Navigation-Solution

November 9, 2019

1 Navigation

First project of NANODEGREE PROGRAM Become a Deep Reinforcement Learning Expet Udacity

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 [1]: !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 environment is already saved in the Workspace and can be accessed at the file path provided below. Please run the next code cell without making any changes.

```
In [2]: import torch
        import numpy as np
        from unityagents import UnityEnvironment
        import random
        import matplotlib.pyplot as plt
        from collections import deque
        from dqn_agent import Agent

        %matplotlib inline
```

```
In [3]: # please do not modify the line below
        env = UnityEnvironment(file_name="/data/Banana_Linux_NoVis/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.

```
In [4]: # 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 [5]: # 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. initialize the game agent

```
In [6]: agent = Agent(state_size=state_size, action_size=action_size, seed=0)

        output_file_name="double_dueling_agent.pth" # file name under which the weights will be
```

1.0.4 4. Train the Agent with DQN

```
In [10]: def dqn(n_episodes=2000, max_t=1000, eps_start=1.0, eps_end=0.01, eps_decay=0.995):
             """Deep Q-Learning.

             Params
             ======
                 n_episodes (int): maximum number of training episodes
                 max_t (int): maximum number of timesteps per episode
                 eps_start (float): starting value of epsilon, for epsilon-greedy action selecti
                 eps_end (float): minimum value of epsilon
                 eps_decay (float): multiplicative factor (per episode) for decreasing epsilon
             """
             scores = []                        # list containing scores from each episode
             scores_window = deque(maxlen=100)  # last 100 scores
             eps = eps_start                    # initialize epsilon
             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 = 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)
                     state = next_state
                     score += reward
                     if done:
                         break
                 scores_window.append(score)       # save most recent score
                 scores.append(score)              # save most recent score
                 eps = max(eps_end, eps_decay*eps) # decrease epsilon
                 print('\rEpisode {}\tAverage Score: {:.2f}'.format(i_episode, np.mean(scores_wi
                 if i_episode % 100 == 0:
                     print('\rEpisode {}\tAverage Score: {:.2f}'.format(i_episode, np.mean(score
                 if np.mean(scores_window)>=13:
                     print('\nEnvironment solved in {:d} episodes!\tAverage Score: {:.2f}'
```

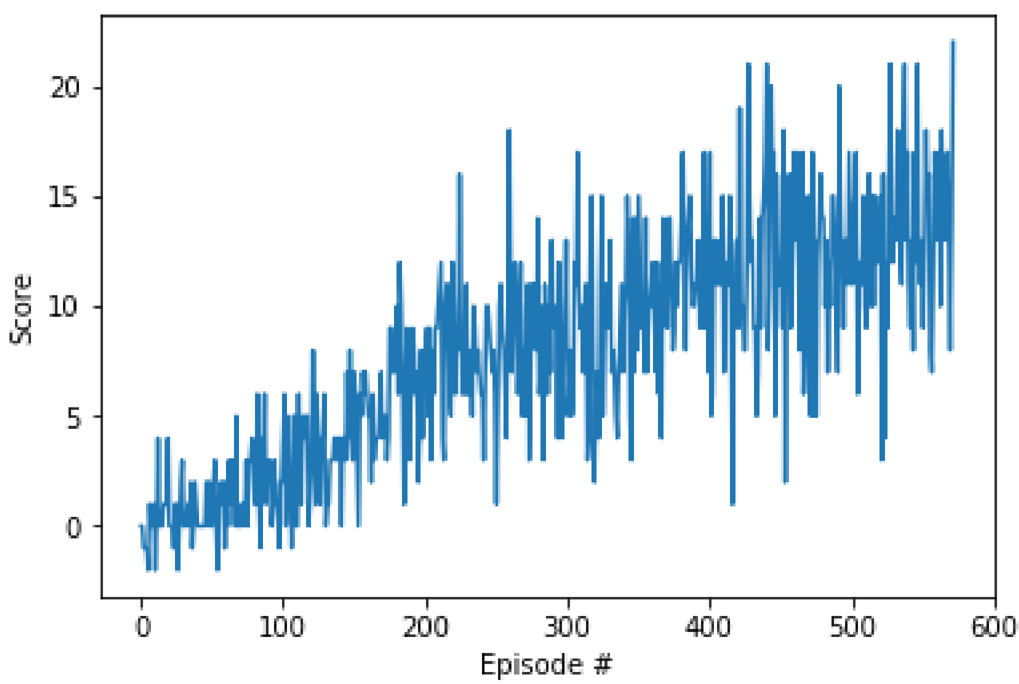
```
                .format(i_episode, np.mean(scores_window)))
            torch.save(agent.qnetwork_local.state_dict(), 'checkpoint.pth')
            break
    return scores
```

1.0.5 4. Start training

```
In [8]: # start training
        scores = dqn()

        # plot the scores
        fig = plt.figure()
        ax = fig.add_subplot(111)
        plt.plot(np.arange(len(scores)), scores)
        plt.ylabel('Score')
        plt.xlabel('Episode #')
        plt.show()
```

```
Episode 100	Average Score: 1.00
Episode 200	Average Score: 4.52
Episode 300	Average Score: 8.26
Episode 400	Average Score: 9.80
Episode 500	Average Score: 11.98
Episode 572	Average Score: 13.00
Environment solved in 572 episodes!	Average Score: 13.00
```



1.0.6 5. Load the trained network weights

```
In [9]: # load the trained network weights
        agent.qnetwork_local.load_state_dict(torch.load('checkpoint.pth'))

        env_info = env.reset(train_mode=True)[brain_name] # reset the environment
        state = env_info.vector_observations[0]            # get the current state
        score = 0                                          # initialize the score
        while True:
            action = agent.act(state)                      # 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 st
            if done:                                       # exit loop if episode finished
                break

        print("Single episode agent score after training: {}".format(score))
        env.close()
```

```
Single episode agent score after training: 10.0
```