浙江工业大学

本科毕业设计（论文、创作）

题目：基于机器学习的多点温控算法研究

作者姓名 汪咏

指导教师 赵云波教授

专业班级 自动化 1402 班

学　　院 信息工程学院

提交日期 2018 年 6 月 1 日

Dissertation Submitted to Zhejiang University of Technology
for the Degree of Bachelor

Research on Multi-point Temperature Control Algorithm Based on Machine Learning

Student: Wang Yong

Advisor: Professor Zhao Yunbo

College of Information Engineering
Zhejiang University of Technology

June 2018

浙江工业大学

本科生毕业设计(论文、创作)诚信承诺书

本人慎重承诺和声明：

1. 本人在毕业设计（论文、创作）撰写过程中，严格遵守学校有关规定，恪守学术规范，所呈交的毕业设计（论文、创作）是在指导教师指导下独立完成的；

2. 毕业设计（论文、创作）中无抄袭、剽窃或不正当引用他人学术观点、思想和学术成果，无虚构、篡改试验结果、统计资料、伪造数据和运算程序等情况；

3. 若有违反学术纪律的行为，本人愿意承担一切责任，并接受学校按有关规定给予的处理。

学生（签名）：汪咏

2018 年 6 月 1 日

浙 江 工 业 大 学

本科生毕业设计（论文、创作）任务书

专 业 自动化 班 级 自动化 1402 学生姓名/学号 汪咏/201403080423

一、设计（论文、创作）题目：

基于机器学习的多点温控算法研究

二、主要任务与目标：

1. 对多源多点温控系统提出合理数学模型；

2. 提出有效的多源多点温控系统的设计算法；

3. 对算法进行分析和实现。

三、主要内容与基本要求：

1. 阅读相关文献，了解该领域基本研究现状；
2. 研究温控系统的相关控制算法；
3. 提出并验证相关的温控算法；
4. 撰写毕业论文。

四、计划进度：

2018 开学前 收集相关资料文献，学习相关知识，完成外文翻译、文献综述；熟悉课题，做好开题准备。 第 1-3 周 完成开题报告，参加开题交流 第 4-8 周 提出算法并作验证等工作，接受中期检查 第 9-14 周 进行算法设计和实现等工作。撰写毕业论文初稿 第 15 周 论文修改，毕业答辩，提交相关文档资料

五、主要参考文献：

1. 《自然通风动态温度调控的智能气动开窗系统》， 司慧萍，浙江大学，博士论文，2015 2. 《智能温度控制系统》，刘美霞，南京航空航天大学，硕士论文，2003 3. 《工业无线温度控制系统设计》，马俊卿，东华大学，硕士论文，2017

任务书下发日期 2017 年 12 月 20 日

设计（论文、创作）工作自 2017 年 12 月 20 日 至 2018 年 6 月 4 日

设计（论文、创作）指导教师 赵云波

系主任（专业负责人） 杨马英

主管院长 张有兵

基于机器学习的多点温控算法研究

摘　要

随着人们生活质量和工业生产需求的增长，对温度控制的要求也随之提高，同时伴随着的是温度控制算法的发展。传统的温度控制算法多针对单点的温度控制，很少对一个空间中强耦合的多点的温度进行控制或者控制效果不佳,无法满足一个空间中不同人群对温度的不同要求。而近些年来，随着机器学习技术的兴起和其在其他领域非线性的优异性能表现，如何将机器学习技术与多点的温度控制进行有效的结合成了一大研究热点。

本文的研究内容是基于机器学习的多点温控算法研究，主要针对温控算法的研究，实现在多台空调作用下对一个空间中多个点的温度控制。利用 Ansys 对实验环境的建模，温度变化过程的仿真，采集各点稳定温度数据和基于机器学习的温度控制算法对数据进行处理，并对各点未知温度需求进行空调设定值的预测，在 Ansys 中完成仿真验证。通过比较不同的机器学习算法对温度预测性能的差异，选择最优秀的控制算法，以达到最好的控制效果。论文的主要工作如下：

1. 综述论文研究背景及意义，介绍温度控制算法的研究现状和描述现在使用比较频繁的几种温度控制算法，了解它们的适用环境。

2. 利用 Ansys 对实验室真实环境进行建模和对温度变化过程的仿真，采集在不同的空调温度设定值的作用下各采样点的温度数据，设计机器学习算法对数据进行处理，并对各点未知温度需求进行空调设定值的预测，在 Ansys 中完成仿真验证。

3. 测试不同的机器学习算法的性能差异，对设计的算法进行改进并选择最好的控制参数，以达到最好的控制。

4. 针对控制算法，设计用户界面程序，转为 windows 系统可执行程序，方便算法的移植和用户操作。

关键词：温度控制、Ansys 仿真、机器学习算法、多输入多输出、界面设计

RESEARCH ON MULTI-POINT TEMPERATURE CONTROL ALGORITHM BASED ON MACHINE LEARNING

ABSTRACT

With the improvement of people's quality of life and the demand for industrial production, the demand for temperature control also continues to increase, accompanied by the development of temperature control algorithms. The traditional temperature control algorithms are mostly aimed at single-point temperature control, it is rare to control the temperature of multiple points that are strongly coupled in one space or poor control effect, and it can't meet different temperature requirements for different people in a space. In recent years, with the rise of machine learning technology and its excellent nonlinear performance in other fields, how to combine machine learning technology with temperature control effectively has become a hot spot of research.

The research content of this paper is the research on multi-point temperature control algorithm based on machine learning. It mainly focuses on the research of temperature control algorithm. It can realize the temperature control of multiple points in a space under the effect of multiple air conditioners. Using Ansys to model the environment of experiment, simulate the process of temperature change, collect the stable temperature data of each point and the temperature control algorithm based on machine learning to process the data and predict the set value of air conditioning for unknown temperature demand of each point, and simulated in Ansys. By comparing the difference in performance of temperature prediction between different machine learning algorithms, the best control algorithm is selected to achieve the best control effect. The main work of the dissertation is as follows:

1. Overview the background and significance of the research, as well as the current research status of temperature control algorithms, introduced several temperature control

algorithms that are currently used more frequently and understand their applicable environment.

2. Use Ansys to model the real environment of the laboratory and simulate the process of temperature change, collect the temperature data of each sampling point under different set values of air temperature, and design the machine learning algorithm to process the data. Predict the set value of air conditioning for unknown temperature demand of each point and simulated in Ansys.

3. Test the performance difference of different machine learning algorithms, and the design algorithm is improved and choose the best control parameters to achieve the best control.

4. For the control algorithm, design the user interface program and convert to windows system executable program to facilitate the algorithm migration and user operations.

Key Words: temperature control, ansys simulation, machine learning algorithm, multiple-input multiple-output, interface design

目　录

第 1 章　绪　论

1.1　课题研究背景及意义

温度与人们生活息息相关。一年四季的变化带给人们最直观的感受就是温度的改变，进而反映到人们的日常就是生活习惯的改变和衣物的增减。正常人类只能在一定的温度条件下生活，温度过高或过低都会影响人类的生存。

温度也与社会生产有关。温度的改变会引发相应的物理和化学变化，比如物体形态的改变，水在 0℃以下是固态，100℃以上为气态，其余为液态。大部分的生产活动需要在一定的温度条件下进行，合适的温度可以使化学反应得以快速进行，而在不同的温度下，获取的最终产物可能会不同。比如乙醇与浓 H_2SO_4 反应，170℃会生成乙烯，而在 140℃则生成乙醚，食盐跟浓 H_2SO_4 微热时生成 $NaHSO_4$，强热时生成 Na_2SO[1]。

由于温度与各个方面都有联系，因此就需要对温度进行精准的控制。温度的控制是随着控制理论的发展而发展的，从古典控制理论到现代控制理论都有控制算法的发展。温度控制按照调节原理进行分类，有位式控制、PID 控制、模糊控制等[2]。温度控制算法大致分为位式控制、简单 PID 及其变式、模糊控制算法、遗传算法，人工神经网络和机器学习算法几种。传统控制的主要特征是基于模型的控制，难以解决复杂对象的控制问题[3]。

针对空调温度控制系统，传统的温度控制主要分为两种：一是单点的温度控制，其温度传感器与控制器相连，最终调节的温度是空调附近的温度，而不是人体附近的温度，不一定满足人们的精细需求。同时一旦室内温度发生突变，由于温度在空气中的传导很慢，导致空调的温度调节会很慢；二是多点的温度控制，但各个点之间是相互独立的，没有考虑到各点之间的相互影响，导致室内的多台空调风机无法进行整体上的规划调节，引起能源上的不必要消耗。国际能源机构的数据显示，住宅和商业建筑的能源消耗约占欧盟最终能源消耗的 40%和二氧化碳排放总量的 36%[4]。这部分消耗的能源大约一半用于供暖/制冷，通风和空调，发达国家这一用量每年增加 0.5-5%[5]。

另外各人对温度的感受不同，在一人觉得热的时候，另一人则觉得刚好。比

如老年人比较怕冷，而青年人比较怕热；又如肥胖和患有皮肤病等特定疾病的人对温度比较敏感，当他们与一般人处于同一区域时，两者对温度的要求就会产生冲突，这就涉及到了一个空间中多点的温度调节，这在传统的温度控制中是没有讨论到的。本课题将就一个空间中多点的温度控制提出控制方案，并对其进行仿真实验和算法设计。

接下来就现在使用比较频繁的几种温度控制算法进行具体的描述。

1.2 温度控制算法和研究现状介绍

温度控制系统属于控制系统，只是其控制对象不同而已，因此其具有控制系统的基本特征。一个合适精准的温度控制系统必须是闭环控制系统，其由测量变送元件，控制器，执行器等组成。其控制流程图可以由图 1-1 表示：

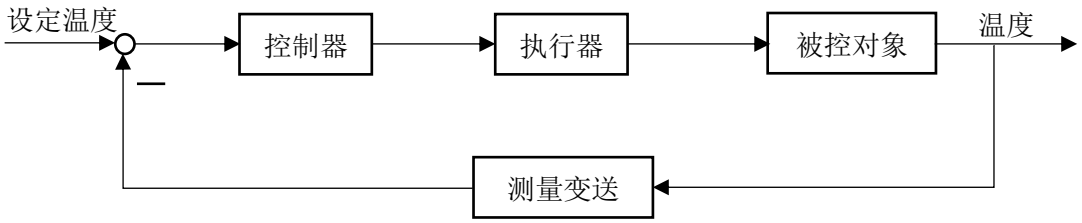


图 1-1 温度控制系统基本控制流程图

温度控制系统包含的各个部分都是非常重要的，测量变送环节提供一个准确的反馈值，执行器将控制算法得到的结果准确输出，带动被控对象的执行，最终进行正确的控制。随着检测技术的发展，出现了一批新型测温仪表，有多点输入式智能温度变送器、光纤辐射温度计、声表面波温度传感器、温度集成传感器[6]。

在确保一个完全正确快速的测量变送和执行器的前提下，一个简单的控制算法就可以解决温度的调节问题。但是实际的控制系统总是存在许多的问题，比如测量变送和执行的时延，执行机构存在误差等，如何在有误差的环节下进行精确的控制才是温度控制的最大问题，这时候一个优秀的控制算法就显得尤为重要。接下来开始阐述适用于不同环境的几种温度控制算法。

1.2.1 位式控制

位式控制是最简单的温度控制算法，当测量温度低于设定温度时，打开加热开关，使温度上升；反之，则打开制冷开关，使温度下降。这与人们的想法是一致的，日常生活中当水温高时掺入冷水，水温低时掺入热水，最终调节到适合的温度。可以用图 1-2 进行表示：

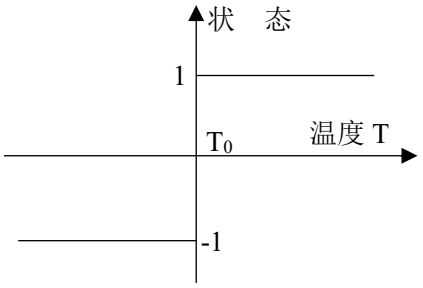


图 1-2 位式控制表示，-1 为开关开通状态，1 为关闭状态，T_0 为设定温度。

位式控制可以分为双位控制和多位控制，双位控制又可以分为带死区和不带死区，而多位控制以三位控制居多，图 1-2 显示的为不带死区的双位控制。带死区的双位控制在图 1-3 中显示，当温度高于 Tmax 时，关闭加热开关，当温度低于 Tmin 时，打开加热开关。由于带有死区，使得加热开关的启动频率降低，可以提高开关的寿命，也可以改善震荡。双位控制系统的输出是一个等幅振荡过程[7]，也就是其调节过程总是存在误差，无法到达一个无稳态误差的状态，因此位式控制的调节效果并不是很好。但是因为其结构简单，在程序上易于实现，常用于对温度要求不高的系统中，例如电烘箱，管式加热炉等。显然，位式控制并不适合需要精确控制的空调温度调节系统。

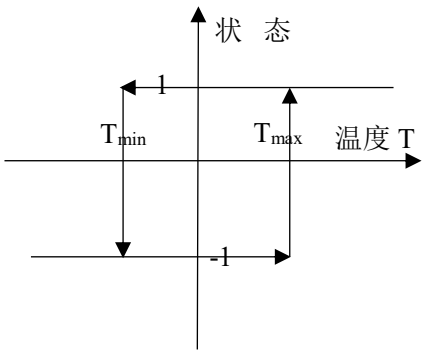


图 1-3 带死区的双位控制

1.2.2 简单 PID 及其变式

(1) PID 基础知识

PID 控制俗称为比例积分微分控制，是一种将过去的信息运用到现今控制的算法，由于其算法简单，鲁棒性高，被广泛应用与过程控制和运动控制。在目前所有的控制系统中，PID 算法约占 80%-90%[8]。PID 算法自从上世纪出现到现在仍然活跃在控制领域，足以体现出它的强大生命力和适应力。一般模拟系统 PID 的表达式可以表示为：

$$u(t) = K_P[e(t) + \frac{1}{T_I}\int_0^t e(t) + T_D\frac{de(t)}{dt}] \tag{1-1}$$

其中 K_P, T_I, T_D 分别为比例增益，积分时间和微分时间，是对调节效果进行改进的参数。而在离散控制系统中，上述表达式则变成了：

$$u_n = K_P[e_n + \frac{T_0}{T_I}\sum_{i=1}^{n} e_i + T_D\frac{e_n - e_{n-1}}{T_0}] \tag{1-2}$$

其中 u_n 为离散系统的第 n 次输出，T_0 为调节周期，e_i 为第 i 次设定值与实际测到值的误差。上述的表达式被称为位置式 PID 表达式，位置式的 PID 算法会将历史上所有的误差进行累计，会占用大量的内存空间，为了解决这个问题，将(1-2)-(1-3)得到增量式的 PID 表达式(1-4)：

$$u_{n\text{-}1} = K_P[e_{n\text{-}1} + \frac{T_0}{T_I}\sum_{i=1}^{n\text{-}1} e_i + T_D\frac{e_{n\text{-}1} - e_{n-2}}{T_0}] \tag{1-3}$$

$$u_n - u_{n-1} = K_P[(e_n - e_{n-1}) + \frac{T_0}{T_I}e_n + T_D\frac{e_n - 2e_{n-1} + e_{n-2}}{T_0}] \tag{1-4}$$

可以看出系统的控制输出只跟前两次和当前一次的误差有关，这样的话就不需要将之前的误差全部带入，进而可以起到节约内存的作用。同时由于需要计算的是增量，参数的调节也变得比较简单，但也导致参数的意义不明。还有就是只考虑了三次的计算，当系统出现可修复故障时，可以快速适应，而故障的数据不会持续影响到系统的输出。

由位置式的 PID 表达式中可以分析出 PID 参数的作用。比例参数跟偏差有关，当比例参数不变时，系统偏离设定值越大时，系统的输出值也越大，也就是

说比例参数可以快速克服偏差的影响。只有比例参数的控制是存在静差的，被控量无法恒定达到设定值。而积分时间跟之前所有的误差有关，只要存在误差，积分部分就会一直累计至系统没有误差，系统保持稳定，因此积分控制可以消除静差。但是也同样是因为引入了系统之前全部的误差，会导致累加的值过大而引起超调，使得系统无法稳定，这样的问题可以通过 PID 的变式来解决。微分时间与误差的变化量有关，误差(带有符号)变小时，微分部分为负，误差变大时，微分部分为正，也就是说微分对系统的变化量有抑制作用。误差的变化量同时体现了误差的变化方向，微分时间可以说是对误差变化方向进行调节，实现提前控制。对微分作用打个比方，我们向一个大桶中注水，刚开始以某一速度加入，后发现还可以以更快的速度加入，当加到一半时候后发现以当前速度加水的话水必定会漏出来了，就慢慢减慢速度，使得水刚好满。微分可以对超调进行抑制，但是微分对干扰的适应差，太强的微分作用会使系统不稳定。

通常采用 PI，PD，PID 控制，这个依照系统的特性来分别选用，PI，PID 控制可以实现无静差控制，PD，PID 控制可以克服滞后效应。选用的算法越复杂，需要调节的参数就越多，PID 的参数调节也是一门学问，这里就不具体阐述了，可以查看文献浅析 PID 参数整定[9]和 PID 参数整定技术的研究及应用[10]。

接下来简单介绍一些 PID 的变式，以适应不同的控制环境。

(2) PID 变式

针对积分项存在的问题，有积分分离 PID 控制算法和抗积分饱和 PID 控制算法。当系统出现一个很大的干扰，积分会导致系统产生超大的超调量而难以进行调节，积分分离算法则可以解决这样问题。在系统出现大于某一阈值的干扰时，停止积分作用，只进行 P，PD 控制，其他的误差则为普通的 PI，PID 控制。当系统的输出值达到最大而积分项还在往输出值继续增大的方向累计，执行结构已经没有无法进行反应的现象称为积分饱和。积分饱和会对执行结构有消耗和被控对象有不好的影响，可以使用抗积分饱和算法来解决问题。抗积分饱和算法就是当系统输出到达极限和积分项还在累积时，将此时的输出值限幅同时积分不进行累积。

针对微分项有不完全微分和微分先行 PID 控制算法。由于微分项对干扰比较敏感，而不完全微分的表达式为(1-5)[8]：

$$u_D(k) = K_D(1-\alpha)(e(k)-e(k-1)) + \alpha u_D(k-1) \tag{1-5}$$

可以看出每次的微分项是跟上一次的微分项有关的，本次的微分所占比例则通过 K_D,α 参数进行调节，这样的话就减缓了这次的误差变化的干扰。微分先行则针对设定值的频繁变化导致微分项的变动，不对误差进行微分，而是对输出值进行微分。

PID 控制适合于大部分的温控系统，对干扰的适应性也很强，但是其参数的调节仍然是一个问题，所以现阶段使用最多的还是对单点和多个独立点的温度控制。还有就是对多点的温度调节必定需要多台空调控制器，由于多控制点之间相互联系的强非线性，无法明确各点的控制器归属。有文献提出了基于单神经元的 PID 控制，尝试解决这些问题[11]。

1.2.3 模糊控制

考虑到温度控制系统动态特性非常复杂，很难得到一个精确的控制模型，运用传统的控制模型不一定可以取得好的控制效果，可以尝试模糊控制。模糊控制类似于专家系统，将人的丰富经验用自然语言表达和建立计算机处理的输入输出模型[12]。简单而言模糊控制就是将人们日常中的模糊话语，比如很大、比较大、中等、比较小、很小等运用到控制中，由于将具体的转为模糊，对控制模型的精确要求不高。

模糊控制的过程大致分为模糊化、模糊推理、清晰化等步骤。模糊化就是具体的控制输入量转为上述提到的模糊话语，而模糊推理则是对模糊话语进行相关判断，得到相应的控制变量，清晰化则为将控制变量转为具体的控制参数，实现对控制对象的控制。控制过程可以用图 1-4 表示。

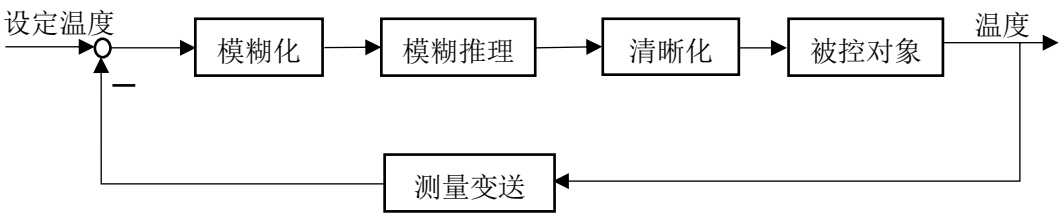


图 1-4　模糊控制系统框图

但是模糊控制算法在实际的成熟性还是不够的，没有像 PID 算法应用的那么广泛，模糊控制无法定义控制目标，其选择参数多采用试凑法，无法快速得到控制良好的参数，因此对复杂系统的控制的实现有一定的难度[13]。所以在实际应用中常将其与其他的控制算法结合使用，比如模糊 PID 控制器，神经网络模糊控制，遗传算法模糊控制。

1.2.4 遗传算法

这里对遗传算法进行简单的阐述，具体的可以参考其他的文献。遗传算法正如其名，是模仿达尔文的生物进化论，“物尽天择，适者生存”，生物通过遗传，变异，自然选择等实现进化。每一个物种都有自己的遗传信息，每一代的遗传信息与上一代的遗传信息总是不同的，都有发生一定的变异以适应这个环境，适应性差的就会被环境所淘汰，适应性强就会继续生存，进而继续进化。而遗传算法同样有模仿了这种优化功能，通过一代一代的进化，直到达到要求。遗传算法起始给定一个随机染色体，将其置于需要求解的问题中，给定环境适应评价准则，通过遗传，变异进行进化，找到适应环境最好的染色体，既为所求问题的解。

遗传算法的主要特点是对对象直接进行操作，而不需要其他的限制条件；可以自动在全局的范围内进行搜索，自动寻找到相对正确的前进路线，而不需要程序者自己设定搜索规则[14]。

1.2.5 人工神经网络和机器学习算法

人是世界上最神奇的动物，人的嗅觉，触觉，味觉等一系列感官器官对世界信息的提取，大脑对信息的传递和处理仍然是一个谜题。从古至今，对人脑的研究一直在继续，渐渐有了许多的发现。其中神经元的作用是用于接受刺激，产生兴奋和传导兴奋。针对神经元的特殊结构、功能和神经元之间的复杂联系，人们对其进行了模仿并发展，产生了人工神经网络科学。

人工神经网络是现今主要的一种人工智能技术，是一种对神经细胞结构和信息存储，处理建立数学模型进行模拟的方法[15]。正如人类需要学习，人工神经网络同样需要学习，才能正常工作，所以它一般分为学习阶段和工作阶段。其中学

习阶段可以通过监督学习、无监督学习、强化学习完成。有监督学习是在训练数据中，确定输入输出之间的关系，只是得到其中关系的方法不同。而无监督学习为给定输入数据，你并不知道正确的输出数据是怎么样的，需要通过聚类等算法发现之中的关系。而强化学习则是通过环境学习，得到环境的强化信号对行为好坏的一个评价。

神经网络控制系统分为基于传统控制理论的神经控制和基于神经网络的智能控制[8]。现阶段存在的神经网络有 BP 神经网络，径向基神经网络和感知机神经网络等，不同的神经网络适应的环境不同。由于温度是一个多因素耦合影响的控制量，而神经网络对物体的特性不敏感，不需要进行建模，所以可以选用神经网络对温度进行控制。有文献提出一种连续时间段聚类与 BP 神经网络相结合的二步日光温室温度预测方法，平均相对误差仅 6.23%[16]，可见神经网络的应用可以对温度取得很好的控制效果。Hipper 等人运用人工神经网络到每日温度分布预测[17]和 Mechaqrane，Thomas 等人运用到建筑物室内温度预测[18, 19]。

机器学习简而言之就是让机器学会像人一样学习，只是机器学习的是数据。机器学习就是计算机从大量的数据中学习潜在规则，并将规则运用到未知数据得到预测的结果。机器学习同样有监督学习，无监督学习。监督学习分为分类和回归，无监督学习有聚类和密度估计。具体到算法部分，监督学习分类部分有 k-近邻算法、朴素贝叶斯、支持向量机(SVM)、决策树，而回归部分则有线性回归、局部加权线性回归、Ridge 回归、Lasso 最小回归系数估计；无监督学习聚类有 k-均值、DBSCAN，密度估计有最大期望算法、Parzen 窗设计。

机器学习一般分为几个步骤：1)获取数据：这里的数据可以来自实验室、网络等，比如多地点的一星期白天各小时的温度数据及最后稳定的温度数据，还有各个空调风机的控制量数据；2)数据的规范化：获取的数据是杂乱的，而算法的输入时标准的，所以要将数据整合和整理成算法要求的格式；3)训练算法：将数据拿出一部分来对设计的算法进行训练，训练的是算法的参数和构建，比如决策树的训练是建立决策树的构成，而支持向量机的训练则是得到合适的 alpha 参数；4)测试算法：算法刚训练完成不一定可以直接应用实践，因为此时的算法只是针对训练的数据，而对未知数据的效果是未知的，所以需要测试算法的性能和误差。测试的数据来自剩余的数据；5)应用算法：应用算法到未知数据，效果不好的话

就需要重新设计算法，重复上述步骤。

在我查询到文献中，对温度的预测中支持向量机算法运用的最广泛。有文献提出基于支持向量机的空调控温过程实时预测[20]，提出基于支持向量机的 ABS 树脂聚合温度控制研究[21]，提出支持向量机的钢坯终轧温度预测[22]，提出最小二乘支持向量机在空调温控系统中的应用[23]。究其原因，支持向量机对环境的适应性比较好，只需要少量的数据就可以达到一个比较好的控制效果，而且对参数不敏感，同时支持向量机采用了核函数，将高维的数据应用到低维处理，避免“维数灾难”。

1.3 主要研究内容

本文的研究内容主要是在多台空调作用下实现对一个空间中多个点的温度控制，设计机器学习算法，完成强非线性条件下的温度控制。主要章节内容如下：

第一章综述课题研究背景与意义，介绍了当前使用比较频繁的几种温度控制算法的原理及适用范围，最后给出了本文主要的研究内容。

第二章介绍针对实验室真实环境，运用集成环境 Ansys 内的 Geometry 建立模型，Fluent 对空气流动的仿真和 CFD_Post 对仿真数据的采集以及温度图像的显示，将仿真数据整合，以便后续的处理。

第三章针对仿真数据，讲述不同的机器学习算法的原理和相关公式的推导，比较它们的训练和预测性能，选取性能表现最好的算法。

第四章一是针对最终算法，尝试不同数量的样本数，不同的参数，测试其性能表现，选取性能最优的参数，对实际系统提出指导意见。二是设计用户界面程序，以便算法的移植和用户操作。

第五章对所做工作进行进一步的总结，并对未来发展做出一定的期望。

1.4 研究思路和整体架构流程图

1) 通过介绍不同的温度控制算法，来突显出相对其他温度控制算法，机器学习算法对于解决强非线性的多点温度控制问题更合适。

2) 考虑到实际采集稳定数据的缓慢，难以满足机器学习训练的要求。可以对真实环境仿真，采集大量的仿真数据。

3) 对于多输入多输出数据的拟合，目前的机器学习算法多以神经网络为主，尝试现在运用最广泛的神经网络：BP 神经网络来解决这个问题，后续可以对其进行改进。

4) 为了算法对任意输入输出问题的解决，参数调节的方便和用户操作的便捷，可以设计界面交互程序。

图 1-5 给出了整体架构流程图。

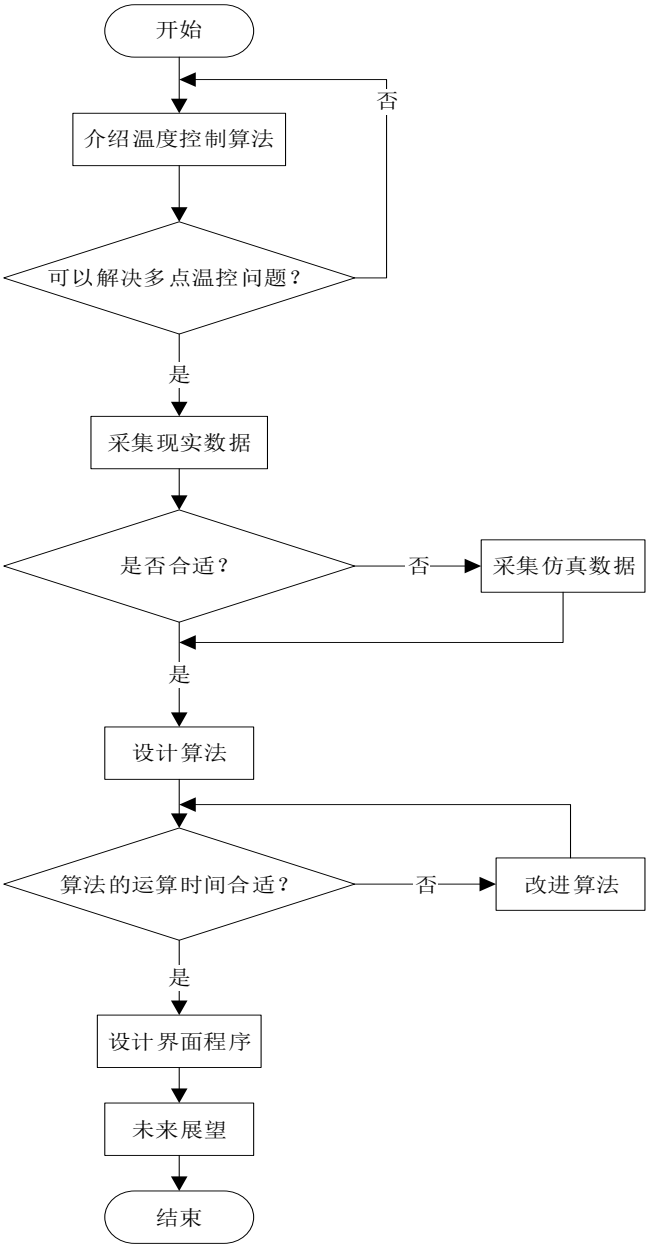


图 1-5 整体架构流程图

1.5 本章小结

本章简要介绍课题的研究背景及意义，描述了几种温度控制算法和研究现状，为后续的算法选择和设计做好准备。最后，介绍了本文的主要研究内容和论文的章节介绍。

第 2 章　实验数据采集

考虑到实际空调系统从设定温度到采集稳定数据的时间很长，难以满足机器学习数据量多的要求，因此采集实验环境的仿真数据就成为了一个解决方案。目前市面上流行的流体动力学仿真软件有 fluent，cfx，comsol 等，考虑到 fluent 的资料完善和具备并行运算的能力，最终选择了它。由于 Ansys 公司收购了 fluent，Ansys 软件就包含有 fluent。

2.1　Ansys 仿真软件简介

Ansys 软件是美国 Ansys 公司研制的大型通用有限元分析软件，是世界范围内增长最快的的计算机辅助软件，非常适合处理结构，流体，电场，磁场，声场及耦合场相关问题[24]。Ansys 的功能完善，可以解决一个问题从前期建模，计算，后期数据分析等的全部过程。

2.2　实验环境建模和数据采集过程

2.2.1　实验环境建模

本课题的仿真建模原型为浙江工业大学网络化智能控制实验室，建模工具为 Ansys 自带的 Geometry 软件，实验室的体积为 10*8*3.5m3，下面显示了实验室的真实环境和建立的仿真模型。

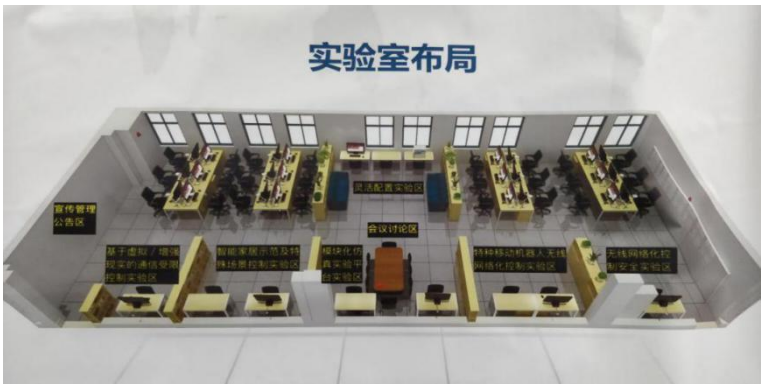


图 2-1　实验室真实环境

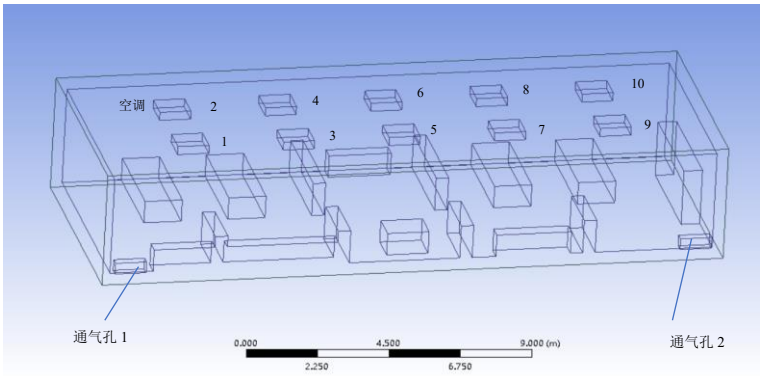


图 2-2　实验室仿真模型

考虑到实际系统会通过各处的缝隙与外界的空气进行交换，这在建模上很难实现，后续假定模型只有两个通气孔与外界进行空气的交换。实际开启空调之后，为了与外界尽量少进行能量交换，通常都是关闭窗户和门，于是在模型中，门窗就没有绘制。为了最大程度与实际系统一致，家具，空调的大小、位置、姿态在建模的时候与实验室保存一致。

建模完成需要对模型划分网格，这部分由 Ansys 内的 ICEM CFD 完成，图 2-3，2-4 显示了对建筑内部空气(流体)，建筑体(固体)的网格划分。

网格划分的密度决定了后面仿真的精度，但是过密的网格划分会导致 fluent 计算量的增加。在计算时间满足的前提下，尽可能提高网格划分密度。

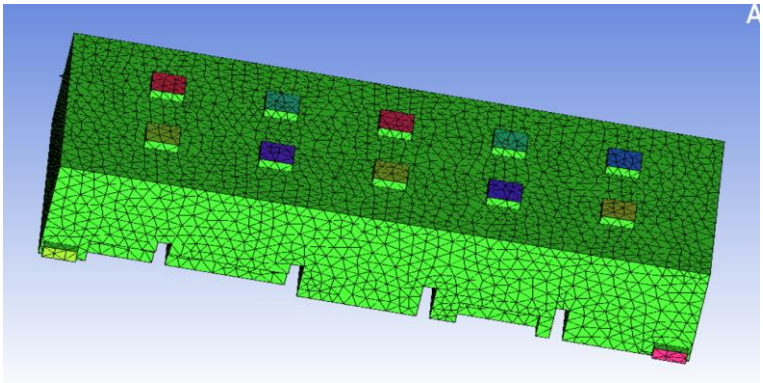


图 2-3　建筑内部空气网格划分

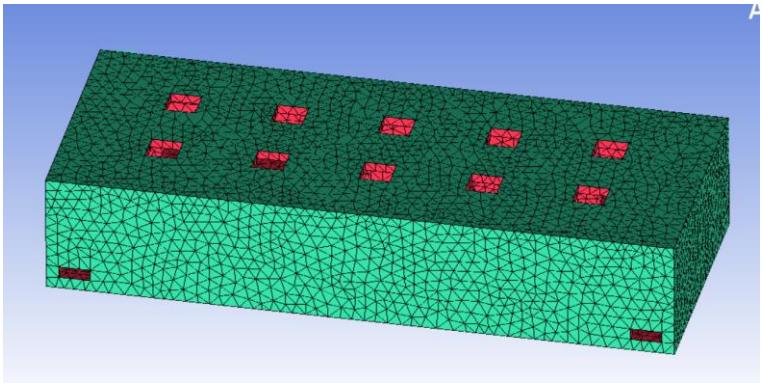


图 2-4　建筑体网格划分

2.2.2　fluent 数据采集

网格划分结束，就可以将网格导入到 fluent 中，进行流体动力学的仿真。fluent 仿真采用的模型为能量模型和普通 k-omega 模型。

假定环境温度为 20℃，具体的热传递分为：空调与空气的对流，建筑与空气的对流和辐射，建筑与外界的辐射。建筑的初始温度定为 20℃，墙体的导热系统设为 10w/(m2*K)，墙体的厚度设定为 0.3m，fluent 仿真对空调主要有两个可调参数：温度、风速。对于这两个参数，温度是一个必调参数，考虑风速，当风速设定为零的时候，就相当于没有开启空调，但这样会产生一个问题：最终预测的是空调的设定值，那如何判断空调的开启和关闭？为了解决这个问题，之后采集数据时，空调默认都是开启的；再者风速设定为不同的数值，会导致空气的对流速度不同，进而导致温度分布会不一样，这对于预测空调的温度设定值是不利的，由于平时空调使用时，并不会频繁设定空调的风速，于是在这里就将风速设定为一个常数，本论文定为 0.1m/s。

空调的温度设定是一个排列组合的问题，空调的调节范围为[21, 30]，数值为整数，每次选择 n 台空调进行调节，实验室共有 10 台空调，那么其中一共的排列组合数为：

$$S = \sum_{n=1}^{10} C_{10}^{n} * 10^{n} \tag{2-1}$$

以上 S 的计算结果很大，超过 10^{15}，全部仿真显然是不可能的，因此只能靠少量的数据来预测空调的设定值。

前期在采集 100 组数据的预测效果不佳的情况下，又采集了 340 组数据。图 2-5 显示了在空调设定为 21，22，23，24，25，26，27，28，29，30(℃)的实验室温度分布图。

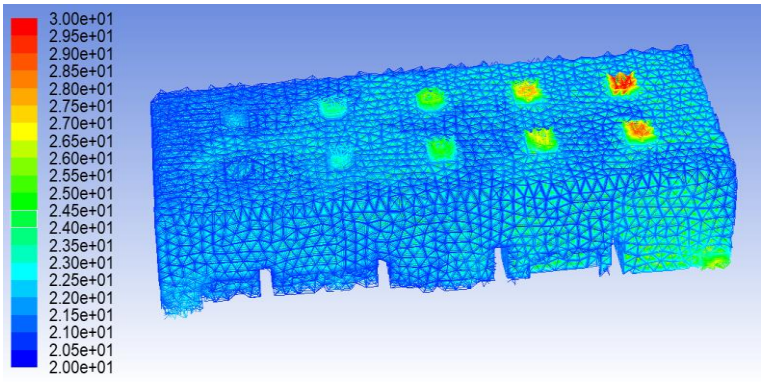


图 2-5　温度分布图

2.2.3　CFD_Post 后数据处理和导出

fluent 只能对流体动力学进行仿真，而数据的后处理可以由 CFD_Post 完成。CFD_Post 是一款专业的后数据处理的软件，Ansys 也集成了这部分软件，它可以显示某一横截面的温度分布，见图 2-6，也可以显示空气流动云图，最重要的是它可以将任何地点的温度导出到 Excel 文档中，以便后续处理。

本论文选取了四个采样点：x=-8，y=2，z=3；x=-8，y=-2，z=3；x=8，y=-2，z=3；x=8，y=2，z=3，具体位置可以见图 2-6 中的“+”号所在位置。后面则需要根据这四个地点的值来预测空调的设定值。

采样点的温度数据单位为开尔文(K)，这点需要注意。

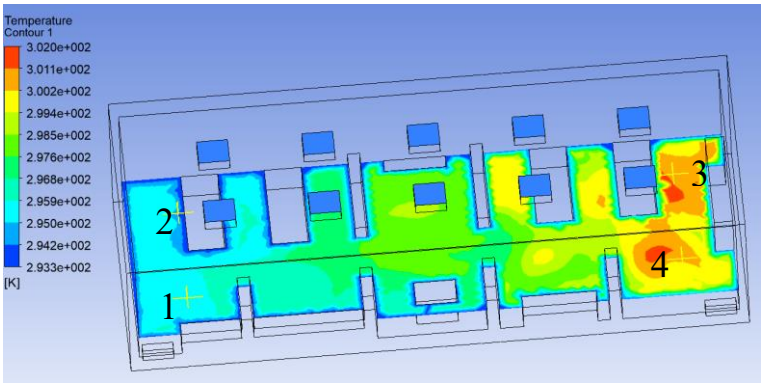


图 2-6　某横截面的温度分布

2.3　原始数据的整合

为了后续处理的方便，需要将采集到的 440 组数据保存在一个文件中，考虑到大部分编程语言对.txt 文件的读取，保存都有相应的库函数，因此将数据保存为.txt 文件是一个不错的选择，保存的格式为：采样点 1 温度 ... 采样点 4 温度 空调 1 设定值 ... 空调 10 设定值，各数据之间由空格相隔。详情可见图 2-7。

```
293.651123  293.150055  293.149963  293.149994  21  20  20  20  20  20  20  20  20  20
294.152466  293.149994  293.149963  293.149994  22  20  20  20  20  20  20  20  20  20
295.656342  293.149994  293.149963  293.150024  25  20  20  20  20  20  20  20  20  20
297.661407  293.149963  293.150024  293.150055  29  20  20  20  20  20  20  20  20  20
298.162659  293.149963  293.149963  293.149994  30  20  20  20  20  20  20  20  20  20
293.192169  293.525299  293.150024  293.149994  20  21  20  20  20  20  20  20  20  20
293.23407   293.900818  293.149994  293.149994  20  22  20  20  20  20  20  20  20  20
293.360107  295.027039  293.149994  293.149994  20  25  20  20  20  20  20  20  20  20
293.528137  296.528595  293.149963  293.149994  20  29  20  20  20  20  20  20  20  20
293.57016   296.903961  293.150024  293.149994  20  30  20  20  20  20  20  20  20  20
293.226166  293.151184  293.1521    293.155884  20  20  21  20  20  20  20  20  20  20
293.302429  293.152527  293.154236  293.161743  20  20  22  20  20  20  20  20  20  20
```

图 2-7　数据保存

2.4　本章小结

本章主要是针对实验室真实环境，对其建立模型和设置不同的空调设定值，仿真空气流动和能量交换，对四个地点采样温度数据，并将全部数据保存为.txt 文件，方便后续程序的读取。

第 3 章　基于机器学习的多点温控算法研究

本论文第一章中提到了位式控制，PID 控制，模糊控制，遗传算法，就单一的算法而言，以上四种算法并不适合于多输入多输出模型的温度控制，而机器学习是一种基于数据的算法，并不关心模型的具体形式，这样的特性使得它对于非线性模型的拟合效果比一般的算法好。本论文就当前使用比较频繁的机器学习算法：BP 神经网络为例，对第二章的数据.txt 文件进行处理，并对未知数据进行预测。本论文选用的输入为 4 个地点的期望温度，输出为 10 个空调的设定值。

3.1　传统 BP 神经网络介绍

3.1.1　算法基本原理

BP(Error Back Propagation Training)算法，又名误差反向传播算法，是一种前馈的算法，而采用 BP 算法的神经网络是目前运用最广泛的神经网络。下面以最简单的三层(输入层、隐藏层、输出层)BP 神经网络为例，简单介绍一下其原理和公式推导过程。BP 神经网络算法分为信号前向传播和误差反向传播两部分，图 3-1 显示了前向传播过程。

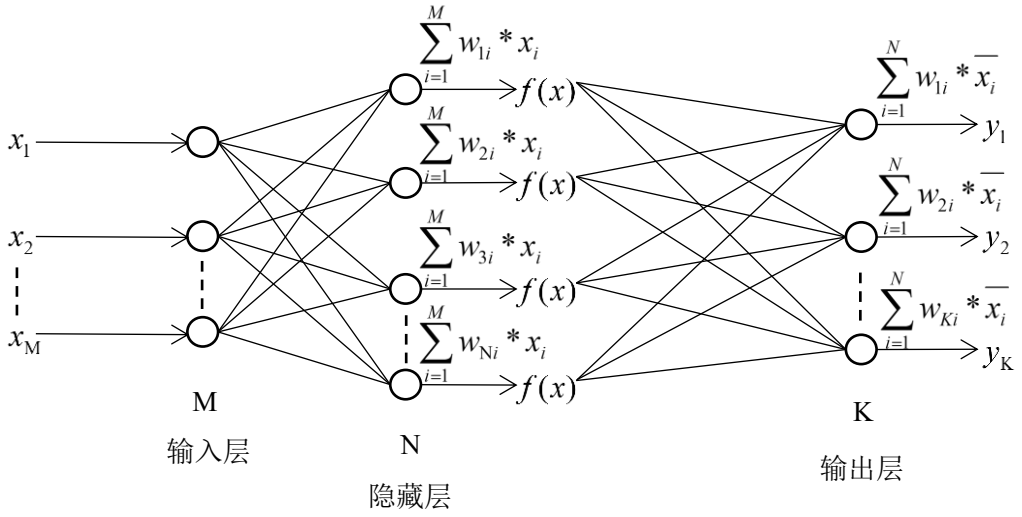


图 3-1　BP 神经网络前向传播过程

系统的实际输入为$x_1, x_2, ..., x_{M-1}$，x_M通常设置为 1，其所在神经元称为偏置神经元，可以这样理解：

$$\sum_{i=1}^{M} w_{ni} * x_i = \sum_{i=1}^{M-1} w_{ni} * x_i + w_{nM} * x_M = \sum_{i=1}^{M-1} w_{ni} * x_i + w_{nM} \tag{3-1}$$

正如$y = kx + b$，b项的加入可以使得直线的拟合效果变得更好，w_{nM}同样起到了这样的作用。很多 BP 神经网络会加入b项，但会导致调节的参数变多，而输入层多添加一项x_M在起到相同作用的时候又减少了计算量。

系统输入确定之后，将每一个输入与隐藏层每一个节点中的M个w参数相乘求和，得到该节点的输出，第n个隐藏层节点的输出为：

$$\sum_{i=1}^{M} w_{ni} * x_i \tag{3-2}$$

则隐藏层需要参数数为$N * M$。

隐藏层的输出到输入层的输入之间需要经过一个激活函数$f(x)$。由$y_n = \sum_{i=1}^{M} w_{ni} * x_i$可见，在$w_{ni}$为常数的时候，$y_n, x_i$之间的关系是线性的。也就是无论参数怎么样变化，该神经网络都只能拟合线性函数，这将神经网络的使用范围大大减小。激活函数的引入，则解决了这个问题。$f(x)$一般选取为 sigmoid 函数：

$$f(x) = \frac{1}{1+e^{-x}} = 1 - \frac{1}{1+e^{x}} \tag{3-3}$$

其形状为图 3-2。

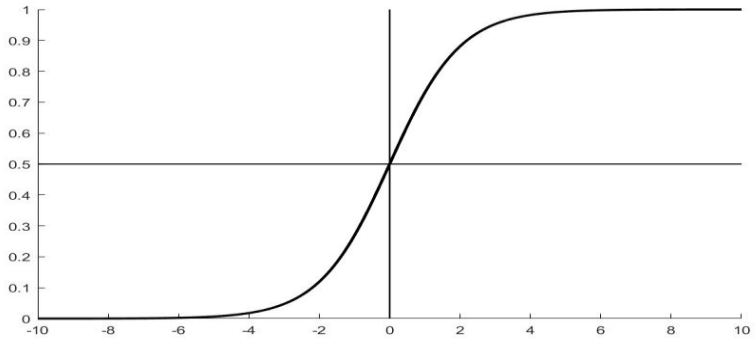


图 3-2　sigmoid 函数

而使用 sigmoid 的原因除了解决神经网络只能拟合线性函数问题，个人觉得还有两个原因：一是将隐藏层的输入归一化到[0, 1]，归一化一方面保证隐藏层输出值发挥的作用是一致的，不会由于存在特别大的数，而小的数被忽略，没有起到作用。另一方面，归一化的数值在计算时不会发生数值异常，这在编程时的作用很明显；二是 sigmoid 函数求导很简单，求导结果为：

$$f'(x) = f(x)*(1-f(x)) \tag{3-4}$$

这个结果对于后面反向传播的相关计算是非常有利的。

隐藏层的输出经过激活函数得到 N 个输出 $\bar{x}$，这些输出作为输出层神经元的输入，可得到第 k 个输出层神经元的输出为：

$$\sum_{i=1}^{N} w_{ki} * \overline{x_i} \tag{3-5}$$

因此输出层需要的参数数为 $N*K$。对于回归模型，上面的前向传播过程已经结束了，但是对于分类的话，还需要将上面的输出再经过一次激活函数，将其归一化到[0, 1]。

下面进行反向传播过程的相关推导。我们将 K 个输出层输出值与期望值的误差平方和的一半定为损失函数：

$$S = \frac{1}{2}\sum_{k=1}^{K}(Y_k - y_k)^2 \tag{3-6}$$

其中 Y_k 为实际输出值，y_k 为期望值，S 的值越小，表明输出值与期望值越接近。那么我们的目标是通过调整参数 w 将 S 的值减小。

这个计算过程需要用到梯度下降法。梯度下降法简而言之就是函数在梯度的方向上下降速度是最快的，就多元函数，梯度就是某一个自变量的偏导数，用下面这个算式进行简单的描述：

$$x = x - \lambda * \Delta y, \Delta y = \frac{\partial y}{\partial x} \tag{3-7}$$

由(3-2)可知，当 $\frac{\partial y}{\partial x}$ 为负数时，自变量 x 增大，反之减小。下面将对 $\frac{\partial S}{\partial w_{kj}}$ 进行求解：

$$
\begin{aligned}
S &= \frac{1}{2}\sum_{k=1}^{K}(Y_k - y_k)^2 \\
Y_k &= \sum_{j=1}^{N} w_{kj} * \overline{x_j} \\
\frac{\partial S}{\partial w_{kj}} &= \frac{\partial S}{\partial Y_k} * \frac{\partial Y_k}{\partial w_{kj}} = (Y_k - y_k) * \overline{x_j} \\
w_{kj} &= w_{kj} - \lambda * \frac{\partial S}{\partial w_{kj}} = w_{kj} + \lambda * (y_k - Y_k) * \overline{x_j}
\end{aligned}
\tag{3-8}
$$

其中 j 为隐藏层变量，范围[1, N]，k 为输出层变量，范围为[1, K]。$\overline{x_j}$ 为隐藏层激活函数的输出，λ 为步长。下面对 $\frac{\partial S}{\partial w_{ji}}$ 进行求解：

$$
\begin{aligned}
S &= \frac{1}{2}\sum_{k=1}^{K}(Y_k - y_k)^2 \\
Y_k &= \sum_{j=1}^{N} w_{kj} * \overline{x_j} \\
\overline{x_j} &= sigmoid(H_j) \\
H_j &= \sum_{i=1}^{M} w_{ji} * x_i
\end{aligned}
\tag{3-9}
$$

$$
\frac{\partial S}{\partial w_{ji}} = (\sum_{k=1}^{K} \frac{\partial S}{\partial Y_k} * \frac{\partial Y_k}{\partial \overline{x_j}}) * \frac{\partial \overline{x_j}}{\partial H_j} * \frac{\partial H_j}{\partial w_{ji}} = (\sum_{k=1}^{K} (Y_k - y_k) * w_{kj}) * \overline{x_j}(1 - \overline{x_j}) * x_i
$$

$$
\begin{aligned}
w_{ji} &= w_{ji} - \lambda * (\sum_{k=1}^{K} (Y_k - y_k) * w_{kj}) * \overline{x_j}(1 - \overline{x_j}) * x_i \\
&= w_{ji} + \lambda * (\sum_{k=1}^{K} (y_k - Y_k) * w_{kj}) * \overline{x_j}(1 - \overline{x_j}) * x_i
\end{aligned}
\tag{3-10}
$$

其中 i 为输入层变量，范围[1, M]，x_i 为系统的输入，λ 为步长。

由此，反向传播部分的推导结束，由上述推导可以对隐藏层参数 w_{ji}，输出层参数 w_{kj} 进行调整。

3.1.2 算法实现流程图和训练结束条件

关于程序训练部分结束的条件，一般有这么几种：

1) 全部训练数据预测输出值与期望值的均方误差小于一个阈值；
2) 迭代次数大于一个阈值；

3) 两次的均方误差相差很小，可以认为均方误差已经不变了；

4) 设定一个按键中断，识别到按下该键的时候就退出，这样的话就可以手动对程序进行干预，不同过分的等待退出条件；

5) 将输入数据分为训练集，验证集和测试集。但验证集的均方误差小于一个阈值的时候，就退出迭代。验证集的存在可以评价神经网络的泛化能力，即对未知数据的预测能力。

图 3-3 显示了程序实现的流程图，实现的语言为 python，该部分代码有参考文献[25]。

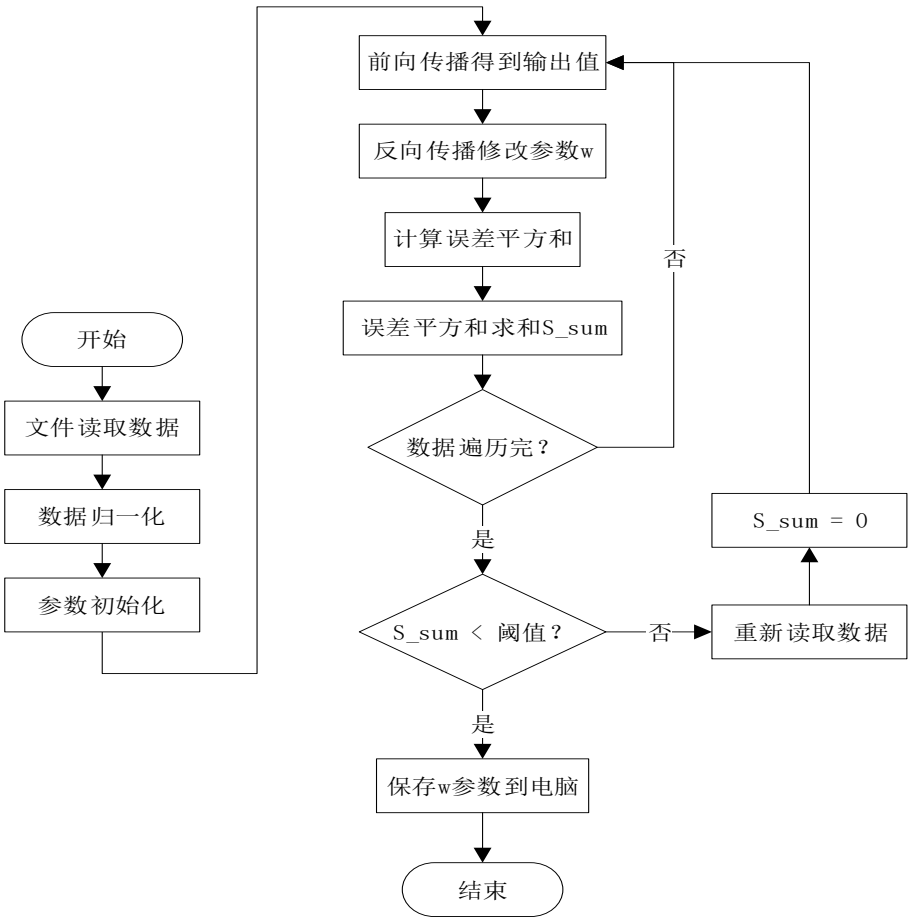


图 3-3 算法实现流程图

3.1.3 算法存在的问题

1) 传统 BP 神经网络采用梯度下降法，而梯度下降法可能会导致算法在找到局部极小值就退出，无法找到全局最小值，可以图 3-4 的 I 处。

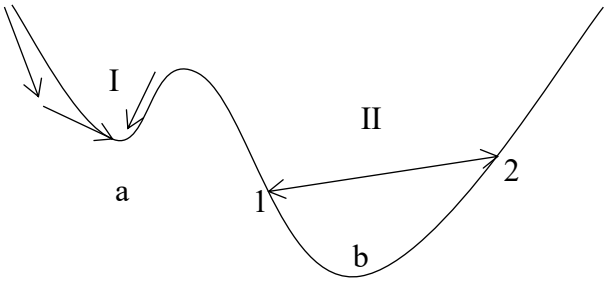


图 3-4　陷入局部极小值和震荡

当从左向右搜索时，算法就会认为 a 点是最小值，这与实际的最小值是不符合的。

2) 传统 BP 神经网络的学习率 λ 是一个常数，当 λ 设置比较大的时候，它可能会绕过局部极小值，找到全局最小值，但是这也会产生另一个问题：算法无法收敛，见 II 处，1，2 点的导数相反，1 处减去导数到 2 处，2 处减去导数到 1 处，这就产生了震荡，导致算法无法收敛。而 λ 设置较小的时候，就会产生训练速度很慢和陷入极小值的问题。

3) λ 是一个常数，导致训练速度完全取决于 $\frac{\partial S}{\partial w}$，如图 3-4，越接近于 a 点，导数的值就越小，训练的速度也就越慢。更有可能像图 3-5 的 I 处，导数的值为 0，训练无法进行。

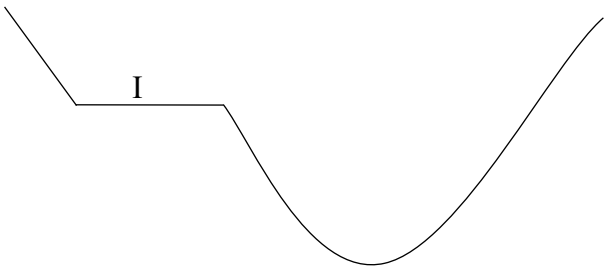


图 3-5　训练过慢

测试的时候，发现输入 20 组数据，到训练结束需要 1 个多小时，而输入 100 组数据，就需要 12 个多小时，更不用说输入 400 组数据了。

显然，传统 BP 神经网络对于该课题不太合适，需要对其进行优化。

3.2 传统 BP 神经网络的优化

3.2.1 运算结构的选择

本次程序的实现语言为 python，其基本的运算结构有两种：List 列表和 Matrix 矩阵。为了了解这两种运算结构的运行速度，我对它们进行了测试：全部输入数据计算完毕为迭代一次，计算迭代 N 次所花费的时间。下图给出了测试结果:

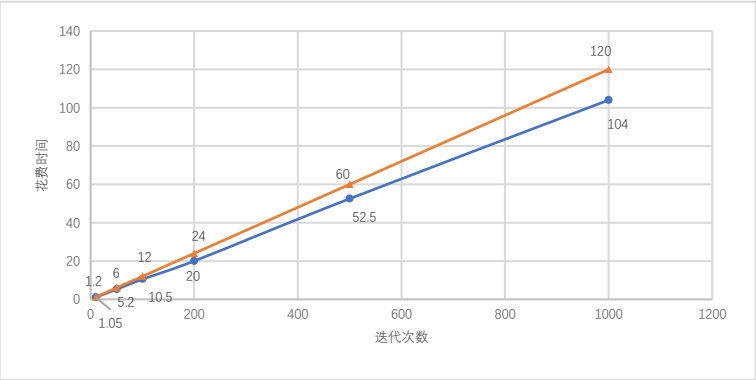


图 3-6 list(三角形)和 matrix(圆形)运算速度比较

从图 3-6 可以看出，随着 N 的增加，Matrix 比 List 花费的时间越来越长。因此，程序的最终实现结构为 List 列表。

3.2.2 自适应学习率和增加动量项

从传统 BP 神经网络存在的问题可以看出，学习率 λ 的选择是一个难点，因此我们可以根据均方误差的变化来动态的给定学习率 λ。均方误差变小，说明参数的变化方向是对的，那么下一次我们就可以给更大的 λ，使得梯度下降的更快；均方误差变大，说明参数的变化方向是错误的，那么我们可以减小 λ 的值，使得参数在往反方向运行的步长减小，进而收敛到最小值。

同时，动态的 λ 对于参数的震荡的抑制也是有效的，以图 3-7 为例。

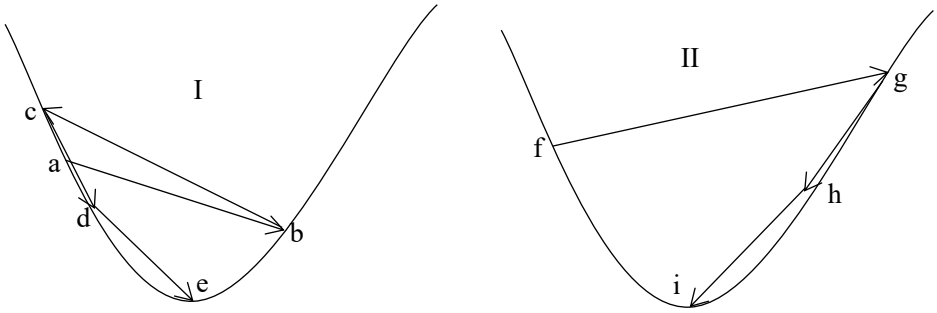


图 3-7　动态 λ 抑制震荡

I 中的 a 点由于学习率过大，导致越过最小值到 b 点，此时误差减小，学习率增大，导致到达 c 点，这时学习率减小，进而算法收敛到 d 点，最后找到最小值。

II 中的 f 点由于学习率过大，导致越过最小值到 g 点，此时误差增大，学习率减小，到达 h 点，进而算法收敛到 i 点。而震荡的情况只会在 a 和 b 点，f 和 g 点之间变化，无法收敛。动态 λ 在程序中可以表示为式子(3-11):

$$\begin{aligned} &if \quad S\_sum < S\_sum\_last \\ &\lambda^* = 1.001 \\ &else \quad if \quad S\_sum > S\_sum\_last \\ &\lambda^* = 0.9 \end{aligned} \tag{3-11}$$

动量项则利用了参数的变化量，可表示为式(3-12)

$$\Delta w_i = \lambda * \frac{\partial S}{\partial w} + \alpha * \Delta w_{i-1} \tag{3-12}$$

Δw 代表了参数的变化趋势，当 $\frac{\partial S}{\partial w}$，$\Delta w_{i-1}$ 符号一致的时候，Δw_{i-1} 的加入可以有效的加快梯度的下降速度。而 $\frac{\partial S}{\partial w}$，$\Delta w_{i-1}$ 符号不一致的时候，Δw_{i-1} 的加入则可以抑制震荡，以图 3-8 为例。a 点过大的增量使得其变化到 b 点，此时 $\frac{\partial S}{\partial w} < 0$，$\Delta w_{i-1} > 0$，使得 $|\Delta w|$ 比没有增量项的时候小而变化到 c 点，此时 $\frac{\partial S}{\partial w} > 0$，$\Delta w_{i-1} < 0$，同样使得 $|\Delta w|$ 比没有增量项的时候小而变化到 d 点，比没有增量的时候震荡次数减少。

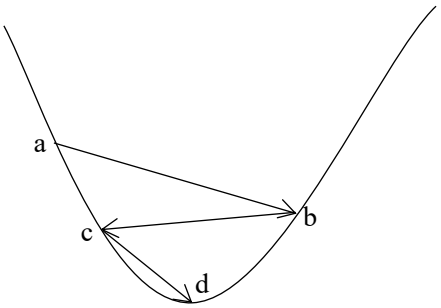


图 3-8　增量项加入抑制震荡

动态学习率和增量项的加入可以有效的加快 BP 神经网络的收敛速度，经测试 100 组输入数据，10 个隐藏层节点的训练时间是 120s，100 组输入数据，50 个隐藏层节点的训练时间是 354s，200 组输入数据，20 个隐藏层节点的训练时间为 607s，而 300 组输入数据，100 个隐藏层节点的训练时间为 2 个小时。但是其预测效果并不是很好，隐藏层节点数小的时候对差别较大的数字无法区分，而隐藏层节点增加，训练时间就会长很多，对本课题来说并不是特别合适，还需要继续优化。

3.2.3　弹性 BP 神经网络

由于传统 BP 神经网络的训练速度取决于 $\frac{\partial S}{\partial w}$，导致其训练速度越来越慢，那么当我们只用到导数的符号，而不是它的值的话，其大小对训练速度就没有影响。弹性 BP 神经网络就是基于这张思想。为每一个参数设置一个学习率 λ_{ij}，参数的变化为：

$$\Delta w_{ij} = \lambda_{ij} * sign(\frac{\partial S}{\partial w_{ij}}) \tag{3-13}$$

其中 $sign(x)$ 为取符号函数，λ_{ij} 的变化根据下面的规则：当这次的偏导数和上次的偏导数符号相同时，表明参数 λ_{ij} 可以继续增加；而符号不同时，表明系统正在震荡，参数 λ_{ij} 就需要减小。

弹性 BP 神经网络的训练速度比动态学习率和增加动量项的神经网络速度快，但预测效果与上述方法类似，并不是特别好。

3.2.4 LM_BP 算法

LM(Levenberg-Marquardt)，中文名为莱文贝格-马夸特，能提供非线性函数最小化的数值解，该算法结合了牛顿高斯法和梯度下降法的优点。为了能够理解 LM 算法，接下来先简单介绍牛顿高斯法。

牛顿高斯法与梯度下降法的目的是一样的，都是为了求取函数 $f(x)$ 的的最小值，而在高数中可以得知在：当 $f'(x)=0$ 时，函数存在极值。对 $f'(x)$ 进行泰勒展开，并只保留前两项可得：

$$\begin{aligned} f'(x) &\approx f'(x_0)+f''(x_0)(x-x_0)=0 \\ x &= x_0-\frac{f'(x_0)}{f''(x_0)} \\ x_n &= x_{n-1}-\frac{f'(x_{n-1})}{f''(x_{n-1})} \end{aligned} \tag{3-14}$$

上式对于一元函数可以直接应用来得到函数最小值，而对于多元的问题就需要用到雅可比矩阵。以本论文所讨论的问题为例，假设训练的样本总数为 A，损失函数为:

$$S=\frac{1}{2}\sum_{i=1}^{A}\left\|\vec{e}_i\right\|_2=\frac{1}{2}\sum_{i=1}^{A}\left\|\vec{y}_i-\vec{Y}_i\right\|_2 \tag{3-15}$$

其中 $\vec{y}_i,\vec{Y}_i$ 分别为第 i 个样本的期望输出向量和实际预测输出向量，$\vec{e}_i$ 为第 i 个样本输出误差向量。

我们的目标是将 S 的值减少，对 S 求偏导和二次偏导，我们可以得到：

$$\begin{aligned} \frac{\partial S}{\partial w_{kj}} &= \sum_{i=1}^{A}(\vec{e}_i*\frac{\partial \vec{e}_i}{\partial w_{kj}}) \\ \frac{\partial^2 S}{\partial^2 w_{kj}} &= \sum_{i=1}^{A}(\frac{\partial \vec{e}_i}{\partial w_{k'j'}}*\frac{\partial \vec{e}_i}{\partial w_{kj}})+\sum_{i=1}^{A}(\vec{e}_i*\frac{\partial^2 \vec{e}_i}{\partial w_{kj}\partial w_{k'j'}}) \end{aligned} \tag{3-16}$$

牛顿高斯法将二次偏导省略，可以得到：

$$\frac{\partial^2 S}{\partial^2 w_{kj}}=\sum_{i=1}^{A}(\frac{\partial \vec{e}_i}{\partial w_{k'j'}}*\frac{\partial \vec{e}_i}{\partial w_{kj}}) \tag{3-17}$$

引入雅可比矩阵 J_e，可得：

$$\frac{\partial S}{\partial w_{kj}} = J_e^T * \vec{E}$$
$$\frac{\partial^2 S}{\partial^2 w_{kj}} = J_e^T * J_e \tag{3-18}$$

其中 $\vec{E}$ 为全部样本的输出误差组成的向量，格式为 $[\vec{e}_1, \vec{e}_2, ..., \vec{e}_{A-1}, \vec{e}_A]^T$,大小为 (K*A)*1。J_e 即为雅可比矩阵，大小为：(K*A)*(M*N+N*K)，格式为：

$$J_e = \begin{bmatrix} \frac{\partial E_1}{\partial W_1} & \frac{\partial E_1}{\partial W_2} & \cdots & \frac{\partial E_1}{\partial W_{B-1}} & \frac{\partial E_1}{\partial W_B} \\ \frac{\partial E_2}{\partial W_1} & \frac{\partial E_2}{\partial W_2} & \cdots & \frac{\partial E_2}{\partial W_{B-1}} & \frac{\partial E_2}{\partial W_B} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial E_K}{\partial W_1} & \frac{\partial E_K}{\partial W_2} & \cdots & \frac{\partial E_K}{\partial W_{B-1}} & \frac{\partial E_K}{\partial W_B} \\ \frac{\partial E_{K+1}}{\partial W_1} & \frac{\partial E_{K+1}}{\partial W_2} & \cdots & \frac{\partial E_{K+1}}{\partial W_{B-1}} & \frac{\partial E_{K+1}}{\partial W_B} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial E_{2K}}{\partial W_1} & \frac{\partial E_{2K}}{\partial W_2} & \cdots & \frac{\partial E_{2K}}{\partial W_{B-1}} & \frac{\partial E_{2K}}{\partial W_B} \\ \frac{\partial E_{2K+1}}{\partial W_1} & \frac{\partial E_{2K+1}}{\partial W_2} & \cdots & \frac{\partial E_{2K+1}}{\partial W_{B-1}} & \frac{\partial E_{2K+1}}{\partial W_B} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial E_{AK}}{\partial W_1} & \frac{\partial E_{AK}}{\partial W_2} & \cdots & \frac{\partial E_{AK}}{\partial W_{B-1}} & \frac{\partial E_{AK}}{\partial W_B} \end{bmatrix} \tag{3-19}$$

其中 B 为 w 参数的总数，大小为 M*N+N*K。参数 w 的集合向量 $W = [w_1, w_2, ..., w_{M*N}, ..., w_{M*N+N*K}]^T$，于是可以得到：

$$\Delta W = \frac{\frac{\partial S}{\partial w_{kj}}}{\frac{\partial^2 S}{\partial^2 w_{kj}}} = \frac{J_e^T * \vec{E}}{J_e^T * J_e} = (J_e^T * J_e)^{-1} * J_e^T * \vec{E} \tag{3-20}$$
$$W = W - \Delta W$$

以上就是牛顿高斯法与本课题的结合和相关公式的推导，而 LM 算法是在牛顿高斯算法的基础上加上梯度下降法，具体的表达式为：

$$\Delta W = (J_e^T * J_e + \lambda * I)^{-1} * J_e^T * \vec{E} \tag{3-21}$$

可以看出当 λ 很小时，上式就为牛顿高斯法；当 λ 越大时,

$$\Delta W = (\lambda * I)^{-1} * J_e^T * \vec{E} = \frac{1}{\lambda} * J_e^T * \vec{E} \tag{3-22}$$

就越接近梯度下降法。之所以采用 LM 法，有以下几点原因：

1) 梯度下降法在接近目标值时，搜索速度会很慢，而牛顿高斯法则可以弥补这点，它在接近目标的速度很快，精度也很高。以二次函数 x^2 为例，梯度下降法为：$x_n = x_{n-1} - 2x * \lambda'$，当 $\lambda' = 0.4$ 时，越接近 $x = 0$,速度就越慢，当 $\lambda' = 0.5$，则可以得到期望值，但是这样的参数很难得到；牛顿高斯法为：$x_n = x_{n-1} - \frac{2x_{n-1}}{2} = 0$，一步到达期望值，不需要设定参数。

2) 牛顿高斯法在远离期望值的时候收敛速度没有梯度下降法快，LM 算法利用了这点，远离期望值时用梯度下降法，接近期望值时用牛顿高斯法。

3) 牛顿高斯法的 $J_e^T * J_e$ 矩阵不一定时正定的，也就是这个矩阵不一定存在逆矩阵，而加上 $\lambda * I$ 之后，在 λ 合适的情况下，这个矩阵一定是正定的，该表达式一定有解。

综合上面，需要对 λ 进行调节，可以这样设定：当 S 比上一次全局迭代的 S 小，λ 减小，趋向于牛顿高斯法；反之，就要将 λ 增大，趋向梯度下降法。一般 λ 初始设定为 0.001。

还剩最后一个问题，就是雅可比矩阵的求解，接下来就进行相关公式的推导，符号的含义与之前一致。

对输出层参数 w_{kj}，有：

$$\begin{aligned} e_k &= y_k - Y_k \\ Y_k &= \sum_{j=1}^{N} w_{kj} * \overline{x_j} \\ \frac{\partial e_k}{\partial w_{kj}} &= \frac{\partial e_k}{\partial Y_k} * \frac{\partial Y_k}{\partial w_{kj}} = -1 * \overline{x_j} \end{aligned} \tag{3-23}$$

对隐藏层参数 w_{ji}，有：

$$\begin{aligned} e_k &= y_k - Y_k \\ Y_k &= \sum_{j=1}^{N} w_{kj} * \overline{x_j} \\ \overline{x_j} &= sigmoid(H_j) \\ H_j &= \sum_{i=1}^{M} w_{ji} * x_i \\ \frac{\partial e_k}{\partial w_{ji}} &= \frac{\partial e_k}{\partial Y_k} * \frac{\partial Y_k}{\partial \overline{x_j}} * \frac{\partial \overline{x_j}}{\partial H_j} * \frac{\partial H_j}{\partial w_{ji}} = -1 * w_{kj} * \overline{x_j}(1 - \overline{x_j}) * x_i \end{aligned} \tag{3-24}$$

其中e_k和w_{kj}的k是不一样的，这时候就将导数设为0，比如$\frac{\partial e_1}{\partial w_{21}}=0$。

至此，关于LM算法设计的公式已全部推导完成，附录部分会给出这部分的python代码，代码中的参数下标与上述公式推导的参数下标是相反的，这点需要注意。

LM算法优化的函数为一个全局函数，因此LM算法具有全局的收敛性，这导致他的预测精度比一般的BP神经网络高，同时由于LM算法用到了雅可比矩阵，这个矩阵的大小为：(K*A)*(M*N+N*K)，导致LM算法的内存占用量比较大，但是算法的效率比较高，因此LM算法的训练速度同样很快，经测试400组输入数据，20个隐藏层节点的训练时间为151s。

考虑到LM算法的超快收敛速度，最终选用LM算法，对BP神经网络优化结束。

3.3 多点温控算法的其他尝试

除了BP神经网络，对多输入多输出数据的拟合算法还有支持向量机。

支持向量机是为了寻找一个分隔类别的超平面，以二分类问题为例，就是找到一条直线，将两个类别区分。直线的表达式为：$\vec{w}*\vec{x}+b=w_1x_1+w_2x_2+...+w_nx_n+b$，定义直线下方的类别$y=1$，直线上方的类别$y=-1$，正常分类的点$y*(\vec{w}*\vec{x}+b)>0$，可以得到平面点到直线的距离为$\frac{y_i*(\vec{w}*\vec{x_i}+b)}{\left\|\vec{w}\right\|_2}$，可以看到$w,b$同时变化时，对距离并没有什么影响，那么我们就可以指定分子或分母为常数1，支持向量机指定分子为1，也就是$y_i*(\vec{w}*\vec{x_i}+b)=1$，也就是我们需要求$\min\left\|w\right\|_2$。

对于这样的凸优化问题，我们可以运用拉格朗日乘子来求解得，$\vec{w}=\sum_{i=1}^{N}(\alpha_i y_i*\vec{x_i})$

而α_i的求解需要用到序列最小优化算法(SMO)，这里就不详细讲述。

为了了解支持向量机对多输入多输出模型的拟合程度，运用python语言的机器学习库scikit-learn里的SVM对其进行了测试，假定输入为{[0, 1], [1, 2], ..., [98, 99], [99, 100]}，输出为{[0, 2], [2, 4], ..., [198, 200]}，即输出均为输入的2

倍，预测[20, 30]，SVM 的拟合输出为[45.37283776, 47.37283776]，可以看到 SVM 对多输入多输出的拟合效果不好，除非与原输入相差很近，不然预测效果会很差，也就是算法的泛化能力比较差。考虑到这种情况，选择放弃了支持向量机。

3.4 本章小结

本章设计了多种机器学习算法对第二章的多输入输出数据进行拟合，对这多种算法从原理，程序流程，运行速度和预测效果等进行了描述。最终 LM_BP 算法在输入数据多的情况下运行速度仍然很快，同时对多点的温度预测均方误差达到了 0.2，与传统的温度控制算法无法对多点的温度进行控制或者控制效果不佳相比，实现了很好的控制效果。

第 4 章　算法测试和用户界面设计

4.1　算法测试

就 LM_BP 算法，下面从训练数据组数、隐藏层节点数、是否发生过拟合、预测效果四个方面进行测试。测试环境为：Intel(R) Core(TM) i5-7300HQ CPU @2.5GHz, 16GB 内存，系统 Windows10 64 位操作系统，未使用 GPU 加速。

4.1.1　训练数据组数

1) 隐藏层节点数为 10，训练数据分别为 100，200，300，400 的 10 组测试数据的输出均方误差(三角形)，平均绝对误差(圆形)和训练时间。

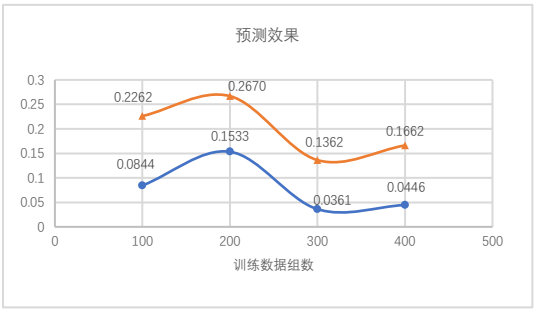


图 4-1a　误差曲线

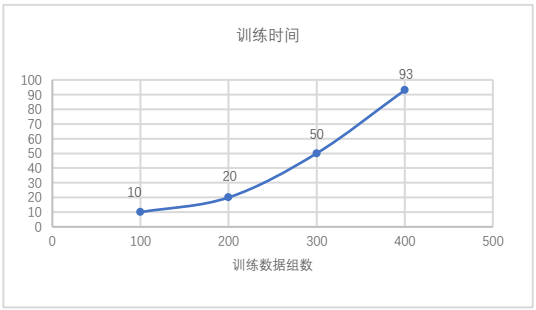


图 4-1b　时间曲线

2) 隐藏层节点数为 20，训练数据分别为 100，200，300，400 的 10 组测试数据的输出均方误差(三角形)，平均绝对误差(圆形)和训练时间。

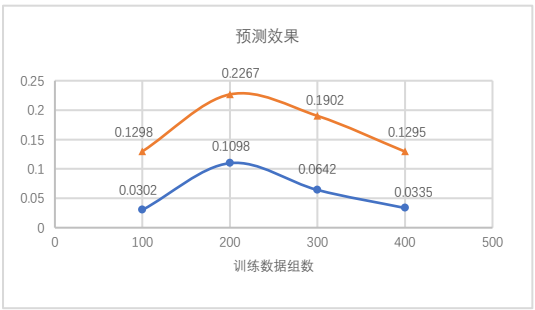


图 4-2a　误差曲线

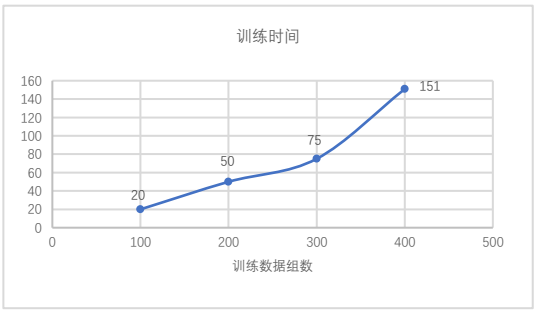


图 4-2b　时间曲线

3) 隐藏层节点数为 50，训练数据分别为 100，200，300，400 的 10 组测试数据的输出均方误差(三角形)，平均绝对误差(圆形)和训练时间。

由于输入 100 组训练数据时，测试出现了过拟合现象，导致输出误差很大，于是将均方误差和平均绝对误差均设为 1。

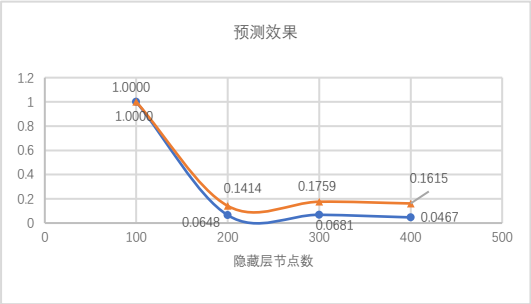


图 4-3a　误差曲线

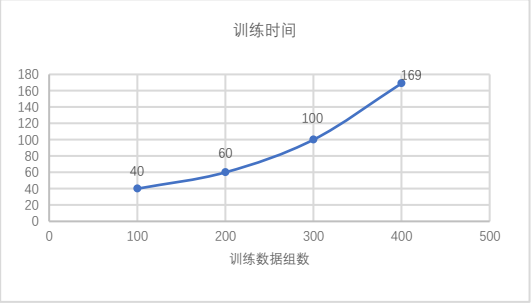


图 4-3b　时间曲线

4) 隐藏层节点数为 100，训练数据分别为 200，300，400，均会发生过拟合的现象。

4.1.2　隐藏层节点数

1) 训练数据分别为 100，隐藏层节点数为 10，20，50，100 的 10 组测试数据的输出均方误差(三角形)，平均绝对误差(圆形)和训练时间。50，100 时均发生过拟合。

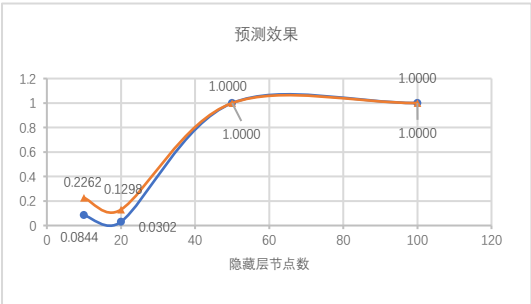


图 4-4a　误差曲线

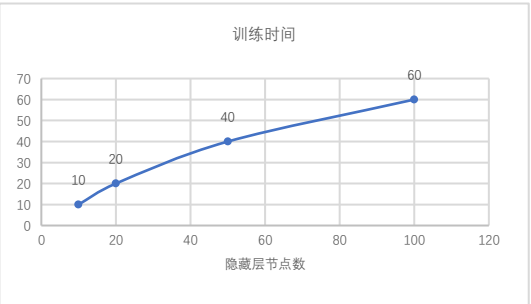


图 4-4b　时间曲线

2) 训练数据分别为 200，隐藏层节点数为 10，20，50，100 的 10 组测试数据的输出均方误差(三角形)，平均绝对误差(圆形)和训练时间。100 时发生过拟合。

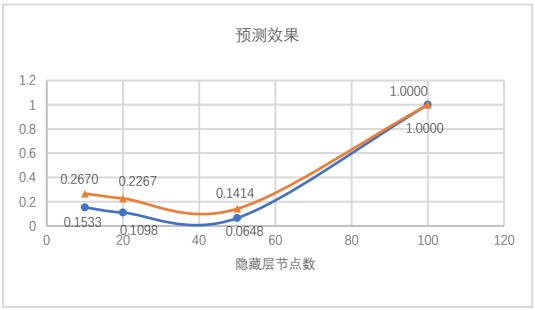


图 4-5a 误差曲线

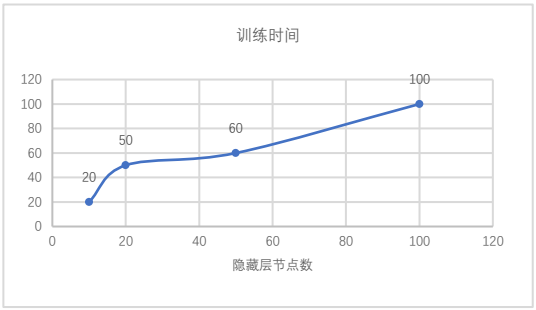


图 4-5b 时间曲线

3) 训练数据分别为 300，隐藏层节点数为 10，20，50，100 的 10 组测试数据的输出均方误差(三角形)，平均绝对误差(圆形)和训练时间。100 时可能会发生过拟合。

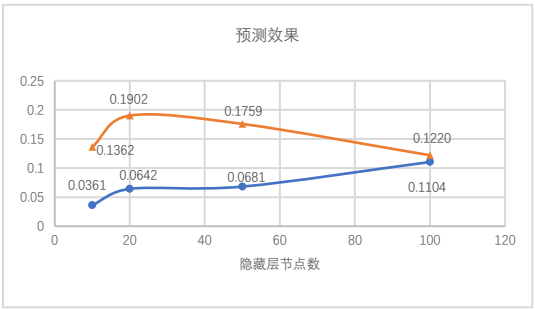


图 4-6a 误差曲线

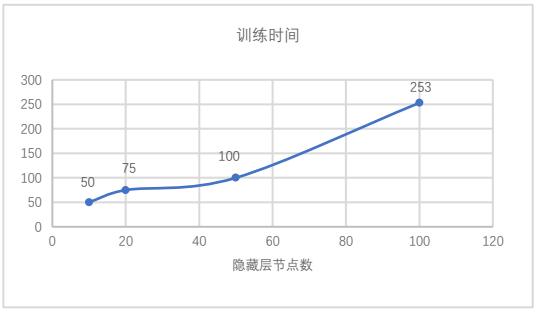


图 4-6b 时间曲线

4) 训练数据分别为 400，隐藏层节点数为 10，20，50，100 的 10 组测试数据的输出均方误差(三角形)，平均绝对误差(圆形)和训练时间。100 时可能会发生过拟合。

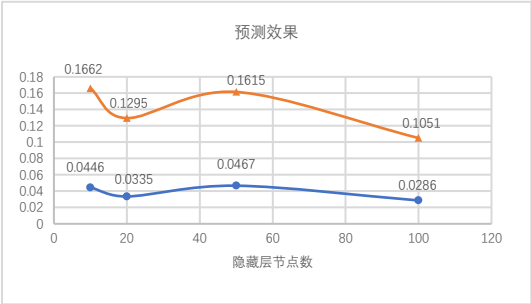


图 4-7a　误差曲线

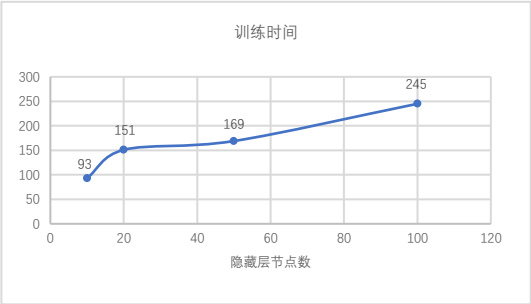


图 4-7b　时间曲线

4.1.3　多数据的预测效果

对 100 组测试数据的预测效果如下：

表 4-1　100 组测试数据预测效果

训练组数	隐藏层节点数	100 组数据预测效果(均方误差)
100	10	5.00
100	20	5.20
200	10	3.46
200	20	3.17
200	50	5.73
200	100	5.73
300	10	2.95
300	20	2.04
300	50	2.06
300	100	3.00

4.1.4　测试总结

1) 训练数据组数和隐藏层节点数的增加会使训练时间加长。

2) 隐藏层节点数达到一定值时会发生过拟合现象，这个值与训练数据组数

相关。

3) 随着训练数据组数的增多，模型对数据的预测效果更好，但是增多到一定值时，组数的影响就会很小。

综上所述，最终选取对本课题而言最佳的参数为：300 组训练数据，20 个隐藏层节点数，预测多点温度的均方误差为 0.19,平均绝对误差为 0.06，控制效果很好。

4.2 用户界面设计

为方便参数的调节和一般用户的使用，设计了界面操作程序，该界面程序适合于任意输入和输出的 LM_BP 模型，应用范围很广。界面外形见图 4-1。

图 4-8 用户操作界面

该用户操作界面分为两个主要部分：左边的训练部分和右边的预测部分。

4.2.1 训练部分

[开始训练]按键：按下之后就可以选择训练所需要的数据，比如本论文所使用的 data.txt，文本格式按照之前所述。这样设置的目的是用户不需要将.txt 文件放在固定的文件夹下面，方便操作。选择数据之后程序就会自动进行训练。该按

键可以在[退出训练]按键按完之后，再次点击，表示重新开始训练。注意：在按键之前需要设定按键界面下方的 6 个参数。

[清空输出]按键：将[训练、验证、测试误差输出值]文本框内的内容清除，在重复训练的时候可以使用。

[退出训练]按键：按下之后，就会终止当前正在训练的进程，并会弹出文件框，这里需要选择保存当前训练参数的文件夹，选择完成之后，会在选择的目录下自动生成 input_weights.txt 和 output_weights.txt 参数保存文件。

[帮助]按键：显示相关提示。

[训练组数]文本框：输入训练数据的样本组数。

[验证组数]文本框：对训练模型进行验证的样本组数，验证数据是用来对当前参数的预测效果的显示，便于中途中断训练。

[测试组数]文本框：对训练完成的模型进行综合测试的样本数。

[调节的区域数]文本框：最终用户所需求的温度个数，也是预测模型的输入层的节点数，也是本论文题目的“多点”的点数。

[隐藏层节点数]文本框：BP 模型隐藏层的节点数。

[调节的空调数]文本框：输出层的节点数，最终预测输出空调设定值的个数。

[训练、验证、测试误差输出值]文本框：各种信息输出的地方，信息包括：选择的数据文件路径、训练样本数、验证样本数、测试样本数、各步的训练均方误差，各步的验证均方误差、保存参数的文件夹路径、测试数据的真实值，测试数据的预测值、测试数据的均方误差、整个训练，测试程序过程的时间。文本框会实时显示，根据这个文本框的信息就可以了解训练的全过程，便于判断什么时候该结束训练，以及什么参数的训练效果好。

4.2.2 预测部分

[开始预测]按键：会根据训练的参数和[请输入各点想要的温度]文本框内容，得到各台空调的预测设定值，输出以空格分隔。训练部分没有执行时，需要先按[设定参数]按键，设置相关参数和选择之前训练参数的数据文件，然后按下[开始预测]按键会打开文件框，选择预测需要的参数所在的文件夹，根据选择的参数来进行预测。之后再次按下按键时，会自动选择之前的参数来进行预测。当你需

要选择其他的预测参数时，可以按[换预测参数]按键，再次按[开始预测]按键就可以重新选择预测参数，主要设置的目的是：可以使预测部分单独工作和比较不同参数的预测效果；而训练部分已执行过时，则会根据保存的参数来进行预测。

[换预测参数]按键：按下之后可以让[开始预测]按键重新选择预测参数，便于比较各种参数的预测效果。

[设定参数]按键：在训练部分未执行时，对网络的各种参数进行初始化，这里的参数包括训练部分的六个参数，之后这里会弹出数据文件框，你可以选择之前训练网络的数据文件，选择完成会在[预测的 n 台空调设定值]文本框中显示成功信息。

[清空预测输出]按键：清空[预测的 n 台空调设定值]文本框的内容。

[请输入各点想要的温度]文本框：输入各点期望温度，单位℃，输入以空格分隔。

[预测的 n 台空调设定值]文本框：输出 n 台空调的设定温度，单位℃，按照这个温度就可以对空调进行调节。

最终将用 python 语言实现的界面程序打包为 exe 文件，可以在没有安装 python 的 Windows 系统上运行，大大提高了算法的可移植性。

4.3 本章小结

本章就 LM_BP 算法，从不同的方向对其进行了测试，并选择了对当前数据而言最好的参数，预测多点温度的均方误差为 0.19,平均绝对误差为 0.06，预测效果很好。而后设计了用户操作的界面程序，包括训练和测试部分，通过该界面，用户可以简单的进行加载数据，保存参数，预测空调设定值等操作，大大方便了用户的操作和程序的移植。

第 5 章　总结与展望

5.1　毕设工作总结

随着人们生活质量和工业生产需求的增长，对温度控制的要求也随之提高，同时伴随着的是温度控制算法的发展。传统的温度控制算法多针对单点的温度控制，很少对一个空间中强耦合的多点的温度进行控制或者控制效果不佳,无法满足一个空间中不同人群对温度的不同要求。而近些年来，随着机器学习技术的兴起和其在其他领域非线性的优异性能表现，如何将机器学习技术与多点的温度控制进行有效的结合成了一大研究热点。

本文的研究内容是基于机器学习的多点温控算法研究，主要针对温控算法的研究方面，实现在多台空调作用下对一个空间中多个点的温度控制。论文的主要工作如下：

1) 查阅相关文献资料，综述课题研究背景与意义，以及温度控制算法的研究现状，介绍了现今存在的各种温度控制算法。

2) 利用 Ansys 对实验室真实环境进行建模和对温度变化过程的仿真，采集在不同的空调温度设定值的作用下各采样点的温度数据，保存为.txt 文件，便于后续的处理。

3) 设计机器学习算法对数据进行处理，并对各点未知温度需求进行空调设定值的预测，在 Ansys 进行仿真验证。讲述不同的机器学习算法的原理和公式推导，测试它们的性能差异，选取最终性能表现最好的算法及其相关参数，以达到最好的控制。

4) 针对控制算法，设计用户界面程序，转为 Windows 系统可执行程序，方便算法的移植和用户操作。

5.2　未来展望

本文经过认真的实验和研究，基本达到了预期的研究目标，完成了基于仿真数据的多点温度控制算法的研究，并设计了界面操作程序，方便算法的移植。但仿真与实际的表现还是有差异的，如何获取大量的实际的温度数据就成了一大难

点，这个方面本论文并没有涉及。同时设计的算法在实际系统的表现如何并没有明确的测试，所有的数据测试都是基于仿真平台的，这可能导致与实际差异较大。

就实际系统的数据采集，目前有想法是：实际测试改变空调设定值，t 时间内可以采集到稳定的温度数据。每隔 t 时间，控制器自动改变空调的设定值，采集各采样点的温度数据，将空调设定值和采样点的温度数据自动上传到云端或者保存到本地的存储器中，过一段时间就可以得到实际系统的数据。

采集到实际系统数据，就可以进行训练得到相关参数，控制器就可以根据参数预测各空调的设定值，从而进行控制。同时数据的采集可以继续进行，对预测参数可以在线升级，进而增加对环境的适应性。

就训练速度而言，由于本论文训练数据并不是很多，速度还是在可以接受的范围。但随着数据的增多，训练速度会慢下来，未来的计算可以尝试 GPU 加速，缩短训练时间。

参 考 文 献

[1] 夏立先. 化学反应与温度控制[J]. 学生园地, 1996, (5): 46-47.

[2] 柯维娜, 朱定强, 蔡国飙. 温度控制技术的发展与应用[J]. 计量学报, 2007, 28(3A): 178-184.

[3] 刘美霞. 智能温度控制系统[D]. 南京: 南京航空航天大学. 2003: 1-58.

[4] IEA international energy agency. (2010). Policy pathway – energy performance certification of buildings, <http://www.iea.org/papers/pathways/buildings_certification.pdf>, 2010.

[5] L. P. Lombard, J. Ortiz, and C. Pout, “A review on buildings energy consumption information”, Energy Build, vol. 40, no. 3, pp. 394–398, 2008.

[6] 卞正岗. 温度的测量和控制综述[N]. PLC&FA, 2011-4.

[7] 马林, 马茵. 温度位式控制系统的分析[J]. 洁净与空调技术, 2010, (4): 45-46.

[8] 张锋. 智能 PID 温度控制技术研究与应用[D]. 南京: 南京工业大学, 2005: 1-116.

[9] 任玉苓. 浅议 PID 参数的整定[J]. 甘肃科技, 2008, 24(16): 63-64.

[10] 刘玲玲. PID 参数整定技术的研究及应用[D]. 郑州: 郑州大学, 2010: 1-57.

[11] 马俊卿. 工业无线温度控制系统设计[D]. 上海: 东华大学, 2017: 1-62.

[12] 蔚东晓, 贾霞彦. 模糊控制的现状和发展[J]. 自动化与仪器仪表, 2006, (6): 4-7.

[13] 吕萍. 常用温度控制算法的分析与研究[J]. 科技视界, 2012, (28): 124-125.

[14] 王银年. 遗传算法的研究与应用-基于 3PM 交叉算子的退火遗传算法及应用研究[D]. 无锡: 江南大学, 2009: 1-65.

[15] 樊军庆, 张宝珍. 温度控制理论的发展概况[J]. 工业炉, 2008, 30(6): 12-14.

[16] 陈昕, 唐湘璐, 李想, 刘天麒, 贾璐, 卢韬. 二次聚类与神经网络结合的日光温室温度二步预测方法[J]. 农业机械学报, 2017, (S1): 353-358.

[17] Hippert, H., & Pedreira, C. (2004). Estimating temperature profiles for short-term load forecasting: Neural networks compared to linear models. IEE Proceedings Generation, Transmission and Distribution, 151 (4) 543–547.

[18] Mechaqrane, A., & Zouak, M. (2004). A comparison of linear and neural network ARX models applied to a prediction of the indoor temperature of a building. Neural Computing & Applications, 13(1), 32–37.

[19] Thomas, B., & Soleimani-Mohseni, M. (2007). Artificial neural network models for indoor temperature prediction: Investigations in two buildings. Neural Computing & Applications, 16,81–89.

[20] 肖晓萍, 李自胜, 蒋刚. 基于支持向量机的空调控温过程实时预测[J]. 计算机工程与应用, 2007, 43(31): 219-221.

[21] 赵凡. 基于向量机的 ABS 树脂聚合温度控制研究[D]. 大连: 大连理工大学, 2004: 1-60.

[22] 肖莎丽. 基于支持向量机的温度控制系统研究[D]. 武汉: 武汉科技大学, 2010: 1-59.

[23] 黄琴, 陈虹, 靳召东. 最小二乘支持向量机在空调温控系统中的应用[J]. 机械研究与应用, 2011, (5): 124-126.

[24] 徐希. 管道系统结构动力特性分析与验证[D]. 南京: 南京航空航天大学, 2012.

[25] -Finley-. BP 神经网络与 Python 实现[DB/OL]. https://www.cnblogs.com/Finley/p/5946000.html, 2016-10-10.

附录 1：LM_BP 算法及界面操作程序

```
#############################################################################
# designer:汪咏
# date:2018-4
# project:基于机器学习的多点温控算法研究—LM_BP
# university：浙江工业大学
# description:运算过程用列表形式，运用 for 遍历进行计算，用留存交叉验证法进行训练和预测，并将相关参数保存在电脑里，便于下
#                    次直接提取
#############################################################################

#####
# 引用库
#####
import math
import random
import numpy as np
import time
import signal
import tkinter as tk
import tkinter.messagebox
import tkinter.filedialog

end_signal = 0  # 训练结束信号
test_signal = 0

#####
# 函数定义
#####
'''
@funtion:LoadDataSet
@description:加载温度的文本数据
@input:fileName:data.txt 所在目录，比如'data/data of air_conditioning.txt'
@output:两个 list 类型列表，形如[[], [], []],保持属性和结果，必要时可以转化为 mat or array
'''
def LoadDataSet(FileName, data_num, label_num):
    DataMat = []; LabelMat = [];
    fr = open(FileName)
    for line in fr.readlines():
        LineArr = line.strip().split('\t')
```

```
            DataMat.append([round(float(i) - 273.15, 4) for i in LineArr[0:data_num]])
            Label = [float(i) for i in LineArr[data_num:data_num + label_num]]
            LabelMat.append(Label)
    return DataMat, LabelMat

'''
@function:normalization
@description:归一化函数，实验发现同一网络，输入归一化后对输出的预测效果会好很多
@input:data:列表形式数据，形如[[ ，],[ ，],[ ，]]
@output:归一化之后的数据，范围为[0.2, 0.8]
'''
def normalization(data):
    max = np.max(data)
    min = np.min(data)
    m, n = len(data), len(data[0])
    for i in range(m):
        for j in range(n):
            data[i][j] = 0.2 + 0.6 * (data[i][j] - min) / (max - min)
    return data, max, min

'''
@function:random_train_test
@description:从原始数据中随机选取一定数量的数据来当中训练集和测试集
@input:data, label:原始数据，train_num:训练集数据的个数，valid_num:验证集数据个数，te
st_num:当为-1 时，训练集，验证集提取之后剩下的数据全部为测试集，否则提取参数个数
数据
@output:trainData, trainLabel:训练集，validData, validLabel:验证集, testData, testLabel:测试
集
'''
def random_train_test(data, label, train_num, valid_num, test_num=-1):
    # 提取训练测试编号
    dataSet = list(range(len(data)))
    trainSet = []; validSet = []; testSet = []  # 随机提取的数据
    for i in range(train_num):
        randIndex = int(np.random.uniform(0, len(dataSet)))
        trainSet.append(dataSet[randIndex])
        del(dataSet[randIndex])
    for i in range(valid_num):
        randIndex = int(np.random.uniform(0, len(dataSet)))
        validSet.append(dataSet[randIndex])
        del(dataSet[randIndex])
    if test_num == -1:  # 将剩下的都作为测试
        testSet = dataSet
    else:  # 只提取一定数量作为测试
```

```
            for i in range(test_num):
                randIndex = int(np.random.uniform(0, len(dataSet)))
                testSet.append(dataSet[randIndex])
                del(dataSet[randIndex])

        # 提取训练测试数据
        trainData = []; trainLabel = []
        validData = []; validLabel = []
        testData = []; testLabel = []
        for i in trainSet:  # 训练集
            trainData.append(data[i])
            trainLabel.append(label[i])
        for i in validSet:  # 训练集
            validData.append(data[i])
            validLabel.append(label[i])
        for i in testSet:  # 测试集
            testData.append(data[i])
            testLabel.append(label[i])
        return trainData, trainLabel, validData, validLabel, testData, testLabel

'''
@function:StoreWeights
@description:将参数保存在电脑里
@input:data:data:待保存的数据,类型随意，FileName:保存成的文件名，诸如'XX.txt'
'''
def StoreWeights(data, FileName):
    import pickle
    fw = open(FileName, 'wb')
    pickle.dump(data, fw)
    fw.close()

'''
@function:GrabWeights
@description:将电脑保存的数据提取
@input:FileName:已保存的数据文件名，诸如'XX.txt'
@output:提取的保存文件
'''
def GrabWeights(FileName):
    import pickle
    fr = open(FileName, 'rb')
    return pickle.load(fr)

'''
@function:rand
```

```
@description:产生[a, b]之间的随机数
@input:a, b 给定的随机数范围，未规定 a,b 的相对大小
@output:[a, b]之间的随机数
'''
def rand(a, b):
    return (b - a) * random.random() + a

'''
@function:make_matrix
@description:产生一个 m 行 n 列的二维列表，
@input:m:行数, n:列数, fill:列表中给定的值，默认为 0
@output:m 行 n 列的二维列表，m=3, n=2 时输出为[[0, 0], [0, 0], [0, 0]]
'''
def make_matrix(m, n, fill=0.0):
    mat = []
    for i in range(m):
        mat.append([fill] * n)
    return mat

'''
@function:sigmoid
@description:计算 sigmoid 函数的值,
@input:x:需要计算的值,注意当 x 很大的时候 math.exp 会溢出,而负数时则不会溢出
@output:sigmoid 函数的值，sidmoid(x) = 1 / (1 + exp(-x)),将 R 范围值转为(0, 1)范围值
'''
def sigmoid(x):
    if x < 0:
        return 1.0 - 1.0 / (1.0 + math.exp(x))
    return 1.0 / (1.0 + math.exp(-x))

'''
@function:sigmoid_derivative
@description:计算 sigmoid 函数求导的值
@input:x:sigmoid 函数值
@output:sigmoid 函数的求导值
'''
def sigmoid_derivative(x):
    return x * (1 - x)

'''
@function:BPNeuralNetwork
@description:BP 神经网络的具体实现过程
'''
class BPNeuralNetwork:  # BP 神经网络类
```

```
def __init__(self):  # 参数初始化
    self.input_n = 0  # 输入层节点数 + 1(偏移)
    self.hidden_n = 0  # 隐藏层节点数
    self.output_n = 0  # 输出层节点数
    self.input_cells = []  # 系统输入值，最后一个数值为 1(偏移)
    self.hidden_cells = []  # 隐藏层的输出值
    self.output_cells = []  # 输出值
    self.input_weights = []  # 隐藏层参数
    self.output_weights = []  # 输出层参数
    self.input_weights_save = []  # 隐藏层参数
    self.output_weights_save = []  # 输出层参数
    self.Jacobian_JM = []  # 保存 Jacobian 矩阵的部分信息
    self.M = 0  # 参数的总数
    self.N = 0  # 样本数
    self.S = 0  # 样本数*输出层节点数

def setup(self, ni, nh, no):  # 参数的初始设定值，设定各层的层数
    self.input_n = ni + 1
    self.hidden_n = nh
    self.output_n = no
    self.M = self.input_n * self.hidden_n + self.hidden_n * self.output_n

    # 各层输出初始值
    self.input_cells = [1.0] * self.input_n
    self.hidden_cells = [1.0] * self.hidden_n
    self.output_cells = [1.0] * self.output_n

    # 参数二维列表赋随机值
    self.input_weights = make_matrix(self.input_n, self.hidden_n)
    self.output_weights = make_matrix(self.hidden_n, self.output_n)
    for i in range(self.input_n):
        for h in range(self.hidden_n):
            self.input_weights[i][h] = rand(-0.6, 0.6)  # 2.4 / no
    for h in range(self.hidden_n):
        for o in range(self.output_n):
            self.output_weights[h][o] = rand(-0.6, 0.6)

    # 暂存二维列表赋值
    self.Jacobian_JM = make_matrix(self.output_n, self.M)

def predict(self, inputs):  # 按照参数得到各层的输出
    for i in range(self.input_n - 1):
        self.input_cells[i] = inputs[i]  # 输入层输出值
```

```
        # 隐藏层输出
        for h in range(self.hidden_n):
            total = 0.0
            for i in range(self.input_n):
                total += self.input_cells[i] * self.input_weights[i][h]  # 隐藏层输入值
            self.hidden_cells[h] = sigmoid(total)  # 隐藏层的输出值

        # 输出层输出
        for o in range(self.output_n):
            total = 0.0
            for j in range(self.hidden_n):
                total += self.hidden_cells[j] * self.output_weights[j][o]
            self.output_cells[o] = total  # 直接输出
        return self.output_cells

    def bp_nn(self, case, label):  # LM 算法核心函数，求取 Jacobian 矩阵
        # 预测输出
        self.predict(case)

        for o in range(self.output_n):
            i = 0
            j = 0
            k = 0
            for m in range(self.M):
                if m < self.input_n * self.hidden_n:
                    self.Jacobian_JM[o][m] = -1 * self.output_weights[i][o] * sigmoid_d
erivative(self.hidden_cells[i]) * self.input_cells[j]
                    i += 1
                    if i == self.hidden_n:  # 一行结束，重新开始
                        i = 0
                        j += 1
                else:
                    if (m - self.hidden_n * self.input_n) % self.output_n == o:
                        self.Jacobian_JM[o][m] = -1 * self.hidden_cells[k]
                        k += 1
                    else:
                        self.Jacobian_JM[o][m] = 0

    def train(self, datas, labels, validdata, validlabel, threshold=1.0, u=0.05):  # 对给定的
输入输出进行 bp 网络的训练，设定的
        error = 1
        index = 0
        error_min = 10
        self.N = len(datas)
```

```
        self.S = self.N * self.output_n
        Jacobian = np.mat(np.zeros((self.S, self.M)))
        Ew = np.mat(np.zeros((self.S, 1)))

        ##############################主训练程序##############################
        while error >= threshold:  # 只有绝对误差小于一定值，才结束参数的调节
            for i in range(self.N):
                data = datas[i]
                label = labels[i]
                self.bp_nn(data, label)
                for j in range(self.output_n):
                    Ew[i * self.output_n + j, 0] = label[j] - self.output_cells[j]  # 偏差
                    for m in range(self.M):
                        Jacobian[i * self.output_n + j, m] = self.Jacobian_JM[j][m]

            ##############################调整 u 参数##############################
            last_error = error
            error = (Ew.T * Ew)[0, 0] / self.S  # 均方误差
            index += 1
            if index > 1:
                if error < last_error:
                    u *= 0.5
                elif error > last_error:
                    u *= 10
            self.output.insert('end', 'error:')
            self.output.insert('end', error)
            self.output.insert('end', '\n')

            ########################调整 Wik, Wkj 参数########################
#
            weights_add = -1 * (Jacobian.T * Jacobian + u * np.identity(self.M)).I * Jac
obian.T * Ew  # 参数增量
            m = 0
            for i in range(self.input_n):
                for k in range(self.hidden_n):
                    self.input_weights[i][k] += weights_add[m, 0]
                    m += 1
            for k in range(self.hidden_n):
                for j in range(self.output_n):
                    self.output_weights[k][j] += weights_add[m, 0]
                    m += 1

            ########################退出训练条件##############################
            # 验证误差小于一定值，表示预测效果良好，直接退出训练
```

```
            valid_error = 0  # 验证误差
            for i in range(len(validdata)):
                predLabel = self.predict(validdata[i])
                for j in range(len(predLabel)):
                    valid_error += (predLabel[j] - validlabel[i][j]) ** 2
            valid_error = valid_error / len(validdata) / self.output_n
            if valid_error < error_min:
                error_min = valid_error
                self.input_weights_save = self.input_weights
                self.output_weights_save = self.output_weights
            self.output.insert('end', 'valid_error:')
            self.output.insert('end', valid_error)
            self.output.insert('end', '\n')
            self.output.see(tk.END)
            self.output.update()

            if valid_error < threshold:
                break

            # 迭代太多次就直接退出
            if index > 1e10:
                self.output.insert('end', '迭代太多次:')
                self.output.insert('end', index)
                self.output.insert('end', '\n')
                break

            # 按下 CTRL-C 就退出训练函数
            if end_signal == 1:
                break

            # 偏差变化小于一定的程度就退出
            if abs(error - last_error) < 1e-10:
                break

    def ListToStr(self, list):
        data = [str(int(i)) for i in list]
        return ' '.join(data)

    def hit1(self):
        global end_signal
        global test_signal
        end_signal = 0
        test_signal = 1
        FileName = tkinter.filedialog.askopenfilename()
```

```
            start = time.clock()
            self.output.insert('end', '选择的数据文件路径:')
            self.output.insert('end', FileName)
            self.output.insert('end', '\n')
            self.output.insert('end', '\n')
            datas, labels = LoadDataSet(FileName, int(self.input4.get()), int(self.input6.get()))
# 加载数据
            data_norm, self.max, self.min = normalization(datas)  # 归一化
            trainData, trainLabel, validData, validLabel, testData, testLabel = random_train_test
(data_norm, labels, int(self.input1.get()), int(self.input2.get()), int(self.input3.get()))  # 提取
训练集和测试集
            self.output.insert('end', '训练样本数:')
            self.output.insert('end', len(trainData))
            self.output.insert('end', '\n')
            self.output.insert('end', '验证样本数:')
            self.output.insert('end', len(validData))
            self.output.insert('end', '\n')
            self.output.insert('end', '测试样本数:')
            self.output.insert('end', len(testData))
            self.output.insert('end', '\n')
            self.output.insert('end', '\n')

            # 训练
            self.setup(int(self.input4.get()), int(self.input5.get()), int(self.input6.get()))
            self.train(trainData, trainLabel, validData, validLabel, 0.5, 0.01)  # 时间最长

            # 存储隐藏层和输出层的参数，便于以后使用
            FileName = tkinter.filedialog.askdirectory()
            self.output.insert('end', '\n')
            self.output.insert('end', '选择保存参数的文件夹路径:')
            self.output.insert('end', FileName)
            self.output.insert('end', '\n')
            self.output.insert('end', '\n')
            StoreWeights(self.input_weights_save, FileName+'/input_weights.txt')  # 保存隐藏层
参数
            StoreWeights(self.output_weights_save, FileName+'/output_weights.txt')  # 保存输出
层参数
            self.input_weights = GrabWeights(FileName+'/input_weights.txt')  # 提取隐藏层参
数
            self.output_weights = GrabWeights(FileName+'/output_weights.txt')  # 提取输出层
参数

            # 测试
            pred_error = 0  # 预测绝对误差
```

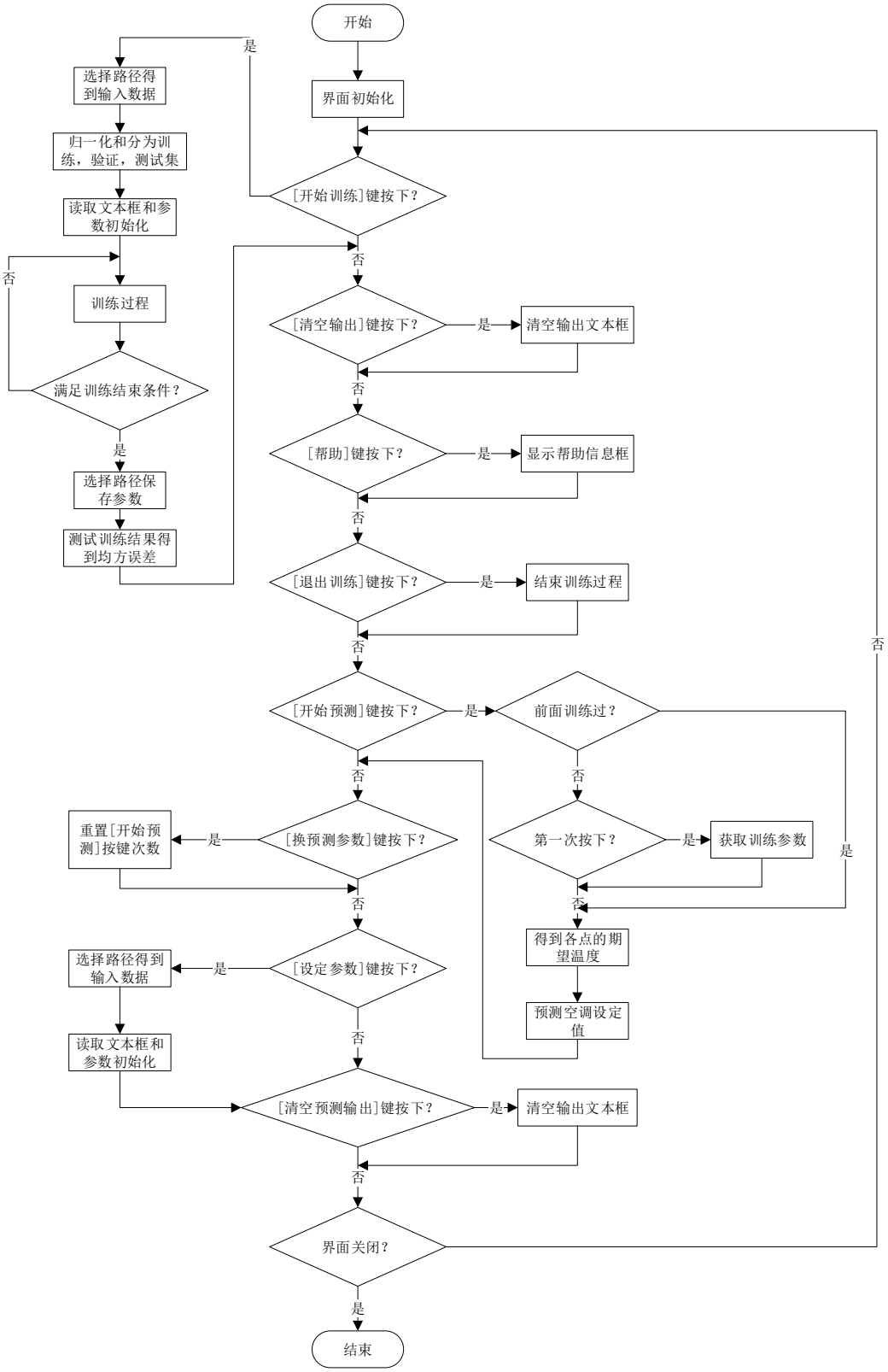
```
            predLabel = []
            for i in range(len(testData)):
                predLabel = [round(i) for i in self.predict(testData[i])]
                self.output.insert('end', '真实值:')
                self.output.insert('end', self.ListToStr(testLabel[i]))
                self.output.insert('end', '\n')
                self.output.insert('end', '预测值:')
                self.output.insert('end', self.ListToStr(predLabel))
                self.output.insert('end', '\n')
                for j in range(len(predLabel)):
                    pred_error += (predLabel[j] - testLabel[i][j]) ** 2
            end = time.clock()
            self.output.insert('end', '\n')
            self.output.insert('end', '均方误差:')
            self.output.insert('end', pred_error / len(testData) / len(predLabel))
            self.output.insert('end', '\n')
            self.output.insert('end', '花费时间:')
            if (end - start) >= 60:
                self.output.insert('end', int((int(end - start) - int(end-start) % 60) / 60))
                self.output.insert('end', 'min')
            self.output.insert('end', round((end-start) % 60, 4))
            self.output.insert('end', 's')
            self.output.insert('end', '\n')
            self.output.see(tk.END)
            self.output.update()


        def hit2(self):
            self.output.delete('1.0', tk.END)

        def hit3(self):
            global test_signal
            if test_signal == 0:
                FileName = tkinter.filedialog.askdirectory()
                self.output1.insert('end', '\n')
                self.output1.insert('end', '参数的文件夹路径:')
                self.output1.insert('end', FileName)
                self.output1.insert('end', '\n')
                self.output1.insert('end', '\n')
                self.input_weights = GrabWeights(FileName+'/input_weights.txt')  # 提取隐藏
层参数
                self.output_weights = GrabWeights(FileName+'/output_weights.txt')  # 提取输
出层参数
            test_signal = 1
```

```
        input_text = self.input7.get()
        pred_data = [float(i) for i in input_text.split()]
        for i in range(len(pred_data)):
            pred_data[i] = 0.2 + 0.6 * (pred_data[i] - self.min) / (self.max - self.min)
        data = [int(round(i)) for i in self.predict(pred_data)]
        self.output1.insert('end', self.ListToStr(data))
        self.output1.insert('end', '\n')
        self.output1.see(tk.END)
        self.output1.update()

    def hit4(self):
        self.output1.delete('1.0', tk.END)

    def hit5(self):
        tk.messagebox.showinfo(title='help', message='按[退出训练]键可以结束训练\n 并选
择保存参数的文件夹')

    def hit6(self):
        global end_signal
        end_signal = 1

    def hit7(self):
        self.setup(int(self.input4.get()), int(self.input5.get()), int(self.input6.get()))
        self.output1.insert('end', '设定参数完成！')
        self.output1.insert('end', '\n')

        FileName = tkinter.filedialog.askopenfilename()
        self.output1.insert('end', '选择的数据文件路径:')
        self.output1.insert('end', FileName)
        self.output1.insert('end', '\n')
        self.output1.insert('end', '\n')
        datas, labels = LoadDataSet(FileName, int(self.input4.get()), int(self.input6.get()))
# 加载数据
        data_norm, self.max, self.min = normalization(datas)  # 归一化

    def hit8(self):
        global test_signal
        test_signal = 0

    def main(self):  # 执行主函数
        ##################################界面#####################################
        window = tk.Tk()
        window.title('基于 LM-BP 神经网络的多点温控界面程序_汪咏_V2.0')
```

```
        window.geometry('610x500')

        canvas = tk.Canvas(window, height=500, width=610)
        canvas.place(x=0, y=0)
        canvas.create_rectangle(5, 5, 605, 495)
        canvas.create_line(300, 5, 300, 495)

        ###############################################################################
        tk.Button(window, text='开始训练', width=8, height=1, command=self.hit1).place(x=
10, y=10)
        tk.Button(window, text='清空输出', width=8, height=1, command=self.hit2).place(x=
80, y=10)
        tk.Button(window, text='帮助', width=8, height=1, command=self.hit5).place(x=220,
y=10)
        tk.Button(window, text='退出训练', width=8, height=1, command=self.hit6).place(x=
150, y=10)

        tk.Label(window, text='训练组数:', fg='red').place(x=10, y=45)
        self.input1 = tk.Entry(window, width=7)
        self.input1.place(x=10, y=70)
        self.input1.insert(0, 300)

        tk.Label(window, text='验证组数:', fg='red').place(x=110, y=45)
        self.input2 = tk.Entry(window, width=7)
        self.input2.place(x=110, y=70)
        self.input2.insert(0, 30)

        tk.Label(window, text='测试组数:', fg='red').place(x=210, y=45)
        self.input3 = tk.Entry(window, width=7)
        self.input3.place(x=210, y=70)
        self.input3.insert(0, 10)

        tk.Label(window, text='调节的区域数:', fg='red').place(x=10, y=95)
        self.input4 = tk.Entry(window, width=7)
        self.input4.place(x=10, y=120)
        self.input4.insert(0, 4)

        tk.Label(window, text='隐藏层节点数:', fg='red').place(x=110, y=95)
        self.input5 = tk.Entry(window, width=7)
        self.input5.place(x=110, y=120)
        self.input5.insert(0, 20)

        tk.Label(window, text='调节的空调数:', fg='red').place(x=210, y=95)
        self.input6 = tk.Entry(window, width=7)
```

```
            self.input6.place(x=210, y=120)
            self.input6.insert(0, 10)

            tk.Label(window, text='训练、验证、测试误差输出值:', fg='red').place(x=10, y=145)
            self.output = tk.Text(window, width=39)
            self.output.place(x=10, y=170)

            ##########################################################################
            tk.Button(window, text='开始预测', width=8, height=1, command=self.hit3).place(x=
310, y=10)
            tk.Button(window, text='换预测参数', width=8, height=1, command=self.hit8).place
(x=380, y=10)
            tk.Button(window, text='设定参数', width=8, height=1, command=self.hit7).place(x=
450, y=10)
            tk.Button(window, text='清空预测输出', width=10, height=1, command=self.hit4).pl
ace(x=520, y=10)

            tk.Label(window, text='请输入各点想要的温度:', fg='red').place(x=310, y=45)
            self.input7 = tk.Entry(window, width=40)
            self.input7.place(x=310, y=70)
            self.input7.insert(0, '22 24 25 21')

            tk.Label(window, text='预测的 n 台空调设定值:', fg='red').place(x=310, y=95)
            self.output1 = tk.Text(window, width=40, height=28)
            self.output1.place(x=310, y=120)  # 这里是窗口的内容

            window.mainloop()

# CTRL-C 信号处理函数
def CtrlCHandler(signum, frame):
    global end_signal
    end_signal = 1

# 主执行部分
if __name__ == '__main__':
    end_signal = 0
    signal.signal(signal.SIGINT, CtrlCHandler)
    nn = BPNeuralNetwork()
    nn.main()
```

附录 2：LM_BP 算法程序流程图



致　谢

大学生活，转瞬即逝。作为人生中不可或缺的四年，一路走来，有收获的喜悦，有辛勤劳作的汗水，有志同道合的伙伴，更有循循善诱的老师。在这个快要结束的时刻，对曾经帮助过我的他们表达深深的感谢。

首先，我要感谢我的母校浙江工业大学。她提供给了我一个优越的学习环境，使我可以安心沉浸于研究当中，没有收到外界的干扰。

然后，我要感谢我的指导教师赵云波教授。赵老师严谨的治学态度深深吸引着我。从最初的选题，初期答辩，到后面的中期答辩，毕业论文的撰写，赵老师给予了我很大的帮助。每次遇到瓶颈的时候，与赵老师的交流总能给我一种豁然开朗的感觉，明确了前进的方面。没有赵老师的指导，我很难完成毕业设计工作和毕业论文的撰写。

最后，我想特别感谢的是我的父母。是你们提供了我优异的学习条件，使我不用担心除学习以外的事情。每次与你们的交流都会成为我继续前进的动力。你们总要我不用担心你们，要好好照顾自己，不要熬夜。这些我都知道，我也会好好加油的，再次感谢我的父母。

美好的日子总是过的很快，大学四年，收获的不只是知识，更是面对未知挑战的勇气。人生总会遇到很多的困难和失败，坚持下去，找到属于自己的道路，不要放弃，相信自己会取得成功！