# Introduction to Reinforcement Learning
# Project 3: Collaboration and Competition

**Jose Lara** | jose.lara.l@outlook.com

## Problem Statement

The goal of this project is to train competitive and collaborative agents to navigate a Reacher Unity environment to maximize rewards. This environment is a Tennis game, where two agents control the motion of two rackets to hit a ball over a net. At the end of each episodic task, the agent receives a reward of +0.1 if an agent hits the ball over the net, and an agent will receive a reward of -0.01 if the agent lets the ball hit the ground or hits the ball out of bounds. The main goal of this training is to keep the ball in play.

Using a Deep Q Network, based on PyTorch library, and a Multi Agent DDPG algorithm, a final set of optimized weights generated through training must provide an optimal solution. The RL agent based of these weights must be able to average a reward of +0.5 over 100 consecutive episodes.

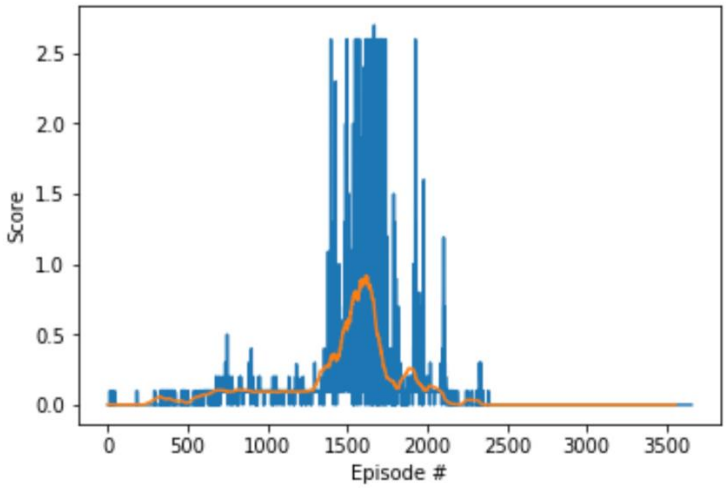


**Figure 1:** Training reward example of the average rewards for a trained agent that meets the assignment's criteria.

An important note added in the project is that due to the multi-agent nature of the problem, training instability may occur. It's important to track the full training and try to not stop training too early.

# Design

## Model: Deep Q Networks:

An Actor Critic Method was used to solve this problem. Both the actor and the critic architecture are composed of neural networks with two hidden layers, the first with 64 nodes and the second with 128 nodes. Both neural networks use the Leaky Rectified Linear Unit activation function, with 'leakiness' parameter that can be changed via a training agent argument to try different training configurations.

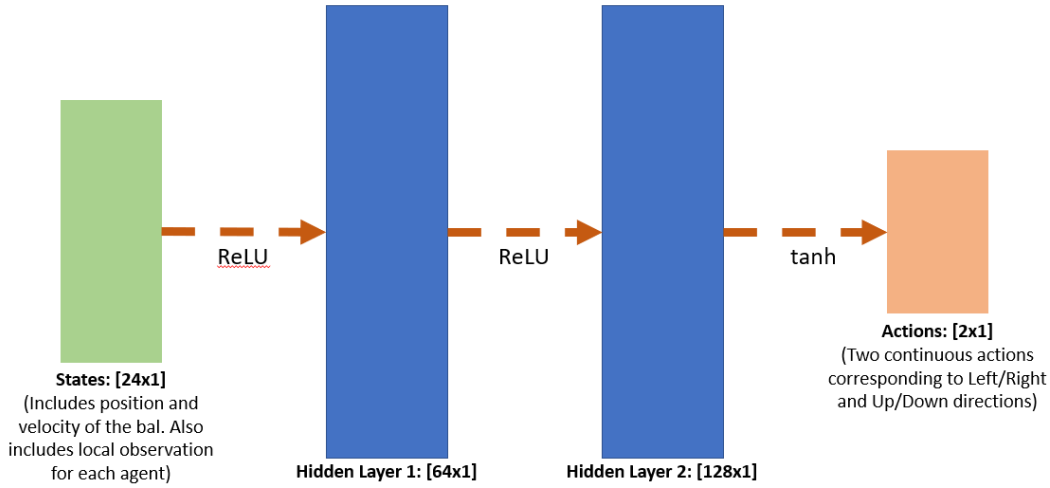


**Figure 2:** Actor Neural Network configuration includes an additional hyperbolic function to normalize inputs to correct range.

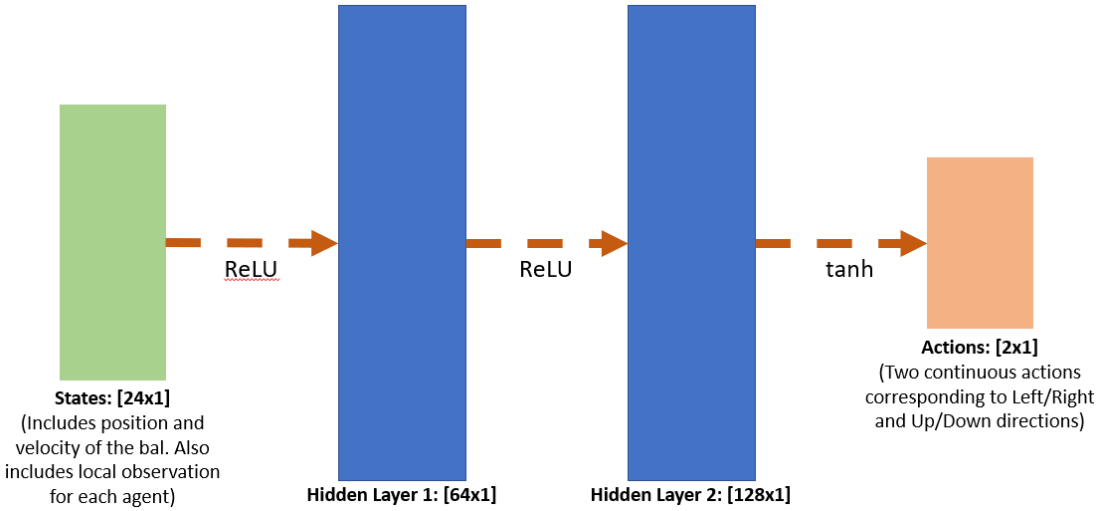


**Figure 3:** Critic Neural Network configuration.

## Algorithm

The algorithm used mainly stems from the example provided in the Deep Deterministic Policy Gradient, Example in a previous lesson, **Lesson 5: Actor-Critic Methods**. In the same algorithm, noise is added to each episode to improve the training. Also, a queue with results from previous episodes to sample experience replay.

Since this was the same algorithm used for the 20 Reacher environment on the previous assignment, it was relatively easy to update the code functionality.

# Training Results

The warning was correct, there was some instability at around ~150 episodes. However, after leaving the agent training for more episodes, the agent was able to solve the problem.

The main parameter that impacted the training was the Learning Rate for both the Actor and Critic networks. Increasing each parameter provided a signal of improved training.

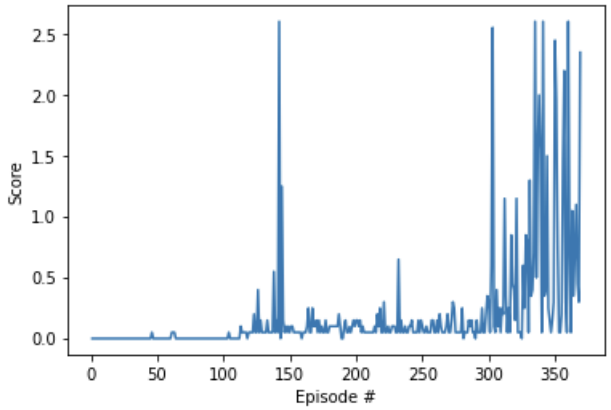


**Figure 3:** Earned rewards through training agent using the Tennis environment, with an average reward of 0.5 within 100 episodes. The solution was reached after 269 episodes.

### Training Hyperparameters

| Hyperparameter Name | Value |
|---|---|
| Buffer Size | 1e6 |
| Batch Size | 64 |
| Gamma (Future Reward) | 0.99 |
| Tau | 6e-3 |
| Actor Learning Rate | 3e-4 |
| Critic Learning Rate | 3e-3 |
| Weight Decay | 0.001 |
| Leakiness | 0.01 |

**Table 1:** Hyperparameter used for training

## Next Steps

For next steps, I want to improve my multi-agent class. First, I want to add an argument that takes the type of algorithm and implement each algorithm that we learned in this course. Also, I want to learn how to implement in the Montel Carlo Tree search and apply it to this multi-agent class so that it can train turn-based games.

I will also build my own Unity environment with multiple agents and test my code. Some of my ideas include a multi-goal ice hockey game and a short racing game where a player can play with agents trained at different average rewards to increase difficulty.