**cart-pole实验报告**

2016202106

张艺邻

1. **背景介绍**
   1. **cartpole**

cartpole-v0和cartpole-v1是openaigym上的一个系列的小车平衡游戏。小车上有一个可摇晃的杆子，小车可被操纵向左移或向右移，在杆子倾斜角度大于十五度或小车距离绝对值大于2.4时，游戏失败；否则，玩家获得点数为1的回报值，游戏继续。

这个平台的接口为玩家提供了方便的接口，其中env.step(parameter)接受action（值为0或1）作为参数，返回observation，reward，done，info。

observation是一个大小为4的一维数组，值分别是小车当前的位置，速度，旋转角度，和旋转速度；reward是action之后的报酬，若小车未失去平衡则为1；done为1则小车失去平衡，反之为0；info为调试信息，通常不用。

observation

| **Num** | **Observation** | **Min** | **Max** |
| --- | --- | --- | --- |
| 0 | Cart Position | -2.4 | 2.4 |
| 1 | Cart Velocity | -Inf | Inf |
| 2 | Pole Angle | ~ -12° | ~ 12° |
| 3 | Pole Velocity At Tip | -Inf | Inf |

action

| **Num** | **Action** |
| --- | --- |
| 0 | Push cart to the left |
| 1 | Push cart to the right |

玩家的任务即是通过观察值observation，制定策略决定左移（action = 0）或右移（action = 1），使得小车越长时间保持平衡越好

**2.pytorch**

Pytorch由Facebook开源的神经网络框架，Torch 是一个经典的对多维矩阵数据进行操作的张量（tensor ）库，在机器学习和其他数学密集型应用有广泛应用。与Tensorflow的静态计算图不同，pytorch的计算图是动态的，可以根据计算需要实时改变计算图。作为经典机器学习库 Torch 的端口，PyTorch 为 Python 语言使用者提供了舒适的写代码选择。

PyTorch的设计追求最少的封装，设计遵循tensor→variable(autograd)→nn.Module 三个由低到高的抽象层次，分别代表高维数组（张量）、自动求导（变量）和神经网络（层/模块），而且这三个抽象之间联系紧密，可以同时进行修改和操作。

在本次实验中，主要利用pytorch完成了机器学习的线性回归，逻辑回归和深度强化学习

1. **强化学习**

**1.相关概念**

* + 1. 行为策略（behavior policy）

我们把用来指导个体产生与环境进行实际交互行为的策略称为行为策略

* + 1. 目标策略（target policy）

用来评价状态或行为价值的策略或者待优化的策略称为目标策略

* + 1. 同步策略学习（on-policy learning）

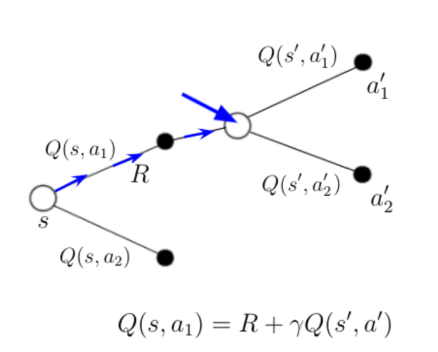
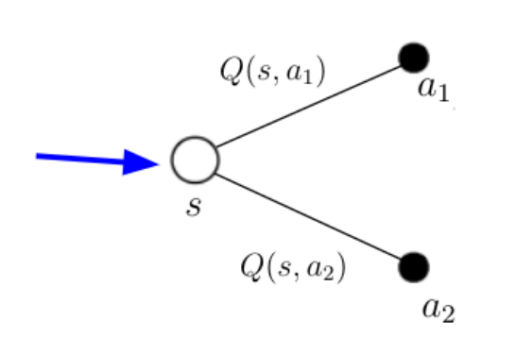
如果个体在学习过程中优化的策略与自己的行为策略是同一个策略时，这种学习方式称为同步策略学习

* + 1. 异步策略学习（off-policy learning）

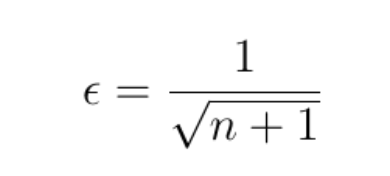
如果个体在学习过程中优化的策略与自己的行为策略是不同的策略时，这种学习方式称为异步策略学习

**2.算法详情**

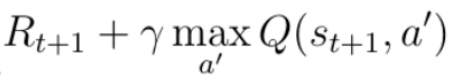
Qlearning试图维护一个表格，通过该表格，我们可以为每一个状态（state）上进行的每一个动作（action）计算出最大的未来奖励（reward）的期望。报酬分为长期报酬和短期报酬，其中，长期报酬与它后一结点的最大报酬成正比，短期报酬则是采取动作后立即获得的报酬。



本次实验中，行为策略是以epsilon的概率，随机选择一个action，以（1-episilon）的概率，选择当前状态下的最优决策。之所以要进行epislon概率下的随机选择，是为了搜集更多的数据，寻求可能的新路径。epsilon参数应当随步数增加而减少。我们采取的是epsilon = 1 / math.sqrt(timestep + 1)

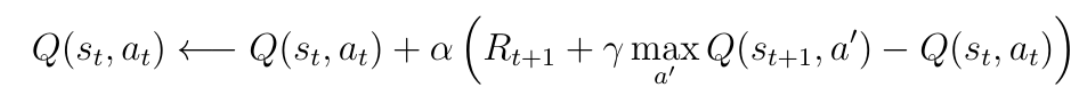


目标函数为



但是，由于state的值是连续的，即状态有无穷多个，所以并没有办法建立这样一个具体的表格。所以，我们用一个深度神经网络来表示Q(s, a)。

我们的数据集为（state， action， reward， next\_state），在每次学习中，在数据集中随机抽取部分数据，如果该数据没有下一状态，则它应当为对应的reward；若它有下一状态，则它应当为reward+gamma\*max(Q(next\_state, any\_action))。我们以一定比率缩小样本与正确数据的距离，最终使网络收敛



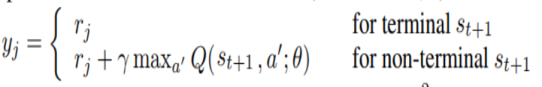
最终我们的算法步骤就是

a）初始化小车

b)以epsilon = 的概率随机选择action；反之 

c)将(state, action, reward, next\_state)加入数据集

d)从数据集随机抽取部分样本

计算y 

e)在损失函数loss(y, Q(s, a))上实行一次梯度下降以更新网络