Continuous Control

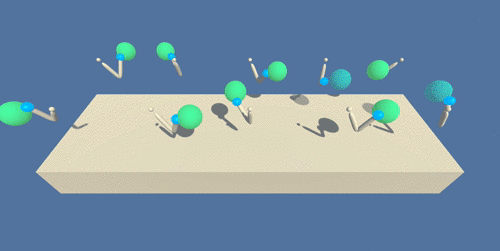
Project Report

# Introduction

This project involves solving an environment in which an agent (a double-jointed arm) which has freedom of motion in 3-D until a certain fixed distance, has to follow a target location and stay in its vicinity for as many time steps as possible. My choice of algorithm for this task is to use DDPG (Deep Deterministic Policy Gradient) Algorithm which is an off-policy actor-critic algorithm, and is well suited for tasks like these.

# Environment

For this project, we will work with the Reacher environment. In this environment, a double-jointed arm can move to target locations. A reward of +0.1 is provided for each step that the agent's hand is in the goal location. Thus, the goal of the agent is to maintain its position at the target location for as many time steps as possible.



The observation space consists of 33 variables corresponding to position, rotation, velocity, and angular velocities of the arm. Each action is a vector with four numbers, corresponding to torque applicable to two joints. Every entry in the action vector should be a number between -1 and 1.

# Implementation

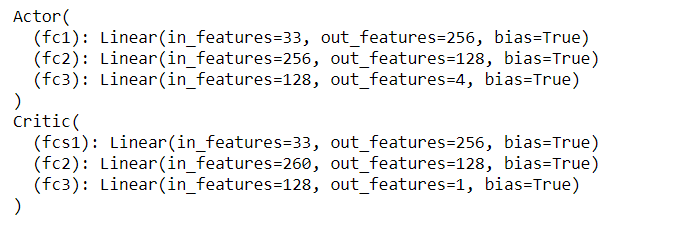
The repository consists of the following files:

1. Continuous\_Control.ipynb
2. Ddpg\_agent.py
3. Model.py
4. Checkpoint\_actor.pth
5. Checkpoint\_critic.pth
6. Reacher\_Windows\_x86\_64

Continuous\_control.ipynb is the jupyter notebook where the training happens. We initially setup the environment instance and do the basic analysis of it, lie the number of agents (1 for version 1), actions size (4), state vector size (33) etc. The agent is required to take the state vector as input and outputs 4 values between -1 and 1 corresponding to torque applicable at two joints.

The training loop consists of n\_episodes of the environment. We collect the episode and use it to train the agent. This training happens until the episode is done without stopping in between. And if the average score crosses 30, we save the weights and break the loop.

Model.py consists of two nn.modules for both actor and critic networks. The architectures are given below. Actor has an output activation function of Tanh to get the appropriate range for action space.

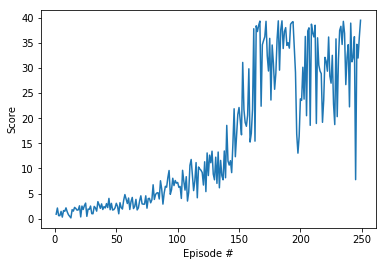


# Learning Algorithm

I used the Adam optimizer to reduce the loss. Loss function of choice is MSE Loss

|  |  |
| --- | --- |
| Parameter | Value |
| Buffer Size | 1e5 |
| Batch Size | 128 |
| Gamma | 0.99 |
| Tau | 1e-3 |
| LR actor | 1e-4 |
| LR critic | 1e-4 |
| Weight Decay Actor | 0 |
| Weight Decay Critic | 0 |
| Epsilon Start | 1 |
| Epsilon End | 0.05 |
| Epsilon Decay | 3e-5 |

# Training



# Ideas for Future work

In addition to DDQN, we can experiment with using the next generation of DQNs, such as Dueling Network Architectures, Dynamic Frame skip Deep Q networks, Rainbow networks, Deep Recurrent Q-Learning.