Continuous_Control

November 24, 2019

1 Continuous Control

In this notebook, you will learn how to use the Unity ML-Agents environment for the second project of the Deep Reinforcement Learning Nanodegree program.

1.0.1 1. Start the Environment

We begin by importing the necessary packages. If the code cell below returns an error, please revisit the project instructions to double-check that you have installed Unity ML-Agents and NumPy.

```
In [2]: !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.
```

```
In [3]: import numpy as np
        import torch
        import matplotlib.pyplot as plt
        import time

        from unityagents import UnityEnvironment
        from collections import deque
        from itertools import count
        import datetime

        from ddpg import DDPG, ReplayBuffer

        %load_ext autoreload
        %autoreload 2
        %matplotlib inline
```

Next, we will start the environment! ***Before running the code cell below***, change the `file_name` parameter to match the location of the Unity environment that you downloaded.

- **Mac**: `"path/to/Reacher.app"`

- **Windows** (x86): "path/to/Reacher_Windows_x86/Reacher.exe"
- **Windows** (x86_64): "path/to/Reacher_Windows_x86_64/Reacher.exe"
- **Linux** (x86): "path/to/Reacher_Linux/Reacher.x86"
- **Linux** (x86_64): "path/to/Reacher_Linux/Reacher.x86_64"
- **Linux** (x86, headless): "path/to/Reacher_Linux_NoVis/Reacher.x86"
- **Linux** (x86_64, headless): "path/to/Reacher_Linux_NoVis/Reacher.x86_64"

For instance, if you are using a Mac, then you downloaded `Reacher.app`. If this file is in the same folder as the notebook, then the line below should appear as follows:

```
env = UnityEnvironment(file_name="Reacher.app")
```

```
In [4]: #env = UnityEnvironment(file_name='envs/Reacher_Linux_NoVis_20/Reacher.x86_64') # Headle
        #env = UnityEnvironment(file_name='envs/Reacher_Linux_20/Reacher.x86_64') # Visual
        env = UnityEnvironment(file_name="/data/Reacher_Linux_NoVis/Reacher")
```

```
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 [5]: # 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

In this environment, a double-jointed arm can move to target locations. A reward of `+0.1` is provided for each step that the agent's hand is in the goal location. Thus, the goal of your agent is to maintain its position at the target location for as many time steps as possible.

The observation space consists of 33 variables corresponding to position, rotation, velocity, and angular velocities of the arm. Each action is a vector with four numbers, corresponding to torque applicable to two joints. Every entry in the action vector must be a number between -1 and 1.

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

```
In [6]: # 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.

Once this cell is executed, you will watch the agent's performance, if it selects an action at random with each time step. A window should pop up that allows you to observe the agent, as it moves through the environment.

Of course, as part of the project, you'll have to change the code so that the agent is able to use its experience to gradually choose better actions when interacting with the environment!

```
In [7]: env_info = env.reset(train_mode=False)[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 step
        if np.any(dones):                                  # exit loop if episode finished
            break
        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.

1.0.4 4. It's Your Turn!

Now it's your turn to train your own agent to solve the environment! 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]
```

```
In [8]: avg_over = 100
        print_every = 10

        def ddpg(agent, n_episodes=200, stopOnSolved=True):
            print('Start: ',datetime.datetime.now())
            scores_deque = deque(maxlen=avg_over)
            scores_global = []
            average_global = []
            min_global = []
            max_global = []
            best_avg = -np.inf

            tic = time.time()

            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 (
                scores = np.zeros(num_agents)                          # initialize the score (f
                agent.reset()

                score_average = 0
```

```
        timestep = time.time()
        for t in count():
            actions = agent.act(states, add_noise=True)
            env_info = env.step(actions)[brain_name]            # send all actions to tn
            next_states = env_info.vector_observations          # get next state (for ea
            rewards = env_info.rewards                          # get reward (for each a
            dones = env_info.local_done                         # see if episode finishe
            agent.step(states, actions, rewards, next_states, dones) # store experience
            states = next_states                                # roll over states to ne
            scores += rewards                                   # update the score (for
            if np.any(dones):                                   # exit loop if episode f
                break

        score = np.mean(scores)
        scores_deque.append(score)
        score_average = np.mean(scores_deque)
        scores_global.append(score)
        average_global.append(score_average)
        min_global.append(np.min(scores))
        max_global.append(np.max(scores))
        print('\r {}, {:.2f}, {:.2f}, {:.2f}, {:.2f}, {:.2f}'\
              .format(str(i_episode).zfill(3), score, score_average, np.max(scores),
                      np.min(scores), time.time() - timestep), end="\n")
        if i_episode % print_every == 0:
            agent.save('./')
        if  stopOnSolved and score_average >= 30.0:
            toc = time.time()
            print('\nSolved in {:d} episodes!\tAvg Score: {:.2f}, time: {}'.format(i_epi
            agent.save('./'+str(i_episode)+'_')
            break

    print('End: ',datetime.datetime.now())
    return scores_global, average_global, max_global, min_global

ddpg
```

Out[8]: <function __main__.ddpg(agent, n_episodes=200, stopOnSolved=True)>

In [8]:
```
agent = DDPG(state_size=state_size, action_size=action_size, CER=False)
scores, averages, maxima, minima = ddpg(agent, n_episodes=200)

plt.plot(np.arange(1, len(averages)+1), averages)
plt.plot(np.arange(1, len(scores)+1), scores)
plt.plot(np.arange(1, len(maxima)+1), maxima)
plt.plot(np.arange(1, len(minima)+1), minima)
plt.title('Learning without CER')
plt.ylabel('Score')
plt.xlabel('Episode #')
```

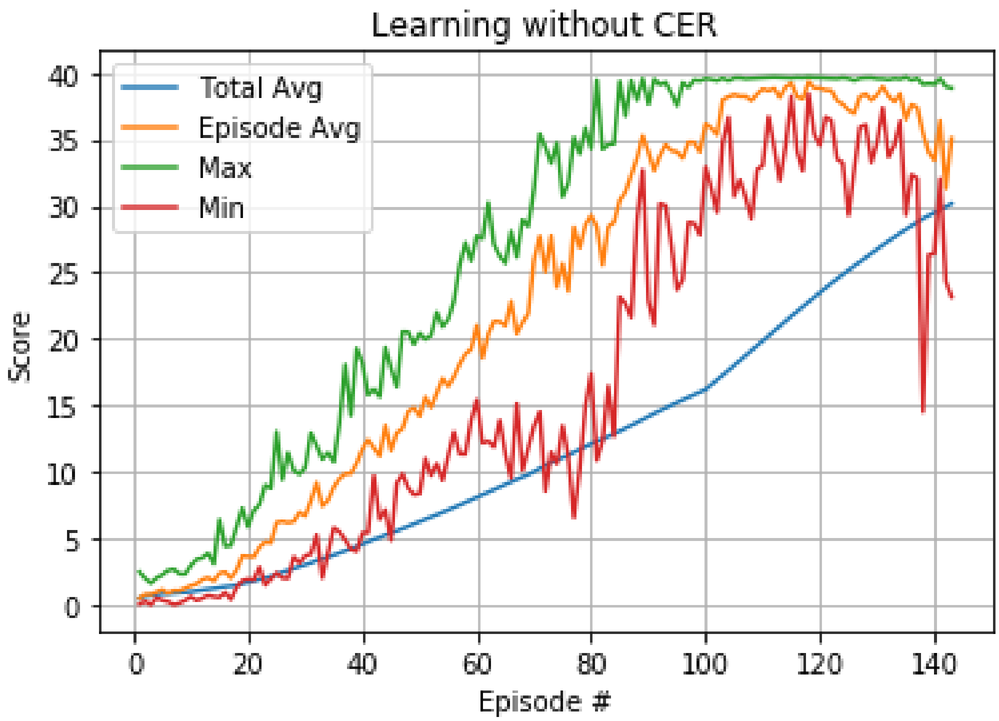
```python
        plt.legend(['Total Avg', 'Episode Avg', 'Max', 'Min'], loc='upper left')
        plt.grid()
        plt.show()
```

```
Start:  2019-11-19 21:38:07.228439
 001, 0.48, 0.48, 2.46, 0.05, 16.86
 002, 0.78, 0.63, 1.98, 0.25, 17.21
 003, 0.77, 0.68, 1.62, 0.00, 17.22
 004, 0.95, 0.74, 2.01, 0.51, 17.23
 005, 1.13, 0.82, 2.24, 0.33, 17.33
 006, 0.86, 0.83, 2.58, 0.23, 17.51
 007, 1.06, 0.86, 2.67, 0.00, 17.54
 008, 1.05, 0.89, 2.32, 0.12, 17.63
 009, 1.23, 0.92, 2.30, 0.33, 17.77
 010, 1.43, 0.97, 2.95, 0.60, 17.96
 011, 1.56, 1.03, 3.41, 0.32, 18.13
 012, 1.89, 1.10, 3.52, 0.48, 18.31
 013, 2.03, 1.17, 3.86, 0.70, 18.68
 014, 1.79, 1.22, 3.08, 0.60, 18.82
 015, 2.36, 1.29, 6.42, 0.54, 19.15
 016, 2.46, 1.36, 4.39, 0.92, 19.35
 017, 2.01, 1.40, 4.45, 0.39, 19.72
 018, 2.60, 1.47, 5.88, 1.30, 19.93
 019, 3.67, 1.58, 7.28, 1.81, 20.51
 020, 3.62, 1.69, 5.87, 1.92, 21.11
 021, 3.60, 1.78, 7.08, 1.85, 20.85
 022, 4.32, 1.89, 7.53, 2.83, 21.14
 023, 4.72, 2.02, 9.00, 1.47, 21.58
 024, 4.65, 2.13, 8.75, 1.94, 21.92
 025, 6.19, 2.29, 13.02, 2.39, 22.28
 026, 6.28, 2.44, 9.41, 2.01, 22.85
 027, 6.18, 2.58, 11.42, 1.99, 23.18
 028, 6.26, 2.71, 10.11, 3.60, 23.64
 029, 6.95, 2.86, 9.82, 3.13, 23.89
 030, 6.68, 2.99, 10.33, 3.68, 24.22
 031, 7.83, 3.14, 12.92, 3.81, 24.60
 032, 9.20, 3.33, 11.96, 5.27, 25.08
 033, 7.42, 3.45, 10.94, 2.01, 25.74
 034, 7.86, 3.58, 11.42, 3.88, 26.02
 035, 8.85, 3.73, 10.74, 5.79, 26.32
 036, 9.49, 3.89, 13.61, 5.48, 26.67
 037, 9.86, 4.06, 18.07, 4.87, 27.01
 038, 9.87, 4.21, 14.23, 4.17, 27.47
 039, 10.64, 4.37, 19.29, 4.01, 27.70
 040, 11.69, 4.56, 18.20, 5.40, 28.83
 041, 12.41, 4.75, 15.78, 5.42, 28.59
 042, 11.78, 4.92, 16.20, 9.70, 28.92
 043, 11.20, 5.06, 15.65, 6.43, 29.26
```

044, 13.49, 5.25, 19.36, 7.14, 30.02
045, 11.58, 5.39, 17.78, 4.85, 30.12
046, 12.93, 5.56, 16.39, 9.15, 30.26
047, 13.24, 5.72, 20.54, 9.84, 30.69
048, 14.54, 5.90, 20.52, 8.78, 31.46
049, 14.81, 6.09, 19.60, 8.28, 31.49
050, 14.19, 6.25, 20.41, 8.36, 31.98
051, 15.62, 6.43, 19.97, 11.01, 32.19
052, 14.83, 6.59, 20.28, 9.72, 32.22
053, 15.91, 6.77, 22.00, 10.61, 31.96
054, 17.03, 6.96, 20.93, 9.36, 31.83
055, 16.42, 7.13, 21.48, 11.24, 31.58
056, 17.16, 7.31, 22.86, 12.99, 31.24
057, 18.11, 7.50, 25.54, 11.36, 31.33
058, 18.86, 7.70, 27.25, 11.33, 31.56
059, 19.22, 7.89, 25.86, 13.86, 32.29
060, 21.00, 8.11, 27.75, 15.41, 32.78
061, 18.57, 8.28, 27.54, 12.15, 32.16
062, 20.36, 8.48, 30.27, 12.30, 32.97
063, 21.33, 8.68, 27.11, 11.87, 32.65
064, 21.30, 8.88, 26.24, 13.86, 32.34
065, 20.99, 9.06, 25.68, 11.34, 32.16
066, 22.83, 9.27, 28.10, 9.52, 31.96
067, 20.41, 9.44, 26.16, 15.13, 32.10
068, 21.22, 9.61, 28.97, 10.12, 32.04
069, 21.91, 9.79, 28.46, 11.80, 32.15
070, 25.93, 10.02, 31.25, 13.45, 32.18
071, 27.73, 10.27, 35.42, 14.51, 32.00
072, 25.05, 10.48, 34.45, 8.54, 32.11
073, 27.78, 10.71, 33.21, 11.48, 32.22
074, 23.92, 10.89, 34.73, 10.61, 32.13
075, 25.65, 11.09, 30.69, 13.46, 32.05
076, 23.57, 11.25, 31.72, 11.13, 32.05
077, 28.41, 11.47, 35.17, 6.56, 32.08
078, 26.85, 11.67, 33.94, 10.23, 32.30
079, 28.57, 11.89, 35.88, 14.93, 32.17
080, 29.29, 12.10, 34.31, 17.38, 32.55
081, 28.42, 12.30, 39.47, 10.88, 32.10
082, 25.55, 12.47, 34.27, 12.29, 32.73
083, 28.44, 12.66, 34.63, 16.47, 31.74
084, 28.77, 12.85, 34.69, 12.65, 32.00
085, 30.42, 13.06, 39.37, 23.16, 31.58
086, 31.06, 13.27, 36.76, 22.66, 31.51
087, 32.36, 13.49, 39.44, 21.62, 31.55
088, 33.60, 13.71, 37.96, 28.97, 31.94
089, 35.28, 13.96, 39.58, 32.71, 31.78
090, 34.05, 14.18, 37.67, 22.87, 31.79
091, 32.64, 14.38, 39.55, 21.06, 31.78

```
092, 33.78, 14.59, 39.13, 30.19, 31.81
093, 34.62, 14.81, 39.29, 30.03, 31.90
094, 34.17, 15.02, 38.51, 27.10, 31.72
095, 34.07, 15.22, 37.55, 23.69, 31.98
096, 33.59, 15.41, 39.27, 24.48, 32.02
097, 34.81, 15.61, 38.93, 28.76, 32.08
098, 34.79, 15.80, 39.49, 28.69, 32.13
099, 34.07, 15.99, 39.41, 27.82, 32.17
100, 36.19, 16.19, 39.58, 32.93, 32.75
101, 35.89, 16.54, 39.52, 31.35, 32.26
102, 35.36, 16.89, 39.42, 29.51, 32.41
103, 38.01, 17.26, 39.61, 34.81, 32.35
104, 38.23, 17.63, 39.42, 36.68, 32.08
105, 38.39, 18.01, 39.63, 30.72, 32.24
106, 38.27, 18.38, 39.64, 31.98, 32.14
107, 38.26, 18.75, 39.57, 30.64, 32.43
108, 37.93, 19.12, 39.60, 29.03, 32.25
109, 38.42, 19.49, 39.64, 32.74, 32.20
110, 38.81, 19.87, 39.66, 33.05, 32.14
111, 38.67, 20.24, 39.67, 36.73, 32.25
112, 38.86, 20.61, 39.69, 34.39, 32.43
113, 38.17, 20.97, 39.67, 31.89, 32.25
114, 38.98, 21.34, 39.65, 34.56, 32.21
115, 39.25, 21.71, 39.63, 38.27, 32.17
116, 38.23, 22.07, 39.67, 34.11, 32.06
117, 38.06, 22.43, 39.65, 32.65, 32.33
118, 39.34, 22.80, 39.70, 38.45, 32.30
119, 38.89, 23.15, 39.66, 35.41, 32.10
120, 38.86, 23.50, 39.66, 34.57, 32.87
121, 38.72, 23.85, 39.62, 36.69, 32.20
122, 38.69, 24.20, 39.61, 36.45, 32.57
123, 37.94, 24.53, 39.58, 33.53, 31.67
124, 37.66, 24.86, 39.65, 33.24, 31.69
125, 37.22, 25.17, 39.59, 29.30, 32.19
126, 36.98, 25.48, 39.43, 32.74, 32.35
127, 38.26, 25.80, 39.60, 35.94, 32.15
128, 38.43, 26.12, 39.64, 36.11, 31.36
129, 37.97, 26.43, 39.65, 33.28, 31.48
130, 38.37, 26.75, 39.57, 34.79, 32.12
131, 38.99, 27.06, 39.56, 37.39, 32.37
132, 38.28, 27.35, 39.50, 33.69, 32.37
133, 37.85, 27.65, 39.59, 34.57, 32.34
134, 38.46, 27.96, 39.54, 36.40, 32.28
135, 36.51, 28.23, 39.69, 29.32, 32.24
136, 37.63, 28.52, 39.46, 32.34, 32.20
137, 37.43, 28.79, 39.60, 32.08, 32.12
138, 35.49, 29.05, 39.15, 14.50, 32.23
139, 34.04, 29.28, 39.21, 26.32, 32.24
```

```
 140, 33.44, 29.50, 39.16, 26.42, 32.84
 141, 36.43, 29.74, 39.58, 32.06, 32.54
 142, 31.30, 29.94, 38.99, 24.32, 32.35
 143, 35.10, 30.17, 38.88, 23.16, 32.35

Solved in 143 episodes!        Avg Score: 30.17, time: 4150.681720256805
End:  2019-11-19 22:47:17.913165
```

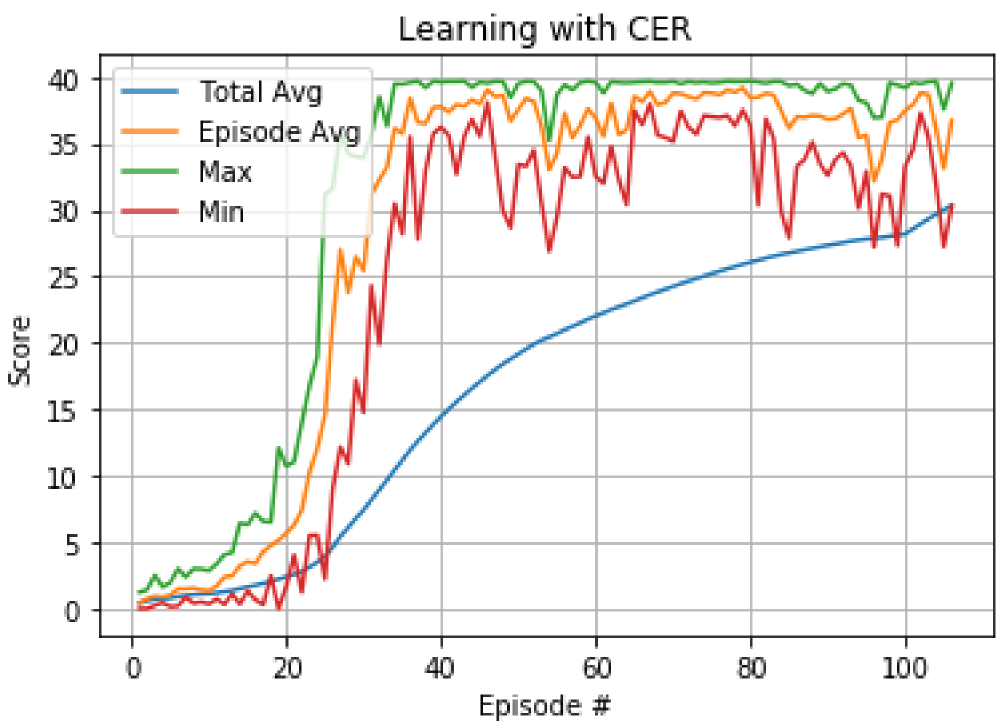


In [8]:
```python
# Agent with CER enabled
agent = DDPG(state_size=state_size, action_size=action_size, CER=True)
scores, averages, maxima, minima = ddpg(agent, n_episodes=200)

plt.plot(np.arange(1, len(averages)+1), averages)
plt.plot(np.arange(1, len(scores)+1), scores)
plt.plot(np.arange(1, len(maxima)+1), maxima)
plt.plot(np.arange(1, len(minima)+1), minima)
plt.title('Learning with CER')
plt.ylabel('Score')
plt.xlabel('Episode #')
plt.legend(['Total Avg', 'Episode Avg', 'Max', 'Min'], loc='upper left')
plt.grid()
plt.show()
```

```
Start:  2019-11-24 18:18:30.753477
 001, 0.42, 0.42, 1.23, 0.00, 20.43
 002, 0.75, 0.59, 1.43, 0.00, 20.48
 003, 0.94, 0.71, 2.52, 0.24, 20.58
 004, 0.86, 0.74, 1.62, 0.45, 20.65
 005, 1.02, 0.80, 1.97, 0.15, 20.69
 006, 1.48, 0.91, 3.02, 0.20, 20.96
 007, 1.48, 0.99, 2.37, 0.87, 21.09
 008, 1.54, 1.06, 3.00, 0.40, 21.27
 009, 1.41, 1.10, 2.98, 0.51, 21.41
 010, 1.35, 1.13, 2.86, 0.35, 21.65
 011, 1.72, 1.18, 3.38, 0.73, 21.79
 012, 2.39, 1.28, 4.07, 0.32, 21.99
 013, 2.48, 1.37, 4.21, 1.13, 22.34
 014, 3.23, 1.51, 6.44, 0.34, 22.73
 015, 3.54, 1.64, 6.32, 1.33, 22.88
 016, 3.41, 1.75, 7.19, 0.67, 23.15
 017, 4.29, 1.90, 6.57, 0.30, 23.51
 018, 4.76, 2.06, 6.51, 2.47, 23.88
 019, 5.16, 2.22, 12.07, 0.00, 24.01
 020, 5.67, 2.39, 10.73, 1.69, 24.70
 021, 6.28, 2.58, 11.05, 4.04, 24.77
 022, 7.41, 2.80, 13.95, 1.24, 25.11
 023, 10.28, 3.12, 16.78, 5.50, 25.46
 024, 12.02, 3.50, 18.92, 5.53, 25.68
 025, 14.58, 3.94, 31.15, 2.22, 26.17
 026, 21.46, 4.61, 31.63, 8.99, 26.63
 027, 27.02, 5.44, 35.90, 12.16, 27.34
 028, 23.78, 6.10, 34.21, 10.90, 27.37
 029, 26.48, 6.80, 34.00, 17.22, 27.85
 030, 25.41, 7.42, 33.92, 14.75, 28.24
 031, 31.13, 8.19, 35.64, 24.32, 28.81
 032, 32.25, 8.94, 38.52, 19.88, 29.04
 033, 33.22, 9.67, 36.32, 26.30, 29.56
 034, 36.13, 10.45, 39.48, 30.47, 29.84
 035, 35.75, 11.17, 39.45, 28.17, 30.29
 036, 38.42, 11.93, 39.58, 35.52, 31.09
 037, 36.59, 12.60, 39.68, 27.79, 31.12
 038, 36.50, 13.23, 39.20, 32.97, 31.75
 039, 37.71, 13.85, 39.67, 35.74, 31.98
 040, 37.77, 14.45, 39.61, 36.24, 32.71
 041, 37.36, 15.01, 39.65, 35.61, 32.54
 042, 37.94, 15.56, 39.64, 32.68, 32.82
 043, 37.79, 16.07, 39.68, 35.54, 33.61
 044, 38.29, 16.58, 39.22, 36.83, 34.15
 045, 37.96, 17.05, 39.61, 35.53, 34.03
 046, 39.05, 17.53, 39.54, 38.09, 34.97
 047, 38.52, 17.98, 39.66, 33.98, 35.53
```

048, 38.63, 18.41, 39.65, 29.94, 35.77
049, 36.67, 18.78, 38.46, 28.65, 35.51
050, 37.57, 19.16, 39.51, 33.42, 35.97
051, 38.40, 19.53, 39.66, 33.28, 36.27
052, 38.16, 19.89, 39.64, 34.51, 36.27
053, 36.29, 20.20, 38.98, 30.51, 36.30
054, 33.01, 20.44, 35.23, 26.88, 36.41
055, 34.27, 20.69, 38.70, 29.47, 36.49
056, 37.27, 20.99, 39.39, 33.18, 36.65
057, 35.43, 21.24, 39.03, 32.46, 36.76
058, 36.35, 21.50, 39.57, 32.50, 36.38
059, 37.61, 21.77, 39.67, 35.49, 36.56
060, 36.90, 22.03, 39.59, 32.53, 37.27
061, 35.54, 22.25, 38.80, 31.98, 36.49
062, 38.05, 22.50, 39.63, 34.77, 36.65
063, 35.67, 22.71, 39.58, 32.27, 36.63
064, 36.02, 22.92, 39.55, 30.39, 36.83
065, 38.54, 23.16, 39.59, 37.48, 36.46
066, 38.16, 23.39, 39.63, 36.42, 36.59
067, 38.91, 23.62, 39.65, 37.97, 36.55
068, 37.91, 23.83, 39.58, 35.62, 36.79
069, 38.09, 24.04, 39.59, 35.45, 36.50
070, 38.79, 24.25, 39.65, 35.15, 36.63
071, 38.73, 24.45, 39.48, 37.43, 36.64
072, 38.53, 24.65, 39.64, 36.21, 36.60
073, 38.35, 24.83, 39.56, 35.70, 36.52
074, 38.83, 25.02, 39.61, 37.09, 36.52
075, 38.78, 25.21, 39.54, 37.02, 36.56
076, 38.66, 25.38, 39.58, 36.95, 36.67
077, 39.02, 25.56, 39.67, 37.10, 37.23
078, 38.86, 25.73, 39.65, 36.32, 36.81
079, 39.17, 25.90, 39.64, 37.53, 36.68
080, 38.42, 26.06, 39.69, 36.35, 37.21
081, 38.49, 26.21, 39.61, 30.36, 36.78
082, 38.78, 26.36, 39.52, 36.88, 37.38
083, 38.66, 26.51, 39.65, 35.37, 37.45
084, 37.42, 26.64, 39.64, 29.81, 37.49
085, 36.14, 26.75, 39.33, 27.88, 37.51
086, 37.02, 26.87, 39.45, 33.27, 37.53
087, 36.96, 26.99, 38.99, 33.80, 37.46
088, 37.11, 27.10, 38.76, 35.11, 37.61
089, 37.06, 27.22, 39.49, 33.46, 37.62
090, 36.82, 27.32, 38.91, 32.59, 37.60
091, 36.86, 27.43, 39.11, 33.80, 37.65
092, 37.23, 27.53, 39.49, 34.30, 37.58
093, 37.64, 27.64, 39.37, 33.32, 37.66
094, 35.43, 27.73, 38.24, 30.13, 37.76
095, 35.64, 27.81, 38.07, 32.89, 37.71

```
096, 32.18, 27.85, 36.93, 27.21, 37.78
097, 33.81, 27.92, 36.96, 31.19, 37.25
098, 36.56, 28.00, 39.56, 31.06, 36.88
099, 36.79, 28.09, 39.38, 27.31, 36.83
100, 37.48, 28.19, 39.20, 33.30, 37.46
101, 37.92, 28.56, 39.57, 34.51, 36.71
102, 38.82, 28.94, 39.49, 37.28, 36.75
103, 38.69, 29.32, 39.62, 35.48, 36.79
104, 36.05, 29.67, 39.67, 31.91, 36.90
105, 33.13, 29.99, 37.59, 27.24, 36.75
106, 36.73, 30.35, 39.52, 30.26, 37.05

Solved in 106 episodes!        Avg Score: 30.35, time: 3425.3794577121735
End:  2019-11-24 19:15:36.136079
```



Saves experiences for training future agents. Warning file is quite large.

```
In [24]: env.close()
```

1.0.5 5. See the pre-trained agent in action

```
In [10]: agent = DDPG(state_size=state_size, action_size=action_size, CER=True)
```

```
In [11]: agent.load('./96_96_actor.pth', './106_96_96_critic.pth')
```

In [12]:
```
def play(agent, episodes=3):
    for i_episode in range(episodes):
        env_info = env.reset(train_mode=False)[brain_name]     # reset the environment
        states = env_info.vector_observations                  # get the current state
        scores = np.zeros(num_agents)                          # initialize the score (
        while True:
            actions = np.random.randn(num_agents, action_size) # select an action (for
            actions = agent.act(states, add_noise=False)       # all actions between -1
            env_info = env.step(actions)[brain_name]           # send all actions to tn
            next_states = env_info.vector_observations         # get next state (for ea
            rewards = env_info.rewards                         # get reward (for each a
            dones = env_info.local_done                        # see if episode finishe
            scores += env_info.rewards                         # update the score (for
            states = next_states                               # roll over states to ne
            if np.any(dones):                                  # exit loop if episode f
                break
            #break
        print('Ep No: {} Total score (averaged over agents): {}'.format(i_episode, np.m
```

In [13]:
```
play(agent, 10)
```

```
Ep No: 0 Total score (averaged over agents): 34.48299922924489
Ep No: 1 Total score (averaged over agents): 31.785499289538713
Ep No: 2 Total score (averaged over agents): 32.320999277569356
Ep No: 3 Total score (averaged over agents): 33.78049924494699
Ep No: 4 Total score (averaged over agents): 31.428999297507108
Ep No: 5 Total score (averaged over agents): 32.94849926354364
Ep No: 6 Total score (averaged over agents): 31.92699928637594
Ep No: 7 Total score (averaged over agents): 32.104499282408504
Ep No: 8 Total score (averaged over agents): 33.96599924080074
Ep No: 9 Total score (averaged over agents): 32.25999927893281
```