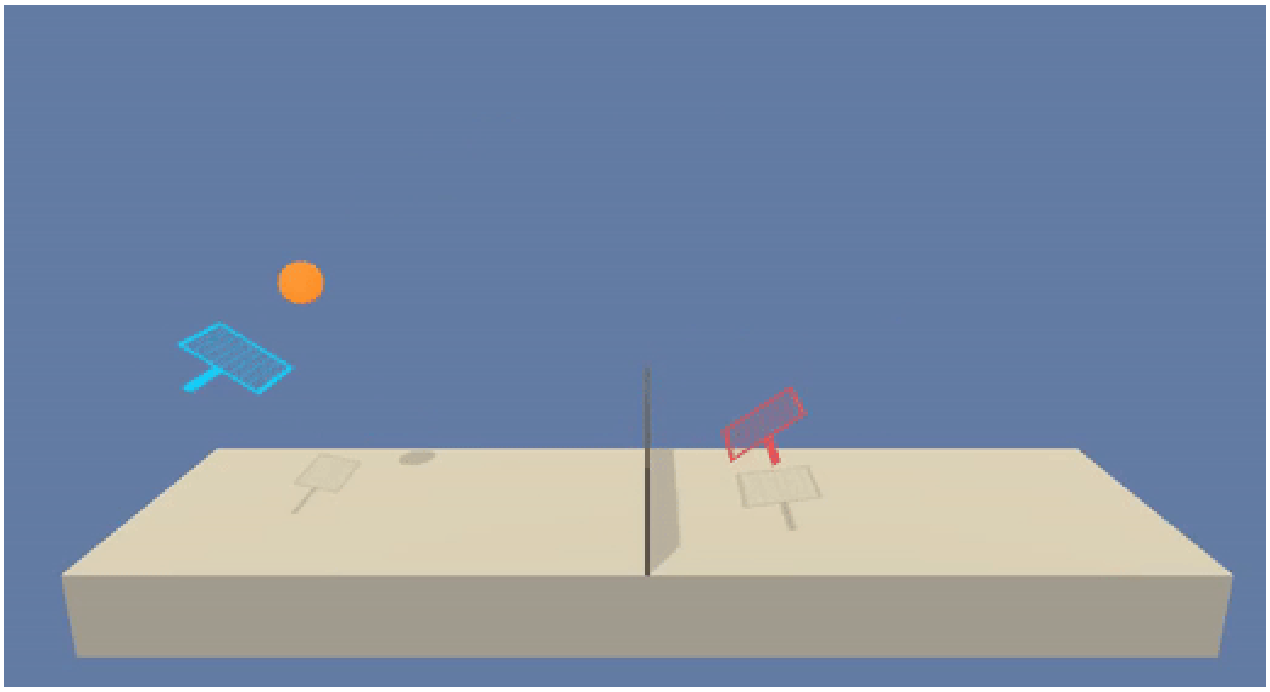
Collaboration and Competition



Problem Statement

Using reinforcement learning techniques to train an agent to keep the ball in play.

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. This is an episodic task and the goal of the agent is to get an average score of +0.5 over 100 consecutive episodes, after taking the maximum over both agents.

Implementation

The learning algorithm takes into account the presence of two agents. In particular, the goal is to train the agents such that they get an average score of +0.5 (over 100 consecutive episodes,).

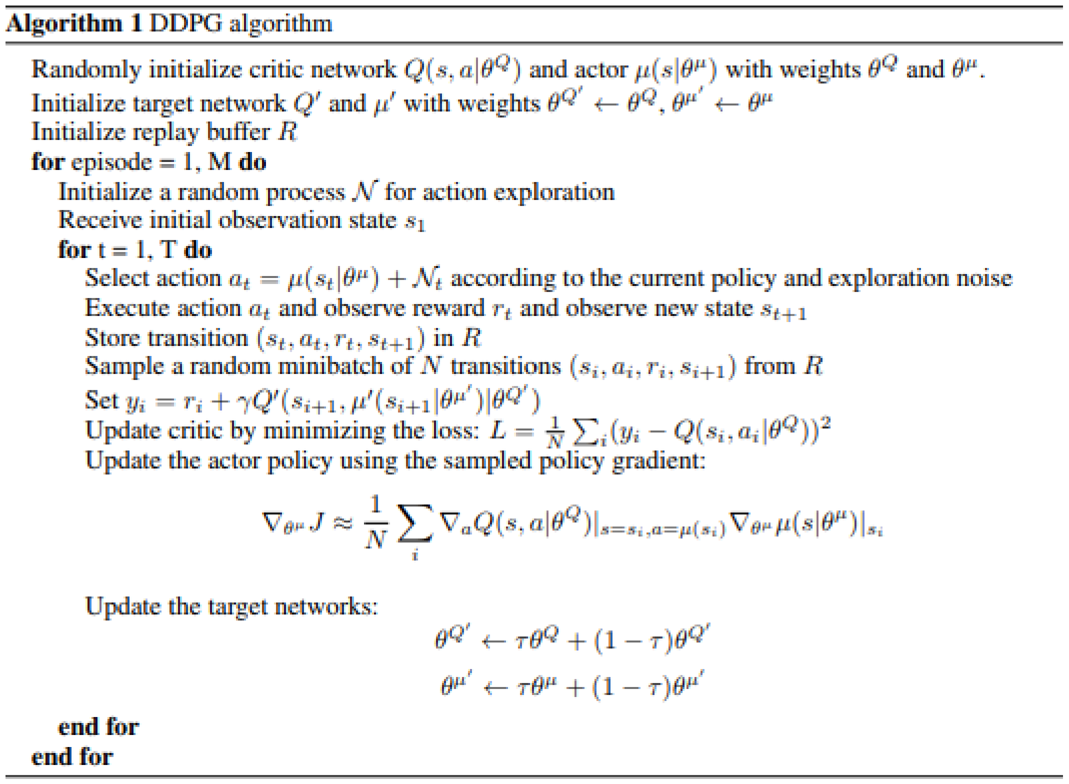
Specifically, after each episode, we add up the rewards that each agent received (without discounting), to get a score for each agent.

DDPG Algorithm

DDPG uses four neural networks: **a Q network**, **a deterministic policy network**, **a target Q network**, and **a target policy network**.

The **Q network** and **policy network** is very much like simple Advantage Actor-Critic, but in DDPG, the Actor directly maps states to actions (the output of the network directly the output) instead of outputting the probability distribution across a discrete action space

Following depicts the pseudocode of the same.



```
Algorithm 1 DDPG algorithm
  Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
  Initialize target network $Q'$ and $\mu'$ 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
```

Figure 1:DDPG Algorithm

The Algorithm used to train our agents is Multi-DDPG Algorithm. Multi Agent DDPG uses a 'decentralised actor, centralised critic training approach'. In this approach, all agents have access to all other agent's state observation and actions during critic training but tries to predict its action with only its own state observation during execution. This helps in easing the training as the environment becomes stationary for each agent. Below diagram explains the process which was present in the published article.

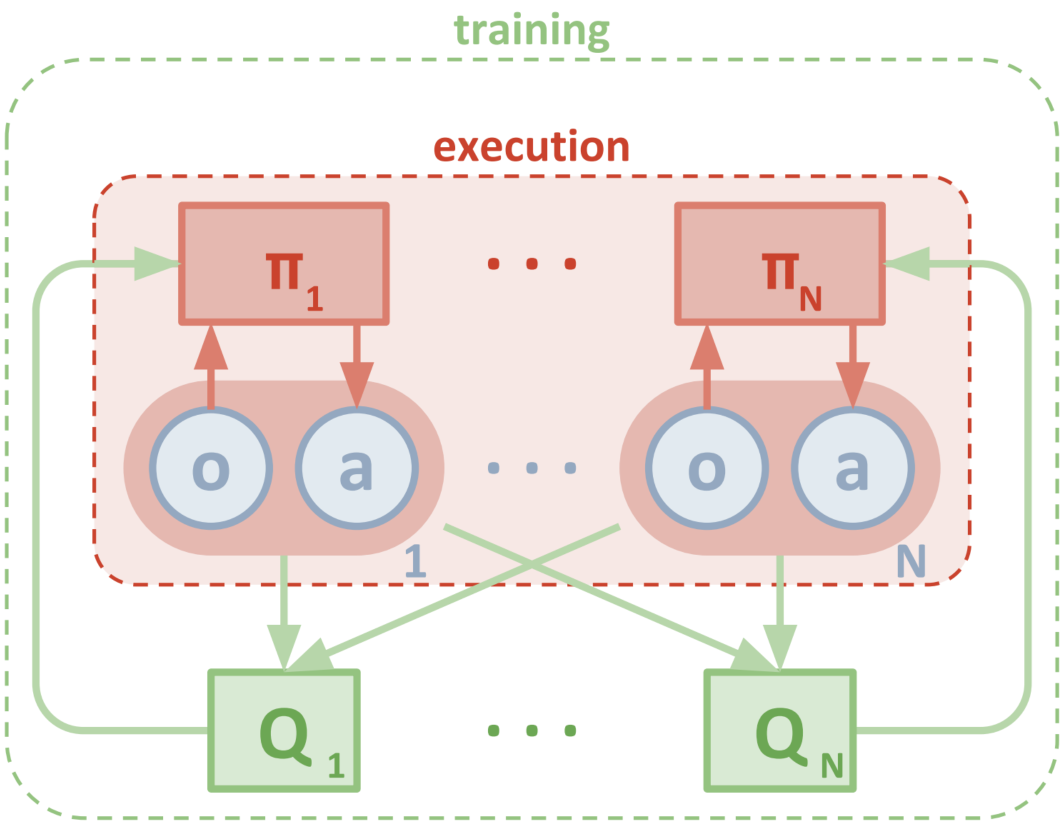
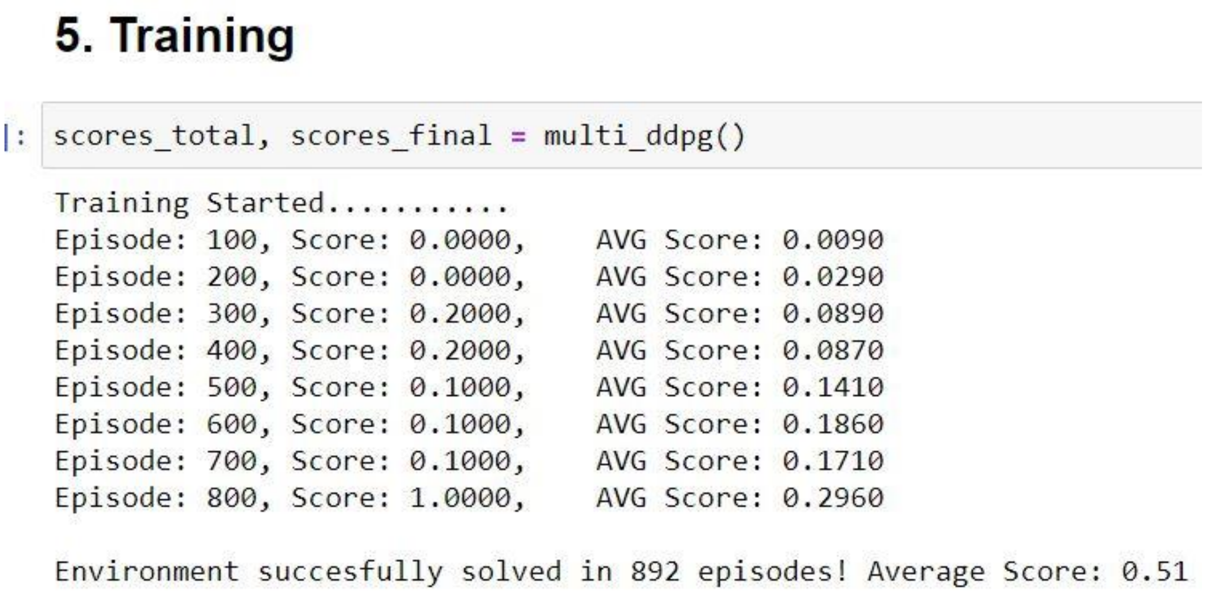


Figure 2:Overview of our multi-agent decentralized actor, centralized critic approach (from openAI paper)

All the agents' states and action are concatenated which serves as input to critic Neural Network. The output is Q-Value for that state. This Q-value is used as baseline to train actor, which gets only particular agent state as input and outputs its action values.

Results

The agents were able to obtain an average reward of value greater than +0.5 within 892 episodes of learning.



5. Training

```
]: scores_total, scores_final = multi_ddpg()

Training Started...........
Episode: 100, Score: 0.0000,    AVG Score: 0.0090
Episode: 200, Score: 0.0000,    AVG Score: 0.0290
Episode: 300, Score: 0.2000,    AVG Score: 0.0890
Episode: 400, Score: 0.2000,    AVG Score: 0.0870
Episode: 500, Score: 0.1000,    AVG Score: 0.1410
Episode: 600, Score: 0.1000,    AVG Score: 0.1860
Episode: 700, Score: 0.1000,    AVG Score: 0.1710
Episode: 800, Score: 1.0000,    AVG Score: 0.2960

Environment succesfully solved in 892 episodes! Average Score: 0.51
```

Figure 3: Training using Multi agent DDPG

Plots

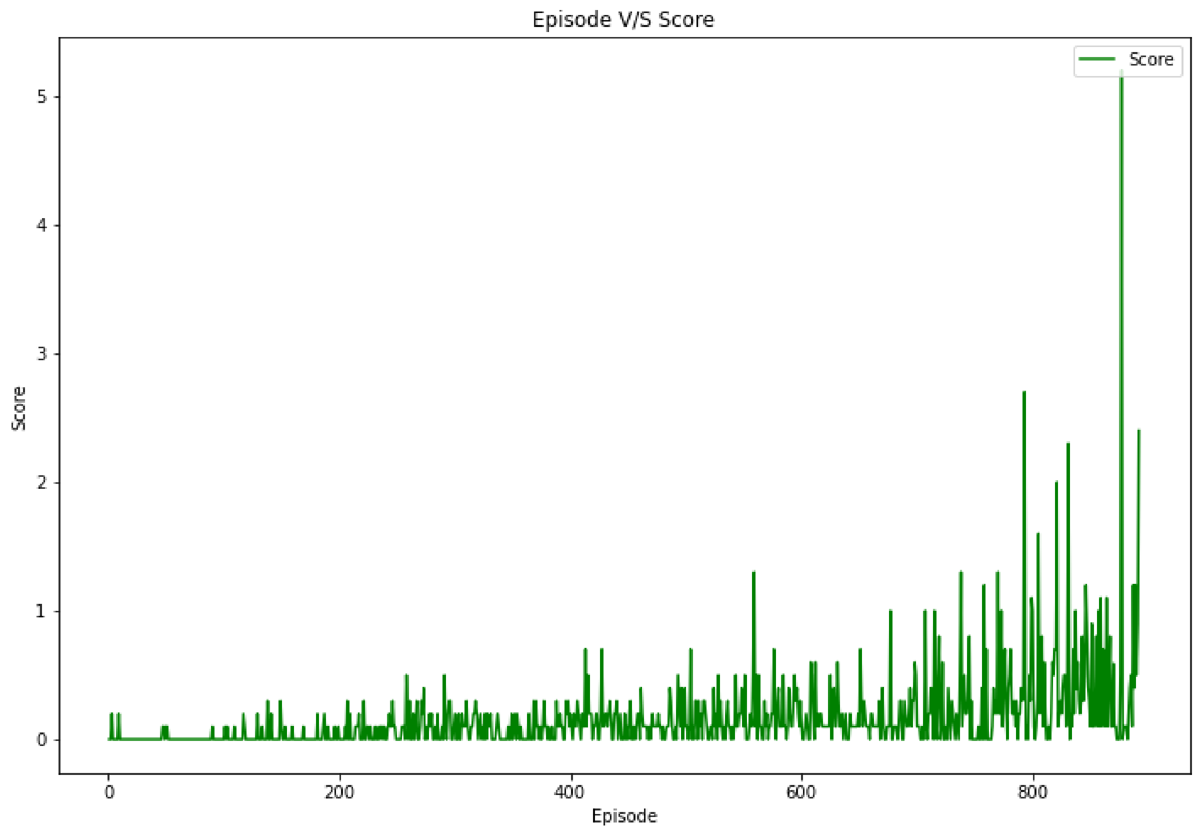


Figure 4: Episode # V/S Score

It can be observed that after about 892 episodes of learning, the agents are slowly able to accumulate a reward of +0.5. And a training up to 892 episodes finally solved the environment.

Conclusion and Future Work

- The agents are able to accumulate an average reward of +0.5 over 892 episodes of learning.
- We can experiment with our model architecture to further observe if there can be any improvements.
- There are several more algorithms like D4PG, PPO etc, we can use these algorithms and performs comparisons to see if it improves the results any further.
- Hyperparameter tuning seemed to be most effective for me during the implementation of this project. Experimenting with the parameters like epsilon, epsilon decay, updating factor etc. can be done further to observe if the results improve.

References

- https://arxiv.org/abs/1509.02971
- https://towardsdatascience.com/deep-deterministic-policy-gradients-explained-2d94655a9b7b
- https://medium.com/brillio-data-science/improving-openai-multi-agent-actor-critic-rl-algorithm-27719f3cafd4