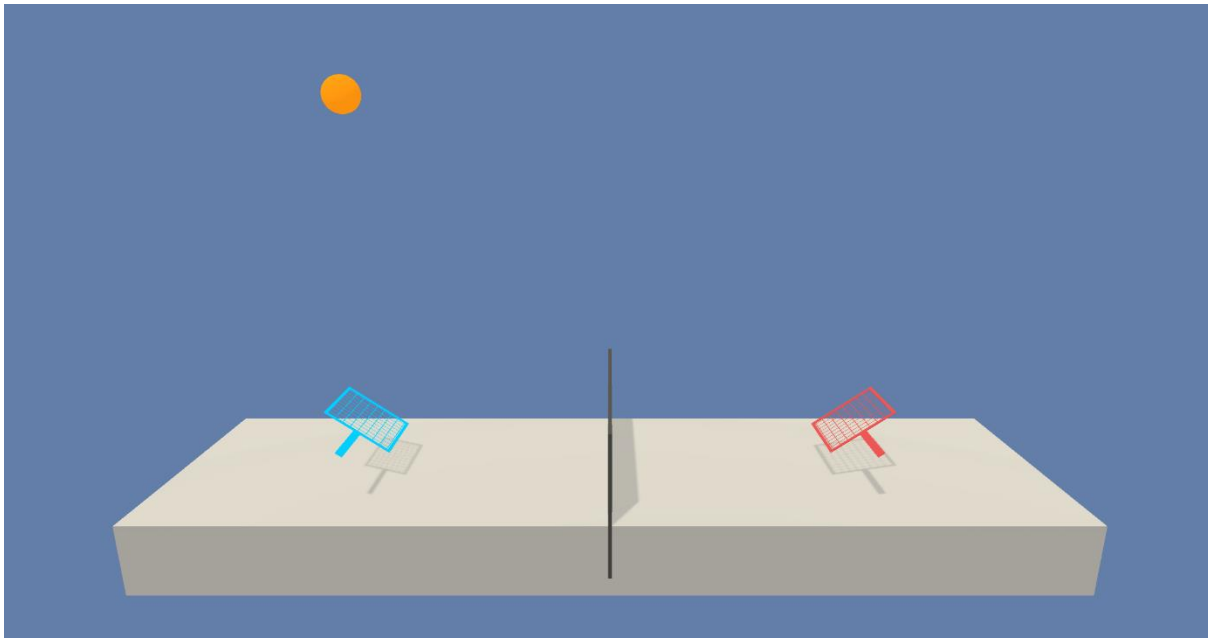
Deep Reinforcement Learning Nanodegree

Project 3: Collaboration and Competition

Pradeep Dayanandam

Introduction:

In this project, will work with the Tennis environment.



Unity ML-Agents Tennis Environment

In this environment, two agents control rackets to bounce a ball over a net. If an agent hits the ball over the net, it receives a reward of +0.1. If an agent lets a ball hit the ground or hits the ball out of bounds, it receives a reward of -0.01. Thus, the goal of each agent is to keep the ball in play.

The observation space consists of 8 variables corresponding to the position and velocity of the ball and racket. Each agent receives its own, local observation. Two continuous actions are available, corresponding to movement toward (or away from) the net, and jumping.

The task is episodic, and in order to solve the environment, your agents must get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents). Specifically,

After each episode, we add up the rewards that each agent received (without discounting), to get a score for each agent. This yields 2 (potentially different) scores. We then take the maximum of these 2 scores.

This yields a single score for each episode.

The environment is considered solved, when the average (over 100 episodes) of those scores is at least +0.5

Algorithm Used:

The Algorithm used to solve this problem is **Deep Deterministic Policy Gradient** (DDPG). DDPG was proposed in Continuous control with deep reinforcement learning (Lillicrap et al, 2015). DDPG is an off-policy algorithm and can only be used only for continuous action spaces. It simultaneously learns a function and a policy. It's an actor critic algorithm and uses two neural networks. The algorithm also applies experience replay and target networks to stabilize the training. To boost exploration, the authors of the DDPG paper have used Ornstein-Uhlenbeck Process to add noise to the action output. But for this training run, we would only be adding Gaussian noise.

Pseudocode:

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

Taken from Continuous control with deep reinforcement learning (Lillicrap et al, 2015)

Model Architecture:

The Actor network consist of three fully connected layer with batchnormalization applied at the first layer. The network maps states to actions. It uses ReLU as activation function except the last layer where it use tanh.

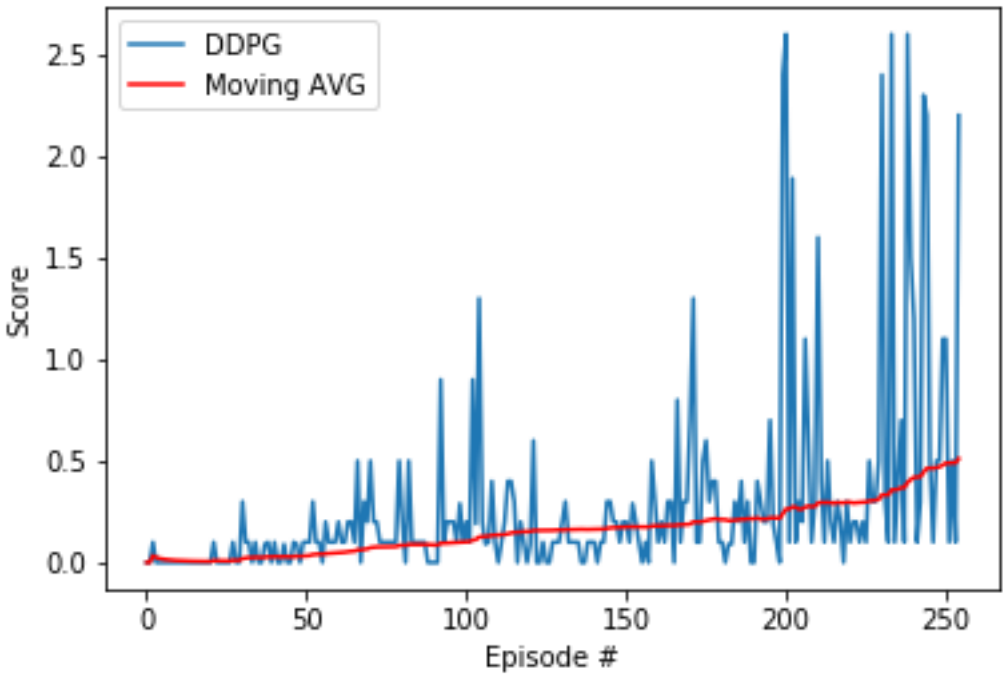
The critic network also consist of three fully connected layer with batchnormalization applied at the first layer. The network maps maps (state,action) pairs to Q-values. It uses ReLU as activation function in the first two layers and no activation function for the last layer.

Hyperparameters

- fc1_units=400 # Number of nodes in the first hidden layer
- fc2_units=300 # Number of nodes in the second hidden layer
- BUFFER_SIZE = int(1e6) # replay buffer size
- BATCH_SIZE = 256 # minibatch size
- GAMMA = 0.99 # discount factor
- TAU = 2e-3 # for soft update of target parameters
- LR_ACTOR = 1e-3 # learning rate of the actor
- LR_CRITIC = 1e-3 # learning rate of the critic
- WEIGHT_DECAY = 0 # L2 weight decay
- LEARN_EVERY = 1 # learning timestep interval
- LEARN_NUM = 10 # number of learning passes
- GRAD_CLIPPING = 1.0 # gradient clipping
- EPSILON = 1.0 # for epsilon in the noise process (act step)
- EPSILON_DECAY = 1e-6 # epsilon decay rate

Results:

The environment is considered to be solved when our agents get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents).



Environment solved in 255 episodes!

Average score (when the env was first solved) : 0.511

Future ideas to improve the agent's performance:

More experiments can be done to increase the performance of agent

- DDPG can be improved by using prioritized experience replay.
- Fine-tuning of hyperparameters can also lead to better results and faster training time
- Applying Multi Agent Actor Critic for Mixed Cooperative Competitive environments