师徒博弈

专业名称: 人工智能与自动化学院
学　　号: U201914634 U201914626
学生姓名: 彭杨哲 贾田旺
指导教师: 罗云峰
完成时间: 2021 年 6 月 24 日

第 1 章 单次师徒博弈

随着知识经济的到来，信息与技术成为时代特征，智力资源正逐步取代劳动力和资本成为创造财富的主要资产。个体拥有知识与技术，才有创造财富的能力，但一个人的力量终究是有限的，故产生了知识的共享与传承。而师徒模式是知识共享和传承的最常见的模式。

在师徒传承中的绝大部分时刻，师父的知识程度都是大于徒弟的；徒弟处于学习的过程中，知识程度是不断增长的。在这个过程中，师父的心理变化是复杂而矛盾的：一方面徒弟日常照顾师父的生活，并且可以替师父完成档次低的任务，使得师父得到了解脱，能够着眼于水平更高的任务，提高了师父的工作效率，并且徒弟的水平增长可以为师父带来成就感，可以提高师父在圈内的名誉，进而为师父带来更好的生源；但另一方面，随着徒弟的水平越来越高，师父也会担心徒弟有一天会超越自己，从而取代自己，抢走自己的饭碗。

为了讨论以上现象，本文将以师徒博弈为研究对象，构建师徒博弈模型[1]，通过探寻二者在参与博弈过程中的行为动机，以及在不同激励政策下的不同表现，分析最大化收益的对策。

1.1 师徒博弈的基本假设

这是一个很有意思的问题，为了讨论这个问题，不妨进行如下假设：

1. 在这个特定问题中，为了简化模型，知识水平用掌握的技能来表征。

2. 知识水平是可以变化的。我们假设师父的知识水平是接近饱和的，师父如果继续坚持学习，知识水平依旧会得到提升，但是提升不大，可近似将其看作接近于饱和并稳定不变。在初始时刻，徒弟几乎没有任何知识，故设其知识水平为 0，即在 $t = 0$ 时，$K_A = 0$。通过学习，徒弟的知识水平可以快速不断增长，随着时间增长，徒弟终究会达到与师父同等水平，并在自己探索创新下有超越师父的可能。

结合以上假设，令 $k = \frac{K_A}{K_M}$, 可以得到以下内容。

以师父的视角而言：

1. $t = 0, K_A = 0$, 初始时刻，徒弟没有任何技能。

2. $0 < t < t_1, 0 < K_A << K_M, 0 < k << 1$, 在初学的一段时间内，我们称之为第一阶段，师父先试验地教导一些内容，看徒弟有没有灵性与前途。徒弟在这段时间学习的知识远不能和师父比较。

3. $t = t_1$，师父初步评判徒弟是否是可造之材，设师父设立的门槛水平为 K_G 若 $K_A \geq K_G$, 则师父会继续培养这个徒弟；否则，师父会将其放弃。

[1]本文所有的代码及仿真结果, 模型训练结果已开源在 https://github.com/PoloWitty/game_theory_master

4. 设 p_1 为师父的忍耐力水平，$t_1 < t < t_2, 0 < K_A < p_1 * K_M, 0 < k < p_1$，在这段时间中（我们称之为师父视角的第二阶段），师父耐心地教导徒弟，徒弟帮助师父做日常的简单活，知识水平增长迅速（但没有达到师父的忍耐水平），师父也能够从日常的琐事中解放出来，做有成就感的事情，二者互相促进，互相提高。

5. $t_2 < t < t_3, p_1 * K_M < K_A < K_M, p_1 < k < 1$，在这段时间中（我们称之为师父视角的第三阶段），师父发现徒弟的知识技能水平上升的太快，害怕有一天徒弟与自己的水平相当之后会取代自己。从 t_2 开始，师父发觉了危机，所以之后的教导师父觉得应该留一手，教导得到的收益减少，故会有些技能故意不教徒弟。

6. $t = t_3, K_A = K_M, k = 1$, 徒弟终究在进步，到了 t_3 时刻，徒弟经过自己的勤奋努力，终于在技术上的造诣完全可以和师父并驾齐驱，取代已显疲态的师父易于反掌。

以徒弟的视角而言：

1. 首先与师父视角一样，徒弟需要从知识水平为 0 开始，直到通过师父的考验。

2. 设 P_2 为徒弟的好学程度，$t_1 < t < t_2', 0 < K_A < p_2 * K_M, 0 < k < p_2$，在这段时间中（我们称之为徒弟视角的第二阶段），徒弟认真地跟随师父学习，知识水平增长迅速，得到收益，获得成就感。

3. $t_2 < t < t_3, p_2 * K_M < K_A < K_M, p_2 < k < 1$，在这段时间中（我们称之为徒弟视角的第三阶段），徒弟发现师父教的技能太过陈旧，并且被外界的诱惑所吸引，希望自己出去单干，脱离师父的束缚，所以学习的激情下降，学习获得的收益减少。

4. $t = t_3, K_A = K_M, k = 1$，通过学习，徒弟的水平与师父相当，选择脱离师父的束缚。

首先，先讨论最普遍的师徒关系状况（即师徒双方均处于第二或第三阶段，$t_1 < t < t_3, 0 < K_A < K_M, 0 < k < 1$）中存在的单次博弈情形。

1.2 博弈模型

1.2.1 模型建立

设：师父教的成本为 U，收益为 R；徒弟学的成本为 V，学习的收益为 Q。师父选择教的概率为 x，则不教的概率为 $1-x$；徒弟学的概率为 y，则不学的概率为 $1-y$。则有关博弈矩阵如下：

表 1-1 师徒博弈模型

师父 \ 徒弟	学	不学
教	$R-U, Q-V$	$-U, 0$
不教	$0, -V$	$0, 0$

1.2.2 得失分析

设师父教的期望收益为 E_T, 不教的期望收益为 E_{NT}, 平均期望收益为 E_M， , 徒弟学的期望收益为 E_L, 不学的期望收益为 E_{NL}, 平均期望收益为 E_A 则有:

$$\begin{cases} E_T = y(R-U) + (1-y)(-U) = Ry - U \\ E_{NT} = 0y + 0(1-y) = 0 \\ E_M = xE_T + (1-x)E_{NT} = xyR - xU = x(Ry-U) \end{cases} \tag{1-1}$$

$$\begin{cases} E_L = x(Q-V) + (1-x)(-V) = xQ - V \\ E_{NL} = 0x + 0(1-x) = 0 \\ E_A = yE_L + (1-y)E_{NL} = y(xQ-V) \end{cases} \tag{1-2}$$

对于师傅来说，由 $E_T = E_{NT}$ 可得 $y = \frac{U}{R}$, 此时, 师傅采取教或者不教的策略所获得的的收益是相同的; 对于徒弟来说, 有 $E_L = E_{NL}$ 可得 $x = \frac{V}{Q}$, 此时, 徒弟选择学或者不学的策略所获得的的收益是相同的.

所以, 可以看出, 此博弈的混合策略纳什均衡为:

- 师父 σ_M（教，不教）=（$\frac{U}{R}, 1-\frac{U}{R}$）
- 徒弟 σ_A（学，不学）=（$\frac{V}{Q}, 1-\frac{V}{Q}$）

实际上，此单次博弈也存在着纯战略纳什均衡：即（教，学），（不教，不学）。

结合原先的假设来看，在师徒均处于第二阶段时，出于完全理性假设，师徒一定会选择收益最大的策略组合，即（教，学）的策略组合；在有一方处于第三阶段时，该方的净收益会下降，如果此单次博弈进行时，收益下降到 0 以下，则双方都会偏离（教，学）而共同选择（不教，不学）。

第 2 章　多阶段师徒博弈

在第1章中, 我们讨论了对于单次师徒博弈的情形, 而当我们考虑实际情况时, 可以想见, 师傅与徒弟之间不会仅仅发生一次博弈, 而是会根据师傅所掌握的技能发生很多次的博弈过程, 在这一部分中, 我们将会对这种具有多个阶段的师徒博弈进行讨论，并编写了相关的代码进行仿真分析，其目的主要在于得到在这种博弈情境下师傅应当采取的策略。

为了简化模型，我们假设徒弟已经度过了师父设立的“门槛”。

2.1　模型建立

假设师父共有 N 种技能，每种技能的教导和学习构成一次博弈，则达成了 N 次有限重复博弈的情况。

因为我们的目的主要是讨论师傅应当采取的策略，所以对于徒弟，我们将其设置为一种固定的模型，其可以根据师傅所作出的决策自动的进行选择学还是不学. 而相应的收益 $Q-V, R-U$ 也会随着徒弟所掌握的技能数 t 发生变化。

2.1.1　徒弟模型建立

对于徒弟而言, 我们假设其面对每一次的博弈时, 都具有三种选择: 学, 不学, 背叛师傅。其中学的收益会随着其学到的技能点数 t 的增长而下降, 而不学的话收益必为 0, 而如果徒弟选择背叛的话, 其会获得 10 点的收益, 师傅会获得-10 点的收益, 即当徒弟背叛师傅, 另立门户后徒弟可以获得更大的收益, 而师傅则会因为失去徒弟, 失去了一个得力的助手, 也增加了自己的技能失传的概率, 还面临着被抢饭碗的风险，所以收益为负值。

而考虑实际情况, 我们可以猜想出徒弟的收益 $Q-V$ 所对应的函数应该具有的图像特征:

- 随徒弟学到的技能点数 t 的增加而总体上呈现出下降的趋势
- 应当具有某种转折点, 在该点之前, 徒弟学的收益相对较大, 而在该点之后徒弟学的收益就相对较小，这个转折点就是第1章中假设的徒弟的好学程度
- 可以体现出徒弟对于师傅能力的感知以及对于自身是否应当背叛, 是否应当另立门户的程度

综合考虑以上几点, 我们选择了 sigmoid 函数作为原型, 进行伸缩平移变换然后得到我们的收益函数为

$$Q-V=-6[\frac{1}{1+e^{-t+p_2*N}}-0.5]+3$$

其对应的函数图像如图2-1所示。

其中 p_2 即为徒弟的好学程度, 其值越大, 就表明徒弟认为自己跟着师傅进行学习所获得的收益随自己的技能点数的提升下降程度并不大, 相应的横轴上的 p_2*N 点对应了

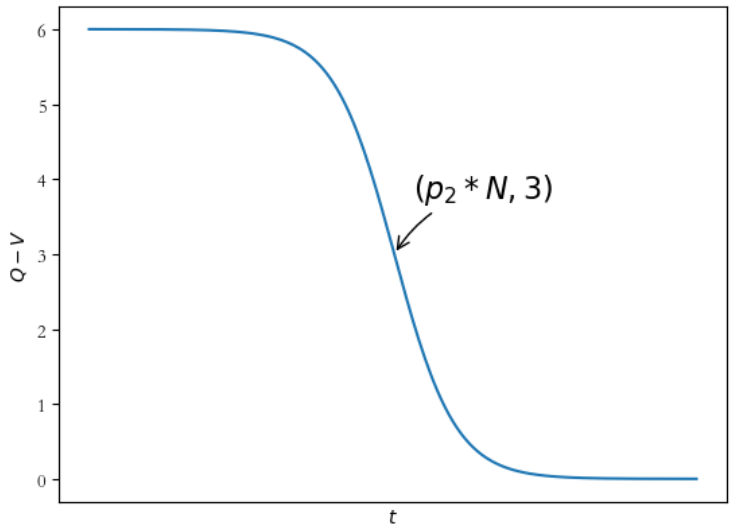

图 2-1 Q-V 的收益函数图像

徒弟选择学所获得的收益的转折点。

而对于背叛, 我们考虑将其设置为基于概率进行选择, 因为在实际中, 对于一个徒弟而言, 其既可能在刚学一点技能时就背叛师傅, 也有可能跟着师傅学了很久却并没有背叛, 所以不能将其直接设置为过了某个阈值就一定会背叛的形式, 所以我们就需要一个徒弟背叛的概率函数, 其应该满足一下几点图像特征:

- 与徒弟的收益 $Q-V$ 存在着一定的相关关系
- 纵坐标范围为零到一之间
- 在徒弟刚开始学习的一段时间内, 其函数值几乎为零, 而在徒弟所学的技能点数较多时, 背叛概率逐渐变大

综合考虑以上几点, 我们发现 sigmoid 函数所具有的特性满足我们的预期需要, 且其与徒弟的学习收益 $Q-V$ 具有相当好的相关关系, 为收益函数沿中心对称点翻转之后再归一化得到, 函数表达式为

$$P(\text{背叛}) = \frac{1}{1+e^{-t+p_2*N}}$$

其中 t 为徒弟所学到的技能点数, p_2 与 N 的含义与徒弟的收益函数中相应变量的含义相同，对应的函数图形如图2-2。

在此处, 我们对于 p_2 的含义可以有更深一层的理解, p_2 也衡量了徒弟背叛概率为 0.5 时徒弟所学到的技能点数与师傅所具有的总技能点数的比值, 其值越大就表明徒弟越好学, 越谦虚, 越想学到师傅的全部技能。

2.1.2 师傅的简单博弈策略

对于师父而言，我们则假设其面对每一次博弈时有两种选择，教与不教。其中教的收益会随着徒弟学到的技能点数而进行相应改变，而不教的收益为 0。此外徒弟若选择

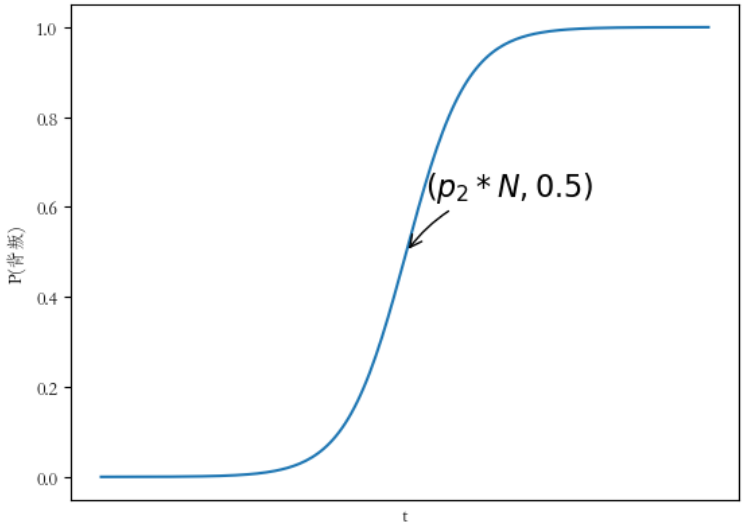

图 2-2 背叛概率

背叛，师父还会受到 10 点惩罚。

我们经过调研，结合实际情况，可以猜想出师父的收益 $R-U$ 所对应的函数应该具有的图像特征：

- 结合之前的假设，师父的收益随着徒弟学到的技能点数增长总体上应该呈现出下降的趋势
- 应当具有某种转折点，在该点之前，师父的收益相对稳定，保持在一个较高值；而该点之后，师父的收益应该迅速下降，这个转折点就是第1章假设中的师父的忍耐力程度 p_1。
- 师父的收益会下降到 0 以下，以致师父会选择不教的策略

综合考虑以上几点，结合我们进行的数据分析，我们选择了以下分段函数作为师父的收益曲线：

$$R-U=\begin{cases}8 & t \le p_1 K_A \\ -(t-p_1 K_A)+8 & t > p_1 K_A\end{cases}$$

其对应的函数图像如图2-3所示。

其中 p_1 为师父的忍耐力程度，对应图像中的转折点，其值越大，代表师父对徒弟的忍耐力越高，越相信徒弟不会背叛自己，进而愿意教给他更多技能。可以得出 $t=8+p_1 K_A$ 时师父教导的收益下降到 0，并还会继续下降，从此时开始，师父为了自己的收益最大化，每一次博弈均会选择不教，进而徒弟会选择不学，所以每一次博弈的纯策略纳什均衡都只剩下了一个，即（不教，不学）。

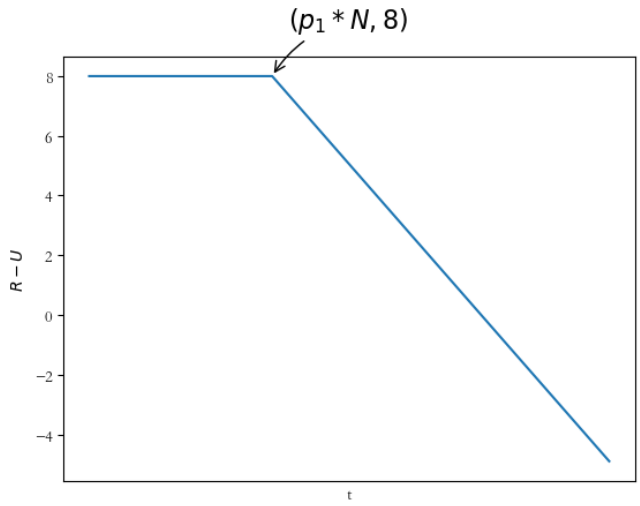

图 2-3 R-U 收益对应函数图像

2.1.3 模拟仿真

结合以上建立的师徒双方的模型进行仿真分析。我们得到了如图2-4输出结果。

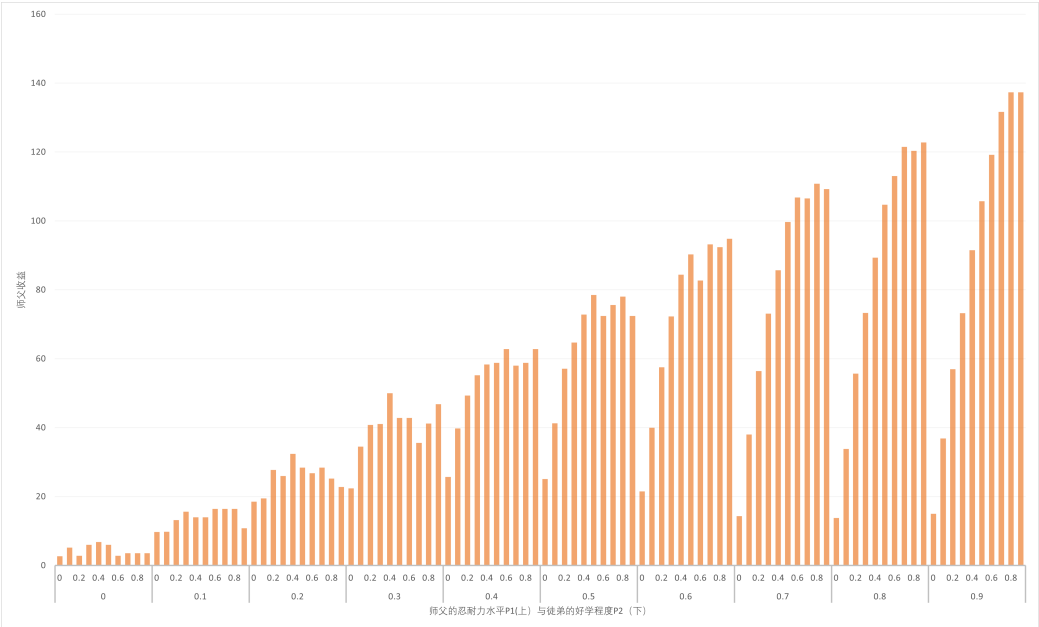

图 2-4 模拟仿真分析结果

按徒弟好学程度 $p_2 = 0 - 0.9$ 进行分组，每组中进行师父的不同策略模拟，即师父忍耐力水平 $p_1 = 0 - 0.9$，纵坐标为师傅的收益。

从图中可以看出，师父的收益总体呈现上升趋势，故徒弟的好学程度越高，师父获得的收益越高；徒弟拥有不同的好学程度时，师父获得最多收益的策略不同，徒弟的好学程度极低时（$0 \leq p_2 < 0.2$），师父的收益较低且相对稳定，采取不同策略对收益的影响不大；徒弟的好学程度处于中间水平时（$0.2 \leq p_2 < 0.6$），师父的收益较高，且师父采

取策略为忍耐力水平 p_1 处于中间水平且与徒弟好学程度相近时，可获得最大收益；徒弟的好学程度较大时（$0.6 \leq p_2 < 1$），师父的收益最高，且师父的忍耐力水平 p_1 相应较高时可获得最大收益。

综上，师父的收益大致上与徒弟的好学程度呈正相关趋势。且对待不同的徒弟应采取不同的策略，徒弟的好学程度较低时，代表徒弟是一个目光短浅的人，只想着赶快学成，背叛师傅外出单干，师父应该降低忍耐力水平，降低自己的损失，以获得相对较大收益；徒弟的好学程度较高时，代表徒弟是一个目光长远的人，乐于钻研知识，并将它发扬光大，不会轻易背叛师父，师父应该提高自己的忍耐力水平，教导徒弟更多知识，期望其传承自己的技能，并帮助自己完成更多任务，以获得最大收益。

第 3 章 利用强化学习对多阶段师徒博弈中师傅应该采取的策略进行求解

在第2章中, 我们利用了一种较简单的策略作为师傅采取的策略进行了模拟, 即当师傅所获得的收益一旦变为负值时, 师傅就一直采取不再教徒弟的策略. 然而, 实际情况时错综复杂的, 首先, 人并不一定只会看着眼前的收益, 人可以是很有远见的, 不会因为眼前收益受损就不再继续走下去; 其次, 师傅在前期也不一定要一直采取教的策略, 而是可以选择教几次, 不教几次的策略, 我们简单地使其采取一直教下去的策略是并不完全合理的。

而反观第2章的仿真模拟结果, 我们可以看出当徒弟好学程度较低, 而师傅的忍耐力水平又较高时, 二者存在着一定程度的错位, 此时师傅采取的策略便会对最终的结果产生较大的影响, 所以我们主要讨论这部分的内容. 所以, 在本部分中, 我们采取强化学习中 Q-Learning 学习的策略对 $p_1 = 0.9, p_2 = 0.3$ 的博弈情形进行结果求解。

3.1 Q-Learning 概述

下面, 我们先简单的对于强化学习中的 Q-Learning 学习策略进行一下概述。

3.1.1 Q-Learning 决策

假设我们处于状态 s1, 当前有两个行为 a1,a2 可供选择, 如果根据经验, 我们可以知道 a1 带来的潜在奖励要比 a2 能够带来的潜在奖励大, 这里的潜在奖励我们可以用一个有关 s 和 a 的 Q 表格代替, 在我们的记忆 Q 表格中,Q(s1, a1)=-2 要小于 Q(s1, a2)=1, 所以我们判断要选择 a2 作为下一个行为. 现在我们的状态更新成 s2 , 我们还是有两个同样的选择, 重复上面的过程, 在行为准则 Q 表中寻找 Q(s2, a1) Q(s2, a2) 的值, 并比较他们的大小, 选取较大的一个. 接着根据 a2 我们到达 s3 并在此重复上面的决策过程. Q learning 的方法也就是这样决策的。

3.1.2 Q-Learning 更新

根据 Q 表的估计, 因为在 s1 中, a2 的值比较大, 通过之前的决策方法, 我们在 s1 采取了 a2, 并到达 s2, 这时我们开始更新用于决策的 Q 表, 接着我们并没有在实际中采取任何行为, 而是再想象自己在 s2 上采取了每种行为, 分别看看两种行为哪一个的 Q 值大, 比如说 Q(s2, a2) 的值比 Q(s2, a1) 的大, 所以我们把大的 Q(s2, a2) 乘上一个衰减值 gamma (比如是 0.9) 并加上到达 s2 时所获取的奖励 R , 因为会获取实实在在的奖励 R , 我们将这个作为我现实中 Q(s1, a2) 的值, 但是我们之前是根据 Q 表估计 Q(s1, a2) 的值. 所以有了现实和估计值, 我们就能更新 Q(s1, a2) , 根据估计与现实的差距, 将这个差距乘以一个学

习效率 alpha 累加上老的 Q(s1, a2) 的值变成新的值. 但时刻记住, 我们虽然用 maxQ(s2) 估算了一下 s2 状态, 但还没有在 s2 做出任何的行为, s2 的行为决策要等到更新完了以后再重新另外做。

3.1.3 Q-Learning 伪代码

Algorithm 1 Q-Learning 算法

Input: 步长 $\alpha \in (0, 1]$, 很小的$\epsilon, \epsilon > 0$

Output: 更新后的" 状态-动作" 二元组

对所有的 $s \in S^+, a \in A(s)$, 任意初始化 $Q(s, a)$, 其中 $Q($终止状态$, \cdot) = 0$
for 每一幕 **do**
    初始化 S
    repeat
        使用从 Q 得到的策略 (例如 ϵ-贪心), 在 S 处选择 A
        执行 A, 观察到 R,S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_\alpha Q(S', \alpha) - Q(S, A)]$
        $S \leftarrow S'$
    until S 为终止状态
end for

3.2 应用 Q-Learning 进行求解

在我们的博弈情景下, 我们可以将 (徒弟学到的技能点数, 博弈的轮数) 二元组作为状态 S, 而师傅可以采取的行动 A 即为教还是不教两种, 而相应的收益按照 Q-Learning 的学习策略进行更新。

在代码中, 我们设置的 $\alpha = 0.1, \gamma = 0.6, \epsilon = 0.9$, 让计算机训练了 2000 轮, 每轮的收益与上一章中的模拟设置保持相同。

训练完成后, 我们使用训练得到的 Q 表进行了一次仿真, 观察经过强化学习的师傅所选择的策略是怎么样的, 并将其获得的总收益与上一章中的采取简单策略的师傅所获得的总收益进行对比, 可得到其中一次的实验结果如下:

```
第 1 轮 RL选择 1 此轮收益为 8
第 2 轮 RL选择 1 此轮收益为 8
第 3 轮 RL选择 1 此轮收益为 8
第 4 轮 RL选择 1 此轮收益为 8
第 5 轮 RL选择 0 此轮收益为 0
第 6 轮 RL选择 1 此轮收益为 8
第 7 轮 RL选择 1 此轮收益为 8
第 8 轮 RL选择 1 此轮收益为 8
第 9 轮 RL选择 1 此轮收益为 8
```

```
第 10 轮 RL选择 1 此轮收益为 -10
强化学习结果总收益为 62.0
简单策略结果总收益为 60.0
```

其中 RL 代表经过强化学习的师傅, 1/0 表示师傅所选择的策略为教 (1) 或不教 (0)。

可以看出, 经过强化学习的师傅并不会在每一次都采取教徒弟, 而是有时会选择不教徒弟, 而从最终的总收益来看, 使用强化学习的策略的师傅所获得的总收益略大于使用简单策略所能获得的总收益, 而如果采取简单策略的话, 师傅在收益降为负值之前一定是一直教徒弟的, 所以不可能出现中间突然不教徒弟的情况, 也对应了对于不好学的徒弟, 一直教徒弟很可能并不是最好的策略。

而从二种策略的最终结果较为相近以及经过强化学习得到的结果与我们模型设计较为相近可以看出, 我们的模型具有较为良好的可解释性, 具有一定的实际价值。

第 4 章 总结

综上，我们进行了单次师徒博弈，多阶段师徒博弈，仿真分析与强化学习分析，现进行总结。

总体来说，对于“好学”的徒弟，师父应该选择多教的策略，即若 p_2 值较大，师父应该使得自己的忍耐力水平 p_1 值较大以获得更多收益；而对于急于脱离师父，想出去单干的徒弟，师父应该选择少教的策略，即若 p_2 值较小，师父应该降低自己的忍耐力水平 p_1 以获得更多收益。

师父应该时刻关注徒弟的状态，并根据徒弟的不同性格更改自己的策略，以使得自己获得最大收益。

我们可以借用这个模型分析跨国公司与当地企业之间的技术交流过程，准确的讲是技术水平高的跨国公司（师父）将技术溢出到东道国当地企业（徒弟）的过程，即分析跨国公司对当地企业的技术溢出效应。并得到结论：

- 东道国当地企业要想获得跨国公司的技术溢出必须要在技术上达到“门槛水平”。
- 跨国公司在一定程度上愿意将技术溢出到当地企业，实现垂直分工，从而降低成本。
- 跨国公司对当地企业的技术溢出要有个限度，一旦感觉到潜在威胁或预感到当地企业具备可能的竞争能力时，就停止技术溢出。
- 跨国公司面对不同倾向的当地企业时需采取不同的策略。如当地企业倾向于仅仅学习技术而不与跨国公司竞争，跨国公司应多溢出些技术并让其承办更多低收益工作，从而降低自己的成本，提高自己的收益；而如当地企业倾向于与跨国公司竞争抢占当地市场，则跨国公司应较早停止技术溢出，防止当地企业抢夺自己的利益。
- 当地企业只要有学习成长的能力，最终能够达到与跨国公司相抗衡的技术水平。
- 当地企业仅靠技术溢出是不够的，要想超越跨国公司，必须培养自己技术创新的能力。

附录 A 源代码

A.1 仿真及强化学习环境搭建

```
import numpy as np
import pandas as pd
import time
import math
import sys
import matplotlib.pyplot as plt
from parameter import INTERACTIVE


G_APPRENTICE_BETRAY=10
G_MASTER_BETRAY=-10
TOTAL_SKILLS=20


class Env():
    def __init__(self):
        self.result =np.zeros(2)#第一个为师傅的累积收益,第二个为徒弟的累积收益
        self.record=pd.DataFrame(np.zeros((TOTAL_SKILLS,2)))
        if INTERACTIVE:
            self.fig, self.ax=plt.subplots()
            plt.ion()

    def G_apprentice(self,t, apprentice_action,P_2):
        y=0
        # y=3
        y=-6*(1/(1+math.exp(-t+P_2*TOTAL_SKILLS))-0.5)+6/2#6为可变参数
        if apprentice_action==0:
            y=0
        return y

    def G_master(self,t, master_action,P_1):
        y=0
        if t<=P_1*TOTAL_SKILLS:
            y=8
        else:
            y=-t+P_1*TOTAL_SKILLS+8
        if master_action==0:
```

```
            y=0
        return y

    def _betray_possibility (self,t,P_2):
        y=0
        # if t<TOTAL_SKILLS:
        #     # y=0.8*(math.exp(t)-1)/(math.exp(TOTAL_SKILLS)-1)#0.8为可调超参
        #     y=0.5*t/TOTAL_SKILLS
        y=1/(1+math.exp(-t+P_2*TOTAL_SKILLS))
        return y

    def if_betray(self,t,P_2):
        p_=self._betray_possibility(t,P_2)
        # print('背叛概率为',p_)
        return np.random.choice([1,0], p=[p_,1-p_])

    def reset(self):
        self.result=np.zeros(2)
        self.record=pd.DataFrame(np.zeros_like(self.record))
        if INTERACTIVE:
            self.fig, self.ax=plt.subplots()

    def step(self, master_action, t, state, P_1,P_2):
        #计算徒弟的行为,是学还是不学,还是背叛
        apprentice_action_=1
        if master_action==1:#教
            apprentice_action_=1#学
        elif master_action==0:#不教
            apprentice_action_=0#不学
        betray=self.if_betray(t,P_2)
        apprentice_action=apprentice_action_*(1-betray)

        done=0#游戏是否结束
        if betray==1 or state==TOTAL_SKILLS:
            done=1
        else:
            done=0

        #结算收益
        g_m=self.G_master(t, master_action, P_1)
        self.result[0]+=g_m
        if not betray:
            g_a=self.G_apprentice(t, apprentice_action, P_2)
            self.result[1]+=g_a
```

```
        if betray:
            g_m=G_MASTER_BETRAY
            self.result[0]+=g_m
            g_a=G_APPRENTICE_BETRAY
            self.result[1]+=g_a

        state_result=[]
        state_result.append(g_m)
        state_result.append(g_a)

        self.record.iloc[state,0]=self.result[0]
        self.record.iloc[state,1]=self.result[1]

        return betray, apprentice_action, self.record, state_result,done

    def render(self):
        if INTERACTIVE:
            self.ax.cla()
            self.record.plot(kind='bar',ax=self.ax)
            plt.pause(0.2)
        else:
            pass
```

A.2 人机交互与第二部分仿真

```
from env import Env, TOTAL_SKILLS
import time
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from RL_brain import simple_strategy
from parameter import INTERACTIVE

P_2_TEST=10
TOTAL_EPISODE=10
BATCH_SIZE=10

def update():
    log=pd.DataFrame(np.zeros((P_2_TEST*TOTAL_EPISODE,3)),columns=['p_2','p_1',"师傅的总
        收益"])
    # if not INTERACTIVE:
    P_1=0
```

```
P_2=0
for i in range(P_2_TEST):#测试p_2
    P_1=0
    for episode in range(TOTAL_EPISODE):
        if INTERACTIVE:
            print('episode: ',episode)
            print('P_1: ',P_1)
        # initial observation
        sum_gain=0
        for batch in range(BATCH_SIZE):
            if INTERACTIVE:
                env.render()
            state =0
            t=0
            state_result =[0,0]
            record = pd.DataFrame(np.zeros((TOTAL_SKILLS, 2)))
            while state <TOTAL_SKILLS:

                if INTERACTIVE:
                    master_action=int(input('此轮你的决策是:'))#人机交互版本
                else:
                    master_action=simple_strategy( state_result [0])

                apprentice_action =0
                betray=0
                betray, apprentice_action ,record, state_result ,done=env.step(master_action,
                    t, state ,P_1,P_2)

                if apprentice_action ==1:
                    t+=1
                if INTERACTIVE:
                    if betray==1:
                        print('徒弟此轮背叛了你')
                    else:
                        print("徒弟此轮选择了",apprentice_action)
                    print('此轮你的收益为', state_result [0], '徒弟的收益为', state_result
                        [1])
                state +=1
                if INTERACTIVE:
                    print('徒弟学到的技能数为:',t,' 当前轮数:', state )
                    print('\n')

                if INTERACTIVE:
                    env.render()
```

```
                    # break while loop when end of this episode
                    if done:
                        break

                # end of game
                # print ('game over')
                sum_gain+=record.iloc[state-1,0]
                env.reset()
            log.iloc[i*P_2_TEST+episode,0]=P_2
            log.iloc[i*P_2_TEST+episode,1]=P_1
            log.iloc[i*P_2_TEST+episode,2]=sum_gain/BATCH_SIZE
            P_1+=1/TOTAL_EPISODE
        P_2+=1/P_2_TEST
    log.to_csv('log.csv')


if __name__=="__main__":
    env=Env()
    update()
```

A.3 强化学习部分

```
from RL_brain import QLearningTable
from env import Env, TOTAL_SKILLS
from tqdm import tqdm
import time
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from RL_brain import simple_strategy
from parameter import INTERACTIVE

P_1=0.9
P_2=0.3


def update():
    for episode in tqdm(range(2000)):
        # initial observation
        t=0;state=0;betray=0
```

```
        observation =(t, state )#状态是(徒弟学会的技能数t,当前进行的轮数state,徒弟是否背叛)

        while state <TOTAL_SKILLS:

            # RL choose action based on observation
            action = RL.choose_action( str ( observation ))

            # RL take action and get next observation and reward
            betray , apprentice_action , record , state_result ,done=env.step( action , t , state ,P_1,
                P_2)

            if apprentice_action :
                t+=1
            state +=1
            observation_ =(t, state )
            if state ==TOTAL_SKILLS-1:
                observation_ ='done'
            # RL learn from this transition
            RL.learn( str ( observation ), action , state_result [0], str ( observation_))

            # swap observation
            observation = observation_

            # break while loop when end of this episode
            if done:
                break

    # end of game
    print ('game over')

def sample():
    env. reset ()
    #read the learning result
    RL.q_table=pd.read_csv(' q_table .csv' , index_col=0)
    ret_result =0
    # initial observation
    t = 0
    state = 0
    betray = 0
    observation = (t, state ) # 观测到的是(徒弟学会的技能数t,当前进行的轮数state,徒弟是否
        背叛)

    while state < TOTAL_SKILLS:
```

```
        # RL choose action based on observation
        action = RL.choose_action(str(observation))

        # RL take action and get next observation and reward
        betray, apprentice_action, record, state_result, done = env.step(
            action, t, state, P_1, P_2)

        if apprentice_action:
            t += 1
        state += 1
        observation_ = (t, state)
        if state == TOTAL_SKILLS-1:
            observation_ = 'done'
        print('第', state, '轮','RL选择',action,'此轮收益为', state_result[0])

        # swap observation
        observation = observation_

        # break while loop when end of this episode
        if done:
            ret_result =record.iloc[state-1,0]
            break
    return ret_result

def sample_compare():
    env.reset()
    state = 0
    t=0
    state_result =[0,0]
    ret_result =0
    while state <TOTAL_SKILLS:
        master_action=simple_strategy(state_result[0])

        apprentice_action =0
        betray=0
        betray, apprentice_action,record, state_result,done=env.step(master_action,t, state,P_1,
            P_2)

        if apprentice_action ==1:
            t+=1
        state +=1

        # break while loop when end of this episode
        if done:
```

```
                ret_result =record.iloc[state-1,0]
                break
    return ret_result



if __name__ == "__main__":
    env = Env()
    RL = QLearningTable(actions=[1,0])
    update()
    RL.q_table.to_csv('q_table.csv')
    rl_res =sample()
    compare_res=sample_compare()
    print('强化学习结果总收益为',rl_res)
    print('简单策略结果总收益为',compare_res)
```

A.4 决策学习模块

```
import numpy as np
import pandas as pd


def simple_strategy(g_m):
    action=1
    if g_m<0:
        action=0
    return action

class QLearningTable:
    def __init__(self, actions, learning_rate=0.1, reward_decay=0.6, e_greedy=0.9):
        self.actions = actions  # a list
        self.lr = learning_rate
        self.gamma = reward_decay
        self.epsilon = e_greedy
        self.q_table = pd.DataFrame(columns=self.actions, dtype=np.float64)

    def choose_action(self, observation):
        self.check_state_exist(observation)
        # action selection
        if np.random.uniform() < self.epsilon:
            # choose best action
            state_action = self.q_table.loc[observation, :]
```

```
            # some actions may have the same value, randomly choose on in these actions
            action = np.random.choice(state_action[state_action == np.max(state_action)].
                index)
        else:
            # choose random action
            action = np.random.choice(self.actions)
        return action

    def learn(self, s, a, r, s_):
        self.check_state_exist(s_)
        q_predict = self.q_table.loc[s, a]
        if s_ != 'done':
            q_target = r + self.gamma * self.q_table.loc[s_, :].max()  # next state is not
                terminal
        else:
            q_target = r  # next state is terminal
        self.q_table.loc[s, a] += self.lr * (q_target - q_predict)  # update

    def check_state_exist(self, state):
        if state not in self.q_table.index:
            # append new state to q table
            self.q_table = self.q_table.append(
                pd.Series(
                    [0]*len(self.actions),
                    index=self.q_table.columns,
                    name=state,
                )
            )
```