人工智能第一次实验

姓名：

郑子浩 2017202117

内容：

一、概念

2016 年 5 月 4 日，OpenAI 发布了人工智能研究工具集 OpenAI Gym。OpenAI Gym 是一款用于研发和比较学习算法的工具包。它与很多数值计算库兼容，比如 tensorflow 和 theano。现在支持的语言主要是 python。

openai gym 是一个增强学习（reinforcement learning,RL）算法的测试床（testbed）。增强学习和有监督学习的评测不一样。有监督学习的评测工具是数据。只要提供一批有标注的数据 18:34:13 就能进行有监督学习的评测。增强学习的评测工具是环境。需要提供一个环境给 Agent 运行，才能评测 Agent 的策略的优劣。OpenAI Gym 是提供各种环境的开源工具包。

OpenAI Gym 由两部分组成：

gym 开源库：测试问题的集合。当你测试增强学习的时候，测试问题就是环境，比如机器人玩游戏，环境的集合就是游戏的画面。这些环境有一个公共的接口，允许用户设计通用的算法。

OpenAI Gym 服务。提供一个站点（比如对于游戏 cartpole-v0：https://gym.openai.com/envs/CartPole-v0）和 api，允许用户对他们的测试结果进行比较。

二、安装

使用 OpenAI Gym 首先需要将其载入：

Windows（有两种方法）：

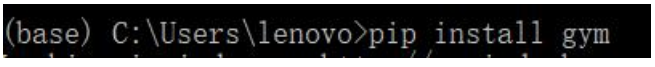
（1）使用 pip：

```
pip install gym
```

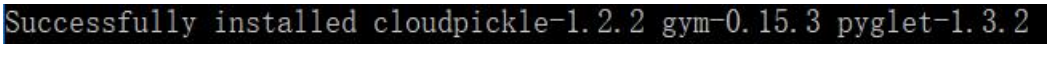
（2）使用 git：

```
git clone https://github.com/openai/gym
cd gym
pip install -e . # minimal install
pip install -e .[all] # full install (this requires cmake and a recent pip version)
```

采用方法 1：

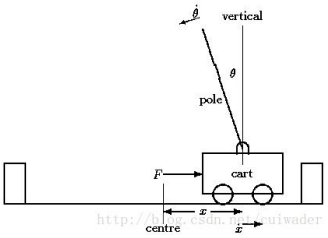


Pip 成功：



三、游戏介绍

Cart Pole 在 OpenAI 的 gym 模拟器里面，是相对比较简单的一个游戏。游戏里面有一个小车，上有竖着一根杆子。小车需要左右移动来保持杆子竖直。如果杆子倾斜的角度大于 15°，那么游戏结束。小车也不能移动出一个范围（中间到两边各 2.4 个单位长度）。如下图所示：



在 gym 的 Cart Pole 环境（env）里面，左移或者右移小车的 action 之后，env 都会返回一个+1 的 reward。到达 200 个 reward 之后，游戏也会结束。

四、使用

熟悉环境：

运行 CartPole-v0 环境 1000 个时间步(timestep)。

构造一个初始环境：

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

通过重置来启动环境：

env.reset()

渲染出当前的智能体以及环境的状态:

env.render()

选择行动：

env.action_space.sample()

行动：

env.step()

最后，关闭环境：

env.close()

具体代码：

```
import gym
env = gym.make('CartPole-v0')
observation = env.reset()
for t in range(1000):
    env.render()
    print(observation)
    observation, reward, done, info=env.step(env.action_space.sample()) # take a random action
    if done:
            print("Episode finished after {} timesteps".format(t+1))
            break
env.close()
```

结果示例：

可以看到随机控制算法发散，游戏很快结束，该游戏环节持续了 15 timesteps。

```
[ 0.00922061  0.03029822  0.00099047 -0.03414432]
[ 9.82657860e-03  2.25405952e-01  3.07584193e-04 -3.26514583e-01]
[ 0.0143347   0.03027962 -0.00622271 -0.03373467]
[ 0.01494029 -0.16475254 -0.0068974   0.25697845]
[ 0.01164524 -0.35977534 -0.00175783  0.54747787]
[ 0.00444973 -0.16462874  0.00919173  0.25424162]
[ 0.00115716 -0.35988072  0.01427656  0.54980954]
[-0.00604046 -0.55520027  0.02527275  0.84695617]
[-0.01714446 -0.36043203  0.04221187  0.5623265 ]
[-0.0243531  -0.55612012  0.0534584   0.86800377]
[-0.03547551 -0.7519271   0.07081848  1.17700395]
[-0.05051405 -0.94789387  0.09435856  1.49102053]
[-0.06947192 -1.14402929  0.12417897  1.81161518]
[-0.09235251 -1.340297    0.16041127  2.14016461]
[-0.11915845 -1.53660003  0.20321456  2.47779778]
Episode finished after 15 timesteps
```

尝试优化：

环境的 step 函数返回四个值：

Observation(object):返回一个特定环境的对象，描述对环境的观察。比如，来自相机的像素数据，机器人的关节角度和关节速度，或棋盘游戏中的棋盘状态。

Reward(float)：返回之前动作收获的总的奖励值。不同的环境计算方式不一样，但总体的目标是增加总奖励。

Done(boolean)：返回是否应该重新设置（reset）环境。大多数游戏任务分为多个环节（episode），当 done=true 的时候，表示这个环节结束了。

Info(dict)：用于调试的诊断信息（一般没用）。

本题是典型的“智能体-环境循环”。每个时间步长（timestep），智能体选择一个行动，环境返回一个观察和奖励值。过程一开始调用 reset，返回一个初始的观察。并根据 done 判断是否再次 reset。算法采用随机算法：

```python
#导入openAI gym 以及 TensorFlow
import gym
import numpy as np
import tensorflow as tf

#用gym.make('CartPole-v0')导入gym定义好的环境，对于更复杂的问题则需要自定义环境
env = gym.make('CartPole-v0')

#第一步不用agent，采用随机策略进行对比
env.reset() #初始化环境
random_episodes = 0
reward_sum = 0
while random_episodes < 10:
    env.render()
    obsevation, reward, done, _ = env.step(np.random.randint(0, 2))
    #np.random.randint创建随机action，env.step执行action
    reward_sum += reward
    #最后一个action也获得奖励
    if done:
        random_episodes += 1
        print("Reward for this episodes was:", reward_sum)
        reward_sum = 0  #重置reward
        env.reset()
env.close()
```

以下结果作为我们的 baseline。

```
Reward for this episodes was: 48.0
Reward for this episodes was: 21.0
Reward for this episodes was: 50.0
Reward for this episodes was: 28.0
Reward for this episodes was: 16.0
Reward for this episodes was: 13.0
Reward for this episodes was: 21.0
Reward for this episodes was: 18.0
Reward for this episodes was: 27.0
Reward for this episodes was: 13.0
```

知识拓展：

空间：

每个游戏都有自己的 action_space 和 observation_space，表示可以执行的动作空间与观察空间。我们可以将其打印出来，看动作空间和观察空间的最大值或者最小值。

```python
import gym
env = gym.make('CartPole-v0')
print(env.action_space)
print(env.observation_space)
print(env.observation_space.high)
print(env.observation_space.low)
```

env.action_space：Discrete(2)　离散值 0 或 1

env.observation_space：Box(4,)　区间值，数组中包含四个数，取值如下

env.observation_space.high：array([2.4 , inf, 0.20943951, inf])

env.observation_space.low：array([-2.4 , -inf, -0.20943951, -inf])

环境：

Gym 包含一个测试问题集，每个问题成为环境（environment），可以用于自己的 RL 算法开发。这些环境有共享的接口，允许用户设计通用的算法。可以列出这些环境。

```
from gym import envs
print(envs.registry.all())
```

程序会列出一系列的 EnvSpec。它们为特定任务定义特定参数，包括运行的实验数目和最多的步数。比如，EnvSpec(Hopper-v1)定义了一个环境，环境的目标是让一个 2D 的模拟机器跳跃。EnvSpec(Go9x9-v0)定义了 9*9 棋盘上的围棋游戏。这些环境 ID 被视为不透明字符串。为了确保与未来的有效比较，环境永远不会以影响性能的方式更改，只能由较新的版本替代。 我们目前使用 v0 为每个环境添加后缀，以便将来的替换可以自然地称为 v1，v2 等。

再度优化：

构建策略网络

我们在策略网络使用一个带有一层隐藏层的 ANN，各种超参数为：nodes = 50, batch_size = 25, learning_rate = 0.1, discout_rate = 0.99。定义策略网络将 agent 对环境的 observation 作为输入，最后输入概率值选择 action。

```
import gym
import numpy as np
import tensorflow as tf

tf.reset_default_graph()

#用gym.make('CartPole-v0')导入gym定义好的环境，对于更复杂的问题则需要自定义环境
env = gym.make('CartPole-v0')
#env.reset() #初始化环境

#可以使用更复杂的深度神经网络
H = 50 #50个neure
batch_size = 25
learning_rate = 0.1
D = 4 #observation维度为4
gamma = 0.99 #discount rate
```

```
#定义策略网络具体结构：输入observation，输出选择action的概率
observations = tf.placeholder(tf.float32, [None, D], name = "input_x")
W1 = tf.get_variable("W1", shape = [D, H], initializer=tf.contrib.layers.xavier_initializer())
layer1 = tf.nn.relu(tf.matmul(observations, W1))
#隐藏层使用ReLu激活
W2 = tf.get_variable("W2", shape = [H, 1], initializer=tf.contrib.layers.xavier_initializer())
score = tf.matmul(layer1, W2)
probability = tf.nn.sigmoid(score)
#输出层使用Sigmoid将输出转化为概率

#定义优化器，梯度占位符，采用batch training更新参数
adam = tf.train.AdamOptimizer(learning_rate = learning_rate)
W1_grad = tf.placeholder(tf.float32,name = "batch_grad1")
W2_grad = tf.placeholder(tf.float32,name = "batch_grad2")
batchGrad = [W1_grad, W2_grad]
tvars = tf.trainable_variables()
updateGrads = adam.apply_gradients(zip(batchGrad, tvars))
```

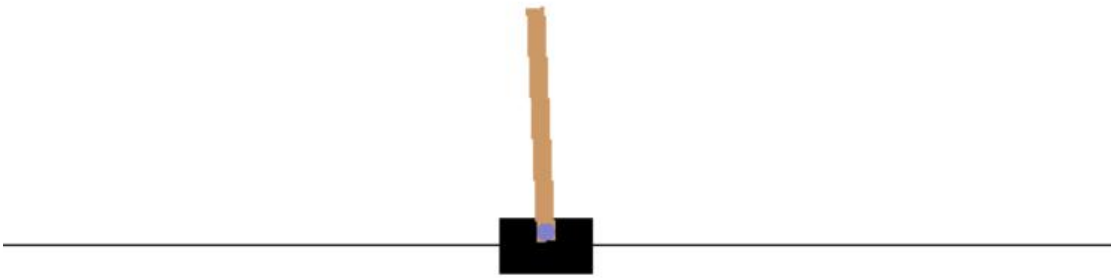
```
#计算每一个action的折现后的总价值
def discount_rewards(r):
    discounted_r = np.zeros_like(r)
    running_add = 0
    for t in reversed(range(r.size)):
        running_add = running_add * gamma + r[t]
        discounted_r[t] = running_add
    return discounted_r

#计算损失函数
input_y = tf.placeholder(tf.float32, [None, 1], name = "input_y")
advantages = tf.placeholder(tf.float32, name = "reward_signal")
#action的潜在价值
loglik = tf.log(input_y*(input_y - probability) + (1-input_y)*(input_y + probability))
#loglik为action的对数概率，P(act=1) = probablility,P(act=0) = 1-probablility
#action=1，loglik = tf.log(probability)
#action=0，loglik = tf.log(1-probability)
loss = -tf.reduce_mean(loglik * advantages)
newGrads = tf.gradients(loss, tvars)
#tvars用于获取全部可训练参数，tf.gradients求解参数关于loss的梯度
```

```
xs = [] #observation的列表
ys = [] #label的列表，  label = 1 - action
drs = [] #每个action的reward
reward_sum = 0 #累计reward
episode_num = 1 #每次实验index
total_episodes = 10000 #总实验次数
```

```
#创建会话
with tf.Session() as sess:
    rendering = False
    init = tf.global_variables_initializer()
    sess.run(init) #初始化状态
    observation = env.reset() #重置环境

    gradBuffer = sess.run(tvars)
    #创建存储参数梯度的缓冲器，执行tvars获取所有参数
    for ix,grad in enumerate(gradBuffer):
        gradBuffer[ix] = grad * 0 #将所有参数全部初始化为零
    #进入实验循环
    while episode_num <= total_episodes:
        #当某batch平均reward>100时，对环境进行展示
        if reward_sum/batch_size > 100 or rendering == True:
            env.render()
            rendering = True

        #将observation变形为网络输入格式
        x = np.reshape(observation, [1, D])
        #计算action=1的概率tfprob
        tfprob = sess.run(probability, feed_dict={observations: x})
        #(0,1)随机抽样，若随机值小于tfprob，action=1
        action = 1 if np.random.uniform() < tfprob else 0

        xs.append(x) #将observation加入列表xs
        y = 1 - action
        ys.append(y) #将label加入列表ys
        observation, reward, done, info = env.step(action)
        #env.step执行action，获取observation,reward,done,info
        reward_sum += reward
        drs.append(reward) #将reward加入列表drs
```

```
# done=True 即实验结束
if done:
    episode_num += 1  #一次实验结束，index+1
    epx = np.vstack(xs)
    epy = np.vstack(ys)
    epr = np.vstack(drs)
    xs, ys, drs = [], [], []

    #计算每一步的总价值，并标准化为均值为0标准差为1的分布
    discounted_epr = discount_rewards(epr)
    discounted_epr -= np.mean(discounted_epr)
    discounted_epr /= np.std(discounted_epr)

    #将epx epy epr 输入神经网络，newGrads求梯度
    tGrad = sess.run(newGrads, feed_dict={observations: epx, input_y: epy, advantages: discounted_epr})
    for ix,grad in enumerate(tGrad):
        gradBuffer[ix] += grad
        #将梯度叠加进gradBuffer

    #当试验次数达到batch_size整数倍时
    if episode_num % batch_size == 0:
        sess.run(updateGrads, feed_dict={W1_grad: gradBuffer[0], W2_grad: gradBuffer[1]})
        #updateGrads将gradBuffer梯度更新到模型参数中

        for ix, grad in enumerate(gradBuffer):
            gradBuffer[ix] = grad * 0
            #清空gradBuffer，为下一个batch做准备

        print('Average reward for episode %d : %f.' %(episode_num, reward_sum/batch_size))
        if reward_sum/batch_size > 200:
            print("Task solved in", episode_num, 'episodes!')
            break
        reward_sum = 0

    observation = env.reset()
    #每次实验结束，重置任务环境
```

结果：



```
Average reward for episode 25 : 20.960000.
Average reward for episode 50 : 37.760000.
Average reward for episode 75 : 69.080000.
Average reward for episode 100 : 86.680000.
Average reward for episode 125 : 107.680000.
Average reward for episode 150 : 157.640000.
Average reward for episode 175 : 182.000000.
Average reward for episode 200 : 198.840000.
Average reward for episode 225 : 200.000000.
Average reward for episode 250 : 200.000000.
Average reward for episode 275 : 200.000000.
Average reward for episode 300 : 200.000000.
Average reward for episode 325 : 200.000000.
Average reward for episode 350 : 200.000000.
Average reward for episode 375 : 200.000000.
Average reward for episode 400 : 200.000000.
Average reward for episode 425 : 200.000000.
Average reward for episode 450 : 200.000000.
Average reward for episode 475 : 200.000000.
Average reward for episode 500 : 200.000000.
Average reward for episode 525 : 195.760000.
Average reward for episode 550 : 195.000000.
Average reward for episode 575 : 200.000000.
Average reward for episode 600 : 200.000000.
Average reward for episode 625 : 200.000000.
Average reward for episode 650 : 200.000000.
Average reward for episode 675 : 200.000000.
```

五、错误分析

错误一：

spyder 3.3.2 requires pyqt5<5.10; python_version >= "3", which is not installed.

```
spyder 3.3.2 requires pyqt5<5.10; python_version >= "3", which is not installed.
```

问题解析：将 pyqt5 更新到 5.11.3 版本之后，spyder 3.3.2 弹出要求说 要求 pyqt5 的版本低于 5.10

解决方法：

1、卸载 pyqt5：命令为：pip uninstall pyqt5

2、安装需要的版本（低于 5.10），使用下面的命令可以得到所有的版本号：pip install pyqt5==6.12.0 (这里输入一个比最新的还要高的版本号，它就会提示错误，并且列出所有的版本号，选择满足要求的最新的一个即可。)

3、使用 pip install pyqt5==5.9.2（这是查询到的低于 5.10 的最高版本）

结果：

```
Successfully installed pyqt5-5.9.2 sip-4.19.8
```

错误二:

记录和加载结果运用 monitor 时出现：

env.monitor is deprecated. Wrap your env with gym.wrappers.Monitor to record data.

```
AttributeError: 'CartPoleEnv' object has no attribute 'monitor'
```

解决方法：

from gym.wrappers import Monitor

```
    env=Monitor(directory='/tmp/cartpole-experiment-0201',video_callable=False,write_upon_
reset=True)(env)
    env.close()
```