# Continuous Control Project
# Report

## Algorithm Description:

I use a ***Deep Deterministic Policy Gradient*** (DDPG), which is s Policy Gradient approach.
The algorithm is using the Actor-Critic approach.

The Actor is a Neural Network that approximates a deterministic policy. It takes in input the state and the output is an array of values, one for each action. It approximates the policy $\pi(a|s;\theta\pi)$. This policy is deterministic, and it represents the policy value in the action space, with values in the range [-1,1].

The Critic is used to give feedback about the value of the input state observed by the actor and the action considered by the actor for that state. The feedback is computed using the gradient of the advantage function computed using the tuple (state, next state, action, reward) sampled from the memory buffer.

Both Actor and Critic have a target and regular networks. The target network is used to define the desired target, avoiding to have the same network for considering the prediction and the target. This is the same idea used in the DQN algorithm.

A soft update approach is implemented, which means that the target network is updated more often but with a smaller change.

The DDPG, as many policy gradient algorithm, suffers from having large variance, due to their Monte Carlo approach in computing the cumulative reword. To reduce the variance, a Temporal Difference approach is used together with bootstrapping of one, this approach reduces the variance but introduces a bias.

The exploration of the action space is done by injecting a noise after the action has been selected from the Actor Network.

## Model Description of the networks

The actor network is composed by 3 linear layers:

Layer 1 : state_size * 256

Layer 2 : 256 *128

Layer 3:  128 * action_size

Between layer 1 and 2, and layer 2 and 3, the network has no-linear activation functions (ReLU), while the last layer is followed by a tangh no-linear function to restrict  the value in [-1,1].

The critic network is composed by 3 linear layers:

Layer 1 : state_size * 256

Layer 2 : 256 +action_space*128

Layer 3:  128 * 1

Between layer 1 and 2, and layer 2 and 3, the network has the no-linear activation functions (ReLU).

## Hyper Parameter Configuration

BUFFER_SIZE = int(1e6)  # replay buffer size

BATCH_SIZE = 128  # minibatch size

GAMMA = 0.99  # discount factor

TAU = 1e-3  # for soft update of target parameters

LR_ACTOR = 1e-4  # learning rate of the actor

LR_CRITIC = 1e-4  # learning rate of the critic

WEIGHT_DECAY = 0.0  # L2 weight decay

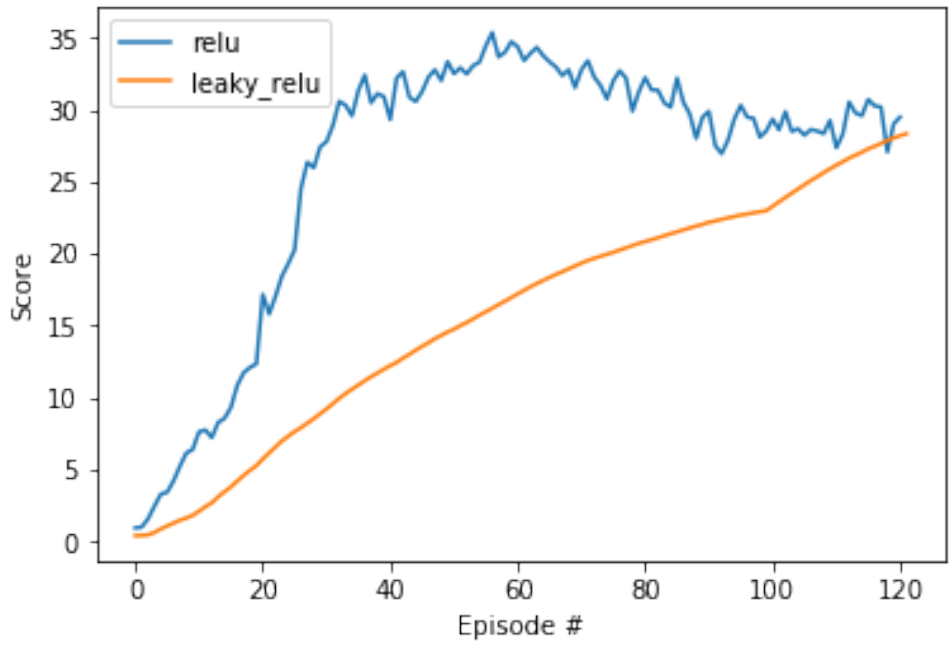
N_LEARN_UPDATES = 10  # number of learning updates

N_TIME_STEPS = 20  # every n time step do update

## Learning Algorithm

- Every N_TIME_STEPS a batch of experiences is sampled from the buffer of size BATCH_SIZE. (*step function in the agent class , file ddpg_agent.py*)
- Then, multiple learning steps are computed ( N_LEARN_UPDATES times) *(step function in the agent class , file ddpg_agent.py*)
- Each experience is made by (state, next_state, action reward ).
- This tuple is used to update the actor and the critic networks. *(learn function in the agent class , file ddpg_agent.py*)
- In particular:
    - The actor network computes the next action using the next state.
    - This next action is used to compute the temporal difference between the Expected and Target Q value.
    - The MSE loss between the target and expected Q value is used in the optimization algorithm (ADAM) to update the weights of the critic network.
    - The optimization algorithm (ADAM) is also used to update the weights of the actor network using the gradient of the negative mean of the predictions of the actions from the local network for the given states.

## Score Plot



- I have observed a **good improvement** in the average score, if I use the **ReLU** activation in the critic instead of the **Leaky ReLU**
- I think that the results are strongly correlated with the diversity of samples within the buffer ,so the addition of prioritized experience replay sampling and importance-sampling (as in the advanced version of the DQN) should be very helpful. This is also well described in this video

**Future works:**

- Use different algorithm like Raimbow
- Change the approach for action exploration using OpenAI approach or noise networks as explained here