

研究生《深度学习》课程

实验报告

|  |  |
| --- | --- |
| **实验名称：** | **循环神经网络实验** |
| **姓 名：** | **梁棋棋** |
| **学 号：** | **20120376** |
| **上课类型：** | **专业课** |
| **日 期：** | **2020.8.28** |

## 一、实验内容

1. 手动实现循环神经网络RNN，并在至少一个数据集上进行实验，从训练时间、预测精度、Loss变化等角度分析实验结果（最好使用图表展示）
2. 使用torch.nn.rnn实现循环神经网络，并在至少一个数据集上进行实验，从训练时间、预测精度、Loss变化等角度分析实验结果（最好使用图表展示）
3. 不同超参数的对比分析（包括hidden\_size、batchsize、lr等）选其中至少1-2个进行分析
4. 使用PyTorch实现LSTM和GRU并在至少一个数据集进行试验分析
5. 设计实验，对比分析LSTM和GRU在相同数据集上的结果

## 二、实验设计

1. RNN实验的设计思路

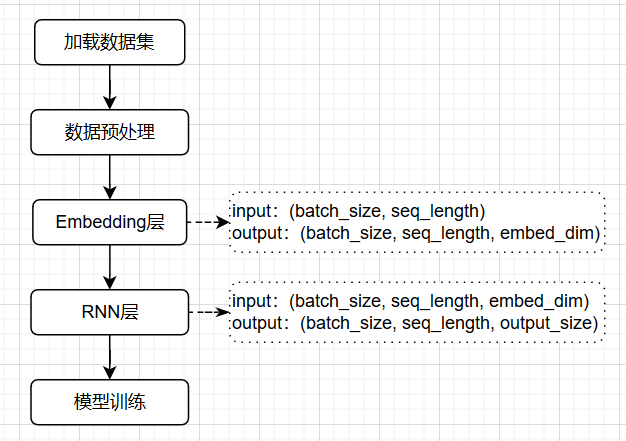


图2.1 RNN实验设计流程图

1. LSTM实验的设计思路

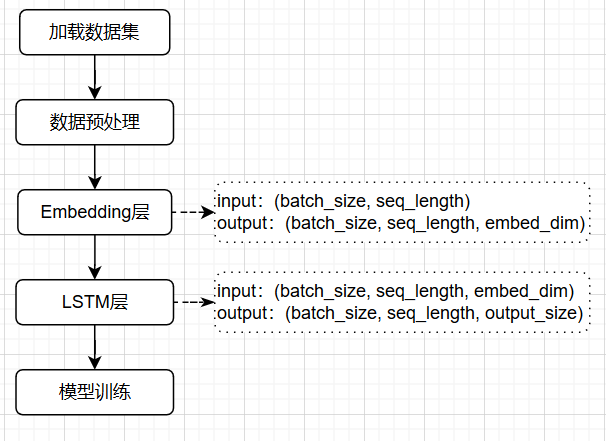


图2.2 LSTM实验设计流程图

1. GRU实验的设计思路

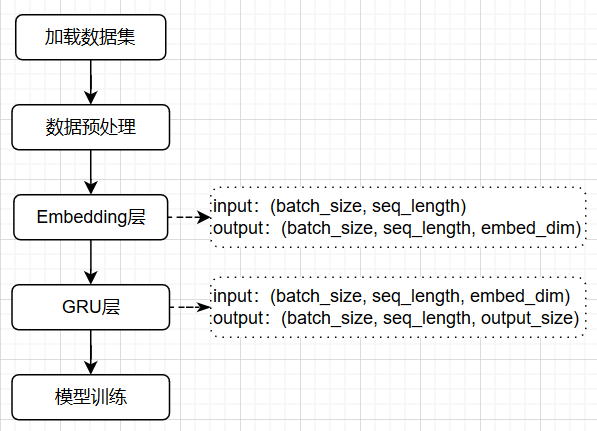


图2.3 GRU实验设计流程图

## 三、实验环境及实验数据集

表3-1 实验环境

|  |  |
| --- | --- |
| 操作系统 | Windows 10 |
| 使用语言 | Python3.6.10 |
| 使用IDE | Jupyter notebook |
| 开发平台 | Pytorch1.5.1 |

实验数据集1：check-in轨迹数据

实验数据集2：高速公路传感器数据

## 四、实验过程

#### 4.1 数据处理模块

1. 加载数据集：载入数据集后只留下userId（用户id）那一列和venueCategory（地点类别）那一列，因为我们需要根据每个人的运动轨迹来进行预测，由于轨迹是按照时间顺序排列的，所以可以不用时间那一列。



图4.1.1 加载数据集代码图

1. 获取地址字典：使用list(set(location))对地点列表进行去重操作，然后使用dict(zip())函数给每个字典分配一个数字并构成字典的形式

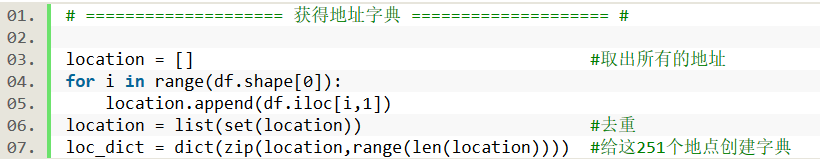


图4.1.2 获取地址字典代码图

1. 分割训练集和测试集：使用replace将地址类别那一列替换成字典里的数字，然后将每个人的运动轨迹通过window\_size=10的滑动窗口来分割并设置训练集和测试集。train\_set是轨迹的训练集，size=[len(train\_set), 10]；train\_set是轨迹的测试集，size=[len(train\_set), 10]。

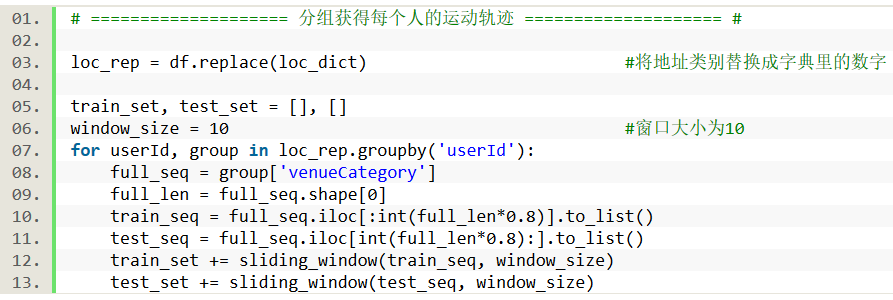


图4.1.3 分割训练集和测试集代码图

#### 4.2 手动实现RNN

1. RNN模型构建：nn.Embedding的输入是(batch\_size, seq\_length)，输出是将维度扩展成词向量的维度，也就是(batch\_size, seq\_length, embed\_dim)，对于文字分类来说，input\_size = output\_size = embed\_dim = voc\_size，此处由于地点类别有251个，将他们归类于数字就是0~250，一个长度为10的句子的size就是[10, 251]（one-hot编码）。w\_h、u\_h、b\_h是隐藏层的参数，w\_y、b\_y是输出层的参数。

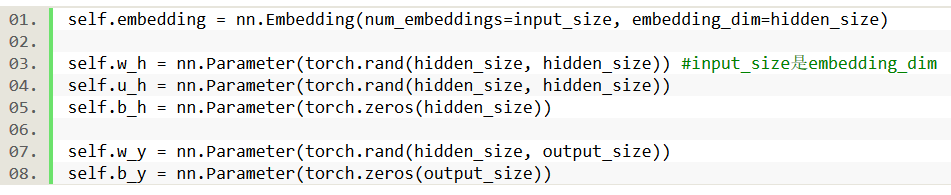


图4.2.1 RNN模型构建代码图

1. RNN前向传播：输入的x经过embedding层后维度为(batch\_size, seq\_length, embed\_dim)，根据RNN的流程得到每一个时间节点的隐藏层h和输出层y，这个输出层y就是预测序列，大小为(batch\_size, embed\_dim)，因为for循环循环10次，所以y\_list的size是(seq\_length, batch\_size, embed\_dim)

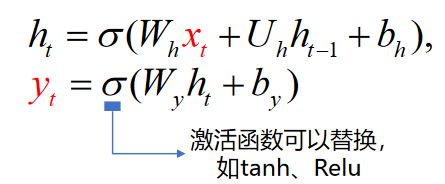


图4.2.2 RNN数学公式图

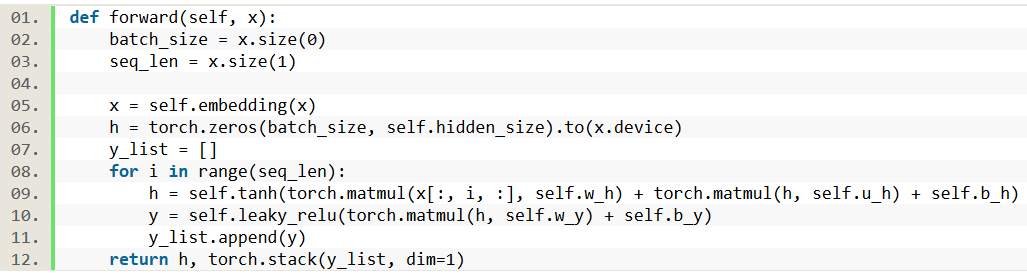


图4.2.3 前向传播代码图

1. 训练模型：假设batch\_size是128（128组），window\_size是10（10个轨迹），embedding\_dim是251（类似于one-hot编码），那么batch就是[128, 10, 251]，且x是[128, 9, 251]（也就是每个batch\_size的前9个轨迹），label是[128, 1, 251]（把最后一个轨迹做为预测结果）。也就是输入x，得到的out也是[128, 9, 251]，但是假设我输入的是0~8段轨迹，那么我的预测就是1~9，取出第9个与我实际的第9个做loss。

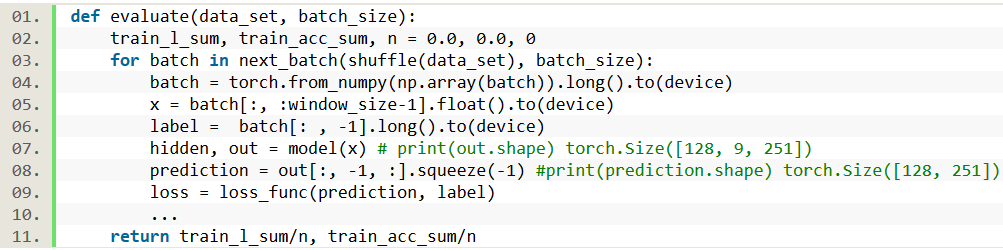


图4.2.4 训练模型代码图

1. 过拟合处理：因为会有过拟合现象，所以在优化器中加入了weight\_decay权重衰减来防止过拟合，并且设置了MultiStepLR即学习率衰减，每到60个epoch学习率衰减为原来的0.1，能够使得精度有一个小的飞跃。

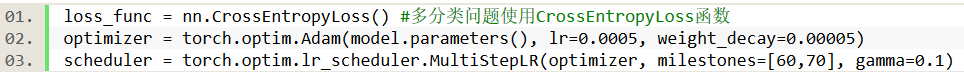


图4.2.5 过拟合处理代码图

#### 4.3 利用torch.nn实现RNN

1. 模型定义：由于batch\_size是128，seq\_len是10，所以embedding层的输入是[128, 10]，经过了embedding层后扩充了维度变成了[128, 10, 32]，其中32是后续连接rnn的维度，所以rnn的有32个隐藏层单元，输入就是32维。y和h代表模型输入x和上一个隐藏时刻的h输入到rnn后获得的当前时刻的输出和当前时刻的隐藏状态，循环seq\_len次。

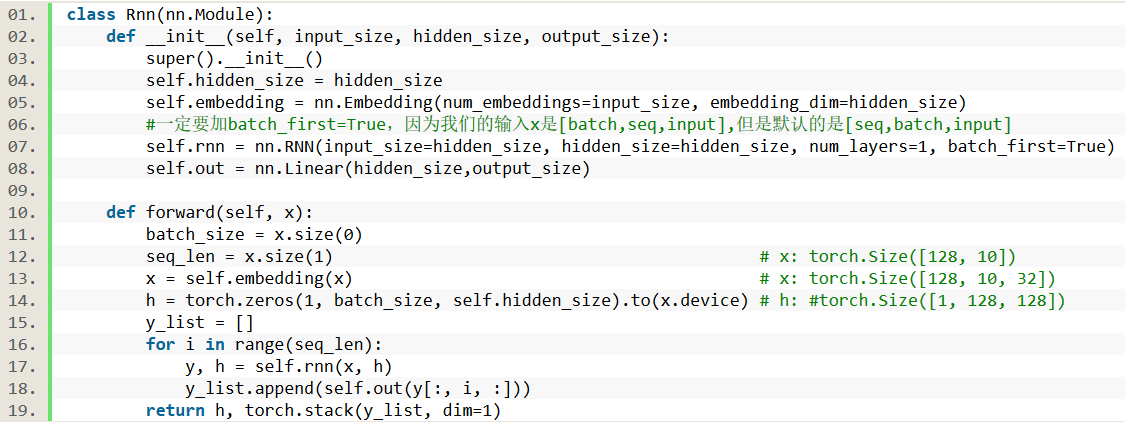


图4.3.1 RNN模型定义代码图

#### 4.4 手动实现LSTM

1. 模型定义：输入x与embedding层和RNN相同，根据LSTM的公式得到模型定义，其中gates层其实是合并矩阵运算，提高并行性。一步实现所有门的计算，再将结果拆分后进行激活。

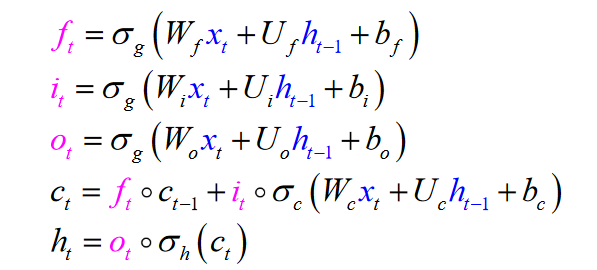


图4.4.1 LSTM公式图

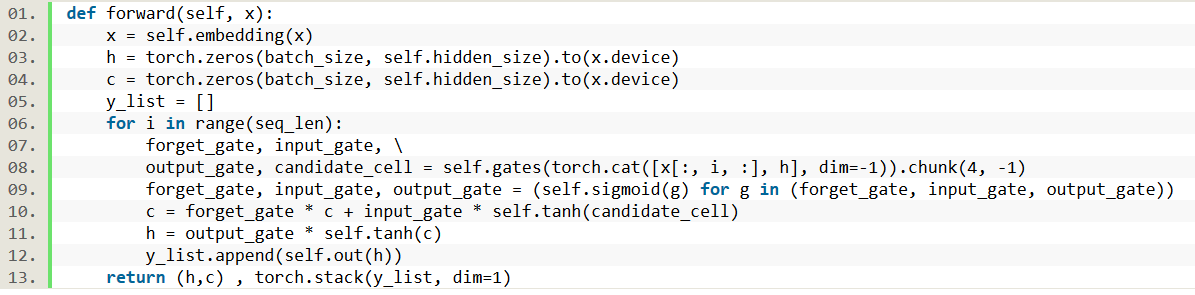


图4.4.2 LSTM模型定义代码图

#### 4.5 利用torch.nn实现LSTM

1. 模型定义：由于batch\_size是128，seq\_len是10，所以embedding层的输入是[128, 10]，经过了embedding层后扩充了维度变成了[128, 10, 32]，其中32是后续连接lstm的维度，所以lstm的有32个隐藏层单元，输入就是32维。y和（h，c）代表模型输入x和上一个隐藏时刻的（h，c）输入到lstm后获得的当前时刻的输出和当前时刻的隐藏状态，循环seq\_len次。

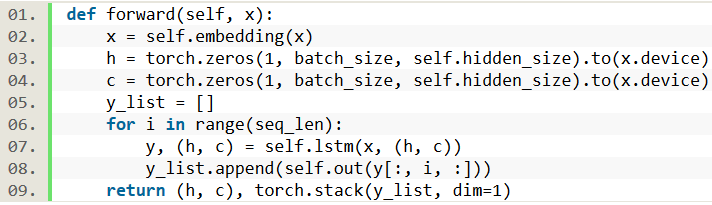


图4.5.1 LSTM模型定义代码图

#### 4.6 手动实现GRU

1. 模型定义：输入x与embedding层和RNN相同，根据GRU的公式得到模型定义，其中gates层其实是合并矩阵运算，提高并行性。既解决了RNN存在的长期依赖问题，又比LSTM提高了计算效率

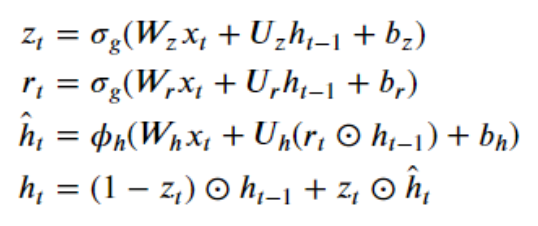


图4.6.1 GRU公式图

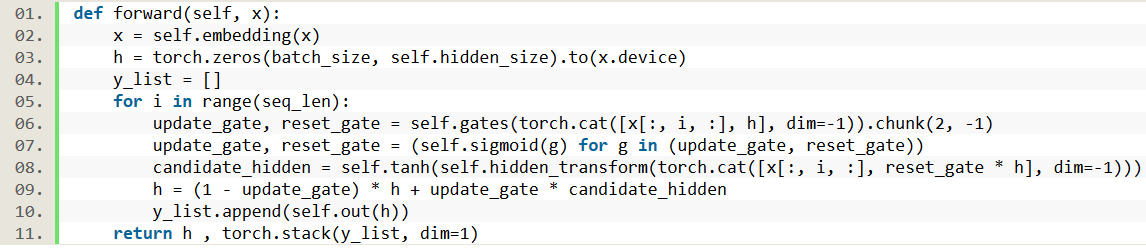


图4.6.2 GRU模型定义代码图

#### 4.7 利用torch.nn实现GRU

1. 模型定义：由于batch\_size是128，seq\_len是10，所以embedding层的输入是[128, 10]，经过了embedding层后扩充了维度变成了[128, 10, 32]，其中32是后续连接GRU的维度，所以GRU的有32个隐藏层单元，输入就是32维。y和h代表模型输入x和上一个隐藏时刻的h输入到GRU后获得的当前时刻的输出和当前时刻的隐藏状态，循环seq\_len次。

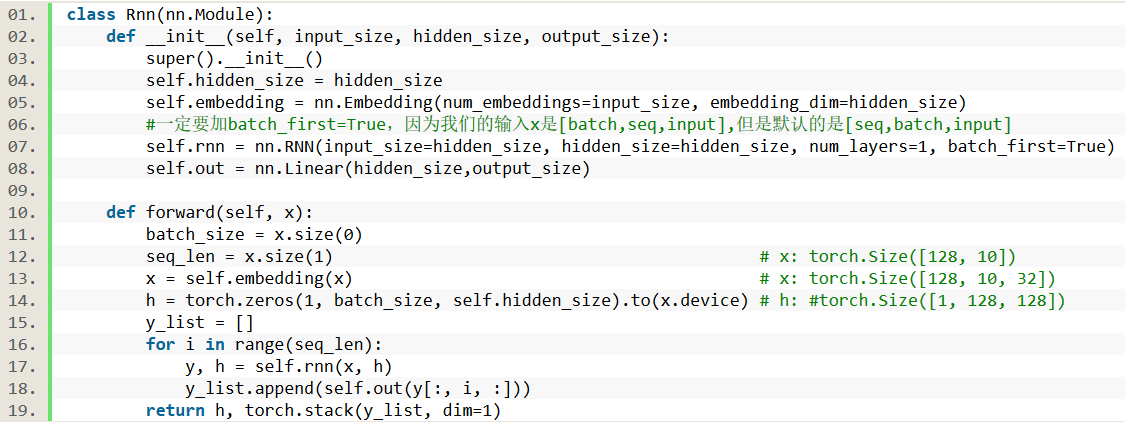


图4.7.1 GRU模型定义代码图

## 五、实验结果

#### 5.1 手动实验RNN

可以看到训练集的loss下降得很快，测试集的loss下降得很慢，同理训练集acc上升得很快，测试集acc上升很慢，并且已经设置了权重衰减weight\_decay=0.00005和学习率衰减gamma=0.1的情况下还有过拟合的迹象。图中60个epoch的即为学习率衰减的时刻，由0.0005衰减为0.00005。设置了学习率衰减后可以看到有明显的下坡飞跃，看来是有一定的效果的。训练集的准确率能到达97%,而测试集的准确率仅能达到56%

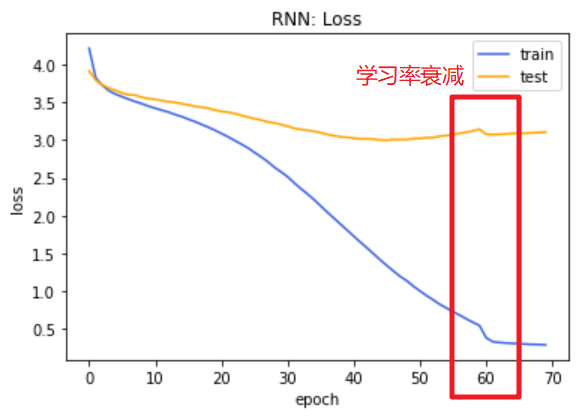


图5.1.1 手动实现RNN的loss结果图

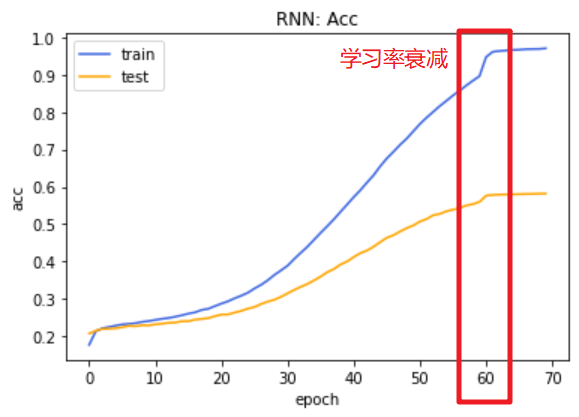


图5.1.2 手动实现RNN的acc结果图

手动实现RNN的时间在6min左右，利用torch.nn实现时间在22min左右，手动实现更快。手动实现一个epoch大概是5s。



图5.1.3 手动实现RNN的时间结果图

#### 5.2 利用torch.nn实现RNN

由图5.2.1和图5.2.2可以看出训练集的训练效果很好，损失能够降到0.909317，准确率最大能达到81%

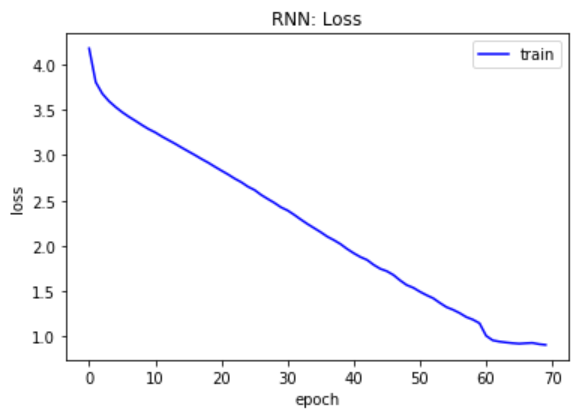


图5.2.1 利用torch.nn实现RNN的训练集loss结果图

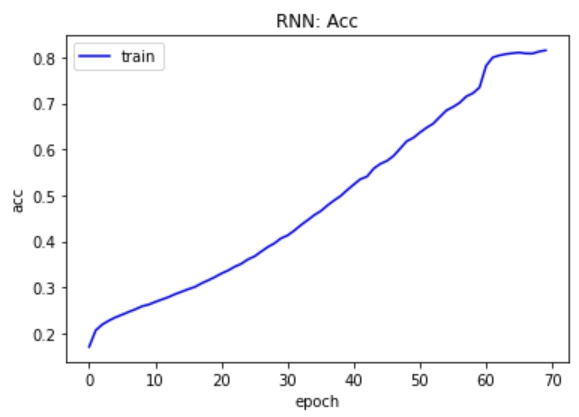


图5.2.2 利用torch.nn实现RNN的训练集acc结果图

在图5.2.3和图5.2.4中，在加入了权重衰减和学习率衰减防止过拟合后，测试集过拟合现象有明显改善，但是过拟合现象仍然很严重，只能归结于人的运动本身就有比较大的随机性。

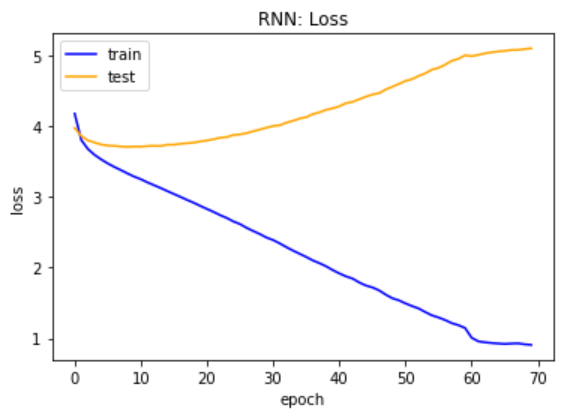


图5.2.3 利用torch.nn实现RNN的loss对比结果图

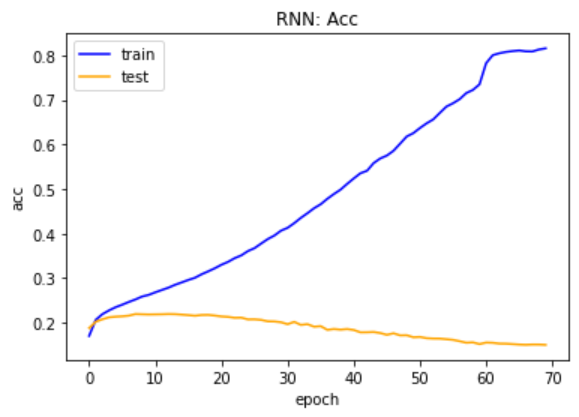


图5.2.4 利用torch.nn实现RNN的acc对比结果图

手动实现RNN的时间在6min左右，利用torch.nn实现时间在22min左右，手动实现更快，快了4倍左右。利用torch.nn实现一个epoch大概是19s。



图5.2.5 利用torch.nn实现RNN的时间结果图

#### 5.3 RNN实验------不同batch\_size对比

由图5.3.1和5.3.2可知，在固定lr = 0.0005和hidden\_size = 1024的情况下，batch\_size\_size = 1024的效果最好，在三者里batch\_size越小，loss就越低，acc就越高，拟合效果就越好，batch\_size\_size = 1024的loss最低可以达到0.126759，准确率最高可以达到99.3598%

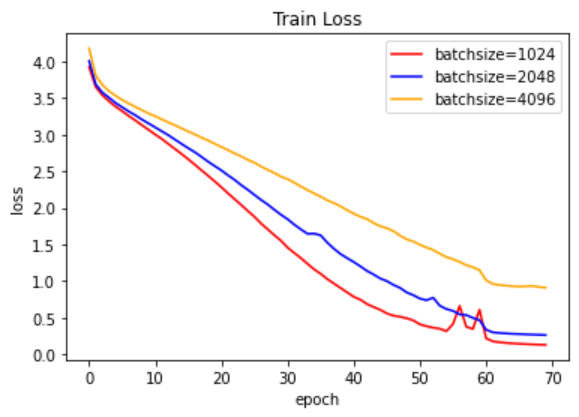


图5.3.1 不同batch\_size的训练集loss对比结果图

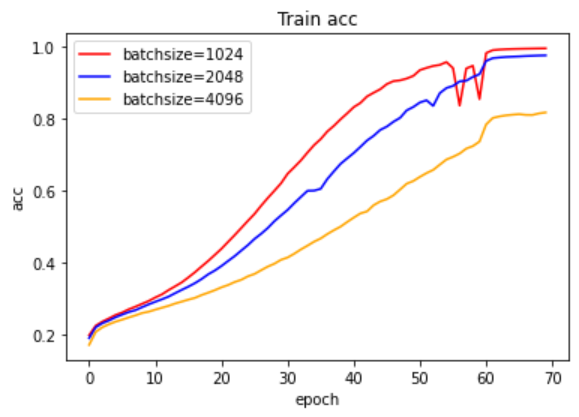


图5.3.2 不同batch\_size的训练集acc对比结果图

在图5.2.3和图5.2.4中，在加入了权重衰减和学习率衰减防止过拟合后，测试集过拟合现象有明显改善，但是过拟合现象仍然很严重，只能归结于人的运动本身就有比较大的随机性。

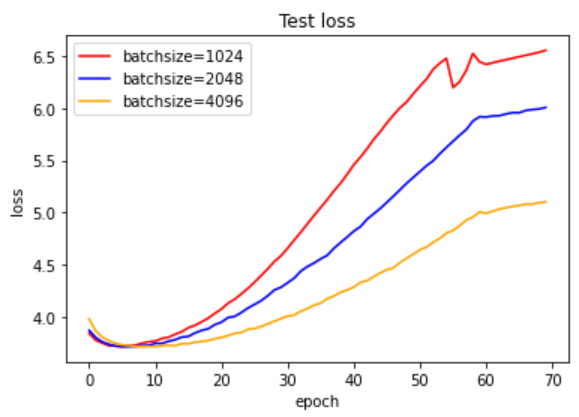


图5.3.3 不同batch\_size的测试集loss对比结果图

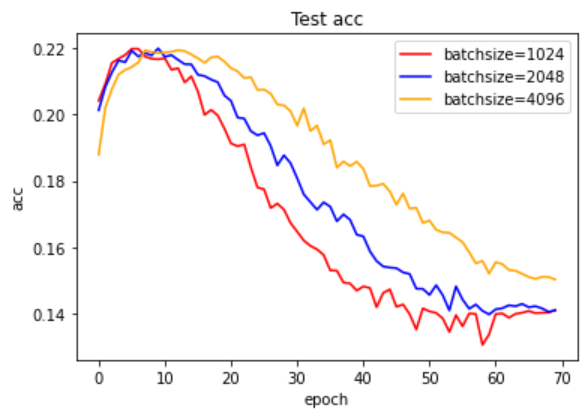


图5.3.4 不同batch\_size的测试集acc对比结果图

虽然时间相差得不多，但是总的来说，batch\_size越小，时间就越多。

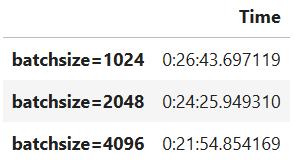


图5.3.5 不同batch\_size的时间对比结果图

#### 5.4 RNN实验------不同lr对比

由图5.4.1和5.4.2可知，在固定batch\_size = 1024和hidden\_size = 1024的情况下，lr = 0.0005的效果最好，在二者里lr = 0.0005的loss就比lr = 0.0001的低，acc就高，拟合效果就好。lr = 0.0005的loss最低可以达到0.126759，准确率最高可以达到99.3598%，但是lr = 0.0001的准确率最高仅能达到42.3112%

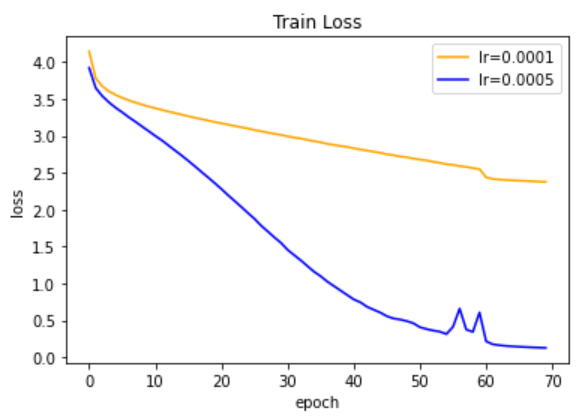


图5.4.1 不同lr的训练集loss对比结果图

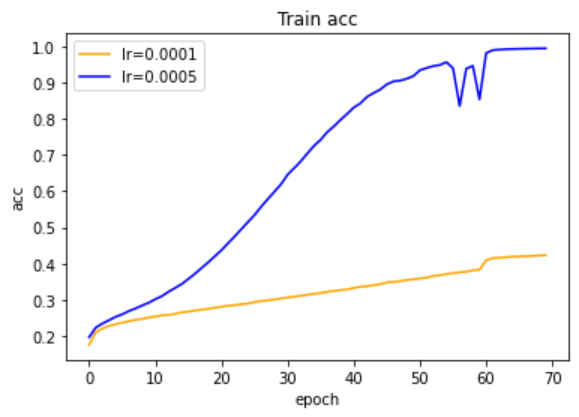


图5.4.2 不同lr的训练集acc对比结果图

在图5.4.3和图5.4.4中，在加入了权重衰减和学习率衰减防止过拟合后，测试集过拟合现象有明显改善，但是过拟合现象仍然很严重，只能归结于人的运动本身就有比较大的随机性。

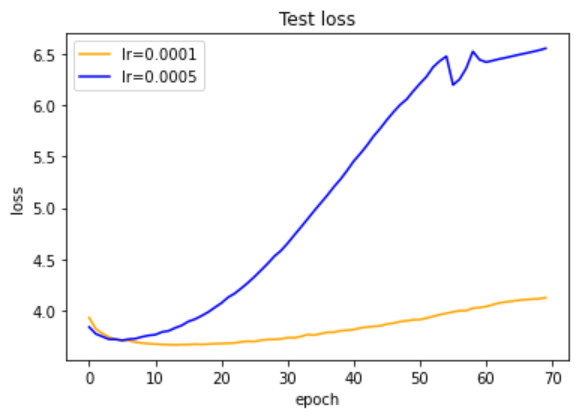


图5.4.3 不同lr的测试集loss对比结果图

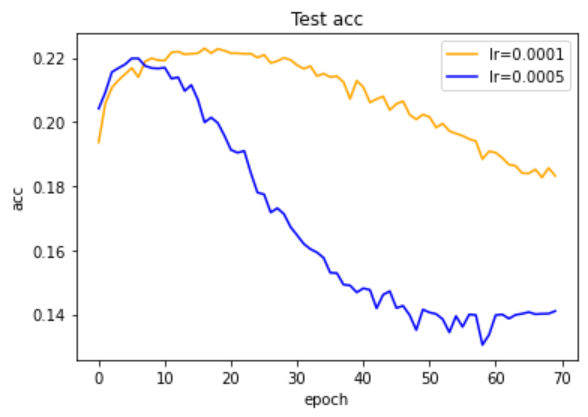


图5.4.4 不同lr的测试集acc对比结果图

由图5.4.5可知，学习率对时间的影响并不大。

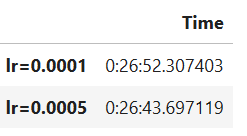


图5.4.5 不同lr的时间对比结果图

#### 5.5 手动实现LSTM

由图5.5.1和图5.5.2可以看出训练集的训练效果很好，损失能够降到0.047521，准确率最大能达到99.4373%。

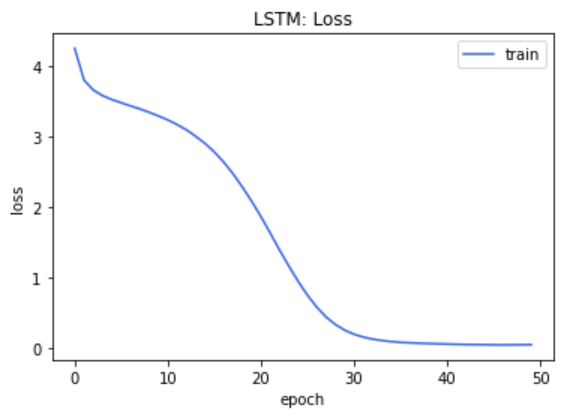


图5.5.1 手动实现LSTM的训练集loss结果图

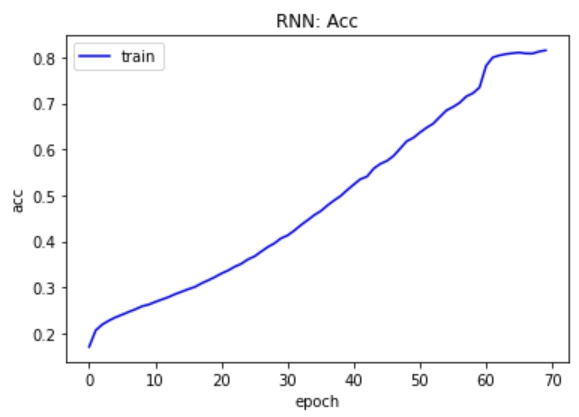


图5.5.2 手动实现LSTM的训练集acc结果图

在图5.5.3和图5.5.4中，在加入了权重衰减和学习率衰减防止过拟合后，测试集过拟合现象有明显改善，但是过拟合现象仍然很严重，只能归结于人的运动本身就有比较大的随机性。

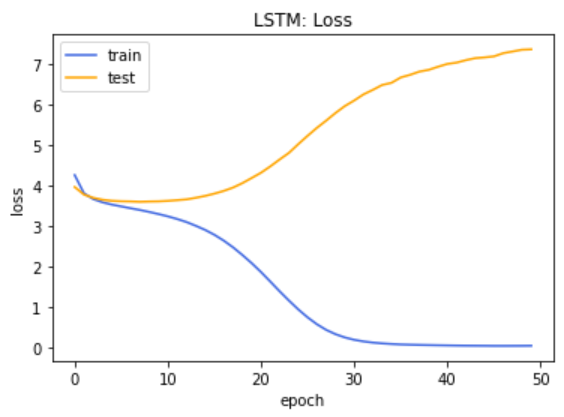


图5.5.3 手动实现LSTM的loss对比结果图

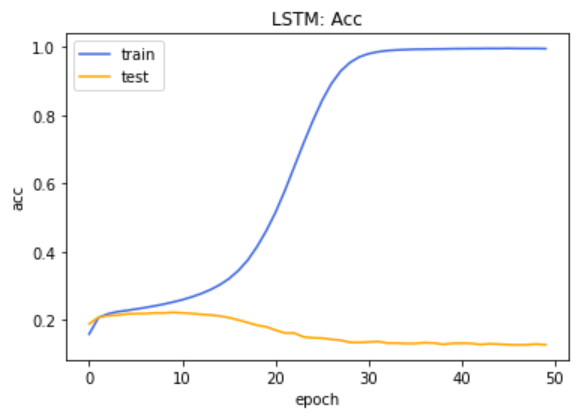


图5.5.4 手动实现LSTM的acc对比结果图

由图可知，手动实现LSTM的时间要低得多，是torch.nn实现的六分之一，跑50个epoch时间是10分钟，大概 12s一个epoch。

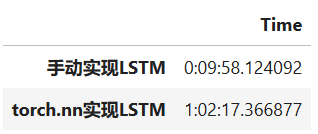


图5.5.5 手动实现LSTM的时间结果图

**5.6 利用torch.nn实现LSTM**

由图5.6.1和图5.6.2可以看出训练集的训练效果很好，损失能够降到 0.068112，准确率最大能达到98.5544 %。

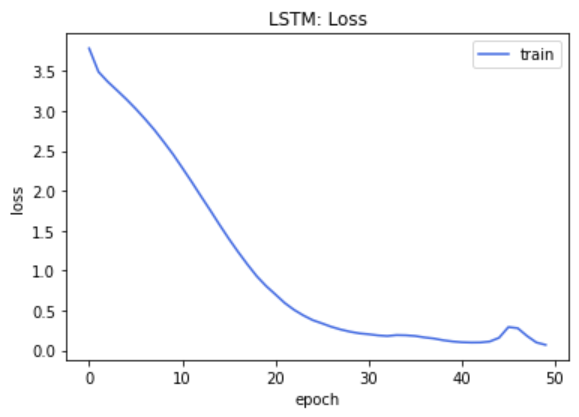


图5.6.1 利用torch.nn实现LSTM的训练集loss结果图

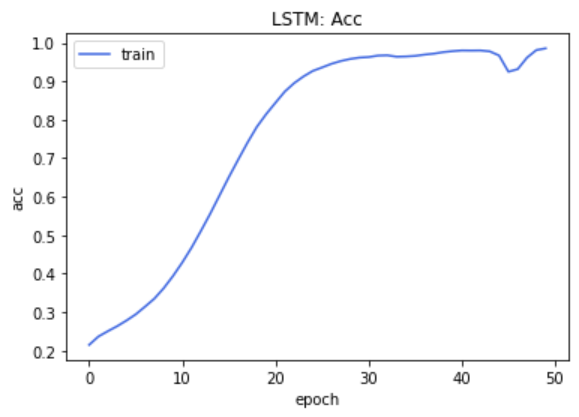


图5.6.2 利用torch.nn实现LSTM的训练集acc结果图

在图5.6.3和图5.6.4中，在加入了权重衰减和学习率衰减防止过拟合后，测试集过拟合现象有明显改善，但是过拟合现象仍然很严重，只能归结于人的运动本身就有比较大的随机性。

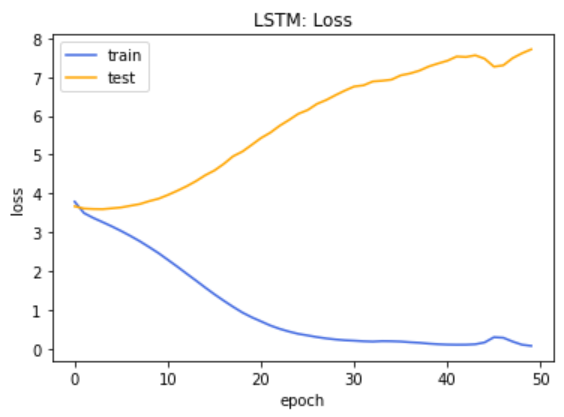


图5.6.3 利用torch.nn实现LSTM的loss对比结果图

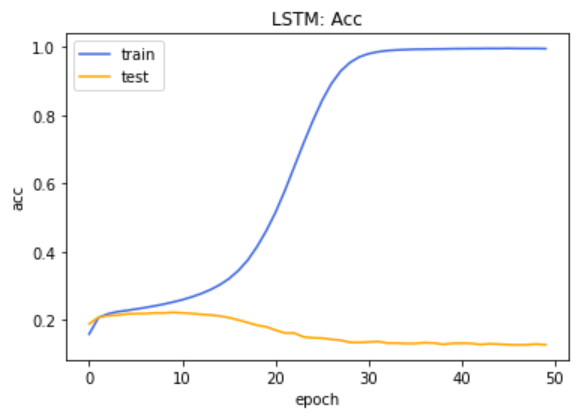


图5.6.4 利用torch.nn实现LSTM的acc对比结果图

由图可知，利用torch.nn实现LSTM的时间要多得多，是手动实现的六倍，跑50个epoch时间是1小时，大概1min14s一个epoch。

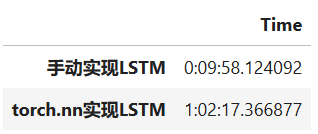


图5.6.5 利用torch.nn实现LSTM的时间结果图

#### 5.7 手动实现GRU

由图5.7.1和图5.7.2可以看出训练集的训练效果很好，损失能够降到0.098222，准确率最大能达到99.3347%。

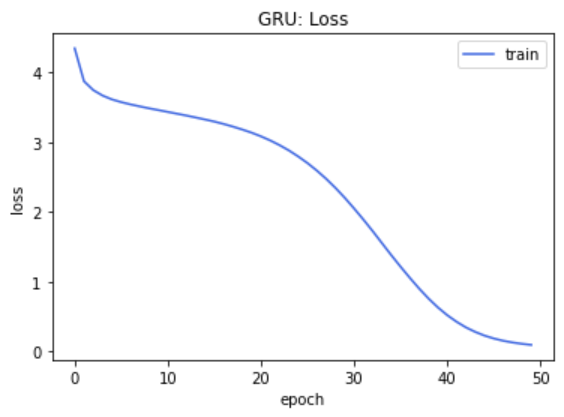


图5.7.1 手动实现GRU的训练集loss结果图

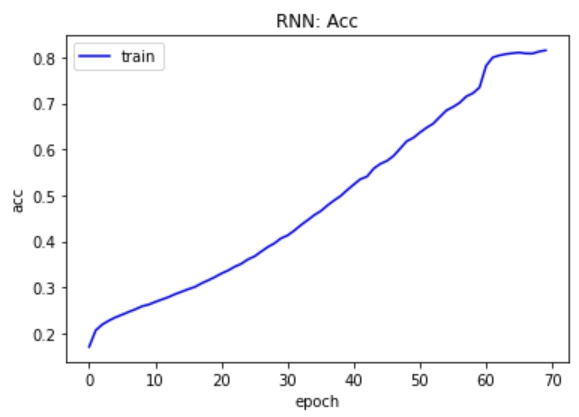


图5.7.2 手动实现GRU的训练集acc结果图

在图5.7.3和图5.7.4中，在加入了权重衰减和学习率衰减防止过拟合后，测试集过拟合现象有明显改善，但是过拟合现象仍然很严重，只能归结于人的运动本身就有比较大的随机性。

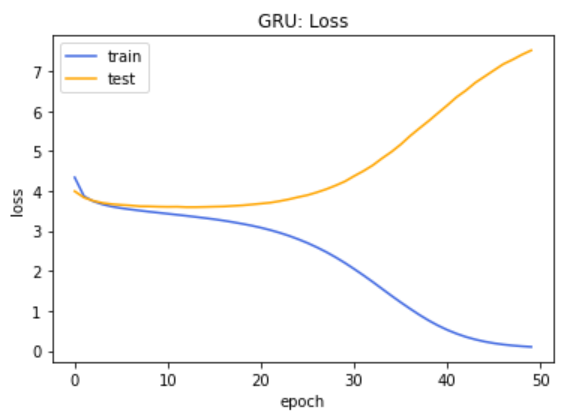


图5.7.3 手动实现GRU的loss对比结果图

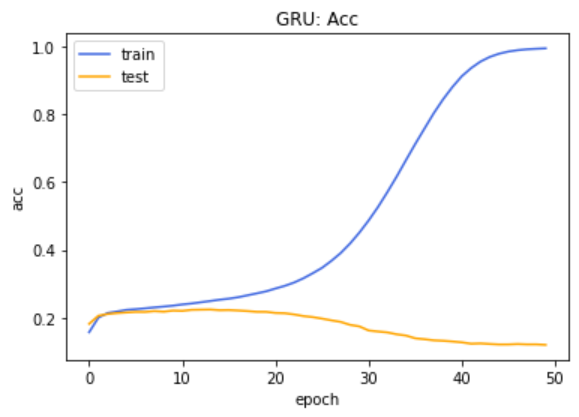


图5.7.4 手动实现GRU的acc对比结果图

由图可知，手动实现比torch.nn实现要快得多跑，50个epoch时间是8分半，大概10s一个epoch。



图5.7.5 手动实现GRU的时间结果图

**5.8 利用torch.nn实现GRU**

由图5.8.1和图5.8.2可以看出训练集的训练效果很好，损失能够降到0.041075，准确率最大能达到98.9695 %。

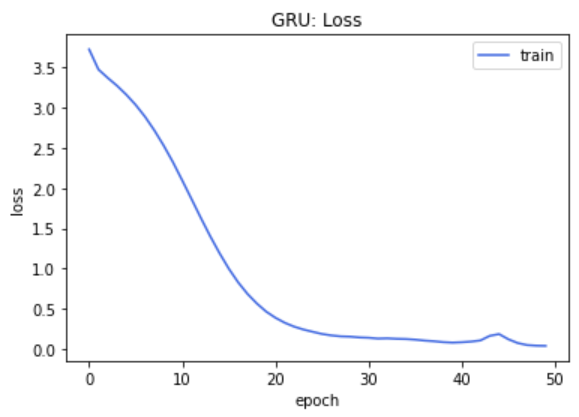


图5.8.1 利用torch.nn实现GRU的训练集loss结果图

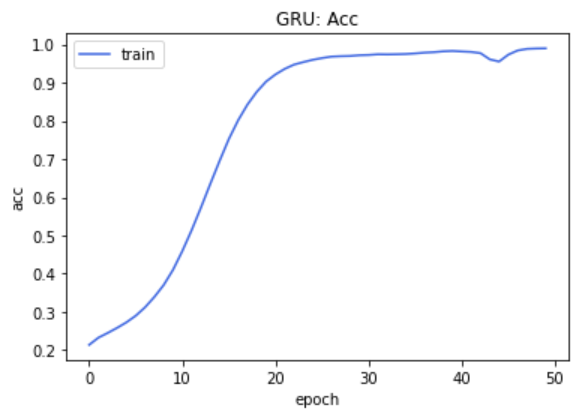


图5.8.2 利用torch.nn实现GRU的训练集acc结果图

在图5.8.3和图5.8.4中，在加入了权重衰减和学习率衰减防止过拟合后，测试集过拟合现象有明显改善，但是过拟合现象仍然很严重，只能归结于人的运动本身就有比较大的随机性。

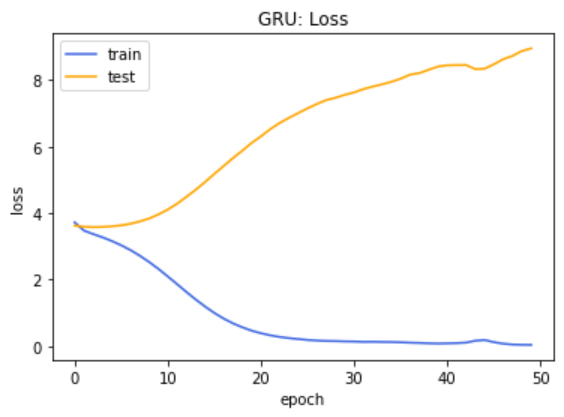


图5.8.3 利用torch.nn实现GRU的loss对比结果图

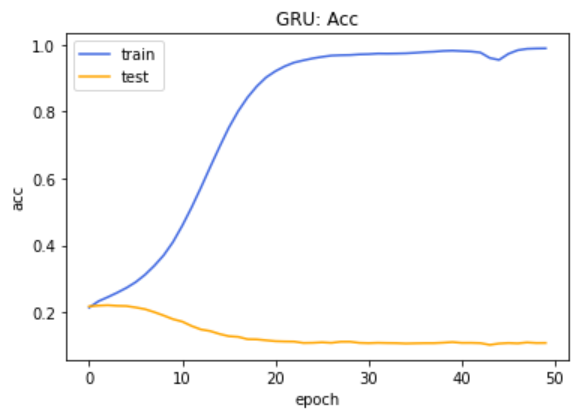


图5.8.4 利用torch.nn实现GRU的acc对比结果图

由图可知，torch.nn实现比手动实现要慢得多，50个epoch时间是55min，大概57s一个epoch。



图5.8.5 利用torch.nn实现GRU的时间结果图

#### 5.8 LSTM和GRU对比

由图5.8.1和图5.8.2可以看出，利用torch.nn的实现效果比手动实现的效果要好得多。利用torch.nn实现里GRU的实现更好一些，一开始的loss就比较低，收敛速度比较快；在手动实现里，LSTM模型的实现效果更好一些。虽然GRU 参数相对少更容易收敛，但是在数据集较大的情况下，LSTM性能更好，收敛速度更快。在相同的参数下，利用torch.nn实现GRU在第20个epoch时准确率就达到了90%，利用torch.nn实现LSTM在第24个epoch，手动实现LSTM在第28个epoch，手动实现GRU是在第41个epoch准确率才达到90%，所以利用torch.nn实现里GRU的效果更好一些。

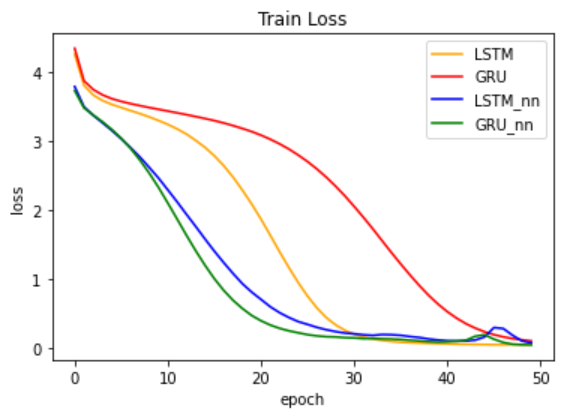


图5.8.1 LSTM和GRU训练集loss对比图

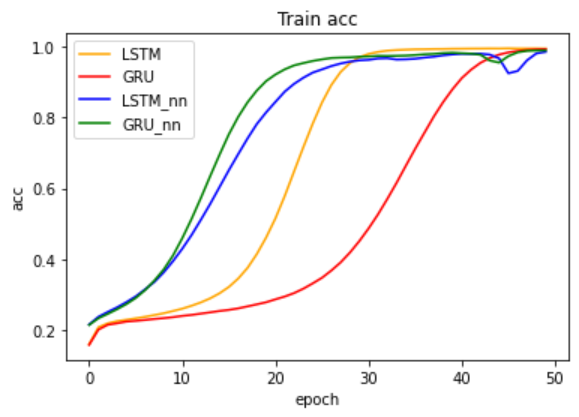


图5.8.2 LSTM和GRU训练集acc对比图

在图5.7.3和图5.7.4中，在加入了权重衰减和学习率衰减防止过拟合后，测试集过拟合现象有明显改善，但是过拟合现象仍然很严重。通过大量的修改batch\_size、hidden\_size和lr之后，过拟合仍然没有得到解决。LSTM和GRU仍有过拟合的迹象。

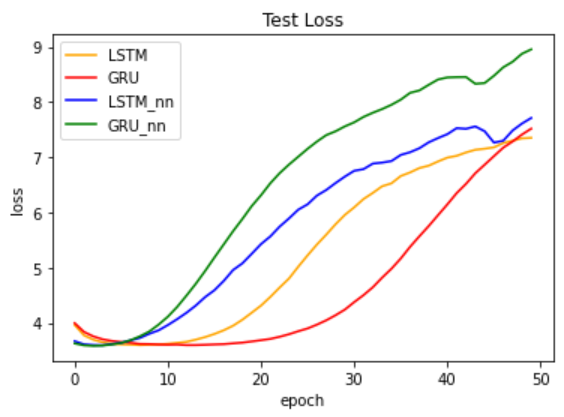


图5.8.3 LSTM和GRU测试集loss对比图

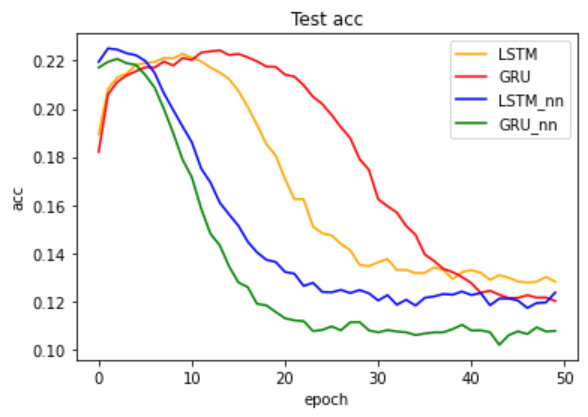


图5.8.4 LSTM和GRU测试集acc对比图

由图可知，在固定相同的参数下，利用torch.nn实现的运行时间都特别长，手动实现的都特别短。



图5.8.5 LSTM和GRU的时间结果图

## 六、实验心得体会

1. nn.Embedding的作用是一个保存了固定字典和大小的简单查找表，用来保存词嵌入和用下标检索它们。模块的输入是一个下标的列表，输出是对应的词嵌入。
2. torch.stack(y\_list, dim=1)解析：y\_list是Tensor的list，也就是说这个list由tensor构成。由于y是(batch\_size, embed\_dim)，y\_list是(seq\_length, batch\_size, embed\_dim)。如下图1.11表示第一个人的第一段轨迹的第1个分量，1.12表示第一个人的第一段轨迹的第2个分量，2.24表示第2个人的第2段轨迹的第4个分量，那么y1就表示第1个人的第1段轨迹的所有分量和第2个人的第1段轨迹的所有分量（都是第1段轨迹，seq\_length是1的时候）；y2就表示第1个人的第2段轨迹的所有分量和第2个人的第2段轨迹的所有分量（都是第2段轨迹，seq\_length是2的时候），y\_list是seq\_len在第一维。

使用torch.stack(y\_list, dim=1)后：(batch\_size, seq\_length, embed\_dim)，也就是【第1个人第1段轨迹的所有分量，第1个人第2段轨迹的所有分量】和【第2个人第1段轨迹的所有分量，第2个人第2段轨迹的所有分量】，这样子结果就是batch\_size在前面

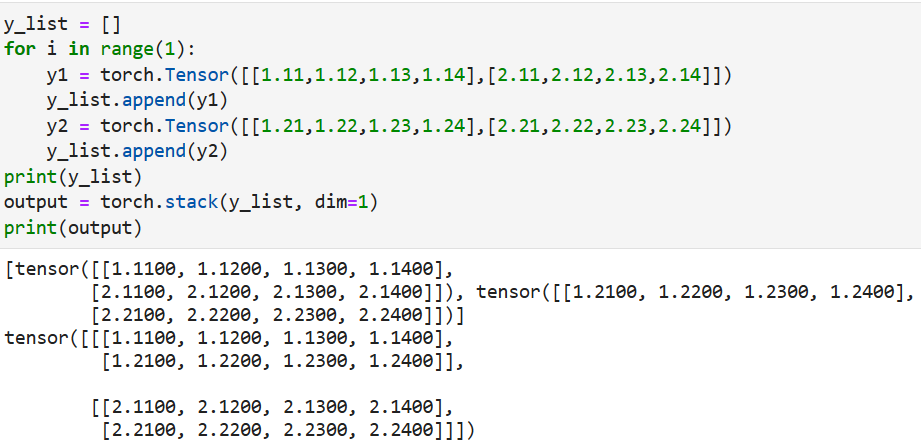


图6.1 torch.stack()测试样例图

1. df=pd.read\_csv(filename,nrows=20,names=["userId",'venueId'])

参数：nrows表示取前几行；names表示自定义每一列的列名，如果没有names参数就是自动把第一行作为每列的列名

1. df.loc[ 行信息 , 列信息] #基于label的索引

df.loc[ : , ['userId','venueCategory'] ] 表示取出所有的行的userId列和venueCategory列

df.iloc[i,1] 基于位置的索引，具体定位

1. test\_ind = test\_rep.set\_index('venueCategory')：将venueCategory列作为index
2. test\_ind.groupby('userId').groups：按照userId进行分组，然后将每组内的index组合成list，然后将[userId,list]组合成字典，也就是将每个分组后的index定位出来
3. 对于图片分类来说，假设图片大小是28\*28的，而分类结果是分3类，那么inputsize=784，outputsize=3

对于文字分类来说，文字的输入维度和分类维度是相同的，文字的输入类似于one-hot编码，输出就是这个one-hot编码的维度，记录着是每个字的概率

1. 因为人的轨迹本身就具有很强的随机性，所以测试集的loss一直很低甚至出现了过拟合的迹象是正常的。

## 七、参考文献

无。

## 八、附录

略。

**实验报告编写要求**

1. 正文要求小四号宋体，行间距1.5倍；

2. 英文要求小四号Times New Roman；

3. 在实验内容、实验过程、实验结果三部分需要针对当次实验不同的实验内容分别填写（模版以实验一为例），实验设计中如有必要也可以分开填写；

4. 实验报告配图的每幅图应有编号和标题，编号和标题应位于图下方处，居中，中文用五号宋体；

5. 表格应为三线表，每个表格应有编号和标题，编号和标题应写在表格上方正中，距正文段前0.5倍行距。表格中量与单位之间用“/”分隔，编号与标题中的中文用五号宋体；

6. 图、表、公式、算式等，一律用阿拉伯数字分别依序连续编排序号。其标注形式应便于互相区别，可分别为：图1、表2、公式(5)等。