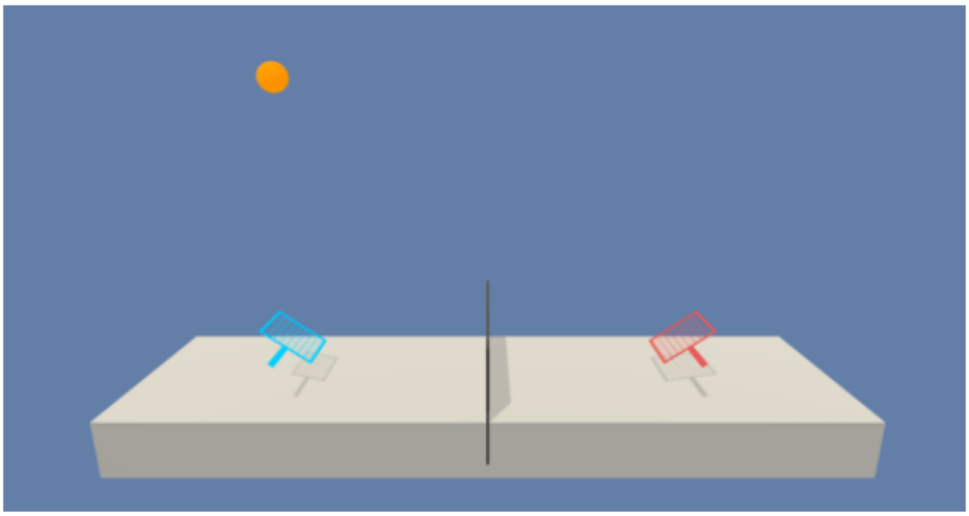
Report – Multi Agent Deep Deterministic Policy Gradient applied for Project 3 – Collaboration and Competition

Objective : To make the two agents to control the rackets on both the ends, so that the ball is made to bounce over the net and it is made sure that the goal is to keep the ball in play.



Unity ML-Agents Tennis Environment

Source : Udacity 3rd Project – The Environment-Introduction

Introduction : The state space has 8 variables corresponding to position, velocity of the ball and the racket. Each agents gets its own observation from the environment. There are two continuous actions are available, one being the movement towards or away from the net and another one being the jumping action.

In this project, a reward of +0.1 is given if the agent hits the ball over the net and a reward of -0.01 is received if the agent lets the ball hit the ground or hits the ball out of the bounds.

The environment is considered to be solved when the average score value is +0.5 i.e. over 100 consecutive episodes, after taking the maximum over both agents.

Algorithm : The algorithm of MADDPG (Multi-Agent Deep Determnistic Policy Gradient) is referenced from MADDPG-Lab of Lesson 2 [1] and Deep Deterministic Policy Gradient (DDPG) program is referenced from the DDPG-Pendulum [2] for applying the actor-critic methods. The 'MADDPG_Agent' class used in this project encapsulates the 'ddpg_agent' class.

Explanation of DDPG :

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Source : Algorithm [3]

The DDPG consists of an Actor and Critic both of which are Neural Networks. This algorithm comes under the Policy based approaches. The objective of the Actor is to approximate the policy in a deterministic way. The input of the Actor network is the state and output is the action taken in that state.

The objective of the Critic is to estimate the value of different state-action pairs i.e. Q-value of the optimal action value function (Based on Bellman's Equation). The weights of the actor and critic network are updated using the soft update strategy.

The learning process takes place by using mini-batches and a replay buffer is utilized here. It is finite sized cache R. The transitions were sampled from the environment according to the exploration policy [3]. Once the buffer gets full, then the oldest samples are discarded [3]. For this algorithm the replay buffer can be large which is beneficial for the learning process.

Explanation of MADDPG :

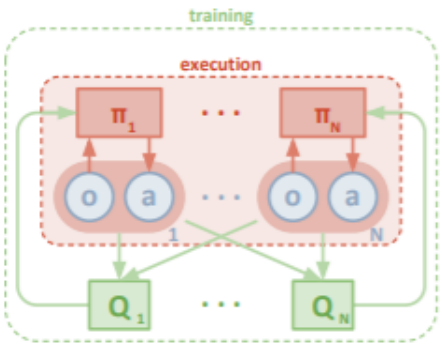


Figure: Overview of Multi-agent decentralized actor, centralized critic approach [4]

MADDPG algorithm, outstretches the existing DDPG into a multi-agent policy gradient algorithm where multiple decentralized agents learn a centralized critic based on the observations and actions of all agents present in the task [4].

The idea behind using a centralized critic (Q-value for the centralized action-value function) is to jointly use the information for updating its parameters. Though the training is carried out in a centralized manner , the execution is decentralized, where only the local actors are used at execution phase .

This algorithm is highly applicable to situations where there is cooperative interaction or a competitive or mixed interaction among the multiple agents. However the most important constraint being :The learned policies can only use their own observations at the execution time [4].

Model Architecture Actor and Critic:

The actor neural network consists of 3 fully connected layers. The number of nodes in the first hidden layer is set as 256 and the second hidden layer consists 256 nodes. ReLU activations are used in the network for including non-linearity. The output layer is followed by the *tanh* activation function.

The critic neural network consists of 3 fully connected layers with two hidden layers with 256 hidden neurons in the first hidden layer followed by 256 neurons in the second hidden layer. ReLU activation functions are used in the network for including non-linearity. Further the states and actions are concatenated as the inputs of the first layer, where the state_size is 24 and action_size is 2.

Hyperparameters :

BUFFER_SIZE = int(1e6)

BATCH_SIZE = 256

GAMMA = 0.99

TAU = 1e-2

LR_ACTOR = 1e-4

LR_CRITIC = 1e-4

WEIGHT_DECAY = 0

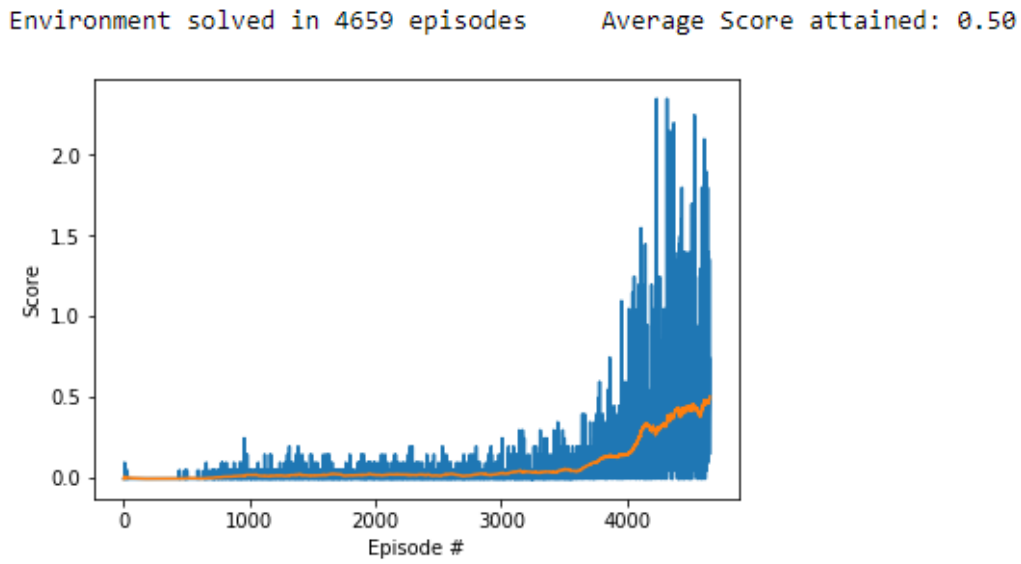
LEARNING_INTERVAL = 2

The value of the discount factor is set to be 0.99. The TAU for the soft update of target parameters is set as 1e-2. The value of the learning rates for the actor and critic network were set to be the same value of 1e-4 and a weight decay value of 0. The

hyperparameter LEARNING_INTERVAL is set to 2. It describes about the frequency of the weight update.

For these set of the hyperparameters the environment was solved in 4659 episodes with an average score of 0.50250.

Plot of Rewards per episode :



The agent attains an average reward of 0.5 after crossing 4659 episodes.

Future scope for improving agent's performance :

Better fine-tuning of Hyperparameters may help in achieving reduced training time or early convergence of the result. Using prioritized experience replay may lead to better results.

References

[1] MADDPG-Lab of Lesson 2 – Introduction to Multi-Agent RL of Udacity's Deep Reinforcement Learning course

[2] Udacity- Deep Reinforcement Learning- Github- https://github.com/udacity/deep-reinforcement-learning/tree/master/ddpg-pendulum

[3] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra: Continuous control with deep reinforcement learning. ICLR (Poster) 2016

[4] Lowe, Ryan et al. "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments." NIPS (2017).