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. 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 wh
ich is incompatible.
ipython 6.5.0 has requirement prompt-toolkit<2.0.0,>=1.0.15, but you'll have pr
ompt-toolkit 2.0.9 which is incompatible.
```

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_Li

# 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_size -> 5.0
                goal_speed -> 1.0
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]
```

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
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   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   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]
```

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

```
Total score (averaged over agents) this episode: 0.09949999777600169
```

When finished, you can close the environment.

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

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!

5. Import agent

In [7]:
```
from collections import deque
import matplotlib.pyplot as plt
%matplotlib inline

import torch
from ddpg_agent import Agent
```

6. Implement DDPG train function

In [8]:
```
def ddpg(n_episodes=300, max_t=1000, print_every=100):

    scores_deque = deque(maxlen=print_every)
    scores, ma_scores = [], []

    for i_episode in range(1, n_episodes+1):
        env_info = env.reset(train_mode=True)[brain_name]
        agent.reset()
        states = env_info.vector_observations            # get the current state
        episode_scores = np.zeros(num_agents)

        for t in range(max_t):
            actions = agent.act(states)          # select an action
            env_info = env.step(actions)[brain_name]        # send the action to
            next_states = env_info.vector_observations   # get the next state
            rewards = env_info.rewards                   # get the reward
            dones = env_info.local_done                  # see if episode has fi
            agent.step(states, actions, rewards, next_states, dones) # take step
            episode_scores += rewards                                # update th
            states = next_states                             # roll over the sta
            if np.any(dones):                                       # exit loop
                break

        scores_deque.append(np.mean(episode_scores))       # save most recent sc
        scores.append(np.mean(episode_scores))

        ma_scores.append(np.mean(scores_deque)) # moving average
        print('\rEpisode {}\tAverage Score: {:.2f}'.format(i_episode, np.mean(sc
        torch.save(agent.actor_local.state_dict(), 'checkpoint_actor.pth')
        torch.save(agent.critic_local.state_dict(), 'checkpoint_critic.pth')
        if i_episode % 10 == 0:
            print('\rEpisode {}\tAverage Score: {:.2f}'.format(i_episode, np.mea

        if np.mean(scores_deque) >=30.0 :
            print('\nEnvironment solved in {:d} episodes!\tAverage Score: {:.2f}
            torch.save(agent.actor_local.state_dict(), 'checkpoint_actor.pth')
            torch.save(agent.critic_local.state_dict(), 'checkpoint_critic.pth')
            break

    return scores, ma_scores
```

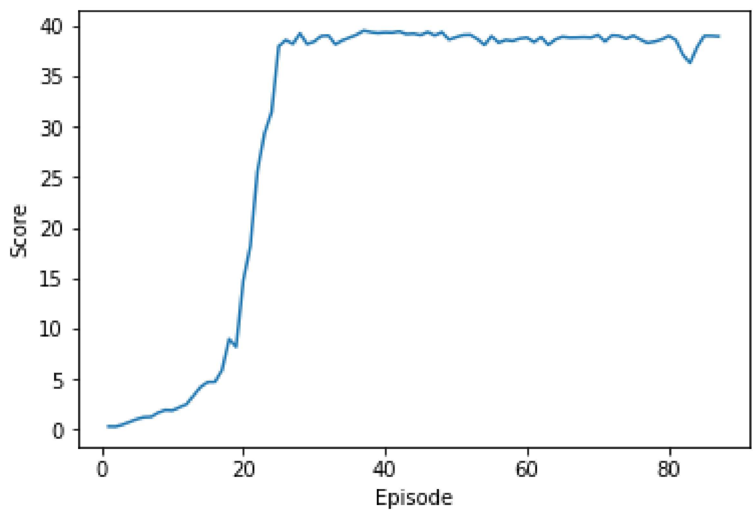
7. Create an Agent and train

In [9]:
```
random_seed = 10
agent = Agent(state_size=state_size, action_size=action_size, num_agents=num_agen
scores, ma_scores = ddpg(n_episodes=200, print_every=100)
```

```
Episode 10      Average Score: 1.09
Episode 20      Average Score: 3.52
Episode 30      Average Score: 13.52
Episode 40      Average Score: 19.89
Episode 50      Average Score: 23.74
Episode 60      Average Score: 26.23
Episode 70      Average Score: 28.02
Episode 80      Average Score: 29.36
Episode 87      Average Score: 30.06
Environment solved in 87 episodes!      Average Score: 30.06
```

8. Plot average score

In [10]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(np.arange(1, len(scores)+1), scores, label='scores per episode')
plt.ylabel('Score')
plt.xlabel('Episode')
plt.show()
```



In [11]:
```
env.close()
```