word2vec

**负采样**

由于训练词向量模型的目标不是为了得到一个多么精准的语言模型，而是为了获得它的副产物——词向量。所以要做到的不是在几万几十万个token中艰难计算下一词，而只需能做到在几个词中找到对的那个词就行。

**缺点是上下文无关(static）**

因而为了让句子有一个整体含义(context)，大家会在下游具体的NLP任务中基与词向量的序列做encoding操作。

Bi-LSTM

从前到后和后到前分别做一遍LSTM的encoding操作，从而获得两个方向的token联系，进而获得句子的context。

但是利用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。举一个例子，“这个餐厅脏得不行，没有隔壁好”，这里的“不行”是对“脏”的程度的一种修饰，通过BiLSTM可以更好的捕捉双向的语义依赖。

**不完全双向**

模型的前向和后向LSTM两个模型是分别训练的，最后得到的隐层向量直接拼接得到结果向量，并且在最后的Loss function中也是前向和后向的loss function直接相加，并非完全同时的双向计算。

**自己看见自己**

模型的前向和后向LSTM两个模型是分别训练的，最后得到的隐层向量直接拼接得到结果向量，并且在最后的Loss function中也是前向和后向的loss function直接相加，并非完全同时的双向计算。

当前位置需要预测的是B，已经在ABCD中出现了，这就会有问题。因而对于Bi-LSTM，只要层数增加，就是会存在“自己看见自己”的问题。

BERT（Bidirectional Encoder Representation from Transformers）

BERT的出现，彻底改变了预训练产生词向量和下游具体NLP任务的关系，提出龙骨级的训练词向量概念。

NLP任务分成两部分，预训练产生词向量，对词向量操作（下游具体NLP任务）。从word2vec到ELMo到BERT，做的其实主要是把下游具体NLP任务的活逐渐移到预训练产生词向量上。

word2vec——>ELMo：

　　结果：上下文无关的static向量变成上下文相关的dynamic向量，比如苹果在不同语境vector不同。

ELMo——>BERT：

　　结果：训练出的word-level向量变成sentence-level的向量，下游具体NLP任务调用更方便，修正了ELMo模型的潜在问题。

**真正的双向encoding**

Masked LM，类似完形填空，尽管仍旧看到所有位置信息，但需要预测的词已被特殊符号代替，可以放心双向encoding。

**Transformer做encoder实现上下文相关（context）**

使用transformer而不是bi-LSTM做encoder，可以有更深的层数、具有更好并行性。并且线性的Transformer比lstm更易免受mask标记影响，只需要通过self-attention减小mask标记权重即可，而lstm类似黑盒模型，很难确定其内部对于mask标记的处理方式。

**提升至句子级别**

学习句子/句对关系表示，句子级负采样。首先给定的一个句子，下一句子正例（正确词），随机采样一句负例（随机采样词）,句子级上来做二分类（即判断句子是当前句子的下一句还是噪声），类似word2vec的单词级负采样。

**Masked Language Model**

随机mask语料中15%的token，然后将masked token 位置输出的最终隐层向量送入softmax，来预测masked token。这样输入一个句子，每次只预测句子中大概15%的词，所以BERT训练很慢。。。

而对于盖住词的特殊标记，在下游NLP任务中不存在。因此，为了和后续任务保持一致，作者按一定的比例在需要预测的词位置上输入原词或者输入某个随机的词。如：my dog is hairy

有80%的概率用“[mask]”标记来替换——my dog is [MASK]

有10%的概率用随机采样的一个单词来替换——my dog is apple

有10%的概率不做替换——my dog is hairy

**Transformer —— attention is all you need**

它是由编码组件、解码组件和它们之间的连接组成。编码组件部分由一堆编码器（encoder）构成，解码组件部分也是由相同数量（与编码器对应）的解码器（decoder）组成的。所有的编码器在结构上都是相同的，但它们没有共享参数。每个解码器都可以分解成两个子层。

从编码器输入的句子首先会经过一个自注意力（self-attention）层，这层帮助编码器在对每个单词编码时关注输入句子的其他单词。自注意力层的输出会传递到前馈（feed-forward）神经网络中。每个位置的单词对应的前馈神经网络都完全一样（译注：另一种解读就是一层窗口为一个单词的一维卷积神经网络）。

解码器中也有编码器的自注意力（self-attention）层和前馈（feed-forward）层。除此之外，这两个层之间还有一个注意力层，用来关注输入句子的相关部分。

像大部分NLP应用一样，我们首先将每个输入单词通过词嵌入算法转换为词向量。每个单词都被嵌入为512维的向量。词嵌入过程只发生在最底层的编码器中。

接下来我们看看Transformer的一个核心特性，在这里输入序列中每个位置的单词都有自己独特的路径流入编码器。在自注意力层中，这些路径之间存在依赖关系。而前馈（feed-forward）层没有这些依赖关系。因此在前馈（feed-forward）层时可以并行执行各种路径。

**自注意力机制**

当模型处理这个单词“it”的时候，自注意力机制会允许“it”与“animal”建立联系。当我们在编码器#5（栈中最上层编码器）中编码“it”这个单词的时，注意力机制的部分会去关注“The Animal”，将它的表示的一部分编入“it”的编码中。

计算自注意力的第一步就是从每个编码器的输入向量（每个单词的词向量）中生成三个向量。也就是说对于每个单词，我们创造一个查询向量、一个键向量和一个值向量。这三个向量是通过词嵌入与三个权重矩阵后相乘创建的。

“多头”注意力机制（“multi-headed” attention）

它给出了注意力层的多个“表示子空间”（representation subspaces）。对于“多头”注意机制，我们有多个查询/键/值权重矩阵集(Transformer使用八个注意力头，因此我们对于每个编码器/解码器有八个矩阵集合)。这些集合中的每一个都是随机初始化的，在训练之后，每个集合都被用来将输入词嵌入(或来自较低编码器/解码器的向量)投影到不同的表示子空间中。

当我们编码“it”一词时，一个注意力头集中在“animal”上，而另一个则集中在“tired”上，从某种意义上说，模型对“it”一词的表达在某种程度上是“animal”和“tired”的代表。

multi-head attention:

　　将一个词的vector切分成h个维度，求attention相似度时每个h维度计算。由于单词映射在高维空间作为向量形式，每一维空间都可以学到不同的特征，相邻空间所学结果更相似，相较于全体空间放到一起对应更加合理。比如对于vector-size=512的词向量，取h=8，每64个空间做一个attention，学到结果更细化。

self-attention：

每个词位的词都可以无视方向和距离，有机会直接和句子中的每个词encoding。每个单词和同句其他单词之间都有一条边作为联系，边的颜色越深表明联系越强。利用self-attention机制实现快速并行。

position encoding:

　　因为transformer既没有RNN的recurrence也没有CNN的convolution，但序列顺序信息很重要，比如你欠我100万明天要还和我欠你100万明天要还的含义截然不同。。。

　　但BERT直接训练一个position embedding来保留位置信息，每个位置随机初始化一个向量，加入模型训练，最后就得到一个包含位置信息的embedding，最后这个position embedding和word embedding的结合方式上，BERT选择直接拼接。

**sentence-level representation**

在很多任务中，仅仅靠encoding是不足以完成任务的（这个只是学到了一堆token级的特征）

句子级负采样

句子级别的连续性预测任务，即预测输入BERT的两端文本是否为连续的文本。

句子级表示

BERT是一个句子级别的语言模型，不像ELMo模型在与下游具体NLP任务拼接时需要每层加上权重做全局池化，BERT可以直接获得一整个句子的唯一向量表示。它在每个input前面加一个特殊的记号[CLS]，然后让Transformer对[CLS]进行深度encoding，由于Transformer是可以无视空间和距离的把全局信息encoding进每个位置的，而[CLS]的最高隐层作为句子/句对的表示直接跟softmax的输出层连接，因此其作为梯度反向传播路径上的“关卡”，可以学到整个input的上层特征。

**segment embedding**

对于句对来说，EA和EB分别代表左句子和右句子；对于句子来说，只有EA。这个EA和EB也是随模型训练出来的。