Result

September 17, 2022

1 Collaboration and Competition

In this notebook, you will learn how to use the Unity ML-Agents environment for the third 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.

[ ]:
```
from unityagents import UnityEnvironment
import numpy as np
```

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/Tennis.app"`
- **Windows** (x86): `"path/to/Tennis_Windows_x86/Tennis.exe"`
- **Windows** (x86_64): `"path/to/Tennis_Windows_x86_64/Tennis.exe"`
- **Linux** (x86): `"path/to/Tennis_Linux/Tennis.x86"`
- **Linux** (x86_64): `"path/to/Tennis_Linux/Tennis.x86_64"`
- **Linux** (x86, headless): `"path/to/Tennis_Linux_NoVis/Tennis.x86"`
- **Linux** (x86_64, headless): `"path/to/Tennis_Linux_NoVis/Tennis.x86_64"`

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

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

[ ]:
```
env = UnityEnvironment(file_name="Tennis.app")
```

```
INFO:unityagents:
'Academy' started successfully!
Unity Academy name: Academy
        Number of Brains: 1
        Number of External Brains : 1
        Lesson number : 0
        Reset Parameters :
```

```
Unity brain name: TennisBrain
        Number of Visual Observations (per agent): 0
        Vector Observation space type: continuous
        Vector Observation space size (per agent): 8
        Number of stacked Vector Observation: 3
        Vector Action space type: continuous
        Vector Action space size (per agent): 2
        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.

[ ]:
```
# 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, two agents control rackets to bounce a ball over a net. If an agent hits the ball over the net, it receives a reward of +0.1. If an agent lets a ball hit the ground or hits the ball out of bounds, it receives a reward of -0.01. Thus, the goal of each agent is to keep the ball in play.

The observation space consists of 8 variables corresponding to the position and velocity of the ball and racket. Two continuous actions are available, corresponding to movement toward (or away from) the net, and jumping.

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

[ ]:
```
# 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], state_size))
print('The state for the first agent looks like:', states[0])
```

```
Number of agents: 2
Size of each action: 2
```

```
There are 2 agents. Each observes a state with length: 24
The state for the first agent looks like: [ 0.          0.          0.
0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.         -6.65278625 -1.5
 -0.          0.          6.83172083  6.         -0.          0.        ]
```

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 agents and receive feedback from the environment.

Once this cell is executed, you will watch the agents' performance, if they select actions at random with each time step. A window should pop up that allows you to observe the agents.

Of course, as part of the project, you'll have to change the code so that the agents are able to use their experiences to gradually choose better actions when interacting with the environment!

[ ]:
```
for i in range(1, 6):                                      # play game for 5␣
↪episodes
    env_info = env.reset(train_mode=False)[brain_name]     # reset the␣
↪environment
    states = env_info.vector_observations                  # get the current␣
↪state (for each agent)
    scores = np.zeros(num_agents)                          # initialize the␣
↪score (for each agent)
    while True:
        actions = np.random.randn(num_agents, action_size) # select an action␣
↪(for each agent)
        actions = np.clip(actions, -1, 1)                  # all actions␣
↪between -1 and 1
        env_info = env.step(actions)[brain_name]           # send all actions␣
↪to tne environment
        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 agent)
        states = next_states                               # roll over states␣
↪to next time step
        if np.any(dones):                                  # exit loop if␣
↪episode finished
            break
    print('Score (max over agents) from episode {}: {}'.format(i, np.
↪max(scores)))
```

When finished, you can close the environment.

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

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]
```

[ ]:
```
from collections import deque
from ddpg_agent import Agent
from maddpg import MADDPG
import torch
import matplotlib.pyplot as plt
%matplotlib inline
```

[ ]:
```
agent = MADDPG(state_size=state_size, action_size=action_size, random_seed=0)

def ddpg(n_episodes=10000, print_every=100):
    scores_deque = deque(maxlen=print_every)
    scores = []
    for i_episode in range(1, n_episodes+1):
        env_info = env.reset(train_mode=True)[brain_name]
        states = env_info.vector_observations
        agent.reset()
        score = np.zeros(num_agents)
        while True:
            actions = agent.act(states)
            env_info = env.step(actions)[brain_name]
            next_states = env_info.vector_observations
            rewards = env_info.rewards
            dones = env_info.local_done
            agent.step(states, actions, rewards, next_states, dones)
            states = next_states
            score += rewards
            if np.any(dones):
                break

        scores_deque.append(np.max(score))
        scores.append(np.max(score))
        print('\rEpisode {}\tAverage Score: {:.2f}'.format(i_episode, np.
↪mean(scores_deque)), end="")
        if i_episode % print_every == 0:
            print('\rEpisode {}\tAverage Score: {:.2f}'.format(i_episode, np.
↪mean(scores_deque)))
            agent.save()
```

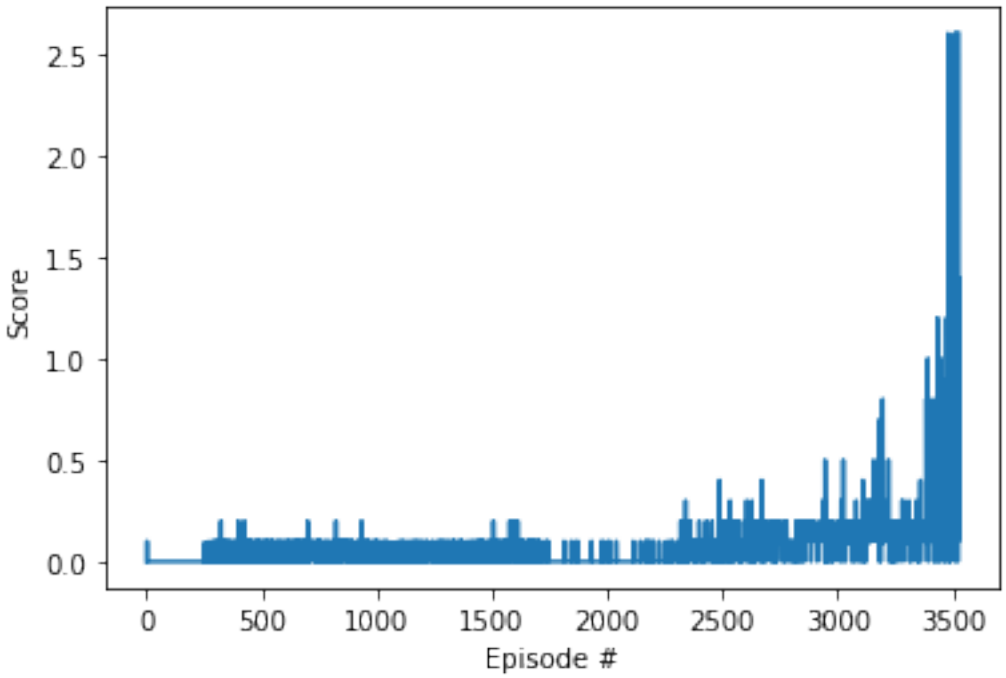
```
        if np.mean(scores_deque)>=0.5:
            print('\nEnvironment solved in {:d} episodes!\tAverage Score: {:.
↪2f}'.format(i_episode, np.mean(scores_deque)))
            agent.save()
            break
    return scores

scores = ddpg()

# plot the scores
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()
```

```
Episode 100     Average Score: 0.00     Current Score: 0.00
Episode 200     Average Score: 0.00     Current Score: 0.00
Episode 300     Average Score: 0.02     Current Score: 0.09
Episode 400     Average Score: 0.06     Current Score: 0.20
Episode 500     Average Score: 0.05     Current Score: 0.00
Episode 600     Average Score: 0.05     Current Score: 0.00
Episode 700     Average Score: 0.05     Current Score: 0.20
Episode 800     Average Score: 0.06     Current Score: 0.00
Episode 900     Average Score: 0.06     Current Score: 0.09
Episode 1000    Average Score: 0.04     Current Score: 0.10
Episode 1100    Average Score: 0.03     Current Score: 0.09
Episode 1200    Average Score: 0.04     Current Score: 0.00
Episode 1300    Average Score: 0.02     Current Score: 0.00
Episode 1400    Average Score: 0.03     Current Score: 0.00
Episode 1500    Average Score: 0.04     Current Score: 0.09
Episode 1600    Average Score: 0.07     Current Score: 0.20
Episode 1700    Average Score: 0.05     Current Score: 0.10
Episode 1800    Average Score: 0.01     Current Score: 0.00
Episode 1900    Average Score: 0.00     Current Score: 0.00
Episode 2000    Average Score: 0.00     Current Score: 0.00
Episode 2100    Average Score: 0.00     Current Score: 0.00
Episode 2200    Average Score: 0.01     Current Score: 0.10
Episode 2300    Average Score: 0.01     Current Score: 0.00
Episode 2400    Average Score: 0.07     Current Score: 0.10
Episode 2500    Average Score: 0.08     Current Score: 0.00
Episode 2600    Average Score: 0.09     Current Score: 0.00
Episode 2700    Average Score: 0.11     Current Score: 0.10
Episode 2800    Average Score: 0.10     Current Score: 0.10
Episode 2900    Average Score: 0.10     Current Score: 0.20
Episode 3000    Average Score: 0.11     Current Score: 0.10
```

```
Episode 3100    Average Score: 0.11     Current Score: 0.10
Episode 3200    Average Score: 0.16     Current Score: 0.30
Episode 3300    Average Score: 0.11     Current Score: 0.10
Episode 3400    Average Score: 0.14     Current Score: 0.10
Episode 3500    Average Score: 0.30     Current Score: 1.00
Episode 3525    Average Score: 0.50     Current Score: 2.60
Environment solved in 3525 episodes!    Average Score: 0.50
```



[ ]:
```
agent = MADDPG(state_size=state_size, action_size=action_size, random_seed=0)
agent.load()

for _ in range(10):
    env_info = env.reset(train_mode=False)[brain_name]     # reset the␣
↪environment
    states = env_info.vector_observations                  # get the current␣
↪state (for each agent)
    scores = np.zeros(num_agents)                          # initialize the␣
↪score (for each agent)
    while True:
        actions = agent.act(states)
        env_info = env.step(actions)[brain_name]           # send all actions␣
↪to tne environment
        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 agent)
        states = next_states                               # roll over states␣
↪to next time step
        if np.any(dones):                                  # exit loop if␣
↪episode finished
            break
    print('Total score (averaged over agents) this episode: {}'.format(np.
↪max(scores)))
```

[ ]: