```
!sudo apt-get install swig  # swig 설치
!pip install box2d gymnasium[box2d]
```

숨겨진 출력 표시

```
!git clone https://github.com/Soggeum/20242R0136COSE47402.git
%cd 20242R0136COSE47402/FinalProject/lunar-lander
```

숨겨진 출력 표시

```
import gymnasium as gym
import pygame
import numpy as np
import matplotlib.pyplot as plt
import itertools
```

```
import agent_class as agent
```

Initialize environment and agent

```
# We first create the environment on which we will later train the agent
env = gym.make('LunarLander-v3')

# We need to know the dimensionality of the state space, as well as how many
# actions are possible
N_actions = env.action_space.n
observation, info = env.reset()
N_state = len(observation)

print('dimension of state space =',N_state)
print('number of actions =',N_actions)
```

```
dimension of state space = 8
number of actions = 4
```

```
# We create an instance of the agent class.
# At initialization, we need to provide
# - the dimensionality of the state space, as well as
# - the number of possible actions

parameters = {'N_state':N_state, 'N_actions':N_actions}

my_agent = agent.dqn(parameters=parameters)
# to train via the actor-critic algorithm, use this line:
# my_agent = agent.actor_critic(parameters=parameters)
```

Train agent

```
# We train the agent on the LunarLander-v3 environment.
# Setting verbose=True allows us to follow the progress of the training

training_results = my_agent.train(environment=env,
                                  verbose=True)
```

```
| episode | return          | minimal return      | mean return        |
|         | (this episode)  | (last 20 episodes)  | (last 20 episodes) |
|-------------------------------------------------------------------|
|     100 |       -6.594    |       -371.114      |      -167.828      |
|     200 |      -88.411    |       -251.395      |      -136.390      |
|     300 |      -23.532    |       -213.534      |       -77.778      |
|     400 |     -216.276    |       -216.276      |       -54.640      |
|     500 |      -46.598    |       -168.718      |       -62.316      |
|     600 |       -4.206    |       -319.626      |       -36.105      |
|     700 |      -49.527    |       -128.284      |        73.911      |
|     800 |      -67.011    |       -148.418      |       104.010      |
|     900 |      271.473    |       -233.651      |       114.051      |
|    1000 |      262.370    |       -225.384      |       130.675      |
|    1100 |      302.630    |       -266.388      |       213.734      |
|    1174 |      271.286    |        217.921      |       261.817      |
```

```
# the method my_agent.train() from the previous cell returns a dictionary
# with training stats, namely:
# - duration of each episode during training,
# - return of each episode during training
# - the total number of training epochs at the end of each episode
# - the total number of steps simulated at the end of each episode

training_results.keys()
```

```
dict_keys(['episode_durations', 'epsiode_returns', 'n_training_epochs', 'n_steps_simulated', 'training_completed'])
```

Plot training stats

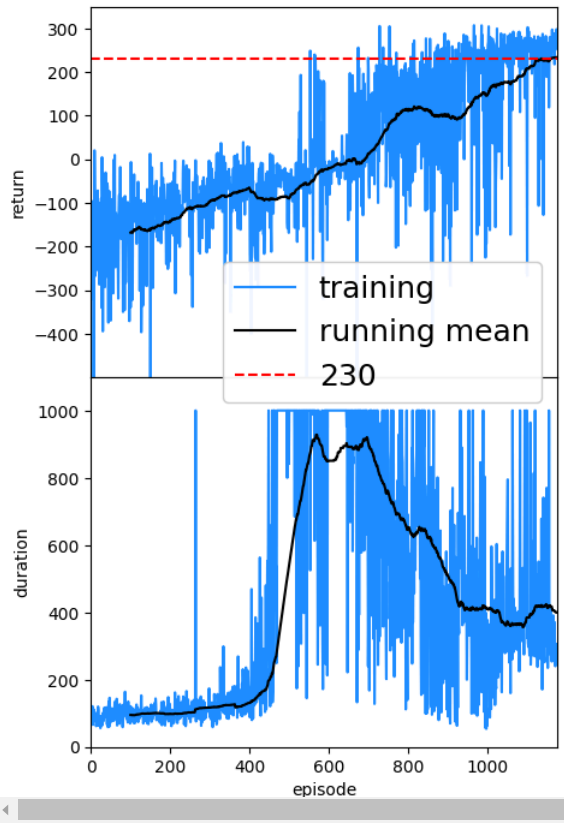
```
# Plot both the return per episode and the duration per episode during
# training, together with their running average over 20 consecutive episodes

N = 100 # number of episodes for running average

def running_mean(x,N=100):
        x_out = np.zeros(len(x)-N,dtype=float)
        for i in range(len(x)-N):
                x_out[i] = np.mean(x[i:i+N+1])
        return x_out

def plot_returns_and_durations(training_results,filename=None):
    fig,axes = plt.subplots(2,1,figsize=(5,8))
    fig.subplots_adjust(hspace=0.0001)
    #
    # return as a function of episode
    ax = axes[0]
    x = training_results['epsiode_returns']
    t = np.arange(len(x)) + 1
    #
    ax.plot(t,x,label='training',color='dodgerblue',)
    # add running mean
    x = running_mean(x=x,N=N)
    t = np.arange(len(x)) + N
    ax.plot(t,x,color='black',label='running mean')
    #
    ax.axhline(230,ls='--',
                label='230',
                        color='red')
    #
    ax.set_ylim(-499,350)
    ax.set_xticks([])
    ax.set_xlim(0,len(t)+100)
    ax.set_xlabel(r'episode')
    ax.set_ylabel(r'return')
    #
    #
    ax = axes[1]
    x = training_results['episode_durations']
    t = np.arange(len(x)) + 1
    #
    ax.plot(t,x,label='training',color='dodgerblue',)
    # add running mean
    x = running_mean(x=x,N=N)
    t = np.arange(len(x)) + N
    ax.plot(t,x,color='black',label='running mean')
    #
    ax.axhline(1200,ls='--', # draw line outside of plot scale,
                label='230', # to get the red dotted line into the legend
                        color='red')
    #
    ax.set_ylim(0,1100)
    ax.set_xlim(0,len(t)+100)
    ax.set_xlabel(r'episode')
    ax.set_ylabel(r'duration')
    ax.legend(loc='upper right',bbox_to_anchor=(1.,1.35),
                                framealpha=0.95,
                        fontsize=18)
    #
    plt.show()
    if filename != None:
        fig.savefig(filename,bbox_inches='tight')
    plt.close(fig)

plot_returns_and_durations(training_results=training_results)
```



Create gameplay video using trained agent

First we create a "live" video that pops up and shows Lunar Lander gameplay performed by the agent

```
# There is the issue that the game window freezes when running gym games
# in jupyter notebooks, see https://github.com/openai/gym/issues/2433
# We here use the fix from that website, which is to use the following
# wrapper class:
class PyGameWrapper(gym.Wrapper):
    def render(self, **kwargs):
        retval = self.env.render( **kwargs)
        for event in pygame.event.get():
            pass
        return retval
```

```
# Create a wrapped environment
env = PyGameWrapper(gym.make('LunarLander-v3',render_mode='human'))

N_episodes = 100

result_string = 'Run {0}: duration = {1}, total return = {2:7.3f}'

for j in range(N_episodes):
    state, info = env.reset()

    total_reward = 0
    for i in itertools.count():
        #env.render()

        action = my_agent.act(state)
        state, reward, terminated, truncated, info = env.step(action)
        done = terminated or truncated
        total_reward += reward

        if done:
            print(result_string.format(j+1,i+1,total_reward))
            break

env.close()
```

```
Run 55: duration = 239, total return = 235.419
Run 56: duration = 494, total return = 246.716
Run 57: duration = 290, total return = 272.497
Run 58: duration = 316, total return = 268.543
Run 59: duration = 305, total return = 282.570
Run 60: duration = 266, total return = 281.784
Run 61: duration = 287, total return = 254.615
Run 62: duration = 253, total return = 258.342
Run 63: duration = 269, total return = 248.640
Run 64: duration = 258, total return = 284.505
Run 65: duration = 418, total return = 237.460
Run 66: duration = 298, total return = 256.810
Run 67: duration = 281, total return = 273.306
Run 68: duration = 241, total return = 267.006
Run 69: duration = 202, total return =   7.432
Run 70: duration = 260, total return = 301.529
Run 71: duration = 228, total return = 287.347
Run 72: duration = 241, total return = 292.207
Run 73: duration = 260, total return = 286.294
Run 74: duration = 284, total return = 297.037
Run 75: duration = 363, total return = 244.174
Run 76: duration = 227, total return = 284.547
Run 77: duration = 235, total return = 285.826
Run 78: duration = 261, total return = 271.271
Run 79: duration = 227, total return = 257.818
Run 80: duration = 274, total return = 282.016
Run 81: duration = 255, total return = 253.038
Run 82: duration = 278, total return = 260.515
Run 83: duration = 287, total return = 293.321
Run 84: duration = 351, total return = 265.774
Run 85: duration = 299, total return = 253.173
Run 86: duration = 241, total return = 243.084
Run 87: duration = 230, total return = 275.609
Run 88: duration = 270, total return = 263.973
Run 89: duration = 241, total return = 281.905
Run 90: duration = 286, total return = 256.052
Run 91: duration = 326, total return = 251.276
Run 92: duration = 380, total return = 204.303
Run 93: duration = 227, total return = 276.534
Run 94: duration = 361, total return = 282.234
Run 95: duration = 239, total return = 246.895
Run 96: duration = 287, total return = 254.795
Run 97: duration = 230, total return = 267.565
Run 98: duration = 241, total return = 258.410
Run 99: duration = 264, total return = 265.239
Run 100: duration = 271, total return = 243.953
```

We also create a video file containing 20 games played by the agent

```
import gymnasium as gym
from gymnasium.wrappers import RecordEpisodeStatistics, RecordVideo

num_eval_episodes = 100

env = gym.make("LunarLander-v3", render_mode="rgb_array")  # replace with your environment
env = RecordVideo(env, video_folder="my_video", name_prefix="eval",
                  episode_trigger=lambda x: True)

env = RecordEpisodeStatistics(env, buffer_length=num_eval_episodes)

for episode_num in range(num_eval_episodes):
    obs, info = env.reset()

    episode_over = False
    while not episode_over:
        action = my_agent.act(state)
        #action = env.action_space.sample()  # replace with actual agent
        obs, reward, terminated, truncated, info = env.step(action)

        episode_over = terminated or truncated
env.close()

print(f'Episode time taken: {env.time_queue}')
print(f'Episode total rewards: {env.return_queue}')
print(f'Episode lengths: {env.length_queue}')
```

```
Episode time taken: deque([0.246945, 0.239493, 0.215686, 0.243252, 0.32205, 0.353517, 0.473401, 0.398357, 0.563515, 0.219236, 0.297372, 0.210349, 0.20769, 0.2263
Episode total rewards: deque([-129.14787369042065, -167.01613602891018, -133.04918564776568, -132.18081214539345, -291.02270670900907, -185.18266855311606, -128.
Episode lengths: deque([75, 73, 69, 71, 97, 76, 86, 71, 90, 69, 102, 67, 71, 75, 85, 85, 52, 84, 88, 89, 83, 56, 76, 65, 73, 66, 86, 88, 77, 62, 67, 67, 57, 60,
```

Video Download

```
import os
import shutil
from google.colab import drive

# Google 드라이브 마운트
drive.mount('/content/drive')

# 원본 경로
source_path = '/content/20242R0136COSE47402/FinalProject/lunar-lander/my_video'
```

```
10
11 # 목적지 경로
12 destination_path = '/content/drive/MyDrive/lunar-lander/mine'
13
14 # 목적지 경로가 없으면 생성
15 if not os.path.exists(destination_path):
16     os.makedirs(destination_path)
17
18 # '/content' 경로에 있는 모든 파일과 폴더를 이동
19 for filename in os.listdir(source_path):
20     file_path = os.path.join(source_path, filename)
21     if os.path.isfile(file_path) or os.path.isdir(file_path):
22         shutil.move(file_path, destination_path)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```