Article I. Environment

Multiagent environment with 2 agents used to train collaborative and competitive agents to play a game of Table Tennis using Multi Agent DDPG algorithm.

Article II. Algorithm

Section 2.01 Multi Agent Deep Deterministic Policy Gradient Algorithm

(a) Description

Deep Deterministic Policy Gradient Algorithm is a type of Actor-Critic Algorithm for RL . The Deep Q Learning algorithms used to calculate Q Value for each state-action pair to find the optimal policy. In order to estimate policy directly without introducing intermediate step of estimating Q values for each action , a Policy Based Learning was introduced. In this algorithm policy is directly estimated with the help of parameter update as following equation.

$$\theta_{t+1} = \theta_t + \alpha\big(G_t \cdot \nabla_\theta \ln \pi(a_t|s_t, \theta_t)\big)$$

G_t is the total accumulated reward from step *t* till the end of the episode. Proof of this update equation can be found in *"Reinforcement Learning, An Introduction" by Sutton & Barto (2nd ed.), pages 325–327*. Unlike Q-Learning, the Policy Gradient algorithm is an *on-policy* algorithm, which means it learns only using state-action transitions made by the current active policy.

With Policy based methods, we now have an Agent that learns a policy without the need to learn the actual value of each action based on total reward it achieves at the end of episode or time period. When we consider overall reward, if the agent fails, it consider al the actions it has taken in an entire period are bad , where as there may be some good moves considering all of them bad , is not a good idea. Therefore we need a method that on one hand, learns a policy in a similar way to the Gradient Policy method, though on the other hand understand the importance of state- and action-specific knowledge, like in the Q-Learning method. Actor Critic Methods combine of both these methods, to get the best of both worlds.

Actor learns the policy which should be acted by , and the Critic learns the Q-Value of each state and action We then change our update rule of our θ to be:

$$\theta_{t+1} = \theta_t + \alpha\big(Q(s_t, a_t) \cdot \nabla_\theta \ln \pi(a_t|s_t, \theta_t)\big)$$

We have replaced the total reward *G* with the Q-Value, but the Q-Value is now also a learned parameter. This is simple Actor-Critic Algorithm.

The DDPG (Deep Deterministic Policy Gradient) algorithm was designed to solve problem of continuous action-space the training phase of an Actor-Critic model is very noisy, as it learns based on its own predictions. To tackle this, DDPG uses idea from DQN, it uses an Experience Replay memory, which makes it an off-policy model. It also uses the same noise-reduction method of the Double DQN model that is, it uses two copies of both the Actor and the Critic, one copy is trained and the second is updated slowly with soft update method.

In this project a version of Multi Agent DDPG is used, where each agent is individual DDGP agent , with individual actor and critic , they collaboratively learn from each other by getting the combined state of both agents and sharing a common replay buffer, on other hand they competitively learn by updating their own copy of actor and critic with individual rewards they have got from environment after each step.

(b) Model Architecture

(i) Actor Model Architecture

fc1_units = 400 , fc2_units = 300

```
super(Actor, self).__init__()
self.seed = torch.manual_seed(seed)
self.fc1 = nn.Linear(state_size, fc1_units)
self.bn1 = nn.BatchNorm1d(fc1_units)
self.fc2 = nn.Linear(fc1_units, fc2_units)
self.fc3 = nn.Linear(fc2_units, action_size)
self.reset_parameters()
```

(ii) Critic Model Architecture

fcs1_units= 400 , fc2_units=300

```
super(Critic, self).__init__()
self.seed = torch.manual_seed(seed)
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.fc3 = nn.Linear(fc2_units, 1)
self.reset_parameters()
```

(c) Hyper Parameters

- Replay Buffer Size is set to $1e^{6}$
- Batch Size is 256
- Discount Factor (gamma) is 0.99
- Soft Update of Target Parameters (tau) is 1e-3
- Learning Rate of actor is 1e-3
- Learning Rate of critic 1e-3
- L2 Weight Decay is 0
- Update Network every 1 timestamp
- Update Network 1 times each time.
- Epsilon for noise added to action is set to 1.0
- Epsilon decay is 1^{e-6}

(d) Result

The environment was first solved in 12913 episodes to achieve average score over 100 episodes of 0.9197

```
Environment solved in 12912 episodes!	Average Score: 0.8986
Environment solved in 12913 episodes!	Average Score: 0.9197
```

Figure 1. from last lines of *nohup.out* file.

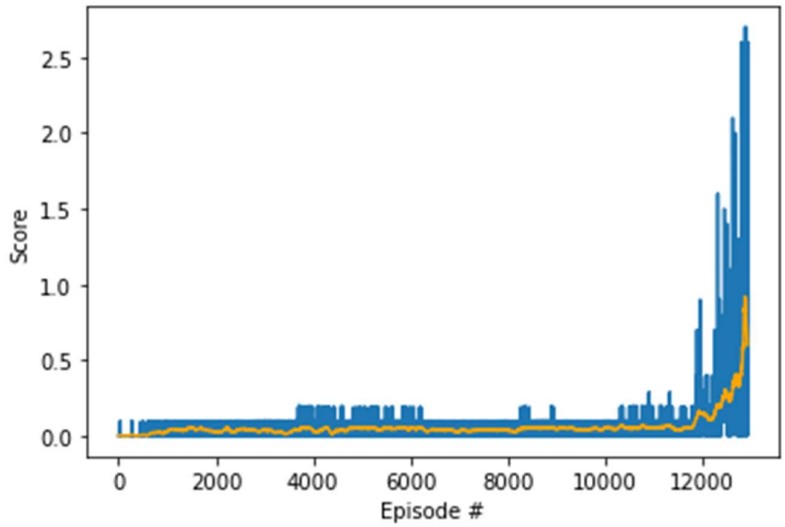

Figure 2 . Graph Plotted in Evaluate_Tennis.ipynb from saved scores.

Article III. Conclusion

The best average reward of 0.9197 over 100 episodes for 2 agents was achieved in 12913 episodes. It took around 7 hrs to train this model completely on p2.x large AWS machine , with K-80 GPU.

Article IV. Future Ideas

1. Agent was trained with a very nice score, but number of epochs used to train it were quite huge. Improvement can be made in model structure or parameters to reduce the convergence time.
2. In order to improve agents performance further, prioritized experience replay can be used instead of random sampling of experiences. The priority can be defined either on basis of loss or using some other priority technique like to replace the new experience with the oldest experience that came in. Definitely priority a/c to loss occurred would be more meaningful.
3. An MADDPG version of shared critic network amongst agents can be used.