队伍编号	728
赛道	A

基于 LSTM 对流量数据的预测

摘 要

随着 5G 时代的发展，流量数据的监测显得尤为重要。本文正是基于此，对题目中给定的数据集设计了相关算法对数据进行了分析处理。而流量数据属于时间序列分析的一种。对于时间序列分析的研究方法有很多，其中使用神经网络是一个合适的方法。

问题一是对给定小区的短期预测，我们使用 LSTM 神经网络作为我们的模型。对于缺失的数据，LRTC 法对数据的复原有着相对更优的效果。所以我们使用 LSTM-LRTC 模型作为我们的训练算法。

问题二是对给定小区的长期预测，在 LSTM 的基础上将其与 BP 神经网络结合起来进行改进，形成 LSTM-BP 神经网络。模型测试结果表明效果相对单纯 LSTM 更优。

问题三是对数据的开放探索，将从时间和空间两个维度对流量数据进行挖掘。同时，对于出现行为异常的数据，进行了行为判断分析。

关键词：流量数据，LSTM，LRTC，时间序列

目录

1. 问题重述

随着 5G 时代的到来，流量数据的监控与预测具有重要意义。本问题正是基于此，对不同小区从 2018 年 3 月 1 日到 2018 年 4 月 19 日的流量数据进行了分析。由于基站数量巨大，无法通过人工实时关注每个基站的流量变化，需要给每个基站设置根据时段自动开关载频的程序。这样就需要知道一段时间内基站流量关于时段的变化值，特别是基站在每个小时的上下行流量值。而且总体而言，总流量的增加需要进行物理扩容，这就需要提早预估出基站需要物理扩容的时间，从而可以更早的进行规划和设计。

问题一对给定小区的流量数据进行分析，以小时为单位，预测 2018 年 4 月 20 日到 2018 年 4 月 26 日内一周的流量变化。属于短期预测问题。

问题二对给定小区的流量数据进行分析，按天为单位将对应小区的流量数据进行求和并预测 2020 年 11 月 1 日到 11 月 25 日的日流量数据，属于长期预测问题。

问题三对给定数据集进行进一步挖掘与探讨，是否能有什么新的发现。

2. 问题分析

2.1 问题一分析

问题一是一个典型的时间序列分析与预测问题。时间序列的常见分析方法有很多，例如回归分析，ARIMA 模型，灰色预测法，神经网络等。而对于本题的数据集，我们使用 LSTM 进行模型训练与预测。相比于传统的 BP 神经网络和 RNN，LSTM 具有更少的参数，训练起来更方便。由于原始数据存在缺失的情况，而且缺失值较为集中，这里使用 LRTC 方法对均值进行填充处理。

2.2 问题二分析

问题二属于长期时间序列预测，同样可以使用 LSTM 神经网络。不过，单纯的 LSTM 神经网络虽然仍然对长期信息保留一定记忆但影响不够，需要对网络结构进行测试调整。可以尝试在 LSTM 神经网络的基础上增加新的层，也可尝试不同神经网络的效果结合来进行测试。

2.3 问题三分析

问题三较为开放。对于本题的数据集，在前面两个问题已经研究了时间与流量之间的关系，那么在问题三中，可以研究空间的流量分布。即以小区编号为自变量，针对每个小区在这段时间内的平均流量进行数据分析，观察流量数据大小，也可以研究数据的聚类等方面。然后时间维度，可以观察一天内的流量变化确定流量数据的高峰期，以及 49 天内的日平均流量（忽略空间因素），这样可以进行平均流量的数据预测与分析。还可以消除空间因素后研究 49 天内每个小时的平均流量，进行流量数据的行为测量与分析，从而挖掘异常。

3. 模型假设和符号描述

3.1 模型假设

1. 缺失值允许被合理填充
2. 每个值都是有意义的不存在错误数据只有异常数据
3. 长期预测的结果允许有一定量的误差

3.2 符号描述

表 1 列出了下文会出现的符号。若有符号不在表 1 内，请以下文阐释为准。

符号	意义
f_t, i_t, o_t	遗忘门输出，输入门输出，输出门输出
h_t, x_t	LSTM 第 t 步接收的序列项和输出
C, c_t	LSTM 候选向量和总线状态
tanh	双曲正切函数
$\sigma(x)$	表示 sigmoid 函数
$W_f, b_f, W_o, b_o, W_i, b_i$	分别表示遗忘门、输出门和输出门的权重和偏置项
α_k, ρ_k	拉格朗日乘子
$rank(x)$	矩阵/张量 x 的秩
T, M	张量，矩阵
$\langle a, b \rangle$	a 和 b 的内积
$L(x)$	x 的拉格朗日函数

4. 建立模型

4.1 问题一模型

4.1.1 LSTM 神经网络

LSTM 神经网络是由 Sepp Hochreiter 等人提出的神经网络模型[1]，在 RNN 的基础上引入了遗忘门。传统的 RNN 可以看作是同一结构的反复迭代，每次迭代的结果都会被传到下一个值进行再处理。RNNs 一旦展开，可以将之视为一个所有层共享同样权值的深度前馈神经网络。虽然它们的目的是学习长期的依赖性，但理论的和经验的证据表明很难学习并长期保存信息[2]。而 LSTM 则是对 RNN 的改进，是一种特殊的 RNN 模型，被更广泛地应用于文本预测、时间序列预测等领域。

LSTM 网络与普通 RNN 一样，都有重复的单元结构。但不同的是，传统的普通 RNN 单元只有比较简单的网络结构，而 LSTM 的单元由三个不同的门组成：输入门，输出门和遗忘门。基本架构如图 1 所示[3]：

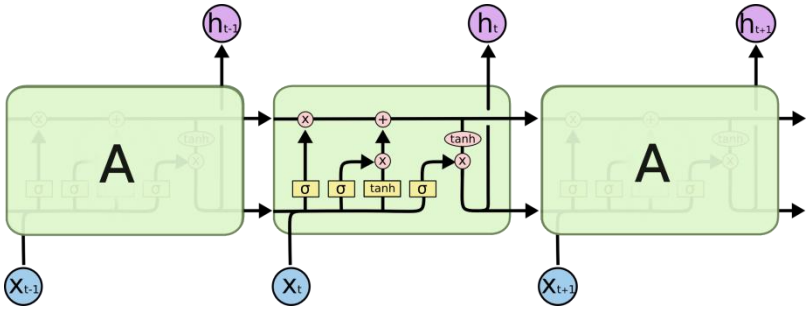


图 1. LSTM 单元的基本架构

图 1 最上方的 c_{t-1} 到 c_t 的总线贯穿始终，是整个 LSTM 网络的核心。在总线的下方是三个门。从左到右的三个门分别为遗忘门，输入门和输出门。

对于遗忘门，它的作用是将接收过的信息进行选择性地遗忘，可以主动调节不同位置信息的作用大小。对此，我们有：

$$(1) \qquad f_t = \sigma(W_f(h_{t-1}, x_t) + b_f)$$

而输入门的作用是更新单元的状态。将新的信息有选择性地输入来代替被遗忘的信息，并生成候选向量 C。下面的方程解释了输入门生成的候选向量：

$$(2) \qquad i_t = \sigma(W_i(h_{t-1}, x_t) + b_i)$$

$$(3) \qquad C = \tanh(W_C(h_{t-1}, x_t) + b_C)$$

输出门可以给出结果，同时将先前的信息保存到隐层中去。同样的，我们有：

$$(4) \qquad o_t = \sigma(W_o(h_{t-1}, x_t) + b_o)$$

$$(5) \qquad c_t = f_t \cdot c_{t-1} + i_t \cdot C$$

$$(6) \qquad h_t = o_t \cdot \tanh(c_t)$$

我们可以看到，LSTM 宏观上也是关于 x_{t-1} 和 x_t 的函数，但是由于多了门控单元对长期信息和短期信息的不同处理模式，网络能够对先前的长期信息保持一定记忆，这克服了传统 RNN 只能针对先前的短期数据进行计算的缺点。

LSTM 网络比较适合于时间序列的中短期预测，对于长期预测效果仍然欠佳。问题一的短期时间预测可以利用 LSTM 进行调参与测试。

4.1.2 LRTC 法对均值进行填充

LRTC-TNN 法是由 Xinyu Chen, Jinming Yang 和 Lijun Sun 等人提出的一种有关缺失值填充的方法[4]。时间序列的缺失主要有两种，分为随机缺失和非随机缺失。而缺失值填充的根本任务是从已经观测到的和后面可被观测到的数据出发，根据这些可观测数据的特征推断出缺失数据的合理特征。

LRTC 方法是张量转换技术的一种，它建立在模型的低秩假设上。假设有一个观测到的有缺失三阶张量 y，若想将它还原为另一个完整的三阶张量，可以建立这样的优化模型：

$$(7)\qquad \min_{x} rank(x)$$
$$s.t. P_{\Omega}(x) = P_{\Omega}(y)$$

而对于三阶张量秩的求解，我们可以通过引入三个参数对它进行简化：

$$(8)\qquad \min_{x} \sum_{k=1}^{3} \alpha_k \left\| x_{(k)} \right\|$$

而对于这一问题的求解算法，我们可以建立 ADMM 模型对其进行分析。在此之前，我们先对（8）列出其广义拉格朗日函数。式（9）中 M 和 T 分别代表矩阵和张量。

$$(9)\qquad L(M,\{x_k,T_k\}_{k=1}^{3}) = \sum_{k=1}^{3}(\alpha_k \left\| x_{k(k)} \right\| + \frac{\rho_k}{2} \left\| x_{k(k)} - M_{(k)} \right\|_F^2 + \langle x_k - M, T_k \rangle)$$

这样，这个拉格朗日函数可被分解为三个子问题：

$$(10)\qquad x_k^{l+1} := \arg\min_{x} L(M,\{x_k^{l+1},T_k^l\}_{k=1}^{3}),$$
$$M^{l+1} := \arg\min_{M} L(M,\{x_k^{l+1},T_k^l\}_{k=1}^{3}),$$
$$T_k^{l+1} := T_k^l + \rho_k(x_k^{l+1} - M^{l+1}),$$

我们可以通过迭代的方式处理这一问题，迭代的顺序为：

$$(11) x_1^{l+1}, x_2^{l+1}, x_3^{l+1}, M^{l+1}, T_1^{l+1}, T_2^{l+1}, T_3^{l+1}$$

经过求解，发现这一问题的最优解是对 M 进行矩阵奇异值分解（SVD）得到的，求解算法的流程图与复原效果如图 2 所示[5]：

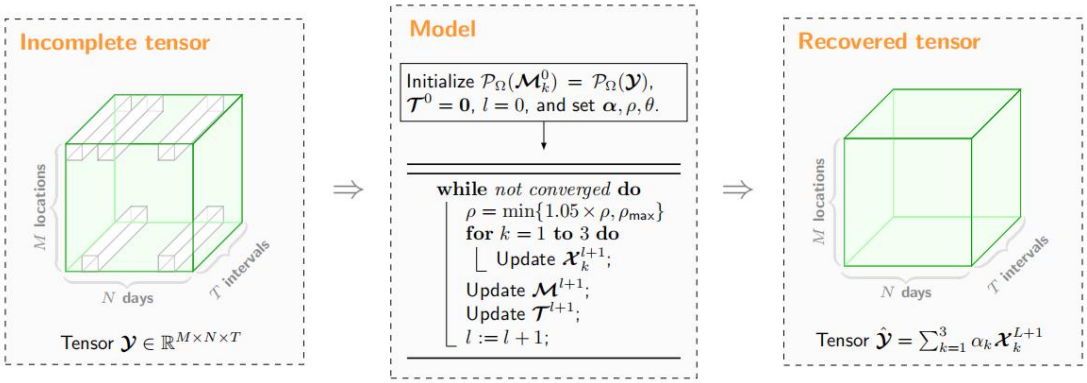


图 2. LRTC 法的算法与效果

可以看到，该算法对于存在缺失值情况的张量填充确实有显著效果。

4.2 问题二模型

问题二属于长期预测的范畴，同样可以使用 LSTM 神经网络进行训练。但在尝试过程中发现单纯的 LSTM 效果并不好。经过测试，我们发现在 LSTM 后面加上三个全连接层构成的一个多隐层 BP 神经网络可以使得效果有显著提升。

我们不妨将我们的改进模型称为 LSTM-BP 神经网络。

表 2 为我们的 LSTM-BP 神经网络的结构：

Layer (type)	Output Shape	Param #
lstm_5 (LSTM)	(None, 20)	1760
dense_20 (Dense)	(None, 20)	420
dense_21 (Dense)	(None, 10)	210
dense_22 (Dense)	(None, 5)	55
dense_23 (Dense)	(None, 1)	6
Total params: 2,451 Trainable params: 2,451 Non-trainable params: 0		

表 2. 使用 LSTM-BP 模型的参数

这一模型结合了传统的 BP 神经网络与 LSTM 神经网络，首先是一个 LSTM 层进行主要的学习，然后我们通过增加全连接层的方式进行逐步形成多对一的映射。首先我们尝试添加了一个拥有 20 个单元的隐藏层（表 2 中 dense_20），发现测试准确率有显著提高。接下来，逐步尝试增加表 2 中的 dense_21，dense_22，发现准确率逐步增大。再增加一个拥有 3 个隐藏单元的隐藏层时发现准确率反而下降了。所以，我们选取这样一个组合模型作为测试用。

提取数据时我们注意到，有很多小区的日数据存在严重缺失，对于这些缺失的数据，我们的模型效果不会很好。这时我们采取数据填充的方式进行处理。不同于问题一，问题二的数据太少，使用 LRTC 法效果并不显著。这时我们对于数据量超过正常值的一半的使用均值填充，对于数据量明显过少甚至没有的选择使用 0 填充。

4.3 问题三模型

对于问题三，考察两个维度的宏观影响：

时间维度上：我们可以忽略空间因素，以小时为单位求解两个流量数据的均值。然后统筹观察流量数据随时间的变化，分析基本趋势与周期性并判断是否有异常值。若出现某个点或者某段流量值突然增大或减小，则应该对流量数据的异常行为进行分析判断是否有特殊原因（例如节假日，自然灾害等）。

另外，对于每一日的平均流量数据，我们也可以对其进行进一步探讨，分析与预测。而一天内 24 小时的流量数据，可以分析出流量大小分布，观察流量更大的时间段方便流量数据的分配。

空间维度上：我们可以忽略时间因素，以小区标号为自变量将数据平均化，观察流量大小随空间的分布。由于小区数量高达 130000，这里我们可以随机抽样分析其空间分布特征。

5. 预测结果与分析

5.1 问题一结果

问题一选用 LSTM 作为我们的模型。同时，我们发现原始数据存在缺失的情况，而且缺失的部分相对较为集中。我们使用了均值填充法和 LRTC 法对缺失值进行了填充处理，并训练神经网络进行分析。

均值填充法相对于 LRTC 法更为直接，但是经过测试，发现拟合效果并不好。而且数据集本身缺失位置过于密集，用均值填充容易造成填充值与实际值不匹配的误差。而 LRTC 法对张量的处理则使得时间序列的缺值预测与填充更具合理性。

另外，对于 LSTM 网络，我们考虑过使用星期作为特征，但实验结果表明差异并不大。所以我们排除了星期的因素进行模型训练。经过在第 221 号小区数据上的测试，LRTC 法的训练结果如图 3 所示，均值填充法结果如图 4 所示。

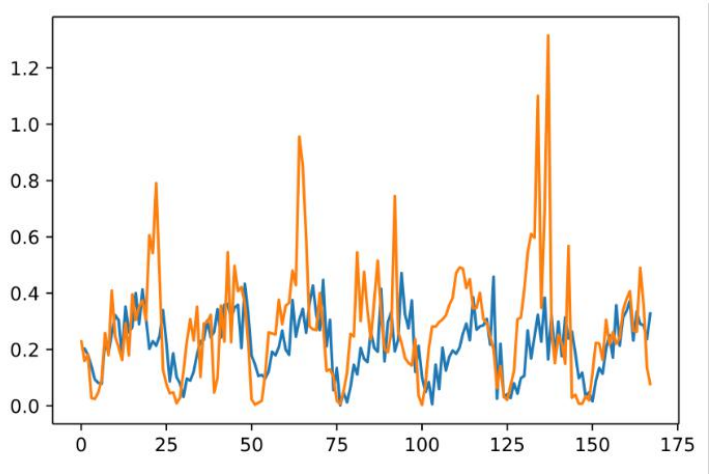


图 3. LRTC 法与 LSTM 配合的预测结果，橙色为实际值蓝色为预测值

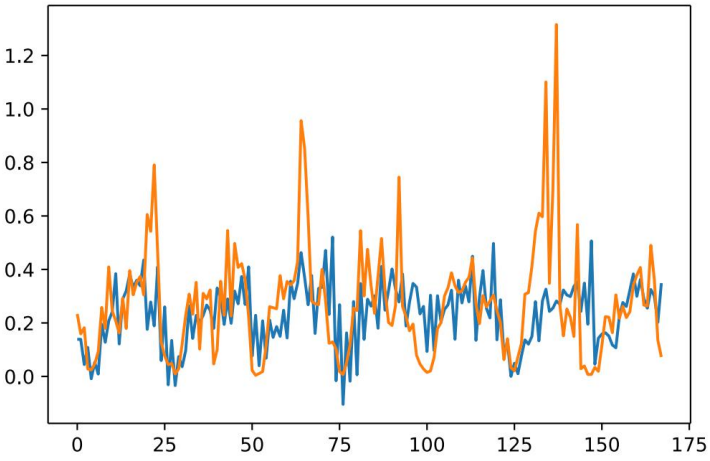


图 4. 均值填充法和 LSTM 的配合结果，橙色为实际值蓝色为预测值

很显然，LRTC 法的预测结果显得更有规律而且更加平稳。测试结果显示 LRTC 法最终的 MSE 更优，为 0.2073。

5.2 问题二结果

问题二使用的 LSTM-BP 模型，我们随机选取了一个小区的日流量数据进行预测。选取的是 7601 号小区的结果。

对于上行流量数据，模型的训练时间为 3.18 秒，实际数据与模型预测数据绘制为图 5 所示：

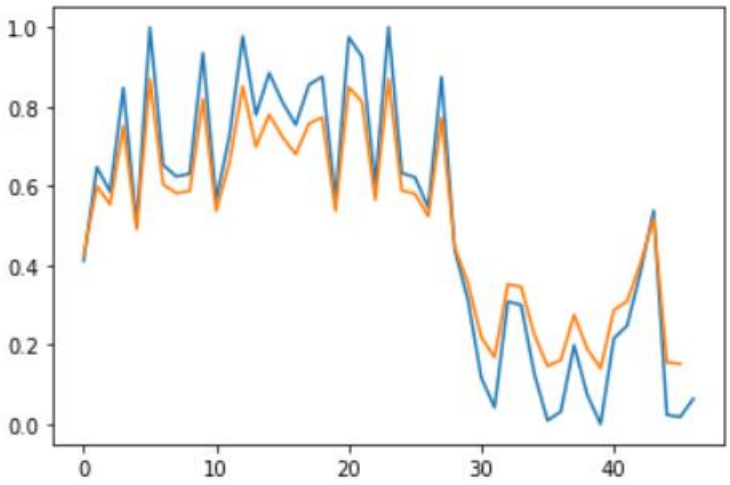


图 5. 7601 号小区的日上行总流量数据

蓝色的线表示预测值，橙色的线表示实际值。可以看到，两条折线的拟合程度还是相当高的。这说明我们的模型取得了比较好的效果。

另外，当以日为单位进行长期多步预测时，我们发现，时间较长的情况下模型的预测结果最终会逐渐收敛到一个值附近。这也是 LSTM 在进行时间序列的长期预测时的一个缺点。

对于缺失严重的数据，我们采取的策略是用数据填充的方式进行处理。不同于问题一，问题二的数据太少，使用 LRTC 法效果并不显著。这时我们对于数据量超过正常值的一半的使用均值填充，对于数据量明显过少甚至没有的选择使用 0 填充。

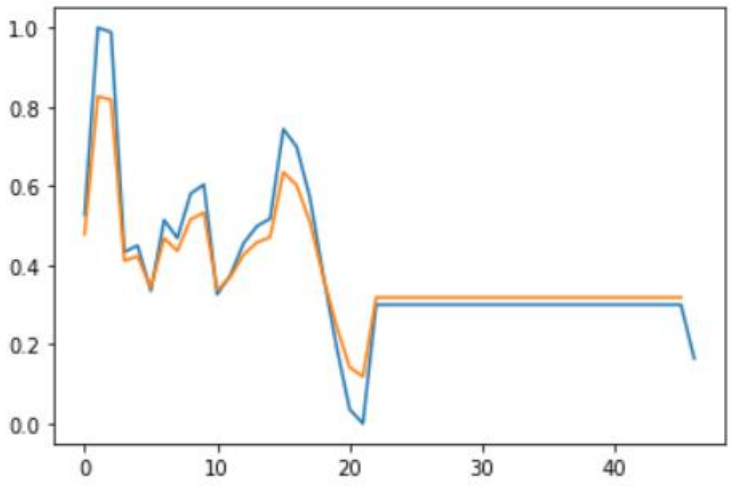


图 6. 279 号小区填充以后的上行流量数据

根据图 6 可以看到，模型对这样的大批量缺失数据仍然有较好的效果。

模型训练的损失随训练次数（epoch）图像如图 7 所示：

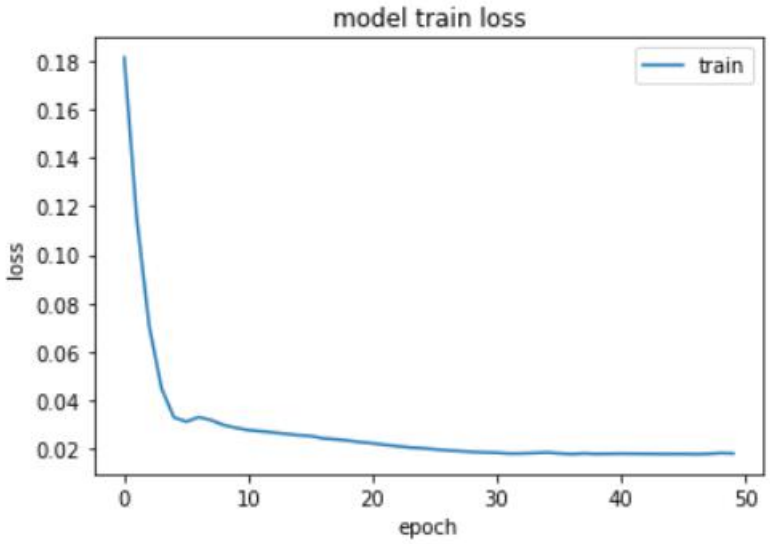


图 7. 模型的训练损失与训练次数的关系图

可以看到，模型的误差是逐步收敛到一个接近于 0 的常数的。对于每个模型我们训练 50 个 epoch 以保证收敛性。

5.3 问题三结果

对于本题给定的数据集，考虑两方面因素：

对于时间因素：我们按小时为单位整理了从 2018 年 3 月 1 日到 2018 年 4 月 19 日内平均流量数据（其中 4 月 14 日数据缺失），现将结果绘制为图 3 和图 4：

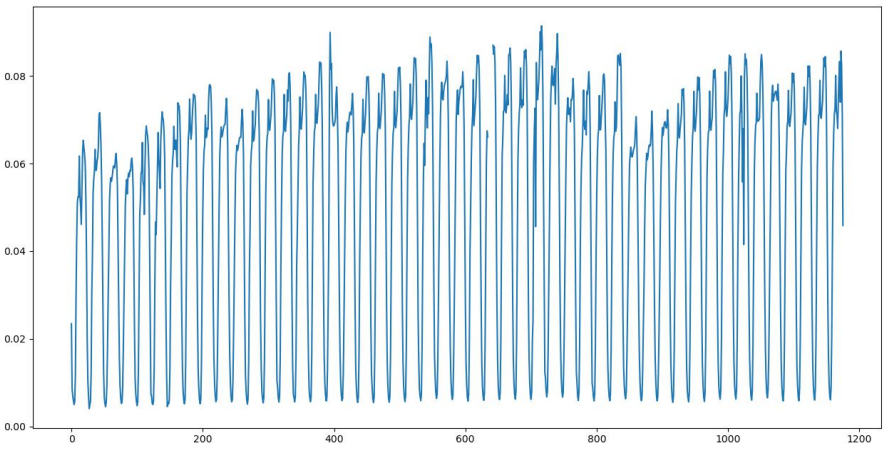


图 3. 总体的上行业务量数据（GB）

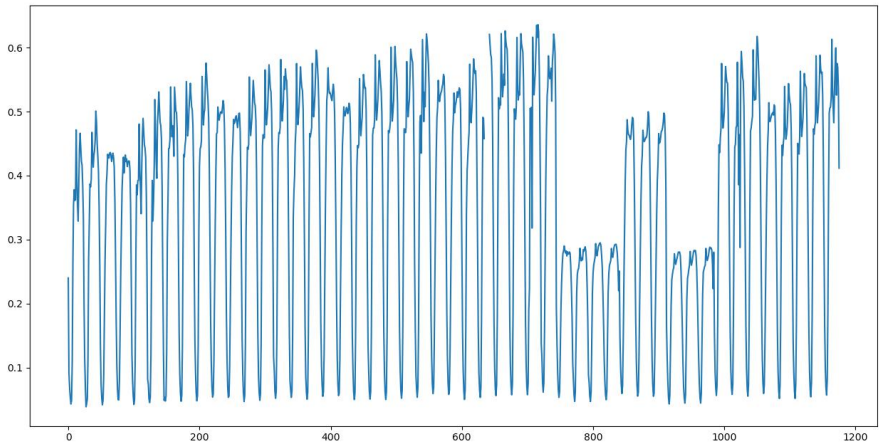


图 4. 总体的下行业务量数据（GB）

我们可以观察到，对于上行业务量数据相对平稳，呈现以 24 小时为周期的一定周期性。但对于下行业务量进行数据测量与行为分析，可以观察到，存在两个明显异常的时间段：第一个是 4 月 1 日到 4 月 4 日这一时间段，第二个是 4 月 8 日到 4 月 10 日这一时间段。这可能是由于 4 月 1 日到 4 月 4 日前后与清明节假期有关，而 4 月 8 日到 10 日可能是由于周末或者发生了其他大事件导致数据出现明显下降）

一天 24 小时的平均流量数据如图 5 所示：

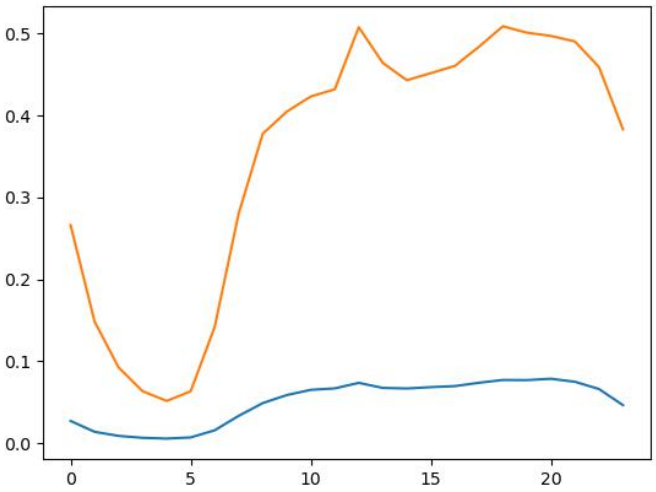


图 5. 一日内平均上行流量，蓝线表示上行业务量，橙色线表示下行业务量

大体上数据呈现有规律变化。相对而言，0 点到 5 点由于多数用户处于休息时间段所以流量数据相对较低。5 点以后更多用户开始使用，这时流量开始增加，到 12 点-13 点达到最大值。下午到晚上 22 点是用户数量较多的一个时间段，所以流量数据在这一时间段保持较高状态。从 22 点以后流量数据开始下降。

49 日内日平均流量数据如图 6 和图 7 所示：

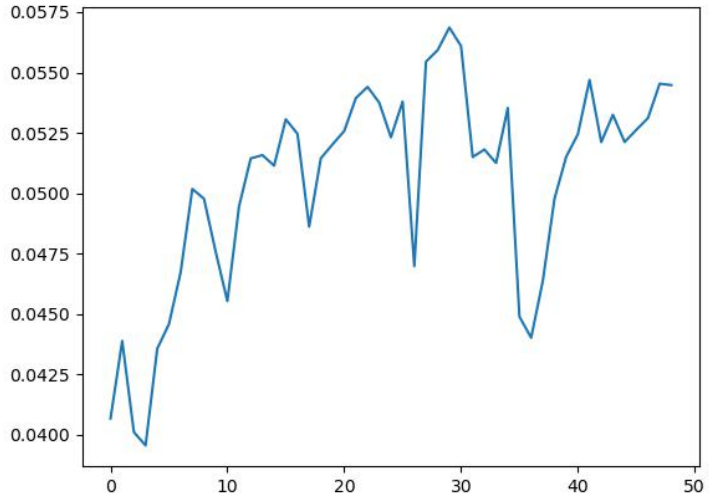


图 6. 日平均上行流量数据

可以看到，日平均上行流量数据虽有波动但大致呈增长趋势，而且增长速度逐渐缓慢，类似于对数增长。

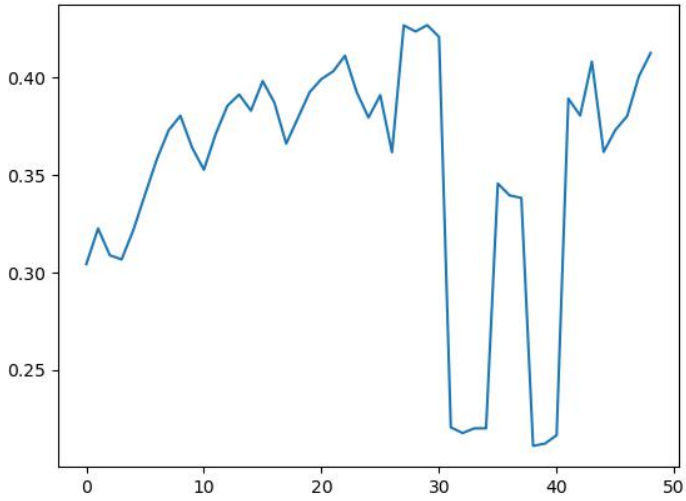


图 7. 每日平均下行流量数据

下行流量大体上也是增长趋势，但在 4 月 1 日到 4 月 4 日和 4 月 8 日到 4 月 10 日有大幅度的下跌。行为异常区间带与图 4 描述高度符合，所以在这两个时间段很有可能出现了影响流量数据的异常事件。

对于空间因素：我们随机抽取了 100 个城市，观察平均流量的空间分布，绘制它们的流量数据与城市代号的散点图如图 8 和图 9 所示：

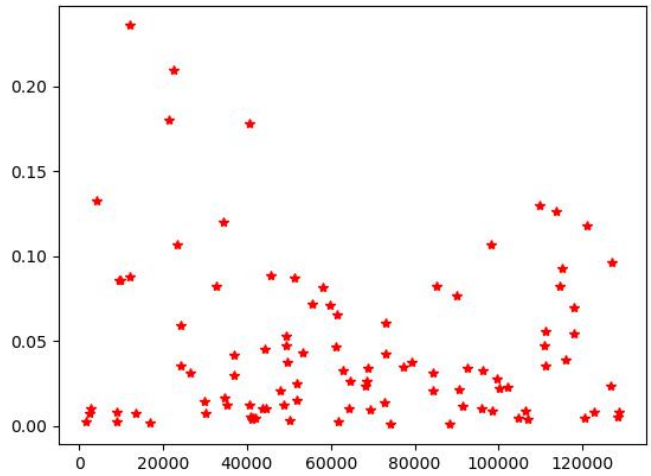


图 8. 城市代号与平均上行流量数据的关系

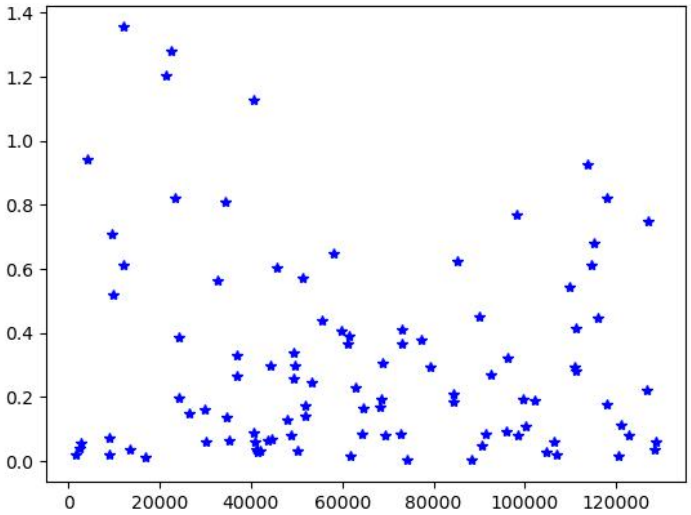


图 9. 城市代号与下行流量数据的关系

可以看出，代号与流量数据虽然没什么关系，但是图 8 和图 9 的位置分布极为相似。这表示，上行流量大的城市，下行流量极有可能也比较大。另外，从空间角度对数据进行分析，有利于日后对新数据进行空间维度上的聚类。

6. 方法优缺点分析

优点：

1. 训练时间不长，开销较低
2. 准确度相对较高
3. LRTC 相比纯粹的均值填充，对缺失数据的复原更合理
4. 模型缺点易于改进

缺点：

1. 得到结果仍然有较大的误差。一方面，这是由于数据的预测时间过长导致的；另一方面，数据缺失比较多
2. 对数据的处理对空间占用比较大，这是由于数据条目本身比较多而且不易清洗
3. 未能很好揭示并利用上行流量数据和下行流量数据的相关性

改进方案：

1. 尝试改进 LSTM 模型结构，观察是否存在更优模型
2. 对于长期预测，可以尝试 LSTNN 等分解式模型[6]
3. 传统方法对长期预测的分析还可尝试 MISMO[7]等方法，即“多输入-多输出”模型进行测试
4. 时间开销的优化问题，可以采取设计并行式机器学习程序，利用分布式计算解决实际问题。
5. 神经网络的参数调节可以引入遗传算法进行自学习

参考文献

[1]. Sepp Hochreiter, Jurgen Schmidhuber, *Long Short Time Memory*, 1997
[2]. Yann LeCun, Yoshua Bengio,Geoffrey Hinton, *Deep Learning*, 2015
[3]. Colah, *Understanding LSTM Networks*, https://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2021.01.19
[4] . Xinyu Chen, Jinming Yang, Lijun Sun, *A Nonconvex Low-Rank Tensor Completion Model for Spatiotemporal Traffic Data Imputation*, 2020
[5] . Xinyu Chen, Jinming Yang, Lijun Sun, *A Nonconvex Low-Rank Tensor Completion Model for Spatiotemporal Traffic Data Imputation*, 2020
[6]. Souhaib Ben Taieb, Gianluca Bontempi, Antti Sorjamaa and Amaury Lendasse *Long-Term Prediction of Time Series by combining Direct and MIMO Strategies,* 2014
[7]. Guokun Lai, Wei-Cheng Chang, Yiming Yang, Hanxiao Liu, *Modeling Long- and Short-Term Temporal Patterns with Deep Neural Networks*, 2018

附录：
编程环境：CPU intel i7, windows 10, python 3.8.2, jupyter notebook, tensorflow 2.2.0，pytorch 1.6.0, Excel
主要代码见下页：
由于 Jupyter notebook 每个 cell 可能重复执行或者乱序执行，所以代码仅供参考
编程过程中有手动的文件操作

experiment2

January 18, 2021

1 Experiment 2

This is an further job of the former experiment about LSTM. In the former jobs, I have done the following things * Altering 2 main parameters, look_back and hidden_size from 0 to 200 or so * exploring the choice of features, including onehot of week and hour

In today's work, I want to have a further understanding of LSTM Model, especially the following items: * How does hidden layers act in LSTM and what's it used for memoring the former information? * How can we combine different models(LSTM and Linear models)to get better effect? * How can we use the tool of feature engeneering to make our paper more talktive? * Is there a possibility to predict accurately using the tools now? ### Let's start!

[233]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
from torch import nn
from torch.autograd import Variable
from numpy.linalg import inv as inv
import datetime
%matplotlib inline
```

2 1.

2.1 1.1

[234]:
```
#    '%Y/%m/%d %H:%M:%S'     time_stamp
#   '%Y/%m'
def str2stamp(str_time):
    datetime_time = datetime.datetime.strptime(str_time, '%Y/%m/%d %H:%M:%S')
    timestamp_time = datetime_time.timestamp()
    return timestamp_time
```

[235]:
```
#
def raw2complete(raw_array, starting_time = '2018/3/1 0:00:00'):
    complete_array = np.zeros((1200, 34), dtype = np.float32)
    starting_stamp = str2stamp(starting_time)
    for day_data in raw_array:
```

```
        time_str = str(day_data[1]) + ' ' + str(day_data[2])
        if '2018' not in time_str:
            time_str = time_str.replace('018', '2018')
        if '-' in str(day_data[1]):
            datetime_time = datetime.datetime.strptime(time_str, '%Y-%m-%d %H:
↪%M:%S')
        else:
            datetime_time = datetime.datetime.strptime(time_str, '%Y/%m/%d %H:
↪%M:%S')

        timestamp_time = datetime_time.timestamp()
        timetuple_time = datetime_time.timetuple()
        rank = int((timestamp_time - starting_stamp) / 3600)
        complete_array[rank][0] = day_data[3]
        complete_array[rank][1] = day_data[4]
        complete_array[rank][2 + timetuple_time.tm_wday] = 1
        complete_array[rank][9 + timetuple_time.tm_hour] = 1
        complete_array[rank][33] = timestamp_time
    for i in range(1200):
        if complete_array[i][0] == 0:
            present_stamp = starting_stamp + 3600 * i
            datetime_time = datetime.datetime.fromtimestamp(present_stamp)
            timetuple_time = datetime_time.timetuple()
            complete_array[i][2 + timetuple_time.tm_wday] = 1
            complete_array[i][9 + timetuple_time.tm_hour] = 1
            complete_array[i][33] = timestamp_time
    return complete_array
```

[14]:
```
def ten2mat(tensor, mode):
    return np.reshape(np.moveaxis(tensor, mode, 0), (tensor.shape[mode], -1),␣
↪order = 'F')
```

[15]:
```
def mat2ten(mat, tensor_size, mode):
    index = list()
    index.append(mode)
    for i in range(tensor_size.shape[0]):
        if i != mode:
            index.append(i)
    return np.moveaxis(np.reshape(mat, list(tensor_size[index]), order = 'F'),␣
↪0, mode)
```

[16]:
```
def svt_tnn(mat, alpha, rho, theta):
    """This is a Numpy dependent singular value thresholding (SVT) process."""
    u, s, v = np.linalg.svd(mat, full_matrices = 0)
    vec = s.copy()
    vec[theta :] = s[theta :] - alpha / rho
    vec[vec < 0] = 0
```

```
    return np.matmul(np.matmul(u, np.diag(vec)), v)
```

[17]:
```
def LRTC(sparse_tensor, alpha, rho, theta, epsilon, maxiter):
    """Low-Rank Tenor Completion with Truncated Nuclear Norm, LRTC-TNN."""

    dim = np.array(sparse_tensor.shape)
    pos_missing = np.where(sparse_tensor == 0)

    X = np.zeros(np.insert(dim, 0, len(dim))) # \boldsymbol{\mathcal{X}}
    T = np.zeros(np.insert(dim, 0, len(dim))) # \boldsymbol{\mathcal{T}}
    Z = sparse_tensor.copy()
    last_tensor = sparse_tensor.copy()
    snorm = np.sqrt(np.sum(sparse_tensor ** 2))
    it = 0
    while True:
        rho = min(rho * 1.05, 1e5)
        for k in range(len(dim)):
            X[k] = mat2ten(svt_tnn(ten2mat(Z - T[k] / rho, k), alpha[k], rho,␣
↪np.int(np.ceil(theta * dim[k]))), dim, k)
        Z[pos_missing] = np.mean(X + T / rho, axis = 0)[pos_missing]
        T = T + rho * (X - np.broadcast_to(Z, np.insert(dim, 0, len(dim))))
        tensor_hat = np.einsum('k, kmnt -> mnt', alpha, X)
        tol = np.sqrt(np.sum((tensor_hat - last_tensor) ** 2)) / snorm
        last_tensor = tensor_hat.copy()
        it += 1
        if (tol < epsilon) or (it >= maxiter):
            break
    return tensor_hat
```

[249]:
```
#          rawdata  0    shape (1200,)
def fill_LRTC(raw_data):
    alpha = np.ones(3) / 3
    rho = 1e-5
    theta = 0.30
    epsilon = 1e-5
    maxiter = 1000
    raw_data_3d1 = raw_data[:, 0].copy().reshape(8, 10, 15)
    result_3d1 = LRTC(raw_data_3d1, alpha, rho, theta, epsilon, maxiter)
    result1 = result_3d1.reshape(1200)
    raw_data_3d2 = raw_data[:, 1].copy().reshape(8, 10, 15)
    result_3d2 = LRTC(raw_data_3d2, alpha, rho, theta, epsilon, maxiter)
    result2 = result_3d2.reshape(1200)
    filled_data = raw_data.copy()
    filled_data[:, 0] = result1
    filled_data[:, 1] = result2
    return filled_data
```

[250]:
```
#
def fill_avg(raw_data):
    result_data = raw_data.copy()
    for i in range(1200):
        if result_data[i][0] == 0:
            begin = i%144
            num = 0
            sum1 = 0
            sum2 = 0
            for j in range(begin, 1200, 168):
                if result_data[j][0] != 0:
                    num += 1
                    sum1 += result_data[j][0]
                    sum2 += result_data[j][1]
            result_data[i][0] = sum1 / num
            result_data[i][1] = sum2 / num
    return result_data
```

2.2 1.2

[20]:
```
import os
filepath_read = 'D:\\  \\MathorCup  Practice\\A  \\short_term_files'
filepath_save = 'D:\\  \\MathorCup  Practice\\data_filled'
short_term_filelst = os.listdir(filepath_read)
```

[28]:
```
#           pd.to_excel
for path in short_term_filelst:
    data = pd.read_excel(filepath_read + '\\' + path)
    dataset = data.values
    try:
        complete_data = raw2complete(dataset)
        fill_avg_data = fill_avg(complete_data)
        fill_LRTC_data = fill_LRTC(complete_data)
        fill_avg_df = pd.DataFrame(fill_avg_data)
        fill_LRTC_df = pd.DataFrame(fill_LRTC_data)
        fill_avg_df.to_excel(filepath_save + '\\avg_' + path)
        fill_LRTC_df.to_excel(filepath_save + '\\LRTC_' + path)
    except BaseException as e:
        print(path)
        print(e)
```

```
1100.xlsx
SVD did not converge
1531.xlsx
division by zero
186.xlsx
unconverted data remains:  0:00:00
3694.xlsx
```

```
division by zero
4469.xlsx
SVD did not converge
6910.xlsx
SVD did not converge
9579.xlsx
division by zero
```

2.3 1.3

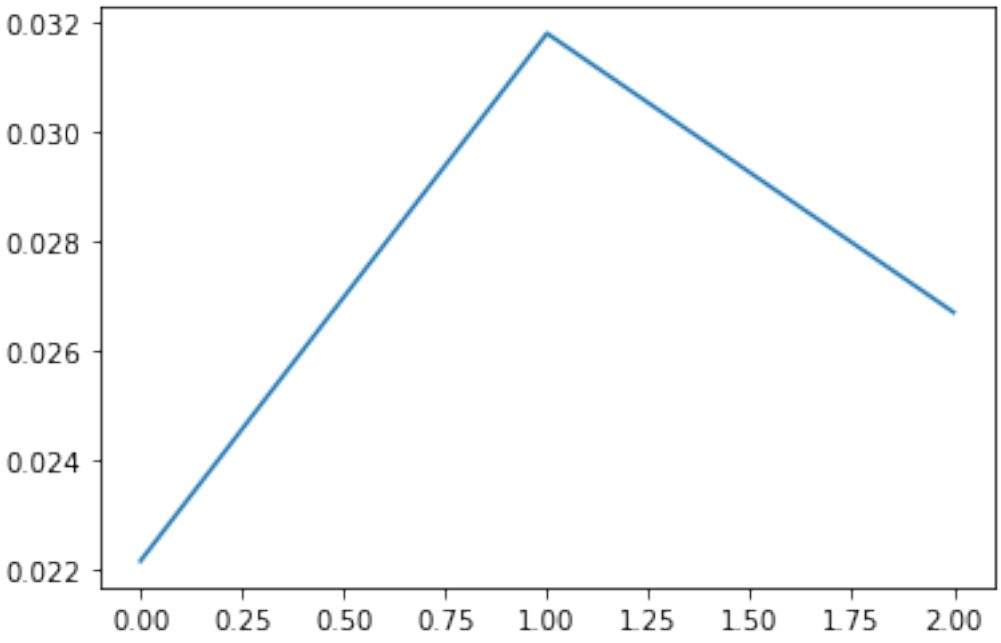
[288]:
```
test_data = complete_np_data.copy()
test_data[300: 320, 0] = np.zeros(20)
```

[289]:
```
test1 = fill_avg(test_data)
test2 = fill_LRTC(test_data)
```

[259]:
```
print(complete_np_data[500: 510, 0])
plt.plot(complete_np_data[500: 503, 0])
```

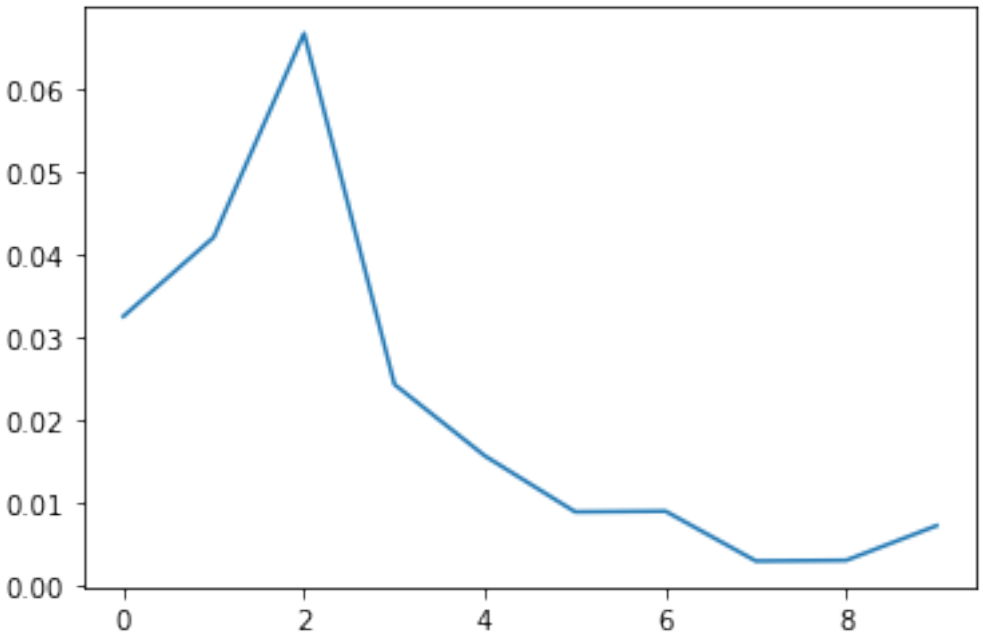
```
[0.02217068 0.03179568 0.02670708]
```

[259]:
```
[<matplotlib.lines.Line2D at 0x187ad961c70>]
```

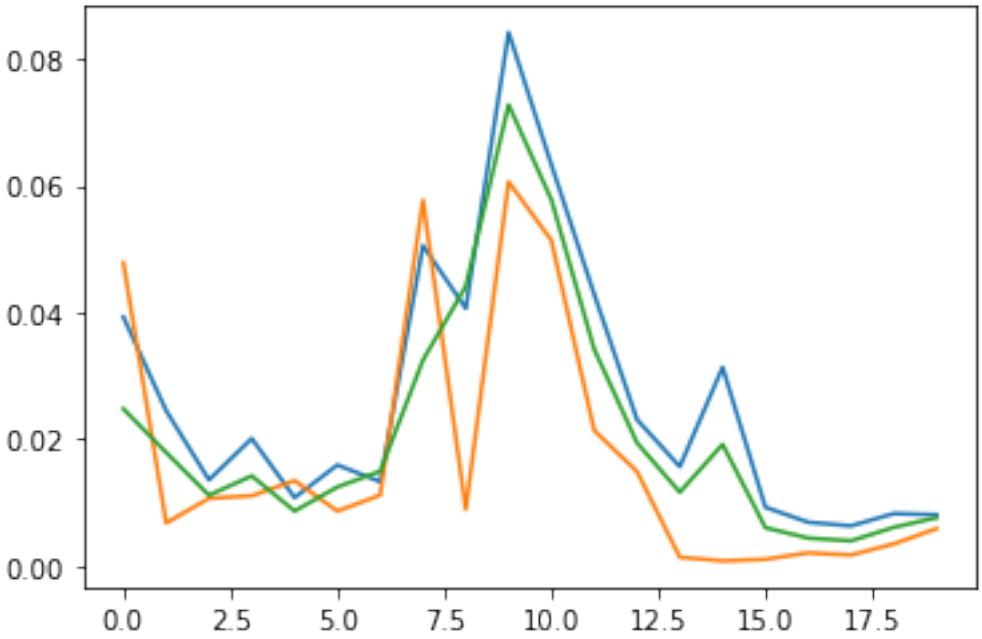


[268]:
```
print(test1[500: 510, 0])
plt.plot(test1[500: 510, 0])
print(test2[500: 510, 0])
```

```
[0.032448   0.04198968 0.06657515 0.02423831 0.01562103 0.00886477
 0.00894377 0.00293699 0.0030183  0.0071983 ]
[0.03471562 0.05029026 0.04336596 0.02423831 0.01562103 0.00886477
 0.00894377 0.00293699 0.0030183  0.00719831]
```



[297]:
```
plt.plot(test2[300: 320, 0])
plt.plot(complete_np_data[300: 320, 0])
plt.plot(test3[300: 320, 0])
test3 = (test1 + test2) / 2
```



[296]:
```
print((complete_np_data[300: 320, 0] - test1[300: 320, 0]).mean())
print((complete_np_data[300: 320, 0] - test3[300: 320, 0]).mean())
```

```
0.0010627706
-0.0041417433
```

3 2.

[206]:
```
## 2.1
    [0, 1]
```

[55]:
```
import numpy as np
```

[35]:
```
#
# succeeded
def Normalize(raw_list, low, high):
    result = raw_list.copy()
    delta = high - low
    if delta != 0:
        for i in range(len(result)):
            result[i] = (result[i] - low) / delta
    return result
def Denormalize(raw_list, low, high):
    result = raw_list.copy()
    delta = high - low
```

```
    if delta != 0:
        for i in range(len(result)):
            result[i] = result[i] * delta + low
    return result
```

3.1 2.2

[93]:
```
class lstm_reg(nn.Module):
    def __init__(self, input_size, hidden_size, output_size = 1, num_layers=2):
        super(lstm_reg, self).__init__()
        self.rnn = nn.LSTM(input_size, hidden_size, num_layers).to(device)
        self.reg = nn.Linear(hidden_size, output_size).to(device)
    def forward(self, x):
        # x
        x, _ = self.rnn(x)
        s, b ,h = x.shape
        x = x.view(s*b, h)
        x = self.reg(x)
        x = x.view(s, b, -1)
        return x[-1, :b, 0]
```

[64]:
```
#   ,numpy list
def MSE(a, b):
    a = a.cpu()
    b = b.cpu()
    sum = 0
    for x, y in zip(a, b):
        sum += pow(x-y, 2)
    return np.sqrt(sum / a.shape[0])
```

[226]:
```
# succeeded
#   data_np(1200*34)     (1200 * 8)     (1200 * 1)
#   dataset   (1200*34 ndarray) updown    updown 0     1
#      ndarray(1200 * 8)     ndarray(1200 * 1)
def generate_data_wday(dataset, updown = 0): #
    dataset_wday = dataset[:, [updown, 2, 3, 4, 5, 6, 7, 8]]
    return dataset_wday
def generate_data_nday(dataset, updown = 0): #
    dataset_nday = dataset[:, updown: updown + 1]
    return dataset_nday
```

[166]:
```
# succeeded
#                   build_model pred_model
#   dataset(ndarray,1200*8or1200*1)   train_size(int)    look_back(int)
#␣
 ↪  high(float)             low(float) trainX(ndarray,(train_size-look_back)*8or1)   trainY(nd
 ↪testX(ndarray,(test_size*8or1))   testY(ndarray,test_size*1)
```

```
def createdata(dataset, train_size, look_back):
    cp_dataset = dataset.copy()
    low = min(cp_dataset[:train_size, 0])
    high = max(cp_dataset[:train_size, 0])
    trainX = []
    trainY = []
    cp_dataset[:, 0] = Normalize(dataset[:, 0], low, high)
    for i in range(train_size - look_back):
        trainX.append(cp_dataset[i:i+look_back, :])
        trainY.append(cp_dataset[i+look_back, 0])
    testX = cp_dataset[:train_size, :]
    testY = cp_dataset[train_size:, 0]
    trainX_np = np.array(trainX).astype(np.float32)
    trainY_np = np.array(trainY).astype(np.float32)
    testX_np = np.array(testX).astype(np.float32)
    testY_np = np.array(testY).astype(np.float32)
    return high, low, trainX_np, trainY_np, testX_np, testY_np
```

[111]:
```
# succeeded
#    trainX trainY    ndarray
#    look_back,hidden_size lstm_layer  batch_size    device
#       net
def build_model(trainX, trainY, look_back, hidden_size, lstm_layer):
    trainX = trainX.transpose(1, 0, 2)
    trainX = torch.from_numpy(trainX).to(device)
    trainY = torch.from_numpy(trainY).to(device)
    input_size = trainX.shape[2]
    net = lstm_reg(input_size, hidden_size, lstm_layer)
    optimizer = torch.optim.Adam(net.parameters(), lr = 1e-2)
    criterion = nn.MSELoss()
    for epoch in range(1000):
        for batch in range(trainX.shape[1]//batch_size):
            var_x = Variable(trainX[:, batch*batch_size:(batch+1)*batch_size, :
 ↪])
            var_y = Variable(trainY[batch*batch_size:(batch+1)*batch_size])
            out = net(var_x)
            loss = criterion(out, var_y)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            print("epoch:{}, batch{}, loss:{}".format(epoch, batch, loss))
    return net
```

[154]:
```
# succeeded
#                     0
#   have_data(ndarray,have_size*8or1)   pred_days(int)
#   draft_data(ndarray,   *8or1)
```

```
def expand_data(have_data, pred_days):
    if have_data.shape[1] == 1:
        draft_data = np.vstack((have_data.astype(np.float32), np.
↪zeros((pred_days, 1), dtype = np.float32)))
    else:
        starting_stamp = str2stamp('2018/3/1 0:00:00')
        draft_data = np.vstack((have_data.astype(np.float32), np.
↪zeros((pred_days, 8), dtype = np.float32)))
        have_days = have_data.shape[0]
        for i in range(pred_days):
            present_stamp = starting_stamp + 3600 * (i + have_days)
            present_datetime = datetime.datetime.fromtimestamp(present_stamp)
            present_tuple = present_datetime.timetuple()
            wday = present_tuple.tm_wday
            draft_data[i + have_days, 1 + wday] = 1
        return draft_data
```

[203]:
```
# some problems
#       net    have_data(ndarray,train_size * 8or1)   look_back      pred_days
def pred_model(net, have_data, look_back, pred_days):
    draft_data = expand_data(have_data, pred_days)
    draft_data_transposed_tsr = torch.from_numpy(draft_data.reshape(draft_data.
↪shape[0], 1, draft_data.shape[1])).to(device)
    have_days = have_data.shape[0]
    previous = draft_data_transposed_tsr[have_days - look_back: have_days, :, :]
    pred_result = []
    for i in range(pred_days):
        pred_value = net(previous)
        draft_data_transposed_tsr[have_days + i, 0, 0] = pred_value
        pred_result.append(pred_value.item())
        previous = draft_data_transposed_tsr[have_days - look_back + i:␣
↪have_days + i, :, :]
    return np.array(pred_result)
```

[207]:
```
## 2.3
```

3.1.1 2.3.1

[262]:
```
import pandas as pd
filepath = "D:\\  \\MathorCup  Practice\\data_filled\\LRTC_221.xlsx"
data = pd.read_excel(filepath)
```

[263]:
```
dataset = data.values[:, 1:]
```

3.1.2 2.3.2

[264]:
```
data_wday = generate_data_wday(dataset)
data_nday = generate_data_nday(dataset)
```

[276]:
```
high, low, trainX, trainY, testX, testY= createdata(data_wday, train_size,␣
↪look_back)
```

3.1.3 2.3.3

[270]:
```
#
device = torch.device("cuda")
look_back = 72
hidden_size = 144
lstm_layer = 1
train_size = int(1200 * 0.8)
test_size = 1200 - train_size
batch_size = 128 # batchsize
```

[272]:
```
net = build_model(trainX, trainY, look_back, 144, 8)
result = pred_model(net, testX, look_back, 168)
loss = MSE(result, testY[:168])
```

```
4010422
epoch:928, batch0, loss:0.00010672054486349225
epoch:928, batch1, loss:0.0001643884024815634
epoch:928, batch2, loss:0.00016461724590044469
epoch:928, batch3, loss:0.00012935981794726104
epoch:928, batch4, loss:8.055505168158561e-05
epoch:928, batch5, loss:0.00016180460806936026
epoch:929, batch0, loss:0.00014215079136192799
epoch:929, batch1, loss:0.00012609972327481955
epoch:929, batch2, loss:8.268481178674847e-05
epoch:929, batch3, loss:8.550500206183642e-05
epoch:929, batch4, loss:0.00012712544412352145
epoch:929, batch5, loss:0.00020516957738436759
epoch:930, batch0, loss:8.711195550858974e-05
epoch:930, batch1, loss:7.421498594339937e-05
epoch:930, batch2, loss:9.411548671778291e-05
epoch:930, batch3, loss:0.00011966171587118879
epoch:930, batch4, loss:0.00012043431343045086
epoch:930, batch5, loss:9.489156946074218e-05
epoch:931, batch0, loss:6.1022627050988376e-05
epoch:931, batch1, loss:9.9234348454047e-05
epoch:931, batch2, loss:0.00011175590771017596
epoch:931, batch3, loss:0.00010960893996525556
epoch:931, batch4, loss:9.164452785626054e-05
epoch:931, batch5, loss:8.08130789664574e-05
```

epoch:932, batch0, loss:9.237607446266338e-05
epoch:932, batch1, loss:9.679422510089353e-05
epoch:932, batch2, loss:8.136741234920919e-05
epoch:932, batch3, loss:6.470651715062559e-05
epoch:932, batch4, loss:7.387044752249494e-05
epoch:932, batch5, loss:0.00010299873247276992
epoch:933, batch0, loss:7.56712252041325e-05
epoch:933, batch1, loss:9.158915054285899e-05
epoch:933, batch2, loss:6.371510244207457e-05
epoch:933, batch3, loss:4.798948430106975e-05
epoch:933, batch4, loss:5.705660078092478e-05
epoch:933, batch5, loss:7.944587559904903e-05
epoch:934, batch0, loss:5.153024903847836e-05
epoch:934, batch1, loss:7.328249921556562e-05
epoch:934, batch2, loss:5.3732554079033434e-05
epoch:934, batch3, loss:7.805493078194559e-05
epoch:934, batch4, loss:7.377400470431894e-05
epoch:934, batch5, loss:7.451882993336767e-05
epoch:935, batch0, loss:4.683378210756928e-05
epoch:935, batch1, loss:3.176018799422309e-05
epoch:935, batch2, loss:6.115868018241599e-05
epoch:935, batch3, loss:6.78558717481792e-05
epoch:935, batch4, loss:6.938057777006179e-05
epoch:935, batch5, loss:5.527965913643129e-05
epoch:936, batch0, loss:3.456111153354868e-05
epoch:936, batch1, loss:5.2361014240887016e-05
epoch:936, batch2, loss:6.52879971312359e-05
epoch:936, batch3, loss:4.88490259158425e-05
epoch:936, batch4, loss:3.606161044444889e-05
epoch:936, batch5, loss:3.935991117032245e-05
epoch:937, batch0, loss:2.886865877371747e-05
epoch:937, batch1, loss:5.846202475368045e-05
epoch:937, batch2, loss:4.455087037058547e-05
epoch:937, batch3, loss:3.206895780749619e-05
epoch:937, batch4, loss:2.758165283012204e-05
epoch:937, batch5, loss:4.0513445128453895e-05
epoch:938, batch0, loss:4.0389582864008844e-05
epoch:938, batch1, loss:2.743626100709662e-05
epoch:938, batch2, loss:3.059801019844599e-05
epoch:938, batch3, loss:1.9507302567944862e-05
epoch:938, batch4, loss:2.717328789003659e-05
epoch:938, batch5, loss:4.487682963372208e-05
epoch:939, batch0, loss:3.5319128073751926e-05
epoch:939, batch1, loss:1.4992707292549312e-05
epoch:939, batch2, loss:2.428948209853843e-05
epoch:939, batch3, loss:3.477624341030605e-05
epoch:939, batch4, loss:3.471592935966328e-05
epoch:939, batch5, loss:3.395853127585724e-05

epoch:940, batch0, loss:1.6696827515261248e-05
epoch:940, batch1, loss:2.30308469326701e-05
epoch:940, batch2, loss:3.4108044928871095e-05
epoch:940, batch3, loss:3.504330379655585e-05
epoch:940, batch4, loss:2.0275718270568177e-05
epoch:940, batch5, loss:1.7565178495715372e-05
epoch:941, batch0, loss:2.909617978730239e-05
epoch:941, batch1, loss:3.3831456676125526e-05
epoch:941, batch2, loss:3.27529851347208e-05
epoch:941, batch3, loss:2.67730065388605e-05
epoch:941, batch4, loss:2.4224191292887554e-05
epoch:941, batch5, loss:2.88646733679343e-05
epoch:942, batch0, loss:4.7042911319294944e-05
epoch:942, batch1, loss:3.431521327001974e-05
epoch:942, batch2, loss:3.554057911969721e-05
epoch:942, batch3, loss:3.154534715577029e-05
epoch:942, batch4, loss:3.657543129520491e-05
epoch:942, batch5, loss:3.955971624236554e-05
epoch:943, batch0, loss:3.757210652111098e-05
epoch:943, batch1, loss:3.9134076359914616e-05
epoch:943, batch2, loss:3.384631418157369e-05
epoch:943, batch3, loss:2.3362794308923185e-05
epoch:943, batch4, loss:3.4990080166608095e-05
epoch:943, batch5, loss:4.521304072113708e-05
epoch:944, batch0, loss:3.550486144376919e-05
epoch:944, batch1, loss:3.2175317755900323e-05
epoch:944, batch2, loss:3.540558464010246e-05
epoch:944, batch3, loss:4.2826788558159024e-05
epoch:944, batch4, loss:4.709426866611466e-05
epoch:944, batch5, loss:2.551840225351043e-05
epoch:945, batch0, loss:2.899364335462451e-05
epoch:945, batch1, loss:4.319813888287172e-05
epoch:945, batch2, loss:5.686294025508687e-05
epoch:945, batch3, loss:3.7223377148620784e-05
epoch:945, batch4, loss:2.3370621420326643e-05
epoch:945, batch5, loss:2.9054892365820706e-05
epoch:946, batch0, loss:5.1832110329996794e-05
epoch:946, batch1, loss:4.7671990614617243e-05
epoch:946, batch2, loss:2.86239264823962e-05
epoch:946, batch3, loss:1.8991733668372035e-05
epoch:946, batch4, loss:4.233834988554008e-05
epoch:946, batch5, loss:6.816369568696246e-05
epoch:947, batch0, loss:4.7692941734567285e-05
epoch:947, batch1, loss:2.4381442926824093e-05
epoch:947, batch2, loss:4.1818973841145635e-05
epoch:947, batch3, loss:5.47693925909698e-05
epoch:947, batch4, loss:6.40190119156614e-05
epoch:947, batch5, loss:3.8218131521716714e-05

epoch:948, batch0, loss:2.93974135274766e-05
epoch:948, batch1, loss:4.9281887186225504e-05
epoch:948, batch2, loss:6.973019480938092e-05
epoch:948, batch3, loss:3.436983024585061e-05
epoch:948, batch4, loss:2.8430424208636396e-05
epoch:948, batch5, loss:3.491532334010117e-05
epoch:949, batch0, loss:5.789143324363977e-05
epoch:949, batch1, loss:5.1175600674469024e-05
epoch:949, batch2, loss:3.2017851481214166e-05
epoch:949, batch3, loss:3.434783502598293e-05
epoch:949, batch4, loss:5.30692923348397e-05
epoch:949, batch5, loss:6.565771764144301e-05
epoch:950, batch0, loss:4.449849802767858e-05
epoch:950, batch1, loss:3.669424768304452e-05
epoch:950, batch2, loss:5.470876203617081e-05
epoch:950, batch3, loss:7.475980237359181e-05
epoch:950, batch4, loss:6.332200428005308e-05
epoch:950, batch5, loss:4.712925147032365e-05
epoch:951, batch0, loss:4.5561057049781084e-05
epoch:951, batch1, loss:6.373669020831585e-05
epoch:951, batch2, loss:7.399742025882006e-05
epoch:951, batch3, loss:3.793747600866482e-05
epoch:951, batch4, loss:3.865562393912114e-05
epoch:951, batch5, loss:5.69979238207452e-05
epoch:952, batch0, loss:6.071134703233838e-05
epoch:952, batch1, loss:5.334667002898641e-05
epoch:952, batch2, loss:4.0500824979972094e-05
epoch:952, batch3, loss:3.0239294574130327e-05
epoch:952, batch4, loss:6.112885603215545e-05
epoch:952, batch5, loss:8.564443123759702e-05
epoch:953, batch0, loss:3.269598528277129e-05
epoch:953, batch1, loss:3.356734669068828e-05
epoch:953, batch2, loss:5.49477081221994e-05
epoch:953, batch3, loss:7.460094639100134e-05
epoch:953, batch4, loss:6.628326082136482e-05
epoch:953, batch5, loss:5.608367791865021e-05
epoch:954, batch0, loss:4.5814012992195785e-05
epoch:954, batch1, loss:7.191274198703468e-05
epoch:954, batch2, loss:8.421987877227366e-05
epoch:954, batch3, loss:5.422732647275552e-05
epoch:954, batch4, loss:4.262708534952253e-05
epoch:954, batch5, loss:6.451500667026266e-05
epoch:955, batch0, loss:6.15566605119966e-05
epoch:955, batch1, loss:6.6612075897865e-05
epoch:955, batch2, loss:3.440943328314461e-05
epoch:955, batch3, loss:2.8000697056995705e-05
epoch:955, batch4, loss:6.410156493075192e-05
epoch:955, batch5, loss:9.179846529150382e-05

epoch:956, batch0, loss:2.840597699105274e-05
epoch:956, batch1, loss:4.3730862671509385e-05
epoch:956, batch2, loss:6.263922114158049e-05
epoch:956, batch3, loss:6.034232865204103e-05
epoch:956, batch4, loss:5.5483462347183377e-05
epoch:956, batch5, loss:4.8915233492152765e-05
epoch:957, batch0, loss:4.04358834202867e-05
epoch:957, batch1, loss:8.642725879326463e-05
epoch:957, batch2, loss:7.452542922692373e-05
epoch:957, batch3, loss:5.083544601802714e-05
epoch:957, batch4, loss:7.43484270060435e-05
epoch:957, batch5, loss:8.766568498685956e-05
epoch:958, batch0, loss:5.196586425881833e-05
epoch:958, batch1, loss:4.5924356527393684e-05
epoch:958, batch2, loss:5.0732116505969316e-05
epoch:958, batch3, loss:5.401071393862367e-05
epoch:958, batch4, loss:8.434787741862237e-05
epoch:958, batch5, loss:6.389230111381039e-05
epoch:959, batch0, loss:5.057618182036094e-05
epoch:959, batch1, loss:9.757316729519516e-05
epoch:959, batch2, loss:0.00010344934707973152
epoch:959, batch3, loss:5.963875446468592e-05
epoch:959, batch4, loss:6.877875421196222e-05
epoch:959, batch5, loss:7.807136717019603e-05
epoch:960, batch0, loss:7.687801553402096e-05
epoch:960, batch1, loss:8.359731873497367e-05
epoch:960, batch2, loss:8.657002763357013e-05
epoch:960, batch3, loss:8.906878065317869e-05
epoch:960, batch4, loss:0.00011315348092466593
epoch:960, batch5, loss:7.459228072548285e-05
epoch:961, batch0, loss:6.27976332907565e-05
epoch:961, batch1, loss:8.950493793236092e-05
epoch:961, batch2, loss:0.00013547799608204514
epoch:961, batch3, loss:5.1672057452378795e-05
epoch:961, batch4, loss:4.933359014103189e-05
epoch:961, batch5, loss:0.00010585829295450822
epoch:962, batch0, loss:0.00014389902935363352
epoch:962, batch1, loss:9.7560667200014e-05
epoch:962, batch2, loss:6.832309009041637e-05
epoch:962, batch3, loss:0.0001023068034555763
epoch:962, batch4, loss:0.00015342948609031737
epoch:962, batch5, loss:0.00012706230336334556
epoch:963, batch0, loss:6.456012488342822e-05
epoch:963, batch1, loss:7.34533168724738e-05
epoch:963, batch2, loss:0.00022611106396652758
epoch:963, batch3, loss:0.0001548461732454598
epoch:963, batch4, loss:5.4665812058374286e-05
epoch:963, batch5, loss:6.408672197721899e-05

```
epoch:964, batch0, loss:0.00015873767551966012
epoch:964, batch1, loss:0.00017044765991158783
epoch:964, batch2, loss:0.00011319164332235232
epoch:964, batch3, loss:5.6966782722156495e-05
epoch:964, batch4, loss:0.00015623918443452567
epoch:964, batch5, loss:0.00021771302272100002
epoch:965, batch0, loss:0.0001743314933264628
epoch:965, batch1, loss:9.038975258590654e-05
epoch:965, batch2, loss:0.00016264728037640452
epoch:965, batch3, loss:0.00018938243738375604
epoch:965, batch4, loss:0.0001656079839449376
epoch:965, batch5, loss:8.331173012265936e-05
epoch:966, batch0, loss:9.392505307914689e-05
epoch:966, batch1, loss:0.0002353246381971985
epoch:966, batch2, loss:0.00026552792405709624
epoch:966, batch3, loss:0.00013332447269931436
epoch:966, batch4, loss:0.00011328917025821283
epoch:966, batch5, loss:0.0002524694427847862
epoch:967, batch0, loss:0.00022824251209385693
epoch:967, batch1, loss:0.00012738248915411532
epoch:967, batch2, loss:8.591049117967486e-05
epoch:967, batch3, loss:0.0001197836099890992
epoch:967, batch4, loss:0.00028373394161462784
epoch:967, batch5, loss:0.00018003243894781917
epoch:968, batch0, loss:0.00011773586447816342
epoch:968, batch1, loss:0.00017528916941955686
epoch:968, batch2, loss:0.00029183411970734596
epoch:968, batch3, loss:0.00018389718024991453
epoch:968, batch4, loss:0.00012766325380653143
epoch:968, batch5, loss:0.00017043144907802343
epoch:969, batch0, loss:0.00013441279588732868
epoch:969, batch1, loss:0.00021682860096916556
epoch:969, batch2, loss:0.00016804676852189004
epoch:969, batch3, loss:0.00014437120989896357
epoch:969, batch4, loss:0.000231990561587736
epoch:969, batch5, loss:0.00025081619969569147
epoch:970, batch0, loss:0.00015852331125643104
epoch:970, batch1, loss:0.00015674243331886828
epoch:970, batch2, loss:0.00026031117886304855
epoch:970, batch3, loss:0.00014920413377694786
epoch:970, batch4, loss:0.00014220498269423842
epoch:970, batch5, loss:0.00017650987138040364
epoch:971, batch0, loss:0.00019566481932997704
epoch:971, batch1, loss:0.000232641115359962
epoch:971, batch2, loss:0.00017028331058099866
epoch:971, batch3, loss:0.00011115844245068729
epoch:971, batch4, loss:0.00023849884746596217
epoch:971, batch5, loss:0.0003764590946957469
```

```
epoch:972, batch0, loss:0.00017370560090057552
epoch:972, batch1, loss:9.397895337315276e-05
epoch:972, batch2, loss:0.00022672642080578953
epoch:972, batch3, loss:0.0002495926746632904
epoch:972, batch4, loss:0.00028680189279839396
epoch:972, batch5, loss:0.00012676219921559095
epoch:973, batch0, loss:0.00013863565982319415
epoch:973, batch1, loss:0.00030023889848962426
epoch:973, batch2, loss:0.0005042387638241053
epoch:973, batch3, loss:0.00025566251133568585
epoch:973, batch4, loss:0.0001411920238751918
epoch:973, batch5, loss:0.0002694698632694781
epoch:974, batch0, loss:0.00029103195993229747
epoch:974, batch1, loss:0.0003566884552128613
epoch:974, batch2, loss:0.0001784183259587735
epoch:974, batch3, loss:0.00014966356684453785
epoch:974, batch4, loss:0.00028516037855297327
epoch:974, batch5, loss:0.0004820318426936865
epoch:975, batch0, loss:0.0002851342724170536
epoch:975, batch1, loss:0.0003186517860740423
epoch:975, batch2, loss:0.00025604391703382134
epoch:975, batch3, loss:0.00022986972180660814
epoch:975, batch4, loss:0.0003413638041820377
epoch:975, batch5, loss:0.0002563579473644495
epoch:976, batch0, loss:0.00024224439403042197
epoch:976, batch1, loss:0.000345385808032006
epoch:976, batch2, loss:0.00029617027030326426
epoch:976, batch3, loss:0.00021584247588180006
epoch:976, batch4, loss:0.00038186582969501615
epoch:976, batch5, loss:0.00038291868986561894
epoch:977, batch0, loss:0.0003281372191850096
epoch:977, batch1, loss:0.0001929534482769668
epoch:977, batch2, loss:0.0002326852991245687
epoch:977, batch3, loss:0.00027096248231828213
epoch:977, batch4, loss:0.0003247035201638937
epoch:977, batch5, loss:0.00032814720179885626
epoch:978, batch0, loss:0.00014355138409882784
epoch:978, batch1, loss:0.00023390517162624747
epoch:978, batch2, loss:0.000332401308696717
epoch:978, batch3, loss:0.00038374075666069984
epoch:978, batch4, loss:0.00020351244893390685
epoch:978, batch5, loss:0.00022786542831454426
epoch:979, batch0, loss:0.0002978882985189557
epoch:979, batch1, loss:0.0003142638597637415
epoch:979, batch2, loss:0.00032328898669220507
epoch:979, batch3, loss:0.00022253862698562443
epoch:979, batch4, loss:0.0003075345593970269
epoch:979, batch5, loss:0.00036513619124889374
```

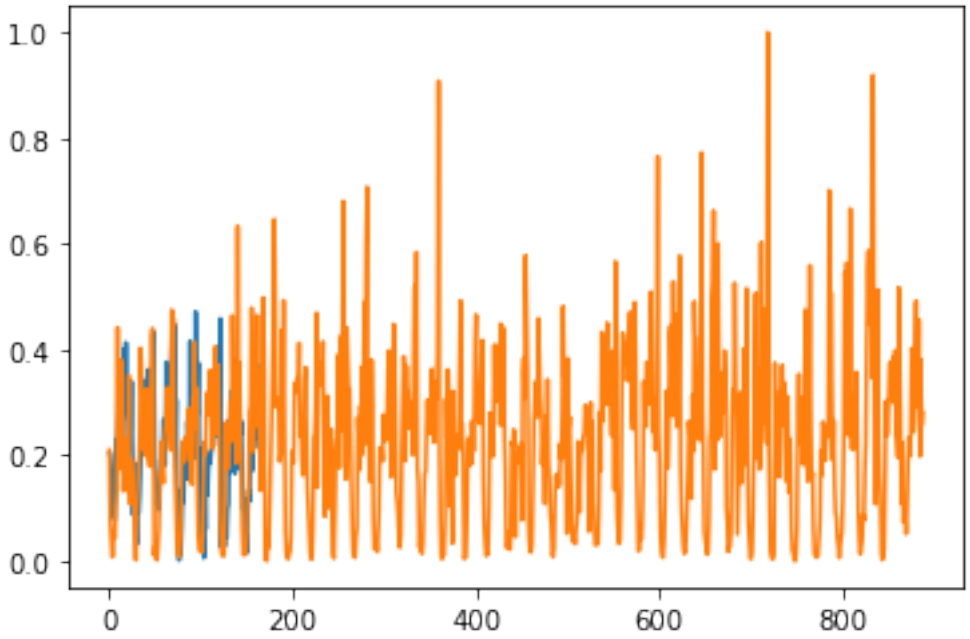
epoch:980, batch0, loss:0.0003111230325885117
epoch:980, batch1, loss:0.00025544670643284917
epoch:980, batch2, loss:0.0003455911064520478
epoch:980, batch3, loss:0.00033115700352936983
epoch:980, batch4, loss:0.0002298594918102026
epoch:980, batch5, loss:0.00037794720265083015
epoch:981, batch0, loss:0.00027002429123967886
epoch:981, batch1, loss:0.00024270749418064952
epoch:981, batch2, loss:0.00019699115364346653
epoch:981, batch3, loss:0.00017851198208518326
epoch:981, batch4, loss:0.0002495469816494733
epoch:981, batch5, loss:0.00030802719993516803
epoch:982, batch0, loss:0.00019550666911527514
epoch:982, batch1, loss:0.0002059076796285808
epoch:982, batch2, loss:0.00017478238441981375
epoch:982, batch3, loss:0.000243629214529097
epoch:982, batch4, loss:0.00028017748263664544
epoch:982, batch5, loss:0.0002217449073214084
epoch:983, batch0, loss:0.0001577716029714793
epoch:983, batch1, loss:0.00016838248120620847
epoch:983, batch2, loss:0.00020805682288482785
epoch:983, batch3, loss:0.0002897219965234399
epoch:983, batch4, loss:0.0001901575888041407
epoch:983, batch5, loss:0.00015308684669435024
epoch:984, batch0, loss:0.00020801450591534376
epoch:984, batch1, loss:0.0002205823693657294
epoch:984, batch2, loss:0.000200379901798442
epoch:984, batch3, loss:0.00011313086724840105
epoch:984, batch4, loss:9.12742834771052e-05
epoch:984, batch5, loss:0.00011768846889026463
epoch:985, batch0, loss:0.00018558579904492944
epoch:985, batch1, loss:0.0001619848480913788
epoch:985, batch2, loss:0.00011792185250669718
epoch:985, batch3, loss:9.81862613116391e-05
epoch:985, batch4, loss:9.33007977437228e-05
epoch:985, batch5, loss:0.0001329699152847752
epoch:986, batch0, loss:9.40418685786426e-05
epoch:986, batch1, loss:0.00012972086551599205
epoch:986, batch2, loss:7.958745845826343e-05
epoch:986, batch3, loss:9.314622002420947e-05
epoch:986, batch4, loss:9.87919484032318e-05
epoch:986, batch5, loss:0.0001089621800929308
epoch:987, batch0, loss:6.557010783581063e-05
epoch:987, batch1, loss:8.3009188529104e-05
epoch:987, batch2, loss:0.00012763946142513305
epoch:987, batch3, loss:7.731839286861941e-05
epoch:987, batch4, loss:7.475406164303422e-05
epoch:987, batch5, loss:4.39528412243817e-05

epoch:988, batch0, loss:7.02140896464698e-05
epoch:988, batch1, loss:0.0001276529219467193
epoch:988, batch2, loss:0.0001412837882526219
epoch:988, batch3, loss:5.891357432119548e-05
epoch:988, batch4, loss:6.794263026677072e-05
epoch:988, batch5, loss:0.00014951752382330596
epoch:989, batch0, loss:0.00010581834067124873
epoch:989, batch1, loss:7.937695772852749e-05
epoch:989, batch2, loss:5.171890734345652e-05
epoch:989, batch3, loss:7.673882646486163e-05
epoch:989, batch4, loss:0.00015408860053867102
epoch:989, batch5, loss:0.00012714295007754117
epoch:990, batch0, loss:5.1434297347441316e-05
epoch:990, batch1, loss:9.764278365764767e-05
epoch:990, batch2, loss:0.0001648454781388864
epoch:990, batch3, loss:0.00011412054300308228
epoch:990, batch4, loss:7.958489732118323e-05
epoch:990, batch5, loss:9.010091162053868e-05
epoch:991, batch0, loss:0.00011745143274310976
epoch:991, batch1, loss:0.0001228843757417053
epoch:991, batch2, loss:9.151217818725854e-05
epoch:991, batch3, loss:7.708747580181807e-05
epoch:991, batch4, loss:0.00013148611469659954
epoch:991, batch5, loss:0.0001510216825408861
epoch:992, batch0, loss:7.54864449845627e-05
epoch:992, batch1, loss:7.592904148623347e-05
epoch:992, batch2, loss:0.00010947763803415
epoch:992, batch3, loss:9.286263957619667e-05
epoch:992, batch4, loss:8.578911365475506e-05
epoch:992, batch5, loss:7.095483306329697e-05
epoch:993, batch0, loss:8.544181764591485e-05
epoch:993, batch1, loss:0.00012311540194787085
epoch:993, batch2, loss:0.00010192936315434054
epoch:993, batch3, loss:7.375993300229311e-05
epoch:993, batch4, loss:7.536551856901497e-05
epoch:993, batch5, loss:9.867520566331223e-05
epoch:994, batch0, loss:8.819514914648607e-05
epoch:994, batch1, loss:5.6960114307003096e-05
epoch:994, batch2, loss:6.596792081836611e-05
epoch:994, batch3, loss:8.265757787739858e-05
epoch:994, batch4, loss:8.730236004339531e-05
epoch:994, batch5, loss:4.5955388486618176e-05
epoch:995, batch0, loss:4.0968097891891375e-05
epoch:995, batch1, loss:7.10904787410982e-05
epoch:995, batch2, loss:8.18783591967076e-05
epoch:995, batch3, loss:4.690484638558701e-05
epoch:995, batch4, loss:2.630346170917619e-05
epoch:995, batch5, loss:6.19890633970499e-05

```
epoch:996, batch0, loss:9.155790030490607e-05
epoch:996, batch1, loss:5.868749212822877e-05
epoch:996, batch2, loss:3.558291064109653e-05
epoch:996, batch3, loss:5.439465894596651e-05
epoch:996, batch4, loss:7.984208059497178e-05
epoch:996, batch5, loss:6.955257413210347e-05
epoch:997, batch0, loss:3.837477197521366e-05
epoch:997, batch1, loss:3.77759788534604e-05
epoch:997, batch2, loss:8.627895294921473e-05
epoch:997, batch3, loss:8.200839511118829e-05
epoch:997, batch4, loss:4.402982449391857e-05
epoch:997, batch5, loss:2.5357781851198524e-05
epoch:998, batch0, loss:6.110468530096114e-05
epoch:998, batch1, loss:8.289240940939635e-05
epoch:998, batch2, loss:4.823099516215734e-05
epoch:998, batch3, loss:1.7988895706366748e-05
epoch:998, batch4, loss:4.433510184753686e-05
epoch:998, batch5, loss:8.991351933218539e-05
epoch:999, batch0, loss:7.927288970677182e-05
epoch:999, batch1, loss:2.9454808100126684e-05
epoch:999, batch2, loss:4.567598080029711e-05
epoch:999, batch3, loss:6.257486529648304e-05
epoch:999, batch4, loss:7.56627632654272e-05
epoch:999, batch5, loss:4.767045902553946e-05
```

[282]:
```
plt.plot(result, )
plt.plot(trainY)
print(MSE(result, testY))
```

```
0.20735464787649574
```



[205]:
```
losses = []
results = []
for layer in range(2, 20, 2):
    net = build_model(trainX, trainY, look_back, 144, layer)
    result = pred_model(net, testX, look_back, 168)
    loss = MSE(result, testY[:168])
    results.append(result)
    losses.append(loss)
```

```
809045374e-05
epoch:928, batch0, loss:4.604194691637531e-05
epoch:928, batch1, loss:7.655860099475831e-05
epoch:928, batch2, loss:6.359111284837127e-05
epoch:928, batch3, loss:3.794200165430084e-05
epoch:928, batch4, loss:2.843969559762627e-05
epoch:928, batch5, loss:2.9950999305583537e-05
epoch:929, batch0, loss:5.782489824923687e-05
epoch:929, batch1, loss:7.507071131840348e-05
epoch:929, batch2, loss:4.629127579391934e-05
epoch:929, batch3, loss:2.7390227842261083e-05
epoch:929, batch4, loss:3.5516670322977006e-05
epoch:929, batch5, loss:3.642477167886682e-05
epoch:930, batch0, loss:2.891042095143348e-05
epoch:930, batch1, loss:3.6741330404765904e-05
epoch:930, batch2, loss:2.6812755095306784e-05
epoch:930, batch3, loss:2.546733958297409e-05
```

epoch:930, batch4, loss:4.086327680852264e-05
epoch:930, batch5, loss:4.934666139888577e-05
epoch:931, batch0, loss:2.5565985197317787e-05
epoch:931, batch1, loss:2.375258736719843e-05
epoch:931, batch2, loss:2.4267912522191182e-05
epoch:931, batch3, loss:2.9164415536797605e-05
epoch:931, batch4, loss:4.004260335932486e-05
epoch:931, batch5, loss:4.825223368243314e-05
epoch:932, batch0, loss:3.257700154790655e-05
epoch:932, batch1, loss:2.8699862014036626e-05
epoch:932, batch2, loss:3.6746358091477305e-05
epoch:932, batch3, loss:4.033712320961058e-05
epoch:932, batch4, loss:4.069307760801166e-05
epoch:932, batch5, loss:3.220983489882201e-05
epoch:933, batch0, loss:2.369141839153599e-05
epoch:933, batch1, loss:4.509984137257561e-05
epoch:933, batch2, loss:4.8292211431544274e-05
epoch:933, batch3, loss:3.6232224374543875e-05
epoch:933, batch4, loss:2.3540977053926326e-05
epoch:933, batch5, loss:2.313139702891931e-05
epoch:934, batch0, loss:2.745773599599488e-05
epoch:934, batch1, loss:5.0005430239252746e-05
epoch:934, batch2, loss:3.0773804610362276e-05
epoch:934, batch3, loss:1.4050367099116556e-05
epoch:934, batch4, loss:2.6044373953482136e-05
epoch:934, batch5, loss:5.1152983360225335e-05
epoch:935, batch0, loss:2.7071702788816765e-05
epoch:935, batch1, loss:2.49831318797078e-05
epoch:935, batch2, loss:1.5871999494265765e-05
epoch:935, batch3, loss:2.0746349036926404e-05
epoch:935, batch4, loss:4.200436524115503e-05
epoch:935, batch5, loss:4.7562123654643074e-05
epoch:936, batch0, loss:1.4504837963613681e-05
epoch:936, batch1, loss:2.1511048544198275e-05
epoch:936, batch2, loss:4.093475217814557e-05
epoch:936, batch3, loss:3.864306927425787e-05
epoch:936, batch4, loss:2.5609046133467928e-05
epoch:936, batch5, loss:1.6649595636408776e-05
epoch:937, batch0, loss:2.1951580492896028e-05
epoch:937, batch1, loss:4.1961215174524114e-05
epoch:937, batch2, loss:4.39206269220449e-05
epoch:937, batch3, loss:2.5028220989042893e-05
epoch:937, batch4, loss:2.01458915398689e-05
epoch:937, batch5, loss:4.163160338066518e-05
epoch:938, batch0, loss:3.122998896287754e-05
epoch:938, batch1, loss:3.295909846201539e-05
epoch:938, batch2, loss:2.386878986726515e-05
epoch:938, batch3, loss:2.6651014195522293e-05

epoch:938, batch4, loss:4.554423503577709e-05
epoch:938, batch5, loss:5.867162690265104e-05
epoch:939, batch0, loss:2.6713896659202874e-05
epoch:939, batch1, loss:3.632369407569058e-05
epoch:939, batch2, loss:4.081197039340623e-05
epoch:939, batch3, loss:3.819158155238256e-05
epoch:939, batch4, loss:4.2093302909052e-05
epoch:939, batch5, loss:3.980487599619664e-05
epoch:940, batch0, loss:3.551470581442118e-05
epoch:940, batch1, loss:5.425595736596733e-05
epoch:940, batch2, loss:5.1232451369287446e-05
epoch:940, batch3, loss:3.032911081390921e-05
epoch:940, batch4, loss:4.136804636800662e-05
epoch:940, batch5, loss:5.0467089749872684e-05
epoch:941, batch0, loss:3.163770816172473e-05
epoch:941, batch1, loss:3.257499702158384e-05
epoch:941, batch2, loss:3.738707164302468e-05
epoch:941, batch3, loss:5.642774704028852e-05
epoch:941, batch4, loss:5.889893145649694e-05
epoch:941, batch5, loss:3.871206354233436e-05
epoch:942, batch0, loss:2.436717477394268e-05
epoch:942, batch1, loss:4.54256878583692e-05
epoch:942, batch2, loss:5.2943236369173974e-05
epoch:942, batch3, loss:6.649861461482942e-05
epoch:942, batch4, loss:2.633153781061992e-05
epoch:942, batch5, loss:2.7985153792542405e-05
epoch:943, batch0, loss:7.70453698351048e-05
epoch:943, batch1, loss:7.976210326887667e-05
epoch:943, batch2, loss:4.2562482121866196e-05
epoch:943, batch3, loss:1.971820165636018e-05
epoch:943, batch4, loss:4.6279259549919516e-05
epoch:943, batch5, loss:8.227416401496157e-05
epoch:944, batch0, loss:6.895136903040111e-05
epoch:944, batch1, loss:4.617086233338341e-05
epoch:944, batch2, loss:2.4953627871582285e-05
epoch:944, batch3, loss:4.60867777292151e-05
epoch:944, batch4, loss:9.575278090778738e-05
epoch:944, batch5, loss:7.611959154019132e-05
epoch:945, batch0, loss:1.9015125872101635e-05
epoch:945, batch1, loss:3.947894219891168e-05
epoch:945, batch2, loss:6.887355993967503e-05
epoch:945, batch3, loss:8.47315532155335e-05
epoch:945, batch4, loss:5.240239261183888e-05
epoch:945, batch5, loss:3.553114947862923e-05
epoch:946, batch0, loss:7.728519267402589e-05
epoch:946, batch1, loss:7.723829185124487e-05
epoch:946, batch2, loss:6.821571878390387e-05
epoch:946, batch3, loss:4.9627160478848964e-05

epoch:946, batch4, loss:5.699962275684811e-05
epoch:946, batch5, loss:8.101440471364185e-05
epoch:947, batch0, loss:8.627233910374343e-05
epoch:947, batch1, loss:6.453051173593849e-05
epoch:947, batch2, loss:4.482167787500657e-05
epoch:947, batch3, loss:5.333559965947643e-05
epoch:947, batch4, loss:9.744943236000836e-05
epoch:947, batch5, loss:7.922676741145551e-05
epoch:948, batch0, loss:3.429840216995217e-05
epoch:948, batch1, loss:7.895071030361578e-05
epoch:948, batch2, loss:9.379794937558472e-05
epoch:948, batch3, loss:5.809682988910936e-05
epoch:948, batch4, loss:5.094351581647061e-05
epoch:948, batch5, loss:6.163507350720465e-05
epoch:949, batch0, loss:9.88457613857463e-05
epoch:949, batch1, loss:8.897438965504989e-05
epoch:949, batch2, loss:7.242047286126763e-05
epoch:949, batch3, loss:2.9668801289517432e-05
epoch:949, batch4, loss:8.280251495307311e-05
epoch:949, batch5, loss:0.00014241650933399796
epoch:950, batch0, loss:8.645945490570739e-05
epoch:950, batch1, loss:4.065369284944609e-05
epoch:950, batch2, loss:5.2316085202619433e-05
epoch:950, batch3, loss:9.331641194876283e-05
epoch:950, batch4, loss:0.00016289604536723346
epoch:950, batch5, loss:0.00010909768752753735
epoch:951, batch0, loss:2.051846240647137e-05
epoch:951, batch1, loss:0.00010083688539452851
epoch:951, batch2, loss:0.00016099889762699604
epoch:951, batch3, loss:0.0001266503968508914
epoch:951, batch4, loss:9.830442286329344e-05
epoch:951, batch5, loss:5.072157728136517e-05
epoch:952, batch0, loss:0.00010082730304915458
epoch:952, batch1, loss:0.00017584598390385509
epoch:952, batch2, loss:0.00014841464872006327
epoch:952, batch3, loss:3.766908776015043e-05
epoch:952, batch4, loss:6.298189691733569e-05
epoch:952, batch5, loss:0.00014495379582513124
epoch:953, batch0, loss:0.00014012529572937638
epoch:953, batch1, loss:9.603404760127887e-05
epoch:953, batch2, loss:7.017273310339078e-05
epoch:953, batch3, loss:7.60049297241494e-05
epoch:953, batch4, loss:0.00015685209655202925
epoch:953, batch5, loss:0.00014684110647067428
epoch:954, batch0, loss:5.319496995070949e-05
epoch:954, batch1, loss:8.532131323590875e-05
epoch:954, batch2, loss:0.00013473123544827104
epoch:954, batch3, loss:0.0001462929940316826

epoch:954, batch4, loss:0.0001350218226434663
epoch:954, batch5, loss:8.648552466183901e-05
epoch:955, batch0, loss:9.800710540730506e-05
epoch:955, batch1, loss:0.00017248844960704446
epoch:955, batch2, loss:0.00015841054846532643
epoch:955, batch3, loss:7.111075683496892e-05
epoch:955, batch4, loss:0.000105309794889763
epoch:955, batch5, loss:0.00017139248666353524
epoch:956, batch0, loss:0.00013675694935955107
epoch:956, batch1, loss:0.00012114403943996876
epoch:956, batch2, loss:9.84287034953013e-05
epoch:956, batch3, loss:8.779436757322401e-05
epoch:956, batch4, loss:0.00017616897821426392
epoch:956, batch5, loss:0.00017253492842428386
epoch:957, batch0, loss:6.625711830565706e-05
epoch:957, batch1, loss:9.525952191324905e-05
epoch:957, batch2, loss:0.00013482518261298537
epoch:957, batch3, loss:0.00015512632671743631
epoch:957, batch4, loss:0.0001221459242515266
epoch:957, batch5, loss:5.288777902023867e-05
epoch:958, batch0, loss:0.00010981935338350013
epoch:958, batch1, loss:0.00017211781232617795
epoch:958, batch2, loss:0.0001437590690329671
epoch:958, batch3, loss:7.31266918592155e-05
epoch:958, batch4, loss:7.508782437071204e-05
epoch:958, batch5, loss:0.00012898864224553108
epoch:959, batch0, loss:0.0001463920925743878
epoch:959, batch1, loss:0.00013949733693152666
epoch:959, batch2, loss:8.039364183787256e-05
epoch:959, batch3, loss:5.261237674858421e-05
epoch:959, batch4, loss:0.00014614635438192636
epoch:959, batch5, loss:0.00022561060904990882
epoch:960, batch0, loss:8.556649117963389e-05
epoch:960, batch1, loss:7.8573779319413e-05
epoch:960, batch2, loss:9.627161489333957e-05
epoch:960, batch3, loss:0.00012292744941078126
epoch:960, batch4, loss:0.00016292899090331048
epoch:960, batch5, loss:0.0001297378184972331
epoch:961, batch0, loss:9.787001181393862e-05
epoch:961, batch1, loss:0.00010882631613640115
epoch:961, batch2, loss:0.00014805342652834952
epoch:961, batch3, loss:0.00011791568977059796
epoch:961, batch4, loss:0.00010182267578784376
epoch:961, batch5, loss:9.728607983561233e-05
epoch:962, batch0, loss:0.00012736054486595094
epoch:962, batch1, loss:0.00012625547242350876
epoch:962, batch2, loss:0.00012649722339119762
epoch:962, batch3, loss:8.982069994090125e-05

epoch:962, batch4, loss:0.0001171815674751997
epoch:962, batch5, loss:0.0001698357955319807
epoch:963, batch0, loss:9.573189890943468e-05
epoch:963, batch1, loss:0.00010196040966548026
epoch:963, batch2, loss:9.659990610089153e-05
epoch:963, batch3, loss:8.662135223858058e-05
epoch:963, batch4, loss:0.00012621452333405614
epoch:963, batch5, loss:0.00014048197772353888
epoch:964, batch0, loss:8.791241270955652e-05
epoch:964, batch1, loss:9.079347364604473e-05
epoch:964, batch2, loss:0.0001004961522994563
epoch:964, batch3, loss:7.986259879544377e-05
epoch:964, batch4, loss:8.391357550863177e-05
epoch:964, batch5, loss:7.853837450966239e-05
epoch:965, batch0, loss:0.00010383898916188627
epoch:965, batch1, loss:8.81637679412961e-05
epoch:965, batch2, loss:0.00011934584472328424
epoch:965, batch3, loss:7.980692316778004e-05
epoch:965, batch4, loss:8.03674993221648e-05
epoch:965, batch5, loss:9.88861866062507e-05
epoch:966, batch0, loss:8.385194814763963e-05
epoch:966, batch1, loss:7.474664744222537e-05
epoch:966, batch2, loss:0.00011331614223308861
epoch:966, batch3, loss:8.868575969245285e-05
epoch:966, batch4, loss:0.00010824386845342815
epoch:966, batch5, loss:0.00013048973050899804
epoch:967, batch0, loss:6.464844045694917e-05
epoch:967, batch1, loss:6.853149534435943e-05
epoch:967, batch2, loss:0.00010114057658938691
epoch:967, batch3, loss:7.435811858158559e-05
epoch:967, batch4, loss:9.299436351284385e-05
epoch:967, batch5, loss:0.00010637767263688147
epoch:968, batch0, loss:8.733807771932334e-05
epoch:968, batch1, loss:8.812727173790336e-05
epoch:968, batch2, loss:0.00010291230864822865
epoch:968, batch3, loss:6.298618973232806e-05
epoch:968, batch4, loss:7.081095827743411e-05
epoch:968, batch5, loss:9.246177796740085e-05
epoch:969, batch0, loss:9.385008888784796e-05
epoch:969, batch1, loss:0.00010297013795934618
epoch:969, batch2, loss:0.00011240753519814461
epoch:969, batch3, loss:6.144371582195163e-05
epoch:969, batch4, loss:6.670286529697478e-05
epoch:969, batch5, loss:9.477634739596397e-05
epoch:970, batch0, loss:6.937114812899381e-05
epoch:970, batch1, loss:6.992557609919459e-05
epoch:970, batch2, loss:8.93985343282111e-05
epoch:970, batch3, loss:7.743603782728314e-05

epoch:970, batch4, loss:9.062160097528249e-05
epoch:970, batch5, loss:8.949209586717188e-05
epoch:971, batch0, loss:5.9068344853585586e-05
epoch:971, batch1, loss:5.146968032931909e-05
epoch:971, batch2, loss:6.498685979750007e-05
epoch:971, batch3, loss:7.859025208745152e-05
epoch:971, batch4, loss:8.341264037881047e-05
epoch:971, batch5, loss:8.022985275601968e-05
epoch:972, batch0, loss:7.463854853995144e-05
epoch:972, batch1, loss:9.514037810731679e-05
epoch:972, batch2, loss:9.515722194919363e-05
epoch:972, batch3, loss:5.239617166807875e-05
epoch:972, batch4, loss:4.838575841858983e-05
epoch:972, batch5, loss:7.893324800534174e-05
epoch:973, batch0, loss:7.636818190803751e-05
epoch:973, batch1, loss:0.00010157956421608105
epoch:973, batch2, loss:8.67250346345827e-05
epoch:973, batch3, loss:4.144278500461951e-05
epoch:973, batch4, loss:6.776051304768771e-05
epoch:973, batch5, loss:0.00010760877921711653
epoch:974, batch0, loss:5.42207999387756e-05
epoch:974, batch1, loss:4.34654502896592e-05
epoch:974, batch2, loss:4.0602448279969394e-05
epoch:974, batch3, loss:5.320679701981135e-05
epoch:974, batch4, loss:8.916777005651966e-05
epoch:974, batch5, loss:8.971922216005623e-05
epoch:975, batch0, loss:4.613916826201603e-05
epoch:975, batch1, loss:3.548310269252397e-05
epoch:975, batch2, loss:5.984940435155295e-05
epoch:975, batch3, loss:6.582232163054869e-05
epoch:975, batch4, loss:6.268640572670847e-05
epoch:975, batch5, loss:4.221709968987852e-05
epoch:976, batch0, loss:4.858107422478497e-05
epoch:976, batch1, loss:7.987819844856858e-05
epoch:976, batch2, loss:9.379004768561572e-05
epoch:976, batch3, loss:5.9689682530006394e-05
epoch:976, batch4, loss:4.719653952633962e-05
epoch:976, batch5, loss:6.215967005118728e-05
epoch:977, batch0, loss:5.3504700190387666e-05
epoch:977, batch1, loss:9.098916780203581e-05
epoch:977, batch2, loss:5.552383663598448e-05
epoch:977, batch3, loss:3.754546560230665e-05
epoch:977, batch4, loss:6.399335688911378e-05
epoch:977, batch5, loss:9.765585127752274e-05
epoch:978, batch0, loss:5.76317761442624e-05
epoch:978, batch1, loss:5.436045103124343e-05
epoch:978, batch2, loss:3.6338438803795725e-05
epoch:978, batch3, loss:4.467157123144716e-05

epoch:978, batch4, loss:7.690013444516808e-05
epoch:978, batch5, loss:7.663243741262704e-05
epoch:979, batch0, loss:4.899925988866016e-05
epoch:979, batch1, loss:3.5984798159915954e-05
epoch:979, batch2, loss:7.87534227129072e-05
epoch:979, batch3, loss:7.052085129544139e-05
epoch:979, batch4, loss:7.241433195304126e-05
epoch:979, batch5, loss:5.445948409033008e-05
epoch:980, batch0, loss:4.095105396118015e-05
epoch:980, batch1, loss:6.994968134677038e-05
epoch:980, batch2, loss:8.495614747516811e-05
epoch:980, batch3, loss:4.7332345275208354e-05
epoch:980, batch4, loss:3.974816354457289e-05
epoch:980, batch5, loss:5.697348387911916e-05
epoch:981, batch0, loss:5.617084025288932e-05
epoch:981, batch1, loss:7.216284575406462e-05
epoch:981, batch2, loss:3.810775524470955e-05
epoch:981, batch3, loss:2.7193271307623945e-05
epoch:981, batch4, loss:4.9619353376328945e-05
epoch:981, batch5, loss:6.717669748468325e-05
epoch:982, batch0, loss:5.16788313689176e-05
epoch:982, batch1, loss:2.8062757337465882e-05
epoch:982, batch2, loss:4.008834366686642e-05
epoch:982, batch3, loss:4.865336086368188e-05
epoch:982, batch4, loss:6.097235018387437e-05
epoch:982, batch5, loss:5.596067785518244e-05
epoch:983, batch0, loss:3.188101254636422e-05
epoch:983, batch1, loss:2.894225144700613e-05
epoch:983, batch2, loss:7.732183439657092e-05
epoch:983, batch3, loss:5.7587829360272735e-05
epoch:983, batch4, loss:4.348225411376916e-05
epoch:983, batch5, loss:3.857266710838303e-05
epoch:984, batch0, loss:3.8045644032536075e-05
epoch:984, batch1, loss:6.045576083124615e-05
epoch:984, batch2, loss:6.021564331604168e-05
epoch:984, batch3, loss:3.4289514587726444e-05
epoch:984, batch4, loss:3.2802268833620474e-05
epoch:984, batch5, loss:5.377264824346639e-05
epoch:985, batch0, loss:5.5399177654180676e-05
epoch:985, batch1, loss:4.8227651859633625e-05
epoch:985, batch2, loss:2.4393633793806657e-05
epoch:985, batch3, loss:2.795069849526044e-05
epoch:985, batch4, loss:4.647694731829688e-05
epoch:985, batch5, loss:7.441984780598432e-05
epoch:986, batch0, loss:4.4751184759661555e-05
epoch:986, batch1, loss:2.6905967388302088e-05
epoch:986, batch2, loss:4.661650018533692e-05
epoch:986, batch3, loss:5.7351422583451495e-05

epoch:986, batch4, loss:5.263230195851065e-05
epoch:986, batch5, loss:4.0571125282440335e-05
epoch:987, batch0, loss:2.6000445359386504e-05
epoch:987, batch1, loss:3.4122582292184234e-05
epoch:987, batch2, loss:7.735463441349566e-05
epoch:987, batch3, loss:5.6387765653198585e-05
epoch:987, batch4, loss:4.4003070797771215e-05
epoch:987, batch5, loss:3.3613665436860174e-05
epoch:988, batch0, loss:4.286185503588058e-05
epoch:988, batch1, loss:5.558644625125453e-05
epoch:988, batch2, loss:4.318209539633244e-05
epoch:988, batch3, loss:2.0692430553026497e-05
epoch:988, batch4, loss:3.165030648233369e-05
epoch:988, batch5, loss:5.946639066678472e-05
epoch:989, batch0, loss:5.24439055880066e-05
epoch:989, batch1, loss:5.3338568250183016e-05
epoch:989, batch2, loss:2.19616213144036e-05
epoch:989, batch3, loss:2.575155667727813e-05
epoch:989, batch4, loss:3.9774888136889786e-05
epoch:989, batch5, loss:5.3513969760388136e-05
epoch:990, batch0, loss:2.4208557078964077e-05
epoch:990, batch1, loss:2.172921085730195e-05
epoch:990, batch2, loss:4.046135290991515e-05
epoch:990, batch3, loss:4.363317202660255e-05
epoch:990, batch4, loss:4.7072044253582135e-05
epoch:990, batch5, loss:3.649115751613863e-05
epoch:991, batch0, loss:1.740766310831532e-05
epoch:991, batch1, loss:2.2454792997450568e-05
epoch:991, batch2, loss:5.56270606466569e-05
epoch:991, batch3, loss:3.982355701737106e-05
epoch:991, batch4, loss:3.713473415700719e-05
epoch:991, batch5, loss:2.957785181934014e-05
epoch:992, batch0, loss:2.855069760698825e-05
epoch:992, batch1, loss:4.743017052533105e-05
epoch:992, batch2, loss:4.251008067512885e-05
epoch:992, batch3, loss:2.3054060875438154e-05
epoch:992, batch4, loss:2.9927155992481858e-05
epoch:992, batch5, loss:3.965992436860688e-05
epoch:993, batch0, loss:3.162080975016579e-05
epoch:993, batch1, loss:4.721516597783193e-05
epoch:993, batch2, loss:2.4778410079306923e-05
epoch:993, batch3, loss:1.8158261809730902e-05
epoch:993, batch4, loss:3.045445555471815e-05
epoch:993, batch5, loss:3.88310436392203e-05
epoch:994, batch0, loss:3.0183255148585886e-05
epoch:994, batch1, loss:3.183250373695046e-05
epoch:994, batch2, loss:2.969333036162425e-05
epoch:994, batch3, loss:3.064916381845251e-05

```
epoch:994, batch4, loss:3.94575072277803e-05
epoch:994, batch5, loss:2.9087093935231678e-05
epoch:995, batch0, loss:1.9837654690491036e-05
epoch:995, batch1, loss:2.200637391069904e-05
epoch:995, batch2, loss:3.6949466448277235e-05
epoch:995, batch3, loss:3.3278709452133626e-05
epoch:995, batch4, loss:4.121192978345789e-05
epoch:995, batch5, loss:3.402773290872574e-05
epoch:996, batch0, loss:1.855433583841659e-05
epoch:996, batch1, loss:3.9434780774172395e-05
epoch:996, batch2, loss:3.5189146728953347e-05
epoch:996, batch3, loss:2.2904088837094605e-05
epoch:996, batch4, loss:3.065337295993231e-05
epoch:996, batch5, loss:3.72620124835521e-05
epoch:997, batch0, loss:2.66337301582098e-05
epoch:997, batch1, loss:4.9737838708097115e-05
epoch:997, batch2, loss:3.1848041544435546e-05
epoch:997, batch3, loss:2.192636748077348e-05
epoch:997, batch4, loss:2.998675154231023e-05
epoch:997, batch5, loss:2.6812875148607418e-05
epoch:998, batch0, loss:2.5382549210917205e-05
epoch:998, batch1, loss:3.304806159576401e-05
epoch:998, batch2, loss:3.0147801226121373e-05
epoch:998, batch3, loss:2.552255136833992e-05
epoch:998, batch4, loss:3.1571093131788075e-05
epoch:998, batch5, loss:2.3873461032053456e-05
epoch:999, batch0, loss:2.2807975256000645e-05
epoch:999, batch1, loss:2.7366806534701027e-05
epoch:999, batch2, loss:2.900266190408729e-05
epoch:999, batch3, loss:2.1746371203335002e-05
epoch:999, batch4, loss:3.406898395041935e-05
epoch:999, batch5, loss:3.84650775231421e-05
```

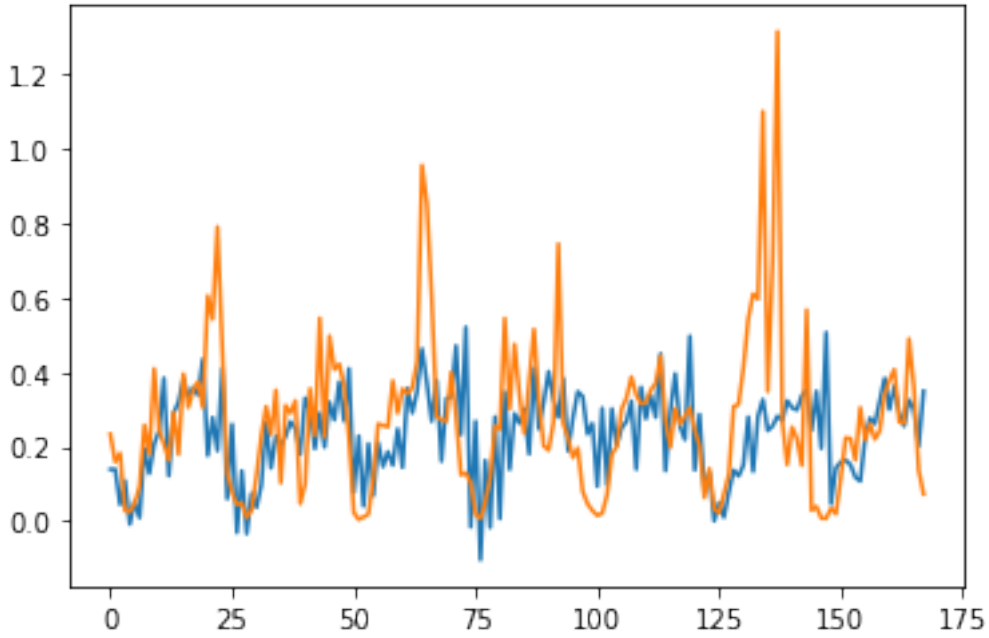
[212]:
```
losses
```

[212]:
```
[0.21883484948750664,
 0.2446167534296321,
 0.216327766289951,
 0.20463085938756365,
 0.21608432002969907,
 0.20334379892486285,
 0.2334473316376589,
 0.22305867227061932,
 0.21807741393100835]
```

[197]:
```
import matplotlib.pyplot as plt
plt.plot(result)
plt.plot(testY[:168])
```

```
print(MSE(testY[168]))
```

[197]: [<matplotlib.lines.Line2D at 0x275ac066cd0>]



4 3.

[114]:
```
timestr = '2018/3/1 0:00:00'
stamp1 = str2stamp(timestr)
stamp2 = stamp1 + 3600
timedatetime = datetime.datetime.fromtimestamp(stamp2)
print(timedatetime)
```

```
2018-03-01 00:59:59.100000
```

In [1]:

```
import numpy
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
import  pandas as pd
import  os
from keras.models import Sequential, load_model
```

In [259]:

```
from sklearn.preprocessing import MinMaxScaler
dataframe = pd.read_excel('ex_long_term/279.xlsx', usecols=[2], encoding='gbk')
dataframe.fillna(1)
ndataframe=pd.read_excel('ex_long_term/279.xlsx', usecols=[3], encoding='gbk')
dataset = dataframe.values
ndataset=ndataframe.values
# 将整型变为float
dataset = dataset.astype('float32')
#归一化 在下一步会讲解
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
ndataset=scaler.fit_transform(ndataset)
```

In [260]:

```
dataset
```

Out[260]:

```
array([[0.6523911 ],
       [0.5272889 ],
       [0.99999994],
       [0.9883291 ],
       [0.43333337],
       [0.4494683 ],
       [0.33503616],
       [0.51416504],
       [0.46831495],
       [0.58037823],
       [0.6035212 ],
       [0.32636094],
       [0.37455058],
       [0.45527276],
       [0.49831766],
       [0.51750696],
       [0.7435779 ],
       [0.6991332 ],
       [0.57122016],
       [0.3764366 ],
       [0.196013  ],
       [0.03599946],
       [0.        ],
       [0.3000834 ],
       [0.3000834 ],
       [0.3000834 ],
       [0.3000834 ],
       [0.3000834 ],
       [0.3000834 ],
       [0.3000834 ],
       [0.3000834 ],
       [0.3000834 ],
       [0.3000834 ],
       [0.3000834 ],
       [0.3000834 ],
       [0.3000834 ],
       [0.3000834 ],
       [0.3000834 ],
       [0.3000834 ],
       [0.3000834 ],
       [0.3000834 ],
       [0.3000834 ],
       [0.3000834 ],
       [0.3000834 ],
       [0.3000834 ],
       [0.3000834 ],
       [0.3000834 ],
       [0.3000834 ],
       [0.1640973 ],
       [0.39016166]], dtype=float32)
```

In [261]:

```
#上面代码的片段讲解
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
```

In [262]:

```
def create_dataset(dataset, look_back):
#这里的look_back与timestep相同
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back)]
        dataX.append(a)
        dataY.append(dataset[i + look_back])
    return numpy.array(dataX),numpy.array(dataY)
#训练数据太少 look_back并不能过大
look_back = 1
trainX,trainY  = create_dataset(dataset,look_back)
```

In [263]:

```
trainY
```

Out[263]:

```
array([[0.527289  ],
       [1.        ],
       [0.98832923],
       [0.43333343],
       [0.44946837],
       [0.3350362 ],
       [0.5141651 ],
       [0.468315  ],
       [0.5803783 ],
       [0.6035213 ],
       [0.32636097],
       [0.3745506 ],
       [0.45527282],
       [0.49831772],
       [0.517507  ],
       [0.74357796],
       [0.6991333 ],
       [0.5712202 ],
       [0.37643665],
       [0.19601303],
       [0.03599947],
       [0.        ],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.16409731]], dtype=float32)
```

In [264]:

```
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, 1))
trainY.shape
```

Out[264]:

(47, 1)

In [265]:

```
# create and fit the LSTM network
model = Sequential()
model.add(LSTM(20, input_shape=(None,1)))
model.add(Dense(20))
model.add(Dense(10))
model.add(Dense(5))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
import time
timestart=time.time()
history=model.fit(trainX, trainY, epochs=50, batch_size=10, verbose=2)
timeend=time.time()
print(timeend-timestart)
# make predictions
```

```
Epoch 1/50
5/5 - 0s - loss: 0.1573
Epoch 2/50
5/5 - 0s - loss: 0.1086
Epoch 3/50
5/5 - 0s - loss: 0.0643
Epoch 4/50
5/5 - 0s - loss: 0.0433
Epoch 5/50
5/5 - 0s - loss: 0.0304
Epoch 6/50
5/5 - 0s - loss: 0.0280
Epoch 7/50
5/5 - 0s - loss: 0.0312
Epoch 8/50
5/5 - 0s - loss: 0.0303
Epoch 9/50
5/5 - 0s - loss: 0.0277
Epoch 10/50
5/5 - 0s - loss: 0.0273
Epoch 11/50
5/5 - 0s - loss: 0.0264
Epoch 12/50
5/5 - 0s - loss: 0.0261
Epoch 13/50
5/5 - 0s - loss: 0.0257
Epoch 14/50
5/5 - 0s - loss: 0.0257
Epoch 15/50
5/5 - 0s - loss: 0.0247
Epoch 16/50
5/5 - 0s - loss: 0.0242
Epoch 17/50
5/5 - 0s - loss: 0.0238
Epoch 18/50
5/5 - 0s - loss: 0.0233
Epoch 19/50
5/5 - 0s - loss: 0.0229
Epoch 20/50
5/5 - 0s - loss: 0.0225
Epoch 21/50
5/5 - 0s - loss: 0.0219
```

```
Epoch 22/50
5/5 - 0s - loss: 0.0215
Epoch 23/50
5/5 - 0s - loss: 0.0212
Epoch 24/50
5/5 - 0s - loss: 0.0205
Epoch 25/50
5/5 - 0s - loss: 0.0203
Epoch 26/50
5/5 - 0s - loss: 0.0199
Epoch 27/50
5/5 - 0s - loss: 0.0196
Epoch 28/50
5/5 - 0s - loss: 0.0193
Epoch 29/50
5/5 - 0s - loss: 0.0189
Epoch 30/50
5/5 - 0s - loss: 0.0188
Epoch 31/50
5/5 - 0s - loss: 0.0189
Epoch 32/50
5/5 - 0s - loss: 0.0185
Epoch 33/50
5/5 - 0s - loss: 0.0186
Epoch 34/50
5/5 - 0s - loss: 0.0185
Epoch 35/50
5/5 - 0s - loss: 0.0182
Epoch 36/50
5/5 - 0s - loss: 0.0181
Epoch 37/50
5/5 - 0s - loss: 0.0180
Epoch 38/50
5/5 - 0s - loss: 0.0180
Epoch 39/50
5/5 - 0s - loss: 0.0179
Epoch 40/50
5/5 - 0s - loss: 0.0178
Epoch 41/50
5/5 - 0s - loss: 0.0180
Epoch 42/50
5/5 - 0s - loss: 0.0179
Epoch 43/50
5/5 - 0s - loss: 0.0177
Epoch 44/50
5/5 - 0s - loss: 0.0178
Epoch 45/50
5/5 - 0s - loss: 0.0177
Epoch 46/50
5/5 - 0s - loss: 0.0181
Epoch 47/50
5/5 - 0s - loss: 0.0178
Epoch 48/50
5/5 - 0s - loss: 0.0178
Epoch 49/50
5/5 - 0s - loss: 0.0178
Epoch 50/50
5/5 - 0s - loss: 0.0176
4.649643898010254
```

In [266]:

```
#model = load_model(os.path.join("DATA","Test" + ".h5"))
trainPredict = model.predict(trainX)

#反归一化
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform(trainY)
```

In [267]:

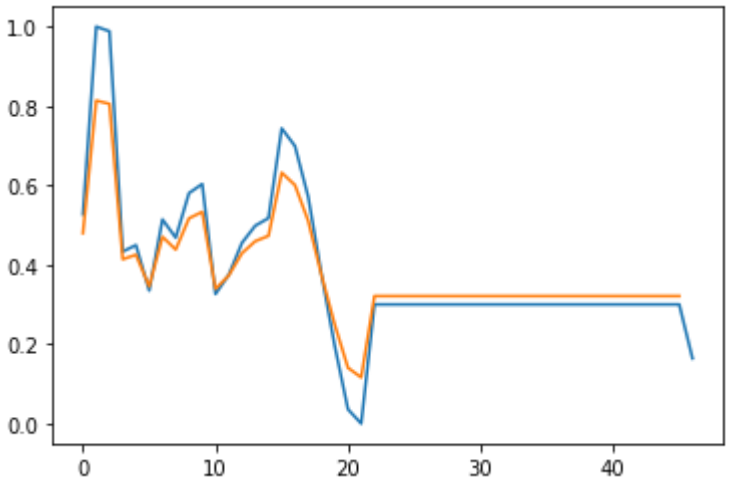
```
trainPredict
```

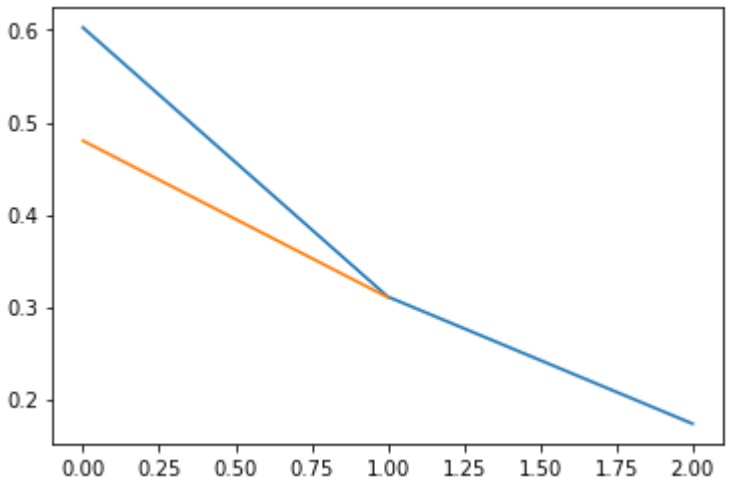
Out[267]:

```
array([[0.5676814 ],
       [0.47964028],
       [0.81367964],
       [0.805413  ],
       [0.4138538 ],
       [0.4251273 ],
       [0.3454193 ],
       [0.47043177],
       [0.43830854],
       [0.51694775],
       [0.5332373 ],
       [0.3394014 ],
       [0.37287608],
       [0.42918536],
       [0.45932025],
       [0.47277614],
       [0.63208944],
       [0.600677  ],
       [0.5105059 ],
       [0.37418845],
       [0.24946733],
       [0.14049992],
       [0.11623319],
       [0.32119647],
       [0.32119647],
       [0.32119647],
       [0.32119647],
       [0.32119647],
       [0.32119647],
       [0.32119647],
       [0.32119647],
       [0.32119647],
       [0.32119647],
       [0.32119647],
       [0.32119647],
       [0.32119647],
       [0.32119647],
       [0.32119647],
       [0.32119647],
       [0.32119647],
       [0.32119647],
       [0.32119647],
       [0.32119647],
       [0.32119647],
       [0.32119647],
       [0.32119647],
       [0.32119647],
       [0.32119647],
       [0.32119647],
       [0.32119647],
       [0.32119647]], dtype=float32)
```

In [268]:

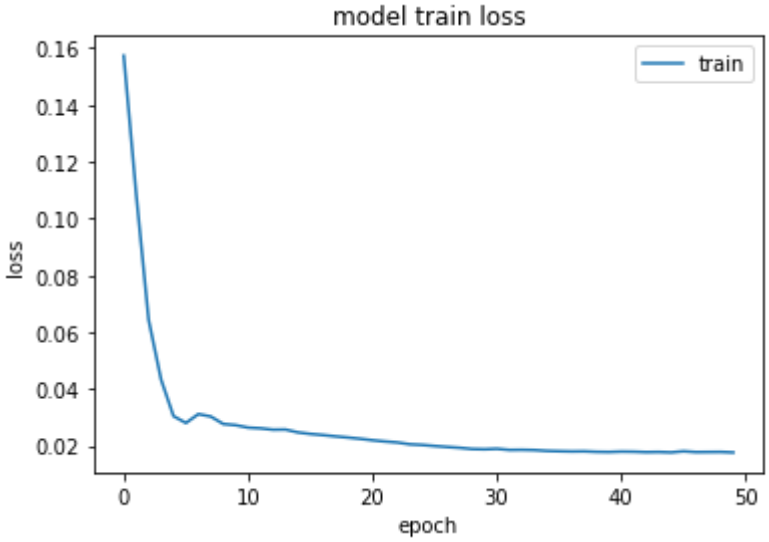
```
plt.plot(trainY)
plt.plot(trainPredict[1:])
plt.show()
plt.plot(testY)
plt.plot(testPredict[1:])
plt.show()
```





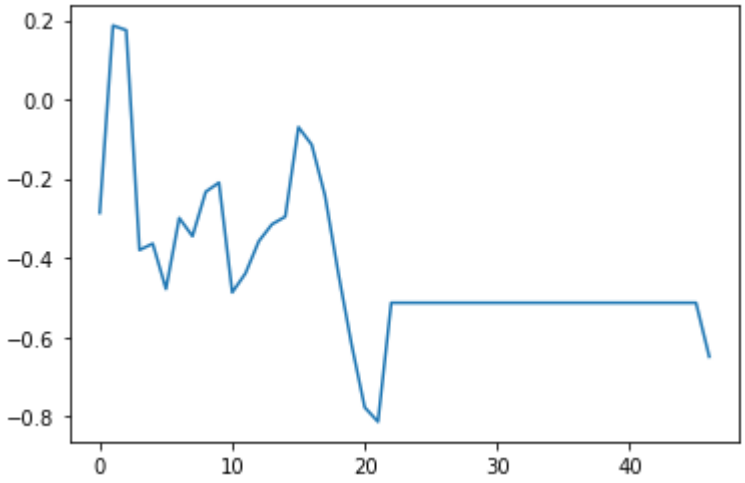
In [269]:

```
plt.plot(history.history['loss'])
plt.title('model train loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper right')
plt.show()
```



In [270]:

```
plt.plot(trainY-trainPredict[1:][1])
plt.show()
```



In [272]:

```
X=dataset
X
```

Out[272]:

```
array([[0.65239114],
       [0.527289  ],
       [1.        ],
       [0.98832923],
       [0.43333343],
       [0.44946837],
       [0.3350362 ],
       [0.5141651 ],
       [0.468315  ],
       [0.5803783 ],
       [0.6035213 ],
       [0.32636097],
       [0.3745506 ],
       [0.45527282],
       [0.49831772],
       [0.517507  ],
       [0.74357796],
       [0.6991333 ],
       [0.5712202 ],
       [0.37643665],
       [0.19601303],
       [0.03599947],
       [0.        ],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.30008343],
       [0.16409731],
       [0.39016172]], dtype=float32)
```

In [273]:

```
import warnings
import numpy as np
warnings.filterwarnings("ignore")
#X=np.append(trainX,testX)
#X=np.delete(X,X[0])
X=X.reshape(49,1,1)
timestart=time.time()
for i in range(102):
    pre=model.predict(X)
    Pre=scaler.inverse_transform(pre)
    X=np.delete(X,X[0],axis=0)
    X=np.append(X,[[Pre[0]]],axis=0)
timeend=time.time()
timeend-timestart
```

Out[273]:

```
3.6318771839141846
```

In [274]:

```
X=X.reshape(49,)
```

In [275]:

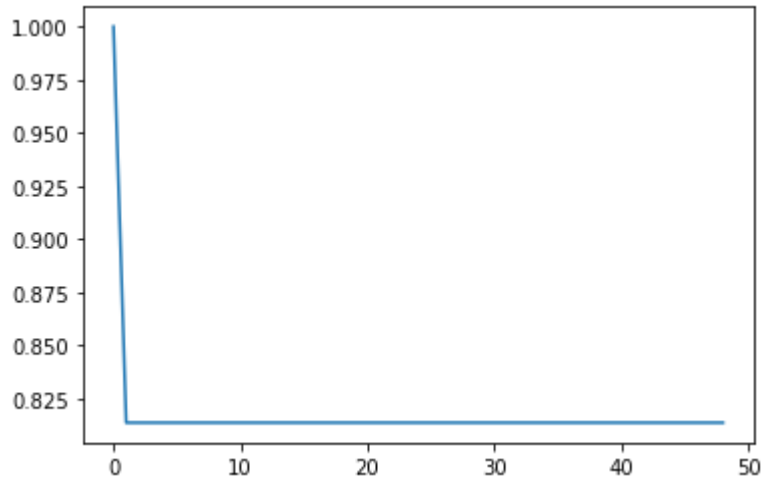
```
X
```

Out[275]:

```
array([1.        , 0.81367964, 0.81367964, 0.81367964, 0.81367964,
       0.81367964, 0.81367964, 0.81367964, 0.81367964, 0.81367964,
       0.81367964, 0.81367964, 0.81367964, 0.81367964, 0.81367964,
       0.81367964, 0.81367964, 0.81367964, 0.81367964, 0.81367964,
       0.81367964, 0.81367964, 0.81367964, 0.81367964, 0.81367964,
       0.81367964, 0.81367964, 0.81367964, 0.81367964, 0.81367964,
       0.81367964, 0.81367964, 0.81367964, 0.81367964, 0.81367964,
       0.81367964, 0.81367964, 0.81367964, 0.81367964, 0.81367964,
       0.81367964, 0.81367964, 0.81367964, 0.81367964], dtype=float32)
```

In [276]:

```
plt.plot(range(49),X)
plt.show()
```



In [277]:

```
X
```

Out[277]:

```
array([1.        , 0.81367964, 0.81367964, 0.81367964, 0.81367964,
       0.81367964, 0.81367964, 0.81367964, 0.81367964, 0.81367964,
       0.81367964, 0.81367964, 0.81367964, 0.81367964, 0.81367964,
       0.81367964, 0.81367964, 0.81367964, 0.81367964, 0.81367964,
       0.81367964, 0.81367964, 0.81367964, 0.81367964, 0.81367964,
       0.81367964, 0.81367964, 0.81367964, 0.81367964, 0.81367964,
       0.81367964, 0.81367964, 0.81367964, 0.81367964, 0.81367964,
       0.81367964, 0.81367964, 0.81367964, 0.81367964, 0.81367964,
       0.81367964, 0.81367964, 0.81367964, 0.81367964], dtype=float32)
```

In [1]:

```
import numpy
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
import  pandas as pd
import  os
from keras.models import Sequential, load_model
```

In [193]:

```
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import warnings
warnings.filterwarnings("ignore")
scaler = MinMaxScaler(feature_range=(0, 1))
def getdata(dataframe):
    dataset = dataframe.values
    # 将整型变为float
    dataset = dataset.astype('float32')
    dataset = scaler.fit_transform(dataset)
    return dataset
def create_dataset(dataset, look_back):
#这里的look_back与timestep相同
    dataX, dataY = [], []
    look_back=1
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back)]
        dataX.append(a)
        dataY.append(dataset[i + look_back])
    return numpy.array(dataX),numpy.array(dataY)
#训练数据太少 look_back并不能过大
look_back = 1
#trainX,trainY  = create_dataset(dataset,look_back)
```

In [194]:

```
model = Sequential()
model.add(LSTM(20, input_shape=(None,1)))
model.add(Dense(20))
model.add(Dense(10))
model.add(Dense(5))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='SGD')
model.summary()
```

```
Model: "sequential_36"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_36 (LSTM)               (None, 20)                1760
_________________________________________________________________
dense_144 (Dense)            (None, 20)                420
_________________________________________________________________
dense_145 (Dense)            (None, 10)                210
_________________________________________________________________
dense_146 (Dense)            (None, 5)                 55
_________________________________________________________________
dense_147 (Dense)            (None, 1)                 6
=================================================================
Total params: 2,451
Trainable params: 2,451
Non-trainable params: 0
_________________________________________________________________
```

In [195]:

```
import time
def train(trainX,trainY):
    timestart=time.time()
    history=model.fit(trainX, trainY, epochs=20, batch_size=10, verbose=2)
    timeend=time.time()
    print(timeend-timestart)
def test(trainX,trainY):
    trainPredict = model.predict(trainX)
    #反归一化
    trainPredict = scaler.inverse_transform(trainPredict)
    trainY = scaler.inverse_transform(trainY)
def predict(X):
    timestart=time.time()
    X=X.reshape(49,1,1)
    for i in range(102):
        pre=model.predict(X)
        Pre=scaler.inverse_transform(pre)
        X=np.delete(X,X[0],axis=0)
        X=np.append(X,[[Pre[0]]],axis=0)
    timeend=time.time()
    timeend-timestart
    X=X.reshape(49,1)
    return X[:-26]
```

In [196]:

```
import os
filelist=os.listdir("long_term_files/long-7")
#filelist=exelist
exlist=os.listdir("ex_long_term")
```

In [197]:

```
import pandas as pd
for files in filelist[87:]:
    if files not in exlist:
        dfup=pd.read_excel('long_term_files/long-7/'+files,usecols=[2],encoding='gbk')
        dfup.fillna(dfdown.mean())
        dfdown=pd.read_excel('long_term_files/long-7/'+files,usecols=[3],encoding='gbk')
        dfdown.fillna(dfdown.mean())
        print(files)
        dataset1=getdata(dfup)
        #dataset1.fillna(dataset1.mean())
        trainX1,trainY1=create_dataset(dataset1,1)
        dataset2=getdata(dfdown)
        #dataset2.fillna(dataset2.mean())
        trainX2,trainY2=create_dataset(dataset2,1)
        train(trainX1,trainY1)
        time.sleep(0.5)
        train(trainX2,trainY2)
        time.sleep(0.5)
        try:
            predX1=predict(dataset1)
            predX2=predict(dataset2)
            if np.nan in predX1 or np.nan in predX2:
                print(files+" error")
                pass
        except ValueError:
            print(files+" error")
            pass
        else:
            newdf=pd.DataFrame(np.c_[np.r_[predX1],np.r_[predX2]])
            writer=pd.ExcelWriter("long_predict/long-7/"+files)
            newdf.to_excel(writer,sheet_name='data',index=False)
            writer.close()
    else:
        continue
```

```
80825.xlsx
Epoch 1/20
5/5 - 0s - loss: 0.1642
Epoch 2/20
5/5 - 0s - loss: 0.0748
Epoch 3/20
5/5 - 0s - loss: 0.0542
Epoch 4/20
5/5 - 0s - loss: 0.0521
Epoch 5/20
5/5 - 0s - loss: 0.0510
Epoch 6/20
5/5 - 0s - loss: 0.0510
Epoch 7/20
5/5 - 0s - loss: 0.0505
Epoch 8/20
5/5 - 0s - loss: 0.0511
Epoch 9/20
5/5 - 0s - loss: 0.0505
```

In [ ]:

```

```

In [ ]:

```
import pandas as pd
df=pd.read_csv('D:/train_data/训练用数据.csv',encoding='gbk')
df.shape
```

In [4]:

```
df.columns
```

Out[4]:

```
Index(['日期', '时间', '小区编号', '上行业务量GB', '下行业务量GB'], dtype='object')
```

In [3]:

```
city=df['小区编号'].unique()
date=df['日期'].unique()
time=df['时间'].unique()
up_flow=df['上行业务量GB']
down_flow=df['下行业务量GB']
```

In [5]:

```
space=df['小区编号'].unique()
space
```

Out[5]:

```
array([     1,      2,      3, ..., 132277, 132278, 132279], dtype=int64)
```

In [8]:

```
import random
n = 0
rand=[]
while n < 100:
    num = random.randint(1, 132279)
    rand.append(num)
    n+=1
print(rand)
```

```
[61317, 69508, 41449, 40991, 59816, 26422, 77429, 111112, 50167, 40581, 74254, 10696
6, 41191, 22628, 92483, 51243, 98505, 24284, 44640, 9460, 61599, 111279, 58126, 1021
58, 41885, 109883, 36858, 118135, 126953, 113867, 8964, 9794, 100314, 12079, 90544,
34409, 29959, 98194, 40602, 122868, 55633, 24223, 2812, 121061, 110981, 126805, 3260
1, 47963, 21410, 4224, 30002, 89955, 13564, 49354, 49301, 23264, 114680, 1454, 8448
8, 43667, 34659, 79348, 62884, 51921, 85081, 64352, 116084, 61723, 35153, 36812, 683
73, 104706, 73072, 128419, 72856, 44331, 9013, 96163, 68866, 64633, 117996, 2303, 68
518, 11931, 115059, 128861, 91504, 53257, 88322, 84465, 49566, 16869, 48671, 120582,
51750, 106386, 96088, 99756, 73013, 45534]
```

In [9]:

```
cup0=[]
cdown0=[]
for t in rand:
        dfi=df.loc[(df['小区编号']==t)]
        aup=dfi['上行业务量GB'].mean()
        adown=dfi['下行业务量GB'].mean()
        cup0.append(aup)
        cdown0.append(adown)
```

In [10]:

```
import numpy as np
import matplotlib.pyplot as plt
cup0=np.array(cup0)
cdown0=np.array(cdown0)
rand=np.array(rand)
df2=pd.DataFrame(np.c_[np.r_[rand],np.r_[cup0],np.r_[cdown0]])
writer=pd.ExcelWriter("space.xlsx")
df2.to_excel(writer,sheet_name='data',index=False)
writer.close()
```

In [26]:

```
date
```

Out[26]:

```
array(['2018/3/1', '2018/3/2', '2018/3/3', '2018/3/4', '2018/3/5',
       '2018/3/6', '2018/3/7', '2018/3/8', '2018/3/9', '2018/3/10',
       '2018/3/11', '2018/3/12', '2018/3/13', '2018/3/14', '2018/3/15',
       '2018/3/16', '2018/3/17', '2018/3/18', '2018/3/19', '2018/3/20',
       '2018/3/21', '2018/3/22', '2018/3/23', '2018/3/24', '2018/3/25',
       '2018/3/26', '2018/3/27', '2018/3/28', '2018/3/29', '2018/3/30',
       '2018/3/31', '018-04-01', '018-04-02', '018-04-03', '018-04-04',
       '2018/4/5', '2018/4/6', '2018/4/7', '018-04-08', '018-04-09',
       '018-04-10', '2018/4/11', '2018/4/12', '2018/4/13', '2018/4/15',
       '2018/4/16', '2018/4/17', '2018/4/18', '2018/4/19'], dtype=object)
```

In [59]:

```
time
```

Out[59]:

```
array(['0:00:00', '1:00:00', '2:00:00', '3:00:00', '4:00:00', '5:00:00',
       '6:00:00', '7:00:00', '8:00:00', '9:00:00', '10:00:00', '11:00:00',
       '12:00:00', '13:00:00', '14:00:00', '15:00:00', '16:00:00',
       '17:00:00', '18:00:00', '19:00:00', '20:00:00', '21:00:00',
       '22:00:00', '23:00:00', '00:00:0', '01:00:0', '02:00:0', '03:00:0',
       '04:00:0', '05:00:0', '06:00:0', '07:00:0', '08:00:0', '09:00:0',
       '10:00:0', '11:00:0', '12:00:0', '13:00:0', '14:00:0', '15:00:0',
       '16:00:0', '17:00:0', '18:00:0', '19:00:0', '20:00:0', '21:00:0',
       '22:00:0', '23:00:0', '04:00:00', '05:00:00', '02:00:00',
       '03:00:00', '07:00:00', '01:00:00', '08:00:00', '06:00:00',
       '09:00:00'], dtype=object)
```

In [23]:

```
cup=[]
cdown=[]
for d in date:
    df0=df.loc[df['日期']==d]
    cup0=[]
    cdown0=[]
    for t in range(0,24):
        dfi=df0.loc[(df0['时间']=='%d:00:00'%t) | (df0['时间']=='%d:00:0'%t) | (df0['时间']=='0%d:0
        aup=dfi['上行业务量GB'].mean()
        adown=dfi['下行业务量GB'].mean()
        cup0.append(aup)
        cdown0.append(adown)
    cup.append(cup0)
    cdown.append(cdown0)
```

In [24]:

```
np.array(cup).shape
```

Out[24]:

(49, 24)

In [27]:

```
cup1=[]
cdown1=[]
for i in cup:
    cup1+=i
for i in cdown:
    cdown1+=i
```

In [30]:

```
cup1[639]
```

Out[30]:

nan

In [51]:

```
import numpy as np
import matplotlib.pyplot as plt
ti=list(range(24))*49
ti=np.array(ti)
cup1=np.array(cup1)
cdown1=np.array(cdown1)
df2=pd.DataFrame(np.c_[np.r_[ti],np.r_[cup1],np.r_[cdown1]])
writer=pd.ExcelWriter("time-2.xlsx")
df2.to_excel(writer,sheet_name='data',index=False)
writer.close()
```

In [57]:

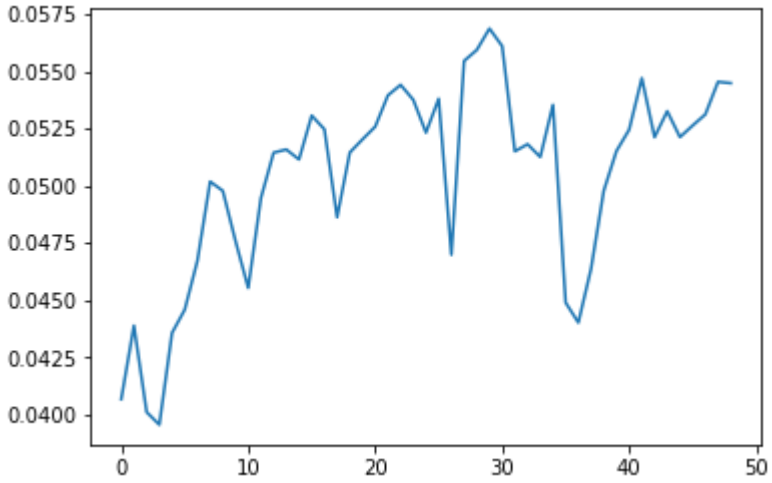
```
df0=df.loc[df['日期']=='2018/3/27']
cup0=[]
cdown0=[]
for t in range(1,10):
        dfi=df0.loc[(df0['时间']=='0%d:00:00'%t)]
        aup=dfi['上行业务量GB'].mean()
        adown=dfi['下行业务量GB'].mean()
        cup0.append(aup)
        cdown0.append(adown)
```

In [58]:

```
dfi0=pd.DataFrame(np.c_[np.r_[cup0],np.r_[cdown0]])
writer=pd.ExcelWriter("time-new.xlsx")
dfi0.to_excel(writer,sheet_name='datanew3',index=False)
writer.close()
```

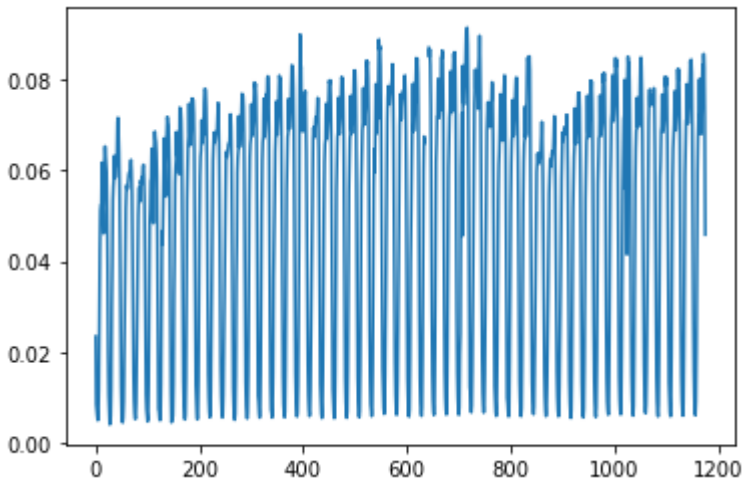
In [29]:

```
df2=pd.read_excel("date.xlsx")
plt.plot(range(49),df2['上行业务量GB'])
plt.show()
```



In [60]:

```
dftime=pd.read_excel("time-final.xlsx")
X=dftime['上行业务量GB']
plt.plot(range(len(X)),X)
plt.show()
```



In [61]:

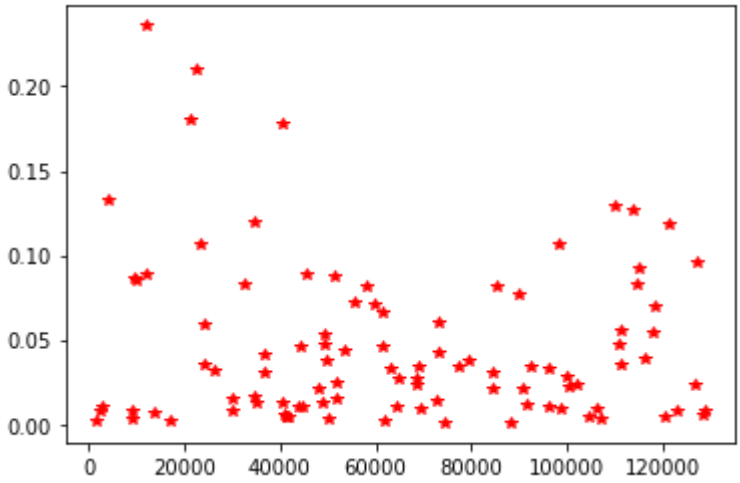
```
ctup=[]
ctdown=[]
for i in range(24):
    dftimei=dftime.loc[dftime['时间']==i]
    aup=dftimei['上行业务量GB'].mean()
    adown=dftimei['下行业务量GB'].mean()
    ctup.append(aup)
    ctdown.append(adown)
```

In [62]:

```
ctup=np.array(ctup)
ctdown=np.array(ctdown)
df2=pd.DataFrame(np.c_[np.r_[ctup],np.r_[ctdown]])
writer=pd.ExcelWriter("time-total.xlsx")
df2.to_excel(writer,sheet_name='data',index=False)
writer.close()
```

In [14]:

```
dfspace=pd.read_excel("space.xlsx")
s=dfspace['小区编号']
x=dfspace['上行业务量GB']
y=dfspace['下行业务量GB']
plt.plot(s,x,'r*')
#plt.plot(s,y,'b*')
plt.show()
```



In [19]:

```
df3=pd.read_excel("time-final.xlsx")
xia=df3['下行业务量GB']
shang=df3['上行业务量GB']
```

In [18]:

```
xia
```

Out[18]:

```
0            3.560744
1            0.830950
2            0.536776
3            0.265031
4            1.537391
                ...
144138195    0.042392
144138196    0.009431
144138197    0.068833
144138198    1.084684
144138199    0.508197
Name: 下行业务量GB, Length: 144138200, dtype: float64
```

In [ ]: