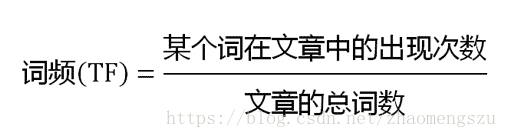
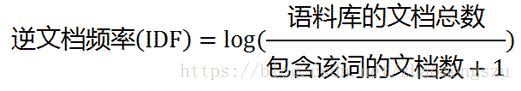
**文本分类**

特征处理

一、TF-IDF

某个词对文章的重要性越高，它的TF-IDF值就越大。







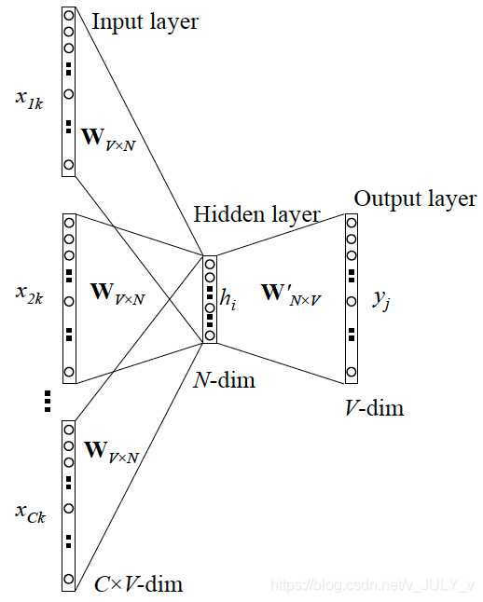
使用sklearn库的TfidfVectorizer，得到文本的关键词。

二、Word2vec（一种word embedding算法）

word embedding指的是将词转化成一种分布式表示，又称词向量。

1、原理

通过训练神经网络语言模型（CBOW和Skip-gram模型）得到的第一个隐含层的参数权重，就是我们需要的词向量。

（1）CBOW

一个三层神经网络。该模型的特点是输入已知上下文，输出对当前单词的预测。

模型计算步骤：

1、输入层：上下文单词的onehot。（假设单词向量空间dim为V，上下文单词个数为C）

2、所有onehot分别乘以共享的输入权重矩阵W（V\*N矩阵，N为自己设定的数，初始化权重 矩阵W）

3.所得的向量相加求平均作为隐层向量, size为1\*N.

4.乘以输出权重矩阵W’{N\*V}

5.得到向量 {1\*V} 激活函数处理得到V-dim概率分布，概率最大的index所指示的单词为预测出的中间词（target word）

6.与true label的onehot做比较，误差越小越好。loss function（一般为交叉熵代价函数）

训练完成后第一个全连接层的参数就是我们要的word embedding矩阵。

（2）Skip-gram

与CBOW类似，只是变成了已知当前词语，预测上下文

2、具体实现

（1）使用re库去除文档的标点符号并进行文档的分词

（2）调用gensim库的Word2vec模型进行训练，得到128维的词向量

3.注意

在这里对所有样本进行不停词操作，可以保证词语字典尽可能多，减少文本分类预测时出现oov情况。

**TextCNN**

一、数据的处理：

1.根据文本中TF-IDF值高的词有选择地构建出停词表

2.对文本进行分词、转换小写、去掉停用词、去掉标点符号的一系列操作

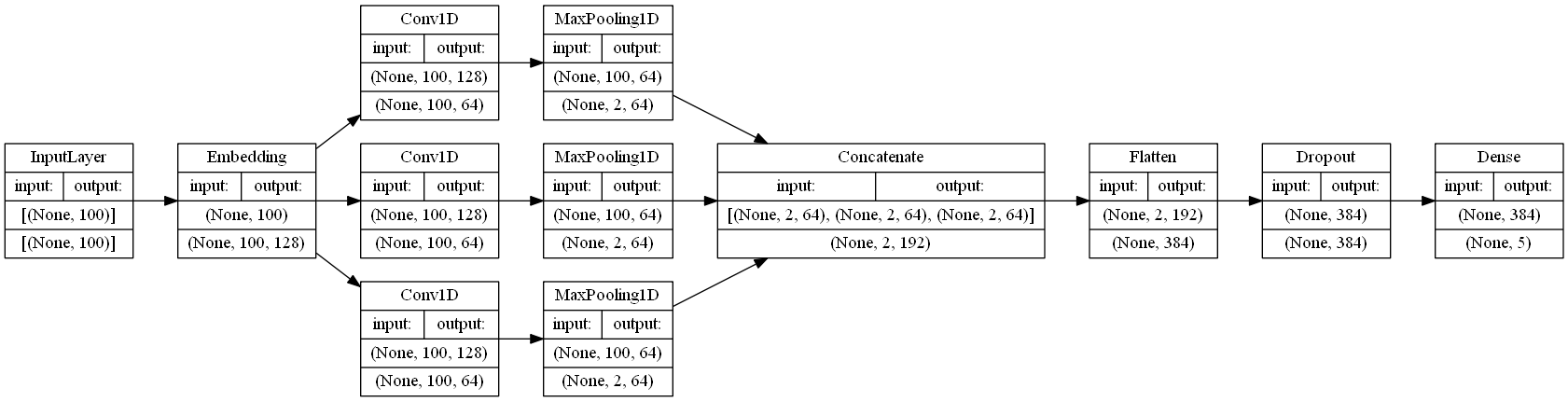
3.对文本所有出现的词从1开始进行编号

4.对应每个词的编号，将文本转换成数字序列并进行相同长度（100）的截断

二、模型的建立：

与图像当中CNN的网络相比，textCNN 最大的不同便是在输入数据的不同

使用keras库构建出textCNN模型，该模型的架构如下：



TextCNN模型结构

1、嵌入层：

对于每一个句子都进行处理，使得句子的长度都为n，对于太长的句子则进行截断。

句子中的每个词，都将其映射到词向量空间，假设词向量为 k 维，则 n 个词映射后，每个句子都可表示成 n\*k 的二维矩阵

2、卷积层

自然语言是一维数据, 虽然经过word-embedding 生成了二维向量，但是对词向量做从左到右滑动来进行卷积没有意义，所以卷积核的宽度和word embedding的宽度一致。

在Keras库中调用Conv1D进行自然语言处理。在这里我创建三个不同kernal size（3，4，5）的卷积层，这样不同的kernel可以获取不同长度范围内词的关系特征，获得的是句子方向上的差异信息，即类似于n-gram。然后在每一层又有多个数量（64个）的卷积核，不同的卷积核有不同的参数，都有自己的关注点，这样多个卷积核就能学习到多个不同的信息。

3、池化层

网络采用了**2-Max**池化，即为从每个滑动窗口产生的特征向量中筛选出两个最大的特征，然后将这些特征通过concatenate操作拼接起来构成向量表示。

4、Dropout层

设置参数为0.6，**防止过拟合**

5、全连接层

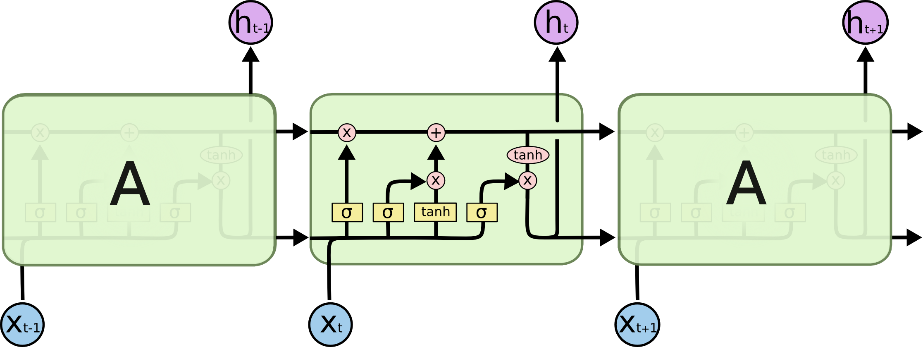
设置激活函数为softmax，结果分5类，即units=5

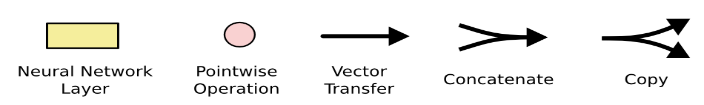
损失函数使用categorical\_crossentropy交叉熵损失函数

优化器使用“adam”,一种随机梯度下降方法

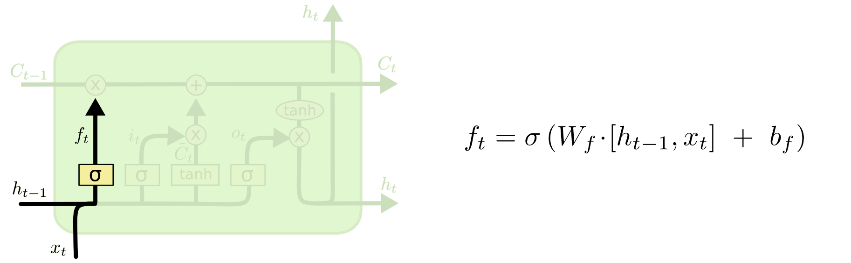
**Lstm**

一、理论基础

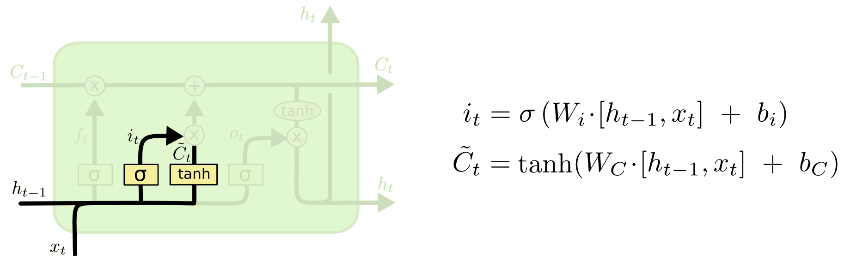


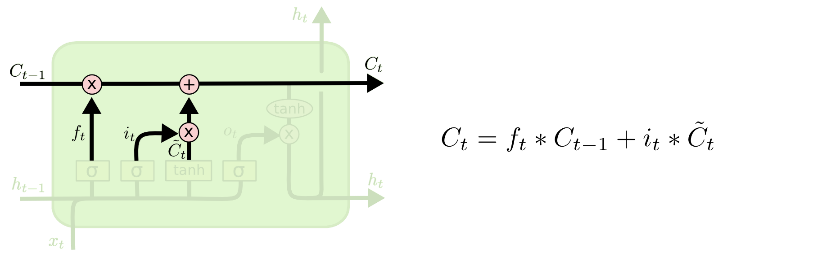


1、遗忘门

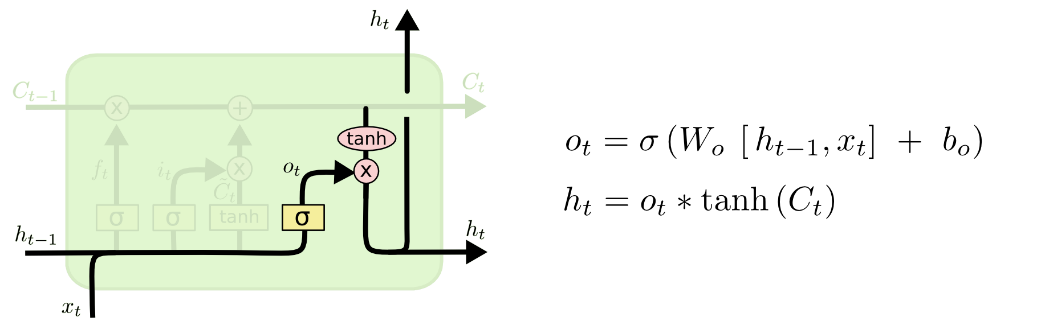


2.更新门





3.输出门

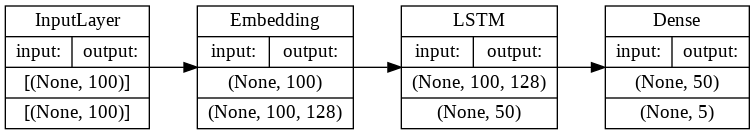


二、数据处理

与textCNN类似

三、模型建立

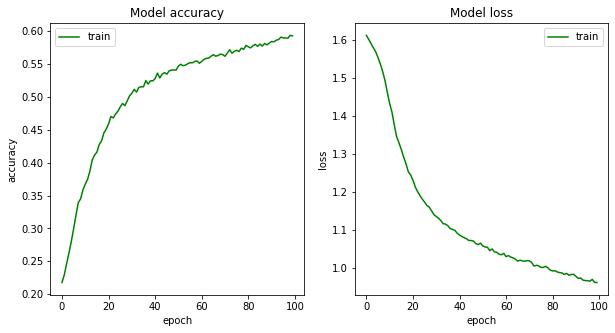
在时间上提取文本的特征



lstm层

unit为50，其中每个unit返回的是序列的最后一个输出，所有最终输出的矩阵为（50，1）

**值得注意**的是，模型训练的epoch在100次左右的分数最高，此时损失值为一点零几左右。猜测可能是由于测试集size大于训练集size，模型迭代多次容易出现对测试集的过拟合，导致预测能力不强。



模型损失值随epoch变化图

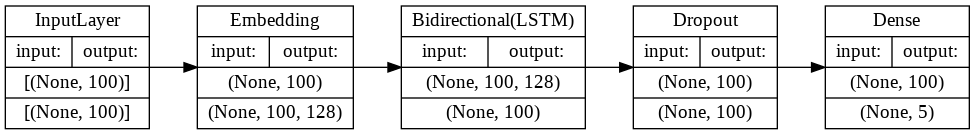
四、模型优化尝试

1. BiLSTM

由于利用LSTM对句子进行建模还存在一个问题：无法编码从后到前的信息。在更细粒度的分类时，如对于强程度的褒义、弱程度的褒义、中性、弱程度的贬义、强程度的贬义的五分类任务需要注意情感词、程度词、否定词之间的交互。可以选择通过BiLSTM更好地捕捉双向的语义依赖。

在代码上使用keras的Bidirectional层进行实现。

该模型的预测分数是所有模型中最高的。



BILSTM模型结构图

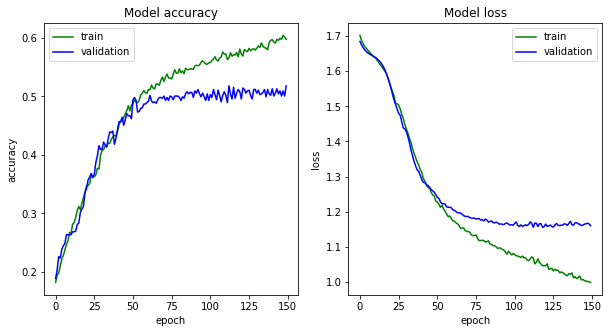
2.数据增强（EDA）

由于经过文本预处理后的长度大于100的句子接近两千，所以分别将训练集的文本句子以‘pre’、‘post’两个方式进行填充和截断，来增加样本数据，可一定程度上**防止过拟合**。

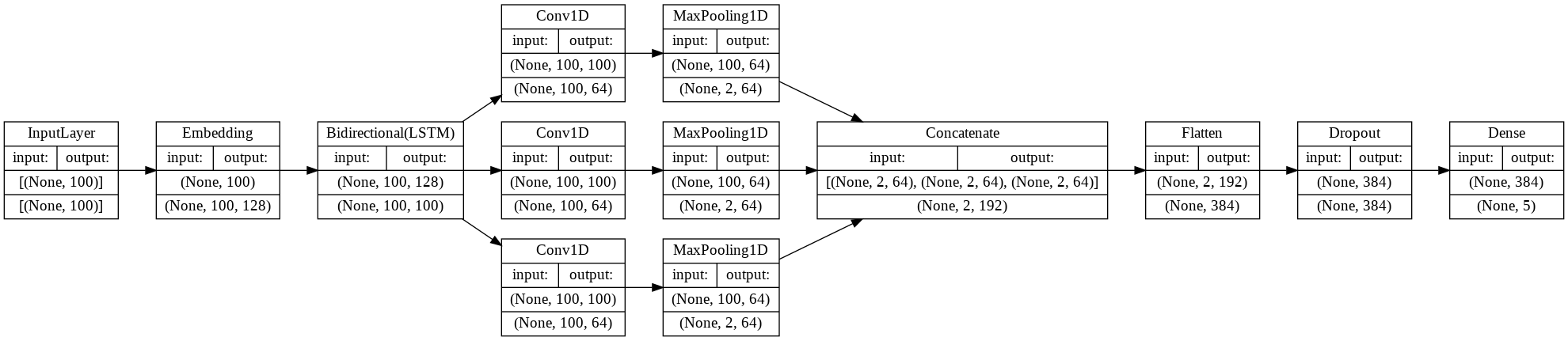
然而将这个新的训练集和原来的训练集放在同一个模型训练出来的结果相差不大。

**3.正则化**

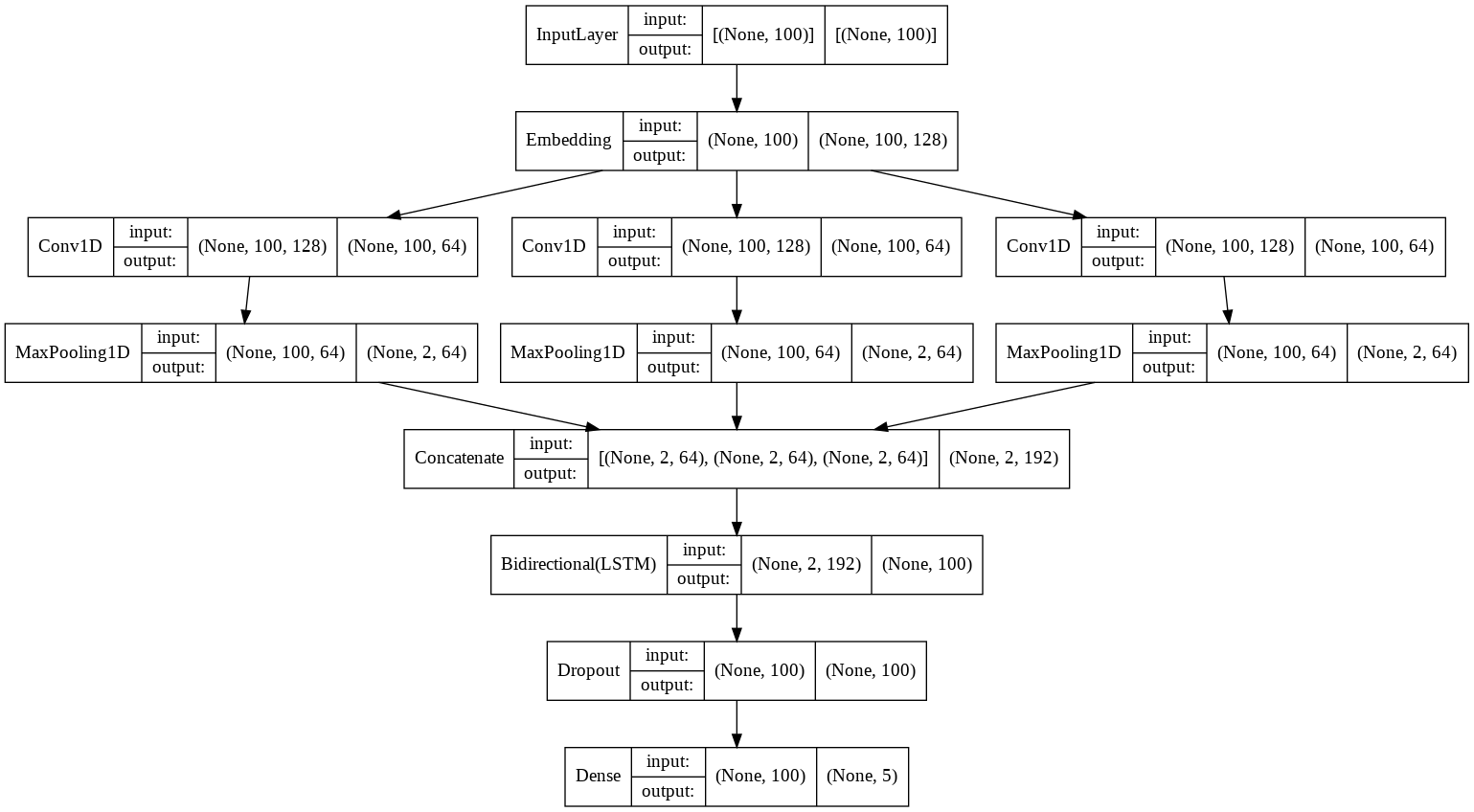
调用keras的正则化器，防止**过拟合**。使用后使模型精确率出现波动



4.混合模型



bilstm+cnn结构图

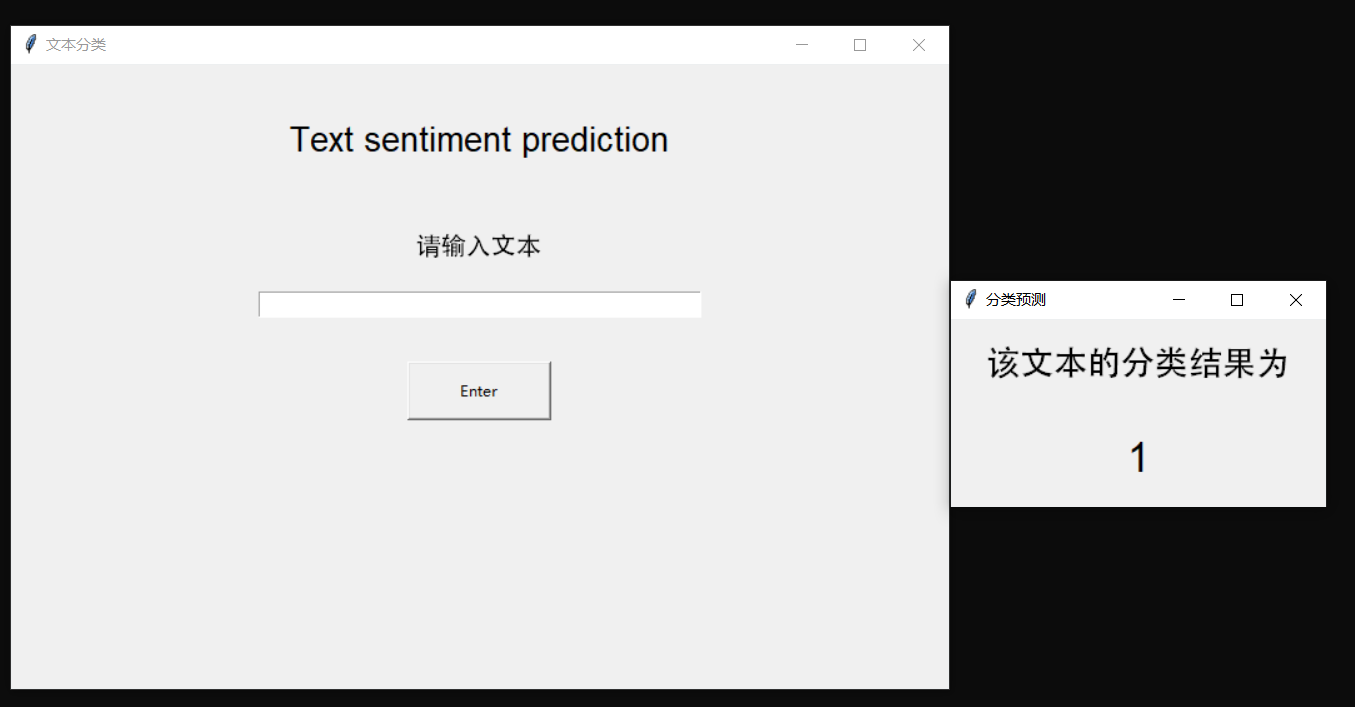


bilstm+cnn结构图

更改两个模型结合的顺序组成混合模型

由于时间原因，并没有对参数的调整做太多的尝试，出来的效果并不明显

GUI

调用python的tkinter库，粗略地编写了一个图形界面

**总结**

本次所做的几个模型的预测效果都不是特别好，个人认为原因有三：

一、 特征处理。这次考核只是简单地将分词、去词后的文档训练出词向量放进模型里。并没有进行诸如词根还原、subword等复杂的处理。

二、 测试数据多于训练数据。过多的训练次数容易造成模型对训练样本的过拟合，泛 化能力低。过少的训练次数又使模型预测不稳定，分数变化大。

三、 模型参数的调节。这次考核的每一步处理基本都调用了库，其中各个函数的参数量众多，对模型的调参提出了比较高的要求。