Project2 利用爬山算法优化倒立摆问题

孙一丹 2016201013

前言

在第一阶段的实验中，完成了对openAI Gym的环境搭建，并熟悉了倒立摆小车游戏。然而在第一阶段中，对小车动作的控制仅停留在random的阶段，于是可以发现游戏启动后，小车很快就滑出了页面，或者倒杆很快就倒下了。

在第二阶段决定实现对倒摆实现有效控制。通过回顾发现，在目前所讲的众多模型中，多数为进行预测或分类的学习模型，仅有少数搜索模型较为适合解决当前问题。经过分析后，决定选择爬山算法来解决此问题。

爬山算法

对于倒立摆模型而言，需要根据输入来判断输出，输入为当前倒立摆的状态。在第一阶段的实验中可知，倒立摆状态由一个四维向量表示，包括小车位置(x)、杆子夹角(θ)、小车速度(d)和角变化率(ρ)。对这个向量求加权和，就可以加权值的符号来决定采取的动作。可以使用$sigmoid$函数将这个问题转化为二分类问题，从而建立控制模型。即：

$$H_s = w_1 x + w_2 \theta + w_3 d + w_4 \rho + b$$

若H_s的符号为正则判定输出为1，否则为0。于是如何确定加权系数成了模型的关键，这里采用爬山算法进行学习优化。

基本思路为，在每次迭代时给当前取得的最优权重加上一组随机值，如果加上这组值使得有效控制倒立摆的持续时间变长了，就更新它为最优权重，如果没有得到改善就保持原来的值不变，直到迭代结束。在迭代过程中，模型的参数会不断得到优化，最终可以得到一组最优的权值作为控制模型的解/

模型的代码如下：

```
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
            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
```

运行代码发现，多数情况下在迭代100以内即可找到适合的权重使得奖励达到最大值，列举结果如下：

iter(迭代次数)	奖励值	权重
28	200.0	[0.86113962 0.06753786 0.81176208 1.58647361 -0.0173375]
73	200.0	[0.66109588 -0.56633614 0.65656425 1.54621575 -0.0323978]
52	200.0	[0.90373154 -0.53520671 0.68904004 1.60874119 -0.03955546]
96	200.0	[0.82962686 -0.22633766 0.70143391 1.56807736 -0.0680506]
94	200.0	[0.80179849 -0.55028972 0.6235989 0.88775632 -0.02865346]