Continuous_Control

September 18, 2022

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.
jupyter-console 6.4.3 has requirement jupyter-client>=7.0.0, but you'll have jupyter-client 5.2.
```

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]
```

```
In [4]: brain_name
```

```
Out[4]: 'ReacherBrain'
```

1.0.2 2. Examine the State and Action Spaces

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

```
In [5]: # 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 [6]: 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: 0.23499999474734068
```

1.0.4 4. Actor-Critic (DDPG Algorithm)

Train model

```
In [7]: import torch
        from matplotlib import pyplot as plt
        from agent import Agent
        from utils import ddpg_train

        DEVICE = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

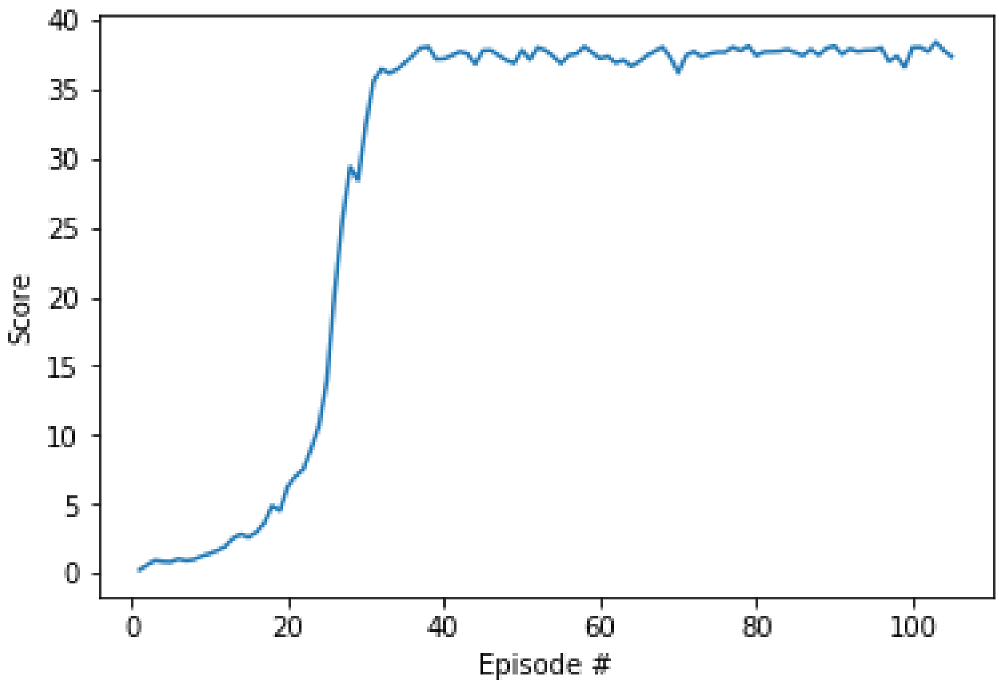
```
In [10]: BUFFER_SIZE = int(1e5)  # replay buffer size
         BATCH_SIZE = 128        # minibatch size
```

```
GAMMA = 0.99            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR_ACTOR = 1e-4         # learning rate of the actor
LR_CRITIC = 1e-4        # learning rate of the critic
WEIGHT_DECAY = 0        # L2 weight decay

CHECKPOINT_FOLDER = './'


agent = Agent(device=DEVICE,
              state_size=state_size,
              n_agents=num_agents,
              action_size=action_size,
              random_seed=123,
              buffer_size=BUFFER_SIZE,
              batch_size=BATCH_SIZE,
              gamma=GAMMA,
              tau=TAU,
              lr_actor=LR_ACTOR,
              lr_critic=LR_CRITIC,
              weight_decay=WEIGHT_DECAY,
              checkpoint_folder=CHECKPOINT_FOLDER,
              restore=False)  # False - Initialize and train agent from scratch! (True
```

In [11]:
```
%%time

# train the agent
scores = ddpg_train(agent=agent, env=env, n_episodes=1000, max_t=1000, print_every=5, m
```

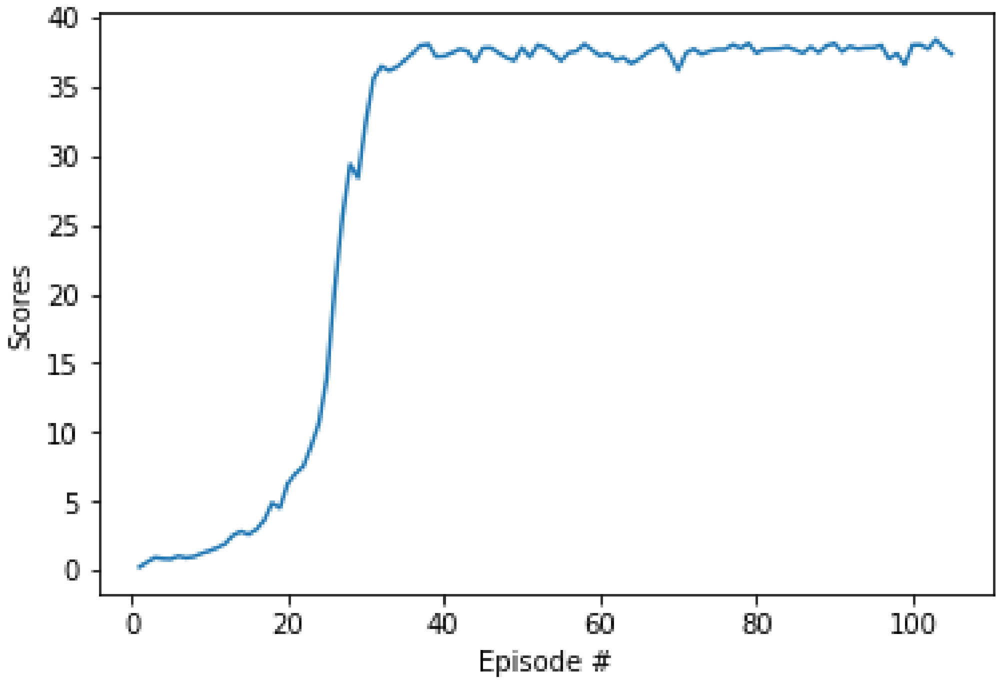
```
Episode:        0        Score:        0.24        Average Score:        0.24ot:
Episode:        5        Score:        1.04        Average Score:        0.75it]INFO:root:
Episode:        10        Score:        1.65        Average Score:        0.97it]INFO:root:
Episode:        15        Score:        2.97        Average Score:        1.48it]INFO:root:
Episode:        20        Score:        7.02        Average Score:        2.38it]INFO:root:
Episode:        25        Score:        20.21        Average Score:        4.28t]INFO:root:
Episode:        30        Score:        35.60        Average Score:        8.47t]INFO:root:
Episode:        35        Score:        37.41        Average Score:        12.39]INFO:root:
Episode:        40        Score:        37.43        Average Score:        15.46]INFO:root:
Episode:        45        Score:        37.81        Average Score:        17.86]INFO:root:
Episode:        50        Score:        37.13        Average Score:        19.76]INFO:root:
Episode:        55        Score:        37.45        Average Score:        21.34]INFO:root:
Episode:        60        Score:        37.37        Average Score:        22.67]INFO:root:
Episode:        65        Score:        37.44        Average Score:        23.76]INFO:root:
Episode:        70        Score:        37.43        Average Score:        24.72]INFO:root:
Episode:        75        Score:        37.67        Average Score:        25.56]INFO:root:
Episode:        80        Score:        37.69        Average Score:        26.32]INFO:root:
Episode:        85        Score:        37.42        Average Score:        26.98]INFO:root:
Episode:        90        Score:        37.53        Average Score:        27.57]INFO:root:
```

```
Episode:         95         Score:         37.98         Average Score:         28.11]INFO:root:
Episode:         100         Score:         38.02         Average Score:         28.85]INFO:root
 10%|          | 104/1000 [25:08<3:32:48, 14.25s/it]INFO:root:
Environment solved in 104 episodes!        Average Score: 30.33
```




```
CPU times: user 20min 30s, sys: 59.3 s, total: 21min 29s
Wall time: 25min 23s
```

In [12]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(np.arange(1, len(scores)+1), scores)
plt.ylabel('Scores')
plt.xlabel('Episode #')
plt.show()
```




```
In [13]: fig.savefig('rewards.png', dpi=fig.dpi)
```

Test model I'll just print 10 episode scores using the pre-trained model:

```
In [14]: # Test the trained agent

         agent = Agent(device=DEVICE, state_size=state_size, n_agents=num_agents, action_size=ac
                       batch_size=BATCH_SIZE, gamma=GAMMA, tau=TAU, lr_actor=LR_ACTOR, lr_critic
                       restore=True)   # restore existing checkpoint

         for episode in range(10):
             env_info = env.reset(train_mode=False)[brain_name]
             states = env_info.vector_observations
             score = np.zeros(num_agents)

             while True:
                 actions = agent.act(states, add_noise=False)

                 env_info = env.step(actions)[brain_name]
                 next_states = env_info.vector_observations
                 rewards = env_info.rewards
                 dones = env_info.local_done
                 score += rewards
                 states = next_states
```

```
            if np.any(dones):
                break

        print('Episode: \t{} \tScore: \t{:.2f}'.format(episode, np.mean(score)))
```

```
INFO:root:Restoring from checkpoint


Episode:        0         Score:          39.54
Episode:        1         Score:          39.54
Episode:        2         Score:          39.47
Episode:        3         Score:          39.42
Episode:        4         Score:          39.47
Episode:        5         Score:          39.44
Episode:        6         Score:          39.46
Episode:        7         Score:          39.53
Episode:        8         Score:          39.49
Episode:        9         Score:          39.52
```

```
In [ ]: print("DONE")
```

1.0.5 Close the environment

env.close()