Continuous_Control

December 2, 2019

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 2.0.
```

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.

```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.

```python
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
```

2

```
   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.

```python
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: 0.0
```

When finished, you can close the environment.

```python
In [6]: # 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:

```python
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!

3

```python
In [7]: import matplotlib.pyplot as plt
        %matplotlib inline

        import numpy as np
        import random
        import time
        import torch

        from collections import deque
        from workspace_utils import active_session
        from ddpgAgent import Agent
        from unityagents import UnityEnvironment

In [8]: seed = 5

In [9]: # DDPG Function

        def ddpg(n_episodes = 2500, max_t = 1000, print_every = 10):

            mean_scores = []
            moving_avgs = []
            best_score = -np.inf
            scores_window = deque(maxlen=100)


            for i_episode in range(1, n_episodes+1):
                env_info = env.reset(train_mode= True)[brain_name]
                states = env_info.vector_observations
                scores = np.zeros(num_agents)
                agent.reset()
                start_time = time.time()
                for t in range(max_t):
                    actions = agent.act(states, add_noise=True)
                    env_info = env.step(actions)[brain_name]
                    next_states = env_info.vector_observations
                    rewards = env_info.rewards
                    dones = env_info.local_done

                    for state, action, reward, next_state, done in zip(states, actions, rewards,
                        agent.step(state,action,reward, next_state,done,t)
                    states = next_states
                    scores += rewards
                    if np.any(dones):
                        break
                duration = time.time() - start_time
                mean_scores.append(np.mean(scores))
                scores_window.append(mean_scores[-1])
                moving_avgs.append(np.mean(scores_window))
```

```
            if i_episode % print_every == 0:
                print('\rEpisode {} ({}s)\tMean: {:.1f}\tMoving Avg: {:.1f}'.format(\
                    i_episode, round(duration), mean_scores[-1], moving_avgs[-1]))

            if moving_avgs[-1] >= 30.00 and i_episode >= 100:
                print('\nEnvironment solved in {:d} episodes!\tAverage Score: {:.2f}'.format
                break

        return mean_scores, moving_avgs

In [10]: state = time.time()

In [11]: agent = Agent(state_size = state_size, action_size = action_size, seed = seed)
         with active_session():
             scores, avgs = ddpg()

Episode 10 (9s)        Mean: 0.2        Moving Avg: 0.3
Episode 20 (9s)        Mean: 1.3        Moving Avg: 0.5
Episode 30 (9s)        Mean: 1.2        Moving Avg: 0.7
Episode 40 (9s)        Mean: 1.7        Moving Avg: 0.8
Episode 50 (9s)        Mean: 2.5        Moving Avg: 1.1
Episode 60 (9s)        Mean: 2.6        Moving Avg: 1.3
Episode 70 (9s)        Mean: 4.4        Moving Avg: 1.5
Episode 80 (9s)        Mean: 4.3        Moving Avg: 1.7
Episode 90 (9s)        Mean: 7.4        Moving Avg: 2.0
Episode 100 (9s)        Mean: 7.8        Moving Avg: 2.5
Episode 110 (9s)        Mean: 6.4        Moving Avg: 3.0
Episode 120 (9s)        Mean: 4.3        Moving Avg: 3.5
Episode 130 (9s)        Mean: 7.2        Moving Avg: 4.2
Episode 140 (9s)        Mean: 15.5        Moving Avg: 5.2
Episode 150 (9s)        Mean: 6.2        Moving Avg: 5.9
Episode 160 (9s)        Mean: 10.4        Moving Avg: 6.7
Episode 170 (9s)        Mean: 15.1        Moving Avg: 7.6
Episode 180 (9s)        Mean: 12.6        Moving Avg: 8.3
Episode 190 (9s)        Mean: 14.1        Moving Avg: 9.2
Episode 200 (9s)        Mean: 12.9        Moving Avg: 10.0
Episode 210 (9s)        Mean: 16.7        Moving Avg: 10.8
Episode 220 (9s)        Mean: 12.7        Moving Avg: 11.8
Episode 230 (9s)        Mean: 14.4        Moving Avg: 12.5
Episode 240 (9s)        Mean: 23.1        Moving Avg: 13.4
Episode 250 (9s)        Mean: 22.5        Moving Avg: 14.6
Episode 260 (9s)        Mean: 26.5        Moving Avg: 15.9
Episode 270 (9s)        Mean: 19.6        Moving Avg: 16.8
Episode 280 (10s)        Mean: 25.7        Moving Avg: 18.1
Episode 290 (10s)        Mean: 17.2        Moving Avg: 19.1
Episode 300 (9s)        Mean: 31.3        Moving Avg: 20.1
Episode 310 (10s)        Mean: 22.3        Moving Avg: 21.3
```

```
Episode 320 (10s)        Mean: 28.2        Moving Avg: 22.6
Episode 330 (10s)        Mean: 22.0        Moving Avg: 23.6
Episode 340 (10s)        Mean: 33.8        Moving Avg: 24.6
Episode 350 (10s)        Mean: 33.7        Moving Avg: 25.2
Episode 360 (10s)        Mean: 28.7        Moving Avg: 25.8
Episode 370 (10s)        Mean: 33.3        Moving Avg: 26.9
Episode 380 (10s)        Mean: 27.8        Moving Avg: 27.7
Episode 390 (10s)        Mean: 31.5        Moving Avg: 28.6
Episode 400 (10s)        Mean: 34.1        Moving Avg: 29.3
Episode 410 (10s)        Mean: 29.1        Moving Avg: 29.9


Environment solved in 311 episodes!        Average Score: 30.05
```
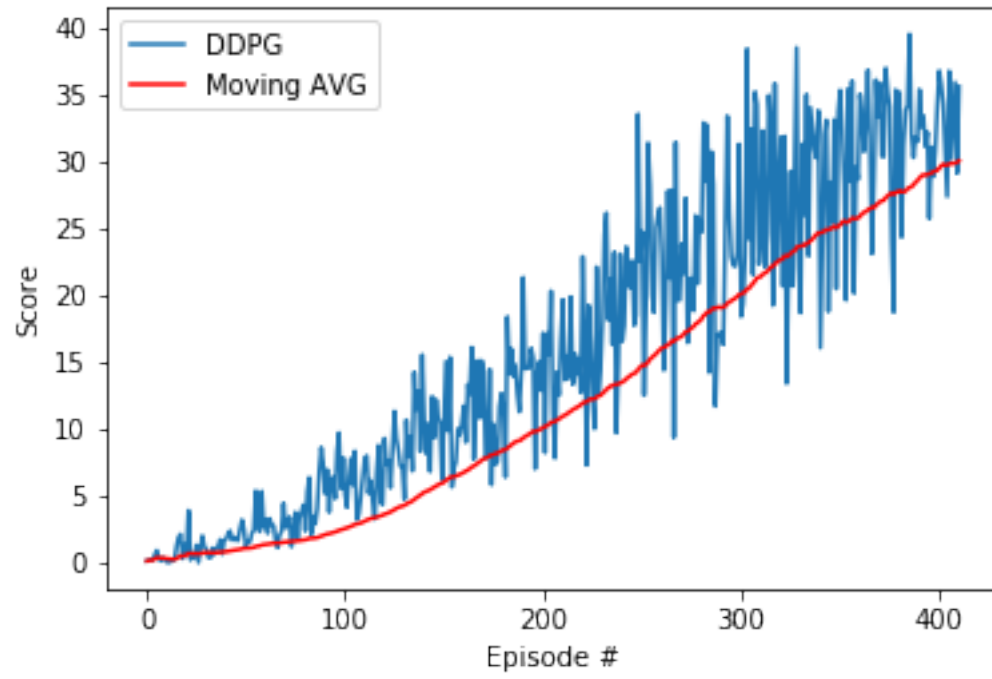
```python
In [13]: end = time.time()
         elapsed = (end - state) / 60.0 # in minutes
         print("\nElapsed Time: {0:3.2f} mins.".format(elapsed))
```

```
Elapsed Time: 64.34 mins.
```

```python
In [14]: # plot the scores
         fig = plt.figure()
         ax = fig.add_subplot(111)
         plt.plot(np.arange(len(scores)), scores, label='DDPG')
         plt.plot(np.arange(len(scores)), avgs, c='r', label='Moving AVG')
         plt.ylabel('Score')
         plt.xlabel('Episode #')
         plt.legend(loc='upper left');
         plt.show()
```

In [15]: env.close()

In [ ]: