# Project 2: continuous control

## 1. Introduction

In this project, I work with the Reacher environment that double-jointed arm can move to target location. The position, rotation, velocity, and angular velocities of the arm are represented by 33 variables. These 33 variables are the observation space, or “state” in our model. Obviously, the states are continuous variables. The torque of the two joins are represented by four numbers. These four numbers are our actions. Each action number can be any continuous value between -1 to 1.

In this environment, if the agent’s hand is in the targeted location, it received a reward as +1. Therefore, to train the arms maintain in the targeted locations means we want to achieve higher accumulated reward.

For this project, we have two different version of the Unity environment.

- Single agent training
- 20 identical agents training

The second option has 20 identical agents to be training simultaneously but independently. This helps us to increase the training speed. Therefore, I chose to train the second version environment.

To train these 20 agents, I used the Deep Deterministic Policy Gradient (DDPG) algorithm which is an actor-critic, model-free algorithm. This algorithm can improve the policy gradient with a critic network. The critic network estimates the value function, this could be the action-value or state-value. The actor network updates the policy distribution in the direction suggested by the critic, such as with the policy gradient.

2. Model architecture

I used similar code as provided in the Udacity github repository. The general structure of actor and critic network is described below.

Actor network:

Linear layer -> relu activation -> linear layer -> relu activation -> linear layer -> tanh activation

Critic network:

Linear layer -> relu activation -> linear layer -> relu activation -> linear layer

The reason why I use the tanh activation function for the output of actor network is because the action's values are between -1 to 1.

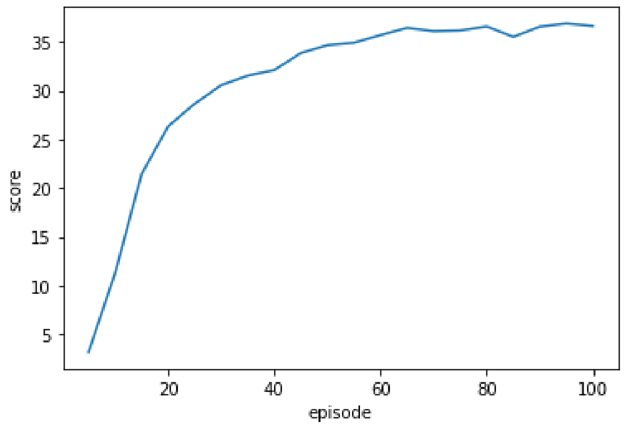
Except these two neural networks, I also created a class named "ReplayBuffer". This buffer can store the steps (with states, rewards, action, next_state, done) information and provide a pool for selection. In this project, the random selection process is controlled by random seed. However, it is possible to improve this step by invoking the weighted selection: the network will tend to select steps information with higher reward.

To adapt to the potential long training process, the learning rate of arctic and critic neural network have a decay factor as 0.999 and this decay process is controlled by the lr_scheduler function which is part of the optimization.

## 3. Results

The training process is kind of slow, so I did not compare any parameter tuning. Since we have 20 agents total in this training process, the scores shown below is the averaged score for all 20 agents. From the figure below, we can see the averaged score of 20 agents reaches to 30 since 30 episodes. This is faster than I expected, maybe it is because I used quite large number of nodes in the network layers.

The first 30 episodes show high increasing rate and the increasing rate gradually slowed down after episode 30. Score keep increasing until when I stopped the training process. I think if I train longer time, the score is expected to keep increasing but won't have significant improvements.



## 4. Future direction

Except of the DDPG algorithm I explored here, there are still many other approaches which suit for this problem. For example, the D4PG, A2C, A3C, and so on.