Navigation

April 26, 2020

1 Navigation

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 [1]: !pip -q install ./python
```

```
ipython 6.5.0 has requirement prompt-toolkit<2.0.0,>=1.0.15, but you'll have prompt-toolkit 3.0.
```

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 numpy as np
        import torch
        import matplotlib.pyplot as plt

        from collections import deque
        from unityagents import UnityEnvironment
        from agent import Agent

        # 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 [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

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

```
In [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.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.

Note that **in this coding environment, you will not be able to watch the agent while it is training**, and you should set train_mode=True to restart the environment.

```
In [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 st
            if done:                                       # exit loop if episode finished
                break

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

```
Score: 0.0
```

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 agent while it is training. However, ***after training the agent***, you can download the saved model weights to watch the agent on your own machine!

```
In [6]: def dqn(n_episodes=2000, max_t=1000, eps_start=1.0, eps_end=0.01, eps_decay=0.995,
                train_mode=True, ckpt_path='checkpoint.pth'):
            """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 selectic
            eps_end (float): minimum value of epsilon
            eps_decay (float): multiplicative factor (per episode) for decreasing epsilon
            train_mode (bool): run training mode if `True`
        """
        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=train_mode)[brain_name] # reset environment
            state = env_info.vector_observations[0]                 # get current state
            score = 0
            for t in range(max_t):
                action = agent.act(state, eps)                      # select an action
                env_info = env.step(action)[brain_name]             # send action to environ
                next_state = env_info.vector_observations[0]        # get next state
                reward = env_info.rewards[0]                        # get reward
                done = env_info.local_done[0]                       # see if episode has fin
                agent.step(state, action, reward, next_state, done) # learning step
                state = next_state
                score += reward
                if done:
                    break

            scores_window.append(score)          # save most recent score to window
            scores.append(score)                 # save most recent score to total
            eps = max(eps_end, eps_decay*eps)    # decrease epsilon

            print('\rEpisode {}\tAverage Score: {:.2f}'.format(i_episode, np.mean(scores_win
            if i_episode % 100 == 0:
                print('\rEpisode {}\tAverage Score: {:.2f}'.format(i_episode, np.mean(scores
            if np.mean(scores_window) >= 13.0:
                print('\nEnvironment solved in {:d} episodes!\tAverage Score: {:.2f}'.format
                if train_mode: torch.save(agent.qnetwork_local.state_dict(), ckpt_path)
                break
        return scores
```

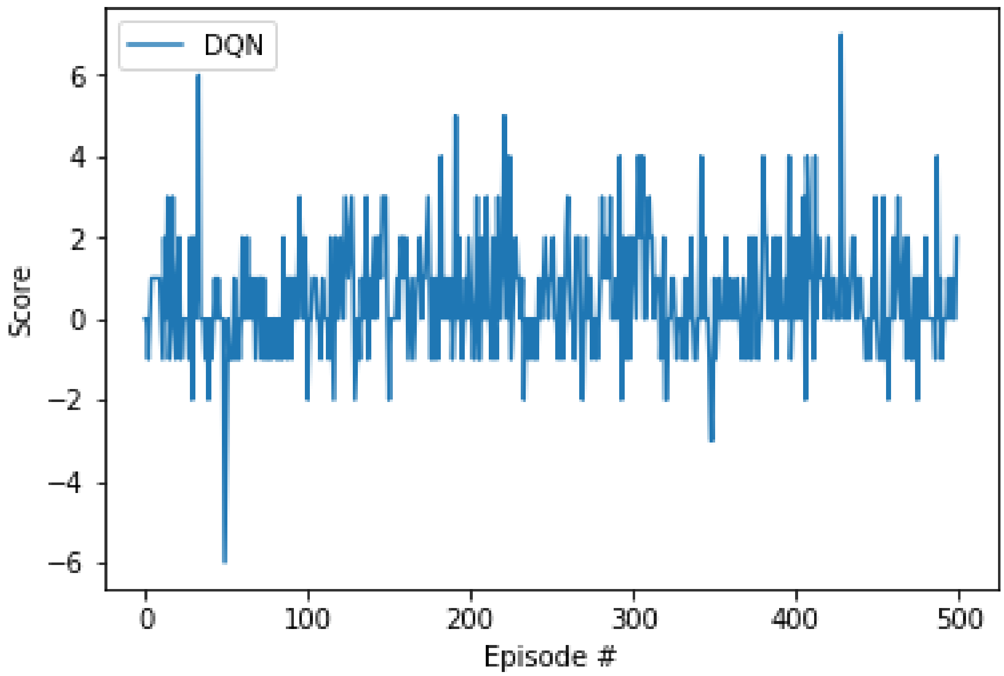
DQN

```
In [9]: agent = Agent(state_size=state_size, action_size=action_size, seed=0)
        scores = dqn(n_episodes=500, eps_decay=0.98, ckpt_path='v1_checkpoint.pth')

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

```
plt.xlabel('Episode #')
plt.legend(loc='upper left')
plt.show()
```

```
Episode 100	Average Score: 0.18
Episode 200	Average Score: 0.68
Episode 300	Average Score: 0.59
Episode 400	Average Score: 0.53
Episode 500	Average Score: 0.60
```



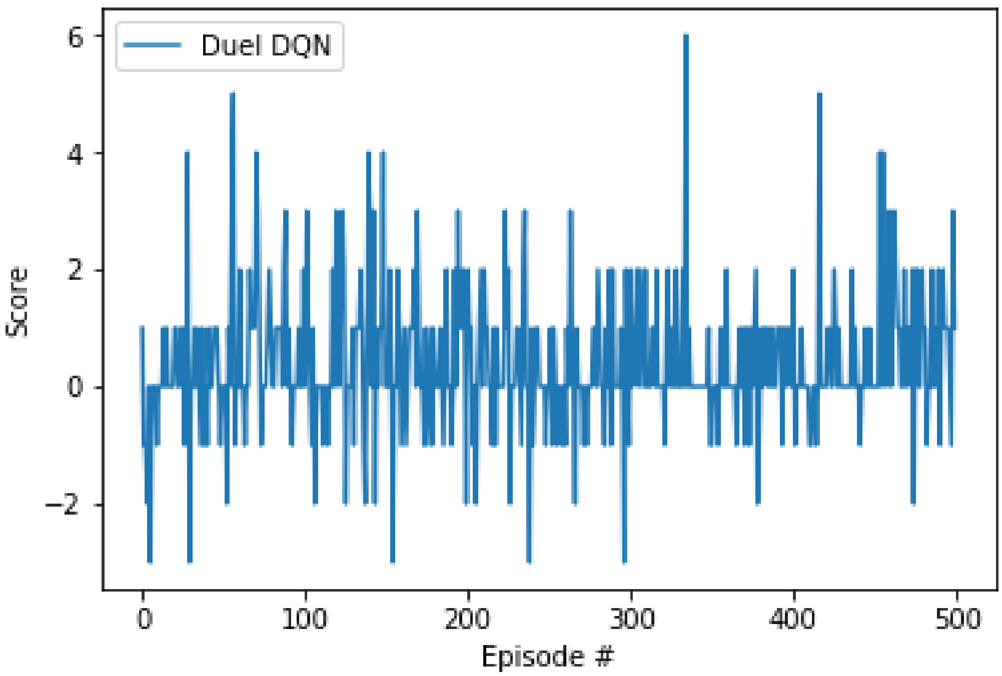
Dueling DQN

```
In [7]: agent = Agent(state_size=state_size, action_size=action_size, seed=0, duel=True)
        scores = dqn(n_episodes=500, eps_decay=0.98, ckpt_path='v2_checkpoint.pth')

        # plot the scores
        fig = plt.figure()
        ax = fig.add_subplot(111)
        plt.plot(np.arange(len(scores)), scores, label='Duel DQN')
        plt.ylabel('Score')
        plt.xlabel('Episode #')
        plt.legend(loc='upper left')
        plt.show()
```

```
Episode 100	Average Score: 0.37
Episode 200	Average Score: 0.51
```

```
Episode 300	Average Score: 0.23
Episode 400	Average Score: 0.44
Episode 500	Average Score: 0.60
```

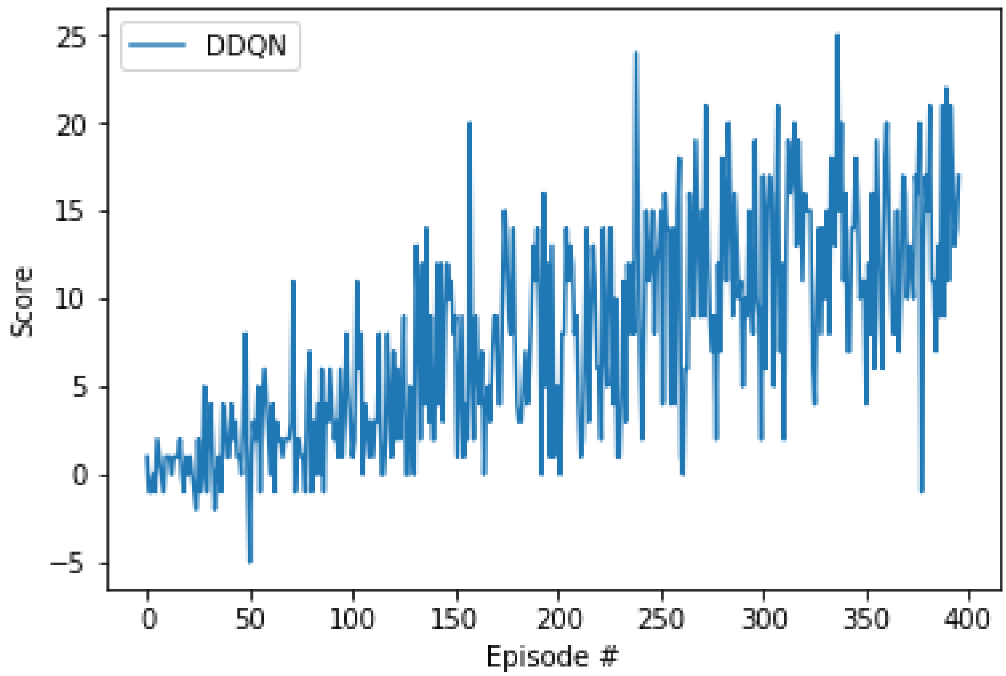


Double DQN

```
In [7]: agent = Agent(state_size=state_size, action_size=action_size, seed=0, double=True)
        scores = dqn(eps_decay=0.98, ckpt_path='v3_checkpoint.pth')

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

```
Episode 100	Average Score: 1.75
Episode 200	Average Score: 6.07
Episode 300	Average Score: 9.88
Episode 396	Average Score: 13.07
Environment solved in 296 episodes!	Average Score: 13.07
```



DDQN + Prioritized Experience Replay

```
In [ ]: agent = Agent(state_size=state_size, action_size=action_size, seed=0, double=True, prior
        scores = dqn(n_episodes=500, eps_decay=0.98, ckpt_path='v4_checkpoint.pth')

        # plot the scores
        fig = plt.figure()
        ax = fig.add_subplot(111)
        plt.plot(np.arange(len(scores)), scores, label='DDQN + PER')
        plt.ylabel('Score')
        plt.xlabel('Episode #')
        plt.legend(loc='upper left')
        plt.show()
```

```
Episode 100       Average Score: 0.77
Episode 159       Average Score: 2.41
```

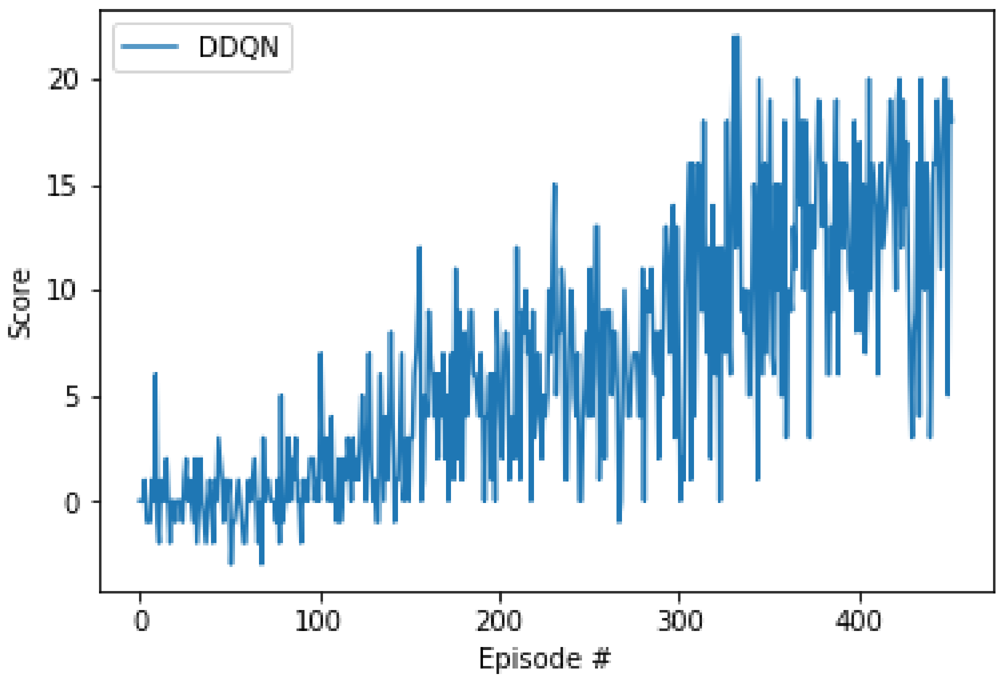
Dueling DDQN

```
In [6]: agent = Agent(state_size=state_size, action_size=action_size, seed=0, double=True, duel=
        scores = dqn(eps_decay=0.98, ckpt_path='v5_checkpoint.pth')

        # plot the scores
        fig = plt.figure()
        ax = fig.add_subplot(111)
        plt.plot(np.arange(len(scores)), scores, label='Duel DDQN')
```

```
        plt.ylabel('Score')
        plt.xlabel('Episode #')
        plt.legend(loc='upper left')
        plt.show()
```

```
Episode 100        Average Score: 0.23
Episode 200        Average Score: 3.45
Episode 300        Average Score: 6.54
Episode 400        Average Score: 11.34
Episode 453        Average Score: 13.01
Environment solved in 353 episodes!        Average Score: 13.01
```




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