Project B

学号：2017202085

姓名：陕杰才

一、实验背景：

Gym是一个研究和开发强化学习相关算法的仿真平台，无需智能体先验知识，并兼容常见的数值运算库如 TensorFlow、Theano等。OpenAI Gym由以下两部分组成：

Gym开源库：测试问题的集合。当你测试强化学习的时候，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：提供一个站点和API（比如经典控制问题：CartPole-v0），允许用户对他们的测试结果进行比较。

简单来说OpenAI Gym提供了许多问题和环境（或游戏）的接口，而用户无需过多了解游戏的内部实现，通过简单地调用就可以用来测试和仿真。

游戏背景：环境中有一辆小车，在一维无阻力的轨道上运动，车上连接一根活动杆，杆会左右摇摆。游戏规则很简单，我们要操纵我们的小车左右移动，使它上面的木棒能够保持平衡。当小车偏离中心4.8个单位，或杆的倾斜超过15度，任务失败。

官方文档示例代码：

import gym

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

for i\_episode in range(20):

observation = env.reset()

for step in range(100):

env.render()

print(observation)

action = env.action\_space.sample()

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

if done:

print("Episode finished after {} timesteps".format(step+1))

break

由于是基于随机生成的action，所以会导致小车一直在山底左右移动

import gym

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

observation = env.reset()

print(observation)

#> [-0.00478028 -0.02917182 0.00313288 0.03160127]

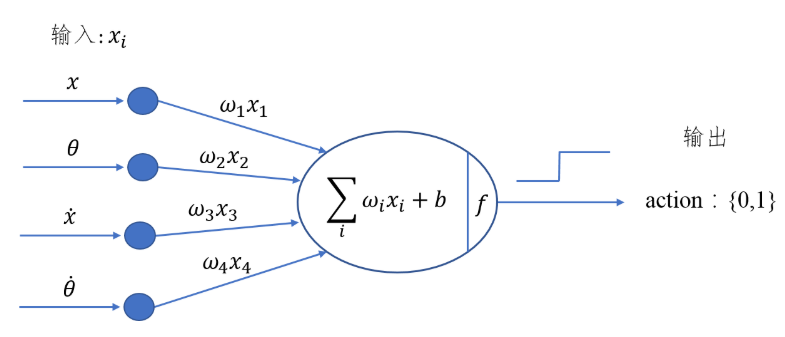
以上代码显示了初始状态下的取值，每次调用*env.reset*( )将重新产生一个初始状态。打印出的*observation*的四个元素分别表示了小车位置、小车速度、杆子夹角及角变化率。

游戏奖励（reward）：

    在gym的Cart Pole环境（env）里面，左移或者右移小车的action之后，env会返回一个+1的reward。其中CartPole-v0中到达200个reward之后，游戏也会结束，而CartPole-v1中则为500。最大奖励（reward）阈值可通过前面介绍的注册表进行修改。

二、基于爬山算法

为了能够有效控制倒立摆首先应建立一个控制模型。明显的，这个控制模型的输入应该是当前倒立摆的状态（*observation*）而输出为对当前状态做出的决策动作（*action*）。决定倒立摆状态的observation是一个四维向量，包含小车位置、杆子夹角、小车速度及角变化率，如果对这个向量求它的加权和，那么就可以根据加权和值的符号来决定采取的动作（action），用sigmoid函数将这个问题转化为二分类问题，从而可以建立一个简单的控制模型。模型如下图所示：

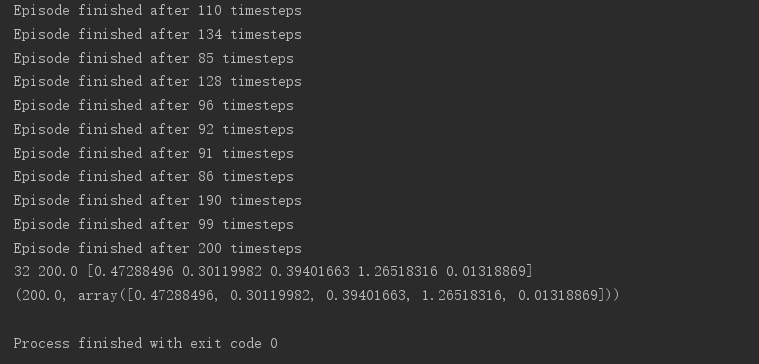


上图的实际功能与神经网络有几分相似，但比神经网络要简单得多。通过加入四个权值，我们可以通过改变权重值来改变决策（*policy*），即有加权和为了得到一组较好的权值从而有效控制倒立摆，我们可以采用爬山算法（*hill climbing algorithm*）进行学习优化。爬山算法是一种启发式方法，是对深度优先搜索的一种改进，它利用反馈信息帮助生成解的决策。

完整代码如下：

# coding: utf8  
  
import numpy as np  
import gym  
import time  
  
def get\_action(weights, observation):# 根据权值对当前状态做出决策  
 wxb = np.dot(weights[:4], observation) + weights[4] # 计算加权和  
 if wxb >= 0:# 加权和大于0时选取动作1，否则选取0  
 return 1  
 else:  
 return 0  
  
def get\_sum\_reward\_by\_weights(env, weights):  
# 测试不同权值的控制模型有效控制的持续时间（或奖励）  
 observation = env.reset() # 重置初始状态  
 sum\_reward = 0 # 记录总的奖励  
 for t in range(1000):  
 # time.sleep(0.01)  
 # env.render()  
 action = get\_action(weights, observation) # 获取当前权值下的决策动作  
 observation, reward, done, info = env.step(action)# 执行动作并获取这一动作下的下一时间步长状态  
 sum\_reward += reward  
 # print(sum\_reward, action, observation, reward, done, info)  
 if done:# 如若游戏结束，返回  
 print("Episode finished after {} timesteps".format(t + 1))  
 break  
 return sum\_reward  
  
  
def get\_weights\_by\_random\_guess():  
# 选取随机猜测的5个随机权值  
 return np.random.rand(5)  
  
def get\_weights\_by\_hill\_climbing(best\_weights):  
# 通过爬山算法选取权值（在当前最好权值上加入随机值）  
 return best\_weights + np.random.normal(0, 0.1, 5)  
  
def get\_best\_result(algo="random\_guess"):  
 env = gym.make("CartPole-v0")  
 np.random.seed(10)  
 best\_reward = 0 # 初始最佳奖励  
 best\_weights = np.random.rand(5) # 初始权值为随机取值  
  
 for iter in range(10000):# 迭代10000次  
 cur\_weights = None  
  
 if algo == "hill\_climbing": # 选取动作决策的算法  
 # print(best\_weights)  
 cur\_weights = get\_weights\_by\_hill\_climbing(best\_weights)  
 else: # 若为随机猜测算法，则选取随机权值  
 cur\_weights = get\_weights\_by\_random\_guess()  
 # 获取当前权值的模型控制的奖励和  
 cur\_sum\_reward = get\_sum\_reward\_by\_weights(env, cur\_weights)  
  
 # print(cur\_sum\_reward, cur\_weights)  
 # 更新当前最优权值  
 if cur\_sum\_reward > best\_reward:  
 best\_reward = cur\_sum\_reward  
 best\_weights = cur\_weights  
 # 达到最佳奖励阈值后结束  
 if best\_reward >= 200:  
 break  
  
 #print(iter, best\_reward, best\_weights)  
 return best\_reward, best\_weights  
  
# 程序从这里开始执行  
print(get\_best\_result("hill\_climbing")) # 调用爬山算法寻优并输出结果  
  
#env = gym.make("CartPole-v0")  
#get\_sum\_reward\_by\_weights(env, [0.22479665, 0.19806286, 0.76053071, 0.16911084, 0.08833981])

结果：



爬山算法本质是一种局部择优的方法，效率高但因为不是全局搜索，所以结果可能不是最优。在这里采用的模型较为简单，如若想要获得更好的学习效果可以考虑更加复杂的模型，如深度神经网络。