Continuous_Control

August 5, 2020

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_NoVis.

        # 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: 1
Size of each action: 4
There are 1 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
  5.55726671e+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: 0.13999999687075615
```

When finished, you can close the environment.

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

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 [11]:
```
from ddpg_agent import Agent
from collections import deque
import torch
```

In [15]:
```
# DDPG function

def ddpg(n_episodes=1000, max_t=1000, consec_episodes=100, print_every=1):
    """Deep Deterministic Policy Gradient (DDPG)

    Params
    ======
        n_episodes (int)      : maximum number of training episodes
        max_t (int)           : maximum number of timesteps per episode
        consec_episodes (int) : number of consecutive episodes used to calculate score
        print_every (int)     : interval to display results


    """

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

    for i_episode in range(1, n_episodes+1):
        env_info = env.reset(train_mode=True)[brain_name]
        agent.reset()
        state = env_info.vector_observations[0]            # get the current state
        score = 0


        #print('start')


        for t in range(max_t):
            action = agent.act(state, add_noise=True)          # select an action

            env_info = env.step(action)[brain_name]        # send the action to the env
            next_state = env_info.vector_observations[0]   # get the next state
            reward = env_info.rewards[0]                   # get the reward
            done = env_info.local_done[0]                  # see if episode has finished
            agent.step(state, action, reward, next_state, done) # take step with agent
            score += reward                                # update the score
            state = next_state                             # roll over the state to next
            if done:                                       # exit loop if episode finis
                break


        scores_deque.append(score)       # save most recent score
        scores.append(score)             # save most recent score
```

```
                #print('end')
                if i_episode % print_every == 0:
                    print('\rEpisode {}\tAverage Score: {:.2f}'.format(i_episode, np.mean(score
                    torch.save(agent.actor_local.state_dict(), 'checkpoint_actor.pth')
                    torch.save(agent.critic_local.state_dict(), 'checkpoint_critic.pth')

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

            return scores
```

```
In [16]: # run the training loop
         agent = Agent(state_size=state_size, action_size=action_size, random_seed=11)
```

```
In [17]: scores = ddpg()
```

```
Episode 1        Average Score: 0.00
Episode 2        Average Score: 0.32
Episode 3        Average Score: 0.00
Episode 4        Average Score: 0.33
Episode 5        Average Score: 0.80
Episode 6        Average Score: 0.00
Episode 7        Average Score: 0.00
Episode 8        Average Score: 0.76
Episode 9        Average Score: 0.26
Episode 10        Average Score: 0.00
Episode 11        Average Score: 0.93
Episode 12        Average Score: 0.40
Episode 13        Average Score: 1.21
Episode 14        Average Score: 1.31
Episode 15        Average Score: 0.49
Episode 16        Average Score: 1.06
Episode 17        Average Score: 1.21
Episode 18        Average Score: 1.29
Episode 19        Average Score: 0.10
Episode 20        Average Score: 1.96
Episode 21        Average Score: 2.41
Episode 22        Average Score: 1.62
Episode 23        Average Score: 0.94
Episode 24        Average Score: 2.00
Episode 25        Average Score: 0.92
Episode 26        Average Score: 2.86
```

```
Episode 27        Average Score: 1.55
Episode 28        Average Score: 2.95
Episode 29        Average Score: 2.49
Episode 30        Average Score: 1.90
Episode 31        Average Score: 2.24
Episode 32        Average Score: 2.72
Episode 33        Average Score: 4.87
Episode 34        Average Score: 2.72
Episode 35        Average Score: 2.78
Episode 36        Average Score: 2.50
Episode 37        Average Score: 3.77
Episode 38        Average Score: 3.69
Episode 39        Average Score: 2.96
Episode 40        Average Score: 6.06
Episode 41        Average Score: 4.31
Episode 42        Average Score: 3.17
Episode 43        Average Score: 2.15
Episode 44        Average Score: 3.86
Episode 45        Average Score: 4.60
Episode 46        Average Score: 3.36
Episode 47        Average Score: 2.50
Episode 48        Average Score: 4.37
Episode 49        Average Score: 2.35
Episode 50        Average Score: 3.19
Episode 51        Average Score: 3.89
Episode 52        Average Score: 6.74
Episode 53        Average Score: 4.66
Episode 54        Average Score: 3.46
Episode 55        Average Score: 6.17
Episode 56        Average Score: 2.35
Episode 57        Average Score: 5.41
Episode 58        Average Score: 4.47
Episode 59        Average Score: 6.83
Episode 60        Average Score: 2.56
Episode 61        Average Score: 2.56
Episode 62        Average Score: 6.95
Episode 63        Average Score: 8.61
Episode 64        Average Score: 3.39
Episode 65        Average Score: 5.62
Episode 66        Average Score: 8.78
Episode 67        Average Score: 11.38
Episode 68        Average Score: 6.07
Episode 69        Average Score: 4.34
Episode 70        Average Score: 7.05
Episode 71        Average Score: 6.25
Episode 72        Average Score: 6.44
Episode 73        Average Score: 10.91
Episode 74        Average Score: 6.67
```

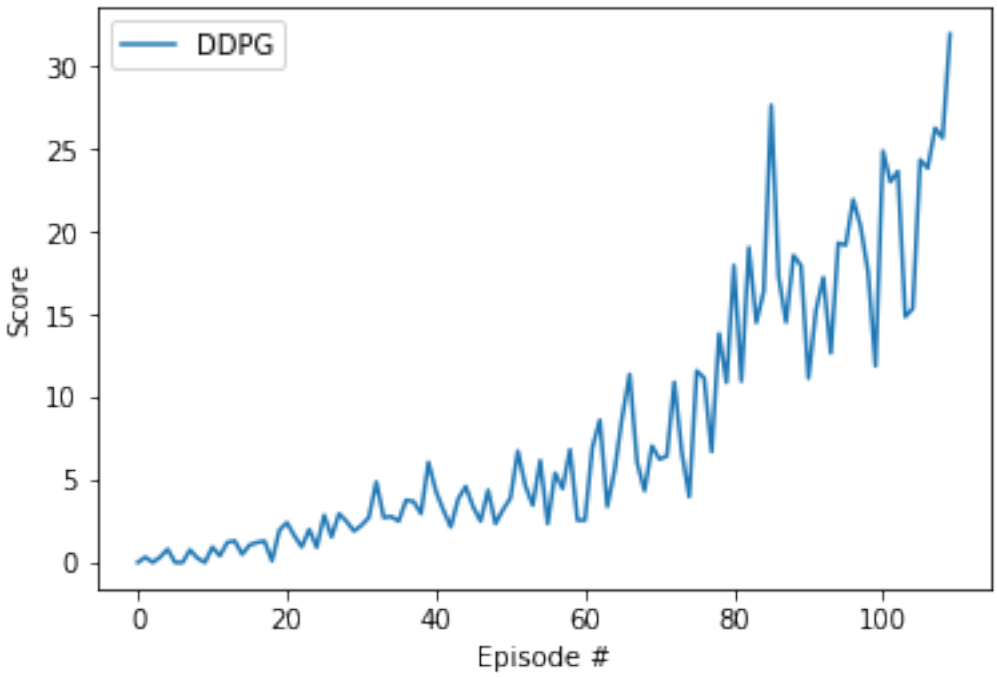
```
Episode 75        Average Score: 3.98
Episode 76        Average Score: 11.59
Episode 77        Average Score: 11.16
Episode 78        Average Score: 6.70
Episode 79        Average Score: 13.85
Episode 80        Average Score: 10.89
Episode 81        Average Score: 17.97
Episode 82        Average Score: 10.97
Episode 83        Average Score: 19.06
Episode 84        Average Score: 14.52
Episode 85        Average Score: 16.42
Episode 86        Average Score: 27.67
Episode 87        Average Score: 17.28
Episode 88        Average Score: 14.54
Episode 89        Average Score: 18.58
Episode 90        Average Score: 17.96
Episode 91        Average Score: 11.15
Episode 92        Average Score: 15.24
Episode 93        Average Score: 17.25
Episode 94        Average Score: 12.67
Episode 95        Average Score: 19.33
Episode 96        Average Score: 19.20
Episode 97        Average Score: 21.96
Episode 98        Average Score: 20.34
Episode 99        Average Score: 17.58
Episode 100        Average Score: 11.89
Episode 101        Average Score: 24.89
Episode 102        Average Score: 23.03
Episode 103        Average Score: 23.67
Episode 104        Average Score: 14.87
Episode 105        Average Score: 15.36
Episode 106        Average Score: 24.36
Episode 107        Average Score: 23.86
Episode 108        Average Score: 26.28
Episode 109        Average Score: 25.69
Episode 110        Average Score: 31.97

Environment solved in 110 episodes!        Average Score: 31.97
```

```
In [18]: import numpy as np
         import matplotlib.pyplot as plt
         %matplotlib inline

         # plot the scores
         fig = plt.figure()
         ax = fig.add_subplot(111)
         plt.plot(np.arange(len(scores)), scores, label='DDPG')
```

```
plt.ylabel('Score')
plt.xlabel('Episode #')
plt.legend(loc='upper left');
plt.show()
```



In [19]:
```
env.close()
```

In [ ]: