华中农业大学
HUAZHONG AGRICULTURAL UNIVERSITY

强化学习课程报告

姓名 卢东黎

学号 2021317220603

院所 信息学院人工智能 2102

2024 年 4 月 4 日

目录

1 本课程总结

1.1 概述

本课程开始于 2 月 26 日（周一三四节课），结束于 4 月 3 日（周三五六节课）。共 12 次课，讲授了七个章节的内容。此外还包含 4 次实验以及 5 次课后作业。涉及主要内容有强化学习概述 (第一章)、多臂老虎机（第二章）、有限马尔可夫过程（第三章）、动态规划（第四章）、蒙特卡洛方法（第五章）、时间差分学习（第六章）、策略梯度（第七章）。

1.2 课程评价

写在前面：学生个人对课程总体评价—— **“强化学习是一门要求数学理论扎实，尤其是矩阵运算和概率论基础，代码理解能力较强，尤其是多臂老虎机章节和蒙特卡洛的代码实操需要很多的课外时间反复研读，思路要求打开的科目。课堂上何老师的讲解生动，但是如果缺少先验知识的积累，课堂上是很难消化推理公式的，这一点深有体会。因为有时候的走神，真的会不理解老师讲什么。所以这门课对课前预习和课后钻研是有要求的。**

1.3 课程建议

以下是我的课程建议：

- 1. 课前发放 PPT 等易理解的导读资料；
- 2. 可以搭配一些动态演示的视频辅助消化；
- 3. 带数值的计算课堂上可以再多一些，帮助了解概率相关的公式推导；
- 4. 答疑问卷收集，重点重讲部分数学公式。

2 第一章回顾

羚羊生存问题、大学新生表现、狂飙电视剧孙子兵法演绎、机器人控制、自动驾驶汽车、股票交易是理解强化学习比较好的例子。

2.1 机器人学习主流范式

重点内容是机器学习三个主流范式，也就是说清楚强化学习这一门学科的定位。包括有监督学习、无监督学习、强化学习。

2.2 强化学习基础概念

强化学习任务：智能体 (agent) 在复杂、不确定环境 (environment)，通过不断交互，最大化它能获得的奖励重要的概念解释：

- 智能体（agent）：强化学习的主体。由谁做动作或决策，谁就是智能体。
- 环境（environment）是与智能体交互的对象，可以抽象地理解为交互过程中的规则或机理。

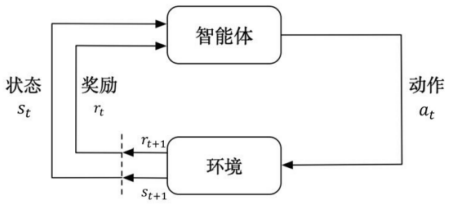

图 1: 强化学习示意图

另外本章节需要重点了解的是强化学习特点：

- 智能体（或者学习器）从环境中拿到的样本是时间序列数据，不满最独立同分布假设
- 没有监督标签，只有奖励信号，需要自己去发现那些动作带来最多的奖励
- 奖励会有延迟，需要发掘具有更多潜在收益的动作
- 动作会影响获取的数据
- 学习过程是一个不断试错探索的过程（trial-and-error exploration）。

3 第二章回顾

掌握多臂老虎机基础概念和代码实操（这个在实验课已经体验过了）。

3.1 多臂老虎机

多臂老虎机问题又称为 MBA 问题。有 K 个赌博机，每个赌博机有一定概率 P 吐出硬币，但是我们不知道这个概率是多少，每个赌博机吐出的硬币价值 V 也是不一样的，现在有 T 次机会选择赌博机，怎么选才能使得到的硬币总价值最大?

概念	说明
智能体	算法
环境	K 臂老虎机
动作	选择哪一个老虎机臂
状态	做出动作后，老虎机在那里不会自己发生改变
动作空间大小	|A|
状态空间大小	1
奖励	老虎机的返回奖励
学习目标	在操作 T 次拉杆后获得尽可能高的累积奖励

表 1: K 臂老虎机其实是简化版的强化学习

3.2 数值计算

下面我具体用数值举例子: 假设 $K = 5$ $P = [0.1, 0.9, 0.3, 0.2, 0.7]$ $V = [5, 3, 1, 7, 4]$ $T = 1000000$

有如下三种情况:

- 如果每次都选期望价值最高的 4 号赌博机，可以获得的最高总价值为 2800000。
- 如果每次都选期望价值最低的 2 号赌博机，可以获得的最低总价值为 300000。
- 如果随机选取赌博机，可以获得的期望总价值为 1540000。

“仅探索”（exploration-only）算法就是将机会平均分配给每一个赌博机，随机挑选赌博机。

“仅利用”（exploitation-only）算法就是选取当前平均价值最高的那台赌博机。

但是由于尝试的次数有限，所以要在探索与利用之间进行权衡，这也是强化学习面临的一大问题：探索-利用窘境（Exploration-Exploitation dilemma）。

重要内容：贪心算法, 每次以 的概率来探索，即以均匀概率随机挑选一个摇臂。以 1 − 的概率利用，即挑选当前平均价值最高的那个摇臂。

由于算法和代码在实验课已经操作过，这里不赘述了。

4 第三章回顾

有限马尔可夫决策过程是第三章重点内容。主要是了解三个过程：马尔可夫过程 MP、马尔可夫奖励过程 MRP 和马尔可夫决策过程 MDP。

4.1 马尔可夫过程

若随机过程满足马尔科夫性，则称为马尔可夫过程，可以用一个二元组 $< S, P >$，其中 S 是有限数量的状态集，P 是状态转移概率矩阵。

4.2 马尔可夫奖励过程

马尔可夫奖励过程 (Markov Reward Process, MRP) 是马尔可夫链再加上了一个奖励函数。在 MRP 中，转移矩阵和状态都是跟马尔可夫链一样的，多了一个奖励函数 (reward function)

$$G_t = R_{t+1} + G_{t+1}$$

是折扣因子,G_t 是 t 时刻最大期望回报,R_{t+1} 是 $t+1$ 时刻的奖励回报。

4.3 马尔可夫决策过程

强化学习的马尔科夫决策过程是一个五元组 $< S, A, P, R, >$，其中 A 表示动作集，R 表示奖励集， 表示折扣因子。

概念	说明
回报公式	$G_t = R_{t+1} + G_{t+1}$
策略公式	$\pi(a\|s) = P(A_t = a\|S_t = s)$
确定性策略	$a = \pi(s)$
随机策略	$\pi(a\|s) = P(A_t = a\|S_t = s)$

表 2: 马尔可夫决策过程有关的几个定义

基于马尔科夫过程的强化学习的流程为：在 t 时刻，Agent 基于状态 S_t ，根据策略 $(a|s)$ 采样出动作 A_t ，然后根据状态转移概率 $p(s'|s, a)$，产生新的状 S_{t+1} ，Agent 可以接收到一个奖励 R_{t+1} ,..., 以此不断运行直到终止。

有限 MDP 特点：

- 状态空间 A 包含有限个状态
- 动作空间 S 包含有限个动作
- 奖励函数 R 包含有限种奖励数值

MDP 环境特点：

- MDP 环境是抽象且灵活的
- 时间步 (time-step) 不一定指固定的实际时间间隔，例如，可以是决策制定和行动执行的任意连续阶段；
- 动作可以是低级控制（例如施加在机器人手臂电机上的电压）或高级决策选择（例如是否吃午饭）；
- 状态可以采用各种形式，例如低级感知，如直接传感器读数，图像和点云；

- 给定一个强化学习任务，借助 MDP 对任务进行形式化描述是理解简化问题的第一步。

最后需要补充的是寻优策略：穷举法、贝尔曼方程。

5 第四章回顾

第四章主要讲述了强化学习中的动态规划方法。动态规划是一种优化方法，主要解决一些具有最优子结构的问题。在给定一个用马尔可夫决策过程（MDP）描述的完备环境模型的情况下，其可以计算最优的策略。

5.1 动态规划

DP 的核心思想是**使用价值函数来结构化地组织对最优策略的搜索**。

这一章节中需要重点掌握的几个概念：策略评估、策略改进和策略迭代。

对于策略评估，给定任意策略 π，如何计算 π 对应的状态价值函数 $v_\pi(\ )$, 即评估当前策略在各状态下的长期回报。计算策略对应的状态价值函数的目的是为了帮助进行策略改进。对于策略改进，假设给定一个确定性策略 π, 我们已经计算其价值 $v_\pi(\ )$。，对于某一状态 s, 通过选择一个新动作 $a(\ \pi(s))$, 是否演进出更好的策略, 如果遵循当前策略 π，选择动作 $\pi(s)$ 能够拿到的价值是 $v_\pi(s)$ ，那么可否考虑选择一个其他动作 $a(\ \pi(s))$，之后在后续状态继续遵循老策略 π，看看价值是否进一步提升。对于策略迭代，实际上就是策略评估、策略改进交替进行的过程，使得长期回报收敛到最优值。由于公式证明太长，在这里不展示了。

6 第五章回顾

第五章我们重点学习了蒙特卡洛方法，涉及 MC 预测、first-visit MC 方法、every-visit MC 方法、on-policy MC 和 off-policy 方法。这一章总体来说难度我认为是比较大的。

6.1 动态规划的局限性

已知动态规划的价值-状态更新函数如下：从上面的公式可以看出，动态规划有以

$$\begin{aligned} v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_k(s')\Big], \end{aligned}$$

图 2: 动态规划公式

下局限：

（1）每更新一个状态的价值，需要遍历后续所有状态的价值，复杂度较高。

（2）很多时候状态转移概率 $p(s', r|s, a)$ 未知，这时候我们无法使用动态规划求解。

蒙特卡洛并非一个特定的算法，而是一类随机算法的统称，其基本思想是：**用事件发生的“频率”来替代事件发生的“概率”，这能解决上面所说的“状态转移概率未知”的问题。通过多次采样，使用该事件发生的频率来替代其发生的概率**。

蒙特卡洛方法的特点如下：

（1）可以通过随机采样得到近似结果

（2）采样次数越多，结果越接近于真实值。

蒙特卡洛的整体思路是：模拟-> 抽样-> 估值

下面是蒙特卡洛方法的一个迭代过程： 策略评估迭代

- 探索 - 选择一个状态-动作。
- 模拟 - 使用当前策略，进行一次模拟，从当前到结束，随机产生一段情节（episode）。
- 抽样 - 获得这段情节上的每个状态的回报，记录到集合。

策略优化 - 优化新的行动价值优化策略

6.2 on-policy 和 off-policy 的区别

on-policy：生成样本的 policy（value function）跟网络更新参数时使用的 policy（value function）相同。典型为 SARSA 算法，基于当前的 policy 直接执行一次动作选择，然后用这个样本更新当前的 policy，因此生成样本的 policy 和学习时的 policy 相同，算法为 on-policy 算法。该方法会遭遇探索-利用的矛盾，光利用目前已知的最优选择，可能学不到最优解，收敛到局部最优，而加入探索又降低了学习效率。epsilon-greedy 算法是这种矛盾下的折衷。优点是直接了当，速度快，劣势是不一定找到最优策略。

off-policy：生成样本的 policy（value function）跟网络更新参数时使用的 policy（value function）不同。典型为 Q-learning 算法，计算下一状态的预期收益时使用了 max 操作，直接选择最优动作，而当前 policy 并不一定能选择到最优动作，因此这里生成样本的 policy 和学习时的 policy 不同，为 off-policy 算法。先产生某概率分布下的大量行为数据（behavior policy），意在探索。从这些偏离（off）最优策略的数据中寻求 target policy。当然这么做是需要满足数学条件的：假設 是目标策略，μ 是行为策略，那么从 μ 学到 的条件是：$(a|s) > 0$ 必然有 $(a|s) > 0$ 成立。两种学习策略的关系是：on-policy 是 off-policy 的特殊情形，其 target policy 和 behavior policy 是一个。劣势是曲折，收敛慢，但优势是更为强大和通用。其强大是因为它确保了数据全面性，所有行为都能覆盖。

6.3 蒙特卡洛方法的优势/劣势

优势

- 蒙特卡洛方法可以从交互中直接学习优化的策略，而不需要一个环境的动态模型。环境的动态模型指的是表示环境的状态变化是可以完全推导的。表明了解环境的所有知识。
- 蒙特卡洛方法可以用于模拟（样本）模型。
- 可以只考虑一个小的状态子集。
- 每个状态价值的计算是独立的。不会影响其他的状态价值。

劣势

- 需要大量的探索（模拟）。
- 基于概率的，不是确定性的。

此外，对于具体的公式推导和证明这里不赘述，因为课堂上已经讲的很清晰了，实验课也有练习到。

7 第六章回顾

第六章主要讲述时间差学习 TD。

7.1 TD 预测

TD 预测和 Monte Carlo (MC) 方法一样, 使用经验数据解决预测问题; 和动态规划一样，用到自举思想。可以看成是两种的结合。TD 方法，仅需要下个时刻数据 $R_(t+1)$ ，即可估算更新:

$$V(S_t) \leftarrow V(S_t) + [R_{t+1} + V(S_{t+1}) - V(S_t)]$$

其中 $R_{t+1} + V(S_{t+1})$ 叫做时序差分更新目标。

总的俩说 TD(0), DP, 和 Monte Carlo 在价值估计分类方法方面可以分为：有模型方法 (Model-based)：DP，不依赖智能体和环境交互数据免模型方法 (Model-Free)：TD, MC, 依赖于智能体和环境交互数据

7.2 策略提升

On-Policy TD Control 是基于预测的价值函数，可进一步对策略进行改进，未知环境中，需要估计状态-动作价值函数 $q_\pi(s, a)$，for all $s\ S, a\ A$，而非状态价值函数 $v_\pi(s)$，for all $s\ S$。Sarsa 算法是直接更新 Q 表，更新后，即可更新策略:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + [R_{t+1} + Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

7.3 Satasa 算法

基于 Sarsa 设计同策略（on policy）控制算法。估计行为策略 π 的动作价值函数 q_π，同时基于 q_π 朝着贪婪方向改进 π。Sarsa 算法中 π 可以使用“e-贪婪”或“e-软”策略。只要所有状态-动作对均被无限次访问，则 Sarsa 以概率 1 达到最优策略和动作-价值函数通过设置 $e = 1/t$，策略收敛到贪婪策略

7.4 Q 学习算法

Off-policy TD Control, 学习到的动作价值 Q，直接逼近 q_* 。最优的动作价值函数和当前遵循的策略无关。

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + [R_{t+1} + max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

行为策略用的是 e-贪婪策略，而目标策略用的是 Q 表上的贪婪策略。

8 第七章回顾

最后一章节主要讲述了策略梯度方法，类似于深度学习神经网络梯度优化计算。在损失地貌上通过不断的梯度迭代，追求最小化损失值。

策略学习的意思是通过求解一个优化问题，学出最优策略函数或它的近似函数。策略梯度算法不需要在动作空间中最大化价值进行学习。

这一部分课堂节奏较快，一些重要的推导在 PPT 中，不再赘述。

9 个人思考

学习了何老师的强化学习课程，动手实操是少不了的，接下来我将尝试使用 python 语言，调用 gym 库实现几个强化学习小 demo 的演示。

9.1 gym-CartPole 火柴棒倒立摆

火柴棒倒立摆的智能体、状态、动作分析见表??，可视化的结果图见3, 算法过程见下面的伪代码表。

9.2 MountainCarContinuous 爬山小车

见图4。小车作为一个智能体，可以选择向前进或者向后退，并且拥有运动惯性，因此动作空间大小是 2，t 时刻小车所处位置即状态，最终目的是到达右上角的小旗子。据我的观察，小车车一开始的轻微幅度的前后移动，后面开始逐渐增大移动幅度，在小坡道来回移动，这个连续性过程可以看作一种“摆动”，直到最后冲过坡找到旗子，否则就掉进沟里重新开始。

概念	说明
智能体	火柴棒倒立摆算法
环境	倒立的火柴棒
动作	左移动、右移动
状态	x, x$_dot$, $theta$, $theta_dot$
动作空间大小	2
奖励	固定值 1
学习目标	在操作 T 移动棒后获得尽可能高的累积奖励并保持稳定

表 3: 火柴棒倒立摆强化学习模型

Algorithm 1 CartPole 算法

Require: import gym 生成环境 env = gym.make('CartPole-v1',render$_mode$ =′ $human$′)

环境初始化 state = env.reset()

循环交互 while True:

选择随机动作 action = env.action$_space.sample$()

环境交互 state,reward,terminated,truncated,info = env.step(action)

打印 print(" 当前状态 state = ; 奖励 reward = ".format(state,reward))

判断当前 episode 是否完成

if terminated:

print("done")

break

time.sleep(1)

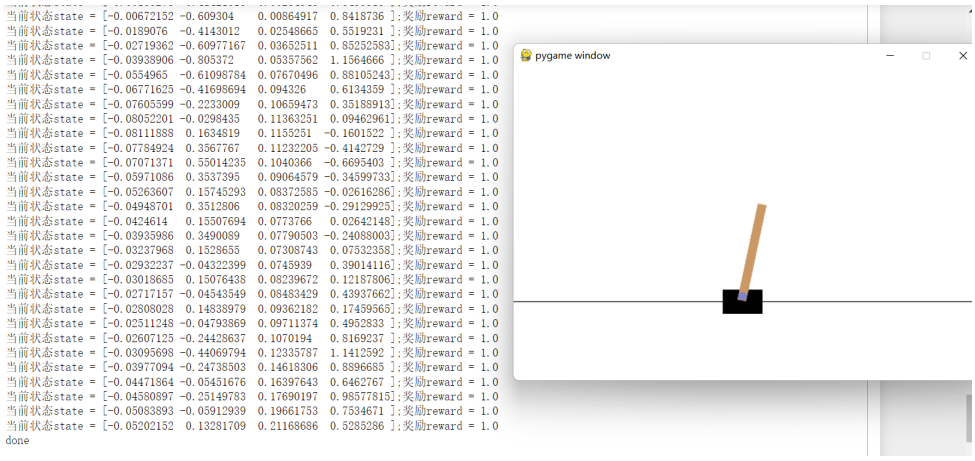

图 3: 火柴摆

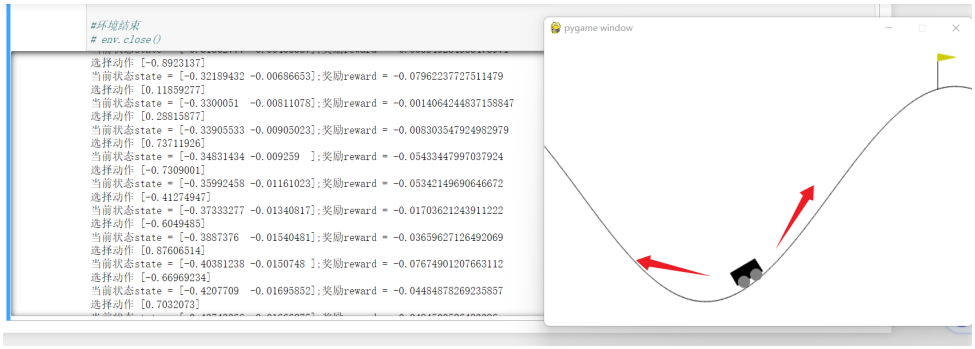
#环境结束
env.close()
选择动作 [-0.8923137]
当前状态state = [-0.32189432 -0.00686653];奖励reward = -0.07962237727511479
选择动作 [0.11859277]
当前状态state = [-0.3300051 -0.00811078];奖励reward = -0.0014064244837158847
选择动作 [0.28815877]
当前状态state = [-0.33905533 -0.00905023];奖励reward = -0.008303547924982979
选择动作 [0.73711926]
当前状态state = [-0.34831434 -0.009259];奖励reward = -0.05433447997037924
选择动作 [-0.7209001]
当前状态state = [-0.35992458 -0.01161023];奖励reward = -0.05342149690646672
选择动作 [-0.41274947]
当前状态state = [-0.37333277 -0.01340817];奖励reward = -0.01703621243911222
选择动作 [-0.6049485]
当前状态state = [-0.3887376 -0.01540481];奖励reward = -0.03659627126492069
选择动作 [0.87606514]
当前状态state = [-0.40381238 -0.0150748];奖励reward = -0.07674901207663112
选择动作 [-0.66969234]
当前状态state = [-0.4207709 -0.01695852];奖励reward = -0.04484878269235857
选择动作 [0.7032073]
pygame window

图 4: 小车爬山