文本情感分析 实验报告

洪昊昀　计82　2017010591

1. 数据处理

1.1 数据清洗

仔细观察所给的文本数据，可以发现文本不仅完成了分词，而且如“的”、“了”等没有具体含义的中文停用词已经被去除了，这为数据清洗提供了便利，不过文本中的数字和英文比较多，它们不包含中文的情感信息，所以需要去掉。

1.2 数据映射

将8种情感标签映射为0-7个类别，然后根据数据归一化训练集和测试集的情感分布，便于在后续过程中求相关系数。将文本中的词对应到实验说明文档链接提供的预训练词向量（sgns.merge.word文档）中的序号上，用词语序号的序列代表整个文本。因为有些文本的词语序列长度很长，甚至达到2000以上，那么输入的代表文本的数据量会太大，而且较短的文本padding后会使词向量很稀疏，所以：

1. 对于在cpu上训练较快的CNN，我选择1000作为词语序列长度，因为去掉不在预训练词向量中的词语后，训练集和测试集中95%以上的样本的词语序列长度都不高于1000；
2. 对于在cpu上训练较慢的RNN，我选择300作为词语序列长度，如果同时去掉不在预训练词向量中的词语和出现次数不多于2次的词语（认为是所含信息不大的词语）后，训练集和测试集中80%以上的样本的词语序列长度都不高于300；
3. 对于作为baseline的MLP，因为我担心词向量会比较稀疏，所以取500作为词语序列长度，然后将500个词向量作平均作为整个文本的代表，相当于不考虑词序关系，认为所有词语对文本的贡献一致，再将平均词向量作为文本向量传入网络中。

最终3种方法都取得了相对不错的效果。

1.3 样本类别分布不均

数据的情感标签类别共有8种，其中训练集的情感类别分布比例为

```
{'感动': 0.1776, '同情': 0.0529, '无聊': 0.0619, '愤怒': 0.4202, '搞笑': 0.1567, '难过': 0.0769, '新奇': 0.0423, '温馨': 0.0115}
```

测试集的情感类别分布比例为

```
{'感动': 0.1495, '同情': 0.0453, '无聊': 0.0575, '愤怒': 0.4776, '搞笑': 0.1562, '难过': 0.0785, '新奇': 0.0305, '温馨': 0.0049}
```

可以看到样本种类的分布很不均匀，其中“愤怒”占了很大的比重。我担心样本类别分布不均会影响训练的准确率，为了不损失训练集中的信息，我使用过采样的方法，就是将数量较少的类别的样本复制几份，使得样本类别分布较均匀，但是结果并不如我所愿，与用原始训练集相比，过采样后反而十分容易过拟合。反思后，我觉得这主要是因为我单纯地把训练集中的样本复制了多份，导致训练集中某些样本，这些样本就容易被过拟合，而原始训练集中的情感类别分布和测试集的情感类别分布相近，所以直接用原始训练集训练再预测测试集比过采样后分布平均的训练集效果更好。

2. 实现思路与评价指标

2.1 文本表示方法

我的文本表示方法是word embedding方法，因为我认为它能保留词义信息（主要是含有情感信息的近义词的信息），将tokenize后的词序号向量作为输入也可以保留词序信息，而且观察原始数据后，我觉得原始数据中已经去掉大部分停用词了，所以我不使用表示能力不如word embedding方法的Bags-of-words 和 TF-IDF 特征表示方法。

2.2 标签表示方法

我的标签表示方法是用最大值转为单标签，也就是将该问题看作分类问题。因为我觉得在表达情感数比较少的情况下，有些网民可能会给出不准确的情感标签，这就导致若将该问题看作回归问题，会被很多脏数据干扰，但是最大值转为单标签就可以减少这种噪声的影响。

2.3 评价指标

我使用了文档里的Accuracy、F-score 、相关系数作为评价指标，其中F-score是weighted F-score，因为样本类别分布很不均，所以适合用weighted F-score作指标，相关系数的求法是在网络最后用softmax函数映射到8维的标签向量上，每个标签都有自己的概率，即该文本对应每个情感标签的概率，然后我用这个概率向量作为真实标签数量分布的预测，再与真实的归一化的标签分布求相关系数。

3. 模型比较、参数分析与实验效果

对于参数的分析：

1. 在所有模型中我的dropout层参数均为0.5，因为这代表每个神经元有1/2的概率会失活，这样所有的神经元的排列组合数取得最大值，这就意味着随机性更大，符合dropout的初衷，太小的dropout会导致过拟合，太大的会导致训练收敛过慢。
2. 在所有模型中的Batch size均为16，它比64表现好，64似乎会落入奇怪的次优点，可能是因为64个样本中“愤怒”的个数很多，会导致预测

结果倾向于都输出“愤怒”，但是batch size 取1又会使各项指标振荡比较严重，所以取16比较合适。

3. 我的优化器为Pytorch的Adam，因为它可以自适应调整学习率，学习率的初始值我就设为默认的0.001，我发现学习率太大会使各项指标振荡很严重，学习率太小会使训练收敛很慢。
4. 对于模型规模参数的选择，其实我并没有尝试很多，因为只能用cpu训练，gpu容量太小而模型embedding层参数太多，综合训练时间、空间和准确率，只能选择合适范围内比较大的值了，模型中参数太多会导致过拟合严重，太少会导致欠拟合。
5. 这个训练集的样本量并不大，只有2342条数据，所以训练的epoch次数应当会比较多，我以300个epoch为最大的轮数，综合准确率和时间，如果10个epoch后验证集指标上升不大，就停止停止训练，选择最好的模型保存为最终模型。

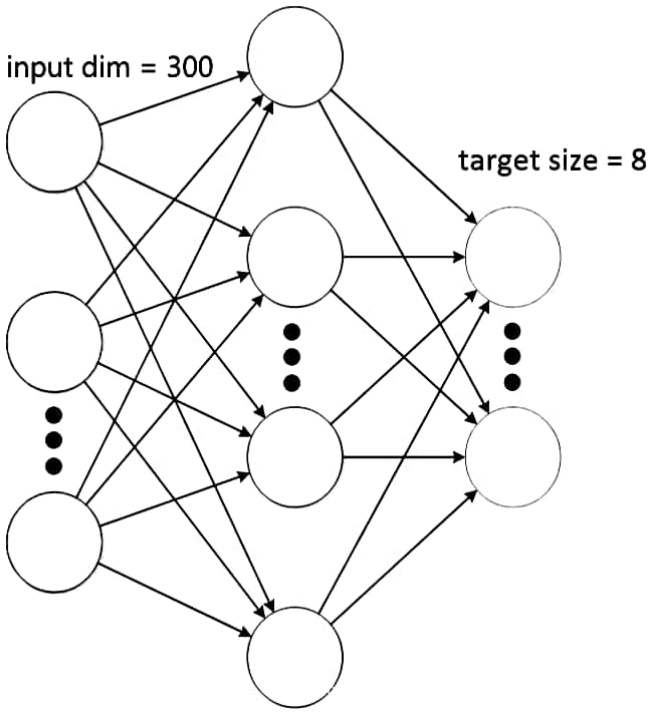
其它参数请见下面关于每个模型的分析。

对于3个指标Accuracy、weighted F-score 、相关系数，我发现在训练过程中，有时虽然Accuracy没有上升，甚至因为振荡有些下降，但是weighted F-score或者相关系数会上升，所以我觉得多个指标评测模型的效果是很有必要的。

3.1 MLP（baseline)

3.1.1 结构图与流程分析

我的MLP结构图大致如下图所示，参数也可见Pytorch输出的信息。



```
MLP(
  (linear1): Linear(in_features=300, out_features=10,
bias=True)
  (dropout): Dropout(p=0.5, inplace=False)
  (linear2): Linear(in_features=10, out_features=8,
bias=True)
)
```

首先，我将每个样本中截取的500个词语的词向量的平均作为整个文本的代表，相当于压缩了词序关系，认为所有词语对文本的贡献一致，再将300维平均词向量作为文本向量传入第一个全连接层，hidden layer的维度为10，是比较小的，因为最开始我觉得文本情感分析没有蕴含很多的特征，hidden layer的维度太高会导致严重的过拟合，再经过一个dropout层，防止全连接层因为参数过多而过拟合，然后经过ReLU函数后，再将向量传入第二个全连接层，再经过softmax后得到标签概率，选择最大概率者为最终分类标签。

3.1.2 结果

共训练了120个epoch，选择表现最好的第112个epoch。

模型在测试集上的Accuracy = 55.61%，weighted F-score = 42.86%，相关系数 = 0.5138。

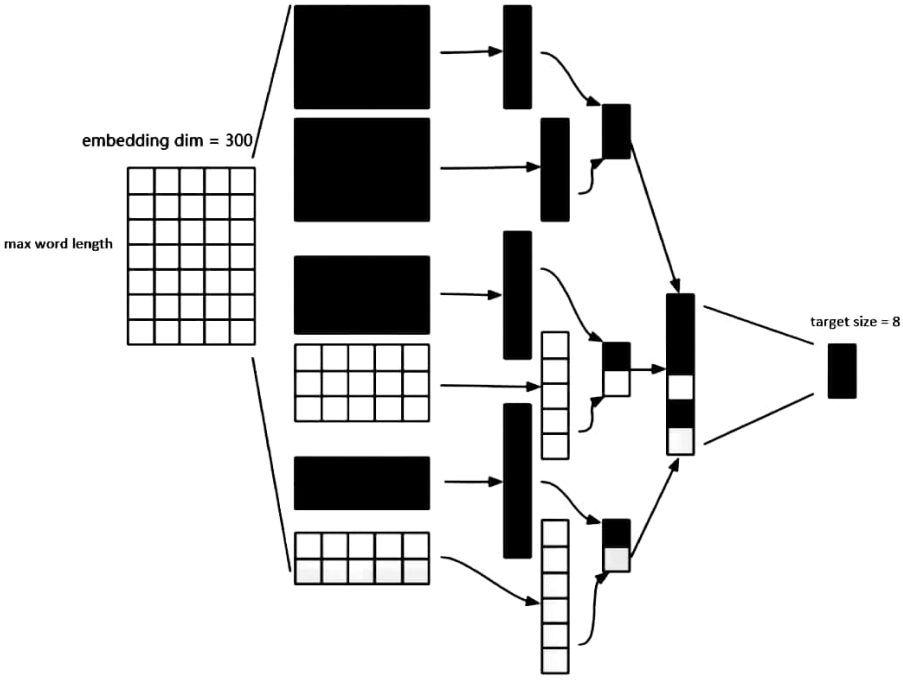
值得一提的是训练集上的Accuracy与测试集相差不多，说明我的MLP的拟合能力还比较有限，也许增加隐藏层维数和增加训练时间会有帮助，但是各项指标在10个epoch内都几乎没有变化了，我判断它接近收敛了，所以应当是增加隐藏层维数或者其它的模型上的复杂度会更有帮助。

（参看Readme的说明，从清华网盘上下载后，模型文件`MLP_model.ckpt`位于`checkpoints`文件夹中，可以运行`python MLP_eval.py`进行模型测试，本机测试结果如上。）

3.2 CNN

3.2.1 结构图与流程分析

我的CNN结构图大致如下图所示，其中我使用了3种卷积核，长为词向量长度即300，宽分别为2，3，4，代表提取2元词、3元词、4元词的信息，虽然论文Convolutional Neural Networks for Sentence Classification 中的卷积核宽度选的是3，4，5，但是经过测试后发现3，4，5的效果没有2，3，4好，我觉得这可能是中文和英文的区别，中文中用2元词表达完整情感的词组比较多，但是英文词组中单词会比较多，所以中文情感分析使用2，3，4元词信息会比较合适。其它参数也可见Pytorch输出的信息。




```
CNN(
  (embedding): Embedding(1291383, 300)
  (convs): ModuleList(
    (0): Conv1d(1, 100, kernel_size=(2, 300), stride=(1,))
    (1): Conv1d(1, 100, kernel_size=(3, 300), stride=(1,))
    (2): Conv1d(1, 100, kernel_size=(4, 300), stride=(1,))
  )
  (dropout): Dropout(p=0.5, inplace=False)
  (linear): Linear(in_features=300, out_features=8,
bias=True)
)
```

首先，我将样本的词序列通过embedding层映射到预训练的300维的词向量上，然后经过3种各100个的卷积核的卷积层，再ReLU整流，然后用最大池化获得最重要的影响因子，再经过dropout层，再加一层全连接层映射到标签空间，最后用softmax得到标签概率，选择最大概率者为最终分类标签。

我发现CNN 之后加dropout层会表现得更好，而且收敛得更快，这可能是因为dropout层相当于提供了更多种类的网络。我还发现在末尾加1层全连接层的效果比加2层好，因为1层全连接层就是把CNN的预测结果映射到标签空间上，2层+ReLU反而损失了信息，导致全预测为“愤怒”。

3.2.2 结果

用cpu，每个epoch耗时2分钟，训练了110个epoch，选择表现最好的第102个epoch。

模型在测试集上的Accuracy = 57.81%，weighted F-score = 52.57%，相关系数 = 0.5717。

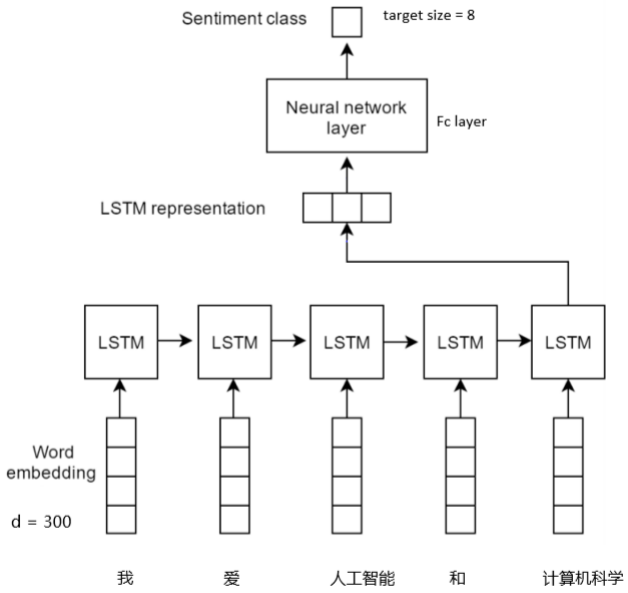
虽然Accuracy没有比MLP高太多，但是weighted F-score和相关系数高了不少，说明CNN的能力还是比MLP强的。

（参看Readme的说明，从清华网盘上下载后，模型文件`CNN_model.ckpt`位于`checkpoints`文件夹中，可以运行`python CNN_eval.py`进行模型测试，本机测试结果如上。）

3.3 RNN—LSTM

3.3.1 结构图与流程分析

我的LSTM结构图大致如下图所示，参数也可见Pytorch输出的信息。




```
LSTM(
  (embedding): Embedding(1291383, 300)
  (lstm): LSTM(300, 100)
  (dropout): Dropout(p=0.5, inplace=False)
  (fc): Linear(in_features=200, out_features=8, bias=True)
)
```

首先，我将样本的词序列通过embedding层映射到预训练的300维的词向量上，然后经过1个单层单向的LSTM层，hidden dim据说取100~300之间普遍比较好，我取了100，再经过dropout层，然后将第一个和最后一个LSTM cell的state首尾相连，这样可以同时利用开头和结尾的信息，再传入全连接层映射到标签空间，最后用softmax得到标签概率，选择最大概率者为最终分类标签。Pytorch的LSTM中的各个参数顺序与想像中有些颠倒，所以要注意在前向传播时进行张量维度的交换。

3.3.2 结果

用CPU训练，每个epoch耗时8分钟，训练了70个epoch，选择表现最好的第47个epoch。

模型在测试集上的Accuracy = 55.43%，weighted F-score = 46.79%，相关系数 = 0.5130。

GRU和BiLSTM的效果并不好，时常维持在test 42.02%，train 47.76%，相当于全输出“愤怒”，但是换成单向LSTM后效果就好一些，可能是因为BiLSTM中参数比较多，很容易进入次优点，比较悲伤的是，花了很长时间训练的LSTM却和MLP的效果差不多，可能是因为计算资源的限制导致epoch不够多，或者是模型还不够复杂，能力还比较弱，也可能是因为文本中存在很多与情感类别无关的噪声信息，导致LSTM的结果不够好。

（参看Readme的说明，从清华网盘上下载后，模型文件`LSTM_model.ckpt`位于`checkpoints`文件夹中，可以运行`python LSTM_eval.py`进行模型测试，本机测试结果如上。）

3.4 实验结果

最终，我保留了3个比较好的模型：

模型	ACCURACY(%)	WEIGHTED F-SCORE(%)	相关系数
MLP	55.61	42.86	0.5138
CNN	57.81	52.57	0.5717
LSTM	55.43	46.79	0.5130

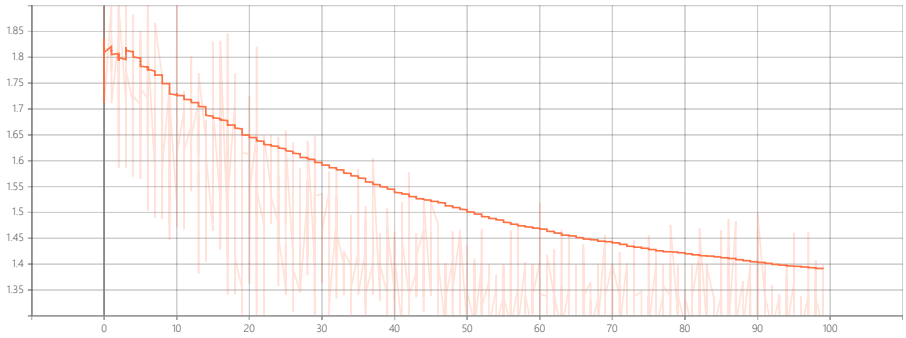
4. 问题思考

1） 实验训练什么时候停止是最合适的？简要陈述你的实现方式，并试分析固定迭代次数与通过验证集调整等方法的优缺点。

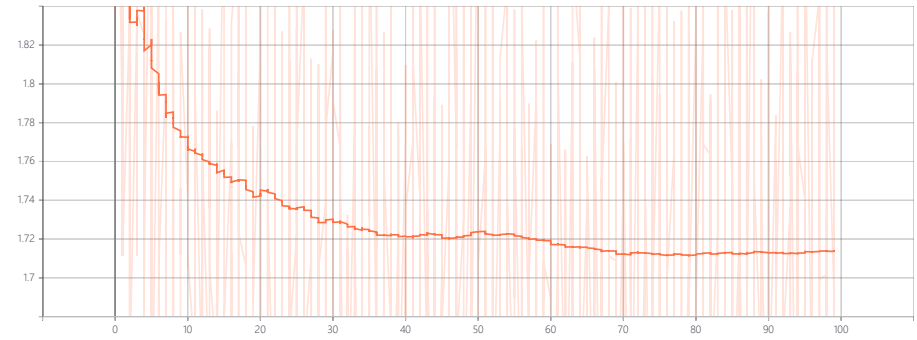
我是通过验证集调整的，若计算机配置较好或模型简单，可以多训练几个epoch，因为我只能用cpu训练，所以时间比较长，因此我的停止标准是比较简单，即如果10个epoch后验证集指标上升不大，就停止。但如果条件允许，最好应该在过拟合之前才停止，然后选取效果最好的模型，这样结果会更好。

比如我用tensorboard对用Pytorch训练的训练集和验证集的loss进行可视化：

Train Loss：



Valid Loss:



可以看到验证集loss曲线后面部分趋向于不变，其它指标改变也很小，然后我就停止训练。

固定迭代次数：优点是可以直观比较不同模型在同样的epoch下的各项指标判断模型的好坏，缺点是比较耗时。

通过验证集调整的方式：优点是在计算机配置不够好或模型很复杂时，可以在比较短的时间内选择出比较好的模型；缺点是每种模型训练的epoch数很可能不一样，不容易直观比较模型的好坏。

2）实验参数的初始化是怎么做的？不同的方法适合哪些地方？（现有的初始化方法为零均值初始化，高斯分布初始化，正交初始化等）

我的模型中的参数都是使用 PyTorch 的默认初始化。

经过查阅资料，我明白了均匀分布和高斯分布的零均值初始化一般用于初始化线性层、卷积层。PyTorch 的线性层的初始化为均匀分布。正交初始化适用于 RNN 中参数的初始化，可以解决递归网络的梯度消失和梯度爆炸问题。

3）过拟合是深度学习常见的问题，有什么方法可以防止训练过程陷入过拟合。

可以用Batch Normalization，或者加Dropout层，还有使用验证集观察，若验证集loss为上升趋势，训练集loss很小，就过拟合了，可以选择过拟合前的最好模型，再调小学习率，进行更精细的训练。

4）试分析 CNN，RNN，全连接神经网络（MLP）三者的优缺点。

	MLP	CNN	RNN
优点	结构简单，可解释性相对其它网络较强，能学习全局信息，能用GPU矩阵加速	学习与位置无关的局部特征时优势很大，能用GPU矩阵加速，模型中的参数比较少，不容易过拟合，在有关图像的训练中每个卷积核可视化后有较强的可解释性	输入长度可变，有时序性和记忆能力，尤其是LSTM对距离较远的上下文也可以进行联系
缺点	输入大小固定，学习能力较弱，一般而言模型中参数规模很大，容易过拟合	超参数很多，调参不容易，不能获取全局信息	模型中参数很多，容易梯度消失和梯度爆炸，除了单元内部的矩阵乘法之外无法用GPU矩阵加速，训练较慢，可解释性较差

5. 心得体会

本次大作业对我来说挑战很大，因为这是我第一次接触深度学习和深度学习框架，我学习了很多课外资料来加深对MLP、CNN、RNN的理解，也从toy model开始学习Pytorch，虽然过程中踩了很多的坑，但是在老师、助教、同学的帮助下也得到了不错的结果。这次作业使我受益良多，让我对使用深度学习网络解决问题有了信心，再次感谢老师、助教对我的关心与帮助！