Continuous_Control

July 17, 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 [8]: !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 [9]: 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
        env = UnityEnvironment(file_name='/data/Reacher_Linux_NoVis/Reacher.x86_64')
        # 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 [11]: # 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 [12]: # 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[
         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]
```

```
In [13]: env_info = env.reset(train_mode=True)[brain_name]      # reset the environment
         states = env_info.vector_observations                  # get the current state (for eac
         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 age
             actions = np.clip(actions, -1, 1)                  # all actions between -1 and 1
             env_info = env.step(actions)[brain_name]           # send all actions to tne enviro
             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 age
             states = next_states                               # roll over states to next time
             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: 0.17149999616667627
```

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.

When finished, you can close the environment.

```
In [14]: print(actions.shape)
         print(next_states.shape)
         print(len(rewards))
```

```
(20, 4)
(20, 33)
20
```

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 [16]:
```
import numpy as np

import torch
import torch.nn as nn
import torch.nn.functional as F

def hidden_init(layer):
    fan_in = layer.weight.data.size()[0]
    lim = 1. / np.sqrt(fan_in)
    return (-lim, lim)

class Actor(nn.Module):
    """Actor (Policy) Model."""

    def __init__(self, state_size, action_size, seed=0, fc1_units=128, fc2_units=128):
        """Initialize parameters and build model.
        Params
        ======
            state_size (int): Dimension of each state
            action_size (int): Dimension of each action
            seed (int): Random seed
            fc1_units (int): Number of nodes in first hidden layer
            fc2_units (int): Number of nodes in second hidden layer
        """
        super(Actor, self).__init__()
        self.seed = torch.manual_seed(seed)
        self.fc1 = nn.Linear(state_size, fc1_units)
        self.bn1 = nn.BatchNorm1d(fc1_units)
        self.fc2 = nn.Linear(fc1_units, fc2_units)
        self.fc3 = nn.Linear(fc2_units, action_size)
        self.reset_parameters()

    def reset_parameters(self):
        self.fc1.weight.data.uniform_(*hidden_init(self.fc1))
        self.fc2.weight.data.uniform_(*hidden_init(self.fc2))
        self.fc3.weight.data.uniform_(-3e-3, 3e-3)

    def forward(self, state):
        """Build an actor (policy) network that maps states -> actions."""
        if state.dim() == 1:
            state = torch.unsqueeze(state,0)
```

```
        x = F.relu(self.fc1(state))
        x = self.bn1(x)
        x = F.relu(self.fc2(x))
        return F.tanh(self.fc3(x))

class Critic(nn.Module):
    """Critic (Value) Model."""

    def __init__(self, state_size, action_size, seed=0, fcs1_units=128, fc2_units=128):
        """Initialize parameters and build model.
        Params
        ======
            state_size (int): Dimension of each state
            action_size (int): Dimension of each action
            seed (int): Random seed
            fcs1_units (int): Number of nodes in the first hidden layer
            fc2_units (int): Number of nodes in the second hidden layer
        """
        super(Critic, self).__init__()
        self.seed = torch.manual_seed(seed)
        self.fcs1 = nn.Linear(state_size, fcs1_units)
        self.bn1 = nn.BatchNorm1d(fcs1_units)
        self.fc2 = nn.Linear(fcs1_units+action_size, fc2_units)
        self.fc3 = nn.Linear(fc2_units, 1)
        self.reset_parameters()

    def reset_parameters(self):
        self.fcs1.weight.data.uniform_(*hidden_init(self.fcs1))
        self.fc2.weight.data.uniform_(*hidden_init(self.fc2))
        self.fc3.weight.data.uniform_(-3e-3, 3e-3)

    def forward(self, state, action):
        """Build a critic (value) network that maps (state, action) pairs -> Q-values.'
        if state.dim() == 1:
            state = torch.unsqueeze(state,0)
        xs = F.relu(self.fcs1(state))
        xs = self.bn1(xs)
        x = torch.cat((xs, action), dim=1)
        x = F.relu(self.fc2(x))
        return self.fc3(x)
```

In [19]:
```
import numpy as np
import random
import copy
from collections import namedtuple, deque

##from model import Actor, Critic
```

```python
import torch
import torch.nn.functional as F
import torch.optim as optim

BUFFER_SIZE = int(1e6)  # 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 = 0.0      # L2 weight decay

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

class Agent():
    """Interacts with and learns from the environment."""

    def __init__(self, state_size, action_size, random_seed):
        """Initialize an Agent object.

        Params
        ======
            state_size (int): dimension of each state
            action_size (int): dimension of each action
            random_seed (int): random seed
        """
        self.state_size = state_size
        self.action_size = action_size
        self.seed = random.seed(random_seed)

        # Actor Network (w/ Target Network)
        self.actor_local = Actor(state_size, action_size, random_seed).to(device)
        self.actor_target = Actor(state_size, action_size, random_seed).to(device)
        self.actor_optimizer = optim.Adam(self.actor_local.parameters(), lr=LR_ACTOR)

        # Critic Network (w/ Target Network)
        self.critic_local = Critic(state_size, action_size, random_seed).to(device)
        self.critic_target = Critic(state_size, action_size, random_seed).to(device)
        self.critic_optimizer = optim.Adam(self.critic_local.parameters(), lr=LR_CRITIC

        # Noise process
        self.noise = OUNoise(action_size, random_seed)

        # Replay memory
        self.memory = ReplayBuffer(action_size, BUFFER_SIZE, BATCH_SIZE, random_seed)

    def step(self, state, action, reward, next_state, done):
        """Save experience in replay memory, and use random sample from buffer to learn
```

```
        # Save experience / reward
        self.memory.add(state, action, reward, next_state, done)

        # Learn, if enough samples are available in memory
        #Commented out to improve performance. Now we learn only when start_learn is ca
        #if len(self.memory) > BATCH_SIZE:
            #experiences = self.memory.sample()
            #self.learn(experiences, GAMMA)

    def act(self, state, add_noise=True):
        """Returns actions for given state as per current policy."""
        state = torch.from_numpy(state).float().to(device)
        self.actor_local.eval()
        with torch.no_grad():
            action = self.actor_local(state).cpu().data.numpy()
        self.actor_local.train()
        if add_noise:
            action += self.noise.sample()
        return np.clip(action, -1, 1)

    def reset(self):
        self.noise.reset()

    def start_learn(self):
        if len(self.memory) > BATCH_SIZE:
            experiences = self.memory.sample()
            self.learn(experiences, GAMMA)

    def learn(self, experiences, gamma):
        """Update policy and value parameters using given batch of experience tuples.
        Q_targets = r +  * critic_target(next_state, actor_target(next_state))
        where:
            actor_target(state) -> action
            critic_target(state, action) -> Q-value
        Params
        ======
            experiences (Tuple[torch.Tensor]): tuple of (s, a, r, s', done) tuples
            gamma (float): discount factor
        """
        states, actions, rewards, next_states, dones = experiences

        # ---------------------------- update critic ---------------------------- #
        # Get predicted next-state actions and Q values from target models
        actions_next = self.actor_target(next_states)
        Q_targets_next = self.critic_target(next_states, actions_next)
        # Compute Q targets for current states (y_i)
        Q_targets = rewards + (gamma * Q_targets_next * (1 - dones))
        # Compute critic loss
```

```
        Q_expected = self.critic_local(states, actions)
        critic_loss = F.mse_loss(Q_expected, Q_targets)
        # Minimize the loss
        self.critic_optimizer.zero_grad()
        critic_loss.backward()
        self.critic_optimizer.step()

        # ---------------------------- update actor ---------------------------- #
        # Compute actor loss
        actions_pred = self.actor_local(states)
        actor_loss = -self.critic_local(states, actions_pred).mean()
        # Minimize the loss
        self.actor_optimizer.zero_grad()
        actor_loss.backward()
        self.actor_optimizer.step()

        # ----------------------- update target networks ----------------------- #
        self.soft_update(self.critic_local, self.critic_target, TAU)
        self.soft_update(self.actor_local, self.actor_target, TAU)

    def soft_update(self, local_model, target_model, tau):
        """Soft update model parameters.
        _target = *_local + (1 - )*_target
        Params
        ======
            local_model: PyTorch model (weights will be copied from)
            target_model: PyTorch model (weights will be copied to)
            tau (float): interpolation parameter
        """
        for target_param, local_param in zip(target_model.parameters(), local_model.par
            target_param.data.copy_(tau*local_param.data + (1.0-tau)*target_param.data)

class OUNoise:
    """Ornstein-Uhlenbeck process."""

    def __init__(self, size, seed, mu=0., theta=0.15, sigma=0.2):
        """Initialize parameters and noise process."""
        self.mu = mu * np.ones(size)
        self.theta = theta
        self.sigma = sigma
        self.size = size
        self.seed = random.seed(seed)
        self.reset()

    def reset(self):
        """Reset the internal state (= noise) to mean (mu)."""
        self.state = copy.copy(self.mu)
```

```
    def sample(self):
        """Update internal state and return it as a noise sample."""
        x = self.state

        # Thanks to Hiu C. for this tip, this really helped get the learning up to the
        dx = self.theta * (self.mu - x) + self.sigma * np.random.standard_normal(self.s

        self.state = x + dx
        return self.state

class ReplayBuffer:
    """Fixed-size buffer to store experience tuples."""

    def __init__(self, action_size, buffer_size, batch_size, seed):
        """Initialize a ReplayBuffer object.
        Params
        ======
            buffer_size (int): maximum size of buffer
            batch_size (int): size of each training batch
        """
        self.action_size = action_size
        self.memory = deque(maxlen=buffer_size)  # internal memory (deque)
        self.batch_size = batch_size
        self.experience = namedtuple("Experience", field_names=["state", "action", "rew
        self.seed = random.seed(seed)

    def add(self, state, action, reward, next_state, done):
        """Add a new experience to memory."""
        e = self.experience(state, action, reward, next_state, done)
        self.memory.append(e)

    def sample(self):
        """Randomly sample a batch of experiences from memory."""
        experiences = random.sample(self.memory, k=self.batch_size)

        states = torch.from_numpy(np.vstack([e.state for e in experiences if e is not N
        actions = torch.from_numpy(np.vstack([e.action for e in experiences if e is not
        rewards = torch.from_numpy(np.vstack([e.reward for e in experiences if e is not
        next_states = torch.from_numpy(np.vstack([e.next_state for e in experiences if
        dones = torch.from_numpy(np.vstack([e.done for e in experiences if e is not Non

        return (states, actions, rewards, next_states, dones)

    def __len__(self):
        """Return the current size of internal memory."""
        return len(self.memory)
```

In [20]:
```
import random
```

```
import torch
import time

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

##from ddpg_agent import Agent, ReplayBuffer

random_seed = random.randint(1,25)
#random_seed = 7
train_mode = True

agent = Agent(state_size=state_size, action_size=action_size, random_seed=random_seed)
```

In [21]:
```
def ddpg(n_episodes=2000, max_t=1000, print_every=10, learn_every=20, num_learn=10, goa
    total_scores_deque = deque(maxlen=100)
    total_scores = []

    for i_episode in range(1, n_episodes+1):
        #Reset the env and the agent
        env_info = env.reset(train_mode=train_mode)[brain_name]   # reset the environme
        states = env_info.vector_observations # get the crrrent states
        scores = np.zeros(num_agents) # initialize the score for each agent
        agent.reset()

        start_time = time.time()

        for t in range(max_t):
            actions = agent.act(states)

            env_info = env.step(actions)[brain_name]            # send all actions to t
            next_states = env_info.vector_observations          # get next state (for e
            rewards = env_info.rewards                          # get reward (for each

            dones = env_info.local_done                         # see if episode finish

            for state, action, reward, next_state, done in zip(states, actions, rewards
                agent.step(state, action, reward, next_state, done) # send actions to t

            scores += env_info.rewards                           # update the score (fo
            states = next_states                                # roll over states to n

            if t%learn_every == 0:
                for _ in range(num_learn):
                    agent.start_learn()

            if np.any(dones):                                   # exit loop if episode
```

```python
                break

        mean_score = np.mean(scores)
        min_score = np.min(scores)
        max_score = np.max(scores)
        total_scores_deque.append(mean_score)
        total_scores.append(mean_score)
        total_average_score = np.mean(total_scores_deque)
        duration = time.time() - start_time

        print('\rEpisode {}\tTotal Average Score: {:.2f}\tMean: {:.2f}\tMin: {:.2f}\tMa
              .format(i_episode, total_average_score, mean_score, min_score, max_score,

        if i_episode % print_every == 0:
            torch.save(agent.actor_local.state_dict(), 'checkpoint_actor.pth')
            torch.save(agent.critic_local.state_dict(), 'checkpoint_critic.pth')
            print('\rEpisode {}\tTotal Average Score: {:.2f}'.format(i_episode, total_a

        if total_average_score >= goal_score and i_episode >= 100:
            print('Problem Solved after {} epsisodes!! Total Average score: {:.2f}'.for
            torch.save(agent.actor_local.state_dict(), 'checkpoint_actor.pth')
            torch.save(agent.critic_local.state_dict(), 'checkpoint_critic.pth')
            break

    return total_scores

scores = ddpg()
```

```
Episode 1        Total Average Score: 0.80        Mean: 0.80        Min: 0.19        Max: 2.43
Episode 2        Total Average Score: 0.76        Mean: 0.72        Min: 0.01        Max: 1.64
Episode 3        Total Average Score: 0.78        Mean: 0.82        Min: 0.11        Max: 2.41
Episode 4        Total Average Score: 0.74        Mean: 0.60        Min: 0.00        Max: 1.56
Episode 5        Total Average Score: 0.77        Mean: 0.89        Min: 0.12        Max: 1.57
Episode 6        Total Average Score: 0.73        Mean: 0.56        Min: 0.11        Max: 1.56
Episode 7        Total Average Score: 0.77        Mean: 0.97        Min: 0.14        Max: 3.52
Episode 8        Total Average Score: 0.79        Mean: 0.99        Min: 0.00        Max: 2.94
Episode 9        Total Average Score: 0.85        Mean: 1.29        Min: 0.38        Max: 2.54
Episode 10        Total Average Score: 0.87        Mean: 1.09        Min: 0.09        Max: 1.92
Episode 10        Total Average Score: 0.87
Episode 11        Total Average Score: 0.90        Mean: 1.20        Min: 0.11        Max: 2.13
Episode 12        Total Average Score: 0.91        Mean: 1.04        Min: 0.00        Max: 2.07
Episode 13        Total Average Score: 0.93        Mean: 1.09        Min: 0.00        Max: 2.96
Episode 14        Total Average Score: 0.91        Mean: 0.74        Min: 0.00        Max: 2.58
Episode 15        Total Average Score: 0.92        Mean: 1.02        Min: 0.06        Max: 3.40
Episode 16        Total Average Score: 0.95        Mean: 1.36        Min: 0.00        Max: 3.64
Episode 17        Total Average Score: 0.99        Mean: 1.59        Min: 0.34        Max: 2.77
Episode 18        Total Average Score: 1.03        Mean: 1.77        Min: 0.87        Max: 3.28
Episode 19        Total Average Score: 1.04        Mean: 1.30        Min: 0.31        Max: 2.51
```

```
Episode 20          Total Average Score: 1.09        Mean: 2.00        Min: 0.52        Max: 5.56
Episode 20          Total Average Score: 1.09
Episode 21          Total Average Score: 1.11        Mean: 1.52        Min: 0.29        Max: 3.41
Episode 22          Total Average Score: 1.14        Mean: 1.71        Min: 0.53        Max: 2.97
Episode 23          Total Average Score: 1.17        Mean: 1.83        Min: 0.91        Max: 3.91
Episode 24          Total Average Score: 1.19        Mean: 1.69        Min: 0.90        Max: 3.35
Episode 25          Total Average Score: 1.23        Mean: 2.15        Min: 0.69        Max: 5.68
Episode 26          Total Average Score: 1.28        Mean: 2.55        Min: 0.83        Max: 6.22
Episode 27          Total Average Score: 1.32        Mean: 2.49        Min: 0.92        Max: 4.04
Episode 28          Total Average Score: 1.38        Mean: 2.84        Min: 0.86        Max: 5.23
Episode 29          Total Average Score: 1.39        Mean: 1.62        Min: 0.40        Max: 2.90
Episode 30          Total Average Score: 1.43        Mean: 2.72        Min: 0.16        Max: 6.47
Episode 30          Total Average Score: 1.43
Episode 31          Total Average Score: 1.49        Mean: 3.15        Min: 1.02        Max: 6.92
Episode 32          Total Average Score: 1.54        Mean: 3.33        Min: 1.00        Max: 6.32
Episode 33          Total Average Score: 1.61        Mean: 3.68        Min: 1.06        Max: 6.39
Episode 34          Total Average Score: 1.67        Mean: 3.54        Min: 1.09        Max: 6.83
Episode 35          Total Average Score: 1.73        Mean: 3.93        Min: 0.65        Max: 6.49
Episode 36          Total Average Score: 1.80        Mean: 4.06        Min: 2.52        Max: 5.76
Episode 37          Total Average Score: 1.87        Mean: 4.36        Min: 1.38        Max: 7.91
Episode 38          Total Average Score: 1.93        Mean: 4.19        Min: 2.02        Max: 6.92
Episode 39          Total Average Score: 1.99        Mean: 4.30        Min: 1.72        Max: 8.69
Episode 40          Total Average Score: 2.05        Mean: 4.38        Min: 1.45        Max: 7.76
Episode 40          Total Average Score: 2.05
Episode 41          Total Average Score: 2.12        Mean: 5.17        Min: 2.14        Max: 9.13
Episode 42          Total Average Score: 2.23        Mean: 6.68        Min: 3.30        Max: 10.58
Episode 43          Total Average Score: 2.29        Mean: 4.64        Min: 1.31        Max: 7.83
Episode 44          Total Average Score: 2.37        Mean: 6.01        Min: 2.76        Max: 11.34
Episode 45          Total Average Score: 2.46        Mean: 6.25        Min: 3.09        Max: 11.42
Episode 46          Total Average Score: 2.57        Mean: 7.78        Min: 4.22        Max: 12.64
Episode 47          Total Average Score: 2.66        Mean: 6.67        Min: 2.12        Max: 12.14
Episode 48          Total Average Score: 2.76        Mean: 7.19        Min: 3.96        Max: 10.08
Episode 49          Total Average Score: 2.85        Mean: 7.47        Min: 3.28        Max: 12.57
Episode 50          Total Average Score: 2.96        Mean: 8.18        Min: 5.10        Max: 12.96
Episode 50          Total Average Score: 2.96
Episode 51          Total Average Score: 3.06        Mean: 8.24        Min: 6.10        Max: 12.28
Episode 52          Total Average Score: 3.15        Mean: 7.88        Min: 4.93        Max: 10.31
Episode 53          Total Average Score: 3.26        Mean: 8.63        Min: 4.72        Max: 14.63
Episode 54          Total Average Score: 3.37        Mean: 9.32        Min: 5.58        Max: 12.65
Episode 55          Total Average Score: 3.49        Mean: 10.06        Min: 3.23        Max: 14.4
Episode 56          Total Average Score: 3.58        Mean: 8.31        Min: 4.39        Max: 12.68
Episode 57          Total Average Score: 3.66        Mean: 8.31        Min: 3.95        Max: 12.41
Episode 58          Total Average Score: 3.76        Mean: 9.30        Min: 4.43        Max: 12.72
Episode 59          Total Average Score: 3.86        Mean: 9.70        Min: 5.45        Max: 14.95
Episode 60          Total Average Score: 3.97        Mean: 10.64        Min: 6.13        Max: 16.0
Episode 60          Total Average Score: 3.97
Episode 61          Total Average Score: 4.09        Mean: 10.89        Min: 5.81        Max: 14.0
Episode 62          Total Average Score: 4.22        Mean: 12.64        Min: 5.78        Max: 18.6
```

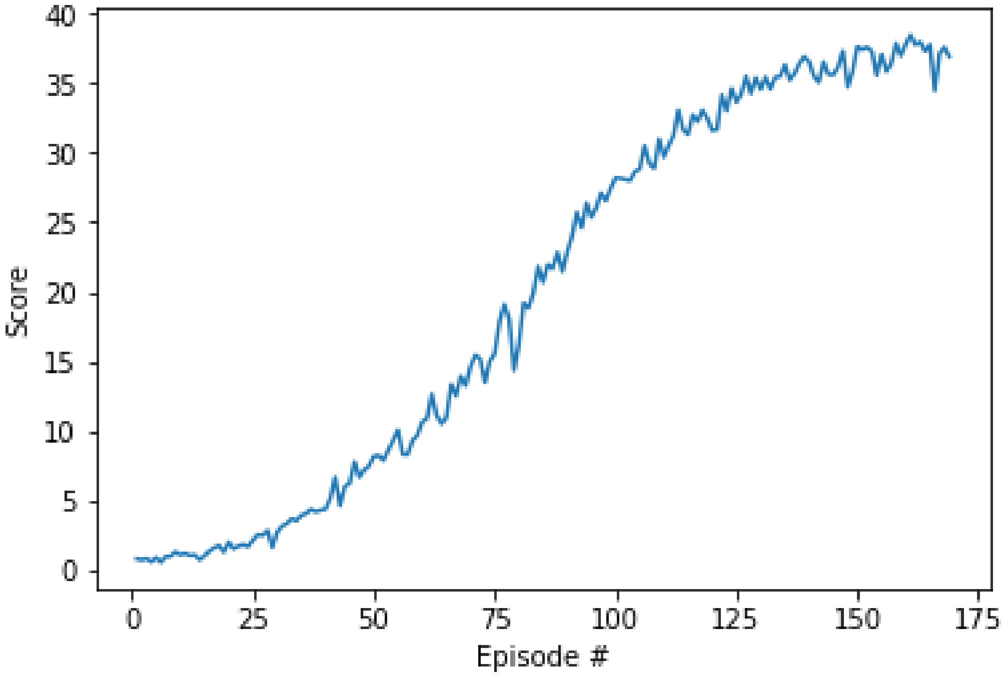
```
Episode 63        Total Average Score: 4.33       Mean: 11.09       Min: 3.52       Max: 17.6
Episode 64        Total Average Score: 4.43       Mean: 10.51       Min: 3.79       Max: 15.2
Episode 65        Total Average Score: 4.53       Mean: 10.95       Min: 3.34       Max: 15.2
Episode 66        Total Average Score: 4.66       Mean: 13.34       Min: 7.82       Max: 17.4
Episode 67        Total Average Score: 4.78       Mean: 12.51       Min: 7.42       Max: 17.4
Episode 68        Total Average Score: 4.91       Mean: 13.92       Min: 9.58       Max: 19.1
Episode 69        Total Average Score: 5.04       Mean: 13.25       Min: 8.44       Max: 21.1
Episode 70        Total Average Score: 5.17       Mean: 14.62       Min: 8.41       Max: 18.7
Episode 70        Total Average Score: 5.17
Episode 71        Total Average Score: 5.32       Mean: 15.44       Min: 10.01       Max: 23.
Episode 72        Total Average Score: 5.45       Mean: 15.19       Min: 9.29       Max: 21.9
Episode 73        Total Average Score: 5.56       Mean: 13.49       Min: 8.71       Max: 19.8
Episode 74        Total Average Score: 5.69       Mean: 15.01       Min: 9.54       Max: 23.9
Episode 75        Total Average Score: 5.82       Mean: 15.56       Min: 8.60       Max: 22.0
Episode 76        Total Average Score: 5.98       Mean: 17.85       Min: 9.57       Max: 27.9
Episode 77        Total Average Score: 6.15       Mean: 19.10       Min: 8.68       Max: 27.4
Episode 78        Total Average Score: 6.30       Mean: 18.04       Min: 7.98       Max: 26.8
Episode 79        Total Average Score: 6.41       Mean: 14.35       Min: 7.41       Max: 22.7
Episode 80        Total Average Score: 6.53       Mean: 16.22       Min: 5.08       Max: 27.8
Episode 80        Total Average Score: 6.53
Episode 81        Total Average Score: 6.68       Mean: 19.20       Min: 11.40       Max: 28.
Episode 82        Total Average Score: 6.83       Mean: 18.83       Min: 12.85       Max: 29.
Episode 83        Total Average Score: 6.99       Mean: 19.92       Min: 11.80       Max: 27.
Episode 84        Total Average Score: 7.17       Mean: 21.82       Min: 8.97       Max: 39.1
Episode 85        Total Average Score: 7.33       Mean: 20.68       Min: 12.41       Max: 29.
Episode 86        Total Average Score: 7.50       Mean: 21.96       Min: 11.37       Max: 31.
Episode 87        Total Average Score: 7.66       Mean: 21.67       Min: 12.82       Max: 27.
Episode 88        Total Average Score: 7.83       Mean: 22.82       Min: 14.61       Max: 29.
Episode 89        Total Average Score: 7.98       Mean: 21.46       Min: 16.03       Max: 29.
Episode 90        Total Average Score: 8.15       Mean: 22.86       Min: 15.89       Max: 31.
Episode 90        Total Average Score: 8.15
Episode 91        Total Average Score: 8.32       Mean: 23.99       Min: 16.12       Max: 38.
Episode 92        Total Average Score: 8.51       Mean: 25.71       Min: 13.08       Max: 39.
Episode 93        Total Average Score: 8.69       Mean: 24.60       Min: 16.72       Max: 35.
Episode 94        Total Average Score: 8.87       Mean: 26.38       Min: 18.28       Max: 37.
Episode 95        Total Average Score: 9.05       Mean: 25.36       Min: 14.94       Max: 31.
Episode 96        Total Average Score: 9.22       Mean: 26.02       Min: 19.93       Max: 30.
Episode 97        Total Average Score: 9.41       Mean: 27.11       Min: 21.53       Max: 31.
Episode 98        Total Average Score: 9.58       Mean: 26.55       Min: 21.78       Max: 33.
Episode 99        Total Average Score: 9.76       Mean: 27.44       Min: 21.11       Max: 34.
Episode 100        Total Average Score: 9.95       Mean: 28.19       Min: 19.54       Max: 37
Episode 100        Total Average Score: 9.95
Episode 101        Total Average Score: 10.22       Mean: 28.16       Min: 19.90       Max: 3
Episode 102        Total Average Score: 10.50       Mean: 28.07       Min: 23.21       Max: 3
Episode 103        Total Average Score: 10.77       Mean: 27.99       Min: 23.93       Max: 3
Episode 104        Total Average Score: 11.05       Mean: 28.63       Min: 23.26       Max: 3
Episode 105        Total Average Score: 11.33       Mean: 28.82       Min: 11.94       Max: 3
Episode 106        Total Average Score: 11.63       Mean: 30.49       Min: 24.58       Max: 3
```

```
Episode 107      Total Average Score: 11.91      Mean: 29.20      Min: 20.82      Max: 3
Episode 108      Total Average Score: 12.19      Mean: 28.89      Min: 23.98      Max: 3
Episode 109      Total Average Score: 12.48      Mean: 30.98      Min: 27.25      Max: 3
Episode 110      Total Average Score: 12.77      Mean: 29.68      Min: 24.00      Max: 3
Episode 110      Total Average Score: 12.77
Episode 111      Total Average Score: 13.06      Mean: 30.52      Min: 24.51      Max: 3
Episode 112      Total Average Score: 13.37      Mean: 31.23      Min: 24.61      Max: 3
Episode 113      Total Average Score: 13.69      Mean: 33.05      Min: 29.27      Max: 3
Episode 114      Total Average Score: 13.99      Mean: 31.65      Min: 22.70      Max: 3
Episode 115      Total Average Score: 14.30      Mean: 31.31      Min: 14.36      Max: 3
Episode 116      Total Average Score: 14.61      Mean: 32.70      Min: 28.65      Max: 3
Episode 117      Total Average Score: 14.92      Mean: 32.23      Min: 26.77      Max: 3
Episode 118      Total Average Score: 15.23      Mean: 33.05      Min: 28.82      Max: 3
Episode 119      Total Average Score: 15.54      Mean: 32.46      Min: 24.74      Max: 3
Episode 120      Total Average Score: 15.84      Mean: 31.60      Min: 22.82      Max: 3
Episode 120      Total Average Score: 15.84
Episode 121      Total Average Score: 16.14      Mean: 31.67      Min: 22.26      Max: 3
Episode 122      Total Average Score: 16.46      Mean: 34.15      Min: 28.22      Max: 3
Episode 123      Total Average Score: 16.78      Mean: 33.02      Min: 26.87      Max: 3
Episode 124      Total Average Score: 17.10      Mean: 34.62      Min: 30.68      Max: 3
Episode 125      Total Average Score: 17.42      Mean: 33.63      Min: 29.76      Max: 3
Episode 126      Total Average Score: 17.74      Mean: 34.22      Min: 28.36      Max: 3
Episode 127      Total Average Score: 18.07      Mean: 35.50      Min: 31.69      Max: 3
Episode 128      Total Average Score: 18.38      Mean: 34.26      Min: 31.54      Max: 3
Episode 129      Total Average Score: 18.72      Mean: 35.38      Min: 32.86      Max: 3
Episode 130      Total Average Score: 19.04      Mean: 34.55      Min: 26.34      Max: 3
Episode 130      Total Average Score: 19.04
Episode 131      Total Average Score: 19.36      Mean: 35.44      Min: 31.65      Max: 3
Episode 132      Total Average Score: 19.67      Mean: 34.58      Min: 30.84      Max: 3
Episode 133      Total Average Score: 19.99      Mean: 35.42      Min: 32.91      Max: 3
Episode 134      Total Average Score: 20.31      Mean: 35.48      Min: 30.98      Max: 3
Episode 135      Total Average Score: 20.63      Mean: 36.30      Min: 30.54      Max: 3
Episode 136      Total Average Score: 20.94      Mean: 35.22      Min: 31.97      Max: 3
Episode 137      Total Average Score: 21.26      Mean: 35.70      Min: 33.55      Max: 3
Episode 138      Total Average Score: 21.58      Mean: 36.39      Min: 31.93      Max: 3
Episode 139      Total Average Score: 21.91      Mean: 36.90      Min: 33.94      Max: 3
Episode 140      Total Average Score: 22.23      Mean: 36.54      Min: 32.31      Max: 3
Episode 140      Total Average Score: 22.23
Episode 141      Total Average Score: 22.53      Mean: 35.50      Min: 29.89      Max: 3
Episode 142      Total Average Score: 22.81      Mean: 35.07      Min: 31.72      Max: 3
Episode 143      Total Average Score: 23.13      Mean: 36.52      Min: 32.19      Max: 3
Episode 144      Total Average Score: 23.43      Mean: 35.63      Min: 31.69      Max: 3
Episode 145      Total Average Score: 23.72      Mean: 35.60      Min: 31.95      Max: 3
Episode 146      Total Average Score: 24.01      Mean: 36.16      Min: 33.58      Max: 3
Episode 147      Total Average Score: 24.31      Mean: 37.30      Min: 35.45      Max: 3
Episode 148      Total Average Score: 24.59      Mean: 34.73      Min: 22.48      Max: 3
Episode 149      Total Average Score: 24.87      Mean: 35.84      Min: 32.22      Max: 3
Episode 150      Total Average Score: 25.17      Mean: 37.62      Min: 34.45      Max: 3
```

```
Episode 150        Total Average Score: 25.17
Episode 151        Total Average Score: 25.46        Mean: 37.41        Min: 34.91        Max: 3
Episode 152        Total Average Score: 25.75        Mean: 37.57        Min: 33.36        Max: 3
Episode 153        Total Average Score: 26.04        Mean: 37.28        Min: 32.34        Max: 3
Episode 154        Total Average Score: 26.30        Mean: 35.56        Min: 32.22        Max: 3
Episode 155        Total Average Score: 26.57        Mean: 37.05        Min: 35.31        Max: 3
Episode 156        Total Average Score: 26.85        Mean: 35.85        Min: 34.31        Max: 3
Episode 157        Total Average Score: 27.13        Mean: 36.39        Min: 33.80        Max: 3
Episode 158        Total Average Score: 27.42        Mean: 37.85        Min: 35.58        Max: 3
Episode 159        Total Average Score: 27.69        Mean: 36.95        Min: 33.52        Max: 3
Episode 160        Total Average Score: 27.96        Mean: 37.79        Min: 34.55        Max: 3
Episode 160        Total Average Score: 27.96
Episode 161        Total Average Score: 28.23        Mean: 38.46        Min: 36.21        Max: 3
Episode 162        Total Average Score: 28.49        Mean: 37.75        Min: 35.65        Max: 3
Episode 163        Total Average Score: 28.75        Mean: 37.95        Min: 36.79        Max: 3
Episode 164        Total Average Score: 29.02        Mean: 37.28        Min: 34.44        Max: 3
Episode 165        Total Average Score: 29.29        Mean: 37.75        Min: 32.75        Max: 3
Episode 166        Total Average Score: 29.50        Mean: 34.45        Min: 28.86        Max: 3
Episode 167        Total Average Score: 29.75        Mean: 37.14        Min: 33.97        Max: 3
Episode 168        Total Average Score: 29.98        Mean: 37.56        Min: 36.11        Max: 3
Episode 169        Total Average Score: 30.22        Mean: 36.90        Min: 34.74        Max: 3
Problem Solved after 169 epsisodes!! Total Average score: 30.22
```

In [22]:
```python
fig = plt.figure()
ax = fig.add_subplot(1,1,1)

plt.plot(np.arange(1, len(scores)+1), scores)
plt.ylabel('Score')
plt.xlabel('Episode #')
plt.show()
```




```
In [23]: # Load the saved weights into Pytorch model
         agent.actor_local.load_state_dict(torch.load('checkpoint_actor.pth', map_location='cpu'
         agent.critic_local.load_state_dict(torch.load('checkpoint_critic.pth', map_location='cp

         env_info = env.reset(train_mode=False)[brain_name]     # reset the environment
         states = env_info.vector_observations                  # get the current state (for eac
         scores = np.zeros(num_agents)                          # initialize the score (for each

         while True:
             actions = agent.act(states)                        # select actions from loaded mod
             env_info = env.step(actions)[brain_name]           # send all actions to tne enviro
             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 age
             states = next_states                               # roll over states to next time
             if np.any(dones):                                  # exit loop if episode finished
                 break
         print('Total score: {}'.format(np.mean(scores)))
```

Total score: 36.58749918220565

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