Project 3: Collaboration and Competition

In this report, I am going to present about the environment and the algorithms that I have used to solve collaboration and competition problem where the agents must bounce ball back and forth while not dropping or sending ball out of bounds.

Environment

We work with Unity ML Environment, Tennis, in this project.

In this environment, there are two agents which control rackets to bounce a ball over a net. The agent receives +0.1 reward if it manages to hit ball over the net without dropping or hitting out of bounds. For dropping or hitting out of bounds, it receives -0.01 .

The observation space consists of 24 variables representing position and velocity of the ball and racket. Each action is a vector with two numbers, corresponding to movement towards or away from the net, and jumping. The action vector should be a number between 1 and -1.

The environment is deemed solved if the agents get an average score of +0.5 over 100 consecutive episodes.

Given below are the characteristics of the agents and the environment.

- Unity brain name: TennisBrain
- Number of Visual Observations (per agent): 0
- Vector Observation space type: continuous
- Vector Observation space size (per agent): 8
- Number of stacked Vector Observation: 3
- Vector Action space type: continuous
- Vector Action space size (per agent): 2

Learning Algorithm

To solve this reinforment learning problem, I am using a Deep Deterministic Policy Gradients (DDPG) with modification to make it suitable for multiagent environment.

Deep Deterministic Policy Gradient (DDPG)

DDPG is an actor-critic algorithm that extends **DQN** to work in continuous spaces. Here, we use two deep neural networks, one as actor and the other as critic. Similar network architectures are used for both actor and critic. **ADAM** optimizer is used with **learning rates 0.0001** and **0.001** for actor and critic, respectively. And the **discount factor** used is **0.99**.

```
GAMMA = 0.99            # discount factor
LR_ACTOR = 1e-4         # learning rate of the actor
LR_CRITIC = 1e-3        # learning rate of the critic
```

Neural Network Architecture

Actor

State → BatchNorm → 128 → ReLU → 64 → ReLU → BatchNorm → action → tanh

Critic

State → BatchNorm → 128 → Relu → 64 → Relu → action

Pytorch Implementation

Actor

```
self.fc1 = nn.Linear(state_size, 128)
    self.bn1 = nn.BatchNorm1d(128)
    self.fc2 = nn.Linear(128, 64)
    self.bn2 = nn.BatchNorm1d(64)
    self.fc3 = nn.Linear(64, 2)
    self.bn3 = nn.BatchNorm1d(2)

    ...

    x = self.fc1(state)
    x = F.relu(x)
    x = self.bn1(x)
    x = self.fc2(x)
    x = F.relu(x)
    x = self.bn2(x)
    x = self.fc3(x)
    x = self.bn3(x)
    x = F.tanh(x)
```

Critic

```
self.bn0 = nn.BatchNorm1d(state_size)
self.fcs1 = nn.Linear(state_size, fcs1_units)
self.bn1 = nn.BatchNorm1d(fcs1_units)
self.fc2 = nn.Linear(fcs1_units+action_size, fc2_units)
self.bn2 = nn.BatchNorm1d(fc2_units)
self.fc3 = nn.Linear(fc2_units, 1)

...

x = self.bn0(state)
x = self.fcs1(x)
x = F.relu(x)
x = torch.cat((x, action), dim=1)
x = self.fc2(x)
x = F.relu(x)
x = self.fc3(x)
```

Experience Replay

We store the last one million experience tuples (S,A,R,S') into a data container called **Replay Buffer** from which we sample **a mini batch of 1024** experiences. This batch ensures that the experiences are independent and stable enough to train the network.

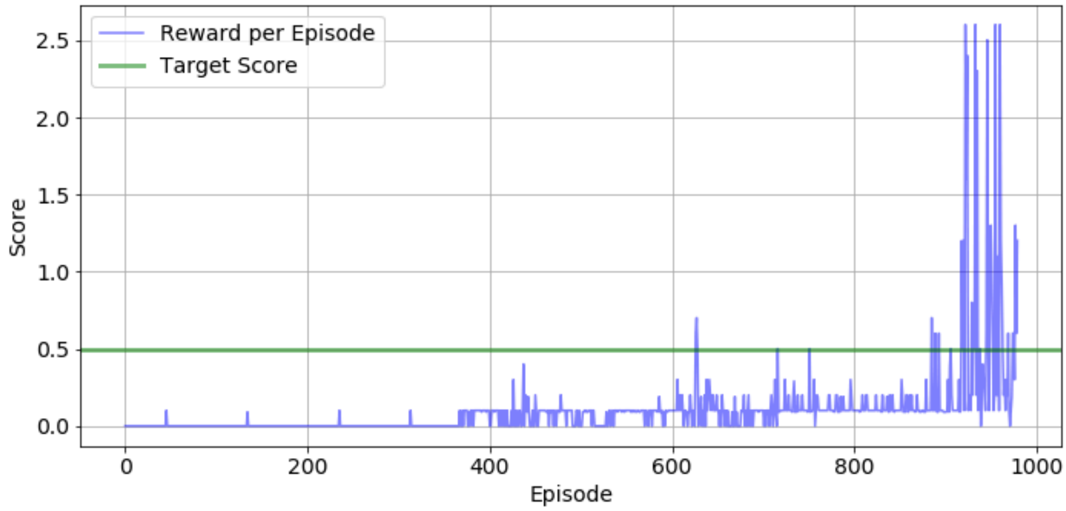
```
BUFFER_SIZE = int(1e6)  # replay buffer size
    BATCH_SIZE = 1024        # minibatch size
```

Soft Target Updates
In order to calculate the target values for both actor and critic networks, we use **Soft Target Update** strategy.

```
TAU = 1e-3              # for soft update of target parameters
```

> In order for the agents to explore the entire environment, replay memory buffer is shared between the agents. Yet, | | > each agent has its own actor and critic networks to train.

Plot of Rewards
After tuning the parameters and tweaking the network by changing the hidden layers size, I could solve the problem in **980 episodes**. The plot below shows the rewards per episode and the target.



Result

Result can be seen in tennis_result.gif

Ideas for Future Work

- After training, when I checked the performance, it seems that the agents are imitating each other. Reason could be the symmetry of the environment. Multi-Agent DDPG would be apt for environment like Soccer. So, I am planning to implement and see the performance of MADDPG in Soccer environment.