实验 3：使用神经网络预测共享单车的使用量

一、　实验内容

实验内容：设计一个有实用价值的人工神经网络，预测未来某地区租赁单车的使用情况。修改预测模型对单车数量进行分类。

实验环境：Python-3.11;PyTorch-cu121;CUDA-12.1;VS code/JupyterNotebook

二、　实验过程和结果

1. 预处理实验数据

1.1 读取数据

```
#神经网络预测共享单车使用
#!wget http://labfile.oss.aliyuncs.com/courses/1073/bike-sharing-dataset.zip
#!unzip bike-sharing-dataset.zip
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
from torch.autograd import Variable
import torch.optim as optim
%matplotlib inline
```

该段代码首先下载了共享单车的数据集并解压，引入 numpy 库进行数值计算，pandas 库用于操作读取数据集文件，matplotlib 用于绘图，torch 用于深度学习，torch.grad.Variable 用于构建自动微分变量，torch.optim 用于对神经网络参数进行优化。

```
data_path='bike-sharing-dataset/hour.csv'
rides=pd.read_csv(data_path)
rides.head()
```

Python

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0.0	3	13	16
1	2	2011-01-01	1	0	1	1	0	6	0	1	0.22	0.2727	0.80	0.0	8	32	40
2	3	2011-01-01	1	0	1	2	0	6	0	1	0.22	0.2727	0.80	0.0	5	27	32
3	4	2011-01-01	1	0	1	3	0	6	0	1	0.24	0.2879	0.75	0.0	3	10	13
4	5	2011-01-01	1	0	1	4	0	6	0	1	0.24	0.2879	0.75	0.0	0	1	1

该段代码读取数据集文件，.heads 方法可以显示数据集前几条数据的样式。

1.2 对类型变量的处理

```
#对类型变量处理
dummy_fields=['season','weathersit','mnth','hr','weekday']
for each in dummy_fields:
    dummies=pd.get_dummies(rides[each],prefix=each,drop_first=0)
    rides=pd.concat([rides,dummies],axis=1)

#删除不需要的特征
fields_to_drop=['season','weathersit','mnth','hr','weekday','instant','dteday','atemp','workingday']
data=rides.drop(fields_to_drop,axis=1)
data.head()
```

Python

	yr	holiday	temp	hum	windspeed	casual	registered	cnt	season_1	season_2	...	hr_21	hr_22	hr_23	weekday_0	weekday_1	weekday_2	weekday
0	0	0	0.24	0.81	0.0	3	13	16	True	False	...	False	False	False	False	False	False	Fal
1	0	0	0.22	0.80	0.0	8	32	40	True	False	...	False	False	False	False	False	False	Fal
2	0	0	0.22	0.80	0.0	5	27	32	True	False	...	False	False	False	False	False	False	Fal
3	0	0	0.24	0.75	0.0	3	10	13	True	False	...	False	False	False	False	False	False	Fal
4	0	0	0.24	0.75	0.0	0	1	1	True	False	...	False	False	False	False	False	False	Fal

5 rows × 59 columns

对于类型变量的数据，使用数值区分会影响效果，应该采用 onehot 编码，某一个特征有多少个取值就将其分割成几个新特征，针对旧特征的取值对新特

征进行赋值 1 或 0，用 pandas 库中的 get_dummies 方法可对数据集中指定特征的列进行 onehot 编码，concat 方法将两个数据集按指定的维度拼接在一起，再通过 drop 方法将旧的特征和不需要用到的特征进行剔除。

1.3 对数值类型变量进行标准化

```
#标准化特征
quant_features=['cnt','temp','hum','windspeed']
scaled_features={}
for each in quant_features:
    mean,std=data[each].mean(),data[each].std()
    scaled_features[each]=[mean,std]
    data.loc[:,each]=(data[each]-mean)/std
```

对于可以连续取值的数值类型特征进行标准化，使其按均值和标准差映射到[-1,1]区间。标准化可以使不同取值范围的特征处于同一取值区间，之间的地位平等。

1.4 将数据集进行分割

```
#数据集分割
test_data=data[-21*24:]
train_data=data[:-21*24]
print('训练数据',len(test_data),'测试数据',len(train_data))
target_fields=['cnt','casual','registered']
features,targets=train_data.drop(target_fields,axis=1),train_data[target_fields]
test_features,test_targets=test_data.drop(target_fields,axis=1),test_data[target_fields]

#数据从pandas dataframe转换为numpy
X=features.values
Y=targets['cnt'].values
Y=Y.astype(float)

Y=np.reshape(Y,[len(Y),1])
losses=[]

features.head()
print(X)
```

```
训练数据 504 测试数据 16875
[[0 0 -1.3346091869412673 ... False False True]
 [0 0 -1.438475009903406 ... False False True]
 [0 0 -1.438475009903406 ... False False True]
 ...
 [1 0 0.11951233452867807 ... False False False]
 [1 0 -0.19208513435773875 ... False False False]
 [1 0 -0.19208513435773875 ... False False False]]
```

数据集分割为训练集和测试集，测试集取出了倒数 21*24 列，每天按小时记录 24 条数据，相当于取出最后 21 天的数据用于测试。然后再分别将输入的特征和标准输出答案取出为 X 和 Y，以及测试集的输入特征和输出答案。

2. 构建神经网络模型

2.1 手动编写用 Tensor 运算的人工神经网络

```python
#手动搭建神经网络
#定义神经网络架构，features. shape[1]个输入层单元，10个隐含层，1个输出层
input_size = features.shape[1] #输入层单元个数
hidden_size = 10 #隐含层单元个数
output_size = 1 #输出层单元个数
batch_size = 128 #每隔batch的记录数
weights1 = Variable(torch.randn([input_size, hidden_size]), requires_grad = True)#第一到二层权重
biases1 = Variable(torch.randn( [hidden_size]),requires_grad = True)#隐含层偏置
weights2 = Variable(torch.randn([hidden_size, output_size]), requires_grad = True)#隐含层到输出层权重
def neu(x):
    #计算隐层输出
    #X:128*inputsize;weights1:inputsize*hiddensize=128*hidden=>sigmoid
    hidden=x.mm(weights1)+biases1.expand(x.size()[0],hidden_size)
    hidden=torch.sigmoid(hidden)

    output=hidden.mm(weights2)
    return output

def cost(x,y):
    return torch.mean((x-y)**2)

def zero_grad():
    if weights1.grad!=None and biases1.grad!=None and weights2.grad!=None:
        weights1.grad.zero_()
        weights2.grad.zero_()
        biases1.grad.zero_()

def optimizer_step(learning_rate):
    weights1.grad.add_(-learning_rate* weights1.grad.data)
    weights2.grad.add_(-learning_rate* weights2.grad.data)
    biases1.data.add_(-learning_rate* biases1.grad.data)
```

手动编写神经网络模型按照机器学习的四要素：数据、模型、目标函数、最优化算法。模型采用前向全连接的神经网络，输入层数量等于输入特征数量，用 features.shape[1]取出特征的列数，隐层神经元 10 个，输出神经元 1 个，因为输出值只有一个预测数量；目标函数仍然采用均方差函数；最优化算法采用反向梯度传播，梯度下降算法。此外还要考虑数据集比较大需要将数据集分成大小相等的若干撮，长度为 128 条数据。

自定义 4 个函数，neu 为神经元模型用于计算预测值，其中先进行加权求和再通过激活函数 sigmoid，层层之间相互矩阵相乘得到输出值；cost 函数为损失函数采用均方差函数；zero_grad 函数首先判断参数梯度是否为空，再对梯度进行清零；optimizer_step 函数传入参数为学习率，对模型进行梯度下降计算，参数修改。

2.2 调用 PyTorch 现成的函数，构建序列化的神经网络

```python
#3.2定义神经网络架构，features. shape[1]个输入层单元，10个隐含层，1个输出层
input_size = features.shape[1]
hidden_size = 10
output_size = 1
batch_size = 128
neu = torch. nn. Sequential(
    torch. nn.Linear(input_size,hidden_size),
    torch. nn.Sigmoid(),
    torch. nn.Linear(hidden_size,output_size),
    )
cost=torch.nn.MSELoss()
optimizer=torch.optim.SGD(neu.parameters(),lr=0.01)
```

调用 pytorch 现成的函数要方便许多，直接设定输入、隐层、输出层神经元数量，Sequential 函数将神经元间的操作按顺序包裹，首先是输入到隐层的线性变化，经过 sigmoid 激活函数，再由隐层到输出的线性变化，损失函数用 MSELoss 均方差损失函数，优化算法为 SGD 随机梯度下降，对 neu 神经网络所有参数优化，学习率为 0.01。

2.3 数据的分批次处理

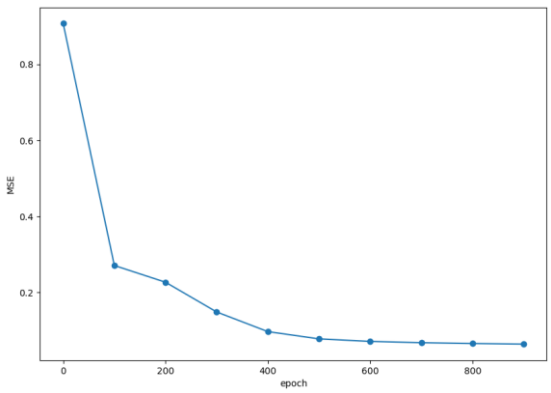
```
#3.3数据分批次处理
#神经网络训练循环
losses=[]
for i in range(1000):
    batch_loss=[]
    for start in range(0,len(X),batch_size):
        end = start +batch_size if start+batch_size<len(X) else len(X)
        X=X.astype(float)
        xx=Variable(torch.FloatTensor((X[start:end])))
        yy=Variable(torch.FloatTensor(Y[start:end]))
        predict=neu(xx)
        loss=cost(predict,yy)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        batch_loss.append(loss.data.numpy())

    if i%100==0:
        losses.append(np.mean(batch_loss))
        print(i,np.mean(batch_loss))
```

```
0 0.9075402
100 0.27027863
200 0.2262065
300 0.14788076
400 0.09620182
500 0.07708397
600 0.07039443
700 0.067039385
800 0.064887576
900 0.06332913
```

然后对神经网络进行分组训练，设置训练次数为 1000 次，循环中对数据集进行分组，每 128 条数据一组传入网络进行一次反向传播和随机梯度下降优化，每 128 条数据得到一条损失值数据追加进 batch_loss 列表中，数据集中每撮数据训练完成后，进行下一轮训练。训练次数为 100 整数倍时输出此时的损失值，并将其追加进列表 losses 中，用于最终绘图。

```
fig = plt.figure(figsize=(10,7))
plt.plot(np.arange(len(losses))*100,losses,'o-')
plt.xlabel('epoch')
plt.ylabel('MSE')
plt.show()
```

绘图将训练次数与对应的损失值绘出，其结果如图，呈现下降趋势，说明随机梯度下降法训练是有效的，损失值不断变小。



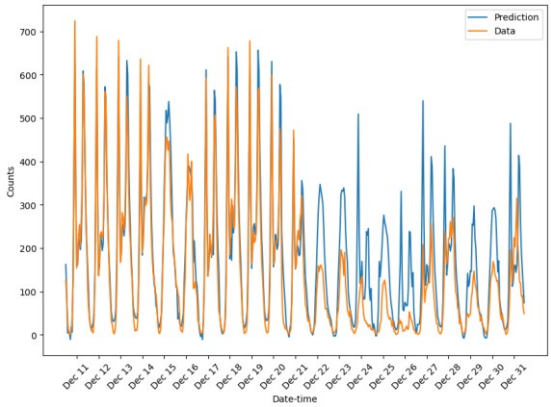
3. 测试网络

3.1 使用测试数据集测试网络

```python
#4.1测试数据测试
targets=test_targets['cnt']#读取测试集的cnt数值
targets=targets.values.reshape([len(targets),1])#将数据转换成合适的tensor形
targets=targets.astype(float)#保证数据为实数
#将属性和预测变量包裹在ariable型变量中
x=Variable(torch.FloatTensor(test_features.values.astype(float)))
y=Variable(torch.FloatTensor(targets))
#用神经网络进行预测
predict=neu(x)
predict=predict.data.numpy()
fig,ax=plt.subplots(figsize=(10,7))
mean,std=scaled_features['cnt']
ax.plot(predict * std+mean,label='Prediction',linestyle ='-')
ax.plot(targets * std+mean,label='Data',linestyle ='-')
ax.legend()
ax.set_xlabel('Date-time')
ax.set_ylabel('Counts')
#对横坐标轴进行标注
dates=pd.to_datetime(rides.loc[test_data.index]['dteday'])
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)
```

Python

对测试集数据进行预测，绘制其标准答案和预测结果的图，将测试集的特征输入神经网络得到预测值，将时间作横坐标，预测值和真值作纵坐标绘入图中。结果如图所示，12 月 21 日前的预测精度较之后好，出现较大偏差的原因可能是 12 月 25 日前后为圣诞节假期，而训练集中没有这部分训练。



4. 修改模型完成分类问题

分类问题和预测问题基本结构相似，区别在于输出层神经元数量，分类问题有多少种分类结果就有几个输出值，而分类问题的输出经过 softmax 函数，各输出值得和为 1，输出的值代表取该种类的概率。因此损失函数不适用均方差损失函数，而采用交叉熵损失函数。

4.1 数据处理

```python
import time
import numpy as np
import torch
import pandas as pd
from torch.utils.data import DataLoader,TensorDataset
import matplotlib.pyplot as plt
#基本参数
batch_size = 128
epoch_num = 2000
data_path='bike-sharing-dataset/hour.csv'
```

✓ 0.0s

```python
#数据处理
rides = pd.read_csv(data_path)
# 形成one-hot编码
dummy_fields = ['season','weathersit','mnth','hr','weekday']
for each in dummy_fields:
    dummies = pd.get_dummies(rides[each], prefix=each,drop_first=False)
    rides = pd.concat([rides,dummies], axis=1)
#把原有的类型特征去掉，将一些不相关的特征去掉
fields_to_drop = ['instant', 'dteday', 'season','weathersit','weekday', 'atemp', 'mnth','workingday','hr']
data = rides.drop(fields_to_drop,axis=1)
#调整所有的特征，标准化处理
quant_features = ['cnt','temp','hum','windspeed']
scaled_features = {}
for each in quant_features:
    mean,std = data[each].mean(), data[each].std()
    scaled_features[each] = [mean,std]
    data.loc[:,each] = (data[each] - mean) / std
```

✓ 0.0s　　Python

4.2 数据集分割

```python
#将所有的数据集分为测试集和训练集，我们以后21天数据一共21*24个数据点作为测试集，其它是训练集
test_data = data[-21 * 24:]
train_data = data[:-21 * 24]
print('训练数据：',len(train_data),'测试数据：',len(test_data))
#将我们的数据列分为特征列和目标列
target_fields = ['cnt','casual','registered']
features,targets = train_data.drop(target_fields,axis=1), train_data[target_fields]
test_features,test_targets = test_data.drop(target_fields, axis=1), test_data[target_fields]
#将数据从pandas dataframe转换为numpy,在转化成tensor
X = features.values
Y = targets['cnt'].values
Y = Y.astype(float)
Y = np.reshape(Y,[len(Y),1])
X_t = test_features.values
Y_t = test_targets['cnt'].values
Y_t = Y_t.astype(float)
Y_t = np.reshape(Y_t,[len(Y_t),1])

X = torch.FloatTensor(X.astype(float))
Y = torch.FloatTensor(Y)
X_t = torch.FloatTensor(X_t.astype(float))
Y_c = torch.sign(torch.sign(Y) + 1).flatten().long()#将Y变为0，1分布,交叉熵函数的target数据只能是一维长整型张量
Y_ct = np.sign(np.sign(Y_t) + 1)#用于计算分类准确率

input_size2 = features.shape[1]
hidden_size2 = 10
output_size2 = 2
```

✓ 0.0s　　Python

上述步骤和预测模型差别不大，最后三行设置了分类模型各层神经元的个数，输出神经元 2 个。

4.3 构建模型

```python
neu2 = torch.nn.Sequential(
    torch.nn.Linear(input_size2,hidden_size2),
    torch.nn.Sigmoid(),
    torch.nn.Linear(hidden_size2,output_size2),
)
cost2 = torch.nn.CrossEntropyLoss()
optimizer2 = torch.optim.SGD(params=neu2.parameters(),lr=0.01)
```

损失函数为交叉熵损失函数，学习率为 0.01

4.4 训练并绘制动态图

```python
#训练模型
def train2(epoch_num,X, Y, is_cuda=0):
    if is_cuda == 1:
        X, Y = X.cuda(),Y.cuda()
        neu2.cuda()
    for epoch in range(epoch_num):
#每128个样本点被划分为一个撮，在循环的时候一批一批地读取# start和end分别是提取一个batch数据的起始和终止下标
        for start in range(0,len(X),batch_size):
            end = start + batch_size if start + batch_size < len(X) else len(X)
            xx= X[start:end]
            yy = Y_c[start:end]
            predict = neu2(xx)
            loss2 = cost2(predict,yy)#注意这里要输入一个一维的数据Y
            optimizer2.zero_grad()
            loss2.backward()
            optimizer2.step()
        if epoch+1 in [1,10,50,100,500,1000,1500,2000] :#epoch % 100 == 99:
            test_plot2(epoch)
```

✓ 0.0s　　Python

这里设置可以将张量传入 GPU 中使用，训练过程基本相同，最后两行的 if 语

句设置在训练次数一定时输出训练的分类结果精度以及绘图，绘图代码如下。

```python
#绘制训练动态图
def test_plot2(epoch):
    plt.cla()
    #prediction
    predict=neu2(X_t)
    _,c=torch.max(predict,dim=1)
    c=c.data.numpy().reshape(-1,1)
    correct = (c == Y_ct).sum()
    accuracy = correct / 504
    print('epoch={}\tAccuracy : {:.3%}'.format(epoch + 1,accuracy))#将后21天的预测数据与真实数据画在一起并比较

    mean, std = scaled_features['cnt']
    plt.scatter(range(504),Y_t * std + mean,c=c)
    plt.plot([-30,550],[mean,mean], linewidth=2)
    plt.xlabel('Date-time')
    plt.ylabel('Counts')

    #对横坐标轴进行标注
    dates = pd.to_datetime(rides.loc[test_data.index]['dteday'])
    dates = dates.apply(lambda d: d.strftime('%b %d'))
    plt.xticks(np.arange(len(dates))[12::24], dates[12::24],rotation=90)
    plt.text(350, 550, 'epoch={} \nAccuracy : {:.3%}'.format(epoch + 1,accuracy),fontsize=10)
    if epoch == epoch_num - 1:
        t2 = time.time()
        print('time of train:',t2 - t1)
        plt.show()
    else:
        plt.pause(0.001)
```

0.0s　　Python

前一部分代码对预测分类输出与标准值进行对比并计数求精度。第二段进行去标准化后绘图输出，对图例和坐标轴进行设置，显示训练次数和精度。

```python
t1=time.time()
train2(epoch_num,X,Y,is_cuda=0)
```

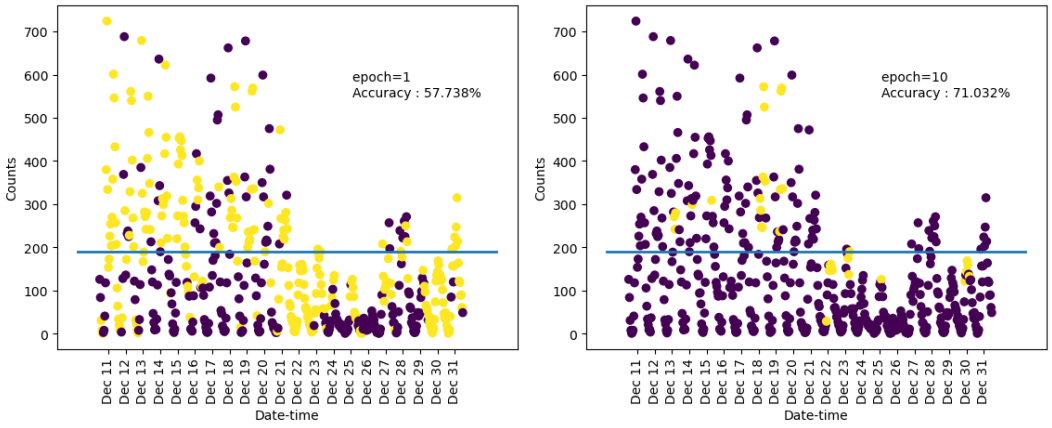
✓ 2m 25.5s

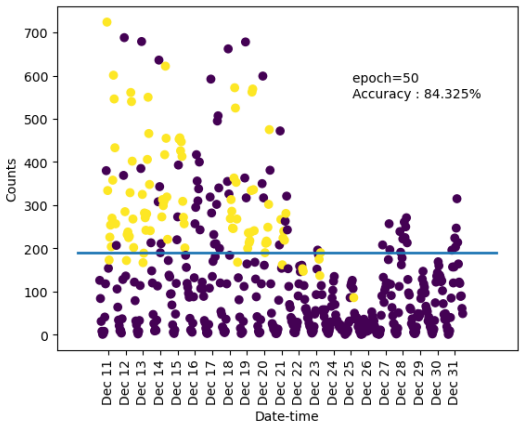
这段代码首先记录开始时间，然后对训练函数调用，程序开始训练神经网络，并将结果绘图输出。

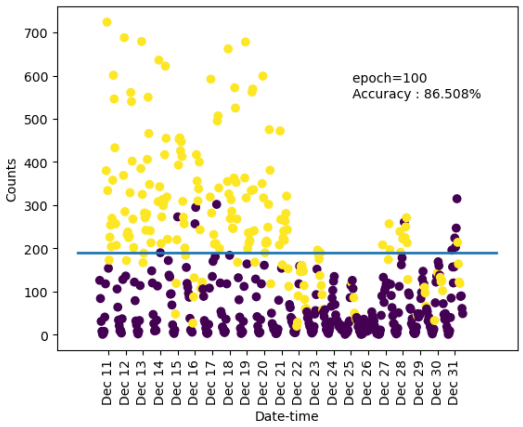
```
epoch=2000      Accuracy : 86.111%
time of train: 145.48251223564148
```

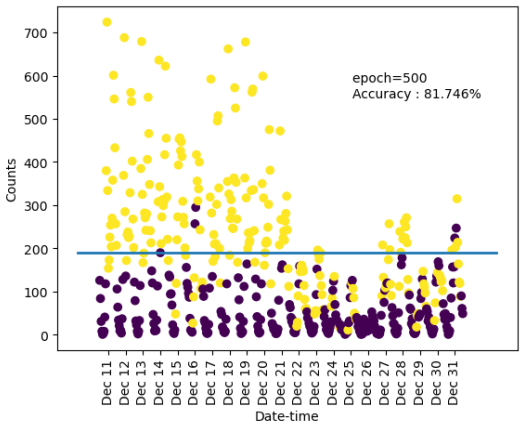
总用时 145.48 秒

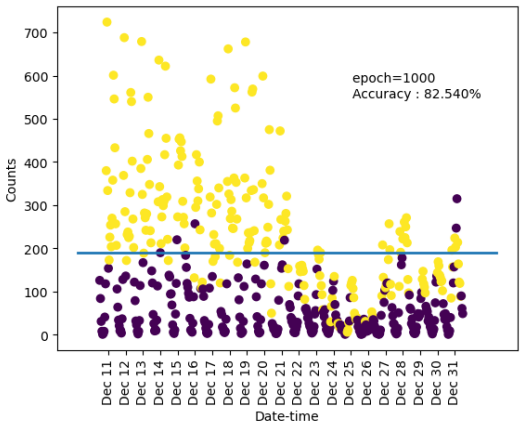
训练结果如图所示，紫色和黄色两类按照均值线逐渐分离，分类模型的准确度随训练轮数逐渐上升。

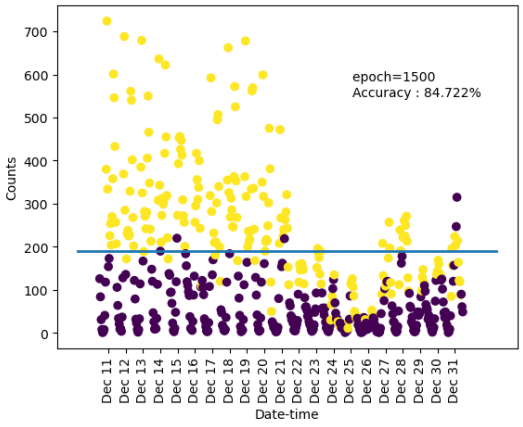


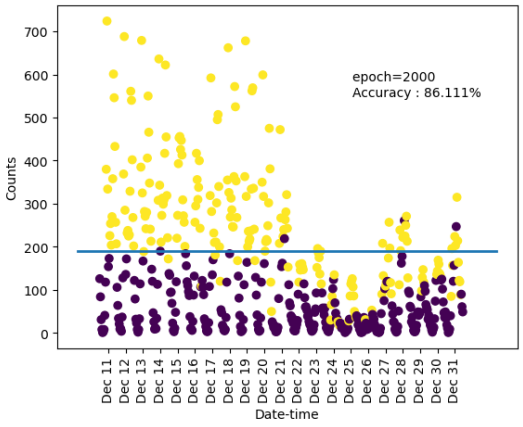

epoch=50
Accuracy : 84.325%
Counts
Date-time


epoch=100
Accuracy : 86.508%
Counts
Date-time


epoch=500
Accuracy : 81.746%
Counts
Date-time


epoch=1000
Accuracy : 82.540%
Counts
Date-time


epoch=1500
Accuracy : 84.722%
Counts
Date-time


epoch=2000
Accuracy : 86.111%
Counts
Date-time

三、　心得体会

本次实验代码长，难度较高。实验针对前向全连接的神经网络从数据集的处理到模型搭建和参数优化进行了全面的练习，针对共享单车数量预测和共享单车数量分类两类问题，分别用手动编写函数和用 pytorch 现成函数两种方式搭建神经网络。编程时牢记机器学习的四个要素：数据、模型、目标函数、最优化算法，就可以清楚每一段代码的大致作用。本次实验学习了使用神经网络模型解决实际问题的方法。了解数据预处理的基本方法。熟悉搭建神经网络的一般流程，测试神经网络、简单分析神经网络的方法。

实验过程中遇到很多程序报错的难题，其中大部分是由于代码抄写错误造成的，可以根据报错提示进行改正。例如在最开始导入 pandas 库时误把 pandas 敲成 panda，两个库并不相同报错没有安装库，安装 panda 库后依然需要安装其他依赖的库，再次对照代码发现错误，经过查阅发现很多初学者会将两个库混淆。此外还有在对输入特征值进行张量化时出现的类型不能转换的问题，经过查阅需要先将特征值转换成标准的类型，X.astype(float)使用该语句解决类型转换的问题。在使用 torch 搭建神经网络时，误把隐层和输出层神经元数量写反，出现在计算过程中矩阵乘法数量不匹配的情况。在完成分类问题时绘图的横坐标标签重叠，以及精确度标签太大不美观，通过添 rotation=90 的参数使坐标轴标签旋转 90 度不重叠，改变显示文字的位置和字体大小使输出的绘制图像更加美观。