Open in Colab

T81-558: Applications of Deep Neural Networks

Module 12: Reinforcement Learning

- Instructor: Jeff Heaton, McKelvey School of Engineering, Washington University in St. Louis
- For more information visit the class website.

Module 12 Video Material

- **Part 12.1: Introduction to the OpenAI Gym** [Video] [Notebook]
- Part 12.2: Introduction to Q-Learning [Video] [Notebook]
- Part 12.3: Keras Q-Learning in the OpenAI Gym [Video] [Notebook]
- Part 12.4: Atari Games with Keras Neural Networks [Video] [Notebook]
- Part 12.5: Application of Reinforcement Learning [Video] [Notebook]

Part 12.1: Introduction to the OpenAI Gym

OpenAI Gym aims to provide an easy-to-setup general-intelligence benchmark with various environments. The goal is to standardize how environments are defined in AI research publications to make published research more easily reproducible. The project claims to provide the user with a simple interface. As of June 2017, developers can only use Gym with Python.

OpenAI gym is pip-installed onto your local machine. There are a few significant limitations to be aware of:

- OpenAI Gym Atari only **directly** supports Linux and Macintosh
- OpenAI Gym Atari can be used with Windows; however, it requires a particular installation procedure
- OpenAI Gym can not directly render animated games in Google CoLab.

Because OpenAI Gym requires a graphics display, an embedded video is the only way to display Gym in Google CoLab. The presentation of OpenAI Gym game animations in Google CoLab is discussed later in this module.

OpenAI Gym Leaderboard

The OpenAI Gym does have a leaderboard, similar to Kaggle; however, the OpenAI Gym's leaderboard is much more informal compared to Kaggle. The user's local machine performs all scoring. As a result, the OpenAI gym's leaderboard is strictly an "honor system." The leaderboard is maintained in the following GitHub repository:

- OpenAI Gym Leaderboard

You must provide a write-up with sufficient instructions to reproduce your result if you submit a score. A video of your results is suggested but not required.

Looking at Gym Environments

The centerpiece of Gym is the environment, which defines the "game" in which your reinforcement algorithm will compete. An environment does not need to be a game; however, it describes the following game-like features:

- **action space**: What actions can we take on the environment at each step/episode to alter the environment.
- **observation space**: What is the current state of the portion of the environment that we can observe. Usually, we can see the entire environment.

Before we begin to look at Gym, it is essential to understand some of the terminology used by this library.

- **Agent** - The machine learning program or model that controls the actions. Step - One round of issuing actions that affect the observation space.
- **Episode** - A collection of steps that terminates when the agent fails to meet the environment's objective or the episode reaches the maximum number of allowed steps.
- **Render** - Gym can render one frame for display after each episode.
- **Reward** - A positive reinforcement that can occur at the end of each episode, after the agent acts.
- **Non-deterministic** - For some environments, randomness is a factor in deciding what effects actions have on reward and changes to the observation space.

It is important to note that many gym environments specify that they are not non-deterministic even though they use random numbers to process actions. Based on the gym GitHub issue tracker, a non-deterministic property means a deterministic environment behaves randomly. Even when you give the environment a consistent seed value, this behavior is confirmed. The program can use the seed method of an environment to seed the random number generator for the environment.

The Gym library allows us to query some of these attributes from environments. I created the following function to query gym environments.

In [1]:
```
import gym


def query_environment(name):
    env = gym.make(name)
    spec = gym.spec(name)
    print(f"Action Space: {env.action_space}")
    print(f"Observation Space: {env.observation_space}")
    print(f"Max Episode Steps: {spec.max_episode_steps}")
    print(f"Nondeterministic: {spec.nondeterministic}")
    print(f"Reward Range: {env.reward_range}")
    print(f"Reward Threshold: {spec.reward_threshold}")
```

We will look at the **MountainCar-v0** environment, which challenges an underpowered car to escape the valley between two mountains. The following code describes the Mountian Car environment.

In [2]:
```
query_environment("MountainCar-v0")
```

```
Action Space: Discrete(3)
Observation Space: Box(-1.2000000476837158, 0.6000000238418579, (2,), float3
2)
Max Episode Steps: 200
Nondeterministic: False
Reward Range: (-inf, inf)
Reward Threshold: -110.0
```

This environment allows three distinct actions: accelerate forward, decelerate, or backward. The observation space contains two continuous (floating point) values, as evident by the box object. The observation space is simply the position and velocity of the car. The car has 200 steps to escape for each episode. You would have to look at the code, but the mountain car receives no incremental reward. The only reward for the vehicle occurs when it escapes the valley.

In [3]:
```
query_environment("CartPole-v1")
```

```
Action Space: Discrete(2)
Observation Space: Box(-3.4028234663852886e+38, 3.4028234663852886e+38,
(4,), float32)
Max Episode Steps: 500
Nondeterministic: False
Reward Range: (-inf, inf)
Reward Threshold: 475.0
```

The **CartPole-v1** environment challenges the agent to balance a pole while the agent. The environment has an observation space of 4 continuous numbers:

- Cart Position
- Cart Velocity

- Pole Angle
- Pole Velocity At Tip

To achieve this goal, the agent can take the following actions:

- Push cart to the left
- Push cart to the right

There is also a continuous variant of the mountain car. This version does not simply have the motor on or off. The action space is a single floating-point number for the continuous cart that specifies how much forward or backward force the cart currently utilizes.

In [4]:
```
query_environment("MountainCarContinuous-v0")
```

```
Action Space: Box(-1.0, 1.0, (1,), float32)
Observation Space: Box(-1.2000000476837158, 0.6000000238418579, (2,), float3
2)
Max Episode Steps: 999
Nondeterministic: False
Reward Range: (-inf, inf)
Reward Threshold: 90.0
```

Note: If you see a warning above, you can safely ignore it; it is a relatively minor bug in OpenAI Gym.

Atari games, like breakout, can use an observation space that is either equal to the size of the Atari screen (210x160) or even use the RAM of the Atari (128 bytes) to determine the state of the game. Yes, that's bytes, not kilobytes!

In [5]:
```
# HIDE OUTPUT
!wget http://www.atarimania.com/roms/Roms.rar
!unrar x -o+ /content/Roms.rar >/dev/nul
!python -m atari_py.import_roms /content/ROMS >/dev/nul
```

```
--2022-04-02 16:20:06--  http://www.atarimania.com/roms/Roms.rar
Resolving www.atarimania.com (www.atarimania.com)... 195.154.81.199
Connecting to www.atarimania.com (www.atarimania.com)|195.154.81.199|:80...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 19583716 (19M) [application/x-rar-compressed]
Saving to: ‘Roms.rar’

Roms.rar            100%[===================>]  18.68M   474KB/s    in 42s

2022-04-02 16:20:49 (456 KB/s) - ‘Roms.rar’ saved [19583716/19583716]
```

In [6]:
```
query_environment("Breakout-v0")
```

```
Action Space: Discrete(4)
Observation Space: Box(0, 255, (210, 160, 3), uint8)
Max Episode Steps: 10000
Nondeterministic: False
Reward Range: (-inf, inf)
Reward Threshold: None
```

In [7]:
```
query_environment("Breakout-ram-v0")
```

```
Action Space: Discrete(4)
Observation Space: Box(0, 255, (128,), uint8)
Max Episode Steps: 10000
Nondeterministic: False
Reward Range: (-inf, inf)
Reward Threshold: None
```

Render OpenAI Gym Environments from CoLab

It is possible to visualize the game your agent is playing, even on CoLab. This section provides information on generating a video in CoLab that shows you an episode of the game your agent is playing. I based this video process on suggestions found here.

Begin by installing **pyvirtualdisplay** and **python-opengl**.

In [8]:
```
# HIDE OUTPUT
!pip install gym pyvirtualdisplay > /dev/null 2>&1
!apt-get install -y xvfb python-opengl ffmpeg > /dev/null 2>&1
```

Next, we install the needed requirements to display an Atari game.

In [9]:
```
# HIDE OUTPUT
!apt-get update > /dev/null 2>&1
!apt-get install cmake > /dev/null 2>&1
!pip install --upgrade setuptools 2>&1
!pip install ez_setup > /dev/null 2>&1
!pip install gym[atari] > /dev/null 2>&1
```

```
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-p
ackages (57.4.0)
Collecting setuptools
  Downloading setuptools-61.3.1-py3-none-any.whl (1.1 MB)
     |████████████████████████████████| 1.1 MB 7.8 MB/s
Installing collected packages: setuptools
  Attempting uninstall: setuptools
    Found existing installation: setuptools 57.4.0
    Uninstalling setuptools-57.4.0:
      Successfully uninstalled setuptools-57.4.0
ERROR: pip's dependency resolver does not currently take into account all th
e packages that are installed. This behaviour is the source of the following
dependency conflicts.
tensorflow 2.8.0 requires tf-estimator-nightly==2.8.0.dev2021122109, which i
s not installed.
datascience 0.10.6 requires folium==0.2.1, but you have folium 0.8.3 which i
s incompatible.
Successfully installed setuptools-61.3.1
```

Next, we define the functions used to show the video by adding it to the CoLab notebook.

In [10]:
```
import gym
from gym.wrappers import Monitor
import glob
import io
import base64
from IPython.display import HTML
from pyvirtualdisplay import Display
from IPython import display as ipythondisplay

display = Display(visible=0, size=(1400, 900))
display.start()

"""
Utility functions to enable video recording of gym environment
and displaying it.
To enable video, just do "env = wrap_env(env)""
"""


def show_video():
    mp4list = glob.glob('video/*.mp4')
    if len(mp4list) > 0:
        mp4 = mp4list[0]
        video = io.open(mp4, 'r+b').read()
        encoded = base64.b64encode(video)
        ipythondisplay.display(HTML(data='''<video alt="test" autoplay
                loop controls style="height: 400px;">
                <source src="data:video/mp4;base64,{0}" type="video/mp4" />
             </video>'''.format(encoded.decode('ascii'))))
    else:
        print("Could not find video")
```

```
def wrap_env(env):
    env = Monitor(env, './video', force=True)
    return env
```

Now we are ready to play the game. We use a simple random agent.

In [11]:
```
# HIDE OUTPUT
#env = wrap_env(gym.make("MountainCar-v0"))
env = wrap_env(gym.make("Atlantis-v0"))

observation = env.reset()

while True:

    env.render()

    # your agent goes here
    action = env.action_space.sample()

    observation, reward, done, info = env.step(action)

    if done:
        break

env.close()
show_video()
```