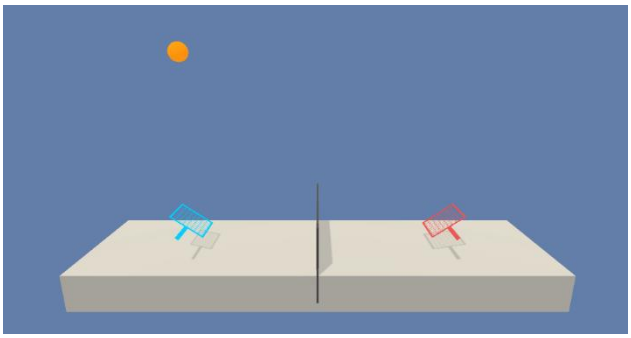
Project 3 – Collaboration and competition

Introduction:

The purpose of this project is to train two agents which 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.

For this project, we will work with the Unity Tennis environment.



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.

Learning Algorithm:

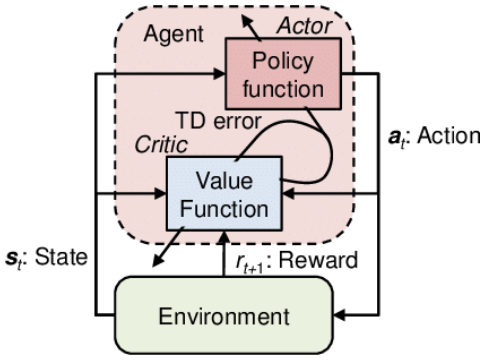
We may consider that both agents are learning to play a game of trying to get the ball staying in the air, so it may be considered as a unique game played by two agents, instead of a sum-zero (competitive) game where both agents want the other to fail.

Based on the above premise, we will start testing with the DDPG algorithm used in the previous project (Continuous Control). Both agents will use the same neural network architecture and the **same replay buffer**.

The DDPG algorithm is an Actor-critic method, these methods main characteristics are:

- The reduction the high-variance commonly associated to policy-based agent,
- It learns faster than policy-based methods and
- have more consistent convergence than value-based agents

For this agent to be an Actor-critic, we have two learning neural networks:



- In a standard Actor-critic method, the **Actor network** will learn the probabilities of good and bad actions. Will take in states and outputs a distribution over the possible actions. But **in the case of DDPG**, the algorithm used in this project, **the Actor will give us the best possible action based on the policy learnt, not a probability distribution.**
- The **Critic network** will learn how to estimate good actions and bad actions to help on the actor decisions. Technically it will learn to evaluate the state value function

(Vπ), will take in states and outputs an expectation of the state value function of the policy, **this state value function will update the actor network.**

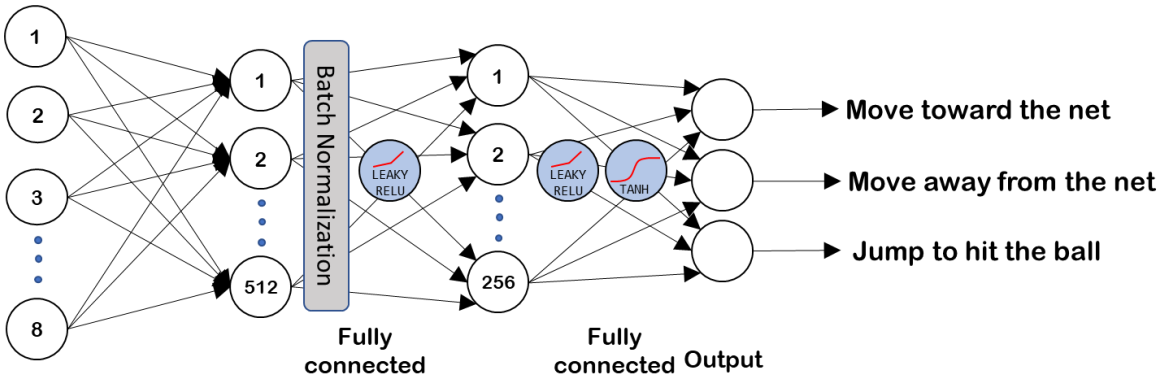
When training is complete, the trained optimal policy will be stored in the actor. The training process for this method are complicated, so a lot of improvements have been implemented to increase it learning efficiency.

The hyperparameters used for the project are the following ones:

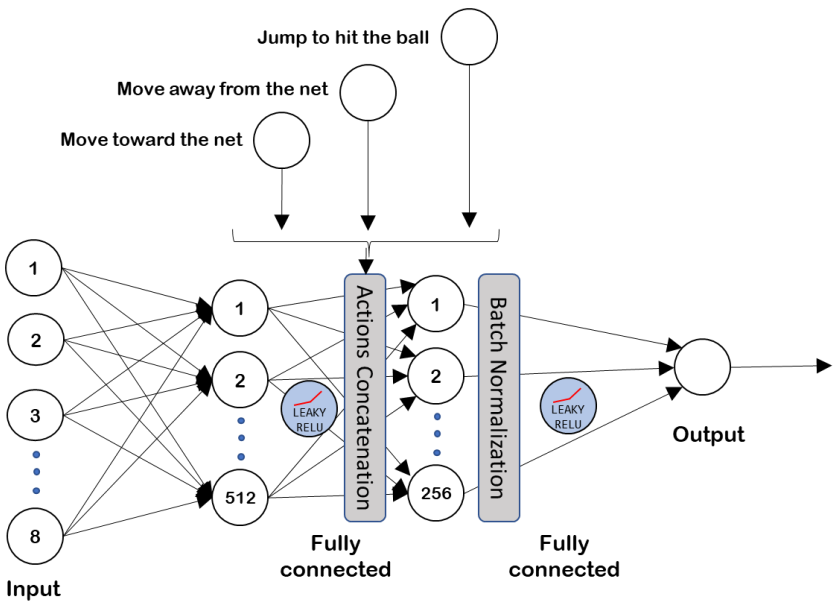
Replay buffer size	BUFFER_SIZE = int(1e8)
Minibatch size	BATCH_SIZE = 128
Discount factor	GAMMA = 0.99
Soft update of target neural network	TAU = 1e-3
Initial Noise	EPS_INITIAL = 1.0
Noise decay	EPS_DECAY = 0.9995
Minimum Noise	EPS_END = 0.01

Actor Neural Network:

The actor NN used is the following one, the ending function used is a Tanh.



Critic Neural Network:



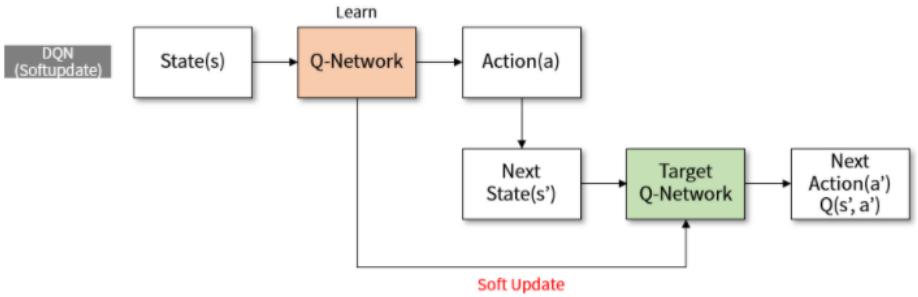
Thanks to the implementation of **experience replay**, the algorithm stores states which are rare and actions that are costly to be able to recall them, each experience (Which is formed by State, Action, Reward and Next

State) is stored in a buffer as the agent is interacting with the environment, afterwards the agent randomly samples a small batch of these experiences in order to learn from them. Thanks to this:

- it learns from individual state-actions multiple times, so it recalls rare occurrences and make a better use of the experience obtained.
- Because of the random sampling, it helps breaking the correlation and prevent actions from oscillating or diverging in wrong ways.

Another critic functionality of the DDPG algorithm is a **soft update** to the target network.

The implementation of this soft update means that the algorithm has two neural networks. The called "Regular" neural network and the "Target" neural network.



In our algorithm, this soft update is done every step, so the networks are blended with a percentage of merging considering the TAU variable, so considering the TAU variable is set to 0,05%, every step a 0,05% of the regular network will be merged with the target network.
Thanks to soft update, the DDPG algorithm gets faster convergence. Soft update logic could be implemented in other off-policy algorithms that use target networks, such as DQN.

To encourage exploration in the training process, we added **noise** using the Ornstein-Uhlenbeck noise process, the noise is reduced through time, because when the agent is more experienced, the need for exploration decreases.

We used **batch normalization** in both networks for stabilizing the learning process and reduce the training time for deep neural networks, batch normalization is especially useful to limit covariate shift in environments where the input distribution changes with each step, like this.

On the critic side, we have **used leaky relu** activation functions for dealing better with the possible negative values coming from the learning (*Dying ReLU problem*) and improve training performance.

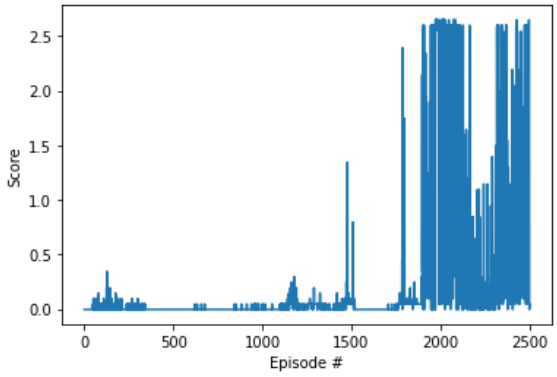
Algorithm performance:

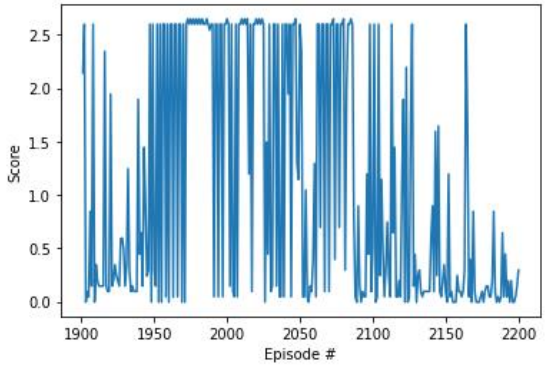
The algorithm showed amazing performance on the first training session, getting to the expected results in 2000 episodes.

I wasn't expecting this to worked on the first try, as I expected needed to mirror the actions or states for the second actor. However, we can see in the video attached of the agents playing that the agent learnt to play the game.

I will try new training sessions to check if this performance is normal.

```
Episode 50      Average Score: -0.00    MinScore: -0.00         MaxScore: 0.00  Hour: 13:16:25
Episode 100     Average Score: 0.01     MinScore: -0.00         MaxScore: 0.01  Hour: 13:16:55
Reset values avgscore, max and minscore
Episode 150     Average Score: 0.02     MinScore: 0.01  MaxScore: 0.02  Hour: 13:17:38
Episode 200     Average Score: 0.02     MinScore: 0.01  MaxScore: 0.03  Hour: 13:18:18
Reset values avgscore, max and minscore
Episode 250     Average Score: 0.01     MinScore: 0.01  MaxScore: 0.02  Hour: 13:18:45
Episode 300     Average Score: 0.01     MinScore: 0.00  MaxScore: 0.02  Hour: 13:19:18
Reset values avgscore, max and minscore
Episode 350     Average Score: 0.01     MinScore: 0.01  MaxScore: 0.01  Hour: 13:19:52
Episode 400     Average Score: -0.00    MinScore: -0.00         MaxScore: 0.01  Hour: 13:20:14
Reset values avgscore, max and minscore
Episode 450     Average Score: -0.00    MinScore: -0.00         MaxScore: 0.00  Hour: 13:20:39
Episode 500     Average Score: -0.00    MinScore: -0.00         MaxScore: 0.00  Hour: 13:21:00
Reset values avgscore, max and minscore
Episode 550     Average Score: -0.00    MinScore: -0.00         MaxScore: 0.00  Hour: 13:21:25
Episode 600     Average Score: -0.00    MinScore: -0.00         MaxScore: 0.00  Hour: 13:21:47
Reset values avgscore, max and minscore
Episode 650     Average Score: -0.00    MinScore: -0.00         MaxScore: 0.00  Hour: 13:22:12
Episode 700     Average Score: -0.00    MinScore: -0.00         MaxScore: 0.00  Hour: 13:22:37
Reset values avgscore, max and minscore
Episode 750     Average Score: -0.00    MinScore: -0.00         MaxScore: 0.00  Hour: 13:22:59
Episode 800     Average Score: -0.00    MinScore: -0.00         MaxScore: 0.00  Hour: 13:23:22
Reset values avgscore, max and minscore
Episode 850     Average Score: -0.00    MinScore: -0.00         MaxScore: 0.00  Hour: 13:23:46
Episode 900     Average Score: -0.00    MinScore: -0.00         MaxScore: 0.00  Hour: 13:24:09
Reset values avgscore, max and minscore
Episode 950     Average Score: -0.00    MinScore: -0.00         MaxScore: 0.00  Hour: 13:24:39
Episode 1000    Average Score: 0.00     MinScore: -0.00         MaxScore: 0.00  Hour: 13:25:09
Reset values avgscore, max and minscore
Episode 1050    Average Score: 0.00     MinScore: 0.00  MaxScore: 0.00  Hour: 13:25:38
Episode 1100    Average Score: -0.00    MinScore: -0.00         MaxScore: 0.00  Hour: 13:26:01
Reset values avgscore, max and minscore
Episode 1150    Average Score: 0.01     MinScore: -0.00         MaxScore: 0.01  Hour: 13:26:378
Episode 1200    Average Score: 0.05     MinScore: -0.00         MaxScore: 0.05  Hour: 13:27:43
Reset values avgscore, max and minscore
Episode 1250    Average Score: 0.04     MinScore: 0.04  MaxScore: 0.05  Hour: 13:28:17
Episode 1300    Average Score: 0.03     MinScore: 0.03  MaxScore: 0.05  Hour: 13:29:02
Reset values avgscore, max and minscore
Episode 1350    Average Score: 0.04     MinScore: 0.03  MaxScore: 0.04  Hour: 13:29:46
Episode 1400    Average Score: 0.02     MinScore: 0.02  MaxScore: 0.04  Hour: 13:30:17
Reset values avgscore, max and minscore
Episode 1450    Average Score: 0.01     MinScore: 0.01  MaxScore: 0.02  Hour: 13:31:19
Episode 1500    Average Score: 0.05     MinScore: 0.01  MaxScore: 0.05  Hour: 13:32:58
Reset values avgscore, max and minscore
Episode 1550    Average Score: 0.06     MinScore: 0.05  MaxScore: 0.07  Hour: 13:33:49
Episode 1600    Average Score: 0.01     MinScore: 0.01  MaxScore: 0.07  Hour: 13:34:10
Reset values avgscore, max and minscore
Episode 1650    Average Score: -0.00    MinScore: -0.00         MaxScore: 0.01  Hour: 13:34:34
Episode 1700    Average Score: -0.00    MinScore: -0.00         MaxScore: 0.01  Hour: 13:34:57
Reset values avgscore, max and minscore
Episode 1750    Average Score: -0.00    MinScore: -0.00         MaxScore: 0.00  Hour: 13:35:20
Episode 1800    Average Score: 0.10     MinScore: -0.00         MaxScore: 0.10  Hour: 13:37:300
Reset values avgscore, max and minscore
Episode 1850    Average Score: 0.13     MinScore: 0.10  MaxScore: 0.13  Hour: 13:38:23
Episode 1900    Average Score: 0.10     MinScore: 0.07  MaxScore: 0.13  Hour: 13:39:41
Reset values avgscore, max and minscore
Episode 1950    Average Score: 0.39     MinScore: 0.12  MaxScore: 0.39  Hour: 13:45:49
Episode 2000    Average Score: 1.31     MinScore: 0.12  MaxScore: 1.31  Hour: 14:02:49
Reset values avgscore, max and minscore
Episode 2050    Average Score: 1.89     MinScore: 1.00  MaxScore: 1.89  Hour: 14:18:33
Episode 2100    Average Score: 1.53     MinScore: 1.00  MaxScore: 1.91  Hour: 14:30:34
Reset values avgscore, max and minscore
Episode 2150    Average Score: 0.94     MinScore: 0.94  MaxScore: 1.53  Hour: 14:36:32
Episode 2200    Average Score: 0.42     MinScore: 0.42  MaxScore: 1.53  Hour: 14:39:02
Reset values avgscore, max and minscore
Episode 2250    Average Score: 0.20     MinScore: 0.20  MaxScore: 0.40  Hour: 14:41:14
Episode 2300    Average Score: 0.18     MinScore: 0.15  MaxScore: 0.40  Hour: 14:43:38
Reset values avgscore, max and minscore
Episode 2350    Average Score: 0.50     MinScore: 0.16  MaxScore: 0.50  Hour: 14:52:59
Episode 2400    Average Score: 0.73     MinScore: 0.16  MaxScore: 0.73  Hour: 15:01:16
Reset values avgscore, max and minscore
Episode 2450    Average Score: 0.71     MinScore: 0.71  MaxScore: 0.81  Hour: 15:10:46
Episode 2500    Average Score: 0.94     MinScore: 0.65  MaxScore: 0.94  Hour: 15:23:44
```





Future improvements:

Some future improvement that could be done in this project are:

1. Use **MaDDPG** algorithm, the learning process should be faster.
2. Implement **Prioritized Experience replay**, the learning process should be more efficient
3. Try making a hard-update on the target network **periodically** to make the learning process faster without losing the soft updates improvements to the learning process.
4. Decrease exploration through the learning process
5. Implementing some features described in the benchmarking DeepRL for Continuous control paper:
    - Rescale the reward by a factor of 0.1 through the learning process, this should improve stability in the learning process.
    - Change learning rates to $5x10^{-3}$, usually this value improves performance.
6. Analyze if it's possible to improve the neural network architecture by using Archai:

    https://www.microsoft.com/en-us/research/blog/archai-can-design-your-neural-network-with-state-of-the-art-neural-architecture-search-nas/