温度系统设计需要从系统设计抽象到数学模型。我们选择的抽象方法是基于UML统一建模语言的系统建模方法。对于温度控制系统问题域，使用用例图描述用户和温控系统的可能使用场景，然后设计MVC体系结构框架解决温控系统与用户的交互问题，使用类图描述如何控制温度这个问题域中出现的对象，建立了一个最基本的问题域模型。最终确定要研究的核心问题是在中央处理器上如何综合传感器发来的和用户发来的数据。这个核心问题就是所要研究的数学模型。下面详细阐述从智能温度系统中抽象数学问题的方案。

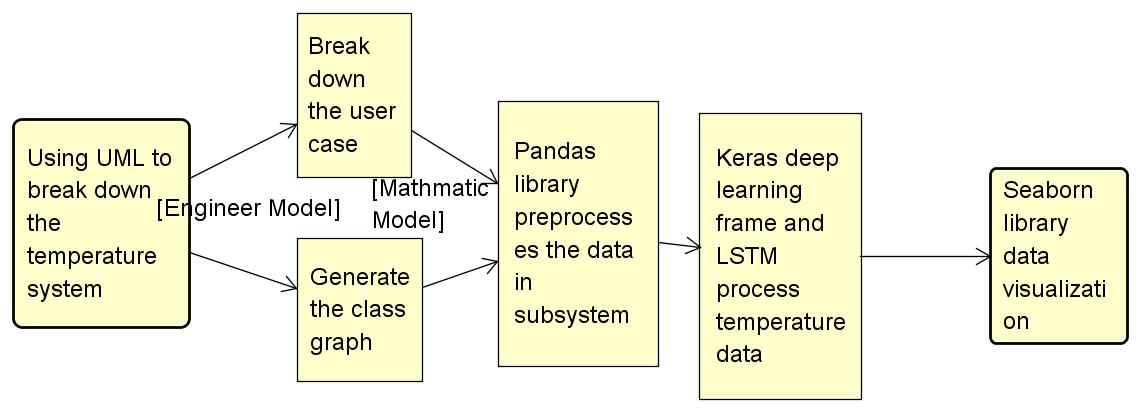


Figure 1工程化建模过程流

1、面向对象分析——用例建模

用例建模帮助我们理解需求，明确任务。用例包括智能温度系统中的场景以及外部对象。外部对象包括用户和中央处理系统。对象的交互过程通过用例完成。用户面对的场景是编程控制温度。假设用户编制的程序是能够表征温度随时间变化的函数。然后通过程序，翻译成MVC框架中的控制器所能识别的数据流，进入学习网络，激活控制器收集传感器数据，进而控制屋内的温度。节省能量、适应用户的喜好这两个用例，在某种程度上存在矛盾，一个系统在运行的过程中必须要先保证用户的满意才能节省能量，这是系统设计的基准点。

用户的喜好判断由系统完成。有关用户喜好的用例包含设置温度程序的子用例。用户编程设定的温度场景通过嵌入式终端实现，如通过手机设定温度。考虑到用户手动编程设定温度非常麻烦，而且效率极低。因此，分析设计一个用户可图形编程的界面与用户交互十分必要。这也就是下文体系结构框架里所述的界面设计。

与系统有关的用例都能和系统作用，从而感知外部的温度变化，温度变化主要从传感器中感知。其中有一个扩展用例感知的是湿度的变化，不仅需要湿度传感器，还要感知空气污染水平的传感器。系统除了要从这几个传感器中接收数据，并计算出温度应该怎么变化，而且应该能够算出在室内有多少个人，根据人数进一步调整室内温度。人数较少时系统工作强度大，人数较多时系统工作强度小。这样能够有效节省能量。

综上所述，智能温度控制系统要有一个面向用户的图形界面，能够在界面上进行显式的编程。同时系统也能够根据用户的数据，学习出一个模型，根据用户的人数，根据传感器发出来的各种数据去学习固定的模式。

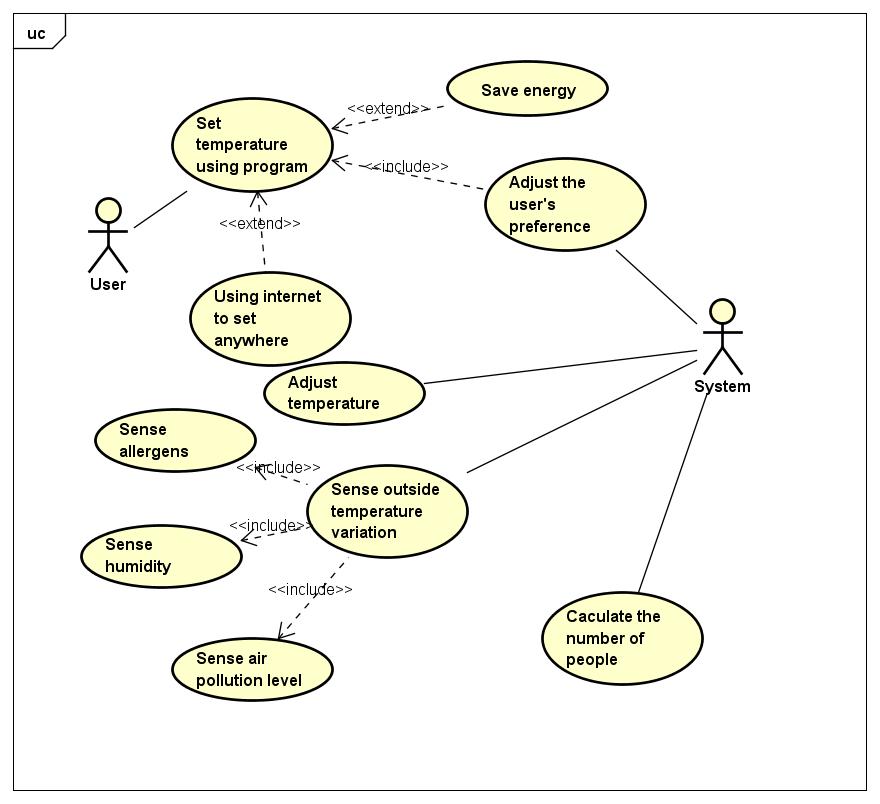


Figure 2温控系统需求分析用例（UML统一建模语言）

2、体系结构框架设计——MVC框架

我们基于MVC设计出面向用户和智能温度系统的体系结构框架。MVC框架是一个体系结构框架，由三部分组成：模型对象，视图对象和控制器对象。

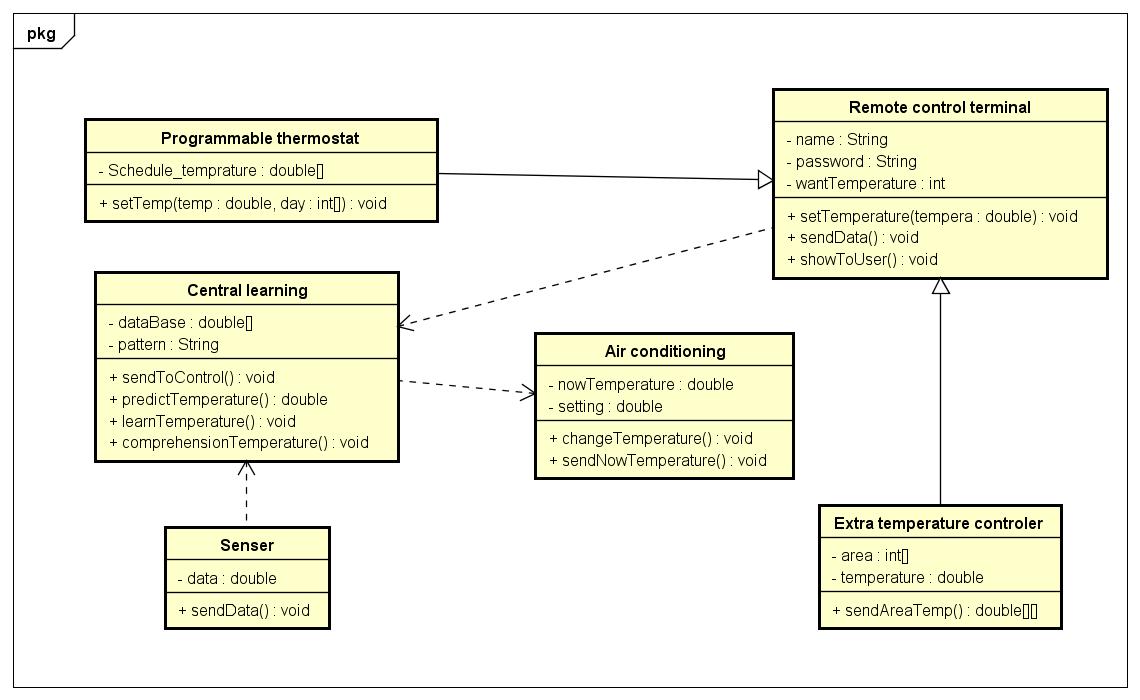
其中的控制器对象实例化为温度系统中心处理器。它控制智能家庭系统中的任务，并完成核心的计算功能。

视图对象是人机交互的窗口，当用户需要访问智能控制家居系统时，首先需要通过手机应用或者电脑端应用访问系统界面，在下面解释中我们会看到，界面应该具有可编程的特性，以便让用户去图形化编程，在控制系统的温度的同时能够方便地与系统交互。

模型对象具体指LSTM神经网络模型。我们应用Keras深度学习框架实现了这个模型，具体的模型将在后文中讨论，相关的代码以及注释作为附录。

3、数学模型生成——基于类的分析

下面根据场景建模，进一步分析需求并系统设计。类图把场景映射成抽象的对象，抽象出数学模型。程控温度设定器的实质是上述建立MVC框架中所述图形温度控制界面。程控温度器对象中的方法把用户的输入、编写的程序转换成移动数据，发送给远程终端。远程终端能够感知到消息。远程终端同时发送数据给MVC框架中的核心处理器，简而言之是一台服务器。在物联网中设置的服务器是嵌入式为主的微服务器。考虑到在建模过程中用到神经网络，在系统设计时，处理器必须预留USB神经网络加速器接口，否则使用会有较大局限。

另一方面，我们要考虑到题目中所给的一些其他限定，只有一个温度调节器和额外温度调节器的情况下，分别对系统有什么影响？额外的温度控制器对象继承于远程控制终端。

综上所述，在基于类的建模过程中，最终把智能温度系统设计这样的实际问题归结为数学问题——处理各个传感器和用户发来的数据。在接下来过程中，我们要把这个对工程的建模转化成对数学问题的建模，更详细的讨论central controler中的数学问题。

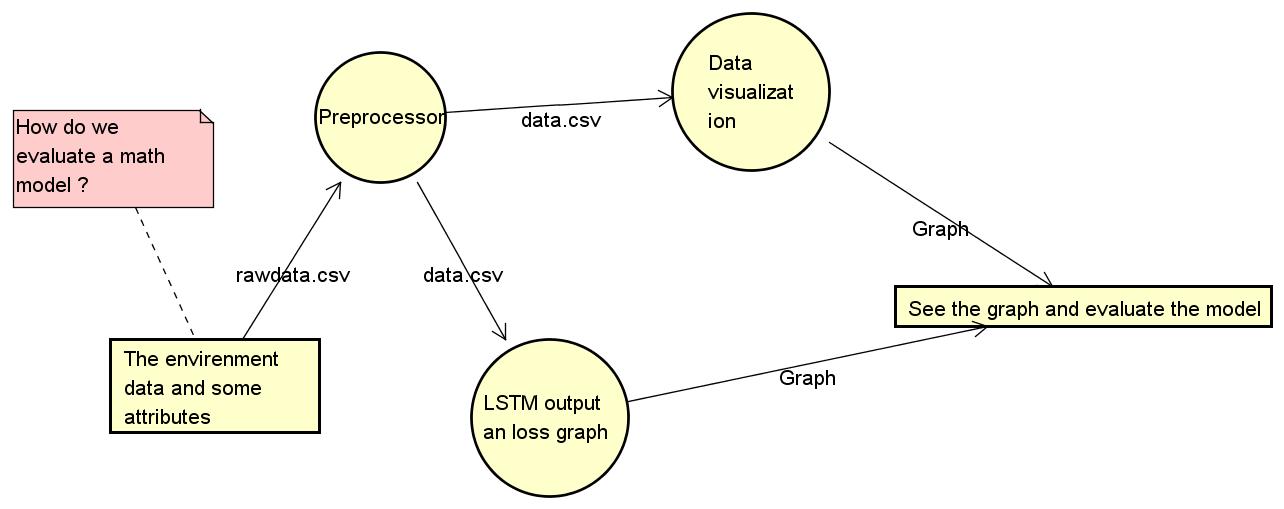
Figure 3基于类的建模

数据收集

本文使用深度学习的方法分析数据，因此需要收集较多的数据。收集到的数据有8760条，来自马塞诸塞大学实验室，反映室内室外的环境变化，以及人体的体感温度。数据集共有9个属性，分别为temperature,icon,humidity,apparentTemperature,pressure, windspeed, time, windBearing, dewpoint.通过数据集分析，找出数据间相互作用规律，进而对中心温度控制器建模，最后验证LSTM神经网络模型。

|  |  |
| --- | --- |
| **Symbol** | **Interpreter** |
| Temperature | The inside temperature |
| Icon | The weather condition |
| Humidity | The inside humidity |
| apparentTemoerature | Sendible temperature |
| pressure | Stmospheric pressure |
| windspeed | Wind Spees |
| time | Local Time |
| windBearing | Wind direction |
| dewpoint | Dew point temperature (Meteorological academic language) |

**数据预处理**

下面展现了通过构建数据流图显示需要进行的数据处理。通过对数据可视化图像的分析，找出数据的一般规律，进而利用LSTM神经网络回归数据。数据集存在如数据缺失、存在字符串属性等部分数据不够规范化的问题。

我们使用Python语言的Pandas库和sklearn库进行数据预处理，运行环境是Linux。代码和注释在附录中，分为两个文件解决不同类型的数据正则化问题。

缺失数据的解决方案。缺失数据集中在三列上，缺失量不大，因此我们采用缺失值填充方法而未采用缺失值剔除方法。根据学界有关学者的研究，连续型数据用该属性的平均值去填充，而离散属性使用众数填充数据。鉴于缺失数据全部为连续型的数据，我们通过平均值法填充。

用户数据的填充方案。原始数据集并未提供用户根据具体环境做出的反映，因此我们作为模拟用户尝试填充了小部分调节温度，反映我们自身对于该项环境下的体验。经过80项人工判断填充，我们大致掌握了温度的调节规律。并依规律大规模人工向下填充。经过方案可行性分析，这种人工填充方法模拟了用户的实际行为，能够较好地反映真实系统下用户的有关操作。

非数值属性的编码问题。原始数据中，表现室外天气情况的icon字段出现了{'clear-night', 'wind', 'partly-cloudy-night', 'rain', 'cloudy', 'fog', 'partly-cloudy-day', 'clear-day', 'snow'}这样的字符串。需要将其映射成为整数。考虑到最终用户的输入数据有可能与光线强弱有关，我们按照不同天气情况对应的光线强弱由弱到强依次编码为0-9.为下一步矩阵计算创造数值方面的条件。

时间序列的生成方案。因为数据集提供了时间字段，因此数据预处理时需要根据时间字段排序，以反映时间序列的特征。由于LSTM神经网络是时间序列的网络，我们需要在原始数据的基础上建立滑动窗口。这里我们设置的滑动窗口长度是2，每次窗口滑动一个单位。这里需要特别注意，建立了时间序列后，原始的数据从二维矩阵变换为三维张量。

数据的归一化问题。神经网络的基本原理是把线性不可分的数据映射到高维空间，增强数据的线性可分性。数据归一化可以让数据集中分布在较小的范围内，既可以增强激活函数的非线性映射能力，又可以增强数据集的表现效果。避免数据在经过多层网络时因为计算机浮点运算能力的问题出现梯度消失或者梯度爆炸问题。我们使用的归一化过程如下:



**数据可视化分析**

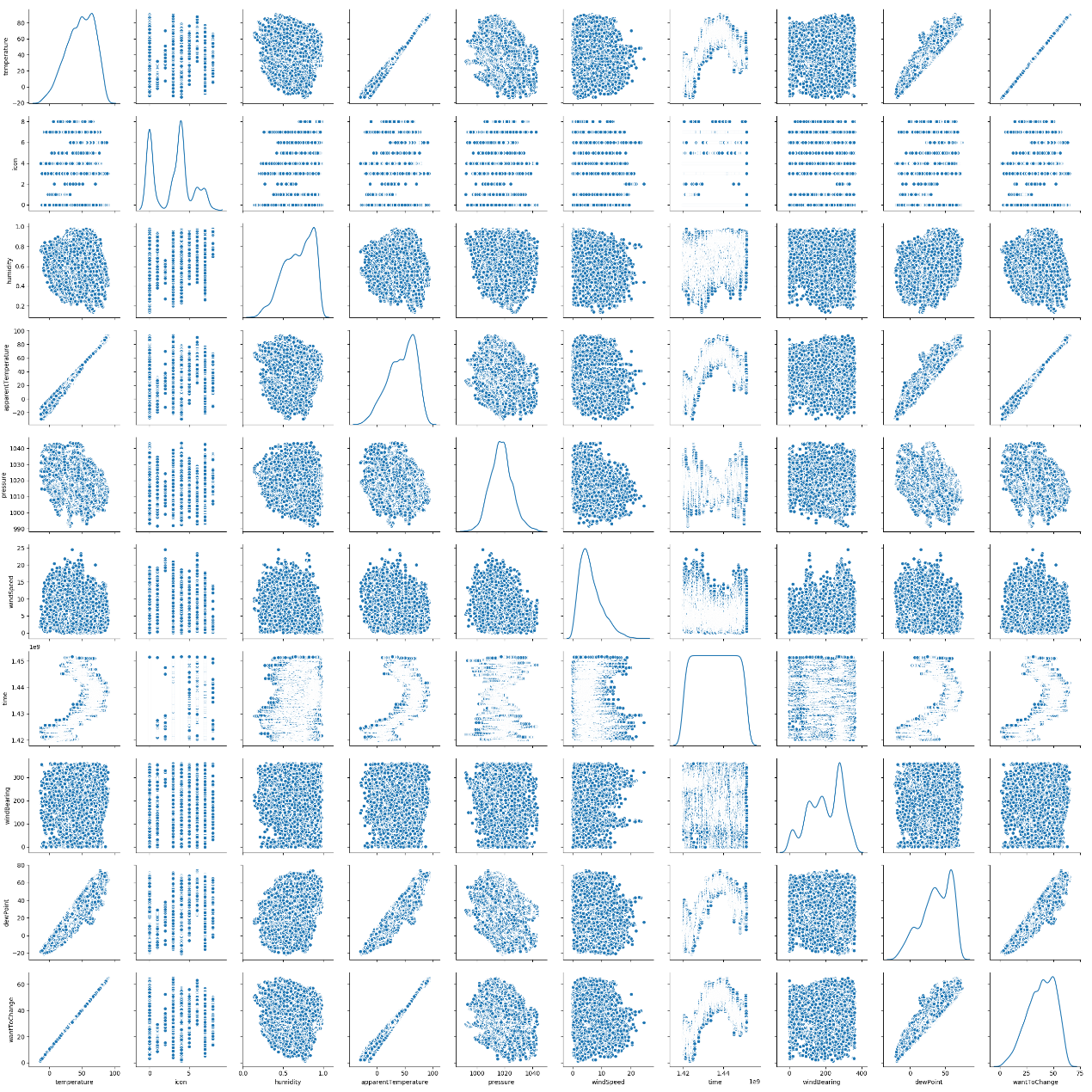
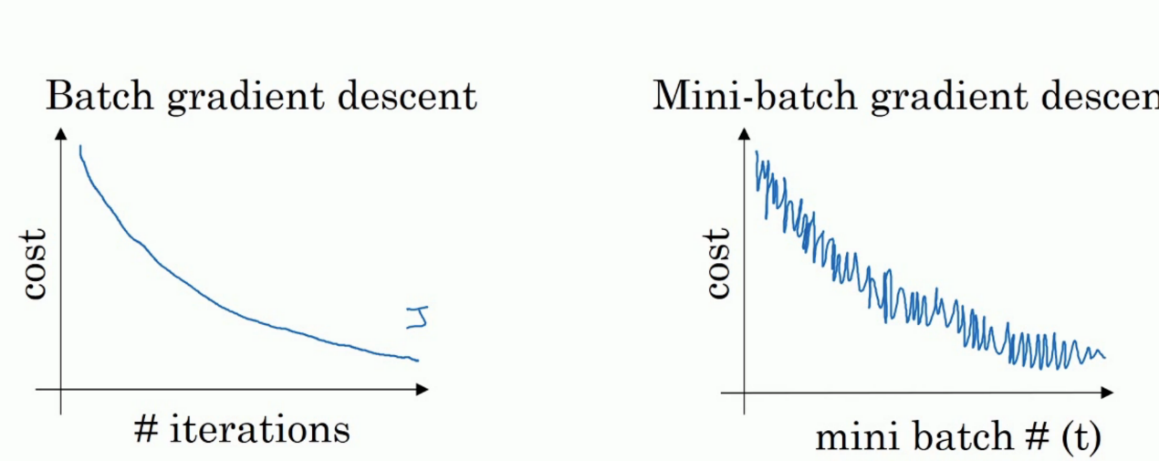


Figure 5 The scatter and distribution Graph

上图展示的是数据集的分布情况，横轴从左到右依次为temperature, icon, humidity, apparentTemperature, pressure, windspeed, time, windBearing, dewpoint, wantToChange.纵轴从上到下与横轴相同。图中对角线上展示的是不同属性的分布情况。可以清晰的发现，数据的分布大多都向正态分布靠近，这也与大数定律的结果相吻合，实验次数越多，分布就向正态分布靠近。

散点图中的time属性具有相当明显的周期性，这说明温度和环境的有关数据和信息都在随着时间的变化呈现周期性的变迁。而温度相关的属性，apparentTemperature, temperature, wantToChange。

LSTM方法建模

数据的正则化方法。为了防止模型过拟合，需要对数据进行正则化处理，这里我们采用数量为20的minibatch方法。如图所示，在不使用minibatch的传统梯度下降算法中，损失函数的下降速率并不规则，这是由于单个梯度下降具有随机性，很难精准地向总体均方误差最小化的方向移动。而minibatch往往具有代表性，能够反映数据总体在误差上的表现。

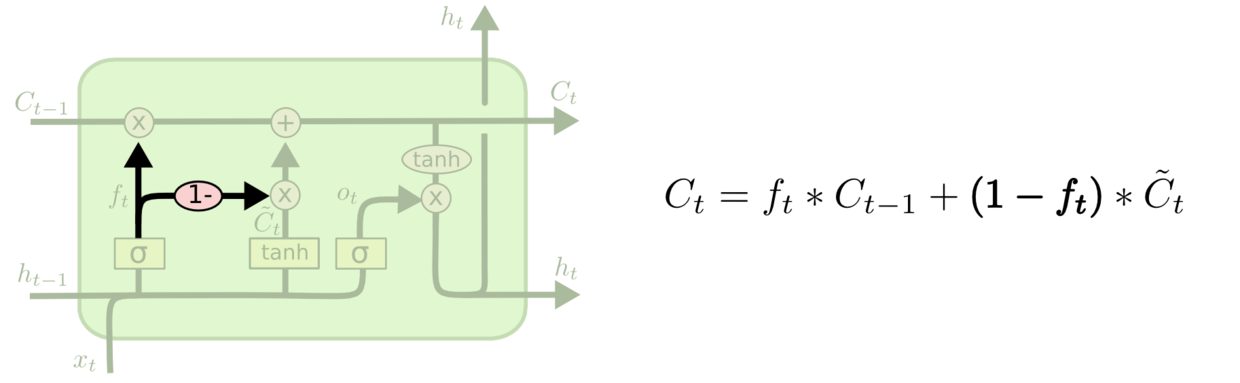
为了精准比较误差，我们控制LSTM网络返回整个序列而不是一个序列的单个输出。

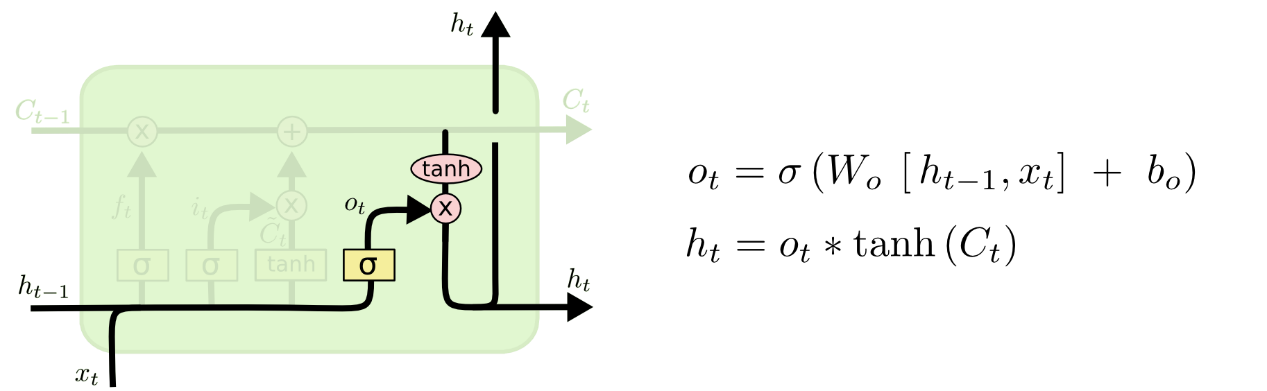
模型选择方面，全连接型神经网络和卷积神经网络不适合。卷积神经网络一般是用在图像的处理里面，解决高维数据压缩问题，但是这个数据集维度仅为9个属性。全连接神经网络无法很好的处理时间序列问题。朴素贝叶斯方法对于样本数量较少的情况下拟合效果好，但是缺乏时间序列的判断性。递归神经网络已经在很多自然言语处理任务中 表现出很强的学习能力，特别是能够对序列数据进行很好的建模，充分挖掘出序列中隐藏的信息。如RNN循环神经网络，能够表现时间序列的变化，但是因为缺乏控制流。因此不能较好地对长时数据建模。

标准的LSTM神经网络模块有4个可交互的神经网络层组成，避免在迭代次数增加的情况下出现梯度消失或者爆炸的问题，保证模型得到充分的训练，从而提高模型的预测准确度。其中σ 为sigmoid，tanh为tanh函数。

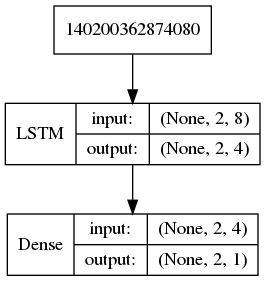


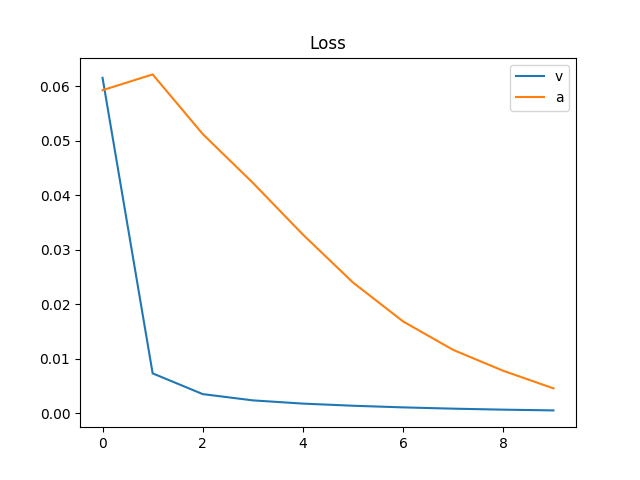


每个神经网络模块可以看作一个Memory cell，其中分别表示第t时刻的输入单元、细胞状态和输出单元，而分别表示第t-1时刻的细胞状态和输出单元。每个“记忆细胞”包含三个“门”结构，分别是：“Forget Gate”,”Input gate”,”Output gate”.”Forget gate “决定从细胞状态中忘记哪些信息；”输入门“决定将要更新哪些信息到细胞状态中；”输出门“决定将要输出哪些信息。“输入门”层采用与“遗忘门”层耦合的方式对向 “记忆细胞”注入的信息进行控制，共同决定细胞状 态的更新。用 Ct表示将要注入“记忆细胞”新信息 的向量，它为tanh神经网络层的输出.下面的公式中，分别用表示Forget gate中需要遗忘信息的比率，表示当前输入的信息需要传递给细胞状态的比率。与此类似，表示经过sigmoid函数放缩过后的预输出，其中sigmoid函数的作用对象是输入数据的一个线性映射。表示经过前一个细胞信息与当前输入综合后的信息流，是预输出与细胞控制流的乘积。

进一步解释模型，tanh起到信息流的非线性映射作用，sigmoid函数起到了信息流开关的控制作用。而sigmoid和tanh函数在自变量较大时都会出现梯度消失的现象，而且他们的输出是有有界的，这进一步论证了我们在数据预处理阶段对温度集所有数据按列整体归一化的原因。必须确保网络输出能够尽可能映射成为实际数据，也要确保控制流能够较好地进行放缩。如果数据做归一化，则实际应用时必须要进行反归一化才能输出正确结果。

下面是我们基于Keras框架设计温度集上LSTM的网络结构：其中140200362874080是输入数据元素的总数,含属性和迭代次数.我们使用的更详细的网络结构参见附录7.



图中所示为LSTM模型拟合过程中的损失下降曲线，V stands for validation set，A stands for train set。我们采用的验证集与样本总体的比率是0.33.可以看出，使用minibatch方法，在迭代少于10次时下降损失已经较低。我们认为，这是因为模拟用户温度的设置偏向于线性设置，因此模型能够在较少的迭代次数下快速完成用户数据的分析。

因为现在并不知道用户设定的实际温度是多少。所以接下来需要用自适应方法优化模型。我们仍然使用传统的均方误差作为损失函数。除了使用minibatch外（在数据预处理部分讲述），还使用Adagrad优化器加速模型的收敛速度。





随机梯度下降方法中，对于每一个参数的训练使用了相同的学习率α。Adagrad算法能够在训练中自动的对learning rate进行调整，对于出现频率较低参数采用较大的α更新；相反，对于出现频率较高的参数采用较小的α更新。因此，Adagrad非常适合处理稀疏数据，而温度数据集中存在的离散数据满足了稀疏数据的条件，适合处理此数据集。

我们设为第t轮第i个参数的梯度，即。因此，SGD中参数更新的过程可写为：

Adagrad在每轮训练中对每个参数的学习率进行更新，参数更新公式如下：



其中，*Gt*∈R*d*×*d*为对角矩阵，每个对角线位置（i，i）为对应参数*θi*从第1轮到第t轮梯度的平方和。ϵ是平滑项，用于避免分母为0，一般取值1。Adagrad的缺点是在训练的中后期，分母上梯度平方的累加将会越来越大，从而梯度趋近于0，使得训练提前结束，但是在温度训练集上我们并没有观察到这样的情况出现。

**LSTM的工程化应用**

上述理论部分已经说明，LSTM用于温度数据分析在理论上是可行的，本节说明LSTM用于Central learning的子系统仍然可以提供给用户最佳体验。

假设物联网系统使用的是Intel up square board作为中心处理器。可以安装tenserflow-lite版本运行上述神经网络，但是并不时时刻刻运转，而是首先记录用户输入数据、输入时间以及传感器数据。等待用户输入的数据量达到一个minibatch然后集中送入LSTM神经网络。这样设计的优点在于可以在优化梯度下降的同时改善功耗，达到节能的目的。此时用户可以通过手机应用访问物联网中心，在手机上编程设定温度，达到温度控制的效果。物联网控制中心的控制子系统随时根据传感器发来的数据做出应答，此时控制子系统使用的是已经建好的数据库，并非正在学习中的。实现控制与计算分离，最大程度降低功耗。

第二问——分布式温度控制器建模

模型的分析

在任务一中我们研究了使用LSTM方法建立单中心温度控制器模型，并且研究了在一个温度控制器的情况下如何调节室内温度。我们将研究在多个控制器的情况下，如何建立温度分布模型。

这里不妨借用任务一的模型。首先定义中央控制器，中央控制器是当前人所在房间的温度控制器，这个控制器产生的数据，即是我们通过第一问中LSTM模型计算出来的数据。而其余的温度控制器定义为边缘控制器.在人较多，并且分布在不同的房间的情况下，需要设定多个不同的中心控制器，同时有多个不同的边缘控制器。这种情况定义为多中心温度控制器情形。

模型研究边缘控制器的温度计算问题。经过实验，我们认为边缘控制器的温度服从一定的分布。随着距离主控制器的增加，边缘控制器的设定温度更趋近于室温。与我们实验最初始的假设有所不同，最初我们认为温度下降模型可以通过KNN算法进行建模。随着实验的深入，我们发现KNN主要针对离散化的数据聚类，而温度控制器可以通过设定温度下降速度的大小去拟合温度这个连续属性。因此，KNN与连续下降相比不适用于此模型。与第一问中定义的均方误差损失相似，在此处我们仍然应用欧氏距离定义与中心控制器的距离，并采用连续方法计算边缘控制器的设定温度。

模型的实现

考虑到有多个中心温度控制器的存在，我们认为在这种情况下，任意一个边缘控制器所对应的最近的中心温度控制器就是他所要计算的欧式距离。在这种假设下，我们把一个多中心控制器问题化为一个单中心控制器的边缘分布问题。即对于每一个边缘控制器，程序只需计算它与最近中心控制器的距离。

下面我们将针对更普遍的情况，对多中心控制器的分布问题实验，给出具体的公式以及实验的结果。与系统分析阶段所说的面向对象的分析相似，在这个阶段仍然用面向对象方法建模。

编程语言选择面向对象语言Java。Java语言有很好的面向对象特性，能够较好地勾画出在现实中不同类之间的关系，以及他们内部的成分构造。这里我们研究的对象仅限于多余的控制器中心控制器以及整个房屋温度控制器设定温度的分布问题。  
 我们模拟了有37个控制器的大房间模型，这个大房间中我们设定有两个中心控制器，在每个中心控制器的房间里面有一个人。中心控制器的模型使用字符数组存放，下面我们利用C/C++语言模型可视化(附录4)，其中红色表示中心控制器的位置，而蓝色表示边缘控制器的位置。左上角坐标为(0,0),作图是按1:30对房间字符数组放缩表示.

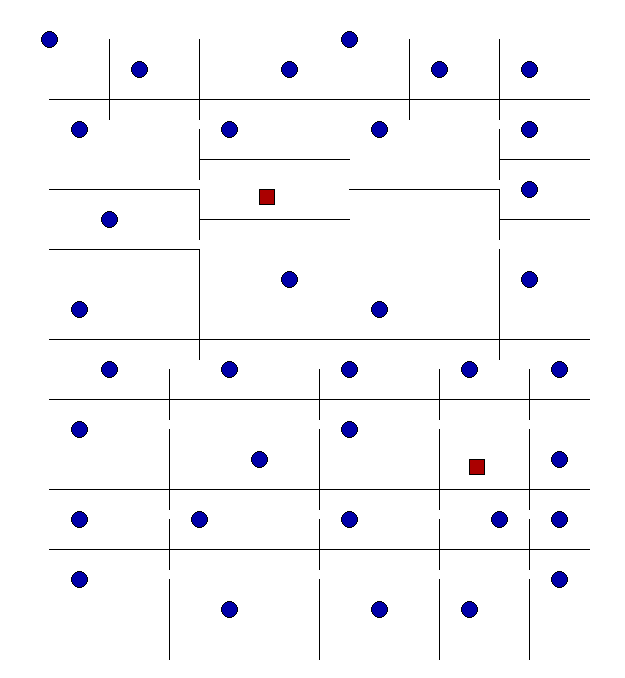


Figure 1 A probable room model

Java语言代码中的具体的算法用s代表边缘控制器的位置，用p代表人和中心控制器的位置。更详细的解释在Java语言代码中均给出了相应的注释以及自解释的变量函数说明，这些算法资料均在附录3中列出。在算法中，我们设定了declinerate变量表示各个房间中控制器相对中间控制器温度趋近于室温速率的大小。declinerate变量越大，表示各个房间中控制器越趋近于中间控制器的设定温度。

根据各个房间的距离，随中心控制器距离增大而温度趋近于室温，我们设定了与距离呈负相关的一组函数去描述在不同季节边缘控制器的设定温度。

算法如下:

|  |
| --- |
| **distance**= *min*(*sqrt*((x-person\_x)\*(x-person\_x)+(y-person\_y)\*(y-person\_y)), *sqrt*((x-person2\_x)\*(x-person2\_x)+(y-person2\_y)\*(y-person2\_y)))  **if in winter then**  setTemperature=min(centralSetTemperature,roomTemperature+(centralSetTemperature-roomTemperature)\*(1.0/(1+distance\*distance))\*declineRate)  **else if in summer then**  setTemperature=max(centralSetTemperature,roomTemperature+(centralSetTemperature-roomTemperature)\*(1.0/(1+distance\*distance))\*declineRate) |

下面介绍我们建模的工程学原理，物联网是一个典型的嵌入式系统，而嵌入式系统要求高实时性，并且在一个资源有限的环境下能够快速计算出所得的结果对实体进行控制。我们的公式尽可能使用有限个四则运算，就可以得到边缘温度相对准确的结果。

模型简化主要体现在边缘传感器与多中心传感器的距离选择仅取决于与它最近的中心传感器，与它较远的中心传感器，虽然和有联系，但是联系相对较弱。从增强嵌入式系统可靠性的角度，我们忽略了这种联系，因此这种简化是合理的。从系统设计的角度，嵌入式系统设计的第二步需要划分软硬件界面，在此我们对软件设计部分做相应的简化处理，使得模型更为简化，也使得硬件的成本有所下降。

需要特别说明，与第一个任务有所不同，在第一个任务中，我们利用中央处理器作为服务器处理数据，中央处理器的运算速度快，因此可以实现LSTM大规模神经网络计算。但是在第三个任务中，需要分别依据每个传感器的数据去计算它所在的房间的空调控制器的温度，它所依据的实体是嵌入式微控制器，嵌入式微控制器不适宜进行大规模运算。这个问题属于边缘计算问题，对于边缘计算问题，我们不能够使用大规模神经网络或者是很复杂的公式进行计算，否则会使系统的可靠性和易用性大幅下降。这就是这一小问与第一个任务之间最大的不同。也是我们算法设计和模型构造的出发点。

模型测试与验证

在测试模型中我们设定中心控制器根据LSTM模型计算出的温度是26℃，此时室温为34℃。根据各个房间中不同控制器的坐标，我们给出了相应计算的距离。测试结果见附录6.

测试后的结果约37个,附录5显示了只有一个人在坐标（7，5）时的情况，附录6显示了两个人分别在坐标（7，5）和（14，14）时的情况。在测试过程中，用户可以自行输入declinerate来控制不同房间距离中心控制器距离的增加，而使设定温度下降或增加的快慢。与第一问类似，我们认为,变化速率可以使用户在手机上设定编程实现的，或者作为LSTM网络的参数输入，系统自动计算出相应的declinerate，但是前提是用户必须提供足够的学习数据支撑网络运行。  
 作为模拟用户我们认为这组输出数据较好地反映了我们所期望的温度设定结果。实现了任务中所要求的在多传感器和多控制调节器下房间温度的设定建模。

As a simulated user, we believe that this set of output data better reflects the temperature setting results we expect. We realize the modeling of the room temperature setting under the multi-sensor and multi-control regulators required in the task.

**Appendix**

Disclaimer: Please refer to my github for source code. Due to page limit, we only show the core code. The relevant data set is also available on my gihub. https://github.com/changkaiyan/mathmodel

|  |
| --- |
| Appendix 1: Python code on Data preprocessing (Based on Pandas library) |
| #--------------------------------------------------------  # Copyright 2018 Kaiyan Chang, Kaiyuan Tian, Ruilin Chen  # For the data preprocessing  # Python 3.6 for Linux. Only run in Linux system.  #--------------------------------------------------------  #Read data from csv  raw\_data=pd.read\_csv("datar.csv")  temp=set(raw\_data['icon'])  print(temp)  #Choose a best index for icon  size\_mapping={label:index for index, label in enumerate(temp)}  raw\_data['icon']=raw\_data['icon'].map(size\_mapping)  #Caculate the user defined temperature  sigma=0.2  mu=0  wanttochange=0.607\*raw\_data['temperature'].values+10.092+np.random.normal(mu,sigma,raw\_data.shape[0])  #Fill NaN  raw\_data['wantToChange']=wanttochange  raw\_data[raw\_data.isnull().values==True]  raw\_data['pressure']=raw\_data['pressure'].fillna(raw\_data['pressure'].mean())  raw\_data['windBearing']=raw\_data['windBearing'].fillna(raw\_data['windBearing'].mean())  raw\_data['windSpeed']=raw\_data['windSpeed'].fillna(raw\_data['windSpeed'].mean())  #Sort by time  raw\_data.sort\_values(by="time")  raw\_data.to\_csv('datanoraw.csv',index=False)  sns.pairplot(raw\_data,diag\_kind='kde')  plt.savefig('datasetprocessingpairplot1')  raw\_data |

|  |
| --- |
| Appendix 2: Python code on Data regression (Based on Keras and tensorflow library) |
| #--------------------------------------------------------  # Copyright 2018 Kaiyan Chang, Kaiyuan Tian, Ruilin Chen  # For the data regression  # Python 3.6 for Linux. Only run in Linux system.  #--------------------------------------------------------  look\_back=2 # A length of time window  epochs=1000 # training Epochs  batch\_size=20 #In minibatch the batch size is 20  #Create a time windows and extends matrix to tensor  def create\_dataset(dataset):  dataX, dataY = [],[]  global look\_back  for i in range(len(dataset) - look\_back - 1):  x= dataset[i:i+look\_back, :dataset.shape[1]-1]  dataX.append(x)  y = dataset[i:i+look\_back, dataset.shape[1]-1:dataset.shape[1]]  dataY.append(y)  dataX=np.reshape(np.array(dataX),((len(dataset) - look\_back - 1),look\_back,dataset.shape[1]-1))  dataY = np.reshape(np.array(dataY), ((len(dataset) - look\_back - 1),look\_back,1))  return dataX, dataY  #Define batch normalization class  scaler\_x = MinMaxScaler()  scaler\_y=MinMaxScaler()  scaler=MinMaxScaler()  #Read data from .csv  data\_set = read\_csv('datanoraw.csv', header=0, index\_col=0)  data\_set=data\_set.values.astype('float32')  #Batch normalization  Train\_x = scaler\_x.fit\_transform(data\_set[:,:data\_set.shape[1]-1])  Train\_y = scaler\_y.fit\_transform(data\_set[:,data\_set.shape[1]-1:data\_set.shape[1]])  data\_set=np.hstack((Train\_x,Train\_y))  train = data\_set[0:, :]  X\_train, y\_train = create\_dataset(train)  #Define model using keras  model = Sequential()  model.add(LSTM(units=4, input\_shape=(look\_back, data\_set.shape[1]-1),return\_sequences=True))  model.add(Dense(units=1))  model.compile(loss='mean\_squared\_error', optimizer='adam')  #Run the model  history=model.fit(X\_train, y\_train, epochs=epochs, batch\_size=batch\_size, verbose=2,validation\_split=0.33)  #Model visulization  model.summary()  plt.plot(history.history['loss'], label='train')  plt.legend('train')  plt.plot(history.history['val\_loss'], label='test')  plt.legend('validation')  plt.title('Loss')  plt.show()  plt.savefig('loss 1-10') |

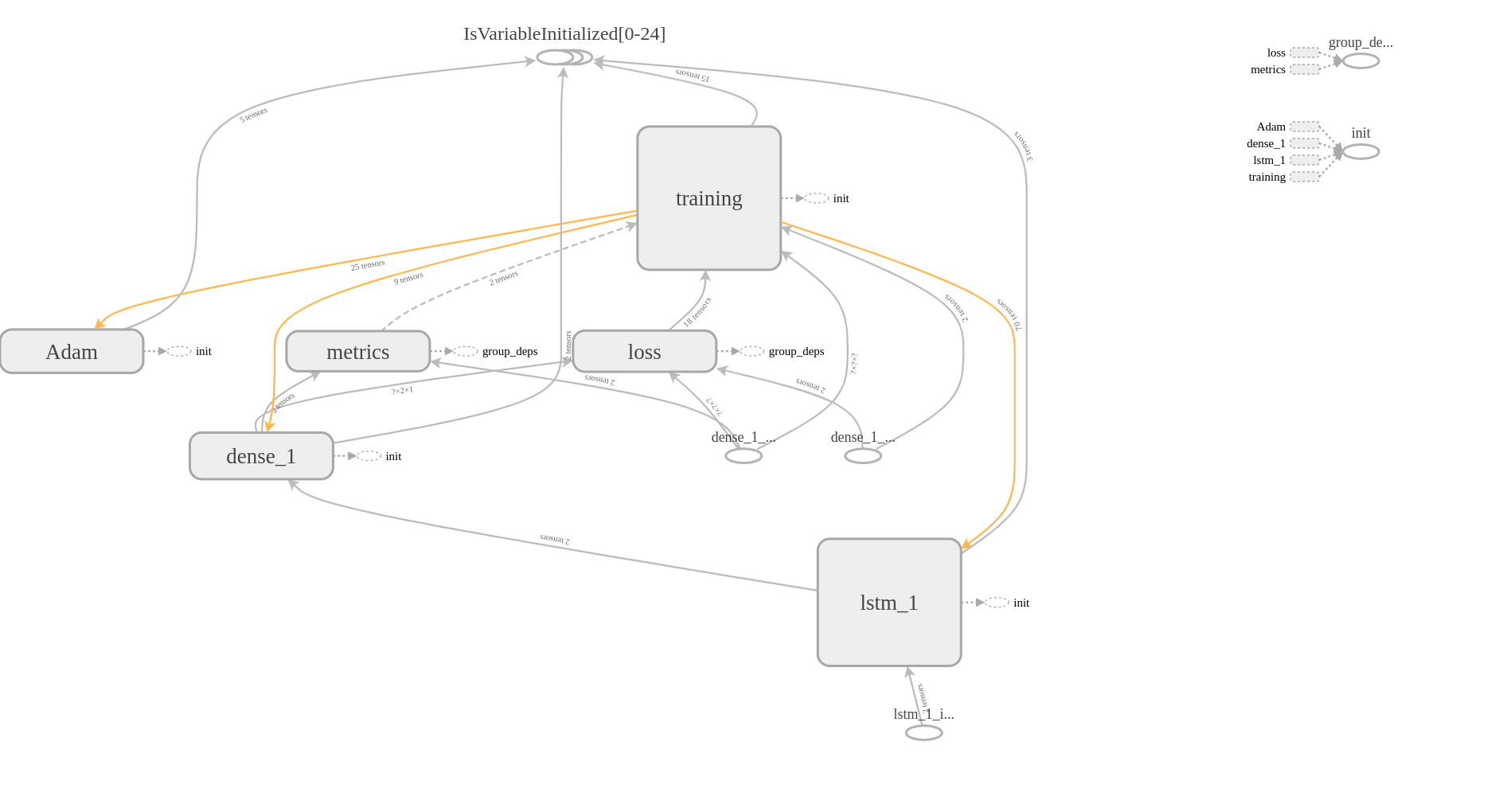
|  |
| --- |
| Appendix 3: Java Code for room model |
| */\*\*  \** ***@file*** *myRoom.java  \** ***@author*** *changkaiyan (changkaiyan@std.uestc.edu.cn)  \** ***@brief*** *Multi-central senser caculation  \** ***@copyright*** *Copyright 2018 Kaiyan Chang, Kaiyuan Tian, Ruilin Chen  \*  \*/* **public class** myRoom{  *//Person is in (7,5) (14,14)* **private char**[][] **room**;//The data is too heavy you can see my github  **private** Senser[] **roomsenser**=**new** Senser[64];  **private int sensercount**=0;  **public static void** main(String[] args){  myRoom mr=**new** myRoom();mr.setSenser();  mr.getRoomTemperature();   }  **public void** setSenser()  {  **for**(**int** i=0;i<**room**.**length**;++i)  {  **for**(**int** j=0;j<**room**[i].**length**;++j)  {  **if**(**room**[i][j]==**'s'**)  {  **roomsenser**[**sensercount**]=**new** Senser(i,j);  **sensercount**++;  }  }  }  }  **public void** getRoomTemperature()  {  **for**(**int** i=0;i<**sensercount**;++i)  **roomsenser**[i].printData();  } } *//I assume the room temperature is same in all rooms* **class** Senser{  **private int x**;  **private int y**;  **private double distance**;  **private double setTemperature**;  *//Multi-central-Senser* Senser(**int** x,**int** y)  {  **int** person\_x=7; **int** person\_y=5;  **int** person2\_x=14; **int** person2\_y=14;  **double** centralSetTemperature = 26.0;  **double** roomTemperature=35;  **this**.**x**=x; **this**.**y**=y;  **double** declineRate=35;*//The recomment data* **this**.**distance**=Math.*min*(Math.*sqrt*((x-person\_x)\*(x-person\_x)+(y-person\_y)\*(y-person\_y)),  Math.*sqrt*((x-person2\_x)\*(x-person2\_x)+(y-person2\_y)\*(y-person2\_y)));  **if**(roomTemperature<centralSetTemperature)*//Often Winter* **setTemperature**=Math.*min*(centralSetTemperature,roomTemperature+(centralSetTemperature-roomTemperature)\*(1.0/(1+**this**.**distance**\***this**.**distance**))\*declineRate);  **else***//Often Summer* **setTemperature**=Math.*max*(centralSetTemperature,roomTemperature+(centralSetTemperature-roomTemperature)\*(1.0/(1+**this**.**distance**\***this**.**distance**))\*declineRate);  }  **public void** printData()  {  System.***out***.println(**"The senser location : ("**+**x**+**","**+**y**+**") Distance is "**+**distance**+**" The auto setting temperature is:"**+**setTemperature**);  } } |

|  |
| --- |
| Appendix 4: C/C++ code for room model viusalization |
| /\*\*  \* @file C\_code\_model.cpp  \* @author changkaiyan (changkaiyan@std.uestc.edu.cn)  \* @brief Multi-central senser Model Visualization  \* @copyright Copyright 2018 Kaiyan Chang, Kaiyuan Tian, Ruilin Chen  \*  \*/  int main()  {      char room[18][20] ;//The data is too heavy you can see my github      initgraph(1024, 768);setorigin(100, 100);      setbkcolor(WHITE);cleardevice();setcolor(BLACK);      for(int i=0;i<18;++i)          for (int j = 0; j < 20; ++j)          {              if (room[i][j] == 's')              {                  setfillcolor(BLUE);                  fillcircle(i \* 30, j \* 30, 8);              }              else if (room[i][j] == 'p')              {                  setfillcolor(RED);                  fillrectangle(i \* 30, j \* 30, i \* 30 + 15, j \* 30 + 15);              }              else if (room[i][j] == '|')              {                  setlinecolor(BLACK);                  line(i \* 30, j \* 30, i \* 30+30, j \* 30 );              }              else if (room[i][j] == '-')              {                  setlinecolor(BLACK);                  line(i \* 30, j \* 30, i \* 30, j \* 30+50 );              }          }      system("Pause");closegraph();  } |

|  |
| --- |
| Appendix 5: Room model only one person(Run in Java 10) |
| Loc : (0,0) Distance 8.602325267042627 Setting temper:30.80  Loc : (1,3) Distance 6.324555320336759 Setting temper:27.32  Loc : (1,9) Distance 7.211102550927978 Setting temper:29.06  Loc : (1,13) Distance 10.0 Setting temper:31.88  Loc : (1,16) Distance 12.529964086141668 Setting temper:33.01  Loc : (1,18) Distance 14.317821063276353 Setting temper:33.47  Loc : (2,6) Distance 5.0990195135927845 Setting temper:26.00  Loc : (2,11) Distance 7.810249675906654 Setting temper:29.92  Loc : (3,1) Distance 5.656854249492381 Setting temper:26.00  Loc : (5,16) Distance 11.180339887498949 Setting temper:32.50  Loc : (6,3) Distance 2.23606797749979 Setting temper:26.00  Loc : (6,11) Distance 6.082762530298219 Setting temper:26.71  Loc : (6,19) Distance 14.035668847618199 Setting temper:33.41  Loc : (7,14) Distance 9.0 Setting temper:31.16  Loc : (8,1) Distance 4.123105625617661 Setting temper:26.00  Loc : (8,8) Distance 3.1622776601683795 Setting temper:26.00  Loc : (10,0) Distance 5.830951894845301 Setting temper:26.00  Loc : (10,11) Distance 6.708203932499369 Setting temper:28.15  Loc : (10,13) Distance 8.54400374531753 Setting temper:30.74  Loc : (10,16) Distance 11.40175425099138 Setting temper:32.60  Loc : (11,3) Distance 4.47213595499958 Setting temper:26.00  Loc : (11,9) Distance 5.656854249492381 Setting temper:26.00  Loc : (11,19) Distance 14.560219778561036 Setting temper:33.52  Loc : (13,1) Distance 7.211102550927978 Setting temper:29.06  Loc : (14,11) Distance 9.219544457292887 Setting temper:31.34  Loc : (14,19) Distance 15.652475842498529 Setting temper:33.72  Loc : (15,16) Distance 13.601470508735444 Setting temper:33.31  Loc : (16,1) Distance 9.848857801796104 Setting temper:31.79  Loc : (16,3) Distance 9.219544457292887 Setting temper:31.34  Loc : (16,5) Distance 9.0 Setting temper:31.16  Loc : (16,8) Distance 9.486832980505138 Setting temper:31.54  Loc : (17,11) Distance 11.661903789690601 Setting temper:32.70  Loc : (17,14) Distance 13.45362404707371 Setting temper:33.27  Loc : (17,16) Distance 14.866068747318506 Setting temper:33.58  Loc : (17,18) Distance 16.401219466856727 Setting temper:33.83 |

|  |
| --- |
| Appendix 6: Room model for two persons(Run in Java 10) |
| Loc : (0,0) Distance 8.602325267042627 Setting temper:30.80  Loc : (1,3) Distance 6.324555320336759 Setting temper:27.32  Loc : (1,9) Distance 7.211102550927978 Setting temper:29.06  Loc : (1,13) Distance 10.0 Setting temper:31.88  Loc : (1,16) Distance 12.529964086141668 Setting temper:33.01  Loc : (1,18) Distance 13.601470508735444 Setting temper:33.31  Loc : (2,6) Distance 5.0990195135927845 Setting temper:26.00  Loc : (2,11) Distance 7.810249675906654 Setting temper:29.92  Loc : (3,1) Distance 5.656854249492381 Setting temper:26.00  Loc : (5,16) Distance 9.219544457292887 Setting temper:31.34  Loc : (6,3) Distance 2.23606797749979 Setting temper:26.00  Loc : (6,11) Distance 6.082762530298219 Setting temper:26.71  Loc : (6,19) Distance 9.433981132056603 Setting temper:31.50  Loc : (7,14) Distance 7.0 Setting temper:28.70  Loc : (8,1) Distance 4.123105625617661 Setting temper:26.00  Loc : (8,8) Distance 3.1622776601683795 Setting temper:26.00  Loc : (10,0) Distance 5.830951894845301 Setting temper:26.00  Loc : (10,11) Distance 5.0 Setting temper:26.00  Loc : (10,13) Distance 4.123105625617661 Setting temper:26.00  Loc : (10,16) Distance 4.47213595499958 Setting temper:26.00  Loc : (11,3) Distance 4.47213595499958 Setting temper:26.00  Loc : (11,9) Distance 5.656854249492381 Setting temper:26.00  Loc : (11,19) Distance 5.830951894845301 Setting temper:26.00  Loc : (13,1) Distance 7.211102550927978 Setting temper:29.06  Loc : (14,11) Distance 3.0 Setting temper:26.00  Loc : (14,19) Distance 5.0 Setting temper:26.00  Loc : (15,16) Distance 2.23606797749979 Setting temper:26.00  Loc : (16,1) Distance 9.848857801796104 Setting temper:31.79  Loc : (16,3) Distance 9.219544457292887 Setting temper:31.34  Loc : (16,5) Distance 9.0 Setting temper:31.16  Loc : (16,8) Distance 6.324555320336759 Setting temper:27.32  Loc : (17,11) Distance 4.242640687119285 Setting temper:26.00  Loc : (17,14) Distance 3.0 Setting temper:26.00  Loc : (17,16) Distance 3.605551275463989 Setting temper:26.00  Loc : (17,18) Distance 5.0 Setting temper:26.00 |

Appendix 7： The LSTM net frame we set.(Based on tensorboard)



**Appendix 8：Data sets**

datanoraw.csv run\_.-tag-loss \_train.csv run\_.-tag-val\_loss.csv

You can find them on my GitHub