Navigation

April 23, 2020

1 Navigation

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

1.0.1 1. Start the Environment

We begin by importing some 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.

[1]:
```
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/Banana.app"`
- **Windows** (x86): `"path/to/Banana_Windows_x86/Banana.exe"`
- **Windows** (x86_64): `"path/to/Banana_Windows_x86_64/Banana.exe"`
- **Linux** (x86): `"path/to/Banana_Linux/Banana.x86"`
- **Linux** (x86_64): `"path/to/Banana_Linux/Banana.x86_64"`
- **Linux** (x86, headless): `"path/to/Banana_Linux_NoVis/Banana.x86"`
- **Linux** (x86_64, headless): `"path/to/Banana_Linux_NoVis/Banana.x86_64"`

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

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

[2]:
```
env = UnityEnvironment(file_name="./Banana_Linux/Banana.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 :
```

```
Unity brain name: BananaBrain
        Number of Visual Observations (per agent): 0
        Vector Observation space type: continuous
        Vector Observation space size (per agent): 37
        Number of stacked Vector Observation: 1
        Vector Action space type: discrete
        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.

[3]:
```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

The simulation contains a single agent that navigates a large environment. At each time step, it has four actions at its disposal: - `0` - walk forward - `1` - walk backward - `2` - turn left - `3` - turn right

The state space has `37` dimensions and contains the agent's velocity, along with ray-based perception of objects around agent's forward direction. A reward of `+1` is provided for collecting a yellow banana, and a reward of `-1` is provided for collecting a blue banana.

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

[4]:
```python
# reset the environment
env_info = env.reset(train_mode=True)[brain_name]

# number of agents in the environment
print('Number of agents:', len(env_info.agents))

# number of actions
action_size = brain.vector_action_space_size
print('Number of actions:', action_size)

# examine the state space
state = env_info.vector_observations[0]
print('States look like:', state)
state_size = len(state)
print('States have length:', state_size)
```

```
Number of agents: 1
Number of actions: 4
States look like: [1.         0.         0.         0.         0.84408134 0.
 0.         1.         0.         0.0748472  0.         1.
 0.         0.         0.25755    1.         0.         0.
 0.         0.74177343 0.         1.         0.         0.
```

```
 0.25854847 0.         0.         1.         0.         0.09355672
 0.         1.         0.         0.         0.31969345 0.
 0.        ]
States have length: 37
```

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 (uniformly) 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!

[5]:
```
env_info = env.reset(train_mode=False)[brain_name] # reset the environment
state = env_info.vector_observations[0]            # get the current state
score = 0                                          # initialize the score
while True:
    action = np.random.randint(action_size)        # select an action
    env_info = env.step(action)[brain_name]        # send the action to the␣
 ↪environment
    next_state = env_info.vector_observations[0]   # get the next state
    reward = env_info.rewards[0]                   # get the reward
    done = env_info.local_done[0]                  # see if episode has finished
    score += reward                                # update the score
    state = next_state                             # roll over the state to␣
 ↪next time step
    if done:                                       # exit loop if episode␣
 ↪finished
        break

print("Score: {}".format(score))
```

```
Score: 0.0
```

When finished, you can close the environment.

[3]:
```
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]
```

[10]:
```
import torch
import numpy as np
import matplotlib.pyplot as plt
from unityagents import UnityEnvironment

from agent import Agent, QNet
```

[2]:
```
default_env_name = "./Banana_Linux/Banana.x86_64"
default_episodes = 5000
```

[3]:
```
env = UnityEnvironment(file_name=default_env_name)
```

```
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: BananaBrain
        Number of Visual Observations (per agent): 0
        Vector Observation space type: continuous
        Vector Observation space size (per agent): 37
        Number of stacked Vector Observation: 1
        Vector Action space type: discrete
        Vector Action space size (per agent): 4
        Vector Action descriptions: , , ,
```

[4]:
```
# get the default brain
brain_name = env.brain_names[0]
brain = env.brains[brain_name]
brain_name
```

[4]:
```
'BananaBrain'
```

[5]:
```
# obtain initial observation
env_info = env.reset(train_mode=True)[brain_name]

# number of agents in the environment
print('Number of agents:', len(env_info.agents))

# number of actions
action_size = brain.vector_action_space_size
print('Number of actions:', action_size)

# observation space size
```

```
state = env_info.vector_observations[0]
state_size = len(state)
print('States have length:', state_size)
```

```
Number of agents: 1
Number of actions: 4
States have length: 37
```

[11]:
```
def train_dqn(agent, env, brain_name, n_episodes=2000, max_t=100, eps_start=1.
↪0, eps_end=0.01, eps_decay=0.995):
    episode_scores = []
    total_scores = []
    eps = eps_start
    for i_episode in range(1, n_episodes+1):
        env_info = env.reset(train_mode=True)[brain_name]
        state = env_info.vector_observations[0]
        score = 0
        for t in range(max_t):
            action = int(agent.act(state, eps))
            env_info = env.step(action)[brain_name]
            next_state = env_info.vector_observations[0]
            reward = env_info.rewards[0]
            done = env_info.local_done[0]
            agent.step(state, action, reward, next_state, done)
            score += reward
            state = next_state
            if done:
                break
        episode_scores.append(score)
        total_scores.append(score)
        eps = max(eps_end, eps_decay*eps)
        print('\rEpisode {}\tAverage Score: {:.2f}'.format(i_episode, np.
↪mean(episode_scores)), end="")

        if i_episode % 100 == 0:
            print('\rEpisode {}\tAverage Episode Score: {:.2f}'.
↪format(i_episode, np.mean(episode_scores)))

        if i_episode % 1000 == 0:
            print('\nSaving target net {:d} episodes!\tAverage Episode Score: {:
↪.2f}\tAverage Total Score: {:.2f}'.format(i_episode, np.
↪mean(episode_scores), np.mean(total_scores)))
            torch.save(agent.target_net.state_dict(), 'target_model.pth')

        if np.mean(episode_scores) >= 13.0:
            print('\nEnvironment solved in {:d} episodes!\tAverage Episode␣
↪Score: {:.2f}'.format(i_episode, np.mean(episode_scores)))
```

```
            torch.save(agent.policy_net.state_dict(), 'policy_model.pth')
            break
        episode_scores = []
    return total_scores
```

[12]:
```
agent = Agent(state_size, action_size=action_size, batch_size=64,␣
 ↪update_every=4)
```

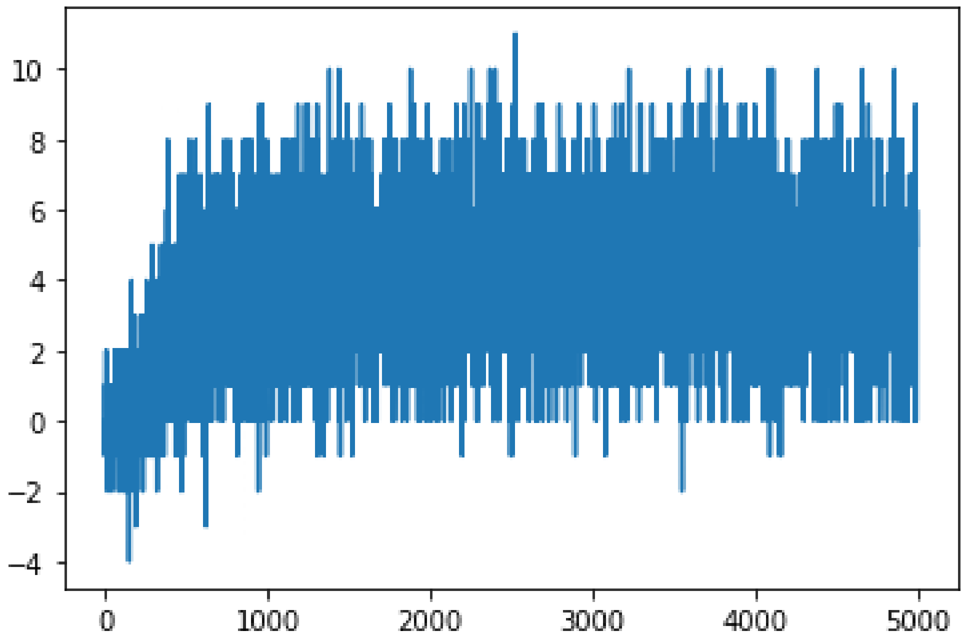
[13]:
```
scores = train_dqn(agent, env, brain_name, 5000)
```

```
Episode 100     Average Episode Score: 0.00
Episode 200     Average Episode Score: 0.00
Episode 300     Average Episode Score: 0.00
Episode 400     Average Episode Score: 1.00
Episode 500     Average Episode Score: 1.00
Episode 600     Average Episode Score: 3.00
Episode 700     Average Episode Score: 2.00
Episode 800     Average Episode Score: 2.00
Episode 900     Average Episode Score: 2.00
Episode 1000    Average Episode Score: 5.00

Saving target net 1000 episodes!        Average Episode Score: 5.00     Average
Total Score: 2.23
Episode 1100    Average Episode Score: 4.00
Episode 1200    Average Episode Score: 0.00
Episode 1300    Average Episode Score: 6.00
Episode 1400    Average Episode Score: 2.00
Episode 1500    Average Episode Score: 5.00
Episode 1600    Average Episode Score: 6.00
Episode 1700    Average Episode Score: 5.00
Episode 1800    Average Episode Score: 5.00
Episode 1900    Average Episode Score: 8.00
Episode 2000    Average Episode Score: 3.00

Saving target net 2000 episodes!        Average Episode Score: 3.00     Average
Total Score: 3.18
Episode 2100    Average Episode Score: 5.00
Episode 2200    Average Episode Score: 6.00
Episode 2300    Average Episode Score: 3.00
Episode 2400    Average Episode Score: 6.00
Episode 2500    Average Episode Score: 1.00
Episode 2600    Average Episode Score: 5.00
Episode 2700    Average Episode Score: 2.00
Episode 2800    Average Episode Score: 4.00
Episode 2900    Average Episode Score: 5.00
Episode 3000    Average Episode Score: 5.00
```

```
Saving target net 3000 episodes!        Average Episode Score: 5.00     Average
Total Score: 3.59
Episode 3100    Average Episode Score: 6.00
Episode 3200    Average Episode Score: 7.00
Episode 3300    Average Episode Score: 4.00
Episode 3400    Average Episode Score: 5.00
Episode 3500    Average Episode Score: 7.00
Episode 3600    Average Episode Score: 6.00
Episode 3700    Average Episode Score: 6.00
Episode 3800    Average Episode Score: 6.00
Episode 3900    Average Episode Score: 8.00
Episode 4000    Average Episode Score: 4.00

Saving target net 4000 episodes!        Average Episode Score: 4.00     Average
Total Score: 3.88
Episode 4100    Average Episode Score: 2.00
Episode 4200    Average Episode Score: 3.00
Episode 4300    Average Episode Score: 4.00
Episode 4400    Average Episode Score: 1.00
Episode 4500    Average Episode Score: 5.00
Episode 4600    Average Episode Score: 7.00
Episode 4700    Average Episode Score: 2.00
Episode 4800    Average Episode Score: 4.00
Episode 4900    Average Episode Score: 4.00
Episode 5000    Average Episode Score: 5.00

Saving target net 5000 episodes!        Average Episode Score: 5.00     Average
Total Score: 4.00
```

[14]:

```python
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(np.arange(len(scores)), scores)

plt.show()
```



1.1 Testing the Agent

[15]:
```
agent = Agent(state_size, action_size=action_size, batch_size=64)
agent.policy_net.load_state_dict(torch.load('target_model.pth'))

for i in range(3):
    env_info = env.reset(train_mode=False)[brain_name]
    state = env_info.vector_observations[0]
    score = 0
    for j in range(200):
        action = agent.act(state)
        env_info = env.step(action)[brain_name]
        next_state = env_info.vector_observations[0]
        reward = env_info.rewards[0]
        done = env_info.local_done[0]
        agent.step(state, action, reward, next_state, done)
        score += reward
        state = next_state
        if done:
            break
    print("Episode {} is done total score is {}".format(i+1, score))
```

```
Episode 1 is done total score is 8.0
Episode 2 is done total score is 2.0
Episode 3 is done total score is 1.0
```

[16]:
```
env.close()
```

1.1.1 Ways to Improve

The Agent needs more training episodes, also giving it eyes :) like capturing the state visually and feeding it to the Agent neural network.