

研究生《深度学习》课程

实验报告

|  |  |
| --- | --- |
| **实验名称：** | **出租车流量预测** |
| **姓 名：** |  |
| **学 号：** |  |
| **日 期：** |  |

## 一、实验内容

**任务概述：**

车流量预测任务是一个回归任务，旨在根据区域历史的车流量情况来预测其未来某一段时间的车流量情况 。使用的数据为纽约市出租车流量数据。输入为纽约市各区域的历史车流量时间序列，输出为对应各区域的未来车流量的预测值。

**任务要求：**

要求以提高在测试集上的效果为目标，自己根据数据特点及需要进行数据预处 理以及模型设计，本任务不对模型的设计和选择进行限制，但根据数据特点可结合CNN与 RNN的特点进行模型设计。

本任务要求各位同学首先将数据进行归一化处理并对数据进行滑动窗口划分，具体划分要求为使用历史6个时间步预测未来一个时间步，模型的输出应该为纽约200个区域的各区域未来半小时的出入流量。在计算评价指标MAE和RMSE时应将数据反归一化后再计算。

实验报告中需要包含但不限于数据归一化的过程介绍、滑动窗口划分的过程介绍、模型图、模型中各部分的作用介绍或者使用理由、超参数设置、训练过程中各指标变化、实验结果分析等。

## 二、实验设计

本实验采用了CNN+GRU模型，用GRU模型替代RNN模型，是为了提高模型效果。整个模型的超参数设置为batch\_size=128，lr=0.0001，epochs=100。模型先进行CNN训练，再进行GRU训练，接着用全连接层，最后输出。具体流程和各种参数设计及原因如下：

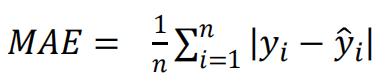
首先，加载数据，对数据进行滑动窗口划分，因为题目要求使用历史6个时间步预测未来一个时间步，所以滑动窗口window\_size设置为7。

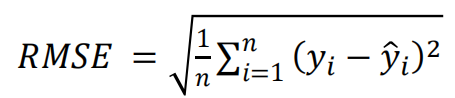
其次，用一层CNN卷积网络提取数据的特征，输入通道数in\_channels=6，输出通道数out\_channels=6，卷积核大小kernel\_size=1，经过卷积后进行平均池化，激活，激活选用的是ReLU函数。

然后，用GRU模型进行数据中时序信息以及语义信息的挖掘，隐藏层hidden\_size=32，输入特征数input\_size=1，循环网络层数为1。

最后得到的结果进入全连接层，从而得到未来半个小时出入的车流量。

本任务以平均绝对误差(MAE)、均方根误差(RMSE) 作为计算评测指标：





GRU（Gate Recurrent Unit）是循环神经网络RNN的一种。和LSTM一样，也是为了解决长期记忆和反向传播中的梯度等问题而提出来的。GRU模型引入两个门函数，分别是更新门和重置门。GRU模型计算公式如图2.1所示，其中，和分别表示更新门和重置门。

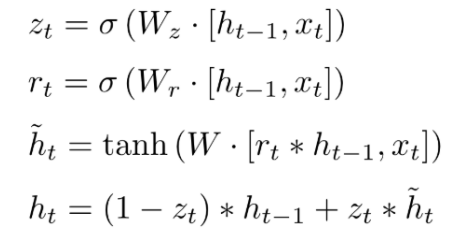


图2.1 GRU模型计算公式

GRU模型结构如图2.2所示。

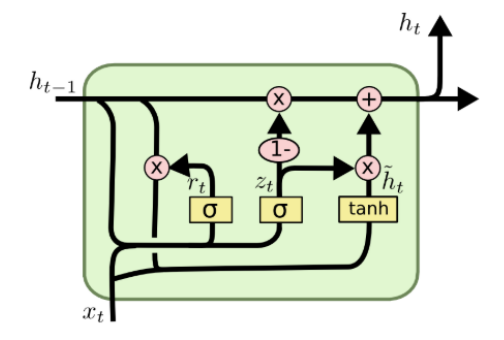


图2.2 GRU模型结构

## 三、实验环境及实验数据集

**实验环境：**Python3.7 + Anaconda + PyTorch + Jupyter Notebook

**实验数据集：**纽约出租车流量数据集

**数据集说明：**

纽约出租车流量数据集，时间跨度为从2015年1月1日到2015年3月1日。数据处理成为网格流量数据时的时间间隔设定为30分钟。后20天数据被划定为测试集，其余数据为训练集。

**数据集使用方式：**

volume\_train=np.load(open(train\_path, "rb"))["volume"]

volume\_test=np.load(open(test\_path, "rb"))["volume"]

**数据格式说明：**

以训练集为例，其shape=(1920\*10\*20\*2) 代表有1920个时间段，10\*20个区域，2个特征分别为区域的入流量与出流量。

对处理后的数据集可视化结果如图3.1所示。

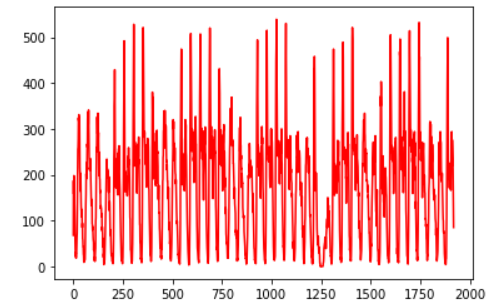


图3.1 数据集可视化结果

## 四、实验过程

#### 4.1 数据处理

数据预处理

1. volume\_train = np.load(open(train\_path, "rb"))["volume"]
2. volume\_test = np.load(open(test\_path, "rb"))["volume"]
3. volume\_train = np.sum(volume\_train.reshape(1920, 200, 2, 1), axis=2)
4. volume\_test = np.sum(volume\_test.reshape(960, 200, 2, 1), axis=2)

数据归一化

1. # min-max归一化 将结果映射在[0, 1]
2. dmin, dmax = volume\_train.min(), volume\_train.max()
3. volume\_train = (volume\_train - dmin) / (dmax - dmin)
4. dmin, dmax = volume\_test.min(), volume\_test.max()
5. volume\_test = (volume\_test - dmin) / (dmax - dmin)

定义固定长度滑动窗口

1. def sliding\_window(seq, window\_size):
2. result = []
3. for i in range(len(seq) - window\_size):
4. result.append(seq[i: i + window\_size])
5. return result

加载数据

1. train\_set, test\_set = [], []
2. *# for i in range(sensor\_num):*
3. train\_set += sliding\_window(volume\_train, window\_size=7)
4. test\_set += sliding\_window(volume\_test, window\_size=7)
5. train\_set, test\_set = np.array(train\_set), np.array(test\_set)
6. *# train\_set = np.swapaxes(train\_set, 1, 2)*
7. *# test\_set = np.swapaxes(test\_set, 1, 2)*
8. print(train\_set.shape,test\_set.shape)
9. print(train\_set,test\_set)

#### 4.2 模型设计

定义CNN + GRU模型

1. device = 'cpu'
2. class MyCNN(nn.Module):
3. def \_\_init\_\_(self):
4. super(MyCNN, self).\_\_init\_\_()
5. self.conv1 = nn.Sequential(
6. nn.Conv2d(in\_channels=6, out\_channels=6, kernel\_size=1, padding=0),
7. nn.AvgPool2d((200, 1)),
8. nn.ReLU(inplace=True)
9. )
11. def forward(self, x):
12. out = self.conv1(x)
13. return out.squeeze()
14. Feature\_extract = MyCNN()
15. torch\_gru = nn.GRU(input\_size=1, hidden\_size=32, num\_layers=1, batch\_first=True).to(device)
16. output\_model = nn.Linear(32, 200).to(device)

初始化模型参数

1. loss\_func = nn.MSELoss()
2. optimizer = torch.optim.Adam(list(torch\_gru.parameters()) + list(output\_model.parameters()), lr=0.0001)

#### 4.3 函数定义

定义评估函数

1. def mape(y\_true, y\_pred):
2. y\_true, y\_pred = np.array(y\_true), np.array(y\_pred)
3. non\_zero\_index = (y\_true > 0)
4. y\_true = y\_true[non\_zero\_index]
5. y\_pred = y\_pred[non\_zero\_index]
6. mape = np.abs((y\_true - y\_pred) / y\_true)
7. mape[np.isinf(mape)] = 0
8. return np.mean(mape) \* 100

读取batch函数

1. def next\_batch(data, batch\_size):
2. data\_length = len(data)
3. num\_batches = math.ceil(data\_length / batch\_size)
4. for batch\_index in range(num\_batches):
5. start\_index = batch\_index \* batch\_size
6. end\_index = min((batch\_index + 1) \* batch\_size, data\_length)
7. yield data[start\_index:end\_index]

定义反归一化函数

1. def denormalize(x):
2. return x \* (dmax - dmin) + dmin

#### 4.4 训练函数

1. train\_log = []
2. test\_log = []
3. *#开始时间*
4. timestart = time.time()
5. trained\_batches = 0 #记录多少个batch
6. for epoch in range(100):
8. total\_1oss = 0 #记录Loss
9. for batch in next\_batch(shuffle(train\_set), batch\_size=128):
10. *#每一个batch的开始时间*
11. batchstart = time.time()
13. batch = torch.from\_numpy(batch).float().to(device)  # (batch, seq\_len)
14. *# 使用短序列的前6个值作为历史，最后一个值作为预测值。*
15. x, label = batch[:, :6], batch[:, -1]
17. x = Feature\_extract(x)
18. out, hidden = torch\_gru(x.unsqueeze(-1))  # out: (batch\_size, seq\_len, hidden\_size)
19. out = output\_model(out[:, -1, :])
20. prediction = out.squeeze(-1)  # (batch)
21. label = label.squeeze(-1)
23. loss = loss\_func(prediction, label)
24. optimizer.zero\_grad()
25. loss.backward()
26. optimizer.step()
27. *#correct += (prediction == label).sum().item()*
28. *#累加loss*
29. *#total\_1oss += loss.item( )*
30. trained\_batches += 1
31. *#计算平均oss与准确率*
32. *#train\_loss = total\_1oss / train\_batch\_num*
33. *#train\_log.append(train\_loss)*
34. *# 每训练一定数量的batch，就在测试集上测试模型效果。*
35. *#if trained\_batches % 100 == 0:*
36. train\_log.append(loss.detach().cpu().numpy().tolist());
37. train\_batch\_time = (time.time() - batchstart)
38. print('batch %d, train\_loss %.6f,Time used %.6fs'%(trained\_batches, loss,train\_batch\_time))

41. # 每训练一定数量的batch，就在测试集上测试模型效果。
42. if trained\_batches % 100 == 0:
43. *#每一个batch的开始时间*
44. batch\_test\_start = time.time()
45. *#在每个epoch上测试*
46. all\_prediction = []
47. for batch in next\_batch(test\_set, batch\_size=128):
48. batch = torch.from\_numpy(batch).float().to(device)  # (batch, seq\_len)
49. x, label = batch[:, :6], batch[:, -1]
51. x = Feature\_extract(x)
52. out, hidden = torch\_gru(x.unsqueeze(-1))  # out: (batch\_size, seq\_len, hidden\_size)
53. out = output\_model(out[:, -1, :])
54. prediction = out  # (batch)
55. all\_prediction.append(prediction.detach().cpu().numpy())
56. all\_prediction = np.concatenate(all\_prediction)
57. all\_label = test\_set[:, -1]
58. all\_label = all\_label.squeeze()
59. *# 没有进行反归一化操作。*
60. *#all\_prediction = denormalize(all\_prediction)*
61. *#all\_label = denormalize(all\_label)*
62. *# 计算测试指标。*
63. rmse\_score = math.sqrt(mse(all\_label, all\_prediction))
64. mae\_score = mae(all\_label, all\_prediction)
65. mape\_score = mape(all\_label, all\_prediction)
66. test\_log.append([rmse\_score, mae\_score, mape\_score])
67. test\_batch\_time = (time.time() - batch\_test\_start)
68. print('——test\_batch %d, test\_rmse\_loss %.6f,test\_mae\_loss %.6f,test\_mape\_loss %.6f,Time used %.6fs'%(trained\_batches, rmse\_score,mae\_score,mape\_score,test\_batch\_time))
70. #每一个epoch的结束时间
71. *#elapsed = (time.time() - epochstart)*
73. #计算总时间
74. timesum = (time.time() - timestart)
75. print('The total time is %fs'%(timesum))
76. print('The total time is %fs'%(timesum))

## 五、实验结果

本实验的train\_loss曲线结果如图5.1所示。

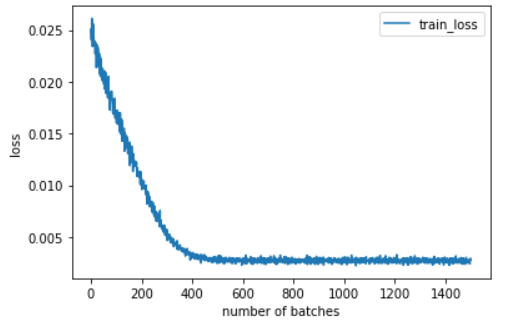


图5.1 train\_loss曲线结果图

结果分析：由图可知，训练集上loss在不断的降低，最后趋近于0，虽然会有轻微抖动，但是模型总体效果还是不错的。

MAE计算评估结果如图5.2所示。指标在不断的下降，最后趋于0。

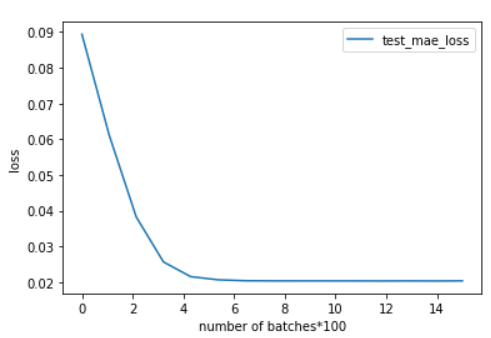


图5.2 MAE计算评估结果图

RMSE计算评估结果如图5.3所示。指标在不断的下降，最后趋于0。

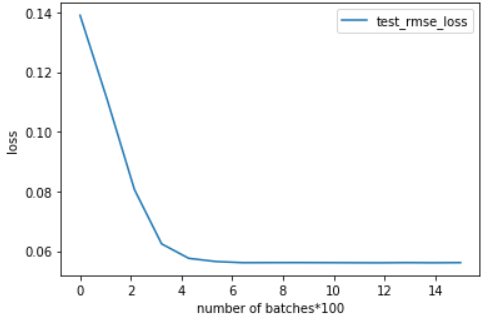


图5.3 RMSE计算评估结果图

## 实验心得体会

经过暑假的深度学习课程学习，让我对研究生知识学习生涯有了入门。这次的大实验是对深度学习很多神经网络模型的一次应用，比如用到的卷积神经网络RNN，门控循环神经网络GRU等等，将理论应用于实践，让我对课堂上的知识有了更深的了解。在设计的过程中，学会了该如何对各种模型内的超参数进行设置，以有一个更好的模型输出结果。

## 七、参考文献

## 八、附录