

Project 3

Collaboration and Competition Report

Udacity Deep Reinforcement Learning NanoDegree

Author: Denis O'Connor

Implementation

The base of this implementation was taken from the Udacity Deep Reinforcement Learning [ddpg-pendulum](#) project. I relied heavily on implementation from Project 2. I was surprised that I didn't have to modify more. In addition to the base code used, I had to play around with several other elements. Initially, while trying to train with the original DDPG code I was encountering issues with `UnityActionExceptions`. I wasn't properly handling the step and act functions in my Agent. After perusing the Udacity Knowledge forum and going through the MADDPG lab closer, I was able to correct this. I updated the Agent constructor to take the number of agents, and then had to update the act function to properly handle and return actions. Additionally, I needed to use a different constructor for the `OuNoise` object by passing the now available number of agents

Learning Algorithm

Per [Medium.com](#), I used this explanation for the basis of my learning algorithm. I used a Deep Deterministic Policy Gradients approach. Also per [Medium.com](#), "The network architecture is comprised of two fully connected hidden layers of 128 units each with ReLU activations. 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." However, through my troubleshooting I commented out the batch normalization after the second layer's activation. At the time, I was hitting problems where training was resulting in an average score of 0.0. Although, I did not try uncommenting it out after solving some of my other workspace issues.

Hyperparameters

Batch size: 128

Replay buffer size: 1e6

Gamma (discount factor): 0.99

TAU: 1e-3

Actor learn rate: 1e-3

Critic learn rate: 1e-3

Weight decay: 0

OUNoise theta: 0.15

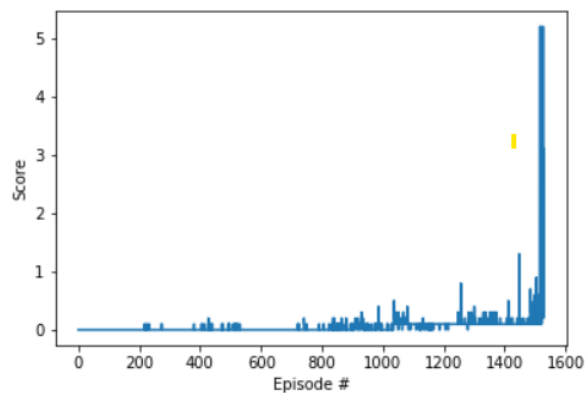
OUNoise sigma: 0.1

Maximum Timesteps Per Episode: 1000

Results

Environment solved in 1530 episodes! Average Score: 0.51

```
In [10]: fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(np.arange(len(scores)), scores)
plt.ylabel('Score')
plt.xlabel('Episode #')
plt.show()
```



Future Improvements

First improvement I would implement is to reintroduce the batch normalization to the second hidden layer. Although, for once I was fairly pleased with my results, I believe tweaking the hyperparameters could yield better results. Also, it would be a great learning experience to try to apply my implementation to the Soccer Environment.