# OpenAI Gym

December 21, 2017

## 1 Testing the Gym

In [ ]:
```
print("Attempting to Import Gym")
print("Remember you can use pip install gym at your command line.")
import gym
print(gym.__version__)
print("It worked!")
```

## 2 Intro to OpenAI

In [ ]:
```
# Gotta import gym!
import gym

# Make the environment, replace this string with any
# from the docs. (Some environments have dependencies)
env = gym.make('CartPole-v0')

# Reset the environment to default beginning
env.reset()

# Using _ as temp placeholder variable
for _ in range(1000):
    # Render the env
    #env.render()

    # Still a lot more explanation to come for this line!
    env.step(env.action_space.sample()) # take a random action
```

## 3 Gym Environment Basics

In [ ]:
```
# Gotta import gym!
import gym

# Make the environment, replace this string with any
# from the docs. (Some environments have dependencies)
env = gym.make('CartPole-v0')
```

```
        # Reset the environment to default beginning
        # Default observation variable
        print("Initial Observation")
        observation = env.reset()
        print(observation)

        print('\n')
        for _ in range(1):


            # Random Action
            action = env.action_space.sample()

            # Get the 4 observation values discussed
            observation, reward, done, info = env.step(action)

            print("Performed One Random Action")
            print('\n')
            print('observation')
            print(observation)
            print('\n')

            print('reward')
            print(reward)
            print('\n')

            print('done')
            print(done)
            print('\n')

            print('info')
            print(info)
            print('\n')
```

## 4 Gym Actions

```
In [ ]: import gym
        env = gym.make('CartPole-v0')
        # print(env.action_space.)
        # #> Discrete(2)
        # print(env.observation_space)
        # #> Box(4,)
        observation = env.reset()

        for t in range(1000):
```

```
    #env.render()

    cart_pos , cart_vel , pole_ang , ang_vel = observation

    # Move Cart Right if Pole is Falling to the Right

    # Angle is measured off straight vertical line
    if pole_ang > 0:
        # Move Right
        action = 1
    else:
        # Move Left
        action = 0

    # Perform Action
    observation , reward, done, info = env.step(action)
```

## 5 Basic Neural Network

```
In [ ]: import tensorflow as tf
        import gym
        import numpy as np
        ################################################
        ######## PART ONE: NETWORK VARIABLES #########
        ################################################

        # Observation Space has 4 inputs
        num_inputs = 4

        num_hidden = 4

        # Outputs the probability it should go left
        num_outputs = 1

        initializer = tf.contrib.layers.variance_scaling_initializer()


        ################################################
        ######## PART TWO: NETWORK LAYERS #########
        ################################################

        X = tf.placeholder(tf.float32, shape=[None,num_inputs])
        hidden_layer_one = tf.layers.dense(
            X,num_hidden,
            activation=tf.nn.relu,kernel_initializer=initializer)
        hidden_layer_two = tf.layers.dense(
            hidden_layer_one,
```

```
    num_hidden,
    activation=tf.nn.relu,kernel_initializer=initializer)

# Probability to go left
output_layer = tf.layers.dense(
    hidden_layer_one,
    num_outputs,
    activation=tf.nn.sigmoid,kernel_initializer=initializer)

# [ Prob to go left , Prob to go right]
probabilties = tf.concat(
    axis=1, values=[output_layer, 1 - output_layer])

# Sample 1 randomly based on probabilities
action = tf.multinomial(probabilties, num_samples=1)


init = tf.global_variables_initializer()



##################################################
######## PART THREE: SESSION #########
##################################################

saver = tf.train.Saver()

epi = 50
step_limit = 500
avg_steps = []
env = gym.make("CartPole-v1")
with tf.Session() as sess:
    init.run()
    for i_episode in range(epi):
        obs = env.reset()

        for step in range(step_limit):
            # env.render()
            action_val = action.eval(
                feed_dict={X: obs.reshape(1, num_inputs)})
            obs, reward, done, info = env.step(action_val[0][0])
            if done:
                avg_steps.append(step)
                print('Done after {} steps'.format(step))
                break
print("After {} episodes the average cart steps before done was {}".format(
    epi,np.mean(avg_steps)))
env.close()
```

## 6 Policy Gradient Neural Network

In [ ]:
```
import tensorflow as tf
import gym
import numpy as np
#########################
### VARIABLES ###########
#########################

num_inputs = 4
num_hidden = 4
num_outputs = 1

learning_rate = 0.01

initializer = tf.contrib.layers.variance_scaling_initializer()


#################################
### CREATING THE NETWORK #######
#################################

X = tf.placeholder(tf.float32, shape=[None, num_inputs])

hidden_layer = tf.layers.dense(
    X,
    num_hidden,
    activation=tf.nn.elu,
    kernel_initializer=initializer)
logits = tf.layers.dense(hidden_layer, num_outputs)
outputs = tf.nn.sigmoid(logits)  # probability of action 0 (left)

probabilties = tf.concat(axis=1, values=[outputs, 1 - outputs])
action = tf.multinomial( probabilties, num_samples=1)

# Convert from Tensor to number for network training
y = 1. - tf.to_float(action)

#######################################
### LOSS FUNCTION AND OPTIMIZATION ####
#######################################
cross_entropy = tf.nn.sigmoid_cross_entropy_with_logits(
    labels=y, logits=logits)
optimizer = tf.train.AdamOptimizer(learning_rate)

# https://stackoverflow.com/questions/41954198/
# optimizer-compute-gradients-how-the-gradients-are-calculated-programatically
# https://www.tensorflow.org/api_docs/python/tf/train/AdamOptimizer
```

```
################################
#### GRADIENTS ################
################################
gradients_and_variables = optimizer.compute_gradients(cross_entropy)



gradients = []
gradient_placeholders = []
grads_and_vars_feed = []

for gradient, variable in gradients_and_variables:
    gradients.append(gradient)
    gradient_placeholder = tf.placeholder(
        tf.float32, shape=gradient.get_shape())
    gradient_placeholders.append(gradient_placeholder)
    grads_and_vars_feed.append((gradient_placeholder, variable))


training_op = optimizer.apply_gradients(grads_and_vars_feed)

init = tf.global_variables_initializer()
saver = tf.train.Saver()

#####################################
#### REWARD FUNCTIONs ################
#####################################
# CHECK OUT: https://medium.com/@awjuliani/super-simple-reinforcement-learning-tutorial-

def helper_discount_rewards(rewards, discount_rate):
    '''
    Takes in rewards and applies discount rate
    '''
    discounted_rewards = np.zeros(len(rewards))
    cumulative_rewards = 0
    for step in reversed(range(len(rewards))):
        cumulative_rewards = rewards[step] + cumulative_rewards * discount_rate
        discounted_rewards[step] = cumulative_rewards
    return discounted_rewards

def discount_and_normalize_rewards(all_rewards, discount_rate):
    '''
    Takes in all rewards, applies helper_discount function and then normalizes
    using mean and std.
    '''
    all_discounted_rewards = []
```

```
    for rewards in all_rewards:
        all_discounted_rewards.append(helper_discount_rewards(rewards,discount_rate))

    flat_rewards = np.concatenate(all_discounted_rewards)
    reward_mean = flat_rewards.mean()
    reward_std = flat_rewards.std()
    return [(discounted_rewards - reward_mean)/reward_std for
            discounted_rewards in all_discounted_rewards]

########################################
#### TRAINING SESSION #################
########################################

env = gym.make("CartPole-v0")

num_game_rounds = 10
max_game_steps = 1000
num_iterations = 250
discount_rate = 0.95

with tf.Session() as sess:
    sess.run(init)


    for iteration in range(num_iterations):
        print("Currently on Iteration: {} \n".format(iteration) )

        all_rewards = []
        all_gradients = []

        # Play n amount of game rounds
        for game in range(num_game_rounds):

            current_rewards = []
            current_gradients = []

            observations = env.reset()

            # Only allow n amount of steps in game
            for step in range(max_game_steps):

                # Get Actions and Gradients
                action_val, gradients_val = sess.run(
                    [action,gradients],
                    feed_dict={X: observations.reshape(1, num_inputs)})

                # Perform Action
                observations, reward, done, info = env.step(action_val[0][0])
```

```
                # Get Current Rewards and Gradients
                current_rewards.append(reward)
                current_gradients.append(gradients_val)

                if done:
                    # Game Ended
                    break

            # Append to list of all rewards
            all_rewards.append(current_rewards)
            all_gradients.append(current_gradients)

        all_rewards = discount_and_normalize_rewards(all_rewards,discount_rate)
        feed_dict = {}


        for var_index, gradient_placeholder in enumerate(gradient_placeholders):
            mean_gradients = np.mean([reward * all_gradients[game_index][step][var_index
                                      for game_index, rewards in enumerate(all_rewards)
                                          for step, reward in enumerate(rewards)], axis=
            feed_dict[gradient_placeholder] = mean_gradients

        sess.run(training_op, feed_dict=feed_dict)

    print('SAVING GRAPH AND SESSION')
    meta_graph_def = tf.train.export_meta_graph(filename='/models/my-650-step-model.meta
    saver.save(sess, '/models/my-650-step-model')




##############################################
### RUN TRAINED MODEL ON ENVIRONMENT #######
##############################################

env = gym.make('CartPole-v0')

observations = env.reset()
with tf.Session() as sess:
    # https://www.tensorflow.org/api_guides/python/meta_graph
    new_saver = tf.train.import_meta_graph('/models/my-650-step-model.meta')
    new_saver.restore(sess,'/models/my-650-step-model')

    for x in range(500):
        env.render()
        action_val, gradients_val = sess.run(
            [action, gradients],
```

```
                feed_dict={X: observations.reshape(1, num_inputs)})
            observations, reward, done, info = env.step(action_val[0][0])
```

# 7 Running Network

```
In [ ]: import tensorflow as tf
        import gym
        import numpy as np

        # Same as 05 file, just no training, just loads the pre-trained model.


        ##########################
        ### VARIABLES ############
        ##########################

        num_inputs = 4
        num_hidden = 4
        num_outputs = 1

        learning_rate = 0.01

        initializer = tf.contrib.layers.variance_scaling_initializer()


        ################################
        ### CREATING THE NETWORK #######
        ################################

        X = tf.placeholder(tf.float32, shape=[None, num_inputs])

        hidden_layer = tf.layers.dense(
            X,
            num_hidden,
            activation=tf.nn.elu,
            kernel_initializer=initializer)
        logits = tf.layers.dense(hidden_layer, num_outputs)
        outputs = tf.nn.sigmoid(logits)  # probability of action 0 (left)


        probabilties = tf.concat(axis=1, values=[outputs, 1 - outputs])
        action = tf.multinomial( probabilties, num_samples=1)

        # Convert from Tensor to number for network training
        y = 1. - tf.to_float(action)

        ########################################
```

```
### LOSS FUNCTION AND OPTIMIZATION ####
#######################################
cross_entropy = tf.nn.sigmoid_cross_entropy_with_logits(labels=y, logits=logits)
optimizer = tf.train.AdamOptimizer(learning_rate)

# https://stackoverflow.com/questions/41954198/
#optimizer-compute-gradients-how-the-gradients-are-calculated-programatically
# https://www.tensorflow.org/api_docs/python/tf/train/AdamOptimizer


################################
#### GRADIENTS ###############
################################
gradients_and_variables = optimizer.compute_gradients(cross_entropy)



gradients = []
gradient_placeholders = []
grads_and_vars_feed = []

for gradient, variable in gradients_and_variables:
    gradients.append(gradient)
    gradient_placeholder = tf.placeholder(tf.float32, shape=gradient.get_shape())
    gradient_placeholders.append(gradient_placeholder)
    grads_and_vars_feed.append((gradient_placeholder, variable))


training_op = optimizer.apply_gradients(grads_and_vars_feed)

init = tf.global_variables_initializer()
saver = tf.train.Saver()

#####################################
#### REWARD FUNCTIONs ###############
#####################################
# CHECK OUT: https://medium.com/@awjuliani/
#super-simple-reinforcement-learning-tutorial-part-2-ded33892c724

def helper_discount_rewards(rewards, discount_rate):
    '''
    Takes in rewards and applies discount rate
    '''
    discounted_rewards = np.zeros(len(rewards))
    cumulative_rewards = 0
    for step in reversed(range(len(rewards))):
        cumulative_rewards = rewards[step] + cumulative_rewards * discount_rate
        discounted_rewards[step] = cumulative_rewards
```

```
    return discounted_rewards

def discount_and_normalize_rewards(all_rewards, discount_rate):
    '''
    Takes in all rewards, applies helper_discount function and then normalizes
    using mean and std.
    '''
    all_discounted_rewards = []
    for rewards in all_rewards:
        all_discounted_rewards.append(helper_discount_rewards(rewards,discount_rate))

    flat_rewards = np.concatenate(all_discounted_rewards)
    reward_mean = flat_rewards.mean()
    reward_std = flat_rewards.std()
    return [(discounted_rewards - reward_mean)/reward_std for
            discounted_rewards in all_discounted_rewards]

############################################
### RUN TRAINED MODEL ON ENVIRONMENT #######
############################################

env = gym.make('CartPole-v0')

observations = env.reset()
with tf.Session() as sess:
    # https://www.tensorflow.org/api_guides/python/meta_graph
    new_saver = tf.train.import_meta_graph('/models/my-650-step-model.meta')
    new_saver.restore(sess,'/models/my-650-step-model')

    for x in range(500):
        env.render()
        action_val, gradients_val = sess.run(
            [action, gradients],
            feed_dict={X: observations.reshape(1, num_inputs)})
        observations, reward, done, info = env.step(action_val[0][0])
```