基于深度学习的电影评论情感分类

姓名学号： 王程钥 17341146
联系方式： wangchy56@mail2.sysu.edu.cn
日期： 2020-7-15

摘要 本文使用了LSTM模型完成了电影情感分类的任务，并尝试在模型的嵌入层中加入预训练的Glove词向量。本文尝试了多种长度的词向量表示，多种不同的预训练方式，注意到了类别不均衡的问题，并尝试了dropout的作用。最终使用本文实现的最优算法产生的分类结果提交到kaggle的电影评论情感分类比赛中，可以获得63.099的分数。

1 导言

1.1 背景

文本分类是自然语言处理领域中的一个经典问题，它根据一个已经被标注的训练文档集合，找到文档特征和文档类别之间的关系模型，然后利用这种学习得到的关系模型对新的文档进行类别判断。

文本分类的方法最早可以追述到基于专家系统的方法，不过由于专家系统在覆盖面上的局限性和数据库过于庞大，这种方法的准确率非常有限，适用性也并不高。后来逐渐出现基于统计学习的方法，这种方法利用人工构造的特征工程和浅层的机器学习分类器（如SVM，LR，Naive Bayes）进行分类。不过由于人工构造的特征表达能力较弱，且特征需要人工构造，人力成本较高，所以这种方法也存在局限性。

1.2 相关工作

随着近年来深度学习的崛起，尤其是在计算机视觉（CV）等领域的广泛应用，一定程度推动了深度学习算法在自然语言处理（NLP）领域的发展。基于深度学习的算法在文本分类任务上也取得了不错的效果。TextCNN[2]将卷积神经网络应用到文本分类任务中，利用多个不同size的kernal来提取句子的关键信息，能够很好地捕捉局部相关性。不过它也存在局限性，善于捕捉较短文本的信息，对于长文本表现一般。TextRNN[3]则将循环神经网络应用到文本分类任务中。相较于CNN，RNN能够处理变长数据，对长文本的表现更好。不过当文本过长时RNN同样存在信息遗忘的问题，在单个模型中反复反向传播也很容易导致模型梯度爆炸。LSTM[1]对RNN进行改进，它引入了门机制，通过这种方式一定程度上解决了梯度爆炸的问题。近年来基于Transformer[4]的算法开始流行，这种算法完全抛弃了RNN的结构，利用Self-Attention的机制，并取得了不错的效果。这种算法的优势在长文本，段落上更是尤其明显。

另外，随着计算机算力的逐年提升，迁移学习近年来也快速发展。从Word2Vec[5]开始，预训练词向量成了基于深度学习的自然语言处理算法中不可获取的一部分。Glove[6]将上下文信息和文本全局信息进行结合，取得了不错的效果。BERT[7]基于Transformer模型和Google庞大的数据构造了一个预训练数据库，基于pre-training和fine-tuning成功横扫NLP领域的11项下游任务。

1.3 我们的工作

本文需要实现的是一个基于电影评论的情感分类的模型，这也算是一个文本分类的问题。鉴于本次实验的文本大体上较短，至少一半的句子单词数小于5，最长的句子包括标点符号也只有53个单词，如果使用基于Transformer的算法，预计会又慢又难调还会过拟合。因此本文最终选择了更常见的LSTM+Glove算法进行实现，并取得了不错的效果。

2 实验过程

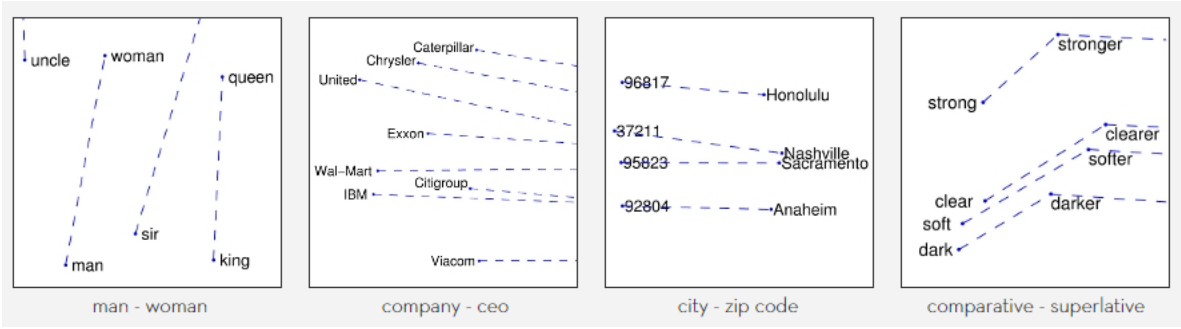
2.1 文本预处理

因为本次实验使用的是英文文本，而英文不需要额外的分词操作，文本预处理部分是比较简单的。对于读入的文本，先把所有大写字母转换为小写，后去掉缩写（如：it's → it is）得到待处理的句子集。最后直接根据空格分词，构造词典并将单词转化成编号即可。

2.2 词嵌入（Word Embadding）

在词嵌入算法盛行之前，单词一般用one-hot向量构成的矩阵或者TF-IDF矩阵表示。这种表示方式把单词当作一个个独立的个体，但其实单词之间是存在联系的。而且如果单词种类很多，使用这种方式会导致矩阵过大，占用过多显存，影响效率。

词嵌入（Word Embadding）是一种单词映射的做法，可以把单词从原始的数据空间映射到新的低维空间上去。这种做法可以让one-hot表示的单词向量迅速降维，而且如果进行合理预训练，还可以表示出单词间的语义关系。



2.2.1 Word2vec

Word2Vec[5]是最早提出的一种词嵌入方法。这种方法是自监督的，不需要额外标注，使用文本本身的上下文信息进行训练。Word2Vec主要有Skip-Gram和CBOW两种形式，其中Skip-Gram是给定输入单词预测上下文，CBOW则是给定上下文预测中间的单词。

两种方法都是构造一个encoder-decoder的结构，假设词典的大小是N嵌入后维度为D，则先接一个$N*D$的全连接层降维，再接一个$D*N$的全连接层升维分类，最后只取前一个全连接层即可生成词向量。对于Skip-Gram方法，输入是中心单词的one-hot向量，输出对应的label是下一个单词的one-hot向量。对于CBOW方法，输入是中心词上下文几个单词的one-hot向量的和，输出对应的label则是中心词对应的one-hot向量。

2.2.2 Glove

Word2vec主要考虑上下文信息，而对于Glove[6]算法，除了上下文信息外还会考虑语料库的全局信息。

Glove首先引入了共现矩阵（Co-occurrence Probabilities Matrix），X_{ij}表示$word_i$上下文中$word_j$的出现次数（$word_i$的上下文可j以表示为以该单词为中心的在句子的一个滑动窗口），$X_i = \sum_k X_{ik}$ 表示$word_i$所有相关上下文的共现矩阵值的和，$P_{ij} = P(j|i) = \frac{x_{ij}}{x_i}$，表示 $word_j$ 出现在 $word_i$ 上下文的共现概率。P_{ij}构建的矩阵即为共现矩阵。

取单词$word_i$，和其相关的两个单词$word_j$，$word_k$，根据共现概率定义三个单词的比率$Ratio_{i,j,k} = \frac{P_{ij}}{P_{ik}}$。根据单词间的相关性，Ratio值有如下的性质。

Ratio	单词 i,j 相关	单词 i,k 相关
单词 i,j 相关	趋近1	很大
单词 i,k 相关	很小	趋近1

Glove模型就是利用这个Ratio值进行定义。对于三个单词embadding的词向量w_i，w_j，w_k，通过函数F变换后应与它们的Ratio值接近。

$$F(w_i, w_j, w_k) = Ratio_{i,j,k} = \frac{P_{ij}}{P_{ik}} \tag{1}$$

因为Ratio是考察对于单词i，单词j和k的相似性关系，所以可以考虑使用线性空间的相似性，即两个向量的差$(w_j - w_k)$。Ratio本质时一个标量，而w_i和$w_j - w_k$是两个向量，所以可以尝试将两个向量点积，和Ratio比较。

$$F(w_i, w_j, w_k) \to F(w_i, w_j - w_k) \to F((w_j - w_k)^T w_i) \to F(w_j^T w_i - w_k^T w_i) \tag{2}$$

取$F(x) = exp(x)$，代入可以得到

$$F(w_j^T w_i - w_k^T w_i) = exp(w_j^T w_i - w_k^T w_i) = \frac{w_j^T w_i}{w_k^T w_i} = \frac{P_{ij}}{P_{ik}} \tag{3}$$

只要让等式两边的分子和分母分别相等，等式就成立，即

$$exp(w_j^T w_i) = P_{ij} = \frac{X_{ij}}{X_i} \tag{4}$$

两边分别取对数

$$w_j^T w_i = log(X_{ij}) - log(X_i) \tag{5}$$

因为$logX_{ij} - logX_i! = logX_{ji} - logX_j$，所以为了解决对称性问题，引入两个偏置项$b_i$，$b_j$，可以得到代价函数。一般情况下两个词共现次数越多的Loss Function中的权重越大，所以引入权重$F(x_{ij})$。以下为最终的Loss Function。

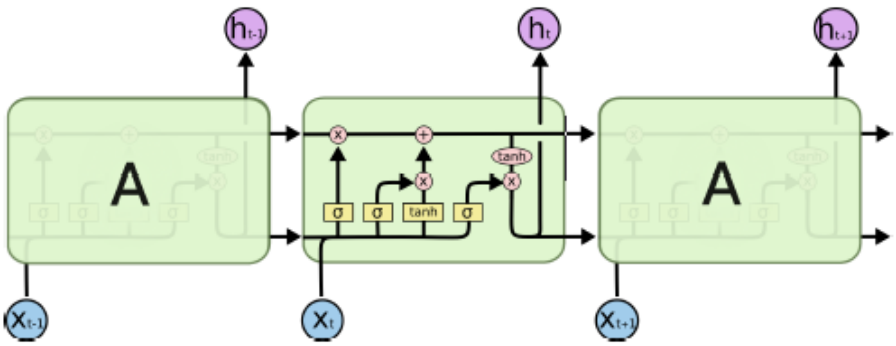
$$J = F(x_{ij}) \sum_{ik} (w_j^T w_i + b_i + b_j - log(X_{ij}))^2 \tag{6}$$

2.3 循环神经网络（RNN）

循环神经网络（RNN）主要用于处理序列数据，比如文本和视频。它基于不同神经元共享参数的思想，对句子的信息进行编码，保留和当前语境相关的信息并丢弃无关信息。权值共享使得它能够使用较少的模型参数训练大量的数据。

根据反向传播的原理，传统的RNN下梯度会在模型中多次反向传播，这很容易造成梯度爆炸或梯度消失，进而导致模型很难训练。同时传统的RNN很难访问距离当前时刻较远的信息。长短期记忆网络（LSTM）的引入一定程度上解决了这些问题。

LSTM[1]的最大贡献是引入了门机制。如下图，LSTM包含两个流h和c，h_i 是短期记忆流，c_i 是长期记忆流。LSTM从左往右有三个门，用于更新状态h和c。



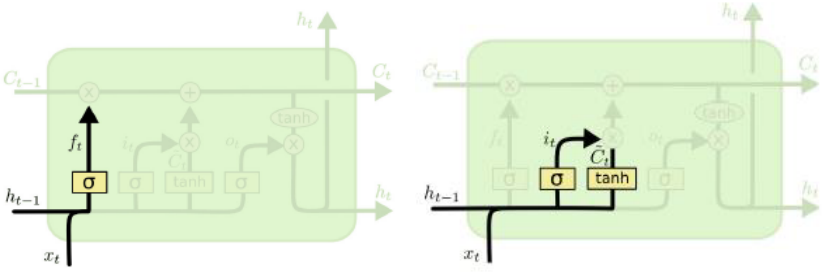
遗忘门（Forget Gate，下图左）让当前时刻t的输入 x_t 通过一个全连接层后接激活函数映射到0或1，与保存的历史状态 C_{i-1} 点乘，目的是根据输入遗忘部分历史状态。

$$f_t = \alpha(W_f * [h_{t-1}, x_t] + b_f) \tag{7}$$

输入门（Input Gate, 下图右）让输入 x_t 信息经过两个不同的全连接层后接激活函数得到两个流 i_t 和 $\widetilde{C}_t$ 。这两个流点乘用于表示当前状态有多少需要被保存到历史信息中。

$$i_t = \alpha(W_i * [h_{t-1}, x_t] + b_i) \tag{8}$$

$$\widetilde{C}_t = tanh(W_c * [h_{t-1}, x_t] + b_c) \tag{9}$$



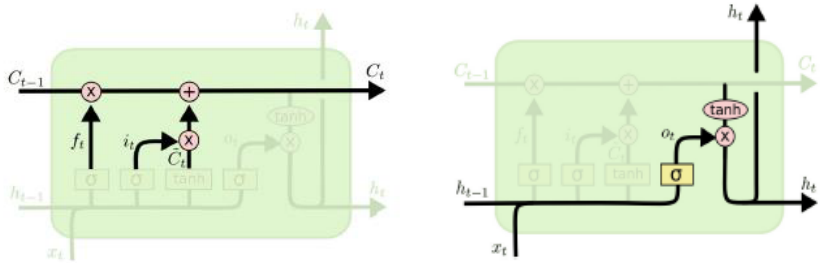
长期记忆状态 c_t (下图左)根据遗忘门和输入门更新，将遗忘后的历史状态加上输入状态，得到新的历史状态

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C}_t \tag{10}$$

输出门（Output Gate, 下图右）将t时刻的输入信息 x_t 接全连接层与激活函数后与接了激活函数的历史信息点乘，得到对于t时刻的短期记忆状态 h_t。

$$O_t = \alpha(W_o * [h_{t-1}, x_t] + b_o) \tag{11}$$

$$h_t = o_t * tanh(C_t) \tag{12}$$

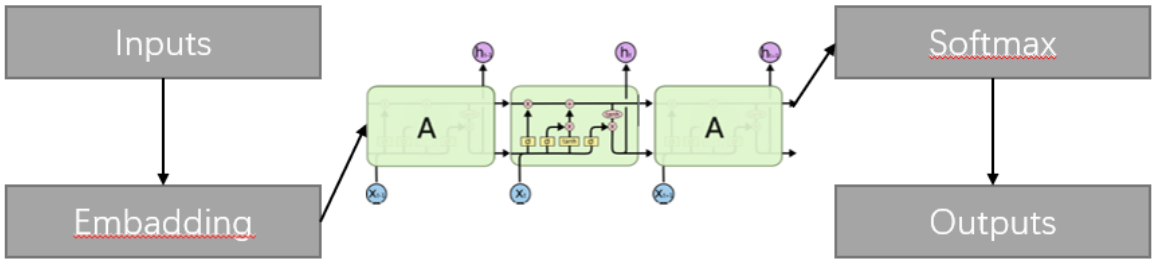


3 实验结果与分析

3.1 数据集与相关说明

本文使用的是kaggle上Sentiment Analysis on Movie Reviews比赛的数据集。训练数据包括156060个句子或短语，这些语料由8544个句子生成，短语就是这些句子截断产生的一部分。测试数据则是由3311个句子生成的156060个句子或短语构成。语料的语言都是英文，标签有5中，分别为
0（negative），1（somewhat negative），2（neutral），3（somewhat positive），
4（positive）。

实验过程中，本文按照以下结构构造了一个神经网络。通过预处理转化成Index的输入数据经过Embadding降维为词向量，随后接入一个两层的双向LSTM中。对于每个句子，根据它的具体长度选择LSTM中对应的句尾的Output向量作为输出，后接一个带Softmax的全连接层得到分类结果。



在比较分析的时候，会构造一个验证集配合kaggle上的feedback比较各种方法的性能。这里选择数据集中80%的数据作为训练集，20%的数据作为验证集。最终提交的时候则会选择使用全量数据集训练的模型产生的分类结果。本章的表格中，Train_Acc为训练100个epoch后最终的训练集准确率，Valid_Acc为训练100个epoch后最终的验证集准确率，Max_Val_Acc为训练过程中验证集准确率的最大值，Test_Acc为测试集上传到kaggle上反馈的得分（准确率）。

在训练参数上，默认batch_size=128，训练100个epoch，使用两层的双向LSTM。初始学习率为1e-4，学习率会根据训练情况动态调整，如果连续5个epoch训练集的loss达不到最低则学习率loss会乘上0.1。使用Adam优化器，weight_decay=1e-4。硬件方面，使用一张GTX1080Ti进行训练。模型的显存消耗不大，大约在1080MB左右。

3.2 类别不均衡问题

本文使用的数据集其实类别并不均衡。简单统计一下训练集中每个类别的占比，可以看到标签为2的数据特别多。同时直接把type=2作为结果交到kaggle上就可以获得51.79的分数。出现这种情况的主要原因是数据集里有很多单词和短语，而这些单词和短语大多都是中性的。

type	0	1	2	3	4
prob	4.51%	17.10%	51.44%	21.08%	5.86%

这里使用不加优化的LSTM模型进行训练。我尝试将类别均衡化，引入pytorch的`WeightedRandomSampler`，将训练集数据中每个类别出现的比率平均化。不过由于整个数据集的标签比例都是类似的，使用类别均衡后的模型在验证集上表现一般。因此本文最终没有采用这种策略。

weight sample	Val_Acc
N	58.12
Y	52.68

3.3 词向量的维度

Glove[6]提供了维度为50，100，200，300的多种预训练词向量，我分别尝试这几种词向量，并比较模型的性能。这里使用LSTM作为网络基本单元，droprate=0.5，使用fine-tine的训练方法。可以看出，虽然50维的词向量维度较小，但是它的性能却是最好的。因此之后的实验我都选择实验50维的预训练词向量。

Dim	Train_Acc	Val_Acc	Max_Val_Acc	Test_Acc
50	78.51	57.10	59.34	63.02
100	76.50	57.05	58.71	62.46
200	77.46	56.04	58.85	62.50
300	77.41	57.04	58.95	62.82

3.4 词向量的训练方式

我们尝试将Glove预训练的词向量加载到embadding层中进行实验，并和不加载预训练词向量的LSTM模型进行性能上的对比。因为Glove预训练词向量的词典和本文使用的数据集不同，对于Glove中不存在的单词，使用随机向量进行嵌入。

Glove官方库中包含不同长度的词向量表示。另外，加载词向量后可以选择是否将embadding的梯度冻结住，即让embadding层在预训练的基础上进行fine-tune还是将embadding层固定住。结果发现，虽然在验证集上表现相近，在测试集上freeze住参数的模型性能明显低于fine-tune的模型，这主要是因为把参数freeze住后会影响模型的表达能力。这点从模型在训练集上的表现也可以看出，模型在训练集上的准确率最终也只有69.31%。因此本文选择使用fine-tune作为加载预训练词向量后对Embadding层的处理方案。

Use_Glove	Freeze	Train_Acc	Val_Acc	Max_Val_Acc	Test_Acc
N	-	75.66	57.29	58.68	62.52
Y	N	78.51	57.10	59.34	63.02
Y	Y	69.31	59.14	59.58	62.18

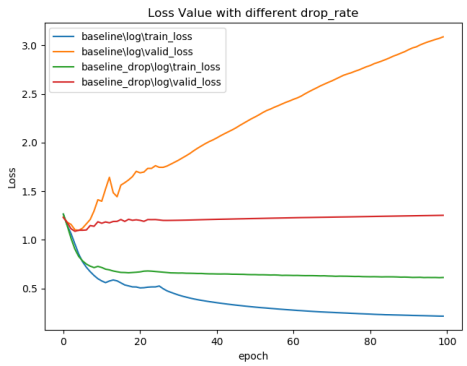
3.5 Dropout的影响

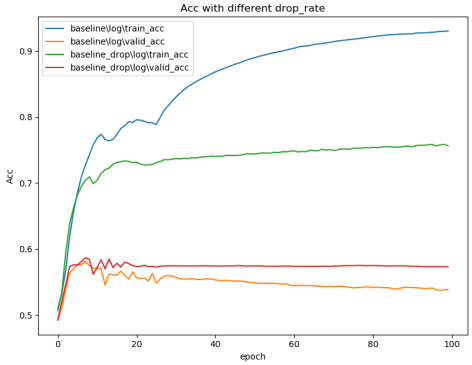
这里我尝试在使用枚举一些drop_rate做对比试验。Pytorch里面LSTM的dropout是在除了最后一层外每层的输出后加上的。

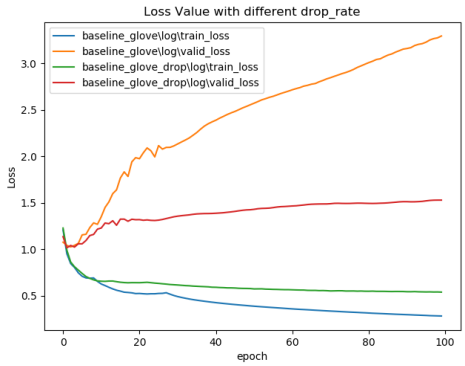
根据下表的实验结果可以发现，虽然幅度有限，但是使用dropout对预测结果还是有一定提升的，比如使用Glove的预训练词向量的情况下测试准确率能提升0.5个百分点（当然，这些结果也和随机种子有一定的关系）。

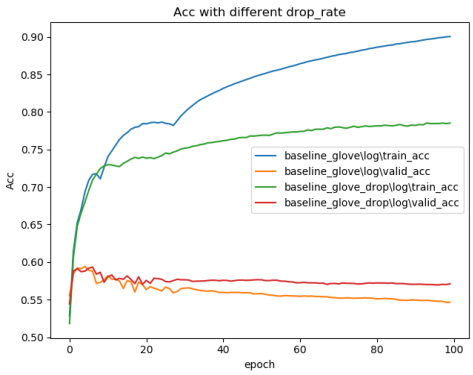
Use_Glove	Drop_Rate	Train_Acc	Val_Acc	Max_Val_Acc	Test_Acc
N	0	93.04	53.85	58.12	62.46
N	0.2	84.92	56.18	57.97	62.55
N	0.5	75.66	57.29	58.68	62.52
Y	0	90.04	54.66	59.22	62.55
Y	0.2	81.42	56.53	59.16	62.84
Y	0.5	78.51	57.10	59.34	63.02

另外，取droprate=0和0.5的情况，对比训练的loss和准确率曲线，并结和上表中间两列的数据，可以发现取dropout=0.5时，随着训练的进行过拟合现象会得到明显的缓解。使用dropout的方法在训练集上的表现劣于没使用dropout的方法，在验证集上则明显优于没使用dropout的方法。因此，我们取droprate=0.5进行全部实验。
下图中第一行的两张图为不适用Glove预训练词向量时不同drop_rate的训练图，第二行为使用gGlove预训练词向量的情况。









同时通过对比也可以看到，导入预训练词向量可以给网络一个比较好的初始解，也可以一定程度上减少过拟合。

3.6 过程与结果分析

我们使用LSTM网络结构，加载50维Glove预训练词向量，不freeze嵌入层，drop_rate=0.5，不使用weight_sample的模型进行测试。3.5节中下面的两张图的baseline_glove_drop模型就是模型训练时的参数图。

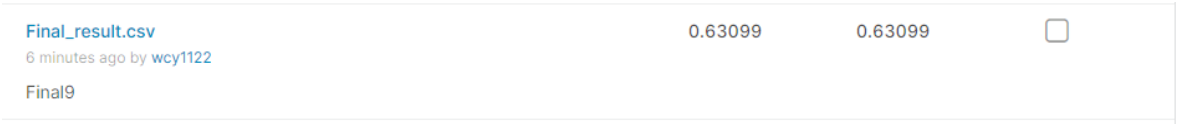
可以看到训练集的准确率随着epoch的增加逐渐增大，loss也逐渐降低。验证集的准确率呈先增大后降低的趋势，两三个epoch后就开始低于测试集。对于基于LSTM的文本分类任务，这其实是比较正常的现象，主要原因还是数据太少，毕竟本文使用的数据集的数据量只有15万左右，真实句子数只有8千多。使用LSTM尚且如此，如果上BERT显然会出现更严重的过拟合。

以下为验证集的混淆矩阵，可以看到占据样本比率超过一半的2这一类的预测结果较好，召回率可以达到79.08%。样本数较多的1，3两类的预测记过也还可以，大体上召回率能超过40%。0，4这良类最惨，一类召回率不足30%，另一类更是5.68%的召回率。验证集中大多数标签为0的数据的预测结果都是1，大多数标签为4的数据预测结果都是3。不过可以看到即使出错，模型预测结果的大体趋势还是正确的，即使出错标签和预测结果的差也基本不会超过1。如果放宽正确标准到误差<=1，该模型的测试准确率可以达到94.29%。

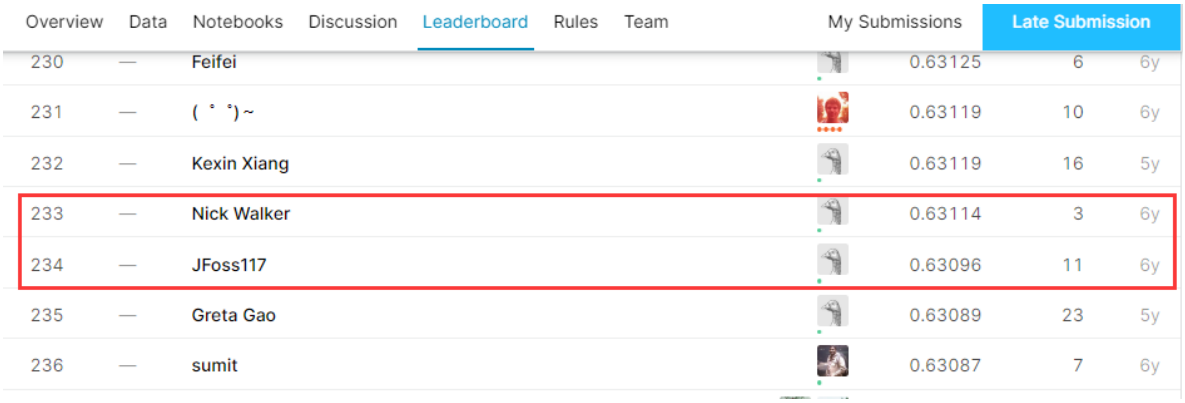
predict \ label	0	1	2	3	4
0	**5.68**	1.35	0.10	0.00	0.16
1	66.53	**48.36**	11.76	8.48	5.47
2	22.80	44.09	**79.08**	41.09	14.38
3	4.78	5.86	8.68	**43.71**	51.33
4	0.21	0.34	0.37	6.65	**28.66**

3.7 提交结果

取3.4节中的模型，取全量数据训练10个epoch，将训练的模型得到的测试结果提交到kaggle比赛中，最终可以取得63.099的得分。



因为是一个已经结束多年的比赛，所以不能上榜，参考Leaderboard，排名大致为234名。



4 结论

本文使用LSTM作为基本的网络单元，将Glove预训练词向量加载入embadding层，最终在kaggle比赛中可以获得63.099的得分。本文测试了不同的droprate，发现了dropout对RNN类网络的作用。另外，本文还对预训练词向量进行探究，测试了不同维度预训练词向量的性能，也对比了freeze和fine-tune两种训练形式在性能上的差异，最后发现使用50维的Glove预训练词向量进行fine-tune的性能最好，预训练词向量的引入对模型的准确性得到了提升。

参考文献

[1] Sak, H., Senior, A.W., & Beaufays, F. (2014). Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition. *ArXiv, abs/1402.1128*.

[2] Kim, Yoon. (2014). Convolutional Neural Networks for Sentence Classification, *EMNLP*.

[3] Liu, P., Qiu, X., & Huang, X. (2016). Recurrent Neural Network for Text Classification with Multi-Task Learning. *ArXiv, abs/1605.05101*.

[4] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., & Polosukhin, I. (2017). Attention is All you Need. *ArXiv, abs/1706.03762*.

[5] Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. *ArXiv, abs/1310.4546*.

[6] Pennington Jeffrey, Socher Richard, Manning Christopher (2014). Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing. *EMNLP*.

[7] Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *NAACL-HLT*.