# Continuous_Control

December 3, 2021

## 1 Continuous Control

---

You are welcome to use this coding environment to train your agent for the project. Follow the instructions below to get started!

#### 1.0.1 1. Start the Environment

Run the next code cell to install a few packages. This line will take a few minutes to run!

```
In [1]: !pip -q install ./python
```

```
tensorflow 1.7.1 has requirement numpy>=1.13.3, but you'll have numpy 1.12.1 which is incompatib
ipython 6.5.0 has requirement prompt-toolkit<2.0.0,>=1.0.15, but you'll have prompt-toolkit 3.0.
```

The environments corresponding to both versions of the environment are already saved in the Workspace and can be accessed at the file paths provided below.

Please select one of the two options below for loading the environment.

```
In [2]: from unityagents import UnityEnvironment
        import numpy as np

        # select this option to load version 1 (with a single agent) of the environment
        # env = UnityEnvironment(file_name='/data/Reacher_One_Linux_NoVis/Reacher_One_Linux_NoVi

        # select this option to load version 2 (with 20 agents) of the environment
        env = UnityEnvironment(file_name='/data/Reacher_Linux_NoVis/Reacher.x86_64')
```

```
INFO:unityagents:
'Academy' started successfully!
Unity Academy name: Academy
        Number of Brains: 1
        Number of External Brains : 1
        Lesson number : 0
        Reset Parameters :
                goal_speed -> 1.0
```

```
                goal_size -> 5.0
Unity brain name: ReacherBrain
        Number of Visual Observations (per agent): 0
        Vector Observation space type: continuous
        Vector Observation space size (per agent): 33
        Number of stacked Vector Observation: 1
        Vector Action space type: continuous
        Vector Action space size (per agent): 4
        Vector Action descriptions: , , ,
```

Environments contain ***brains*** which are responsible for deciding the actions of their associated agents. Here we check for the first brain available, and set it as the default brain we will be controlling from Python.

```
In [3]: # get the default brain
        brain_name = env.brain_names[0]
        brain = env.brains[brain_name]
```

### 1.0.2 2. Examine the State and Action Spaces

Run the code cell below to print some information about the environment.

```
In [4]: # reset the environment
        env_info = env.reset(train_mode=True)[brain_name]

        # number of agents
        num_agents = len(env_info.agents)
        print('Number of agents:', num_agents)

        # size of each action
        action_size = brain.vector_action_space_size
        print('Size of each action:', action_size)

        # examine the state space
        states = env_info.vector_observations
        state_size = states.shape[1]
        print('There are {} agents. Each observes a state with length: {}'.format(states.shape[0
        print('The state for the first agent looks like:', states[0])
```

```
Number of agents: 20
Size of each action: 4
There are 20 agents. Each observes a state with length: 33
The state for the first agent looks like: [  0.00000000e+00  -4.00000000e+00   0.00000000e+00
  -0.00000000e+00  -0.00000000e+00  -4.37113883e-08   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00  -1.00000000e+01   0.00000000e+00
   1.00000000e+00  -0.00000000e+00  -0.00000000e+00  -4.37113883e-08
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
```

```
  0.00000000e+00   0.00000000e+00   5.75471878e+00  -1.00000000e+00
  5.55726624e+00   0.00000000e+00   1.00000000e+00   0.00000000e+00
 -1.68164849e-01]
```

### 1.0.3 3. Take Random Actions in the Environment

In the next code cell, you will learn how to use the Python API to control the agent and receive feedback from the environment.

Note that **in this coding environment, you will not be able to watch the agents while they are training**, and you should set `train_mode=True` to restart the environment.

```
In [5]: env_info = env.reset(train_mode=True)[brain_name]      # reset the environment
        states = env_info.vector_observations                  # get the current state (for each
        scores = np.zeros(num_agents)                          # initialize the score (for each
        while True:
            actions = np.random.randn(num_agents, action_size) # select an action (for each agen
            actions = np.clip(actions, -1, 1)                  # all actions between -1 and 1
            env_info = env.step(actions)[brain_name]           # send all actions to tne environ
            next_states = env_info.vector_observations         # get next state (for each agent)
            rewards = env_info.rewards                         # get reward (for each agent)
            dones = env_info.local_done                        # see if episode finished
            scores += env_info.rewards                         # update the score (for each agen
            states = next_states                               # roll over states to next time s
            if np.any(dones):                                  # exit loop if episode finished
                break
        print('Total score (averaged over agents) this episode: {}'.format(np.mean(scores)))
```

```
Total score (averaged over agents) this episode: 0.11099999751895666
```

When finished, you can close the environment.

```
In [6]: env.close()
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

### 1.0.4 4. It's Your Turn!

Now it's your turn to train your own agent to solve the environment! A few **important notes**: - When training the environment, set `train_mode=True`, so that the line for resetting the environment looks like the following:

```
env_info = env.reset(train_mode=True)[brain_name]
```

- To structure your work, you're welcome to work directly in this Jupyter notebook, or you might like to start over with a new file! You can see the list of files in the workspace by clicking on ***Jupyter*** in the top left corner of the notebook.
- In this coding environment, you will not be able to watch the agents while they are training. However, ***after training the agents***, you can download the saved model weights to watch the agents on your own machine!

In [6]:
```
!pip install torchsummary
```

```
Collecting torchsummary
  Downloading https://files.pythonhosted.org/packages/7d/18/1474d06f721b86e6a9b9d7392ad68bed711a
Installing collected packages: torchsummary
Successfully installed torchsummary-1.5.1
```

In [7]:
```
# reset the environment
env_info = env.reset(train_mode=True)[brain_name]

# number of agents
num_agents = len(env_info.agents)
print('Number of agents:', num_agents)

# size of each action
action_size = brain.vector_action_space_size
print('Size of each action:', action_size)

# examine the state space
states = env_info.vector_observations
state_size = states.shape[1]
print('There are {} agents. Each observes a state with length: {}'.format(states.shape[0
print('The state for the first agent looks like:', states[0])
```

```
Number of agents: 20
Size of each action: 4
There are 20 agents. Each observes a state with length: 33
The state for the first agent looks like: [  0.00000000e+00  -4.00000000e+00   0.00000000e+00
  -0.00000000e+00  -0.00000000e+00  -4.37113883e-08   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00  -1.00000000e+01   0.00000000e+00
   1.00000000e+00  -0.00000000e+00  -0.00000000e+00  -4.37113883e-08
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   7.90150833e+00  -1.00000000e+00
   1.25147629e+00   0.00000000e+00   1.00000000e+00   0.00000000e+00
  -1.29508138e-01]
```

In [8]:
```
from collections import deque
import matplotlib.pyplot as plt
%matplotlib inline
```

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

# from ddpg_agent import Agent
from ddpg_agent import Agent
from torchsummary import summary
import time
plt.ion()


seed = 42

# Create agent
agent = Agent(state_size=state_size, action_size=action_size, n_agents=num_agents, seed=
```

In [9]:
```
def ddpg(n_episodes=2000, max_t = 1000, window_size=100, score_threshold=30.0,
         print_interval=10, epochs=1000):

    scores_deque = deque(maxlen=window_size)
    scores = []
    best_average_score = -np.inf
    print("Training on {} started...".format(agent.device))

    for i_episode in range(1, epochs+1):

        env_info = env.reset(train_mode=True)[brain_name]
        states = env_info.vector_observations

        agent.reset()
        episode_scores = np.zeros(num_agents)

        for t in range(max_t):
            actions = agent.act(states)
            env_info = env.step(actions)[brain_name]
            next_states = env_info.vector_observations
            rewards = env_info.rewards
            dones = env_info.local_done

            agent.step(states=states, actions=actions, rewards=rewards, next_states=next
            episode_scores += np.array(rewards)
            states = next_states
            if np.any(dones):
                break

        episode_score = np.mean(episode_scores)
```

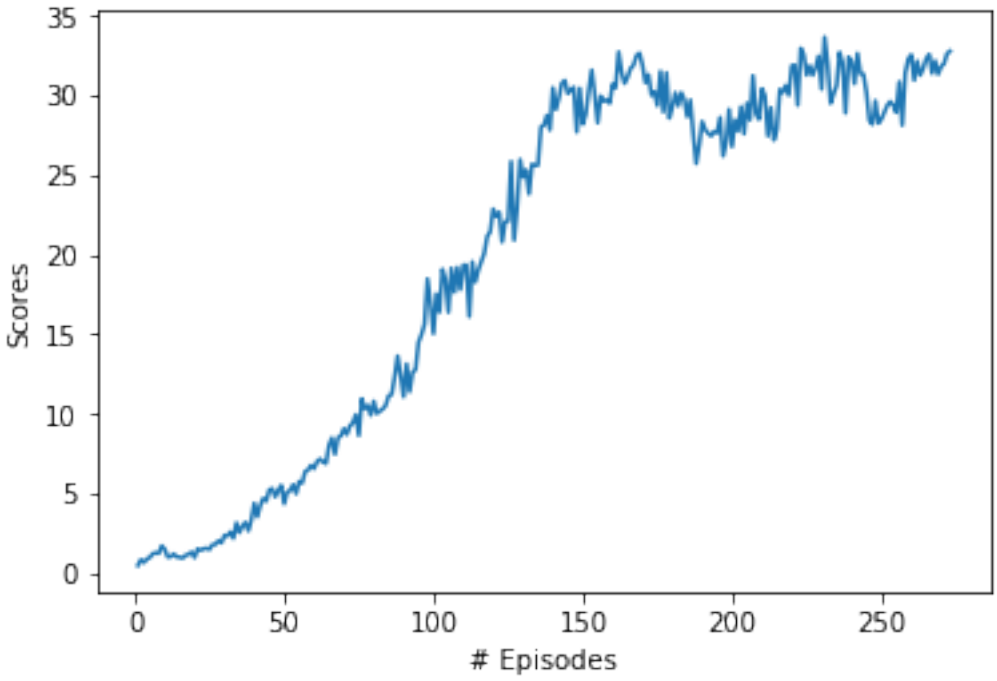
```
            scores_deque.append(episode_score)
            scores.append(episode_score)
            average_score = np.mean(scores_deque)

            print('\rEpisode: {}\tAverage Score: {:.2f}\tCurrent Score: {:.2f}'.format(i_epi
            if i_episode % print_interval == 0:
                print('\rEpisode: {}\tAverage Score: {:.2f}\tCurrent Score: {:.2f}'.format(i

            if average_score >= score_threshold:
                print('\nEnvironment solved in {} episodes!\tAverage Score: {:.2f}'.format(i
                torch.save(agent.actor_local.state_dict(), 'checkpoint_actor.pth')
                torch.save(agent.critic_local.state_dict(), 'checkpoint_critic.pth')
                break

        np.save('scores.npy', scores)
        return scores
```

```
In [10]: scores = ddpg()
```

```
Training on cuda:0 started...
Episode: 10        Average Score: 1.07        Current Score: 1.48
Episode: 20        Average Score: 1.07        Current Score: 0.97
Episode: 30        Average Score: 1.29        Current Score: 2.36
Episode: 40        Average Score: 1.70        Current Score: 4.38
Episode: 50        Average Score: 2.31        Current Score: 4.35
Episode: 60        Average Score: 2.89        Current Score: 6.55
Episode: 70        Average Score: 3.60        Current Score: 9.07
Episode: 80        Average Score: 4.38        Current Score: 10.79
Episode: 90        Average Score: 5.15        Current Score: 11.07
Episode: 100        Average Score: 6.09        Current Score: 15.04
Episode: 110        Average Score: 7.79        Current Score: 19.34
Episode: 120        Average Score: 9.65        Current Score: 22.86
Episode: 130        Average Score: 11.79        Current Score: 24.87
Episode: 140        Average Score: 14.19        Current Score: 30.48
Episode: 150        Average Score: 16.69        Current Score: 28.19
Episode: 160        Average Score: 19.09        Current Score: 30.70
Episode: 170        Average Score: 21.49        Current Score: 31.90
Episode: 180        Average Score: 23.52        Current Score: 29.28
Episode: 190        Average Score: 25.25        Current Score: 28.36
Episode: 200        Average Score: 26.56        Current Score: 26.78
Episode: 210        Average Score: 27.65        Current Score: 30.41
Episode: 220        Average Score: 28.62        Current Score: 31.86
Episode: 230        Average Score: 29.48        Current Score: 30.39
Episode: 240        Average Score: 29.92        Current Score: 32.14
Episode: 250        Average Score: 29.94        Current Score: 28.50
Episode: 260        Average Score: 29.97        Current Score: 32.55
Episode: 270        Average Score: 29.97        Current Score: 31.81
Episode: 273        Average Score: 30.02        Current Score: 32.78
```

```
Environment solved in 173 episodes!        Average Score: 30.02
```

```
In [11]: f = plt.figure()
         ax = f.add_subplot(111)
         plt.plot(range(1, len(scores) + 1), scores)
         plt.xlabel('# Episodes')
         plt.ylabel('Scores')
         plt.savefig('scores_plot.png')
         plt.show()
```




```
In [12]: env_info = env.reset(train_mode=False)[brain_name]     # reset the environment
         states = env_info.vector_observations                  # get the current state (for eac
         scores = np.zeros(num_agents)                          # initialize the score (for each
         while True:
             actions = np.random.randn(num_agents, action_size) # select an action (for each age
             actions = np.clip(actions, -1, 1)                  # all actions between -1 and 1
             env_info = env.step(actions)[brain_name]           # send all actions to tne enviro
             next_states = env_info.vector_observations         # get next state (for each agent
             rewards = env_info.rewards                         # get reward (for each agent)
             dones = env_info.local_done                        # see if episode finished
             scores += env_info.rewards                         # update the score (for each age
             states = next_states                               # roll over states to next time
             if np.any(dones):                                  # exit loop if episode finished
                 break
         print('Total score (averaged over agents) this episode: {}'.format(np.mean(scores)))
```

```
Total score (averaged over agents) this episode: 0.10149999773129821
```

In [13]:
```
env.close()
```

In [ ]:
```
```