1、transformer介绍（图，原理，公式）

位置向量那里不需要详细讲，只需要知道这是通过一个数学公式来计算位置信息的一个向量。

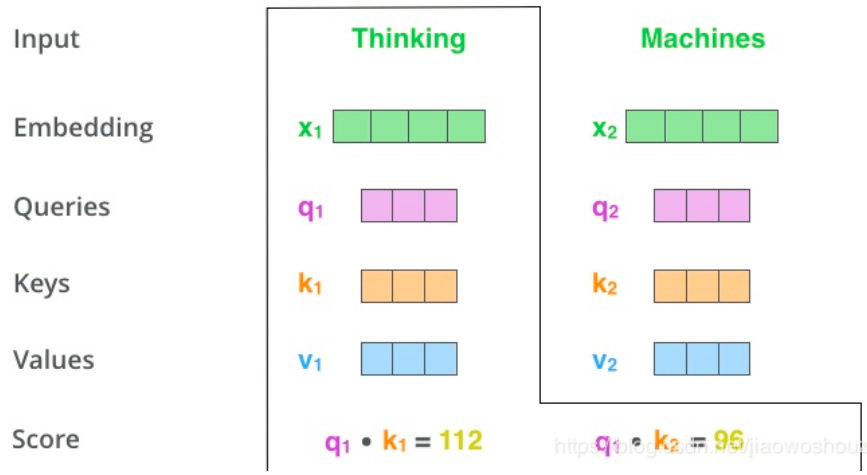
Decoder部分不需要详细讲，因为BERT里面没用到Decoder。

残差网络的作用：使得训练能够在比较深的网络中也能够较好地收敛，如果没有残差网络，深的网络可能会比浅层网络的准确率还低。这里的证明和理解消化需要一定的基础。

这种通过 query 和 key 的相似性程度来确定 value 的权重分布的方法被称为scaled dot-product attention。其实scaled dot-Product attention就是我们常用的使用点积进行相似度计算的attention，只是多除了一个（为K的维度）起到调节作用，使得内积不至于太大。

首先，self-attention会计算出三个新的向量，在论文中，向量的维度是512维，我们把这三个向量分别称为Query、Key、Value，这三个向量是用embedding向量与一个矩阵相乘得到的结果，这个矩阵是随机初始化的，维度为（64，512）注意第二个维度需要和embedding的维度一样，其值在BP的过程中会一直进行更新，得到的这三个向量的维度是64低于embedding维度的。

计算self-attention的分数值，该分数值决定了当我们在某个位置encode一个词时，对输入句子的其他部分的关注程度。这个分数值的计算方法是Query与Key做点乘，以下图为例，首先我们需要针对Thinking这个词，计算出其他词对于该词的一个分数值，首先是针对于自己本身即q1·k1，然后是针对于第二个词即q1·k2



Self-attention即K=V=Q，例如输入一个句子，那么里面的每个词都要和该句子中的所有词进行attention计算。目的是学习句子内部的词依赖关系，捕获句子的内部结构。

今天天气真好，“今天”这个词与“今天” “天气” “真好”这三个词进行计算相似度，再将这个相似度向量乘以“今天”本身的词向量，那么相当于此时“今天”这个词向量携带了更多的信息，这个信息能说明“今天”这个词和句子中其他词的依赖关系和联系，也能同时表示句子的内部结构。

如果self-attention这里没有完全听懂，其实总结下来就两点，第一，从宏观上，整体上来讲，self-attention这个结构就类似于CNN，RNN，作用是用于提取特征，所以这里的我的输入是一个句子，encoder之后，得到的是句子中每个词的特征向量（tf-idf也是词的特征的一种，这里是把词表示成了一个向量，也能表示词的更多信息）。第二，self-attention（之所以不用CNN，RNN这样的结构来辅助）（因为之前有论文提出了在CNN和RNN的基础上再加个attention）最重要的点在于它提出了QKV这样的形式去计算句子中的词的特征，在这个计算过程中得到了句子中词之间的联系和句子的内部结构。

mask 表示掩码，它对某些值进行掩盖，使其在参数更新时不产生效果。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？因为每个批次输入序列长度是不一样的也就是说，我们要对输入序列进行对齐。具体来说，就是给在较短的序列后面填充 0。但是如果输入的序列太长，则是截取左边的内容，把多余的直接舍弃。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Boolean，值为 false 的地方就是我们要进行处理的地方。

2、语言模型介绍（正常的语言模型原理及公式）
词向量是语言模型的副产物。

3、bert的语言模型介绍（？？这里要再看看）及创新点，bert的使用优点。

思想上的创新点（其实也不算它想到的），提出了双向的语言模型（其实这个双向的方式也不是他最早提出，双向的语言模型在它之前也有比如ELMO，但BERT的作者认为ELMO并不能很好地联系上下文语境，他认为ELMO的双向并不充分，ELMO的融合方式过于简单，因为ELMO是先分别计算训练左右语境再进行拼接结合，他评价ELMO用的词是shallow concatenation，然后BERT作者认为他提出的这个是Deep concatenation，能够充分融合上下文语境。）
图中的transformer可以先理解为LSTM这样的一个处理词向量的结构，后面我会再详细介绍。
（图中的T1，T2是什么意思？是经过transformer处理后的输出？）
那么BERT是如何做到深层的结合上下文语境的，主要在于他提出的训练语言模型的方式，这也是他的两个创新点（这是他训练语言模型方式的创新）：
1、MLM的方式 2、在双向语言模型的基础上增加了一个句子级别的连续性预测任务。这里的细节后面会再提到。
BERT的模型架构就是这样，论文也通过实验证明了，LARGE版本的BERT效果要比BASE版本的效果要好，但LARGE版本需要更多的资源去运行。我做实验一般都用base版本。下面先说一下图中的Trm，也就是transformer。等一下可以再回头过来看看这几个参数的含义。
巴拉巴拉说transformer。
BERT的输入由三部分组成，巴拉巴拉。
得到BERT的输入之后，我们要进行预训练任务，这也是它的创新点，

为了防止微调时没见过，所以有5%时正常的？？

参数等于340M是所有参数的个数

使用学习的位置嵌入是什么意思？We use learned positional embeddings with supported sequence lengths up to 512 tokens。

要预测的词的原先的样子（比如是mask或者随机词或者正常词）有什么影响？。。我之前觉得都是相当于挖空了让算法预测出正常的词。

如何存储词向量（微调的时候怎么搞的）。这里是导入模型后，利用函数得到模型的各个transformer层的输出结果。

为什么要搞个50%的正常句子，50%的异常句子这样的形式。当然是想让模型学习到哪些情况是正确的，哪些是错误的。

BERT的预训练模型的训练数据大小，训练时间长度，训练的仪器资源（论文）

CLS这个东西代表的是整个句子的特征向量。所以才会在预测句子是否为下一句这个任务中直接使用CLS的特征向量进行使用。

现在基本性能好一些的NLP模型，例如OpenAI GPT，google的BERT，在数据预处理的时候都会有WordPiece的过程。
WordPiece字面理解是把word拆成piece一片一片，其实就是这个意思。
WordPiece的一种主要的实现方式叫做BPE（Byte-Pair Encoding）双字节编码。
BPE的过程可以理解为把一个单词再拆分，使得我们的此表会变得精简，并且寓意更加清晰。
比如"loved","loving","loves"这三个单词。其实本身的语义都是"爱"的意思，但是如果我们以单词为单位，那它们就算不一样的词，在英语中不同后缀的词非常的多，就会使得词表变的很大，训练速度变慢，训练的效果也不是太好。
BPE算法通过训练，能够把上面的3个单词拆分成"lov","ed","ing","es"几部分，这样可以把词的本身的意思和时态分开，有效的减少了词表的数量。

两个任务的预测损失是这样的：1、预测mask的词，是预测mask掉的词是否是真的正常的词，是一个分类问题，最终得到的是字典大小的所有词的概率，我们取概率最大的词和正常的词进行计算他们的损失值。2、预测下一个句子这个任务，我们在做预处理数据的时候，做了一个标签来表示这条训练数据中的第二个句子是否是第一个句子的下一句，如果是下一句，那就记下标签为1，否则就是0，这样在预训练的时候，我们把CLS的特征向量（这个特征向量可以代表整个句子）再接一个全连接层和softmax的处理，将这个结果与实际的标签1或0进行计算他们的损失值，这是一个二分类问题。 预训练的总的损失值是两个任务的损失值进行相加。

需要多说一句的是，masked_lm_loss，用到了模型的sequence_output和embedding_table，这是因为对多个MASK的标签进行预测是一个标注问题，所以需要获取最后一层的整个sequence，而embedding_table用来反

embedding，这样就映射到token的学习了。而next_sentence_loss用到的是pooled_output，对应的是第一个token [CLS]，它一般用于分类任务的学习。

注意微调和基于特征的区别，根据前面所看到的微调使用时的图片可以知道，微调实际上是利用了语言模型的结构，也就是前面所展示的包含了多个transformer这样的一个结构，而基于特征则是将词向量或者句向量从语言模型中拿出来再对这个词向量根据自己的任务进行使用，举个最简单的例子，我可以取出bert训练好的词向量，比如“中国”和“美国”这两个词的向量，进行一个相似度比较，那么实际上他们的相似度应该是比较接近1的。当然这里的词向量也可以用在文本分类，匹配这些地方，NLP很多任务的第一步进行进行embedding做词嵌入，这里的词嵌入矩阵就可以用bert的词向量矩阵来代替。

用词的特征向量的好处我觉得主要是1、可以取与训练模型中的某一个网络层的输出作为词特征向量，并不一定是最后一层，最后一层也并不一定是最好的，而预训练模型只能取最后一层的输出，在该基础上我们再加任务所特定的网络结构。2、计算一个好的词向量特征是一个比较消耗资源和时间的事情，所以如果能把词向量特征单独取出来再进行下游任务，这样能减少资源和时间的开销，而我们也可以在这个词向量特征上快速地进行多次实验。

怎么固定住已经训练好的词向量使其不在根据模型的反向传播进行调整？（要清楚词嵌入那里的反向传播是怎么回事）

4、bert的github代码简介，以及如何使用bert（微调和提取词向量特征）

关于预训练模型这块：要介绍，模型输入，输出，损失函数（遮蔽和下一句预测），优化函数。
微调这块：介绍微调原理和如何微调。

讲一下run_classifier.py的使用和改写用作来自己的分类任务。

简单介绍下bert项目中各个文件的一些用法，在构建预训练模型的时候，我们可以得到不同的网络层的结果，比如transformer的最后一个输出层，也可以得到第一个CLS的输出结果。

提取词的特征向量的命令中，注意一下layers这个参数，这个参数表示要取出哪一层的特征向量。可以最后一层的或者其中某一层的特征向量，也可以取出多层输出结果concat之后的特征向量，也可以对多个层的输出向量进行相加处理再拿去使用。

训练和预测的时间统计一下。（工程会更关心这个）

很可惜的就是BERT的官方论文并没有中文方面的测评比较。百度的ERNIE倒是做了一些中文测评比较，但它没说明是用的bert-base还是bert-large版本。

BERT-wwm的测试结果在大部分常见的NLP任务中预测效果要比谷歌的BERT要好一点或者一样，会有0.几的提升。

BERT-BiLSTM-CRF-NER:虽然名字中有NER（实体命名识别），但并不是说只能用来做NER，这里主要是用它的服务端的功能，启动它的服务端之后可以加载训练好的模型，然后我们在客户端就可以根据服务端的IP和端口对要预测的句子进行预测。之前介绍的bert-as-service只能用来输出文本向量化之后的结果。预测时间的话，它自己的github上并没有给出具体的测评。

5、bert相关的开源工具库介绍（服务，部署等）（bert-as-service，keras-bert，bert-NER那个）
6、一些bert基础上的创新（百度ERNIE，哈工大的中文bert），一些新论文的方法（XLNET）

中文注释：
https://github.com/cnfive/cnbert/blob/master/run_pretraining.py

https://www.jianshu.com/p/22e462f01d8c
keras版本代码：
https://github.com/Separius/BERT-keras/blob/master/transformer/train.py

看看transformer的源码解析。https://yiyibooks.cn/yiyibooks/Attention_Is_All_You_Need/index.html
https://yiyibooks.cn/yiyibooks/Neural_Machine_Translation_by_Jointly_Learning_to_Align_and_Translate/index.html
https://blog.csdn.net/jiaowoshouzi/article/details/89073944

13：52-14：27

BERT的缺点：在生成类NLP中，就面临训练过程和应用过程不一致的问题，导致生成类的NLP任务做不太好。

MNLI:文本蕴含任务，句子对的分类任务。
MRPC:判断两个句子语义是否相同，二分类任务。
SST:情感分类任务。
SQUAD:阅读理解数据集。查找问题的答案的位置。