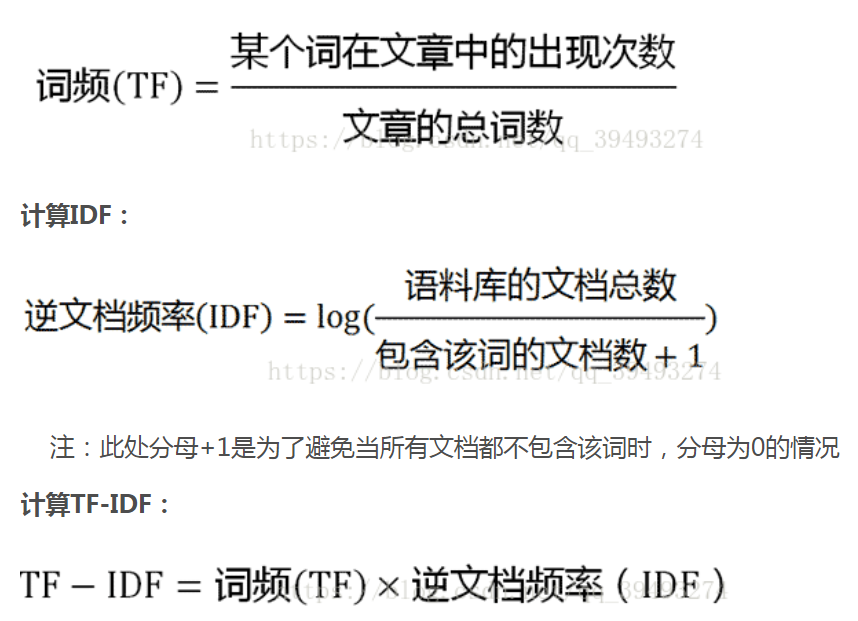
# 一、中文关键词抽取的三种方法

**TF-IDF、textrank、word2vec**

## TF-IDF

TFIDF全称叫做term frequency–inverse document frequency，翻译过来可以叫做文本频率与逆文档频率, TFIDF就是为了表征一个token（可以是一个字或一个词）的重要程度。

主要用于提取文本中的关键词。



TF-IDF是对文本所有候选关键词进行加权处理，根据权值对关键词进行排序。

优点:简单快速，结果比较符合实际情况。

缺点:单纯以"词频"衡量一个词的重要性，不够全面，有时重要的词可能出现次数并不多。而且，这种算法无法体现词的位置信息，出现位置靠前的词与出现位置靠后的词，都被视为重要性相同，这是不正确的。

算法的关键词抽取步骤：

（1） 对于给定的文本D进行分词、词性标注和去除停用词等数据预处理操作。采用结巴分词，保留'n','nz','v','vd','vn','l','a','d'这几个词性的词语，最终得到n个候选关键词，即D=[t1,t2,…,tn];

（2） 计算词语ti 在文本D中的词频TF=Count(ti)/Count(D\_word)；

（3） 计算词语ti 在整个语料的IDF=log (Dn /(Dt +1))，Dt 为语料库中词语ti 出现的文档个数；

（4） 计算得到词语ti 的TF-IDF=TF\*IDF，并重复（2）—（4）得到所有候选关键词的TF-IDF数值；

（5） 对候选关键词计算结果进行倒序排列，得到排名前TopN个词汇作为文本关键词。

实现代码：

*# 计算TF值*def tf(number\_counts):  
 for m in number\_counts:  
 fenmu = 0  
 for j in number\_counts[m]:  
 fenmu += number\_counts[m][j]#文章中出现词的总数  
 for k in number\_counts[m]:  
 number\_counts[m][k] = number\_counts[m][k] / fenmu  
 print(number\_counts[m][k])  
 return number\_counts  
  
*# 计算IDF的值*def idf(number\_counts):  
 idf = {"行政": 0, "规划": 0, "政策": 0}  
 for l in idf:  
 count = 0  
 D = 0  
 for m in number\_counts:  
 D += 1  
 if l in number\_counts[m].keys():  
 count += 1  
 idf[l] = math.log(D /count)return idf  
  
  
*# 计算TF-IDF的值*def TFIDF(tf, idf):  
 for m in tf:  
 for k in idf:  
 if k in tf[m].keys():  
 tf[m][k] = tf[m][k] \* idf[k]  
 return tf

原始数据：

{'1': {'行政': 2, '规划': 4, '政策': 1}, '2': {'规划': 2, '政策': 1}, '3': {'行政': 1, '规划': 1}}

结果数据：

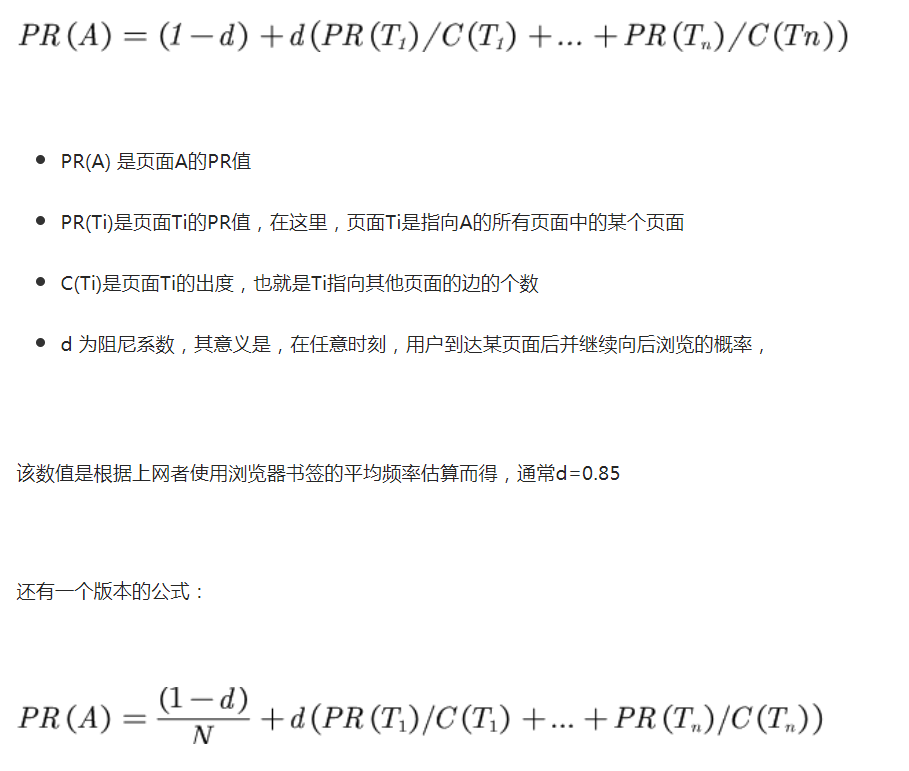
{'1': {'行政': 0.11584717374518982, '规划': 0.0, '政策': 0.05792358687259491}, '2': {'规划': 0.0, '政策': 0.13515503603605478}, '3': {'行政': 0.2027325540540822, '规划': 0.0}}

## Textrank

TextRank算法是基于PageRank算法，PageRank算法的核心思想是，认为网页重要性由两部分组成：

① 如果一个网页被大量其他网页链接到说明这个网页比较重要，即被链接网页的数量；

② 如果一个网页被排名很高的网页链接说明这个网页比较重要，即被链接网页的权重。



N为页面总数

Textrank的思想是把文本拆分成词汇作为网络节点，组成词汇网络图模型，将词语间的相似关系看成是一种推荐或投票关系，使其可以计算每一个词语的重要性。

基于TextRank的文本关键词抽取是利用局部词汇关系，即共现窗口，对候选关键词进行排序，该方法的步骤如下：

（1） 对文本D进行分词、词性标注和去除停用词等数据预处理操作。本分采用结巴分词，保留'n','nz','v','vd','vn','l','a','d'这几个词性的词语，最终得到n个候选关键词，即D=[t1,t2,…,tn] ；

（2） 构建候选关键词图G=(V,E)，其中V为节点集（由候选关键词组成），并采用共现关系构造任两点之间的边，两个节点之间仅当它们对应的词汇在长度为K的窗口中共现则存在边，K表示窗口大小即最多共现K个词汇；

（3） 根据 公式 迭代 计算 各节点的权重，直至收敛；

（4） 对节点权重进行倒序排列，得到排名前TopN个词汇作为文本关键词。

说明：Jieba库中包含jieba.analyse.textrank函数可直接实现TextRank算法，本文采用该函数进行实验。

代码实现：



## Word2vec

对于用词向量表示的文本词语，通过K-Means算法对文章中的词进行聚类，选择聚类中心作为文章的一个主要关键词，计算其他词与聚类中心的距离即相似度，选择topN个距离聚类中心最近的词作为文本关键词，而这个词间相似度可用Word2Vec生成的向量计算得到。

假设Dn为测试语料的大小，使用该方法进行文本关键词抽取的步骤如下所示：

（1） 对Wiki中文语料进行Word2vec模型训练，参考文章“利用Python实现wiki中文语料的word2vec模型构建”（ <http://www.jianshu.com/p/ec27062bd453> ），得到词向量文件“wiki.zh.text.vector”；

（2） 对于给定的文本D进行分词、词性标注、去重和去除停用词等数据预处理操作。本分采用结巴分词，保留'n','nz','v','vd','vn','l','a','d'这几个词性的词语，最终得到n个候选关键词，即D=[t1,t2,…,tn] ；

（3） 遍历候选关键词，从词向量文件中抽取候选关键词的词向量表示，即WV=[v1，v2，…，vm]；

（4） 对候选关键词进行K-Means聚类，得到各个类别的聚类中心；

（5） 计算各类别下，组内词语与聚类中心的距离（欧几里得距离），按聚类大小进行升序排序；

（6） 对候选关键词计算结果得到排名前TopN个词汇作为文本关键词。

步骤（4）中需要人为给定聚类的个数，本文测试语料是新闻文本，因此只需聚为1类，各位可根据自己的数据情况进行调整；步骤（5）中计算各词语与聚类中心的距离，常见的方法有欧式距离和曼哈顿距离，本文采用的是欧式距离，计算公式如下：

IMG_256

Word2vec代码实现

from gensim.models.word2vec import Word2Vec  
import gensim  
*# 读取数据，用gensim中的word2vec训练词向量*file = open('sentence.txt')  
sss = []  
while True:  
 ss = file.readline().replace('\n', '').strip()  
 if ss == '':  
 break  
 s1 = ss.split(" ")  
 sss.append(s1)  
file.close()  
model = Word2Vec(size=200, workers=5, sg=1) *# 生成词向量为200维，考虑上下5个单词共10个单词，采用sg=1的方法也就是skip-gram*model.build\_vocab(sss)  
model.train(sss, total\_examples=model.corpus\_count, epochs=model.iter)  
model.save('w2v\_model') *# 保存模型*new\_model = gensim.models.Word2Vec.load('w2v\_model') *# 调用模型*sim\_words = new\_model.most\_similar(positive=['女人'])  
for word, similarity in sim\_words:  
 print(word, similarity) *# 输出’女人‘相近的词语和概率*print(model['女孩'])

三种方法对比：

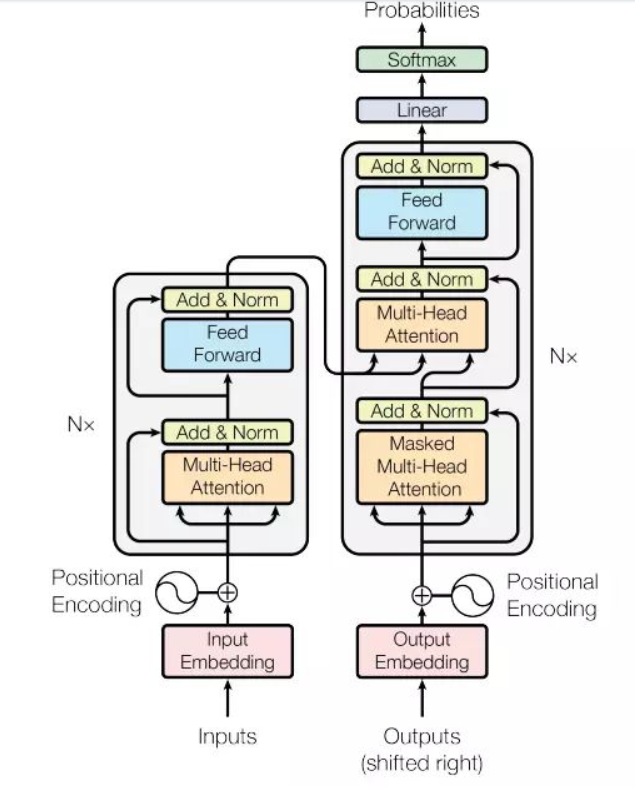
本文总结了三种常用的抽取文本关键词的方法：TF-IDF、TextRank和Word2Vec词向量聚类，因本文使用的测试语料较为特殊且数量较少，未做相应的结果分析，根据观察可以发现，得到的十个文本关键词都包含有文本的主旨信息，其中TF-IDF和TextRank方法的结果较好，Word2Vec词向量聚类方法的效果不佳，这与文献[8]中的结论是一致的。

文献[8]中提到，对单文档直接应用Word2Vec词向量聚类方法时，选择聚类中心作为文本的关键词本身就是不准确的，因此与其距离最近的N个词语也不一定是关键词，因此用这种方法得到的结果效果不佳；而TextRank方法是基于图模型的排序算法，在单文档关键词抽取方面有较为稳定的效果，因此较多的论文是在TextRank的方法上进行改进而提升关键词抽取的准确率。

另外，本文的实验目的主要在于讲解三种方法的思路和流程，实验过程中的某些细节仍然可以改进。例如Word2Vec模型训练的原始语料可加入相应的专业性文本语料；标题文本往往包含文档的重要信息，可对标题文本包含的词语给予一定的初始权重；测试数据集可采集多个分类的长文本，与之对应的聚类算法KMeans()函数中的n\_clusters参数就应当设置成分类的个数；根据文档的分词结果，去除掉所有文档中都包含某一出现频次超过指定阈值的词语。

# 二、TRANSFORMER

Transformer是谷歌在2017年发布的一个用来替代RNN和CNN的新的网络结构，Transformer本质上就是一个Attention结构，它能够直接获取全局的信息，而不像RNN需要逐步递归才能获得全局信息，也不像CNN只能获取局部信息，并且其能够进行并行运算，要比RNN快上很多倍。



## 1、Encoder和Decoder结构

Encoder: 这里面有 N=6 个 一样的layers, 每一层包含了两个sub-layers. 第一个sub-layer 就是多头注意力层（multi-head attention layer） 然后是一个简单的全连接层。

Multi-Head Self-Attention

Position-Wise Feed-Forward Network (全连接层)

Encoder 的输入由 Input Embedding 和 Positional Embedding 求和组成，这里还有一个残差连接 （residual connection), 在这个基础上， 还有一个layer norm.。

Decoder: 这里同样是有六个一样的Layer是，但是这里的layer 和encoder 不一样， 这里的layer 包含了三个sub-layers,  其中有 一个self-attention layer, encoder-decoder attention layer 最后是一个全连接层。

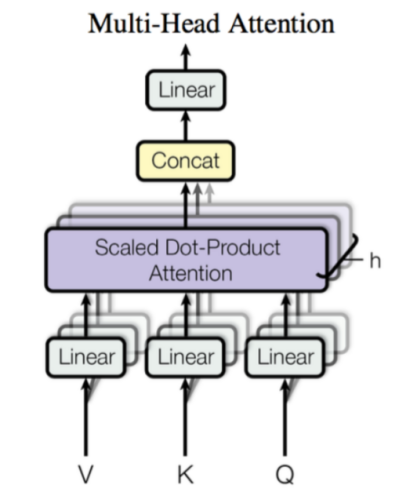
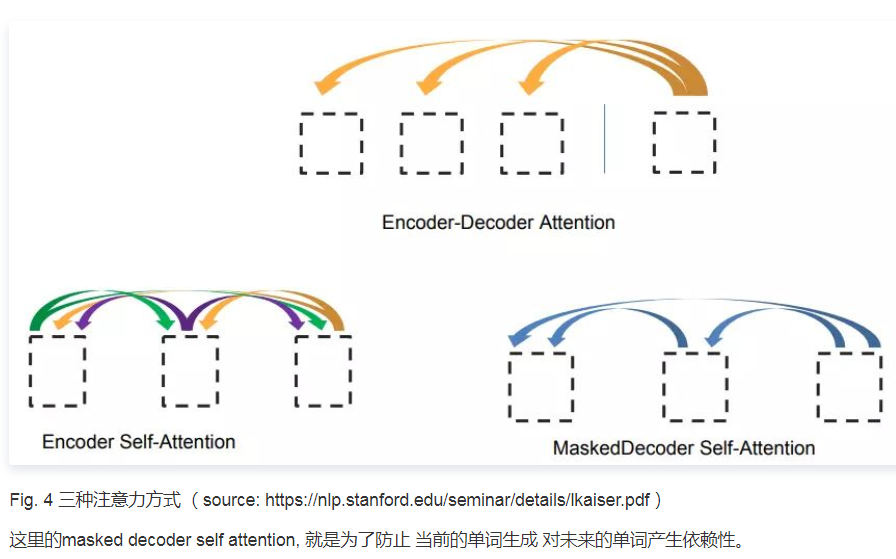
Multi-Head Self-Attention

Multi-Head Context-Attention

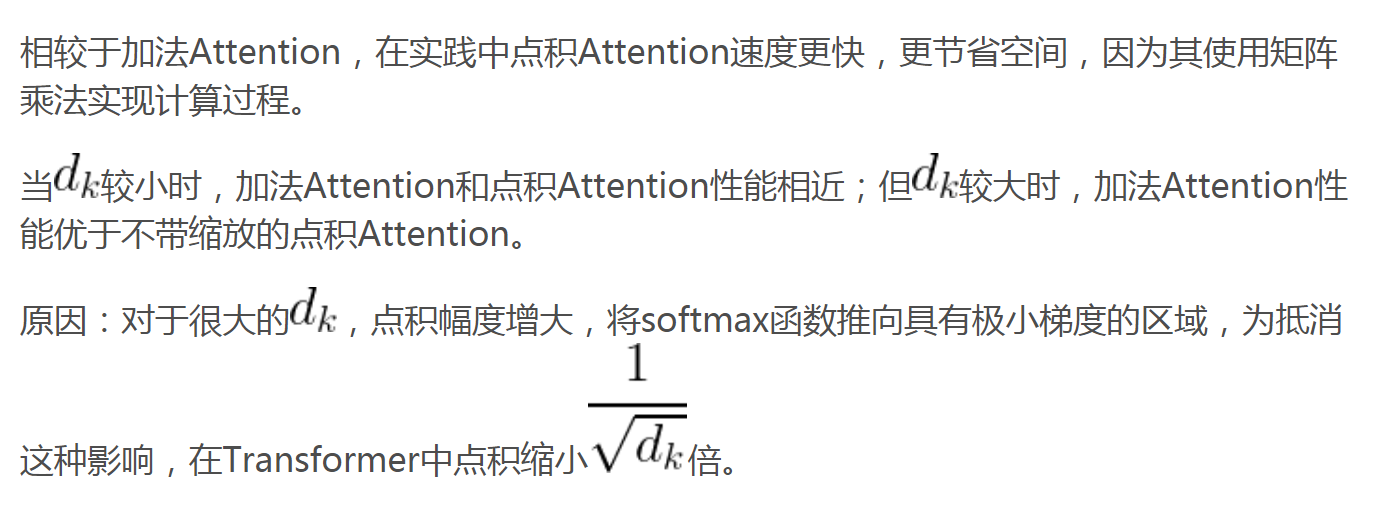
Position-Wise Feed-Forward Network

前两个sub-layer 都是基于multi-head attention layer.  这里有个特别点就是masking,  masking 的作用就是防止在训练的时候 使用未来的输出的单词。 比如训练时， 第一个单词是不能参考第二个单词的生成结果的。 Masking就会把这个信息变成0， 用来保证预测位置 i 的信息只能基于比 i 小的输出。Decoder 的初始输入由 Output Embedding 和 Positional Embedding 求和得到。

## 2、Multi-Head Self-Attention

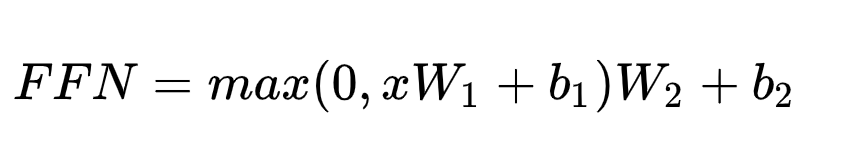
 

关于使用缩放dk的原因：



## 3、Position-wise Feed-Forward network

这是一个全连接网络，包含两个线性变换和一个非线性函数 (实际上就是 ReLU)。公式如下



代码

这个线性变换在不同的位置都表现地一样，并且在不同的层之间使用不同的参数。

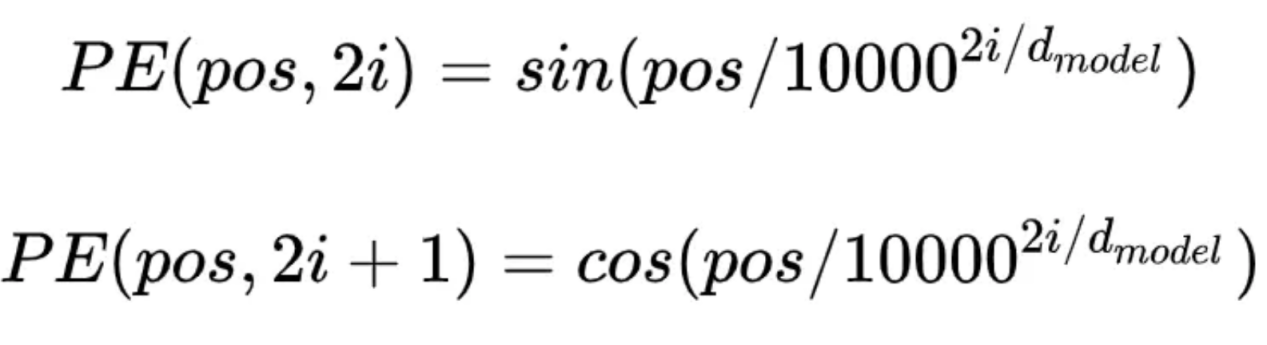
这里实现上用到了两个一维卷积。

残差网络：

残差网络有什么好处呢？显而易见：因为增加了 x 项，那么该网络求 x 的偏导的时候，多了一项常数 1，所以反向传播过程，梯度连乘，也不会造成梯度消失。

## 4、Positional embedding

因为 Transformer 利用 Attention 的原因，少了对序列的顺序约束，这样就无法组成有意义的语句。为了解决这个问题，Transformer 对位置信息进行编码。

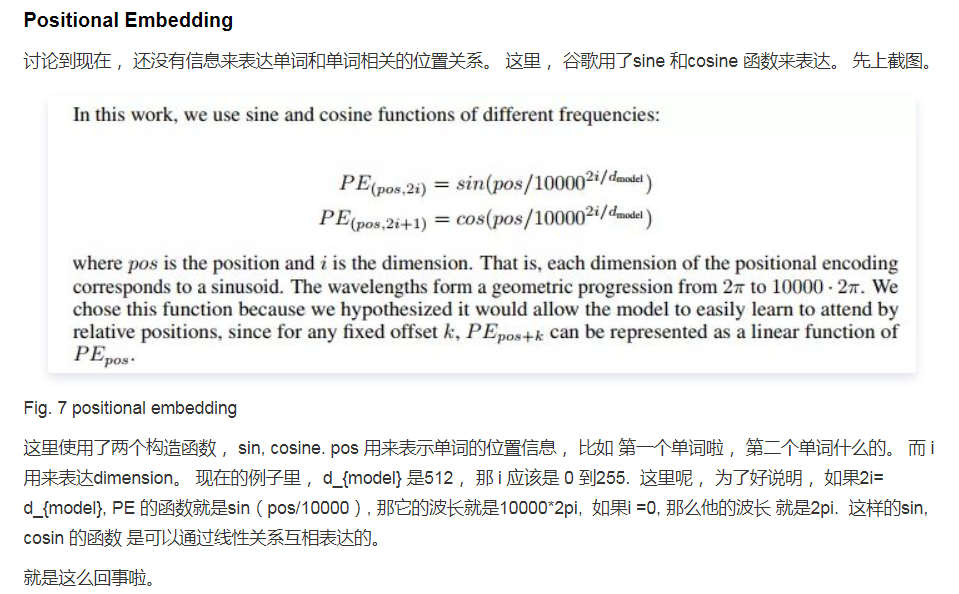


pos 指词语在序列中的位置，偶数位置，使用正弦编码，奇数位置，使用余弦编码。

上述公式解释：给定词语的位置 pos，我们可以把它编码成 d\_model 维的向量！也就是说，位置编码的每一个维度对应正弦曲线，波长构成了从 IMG_256到 IMG_257的等比序列。

上面的位置编码是绝对位置编码。但是词语的相对位置也非常重要。这就是论文为什么要使用三角函数的原因！

正弦函数能够表达相对位置信息



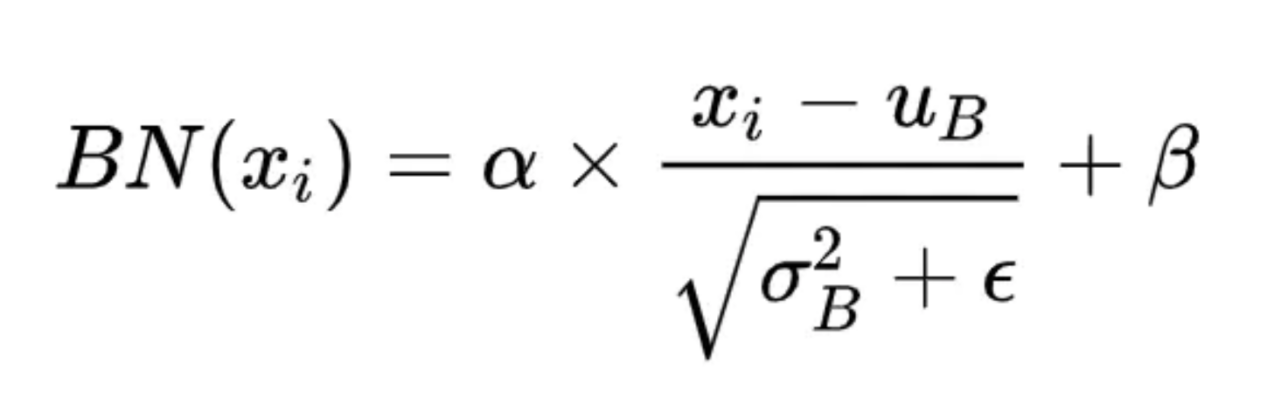
## **5、两种Normalization layer**

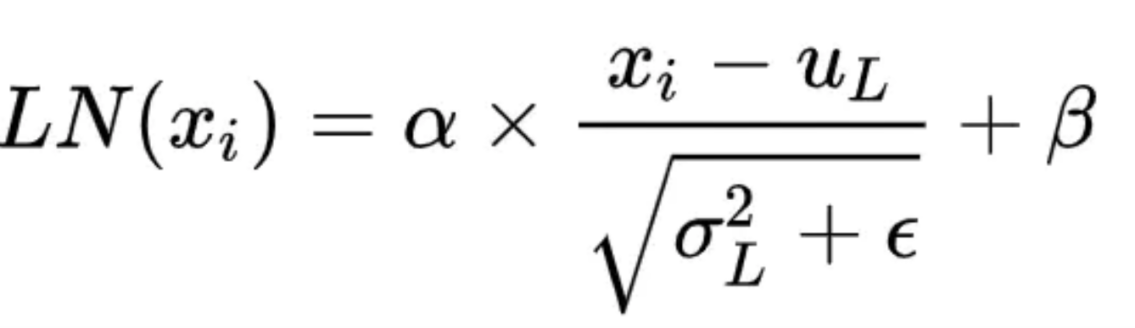
Normalization 有很多种，但是它们都有一个共同的目的，那就是把输入转化成均值为 0 方差为 1 的数据。我们在把数据送入激活函数之前进行 Normalization（归一化），因为我们不希望输入数据落在激活函数的饱和区。

在深层网络训练过程中，由于网络中参数变化而引起内部节点数据分布发生变化的这一过程被称作 Internal Covariate Shift。（内部协变量偏移）

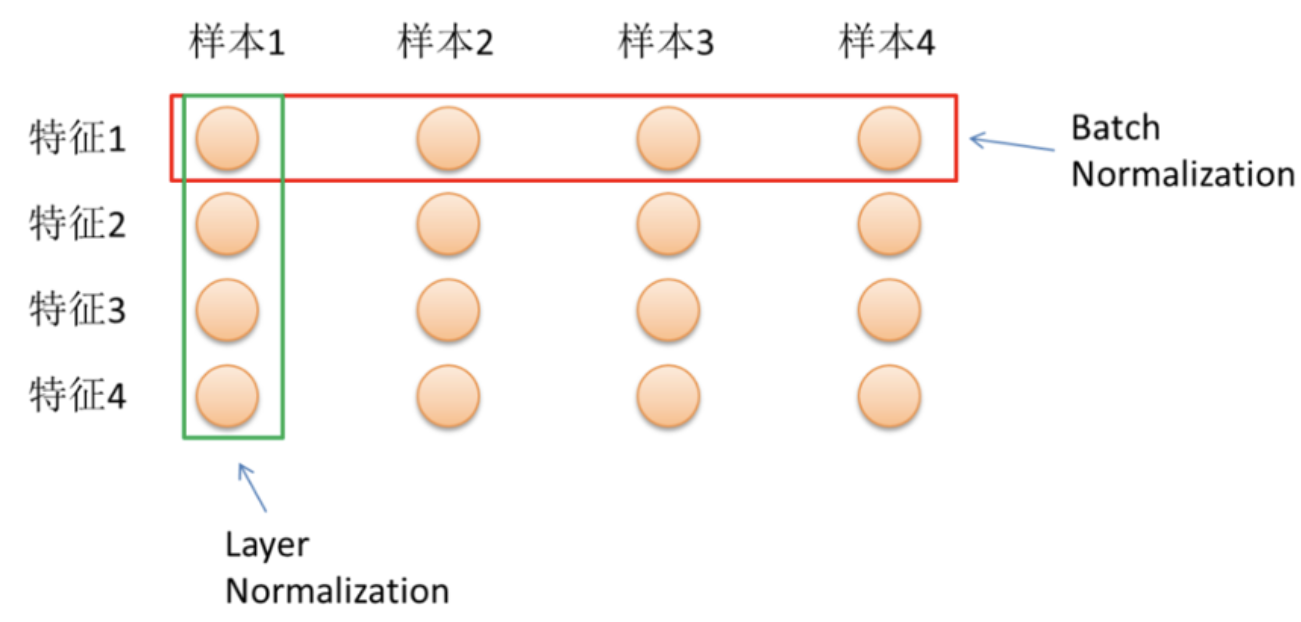
BN 就是为了解决这一问题，一方面可以简化计算过程，一方面经过规范化处理后让数据尽可能保留原始表达能力。

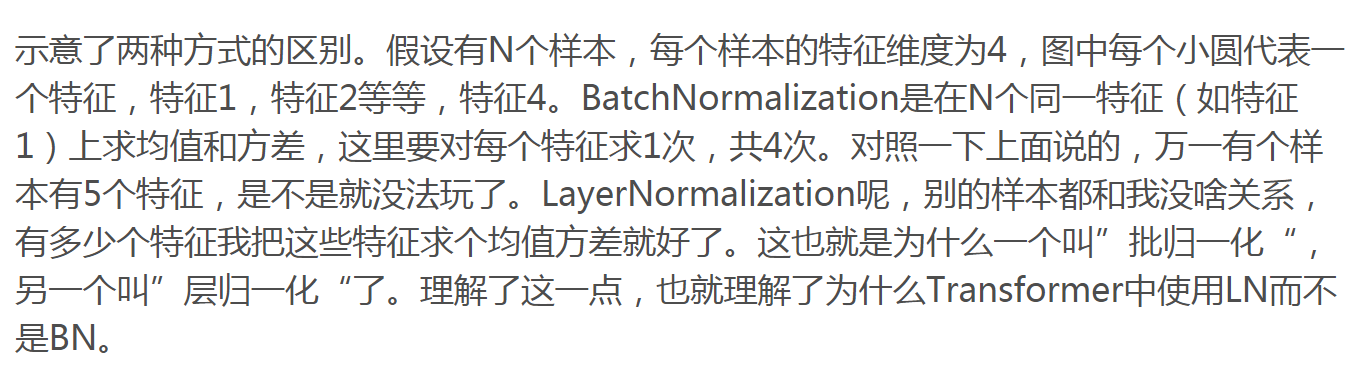
BN 的主要思想是：在每一层的每一批数据上进行归一化。

  
  
 LN 是****在每一个样本上计算均值和方差，而不是 BN 那种在批方向计算均值和方差****



**为什么用layer normalization而不是batch normalization**





## **6、两种mask**

Transformer 模型里面涉及两种 Mask。分别是 Padding Mask 和 Sequence Mask。

其中，Padding Mask 在所有的 Scaled Dot-Product Attention 里面都需要用到，而 Sequence Mask 只有在 Decoder 的 Self-Attention 里面用到。

**Padding Mask**

什么是 Padding Mask 呢？回想一下，我们的每个批次输入序列长度是不一样的。我们要对输入序列进行对齐！就是给在较短的序列后面填充 0。因为这些填充的位置，其实是没什么意义的，所以我们的 Attention 机制不应该把注意力放在这些位置上，所以我们需要进行一些处理。

具体的做法是，把这些位置的值加上一个非常大的负数(负无穷)，这样的话，经过 Softmax，这些位置的概率就会接近 0 ！

而我们的 Padding Mask 实际上是一个张量，每个值都是一个 Boolen，值为 False 的地方就是我们要进行处理的地方。

def padding\_mask(seq\_k, seq\_q):

# seq\_k 和 seq\_q 的形状都是 [B,L]

len\_q = seq\_q.size(1)

# `PAD` is 0

pad\_mask = seq\_k.eq(0)

# shape [B, L\_q, L\_k]

pad\_mask = pad\_mask.unsqueeze(1).expand(-1, len\_q, -1)

return pad\_mask

**Sequence mask**

文章前面也提到，Sequence Mask 是为了使得 Decoder 不能看见未来的信息。也就是对于一个序列，在 time\_step 为 t 的时刻，我们的解码输出应该只能依赖于 t 时刻之前的输出，而不能依赖 t 之后的输出。因此我们需要想一个办法，把 t 之后的信息给隐藏起来。

那么具体怎么做呢？也很简单：产生一个上三角矩阵，上三角的值全为 1，下三角的权值为 0，对角线也是 0。把这个矩阵作用在每一个序列上，就可以达到我们的目的啦。

本来 Mask 只需要二维的矩阵即可，但是考虑到我们的输入序列都是批量的，所以我们要把原本 2 维的矩阵扩张成 3 维的张量。

def sequence\_mask(seq):

batch\_size, seq\_len = seq.size()

mask = torch.triu(torch.ones((seq\_len, seq\_len), dtype=torch.uint8),diagonal=1)

mask = mask.unsqueeze(0).expand(batch\_size, -1, -1) # [B, L, L]

return mask

对于decoder的self-attention，里面使用到的scaled dot-product attention，同时需要padding mask 和 sequence mask 作为 attn\_mask，具体实现就是两个 mask 相加作为attn\_mask，其他情况，attn\_mask 一律等于 padding mask。

# **三、BERT**

## 1、BERT的基本原理是什么？

BERT可以看成一个自编码的语言模型，主要用两个任务训练该模型:

MLM(Masked LM)

输入一句话的时候，随机地选一些要预测的词，然后用一个特殊的符号[MASK]来代替它们，之后让模型根据所给的标签去学习这些地方该填的词

NSP(Next Sentence Prediction)

在双向语言模型的基础上额外增加了一个句子级别的连续性预测任务，即预测输入BERT的两段文本是否为连续的文本

## 2. BERT是怎么用Transformer的？

BERT只使用了Transformer的Encoder模块,与Transformer本身的Encoder端相比，BERT的Transformer Encoder端输入的向量表示，多了Segment Embeddings。

## 3. 请简要介绍一下Masked LM

MLM可以看作是一种引入噪声的手段，增强模型的泛化能力

具体步骤：

如果某个Token在被选中的15%个Token里，则按照下面的方式随机的执行：

80%的概率替换成[MASK]，比如my dog is hairy → my dog is [MASK]

10%的概率替换成随机的一个词，比如my dog is hairy → my dog is apple

10%的概率替换成它本身，比如my dog is hairy → my dog is hairy

好处：

BERT并不知道[MASK]替换的是这15%个Token中的哪一个词(意思是输入的时候不知道[MASK]替换的是哪一个词，但是输出还是知道要预测哪个词的)，而且任何一个词都有可能是被替换掉的，比如它看到的apple可能是被替换的词。这样强迫模型在编码当前时刻的时候不能太依赖于当前的词，而要考虑它的上下文，甚至对其上下文进行”纠错”。比如上面的例子模型在编码apple是根据上下文my dog is应该把apple(部分)编码成hairy的语义而不是apple的语义。

## 4. 请简要介绍一下NSP(Next Sentence Prediction)

判断第二句话在文本中是否紧跟在第一句话之后，类似于段落排序

将一篇文章的各段打乱，通过重新排序把原文还原出来，这其实需要对全文大意有充分、准确的理解。Next Sentence Prediction 任务实际上就是段落重排序的简化版：只考虑两句话，判断是否是一篇文章中的前后句。在实际预训练过程中，作者从文本语料库中随机选择 50% 正确语句对和 50% 错误语句对进行训练，与 Masked LM 任务相结合，让模型能够更准确地刻画语句乃至篇章层面的语义信息。

## 5. EMO、GPT、BERT三者之间有什么区别？

特征提取器：

ELMO采用LSTM进行提取，GPT和BERT则采用Transformer进行提取。很多任务表明Transformer特征提取能力强于LSTM

ELMO用1层静态向量+2层LSTM，多层提取能力有限，而GPT和BERT中的Transformer可采用多层，并行计算能力强。

单/双向语言模型：

GPT采用单向语言模型，EMLo和BERT采用双向语言模型。但是EMLo实际上是两个单向语言模型（方向相反）的拼接，这种融合特征的能力比BERT一体化融合特征方式弱。

GPT和BERT都采用Transformer，Transformer是encoder-decoder结构，GPT的单向语言模型采用decoder部分，decoder的部分见到的都是不完整的句子；bert的双向语言模型则采用encoder部分，采用了完整句子

## 6. BERT的输入和输出分别是什么？

bert的输入

bert的输入是三部分，input\_id segment\_id 和position\_id  
input\_id 就是将输入转换成词表中的id，通过查询字向量表将文本中的每个字转换为一维向量，也可以利用Word2Vector等算法进行预训练以作为初始值；  
segment\_id（token\_type\_id）就是区分句子是第几个句子 用0和1表示  
position\_id是记录句子中词的顺序

形式：[cls]上一句话，[sep]下一句话.[sep]

bert的输出

bert的输出有两种，一种是get\_sequence\_out(),获取的是整个句子每一个token的向量表示，输出shape是[batch\_size, seq\_length, hidden\_size]；  
另一种是get\_pooled\_out()，获取的是整个句子中的[cls]的表示,输出shape是[batch size,hidden size]。

## 7. Mask LM相对于CBOW有什么异同点？

相同点：

都是根据上下文预测中心词

不同点：

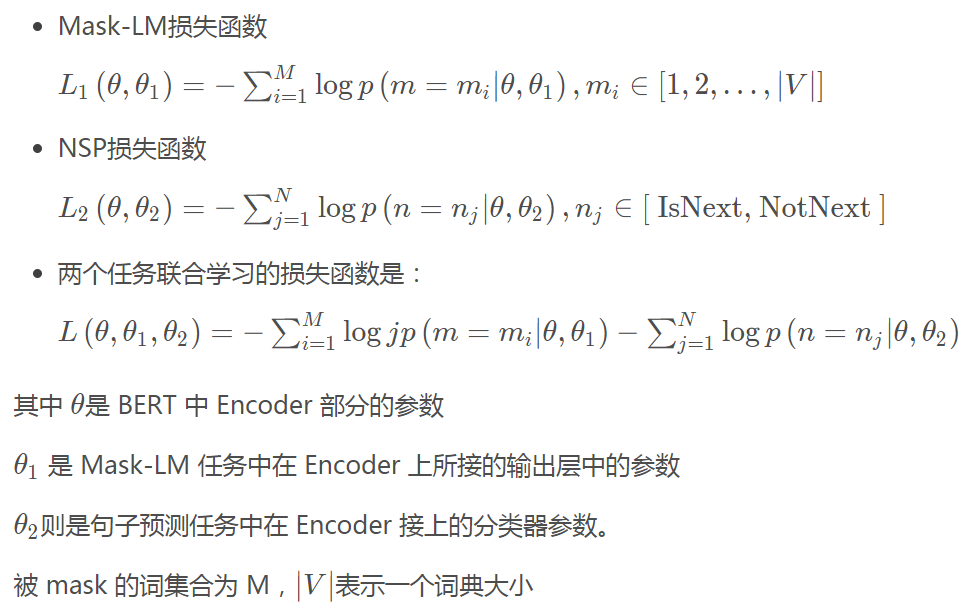
CBOW中的输入数据只有待预测单词的上下文，而BERT的输入是带有\*\*[MASK] token\*\*的“完整”句子

CBOW模型训练后，每个单词的word embedding是唯一的，因此并不能很好的处理一词多义的问题，而BERT模型得到的word embedding(token embedding)融合了上下文的信息，就算是同一个单词，在不同的上下文环境下，得到的word embedding是不一样的。

Word2vec的inference就是对该矩阵的embedding\_lookup，对同一个词得到的词向量肯定是不变的，所以是静态的。

而bert的inference过程会复杂很多，会利用到上下文的信息经过transformer编码（即self-attention交互），同一个词如果上下文信息不同，那么就会得到不同的embedding，所以是动态的。

## 8. BERT的两个预训练任务对应的损失函数是什么



## **9. BERT优缺点**

优点  
bert将双向 Transformer 用于语言模型，Transformer 的 encoder 是一次性读取整个文本序列，而不是从左到右或从右到左地按顺序读取，这个特征使得模型能够基于单词的两侧学习，相当于是一个双向的功能。

bert 相较于rnn,lstm可以并发进行，并且可以提取不同层次的信息,反应更全面的句子语义。相较于 word2vec，根据句子上下文获取词义，避免了歧义出现。

缺点  
bert模型参数太多，而且模型太大，少量数据训练时，容易发生过拟合。  
bert模型预训练会出现mask,而在下游的微调任务中并不会出现，使得不匹配。  
bert模型会将词分成词根一样的词片，若随机遮掩一些词的时候，若遮掩住中间的的词片，则会发生不是根据上下文的语义预测的。

## **10、简述 bert**

Bert 是一个预训练语言模型，它主要有两个任务。第一个任务是将数据集中的句子随机遮掩一部分，通过训练，来预测这些词是什么，加强了句子内部之间的联系；第二个任务是判断两个句子是否是连续的上下句，通过训练来识别，加强了句子外部之间的联系。

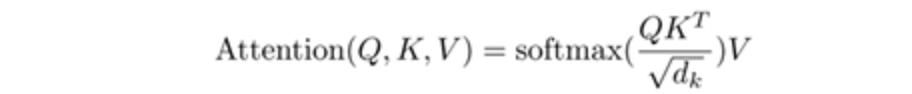
bert 的创新点在于它将双向 Transformer 用于语言模型，Transformer 的 encoder 是一次性读取整个文本序列，而不是从左到右或从右到左地按顺序读取，这个特征使得模型能够基于单词的两侧学习，相当于是一个双向的功能。

bert 相较于rnn,lstm可以并行进行，并且可以提取不同层次的信息,反应更全面的句子语义。相较于 word2vec，根据句子上下文获取词义，避免了歧义出现。缺点就是模型参数太多，而且模型太大，少量数据训练时，容易发生过拟合。

## **11、bert的模型结构**

bert 是由transformer的编码器构成的。  
小Bert是由12个transformer的编码器组成，大bert是由24个transformer的编码器组成，同时相较于transformer的输入而言，bert的输入多了segment id。

## **12、Multi-head Attention 多头注意力机制**



多头注意力机制定义一个超参数h(head数量)来提取多重语义，先将Query，Key，Value做线性变化，然后把Query，Key，Value的embedding维度分割成了h份，q与k做相似度计算，然后与v做加权求和得到注意力向量，再把这些注意力向量拼接起来，再通过线性变换得到最终的结果。注意头之间参数不共享，每次Q，K，V进行线性变换的参数是不一样的。

## **13、bert适合的任务**

bert可以用来做分类、句子语义相似度、阅读理解等任务  
对于分类：获得输出的[cls]的向量，做softmax得到每个类别的概率  
对于语义相似度：获得输出的[cls]的向量，做softmax得到相似度  
对于阅读理解：输入[cls]问题[sep]文本[sep]，输出的是整句话的向量，将其分成两部分，转换成对应的答案开始和结束位置的概率。

## **14、Bert中如何获得词意和句意？**

get\_pooled\_out代表了涵盖了整条语句的信息 cls向量 (bacth\_size\*hidden\_size)

get\_sentence\_out代表了这个获取每个token的output 输出  (bacth\_size\*sen\_len\*hidden\_size)

## **15、源码中Attention后实际的流程是如何的？**

Transform模块中：在残差连接之前，对output\_layer进行了dense+dropout后再合并input\_layer进行的layer\_norm得到的attention\_output

所有attention\_output得到并合并后，也是先进行了全连接，而后再进行了dense+dropout再合并的attention\_output之后才进行layer\_norm得到最终的layer\_output

## **16、为什么要在Attention后使用残差结构？**

残差结构能够很好的消除层数加深所带来的信息损失问题

## **17、平时用官方Bert包么？耗时怎么样？**

第三方：bert\_serving

官方：bert\_base

耗时：64GTesla，64max\_seq\_length，80-90doc/s

在线预测只能一条一条的入参，实际上在可承受的计算量内batch越大整体的计算性能性价比越高

## **18、Bert的双向体现在什么地方？**

mask+attention，mask的word结合全部其他encoder word的信息

## **19、为什么选取的15%的词中有80%用[mask]来替换，10%用原词，剩余的10%用随机词来替换，不全部用mask来遮掩:**

是因为下游任务微调中不会出现mask，这样会导致预训练模型和下游任务不匹配。同时在预测时，因为不知道这个词是否是正确的，会使模型更加依赖上下文，有一定的纠错能力。

=====另一版本======

mask只会出现在构造句子中，当真实场景下是不会出现mask的，全mask不match句型了

随机替换也帮助训练修正了[unused]和[UNK]；强迫文本记忆上下文信息

## **20、为什么BERT有3个嵌入层，它们都是如何实现的？**

input\_id是语义表达，和传统的w2v一样，方法也一样的lookup

segment\_id是辅助BERT区别句子对中的两个句子的向量表示，从[1,embedding\_size]里面lookup

position\_id是为了获取文本天生的有序信息，否则就和传统词袋模型一样了，从[511,embedding\_size]里面lookup

## **21、bert的具体网络结构，以及训练过程，bert为什么火，它在什么的基础上改进了些什么？**

bert是用了transformer的encoder侧的网络，作为一个文本编码器，使用大规模数据进行预训练，预训练使用两个loss，一个是mask LM，遮蔽掉源端的一些字（可能会被问到mask的具体做法，15%概率mask词，这其中80%用[mask]替换，10%随机替换一个其他字，10%不替换，至于为什么这么做，那就得问问BERT的作者了\{捂脸}），然后根据上下文去预测这些字，一个是next sentence，判断两个句子是否在文章中互为上下句，然后使用了大规模的语料去预训练。在它之前是GPT，GPT是一个单向语言模型的预训练过程（它和gpt的区别就是bert为啥叫双向 bi-directional），更适用于文本生成，通过前文去预测当前的字。下图为transformer的结构，bert的网络结构则用了左边的encoder。

## **22、讲讲multi-head attention的具体结构**

BERT由12层transformer layer（encoder端）构成，首先word emb , pos emb（可能会被问到有哪几种position embedding的方式，bert是使用的哪种）, sent emb做加和作为网络输入，每层由一个multi-head attention, 一个feed forward 以及两层layerNorm构成，一般会被问到multi-head attention的结构，具体可以描述为，一个768的hidden向量，被映射成query， key， value。 然后三个向量分别切分成12个小的64维的向量，每一组小向量之间做attention。

hidden(768) -> query(768) -> 12 x 64

hidden(768) -> key(768) -> 12 x 64

hidden(768) -> val(768) -> 12 x 64

然后query和key之间做attention，得到一个12乘以12的权重矩阵，然后根据这个权重矩阵加权val中切分好的12个64维向量，得到一个12 x 64的向量，拉平输出为768向量。

问题2.5: Bert 采用哪种Normalization结构，LayerNorm和BatchNorm区别，LayerNorm结构有参数吗，参数的作用？

采用LayerNorm结构，和BatchNorm的区别主要是做规范化的维度不同，BatchNorm针对一个batch里面的数据进行规范化，针对单个神经元进行，比如batch里面有64个样本，那么规范化输入的这64个样本各自经过这个神经元后的值（64维），LayerNorm则是针对单个样本，不依赖于其他数据，常被用于小mini-batch场景、动态网络场景和 RNN，特别是自然语言处理领域，就bert来说就是对每层输出的隐层向量（768维）做规范化，图像领域用BN比较多的原因是因为每一个卷积核的参数在不同位置的神经元当中是共享的，因此也应该被一起规范化。

## **23、self-attention相比lstm优点是什么？**

bert通过使用self-attention + position embedding对序列进行编码，lstm的计算过程是从左到右从上到下（如果是多层lstm的话），后一个时间节点的emb需要等前面的算完，而bert这种方式相当于并行计算，虽然模型复杂了很多，速度其实差不多。