LunarLander with Policy Gradient

Made by

- Adrian Biller A01018940
- Edgar Garcia A01021730
- Jose Manuel Beauregard A01021716

In [0]:

```
#remove " > /dev/null 2>&1" to see what is going on under the hood
!pip install gym pyvirtualdisplay > /dev/null 2>&1
!apt-get install -y xvfb python-opengl ffmpeg > /dev/null 2>&1
!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
!pip install gym[box2d] > /dev/null 2>&1
```

```
Requirement already up-to-date: setuptools in /usr/local/lib/python3.6/di
st-packages (41.0.1)
```

In [0]:

```
import gym
from gym import logger as gymlogger
from gym.wrappers import Monitor

import matplotlib
import matplotlib.pyplot as plt
import itertools
import cv2
import numpy as np
import random, math
from keras import models, layers, optimizers

from collections import deque

import glob, io, base64

from tensorflow import convert_to_tensor
from IPython.display import HTML
from IPython import display as ipythondisplay
from pyvirtualdisplay import Display
from sklearn.preprocessing import normalize
gymlogger.set_level(40) #error only
%matplotlib inline
```

In [0]:
```
"""
Utility functions to enable video recording of gym environment and displayi
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
```

In [0]:
```
display = Display(visible=0, size=(1400, 900))
display.start()
```

Out[4]:
```
<Display cmd_param=['Xvfb', '-br', '-nolisten', 'tcp', '-screen', '0', '1
400x900x24', ':1001'] cmd=['Xvfb', '-br', '-nolisten', 'tcp', '-screen',
'0', '1400x900x24', ':1001'] oserror=None return_code=None stdout="None"
stderr="None" timeout_happened=False>
```

In this project we will solve Lunar Lander v2 environment where the user must land a spaceship in a desiered area. For this the user can do 4 different actions and we will use 8 inputs that will help discretize the env.

Available Actions

- Do nothing
- Left engine
- Main engine
- Right engine

Inputs

1. X Position
2. Y Position
3. X Velocity
4. Y Velocity
5. Angle
6. Angular Velocity
7. Left leg in contact with ground

8. Right leg in contact with ground

Rewards

- Reward for moving from the top of the screen to landing pad and zero speed is about 100..140 points.
- If lander moves away from landing pad it loses reward back.
- Episode finishes if the lander crashes or comes to rest, receiving additional -100 or +100 points.
- Each leg ground contact is +10.
- Firing mainengine is -0.3 points each frame. (DOWN)
- Firing side engine is -0.03 points each frame. (LEFT-RIGHT)
- Solved is 200 points.

In [0]:
```
# Loads the cartpole environment
env = wrap_env(gym.make('LunarLander-v2'))

state_size   = env.observation_space.shape[0]
action_size = env.action_space.n

n_episodes   = 10100

print(state_size, action_size)
```

```
8 4
```

In [0]:
```
env = wrap_env(gym.make('LunarLander-v2'))
observation = env.reset()

while True:

    env.render()

    #your agent goes here
    action = np.random.choice([0,1,2,3])
    #action = env.action_space.sample()

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

    if done:
      break;

env.close()
show_video()
```

0:00

In [0]:
```
def discount_rewards(reward, gamma):
  r = np.array(reward)
  discounted_r = np.zeros_like(r)
  running_add = 0

  for t in reversed(range(0, r.size)):
    if r[t] != 0:
      running_add = 0
    # the point here is to use Horner's method to compute those rewards eff
    running_add = running_add * gamma + r[t]
    discounted_r[t] = running_add
  # normalizing the result
  discounted_r -= np.mean(discounted_r)
  # idem
  discounted_r /= np.std(discounted_r)

  return discounted_r
```

In [0]:
```
class DQNAgent:
    def __init__(self, state_size, action_size, gamma = 0.99, alpha = 0.001
        self.state_size    = state_size
        self.action_size   = action_size
        self.gamma         = gamma
        self.alpha         = alpha
        self.model         = self._build_model()

    def _build_model(self):
        model = models.Sequential()

        model.add(layers.Dense(8,
                               activation='relu',
                               input_dim=self.state_size,
                               kernel_initializer='glorot_uniform'))
        model.add(layers.Dense(10,
                               activation='sigmoid',
                               kernel_initializer='RandomNormal'))
        model.add(layers.Dense(10, activation='sigmoid'))
        model.add(layers.Dense(4,   activation='softmax'))

        model.compile(loss= 'binary_crossentropy',
                                optimizer= optimizers.Adam(lr=self.alpha),
                                metrics= ['accuracy'])
        return model

    def train(self, states, labels, rewards):
        # sample_weight: Optional Numpy array of weights for the training s
        # used for weighting the loss function (during training only).
        self.model.fit(x = states,
                             y = labels,
                             verbose = 0,
                             steps_per_epoch = states.shape[1],
                             sample_weight = discount_rewards(rewards, self

    def load(self, name):
        self.model.load_weights(name)

    def save(self, name):
        self.model.save_weights(name)
```

In [0]:
```
def oneHotEncoding(action):
    '''
     * Do nothing
     * Left engine
     * Main engine
     * Right engine
    '''
    one_hot = np.zeros(4)
    one_hot[action] = 1
    return one_hot
```

In [0]:
```
"""
  inputs Indexes
    0: X Position
    1: Y Position
    2: X Velocity
    3: Y Velocity
    4: Angle
    5: Angular Velocity
    6: Left leg in contact with ground
    7: Right leg in contact with ground
"""
# normalize is a function from sklearn
preprocessInput = lambda inputs : normalize(inputs[:, np.newaxis], axis= 0)
```

In [0]:
```
# convert_to_tensor is a function from tensorflow
to_tensor = lambda tensor : convert_to_tensor(np.array(tensor), dtype= tf.f
```

In [0]:
```
env = wrap_env(gym.make('LunarLander-v2'))
running_reward = None

agent = DQNAgent(state_size, action_size, 0.91, 0.002)
agent.model.summary()

'''
Action mapping
  [
    going left: 0
    going right: 1
    rotate left: 2
    rotate right: 3
  ]
'''
actionsMap = {
    0: [1],
    1: [3],
    2: [1,2],
    3: [2,3]
}


try:
    for e in range(n_episodes):

        states_train, labels_train, rewards = [], [], []
        total_reward = 0

        next_state = env.reset()
        prev_state = None

        done = False

        while not done:

            #env.render()
            current_state = np.copy(preprocessInput(next_state))

            delta_state = current_state - prev_state if prev_state is not N

            prev_state =  np.copy(current_state)

            current_state = np.reshape(current_state, (1,8))

            prediction = agent.model.predict(current_state)

            action = random.choice(actionsMap[np.argmax(prediction)])

            label = oneHotEncoding(action)

            states_train.append(delta_state)
            labels_train.append(label)

            next_state, reward, done, _ = env.step(action)
            rewards.append(reward)
```

```
            total_reward += reward


        if e % 100 == 0:
          running_reward = total_reward if running_reward is None else runn
          print('Episode: {} Reward {}'.format(e, total_reward))
          agent.save('model_weights_{}_{}_.hdf5'.format(e, running_reward))


        agent.train(to_tensor(states_train),
                    to_tensor(labels_train),
                    rewards)


finally:
    env.close()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorfl
ow/python/framework/op_def_library.py:263: colocate_with (from tensorfl
ow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 8)                 72
_________________________________________________________________
dense_2 (Dense)              (None, 10)                90
_________________________________________________________________
dense_3 (Dense)              (None, 10)                110
_________________________________________________________________
dense_4 (Dense)              (None, 4)                 44
=================================================================
Total params: 316
Trainable params: 316
```

In [0]:

```
env = wrap_env(gym.make('LunarLander-v2'))
agent = DQNAgent(state_size, action_size)
# agent.load('model_weights_9800_0.28.hdf5')
agent.load('model_weights_6000_-613.3778618601141_.hdf5')
actionsMap = {
    0: [1],
    1: [3],
    2: [1,2],
    3: [2,3]
}

try:
      states_train, labels_train, rewards = [], [], []
      total_reward = 0

      next_state = env.reset()
      prev_state = None

      done = False

      while not done:

          env.render()
          current_state = np.copy(preprocessInput(next_state))
          current_state = np.reshape(current_state, (1,8))
          # Takes a random action from the action space of the environment
          prediction = agent.model.predict(current_state)
          action = random.choice(actionsMap[np.argmax(prediction)])

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

          total_reward += reward

          next_state, reward, done, _ = env.step(action)
          rewards.append(reward)
          total_reward += reward

finally:
    env.close()
    show_video()
```

0:00

Final Results

Here we will present the evolution of our model.

We trained our model for **11,000** episodes and it took around 13-14 hours in total to train. Given how the rewards are treated in this game, we can give ourselfs an idea of what its learning in each 1000's iteration.

Iteration 2000

- Reward: -495.759

In [0]:
```
HTML("""
  <iframe src="https://player.vimeo.com/video/335758641" width="640" height
""")
```

Out[5]:

Iteration 4000

- Reward: -84.598

In [0]:
```
HTML("""
  <iframe src="https://player.vimeo.com/video/335715611" width="640" height
""")
```

Out[34]: "

Iteration 5000

- Reward: -136.116

In [0]:

```
HTML("""
  <iframe src="https://player.vimeo.com/video/335715616" width="640" height
""")
```

Out[33]:

Iteration 6000

- Reward: -31.450

In [0]:
```
HTML("""
  <iframe src="https://player.vimeo.com/video/335715621" width="640" height
""")
```

Out[35]:

Iteration 9000

- Reward: -52.414

In [0]:
```
HTML("""
  <iframe src="https://player.vimeo.com/video/335715627" width="640" height
""")
```

Out[36]:

Best Iteration

- Episode 9800
- Reward: 0.2836
- Best reward
- This video was made loading the weights and using model.predict

In [0]:
```
HTML("""
  <iframe src="https://player.vimeo.com/video/335722271" width="640" height
""")
```

Out[37]: