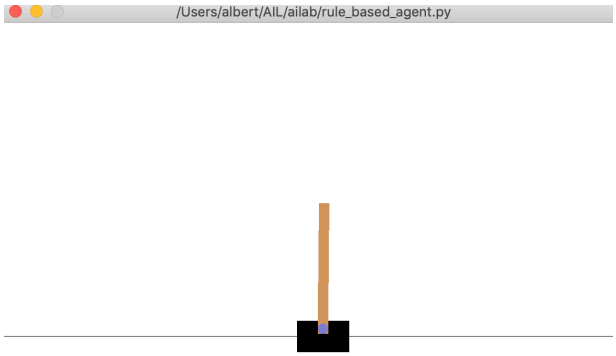
李昕旸(2017202105) 实验三报告

实验选题： open ai gym
实验环境： Mac OS X Python 3.7
实验概述：

如图所示，本实验装置由一个可以移动的黑色小车和一个细长的木棍组成，我们的主要任务是左右移动小车使得木棍在长时间内不倾倒且小车不撞上左右边缘。本实验在每个状态给我们提供一个长度为4的向量，分别代表：小车位置、小车速度、木棒角度正弦值、木棒角度变化率，上面四个值的负数表示向左、正数表示向右；而我们需要返回一个决策给系统，返回0代表往左走，返回1代表往右走。



实验过程：

1. 刚开始打算使用强化学习的 Q-learning 方法 但经过实验发现 这个实验的 reward 反馈不明显（具体因为：木棍倒下可能由于之前5-10步决策的错误，但前面错误的步数的决策还是得到了1的reward，导致训练数据很差、没有区分度）
2. 解决上述问题的思路，由于上次的经典机器学习模型已经得到了很高的评分，所以打算利用上次经典机器学习的得到的模型，运行若干次得到相对很好的测试数据（即每次的观察向量（四维）+ 经典机器学习模型得到的向左或者向右的决策（0或者1））作为训练数据，那么这个问题已经转化为使用深度学习的二分类问题，得到训练数据的具体代码为：

```
def get_train_data():
    X = []
    y = []
    agent = ClassicMachineLearningAgent()
    agent.train()
    env = gym.make(ENV_NAME)
    for i in range(30):
        obs = env.reset()
        sum_reward = 0.0
        while True:
            action = agent.get_action(obs)
            X.append(obs[:])
            y.append(action)
            obs, reward, done, info = env.step(action)
            sum_reward += reward
            if done:
                # print(sum_reward)
                break
    env.close()
    print("train_data_ready")
    return np.array(X), np.array(y)
```

3. 初始化深度学习模型，使用简单的全连接神经网络，中间加两个全连接层，使用relu作为激活函数，使用交叉熵作为损失函数（训练集的label只有两个 0 1），使用AdamOptimizer（自适应矩估计优化器），迭代次数设为25，优化率设为 0.001，如下图：

```
def __init__(self):
    tf.reset_default_graph()
    self.X_data, self.y_data = get_train_data()
    # 定义训练数据的占位符， x是特征值， y是标签值(0 或 1)
    self.x = tf.placeholder(tf.float32, [1, 4], name="X")
    self.y = tf.placeholder(tf.float32, [1, 1], name="Y")

    with tf.name_scope("Model1"):
        def model(x):
            d1 = tf.nn.relu(tf.layers.dense(x, 128))
            d2 = tf.nn.relu(tf.layers.dense(d1, 32))
            d3 = tf.layers.dense(d2, 1)
            return d3

        self.pred = model(self.x)

    self.train_epochs = 25  # 迭代次数
    self.learning_rate = 0.001  # 学习率

    # 定义损失函数 采用交叉熵作为损失函数
    with tf.name_scope("LossFunction"):
        self.loss_function = tf.nn.sigmoid_cross_entropy_with_logits(labels=self.y, logits=self.pred)

    # 使用自适应矩估计优化器 设置学习率和优化目标损失最小化
    self.optimizer = tf.train.AdamOptimizer(self.learning_rate).minimize(self.loss_function)

    self.sess = tf.Session()
    init = tf.global_variables_initializer()
    self.sess.run(init)
```

4. 迭代训练集进行训练，训练过程特别加了一步打乱数据顺序，防止过拟合，如下图

```
def train(self):

    loss_list = []  # 用于保存loss的值
    # 迭代训练
    for epoch in range(self.train_epochs):
        loss_sum = 0.0
        for xs, ys in zip(self.X_data, self.y_data):
            xs = xs.reshape(1, 4)
            ys = ys.reshape(1, 1)

            _, loss = self.sess.run([self.optimizer, self.loss_function], feed_dict={self.x: xs, self.y: ys})

            loss_sum = loss_sum + loss  # 累加损失

        X_train, y_train = shuffle(self.X_data, self.y_data)  # 打乱数据顺序 避免过拟合假性学习

        loss_average = loss_sum / len(y_train)  # 所有数据的平均损失
        loss_list.append(loss_average)

    # print(loss_list)
```

5. 进行测试，由于分类是 0 和 1，则全连接神经网络拟合出的 < 0.5 算作类型 0（向左）；
>= 0.5的算作类型 1（向右）

```
def car_test(self):

    env = gym.make(ENV_NAME)
    for i in range(3):
        obs = env.reset()
        sum_reward = 0.0
        while True:
            time.sleep(0.01)
            env.render()
            obs_xs = np.array(obs).reshape(1, 4)
            choice_tmp = self.sess.run([self.pred], feed_dict={self.x: obs_xs})
            real_choice = 0 if choice_tmp[0] < 0.5 else 1
            obs, reward_tmp, done, info = env.step(real_choice)
            sum_reward += reward_tmp
            if done:
                print(sum_reward)
                break
```

实验结果
实验效果优 在三次测试中 小车均拿到了500分的最高分

```
500.0
500.0
500.0
```