# Udacity's Deep Reinforcement Learning Nanodegree Project 3 Report: Unity's Tennis Environment solved by DDPG

December 25, 2018

## 1 Introduction

This project involves solving the environment of a tennis match simulated in the Unity MLAgents environment. This environment has a continuous action space. Thus, we will solve this environment with a Deep Deterministic Policy Gradient (DDPG) [1, 2] method.

The environment includes two agents that control rackets that play tennis with each other. An agent receives a reward of 0.1 if it hits the ball over the net. It receives a reward of -0.01 if the ball hits the ground or is out of bounds. The goal is to keep the ball in play.

The observation space for each agent is 8 continuous variables representing the position and velocity of the ball and racket. There are 2 continuous action variables — moving towards or away from the next and jumping.

The solved criteria is an average score of +0.5 over the last 100 consecutive episodes after taking the maximum over both agents.

## 2 Learning Algorithm

The Deep Deterministic Policy Gradient (DDPG) algorithm [1, 2] is implemented in this project:

Initialize replay memory $D$ with capacity $N$
Initialize critic network $\hat{q}$ with random weights $w^q$
Initialize actor network $\mu$ with weights $w^\mu$
Initialize target critic weights $w^{q-} \leftarrow w^q$
Initialize target actor weights $w^{\mu-} \leftarrow w^\mu$
**for** the episode $e \leftarrow 1$ to $M$ **do**
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial input state $S$
    **for** time step $t \leftarrow 1$ to $T$ **do**
        Choose action $A = \mu(S|w^\mu) + \mathcal{N}$

Take action $A$, observe reward $R$, and next input frame $S'$
Store experience $(S, A, R, S')$ in replay memory $D$
$S \leftarrow S'$
Obtain minibatch of tuples $(s_j, a_j, r_j, s_{j+1})$ from $D$ of size $K$.
Set target $y_j = r_j + \gamma \hat{q}(s_{j+1}, \mu(S', w^{\mu-}), w^{q-})$
Update $w^q$: $\Delta w^q = -\alpha \frac{1}{N} \sum_j (y_j - \hat{q}(s_j, a_j, w^q)) \nabla_{w^q} \hat{q}(s_j, a_j, w^q)$
Update $w^\mu$ with policy gradient:

$$\nabla_{w^\mu} J \approx \frac{1}{N} \sum_i \nabla_a \hat{q}(s_i, a\mu(s_i)) \nabla_{w^\mu} \mu(s_i | w^\mu)$$

Soft update $w^{q-}$: $w^{q-} \leftarrow (1-\tau) w^{q-} + \tau w^q$
Soft update $w^{\mu-}$: $w^{\mu-} \leftarrow (1-\tau) w^{\mu-} + \tau w^\mu$

The hyperparameters were set to the following: buffer size $N = 10^6$, discount factor $\gamma = 0.99$, batch size $K = 256$, soft update parameter $\tau = 10^{-1}$, learning rate $\alpha = 1 \times 10^{-4}$ for the actor and learning rate $\alpha = 3 \times 10^{-4}$ for the critic.

The critic network used to represent $\mu$ is a fully connected neural network with an input layer with 8 input nodes. The second layer is fully connected to the first input layer with 256 nodes and is concatenated with a series of new input nodes for the action. The third layer has 128 nodes. The final layer has a single node resenting the action value for the state and action.

The actor network used to represent $\hat{q}$ is a fully connected neural network with an input layer of 8 nodes, a hidden layer of 256 nodes, and the final output layer with 2 nodes (one for each action value). The activation used for the hidden layer was a ReLU function and a tanh function for the final layer.

# 3 Results and Plots of Rewards

Figure 1 shows how the rewards vary over time. The solved criteria was fulfilled around the 800 episode.

# 4 Ideas for Future Work

I am interested to see how MADDPG compares to the rate of learning of DDPG. I would expect it to do better since it is specifically designed for the multi-agent setting.

# References

[1] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, sep 2015.

[2] Udacity. Deep Reinforcement Learning Nanodegree Course Material, 2018.

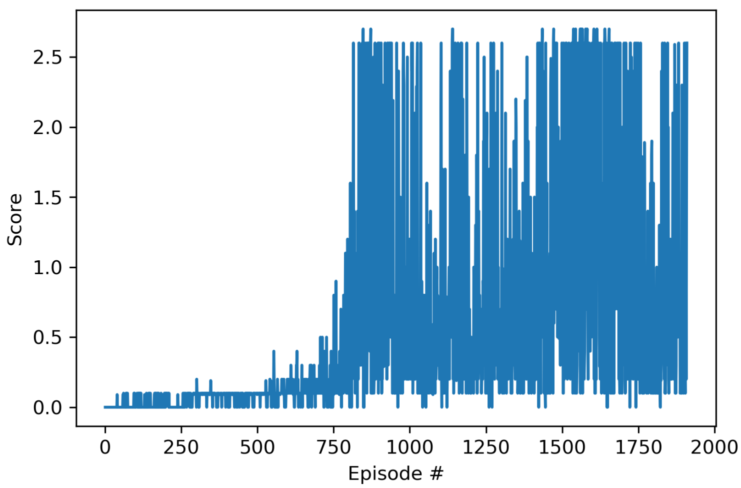


Figure 1: Rewards over time.