# Report – Collaboration and Competition

## Introduction

The project uses a multi-agent version of the Deep Deterministic Policy Gradient (DDPG) algorithm to train agents to pass a tennis ball over the net in Udacity's version of the Unity ML Tennis environment. The players are rewarded for hitting the ball over the net. The goal is to achieve an average score of at least +0.5 (roughly corresponds to 10 successful passes) over 100 episodes.

The implementation is based on the ddpg-pendulum example in Udacity's Deep Reinforcement Learning Nanodegree program.

## Environment

Two agents (tennis rackets) pass a tennis ball. Every time an agent hits the ball over the net, it receives a reward of +0.1. If an agent drops the ball or hits it out of the field, it receives a punishment of -0.01 and the episode ends.

### States

Each agent observes 8 variables corresponding to the position and velocity of the ball and racket. However, the Unity environment returns three observation; thus, the actual state size for each agent is 24.

### Actions

The action space consists of two continuous variables: jumping and moving towards or away from the net.

### Helper functions

The implementation uses two helper functions to reshape outputs from the environment. The `open_brain_surgery` function unpacks the next state, rewards, and episode termination information.

For additional tests (see below), the `transform_rewards` function transforms the rewards to correspond to the rewards from purely competitive play where a player gains +1.0 if it wins the match and -1.0 if it loses the match, with no explicit reward for hitting the ball over the net.

## Learning Algorithm

The implementation uses a multi-agent version of the Deep Deterministic Policy Gradient algorithm, an extension of Deep Q-Learning to continuous state and action spaces.

The algorithm uses two neural networks (each with a local and a target network): an Actor Network to map states to actions, and a Critic Network to estimate Q-values for different state-action pairs.

## Agent Class

The DDPG agent is implemented in the `Agent` class. In addition to the usual inputs (the size of the state and action spaces and a random seed), the number of hidden layers (at least one) and their neurons for the actor and the critic networks can be passed in lists to the `hidden_sizes_actor` and `hidden_sizes_critic` parameters.

## Hyperparameters

The hyperparameters for training in all settings were set to the following values:

```
BUFFER_SIZE = int(1e5)

BATCH_SIZE = 128

GAMMA = 0.99

TAU = 1e-3

LR_ACTOR = 1e-4

LR_CRITIC = 1e-3

WEIGHT_DECAY_AC = 0

WEIGHT_DECAY_CR = 0
```

## Model Architectures

The local and the target networks for both the actor and the critic are initialized with the same random weights. The Adam optimizer is used for the learning rates in both networks, and soft-updates are used to update the values of the target networks. To stabilize learning, the critic also uses gradient clipping.

The actor networks (depicted below) determines the policy to map states to actions and consist of three fully connected layers. There are two hidden layers with 64 and 32 neurons, respectively, using leaky RELU activation, and a final layer that maps outputs from the hidden layers to the action space using the hyperbolic tangent function.

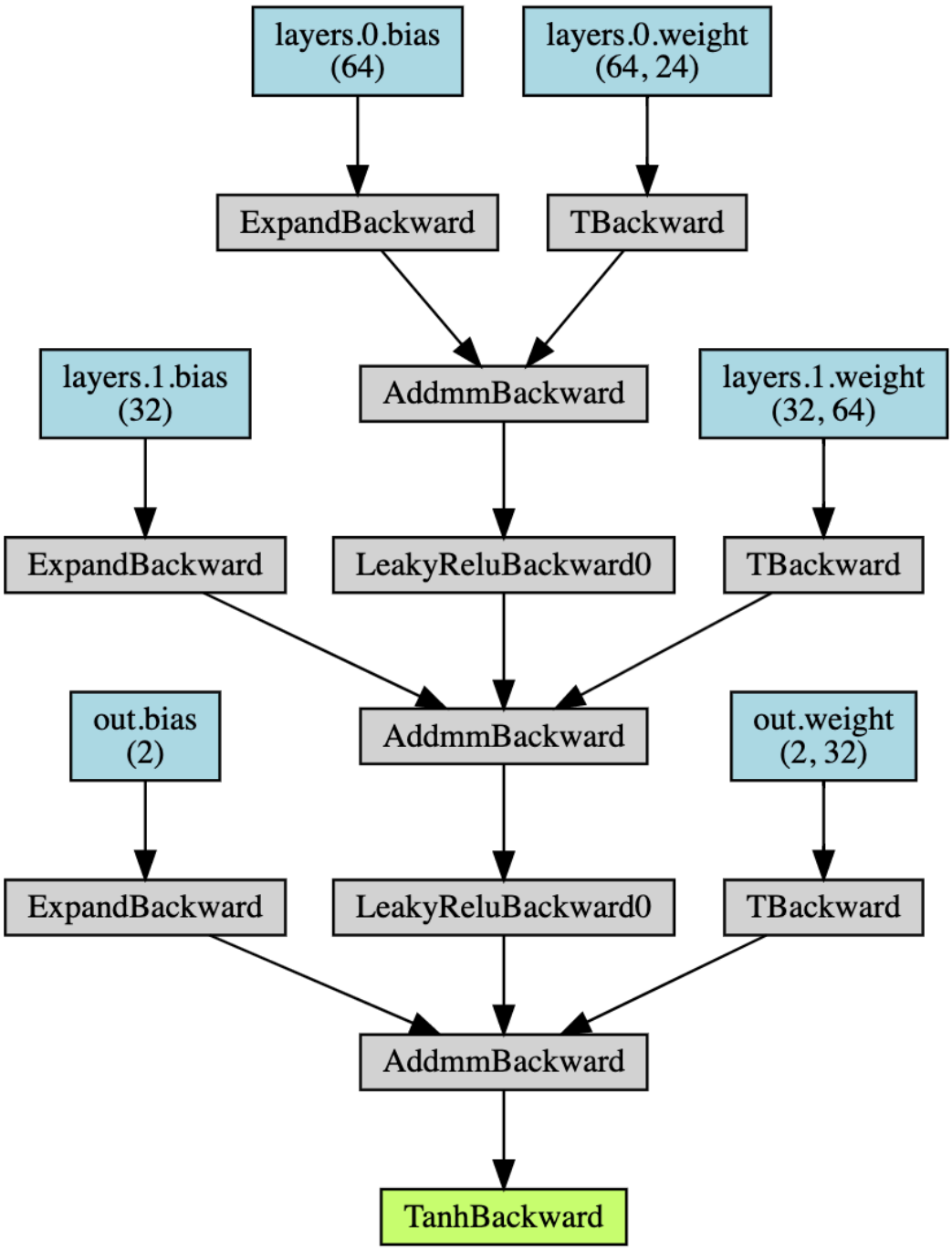


**Figure 1: Actor Network**

The critic networks (depicted below) estimate Q-values for different state-action pairs and consist of four fully connected layers. There are three hidden layers with 128, 64 and 32 neurons, respectively, using leaky RELU activation, and a final layer that maps outputs from the hidden layers to Q-values, again using leaky RELU activation.

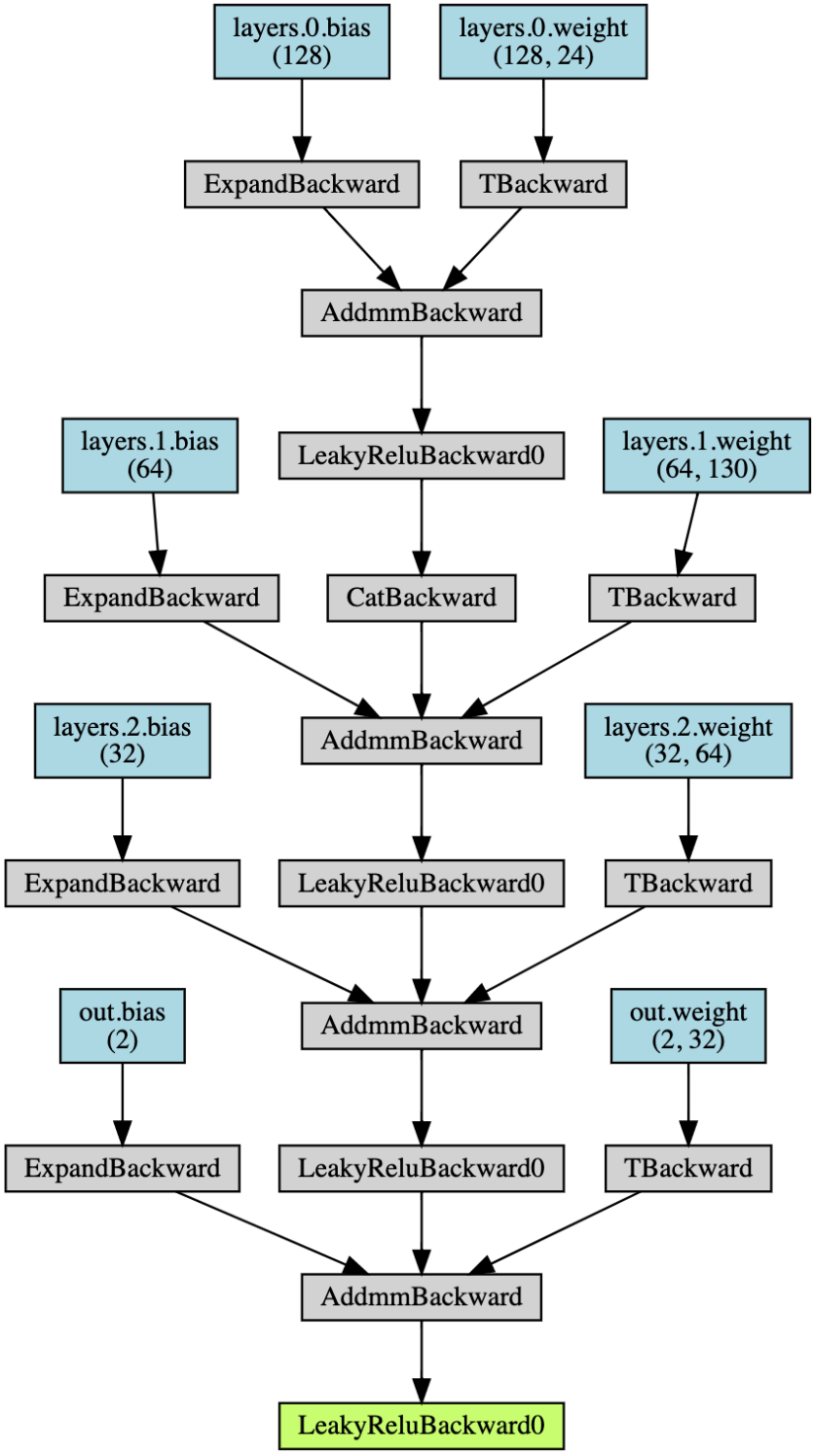


**Figure 2: Critic Network**

## Replay Buffer

The implementation uses a Replay Buffer to store past experiences (tuples of the state, action, reward, next state, and episode termination signal) and use random batches of past experiences to learn.

## Noise

Ornstein-Uhlenbeck noise is added to the action recommended by the actor network to facilitate exploration.

# Training

The agent was trained in two different settings:

1. Using one Actor and one Critic to train both tennis players jointly.

2. Using two Actors and Critics to each train one tennis player.

In addition, I tested whether the agents can still learn to pass the ball under zero-sum tennis rewards – instead of being rewarded for passes over the net, players receive +1.0 for winning the game and -1.0 for losing. Again, the agents were trained in two different settings:

3. Using one Actor and one Critic with competing rewards.

4. Using two Actors and Critics with competing rewards.

## 1. Joint Training, Normal Rewards

The algorithm is learning relatively quickly, solving the environment in less than 1300 episodes. Learning is also relatively reliable (previous runs took between 800 and 1800 episodes to solve the environment).

```
Environment solved in 1296 episodes!    Average Score: 0.51
```

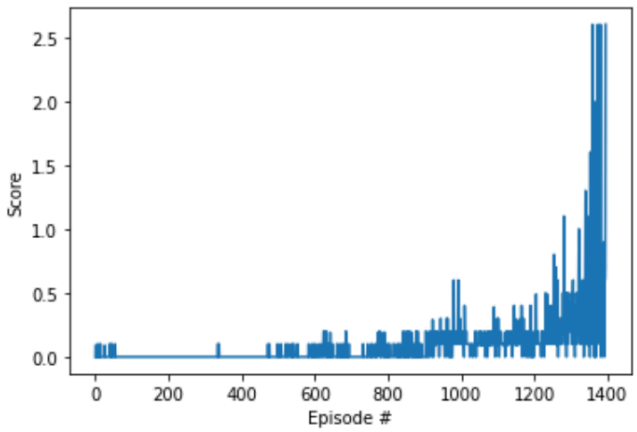


**Figure 3: Plot of Rewards**

## 2. Separate Training, Normal Rewards

If two separate agents have to learn to play the game together, learning takes more time (almost 2300 episodes) and learning is less reliable. While some previous runs took as little as 1400 episodes, many others wouldn't solve the environment even in 10000 episodes, cycling between average scores of 0.1 and 0.4.

```
Environment solved in 2291 episodes!    Average Score: 0.50
```

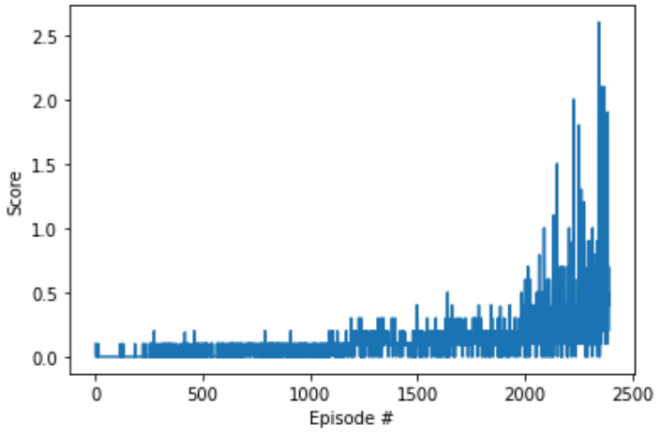


**Figure 4: Plot of Rewards**

## 3. Joint Training, Competitive Rewards

Training with competitive rewards (where players just want to win the game) took a bit more than 2150 episodes. As in the previous setting with jointly trained agents, the training is relatively reliable.

```
Environment solved in 2161 episodes!    Average Score: 0.50
```

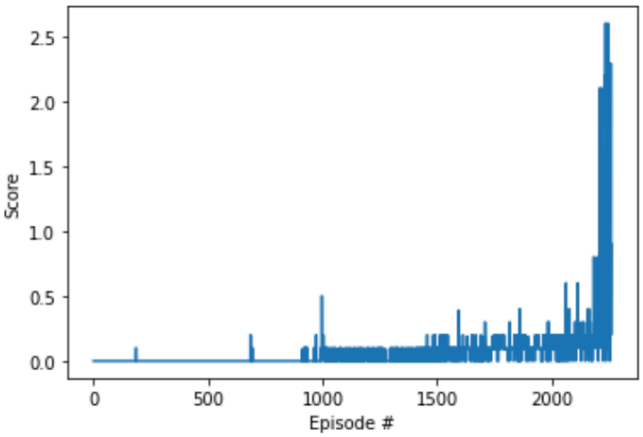


**Figure 5: Plot of Rewards**

## 4. Separate Training, Competitive Rewards

Finally, with separate training and competitive rewards, learning again takes considerably more time (just over 6000 episodes) and is much less reliable.

```
Environment solved in 6002 episodes!    Average Score: 0.51
```

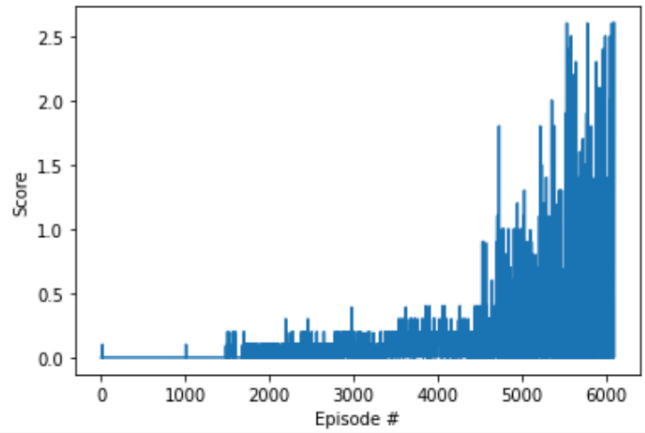


**Figure 6: Plot of Rewards**

# Conclusion and Possible Extensions

As an extension, I tested whether the agents can still learn to pass the ball under zero-sum tennis rewards – instead of being rewarded for passes over the net, players receive +1.0 for winning the game and -1.0 for losing.

While the results suggest that training with competitive rewards takes somewhat longer, the agents were nevertheless able to solve the environment. The longer training time is to be expected, as the purely competitive reward signal is necessarily sparser than the original rewards, because they only occur when the episode has terminated. It's quite remarkable that the agents achieve a sufficiently high score in terms of the original rewards to solve the environment, even though they do not have an explicit incentive to pass the ball.

Performance and training speed could potentially be increased further:

1. The hyperparameters used are likely not optimal. In particular, it is likely that the optimal learning, update, and decay rates will differ in the setting where agents are trained separately. Additional tuning or grid search could be used to make the training more efficient.

2. The ReplayBuffer class could be appended to support prioritized experience replay.

3. Switching to more advanced algorithms, such as D4PG, A3C, TPRO, or PPO might yield better performance.