中国研究生创新实践系列大赛

“华为杯”第十八届中国研究生数学建模竞赛

学　　校 _____

参赛队号 _____

队员姓名 _____

_____

_____

中国研究生创新实践系列大赛
“华为杯”第十八届中国研究生
数学建模竞赛

题 目 **空气质量预报二次建模**

摘 要：

本文基于 WRF-CMAQ 模型的预测结果，在数据进行处理和实测数据进行整合的基础上，分别构建了基于遗传模拟退火算法的 FCM 聚类模型、支持向量机分类算法模型、基于循环神经网络(RNN)的预测模型和基于有向图传播网络的区域协同预报模型。本文使用 MATLAB 和 Python 编程，对空气质量进行二次建模，以期得到更准确的污染物浓度预报。

针对问题一，要求计算监测点指定日期的每天实测的空气质量指数（AQI）和对应的首要污染物，首先需要根据相关技术规定计算出各污染物的空气质量分指数（IAQI），其中臭氧需要采用滑动平均的方法进行计算，取各污染物 IAQI 中的最大值作为 AQI，并确定首要污染物。

针对问题二，要求根据对污染物的影响程度，对气象条件进行分类，并分析其特征。本文首先对数据进行处理，去除存在缺失和偏离正常值严重的数据，之后使用基于遗传模拟退火算法的 FCM 聚类模型对目标时间段内的气象条件进行分类，由聚类后的指标推断出，当聚类数为 30 时，分类效果最好。因为本文认为大多数气候符合基本规律，所以本文选取数据量较多的前 4 类数据进行分类，使用有监督的支持向量机分类算法确定分割平面，通过交叉验证和网格搜索选择适当参数，最终模型在训练集上的准确率达到了 100%。之后用训练好的模型对提取出的四类数据进行分类，根据样本点到分割平面的距离可以知道该样本点所属类别。

针对问题三，要求建立一个同时适用于三个监测点的二次预报数学模型，考虑到对时间进行建模，因此本文对六种污染物建立循环神经网络的二次预测模型，首先去除存在信息缺失的数据以及异常数据，将一次模型的预测值进行归一化作为输入，将实际值与一次模型预测的值做差进行归一化后作为理想输出，输入到循环神经网络加线性层的结构进行预测。为了更好地增强模型的鲁棒性，提升首要污染物预测准确度，本文在一次预报的基础上增加了实测数据。同时，由于臭氧是一种二次污染物，本文在预测臭氧时候，考虑了其他污染物对其的影响。所搭建的模型能够有效拟合常规污染物的浓度变化。

针对问题四，要求建立区域协同预报模型来提升空气预报质量的准确度，因此本文基于有向环图传播网络建立模型。在有向环图 G=(V,E,X,H)中，本文认为每个监测站代表一个结点，该结点主要特征为当前六种污染物的浓度。由问题二可知，污染物浓度变化与气象条件有关，因此在问题四中对气象条件进行进一步处理，对于风速、温度、湿度和气压，本文认为这些条件能够作用于所有结点的浓度，因此污染物的传出与这些因素有关。但是对于污染物的传入，本文认为只有由风速导致的那部分才会传入另一个监测站。所以对于风向和风速，本文认为这两个条件会作用于区域间的浓度传播，即权值正比于风向与单位位置向量的点乘，反比于传播时间。权值越大浓度传播越多，反之则减少，同时由于点乘的存在，传播浓度与风向和位置向量的角度有关，权值可正向传播也可以反向传播。最后

通过非线性回归网络训练传出参数和传入参数，并与问题三中的模型进行对比可知，区域协同预报模型能够有效提示模型预报准确率。

最后本文对模型的优缺点进行了分析，并提出了可行的改进方案以供参考，综上所述，对 WRF-CMAQ 模型的预测结果进行二次建模能够有效提升模型性能。

关键词：WRF-CMAQ 模型　二次建模　遗传模拟退火算法　支持向量机　循环神经网络　有向环图传播网络

目录

1.问题重述

1.1 问题背景

随着人民生活水平的提高，人们越来越关注生活所需要的空气质量。大气污染是指人类活动或自然过程引起某些物质进入大气中，这些物质具有足够的浓度，达到了足够的时间，且因此危害了人体的舒适、健康和福利或危害了生态环境。研究实践中表明，建立良好的空气质量预报模型，能预测大气污染情况并做出相应的防护措施，可以在一定程度上降低大气污染对人类生活造成的危害。根据《环境空气质量标准》（GB3095-2012），用于衡量空气质量的常规大气污染物共有六种，分别为二氧化硫（SO_2）、二氧化氮（NO_2）、粒径小于 10μm 的颗粒物（PM_{10}）、粒径小于 2.5μm 的颗粒物（$PM_{2.5}$）、臭氧（O_3）、一氧化碳（CO）。其中臭氧预报难度较大。空气质量等级是以空气质量指数（Air Quality Index，AQI）为指标进行划分，当某日 AQI 超过 100 时，视当日天气为污染天气。

目前针对空气质量预测多用 WRF-CMAQ 模拟体系。但受模拟的气象场、排放清单的不确定性和对臭氧在内的污染物生成机理的缺少了解，WRF-CMAQ 模拟体系的结果并不理想。故本文采用了二次建模的方法，在该模拟体系一次预报建模的结果上，结合更多的数据源再次建模，以提高预报的准确性。由于实际气象条件对空气质量影响很大，且污染物浓度实测数据的变化情况对空气质量预报具有一定参考价值，故也会参考空气质量监测点获得的气象与污染物数据进行二次建模，相邻区域的污染物浓度往往具有一定的相关性，区域协同预报可能会提升空气质量预报的准确度。因此本文也利用监测点 A 及附近监测点数据建立协同预测模型一起预测污染物的浓度。

1.2 问题描述

根据目前的情况，基于已知的一次预报数据及实测数据进行空气质量预报二次数学建模，实现以下：

1) 根据已获得数据，计算监测点 A 从 2020 年 8 月 25 日到 8 月 28 日每天实测的 AQI 和首要污染物。
2) 若污染物排放情况不变，当某地气象条件有利于污染物扩散或沉降时，该地的 AQI 会下降，反之会上升。则需根据对污染物浓度的影响程度，对气象条件进行合理分类，并阐述各类气象条件的特征。
3) 利用数据建立可适用于 A、B、C 三个监测点的二次预报数学模型，预测三天六种常规污染物单日浓度值，使二次预报模型结果 AQI 预报值得最大相对误差尽量小，且首要污染物预测准确度尽量高。并利用新模型预测三点在 2021 年 7 月 13 日至 7 月 15 日六种污染物的单日浓度，计算相应的 AQI 和首要污染物。
4) 利用监测点 A 和附近三点的数据，建立协同预报模型。利用该模型预测四点在 2021 年 7 月 13 日至 7 月 15 日六种污染物单日浓度值并计算相应的 AQI 和首要污染物。并对比上一个模型，观察协同预报模型能否提升对监测点 A 的污染物浓度预报准确度。

2.基本假设和符号、监测点说明

2.1 基本假设

假设一：假设监测点 A、B、C 三点距离足够远，可以忽略它们之间的相互影响；

假设二：假设已获得数据中偏离正常值的数据属于无效数据，不参与建模不会影响模型的准确性；

假设三：假设相同气象条件下，对污染物浓度的影响程度相同，该地区的 AQI 的变化量相同；

假设四：假设类别中包含事件数较多的类是主题类，因为气候变换本身是符合规律的，类别中事件数较少的类可能是因为数据中存在误差或天气变化较为剧烈导致的；

假设五：假设问题四中浓度降低只可能与当前监测站气象条件有关，浓度增加只可能与其他观测站传入有关；

假设六：假设在预测样本污染物浓度时，污染物排放情况不变。

2.2 符号说明

符号	注释
$IAQI_P$	污染物 P 的空气质量分指数，结果进位取整数；
C_P	污染物 P 的质量浓度值；
BP_{Hi}	与 C_P 相近的污染物浓度限值的高位值；
BP_{Lo}	与 C_P 相近的污染物浓度限值的低位值；
$IAQI_{Hi}$	与 BP_{Hi} 对应的空气质量分指数；
$IAQI_{Lo}$	与 BP_{Lo} 对应的空气质量分指数；
q	比湿，空气中的水汽质量在混合空气中的质量占比
J_b	算法中的目标函数值
γ_i	几何间隔
γ	超平面关于所有样本点的几何间隔的最小值
ε_i	松弛变量
C	惩罚系数
Gamma	用于非线性支持向量机的超参数
δ_j^t	损失函数对网络权重的偏导数
MSE	均方损失函数
$G = (V,E,X,H)$	有向环图传播网络
$\omega_{in}(i,j)$	由监测站 i 到监测站 j 的传入系数
ω_{out}	监测站 i 的传出系数

2.3 监测点说明

监测点 A、B、C 三点两两之间直线距离较远，大于 100 千米。监测点 A1、A2、A3 是存在于监测点 A 临近区域的三点。具体相对位置如图 2-1 所示。

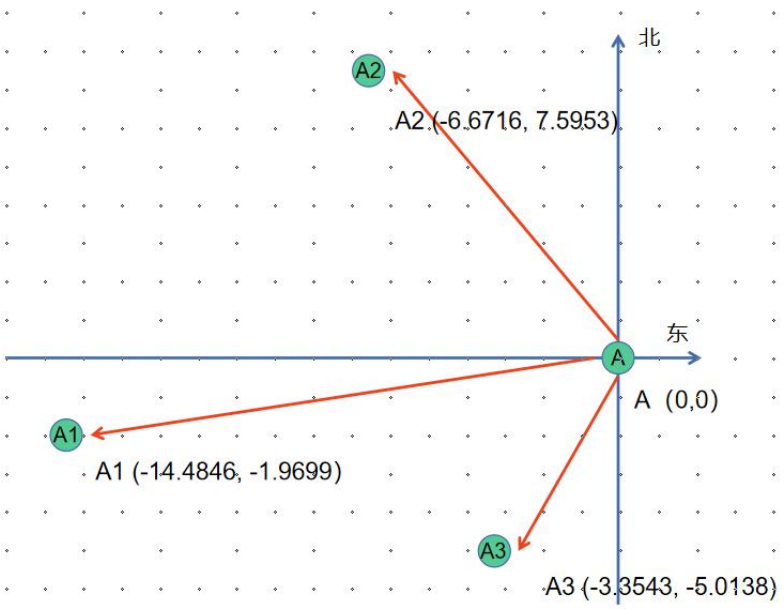

图 2-1 各监测点相对位置示意图

在图 2-1 中，正东方向为 x 轴，正北方向为 y 轴，单位为千米。A 点坐标为(0, 0)，A1 点坐标为(-14.4846，-1.9699)，A2 点坐标为(-6.6716, 7.5953) ，A3 点坐标为(-3.3543, -5.0138)。

3.问题一计算实测 AQI

根据《环境空气质量指数（AQI）技术规定（试行）》（HJ633-2012），空气质量指数（AQI）可用于判别空气质量等级[1]。首先需得到各项污染物的空气质量分指数（IAQI），其计算公式 3-1：

$$\mathrm{IAQI_P}=\frac{\mathrm{IAQI_{Hi}}-\mathrm{IAQI_{Lo}}}{\mathrm{BP_{Hi}}-\mathrm{BP_{Lo}}}\cdot(\mathrm{C_P}-\mathrm{BP_{Lo}})+\mathrm{IAQI_{Lo}} \tag{3-1}$$

其中 O_3 的浓度与其他污染物的浓度不同，需要做进一步处理，求得最大 8 小时滑动平均，这指一天中 8 时到 24 时之间所有 8 小时滑动平均浓度最大值，所谓 8 小时滑动平均值指连续 8 小时的平均浓度的算术平均值。其计算公式 3-2：

$$\mathrm{C_{O_3}}=\max_{t=8,9,\ldots,24}\{\frac{1}{8}\sum_{i=t-7}^{t}\mathrm{c}_t\} \tag{3-2}$$

其中 c_t 为臭氧在某日 t-1 时至 t 时的平均污染物浓度。

最后取各污染物的空气质分指数中的最大值作为空气质量指数（AQI），即

$$\mathrm{AQI}=\max\{\mathrm{IAQI}_1,\mathrm{IAQI}_2,\mathrm{IAQI}_3,\cdots,\mathrm{IAQI}_n\} \tag{3-3}$$

式 3-3 中，$\mathrm{IAQI}_1,\mathrm{IAQI}_2,\mathrm{IAQI}_3,\cdots,\mathrm{IAQI}_n$为各污染物项目的分指数。在本题中，对于 AQI 的计算仅涉及题目提供的六种污染物，因此计算公式 3-4：

$$\mathrm{AQI}=\max\{\mathrm{IAQI_{SO_2}},\mathrm{IAQI_{NO_2}},\mathrm{IAQI_{PM_{10}}},\mathrm{IAQI_{PM_{2.5}}},\mathrm{IAQI_{O_3}},\mathrm{IAQI_{CO}}\} \tag{3-4}$$

根据技术规定，当 AQI 小于或等于 50 时，这一天的空气质量可评为“优”，故一般会称这一天没有首要污染物；当 AQI 大于 50 时，IAQI 最大的污染物为首要污染物。

利用上述公式和理论，通过 MATLAB 编程（程序见附程序 1-2）计算出这四天每日每种污染物的 IAQI 的值，如图 3-1 所示。由图 3-1 所示，这四日 IAQI 最大的都是 O_3，其中 26 日时，O_3 的 IAQI 最大为 46，这一天空气质量评价为优，称当天无首要污染物。25、27、28 日最大 IAQI 均大于 50，且首要污染物均为当日 IAQI 最大的污染物 O_3。

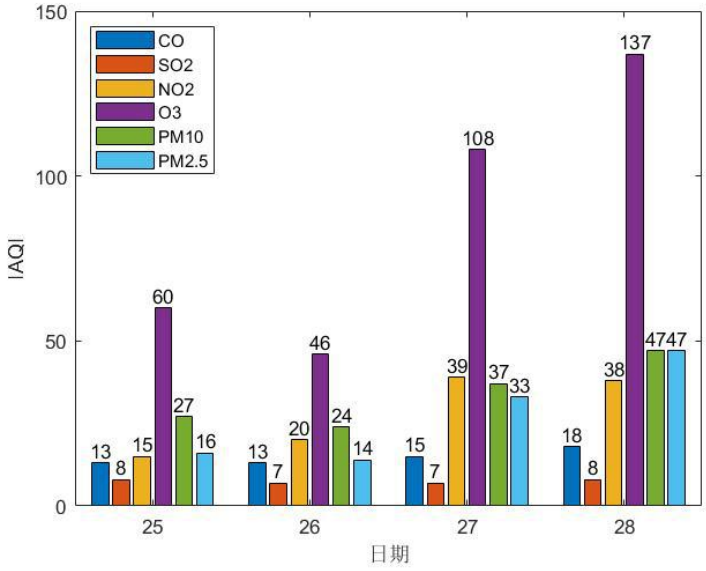

图 3-1 六种污染物每日 IAQI 值

把这些数据进一步整理得出问题一的结果如表 3-1：

表 3-1 AQI 计算结果表

监测日期	地点	AQI 计算	
		AQI	首要污染物
2020/8/25	监测点 A	60	O_3
2020/8/26	监测点 A	46	无首要污染物
2020/8/27	监测点 A	108	O_3
2020/8/28	监测点 A	137	O_3

4.问题二的模型建立与求解

4.1 模型的理论分析

4.1.1 基于遗传模拟退火算法的 FCM 聚类模型

随着研究领域的不断拓宽，无论科研还是实际生产中对聚类的结果从各个方面都提出了更高的要求。针对问题二提出的对气象条件进行合理的分离，本文考虑使用模糊 C-均值聚类（FCM，Fuzzy C-means），它在欧氏空间确定数据点的几何贴近度，将这些数分配到不同的聚类，最后确定各个聚类的几何距离。目标函数 J_b 公式如式 4-1：

$$J_{b(U,\ v)} = \sum_{k=1}^{n} \sum_{i=1}^{c} (\mu_{ik})^b (d_{ik})^2 \tag{4-1}$$

其中 n 是数据样本数，c 是相应的类别数，U 是其相似分类矩阵，v 是各类别聚类中心，μ_{ik} 是样本 x_i 对于类 A_k 的隶属度。FCM 聚类方法就是寻找一个最佳的分类，得到最小的目标函数值。这种算法重点在于聚类中心的初值，初始值选择不当很容易就收敛到局部极小点上，因此存在一定的局限性[2]。

而本问题中涉及到了 17044 个时间点的气象条件和空气质量指数，数据量极大，数据随机初始化复杂，为了避免 FCM 算法收敛到局部最优解，将其与遗传算法和模拟退火结合起来使用，各取其长，能够有效提升遗传算法的收敛速度。遗传算法是最近几年快速发展起来的，模仿生物界的自然遗传机制的全局随机搜索算法，可以解决复杂的寻找最优解问题。这样就对 FCM 算法的局限性做出了改进，得到能够高效收敛到全局最优解的混合算法。混合算法的过程如表 4-1 所示。

表 4-1 模型的具体流程

Algorithm FCM
Input: num % num 监测点 A 气候条件 M %M 是种群的个数 S % S 是内循环次数 P_c % env 是环境参数 P_m % P_m 是可能发生变异的比例 T_0 % T_0 是退火的初始温度 k %k 是温度衰减的系数 T_{end} % T_{end} 是退火的终止温度
Output: J_b % J_b 是目标函数值
1. **function** FCM (num, M, S, P_c, P_m, T_0, k, T_{end})
2. 随机初始化 C 个聚类中心
3. 生成初始种群 M
4. C,M → 每个聚类中心数据的隶属度 J_b 和每个体的适应值 f_i
5. **while(**loop < S**)**
6. 对群体 M 实施选择、交叉和变异
7. C,M → 每个聚类中心数据的隶属度 J_b 和每个体的适应值 f_i
8. **if(**f_i < 阈值**)**
9. 以旧个体替换新个体

```
10.      else
11.        以概率 P = exp（f－f/T）接收新个体，舍弃旧个体
12.      end if
13.      loop = loop+1
14.      if(Ti ＜ Tend)
15.        break
16.      else
17.        Ti+1 = kTi
18.      end if
19.    end while
20.    return Jb
21. end function
```

4.1.2 支持向量机分类模型

本问题中要求根据对污染物浓度的影响程度，对气象条件进行合理分类，这里本文采用支持向量机（SVM，Support Vector Machine）分类算法解答，采用 Python 进行编程（程序见附程序 4）。SVM 分类算法能够求解能够正确划分训练数据集的分离超平面，取几何间隔最大的超平面。对于给定一个特征空间上的训练数据集和超平面$\omega^T x + b = 0$,定义超平面关于样本点（x_i，y_i）的几何间隔为:

$$\gamma_i = y_i（\frac{\omega}{\|\omega\|} \times x_i + \frac{b}{\|\omega\|}）\qquad（4\text{-}2）$$

超平面关于所有样本点的几何间隔的最小值为:

$$\gamma = min_{i=1,2.....N}\gamma_i \qquad（4\text{-}3）$$

实际上这个距离就是支持向量到超平面的距离。根据以上定义，模型的求解最大分割超平面问题可以表示为有约束的优化问题。对于多分类问题可以采用一对多的 SVM 算法和一对一的 SVM 算法，从而将多分类问题转化成二分类问题，进而求解。本文采用一对多的 SVM 算法对 n 类数据集，构造 n 个两类子分类器，将每类数据与其它类数据分开。

SVM 算法的参数设置：

1) 对特征向量做归一化处理；
2) 选取核函数：采用在通常情况下性能较好的高斯径向基核函数（RBF）；
3) 参数设置：对于 RBF 核函数有两个重要的参数，一个是 gamma 参数默认值是 1/n，一个是惩罚参数 C 默认值为 1。但是采用默认值模型训练后准确度不高，通过交叉验证和网格搜索寻优获得最优的 C 和 gamma，且不会过拟合。这里最终选取 C=5 和 gamma=0.2；
4) 用取好的 C 和 gamma 训练模型；
5) 用测试样本检验模型的优劣；
6) 使用训练好的模型对气象条件分类[2]。

4.2 模型的建立

4.2.1 数据处理

根据附件给出的监测点 A 逐小时污染物浓度与气象实测数据，选取温度(℃)、湿度(%)、气压(MBar)、风速(m/s)、风向(°)五个指标来研究气象条件对污染物浓度的影响。将数据中某些数据缺失和偏离正常情况较严重（即使选用分类时这些组数据也属于少数类别，从而被排除掉）的数据组清洗掉，之后把剩余的这些指标做归一化处理，最后得到五个指标构成的 10744 组数据。

4.2.2 样本聚类

开始之前需要把所有样本按顺序编号，以此为事件序号。利用基于遗传模拟退火算法的 FCM 聚类算法，用 MATLAB 进行编程建模。对于聚类数，由于样本的类别数未知，所以尝试不同的类别数，计算目标函数值，从中取目标函数值最小的，说明个体适应度高，将之作为衡量模型效果的指标，最后选取效果最优的模型。

本次建模首先设置聚类数为 10，聚类效果如表 4-2：

表 4-2 10 类聚类部分结果表

类别	包含事件数	部分事件序号	
1	2045	2021/5/1 9:00	2021/5/1 11:00
2	2575	2019/4/24 10:00	2019/4/29 18:00
3	923	2019/9/19 17:00	2019/10/6 12:00
4	727	2019/11/12 6:00	2019/5/2 11:00
5	521	2019/10/25 11:00	2019/11/11 18:00
6	3501	2019/4/22 18:00	2020/8/28 14:00
7	3253	2019/9/19 14:00	2019/9/22 10:00
8	2621	2019/9/26 9:00	2019/9/28 6:00
9	174	2019/7/7 4:00	2019/8/9 1:00
10	704	2019/4/16 4:00	2019/4/18 12:00

由上述类别可以看出，类别相同时，时间较近，即季节相同时气候条件类似，因此从侧面印证了模型的准确性。但是 10 类时求出 J_b 值为 34.7474，这个值比较大，说明这样个体适应度不够高，这个聚类模型需要进一步改善。对于这样有 10744 个样本的模型，分为 10 类存在很多不合理的地方。所以本文接下来尝试了 20 和 30 类，再运行程序得到了各自的目标函数值进行比对如表 4-3。

表 4-3 不同聚类数下 J_b 值

聚类数	10	20	30
J_b	34.7474	8.6848	3.8588

表 4-3 表明，J_b 值与聚类数成反比，随着聚类数变大，J_b 值逐渐变小。当聚类数取 30 时，J_b 值为 3.8588，这个结果比较合理，因此本文采用 30 个聚类数进行分类，具体分类结果如表 4-4 所示。

表 4-4 30 类聚类部分结果表

类别	包含事件数	类别	包含事件数
1	3024	16	565
2	733	17	373
3	1008	18	214

4	1629	19	267
5	286	20	297
6	13	21	28
7	75	22	415
8	2479	23	93
9	0	24	376
10	38	25	257
11	2340	26	105
12	117	27	362
13	42	28	1434
14	346	29	2
15	125	30	1

由表 4-4 中观察聚类数为 30 时的聚类结果有的类别中已经出现没有样本的情况说明聚类数不能再变大了，且此时 J_b 值较小，故本文选择聚类数为 30 进行建模。由表 4-4，可以得出包含事件数较多的有类别 1、类别 4、类别 8、类别 11，由假设四得出这四个类别假设为主题类。总共有 17044 个事件，这四类包括了 9472 个事件。数据归一化之后 30 类别的五个数据量的聚类中心如表 4-5 所示。

表 4-5 聚类数为 30 的聚类中心

类别	温度	湿度	气压	风速	风向
1	0.71901	0.671083	0.354587	0.257561	0.677471
2	0.68634	0.681799	0.383463	0.229292	0.690778
3	0.633314	0.728222	0.42573	0.214339	0.249136
4	0.516445	0.594568	0.58797	0.231769	0.197458
5	0.52032	0.60159	0.58218	0.229922	0.202526
6	0.711437	0.675273	0.360279	0.249917	0.680618
7	0.635314	0.731119	0.422639	0.213503	0.248346
8	0.701557	0.681634	0.367414	0.239786	0.685937
9	0.443392	0.504275	0.667606	0.261515	0.13086
10	0.528655	0.610954	0.571114	0.227811	0.211322
11	0.52553	0.610492	0.574338	0.227727	0.20886
12	0.708079	0.675648	0.363644	0.247135	0.681534
13	0.711588	0.677421	0.358926	0.249303	0.68141
14	0.517521	0.599422	0.585508	0.230429	0.199822
15	0.699894	0.681408	0.369304	0.238658	0.686319
16	0.638662	0.735538	0.417608	0.212215	0.247196
17	0.464089	0.601356	0.606409	0.197112	0.866134
18	0.466972	0.603006	0.603977	0.195804	0.863194
19	0.523915	0.605022	0.577643	0.229139	0.206206
20	0.466356	0.602283	0.604571	0.196164	0.863875
21	0.640927	0.735493	0.415292	0.212564	0.247692
22	0.53292	0.616335	0.565149	0.226699	0.215542
23	0.633178	0.731583	0.424656	0.212992	0.247636

24	0.515972	0.595368	0.588177	0.231534	0.197393
25	0.522215	0.602115	0.580189	0.229851	0.204182
26	0.545968	0.63025	0.547523	0.224447	0.226796
27	0.694875	0.683748	0.373463	0.234012	0.689081
28	0.633263	0.723229	0.427602	0.21626	0.251337
29	0.683176	0.678405	0.388442	0.228947	0.68973
30	0.705856	0.677868	0.36481	0.244585	0.683034

4.2.3 各个类别的影响程度

表 4-6 30 类气象条件对污染物影响程度

类别	影响程度	类别	影响程度
1	-0.015873016	16	-0.100884956
2	-0.064120055	17	-0.042895442
3	-0.063492063	18	-0.303738318
4	-0.036218539	19	-0.209737828
5	-0.132867133	20	-0.212121212
6	-2.307692308	21	-0.714285714
7	-0.266666667	22	-0.098795181
8	0.003630496	23	-0.096774194
9	NaN	24	-0.002659574
10	0.052631579	25	-0.073929961
11	0.045299145	26	-0.219047619
12	-0.452991453	27	-0.055248619
13	-0.571428571	28	-0.037656904
14	-0.138728324	29	-9.5
15	-0.512	30	-14

根据题目要求，分别计算各类别对污染物的影响程度。由于本题属于熵减的情况，污染物浓度高时会自发地向浓度低变换，所以本文用样本中本小时和其后一个小时的 AQI 差值，当作气象条件对样本 AQI 的影响程度，再把每一类别所有样本影响程度的平均值作为这类气象条件对污染物的影响程度。具体每一类的影响程度如表 4-6，其中有一项为 NaN 是因为该类别中没有样本事件。由表 4-6 可以得出，主题类的影响程度分别为-0.015873016、-0.036218539、0.003630496、0.045299145。这四个影响程度有两正、两负，而且数量级类似，可以验证假设四合理。

4.2.4 新样本分类

利用 SVM 算法，把这四个主题类所包含的 9472 个样本，取其中 70%为训练集，剩余 30%为验证集，利用 Python 编程，得出训练集的预测准确度为 100%，验证集的预测准确度为 100%，2842 条验证数据都是正确的，证明分类准确度很高，模型训练效果很好。利

用训练好的模型对验证数据进行分类，输出每个事件归属于某一类别时到分割超平面的函数距离，距离越大，样本点归属于该类的可能性越大。

表 4-7 部分验证样本的函数距离表

监测时间	第 1 类	第 4 类	第 8 类	第 11 类	所属类别
2020/6/28 1:00:00'	0.815229	2.223347	3.277903	-0.28432	8
2021/2/17'	2.26701	-0.28637	0.758702	3.274787	11
2019/9/24 17:00:00'	2.077798	-0.29652	1.260613	3.268817	11
2020/8/11 22:00:00'	3.273093	0.746355	2.226906	-0.25897	1
2020/4/24 5:00:00'	2.277355	0.751092	-0.28489	3.26495	11
2021/5/23 4:00:00'	3.299759	0.715064	1.891848	-0.23971	1
2019/8/10 21:00:00'	1.172299	2.172591	3.285312	-0.29664	8
2019/8/28 7:00:00'	0.748131	3.296372	2.241139	-0.29431	4
2021/1/11 19:00:00'	1.785147	0.737313	-0.28785	3.307416	11
2019/7/5 21:00:00'	0.941227	1.784263	3.296246	-0.2853	8
2019/7/27 18:00:00'	3.264723	0.712364	2.290708	-0.27142	1

由模型可以得出各类气象条件的特征，如表 4-5，表 4-6 所示，比如当温度、湿度、气压、风速、风向归一化之后分别为 0.71901、0.671083、0.354587、0.257561、0.677471 时，AQI 平均变化可能为-0.015873016。

4.3 模型的灵敏度分析

在 SVM 中，训练过程中存在大量线性不可分的数据，对于这些数据引入松弛变量，对于部分不可分数据进行容忍，具体来说如式 4-4 所示：

$$y_i w^T x_i \geq 1-\varepsilon_i \tag{4-4}$$

式中 y 为类别，x 为数据点，ε_i 表示远离程度，引入松弛变量后 SVM 的优化目标可以扩展如式 4-5、4-6[5]。

$$\min_{w,\varepsilon} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\varepsilon_i \tag{4-5}$$

$$s.t. y_i w^T x_i \geq 1-\varepsilon_i, \forall i = 1,\cdots, N \quad \varepsilon_i \geq 0, \forall i = 1,\cdots,N \tag{4-6}$$

本文将原理群体的样本点成为离群点，每个离群点都通过表示远离程度，其中惩罚系数 C 是一个超参，其值越大表示越重视离群点，则不希望舍弃他们，反之则希望舍弃他们。gamma 是用于非线性 SVM 的超参数[6]，本文中使用高斯核函数 gamma 参数控制每个训练数据所产生的影响距离[7]，具体高斯函数如式 4-7：

$$K(x_1, x_2) = \exp(-\frac{\|x_1 - x_2\|^2}{2\sigma^2}) \tag{4-7}$$

为了验证惩罚系数 C 和 gamma 对于训练集准确率的影响，具体结果如表 4-8 所示：

表 4-8 不同参数组合条件下的准确率

C \ gamma	0.0001	0.001	0.01	0.2	1
0.0001	0.315	0.315	0.315	0.315	0.315
0.001	0.315	0.315	0.315	0.315	0.474
0.01	0.315	0.315	0.315	0.824	0.987
0.1	0.315	0.315	0.782	0.996	0.987
1	0.315	0.782	0.99	0.998	1
5	0.332	0.911	0.998	1	0.999

同时为了更形象地表示二者对于 SVM 分类的影响，本文绘制了二者对于测试集准确率影响的曲面图，如图 4-1 所示：

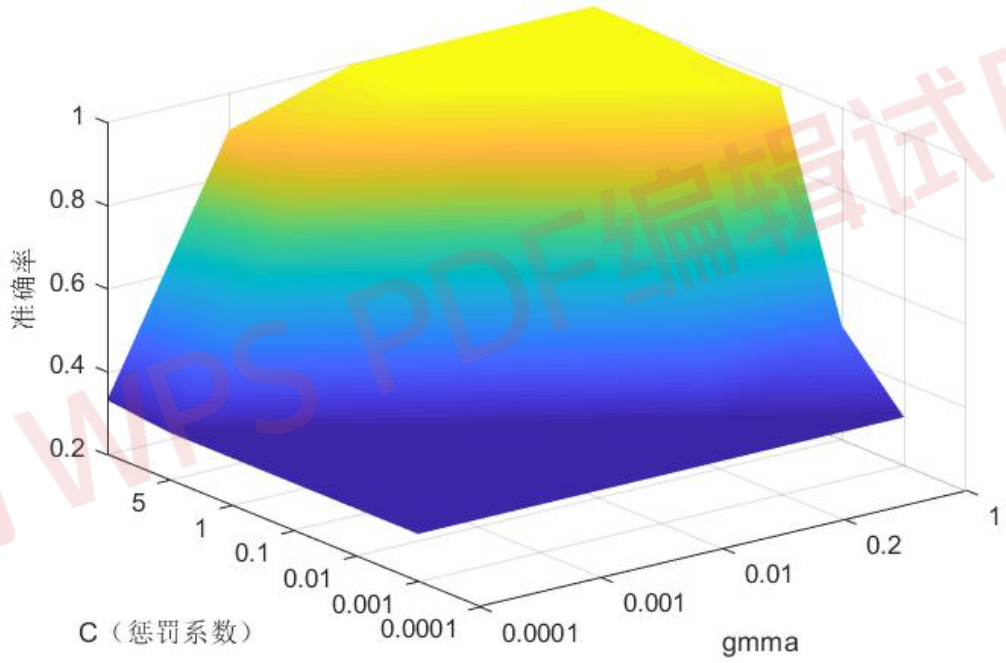

图 4-1 测试集准确率与参数关系

由图 4-1 和表 4-7 可知，随着惩罚系数 C 和 gamma 的增加，准确率不断上升，当惩罚系数 C 超过 0.1 且 gamma 超过 0.01 时，准确率上升较慢。通过灵敏度分析可知，超参数对于 SVM 的准确率影响较高，当 gamma 较高时惩罚系数 C 影响较小，当 gamma 较低时惩罚系数 C 影响较大，类似于对于线性 SVM 的影响。通过灵敏度分析发现，不同的参数取值，得到模型效果差距很大，因此参数寻优是非常有必要的。

5.问题三的模型建立与求解

5.1 模型的理论分析

目前常用 WRF-CMAQ 模拟体系对空气质量进行预报。其中 CMAQ 模型采用了“一个大气”的理念，使其能够全面地模拟出污染物排放进入大气的物理输送扩散以及化学反应过程，目前已广泛地应用于空气污染的研究以及业务预报工作[8]。WRF-CMAQ 模拟体系一种是根据动力-统计相结合的预报方法，采用多元线性统计模型，即通过逐步回归的方法，用前期的污染物观测资料和气象要素观测资料以及 CMAQ 模式产品，建立模式产品和多类预报因子相结合的日污染物浓度预报模型。模型如式 5-1：

$$\langle C_n \rangle = g + a[C_{n-1}] + b[CMAQ_n] + \tilde{c}\tilde{M} \quad (5\text{-}1)$$

设第 n 天为预报日，第 n－1 天为起报日，$\langle C_n \rangle$为预报量，表示预报日某种污染物的浓度；$[C_{n-1}]$代表起报日该种污染物的观测浓度；$[CMAQ_n]$代表预报日该种污染物的 CMAQ 模式预报产品；$\tilde{M} = \begin{pmatrix} M_1^{n-1} \\ M_2^{n-1} \\ \vdots \\ M_m^{n-1} \end{pmatrix}$代表每日 m 种气象要素观测值；a，b 为系数，$\tilde{c} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{pmatrix}$为系数矩阵，g 为常数[9]。但由于模拟的气象场以及排放清单不确定性地限制，以及对包括臭氧在内的污染物生成机理的不完全明晰，然而 WRF-CMAQ 预报模型只考虑了线性部分，未考虑非线性部分，所以模型预测的结果并不理想。在一次模型线性拟合的基础上，拟合其非线性部分，即实测值与一次预测模型的差值。本文二次模型采用 RNN 模型，输入为 15 种气象条件，输出为实测值与一次预测模型的差值。

循环神经网络（RNN，Recurrent Neural Network）是传统的前馈经网络（FNN，Feedforward Neural Network）的一个变种，二者的区别在于 FNN 的神经元仅仅通过层和层之间的连接来完成信息传递[10]，而 RNN 在网络中引入了环状结构，即建立了神经元到自身的连接。对于一个基础的 RNN 模型：输入层（含 I 个神经元），隐含层（含 H 个神经元）和输出层（含 K 个神经元），模型输入是长度为 T 的序列 X 。

RNN 的前向传播算法如式 5-2、5-3、5-4。

$$a_h^t = \sum_{i=1}^{I} \omega_{ih} x_i^t + \sum_{h'=1}^{H} \omega_{h'h} b_{h'}^{t-1} \quad (5\text{-}2)$$

$$b_h^t = \theta_h(a_h^t) \quad (5\text{-}3)$$

$$a_k^t = \sum_{h=1}^{H} \omega_{kh} b_h^t \quad (5\text{-}4)$$式

5-2、5-3、5-4 中：x_i^t是第 i 维输入在时刻 t 的值；ω_{ij}表示神经元 i 与 j 的连接权重；a_j^t和b_j^t分别表示神经元 j 在时刻 t 的输入值和激活值；θ_h表示神经元 h 的激活函数[11]。

RNN 的时间后向传播算法定义了损失函数对神经元 j 在时刻 t 输入值的偏导数，根据链式求导法则计算损失函数对网络权重的偏导数[12]，如式 5-5。

$$\delta_j^t = \frac{\partial L}{\partial a_j^t} \quad (5\text{-}5)$$

本文采用 Python 实现该模型，采用 Pytorch 架构实现（程序见附程序 4-8）。

5.2 数据处理

根据题目的要求，所建立的一个模型需要能够预测 A、B、C 三个监测站点的监测数据，同时要能够预测 2021 年 7 月 13 日-2021 年 7 月 15 日的数据，鉴于 2021 年 7 月 13-2021 年 7 月 15 日的数据的已知量仅有该时期一次模型预测的气象信息和污染物浓度，因此采用一次模型预测的数据进行对污染物浓度进行二次预测。数据处理主要包括以下四个方面：

1）数据清洗：由于工作环境和实际困难，监测站点的数据出现缺失现象，站点的数据质量无法保证，这将影响后续的分析工作，因此对数据进行清洗是及其与必要的，去除存在信息缺失的数据以及异常数据，得到 A 站点数据 8322 组，B 站点数据 8024 组，C 站点数据 7876 组。

2）污染物数据数值处理：本文采用深度学习网络构建二次预测模型，主要是将更多的影响因素纳入二次模型的考虑范围和拟合一次模型所忽视的实际模型的线性部分，因此模型的理想输出设定为实际模型和一次模型的差值，因此需要对污染物的浓度数据进行处理，具体而言，是将污染物的实测值和一次预测模型的预测值做差，得到差值数据作为模型的理想输出结果。

3）数据归一化：由于需要用同一的模型对 A，B，C 三个站点进行预测，不同站点之间，不同污染物和影响因素的数据差值很大，如图 5-1 是 2019 年 4 月 16 日 A 检测站点在不同时刻对 SO_2 和 NO_2 的实测浓度，可以看到由于空气中 SO_2 的浓度是大于 NO_2，在建立模型时对这样的数据难以纳入统一的标准，这对模型的建立、数据的预测是极为不利的，考虑到此类因素，对数据进行归一化处理如式 5-6。

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{5-6}$$

对 2019 年 4 月 16 日 A 检测站点在不同时刻对 SO_2 和 NO_2 的实测浓度如图 5-1 进行归一化后的相对浓度如图 5-2。在模型的训练中，数据归一化能够提升模型训练时的收敛速度和训练出来的模型性能。

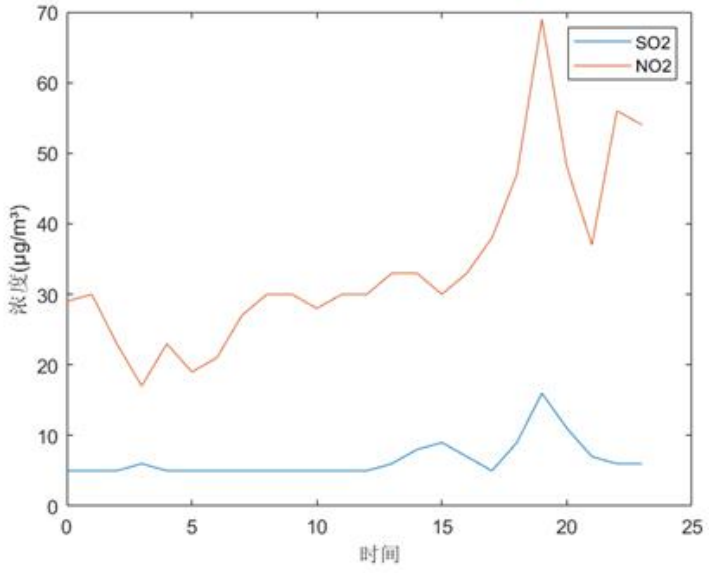

图 5-1 2019 年 4 月 16 日 A 站点 SO_2 和 NO_2 的实测浓度

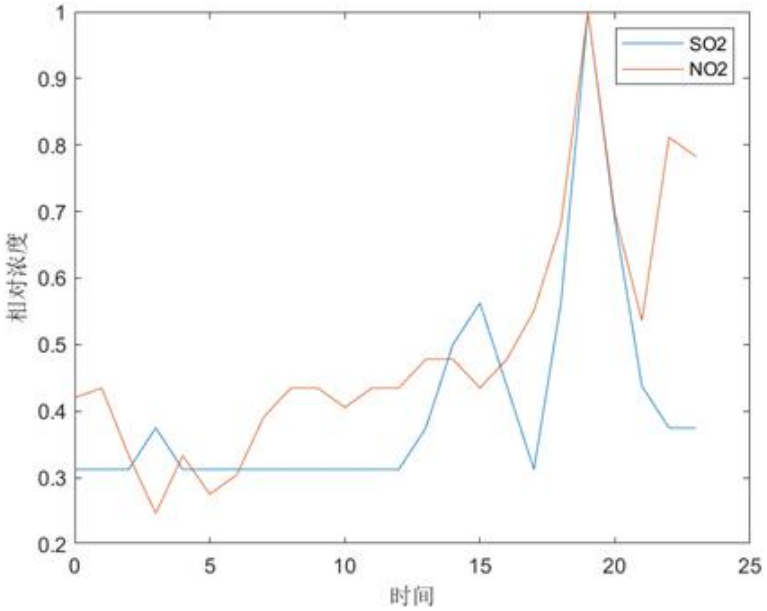

图 5-2 2019 年 4 月 16 日 A 站点归一化后的 SO_2 和 NO_2 的实测浓度

4）数据分组：本题数据的一大特点就是数据暗含时序信息，因为数据序列中的位置决定了数据时间先后顺序，相邻数据之间具有时间相关性。如果两个数据位置相邻，那么它们之间通常相似度很高，由此可见时序特征对于模型预测的准确率而言非常重要。因此，利用好时序信息对于精准预测具有重要作用，基于此，本研究将 A、B、C 三个站点数据进行分组，基本方法为将相邻的八条数据作为一组数据，这样再对模型进行训练，数据就可以对时序信息加以利用。

5.3 模型的求解

5.3.1RNN 预测模型

为了增强模型的鲁棒性能，本文将更多的影响因素纳入预测模型之中，不仅包括近地 2 米温度、地表温度，湿度，大气压，近地 10 米风速，比湿，近地 10 米风向等因素，还包括雨量，云量，边界层高度，感热通量，潜热通量，长波辐射， 短波辐射，地面太阳能辐射等对污染物浓度有影响的天气特征，本文建立了一个 15 维输入，6 维输出的二次预测模型，同时为了能够更好地利用数据中的时序特征预测污染物浓度，网络采用 RNN 加线性层的结构，具体结构如图 5-3：

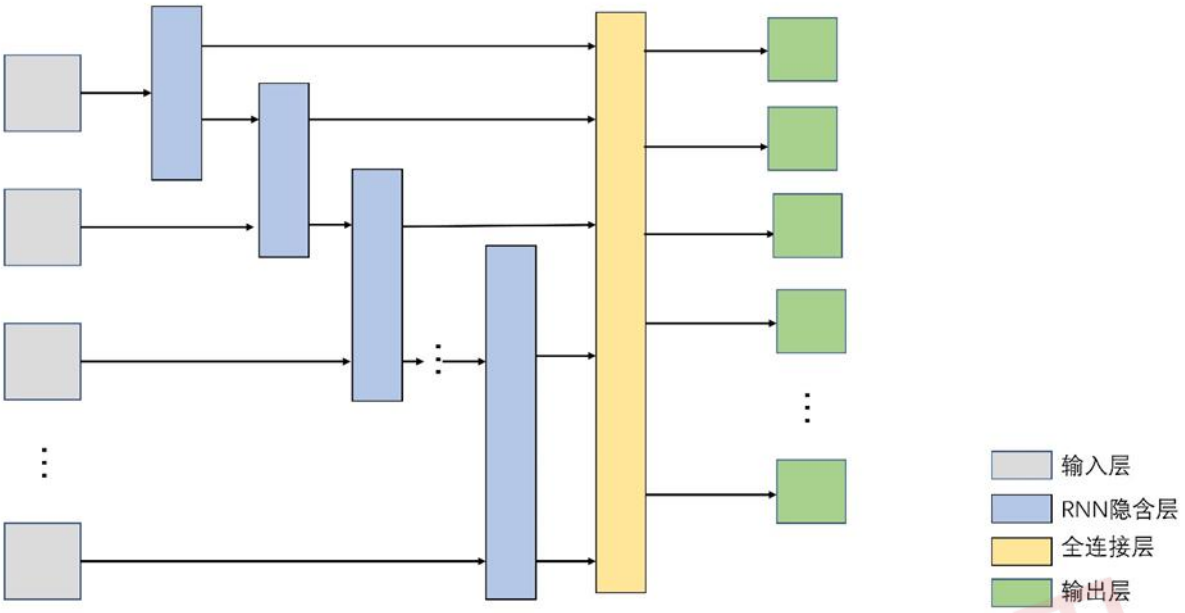

图 5-3 RNN 网络结构

对模型进行训练，训练过程如图 5-4，可以看到，模型在经过 50 次训练后，损失函数基本不变，说明模型已经具备一定的拟合能力，此时停止训练，保存模型作为预测模型。

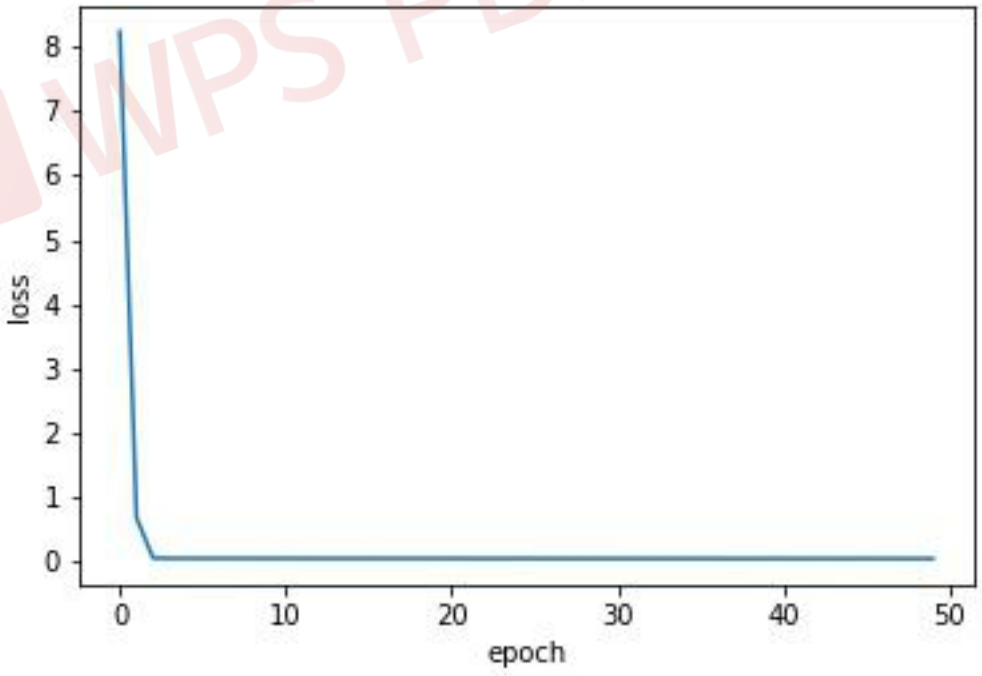

图 5-4 RNN 预测模型训练过程

对其进行性能进行测试，本文采用均方损失函数（MSE）[13]如式 5-7：

$$\mathrm{MSE} = \frac{1}{n}\sum i = 1^{m}\omega i(yi - \widehat{yi})^{2} \qquad (5\text{-}7)$$

其中，n 为样本的个数。

模型在测试集上的表现性能如表 5-1，这里的均方差损失是由归一化数据求得的。

表 5-1：RNN 模型在测试集上的性能

污染物	SO2	NO2	PM10	PM2.5	O3	NO	整体
均方差	0.019	0.014	0.0073	0.014	0.0079	0.014	0.013

如图 5-5 可以形象显示模型的预测值和理想值之间的差距。

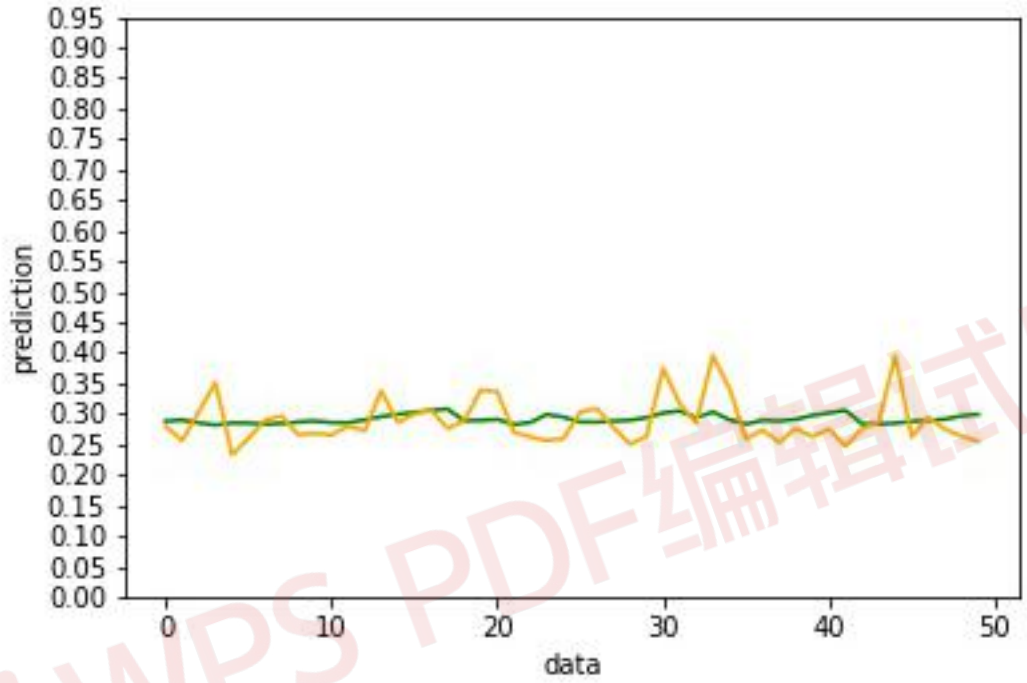

图 5-5 NO 实际差值(黄)和 RNN 模型的预测值(绿)

同时，由于本文构建模型输出为实测值与一次预测模型输出的归一化后的差值，因此模型的输出数据需要进行进一步处理如式 5-8

$$y = \text{predition} * (y_{max} - y_{min}) + y_{min} \tag{5-8}$$

其中，prediction 为模型的输出值，y_{min}为对原始数据进行归一化操作时使用的最小值，y_{max}为对原始数据进行归一化操作时使用的最大值，y 为对实测数据和一次模型预测数据的预测值，因此求解二次模型对污染物浓度的预测值的公式如式 5-9

$$y^{'} = y + y_{old} \tag{5-9}$$

其中$y^{'}$为二次预测模型对污染物浓度预测值，y_{old}为一次预测模型对污染物浓度预测值。

5.3.2 模型的改进

考虑到臭氧作为一种二次污染物，它的浓度不仅受到各种气象条件的影响，还受到了一次污染物的作用，如：NO 在一定条件下可以不断转化为 NO_2，NO_2 的光解使 O_3 逐渐积累，导致污染的产生。。因此为了更精准的预测臭氧的浓度，臭氧浓度预测模型还应该考虑其他污染物，因此将，上述的预测模型预测出来的 SO_2，….等一次污染物数据和气象条件一起，用于预测臭氧的浓度。模型输入为 20 个影响因素，输出为臭氧的实测值和一次预测的差值，模型采用 RNN 模型，其训练过程如图 5-6。

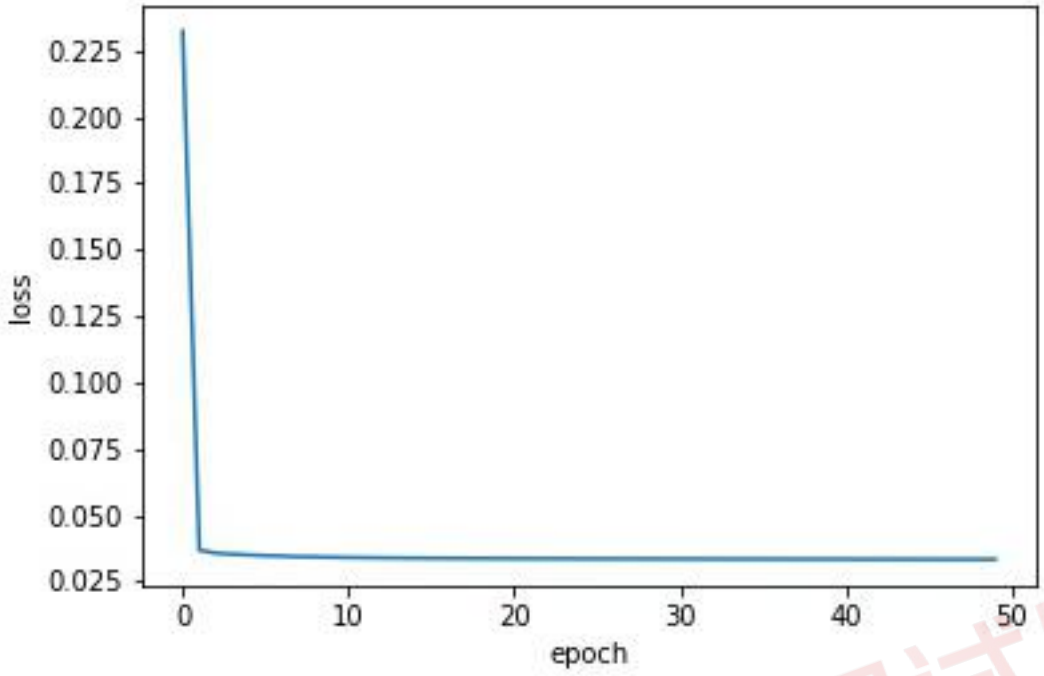

图 5-6 改进后 RNN 预测模型训练过程

训练后，模型具有预测实测值和一次预测模型差值的能力，如图 5-7 可以形象显示模型的预测值和理想值之间的差距。

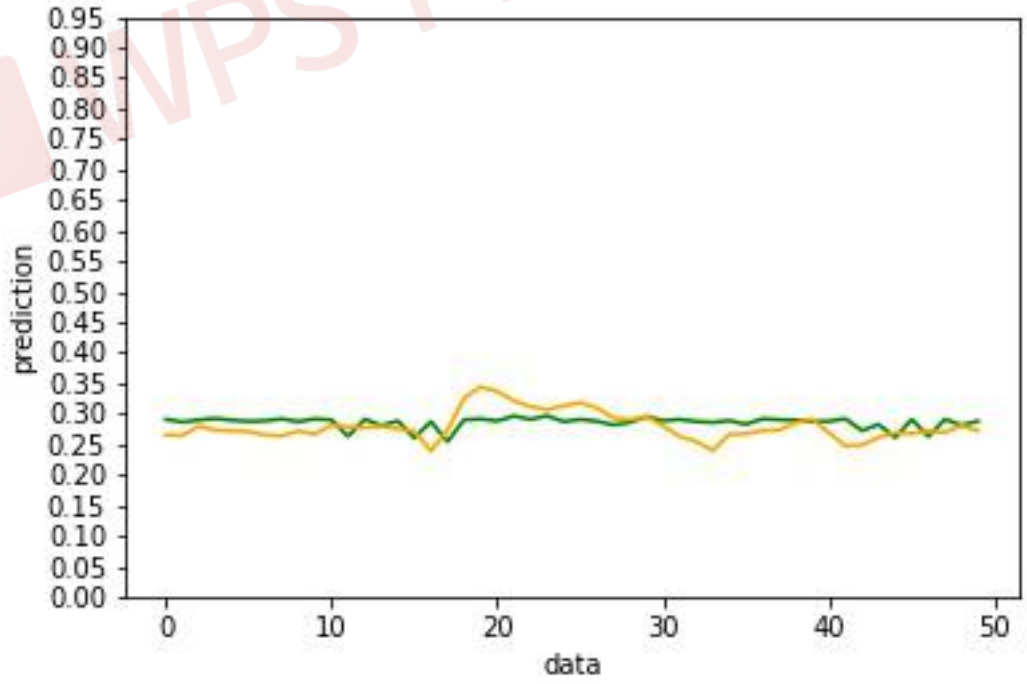

图 5-7 O3 实际差值(黄)和改进后 RNN 模型的预测值(绿)

5.3.3 模型改进前后对比分析

根据题目要求，二次预报模型预测结果中 AQI 预报值的最大相对误差应尽量小，因此选用了均方损失函数，以使模型在训练过程中与理想值尽可能一致。对于成因最为复杂的臭氧浓度的预测，采用了更多的影响因素，以保证其预测的准确性。同时根据题目要求，二次预报数学模型，首要污染物预测准确度尽量高。而在题目一中，首要污染物均为臭氧，因此对臭氧浓度的拟合更要考虑更多因素，提高其预测的精度，为说明改进后模型的性能。下面对改进模型在测试集上性能做简要分析，这里的均方差损失是由归一化数据求得的。

表 5-2 改进前后 RNN 模型对 O_3 的预测性能

模型	改进前	改进后
均方差	0.007915770	0.006212349

由表 5-2 可见，模型在验证集上提升了约 21.5%的性能，这说明，对于 O_3 这类的二次污染物的预测不仅要考虑气象条件的影响，还要考虑其具体的形成过程，考虑二次污染物和一次污染物的浓度关系。由表 5-2 可知，考虑一次污染物对 O_3 的影响的模型对 O_3 的预测更为精准。

5.4 模型的输出结果与分析

综上，本文得到了同时适用于 A、B、C 三点的二次预报模型，且经过验证，在数据归一化的情况下，模型在测试集上的均方差可以达到 0.02 以下，可以完成预测任务，同时，根据 O_3 的成因对模型进行改进。

根据题目要求，下面对模型预测结果进行处理，得到监测点 A、B、C 的 7-13 日至 7-15 日的污染物预测结果分别如表 5-3、表 5-4、表 5-5。其中，一次污染物 SO_2、NO_2、$PM_{2.5}$、PM_{10}、CO 为原始二次模型的输出数据经过处理得到的预测结果，二次污染物 O_3 为改进后的二次预测模型输出的数据经过处理得到的结果。

表 5-3 二次预测模型对 A 站点污染物浓度预测值

预报日期	地点	二次模型日值预测							
		SO_2 ($\mu g/m^3$)	NO_2 ($\mu g/m^3$)	PM_{10} ($\mu g/m^3$)	$PM_{2.5}$ ($\mu g/m^3$)	O_3 最大八小时滑动平均 ($\mu g/m^3$)	CO (mg/m^3)	AQI	首要污染物
2021/7/13	监测点 A	4.85289	10.3497	23.0621	5.34746	88.569	0.64774	44	无首要污染物
2021/7/14	监测点 A	4.94564	17.659	21.7228	5.12516	106.852	0.64605	56	O3
2021/7/15	监测点 A	5.01268	20.6473	22.6597	6.31635	96.847	0.69908	48	无首要污染物

表 5-4 二次预测模型对 B 站点污染物浓度预测值

预报日期	地点	二次模型日值预测							
		SO_2 ($\mu g/m^3$)	NO_2 ($\mu g/m^3$)	PM_{10} ($\mu g/m^3$)	$PM_{2.5}$ ($\mu g/m^3$)	O_3 最大八小时滑动平均 ($\mu g/m^3$)	CO (mg/m^3)	AQI	首要污染物
2021/7/13	监测点 B	5.018184	12.4459	22.90046	2.591779	59.726	0.509146	30	无首要污染物
2021/7/14	监测点 B	1.54925	12.723	19.67035	1.829698	53.9722	0.50867	27	无首要污染物
2021/7/15	监测点 B	4.79759	13.964	19.10251	1.146978	64.8849	0.500266	32	无首要污染物

表 5-5 二次预测模型对 C 站点污染物浓度预测值

预报日期	地点	二次模型日值预测							
		SO_2 ($\mu g/m^3$)	NO_2 ($\mu g/m^3$)	PM_{10} ($\mu g/m^3$)	$PM_{2.5}$ ($\mu g/m^3$)	O_3 最大八小时滑动平均 ($\mu g/m^3$)	CO (mg/m^3)	AQI	首要污染物
2021/7/13	监测点 C	4.761324	0.62876	29.82069	10.10463	59.7598	0.617835	30	无首要污染物
2021/7/14	监测点 C	4.814836	0.0346	30.94028	11.82209	135.9279	0.61493	80	无首要污染物
2021/7/15	监测点 C	46.5192	17.67209	38.59383	17.59553	137.8573	0.663204	82	无首要污染物

6.问题四的模型与求解

6.1 问题分析

鉴于相邻区域的污染物浓度往往具有一定的相关性，空气质量预报往往由多个监测站共同完成，区域协同预报能够有效提升预测的准确性。因此本文考虑多个监测站协同工作，重点考虑问题二中的分类条件，来预测监测点 A 和临近区域内存在的监测点 A1、A2、A3 共四个站点未来三天的 6 种常规污染物的浓度，并计算其相应 AQI，得出首要污染物。同时，本文考虑第三题中提出的方法，利用二次预测模型来预测实测数据，在有效改良问题三的方法的同时又能做到协同预报，由结果可知，协同预报能够有效提升模型预测的准确率。

6.2 模型的建立

6.2.1 有向环图传播网络

由于是临近区域协同工作，而且数据量较少，本文希望先验地建立模型来加快拟合。本文考虑使用简化的图神经网络进行预测，即向图神经网络提供已有的改进信息来加快拟合[14]。因此本文将每个观测站视为一个结点，针对这个结点，每个观测站包含四个参数如式 6-1，即用三元组表示该有向环图。

$$G = (V,E,X,H) \tag{6-1}$$

其中 V 表示当前每个监测站结点，E 为权重集表示每条边的权重，X 为特征集表示每个观测站结点所在区域的 6 种常规污染物的浓度，H 为环境集表示当前所处的环境特征。具体来说该有向图网络如图 6-1 所示：

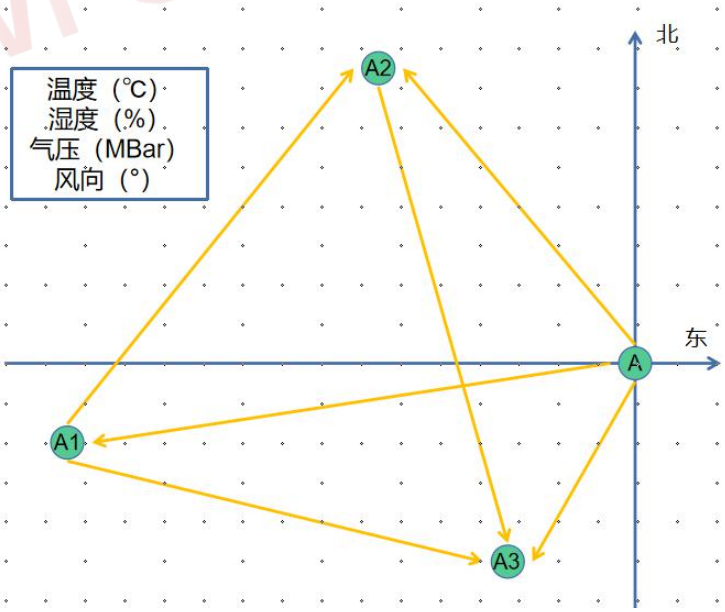

图 6-1 有向图网络

6.2.2 监测站结点 V

对于观测站结点 V，本文主要考虑其位置因素，在结点 V 中存储其位置信息后，进而

得到一个邻接表来表示其向量信息如式 6-2、6-3。

$$V=\begin{bmatrix}0 & 0\\ -14.4846 & -1.9699\\ -6.6716 & 7.5953\\ -3.3543 & -5.0138\end{bmatrix} \tag{6-2}$$

$$V_n=\begin{bmatrix}-14.4846 & -1.9699\\ -6.6716 & 7.5953\\ -3.3543 & -5.0138\\ 7.813 & 9.5652\\ 11.1303 & -3.0439\\ 3.3173 & -12.6091\end{bmatrix} \tag{6-3}$$

由于该图为有向图，因此 V_n 分别表示由 A 至 A1、A 至 A2、A 至 A3、A1 至 A2、A1 至 A3、A2 至 A3 的向量。

6.2.3 特征集 X

本题需要计算出预报四个监测 1 点 2021 年 7 月 13 日至 7 月 15 日 6 种常规污染物的单日浓度值，因此特征集 X 中分别存储每个监测站点所在地区的 6 种常规污染物的单日浓度值。由于该值是个变化量，因此仿照问题二考虑每天六种常规污染物的单日浓度值变化量，通过前三日的变化量预测接下来该日的变化量。

6.2.4 环境集 H

由问题二知，在污染物排放情况不变的条件下，某一地区的气象条件有利于污染物扩散或沉降时，该地区的 AQI 会下降，反之会上升。因此本文考虑当前环境对于常规污染物浓度变化的影响。相对于问题二的分类，由于本问加入了位置信息，因此对于环境因素应当分别考虑，具体作用如表 6-1 所示：

表 6-1 气象条件的具体作用

气象条件	作用
温度（℃）	影响当地常规污染物浓度
湿度(%)	影响当地常规污染物浓度
气压(MBar)	影响当地常规污染物浓度
风速(m/s)	加快污染物浓度的传播速度
风向(°)	影响污染物浓度的传播方向

对于温度(℃)、湿度(%)和气压(MBar)，本文分别做归一化处理。对于风速由于距离的单位为 km，因此本文将其转化为 km/h。对于风向，由题意可知，定义自正北方向至监测点的风向为 0°风向，以顺时针旋转角（单位：°）为正值记录风向，为了与坐标系匹配，定义自正西方向至监测点的风向为 0°风向，具体处理如式 6-4、6-5、6-6、6-7、6-8。

$$H_{\text{temperature}}=\frac{H_{\text{temperature}}-\min(H_{\text{temperature}})}{\max(H_{\text{temperature}})-\min(H_{\text{temperature}})} \tag{6-4}$$

$$H_{\text{humidity}}=\frac{H_{humidity}-\min(H_{humidity})}{\max(H_{humidity})-\min(H_{humidity})} \tag{6-5}$$

$$H_{\text{pressure}}=\frac{H_{\text{pressure}}-\min(H_{pressure})}{\max(H_{\text{pressure}})-\min(H_{pressure})} \tag{6-6}$$

$$H_{speed} = H_{speed} * 3.6 \quad (6\text{-}7)$$

$$H_{direction} = \frac{|H_{direction} - 270^{\circ}| * \pi}{180^{\circ}} \quad (6\text{-}8)$$

6.2.5 权重集 E

为了能够表示传播模型，其具体示意图如图 6-2。

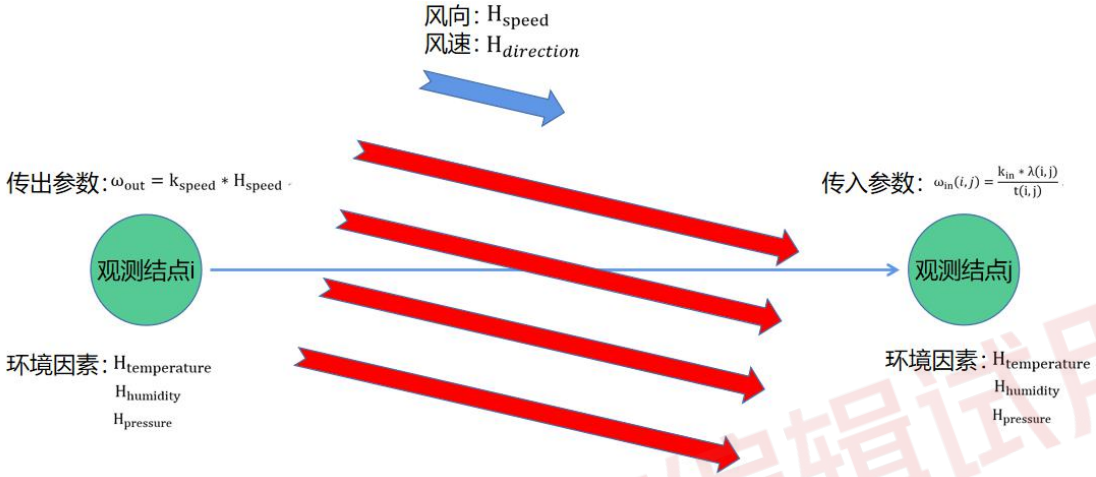

图 6-2 传播模型示意图

本文定义传出参数ω_{out}和传入参数$\omega_{in}(i,j)$。由问题二可知，常规污染物从某地区传出主要和风速、温度、湿度和压强有关，则有了式 6-9。

$$\omega_{out} = k_{speed} * H_{speed} + k_{temperature} * H_{temperature} + k_{humidity} * H_{humidity} + k_{pressure} * H_{pressure} \quad (6\text{-}9)$$

其中H_{speed}为当前风速，k_{speed}为风速的传播系数；$H_{temperature}$为当前温度，$k_{temperature}$为温度的传播系数；$H_{humidity}$为当前湿度，$k_{humidity}$为湿度的传播系数；$H_{pressure}$为当前压强，$k_{pressure}$为压强的传播系数。对于传入参数$\omega_{in}(i,j)$，只有由风传播的部分才会传导到目标地区，故本文认为风速越大会加快污染物传播，距离越长会使得传播的浓度衰减。因此定义时间参数 t(i,j)如式 6-10。

$$t(i,j) = \frac{\sqrt{V_n(i,j)(1)^2 - V_n(i,j)(2)^2}}{\omega_{out}} \quad (6\text{-}10)$$

其中分子为观测站 i 到 j 的距离，分母为传出参数，通过时间参数 t 能够将传出参数与传入参数相结合。

同时仿照物理学中功的定义，本文认为风向与观测站 i 到 j 的向量的点乘代表ω_{out}中被成功传播到观测站 j 的常规污染物浓度如式 6-11。

$$norm(V_n) = \begin{bmatrix} -0.990878393538093 & -0.893349194928462 \\ -0.659943933405330 & 0.996246437721564 \\ -0.556049871946452 & -0.993906320161714 \\ 0.330934460132763 & 1.88959563265003 \\ 0.434828521591641 & -0.100557125233252 \\ 0.103894061458878 & -1.99015275788328 \end{bmatrix} \quad (6\text{-}11)$$

$$\lambda(i,j) = \mathrm{norm}(V_n)(i,j) \cdot H_{direction} \quad (6\text{-}12)$$

$\mathrm{norm}(V_n)$表示对于V_n的归一化，将其与风向进行点成代表成功由观测站 i 传播到观测站 j 的常规污染物浓度。如果$\lambda(i,j)$为正数，代表传播方向和给定方向一致，即 i 传播向 j；如果$\lambda(i,j)$为负数，代表传播方向和给定方向不一致，即 j 传播向 i。

最后传入参数$\omega_{in}(i,j)$与$\lambda(i,j)$成正比，和 $t(i,j)$成反比，并乘以系数k_{in}，具体公式如式 6-13。

$$\omega_{in}(i,j) = \frac{k_{in} * \lambda(i,j)}{t(i,j)} \quad (6\text{-}13)$$

6.3 模型的求解

6.3.1 模型的搭建

为了保证数据能够正确回传，对于如果数据行中存在确实的数据，本文选择放弃这一数据行，对于常规污染物浓度，本文仿照问题一的做法将其归一化，之后搭建非线性回归网络对参数k_{in}和k_{speed}进行学习。本文采用 MATLAB 编程（程序见附程序 9），具体实流程如表 6-2。

表 6-2 模型的具体流程

Algorithm Graph
Input:num　% num 是四个观测站的六个常规污染物浓度变化 location　% location 是四个监测点的位置参数 env　% env 是环境参数 Output:w　% w 是传入参数和传出参数矩阵 22. **function** Graph (num) 23. location → 位置向量 loc → 位置单位向量 loci 24. env→风向和风速矩阵 wind 25. 初始化损失 loss 26. 初始化系数 w 27. **for** time = 1:size(num) 28. wind, location,loci → 传入参数初始值 w_in 29. env → 传出参数初始值 w_out 30. w_in,w_out,w 对应系数相乘 → 传入参数 w_in 和传出参数 w_out 31. w_in,w_out, num → 预测下一天浓度 predict 32. w_in = **tanh**(w_in) 33. w_out = **tanh**(w_out) 34. loss = predict-num 35. loss 回传修改系数 w 36. **end for** 37. **return** w 38. **end function**

6.3.2 结果分析

最终输出结果如图 6-3 所示，这里展示六个常规污染变量在监测点 A 和 A1 的预测值和真实值。

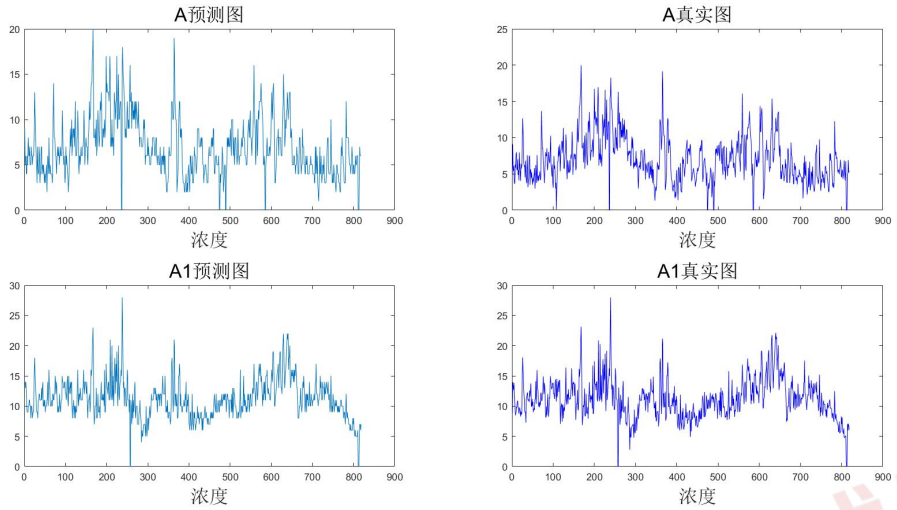

图 6-3（a）SO_2 在监测点 A 和 A1 的预测图和真实图对比

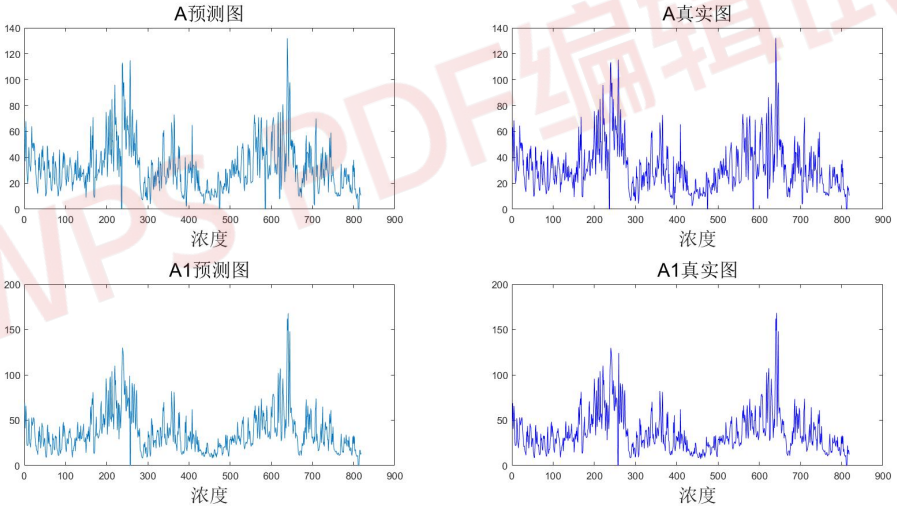

图 6-3（b）NO_2 在监测点 A 和 A1 的预测图和真实图对比

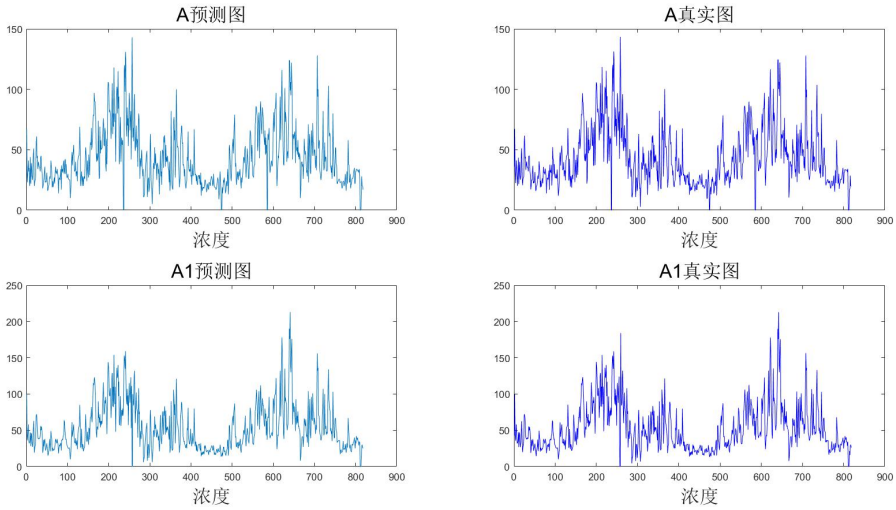

图 6-3（c）PM_{10} 在监测点 A 和 A1 的预测图和真实图对比

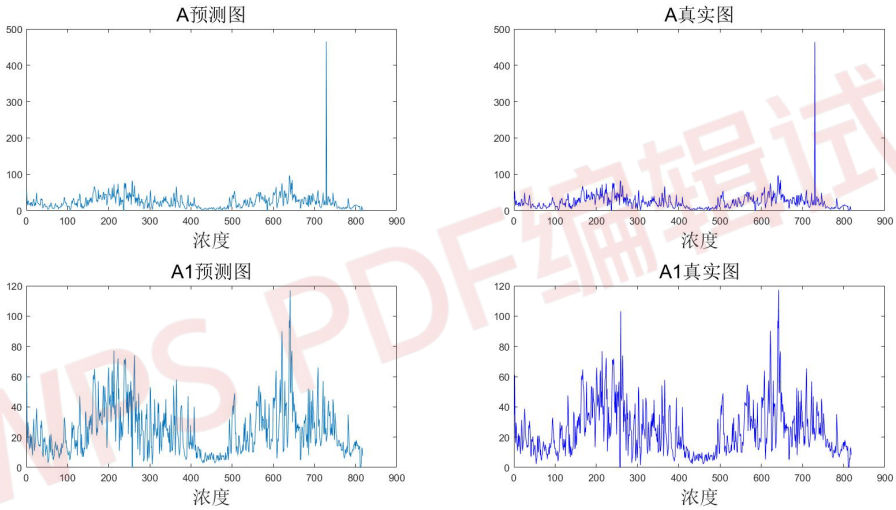

图 6-3（d）$PM_{2.5}$ 在监测点 A 和 A1 的预测图和真实图对比

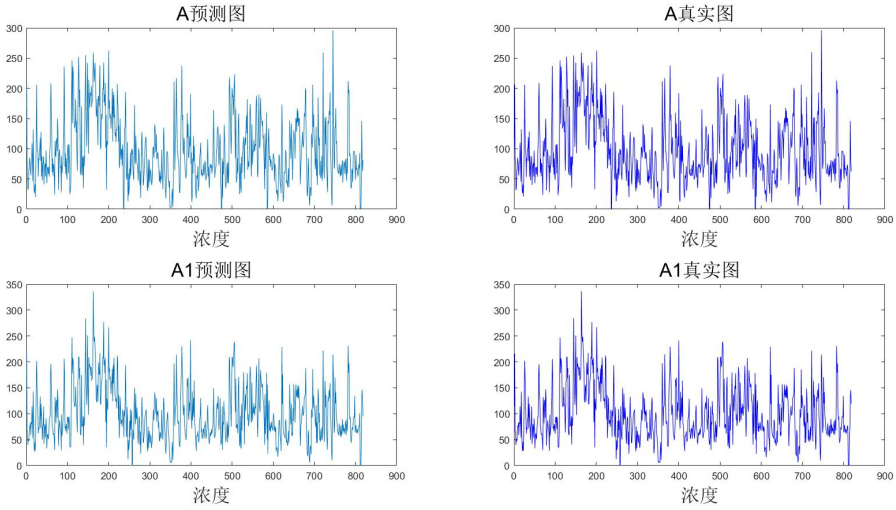

图 6-3（e）O_3 在监测点 A 和 A1 的预测图和真实图对比

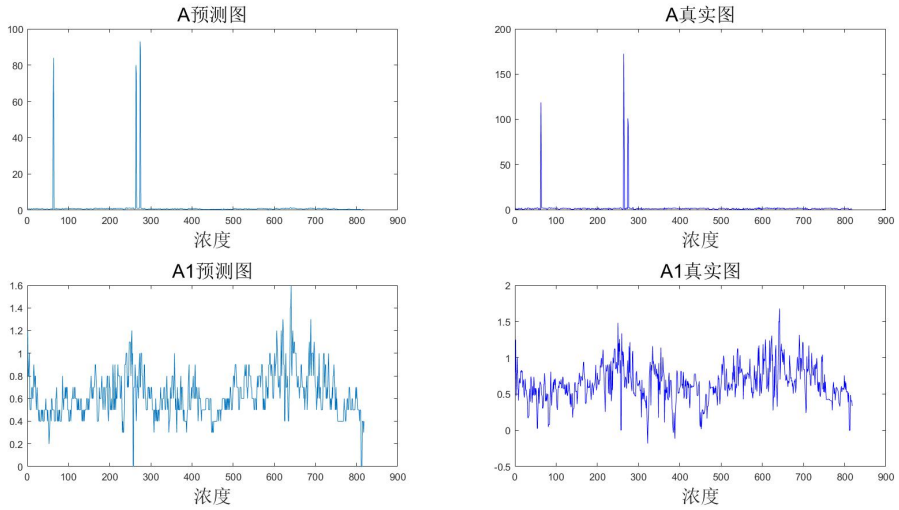

图 6-3（f）CO 在监测点 A 和 A1 的预测图和真实图对比

按照要求，预测监测点 A、A1、A2、A3 在 2021 年 7 月 13 日至 7 月 15 日 6 种常规污染物的单日浓度值和相应的 AQI 和首要污染物如表 6-3。

表 6-3 四个监测点的二次模型日值预测

预报日期	地点	二次模型日值预测								
		SO_2 ($\mu g/m^3$)	NO_2 ($\mu g/m^3$)	PM_{10} ($\mu g/m^3$)	$PM_{2.5}$ ($\mu g/m^3$)	O_3 最大八小时滑动平均 ($\mu g/m^3$)		CO (mg/m^3)	AQI	首要污染物
2021/7/13	监测点 A	5.99599	11.00146	16.99829	4.99386	81.00217	0.39576	5.99599	41	无首要污染物
2021/7/14	监测点 A	6.46518	10.84079	17.25516	5.76825	80.67493	0.90126	6.46518	40	无首要污染物
2021/7/15	监测点 A	6.83851	10.71962	17.48662	6.41006	80.37668	1.30988	6.83851	40	无首要污染物
2021/7/13	监测点 A1	6.99847	13.99871	24.99929	7.99857	94.99885	0.40000	6.99847	47	无首要污染物
2021/7/14	监测点 A1	7.19042	14.15757	25.09330	8.19017	95.20687	0.39983	7.19042	48	无首要污染物
2021/7/15	监测点 A1	7.34701	14.28949	25.17508	8.35438	95.40787	0.39975	7.34701	48	无首要污染物
2021/7/13	监测点 A2	5.00065	18.00332	25.00331	7.00413	88.99688	0.49886	5.00065	44	无首要污染物
2021/7/14	监测点 A2	4.94980	17.53038	24.53437	6.43688	89.40698	0.67310	4.94980	45	无首要污染物
2021/7/15	监测点 A2	4.92801	17.10956	24.12027	5.94270	89.75427	0.83337	4.92801	45	无首要污染物
2021/7/13	监测点 A2	4.00490	8.99651	13.99911	6.00344	86.00210	0.30537	4.00490	43	无首要污染物
2021/7/14	监测点 A2	3.39460	9.47127	14.11716	5.60470	85.71122	0.37419	3.39460	43	无首要

										污染物
2021/7/15	监测点 A2	2.88647	9.88133	14.21803	5.29287	85.46119	0.94299	2.88647	43	无首要污染物

6.4 本模型与问题三模型对比

由题意可知，题目要求回答与问题三的模型相比，协同预报模型能否提升针对监测点 A 的污染物浓度预报准确度，因此本文分别使用问题四中的模型和问题三中的模型对监测点 A 的部分数据进行预测，它们与真实值的结果对比图 6-4。

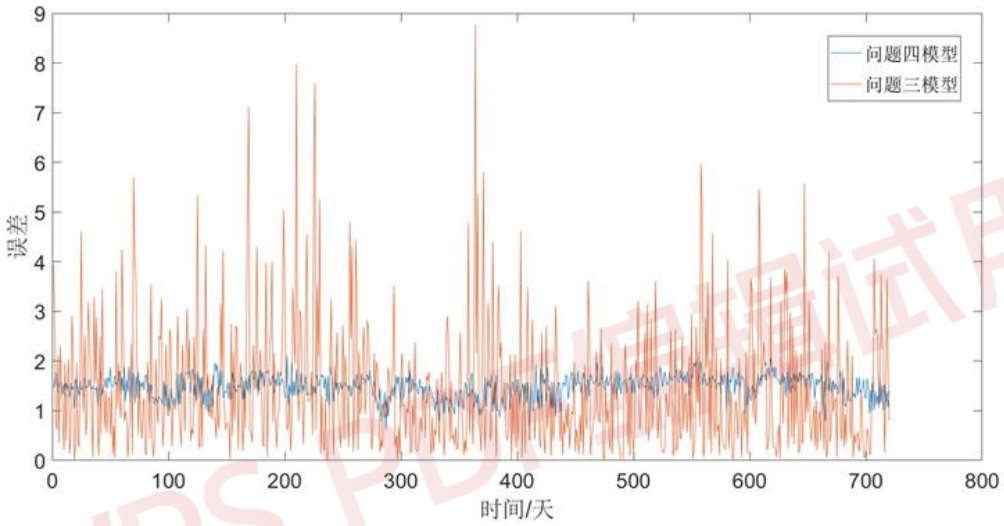

图 6-4 监测点 A 的 O_3 在两个模型下的误差绝对值对比

由图 6-4 可以看出，问题三模型同时采用区域协同预测模型的预测结果较为准确。本文认为对于空气质量预报这个问题：

1) 首先，由于空气存在流动性，相邻区域的污染物浓度往往具有一定的相关性，如果只考虑本地区的气象条件所带来的空气质量变化，不能足够准确地预测目标污染物浓度值。
2) 其次，由于数据记录存在误差，因此多个区域协同预报能够有效解决这一问题，不同的地区的数据也能够弥补数据的缺失。
3) 最后，不同地区的气象条件不同，单个模型只能解决本地的气象条件，无法考虑到其他地区气象条件所带来的影响与不利因素。

综上所述，与问题三的模型相比，协同预报模型能够有效提升模型的预测质量。

7.模型总结与评价

7.1 模型总结

由于人类活动或自然过程引起某些物质进入大气中，呈现足够的浓度，达到了足够的时间，就会危害了人体的舒适、健康和福利或危害了生态环境，因此建立空气质量预报模型，提前获知可能发生的大气污染过程并采取相应控制措施是十分重要的。针对目前预报空气质量的 WRF-CMAQ 模拟体系的不足，进行了二次数学建模。首先利用已获得的数据计算出了 2020 年 8 月 25 日到 8 月 28 日每天实测的 AQI 和首要污染物。由于气象条件对污染物的扩散或沉降有一定的影响，本文采用基于遗传模拟退火算法的 FCM 聚类模型对气象条件进行分类来研究对污染物的影响，经过多次计算聚类数为 30 时，模型效果最好。然后选取数据量较多的前 4 类数据进行分类，使用有监督的 SVM 分类算法进行分离，通过交叉验证和网格搜索选择适当参数，最终模型在训练集上的准确率达到了 100%。为了建立同时适用于 A、B、C 三个监测点的预报模型，将一次模型的预测值进行归一化作为输入，将实际值与一次模型预测的值做差作为输出，输入到 RNN 加线性层的结构进行预测未来时间的 AQI 值等。之后利用监测点 A 和临近区域内的监测点 A1、A2、A3 建立基于有向环图网络的协同预报模型，分别考虑浓度的传出和传入的影响因素，通过非线性回归网络进行回归，并与问题三中的模型进行对比可知，区域协同预报模型能够有效提示模型预报准确率。

7.2 模型评价

7.2.1 模型的优点

优点一：本文采用遗传算法和模拟退火算法相结合的方式，对目标地区的气象条件进行聚类，模拟退火算法能够加速遗传算法收敛，快速达到局部最优值完成聚类，这样能够对气候条件进行有效的无监督聚类。

优点二：本文首先使用无监督的 FCM 算法进行分类，之后使用有监督的 SVM 确定分类平面，这样不仅能提升模型分类的准确性和鲁棒性，同时也确定了所属类别的特征。

优点三：本文使用 RNN 对时间进行建模，这样使输入的时间能够无限延展，从而为模型进一步提升性能提供了可能性。同时考虑到时间序列对于结果的影响，这样建模也有利于提升预测准确率。

优点四：本文对区域协同预报模型进行建模，建立有向环图传播网络，先验地确定污染气体浓度升降的原因和影响因素，解决了数据匮乏的问题。同时区域协同预报有效地提升了模型预测的准确性。

7.2.2 模型的缺点

缺点一：由于数据较少，本文没有使用能够抑制梯度消失和梯度爆炸的长短时记忆网络(LSTM，Long Short-Term Memory)，理论上使用该网络在数据量较大的情况下能够保证模型的正确性。

缺点二：考虑到数据样本较少，问题四采用经验主义的方法，对于系数的迭代也没有采用较为深层的神经网络的办法。

7.2.3 模型的改进

对于问题二，可以采用均值漂移算法、距离聚类算法等不需要确定聚类数量的算法。通过得到的结果与本文进行比较。同时在确定分类平面的方法上，可以采用随机森林、K-临近算法和人工神经网络的办法对模型进行改进，以期待在类别更小的情况下获得更为精准的分类。

对于问题三，可以采用隐马尔可夫模型对数据进行处理，将其为不同的天气状况，之后将得到的数据再输入到 LSTM 神经网络中进行训练，使模型具有更好的可解释性，降低了模型对于数据的依赖。

对于问题四，可以加入 Prime 算法、Kruskal 算法搜索有向图中的网络，这样当数据规模较大，结点较多时可以加快运算速度，提升模型性能。

参考文献

[1]环境空气质量指数(AQI)技术规定(试行)[J].中国环境管理干部学院学报,2012,22(01):44.
[2]Fu Zexian,An Jing,Yang Qiuyu,Yuan Haojun,Sun Yuhang,Ebrahimian Homayoun. Skin cancer detection using Kernel Fuzzy C-means and Developed Red Fox Optimization algorithm[J]. Biomedical Signal Processing and Control,2022,71(PA):
[3]刘秋菊王仲英,刘素华.基于遗传模拟退火算法的模糊聚类方法[J].微计算机信息,2006(05):270-272.
[4]孙志刚,王国涛,高萌萌,郜雷阵,蒋爱平.参数优化支持向量机的密封电子设备多余物定位方法研究[J/OL].电子测量与仪器学报:1-11[2021-10-16].
[5]戴连铭. 基于支持向量机的微电网故障诊断研究[D].江苏科技大学,2020.
[6]Gopi Battineni,Nalini Chintalapudi,Francesco Amenta. Machine learning in medicine: Performance calculation of dementia prediction by support vector machines (SVM)[J]. Informatics in Medicine Unlocked,2019,16:
[7]张晓鹏,张兴忠.基于高斯核函数的支持向量机光伏故障诊断研究[J].可再生能源,2021,39(06):760-765.
[8]陈欣昊. 基于 WRF/CMAQ 模型对江苏省霾日的模拟与评估研究[D].南京信息工程大学,2017.
[9]陈彬彬,林长城,杨凯,林文,王宏,余永江.基于CMAQ模式产品的福州市空气质量预报系统[J].中国环境科学,2012,32(10):1744-1752.
[10]Dutt, Sarthika,Ahuja, Neelu Jyothi,Kumar, Manoj. An intelligent tutoring system architecture based on fuzzy neural network (FNN) for special education of learning disabled learners[J]. Education and Information Technologies,2021(prepublish):
[11]杜康吉. 改进的循环神经网络方法及其应用研究[D].东北电力大学,2021.
[12]范竣翔,李琦,朱亚杰,侯俊雄,冯逍.基于 RNN 的空气污染时空预报模型研究[J].测绘科学,2017,42(07):76-83+120.
[13]陈聪,候磊,李乐乐,杨鑫涛.基于GRU改进RNN神经网络的飞机燃油流量预测[J].科学技术与工程,2021,21(27):11663-11673.
[14]Yang Qiong,Ji Hongchao,Fan Xiaqiong,Zhang Zhimin,Lu Hongmei. Retention time prediction in hydrophilic interaction liquid chromatography with graph neural network and transfer learning[J]. Journal of Chromatography A,2021,1656:

附录

附程序 1 问题一 污染物浓度换算 $IAQI_p$ 值函数 IAQI.m MATLAB

```
function IAQIn = IAQI(data,type)
    %阶梯数据
    step_co = [0,2,4,14,24,36,48,60];
    step_so2 =[0,50,150,475,500,1600,2100,2620];
    step_no2 =[0,40,80,180,280,565,750,940];
    step_o3 = [0,100,160,215,265,800];
    step_PM10 =[0,50,150,250,350,420,500,600];
    step_PM2_5=[0,35,75,115,150,250,350,500];
    step_IAQI = [0,50,100,150,200,300,400,500];
     %元胞数组整合污染物的阶梯数据
     step = {step_co,step_so2,step_no2,step_o3,step_PM10,step_PM2_5};
     %数值超范围的考虑
     if type == 4 && data >800
            fprintf("臭氧（O3）最大 8 小时滑动平均浓度值高于 800 μg/m^3 的，不再进行其空气质量分指数计算。");
            IAQIn = 300;
     elseif type ~= 4 && data >step{type}(8)
           fprintf("污染物浓度高于 IAQI=500 对应限值时，不再进行其空气质量分指数计算。");
           IAQIn = 500;
     %计算 IAQ 值
     else
        for i = 1:8
            if data < step{type}(i)
                IAQIn = (step_IAQI(i)-step_IAQI(i-1))/(step{type}(i)-step{type}(i-1))*(data-step{type}(i-1))+step_IAQI(i-1);
                break
            end
        end
     end
     %四舍五入
     IAQIn = round(IAQIn);
end
```

附程序 2 问题一 IAQ 求值函数 task1.m MATLAB

```
function task1()
    %加载原始数据
    row_data=[0.5,8,12,112,27,11;
           0.5,7,16,92,24,10;
           0.6,7,31,169,37,23;
           0.7,8,30,201,47,33];
    %污染名称
    factors=["CO","SO2","NO2","O3","PM10","PM2.5"];
    %记录所有的 IAQ 值
     IAQI6 = zeros(4,6);
     %计算所有 IAQ 值
     for i = 1:4
        for j = 1:6
            IAQI6(i,j)= IAQI(row_data(i,j),j);
        end
     end
     %取最大值
     [IAQ,~]=max(IAQI6,[],2);
     %考虑数值一样时的污染名称
     for i =1:4
        if IAQ(i)>50
            IAQ_name(i) = {factors(IAQI6(i,:)== IAQ(i))};
        else
            IAQ_name(i) = {["none"]};
        end
     end
     x = [25,26,27,28];
     y = IAQI6;
     b= bar(x,y);
     legend(factors,'Location','NorthWest');
     for i = 1:6
        xtips1 = b(i).XEndPoints;
        ytips1 = b(i).YEndPoints;
        labels1 = string(b(i).YData);
        text(xtips1,ytips1,labels1,'HorizontalAlignment','center',...
        'VerticalAlignment','bottom')
     end
     xlabel('日期');
     ylabel('IAQI');
     ylim([0,150]);
     %输出结果
     for i = 1:4
```

```
        if IAQ(i)>50
            fprintf("第%d 日的 IAQ 为%d，首要污染物为",i+24,IAQ(i));
            for j = IAQ_name{i}
                fprintf(j+",\n");
            end
        else
            fprintf("第%d 日的 IAQ 为%d，当天无首要污染物\n",i+24,IAQ(i))
        end
    end
% 第 25 日的 IAQ 为 60，首要污染物为 O3，
% 第 26 日的 IAQ 为 46，当天无首要污染物
% 第 27 日的 IAQ 为 108，首要污染物为 O3，
% 第 28 日的 IAQ 为 137，首要污染物为 O3，
end
```

附程序 3 问题二 基于遗传模拟退火算法的 FCM 聚类 FCM.m MATLAB

```
%% 数据处理
clc
clear
[num,txt,raw]=xlsread("附件 1_2_监测点 A 逐小时污染物浓度与气象实测数据.xlsx");
array1 = isnan (sum(num(:,:),2));
array2 = (num>0);
array2 = any(~array2,2);
array = bitor(array1,array2);
array = ~[0;array];
raw1 = raw(array,:);%%没有负数且没有 NAN 的值
xlswrite('附件 1_2_监测点 A 逐小时污染物浓度与气象实测数据(清洗).xlsx',raw1);
[num1,txt1,raw1]=xlsread('附件 1_2_监测点 A 逐小时污染物浓度与气象实测数据(清洗).xlsx');
size_n = size(num1);
type = [2,3,5,6,4,1];
IAQ = zeros(1,size_n(1));%存储当前小时 IAQ
data = zeros(6,1);
for i  = 1: size_n(1)
   for j = 1:6
       data(j) = IAQI(num1(i,j),type(j));
   [IAQ(i),~] =  max(data(:));
   end
end
%% 基于遗传模拟退火算法的 FCM 聚类
IAQ = IAQ';%第一列为每天 IAQ 第二列为每天 IAQ 的变换
for i = 1:size(IAQ,1)-1
    IAQ(i,3) = (IAQ(i+1,1)-IAQ(i,1))/IAQ(i,1);
    IAQ(i,2) = (IAQ(i+1,1)-IAQ(i,1));
end
num1 =  num1(:,7:11);%输入每天的环境参数
% 对数据归一化处理
for i = 1 : size(num1, 2)
    temp = num1(:, i);
    temp = (temp-min(temp))/(max(temp)-min(temp));
    num1(:, i) = temp;
end
m=size(num1,2);% 样本特征维数
% 中心点范围[lb;ub]
lb=min(num1);
ub=max(num1);
%% 模糊 C 均值聚类参数
% 设置幂指数为 3，最大迭代次数为 20，目标函数的终止容限为 1e-6
options=[3,20,1e-6];
```

```
%类别数 cn
cn=30;
%% 模拟退火算法参数
q =0.8;
T0=100; %初始温度
Tend=1;%终止温度
%% 定义遗传算法参数
sizepop=10; %个体数目(Numbe of individuals)
MAXGEN=10; %最大遗传代数(Maximum number of generations)
NVAR=m*cn; %变量的维数
PRECI=10; %变量的二进制位数(Precision of variables)
GGAP=0.9; %代沟(Generation gap)
pc=0.7;
pm=0.01;
trace=zeros(NVAR+1,MAXGEN);
%建立区域描述器(Build field descriptor)
FieldD=[rep([PRECI],[1,NVAR]);rep([lb;ub],[1,cn]);rep([1;0;1;1],[1,NVAR])];
Chrom=crtbp(sizepop, NVAR*PRECI); % 创建初始种群
V=bs2rv(Chrom, FieldD);
ObjV=ObjFun(num1,cn,V,options); %计算初始种群个体的目标函数值
T=T0;
while T>Tend
    gen=0;
    while gen<MAXGEN
        %分配适应度值
        FitnV=ranking(ObjV);
        SelCh=select('sus', Chrom, FitnV, GGAP); %选择
        SelCh=recombin('xovsp', SelCh,pc); %重组
        SelCh=mut(SelCh,pm); %变异
        V=bs2rv(SelCh, FieldD);
        ObjVSel=ObjFun(num1,cn,V,options); %计算子代目标函数值
        [newChrom newObjV]=reins(Chrom, SelCh, 1, 1, ObjV, ObjVSel);
        V=bs2rv(newChrom,FieldD);
        %是否替换旧个体
        for i=1:sizepop
            if ObjV(i)>newObjV(i)
                ObjV(i)=newObjV(i);
                Chrom(i,:)=newChrom(i,:);
            else
                p=rand;
                if p<=exp((newObjV(i)-ObjV(i))/T)
                    ObjV(i)=newObjV(i);
                    Chrom(i,:)=newChrom(i,:);
                end
```

```
            end
        end
        gen=gen+1; %代计数器增加
        [trace(end,gen),index]=min(ObjV); %遗传算法性能跟踪
        trace(1:NVAR,gen)=V(index,:);
        fprintf(1,'%d ',gen);
    end
    T=T*q;
    fprintf(1,'\n 温度:%1.3f\n',T);
end
[newObjV,center,U]=ObjFun(num1,cn,[trace(1:NVAR,end)]',options); %计算最佳初始聚类中心的目标函数值
% 查看聚类结果
Jb=newObjV
    U=U{1}
    center=center{1}
    maxU = max(U);
    %记录每一类别的行索引
    index1 = find(U(1,:) == maxU);
    index2 = find(U(2, :) == maxU);
    index3 = find(U(3, :) == maxU);
    index4 = find(U(4, :) == maxU);
    index5 = find(U(5,:) == maxU);
    index6 = find(U(6, :) == maxU);
    index7 = find(U(7, :) == maxU);
    index8 = find(U(8, :) == maxU);
    index9 = find(U(9, :) == maxU);
    index10 = find(U(10, :) == maxU);

    index11 = find(U(11,:) == maxU);
    index12 = find(U(12, :) == maxU);
    index13 = find(U(13, :) == maxU);
    index14 = find(U(14, :) == maxU);
    index15 = find(U(15,:) == maxU);
    index16 = find(U(16, :) == maxU);
    index17 = find(U(17, :) == maxU);
    index18 = find(U(18, :) == maxU);
    index19 = find(U(19, :) == maxU);
    index20 = find(U(20, :) == maxU);

    index21 = find(U(21,:) == maxU);
    index22 = find(U(22, :) == maxU);
    index23 = find(U(23, :) == maxU);
    index24 = find(U(24, :) == maxU);
```

```
    index25 = find(U(25,:) == maxU);
    index26 = find(U(26, :) == maxU);
    index27 = find(U(27, :) == maxU);
    index28 = find(U(28, :) == maxU);
    index29 = find(U(29, :) == maxU);
    index30 = find(U(30, :) == maxU);
    index = {index1,index2,index3,index4,index5,index6,index7,index8,index9,index10,index11,index12,index13,index14,index15,index16,index17,index18,index19,index20,index21,index22,index23,index24,index25,index26,index27,index28,index29,index30};
    %计算每一类对于环境的平均影响
    class_sum = zeros(cn,1);
    for i=1:size(index,2)
        for j=1:size(index{i},2)
            class_sum(i,1) = IAQ(j,2)+class_sum(i,1);
        end
    end
    for i=1:size(index,2)
        class_sum(i,1) = class_sum(i,1)/size(index{i},2);
end
```

附程序 4 问题二 支持向量机算法 SVM.py Python

```
import numpy as np
import pandas as pd
from sklearn import svm
from sklearn.model_selection import train_test_split
def show_accuracy(a, b, tip):
    acc = a.ravel() == b.ravel()
    print('%s Accuracy:%.3f' %(tip, np.mean(acc)))

if __name__ == '__main__':
    csvPD=pd.read_csv("SVM_data.csv",header=None)
    csvPD= csvPD.iloc[0:9472,0:6]
    x, y = np.split(csvPD, (5,), axis=1)
    x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=1,
train_size=0.70)
    # 高斯核
    clf1 = svm.SVC(C=5, kernel='rbf', gamma=0.2, decision_function_shape='ovr')
    # 线性核
    # clf1 = svm.SVC(C=0.5, kernel='linear', decision_function_shape='ovr')
    clf1.fit(x_train, y_train.values.ravel())
    y_hat = clf1.predict(x_train)
    show_accuracy(y_hat, y_train.values, 'traing data')
    y_hat_test = clf1.predict(x_test)
    show_accuracy(y_hat_test, y_test.values, 'testing data')
    a = clf1.decision_function(x_test)
    distance = pd.DataFrame(a) #函数距离表
    b = clf1.predict(x_test)
    probability = pd.DataFrame(b) #可能性及预测类别
    # distance.to_csv('函数距离表.csv', header=0, index=0)
   # probability.to_csv('可能性及预测类别.csv', header=0, index=0)
```

附程序 5 问题三 RNN 污染物预测模型 firstmodel.py Python

```
import torch
from torch import nn
from torch.utils.data import DataLoader
import numpy as np
from torch.autograd import Variable
import os
dir = "C:\\Users\\pengz\\Desktop\\taskthree\\task2\\"
os.chdir(dir )
from torch.utils.data import Dataset

# 导入数据集的类
class MyDataset(Dataset):
    def __init__(self, csv_file):
        self.lines = open(csv_file).readlines()

    def __getitem__(self, index):
        # 获取索引对应位置的一条数据
        cur_line = self.lines[index].split(',')
        cur_line1 = self.lines[index].split(',')
        sin_input = np.float32(np.vstack((cur_line[0:15],cur_line1[0:15])))
        cos_output = np.float32(np.vstack((cur_line[15:21],cur_line1[15:21])))
        return sin_input, cos_output
    def __len__(self):
        return len(self.lines)  # MyDataSet 的行数

from torch.utils.data import Dataset
time_step = 8
# 导入数据集的类
class MyDataset(Dataset):
    def __init__(self, csv_file):
        self.lines = open(csv_file).readlines()

    def __getitem__(self, index):
        # 获取索引对应位置的一条数据
        cur_line = self.lines[index].split(',')
        sin_input=np.float32(cur_line[0:15])
        cos_output = np.float32(cur_line[15:21])
        for i in range(time_step-1):
          cur_line = self.lines[index+1+i].split(',')
          sin_input = np.vstack((sin_input,cur_line[0:15]))
          cos_output = np.vstack((cos_output,cur_line[15:21]))

```

```
43.         return np.float32(sin_input),  np.float32(cos_output)
44.
45.     def __len__(self):
46.         return int(len(self.lines)/time_step)  # MyDataSet 的行数
47.
48. from torch import nn
49.
50.
51. class Rnn(nn.Module):
52.     def __init__(self, input_num=15, hidden_num=32, layer_num=3, output_num=6,
    seq_len=1000):
53.         super(Rnn, self).__init__()
54.         self.hidden_num = hidden_num
55.         self.output_num = output_num
56.         self.seq_len = seq_len  # 序列长度
57.         self.rnn = nn.RNN(
58.             input_size=input_num,
59.             hidden_size=hidden_num,
60.             num_layers=layer_num,
61.             nonlinearity='relu',
62.             batch_first=True  # 输入(batch, seq, feature)
63.         )
64.
65.         self.Out = nn.Linear(hidden_num, output_num)
66.
67.     def forward(self, u, h_state):
68.         """
69.         :param u: input 输入
70.         :param h_state: 循环神经网络状态量
71.         :return:
72.         """
73.         # print(u.shape)
74.         r_out, h_state_next = self.rnn(u, h_state)
75.         # print(r_out.shape)
76.         # r_out_reshaped = r_out.view(-1,2, self.hidden_num)  # to 2D data
77.         # print(r_out_reshaped.shape)
78.         outs = self.Out(r_out)
79.         # print(outs.shape)
80.         outs = outs.view(-1, self.seq_len, self.output_num)  # to 3D data
81.         # print(outs.shape)
82.         return outs, h_state_next
83.
84. # device GPU or CPU
85. device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
print('You are using: ' + str(device))

# batch size
batch_size_train = 100
# total epoch(总共训练多少轮)
total_epoch = 50

# 1. 导入训练数据
filename1 =dir + 'alldata.csv'

dataset_train = MyDataset(filename1)
train_size = int(len(dataset_train)*0.8)
test_size = len(dataset_train) - train_size
train_dataset,  test_dataset = torch.utils.data.random_split(dataset_train, [train_size, test_size])

    train_loader = DataLoader(train_dataset, batch_size=batch_size_train, shuffle=True, drop_last=True)

    # 2. 构建模型，优化器
    rnn = Rnn(seq_len=batch_size_train).to(device)
    optimizer = torch.optim.SGD(rnn.parameters(), lr=0.01, momentum=0.8)
    # scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=500, gamma=0.1)  # Learning Rate Decay
    criterion = nn.MSELoss()  # mean square error
    train_loss_list = []  # 每次 epoch 的 loss 保存起来
    total_loss = 1  # 网络训练过程中最大的 loss


    # 3. 模型训练
    def train_rnn(epoch):
        hidden_state = None  # 隐藏状态初始化
        global total_loss
        mode = True
        rnn.train(mode=mode)  # 模型设置为训练模式
        loss_epoch = 0  # 一次 epoch 的 loss 总和

        for idx, (sin_input, cos_output) in enumerate(train_loader):
            sin_input_np = sin_input.numpy()  # 1D
            cos_output_np = cos_output.numpy()  # 1D
            # print(sin_input_np.shape)
            sin_input_torch = Variable(torch.from_numpy(sin_input_np))  # 3D
            cos_output_torch = Variable(torch.from_numpy(cos_output_np))  # 3D
            prediction, hidden_state = rnn(sin_input_torch.to(device), hidden_state)
```

```
127.
128.            # 要把 h_state 重新包装一下才能放入下一个 iteration，不然会报错!!!
129.            hidden_state = Variable(hidden_state.data).to(device)
130.            # print(prediction.transpose(1,0).shape,cos_output_torch.shape)
131.            loss = criterion(prediction.transpose(1,0), cos_output_torch.to(device))  #
    cross entropy loss
132.            optimizer.zero_grad()  # clear gradients for this training step
133.            loss.backward()  # back propagation, compute gradients
134.            optimizer.step()  # apply gradients
135.
136.            loss_epoch += loss.item()  # 将每个 batch 的 loss 累加，直到所有数据都计算完毕
137.            if idx == len(train_loader) - 1:
138.                print('Train Epoch:{}\tLoss:{:.9f}'.format(epoch, loss_epoch))
139.                train_loss_list.append(loss_epoch)
140.                if loss_epoch < total_loss:
141.                    total_loss = loss_epoch
142.                    torch.save(rnn, '..\\model\\rnn_model.pkl')  # save model
143.
144.    if __name__ == '__main__':
145.        # 模型训练
146.        import torch.backends.cudnn as cudnn
147.        cudnn.benchmark = True
148.        cudnn.deterministic = False
149.        print("Start Training...")
150.        for i in range(total_epoch):  # 模型训练 1000 轮
151.            train_rnn(i)
152.        torch.save(rnn, dir + '/rnn_model3.pkl')
153.        print("Stop Training!")
154.    plt.plot(train_loss_list)
155.    plt.ylabel('loss')
156.    plt.xlabel('epoch')
157.    plt.savefig(dir +'/trainloss.jpg')
158.    batch_size_test = 100
159.    # 导入数据
160.    filename1 = dir + 'alldata.csv'
161.    test_loader = DataLoader(test_dataset, batch_size=batch_size_test, shuffle=False,
    drop_last=True)
162.
163.    criterion = nn.MSELoss()  # mean square error
164.
165.    import matplotlib.pyplot as plt
166.
167.
168.    # rnn 测试
```

```
def test_rnn():
    net_test = Rnn(seq_len=batch_size_test).to(device)
    net_test = torch.load(dir + 'rnn_model3.pkl')  # load model
    hidden_state = None
    test_loss = 0
    net_test.eval()
    with torch.no_grad():
        for idx, (sin_input, cos_output) in enumerate(test_loader):
            sin_input_np = sin_input.numpy()  # 1D
            cos_output_np = cos_output.numpy()  # 1D
            sin_input_torch = Variable(torch.from_numpy(sin_input_np))  # 3D
            cos_output_torch = Variable(torch.from_numpy(cos_output_np))  # 3D
            prediction, hidden_state = net_test(sin_input_torch.to(device), hidden_state)
            # print(prediction.shape)
            prediction = prediction.transpose(1,0)
            if idx == 0:
                predict_value = prediction.squeeze()
                real_value = cos_output_torch.squeeze()
            else:
                predict_value = torch.cat([predict_value, prediction.squeeze()], dim=0)
                real_value = torch.cat([real_value, cos_output_torch.squeeze()], dim=0)
            loss = criterion(prediction,cos_output_torch.to(device))
            test_loss += loss.item()

    print('Test set: Avg. loss: {:.9f}'.format(test_loss))
    return predict_value, real_value
if __name__ == '__main__':
    # 模型测试
    print("testing...")
    p_v, r_v = test_rnn()
    print(r_v.shape)
    # 对比图
    data = p_v[350:400,1,:].reshape(-1,6)
    data1 = r_v[350:400,1,:].reshape(-1,6)
    plt.plot(data[:,4].cpu(), c='green')
    plt.plot(data1[:,4].cpu(), c='orange')
    plt.ylabel('prediction')
    plt.xlabel('data')
    plt.yticks(np.arange(0,1,0.05))

    plt.savefig(dir + '训练 50 次第 O3 气体气体 50 时间检测结果.jpg')
```

```
210.
211.         plt.show()
212.     print("stop testing!")
```

附程序 6 问题三 RNN 预测 O3 模型 modifimodel.py Python

```
import torch
from torch import nn
from torch.utils.data import DataLoader
import numpy as np
from torch.autograd import Variable
import os
import os
dir = "C:\\Users\\pengz\\Desktop\\taskthree\\task2\\"
os.chdir(dir)

from torch.utils.data import Dataset
time_step = 8
# 导入数据集的类
class MyDataset(Dataset):
    def __init__(self, csv_file):
        self.lines = open(csv_file).readlines()

    def __getitem__(self, index):
        # 获取索引对应位置的一条数据
        cur_line = self.lines[index].split(',')
        sin_input=np.float32(cur_line[0:20])
        cos_output = np.float32(cur_line[20])
        for i in range(time_step-1):
          cur_line = self.lines[index+1+i].split(',')
          sin_input = np.vstack((sin_input,cur_line[0:20]))
          cos_output = np.vstack((cos_output,cur_line[20]))

        return np.float32(sin_input),  np.float32(cos_output)

    def __len__(self):
        return int(len(self.lines)/time_step)  # MyDataSet 的行数

from torch import nn


class Rnn(nn.Module):
    def __init__(self, input_num=20, hidden_num=32, layer_num=2, output_num=1,
seq_len=1000):
        super(Rnn, self).__init__()
        self.hidden_num = hidden_num
        self.output_num = output_num
        self.seq_len = seq_len  # 序列长度
```

```
42.
43.         self.rnn = nn.RNN(
44.             input_size=input_num,
45.             hidden_size=hidden_num,
46.             num_layers=layer_num,
47.             nonlinearity='relu',
48.             batch_first=True  # 输入(batch, seq, feature)
49.         )
50.
51.         self.Out = nn.Linear(hidden_num, output_num)
52.
53.     def forward(self, u, h_state):
54.         """
55.         :param u: input 输入
56.         :param h_state: 循环神经网络状态量
57.         :return:
58.         """
59.         # print(u.shape)
60.         r_out, h_state_next = self.rnn(u, h_state)
61.         # print(r_out.shape)
62.         # r_out_reshaped = r_out.view(-1,2, self.hidden_num)  # to 2D data
63.         # print(r_out_reshaped.shape)
64.         outs = self.Out(r_out)
65.         # print(outs.shape)
66.         outs = outs.view(-1, self.seq_len, self.output_num)  # to 3D data
67.         # print(outs.shape)
68.         return outs, h_state_next
69.
70.
71.
72. # device GPU or CPU
73. device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
74. print('You are using: ' + str(device))
75.
76. # batch size
77. batch_size_train = 100
78. # total epoch(总共训练多少轮)
79. total_epoch = 50
80.
81. # 1. 导入训练数据
82. filename = dir + 'O3d.csv'
83. dataset_train = MyDataset(filename)
84. train_size = int(len(dataset_train)*0.8)
85. test_size = len(dataset_train) - train_size
```

```
train_dataset,  test_dataset = torch.utils.data.random_split(dataset_train, [train_size, test_size])

train_loader = DataLoader(train_dataset, batch_size=batch_size_train, shuffle=False, drop_last=True)

# 2. 构建模型，优化器
rnn = Rnn(seq_len=batch_size_train).to(device)
optimizer = torch.optim.SGD(rnn.parameters(), lr=0.02, momentum=0.8)
# scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=500, gamma=0.1)  # Learning Rate Decay
criterion = nn.MSELoss()  # mean square error
train_loss_list = []  # 每次 epoch 的 loss 保存起来
total_loss = 1  # 网络训练过程中最大的 loss


# 3. 模型训练
    def train_rnn(epoch):
        hidden_state = None  # 隐藏状态初始化
        global total_loss
        mode = True
        rnn.train(mode=mode)  # 模型设置为训练模式
        loss_epoch = 0  # 一次 epoch 的 loss 总和

        for idx, (sin_input, cos_output) in enumerate(train_loader):
            sin_input_np = sin_input.numpy()  # 1D
            cos_output_np = cos_output.numpy()  # 1D
            # print(sin_input_np.shape)
            sin_input_torch = Variable(torch.from_numpy(sin_input_np))  # 3D
            cos_output_torch = Variable(torch.from_numpy(cos_output_np))  # 3D
            prediction, hidden_state = rnn(sin_input_torch.to(device), hidden_state)

            # 要把 h_state 重新包装一下才能放入下一个 iteration，不然会报错!!!
            hidden_state = Variable(hidden_state.data).to(device)
            # print(prediction.transpose(1,0).shape,cos_output_torch.shape)
            loss = criterion(prediction.transpose(1,0), cos_output_torch.to(device))  # cross entropy loss
            optimizer.zero_grad()  # clear gradients for this training step
            loss.backward()  # back propagation, compute gradients
            optimizer.step()  # apply gradients

            loss_epoch += loss.item()  # 将每个 batch 的 loss 累加，直到所有数据都计算完毕
            if idx == len(train_loader) - 1:
                print('Train Epoch:{}\tLoss:{:.9f}'.format(epoch, loss_epoch))
```

```
            train_loss_list.append(loss_epoch)
            if loss_epoch < total_loss:
                total_loss = loss_epoch
                torch.save(rnn, '..\\model\\rnn_model.pkl')  # save model


if __name__ == '__main__':
    # 模型训练
    import torch.backends.cudnn as cudnn
    cudnn.benchmark = True
    cudnn.deterministic = False
    print("Start Training...")
    for i in range(total_epoch):  # 模型训练1000轮
        train_rnn(i)
    torch.save(rnn, dir +'O3rnn_model.pkl')
    print("Stop Training!")

import matplotlib.pyplot as plt

plt.plot(train_loss_list)
plt.ylabel('loss')
plt.xlabel('epoch')
plt.savefig(dir +'O3trainloss.jpg')

batch_size_test = 100

# 导入数据
filename = dir + 'O3d.csv'

test_loader = DataLoader(test_dataset, batch_size=batch_size_test, shuffle=False,
drop_last=True)

criterion = nn.MSELoss()  # mean square error

import matplotlib.pyplot as plt


# rnn 测试
def test_rnn():
    net_test = Rnn(seq_len=batch_size_test).to(device)
    net_test = torch.load(dir +'O3rnn_model.pkl')  # load model
    hidden_state = None
    test_loss = 0
    net_test.eval()
```

```
        with torch.no_grad():
            for idx, (sin_input, cos_output) in enumerate(test_loader):
                sin_input_np = sin_input.numpy()  # 1D
                cos_output_np = cos_output.numpy()  # 1D
                sin_input_torch = Variable(torch.from_numpy(sin_input_np))  # 3D
                cos_output_torch = Variable(torch.from_numpy(cos_output_np))  # 3D

                prediction, hidden_state = net_test(sin_input_torch.to(device), hidden_state)
                # print(prediction.shape)
                prediction = prediction.transpose(1,0)
                if idx == 0:
                    predict_value = prediction.squeeze()
                    real_value = cos_output_torch.squeeze()
                else:
                    predict_value = torch.cat([predict_value, prediction.squeeze()], dim=0)
                    real_value = torch.cat([real_value, cos_output_torch.squeeze()], dim=0)

                loss = criterion(prediction, cos_output_torch.to(device))
                test_loss += loss.item()

        print('Test set: Avg. loss: {:.9f}'.format(test_loss))
        return predict_value, real_value


    if __name__ == '__main__':
        # 模型测试
        print("testing...")
        p_v, r_v = test_rnn()
        print(r_v.shape)
        # 对比图 3
        data = p_v.reshape(-1,1)
        data2 = r_v.reshape(-1,1)
        plt.plot(data[150:200].cpu(), c='green')
        plt.plot(data2[150:200].cpu(), c='orange')
        plt.xlabel('data')
        plt.yticks(np.arange(0,1,0.05))
        plt.ylabel('prediction')
        plt.savefig(dir +'O3 数据训练 50 次气体检测结果总.jpg')

        plt.yticks(np.arange(0,1,0.05))
        plt.show()
```

```
210.        print("stop testing!")
```

附程序 7 问题三 RNN 预测空气污染物浓度 prediction.py Python

```
import torch
from torch import nn
from torch.utils.data import DataLoader
import numpy as np
from torch.autograd import Variable
import os
dir = "C:\\Users\\pengz\\Desktop\\taskthree\\task2\\"
os.chdir(dir )

from torch.utils.data import Dataset


# 导入数据集的类
class MyDataset(Dataset):
    def __init__(self, csv_file):
        self.lines = open(csv_file).readlines()

    def __getitem__(self, index):
        # 获取索引对应位置的一条数据
        cur_line = self.lines[index].split(',')
        cur_line1 = self.lines[index].split(',')

        sin_input = np.float32(np.vstack((cur_line[0:15],cur_line1[0:15])))
        cos_output = np.float32(np.vstack((cur_line[15:21],cur_line1[15:21])))

        return sin_input, cos_output

    def __len__(self):
        return len(self.lines)  # MyDataSet 的行数


from torch.utils.data import Dataset
time_step = 8
# 导入数据集的类
class MyDataset(Dataset):
    def __init__(self, csv_file):
        self.lines = open(csv_file).readlines()

    def __getitem__(self, index):
        # 获取索引对应位置的一条数据
        cur_line = self.lines[index].split(',')
        sin_input=np.float32(cur_line[0:15])
        cos_output = np.float32(cur_line[15:21])
```

```
        for i in range(time_step-1):
          cur_line = self.lines[index+1+i].split(',')
          sin_input = np.vstack((sin_input,cur_line[0:15]))
          cos_output = np.vstack((cos_output,cur_line[15:21]))

        return np.float32(sin_input),  np.float32(cos_output)

    def __len__(self):
        return int(len(self.lines)/time_step)  # MyDataSet 的行数

from torch import nn


class Rnn(nn.Module):
    def __init__(self, input_num=15, hidden_num=32, layer_num=3, output_num=6, seq_len=1000):
        super(Rnn, self).__init__()
        self.hidden_num = hidden_num
        self.output_num = output_num
        self.seq_len = seq_len  # 序列长度

        self.rnn = nn.RNN(
            input_size=input_num,
            hidden_size=hidden_num,
            num_layers=layer_num,
            nonlinearity='relu',
            batch_first=True  # 输入(batch, seq, feature)
        )

        self.Out = nn.Linear(hidden_num, output_num)

    def forward(self, u, h_state):
        """
        :param u: input 输入
        :param h_state: 循环神经网络状态量
        :return:
        """
        # print(u.shape)
        r_out, h_state_next = self.rnn(u, h_state)
        # print(r_out.shape)
        # r_out_reshaped = r_out.view(-1,2, self.hidden_num)  # to 2D data
        # print(r_out_reshaped.shape)
        outs = self.Out(r_out)
        # print(outs.shape)
```

```
        outs = outs.view(-1, self.seq_len, self.output_num)  # to 3D data
        # print(outs.shape)
        return outs, h_state_next

# device GPU or CPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print('You are using: ' + str(device))

# batch size
batch_size_train = 100
# total epoch(总共训练多少轮)
total_epoch = 500

    # 1. 导入训练数据
    filename1 = dir+'alldata.csv'

    dataset_train = MyDataset(filename1)
    train_size = int(len(dataset_train)*0.8)
    test_size = len(dataset_train) - train_size
    train_dataset,  test_dataset = torch.utils.data.random_split(dataset_train,
   [train_size, test_size])

    train_loader = DataLoader(train_dataset, batch_size=batch_size_train, shuffle=True,
   drop_last=True)


    batch_size_test = 100

    # 导入数据
    filename1 =dir+ 'alldata.csv'

    test_loader = DataLoader(test_dataset, batch_size=batch_size_test, shuffle=False,
   drop_last=True)

    criterion = nn.MSELoss()  # mean square error

    # 存储差值
    # 导入数据
    import matplotlib.pyplot as plt
    batch_size_test = 100
    filename1 = dir+"call.csv"
    dataset_train = MyDataset(filename1)

```

```
    test_loader = DataLoader(dataset_train, batch_size=batch_size_test, shuffle=False, drop_last=True)

    criterion = nn.MSELoss()  # mean square error


    # rnn 测试
    def test_rnn():
        net_test = Rnn(seq_len=batch_size_test).to(device)
        net_test = torch.load(dir+'rnn_model3.pkl')  # load model
        hidden_state = None
        test_loss = 0
        net_test.eval()
        with torch.no_grad():
            for idx, (sin_input, cos_output) in enumerate(test_loader):
                sin_input_np = sin_input.numpy()  # 1D
                cos_output_np = cos_output.numpy()  # 1D
                sin_input_torch = Variable(torch.from_numpy(sin_input_np))  # 3D
                cos_output_torch = Variable(torch.from_numpy(cos_output_np))  # 3D
                prediction, hidden_state = net_test(sin_input_torch.to(device), hidden_state)
                # print(prediction.shape)
                prediction = prediction.transpose(1,0)
                if idx == 0:
                    predict_value = prediction.squeeze()
                    real_value = cos_output_torch.squeeze()
                else:
                    predict_value = torch.cat([predict_value, prediction.squeeze()], dim=0)
                    real_value = torch.cat([real_value, cos_output_torch.squeeze()], dim=0)

                loss = criterion(prediction, cos_output_torch.to(device))
                test_loss += loss.item()

        print('Test set: Avg. loss: {:.9f}'.format(test_loss))
        return predict_value, real_value


    if __name__ == '__main__':
        # 模型测试
        print("testing...")
        p_v, r_v = test_rnn()
        print(p_v.shape)
```

```
        # 对比图
        data = p_v[891:900,:,:].reshape(-1,6)
        plt.plot(data[:,0].cpu(), c='green')
        plt.plot(data[:,1].cpu(), c='orange')
        plt.plot(data[:,2].cpu(), c='blue')
        plt.plot(data[:,3].cpu(), c='cyan')
        # plt.plot(data[:,4].cpu(), c='black')
        plt.plot(data[:,5].cpu(), c='yellow')
        plt.ylabel('prediction')
        plt.xlabel('time')
        plt.savefig(dir+'某次训练 c 的预测结果.jpg')


    p_np = p_v.numpy()
    import pandas as pd
    data1 = pd.DataFrame(p_v.reshape(-1,6))
    data1.to_csv(dir+'C 总.csv')
```

附程序 8 问题三 RNN 预测 O_3 程序 predictionO3.py Python

```
import torch
from torch import nn
from torch.utils.data import DataLoader
import numpy as np
from torch.autograd import Variable
import os
dir = "C:\\Users\\pengz\\Desktop\\taskthree\\task2\\"
os.chdir(dir)


from torch.utils.data import Dataset
time_step = 8
# 导入数据集的类
class MyDataset(Dataset):
    def __init__(self, csv_file):
        self.lines = open(csv_file).readlines()

    def __getitem__(self, index):
        # 获取索引对应位置的一条数据
        cur_line = self.lines[index].split(',')
        sin_input=np.float32(cur_line[0:20])
        cos_output = np.float32(cur_line[20])
        for i in range(time_step-1):
          cur_line = self.lines[index+1+i].split(',')
          sin_input = np.vstack((sin_input,cur_line[0:20]))
          cos_output = np.vstack((cos_output,cur_line[20]))

        return np.float32(sin_input),  np.float32(cos_output)

    def __len__(self):
        return int(len(self.lines)/time_step)  # MyDataSet 的行数

from torch import nn

class Rnn(nn.Module):
    def __init__(self, input_num=20, hidden_num=32, layer_num=2, output_num=1,
seq_len=1000):
        super(Rnn, self).__init__()
        self.hidden_num = hidden_num
        self.output_num = output_num
        self.seq_len = seq_len  # 序列长度

        self.rnn = nn.RNN(
```

```
            input_size=input_num,
            hidden_size=hidden_num,
            num_layers=layer_num,
            nonlinearity='relu',
            batch_first=True  # 输入(batch, seq, feature)
        )

        self.Out = nn.Linear(hidden_num, output_num)

    def forward(self, u, h_state):
        """
        :param u: input 输入
        :param h_state: 循环神经网络状态量
        :return:
        """
        # print(u.shape)
        r_out, h_state_next = self.rnn(u, h_state)
        # print(r_out.shape)
        # r_out_reshaped = r_out.view(-1,2, self.hidden_num)  # to 2D data
        # print(r_out_reshaped.shape)
        outs = self.Out(r_out)
        # print(outs.shape)
        outs = outs.view(-1, self.seq_len, self.output_num)  # to 3D data
        # print(outs.shape)
        return outs, h_state_next

# device GPU or CPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print('You are using: ' + str(device))

# batch size
batch_size_train = 100
# total epoch(总共训练多少轮)
total_epoch = 50


filename = dir+ 'O3d.csv'
dataset_train = MyDataset(filename)
train_size = int(len(dataset_train)*0.8)
test_size = len(dataset_train) - train_size
train_dataset,  test_dataset = torch.utils.data.random_split(dataset_train, [train_size,
   test_size])

```

```
train_loader = DataLoader(train_dataset, batch_size=batch_size_train, shuffle=False, drop_last=True)

batch_size_test = 100


# rnn 测试
def test_rnn():
    net_test = Rnn(seq_len=batch_size_test).to(device)
    net_test = torch.load(dir + 'O3rnn_model.pkl')  # load model
    hidden_state = None
    test_loss = 0
    net_test.eval()
    with torch.no_grad():
        for idx, (sin_input, cos_output) in enumerate(test_loader):
            sin_input_np = sin_input.numpy()  # 1D
                cos_output_np = cos_output.numpy()  # 1D
                sin_input_torch = Variable(torch.from_numpy(sin_input_np))  # 3D
                cos_output_torch = Variable(torch.from_numpy(cos_output_np))  # 3D
                prediction, hidden_state = net_test(sin_input_torch.to(device), hidden_state)
                # print(prediction.shape)
                prediction = prediction.transpose(1,0)
                if idx == 0:
                    predict_value = prediction.squeeze()
                    real_value = cos_output_torch.squeeze()
                else:
                    predict_value = torch.cat([predict_value, prediction.squeeze()], dim=0)
                    real_value = torch.cat([real_value, cos_output_torch.squeeze()], dim=0)

                loss = criterion(prediction, cos_output_torch.to(device))
                test_loss += loss.item()

        print('Test set: Avg. loss: {:.9f}'.format(test_loss))
        return predict_value, real_value


 if __name__ == '__main__':
        # 模型测试
        print("testing...")
        p_v, r_v = test_rnn()
        print(p_v.shape)
```

```
        # 对比图
        data = p_v[891:1000,:].reshape(-1,1)
        plt.plot(data[:,0].cpu(), c='green')
        plt.xlabel('data')
        plt.ylabel('time')
        plt.savefig(dir+'c 臭氧预测结果.jpg')

    p_np = p_v.numpy()
    import pandas as pd
    data1 = pd.DataFrame(data)
    data1.to_csv(dir + 'C 臭氧预测结果.csv')
```

附程序 9 问题四 有向环图传播网络 Graph.py MATLAB

```
%% 有向环图传播网络
clc
clear
output=zeros(18,4);
for  class = 1:6
    % 浓度变化矩阵
    [num_h,txt_h,raw_h]=xlsread("环境变化.xlsx");
    [num_a,txt_a,raw_a]=xlsread("A_浓度变化.xlsx");
    [num_a1,txt_a1,raw_a1]=xlsread("A1_浓度变化.xlsx");
     [num_a2,txt_a2,raw_a2]=xlsread("A2_浓度变化.xlsx");
     [num_a3,txt_a3,raw_a3]=xlsread("A3_浓度变化.xlsx");
     % 位置矩阵 (0, 1, 2, 3)
     location = [0,0;-14.4846, -1.9699;-6.6716, 7.5953;-3.3543, -5.0138];
     % 数据处理
     MAX = [max(num_h(:,1:3)),1,1];%不处理风速风向
     MIN = [min(num_h(:,1:3)),0,0];
     num_h = (num_h-MIN)./(MAX-MIN);
     wind = num_h(:,4:5);
     wind(:,1) =  num_h(:,4)*3.6;%风速单位化为 km/h
     % 方向处理 (为方便计算, 规定自正西方向至监测点时角度为 0)
     wind(:,2)=abs(wind(:,2)-ones(size(wind,1),1)*270);
     wind(:,2) = wind(:,2)/180*pi;
     % 环境影响因子 (0-1, 0-2, 0-3, 1-2, 1-3, 2-3)
     loc = zeros(6,2);%位置向量
     loc(1,1)=location(2,1)-location(1,1);loc(1,2)=location(2,2)-location(1,2);
     loc(2,1)=location(3,1)-location(1,1);loc(2,2)=location(3,2)-location(1,2);
     loc(3,1)=location(4,1)-location(1,1);loc(3,2)=location(4,2)-location(1,2);
     loc(4,1)=location(3,1)-location(2,1);loc(4,2)=location(3,2)-location(2,2);
     loc(5,1)=location(4,1)-location(2,1);loc(5,2)=location(4,2)-location(2,2);
     loc(6,1)=location(4,1)-location(3,1);loc(6,2)=location(4,2)-location(3,2);
     loc1 = zeros(6,2);%位置向量归一
     for i = 1:size(loc,1)
         loc1(i,1)= loc(i,1)/sqrt(loc(i,1)^2+loc(i,2)^2);
         loc1(i,2)= loc(i,2)/sqrt(loc(i,1)^2+loc(i,2)^2);
     end
     w = zeros(1,6);%传递系数
     w_k = ones(1,6);%传递参数
     w_out = ones(1,4);%传递参数
     loss_c = zeros(1,4);%浓度的误差
     loss_k = zeros(1,6);%每个参数的损失
     % 位置浓度
     for time = 1:size(num_h,1)-1-3
```

```
        concentration = [num_a(time,class),num_a1(time,class),num_a2(time,class),num_a3(time,class)];%浓度
        env = num_h(time,:);
        for i = 1:size(w,2)
            w(1,i) = wind(time,1)*(cos(wind(time,2))* loc1(i,1)+ sin(wind(time,2))* loc1(i,2)+eps)/sqrt(loc(i,1)^2+loc(i,2)^2);%风速/风向·位置向量
        end
        if(any(isnan(w)))%只要有一条通道为 NaN,就跳过
            continue;
        end
        %参数更新
        w_k = tanh(w_k); %激活函数
        %w_k = max(w_k,0); %激活函数

concentration(1,1)=concentration(1,1)+w(1,1)*w_k(1,1)+w(1,2)*w_k(1,2)+w(1,3)*w_k(1,3);

concentration(1,2)=concentration(1,2)-w(1,1)*w_k(1,1)+w(1,4)*w_k(1,4)+w(1,5)*w_k(1,5);

concentration(1,3)=concentration(1,3)-w(1,2)*w_k(1,2)-w(1,4)*w_k(1,4)+w(1,6)*w_k(1,6);

concentration(1,4)=concentration(1,4)-w(1,3)*w_k(1,3)-w(1,5)*w_k(1,5)-w(1,6)*w_k(1,6);
        %计算损失
        if(~isnan(num_a(time+1,class))&&~isnan(num_a(time,class)))
            loss_c(1,1) =-(num_a(time+1,class) - concentration(1,1));%浓度的误差
            loss_all(1,time)=num_a(time+1,class);
            loss_all(2,time)=concentration(1,1);
        end
        if(~isnan(num_a1(time+1,class))&&~isnan(num_a1(time,class)))
            loss_c(1,2) =-(num_a1(time+1,class) - concentration(1,2));%浓度的误差
            loss_all(3,time)=num_a1(time+1,class);
            loss_all(4,time)=concentration(1,2);
        end
        if(~isnan(num_a2(time+1,class))&&~isnan(num_a2(time,class)))
            loss_c(1,3) =-(num_a2(time+1,class) - concentration(1,3));%浓度的误差
            loss_all(5,time)=num_a2(time+1,class);
            loss_all(6,time)=concentration(1,3);
        end
        if(~isnan(num_a3(time+1,class))&&~isnan(num_a3(time,class)))
            loss_c(1,4) =-(num_a3(time+1,class) - concentration(1,4));%浓度的误差
            loss_all(7,time)=num_a3(time+1,class);
            loss_all(8,time)=concentration(1,4);
        end
        disp(sum(abs(loss_c)));
        if (isnan(sum(loss_c)))
```

```
            break
        end
        %梯度回传
        for i = size(loss_c,2)
            loss_c(1,i) = 1-tanh(loss_c(1,i))^2;
        end
        w_k(1,1) =
w_k(1,1)-loss_c(1,1)/(w(1,1)*w_k(1,1)+w(1,2)*w_k(1,2)+w(1,3)*w_k(1,3))*(w(1,1)*w_k(1,1))
+  loss_c(1,2)/(-w(1,1)*w_k(1,1)+w(1,4)*w_k(1,4)+w(1,5)*w_k(1,5))*(w(1,1)*w_k(1,1));
        w_k(1,2) =
w_k(1,2)-loss_c(1,1)/(w(1,1)*w_k(1,1)+w(1,2)*w_k(1,2)+w(1,3)*w_k(1,3))*(w(1,2)*w_k(1,2))
+  loss_c(1,3)/(-w(1,2)*w_k(1,2)-w(1,4)*w_k(1,4)+w(1,6)*w_k(1,6))*(w(1,2)*w_k(1,2));
        w_k(1,3) =
w_k(1,3)-loss_c(1,1)/(w(1,1)*w_k(1,1)+w(1,2)*w_k(1,2)+w(1,3)*w_k(1,3))*(w(1,3)*w_k(1,3))
+  loss_c(1,4)/(-w(1,3)*w_k(1,3)-w(1,5)*w_k(1,5)-w(1,6)*w_k(1,6))*(w(1,3)*w_k(1,3));
        w_k(1,4) =
w_k(1,4)-loss_c(1,2)/(-w(1,1)*w_k(1,1)+w(1,4)*w_k(1,4)+w(1,5)*w_k(1,5))*(w(1,4)*w_k(1,4)
) +  loss_c(1,3)/(-w(1,2)*w_k(1,2)-w(1,4)*w_k(1,4)+w(1,6)*w_k(1,6))*(w(1,4)*w_k(1,4));
        w_k(1,5) =
w_k(1,5)-loss_c(1,2)/(-w(1,1)*w_k(1,1)+w(1,4)*w_k(1,4)+w(1,5)*w_k(1,5))*(w(1,5)*w_k(1,5)
) +  loss_c(1,4)/(-w(1,3)*w_k(1,3)-w(1,5)*w_k(1,5)-w(1,6)*w_k(1,6))*(w(1,5)*w_k(1,5));
        w_k(1,6) =
w_k(1,6)-loss_c(1,3)/(-w(1,2)*w_k(1,2)-w(1,4)*w_k(1,4)+w(1,6)*w_k(1,6))*(w(1,6)*w_k(1,6)
) + loss_c(1,4)/(-w(1,3)*w_k(1,3)-w(1,5)*w_k(1,5)-w(1,6)*w_k(1,6))*(w(1,6)*w_k(1,6)) ;
    end
    figure
    subplot(2,2,1)
    draw= loss_all(1,:);
    plot(draw');
    title('A预测图','FontSize',20);
    xlabel('时间/天','FontSize',20);
        xlabel('浓度','FontSize',20);
        subplot(2,2,2)
        draw= loss_all(2,:);
        plot(draw','Color',[0 0 1]);
        title('A真实图','FontSize',20);
        xlabel('时间/天','FontSize',20);
        xlabel('浓度','FontSize',20);
        subplot(2,2,3)
        draw= loss_all(3,:);
        plot(draw');
        title('A1预测图','FontSize',20);
        xlabel('时间/天','FontSize',20);
        xlabel('浓度','FontSize',20);
```

```
        subplot(2,2,4)
        draw= loss_all(4,:);
        plot(draw','Color',[0 0 1]);
        title('A1 真实图','FontSize',20);
        xlabel('时间/天','FontSize',20);
        xlabel('浓度','FontSize',20);
        time = 819;
        concentration =
   [num_a(time,class),num_a1(time,class),num_a2(time,class),num_a3(time,class)];
        for j = 1:3
            for i = 1:size(w,2)
                w(1,i) = wind(time,1)*(cos(wind(time,2))* loc1(i,1)+ sin(wind(time,2))*
   loc1(i,2)+eps)/sqrt(loc(i,1)^2+loc(i,2)^2);%风速/风向·位置向量
            end
            time = time+1;
            w_k = tanh(w_k); %激活函数
   concentration(1,1)=concentration(1,1)+w(1,1)*w_k(1,1)+w(1,2)*w_k(1,2)+w(1,3)*w_k(1,3);
   concentration(1,2)=concentration(1,2)-w(1,1)*w_k(1,1)+w(1,4)*w_k(1,4)+w(1,5)*w_k(1,5);
   concentration(1,3)=concentration(1,3)-w(1,2)*w_k(1,2)-w(1,4)*w_k(1,4)+w(1,6)*w_k(1,6);
   concentration(1,4)=concentration(1,4)-w(1,3)*w_k(1,3)-w(1,5)*w_k(1,5)-w(1,6)*w_k(1,6);
            output(j+3* class-3,:)=concentration(1,:);
        end
    end
    out = zeros(12,6);
    for i =1:4
        for j = 1:6
            temp1 = j*3-3+1;
            temp2 = j*3;
            temp3 = i*3-3+1;
            temp4 = i*3;
            out(temp3:temp4,j) = output(temp1:temp2,i);
        end
   end
```