Continuous Control

Architecture:

Actor model input layer only focuses on the states of its corresponding agent. On the other hand, critic model input layer observes the states of both agents. Both models (Actor and Critic) have 24 nodes for input layer, two hidden layers and an output layer. For the first hidden layer, actor model has 250 nodes whereas critic model has 250 state nodes plus 2 action nodes. For the second hidden layer, both have 150 nodes. Finally, the output layer for actor model has 2 nodes ranging between -1 to 1 therefore hyperbolic tangent is used for the activation function whereas critic model has only 1 node without activation function.

self.actor_local = Actor(state_size, action_size, random_seed).to(device)

self.actor_target = Actor(state_size, action_size, random_seed).to(device)

self.critic_local = Critic(state_size, action_size , random_seed).to(device)

self.critic_target = Critic(state_size, action_size , random_seed).to(device)

Implementation:

As this problem is highly unstable and tends to fall into local maxima, this implementation uses actor-critic methods to mitigate the issues. Actor model has low bias and it also helps to exit local maxima with its high variance nature whereas critic model helps the models to learn faster with its low variance nature although it has high bias.

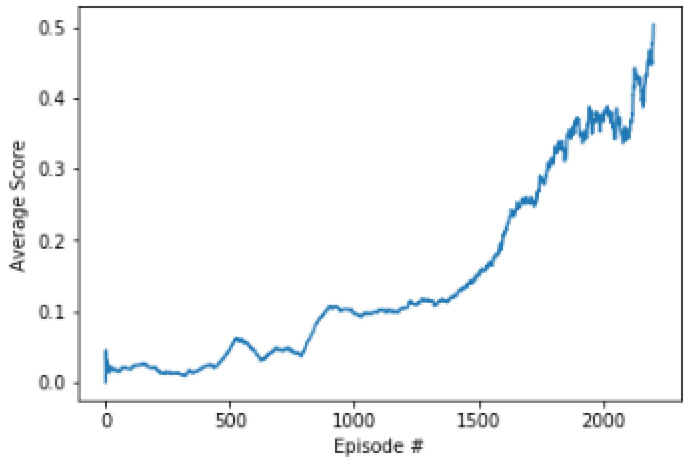
To reduce correlations, this implementation uses replay buffer shared by both agents. By sharing common replay buffer, we ensure that we explore all the environment. Note that agent still have their own actor and critic network to train.

It records state and action taken by both agents at every time step. This implementation also injects Ornstein-Uhlenbeck noise into the actions to encourage exploration around the mean value.

This implementation uses Deep Deterministic Policy Gradient (DDPG) algorithm to train multiple agents with decentralized-actor and centralized-critic method. DDPG features like continuous action-space is useful to significantly reduce the number of nodes required by the neural networks. Features like soft update is useful to perform a distinguishable targeted learning direction. Clipping is also implemented. Action clipping is useful to keep the actions within -1 and 1 due to noise injection.

I modified a standard DDPG algorithm to solve the environment in a multi-agent way. I implemented two DDPG agents that have its critic and actor-network. The agents have shared experience.

Plot of Rewards:



Hyperparameters:

It was fun to finetune the hyperparameters while working on the Tennis environment with MADDPG algorithm. I received the fastest learning with the Actor and Critic having 3 layers (fc1_units=250, fc2_units=150). The Actor uses relu activation on the first two and tanh on the final layer. The Critic uses relu on the first two layers only.

The following hyperparameters I used while training the agent:

Parameter	Value	Description
BUFFER_SIZE	int(1e5)	replay buffer size
BATCH_SIZE	2.50E+02	minibatch size
GAMMA	9.90E-01	discount factor
TAU	1.00E-03	for soft update of target parameters
LR_ACTOR	1.00E-04	learning rate
LR_CRITIC	1.00E-03	learning rate
n_episodes	2.00E+03	number of episodes
EXPLORE_TIMESTEPS	5.00E+03	timesteps until the agents just explore

Future ideas for improving the agent's performance:

- **Prioritized Experience Replay** could be implemented to further enhance the performance of the agent. Its suggested to use ‘**Sum Tree**’ data structure as it helps in a faster implementation of Prioritized Experience Replay.
- Tweak hyperparameter to achieve goal within less episode.