

研究生《深度学习》课程

实验报告

|  |  |
| --- | --- |
| **实验名称：** | **疫情微博情绪分类** |
| **姓 名：** | **梁棋棋** |
| **学 号：** | **20120376** |
| **上课类型：** | **专业课** |
| **日 期：** | **2020.9.16** |

## 一、实验内容

微博情绪分类任务旨在识别微博中蕴含的情绪，输入是一条微博，输出是该微博所蕴含的情绪类别。在本次任务中，我们将微博按照其蕴含的情绪分为以下六个类别之一：积极、愤怒、悲伤、恐惧、惊奇和无情绪。

要求同学们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。

本任务不对模型的选择和设计进行限制，同时要求训练所用数据不脱离给定的数据集范围，不可引入外部语料（例如通过外部语料得到的预训练词向量等），可以以CNN、RNN等模型为基础进行设计。

实验报告中需要包含但不限于数据的处理过程介绍（例如分词等）、词向量、模型图、模型中各部分的作用介绍或者使用理由、超参数设置以及训练过程中各指标变化和实验结果分析等。

## 二、实验设计

1. 处理数据集

首先根据json格式的数据集划分出文本（content）和情绪标签（label）字段，数据编号（id）字段舍弃。将文本进行分词处理sample\_list（含训练集和测试集），特殊符号、标点符号、英文等非词形式都使用停词‘Stop’表示。同理将情绪标签映射成id获得label\_list。

将分词后的feature\_list经过Word2Vec处理得到预训练的词向量，并根据词映射成id的形式得到词汇字典，将训练集和测试集中分词后的结果使用字典替换成数字来得到训练集train\_list和测试集test \_list。

1. 模型设计

在模型层面使用了LSTM、GRU和CNN来进行对比，选取其中指标呈现效果最好的模型作为本实验的最终结果，具体设计内容在实验过程中叙述。

1. 模型训练

设置初始batch\_size和learning\_rate，通过weight\_decay和学习率衰减来防止过拟合，具体训练内容在实验过程中叙述。

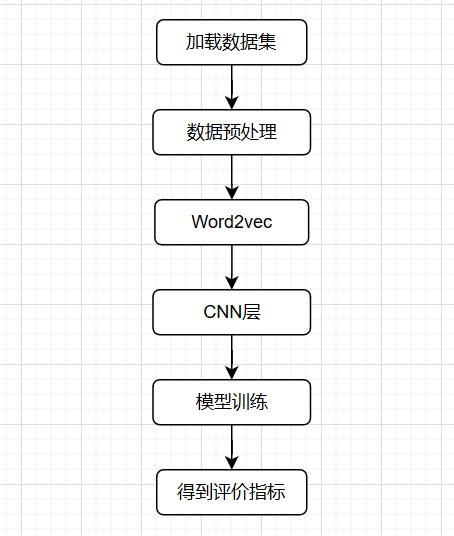


图2.1 疫情微博情绪分类实验流程图

## 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 |

1. 数据集来源

本数据集（疫情微博数据集）内的微博内容是在疫情期间使用相关关键字筛选获得的疫情微博，其内容与新冠疫情相关。

1. 数据集标签

每条微博被标注为以下六个类别之一：neural（无情绪）、happy（积极）、angry（愤怒）、sad（悲伤）、fear（恐惧）、surprise（惊奇）

1. 数据集规模

疫情微博训练数据集包括6,606条微博，测试数据集包含5,000条微博。

1. 数据集形式

数据集为json格式，包含三个字段：数据编号（id），文本（content），情绪标签（label）。示例：{"id": 11, "content": "武汉加油！中国加油！安徽加油！", "label": "happy"}

## 四、实验过程

### 4.1 数据集处理

1. 分词处理：

划分出文本（content）和情绪标签（label）字段，数据编号（id）字段舍弃。写一个函数，将文本进行分词处理sample\_list（含训练集和测试集），特殊符号、标点符号、英文等非词形式都使用停词‘Stop’表示。同理将情绪标签映射成id获得label\_list。

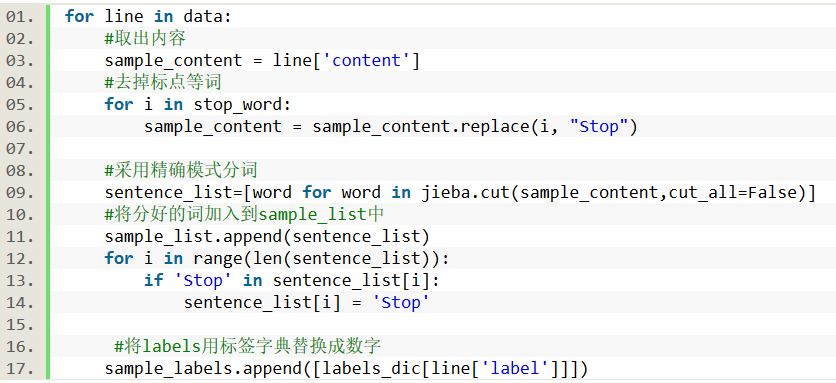


图4.1.1 分词处理代码图

1. 预处理词向量：

使用word2vec得到预处理的词向量，sg=1表示使用skip-gram算法，size=128表示输出维度（类似于embedding层的输出维度），window=5表示窗口大小（即看中心词的前5个词和后5个词），min\_count=3表示词频小于3的词会被丢弃。训练完后使用word2vec.save保存预训练词向量，使用Word2Vec.load加载预处理词向量。

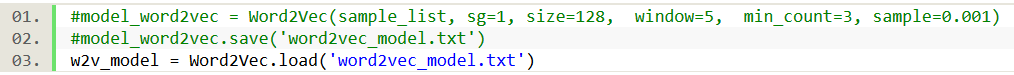


图4.1.2 预训练词向量代码图

1. 得到预处理的训练集和测试集：

使用w2v\_model.wv.vocab.keys()来查看所有词，并通过enumerate(vocab\_list)获得词字典。通过该字典中词到id的映射将训练集和测试集的每句话中的词都映射成id

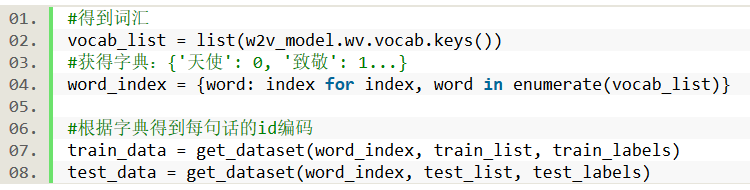


图4.1.3 预处理数据集代码图

1. 处理可变长度的序列：

由于每句话的长度不是像诗句一样的固定5个字和7个字，没办法输入到神经网络中，所以需要将每句话处理成相同的长度。在DataLoader中设置collate\_fn函数，并使用torch.nn.utils.rnn.pad\_sequence将每个batch的数据填充成一样长的，其中padding\_value就是数据预处理中‘Stop’停词所映射的id

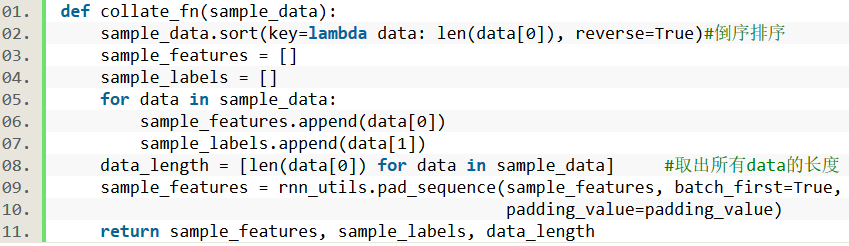


图4.1.4 处理变长序列代码图

### 4.2 模型设计

1. LSTM模型设计：

模型定义：由于训练集和测试集的样本数比较小，所以不需要太复杂的网络。首先设置一层embedding层，使用预处理的词向量去初始化embedding层，并设置不能梯度更新，其中embedding层的输入维度是词汇表大小7748，输出维度是预处理词向量的输出维度（二者要保持一致，不然无法初始化）。然后设置一层LSTM层，最后通过全连接层输出，全连接层的输出维度就是情绪标签的个数6。

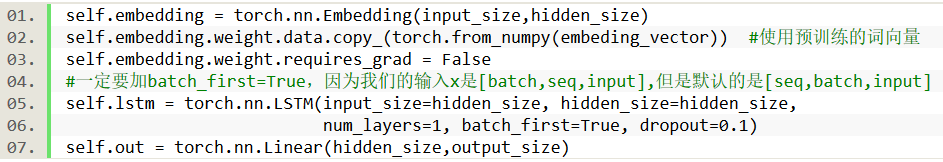
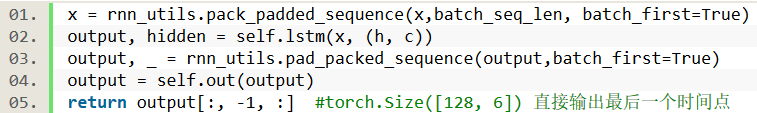


图4.2.1 LSTM模型定义代码图

前向传播：假设batch\_size是2，有一个序列是[1, 2, 3, 4, 5]，另一个序列是[1, 2]，那么经过填充后另一个序列是[1, 2, 0, 0, 0]，在经过LSTM运算时，第二个序列的0也参与了运算，实际上填充的部分并没有输出有用的数据, 而且会浪费算力资源，所以需要pack\_padded\_sequence进行压缩，按列来压缩则batch\_size就变成了[2, 2, 1, 1, 1]，经过LSTM后再使用pad\_packed\_sequence还原回来，还原后的batch\_size就是[5, 2]。



4.2.2 LSTM前向传播代码图

1. GRU模型设计：（与LSTM相同）

模型定义：由于训练集和测试集的样本数比较小，所以不需要太复杂的网络。首先设置一层embedding层，使用预处理的词向量去初始化embedding层，并设置不能梯度更新，其中embedding层的输入维度是词汇表大小7748，输出维度是预处理词向量的输出维度（二者要保持一致，不然无法初始化）。然后设置一层GRU层，最后通过全连接层输出，全连接层的输出维度就是情绪标签的个数6。

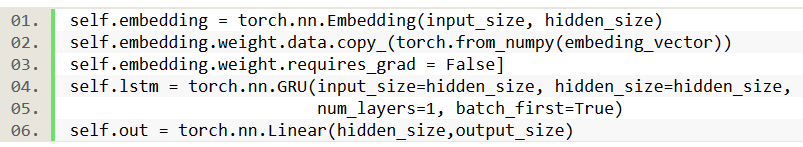
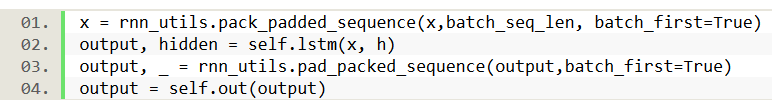


图4.2.3 GRU模型定义代码图

前向传播：在经过GRU层前也需要 pack\_padded\_sequence进行压缩，经过GRU后再使用pad\_packed\_sequence还原回来。



4.2.4 GRU前向传播代码图

1. CNN模型设计：

模型定义：设置两个embedding层并使用预训练词向量来初始化，其中一个参与训练，另一个不参与训练，权重固定。设置dropout=0.5和输出的全连接层。GlobalMaxPool1d表示时序最大池化层，torch.nn.ModuleList()用于创建多个一维卷积层。由于前向传播会将两个embedding层连接，所以一维卷积层的in\_channels是2倍的hidden\_size，out\_channels是设置的[30, 30, 30]（即每一个一维卷积层的out\_channels都是30），kernel\_size设置的[3, 4, 5]

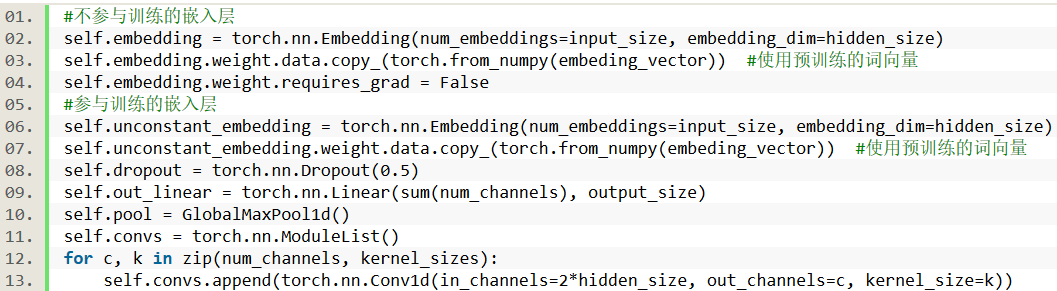


图4.2.5 CNN模型定义代码图

前向传播：

将两个形状是（batch\_size，seq\_size，emb\_size）的embedding层的输出按词向量连接，就会得到（batch\_size，seq\_size，2\*emb\_size），使用permute将词向量变化到前维就是（batch\_size， 2\*emb\_size，seq\_size），因为一维卷积是在最后维度上扫的。

经过relu激活函数和池化层后，维度变成了（batch\_size， out\_channels，1），所以需要使用squeeze压缩一下。

最后经过全连接层输出。由于有三层卷积层，每一层卷积层的out\_channels是num\_channels这个列表里的数，所以最后的全连接层的input其实就是sum(num\_channels)

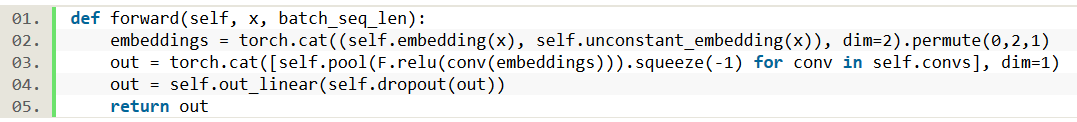


图4.2.6 CNN前向传播代码图

### 4.3 模型训练

1. 参数设置：

从预训练的词向量中得到权重信息（使用w2v\_model.wv.vectors可得到），从权重信息中可得到词典的大小7748（权重信息的第一维）和词向量隐藏层单元个数128（权重信息的第二维）。

设置三层卷积层，卷积核的大小分别为3、4、5，输出通道分别为30、30、30。

加载之前还未训练好的模型继续训练并加载权重参数；设置优化器，学习率最开始为0.1，逐渐减小；batch\_size从1024开始逐渐减小。

因为主要关注测试集上的指标变化，所以开始训练后设置保存模型参数的条件：测试集的loss低于当前的最小值就保存模型。

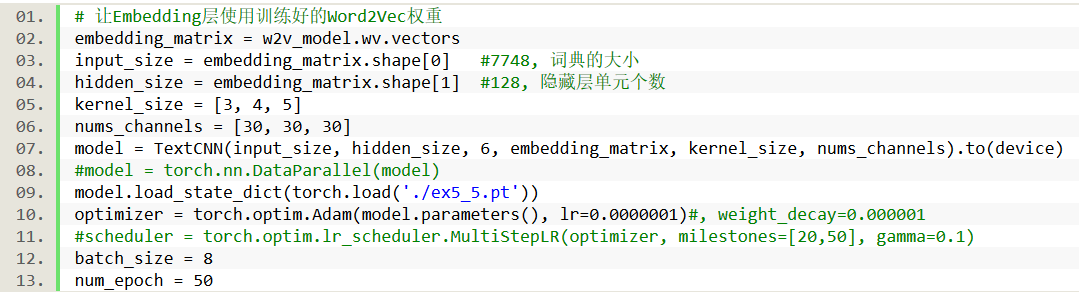


图4.3.1 参数设置代码图

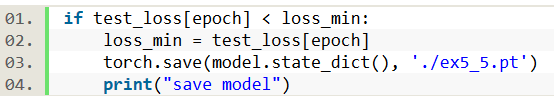


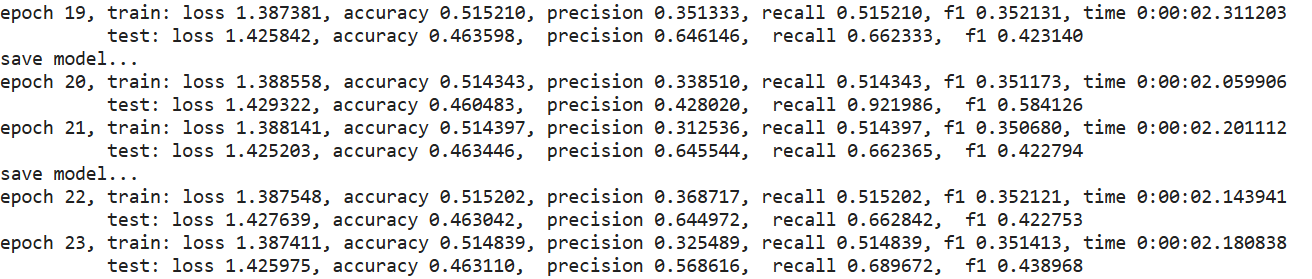
图4.3.2 模型保存条件代码图

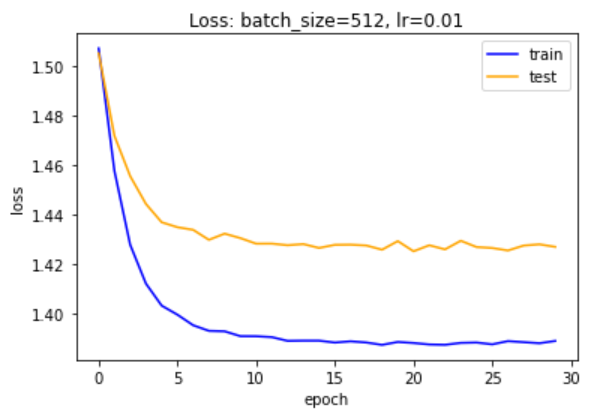
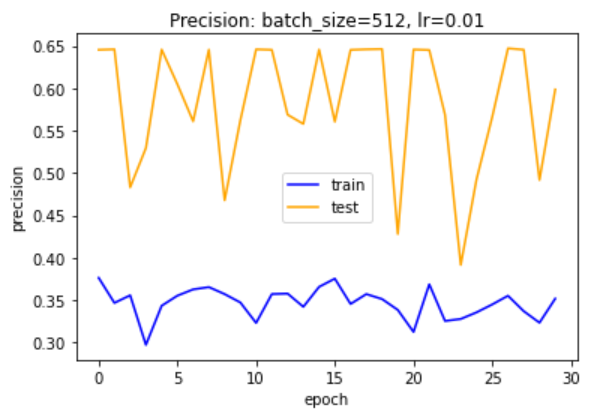
## 五、实验结果

### 5.1 LSTM实验结果

如图5.1.1所示，LSTM模型的loss由最开始的1.51经过20个epoch降到1.42；训练集的F1保持在0.35左右，测试集的F1在0.4-0.6之间波动，此时的batch\_size=512，lr=0.01。

经过大约500次的epoch（其中有调整batch\_size、lr和weight\_decay来使得模型不过拟合并且有更好的学习效果），训练集loss降到了1.383230，测试集loss降到了1. 419484；训练集F1升到了0.376912，测试集F1升到了0.478589，相比起前30个epoch变化不大，且容易过拟合。



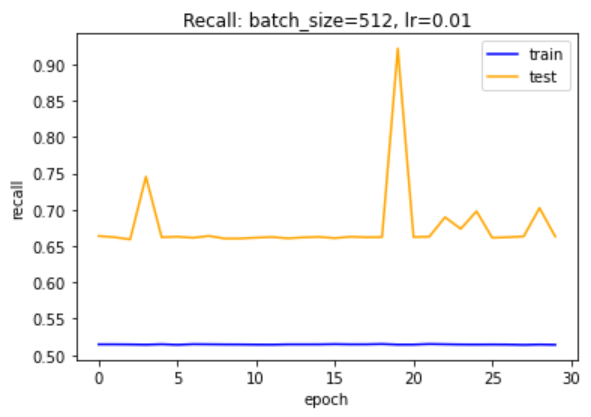
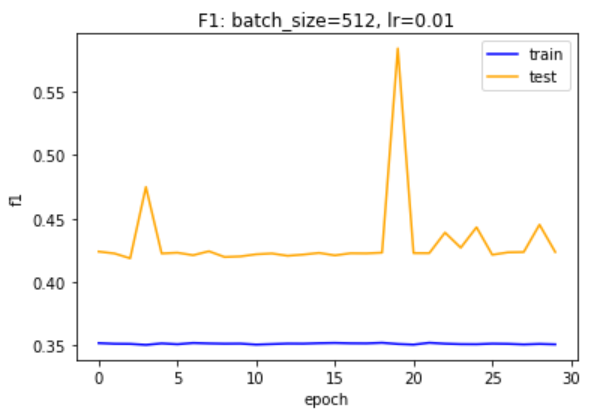
 

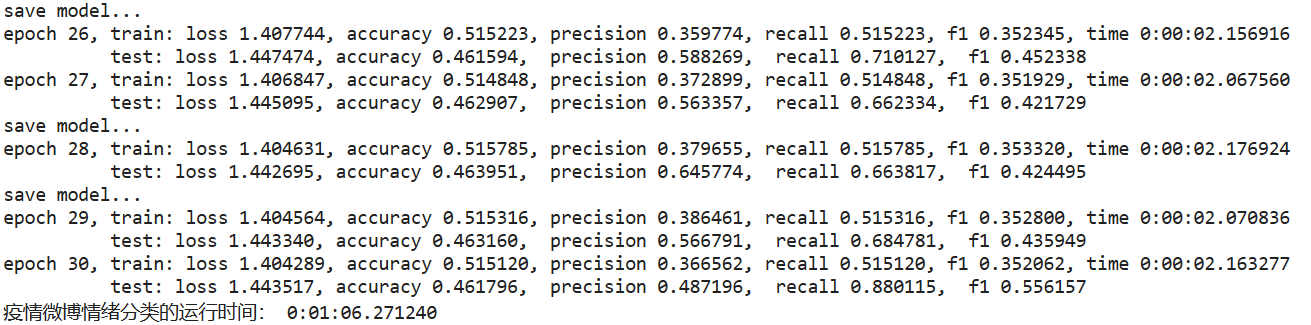
图5.1.1 LSTM训练结果图

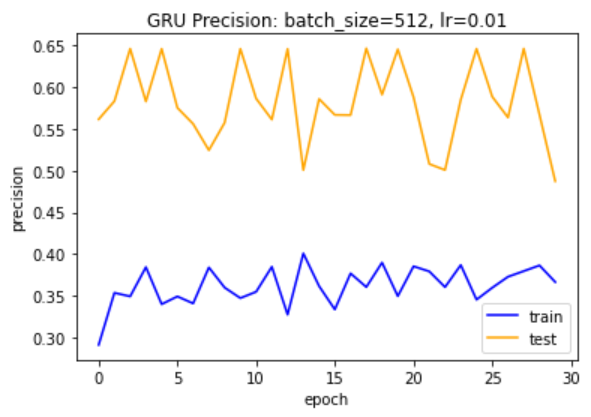
原因分析：该模型的数据集比较小，只有6,606条训练集和5,000条测试集，而LSTM模型太复杂了，导致了过拟合。虽然LSTM层数已经由3层降为了1层，但是还是有过拟合的趋势。

### 5.2 GRU实验结果

如图5.2.1所示，GRU模型的loss由最开始的1.575经过30个epoch降到1.400；训练集的F1保持在0.35左右，测试集的F1在0.4-0.6之间波动，此时的batch\_size=512，lr=0.01。

经过大约500次的epoch（其中有调整batch\_size、lr和weight\_decay来使得模型不过拟合并且有更好的学习效果），训练集loss降到了1.405206，测试集loss降到了1.427701；训练集F1升到了0.378561，测试集F1升到了0.490252，相比起前30个epoch变化不大，但是比LSTM效果要好一些。



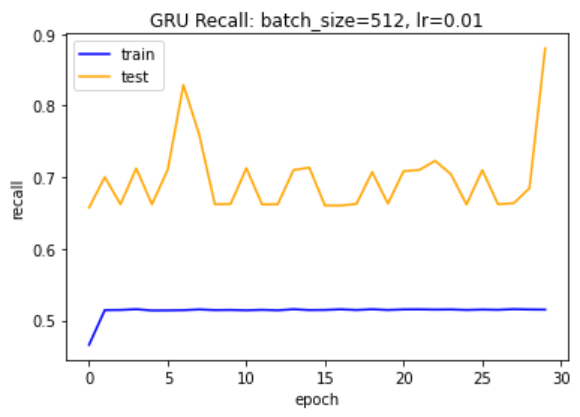
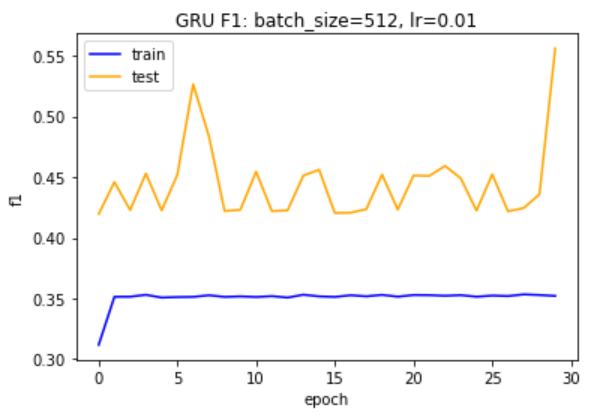
 

图5.2.1 GRU训练结果图

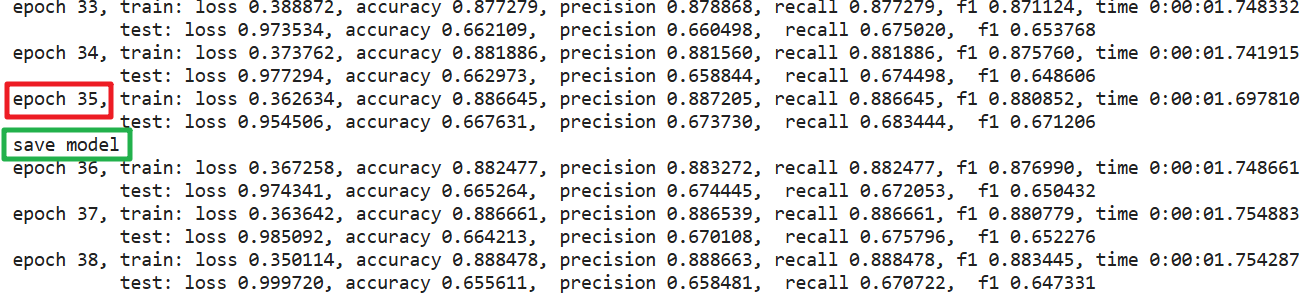
原因分析：该模型的数据集比较小，只有6,606条训练集和5,000条测试集，而GRU模型太复杂了，导致了过拟合。虽然LSTM层数已经由3层降为了1层，但是还是有过拟合的趋势。

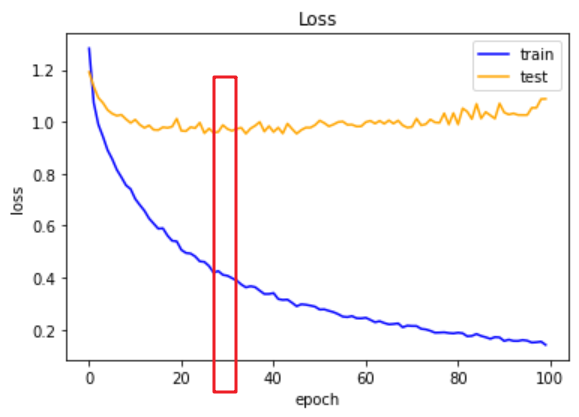
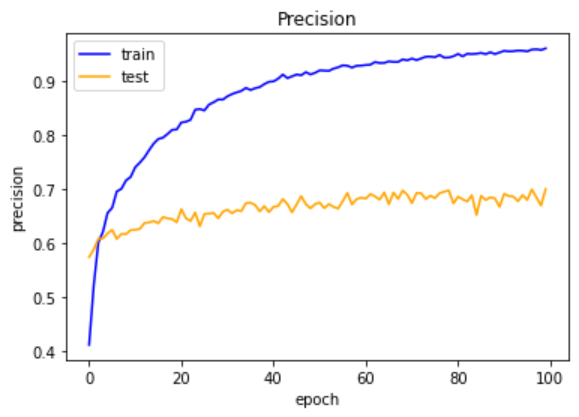
总结：从loss的变化趋势以及precision、recall和F1的数值上看，相比起LSTM，二者差别不大，都是由于模型太复杂从而导致的训练效果差。

### 5.3 CNN实验结果

如图5.3.1所示，前50个epoch训练集的超参数设置为batch\_size=512，lr=0.01。可看出loss由1.3降到了0.2，测试集loss由1.2降到了1.0，由于在35epoch左右测试集开始有过拟合的趋势，所以及时止损，在epoch35的时候保存此时的模型参数用于后面继续训练。

Precision、Recall和F1的变化幅度基本一致。训练集的Precision由最开始的0.4升到0.95，测试集的Precision由最开始的0.57升到0.67；训练集的Recall由最开始的0.5升到0.95，测试集的Recall由最开始的0.6升到0.68；训练集的F1由最开始的0.4升到0.95，测试集的F1由最开始的0.57升到0.67。



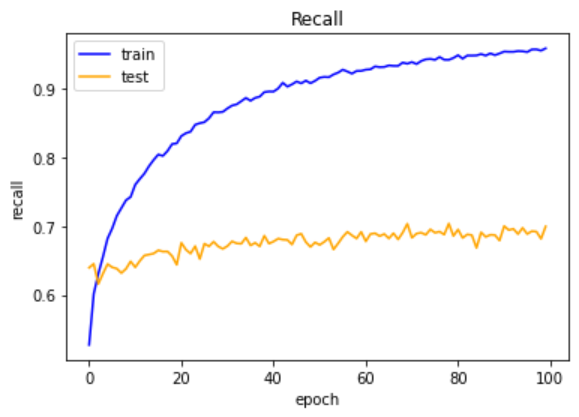
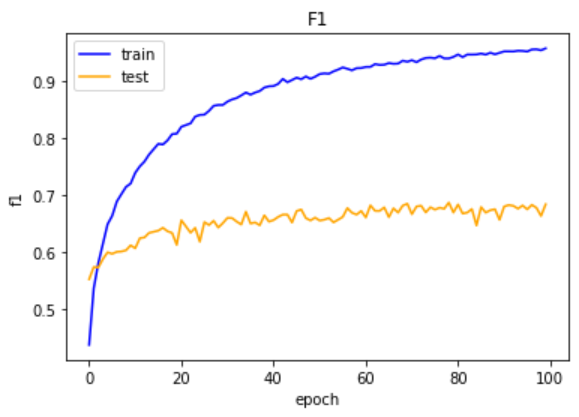
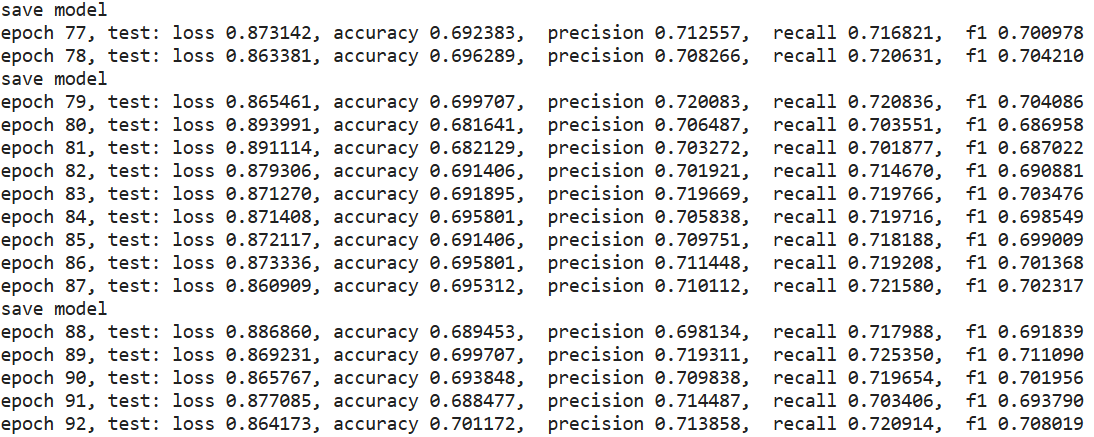
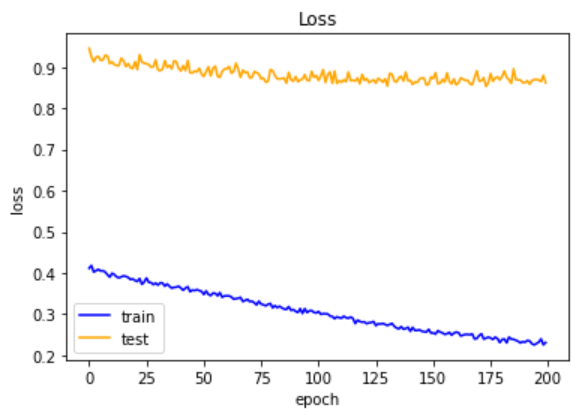
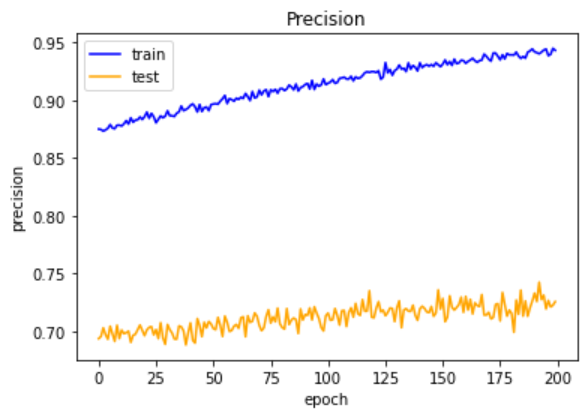
 

图5.3.1 前50epochCNN训练结果图

前50个epoch总结：loss大幅度下降，Precision、Recall和F1在训练集上效果比较明显，在测试集上效果不明显但是也在稳定小幅度上升。这是由于CNN模型比较简单，适合于该数据量小的数据集。如果一开始就将batch\_size调至1024，很容易导致过拟合，经过测试：一开始batch\_size=512的时候实验结果是最好的，经过50个epoch之后再逐渐降低batch\_size和lr。



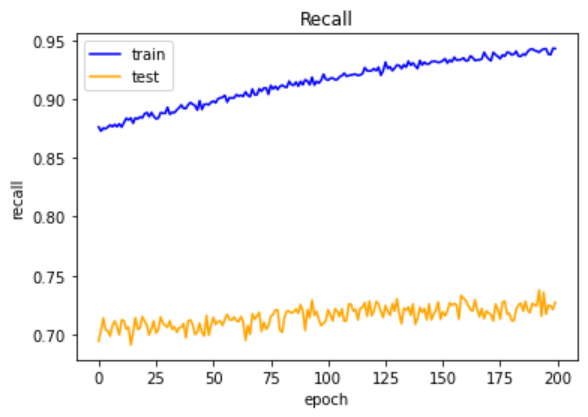
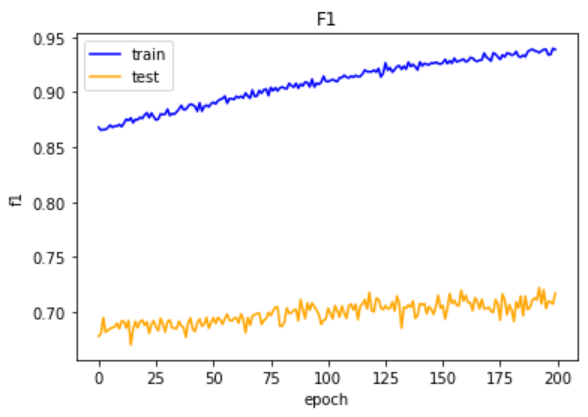
 

图5.3.2 900-1000次epoech的CNN训练结果图



图5.3.3 最好的训练结果图

从图5.3.2和图5.3.3中可以得到训练趋于饱和，训练集loss最低能到0.257，测试集loss最低能到0.849；训练集Precision最高能到0.940，测试集Precision最高能到0.805；训练集Recall最高能到0.937，测试集Recall最高能到0.791；训练集F1最高能到0.932，测试集F1最高能到0.771。

总结：使用CNN训练该疫情微博数据集效果最好，训练集的各项指标均能达到90%以上，测试集的各项指标在77~80%左右。

## 六、实验心得体会

1. CNN也可以用于处理时序问题。对于一些数据量没有那么大的数据集来说，使用CNN效果可能会比LSTM、GRU等好。
2. 处理变长序列的方式：
3. torch.nn.utils.rnn.pad\_sequence()：把不等长的tensor数据, 补充成等长的tensor数据。
4. torch.nn.utils.rnn.pack\_padded\_sequence()：把等长的tensor根据所输入的参数压缩成实际的数据, 同时数据格式变成PackedSequence
5. torch.nn.utils.rnn.pad\_packed\_sequence()：把压缩成PackedSequence的数据还原成tensor类型, 并补成等长的数据
6. class torch.nn.Conv1d(in\_channels, out\_channels, kernel\_size, stride=1, padding=0, dilation=1, groups=1, bias=True)：一维卷积是在最后维度上扫的

## 七、参考文献

无。

## 八、附录

代码文件详情：

ex5\_lstm.py：使用LSTM实现实验5

ex5\_gru.py：使用GRU实现实验5

ex5\_TextCNN.py：使用CNN实现实验5

ex5\_0.pt：LSTM模型的结构和参数

ex5\_1.pt：GRU模型的结构和参数

ex5\_2.pt：CNN模型的结构和参数

word2vec\_model.txt：Word2Vec的结构和参数（通过gensim.Word2Vec生成）

#word2vec.py：简单实现Word2Vec（实验中未使用到）

#negative\_sampling.py：带有负采样的Word2Vec（实验中未使用到）

**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。