# 多模型中文NLP文本分类结果对比

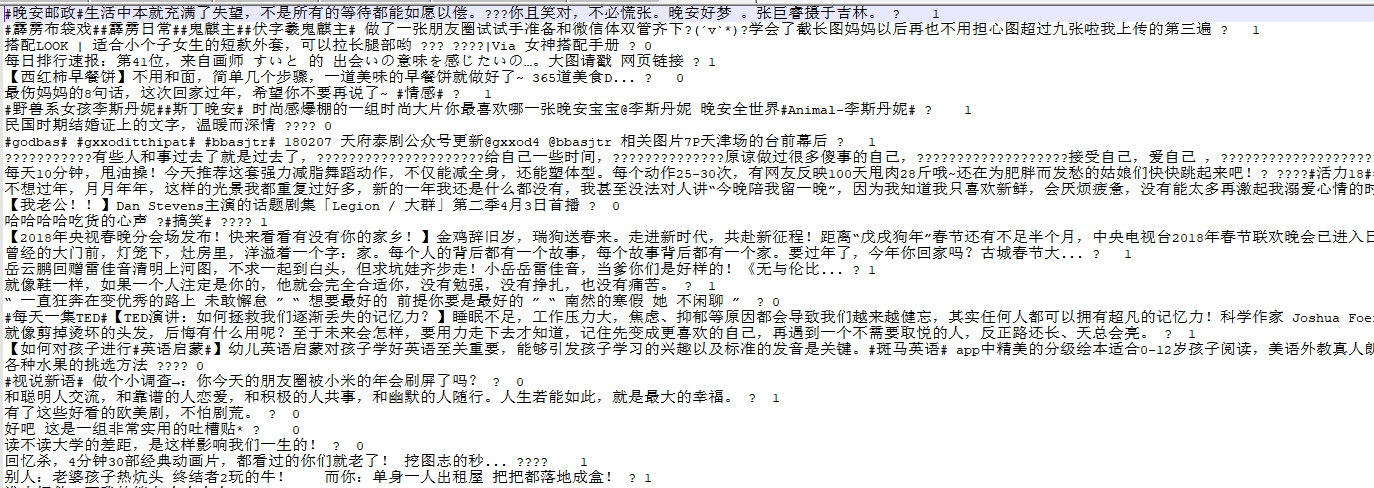
1. 文本清洗

由于摘取的文本是微博中的评论，含有大量的表情符号，中英文标点等并无含义的符号，因此在文本清洗阶段需要将这些符号删除。

* 1. 将中文的标点符号统一为英文标点符号
  2. 删除一些不需要的英文标点符号
  3. 删除由字符组成的表情符号
  4. 删除其他不知名符号

最终，文本中只剩下中文词汇，英文标点和数据标签 （可自行扩充）

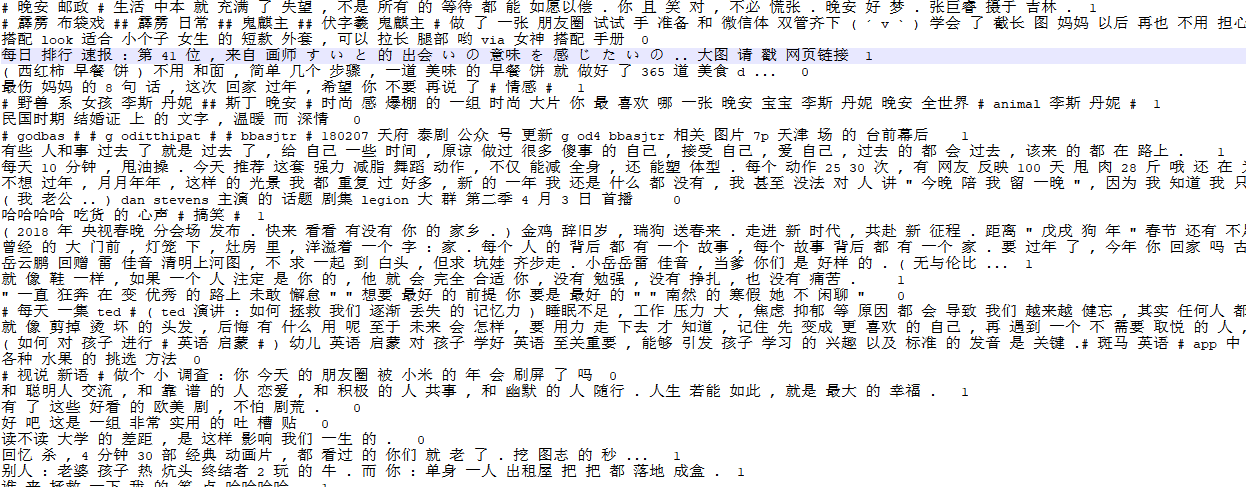
文本清洗前：



1. 分词

一句话是由许多的词汇组合而成的，在模型输入结构中，输入的是词汇信息，因此需要将文本数据中的一句话拆分成多个词汇的组合。Python中有现有的分词包:jieba分词。在这一步中，使用jieba分词中的隐马尔科夫模型将句子拆分成词汇。最终使用空白符将词汇分开。 （可自行扩充）

文本清洗和分词后：



1. 词向量化

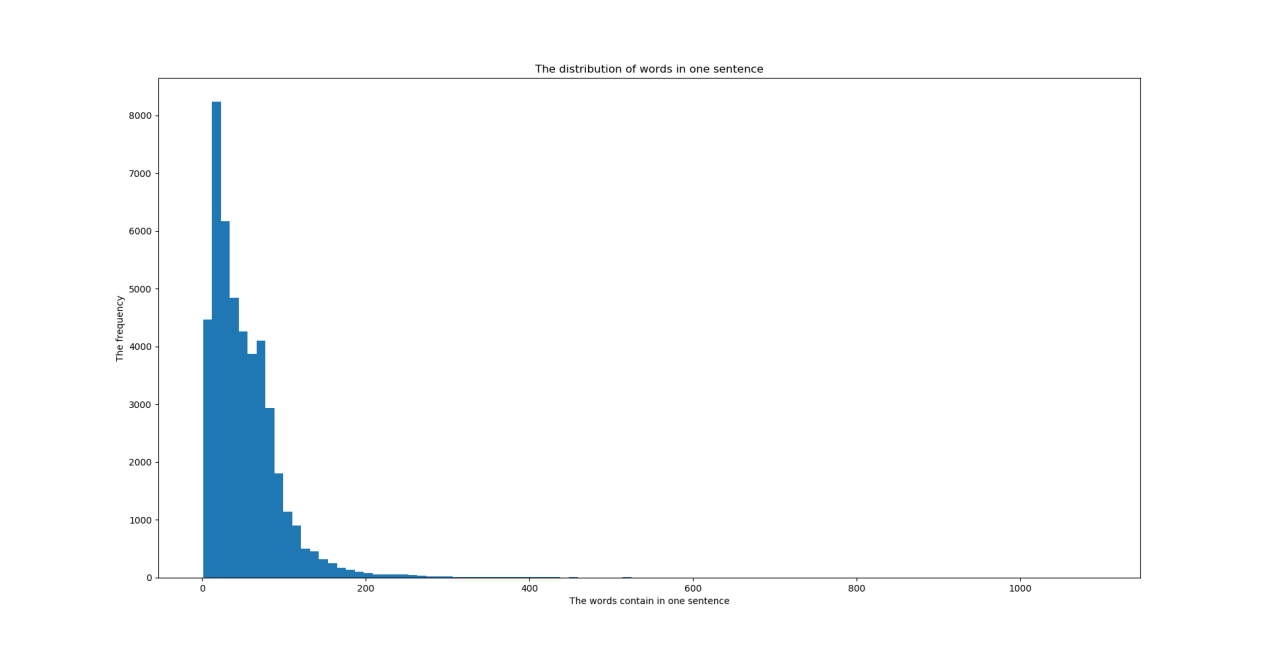
深度学习模型不能处理文本数据，需要用数字向量表示对应的词汇。最简单的就是one-hot编码，将所有单词变为类似[0,0,0....1,0,0...]的向量。但是这样的编码是相互独立的，缺少词汇之间的相互关系，因此不可取。词的向量化通常有两种不同的模型：CBOW模型和Skip-Gram模型。CBOW的目标是根据上下文来预测当前词语的概率；Skip-gram刚好相反：根据当前词语来预测上下文的概率。

使用Python中的gensim包中的Word2Vec函数，使用窗口大小为7的窗口，使用CBOW模型将词表示成拥有128维度的向量。 （可自行扩充）

词向量化矩阵：  


1. 模型的构建

因为每一个训练样本中，不同文本之间的文本长度并不一致，这就要求对文本长度有个统一要求。可以先观察样本集中的文本长度分布：

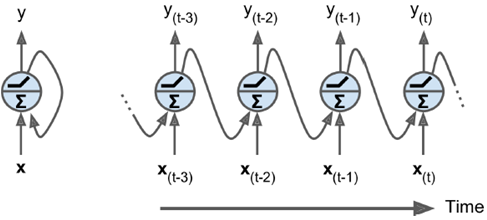


从图中可以看出，95%的句子长度都在200个词汇以内，因此，最大的句子长度选取为200个词汇，对超过最大长度的句子进行截取，不足的进行填充(padding)。

将所有的词汇组成一个词典库，词典库上建立索引，每一个索引表示不同的词，索引从1开始，因为0表示文本长度不够的时候补齐的符号。最后在embeding层中输入一个句子的词汇索引得到一句话的矩阵表示。这样，就可以将多个句子组合成一个批次数据放入模型中进行训练。在本次训练中，同时将64条句子放入一个训练批次中进行训练。

* 1. RNN：

RNN的结构如图所示：

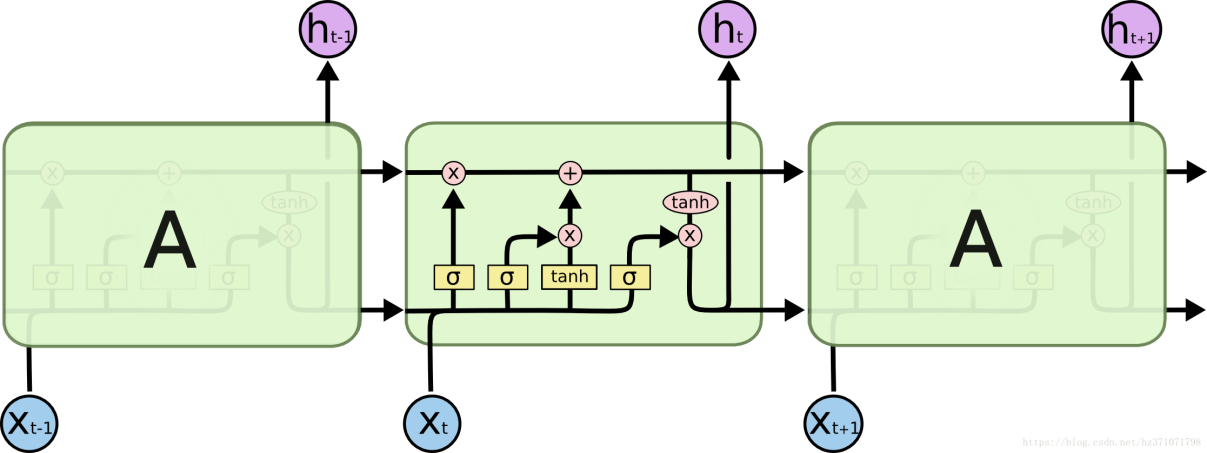


有一个输入x和一个输出y，中间的循环箭头表示前一个时间点的隐藏层输出。隐藏层中的矩阵信息包含了从0时刻到第t-1时刻的所有输入信息，是一个信息的缩影，在t时刻输入x和前t-1时刻的信息做数学运算，因此来达到记忆0~t-1信息的效果。第0时刻的隐藏层所有的矩阵元素都是0，不包含任何信息。因此这套网络系统称为循环神经网络。循环网络的展开图如右图。

* 1. LSTM

RNN中有一个很大的缺陷就是如果句子太长，就会出现遗忘的现象。也就是说隐藏层的矩阵无法完整表示0~t-1时刻的信息，可能会将0~某时刻的信息忘记，这样对模型来说很不利，因此，LSTM就是为了解决传统RNN的遗忘缺陷的。

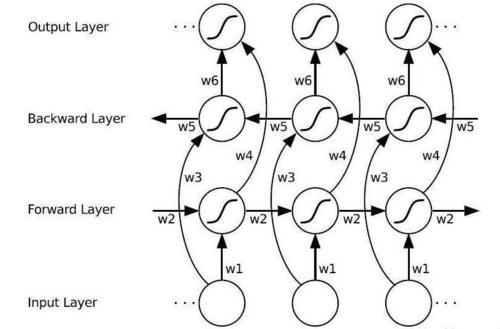
LSTM结构：



LSTM的隐藏层包含了两个输出，一个是ht（隐藏状态）,一个是ct（细胞状态）。并且通过sigmoid函数和tanh函数控制隐藏状态和细胞状态的输出，达到在一定程度上改善传统RNN中出现的遗忘现象。（具体的函数公式可以自行百度）

* 1. Bi-LSTM

单向的LSTM也有一个缺点，在某一时刻t，单向LSTM只可以看见0~t-1时刻的信息，在t之后的信息并不能很好的利用。也就是说，单向LSTM无法完整的应用上下文信息，这样对模型的预测是很不利的，因为在t时刻，某些很重要的上下文信息是在t时刻之后的。因此出现了双向的LSTM模型。结构如图所示：



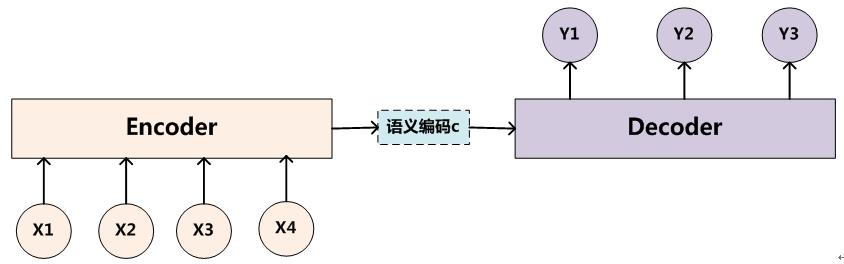
如图所示，输出成融合了来自t时刻之前的信息和t时刻之后的信息，这样，在t时刻，模型就拥有了整体上下文的综合信息，有利于模型通过上下文信息进行预测。

* 1. Attention机制

注意力机制在NLP中分为两种，一种是seq2seq模型中的注意力机制，一种是在2017年谷歌提出的自注意力机制。两种都叫注意力机制，但是原理完全不同。

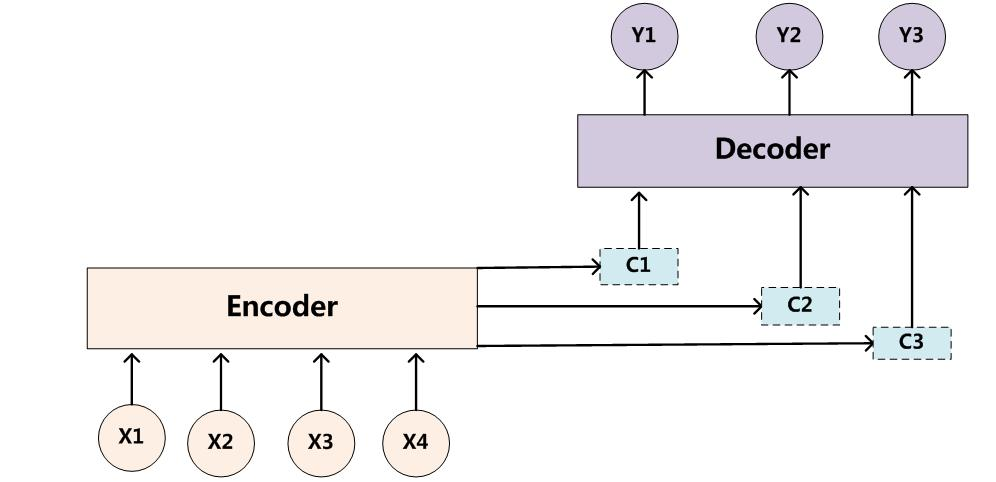
* + 1. Seq2Seq

Seq2Seq模型一般用于文本生成或者文本翻译的模型中，意思是，一个序列转换为另一个序列。这个模型由两个部分组成，一个是encoder编码模块和decoder解码模块。编码模块将原始序列进行编码，一般编码成矩阵的形式；解码模块将编码后的矩阵进行解码成另一种序列。像将中文翻译为英文。结构如图



句子x中的任意单词xi对生成某个单词yi的影响力都是一样的，并没有任何区别。

但是加了注意力机制之后，句子x的任意单词xi对某个yi的影响力会根据对应权重的改变而改变。结构如图：

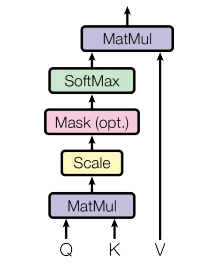


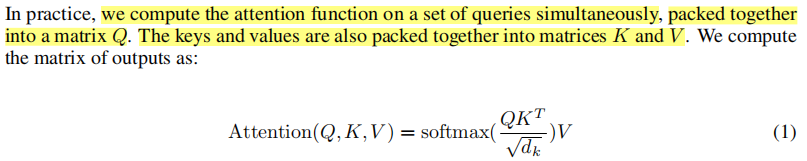
从原先的都是相同的中间语义编码c变成了根据当前时刻t不同而生成不同的语义编码ci。通过这样的注意力机制，使得解码模块更好的理解语义编码向量，做出更准确的预测。

* + 1. 自注意力机制

自注意力机制来自于谷歌的一篇论文:Attention is all you need。这篇论文提出了自使用自注意力机制就可以完美代替RNN以及它的变种模型。RNN以及它的变种都有一个共同的缺点，那就是因为它们对数据的处理都是顺序性的，只能从0时刻按顺序处理到t时刻，这样数据就没法并行处理，大大降低了处理数据的能力与速度，使得RNN的训练都很慢，没有像CNN那样迅速。因此，谷歌的自注意力机制取消了在时间步长上的限制，可以同时并行处理数据，使得数据的训练大大加快。

自注意力机制的结构如图：





Q,K,V都是输入的序列。

我们采用的就是谷歌的自注意力机制模型，而不是seq2seq中的注意力机制。

1. 我们采用对比实验，使用tensorflow2.0构建相应的模型。
   1. 直接构造LSTM模型
   2. 构造双向LSTM模型
   3. 构造双向LSTM模型加上自注意力机制
   4. 构造论文Attention is all you need中的Transformer模型

对比这4个模型在相同文本数据上的训练结果。