**基于搜索算法辅助强化学习的多轿电梯调度算法**

三菱电梯合作项目电梯调度算法相关技术报告

日期：11/22/2021

作者：严骏驰，赵泽林，袁健勇，董智辰，王玙璠

**目录**

[一．概要 4](#_Toc94189802)

[相关定义 5](#_Toc94189803)

[数据分析 5](#_Toc94189804)

[数据特征 6](#_Toc94189805)

[二．电梯调度技术传统算法 8](#_Toc94189806)

[先来先服务算法(FCFS) 8](#_Toc94189807)

[最短寻找楼层时间优先算法(SSTF) 8](#_Toc94189808)

[扫描算法(SCAN) 8](#_Toc94189809)

[最早截止期优先调度算法(EDF) 8](#_Toc94189810)

[SCAN-EDF 算法 8](#_Toc94189811)

[PI 算法 8](#_Toc94189812)

[基于专家系统的电梯群控调度算法 8](#_Toc94189813)

[基于模糊控制的电梯群控调度技术 9](#_Toc94189814)

[基于遗传算法的电梯群控调度技术 9](#_Toc94189815)

[三. 强化学习 11](#_Toc94189816)

[什么是强化学习 11](#_Toc94189817)

[强化学习中的基本概念 12](#_Toc94189818)

[本项目研究多轿电梯分派问题采用强化学习算法的原因 14](#_Toc94189819)

[四. 问题建模 15](#_Toc94189820)

[强化学习环境 15](#_Toc94189821)

[目标与奖励 15](#_Toc94189822)

[状态的选择与表示 15](#_Toc94189823)

[特征提取网络 16](#_Toc94189824)

[多层感知器（MLP） 16](#_Toc94189825)

[嵌入层（Embedding） 17](#_Toc94189826)

[图卷积神经网络 18](#_Toc94189827)

[图的定义 18](#_Toc94189828)

[图卷积的通式以及实现 18](#_Toc94189829)

[特征提取网络 18](#_Toc94189830)

[五. 强化学习模型训练和算法选择 19](#_Toc94189831)

[为什么要训练 19](#_Toc94189832)

[怎么训练 20](#_Toc94189833)

[值基础的方法 20](#_Toc94189834)

[策略基础的方法 22](#_Toc94189835)

[训练的指标 24](#_Toc94189836)

[六. 实验 25](#_Toc94189837)

[实验设置及超参数 25](#_Toc94189838)

[实验一：以整个上行高峰为训练数据 25](#_Toc94189839)

[以一天上行高峰为训练数据 25](#_Toc94189840)

[以31天上行高峰为训练数据 28](#_Toc94189841)

[实验分析 28](#_Toc94189842)

[实验二：以上行高峰中的一段时间为训练数据 29](#_Toc94189843)

[以31天中10：00到20：00时间段为训练数据。 29](#_Toc94189844)

[以31天中30：00到40：00时间段为训练数据 33](#_Toc94189845)

[以31天中30：00到45：00时间段为训练数据 37](#_Toc94189846)

[实验分析 41](#_Toc94189847)

[实验结论 41](#_Toc94189848)

[七. 强化学习结合搜索框架图 42](#_Toc94189849)

[八. 搜索算法 43](#_Toc94189850)

[蒙特卡洛树搜索 43](#_Toc94189851)

[遗传算法 43](#_Toc94189852)

[局部搜索算法 44](#_Toc94189853)

[损失计算 44](#_Toc94189854)

[电梯模拟运行 44](#_Toc94189855)

[实验公式计算 44](#_Toc94189856)

[贪心算法 44](#_Toc94189857)

[探索更优解 44](#_Toc94189858)

[结合历史数据赋予权重 45](#_Toc94189859)

[九. 实验 45](#_Toc94189860)

[实验数据——午餐高峰 45](#_Toc94189861)

[蒙特卡洛搜索 47](#_Toc94189862)

[遗传算法 48](#_Toc94189863)

[局部搜索 48](#_Toc94189864)

[十. 与优化结合的RL 49](#_Toc94189865)

[基于优化的方法 49](#_Toc94189866)

[结合优化方法的RL方法 49](#_Toc94189867)

[十一. 目前工作完成情况 51](#_Toc94189868)

[第一阶段：九月 51](#_Toc94189869)

[第二阶段：十月 51](#_Toc94189870)

[第三阶段：十一月 52](#_Toc94189871)

[第四阶段：十二月 53](#_Toc94189872)

[第五阶段：一月 53](#_Toc94189873)

[十二. 工作计划 55](#_Toc94189874)

# 

# 一．概要

多轿电梯调度问题是指在高层建筑中，有多部电梯用于服务乘客的召唤，由于工作作息时间多多关系，乘客在各层的出现时间具有一定的规律性，电梯调度系统需要设定一定的策略，利用乘客的召唤信息以及乘客的出现规律，选取最优的电梯来响应乘客的召唤请求，尽可能地减少乘客的等待时间与尽快地将乘客送往目的地，从而达到高效率的目的。本项目主要研究基于强化学习的多轿电梯调度算法。

## 相关定义

• 层站召唤（hall call）

在一栋16层4电梯的大楼中，每一层都会有乘客呼叫电梯，称为层站召唤（hall call）。层站召唤只确定上行或下行，电梯并不知道乘客的目的楼层。

• 轿内召唤（car call）

乘客进入电梯后，按下目的楼层，称为轿内召唤（car call）。

• 处理召唤

层站召唤发生后，系统会为该层站召唤分配一台电梯。电梯到达后，该层的乘客一个接一个进入电梯，直到全部上完或者电梯已达到核定载荷。

轿内召唤发生后，电梯会根据当前运行方向和被召唤情况继续运行。电梯的运行方向不会突然改变，直到当前方向的层站召唤和轿内召唤全部处理完。

• 评价标准

对于某种处理召唤的策略，以每位乘客的平均等待时间（从乘客到达，直到进入电梯经过的时间）和平均运输时间（乘客上梯后，从到达楼层运输到目的楼层经过的时间）作为评价标准。

### 数据分析

为了便于强化学习模型的训练，我们使用开源的百度电梯大赛中的电梯模拟器，并在其基础上对电梯的运行参数、召唤的处理逻辑等方面作了修改，使其更符合真实场景的设定。

对于人流数据，我们采用的是闭源电梯模拟器Elevate提供的各种模式下的模拟人流数据，它是csv格式的，主要包含数据如下：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 乘客到达时间 | 乘客到达楼层 |  | 乘客目的楼层 | 三菱算法为此乘客分配的电梯 | 乘客乘坐该电梯的上梯时间 | 乘客乘坐该电梯到达目的楼层时间 |
| 00 ：04.4 | 1 |  | 9 | 4 | 00 ：04.4 | 00 ：36.4 |

表1.1

包括每位乘客的出现时间、出现楼层、目的楼层、重量、三菱调度算法为它分配的电梯号、乘客的上梯时间和到达目的地的时间。由此也可以计算出该乘客的等待时间和运输时间。

为了方便在相同数据集上对三菱的算法以及我们的算法进行比较，我们使用这些人流数据，在开源的模拟器上模拟这些人流的生成：乘客出现的情况使用数据文件（csv格式）指定。每隔dt时间，模拟器检测当前时间是否 ≥ 数据文件中下一个乘客的出现时间。若当前时间达到该乘客的出现时间，则在模拟器中生成该乘客，并执行该乘客的按梯（层站召唤）动作。

### 数据特征

数据有上行高峰和下行高峰两种模式。在上行高峰中，乘客都从1楼出发，乘坐电梯上行到其他楼层；在下行高峰中，乘客从其他楼层出发，都下行到1楼。以下行高峰为例，以各楼层为出发层的人数与时间也存在一定的关系，下图展示了31天数据中，以2楼、16楼为出发层的出现人数与时间的关系。从图中可以看出，10-20分钟是下行高峰中的最高峰。其余时刻大致服从正态分布:

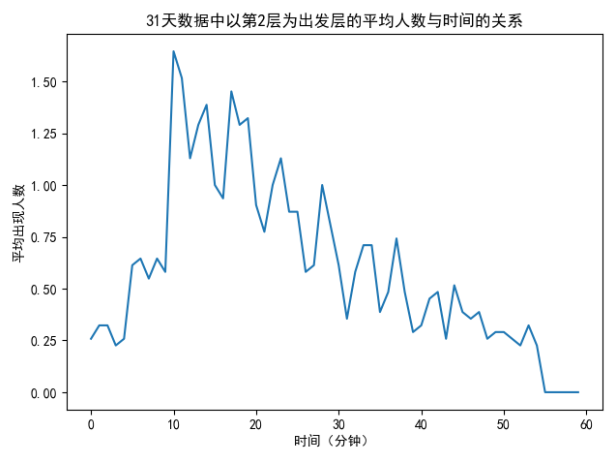


图1.1

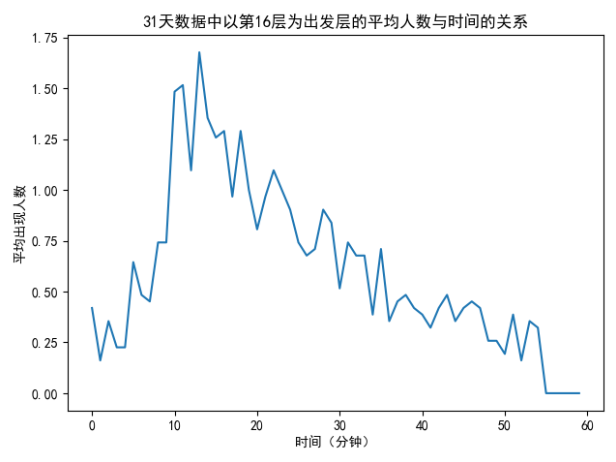


图1.2

# 二．电梯调度技术传统算法

### 先来先服务算法(FCFS)

先来先服务(FCFS-First Come First Serve)算法，是最简单的电梯调度算法。它根据乘客请求乘坐电梯的先后次序进行调度。此算法的优点是公平、简单，且每个乘客的请求都能依次地得到处理，不会出现某一乘客的请求长期得不到满足的情况。这种方法在载荷较轻松的环境下，性能尚可接受；但是在载荷较大的情况下，算法的性能就会严重下降。

### 最短寻找楼层时间优先算法(SSTF)

最短寻找楼层时间优先(SSTF-Shortest Seek Time First)算法，注重电梯寻找楼层的优化。最短寻找楼层时间优先算法选择下一个服务对象的原则是寻找楼层的时间最短。在重载荷的情况下，最短寻找楼层时间优先算法的平均响应时间较短，但响应时间的方差较大，原因是某些请求可能长时间得不到响应。

### 扫描算法(SCAN)

扫描算法(SCAN)是一种按照楼层顺序依次服务请求。它让电梯在最底层和最顶层之间连续往返运行，在运行过程中响应处在于电梯运行方向相同的各楼层上的请求。算法中，每个电梯响应乘客请求的次序是由其发出请求的乘客的位置与当前电梯位置之间的距离来决定的，所有与电梯运行方向相同的乘客的请求在一次电梯向上运行或向下运行的过程中完成。

LOOK算法是扫描算法的一种改进。对LOOK算法而言，电梯同样在最底层和最顶层之间运行。但当LOOK算法发现电梯所移动的方向上不再有请求时立即改变运行方向，而扫描算法则需要移动到最底层或者最顶层时才改变运行方向。

### 最早截止期优先调度算法(EDF)

最早截止期优先(EDF-Earliest Deadline First)调度算法是最简单的实时电梯调度算法，它响应请求队列中时限最早的请求。它的缺点就是造成电梯任意地寻找楼层，导致极低的电梯吞吐率。

### SCAN-EDF 算法

SCAN-EDF算法是SCAN算法和EDF算法相结合的产物。SCAN-EDF算法先按照EDF算法选择请求列队中哪一个是下一个服务对象，而对于具有相同时限的请求，则按照SCAN算法服务每一个请求。

### PI 算法

PI(Priority Inversion)算法将请求队列中的请求分成多个优先级。它首先保证高优先级队列中的请求得到及时响应，在高优先级队列为空的情况下再响应低优先级队列中的请求。

### 基于专家系统的电梯群控调度算法

基于专家系统的电梯群控调度算法，就是用专家的经验知识建立规则，利用这些规则比较各种可能的电梯运行途径，从中选出最佳路线。在电梯控制中，电梯群控系统的特性不能完全用数学进行精确描述，控制经验就显得十分重要，因此专家知识对电梯群控系统的性能具有很大作用。专家系统就是研究这类知识的表述、使用和获取的方法。它是一种知识信息的加工处理系统，其工作过程是一个求解和探索的过程。

专家系统由知识库、数据库、推理机、解释部分及知识获取部分组成。首先，通过知识获取部分获取电梯界专家的知识及经验，经过“知识表达”表达专家的思维与知识，形成一定的控制规则存入到知识库中。数据库存放专家系统当前工作的已知情况、用户提供的事实以及由推理得到的中间结果。例如：电梯呼梯信号分布情况，各电梯位置、乘客人数，预测的乘客等待时间及预测的电梯运行时间等。推理机用于协调整个专家系统的工作，它根据目前的输入数据或信息，利用知识库中的知识按一定的推理策略去控制派梯。解释部分主要是解释系统本身的推理结构，回答用户的问题。由于专家系统应用了人工智能的技术，把一个或多个专家的知识和经验积累起来，并进行推理和判断，解决了许多用传统控制方法难以解决的问题，达到了较为优化的控制目标。

电梯群控系统中的专家系统包括群控管理、通信接口、用户程序设计以及各台控制装置、监视系统等部分，实现厅层呼叫分配、预报的群控功能。首先，根据过去的交通数据预设交通流，以候梯时间最小的最优运行作为大规模组合问题；之后，使用模拟退火法求解此大规模组合问题；最后，将求出的最优运行与原来系统的运行进行仿真比较，并将成为最优运行的分配方法归纳为一般性的规则。

### 基于模糊控制的电梯群控调度技术

模糊逻辑是一种在数学和逻辑上用来处理带有模糊不清的边界对象和问题的有效方法。模糊控制的基本思想，是把人类专家对特定的被控对象或过程的控制策略总结成一系列以"IF（条件）THEN（作用）"形式表示的控制规则，通过模糊推理得到控制作用集，再将控制作用集作用于被控对象或过程，处理控制系统在语言表达上不确定的变量信息。

模糊控制系统的架构包含了五个主要部分，包括：定义变量、模糊化、知识库、逻辑判断及反模糊化。首先，决定被观察的状况及考虑控制的动作，定义模糊变量。之后，通过模糊化，将输入值以适当的比例转换到论域的数值，将系统的各个分量模糊成离散的指标，如候梯时间用“短”、“中”、“长”三种模糊集表示。知识库包括数据库与规则库两部分，其中数据库提供处理模糊数据的相关定义，而规则库则藉由一群语言控制规则描述控制目标和策略。逻辑判断部分模仿人类下判断时的模糊概念，运用模糊逻辑和模糊推论法进行推论，得到模糊控制讯号。反模糊化将推论所得到的模糊值转换为明确的控制讯号，作为系统的输入值。

模糊控制技术在电梯控制系统中的应用显示了高度的优越性。电梯系统中含有许多模糊、不完整的信息，这些模糊信息需要用模糊集来描述。计算机不能接受含糊的回答，但却能用模糊逻辑来推理，它能模仿人脑的推理能力，简化许多复杂问题。模糊控制方法无需建立数学模型，具有较强的鲁棒性和实时性，并且符合人们对过程控制作用的直观描述和思维逻辑。

### 基于遗传算法的电梯群控调度技术

遗传算法是模拟生物进化过程的计算模型，是一种通过模拟自然进化过程搜索最优解的方法，目前已普遍运用于实际问题的解决中。遗传算法的基本要素有以下六个：编码规则与初始化、适应度函数、选择运算、交叉运算、变异运算、终止条件判断。

一般而言，电梯调度模型中的编码规则按照以下方式构建：一个染色体代表一个完整的调度方案；染色体上的每一个基因代表一个已经被分配的楼层，由两部分构成，第一部分确定具体的楼层，第二部分确定该楼层被分配到哪一部电梯。适应度函数一般用电梯调度方案运行所需移动时间与停留时间的总和表示。在具体计算过程中，对个体编码进行反编码得到具体的调度方案，然后就可以根据调度方案计算出相应的适应度。而选择运算、交叉运算、变异运算，则模仿自然中生物遗传的原理，筛选出适应度高的染色体、将不同染色体间的基因互换、将某一染色体的某一基因突变为另一基因。

电梯系统要求根据各个楼层的使用情况来改变系统的设定，使用遗传算法可以根据交通量的变化而变化，实现在线调整几十个控制参数，既保证了系统的稳定性，又可提高系统在使用情况发生变化时的跟随能力。但是遗传算法的计算量较大，为保证采用遗传算法优化派梯的实时性，需要对遗传算法进行速度测试，在启动遗传算法搜索最优派梯方案时每次在速度允许范围内进行若干代搜索即停止搜索，按最优个体派梯。

# 三. 强化学习

### 什么是强化学习

强化学习并不是某一种特定的算法，而是一类算法的统称。强化学习是机器学习中的一个领域，强调如何基于环境而行动，以取得最大化的预期利益。强化学习是除了监督学习和非监督学习之外的第三种基本的机器学习方法。与监督学习不同的是，强化学习不需要带标签的输入输出对，同时也无需对非最优解的精确地纠正。其关注点在于寻找探索（对未知领域的）和利用（对已有知识的）的平衡。以游戏为例，如果在游戏中采取某种策略可以取得较高的得分，那么就进一步“强化”这种策略，以期继续取得较好的结果。这种策略与日常生活中的各种“绩效奖励”非常类似。我们平时也常常用这样的策略来提高自己的游戏水平。

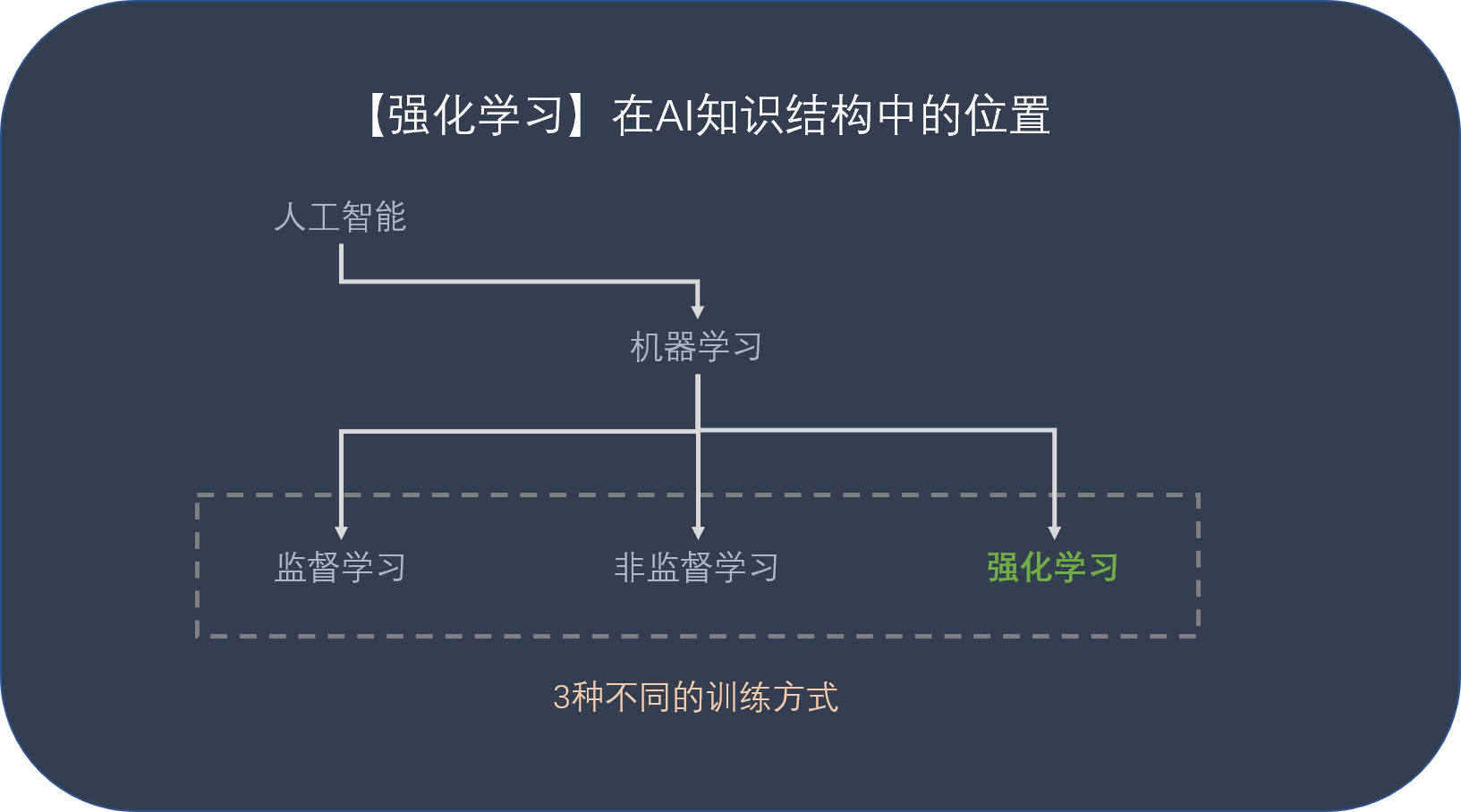


图3.1

在 Flappy bird 这个游戏中，我们需要简单的点击操作来控制小鸟，躲过各种水管，飞的越远越好，因为飞的越远就能获得更高的积分奖励。这就是一个典型的强化学习场景：

* 机器有一个明确的小鸟角色——代理
* 需要控制小鸟飞的更远——目标
* 整个游戏过程中需要躲避各种水管——环境
* 躲避水管的方法是让小鸟用力飞一下——行动
* 飞的越远，就会获得越多的积分——奖励



图3.2

强化学习和监督学习、无监督学习最大的不同就是不需要大量的“数据喂养”。而是通过自己不停的尝试来学会某些技能。

### 强化学习中的基本概念

#### 马尔可夫决策过程（MDP）

在强化学习中，环境通常被抽象为一个马尔科夫决策过程，因为很多强化学习算法在这种假设下才能使用动态规划的方法。传统的动态规划方法和强化学习算法的主要区别是，强化学习算法不需要关于MDP的知识，而且针对无法找到确切方法的大规模MDP。

在一个环境中，如果任意某一个状态信息包含了所有相关的历史，只要当前状态可知，所有的历史信息都不再需要，当前状态就可以决定未来，则认为该状态具有马尔科夫性。如在电梯分派问题中，我们可以把电梯当前的位置、速度、方向、响应楼层作为状态，我们为当前的乘客作分派时，只需要关注当前状态的信息，而不需要知道电梯之前的位置、速度等，我们就可以认为这个状态具有马尔科夫性。可以用下面的状态转移概率公式来描述马尔科夫性：

公式描述了状态为时，下一状态转移到的概率。

马尔科夫过程又叫马尔科夫链(Markov Chain)，它是一个无记忆的随机过程，可以用一个元组<S，P>表示，其中S是有限数量的状态集，P是状态转移概率矩阵。如在电梯分派问题中，1号梯在t0时刻在1楼，被分派了一个位于9楼的下行召唤；下一时刻t1该电梯位于2楼，速度向上；t3时刻位于3楼，速度向上……t9时刻位于9楼，速度为0，接到该乘客，前往7楼；t10时刻位于8楼，速度向下，前往7楼；t11时刻位于7楼，速度为0，到达乘客目的楼层。这个过程就是一个马尔科夫过程，每个时刻对应一个状态，这个状态包含的信息足以决定电梯接下来的运动情况，因此基于这个状态的决策无需考虑以前的状态。

#### 马尔科夫奖励过程 （Markov Reward Process）

马尔科夫奖励过程在马尔科夫过程的基础上增加了奖励R和衰减系数γ：<S，P，R，γ>。R是一个奖励函数。S状态下的奖励是某一时刻(t)处在状态s下在下一个时刻(t+1)能获得的奖励期望：

衰减系数 Discount Factor: γ∈ [0， 1]，执行动作的结果是短期奖励和远期奖励，我们引入衰减系数到远期奖励中，从而保证我们既考虑短期利益，也考虑长期利益，同时把关注点更多的放到能够尽快获得的奖励中。远期利益具有一定的不确定性，因此这样的设定符合人类对于眼前利益的追求，符合金融学上获得的利益能够产生新的利益因而更有价值等等。

#### 强化学习中的基本要素

一个典型的强化学习可以用下图来刻画：



图3.3

强化学习的关键要素有：环境，状态，动作，奖励和策略。有了这些要素我们就能建立一个强化学习模型。强化学习解决的问题是，针对一个具体问题得到一个最优的策略，使得在该策略下获得的奖励最大。所谓的策略其实就是针对不同状态下给出的一系列动作。

* 环境：一个强化学习环境必须能够一步一步地和算法进行交互，交互指的是告诉算法当前环境的一些状态，以及接受算法决策的动作，并且执行这个动作后向算法给出一个反馈，更新当前状态，以此往复。
* 状态：状态通常是算法的输入，它包含了使智能体作出动作所需要的所有信息。所以可以说它需要满足的是马尔科夫性质：如果有了当前状态，过去状态都可以丢弃（果包含t时刻的状态，那么1～t-1时刻的状态即没有包含的必要）。
* 动作：动作一般就是算法的输出。对于强化学习的智能体而言，它的目标就是给出在每个状态应该作出怎么样的动作。动作是智能体能够对环境产生影响的手段，前面提到强化学习与监督学习的一个主要区别就在于强化学习能够通过action对environment进行作用，决定数据的产生方式，从而在获取数据上得到更大的空间。所以一个任务的action设置的最基本要求是能够对environment产生有效影响。在一些任务中，action是固定的，例如游戏，围棋等等；但是在大部分任务中，action需要根据任务场景进行构造，复杂的action space会导致优化难度提高。
* 奖励：奖励是算法学习的指导，在representation+evaluation+optimization的角度来看，奖励是evalution的重要组成成分。奖励的设置往往决定了算法的最终效果是否理想。由于强化学习的目标是最大化累计奖励，应为奖励的设置应当满足reward hypothesis：也就是说我们学习的目标是与期望奖励是一致的。如何有效的设置奖励也成为了强化学习应用的一个重要问题，特别是在实际中，仅仅依靠任务本身的反馈，奖励会变得十分稀疏，因而更近一步提高了训练难度。这其中，不但需要对任务场景的了解，从而将任务目标转化为奖励，而且还需要有足够的计算资源进行尝试。
* 策略：策略是智能体学习的最终目标，它是一个从状态到动作的映射，表示智能体在每个状态选择各个动作的概率。对于随机策略，会按照策略对应的概率选择动作，而确知策略则是直接根据某个规则（例如最大概率）选择固定的最优概率。

#### 强化学习的优缺点

* 对比神经网络，RL一般不需要大量带标签的数据集，而是通过agent和环境的不断交互试错获得用于训练的样本。这就节省了大量的获取带标签样本的时间，也更加接近于人类的学习过程。
* 对比某些传统的控制算法，RL （特别是DRL）可以实现无模型的、端对端的、状态到动作的高维映射关系的自学习。这个优势还是比较明显的，毕竟很多情况下，特别是现实生活中，很多系统的精确模型是很难获取的。（这里的模型你可以简单理解为马尔科夫决策过程（MDP）中的状态转移概率P是否已知）。例如，agent学习走迷宫，基于模型的RL，可以通过某些办法学习拟合获得迷宫的模型（迷宫全局或者局部是怎样子的），也就是说agent可以利用这个模型知道执行了本次动作会走到迷宫的哪里，甚至可以预测未来的几步。相反，无模型的RL 就不知道这个迷宫模型，前期纯粹依靠agent随便走试错撞墙，然后记录下这些样本，用于学习。
* RL具有自学习的能力。例如RL 在自动驾驶上的应用，RL就可以利用车辆遇到的新的复杂情况，学习到新的处理能力，甚至可以泛化到其他的类似的情况。

### 本项目研究多轿电梯分派问题采用强化学习算法的原因

强化学习智能体与环境交互，针对序列决策问题，通过试错的方式学习最优策略。强化学习是一类学习、预测、决策的方法框架。如果一个问题可以描述成或转化成序列决策问题，可以对状态、动作、奖赏进行定义，那么强化学习很可能可以帮助解决这个问题。更宽泛地说，强化学习有可能帮助自动化、最优化手动设计的策略。强化学习考虑序列问题，具有长远眼光，考虑长期回报；而监督学习一般考虑一次性的问题，关注短期效益，考虑即时回报。强化学习的这种长远眼光对很多问题找到最优解非常关键。比如，在最短路径问题中，如果只考虑最近邻居节点，则可能无法找到最短路径。而多轿电梯分派问题正好可以抽象成一个序列决策问题：每一时刻各楼层、召唤请求、电梯的运动可以视为一个状态，状态的变化是确定的，每到达一个人，智能体都为其决策分派一个电梯，在整个事件序列中，智能体为整个人流做出一系列的决策动作，期望达到整体平均等待时间和运输时间最短的目标。同时，我们实现的电梯模拟环境也满足了强化学习算法的训练需求。

# 四. 问题建模

## 强化学习环境

在这个问题基础上，定义强化学习环境如下，整栋大楼以dt为时间单位运转，每隔dt时间，我们：

1. 检测提供的csv数据中是否有人来到某一层呼叫电梯。若有，则在模拟器中生成该乘客，并模拟其按下层站召唤的动作。

2. 检测所有楼层是否存在未响应的层站召唤（尚未给该召唤分配电梯）。若存在，则调用强化学习模型为其指派一辆电梯；同时检测所有已分配电梯的层站召唤，若已分派的电梯尚未服务该召唤，则允许强化学习模型重新为其指派一辆新的电梯。

3. 让所有电梯根据自己的被召唤情况更新目的楼层与前进方向，并根据自己当前的运动状态（运行方向、速度、加速度、开关门速度等物理信息），更新dt之后的运动状态和开关门情况。

4. 接到乘客时，取消该乘客所在层及其方向的层站召唤，模拟该乘客以及该层同方向的乘客按一定的速度进入电梯，并记录此时间用于计算乘客的等待时间，进入电梯后，模拟这些乘客执行轿内召唤按下目的楼层，改变电梯的状态。

5. 乘客到达后，取消该乘客在电梯上的轿内召唤目的楼层，电梯门开后，电梯内以当前楼层为目的楼层的乘客陆续下电梯，并记录此时间用于计算乘客的运输时间。

## 目标与奖励

目标是优化总体乘客的平均等待时间（average waiting time）和平均运输时间（average transmit time）。

因为需要给强化学习模型的选择给予一定的反馈，以便让其学习到其选择策略的好坏，因此要给每一dt的选择动作设计一个合理的奖励。把奖励设定为一个与这两个指标负相关的函数，这样训练模型的时候，模型会不断调整策略来追求获得更多的奖励，也即更短的平均等待时间和平均运输时间。

奖励的设计：每个dt内各个层站召分配动作对应的奖励是一个与当前层站召唤等待人数和当前层站召唤所选择的电梯中car call人数的负相关函数。即

， 表示楼层fi所选择的电梯号。

## 状态的选择与表示

要训练一个模型来为层站做决策，首先要对环境的信息进行整理、打包，得到有用的状态信息，并对这些状态信息进行特征提取，得到神经网络能够理解的特征信息，来作为强化学习模型的输入。在这个电梯环境中，主要关注以下三种状态：

1. 乘客的位置、等待时间。

2. 电梯的位置、运行方向、轿内人数 、已分派的召唤、电梯运行/停止、轿内指令、运行速度。

3. 乘客与电梯的实时分派情况。

前两种状态主要通过邻接矩阵来表示，即楼层与电梯本身的特征。第三种状态中，为了更好地抽象出电梯与乘客的供求关系，以图的形式来表示二者的关系：

1. 节点表示层站乘客和电梯，节点特征为乘客/电梯特征的编码

2. 边表示分派情况和服务模式（上下行）的联系。

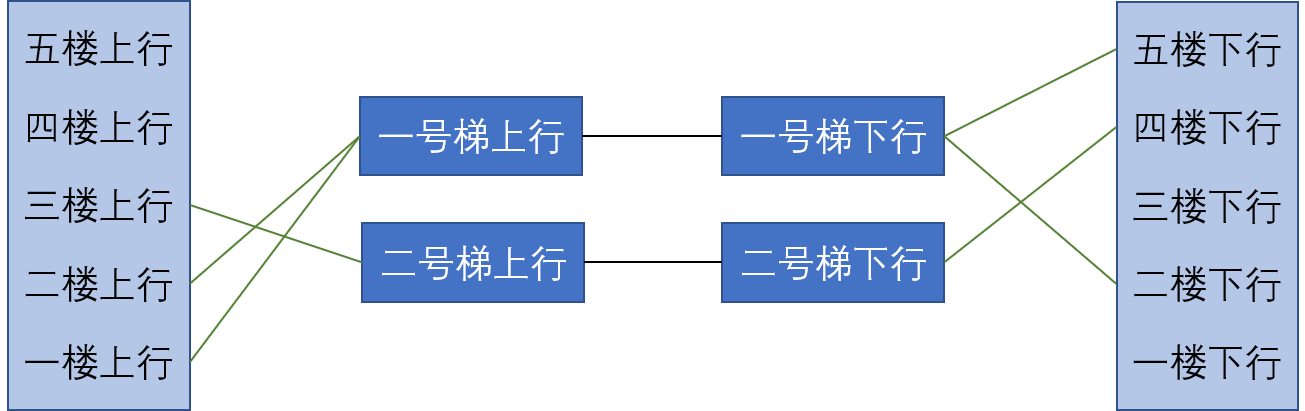


图4.1

## 特征提取网络

选取好环境中有用的信息来作为状态后，我们不能直接把这些信息作为强化学习模型的输入，因为这些信息过于离散且难以被机器理解，我们需要先对这些信息进行预处理，将这些信息编码成利于算法训练的向量，这个过程通常称为特征提取。在深度学习中，常用的特征提取方式有多层感知器（MLP）、嵌入层（Embedding）、卷积神经网络（CNN）和图卷积神经网络（GCN）。这里我们主要采用图卷积神经网络来对选取的状态进行特征提取。

## 多层感知器（MLP）

神经网络其实是对生物神经元的模拟和简化，生物神经元由树突、细胞体、轴突等部分组成。树突是细胞体的输入端，其接受四周的神经冲动；轴突是细胞体的输出端，其发挥传递神经冲动给其他神经元的作用，生物神经元具有兴奋和抑制两种状态，当接受的刺激高于一定阈值时，则会进入兴奋状态并将神经冲动由轴突传出，反之则没有神经冲动。我们基于生物神经元模型可得到多层感知器MLP的基本结构，最典型的MLP包括包括三层：输入层、隐层和输出层，MLP神经网络不同层之间是全连接的（全连接的意思就是：上一层的任何一个神经元与下一层的所有神经元都有连接）。

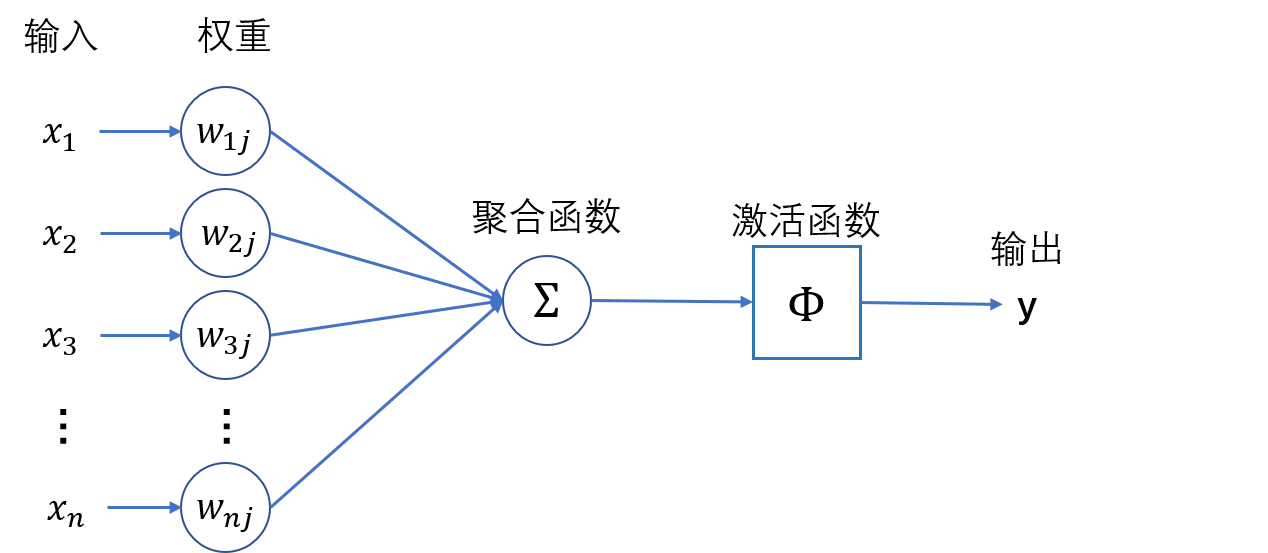


图4.2

由上图可见神经网络主要有三个基本要素：权重、偏置和激活函数。

* 权重：神经元之间的连接强度由权重表示，权重的大小表示可能性的大小
* 偏置：偏置的设置是为了正确分类样本，是模型中一个重要的参数，即保证通过输入算出的输出值不能随便激活。
* 激活函数：起非线性映射的作用，其可将神经元的输出幅度限制在一定范围内，一般限制在（-1~1）或（0~1）之间。最常用的激活函数是Sigmoid函数，其可将（-∞，+∞）的数映射到（0~1）的范围内。

通过训练上述一个MLP网络，我们就可以得到一个能够处理简单分类任务的一个神经网络模型。其中的隐藏层信息包含了神经网络学习到的抽象信息。利用MLP的这个特征，我们可以去掉MLP中最后一层的输出层，仅保留输入层和隐藏层，来作为一个特征编码器，即MLPEncoder。

## 嵌入层（Embedding）

嵌入层也是机器学习任务中常用的一种特征编码方式。

Embedding在数学上表示一个maping：， 也就是一个映射函数。

其中该函数满足两个性质：

1. injective （单射的）：就是我们所说的单射函数，每个X只有唯一的Y对应;

2. structure-preserving（结构保存）：比如在X所属的空间上x1<=x2，那么映射后在Y所属空间上同理y1<=y2。

以常见的Word Embedding为例，把单词 w 映射到向量 x。如果两个词的原意接近，比如coronavirus和covid，那么它们映射后得到的两个词向量 x1 和 x2 的欧式距离很小。又如在推荐系统中，User Embedding，把用户 ID 映射到向量 x。推荐系统中需要用一个向量表示一个用户。如果两个用户的行为习惯接近，那么他们对应的向量 x1 和 x2 的欧式距离很小.

## 图卷积神经网络

神经网络最近的成功推动了模式识别和数据挖掘的进展。许多机器学习任务，例如目标检测、机器翻译、语音识别，曾严重依赖于手工特征工程抽取信息特征集，现在通过各种端到端深度学习范式得到了深刻的改变，例如 CNN、RNN、自编码器。深度学习在许多领域的成功部分归因于快速发展的计算资源（例如 GPU），大量训练数据的可用性、以及深度学习从欧几里得数据（图像、文本和视频）抽取的潜在表示的有效性。以图像为例，我们可以将图像表示为欧几里得空间中的规则网格。卷积神经网络（CNN）能够利用图像数据的平移不变性，局部连通性和合成性。结果，CNN 可以提取与整个数据集共享的局部有意义的特征，以进行各种图像分析。

虽然CNN有效地捕获了欧几里得数据的隐藏模式，但越来越多的应用以图的形式表示数据。例如，在电子商务中，基于图的学习系统能够利用用户和产品之间的交互来提出高度准确的建议。在化学中，分子被建模为图，需要鉴定其生物活性以进行药物发现。在引文网络中，论文通过引文相互链接，因此需要将它们分为不同的组。在我们的多轿电梯分配问题中，需要以图的方式来表示电梯层站召唤的分派与电梯的关系（上图）。图数据的复杂性对现有的机器学习算法提出了重大挑战。由于图可能是不规则的，因此图可能具有可变大小的无序节点，并且图中的节点可能具有不同数量的邻居，从而导致一些重要的操作（例如卷积）在图像域中易于计算，但是应用于图域却是困难的。

## 图的定义

对于图G=(V，E)，V为节点的集合，E为边的集合，对于每个节点i，均有其特征，可以用矩阵表示。其中 N 表示节点数，D表示每个节点的特征数，也可以说是特征向量的维度。通常我们用邻接矩阵来表示图中节点之间的关系。

## 图卷积的通式以及实现

任何一个图卷积层都可以写成这样一个非线性函数：

，其中为第一层的输入，，N为图的节点个数，D为每个节点特征向量的维度，A为邻接矩阵，不同模型的差异点在于函数f的实现不同。一种简单的实现方式是，其中为第

层的权重参数矩阵，为非线性激活函数，例如ReLU。

这种思路是基于节点特征与其所有邻居节点有关的思想。邻接矩阵 与特征 相乘，等价于， 某节点的邻居节点的特征相加。这样多层隐含层曡加，能利用多层邻居的信息。

## 特征提取网络

在我们的特征提取网络中，我们用多层感知器（MLP）分别对每层楼的上行方向等待人数、下行等待人数、电梯内以各层楼为目的楼层的人数、各电梯所在的位置、各电梯距离各个楼层的距离，进行特征提取，得到隐藏层特征向量，并将他们合并起来。用图卷积神经网络（GCN）对电梯与各层站召唤的图邻接矩阵和节点特征进行图卷积，得到隐藏层特征向量，将他们合并起来作为强化学习模型的输入。

# 五. 强化学习模型训练和算法选择

## 为什么要训练

在有监督学习或无监督学习中，我们假设现实世界有一个未知的分布，代表着“大自然的规律”。训练数据是由这个未知分布产生的，我们的目标就是通过这些数据去学习出隐藏在数据背后的未知分布，这就像是通过知识去探求大自然中隐藏的真理。在这个过程中，我们对于所有的知识一视同仁，目标是平等地认识大自然中存在的所有客观规律。我们选用的不同模型就相当于认识世界的不同方式。

在强化学习中，我们的目标不是“学习出环境”而是“最大化累积奖励”。在这个过程中，我们会按照需要去产生数据。举一个下象棋的例子：在求解下象棋的策略时，S1是一个常见而“重要”的状态，而S2是一个罕见且“不重要”的状态。它们都是“世界的组成部分”，地位应该是平等的。但是，由于我们的目标是“在标准对局中最大可能取胜”，所以我们会更加重视S1 。大千世界中有无数的信息，而你只关注你感兴趣的或者和你生活息息相关的信息，这二者是一样的道理。利用这种强化学习的思维模式，我们可以更加高效地完成真正需要完成的任务。

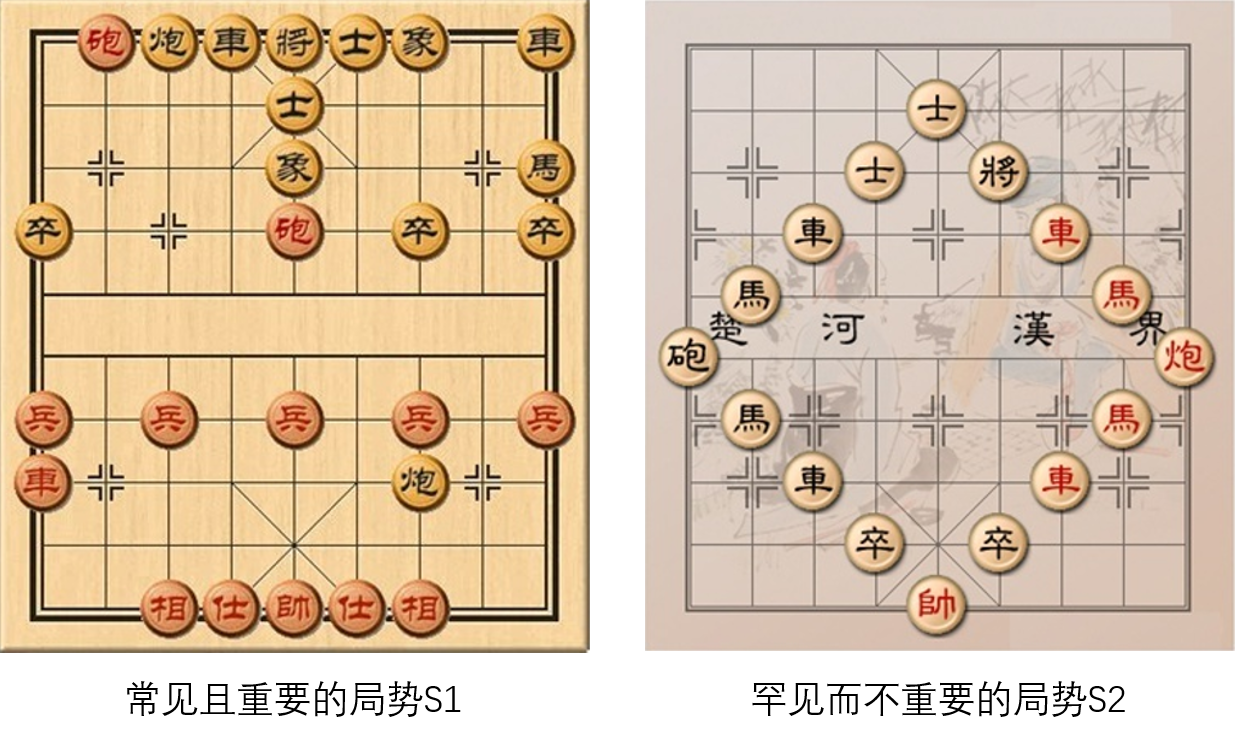


图5.1

同样的，在电梯分派问题中，我们只需要关注相对来说比较“均匀”的分派状态，这里均匀指的是充分利用每一个电梯，而不需要关注每个召唤都分配给同一个电梯这种常人理解不合理的分派状态。总的来说，强化学习是一个不断试错（trial and error）的过程：一开始，模型在各个状态随机尝试各种动作，得到各种状态下，执行特定动作的回报，从而可以不断估算每一个状态的潜在价值，以及在每一个状态下，执行不同动作的价值；通过不断的尝试和训练，这个估算的结果越来越接近奖励函数设定下的真实结果，从而使得模型逐渐收敛，把训练的重心更多地放在价值更高的状态和价值更高的动作上，即上面描述的“更常见而重要的局势”，以此来学习到一个通过执行动作趋向于到达价值更高状态、获得更多奖励的模型。

## 怎么训练

强化学习的训练算法可以分为值基础（Value Based）的算法、策略基础（Policy Based）的算法以及将二者结合的算法。

### 值基础的方法

值基础的方法是通过估算每一个状态下每一种选择的价值Value的大小，通过不断的迭代更新训练，各个状态与对应选择的价值估算慢慢收敛于真实值。此时最优的策略就是每次都选择当前状态中价值最大的动作来执行。以最简单的Q-Learning算法为例，每一个时间片的Q(s，a)和当前得到的Reward以及下一个时间片的Q(s，a)有关。Q-Learning建立在虚拟环境下无限次的实验，这意味着可以把上一次实验计算得到的Q值来估算下一个时间片的Q(s，a)，从而更新当前的Q值。Q-learning的算法如下：

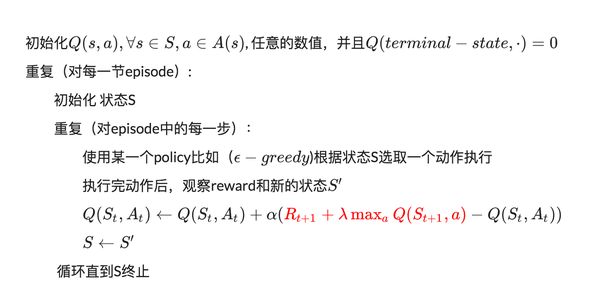


图5.2

对于Q-Learning，首先就是要确定如何存储Q值，最简单的想法就是用矩阵，一个s一个a对应一个Q值，所以可以把Q值想象为一个很大的表格，横列代表s，纵列代表a，里面的数字代表Q值，如下表示：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | a1 | a2 | a3 | a4 |
| s1 | Q(1,1) | Q(1,2) | Q(1,3) | Q(1,4) |
| s2 | Q(2,1) | Q(2,2) | Q(2,3) | Q(2,4) |
| s3 | Q(3,1) | Q(3,2) | Q(3,3) | Q(3,4) |
| s4 | Q(4,1) | Q(4,2) | Q(4,3) | Q(4,4) |

表5.1

在Q-learning中，我们使用表格来表示Q(s，a)，但是这个在现实的很多问题上是几乎不可行的，因为状态实在是太多，使用表格的方式根本存不下。如在电梯分派问题中，一个16层楼4个电梯的大楼，只考虑电梯的位置和层站召唤请求与轿内召唤请求的情况下就有(16^4)\*(2^32)\*(2^64)=2^100≈10^30种状态，这是不可能通过表格来存储并且执行计算的。因此我们需要用一个函数来对价值函数做近似，即用一个函数f去表示Q(s，a)：Q(s，a) = f(s，a)，函数f可以是任意类型的函数，如线性函数：Q(s，a) = w1\*s + w2\*a + b，其中w1、w2、b是函数的参数。这样我们就可以通过矩阵运算来处理高维度的状态与动作。由此我们引入神经网络，用神经网络去近似我们的价值函数，即用神经网络去代替q-table。DQN算法是最经典的值基础方法。神经网络的训练是一个最优化问题，最优化一个损失函数loss function，也就是标签和网络输出的偏差，目标是让损失函数最小化。为此，我们需要有样本，巨量的有标签数据，然后通过反向传播使用梯度下降的方法来更新神经网络的参数。所以，要训练Q网络，我们要能够为Q网络提供有标签的样本。DQN算法通过结合神经网络与Q-Learning算法，用Q-Learning为神经网络的训练提供样本和标签，使得训练的神经网络对当前状态的近似价值更加接近当前状态下得到的奖励与加入衰减系数的下一状态价值之和（这个和称为目标Q值）。

因此，Q网络训练的损失函数设置为如下：

上面公式是s'和a'表示下一个状态和动作。确定了损失函数，也就是cost，确定了获取样本的方式，我们就可以按照深度学习的梯度下降方式来训练我们的Q网络。

具体的DQN算法主要涉及到Experience Replay，也就是经验池的技巧，经验池用于存储样本及采样。由于模拟电梯运动采集的样本是一个时间序列，样本之间具有连续性，如果每次得到样本就更新Q值，受样本分布影响，效果会不好。因此，一个很直接的想法就是把样本先存起来，然后随机采样，这就是Experience Replay。按照脑科学的观点，人的大脑也具有这样的机制，就是在回忆中学习。

总的来说，DQN是通过Q-Learning与环境交互反复试验，然后存储数据。待数据存到一定程度，就每次随机采用数据，对神经网络进行梯度下降，从而学习到一个能够对每个状态和动作都准确估值的神经网络。

### 策略基础的方法

值基础的方法是通过估计状态-动作对的价值，再选择价值最大的动作来执行的。相比之下，算法策略梯度算法是一种更为直接的方法，它让神经网络直接输出策略函数 π(s)，即在状态s下应该执行何种动作。对于非确定性策略，输出的是这种状态下执行各种动作的概率值，即如下的条件概率：

，所谓确定性策略，是只在某种状态下要执行的动作是确定即唯一的，而非确定性动作在每种状态下要执行的动作是随机的，可以按照一定的概率值进行选择。这种做法的原理如下图所示。

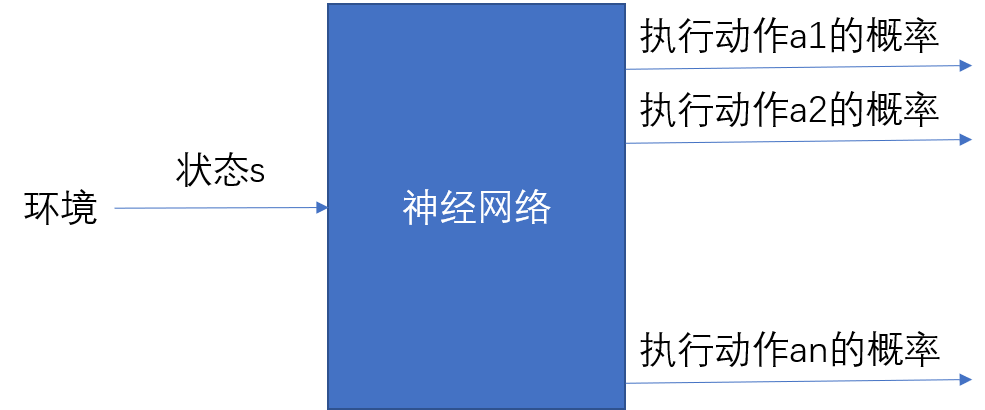


图5.3

此时的神经网络输出层的作用类似于多分类问题的softmax回归，输出的是一个概率分布，只不过这里的概率分布不是用来进行分类，而是执行动作。在电梯分派问题中，由于未来人流的不确定性，我们的策略也不应该是确定性的，我们不能准确的估算出每个状态的具体价值，而应该采用非确定性策略，即每个状态下，以一定的概率来执行每个分派动作，我们要做的就是去学习这个概率应该取什么值。因此，如果我们构造目标损失函数L:

，这里使用了折扣因子。损失函数的输入是神经网络输出的策略函数，通过优化此目标函数，即可确定神经网络的参数θ，从而确定策略函数。这可以通过梯度上升法实现（与梯度下降法相反，向着梯度方向迭代，用于求函数的极大值）。训练时的迭代公式为

，

这里假设策略函数对参数的梯度存在，从而保证存在。策略基础的方法的采样方式采用了与DQN类似的思路，即按 照当前策略随机地执行动作，并观察其回报值，以生成样本。

**策略梯度定理**：对于任何可微的策略，对于任何策略的目标函数，其梯度都如下式所示，转换成了策略梯度;

在监督学习算法中，我们有大量的样本数据，并且数据都有标签，于是在训练的时候知道当前参数下得到的结果是好是坏 (能与监督信息对比)，然后可以根据对比结果调整参数优化的方向 (梯度) 。而强化学习里没有一个监督信息告诉我们当前的策略是好是坏，我们得去计算价值函数来起到标签的作用。

蒙特卡洛策略梯度（REINFORCE）是一个典型的策略基础的算法，具体流程如下：

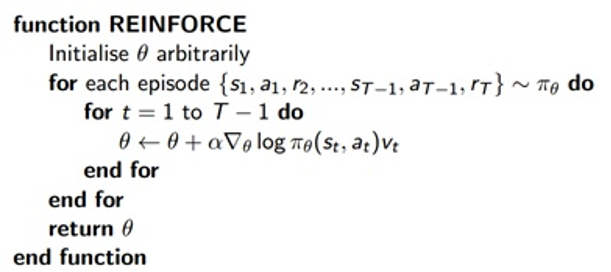


图5.4

首先随机初始化参数，对每个episode，计算其t=1到t=T-1的return ，然后使用随机梯度上升法更新参数。对于策略梯度定理公式里的期望，我们通过采样的形式来替代，即使用t时刻的return作为当前策略下动作价值函数的无偏估计。

#### 演员-评论家算法（Actor-critic算法）

上述策略基础算法也存在一定的问题：由于agent在一个episode中会采取很多动作，我们很难说哪个动作对最后结果是有用的，换句话说，这种算法存在高方差（variance）;同时也造成了收敛速度慢的问题。为了解决这个问题，演员-评论家算法通过引入一种评价机制来解决高方差的问题，它是一种把值基础与策略基础结合起来的算法。具体来说，Critic就类似于**策略评估**，去估计动作值函数：

于是，Actor-Critic算法中就有两组参数：Critic：更新动作值函数参数 ；Actor：以Critic所指导的方向更新策略参数 。所以说，Actor-Critic算法是一种近似的策略梯度

，actor-critic算法的伪代码如下：

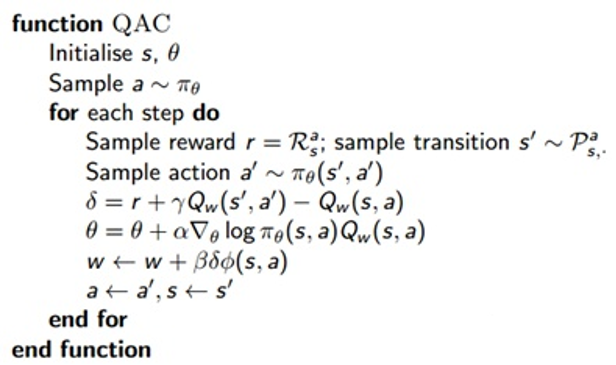


图5.5

训练过程中，我们用线性TD（0）算法来更新critic网络的参数，用策略梯度来更新actor网络。由于这个算法中我们用了critic对状态的估计值来替代了真实的价值函数，因此它用的是一个近似的策略梯度，这样会引入一个偏差，导致最后无法收敛到一个合适的策略。一个解决办法就是通过精心设计这个估计函数 使得其满足以下条件：

1. 近似价值函数的梯度完全等同于策略函数对数的梯度。
2. 值函数参数 使得均方差最小。

满足这两个条件，那么 ，即满足策略梯度定理。

#### Advantage actor critic(A2C)算法

A2C算法是一个经典的actor-critic算法的改进版本，算法通过引入一个advantage函数来减小值函数估计的方差，从而使得actor网络的更新满足策略梯度定理，加速它的收敛。优势函数Advantage function定义为，则策略梯度为。其中状态值函数V(s)表示该状态下所有可能动作所对应的动作值函数乘以采取该动作的概率的和。动作值函数Q(s，a)表示该状态下的a动作对应的值函数。

则优势函数的含义为动作值函数相比于当前状态值函数的优势。

如果优势函数大于零，则说明该动作比平均动作好，如果优势函数小于零，则说明当前动作还不如平均动作好。在工程实现中，只需要维持一套参数用于估计V(s)，并且做梯度下降更新参数的时候可以使用。

A2C算法也是我们在实验中采用的算法，具有方差小、训练快、易于收敛等优点。

## 训练的指标

损失（loss）和奖励（reward）是以A2C为例的强化学习算法中两个比较重要的指标。这里奖励对应的是一个反映平均等待时间和平均运输时间的函数。训练算法时，我们运行模拟器进行模拟，每一步都通过智能体对当前状态进行决策，并得到奖励。训练的过程可以概括为：智能体首先根据当前状态所得到的的奖励来估计每一个状态的价值，然后在这个估计的基础上，选择对当前最有利的动作（选择智能体认为最合适的电梯），而损失（loss）反映的就是智能体对状态价值的估计是否准确。训练的目标就是通过不断优化网络来降低评估的损失，来使得智能体的估计更加准确，从而更准确地选择它认为最好的动作，来得到更大的奖励。

# 六. 实验

## 实验设置及超参数

使用了A2C算法，学习率为2.5\*10^-5，算法的向后执行步数num step为128，损失函数为价值评估损失、动作损失、交叉熵损失的加权和，权重比例为1：1：100，并行环境数量env num为8，在有GPU的Linux/Win10机器上训练，占用GPU内存1133MB，训练时间约为12小时。实验中采用的数据是上行高峰的数据，上行高峰的数据特征如下图所示。上行高峰中，以第一层为出发层的出现人数大致服从成正态分布，随着时间的流逝，人流量先递增后递减，在30-40分钟时，人流量最大，最大值达到每分钟出现约20人，其余时间基本每分钟至少出现5人。

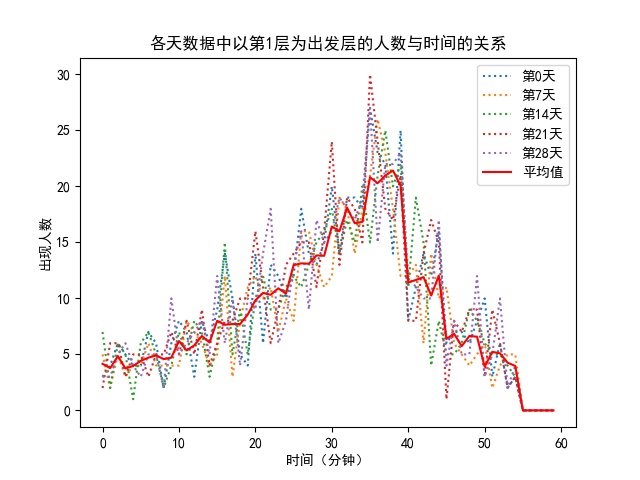


图6.1

## 实验一：以整个上行高峰为训练数据

### 以一天上行高峰为训练数据

我们以一天上行高峰（00：00~59：59）共一个小时为训练数据进行训练时，得到的loss随着训练过程的变化曲线如下。可以看到loss随着训练的过程在平稳下降，最后慢慢收敛，符合我们的预期。

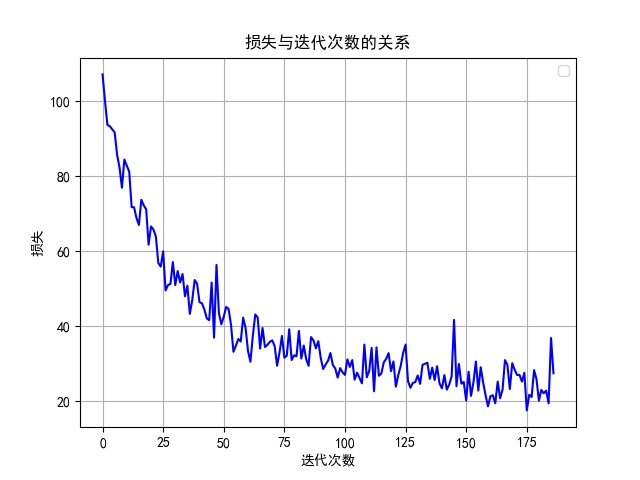


图6.2

得到的reward随着训练过程的变化曲线如下，红线为过去50个迭代到当前迭代的平均值。可以看到问题是reward基本没有得到优化，而上述loss曲线也存在的一个问题是loss的初始值以及收敛值都很大，收敛值为30左右。

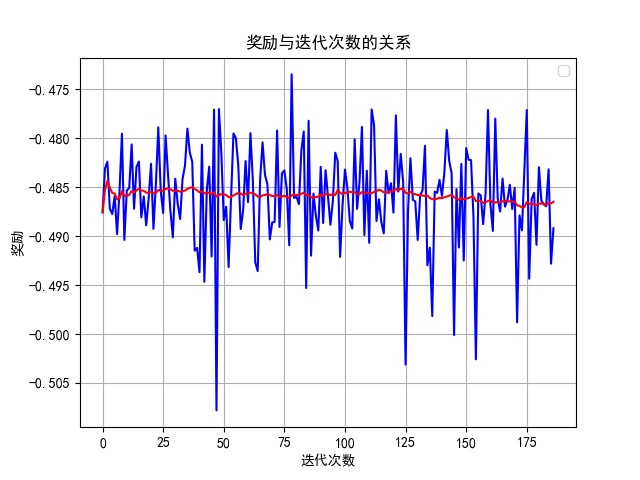


图6.3

实验中我们每50个迭代对模型进行一次评测，得到的评测时间（评测时间为平均等待时间与平均运输时间之和）随着训练过程的变化曲线如下，可以看到问题是评测时间并没有随着训练的进行而得到逐步的优化，只是偶尔会有一些抖动，存在随机性。

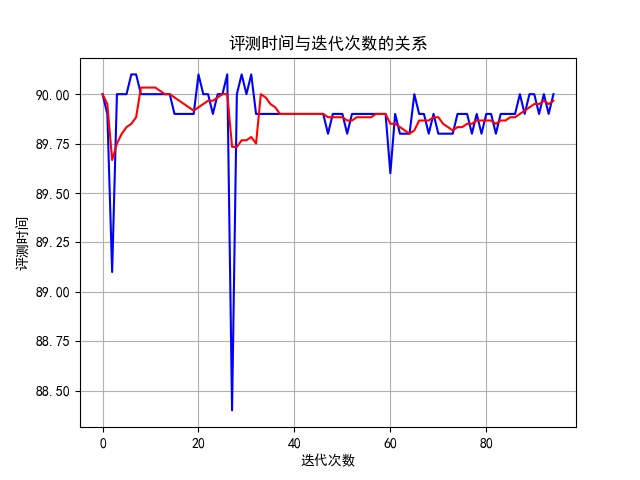


图6.4

### 以31天上行高峰为训练数据

我们扩大了数据集，以31天数据来训练，也存在同样的问题，并且loss的初始值和收敛值更加的大。

### 实验分析

分析猜测可能的原因如下：

1. 人流出现的不确定性过大导致环境过于复杂。由上行高峰数据特征分析可知，人流的出现情况与时间大致成正态分布关系。但是在我们的模型输入中，是没有时间的概念的，也就是说，模型是无法学习到“在0-20分钟的时候，人流量为每分钟5-10人；在30-40分钟的时候人流量为每分钟15-20人”这个特征的。模型只能把把这个特征归咎为当前这个环境是复杂多变且非常不稳定的，人流量在每分钟5人到每分钟20人之间变化。每分钟5人和每分钟20人差别还是很大的，这种环境的不确定性使得模型的价值评估变得有些困难，即损失（loss）会比较大，模型也很难学到一个能够在人流量为每分钟5人到每分钟20人之间都通用的好策略（相信实际中即使存在这样的策略，它也不会是一个优秀的策略）。
2. 在人流量过大时（如在30-40分钟时，人流量为每分钟20人），各部电梯实际上一直在运转，可视化可以发现，这个时候1楼永远有人在等待，且等待人数超过十人（一部电梯的载客量），这时候就没有空闲的电梯，所有电梯要么在执行电梯内乘客的轿内召唤，在前往高楼层的路上，要么是在去往1楼响应1楼的层站召唤的路上。在这种情况下，可以优化的空间非常小，优秀的策略基本就是哪个电梯离1楼近就选哪个电梯（如果允许的话，还可以加上电梯到达电梯内乘客最高的轿内召唤楼层后自主前往1楼这样的设定），这也会造成我们的模型难以改进优化。

为此我们又设计了以下实验来验证我们的猜测，只取上行高峰中的某一个时间片段作为数据来进行训练，其中的三个子实验分别是：

1. 只取31天上行高峰数据中的10：00到20：00这段时间的人流数据作为样本对模型进行训练。
2. 只取31天上行高峰数据中的30：00到40：00这段时间的人流数据作为样本对模型进行训练。
3. 只取31天上行高峰数据中的30：00到45：00这段时间的人流数据作为样本对模型进行训练。

## 实验二：以上行高峰中的一段时间为训练数据

### 以31天中10：00到20：00时间段为训练数据。

取31天上行高峰数据中的10：00到20：00这段时间的人流数据，如下图所示。可以看到平均的人流量在每分钟5人到每分钟10人之间，相对来说比较稳定，人流量也不是很大，环境有所简化。（当然不同天的数据还是有差异的，如在16分钟的时候，人流量最小的一天为每分钟3人，最大的达到每分钟15人。）

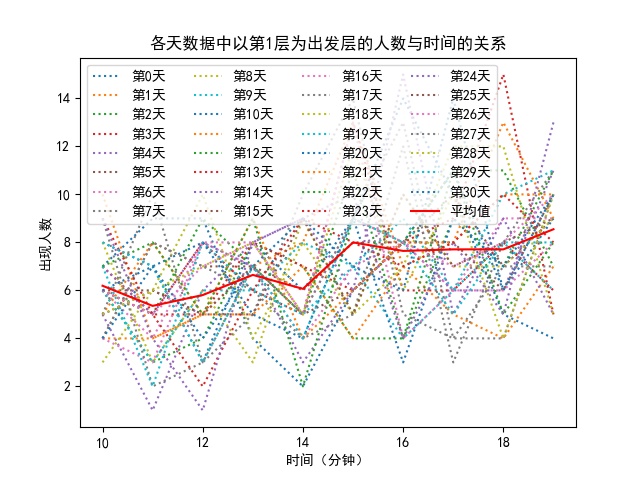


图6.5

我们以这段时间为训练数据训练我们的强化学习模型，得到的loss图如下。loss始终在一个较小的值（0~3.5之间），不像前面的实验那样loss的值为0~100。随着训练的过程，loss逐渐下降并收敛到一个较小的值。

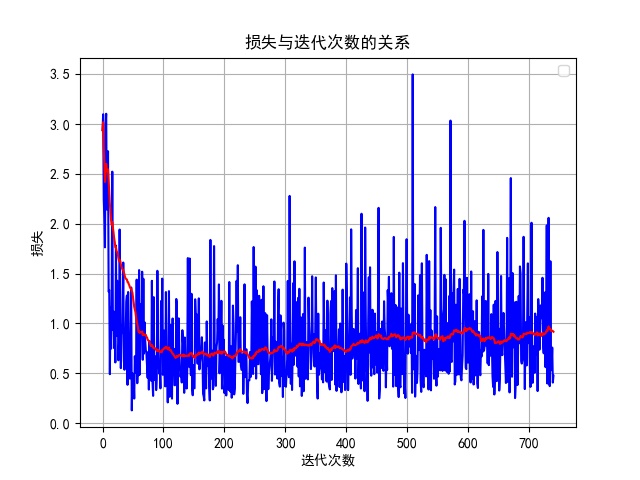


图6.6

平均奖励如下图所示。随着训练的进行也有所提升。

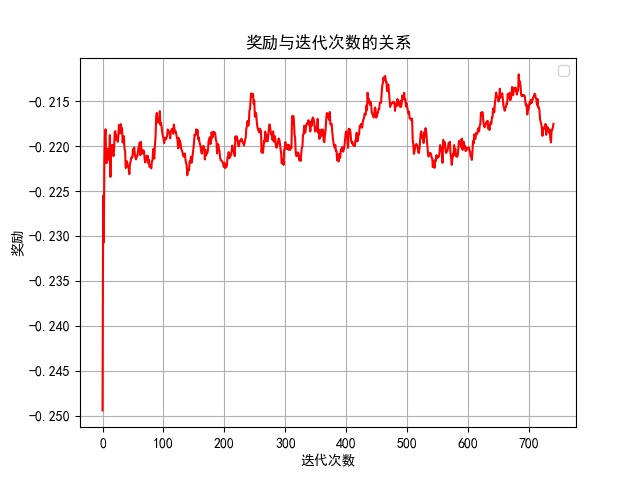


图6.7

实验中我们每300个迭代对模型进行一次评测，得到的评测时间（评测时间为平均等待时间与平均运输时间之和）随着训练过程的变化曲线如下，红线是每20次评测的平均值，可以看到效果是评测时间整体上随着训练的进行逐渐有所优化。

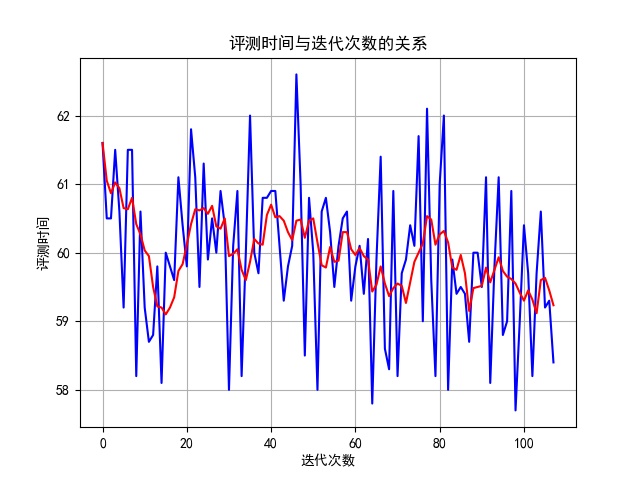


图6.8

与上两个实验相比，这个实验中的模型的训练过程更加的稳定，训练优化效果更加明显。

### 以31天中30：00到40：00时间段为训练数据

取31天上行高峰数据中的30：00到40：00这段时间的人流数据，如下图所示。可以看到平均的人流量在每分钟15人到每分钟20人之间，在不同的天数中，差异较大，人流量在每分钟10人到每分钟30人之间。相对10：00到20：00的数据来说，环境的变化比较大，人流量比较大，环境比较复杂。

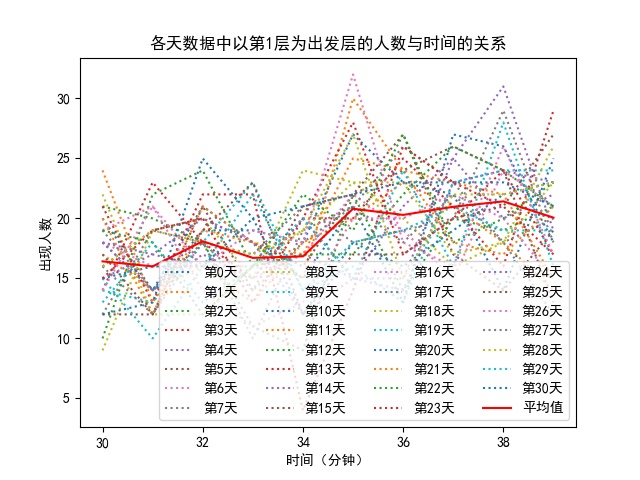


图6.9

我们以这段时间为训练数据训练我们的强化学习模型，得到的loss图如下。loss随着训练的过程从1400慢慢下降，最后收敛到一个相对较小的值（80左右），但是相对10：00到20：00的实验来看，loss还是比较大，且最后loss的震荡也比较厉害。

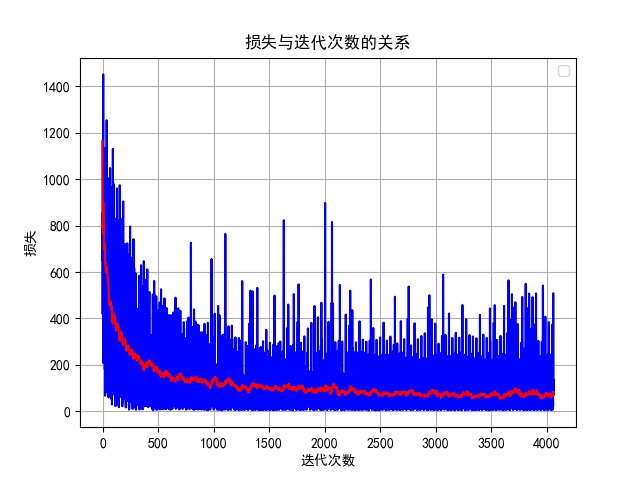


图6.10

平均奖励如下图所示。随着训练的进行抖动较大，在训练的初始阶段有所提升，后续提升不明显。

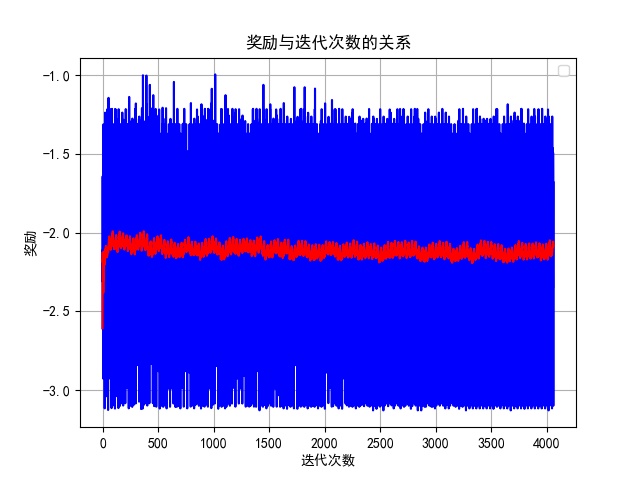


图6.11

实验中我们每300个迭代对模型进行一次评测，得到的评测时间（评测时间为平均等待时间与平均运输时间之和）随着训练过程的变化曲线如下，红线是每20次评测的平均值，可以看到评测时间随着训练的进行有一定程度的下降。

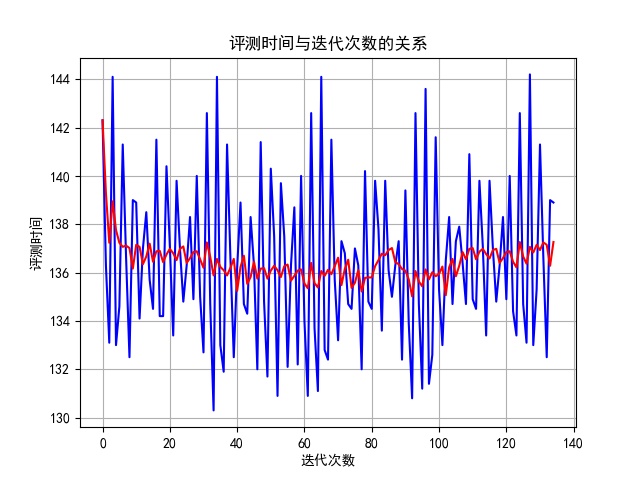


图6.12

### 以31天中30：00到45：00时间段为训练数据

取31天上行高峰数据中的30：00到45：00这段时间的人流数据，如下图所示。可以看到平均的人流量在每分钟10人到每分钟25人之间，在不同的天数中，差异较大，人流量在每分钟5人到每分钟30人之间。相对10：00到20：00的数据来说，环境的变化更加大，人流量也很大，环境更加复杂；相对30：00-40：00的数据来说，增加了40：00-45：00这段时间的人流量的变化，这段时间的平均人流量在每分钟10人到每分钟15人之间，主要为了验证人流量变化对环境和模型的影响。

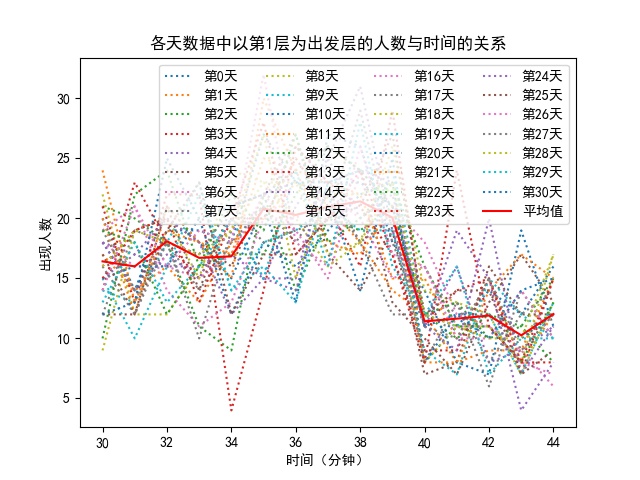


图6.13

我们以这段时间为训练数据训练我们的强化学习模型，得到的loss图如下。loss随着训练的过程从4000慢慢下降，最后收敛到一个相对较小的值（100左右），但是相对10：00到20：00的实验来看，loss还是比较大。

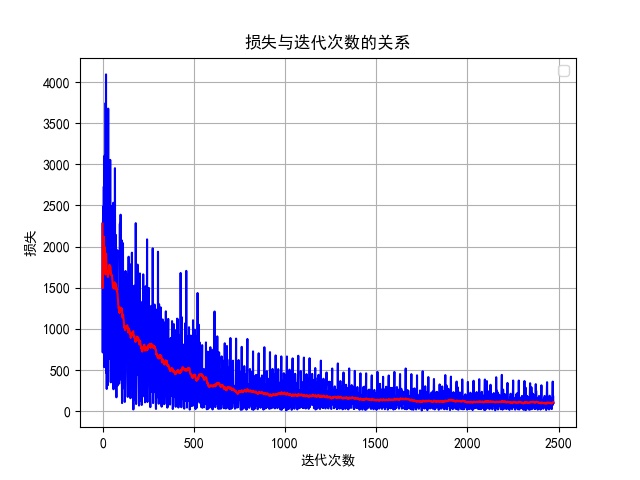


图6.14

平均奖励如下图所示。随着训练的进行抖动较大，提升不明显。

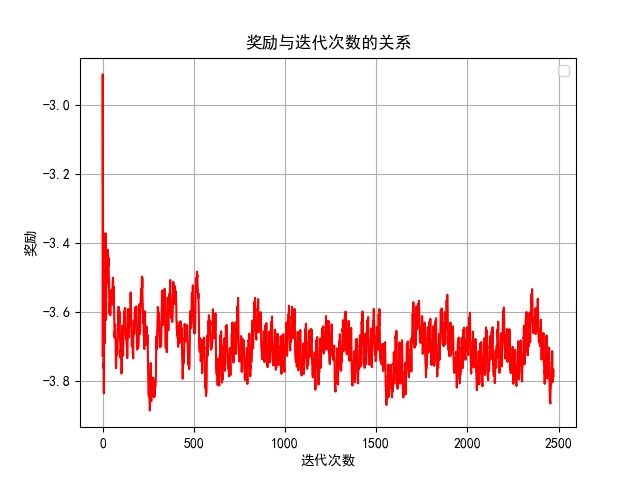


图6.15

实验中我们每300个迭代对模型进行一次评测，得到的评测时间（评测时间为平均等待时间与平均运输时间之和）随着训练过程的变化曲线如下，红线是每20次评测的平均值，可以看到评测时间的优化也不太明显。



图6.16

### 实验分析

以10：00到20：00为训练数据的实验中，我们可以训练得到一个优化效果较明显的模型，在以30：00到40：00为训练数据和以30：00到45：00为训练数据的实验中，从损失函数来看，我们的模型确实有在训练并优化最终收敛，但由于环境过于复杂，损失函数最终收敛的值仍然过大，从奖励函数和评测时间来看也可以看出来模型的优化效果不明显。通过第一个实验和第二个实验的对比，我们可以得知在人流量过大时，由于所有电梯均在不停地运转，优化的空间较小；通过第二个实验和第三个实验的比较，我们可以得知人流量的变化较大会使得强化学习的环境更加复杂，从而导致模型的损失过大，难以学习出一个优秀的策略，优化效果不明显。

## 实验结论

1. 在人流量相对固定且不太大时，强化学习算法能够较通过训练稳定地学习到一定的策略，优化乘客的平均等待时间和平均运输时间。
2. 在人流量变化较大时，强化学习的环境会变得更加复杂，从而导致模型的损失过大，难以学习出一个优秀的策略，优化效果不明显。
3. 在人流量过大时，由于所有电梯均在不停地运转，选择的空间较小，强化学习的优化的空间较小。

# 七. 强化学习结合搜索框架图

为了能够使得强化学习更加稳定，我们在强化学习模型结合搜索算法，具体框架图如下：

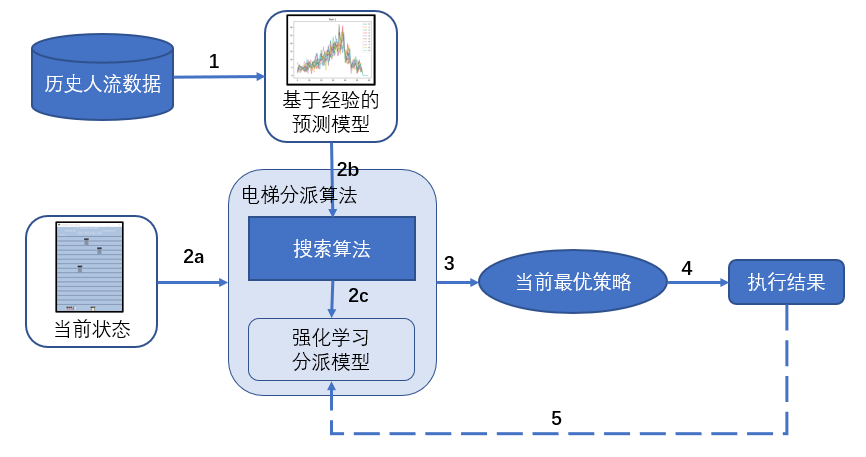


图7.1

通过分析历史人流数据，可以得到各楼层出现人数与时间的关系，我们可以用数学分布去拟合这个关系，得到一个基于历史人流数据预测某一时间点各楼层出现人数的模型。

根据当前的状态（如各楼层层站召唤的请求情况）以及上述经验模型预测结果，我们可以利用搜索算法搜索出一些模拟的较优分派方案。

根据当前状态和上述搜索得到的较优分配方案，我们拿来作为输入去训练我们的强化学习分派模型，得到当前最优的策略。

将这个电梯执行策略在百度电梯模拟器上执行，得到一段时间内各个乘客的平均等待时间和平均运输时间。

将这个结果反馈更新到强化学习分派中，更新神经网络的权重参数，使其能够更好的预测。

# 八. 搜索算法

由于环境过于复杂，对强化学习来说难以直接学习到一套完整的分派策略。为了改善这个问题，我们可以将电梯分派问题分而治之：首先通过神经网络、强化学习等方法对人流情况进行预测，再把这个预测结果和大楼以及电梯的当前状态、历史的人流数据结合起来，通过搜索的方法来得到一个当前最优的策略，把这个策略放到模拟器上执行，得到一个执行的结果（包括平均等待时间和平均运输时间），最后再把这个结果反馈到预测模型中对模型进行优化和更新。

我们最近的工作就是在这个基础上，尝试用搜索的方法来求解电梯分派问题。为了让问题拥有更大的搜索空间，我们选取了午餐高峰来作为我们的实验研究对象（因为既有上行、又有下行，且上下行的楼层足够丰富，问题也非常复杂，在4电梯16层的背景中，问题的解空间最多可达4^32之大）。

## 蒙特卡洛树搜索

蒙特卡洛树搜索（英语：Monte Carlo tree search；简称：**MCTS**）是一种用于某些决策过程的启发式搜索算法，经典的应用场景就是在围棋程序中的应用。著名的围棋程序AlphaGo就是将深度学习与蒙特卡洛树搜索结合的产物，可见蒙特卡洛树搜索是很强大的。

这里我们通过事件触发决策，（有新的召唤到来即视为一个事件），每次决策为这个新的召唤通过蒙特卡洛树搜索，寻找最优的电梯分配。蒙特卡洛树搜索需要模拟执行，在这段模拟执行中，为了简化实验，以便验证效果，我们假设未来的人流数据已知。同时我们还要定义树搜索的终点，在这个场景中，我们以向后执行60秒为一次搜索的终点，以这个终点的平均等待时间和平均运输时间来衡量当前决策的好坏。而对于这60秒中的模拟执行中产生的新的召唤，我们则是通过大量的（playout设为500）探索，来更加准确地估测当前时间所做决策的好坏。

## 遗传算法

由于未来人流情况是未知的，如果通过预测来假设未来人流数据已知，会引入新的不准确因素，造成性能的不稳定。我们还是转向只基于当前电梯情况的搜索，即搜索当前的最优分配，同时，由于只要楼层的hallcall未被电梯服务（即接到该层站召唤的人），就允许重新为其分派别的电梯，我们可以不断动态调整电梯分派方案，每一时刻都选取当前最优解，从而达到一个整体上的近似最优解。

这里我们尝试了经典的启发式搜索算法——遗传算法。我们以每个层站召唤为一个基因节点，如5层楼2个电梯的场景中，假如在0层、1层、2层有上行召唤，3层有下行召唤，那么一个可能的解就是【1，1，0，-1，-1，-1，-1，1，-1，-1】（-1表示没有召唤，0和1代表不同的电梯号）。我们在搜索最优解时，选取200个个体，根据个体的得分以相应概率选择出来，以80%的概率将此解的某一个片段与别的解进行交叉，以10%的概率对此解的某些节点进行变异，形成子代，并计算子代的得分，并对子代计算得分和排序，（这里我们对遗传算法做一个改进：在保留子代时，我们将子代与其父代一起比较，最后只保留得分前200的个体作为下一代），最后经过50代后，选择得分最高的为最优解。

## 局部搜索算法

### 损失计算

对一个电梯而言，在没有新增的外来变化的情况下，当前时刻我们已知的信息就只有这个电梯的位置、运动状态（速度、加速度）、开关门信息、已分配层站召唤和轿厢召唤。为了使得搜索算法能够正确运行，寻找更优解，我们需要对电梯的状态进行评估，这里我们通过计算电梯完成当前前往层站召唤和轿厢召唤所需的总时间来作为这个电梯的损失，对所有电梯求损失并求和即可评估当前的分派方案的好坏。（因为正在执行的层站召唤所服务的乘客要去的楼层未知，因此我们优化的时候不考虑这个指标，只考虑优化正在执行的层站召唤的等待时间和正在执行的轿厢召唤的运输时间。）这里计算电梯的完成时间有两种方法：电梯模拟运行和实验公式计算。

### 电梯模拟运行

在我们的模拟器中，复制电梯当前状态，用模拟器以dt=0.1s为间隔模拟电梯运行，直到电梯完成当前所有召唤停下，记录其中每个召唤完成所需要的时间。

这个方法的优点就是实现简单、计算精度取决于模拟器的对齐情况；缺点是因为要以dt时间间隔来模拟运行，所以运行速度上相对比较缓慢，特别是搜索空间大的话。

### 实验公式计算

因为电梯在两个召唤之间，始终以速度为0出发， 再到速度为0停下，运动的状态相对来说是比较固定的，也就是说，电梯从运行x楼的时间是固定的，我们可以通过实验来测量出电梯运行Δf楼需要的时间。评估电梯的运行时间时，我们可以根据电梯的层站召唤和轿厢召唤情况求出电梯的全部旅程，如：电梯当前在3楼，即将上行到5楼，再上行到13楼，再下行到7楼，最后下行到1楼。这样我们就可以通过公式估计出电梯的运行时间。（当然，在第一段路中，电梯在3楼的时候可能初始速度不为0，因此第一段需要特殊处理。）

### 贪心算法

我们的贪心算法如下：决策由事件触发，（因为如果之前的决策是最优的决策的话，在没有新的事件产生的话，电梯的运行不会使得之前的决策变好或变坏，因此无需重新搜索分配），当产生新的召唤时，将此召唤分别尝试加入到各个电梯中，计算加入此召唤后，总损失的变化大小，贪心地为此召唤选择使得总损失增加最少的那个电梯。

### 探索更优解

由于贪心只针对新增的召唤，对之前已分配当还没服务的召唤来说，旧的分配可能由于新事件的产生已经不是最优了，因此需要重新探索是否存在更优解。这里我们用了类似广度优先搜索的局部搜索，首先把当前最优方案加入到待搜索队列，遍历待搜索队列，对待搜索方案的每一个层站召唤的分派，替换成别的电梯后，计算新的分派的总损失，若总损失小于当前最优解的损失，则把此分派加入到待搜索队列，重复此过程直到搜索队列为空。得到最优解。

实验表明，通过公式计算的方法对比电梯模拟运行的损失计算方法可以加速3倍，决策速度可以达到0.1秒到0.5秒之间。

### 结合历史数据赋予权重

考虑到不同的层站召唤的重要程度是不一样的，如：在午餐高峰中，一层上行的召唤就比五层下行的召唤更重要，因为一层上行的人更多，可能有十个人在等候，但是五层下行只有一个人在等候，所谓一个人等一分钟就是一分钟，但是十个人等一分钟就是十分钟，所以我们要给不同的层站召唤赋予不同的权重。

这里我们采取的权重计算方式是：首先计算历史人流数据中，每个楼层每分钟出现的平均人数，则权重等于（当前层站召唤的时间-上次服务此层站召唤的时间）X 该楼层每分钟出现的平均人数。这个权重的含义是，预测此时该楼层有多少人在等待。

# 九. 实验

## 实验数据——午餐高峰

午餐高峰的数据特征为每分钟约有1~2人从2~16楼下行到1楼，每分钟约有5~10人从1楼上行到各个楼层。30天数据中以1楼为出发层到其他楼层的人数与时间的关系图如下：

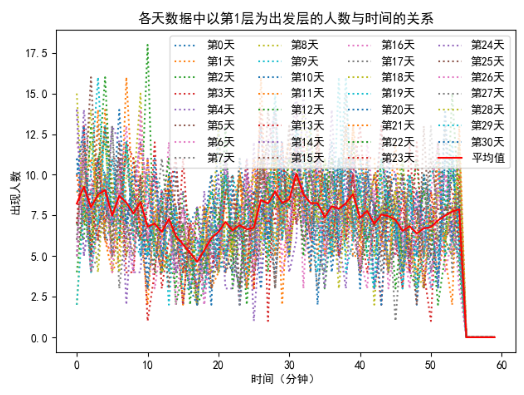


图9.1

以第8层为出发层的人数与时间的关系图如下：



图9.2

第27分钟时，以各楼层为出发层到各个目的层的人数如下：

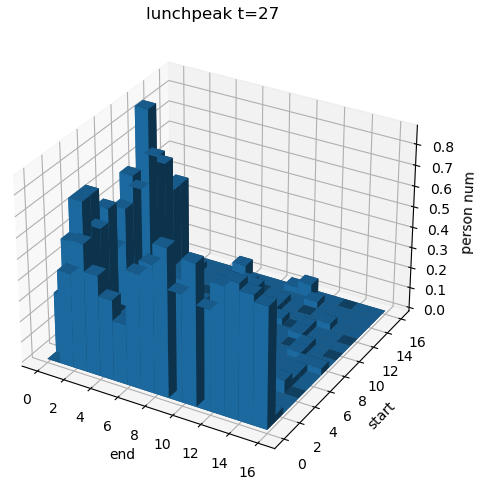


图9.3

可以绝大多数人都是从1楼出发或以1楼为目的层。

## 蒙特卡洛搜索

在上行高峰前十分钟的数据集中，蒙特卡洛树搜索与最短距离算法相比结果如下：

|  |  |  |
| --- | --- | --- |
|  | 平均等待时间 | 平均运输时间 |
| 最短距离算法 | 7.61 | 56.17 |
| 蒙特卡洛树搜索 | *9*.*04*, | *46*.*11* |

蒙特卡洛树搜索在平均等待时间只差了2秒的情况下平均等待时间优化了近10秒。可见蒙特卡洛树搜索算法效果还是很好的，但是，蒙特卡洛树搜索一个很大的缺陷就是运行速度非常慢，因为需要大量的模拟，10分钟的模拟数据运行了将近40分钟，这在实际应用中可能实用性很低，而且实验中为了简化，我们假设了未来数据已知，这都是蒙特卡洛搜索的缺陷。

## 遗传算法

这里遗传算法表现出和蒙特卡洛一样的一大缺点就是非常慢，10分钟模拟数据要运行40分钟，我们通过减少后代数量、减少代数的方法来加快搜索速度，但是仍然难以达到有实用性的水平。因为遗传算法是一个随机算法，进行了很多无用的搜索，为此我们改用更加直接的局部搜索算法。

## 局部搜索

我们选取午餐高峰的10-20分钟的人流数据与三菱算法给出的表格数据进行了比较，结果如下：

|  |  |  |
| --- | --- | --- |
|  | 平均等待时间 | 平均运输时间 |
| 三菱算法 | 30.9 | 47.12 |
| 局部搜索算法 | 26.3 | 45.89 |

可以看到，在平均等待时间和平均运输时间上都有所优化。不过因为局部搜索算法是在本地模拟器上测的而三菱算法是在Elevate上测的，可能也不完全准确，更精确的结果还要进一步在Elevate上测试才能给出。

# 十. 与优化结合的RL

## 基于优化的方法

我们复现了文章《Submodular Function Maximization for Group Elevator Scheduling》中提出的优化方法（以下简称为SFM），并在Elevate中进行了测试，发现它可以超过Elevate中的ETA算法，并与三菱算法csv表格中给出的结果相差较少，因此我们在百度模拟器上探索时，主要以此算法来作为基准。这篇文章提出的方法以

为优化目标，其中，代表的是i楼层的等待时间的一个最坏情况，A是分配情况，是i楼层的等待情况，C是一个常数。如果策略没有执行任何分配，即A为空集，则将目标函数规约到，而使用贪心策略进行分配（把i楼层分配给电梯c，即），可以保证

即目标函数一直在被优化。分配策略的贪心算法被设计如下：



可以证明这个算法的结果至少能达到1/2的最优结果。

我们在百度模拟器上对这个算法进行了调优，并实现了其他传统算法，在同一数据集（午餐高峰）上进行了对比，记过如下：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 平均等待时间 | 平均运输时间 | 总时间 |
| 最近电梯 | 56.36 | 58.47 | 115 |
| Roundrobin | 45.34 | 51.96 | 97.30 |
| LongestQueueFirst | 41.43 | 55.28 | 96.71 |
| ETA | 38.85 | 56.28 | 95 |
| SFM | 36.6 | 52.35 | 89 |
| RL(no mask) | 37.9 | 55.6 | 93.5 |
| RL(mask) | 32.3 | 52.9 | 85.2 |

表10.1 百度模拟器中不同算法在午餐高峰下的平均等待时间和平均运输时间

我们多次测量了SFM方法，发现其结果不太稳定，效果在85到93之间，但最好结果也接近RL的最好结果。并且其拥有不需要训练，能应对多种人流模式的优点，因此，我们接下来在RL中结合了这种方法的优势。

## 结合优化方法的RL方法

首先为了能提高RL的泛化能力，我们注意到使用给定的30天数据来训练对RL来说数据量太少，难以泛化到其他数据。为此我们对30天数据的人流特征进行分析与提取，在训练的每次迭代中都根据分析的人流特征来生成一个新的人流数据，这样可以大大提高数据量，同时人流数据也可以保存下来，进行重复测试。我们分析SFM与RL的可视化对比结果，发现了SFM的优势有：任意时刻都进行重分配，找到当前时刻算法认为的最优；引入了超重的考虑，当电梯已超载，则不再给此电梯分配电梯。因此，我们修改了RL算法中的mask函数，增加了对超载的考虑，以及修改了重分配机制，当只有当电梯的下一目标为该楼层时，才认为该楼层已被分配了电梯，否则，均允许重新分配电梯。我们为其修改了奖励函数，在我们生成的午餐高峰数据集上进行训练，并与SFM算法进行十次测试取平均进行对比，结果如下：

|  |  |
| --- | --- |
|  | 十次测量的平均总时间（秒） |
| SFM | 72.05 |
| RL(mask) | 71.59 |
| 纯mask | 72.82 |

表10.2 百度模拟器中不同算法在午餐高峰下的平均总时间

我们还尝试了加入能量损耗到我们的奖励函数中来引导训练，得到的测试结果如下：

|  |  |
| --- | --- |
|  | 十次测量的平均消耗能量（焦耳） |
| SFM | 2603389.86 |
| RL(mask) | 2507749.19 |

表10.3 百度模拟器中不同算法在午餐高峰下的平均总消耗能量

可以看到我们的模型在午餐高峰中有提升效果。

我们也对下行高峰的数据进行了训练，得到的结果如下：

|  |  |
| --- | --- |
|  | 十次测量的平均总时间（秒） |
| SFM | 72.63 |
| RL(mask) | 67.5 |

表10.4 百度模拟器中不同算法在下行高峰下的平均总时间

但是RL在迁移到不同人流模式方面上还有些不足，mask的强硬限制在减少无用探索、提高训练效率的同时，也过滤去一些可能更优的方案，可能会漏掉了潜在的更优的策略，我们后续打算进一步调整mask来引导RL做更广泛地探索，进而使得模型能够更灵活的迁移到不同人流模式中。另外我们在加入了能量损耗作为引导后，训练也遇到了些问题，有时候会卡住，相关原因也在探索。

## 最终框架—软规则引导的层次化强化学习

综合以上实验探索和结论，我们总结出使用强化学习算法解决电梯调度问题的四个挑战：

1. 状态空间是离散的，并且受到不稳定客流的影响，这增加了MDP中状态转移过程的不确定性。
2. 奖励函数十分稀疏，因为乘客的请求随时都可能发生，我们必须派遣一些电梯尽快响应，但乘客的等待时间和传输时间只能在指定的电梯到达乘客的源楼层和目的楼层后计算。
3. 决策中一个动作变化可能会导致后期乘客的状态出现显著差异，使RL模型更加难以收敛。
4. RL算法需要满足保证用户体验等需求的限制，因此它的探索范围应受限制。

为了克服上述挑战，我们将我们的最终框架分为三部分：状态特征提取模块，上层软规则表示层与下层决策层，如下图所示。

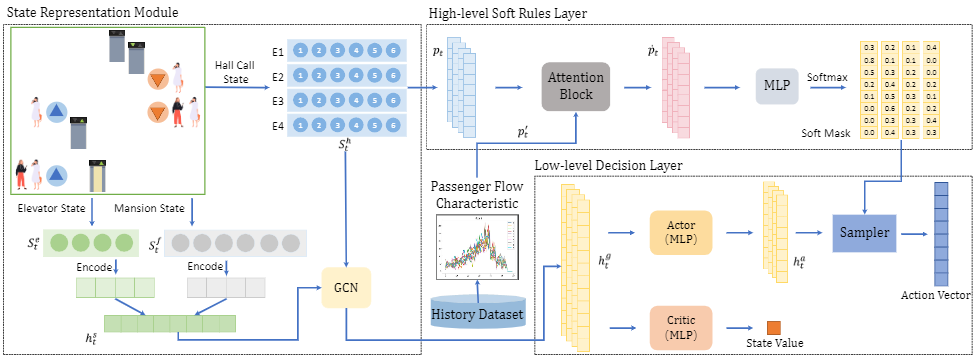


图10.1

### 状态特征提取模块

在状态特征提取模块中，为了克服挑战1，并从环境的可用变量中充分提取特征，我们把环境的状态分为三部分：电梯的状态、大楼的状态与层站召唤的分配状态。电梯的状态包括电梯的位置、速度、加速度、方向、开关门状态、载重、轿厢召唤；大楼的状态包括各楼层的等待人数；层站召唤的分配状态是一个电梯和层站召唤对应情况的邻接布尔矩阵。我们这样划分是因为电梯状态和大楼状态相对稳定且随模拟时间平稳变化，而相比之下，层站召唤的分配状态不稳定，当代理将不同的电梯重新分配给之前的层站召唤时，这个状态将发生十分陡峭不平滑的变化。我们分别用两层神经网络对电梯状态和大楼状态进行编码，并将编码结果连接成，然后与层站召唤状态一起输入到一个深度图神经网络中进行特征提取，得到特征向量。该向量表示整个环境的当前状态。实验中，我们尝试了不用这个模块来表示特征，直接把状态变量输入到决策模块中进行训练，模型最后无法收敛，这有力地验证了我们设计的状态特征提取模块的有效性。

### 上层软规则表示层

上层软规则表示层在提高RL agent的探测效率和保证不同场景下不同电梯调度目标的可靠性能方面发挥着重要作用。在不同的客流模式中，不同模式之间的主要差异在于乘客源层和目的层的分布，下图展示了上行高峰、下行高峰、午餐高峰、非高峰的t=1时各楼层到各楼层的出现人数。

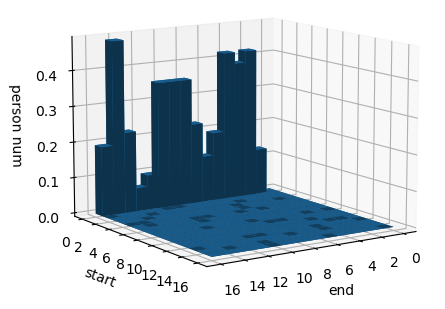
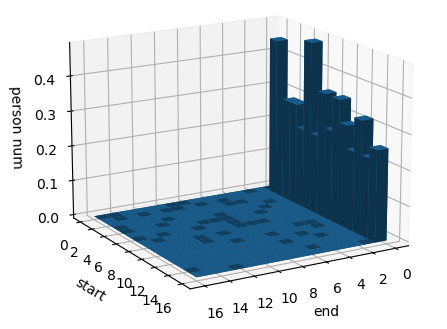
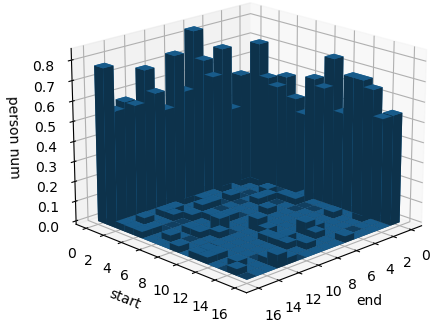
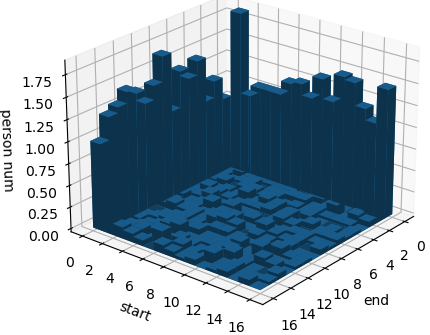
   

图10.2

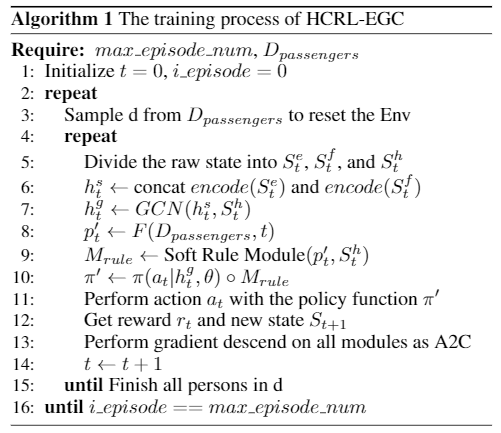
相比于在不同的模式中设计不同的奖励函数或调整超参数以实现不同的优化目标，我们创建了上层软规则层来捕获当前状态和客流特征之间的相关性。它输出针对不同模式的软规则掩码来指导底层决策层。在此之前，我们使用概率模型将统计客流数据描述为历史数据集的特征向量，该历史数据集是由Elevate 8根据不同客流模式生成的，（当然也可以从真实建筑中收集）。然后，根据当前层站召唤分配矩阵和电梯状态，计算各电梯和楼层之间的便利系数，得到表示为的潜在模式特征向量。接下来，我们将作为查询输入向量，而作为键和值输入向量，并将它们输入到一个多头注意力网络。因此，在历史客流特征和之间，我们得到了表示为的注意力系数。之后，我们将$输入到一个前馈网络，并执行Softmax函数来生成面向模式的软规则掩码，以指导决策层。请注意，软规则掩码表示特定流动模式下电梯和楼层之间的便利关系。例如，在午餐模式中，与在3楼运行的电梯相比，将2楼运行的电梯安排到一楼的大厅上行呼叫更合理。因为根据历史数据，一楼可能会有很多乘客在等待，而后面那个电梯对他们来说更方便。通过这种方式，上层软规则层的主要作用是引导RL代理更有效地探索，这与模仿学习类似。我们在后面的实验中进行了消融研究，以验证上层软规则层的有效性。

### 下层决策层

决策层用状态表示和软规则掩码给出最终调度动作向量。我们采用了优势-演员-评论家（A2C）结构来做出决策。评论家网络负责估计当前状态的值，演员网络尝试学习策略函数，以提供一个动作向量，作为电梯组在当前时间步的调度方案。然而，正如挑战4所述，Agent在决策时会存在大量无效操作或者说非法动作，例如，将电梯分配到无人楼层。此外，许多有效的行动违反常识，或者在特定的客流模式下效率低下，例如，将同一部电梯分配给两个相反方向的大厅呼叫。不幸的是，这些行动导致了RL agent进行了大量无用的探索，浪费了大量时间和资源，并且会导致性能不佳。因此，我们引入我们的上层软规则层来生成一个面向模式的规则掩码，其形状大小相同，为。最后，我们采用分类采样器，以规则掩码和的乘积的概率对动作向量（形状为）进行采样。通过这种方式，我们可以利用客流特征来指导Agent更高效地学习以及拥有更好的收敛值。

### 模型训练

框架的训练过程如下述算法所示。



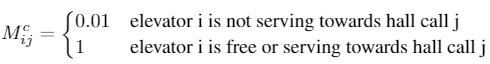
#### 训练数据来源

针对每种模式，我们先准备30个客流数据文件（表示为），每个数据文件记录所有乘客的出现时间、源楼层和目的楼层。我们先统计了这些客流数据，得到了一个记录从i层到j层的乘客出现概率随时间变化的矩阵。训练时，我们每次根据这个矩阵和模拟时间随机生成一组客流，这样做的目的是增广数据量，提高我们模型的泛化能力。

#### 训练过程

如上所述，每个episode我们都随机生成一组服从客流模式的客流数据。针对每个time step，，我们首先将来自环境的原始状态划分为电梯状态（表示为）、楼层状态（表示为）和大厅呼叫状态（表示为）。然后利用图卷积神经网络（GCN）计算当前状态的隐藏特征，并将其分别反馈给演员者网络和评论家网络。接下来，我们使用概率模型来表示时间t时的统计特征。它输出时间t时历史数据的特征向量。然后，我们使用上层软规则层生成软掩码，并将其应用于策略函数：

在上层软规则层中，我们首先利用当前电梯状态和层站召唤分配状态来计算方便系数矩阵，矩阵的大小为，其中每个元素的设置为：



然后，我们将大小为的客流特征向量缩放为与大小相同的矩阵$\mathbf{M}{p}{u t^\prime}$。接下来，我们将它们输入到注意力网络模型中并计算：

，

其中，、和是线性网络，我们使用作为查询网络的输入，使用作为键和值网络的输入。最后，我们将馈送到MLP层，并在电梯维度中执行softmax函数，得到软规则掩码。

为了优化所有乘客的平均等待时间（）和平均传输时间（）并节约能源消耗（），我们将奖励设置为、和的组合。然而，由于动作是分配电梯到层站召唤，在分配之后，到达楼层之前，并没有真正服务。换言之，该行动不是立即采取的，也不对当前的和负责，这也是我们上述的挑战2。因此，将设置为当前的和组合是不合理的。为了克服这一困境，我们计算将每个层站召唤分派的报酬计算为：

其中，是等待大厅呼叫调度的乘客人数，而是电梯中与大厅呼叫有关的乘客人数，、和是平衡、和能耗权重的系数。在这里，我们使用n-step A2C算法来训练我们的代理，n设置为128，因此回报返回值由以下公式给出：

其中，是奖励的折扣系数，由价值网络近似。A2C的优势函数计算如下：

最终损失函数是价值损失、行动损失和熵损失的组合：

其中，和是价值损失和熵损失的系数。利用损失函数，我们按照A2C算法更新演员-评论家网络和上层软规则层的参数。

## 实验

### 实验数据

为了增大数据量，我们的训练数据和测试数据均来自上述“训练数据来源”的生成方式。针对不同的人流模式，为了减小人流量变化对模型稳定性的影响，我们截取不同的相对来说比较高峰的时间段来计算概率矩阵以生成人流数据。上行高峰中截取的是30-36分钟，下行高峰中截取的是6-12分钟，午餐高峰和非高峰中截取的是0-6分钟。对于测试数据，我们设置不同的随机种子（训练时随机种子设为1，测试时随机种子设为10），生成20个6分钟的数据即总计120分钟的人流数据对模型进行测试。

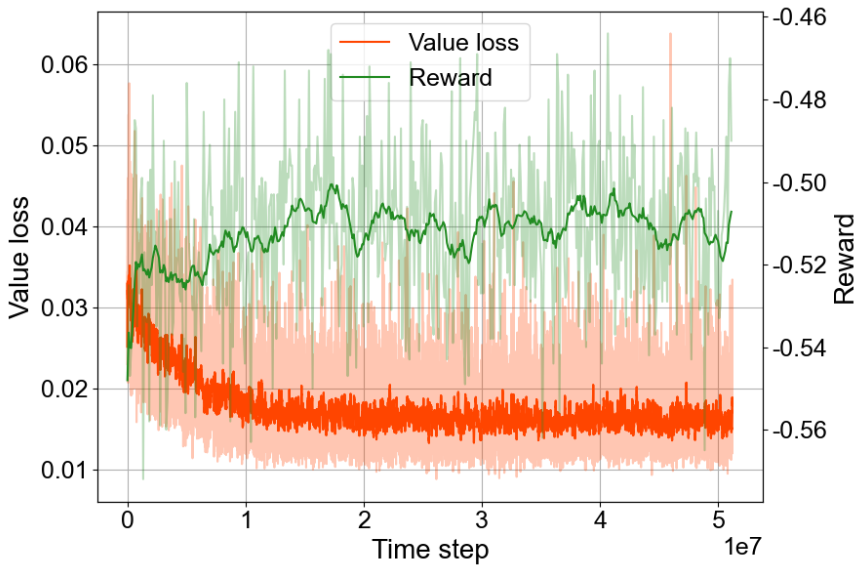
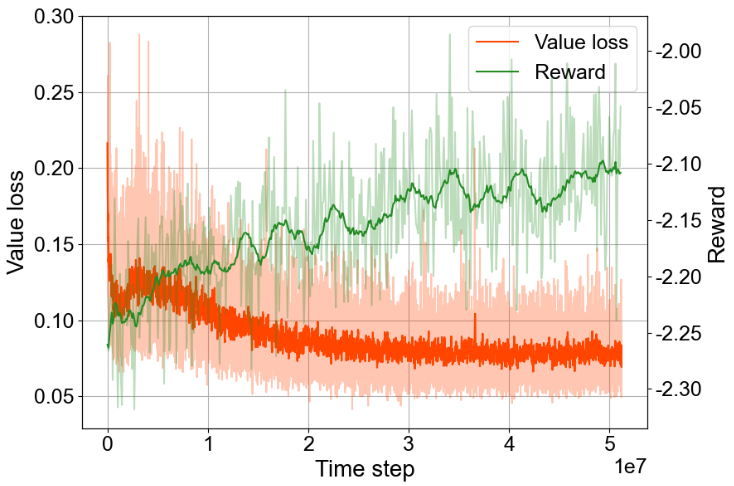
### 实验参数与设备

对于奖励函数的系数，我们将设置为0.1，设置为0.6，设置为，奖励折扣系数设置为0.99。我们选择步数为128，学习率为。对于损失函数L的系数，我们将设置为1，将设置为5。我们在Pytorh中实现了该代码，并在一台具有4个GTX 1080 GPU内核的NVIDIA虚拟机上进行训练。训练过程占用2327MB的GPU内存。完成训练过程需要将近两天的时间。

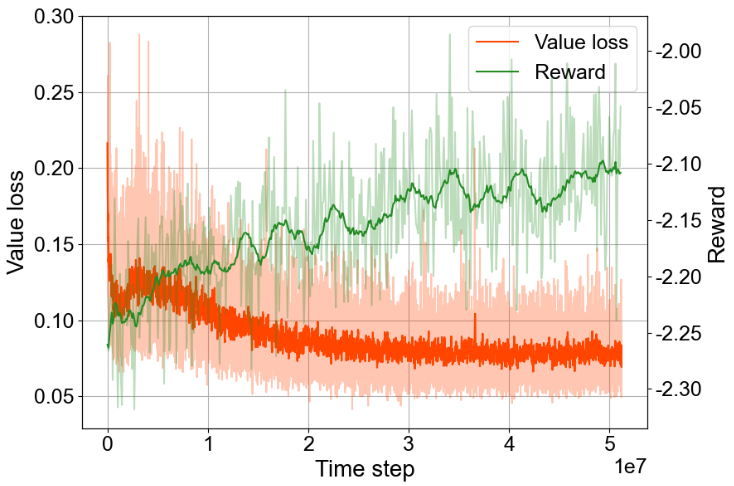
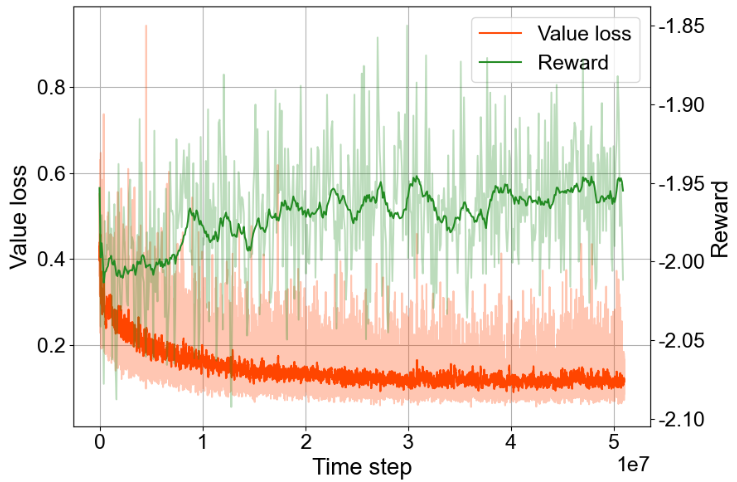
### 训练过程分析

由于我们的RL智能体的决策层中的演员-评论家网络估计当前状态的价值，并相应地生成策略，因此分析培训过程中的价值损失非常重要。此外，智能体的奖励趋势直接反映了其所学策略的效果。因此，我们每50 episode（51200个时间步）在测试数据集上运行我们的模型，并记录其在训练过程中获得的奖励。

下图展示了在各种人流模式中，模型的价值损失与奖励值随着训练过程的变化曲线。阴影曲线是真实值，实线部分是真实值的滑动平均。如图所示，各个人流模式下，值损失随着训练过程而下降，经过近20000K的时间步长更新后，逐渐收敛到一个相对较小的值。随着训练过程的前两万个时间步，奖励函数大大提升，后期增长速度减慢。这是因为代理在早期阶段学习优化平均等待时间和能耗的策略。而在后期训练阶段，随着平均和时间接近峰值，它会继续优化能耗，并保持平均完成时间不变。因此，我们的模型最终可以学习一种节能高效的策略，平衡用户体验和能源消耗。

上行高峰 下行高峰

午餐高峰 非高峰

图10.3

### 与其他算法对比

我们在四种人流模式中对不同的算法进行了比较。为了比较的公平性，我们比较不同算法所使用的客流数据为同样的数据，即先根据客流模式的生成概率矩阵生成4个客流模式的4组客流数据，然后用这4组客流数据分别测试不同算法的性能。对比的算法包括传统算法的代表——ETA，基于子模块优化的算法SFM（参考文章《Submodular Function Maximization for Group Elevator Scheduling》），基于Q-learning的方法Robert（参考文章《Elevator group control using multiple reinforcement learning agents》），基于A3C的方法DRL-EGC（参考文章《Optimal Elevator Group Control via Deep Asynchronous Actor–Critic Learning》）。对比结果如下表所示，其中AWT为平均等待时间，ACT为平均完成时间（即平均等待时间+平均运输时间）。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 方法 | 午餐高峰 | |  | 上行高峰 | | 下行高峰 | | 非高峰 | |
| AWT | ACT |  | AWT | ACT | AWT | ACT | AWT | ACT |
| ETA | 34.01 | 93.05 |  | 49.95 | 122.14 | 25.62 | 65.79 | 53.30 | 119.71 |
| SFM | 38.37 | 90.12 |  | 48.53 | 119.27 | 35.90 | 70.63 | 52.78 | 111.89 |
| Robert | - | 159.31 |  | - | 126.70 | - | 177.42 | - | 164.17 |
| DRL | - | 112.93 |  | - | 123.60 | - | 82.21 | - | 140.42 |
| 我们的 | 32.79 | 89.36 |  | 43.81 | 115.37 | 30.21 | 72.99 | 52.62 | 117.15 |

表10.5

可以看到，与其他基于学习的方法相比，我们的模型在各个客流模式中都大幅领先，而与传统方法和基于优化的方法比，我们的算法在午餐高峰、上行高峰、非高峰数据集上有一定程度上的优化，但在下行高峰数据集上的表现不如ETA算法。另外在算法的运行速度上，传统算法>我们的算法>Robert>DRL> SFM，无论当前需要分配多少个层站召唤，基于RL的算法都是常量时间的，而SFM和其他基于搜索的算法会随着待分配的层站召唤的增多而增加，可见我们的算法在运行速度和吞吐量上也有一定的优势。

### 消融实验

我们在午餐高峰数据集上对我们框架的各个模块做了消融实验来验证各个模块的作用。实验结果如下表所示：

|  |  |  |  |
| --- | --- | --- | --- |
| 智能体 | 平均等待时间 | 平均完成时间 | 总耗能 |
| 不含GCN和软规则层 | 111.73 | 173.55 | 1658499 |
| 不含软规则层 | 65.44 | 128.02 | 1528082 |
| 只含软规则层 | 35.13 | 91.08 | 1965487 |
| 完整版本 | 32.79 | 89.36 | 1946399 |

表10.6

从实验结果我们可以看到，我们在状态表示模块中使用的图形神经网络可以充分地从状态变量中提取丰富的特征。高级软规则层在不了解目标的情况下，仍能提供有保证且可接受的性能。通过结合软规则和RL的优点，我们的框架可以继承软规则的可控性和可靠性，提高RL的探索效率，使得RL模型能够收敛到一个更好的值，获得更好的优化。

### 结论

RL在电梯调度问题上的优势有：

1. 能够更好进行多目标优化，如等候时间、运输时间和电梯能耗的综合优化。
2. 与传统方法相比，不需要复杂的启发函数，调参工作也相对更加简单。
3. 能够针对不同的人流特征，学习出不同的策略，在变化不复杂的场景下效果比传统方法有一定的提升。

劣势有：

1. 在给定的调度规则下，RL能够进行的优化手段有限，优化空间较为扁平；在高峰期下，电梯的运动模式相对固定，可供优化的空间较小。
2. 不同人流模式差异大，导致RL的学习环境更加复杂，难以学出一个普适的优秀策略。
3. 环境提供的信息有限，对RL来说，信息越丰富越容易收敛，不完全信息会让RL的优势无法发挥。
4. RL模型依赖于环境，它在与模型训练环境相同的环境下表现最佳。而在其他不同的环境中，如当电梯运行方式、人流模式等与训练环境差异较大时，难以表现出其最优性能。

# 十一. 目前工作完成情况

## 第一阶段：九月

我们首先基于百度模拟器进行上行高峰中RL模型的训练。在模型训练中，我们提出了多种优化方案，包括：修改RL训练过程中更新状态的单位时间间隔，并加入动作延迟以合并多步动作；加入了用于引导训练的电梯mask，并加入动作mask，以忽略训练过程中的无效动作；每一时刻检查电梯分配状态、允许通过重分配对调度策略进行优化等。我们在百度模拟器中对RL模型和最短距离算法进行了性能测试。在测试中，RL模型的表现优于最短距离算法，测试结果如表1所示。但由于百度模拟器没有与Elevate对齐，RL模型的平均等待时间与三菱调度算法在Elevate的结果仍有一定的出入。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 最短距离算法 | RL | 三菱在Elevate的结果 |
| 平均等待时间/s | 78.99 | 72.40 | 60.90 |

表11.1 不同算法在上行高峰下的平均等待时间

我们之后的工作聚焦于如何让百度模拟器向三菱Elevate对齐。我们调整了百度模拟器的环境运行参数，让其环境与Elevate的运行环境尽可能对齐。由于RL模型的性能依赖于特定的环境，因此每次对百度模拟器进行改动后，都需要对模型进行调整，并重新进行模型的训练。此外，我们也基于训练效果的需要，对百度模拟器进行了优化。例如，我们修改了模拟器中已被分配的电梯在服务hallcall时其按钮的状态，使得电梯的运行状态更加贴合实际情况，这也为训练RL提供了便利。

我们也尝试在百度模拟器上复现了三菱调度算法。我们在模拟器中依据三菱算法的调度结果，实施相应的调度操作，重现了调度场景。但三菱算法在百度模拟器上的复现结果与在Elevate上的结果之间有着不可调和的偏差，因此在百度模拟器上的复现的结果无法直接与RL进行比较。

## 第二阶段：十月

为了验证不同人流数据对RL训练过程的影响，我们在上行高峰中进行了更加细致的对比研究实验，并得出了以下结论：第一，在人流量过大时，由于所有电梯的工作模式都相对固定，可供优化的空间较小；第二，人流量的波动过大，会使强化学习的环境变得更加复杂，导致模型训练中的损失过大，难以学习出一个优秀的策略。因此，RL的适用条件为：人流量和人流量的波动不能太大。

之后，我们提出了针对空闲电梯进行优化的双模型结构，通过命令空闲电梯主动前往某一楼层以进行优化。双模型结构包括一个为hallcall分配电梯的RL模型，和一个为空闲电梯分配parkcall楼层的模型。但是实际训练过程我们发现这样的结构训练非常不稳定，难以收敛，优化效果并不明显。且与三菱沟通后，认为这一策略在实际应用中并不可取，因此我们没有继续研究这一优化方案。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 等待时间/s | 运输时间/s | 运行环境 |
| 最短距离 | 73.2 | 49.6 | 百度模拟器 |
| 随机算法 | 46.3 | 47.5 | 百度模拟器 |
| 校准三菱算法结果后的估计值 | 62.7 | 51.9 | 百度模拟器 |
| RL | 33.1 | 46.9 | 百度模拟器 |

表11.2 百度模拟器中不同算法在上行高峰下的平均等待时间和平均运输时间

我们也提出了一种基于人流预测的模型方案，方案包括两个阶段：第一，预测下一时刻各楼层的出现人数；第二，在所做预测的基础上，计算电梯的最佳停靠方案。对于第一部分，我们首先使用LSTM网络进行人流情况的预测。但由于人流数据是按照某一概率随机生成的，数据随机性较大，且不同数据间存在偏差，导致LSTM网络的预测结果不准确。因此，我们改用概率模型来预测各楼层的出现人数，并得到了较好的预测结果，在进一步研究后得出结论：可以认为人流服从某一特定的概率分布。而在第二部分中，我们尝试使用神经网络来预测停靠楼层，并结合RL决策网络进行训练。但由于在训练过程中不能有效地区分预测网络和决策网络这两个模型的奖励，无法进行有效训练。

为了让RL模型在Elevate上运行，我们也开始开发调用RL算法的DLL。

## 第三阶段：十一月

我们针对上行高峰的10-20分钟数据进行训练，并对比了使用电梯mask协助的RL模型和不使用mask协助的RL模型。我们发现，不使用mask的RL模型的学习曲线在形状上符合一般RL训练的指数型下降曲线，但其收敛结果很差；而对于加入了mask的RL模型而言，其学习曲线的形状又十分反常。这说明，在上行高峰场景下，RL的提升空间有限，进行优化更需要依靠规则。

我们也在RL中加入了蒙特卡洛树搜索，以优化RL算法的表现效果。我们在上行高峰数据中截取了十分钟的人流数据来进行蒙特卡洛树搜索。加入了蒙特卡洛树搜索的RL算法，与最短路径算法相比，在性能上有10%的提升。然而，由于蒙特卡洛树搜索需要获知未来的人流情况，而Elevate中并没有提供获取这一信息的接口，因此我们只能放弃这个方向。

在开发DLL的工作中，我们实现了一个通用的python和C++通讯的调用算法，解决了Elevate版本过旧不能使用64位pytorch的问题，实现了在Elevate调用百度模拟器训练的RL模型的目标。但由于环境无法对齐，以及Elevate提供的环境信息太少（如：各楼层已到达而尚未被服务的人的信息），模型在Elevate环境下欠拟合，运行效果很差。

## 第四阶段：十二月

我们实现了针对午餐高峰的RL算法，在午餐高峰数据集上训练RL。与上行高峰数据集相比，我们认为午餐高峰数据集更加具有优化空间，因为大楼中hallcall的分布更加均匀。各算法在午餐高峰下的运行结果如表3所示。从表中可以看出，在午餐高峰下，RL的优化效果有限，进行优化更多也需要依靠规则。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 平均等待时间/s | 平均运输时间/s | 运行环境 |
| RL | 56.8 | 72.3 | python模拟器 |
| pure mask | 60.9 | 68.6 | python模拟器 |
| 最短距离算法 | 85.0 | 80.1 | python模拟器 |

表11.3 百度模拟器中不同算法在午餐高峰下的平均等待时间和平均运输时间

由于环境的限制过多，我们加入了搜索算法和贪心算法来作为RL的辅助。算法不考虑未来的人流情况，只针对当前大楼和电梯的状态求当前的最优解；而每当有新的hallcall产生时，都对所有hallcall进行重分配。我们首先实现了用遗传算法求近似解的方案。但由于遗传算法每次重分配耗时太久，因此我们转而使用局部搜索算法，并在局部搜索算法上进行改进。为了利用人流模式特征，我们在局部搜索算法评估一种分配方案的优劣时，为每个楼层添加了不同的权重。我们在Elevate上对这一算法进行了测试，发现其表现效果优于传统的ETA算法。经过进一步调整权重，我们发现权重的指派对运行结果影响很大，于是尝试使用神经网络和RL来学习这个权重，但学习效果并不理想。

我们也实现了在给定分配方案下的完成时间预测模型。我们首先设置数据集的结构为：输入为当前电梯分配方案，预测目标为为完成这个方案时乘客预计的平均等待时间。利用这一数据集，我们通过监督学习来训练神经网络。但由于输入过于离散，在尝试了很多网络结构和输入的表示方式之后，网络收敛的loss仍然很高，无法进行准确的预测。之后，我们参考了一篇使用最优化方法的论文，修改了数据集的结构，将输入更改为电梯在各个楼层的停靠概率，而将预测目标更改为电梯预计到达各个楼层的时间。模型通过监督学习进行训练，训练的收敛效果很好，得到的模型能够在给定的分配方案下，较为准确预测完成时间，而这一预测结果可以结合贪心算法来进行决策。但这一方案也存在着问题，因为它在一定程度上偏离了通过RL进行决策的研究方向。

## 第五阶段：一月

我们复现了众多传统算法，基于优化的算法，并结合基于优化的方法的优点改进我们的模型。

我们重新设计了客流数据的生成和导入方式，从而构造更大量的数据供RL进行学习和探索。

我们在午餐高峰和下行高峰数据集中分别训练了模型，验证了我们模型的性能，同时也探究了在午餐高峰模式下训练的模型在下行高峰数据集中测试的性能表现，发现其泛化能力还有待提高。

另外，我们也在根据文章《Optimal Elevator Group Control via Deep Asynchronous Actor–Critic Learning》复现百度电梯大赛中表现较优异的模型来作比较。

# 十二. 工作计划

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **任务排序** | **任务名称** | **预估工期** | **依赖关系** | **开工时间** | **完工时间** | **任务完成情况** |
|  | **三菱项目基于深度学习的电梯调度算法研究** | **28 天** |  | 11/22/21 | 12/28/21 |  |
| 任务1 | 通过经典概率模型、神经网络等方法对人流情况进行预测 | 7 天 |  | 11/22/21 | 11/29/21 | 已探索 |
| 任务 2 | 结合任务1的预测结果和大楼及电梯的当前状态，实现针对各人流模式的搜索算法，使其能够针对不同人流模式求解出预测的最优解 | 14 天 | 任务1 | 11/30/21 | 12/17/21 | 已探索 |
| 任务3 | 结合任务1的预测结果和大楼及电梯的当前状态，训练针对各人流模式下不同人流量的强化学习模型，使其能够针对不同人流量均能学习到最优的策略 | 7 天 | 任务 1 | 11/30/21 | 12/8/21 | 已探索 |
| 任务4 | 将任务2的搜索算法模型和任务3的强化学习模型相结合，使其应对不同人流模式、不同人流量时均能有最优的表现 | 7 天 | 任务 2/任务3 | 12/20/21 | 12/28/21 | 已探索 |
| 任务5 | 目前权重的计算还比较朴素，历史人流数据的利用也比较简单，后续我们打算结合RL来训练出更精确的层站召唤对应的权重，从而提升搜索时预测的损失值，从而得到更好的搜索结果。 | 14 天 | 任务4 | 12/28/21 | 1/11/22 | 已探索 |
| 任务6 | 将RL的mask改成更加柔性的规则参数，增加RL的探索范围，提高性能。 | 7天 |  | 1/28/22 | 2/7/22 |  |
| 任务7 | 加入神经网络来学习历史人流信息与当前人流信息的特征，提高RL的泛化能力，使其能用同一个模型参数处理不同人流模式。 | 7天 |  | 2/7/22 | 2/14/22 |  |

表12.1