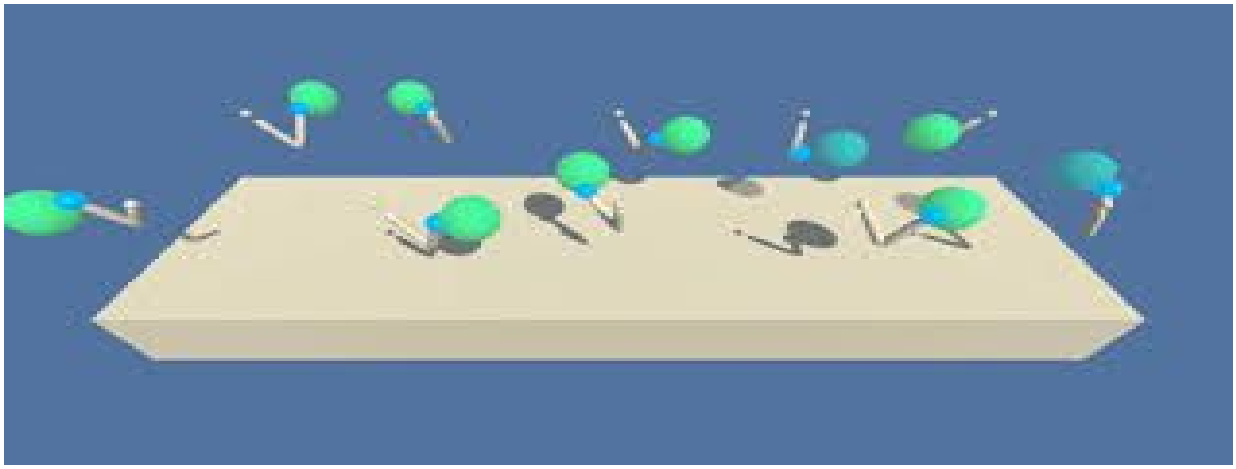
UDACITY DEEP REINFORCEMENT NANODEGREE
PROJECT 2
CONTINUOUS CONTROL

INTRODUCTION

This project was carried out as a part of the deep reinforcement learning nanodegree from Udacity. For this project the environment provided is Reacher made in unity3D.

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 your 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 implementation consists of files ddgn_agent.py, model.py and ddpg_trainer.py. The code is organized as follows:

1. ddgn_agent.py contains code for the agent.
2. model.py contains the neural network code that is used by the agent.
3. ddpg_trainer.py contains code that is used to train the agent.

For information about the project and the environment can be found in the file README.md.

LEARNING ALGORITHM

The algorithm chosen to solve the Reacher problem is Deep Deterministic Policy Gradient (DDPG). The algorithm from spinningup.openai.com is given below for reference.

Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for however many updates **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets

$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$$

    end for
  end if
until convergence

Network hyperparameters

Parameter and its value	Comments
BUFFER_SIZE = int(1e5)	# Replay buffer size
BATCH_SIZE = 128	# Minibatch size
GAMMA = 0.99	# Discount factor
TAU = 1e-3	# For soft update of target parameters
LR_ACTOR = 3e-4	# Learning rate of the actor
LR_CRITIC = 1e-3	# Learning rate of the critic
WEIGHT_DECAY = 0	# L2 weight decay

MODEL SUMMARY

The architecture of the actor and critic neural network is given below

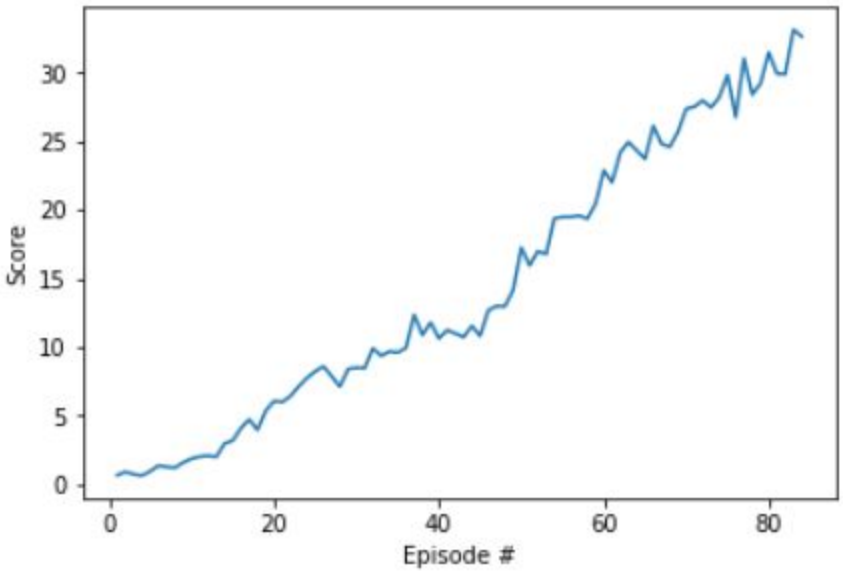
```
Out[12]: Actor(
           (fc1): Linear(in_features=33, out_features=400, bias=True)
           (fc2): Linear(in_features=400, out_features=300, bias=True)
           (fc3): Linear(in_features=300, out_features=4, bias=True)
         )


Critic(
  (fcs1): Linear(in_features=33, out_features=400, bias=True)
  (fc2): Linear(in_features=404, out_features=300, bias=True)
  (fc3): Linear(in_features=300, out_features=1, bias=True)
)
```

PLOT OF REWARDS

Below is the summary of the training with average score and episode count

```
Episode 10      Average Score: 1.14     Score: 1.89
Episode 20      Average Score: 3.66     Score: 6.08
Episode 30      Average Score: 7.61     Score: 8.51
Episode 40      Average Score: 10.27    Score: 10.64
Episode 50      Average Score: 12.54    Score: 17.24
Episode 60      Average Score: 19.03    Score: 22.87
Episode 70      Average Score: 24.77    Score: 27.36
Episode 80      Average Score: 28.77    Score: 31.46
Episode 84      Average Score: 30.22    Score: 32.63
Environment solved in -16 episodes!     Average Score: 30.22
```

A plot of the score Vs Episode is given below



The training ended when the target of average score of 30 was reached as shown in the plot.

After training the weight of the networks are stored as

1. checkpoint_critic1.pth
2. checkpoint_actor2.pth

These files are available as part of the repo

IDEAS FOR FUTURE WORK

So far the agent is trained only using DDPG which can also be implemented using the following algorithms

1. Proximal Policy Optimization(PPO)
2. Generalized Advantage Estimation(GAE)
3. Advantage Actor Critic(A2C)
4. Asynchronous Advantage Actor-Critic(A3C)
5. Distributed Distributional Deterministic Policy Gradients(D4PG)

Also in this attempt the agent interacted with the environment but the agent can be trained using raw pixels from the environment as input.

REFERENCE

[1] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, Proximal Policy Optimization Algorithms

[2] Juliani, A., Berges, V., Vckay, E., Gao, Y., Henry, H., Mattar, M., Lange, D. (2018). Unity: A General Platform for Intelligent Agents. [arXiv preprint arXiv:1809.02627.] (https://github.com/Unity-Technologies/ml-agents)

[3] R. S. Sutton and A. G. Barto, Introduction to Reinforcement Learning, 2nd ed. Cambridge, MA, USA: MIT Press, 2017

[4] Deterministic Policy Gradient Algorithms, Silver et al. 2014

[5]Continuous Control With Deep Reinforcement Learning, Lillicrap et al. 2016