華東理工大學

模式识别大作业

题　　目　共享单车—预测每时段共享单车使用数量概率

学　　院　信息科学与工程

专　　业　控制科学与工程

组　　员　张嘉琛、周颖

指导教师　赵海涛

完成日期：　2019 年 12 月 12 日

1 实验目的

共享单车的使用量不仅可用于研究城市交通的移动性，还可帮助企业进行收支分析：例如对于一个区域需要投放的单车数量，不同时间所需要的维护人员数量，这些问题对于企业发展都非常重要。在实际生活中，共享单车的使用量并非一成不变的，针对不同的时段、日期，其使用量存在较大的差异。

本题给出了某个地区在一段时间内的共享单车使用量，需要解决的问题为根据测试集给出的数据，预测该时段内的单车使用量。为了解决这个问题，我们决定使用 BP 神经网络。

2 实验原理

人工神经网络(Artificial Neural Network, ANN)是通过模仿生物的神经网络的信息处理方式，来对信息进行非线性转换和并行处理的复杂网络系统。它由一个一个的人工神经元联结起来进行计算。人工神经网络常用于复杂的输入输出的系统进行建模，是一种自适应系统，同时可以用于非线性统计性数据进行建模。

人工神经元模型的基本结构如图 2.1 所示：

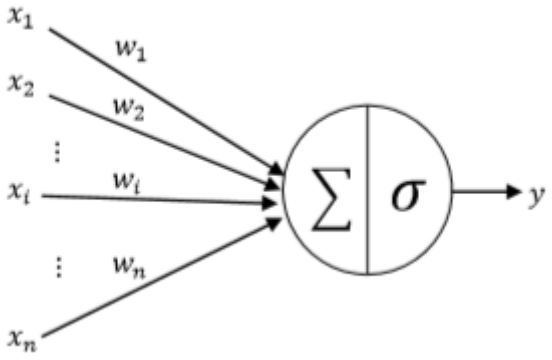


图 2.1 神经元模型

该神经元的输入输出关系为：

$$y = \sigma(\sum_{i=1}^{n} w_i x_i + b)$$

其中x_i为神经元的输入，w_i为各神经元之间的权重，b为神经元的阈值(这里约定阈值写成与权值相似的形式，即把阈值视为样本输入为 1 的随机数)，y为神经元的输出，σ(·)为神经元的激励函数，可以理解为某种非线性变化，常用的转化函数如图 2.2。

单个神经元的功能并不强大，但是多个不同形式构成的人工神经网络，是一个并行和分布式的网络结构，前馈神经网络的典型结构模型如图 2.3 所示：

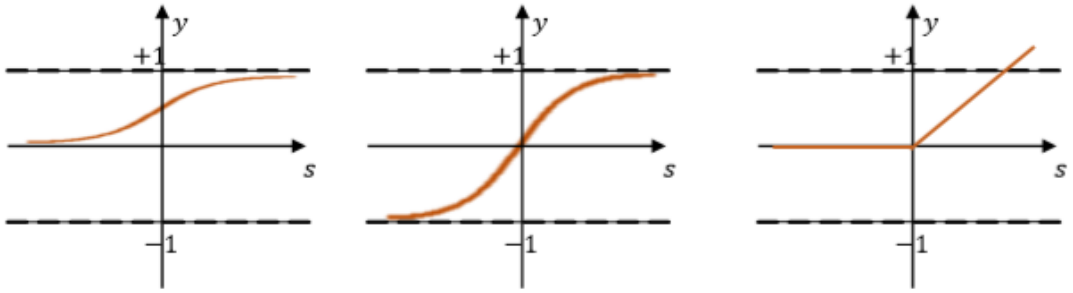


图 2.2 常用的转换函数(从左到右为 sigmod 函数，tanh 函数，ReLU 函数)

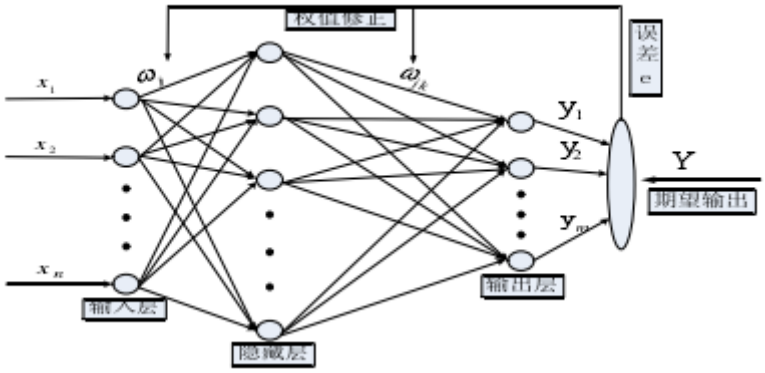


图 2.3 BP 神经网络结构图

BP 神经网络(Backpropagation Neural Network)，又叫反向传播神经网络，这是一种监督学习算法，常用于训练多层感知机。BP 神经网络用于解决非线性连续函数的多层前馈神经网络权重调整问题的效果很好。BP 算法是 Delta 规则的推广，要求每个节点所使用的激励函数必须是可微的。典型的 BP 神经网络采用的是梯度下降算法。一般具有三层或者三层以上的结构，分别为输入层，一层或者多层隐藏层，输出层，上下层层之间的神经元是全连接，层内的神经元之间无连接，各隐含层节点一般使用 Sigmoid 激励函数。

BP 算法的核心在于梯度下降法，即 BP 网络的误差调整方向总是沿着误差下降最快的方向进行。

在正向传播过程中，隐含层或输出层任一神经元将来自前一神经元传来的信息进行加权整合，通常还会在整合过的信息中添加一个阈值，这主要是模仿生物学中神经元必须达到一行的阈值才会触发的原理，再经激活函数变换后传到下一层神经元。

(1) 从输入层到中间层

$$z_i^{(1)} = w_i^{(1)} x = \sum_{j=1}^{do} w_{ij}^{(1)} + b_i^{(1)}$$

$$h_i = \sigma\left(z_i^{(1)}\right) = \sigma(\sum_{j=1}^{d_0} w_{ij}^{(1)} x_j + b_i^{(1)})$$

其中：

$$i = 1,2 \dots d_1, \boldsymbol{x} = \left[x_1, x_2, \dots, x_{d_0}\right]^{\tau}, \boldsymbol{w_i^{(1)}} = [w_{i1}^{(1)}, w_{i2}^{(1)}, \dots, w_{id_0}^{(1)}]$$

(2) 从中间层到输出层

$$z_i^{(2)} = w_i^{(2)} x = \sum_{j=1}^{do} w_{ij}^{(2)} + b_i^{(2)}$$

$$h_i = \sigma\left(z_i^{(2)}\right) = \sigma(\sum_{j=1}^{d_1} w_{ij}^{(2)} x_j + b_i^{(2)})$$

其中：

$$i = 1,2 \dots d_2, \boldsymbol{h} = \left[h_1, h_2, \dots, h_{d_1}\right]^{\tau}, \boldsymbol{w_i^{(2)}} = [w_{i1}^{(2)}, w_{i2}^{(2)}, \dots, w_{id_0}^{(2)}]$$

在反向传播过程中，构造关于输出值与真实值的目标函数，按照减少输出值与真实值之间误差的方向，按照梯度下降法，从输出层反向经过各中间层回到输入层，逐步修正各连接权值与阈值。

构造目标函数：

$$J = \frac{1}{2}\sum_{i=1}^{d_2}(t_i - y_i)^2$$

(1) 从输出层到中间层

$$\frac{\partial J}{\partial z_i^{(2)}} = \frac{\partial J}{\partial y_i \partial z_i^{(2)}} = -(t_i - y_i)\sigma'(z_i^{(2)})$$

其中$i = 1,2 \dots d_2$。下面为书写方便，令：$\delta_i^{(2)} = -(t_i - y_i)\sigma'(z_i^{(2)})$

$$\frac{\partial J}{\partial w_{ij}^{(2)}} = \frac{\partial J \partial z_i^{(2)}}{\partial z_i^{(2)} \partial w_{ij}^{(2)}} = \delta_i^{(2)} h_j$$

$$w_{ij}^{(2)} \leftarrow w_{ij}^{(2)} \leftarrow \alpha \delta_i^{(2)} h_j$$

$$\frac{\partial J}{\partial b_i^{(2)}} = \frac{\partial J \partial z_i^{(2)}}{\partial z_i^{(2)} \partial b_i^{(2)}} = \delta_i^{(2)}$$

$$b_i^{(2)} \leftarrow b_i^{(2)} \leftarrow \alpha \delta_i^{(2)}$$

其中$i = 1,2 \dots d_2, j = 1,2, \dots, d_1$。

(2) 从中间层到输入层

因为$z_i^{(2)}$和每个$z_i^{(2)}(i=1,2\dots d_2)$都有关系，因此求导时要对每个$z_i^{(2)}$进行

$$\frac{\partial J}{\partial b_i^{(1)}}=\sum_{k=1}^{d_2}\frac{\partial J}{\partial z_k^{(2)}}\frac{\partial z_k^{(2)}}{\partial z_i^{(2)}}=\sum_{k=1}^{d_2}\frac{\partial J}{\partial z_k^{(2)}}\frac{\partial z_k^{(2)}\partial h_i}{\partial h_i\partial z_i^{(1)}}=\sum_{k=1}^{d_2}\delta_k^{(2)}w_{k_i}^{(2)}\sigma'(z_i^{(1)})$$

其中$i=1,2\dots d_2$，$i=1,2\dots d_1$为下面书写方便，令$\delta_i^{(2)}=\sum_{k=1}^{d_2}\delta_k^{(2)}w_{k_i}^{(2)}\sigma'(z_i^{(1)})$

$$\frac{\partial J}{\partial w_{ij}^{(2)}}=\frac{\partial J\partial z_i^{(1)}}{\partial z_i^{(1)}\partial w_{ij}^{(1)}}=\delta_i^{(1)}x_j$$

$$w_{ij}^{(1)}\leftarrow w_{ij}^{(1)}\leftarrow\alpha\delta_i^{(1)}x_j$$

$$\frac{\partial J}{\partial b_i^{(1)}}=\frac{\partial J\partial z_i^{(1)}}{\partial z_i^{(1)}\partial b_i^{(1)}}=\delta_i^{(1)}$$

$$b_i^{(1)}\leftarrow b_i^{(1)}\leftarrow\alpha\delta_i^{(1)}$$

其中$i=1,2\dots d_1$，$j=1,2\dots d_0$。

3 数据集简介

本次实验所用数据集来自 lintcode，其网址在附录中给出。实验主要使用的数据文件有 train.csv 和 test.csv。数据字段含义如表 3-1 所示。

表 3-1 特征含义

datetime	日期时间
season	1、2、3、4 分别对应春夏秋冬
holiday	今天是否是节假日
workingday	今天是否是工作日
weather	1 晴天、多云;2 雾天; 3 小雨、小雪;4 大雨、冰雹、暴雪
temp	摄氏温度
atemp	体感摄氏温度
humidity	湿度
windspeed	风速
casual	非登记用户用车
registered	登记用户用车
count	总用车量

其中，train.csv 记录了日期从每月 1-15 日的数据，test.csv 记录了每月 16-19 日的数据，也是将要进行预测的时间段。本题要求按 test 集的顺序输出相应的预

测结果。

4 实验步骤

4.1 数据预处理

在处理数据时，首先应对明显具有偏差的数据进行去除。我们按照拉依达准则对数据进行处理。拉依达准则的具体定义为：假设一组检测数据只含有随机误差，对其计算处理得到标准偏差，按一定概率确定一个区间，认为凡是超过这个区间的误差，就不属于随机误差而是粗大误差，含有该误差的数据应当予以提出。因此，我们首先使用拉依达准则对 count 按下述公式进行处理。

若当前 count 值满足$|\text{剩余误差}| = |\text{测量值} - \text{算术平均值}| > 3\sigma$，的条件，则认为这个测量值是含有粗大误差的坏值，应当予以剔除。

将坏值剔除后，对特征值本身进行处理。首先，注意到日期信息中包含大量信息，其中的小时及年份显然是重要特征。但数据文件所给出的是字符串格式，因此我们首先分割这一部分数据，将日期转化为年份、月份、小时这三列特征。

我们注意到，条件给出的特征可以分为两类。其中如年份、季节等无序多分类变量，虽然用数字表示，但本身并不具有数值意义。因而在引入模型时需要将其作为哑变量处理。我们对其进行特征编码以合理表示其代表的特征类别。在这一步骤中，采用了 One-Hot 编码。One-Hot 即独热编码，又称一位有效编码，其方法是使用 N 位状态寄存器来对 N 个状态进行编码，每个状态都具有独立的寄存器位，并且在任意时候，其中只有一位有效。

第二类数据为具有数值意义的数据，如温度、湿度等，考虑对其进行归一化处理以提高提高模型的收敛速度和模型精度。首先观察该类数据并查阅相关资料，可知温度、湿度、风速的分布大约满足高斯分布，因此可以使用 z-score 标准化方法来进行处理。由于体感温度与温度大致成线性关系，因此只需保留一个特征即可。我们选择保留温度作为特征。对于 count 数据，为使预测结果不出现负值，因此取对数后进行处理。由于原本的数据含有零数据，因此在取对数前先对各数值加一。查阅资料，发现该类处理之后的 count 数据满足高斯分布，因此同样可以用 z-score 进行处理。z-score 标准化也被称为标准差标准化。这种方法基于原始数据的均值和标准差进行数据的标准化。经过处理的数据符合标准正态分布，即均值为 0，标准差为 1，转化函数为：

$$x^* = (x - \mu)/\sigma$$

4.2 实验过程与结果

预处理过程中，针对温度、湿度、风速，将训练集与测试集数据拼接后求得

总体均值与标准差，之后按上述预处理步骤进行处理。模型得到预测值后，通过进行指数运算及减一处理，得到最终的预测值。

经过预处理之后的部分特征值如表 4-1 所示。

表 4-1 预处理特征值结果示例

	holiday	workingday	temp	humidity	windspeed	season_1	season_2
0	0	0	-1.3251	0.986965	-1.56481	1	0
1	0	0	-1.4305	0.935	-1.56481	1	0
2	0	0	-1.4305	0.935	-1.56481	1	0
3	0	0	-1.3251	0.675173	-1.56481	1	0
4	0	0	-1.3251	0.675173	-1.56481	1	0
5	0	0	-1.3251	0.675173	-0.83006	1	0
6	0	0	-1.4305	0.935	-1.56481	1	0
7	0	0	-1.5359	1.246792	-1.56481	1	0
season_3	season_4	weather_1	weather_2	weather_3	weather_4	hour_00	hour_01
0	0	1	0	0	0	1	0
0	0	1	0	0	0	0	1
0	0	1	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	0	0	0	0	0
hour_02	hour_03	hour_04	hour_05	hour_06	hour_07	hour_08	hour_09
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
hour_10	hour_11	hour_12	hour_13	hour_14	hour_15	hour_16	hour_17
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
hour_18	hour_19	hour_20	hour_21	hour_22	hour_23	month_01	month_02
0	0	0	0	0	0	1	0

0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0
month_03	month_04	month_05	month_06	month_07	month_08	month_09	month_10
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
month_11	month_12	year_2011	year_2012				
0	0	1	0				
0	0	1	0				
0	0	1	0				
0	0	1	0				
0	0	1	0				
0	0	1	0				
0	0	1	0				
0	0	1	0				

图 4.1 给出部分训练集中预处理后的结果数据。

	casual	registered	count
0	-0.85819	-1.2383028	-1.21702
1	-0.31559	-0.6209407	-0.59405
2	-0.58689	-0.7392385	-0.74765
3	-0.85819	-1.4119389	-1.35441
4	-1.78578	-2.6393533	-2.73139
5	-1.78578	-2.6393533	-2.73139
6	-1.05069	-3.1384175	-2.44447
7	-1.32199	-2.3474194	-2.2409
8	-1.32199	-1.6412248	-1.66706
9	-0.31559	-1.737367	-1.30559
10	-0.06954	-0.8208349	-0.66669
11	0.419505	-0.6659552	-0.36091
12	0.490003	-0.2401742	-0.07814
13	0.804488	-0.3511623	0.000568
14	0.611996	-0.0592285	0.084742
15	0.699016	-0.0692985	0.110712

图 4.1 训练集结果数据预处理示例

图 4.2 给出部分预测结果：

datetime	count
2011/1/16 0:00	28
2011/1/16 1:00	20
2011/1/16 2:00	14
2011/1/16 3:00	6
2011/1/16 4:00	2
2011/1/16 5:00	2
2011/1/16 6:00	5
2011/1/16 7:00	12
2011/1/16 8:00	28
2011/1/16 9:00	50
2011/1/16 10:00	75
2011/1/16 11:00	90
2011/1/16 12:00	102
2011/1/16 13:00	100
2011/1/16 14:00	107
2011/1/16 15:00	93
2011/1/16 16:00	90

图 4.1 预测数据示例

训练过程中的 loss 曲线变化在图 4.3 中给出。

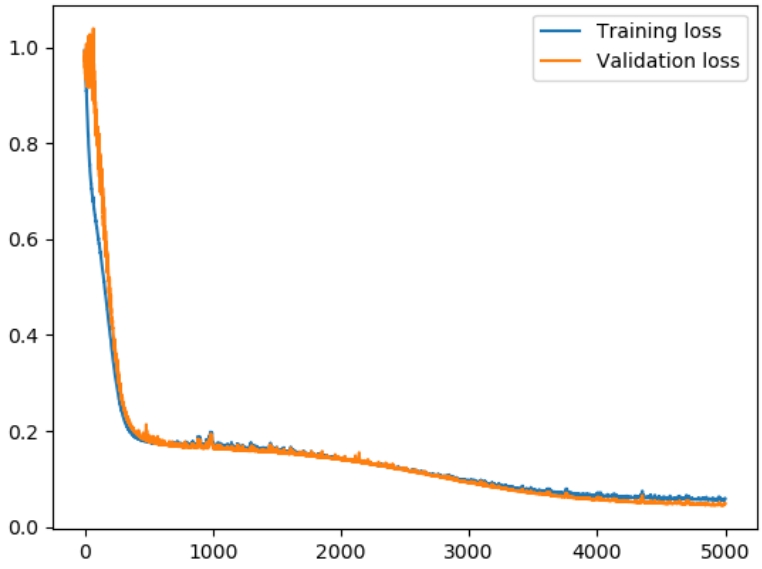


图 4.3 loss 曲线

图 4.4 给出由验证集数据所作出的预测结果与真实结果对比图。

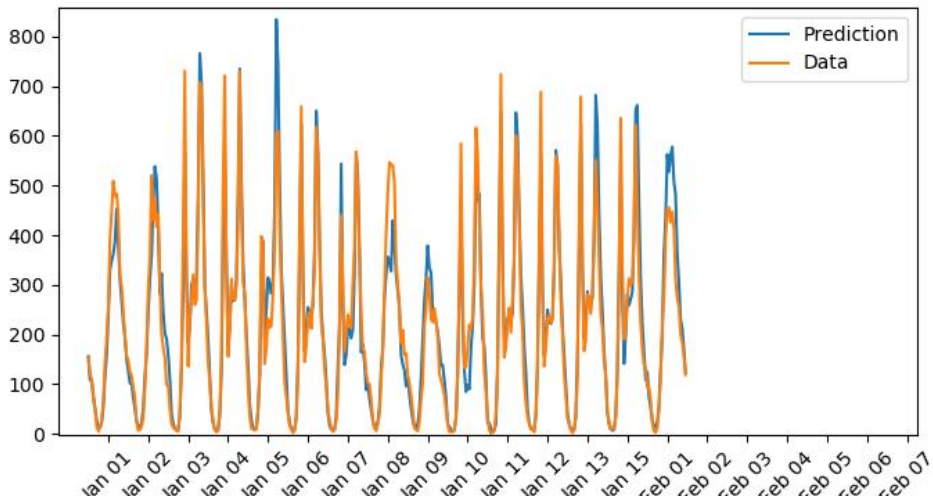
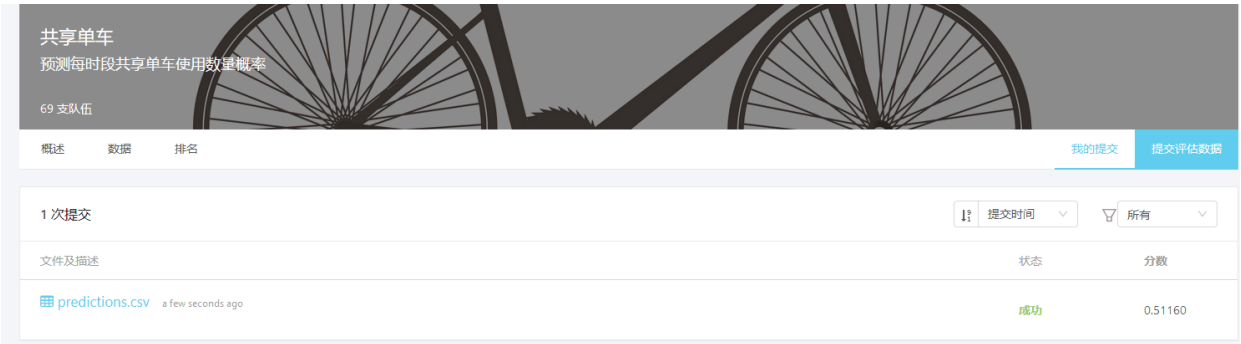


图 4.3 验证集中预测结果与真实数据对比

最终上传分数如下：



由于预测仅需要提交总用车人数，因此实际没有对未注册会员人数及注册会员人数进行预测。

在实验中，我们尝试了多种隐层神经元数目选取，发现只要该数值在满足$\sqrt{n+l}+\alpha$的范围内，对实验结果影响不大。其中，n 为输入层节点数，l 为输出层节点数，α取 1 到 10 之间的常数。此外，我们尝试了增加隐藏层的数目，可能由于该题处理数据比较简单，一层隐藏层的效果就已经较好，因此在收敛速度和准确率上没有明显改善。在迭代过程中，增大学习率可以加快迭代速度，但是 loss 振荡程度也会加剧，我们最终将学习率定为 0.25，隐层节点数设为 12。网络中所使用的激活函数为 Sigmoid 函数。

实验过程中将所给出的测试集数据分为测试集与验证集，将原数据最后 15 天的数据作为验证集进行模型实时验证。最终的 loss 如下图所示，可见预测效果还是较好的。

```
特征处理完成
神经网络
训练网络
Progress: 100.0% ... Training loss: 0.061 ... Validation loss: 0.051
```

5 小组分工

程序设计及编写：张嘉琛
程序调试及修改：张嘉琛
实验报告撰写：周颖、张嘉琛

6 实验小结及心得

本次实验让我们对最基础的 BP 神经网络有了更深的认识，也为之后进行深度学习的后续学习奠定了更坚实的基础。在此之前，虽然也编写过相应领域的程序，但或多或少有调用 tensorflow 或 pytorch 的函数，而不像这次实验一样，完全由自己构建整个模型。在这一过程中，我们对于该模型的数学原理的理解更加深刻与透彻。

实验最初运行时，loss 一直居高不下。经过检查，发现是没有利用日期信息。显然，单车使用人数与每天的时段是有着重要相关性的，忽略该特征就不可能得到正确的预测值。这也让我们深刻明白了合理选取特征的重要性。这个道理虽然浅显，但在实践中我们极有可能因为大意忽略一些重要特征，需要注意。

在改正了这一点后，我们重新训练模型，但发现得到的预测结果趋势突然出现异常，高峰期与低峰期正好相反。我们怀疑是模型结构或者预处理过程中出现了问题，因此进行了两天的查找，反复调试参数，更改隐层节点数、学习率、隐层数目，仔细核对程序中编写的矩阵维度，对预处理归一化也进行了调整实验，并查找大量资料，但无功而返。最终，我们抱着尝试的心态将处理完的训练集特征及测试集特征输出 csv 文件进行观察，发现我们可能在查看原始数据时进行了误操作，导致两者的格式不对应，使得哑变量处理中对小时编码无法匹配上（训练集中为 hour_00,hour_01,hour_02,…,hour_10,…,hour_23;而测试集为 hour_0,hour_1,hour_10,…,hour_23,hour_2,hour_3,…，因此正好导致预测结果高峰期与低峰期相反），而非模型本身或预处理的问题。在重新下载数据文件之后，该问题得到解决。虽然对浪费的两天时间非常痛惜，但也提醒了我们，数据集本身对于模型也是至关重要的，不可以轻易地忽视。

本次的作业真的给我们带来了许多感悟，不仅是对于原理和知识的理解，更有在实际编程中需要注意的要点。比如对于数据集的处理及特征选取问题，光

看理论是不会注意到这两个问题的，只有实践才能让我们有所感受。而在成功得到结果之后，虽然只是一个较容易的问题，我们也得到了简单的快乐，也更有兴趣去探索机器学习的知识。感谢赵老师在这一个学期中耐心细微的教学，让我们对模式相关的各种知识都有了一些初步的了解和认识。这些知识让我们触类旁通，在学习其他课程时也能更快更好地对其进行理解。

7 附录

1. 题目网址

https://www.lintcode.com/ai/bike-sharing/overview

2. 程序

```
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import pandas as pd
import datetime
import csv
import sys

# Sigmoid函数
def sigmoid(x):
    s = 1 / (1 + np / exp(-x))
    return s

#  均方误差
def MSE(y, Y):
    return np.mean((y - Y)**2)

# NN
#  一层隐藏层
class NeuralNetwork(object):
    def __init__( self, input_nodes, hidden_nodes, output_nodes, learning_rate ):
        #  各层节点数
        self.input_nodes = input_nodes
        self.hidden_nodes = hidden_nodes
        self.output_nodes = output_nodes

        #  学习率
        self.lr = learning_rate

        #  权重初始化
        self.weights_input_to_hidden = np.random.normal( 0.0, self.input_nodes**-0.5,
```

```
(self.input_nodes, self.hidden_nodes) )
            self.b_input_to_hidden = np.zeros(self.hidden_nodes)
            self.weights_hidden_to_output = np.random.normal( 0.0, self.hidden_nodes**-0.5,
(self.hidden_nodes, self.output_nodes) )
            self.b_hidden_to_output = np.zeros(self.output_nodes)

            # 激活函数Sigmoid
            self.activation_function = lambda x : sigmoid(x)

        # 训练网络
        def train(self, features, targets):
            # 初始化
            n_record = features.shape[0]       # 特征数
            delta_weights_i_h = np.zeros( self.weights_input_to_hidden.shape )   # 输入至隐层
            delta_b_i_h = np.zeros(self.b_input_to_hidden.shape)
            delta_weights_h_o = np.zeros( self.weights_hidden_to_output.shape ) # 隐层至输出
            delta_b_h_o = np.zeros(self.b_hidden_to_output.shape)



            for X, y in zip( features, targets ):
                # 正向传播
                # 输入层
                hidden_inputs = np.dot( X, self.weights_input_to_hidden ) +
self.b_input_to_hidden
                hidden_outputs = self.activation_function( hidden_inputs )

                # 输出层
                final_inputs = np.dot( hidden_outputs, self.weights_hidden_to_output ) +
self.b_hidden_to_output
                final_outputs = final_inputs

                # BP
                error = y - final_outputs
                output_error_term = error # error * 1

                hidden_error = np.dot( self.weights_hidden_to_output, output_error_term )
                hidden_error_term = hidden_error * hidden_outputs * (1 - hidden_outputs) #
f'(hidden_input)

                # delta_weight
                delta_weights_i_h += hidden_error_term * X[:,None]
                delta_b_i_h += hidden_error_term

                delta_weights_h_o += output_error_term * hidden_outputs[:,None]
```

```
                delta_b_h_o += output_error_term

            # 权重更新
            self.weights_input_to_hidden += self.lr * delta_weights_i_h/n_record
            self.b_input_to_hidden += self.lr * delta_b_i_h/n_record
            self.weights_hidden_to_output += self.lr * delta_weights_h_o/n_record
            self.b_hidden_to_output = self.lr * delta_b_h_o/n_record

        # 运行网络
        def run(self, features):
            hidden_inputs = np.dot( features, self.weights_input_to_hidden ) +
self.b_input_to_hidden
            hidden_output = self.activation_function( hidden_inputs )

            final_inputs = np.dot( hidden_output, self.weights_hidden_to_output ) +
self.b_hidden_to_output
            final_outputs = final_inputs

            return final_outputs

print('start')
# data
data_path = './data/train.csv'
train = pd.read_csv('./data/train.csv')
test = pd.read_csv('./data/test.csv')

# 去除离群数据
outliers_removed = train[np.abs(train['count'] - train['count'].mean()) <= (3 *
train['count'].std())]
train = outliers_removed.reset_index(drop = True)

# 日期分为年月时备用
train['hour'] = train.datetime.apply(lambda x: x.split()[1].split(':')[0])
train['month'] = train.datetime.apply(lambda x: x.split('/')[1])
train['year'] = train.datetime.apply(lambda x: x.split('/')[0])
test['hour'] = test.datetime.apply(lambda x: x.split()[1].split(':')[0])
test['month'] = test.datetime.apply(lambda x: x.split('/')[1])
test['year'] = test.datetime.apply(lambda x: x.split('/')[0])

# 处理哑变量
dummies_fields = ['season', 'weather', 'hour', 'month', 'year']
for each in dummies_fields:
    dummies = pd.get_dummies(train.loc[:, each], prefix=each)
    train = pd.concat([train, dummies], axis=1)
```

```
        dummies = pd.get_dummies(test.loc[:, each], prefix=each)
        test = pd.concat([test, dummies], axis=1)

# 删去处理前数据
# 温度与体感温度基本成线性相关，故删去体感温度atemp
drop_fields = ['season', 'weather', 'atemp', 'datetime', 'hour', 'month', 'year']
train_data = train.drop(drop_fields, axis=1)
test_data = test.drop(drop_fields, axis=1)

# normalization
# 提取train set与test set的温度湿度风速部分得到均值与标准差
target_fields = ['temp', 'humidity', 'windspeed']
train_features = train_data.loc[:, target_fields]
test_features = test_data.loc[:, target_fields]
total_data = pd.concat( [train_features, test_features], axis=0 )
total_data = total_data.reset_index(drop = True)
train_features = {}
test_features = {}
# 对这三类特征进行归一化处理
standard_field = ['temp', 'humidity', 'windspeed']
scaled_feature = {}
for each in standard_field:
        mean = total_data[each].mean()
        std = total_data[each].std()
        scaled_feature[each] = [mean, std]
        train_data.loc[:, each] = (train_data.loc[:, each] - mean) / std
        test_data.loc[:, each] = (test_data.loc[:, each] - mean) / std

# 对train_set中的结果值进行处理
# 为确保不为负，在归一化前先取对数
standard_field = ['casual', 'registered', 'count']
for each in standard_field:
        train_data.loc[:, each] = train_data.loc[:, each] + np.ones(train_data.loc[:, each].shape)
        train_data.loc[:, each] = np.log(train_data.loc[:, each] )
        train_mean = train_data[each].mean()
        train_std = train_data[each].std()
        scaled_feature[each] = [train_mean, train_std]
        train_data.loc[:, each] = (train_data.loc[:, each] - train_mean) / train_std

# 分割特征与count，取最后一部分数据作为验证集
target_fields = ['casual', 'registered', 'count']
features = train_data.drop(target_fields, axis=1)
targets = train_data.loc[:, target_fields]
val_features = features[-15*24:]
```

```
val_targets = targets[-15*24:]
train_features = features[:-15*24]
train_targets = targets[:-15*24]

# 记录处理完的特征与data，作为csv文件输出
write = pd.DataFrame(data=train_targets)
write.to_csv('./train_targets.csv',encoding='gbk')
write = pd.DataFrame(data=train_features)
write.to_csv('./train_features.csv',encoding='gbk')
write = pd.DataFrame(data=test_data)
write.to_csv('./test_features.csv',encoding='gbk')
print('特征处理完成')

# 对网络进行训练
print('训练网络')
# 设置网络参数
iterations = 5000
learning_rate = 0.25
hidden_nodes = 12
output_nodes = 1

N_i = train_features.shape[1]
network = NeuralNetwork(N_i, hidden_nodes, output_nodes, learning_rate)

losses = {'train':[], 'validation':[]}
for ii in range(iterations):
    batch = np.random.choice(train_features.index, size=int(train_features.shape[0]/10))
    X = train_features.iloc[batch].values
    y = train_targets.iloc[batch]['count']

    network.train(X, y)

    # 实时输出loss并记录数据以便后续作图
    train_loss = MSE(network.run(train_features).T, train_targets['count'].values)
    val_loss = MSE(network.run(val_features).T, val_targets['count'].values)

    sys.stdout.write("\rProgress: {:2.1f}".format(100 * ii/float(iterations)) \
                     + "% ... Training loss: " + str(train_loss)[:5] \
                     + " ... Validation loss: " + str(val_loss)[:5])
    sys.stdout.flush()

    losses['train'].append(train_loss)
    losses['validation'].append(val_loss)
```

```
# 作loss曲线
plt.plot(losses['train'], label='Training loss')
plt.plot(losses['validation'], label='Validation loss')
plt.legend()
plt.show()

# 作验证集上的预测结果与实际数据对比曲线
ig, ax = plt.subplots(figsize=(8,4))
mean, std = scaled_feature['count']
predictions = network.run(val_features).T*std + mean
predictions = np.exp(predictions) - 1

ax.plot(predictions[0], label='Prediction')
ax.plot((np.exp(val_targets['count']*std + mean)-1).values, label='Data')
ax.set_xlim(right=len(predictions))
ax.legend()

dates = pd.to_datetime(train.loc[test_data.index]['datetime'])
dates = dates.apply(lambda d: d.strftime('%b %d'))
ax.set_xticks(np.arange(len(dates))[12::24])
_ = ax.set_xticklabels(dates[12::24], rotation=45)

plt.show()

# 使用训练好的网络预测结果
print('预测结果')
fig, ax = plt.subplots(figsize=(8,4))
test_features = test_data
mean, std = scaled_feature['count']
test_targets = network.run(test_features).T * std + mean
test_targets = test_targets * std + mean
test_targets = np.exp(test_targets) - 1
test_targets = test_targets.astype(np.int16)       #取整输出
print(test_targets)

# 将预测结果写为csv文件
print('csv文件输出')
with    open('./prediction.csv', 'w') as csvFile:
        writer = csv.writer(csvFile)
        writer.writerows(test_targets)

print('end')
```

8 附件

附件 1 train.csv：训练集

附件 2 test.csv：测试集

附件 3 cycle_final.py：分析及预测程序

附件 4 submission.csv：最终提交文件

附件 5 train_features.csv、test_features.csv：预处理后特征文件

附件 6 train_targets.csv：测试集 count 预处理结果