Module 16

Name: Nithin Dsouza

Batch ID: 05062021(9AM-1PM weekend)

Topic: Reinforcement learning

Use OpenAI Gym and reinforcement learning to train a game. You may choose any game here.

Game chosen: SpaceInvaders from OpenAI Gym

Description:

SpaceInvaders-v0
Maximize your score in the Atari 2600 game SpaceInvaders.
In this environment, the observation is an RGB image of the screen,
Which is an array of shape (210, 160, 3) each action is repeatedly
Performed for duration of kk frames, where kk is uniformly sampled
From {2, 3, 4 }

Installing dependencies

```
pip install tensorflow==2.3.1 gym keras-rl2 gym[atari]
```

```
import gym
import random
```

```
env = gym.make('SpaceInvaders-v0')
'''If gives ROM missing error run this below command in anaconda cmd
ROM file is given in folder copy file path and paste it inplace of folder-path

python -m atari_py import_roms "folder-path"
'''
height, width, channels = env.observation_space.shape
actions = env.action_space.n

env.unwrapped.get_action_meanings()
```

1. Test Random Environment with OpenAI Gym

```
episodes = 5
for episode in range(1, episodes+1):
    state = env.reset()
    done = False
    score = 0

    while not done:
        env.render()
        action = random.choice([0,1,2,3,4,5])
        n_state, reward, done, info = env.step(action)
```

```
        score+=reward
    print('Episode:{} Score:{}'.format(episode, score))
env.close()
```

2. Create a Deep Learning Model with Keras

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Convolution2D
from tensorflow.keras.optimizers import Adam

def build_model(height, width, channels, actions):
    model = Sequential()
    model.add(Convolution2D(32, (8,8), strides=(4,4), activation='relu', input_shape=(3,height, width, channels)))
    model.add(Convolution2D(64, (4,4), strides=(2,2), activation='relu'))
    model.add(Convolution2D(64, (3,3), activation='relu'))
    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dense(256, activation='relu'))
    model.add(Dense(actions, activation='linear'))
    return model

'''#run this del model if line 75 gives value error'''
#del model

model = build_model(height, width, channels, actions)

model.summary()
```

3. Build Agent with Keras-RL

```
from rl.agents import DQNAgent
from rl.memory import SequentialMemory
from rl.policy import LinearAnnealedPolicy, EpsGreedyQPolicy

def build_agent(model, actions):
    policy = LinearAnnealedPolicy(EpsGreedyQPolicy(), attr='eps', value_max=1., value_min=.1, value_test=.2, nb_steps=10000)
    memory = SequentialMemory(limit=1000, window_length=3)
    dqn = DQNAgent(model=model, memory=memory, policy=policy,
                   enable_dueling_network=True, dueling_type='avg',
                    nb_actions=actions, nb_steps_warmup=1000
                   )
    return dqn

dqn = build_agent(model, actions) #If getting Vlaue error here must delete model (line 53) and run all codes from there.
dqn.compile(Adam(lr=1e-4))
```

Fitting model

```
dqn.fit(env, nb_steps=1000, visualize=False, verbose=2)
```

Testing/Watching our agent playing for 1K steps later we will load pre train weights for 1M steps and test

```
scores = dqn.test(env, nb_episodes=5, visualize=True)
print(np.mean(scores.history['episode_reward']))

env.close()
```

4. Saving weights Agent from Memory

```
import os
dqn.save_weights(os.getcwd() + os.path.sep +'SavedWeights/1k/dqn_weights_1K.h5f')
```

Now load pre trained agents with 1M Steps and building model again

```
del model,dqn

model = build_model(height, width, channels, actions)
model.summary()

dqn = build_agent(model, actions) #If getting Vlaue error here must delete model (line 53) and run all codes from there.
dqn.compile(Adam(lr=1e-4))

dqn.load_weights(os.getcwd() + os.path.sep +'SavedWeights/1m/dqn_weights.h5f')
```

Testing/Watching our agent playing for 1M steps

```
scores = dqn.test(env, nb_episodes=5, visualize=True)
print(np.mean(scores.history['episode_reward']))

env.close()
```

Now our agent performs better.