Article I. Environment

Multiagent environment with 20 agents is used to train an AI Agent to maximize the reward by using DDPG algorithm

Article II. Algorithm

Section 2.01 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.

(b) Model Architecture

(i) Actor Model Architecture

```
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

```
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

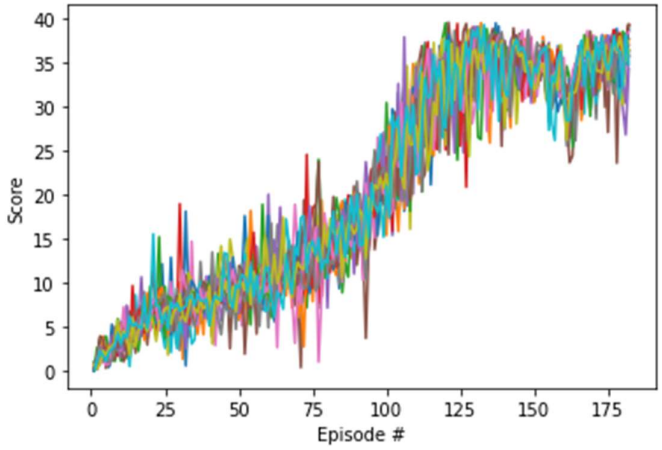
- Replay Buffer Size is set to $1e^6$
- Batch Size is 256
- Discount Factor (gamma) is 0.99
- Soft Update of Target Parameters (tau) is 1e-3
- Learning Rate of actor is 1e-3
- Learning Rate of critic 1e-3
- L2 Weight Decay is 0
- Update Network every 20 timestamps
- Update Network 10 times each time.
- Epsilon for noise added to action is set to 1.0
- Epsilon decay is $^{1e\text{-}6}$

(d) Result

The environment was first solved in 182 episodes to achieve average score of 30.1

```
Episode 181     Average Score: 29.88
Episode 182      Timestamp: 0
Episode 182      Timestamp: 100
Episode 182      Timestamp: 200
Episode 182      Timestamp: 300
Episode 182      Timestamp: 400
Episode 182      Timestamp: 500
Episode 182      Timestamp: 600
Episode 182      Timestamp: 700
Episode 182      Timestamp: 800
Episode 182      Timestamp: 900
Episode 182     Average Score: 30.10

Environment solved in 82 episodes!      Average Score: 30.10
```



Article III. Conclusion

The best average reward over 100 episodes over 20 agents was achieved in 182 episodes. It took around 30 hrs to train this model completely on local machine with gpu NVIDIA GeForce Mx250 2GB.

Article IV. Future Ideas

1. 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.
2. Algorithms like PPO, A3C, and D4PG can be implemented and evaluated to improve agents performance.
3. Agent can be trained for longer time to get score much greater than 30.