Udacity Project 3 – Competition / Collaboration
by Jon F. Hauris

Introduction

For this project I used a MARL (multi-agent) version of DDPG (Deep Deterministic Policy Gradient) algorithm with Experience Replay to train 2 agents to cooperate at playing a game of "tennis". The 2 agents control tennis rackets that have 2 actions: move in the x direction and have a "jump" or "hit" action. The rackets are to keep the ball in the air hitting back and forth over the net. Each time the ball goes over the net the reward is +0.1. If the ball hits the ground or goes out of bounds the reward is -0.01. The goal is to keep the ball in play and gain a total reward of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents).

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.

The control actions are continuous, as is the state space.

This work is based on and inspired by the code developed in the lessons, code by Shangtong Zhang, and code by Jeremi Kaczmarczyk, as well as papers recommended by the class.

Learning Algorithm

The multi-agent version of the DDPG algorithm is very similar to the regular DDPG algorithm except that it each agent has it's own actor and critic networks (targets and locals). Additionally instead of periodic updating of the actor weights, the weights are updated via a "soft" mechanism each time step.

The basic operation for each episode is as follows. In the main training loop, the environment is reset and an initial state observation is obtained from the environment. Next, for specified time steps the following sequence is executed for each agent:

- randomly select from the action space for specified steps, add noise, and decay noise parameter and clips the action (done in MADDPGAgent-act function)
- use this action to execute a step within the environment
- obtain the next state, reward, and done parameters
- using these parameters have the agent(s) step through the learning method
    - see below for more explanation
- update the state with next state
- gather rewards

The above "agent" is derived from the MADDPGAgent routine. However this code uses the DDPGAgent class within the ddpgagnet.py code to create a variable multi-level fully connected neural network (taget and local) for each agent. The MADDPGAgent-act function calls the DDPGAgent-act function which, for the actor_local network, takes the action, adds

noise, decays the noise, rescales the action, and finally clips the action. This is performed for the local net for each agent.

Once the next state, reward, and done is obtained (using this action) the main part of the learning is performed. The MADDPGAgent-step function recalls: states, actions, rewards, dones, and next states, from the replay buffer. If sufficient elements are obtained the MADDPGAgent-learn function is called. This function formats the above paramters and calls the DDPGAgent-step function for each agent.

This is the core of the learning. Here Q(s',a') = critic_target(s',a'), that is q_next is obtained form the output of the critic_local network. Then q_expected = q_exp is determined from the output of the critic_local network, Q(s,a) = criticl_local(s,a).

Then Q target is essentially calculated as: q_t = rewards + gamma*q_next, and the loss then equals: loss = mse [q_t – q_exp] = mse [r + gamma*Q(s',a') - Q(s,a)]
This loss is then used to optimize the critic network.

The actor loss is: action_pred = actions from actor_local. Then the states and the action_pred is applied to the critic_local network and this output calculates the loss as:
actor_loss = - critic_local(states, actions_pred)
This loss is used to optimize the actor network.

Finally, the actor_target and critic_target networks are soft updated with the Tau parameter from the actor_local and critic_local networks.
And the process repeats for the number of specified episodes.

The general structure of the networks are as follows, however, more layers may be added if desired:

actor_local
Network(
  (input): Linear(in_features=24, out_features=256, bias=True)
  (output): Linear(in_features=256, out_features=2, bias=True)
)

actor_target
Network(
  (input): Linear(in_features=24, out_features=256, bias=True)
  (output): Linear(in_features=256, out_features=2, bias=True)
)

critic_local
Network(
  (input): Linear(in_features=52, out_features=512, bias=True)
  (output): Linear(in_features=512, out_features=1, bias=True)
)

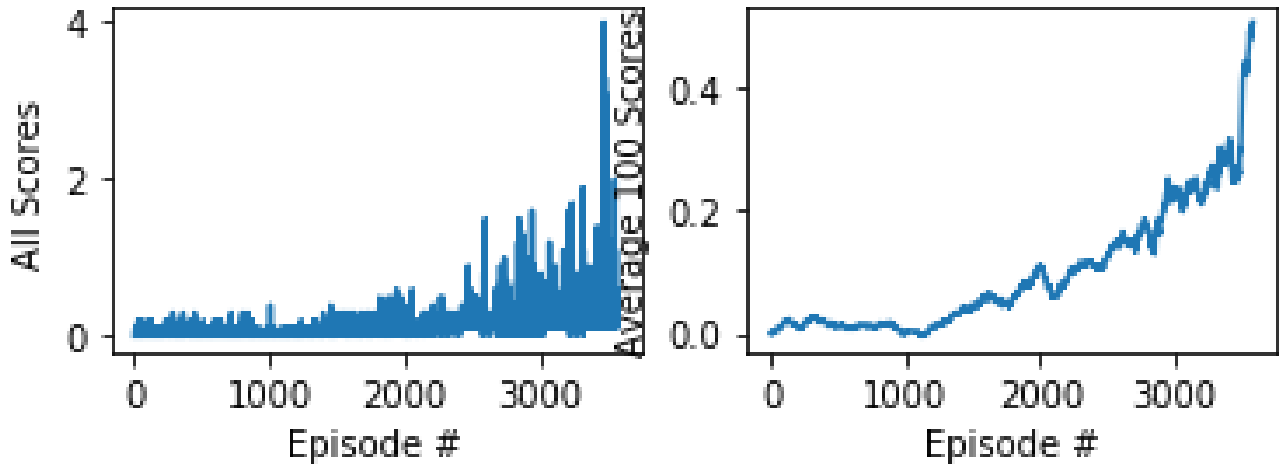
critic_target
Network(

```
  (input): Linear(in_features=52, out_features=512, bias=True)
  (output): Linear(in_features=512, out_features=1, bias=True)
)
```

After exploring the hyper-parameters and various network sizes, the parameters in the config.py were chosen and the following results obtained. As can be seen success was achieved at episode 3577.

scores, avg_scores = training_loop_scores(env, brain_name, agent, config)

```
E:    100 | Average:     0.0150 | Best average:     0.0157 | Last score:    -0.0100
E:    200 | Average:     0.0130 | Best average:     0.0230 | Last score:     0.0900
E:    300 | Average:     0.0240 | Best average:     0.0250 | Last score:    -0.0100
E:    400 | Average:     0.0199 | Best average:     0.0280 | Last score:    -0.0100
E:    500 | Average:     0.0100 | Best average:     0.0280 | Last score:    -0.0100
E:    600 | Average:     0.0090 | Best average:     0.0280 | Last score:    -0.0100
E:    700 | Average:     0.0120 | Best average:     0.0280 | Last score:    -0.0100
E:    800 | Average:     0.0130 | Best average:     0.0280 | Last score:     0.2900
E:    900 | Average:     0.0140 | Best average:     0.0280 | Last score:    -0.0100
E:   1000 | Average:     0.0010 | Best average:     0.0280 | Last score:     0.0900
E:   1100 | Average:    -0.0010 | Best average:     0.0280 | Last score:    -0.0100
E:   1200 | Average:     0.0110 | Best average:     0.0280 | Last score:    -0.0100
E:   1300 | Average:     0.0168 | Best average:     0.0280 | Last score:    -0.0100
E:   1400 | Average:     0.0330 | Best average:     0.0340 | Last score:    -0.0100
E:   1500 | Average:     0.0480 | Best average:     0.0480 | Last score:     0.0900
E:   1600 | Average:     0.0550 | Best average:     0.0630 | Last score:    -0.0100
E:   1700 | Average:     0.0530 | Best average:     0.0650 | Last score:    -0.0100
E:   1800 | Average:     0.0590 | Best average:     0.0650 | Last score:     0.0900
E:   1900 | Average:     0.0790 | Best average:     0.0850 | Last score:     0.1900
E:   2000 | Average:     0.1100 | Best average:     0.1120 | Last score:     0.0900
E:   2100 | Average:     0.0590 | Best average:     0.1120 | Last score:     0.0900
E:   2200 | Average:     0.0880 | Best average:     0.1120 | Last score:     0.1900
E:   2300 | Average:     0.1110 | Best average:     0.1160 | Last score:     0.0900
E:   2400 | Average:     0.1100 | Best average:     0.1180 | Last score:     0.0900
E:   2500 | Average:     0.1270 | Best average:     0.1270 | Last score:     0.0900
E:   2600 | Average:     0.1500 | Best average:     0.1530 | Last score:     0.0900
E:   2700 | Average:     0.1430 | Best average:     0.1640 | Last score:     0.0900
E:   2800 | Average:     0.1680 | Best average:     0.1860 | Last score:     0.0900
E:   2900 | Average:     0.1860 | Best average:     0.1860 | Last score:     0.0900
E:   3000 | Average:     0.2270 | Best average:     0.2510 | Last score:     0.2900
E:   3100 | Average:     0.2380 | Best average:     0.2510 | Last score:     0.0900
E:   3200 | Average:     0.2238 | Best average:     0.2510 | Last score:     0.7900
E:   3300 | Average:     0.2389 | Best average:     0.2728 | Last score:     0.0900
E:   3400 | Average:     0.3099 | Best average:     0.3099 | Last score:     0.1900
E:   3500 | Average:     0.4059 | Best average:     0.4059 | Last score:     1.6900
E:   3577 | Average:     0.5039 | Best average:     0.5039 | Last score:     0.5900
```



Suggestions For Future Work

Options for future work in this area seem a little limited. Being a multi-agent and continuous action space leaves fewer options. Adapting PPO or TRPO to a multi-agent version may work very well, especially PPO. It would be interesting to explore adding multiple environments with multiple agents and seeing of this could speed up training and provided greated reduction in variation. This would be similar to adapting A2C methods to PPO or DDPG.