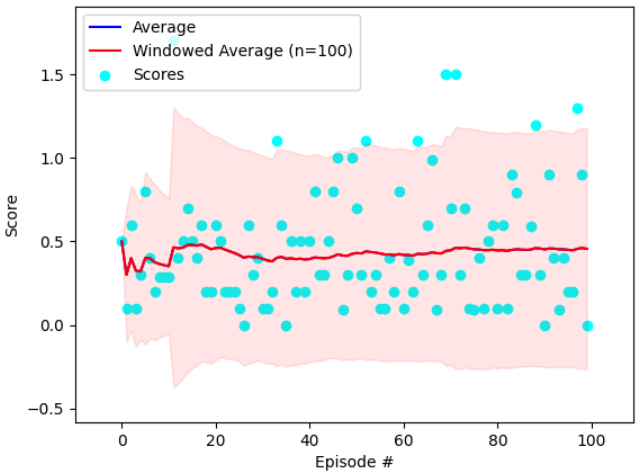

Figure 2 – MADDPG Evaluation

Deep Deterministic Policy Gradient (DDPG) Agent

The DDPG agent used was based off the DDPG bipedal solution provided in the Udacity Deep Reinforcement Learning repository. Additional code was added to save the best performing model during training. Further details about the agents & associated Actor-Critic models are provided in the sections below.

DDPG Model Architecture

A straightforward multilayer perceptron approach was used for each DDPG Actor & Critic local & target networks. Rectified Linear Unit (ReLu) activation functions were used for Actor layers except for final output which used a hyperbolic tangent (tanh). Leaky ReLu activation functions were used for all Critic layers except for final output. A summary of the network architecture is provided below. Critic networks were also clipped during learning to improve stability during training.

Actor Local & Target Networks for Both Agents

```
Actor(
   (fc1): Linear(in_features=24, out_features=256, bias=True)
   (fc2): Linear(in_features=256, out_features=128, bias=True)
   (fc3): Linear(in_features=128, out_features=2, bias=True)
)
```

Critic Local & Target Networks for Both Agents

```
Critic(
   (fcs1): Linear(in_features=24, out_features=256, bias=True)
   (fc2): Linear(in_features=258, out_features=128, bias=True)
   (fc3): Linear(in_features=128, out_features=1, bias=True)
)
```

Agent Hyperparameters

Default values provided from the DDPG bipedal example were modified until an agent successfully passed the 0.5 point reward goal. A summary of agent training parameters is provided below.

Table 1 - DDPG Training Parameters for Each Agent

Buffer Size: 1000000	*Batch Size: 256*	*Gamma: 0.99*	*Tau: 1e-3*	*LR_Actor: 1e-4*
LR_Critic: 3e-4	*Weight Decay: 0.0*	*Seed: 0*	*Optimizer: Adam*	

Future Work

There are many improvements that could potentially improve DDPG agent performance. Goals would be not only to increase averaged windowed score of DDPG agent, but also reduce standard deviation to demonstrate more stable agent performance between episodes (example below) & improve training time / early stop conditions after reaching target goal.

Episode 0 Average Score: 0.00 ± 0.00 Elapsed time: 00:00:00
Episode 100 Average Score: 0.00 ± 0.01 Elapsed time: 00:00:24
Episode 200 Average Score: 0.00 ± 0.02 Elapsed time: 00:00:53
…
Episode 3600 Average Score: 0.21 ± 0.34 Elapsed time: 00:32:26
Episode 3700 Average Score: 0.17 ± 0.20 Elapsed time: 00:34:55
Episode 3800 Average Score: 0.52 ± 0.55 Elapsed time: 00:42:17
Episode 3900 Average Score: 0.48 ± 0.48 Elapsed time: 00:49:05
Episode 4000 Average Score: 0.46 ± 0.40 Elapsed time: 00:55:32
Episode 4100 Average Score: 0.40 ± 0.42 Elapsed time: 01:01:32
Episode 4200 Average Score: 0.23 ± 0.16 Elapsed time: 01:05:07
Episode 4300 Average Score: 0.20 ± 0.13 Elapsed time: 01:08:10
Episode 4400 Average Score: 0.39 ± 0.41 Elapsed time: 01:14:11
Episode 4500 Average Score: 0.56 ± 0.44 Elapsed time: 01:22:53
Episode 4600 Average Score: 0.82 ± 0.78 Elapsed time: 01:35:46
Episode 4700 Average Score: 0.62 ± 0.59 Elapsed time: 01:46:03
…

A list of potential improvements, grouped by category, are provided below.

Agent Algorithm Modifications

- First approach would be to modify provided MADPPG method for the Tennis simulation environment
- Prioritized experience replay would be another approach as it may be the easiest to implement by sampling with respect to memory error as distribution probability

Model Modifications

- Add dropout layers between fully connected Actor & Critic MLP layers to prevent overfitting & improve generalizability
- Investigate the use of batch normalization layers for both Actor & Critic networks

Hyperparameter Optimization

- Optimize hyperparameters by empirically testing various parameter combinations or using some Bayesian approach
- Use a learning rate scheduler to decrease learning rate throughout training to promote the discovery of better performing policies
- Similarly, scaling noise over time might help agent exploration