Navigation_Pixels

July 18, 2021

1 Navigation

Congratulations for completing the first project of the Deep Reinforcement Learning Nanodegree! In this notebook, you will learn how to control an agent in a more challenging environment, where it can learn directly from raw pixels! **Note that this exercise is optional!**

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.

[12]:
```
from unityagents import UnityEnvironment
import numpy as np
import torch
import matplotlib.pyplot as plt
%matplotlib inline

from lib.agents import AgentExperienceReplay, AgentPrioritizedExperienceReplay
from lib.models import QNetwork, DuelingQNetwork, ConvQNetwork,␣
 ↪DuelingConvQNetwork
from lib.dqn import dqn
```

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/VisualBanana.app"`
- **Windows** (x86): `"path/to/VisualBanana_Windows_x86/Banana.exe"`
- **Windows** (x86_64): `"path/to/VisualBanana_Windows_x86_64/Banana.exe"`
- **Linux** (x86): `"path/to/VisualBanana_Linux/Banana.x86"`
- **Linux** (x86_64): `"path/to/VisualBanana_Linux/Banana.x86_64"`
- **Linux** (x86, headless): `"path/to/VisualBanana_Linux_NoVis/Banana.x86"`
- **Linux** (x86_64, headless): `"path/to/VisualBanana_Linux_NoVis/Banana.x86_64"`

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

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

[2]:
```
#env = UnityEnvironment(file_name="VisualBanana_Linux/Banana.x86_64")
env = UnityEnvironment(file_name="VisualBanana.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: BananaBrain
        Number of Visual Observations (per agent): 1
        Vector Observation space type: continuous
        Vector Observation space size (per agent): 0
        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]:
```
# 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

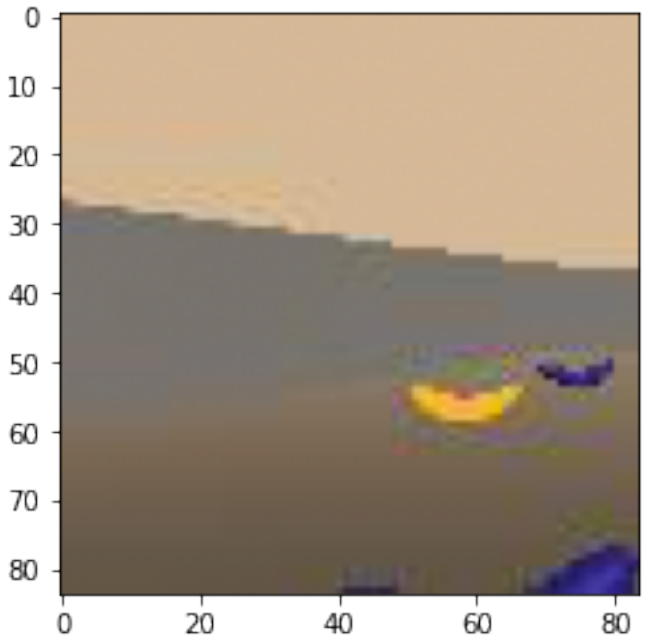
The environment state is an array of raw pixels with shape `(1, 84, 84, 3)`. *Note that this code differs from the notebook for the project, where we are grabbing* ***`visual_observations`*** *(the raw pixels) instead of* ***`vector_observations`***. A reward of `+1` is provided for collecting a yellow 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]:
```
# 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.visual_observations[0]
print('States look like:')
plt.imshow(np.squeeze(state))
plt.show()
state_size = state.shape
print('States have shape:', state.shape)
```

```
Number of agents: 1
Number of actions: 4
States look like:
```

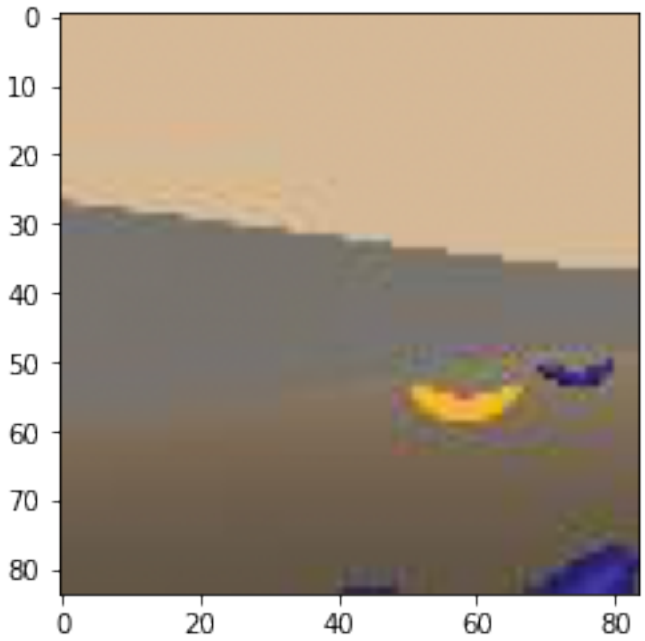



```
States have shape: (1, 84, 84, 3)
```

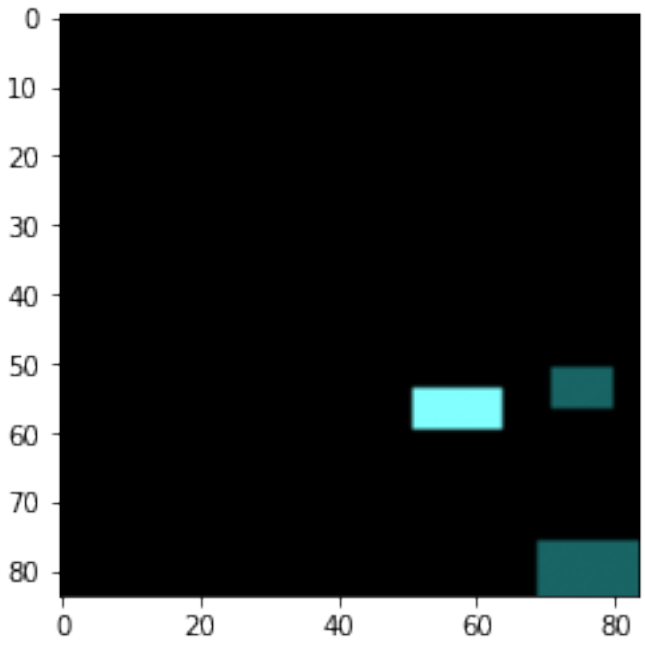
[ ]:
```
# reset the environment
env_info = env.reset(train_mode=True)[brain_name]
action_size = brain.vector_action_space_size
```

[5]:
```
from lib.image import process_image

state = env_info.visual_observations[0]
#blue_boundaries=([100,150,0], [140,255,255])
blue_boundaries=([120,145,50], [130,255,130])

img = state[0]
```

```
plt.imshow(img)
plt.show()

#yellow_boundaries=([20, 100, 100], [30, 117, 117])
yellow_boundaries=([18, 176, 235], [22, 227, 255])

processed_image, maxYellowArea, maxBlueArea = process_image(img,
                                                            ␣
 ↪colour1_boundaries=blue_boundaries,
                                                            ␣
 ↪colour2_boundaries=yellow_boundaries)
print('maxYellowArea', maxYellowArea, 'maxBlueArea', maxBlueArea)
max_area = 84 * 84
print('yellow reward', maxYellowArea/max_area, 'blue reward', -maxBlueArea/
 ↪max_area)
plt.imshow(processed_image)
plt.show()
```




```
maxYellowArea 60 maxBlueArea 120
yellow reward 0.008503401360544218 blue reward -0.017006802721088437
```



1.0.3 Training with a convolution network

Below are the results of attempting to train the agent using a convolutional neural network using double Q learning

The processing was much slower, although it did appear to be learning. Unfortunately the game seemed to stop working after about 600 episodes so wasn't able to assess if this would ultimately produce a score of greater than 13

[13]:
```
def create_model(state_size, action_size, seed):
    # return ConvQNetwork(state_size, action_size, seed, fc_units=[512])
    return ConvQNetwork(state_size, action_size, seed, fc_units=[64, 64])
    # return DuelingConvQNetwork(state_size, action_size, seed)

def get_state(env_info):
    state = env_info.visual_observations[0]
    blue_boundaries=([120,145,50], [130,255,130])
    yellow_boundaries=([18, 176, 235], [22, 227, 255])

    state[0], _, _ = process_image(state[0],
                                   colour1_boundaries=blue_boundaries,
                                   colour2_boundaries=yellow_boundaries)
    color_reward = 0
    return state, color_reward
```

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

# 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 = get_state(env_info)
# print('States look like:', state)
state_size = len(state)
print('States have length:', state_size)

#agent = AgentPrioritizedExperienceReplay(state_size=state_size,
#                                         action_size=action_size,
#                                         seed=0,
#                                         train_mode=True,
#                                         #batch_size=32,
#                                         double_dqn=True,
#                                         create_model=create_model
#                                )

agent = AgentExperienceReplay(state_size=state_size,
                              action_size=action_size,
                              seed=0,
                              train_mode=True,
                              #batch_size=32,
                              double_dqn=True,
                              create_model=create_model
                               )

agent_name = 'pixel_conv'

# agent.load_model(agent_name)
n_episodes = 600


scores = dqn(env,
             brain_name,
             agent,
             n_episodes=n_episodes,
             #eps_end=0.1,
```

```
            checkpoint=13,
            agent_name=agent_name,
            get_state=get_state,
            batched_state=True)

agent.save_model(agent_name)
```

```
Number of agents: 1
Number of actions: 4
States have length: 2
Episode 100     Average Score: 0.02
Episode 200     Average Score: 0.071
Episode 300     Average Score: 0.42
Episode 400     Average Score: 0.23
Episode 500     Average Score: 0.60
Episode 600     Average Score: 0.60
```

1.0.4 Training with convolution Network and 'adjusted' reward

I experimented with adjusting the reward proportional to the size of the largest banana rectangle of each color. I chose a reward of the area of the largest rectangle for a given color divided by the area of the image. So for yellow bananas this was a positive reward and for blue bananas a negative reward.

The results of the code below illustrates this

[6]:
```
def create_model(state_size, action_size, seed):
    # return ConvQNetwork(state_size, action_size, seed, fc_units=[512])
    return ConvQNetwork(state_size, action_size, seed, fc_units=[64, 64])
    # return DuelingConvQNetwork(state_size, action_size, seed)

def get_state(env_info):
    state = env_info.visual_observations[0]
    blue_boundaries=([120,145,50], [130,255,130])
    yellow_boundaries=([18, 176, 235], [22, 227, 255])

    image_area = state[0].shape[0] * state[0].shape[1]

    state[0], max_yellow_area, max_blue_area = process_image(state[0],
                                                             ␣
 ↪colour1_boundaries=blue_boundaries,
                                                             ␣
 ↪colour2_boundaries=yellow_boundaries)
    # experiment of adjusting the reward with an value proportional to the
    # size of the banana
    # this actually seemed to make training worse
    color_reward = max_yellow_area/image_area + max_blue_area/image_area
    return state, color_reward
```

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

# 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 = get_state(env_info)
# print('States look like:', state)
state_size = len(state)
print('States have length:', state_size)

#agent = AgentPrioritizedExperienceReplay(state_size=state_size,
#                                         action_size=action_size,
#                                         seed=0,
#                                         train_mode=True,
#                                         #batch_size=32,
#                                         double_dqn=True,
#                                         create_model=create_model
#                                )

agent = AgentExperienceReplay(state_size=state_size,
                              action_size=action_size,
                              seed=0,
                              train_mode=True,
                              #batch_size=32,
                              double_dqn=True,
                              create_model=create_model
                               )

agent_name = 'pixel_conv'

# agent.load_model(agent_name)
n_episodes = 600


scores = dqn(env,
             brain_name,
             agent,
```

```
            n_episodes=n_episodes,
            #eps_end=0.1,
            checkpoint=13,
            agent_name=agent_name,
            get_state=get_state,
            batched_state=True)

agent.save_model(agent_name)
```

```
Number of agents: 1
Number of actions: 4
States have length: 2
Episode 100     Average Score: 0.13
Episode 200     Average Score: 0.051
Episode 300     Average Score: 0.012
Episode 400     Average Score: 0.39
Episode 500     Average Score: 0.25
Episode 600     Average Score: 0.22
```