# 期末展示实验报告

——基于三种神经网络进行情感分析以及效果比较

10175501113 雷镇豪

## **选题**

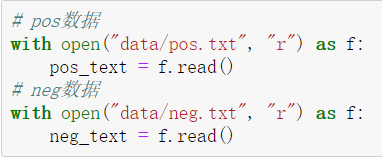
之前浏览到一篇2014年的论文，Yoon Kim的《Convolutional Neural Networks for Sentence Classification》，其中提到了两层channel的卷积神经网络模型进行句子分类，这种Text-CNN类型的卷积神经网络思路简单，但是分类效果的确非常好，另外这篇论文也阐述了几种不同的神经网络模型在句子分类上的效果，因此我打算实现三种不同的神经网络模型，来进行一个较为简单的情感分析，比较它们的性能差异，并加深对不同神经网络模型的理解。

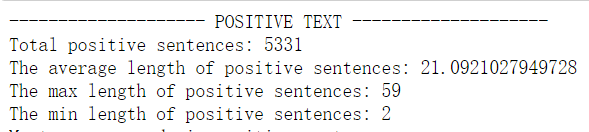
此次试验选取的模型为CNN、两层channel的CNN模型(如论文所述)、LSTM模型(较为经典)。以下将从数据处理开始介绍这三个模型，并对比它们的性能差异。

## **数据整理**

本次实验采用的是cornell大学公开的一个情感分析数据集，包含了5331条positive的文本和5331条negative的文本。下面所有的代码都将基于这个数据来构建分类模型。

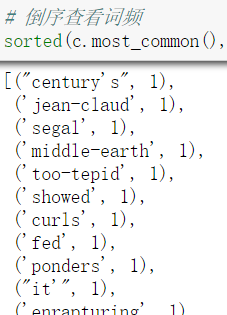
①、首先加载数据，并对数据集进行描述性统计：





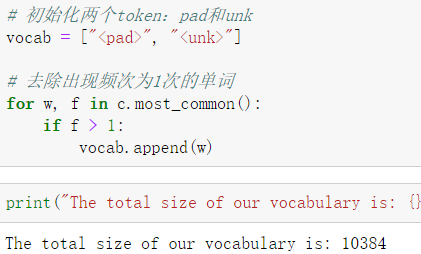
如同积极文本的统计所示，可以看到正负评论文本各有5331条，平均长度在20左右。

②、构造词典



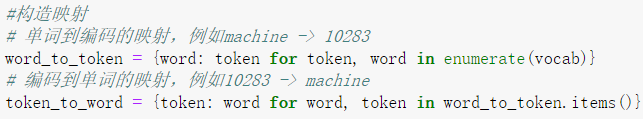
接下来要基于这些语料来构建词典，构造词典的步骤一般就是对文本进行分词再进行去重。当对文本的单词进行统计后，会发现有很多出现频次仅为1次的单词，这类单词会增加词典容量，并且还会给文本处理带来一定的噪声，去除这些词以后一方面将会极大减小词典容量，加速模型训练，另一方面也会减缓一定的噪声。

因此在构造词典过程中仅保留在语料中出现频次大于1的单词。其中<pad>和<unk>是两个初始化的token，<pad>用来做句子填补，<unk>用来替代语料中未出现过的单词。最后我们得到一个包含10384个单词的词典。



③、构造映射

有了词典以后就需要构造word到token的映射和token到word的映射：



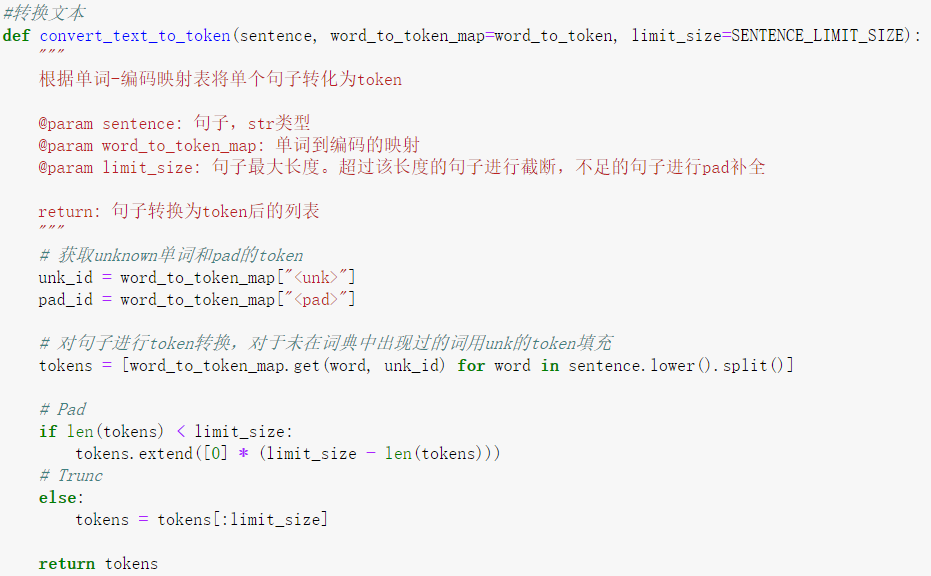
④、转换文本

有了映射表的基础就可以对原始文本进行转换，即将文本转换为机器可识别的编码。除此之外，为了保证句子有相同的长度，需要对句子长度进行处理。在描述性统计阶段可以发现，语料中句子的平均长度为20个单词，因此在这里就设置20作为句子的标准长度：

对于超过20个单词的句子进行截断；

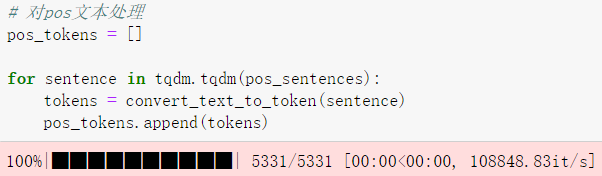
对于不足20个单词的句子进行PAD补全。

接着构造一个函数，它可以接收一个完整的str类型的句子，并根据映射表将其转化为tokens。



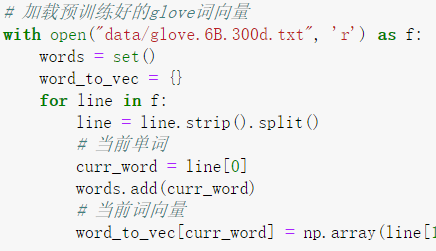
这个函数中，我们首先要获取unk的编码和pad的编码，以备后面句子转换使用。接下来对句子进行映射，如果出现没有见过的单词，则用unk的token替代。最后再对句子的长度进行标准化。

接下来我们分别对pos文本和neg文本进行转换：

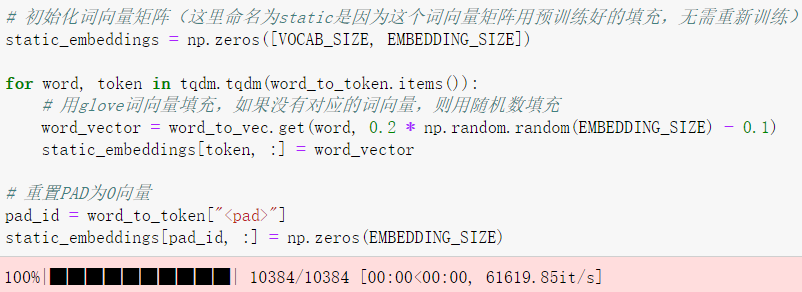


⑤加载预训练词向量进行迁移学习

此次试验使用Glove中预训练好300维的词向量作为模型的word embeddings。



可以发现，通过语料构造的词典中，有9801个单词有pre-trained的词向量，583个单词没有对应的pre-trained词向量，此次实验将基于这个预训练模型构造一个vocab\_size \* embedding\_size大小的矩阵。



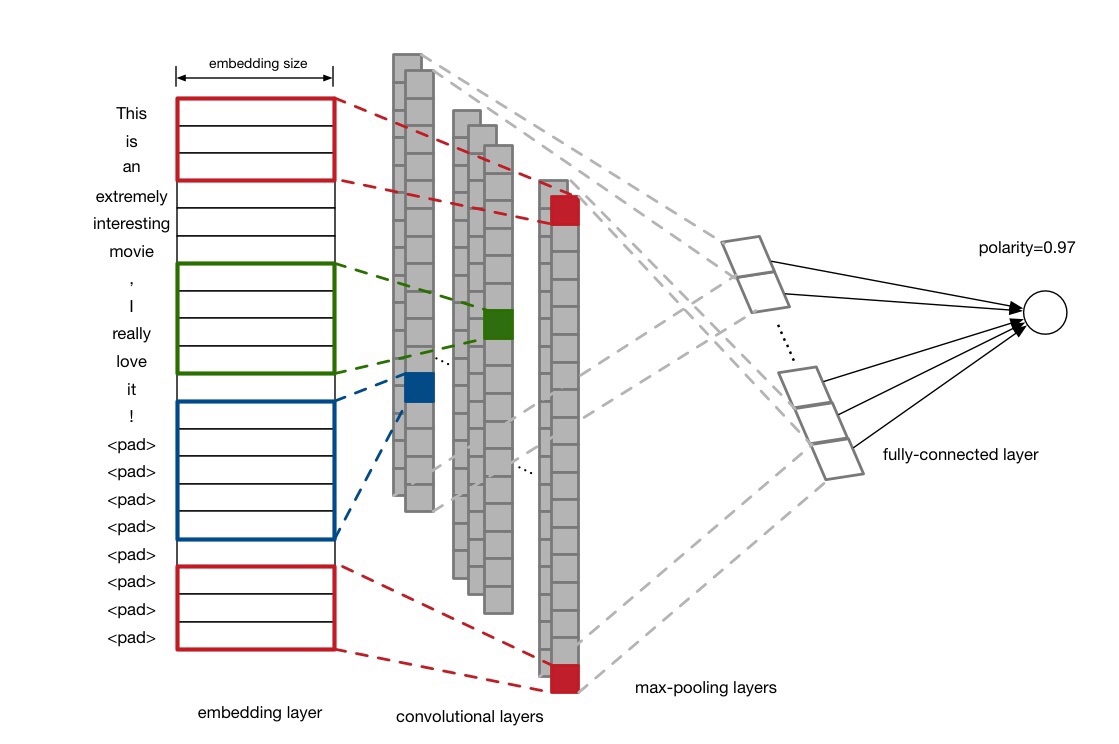
上述代码执行后将得到一个static\_embeddings矩阵，这个矩阵每一行是词典中的一个单词所对应的词向量（300维）。

综上，数据处理步骤主要完成了:将原始文本转换为了tokens、构建了自己的word embeddings。接下来进行模型搭建。

## **模型实现**

①CNN模型

众所周知CNN在图像处理中取得了很不错的效果，这是因为它的卷积和池化操作可以捕捉到图像的局部特征。通过此次实验，完全可以印证CNN用在文本处理上，也可以捕捉到文本中的局部信息，并且效果也比较好。模型结构如图:



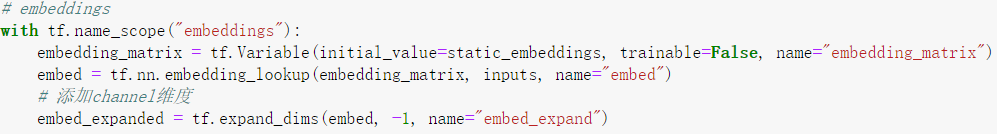
如结构图所示，假设句子长度为20，不足的用<pad>补全。输入句子序列后，经过embedding，获得每个单词的词向量（假设词向量维度为300），就得到一个sentence\_len \* embedding\_size的矩阵，如上图中embedding layer所示。在这里可以将它看做是width=embedding\_size，height=sentence\_len，channel=1的一张图片，那么就可以用filter去做卷积操作。

接下来采用了3种filter，三种filter size分别是3，4，5，每种filter有100个。如结构图所示，红色的filter为height=3，width=embedding\_size，channel=1；绿色height=4，蓝色height=5。

经过卷积操作后，就得到了如上图中所示convolutional layers的输出，多个列向量；再经过max-pooling操作来提取每个列向量中的最重要的信息。最终连一层fully-connected layer得到输出结果。

**Embeddings代码**：

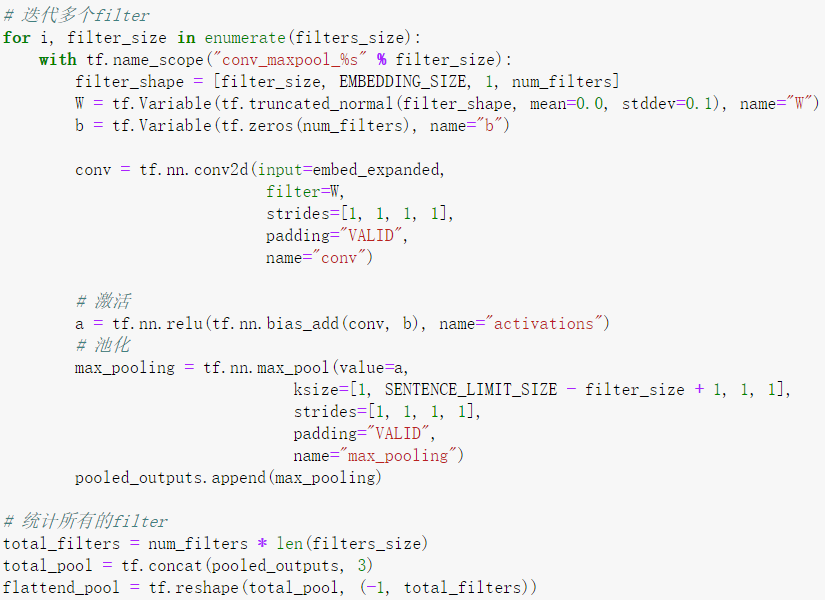
因为卷积操作要求有channels这个维度，因此在构造完embed以后，实际上shape=(batch\_size, vocab\_size, embedding\_size)，但是Tensorflow中卷积要求的维度为(batch\_size, heights, widths, channels)，因此通过expand\_dims增加channels维度。



**卷积-池化层代码:**

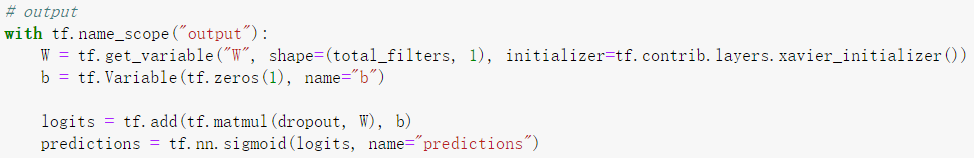
由于采用多种filter（filter size=2, 3, 4, 5, 6），因此对于卷积池化操作要分filter处理。首先定义了pooled\_outputs用来存储每种filter的卷积池化操作输出结果。

对于每个filter，先经过卷积操作得到conv，再经过relu函数激活后进行池化，得到max\_pooling。由于每个filter有100个，因此最终经过flatten后可以得到100\*5=500维向量，用于连接全连接层。



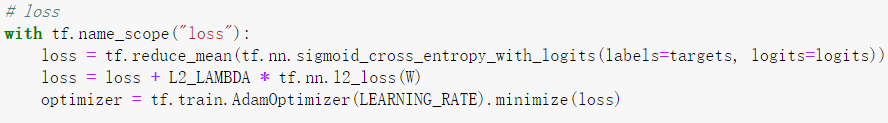
**输出:**

发现添加dropout可以提升模型性能，最后在output层进行全连接，经过sigmoid激活输出最终的结果。



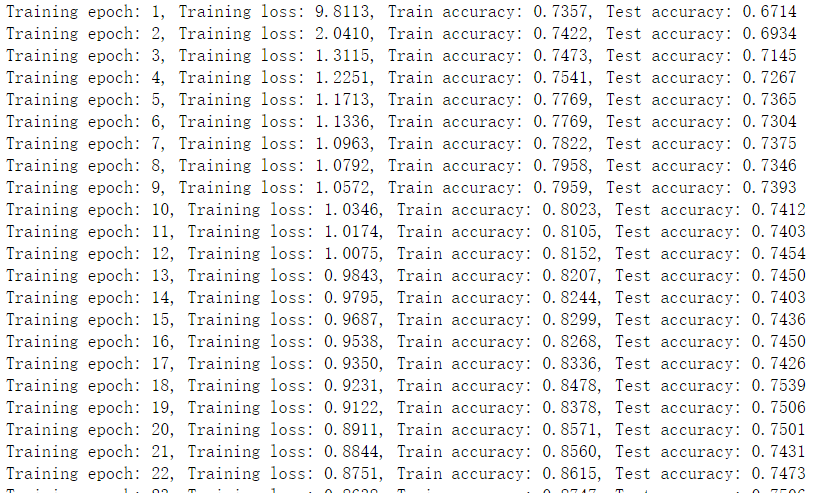
**Loss**：

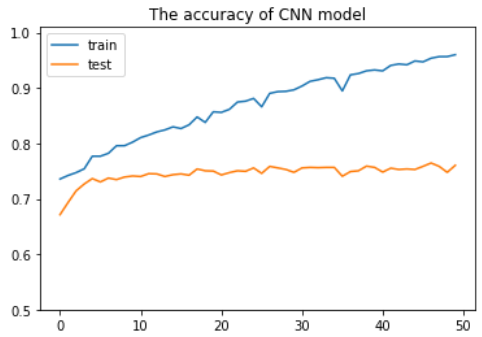
本次实验的CNN模型损失函数上添加了全连接层权重W的L2正则，尝试以后发现加了L2会带来模型在test上的性能提升。



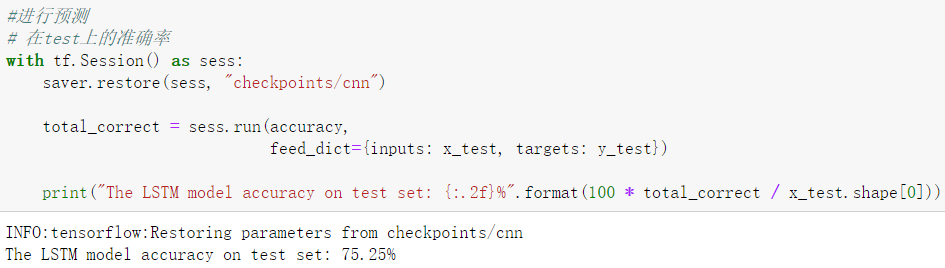
**训练模型**:

由于加入了L2正则和dropout，可以发现模型在train上并没有那么过拟合，并且迭代50个epoch以后train上准确率还有上升空间。在test上，模型基本在第10轮以后就差不多稳定了。





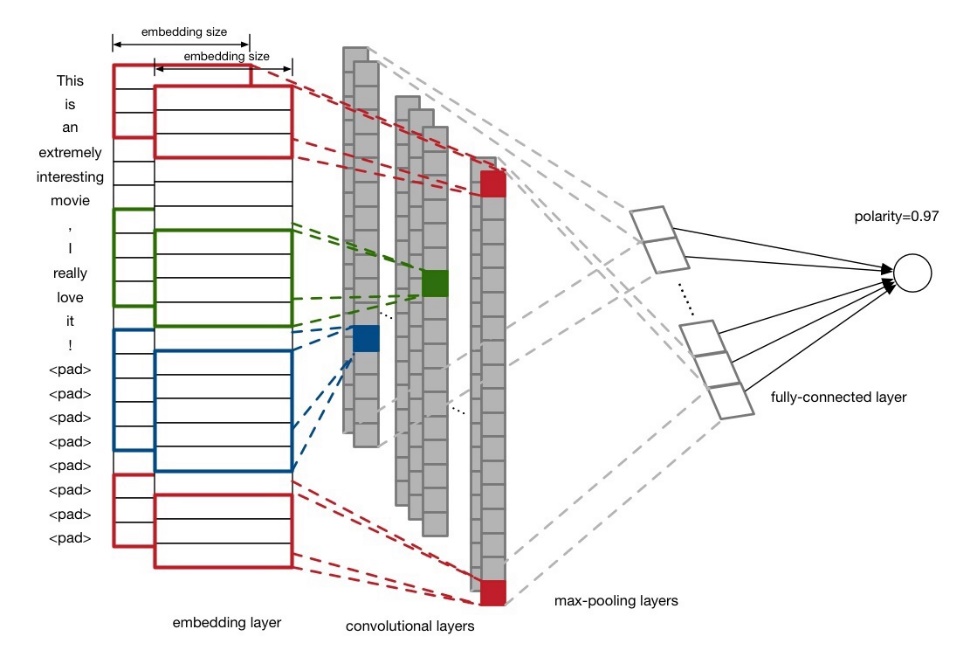
**测试模型:**



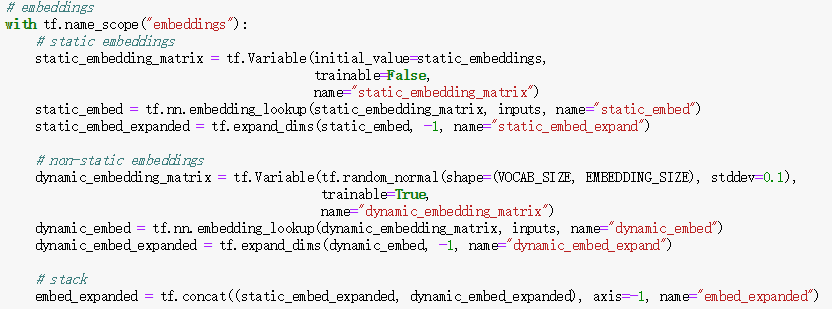
在test上准确率为75.25%。可以看到TextCNN如此简单的模型却能达到相对较高的准确率,还是比较令人欣喜。

②CNN优化模型

CNN模型的优化版本即在上述CNN的基础上为embedding多添加一个channel。之前因为使用pre-trained word embeddings构造了shape=(sentence\_len, embedding\_size, 1)的图；这一层可以采用随机初始化，也可以用pre-trained word embedding初始化并进行fine-tune，它的目的在于捕捉task-specific的信息。具体模型结构如图：

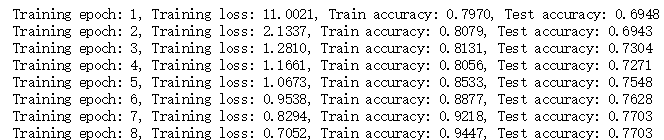


multi-channel模型在CNN模型上添加一个可以trainable的embedding。这里除了static的embeddings以外，还要构造一个non-static的embedding，选取采用随机初始化来填充non-static embedding。得到这两个embedding以后，只需要按照channel堆叠起来就可以得到shape=(sentence\_len, embedding\_size, 2)的tensor。

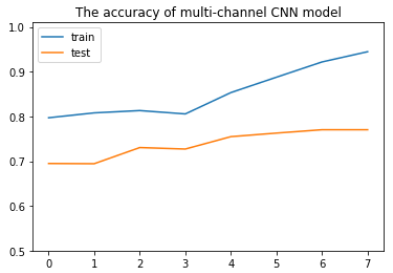


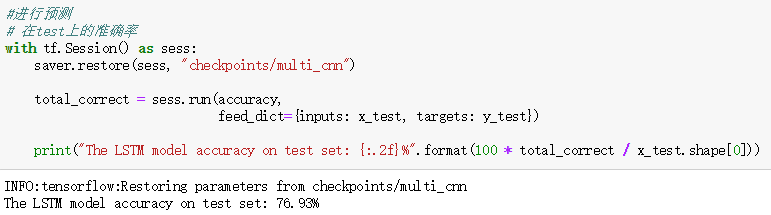
**训练以及测试**

这里只训练了8轮，因为发现往后训练的话会发生严重过拟合。由于训练语料其实不是很多，而且又加了一层embedding，这就使模型非常容易过拟合。因此只设置了8轮epoch，这样模型在test上也能取得不错的效果。



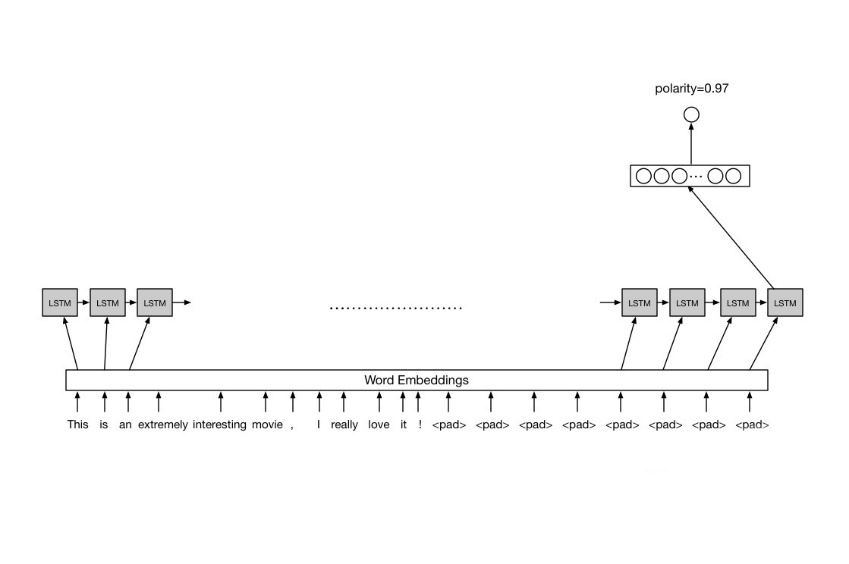
如图所示，multi-channels在test上的准确率为76.93%，算比较不错。





③LSTM模型

LSTM擅长捕捉长序列关系。LSTM由于有gate的存在，所以能够很好地学习和把握序列中的前后依赖关系，因此也就更加适合用来处理长序列的NLP问题。模型结构如图：



思路为先将句子进行word embedding后，传入LSTM序列进行训练，将LSTM的最后一个hidden state拿出来，加入全连接层得到最终输出结果。

**模型代码**

在embeddings中， LSTM不需要对词向量求和，而是直接对词向量本身进行学习。其中无论是求和还是求平均，这种聚合性操作都会损失一定的信息。

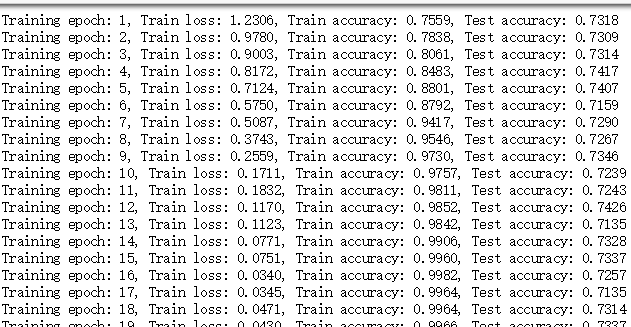
在model中，首先构造了LSTM单元，并且为了防止过拟合，添加了dropout；执行dynamic\_rnn以后，就会得到lstm最后的state，这是一个tuple结构，包含了cell state和hidden state（经过output gate的结果），这里只取hidden state输出，即lstm\_state.h，对这个向量进行连接，最终得到输出结果。

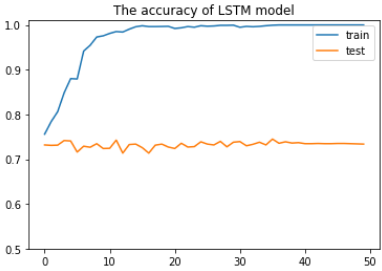


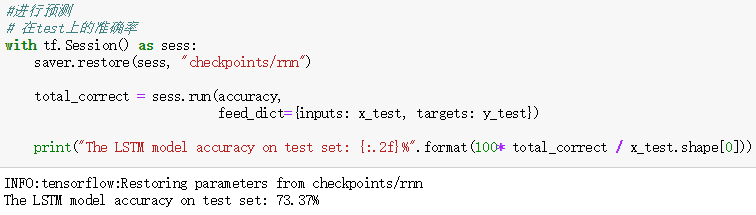
**训练及测试**

模型构建完毕后，设置超参数进行训练，这里采用了单层512个结点的LSTM模型。

用训练好的模型对test数据进行预测，得到准确率如图，为73.37%。







## **项目结果总结**

此次试验完成了利用CNN、优化后的CNN、LSTM三种模型进行情感分析的任务。其中，CNN的准确率达到75.25%。multi-channels CNN在test准确率高达76.93%，并且训练次数也较少。LSTM模型的准确度最差，为73.37%

总体来说，本次实验的模型都相对比较简单，但总体来说这几个模型都取得了不错的准确率，其很大程度上来自于pre-trained word embedding，可见word embedding在NLP模型中的重要性，而multi-channels CNN的准确率提升则在于加入了task-specific的信息。

通过这次实验，加深了我对这三种基本神经网络模型的认识，也更能熟练地运用代码去实现自己的想法。