Udacity Deep Reinforcement Learning Nanodegree Project 3: Collaboration and Competition

Submitted by: Tamoghna Das

1. Network architecture and Learning Algorithm

A Deep Reinforcement Learning Actor-Critic Agent has been trained using Multi Agent DDPG algorithm to solve the Tennis Environment. The implementation is in Tennis.ipynb notebook. The Multi Agent DDPG Agent is coded in deep_rl\agent\DDPG_agent.py. The Actor and Critic Networks are coded inside deep_rl\network\network.py.

The codes are mainly taken from udacity ddpg-pendulam project and MADDPG project and adapted to suit the current project.

Deep Deterministic Policy Gradients (DDPG) algorithm is an off–Policy Actor-Critic algorithm that uses the concept of target networks. The input to the Actor network is the current state while the output is a real value or a vector representing an Action chosen from a continuous Action space.

The Multi Agent Actor Critic algorithm (MADDPG) used here to solve the Tennis environment is derived from the DDPG algorithm but adapted for multi agent cooperative – competitive environment. The Actor network is trained for both Tennis player agents individually. But during training, the Critic network has access to other agent's information also e.g. States observed and Action taken by the other tennis player agent. The Critic tries to estimate the optimal Q value function and learns by using Q learning approach using reward and next state from the Action estimated by Actor network. An Advantage value is calculated using the critic's estimated State value output. The Actor Network tries to estimate Optimal Policy (an Action value for a state) and learns by using calculated Advantage as a baseline.

During execution time, only the Actor network is present and hence each player agent estimates own Actions based on its own observation.

Actor Network Architecture:

- Batch Normalization
- Inputs: 24 units (state_size).
- hidden layers
    - fc1: fully connected. 24 x 128. (Leaky ReLU)
    - fc2: fully connected. 128 x 128. (Leaky ReLU)
- Output layer: fully connected layer. 2 tanh output units for 2 actions(action_size).

Critic Network Architecture:

- Batch Normalization
- Inputs: 24 units (state_size).
- hidden layers
    - fc1: fully connected. 24 x 128. (Leaky ReLU)
    - fc2: fully connected. 132(4 Actions, 2 from each agent, from Actor network added) x 128. (Leaky ReLU)
- Output layer: fully connected layer. 1 linear output unit.

Optimizer : Used for local Actor and local Critic. Adam optimizer (torch.optim.Adam), LR_Actor = 1e-4, LR_Critic = 1e-3.

Hyperparameters:

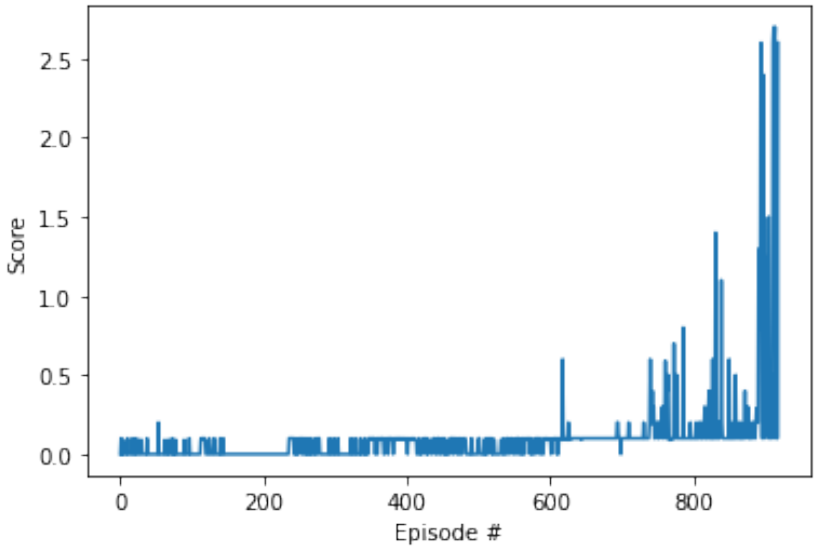
- Replay buffer size: 1e6
- Batch size: 256
- Discount factor (gamma): 0.99
- Tau (used for soft blending of target parameters): 0.001
- Learns every 2 steps

2. Results

The environment is solved after 817 episodes:

```
Environment solved in 817 episodes!  Average Score: 0.50
```

Plot of Scores vs Episode# is shown below:



3. **Ideas of future work**
    - To optimize the hyperparameters to improve performance.
    - To implement other Actor Critic algorithms e.g. A3C on the same environment and compare performances.
    - To apply DQN improvement techniques e.g. prioritized experience replay for performance improvement.