Continuous_Control

July 5, 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_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]
```

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 [ ]: 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)))
```

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 [5]: # 1. Import the Necessary Packages
        from ddpg_agent import Agent
```

```
from collections import deque
import torch
import matplotlib.pyplot as plt
%matplotlib inline

# 2. Instantiate the Agent
agent = Agent(state_size=state_size, action_size=action_size, random_seed=0)
```

In [6]:
```
# 3. Train the Agent with DDPG
def ddpg(n_episodes=1000, max_t=1000, print_every=10):

    scores_deque = deque(maxlen=100)
    scores = []

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

        for step in range(max_t):
            actions = agent.act(states)
            env_info = env.step(actions)[brain_name]           # send all actions to tne
            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
            agent.step(states, actions, rewards, next_states, dones)
            score += rewards                           # update the score (for each agent
            states = next_states                               # roll over states to nex
            if np.any(dones):                                  # exit loop if episode fi
                break

        scores_deque.append(np.mean(score))
        scores.append(np.mean(score))
        avg_score = np.mean(scores_deque)

        print('\rEpisode {}\tAverage Score: {:.2f}\tEpisode Score: {:.2f}'.format(i_epis
        if i_episode % print_every == 0:
            print('\rEpisode {}\tAverage Score: {:.2f}\tEpisode Score: {:.2f}'.format(i_

        if avg_score > 30:
            # save the model
            torch.save(agent.actor_local.state_dict(), 'checkpoint_actor.pth')
            torch.save(agent.critic_local.state_dict(), 'checkpoint_critic.pth')
            print('\rEnvironment solved in {:d} episodes'.format(i_episode))
            break
```

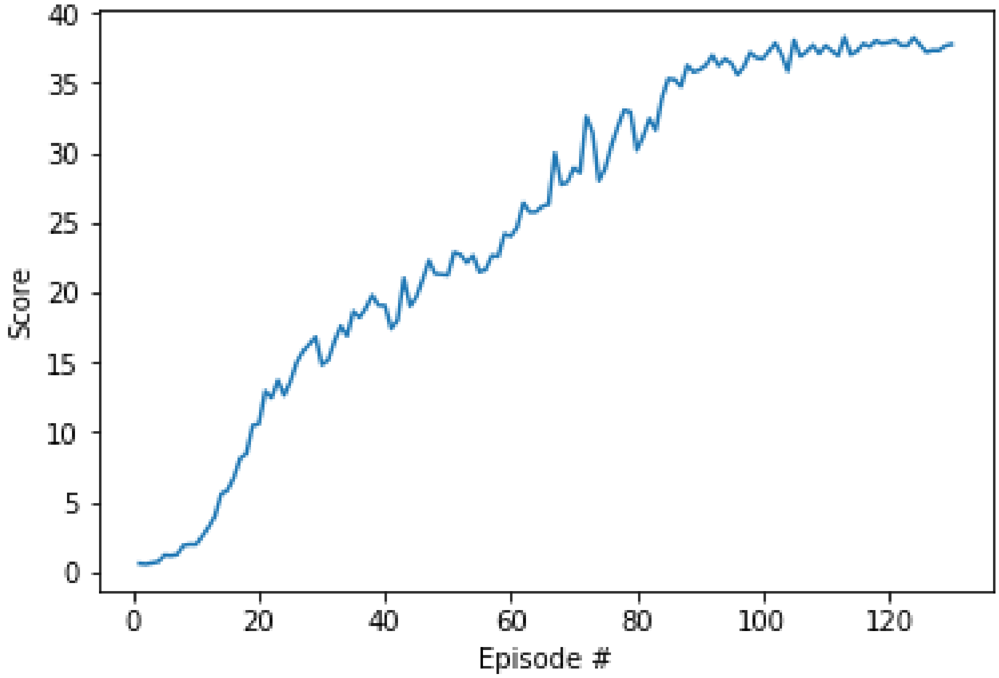
```
    return scores

scores = ddpg()

fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(np.arange(1, len(scores)+1), scores)
plt.ylabel('Score')
plt.xlabel('Episode #')
plt.show()
```

```
/home/workspace/ddpg_agent.py:113: UserWarning: torch.nn.utils.clip_grad_norm is now deprecated
  torch.nn.utils.clip_grad_norm(self.critic_local.parameters(), 1)


Episode 10		Average Score: 1.19		Episode Score: 1.96
Episode 20		Average Score: 3.87		Episode Score: 10.60
Episode 30		Average Score: 7.39		Episode Score: 14.82
Episode 40		Average Score: 10.04		Episode Score: 19.07
Episode 50		Average Score: 12.07		Episode Score: 21.24
Episode 60		Average Score: 13.84		Episode Score: 24.01
Episode 70		Average Score: 15.72		Episode Score: 28.91
Episode 80		Average Score: 17.60		Episode Score: 30.17
Episode 90		Average Score: 19.45		Episode Score: 35.95
Episode 100		Average Score: 21.16		Episode Score: 36.73
Episode 110		Average Score: 24.76		Episode Score: 37.66
Episode 120		Average Score: 27.87		Episode Score: 37.90
Episode 130		Average Score: 30.19		Episode Score: 37.76
Environment solved in 130 episodes
```



In [ ]: