bert细节

1.背景结构

1.1基础知识

BERT（Bidirectional Encoder Representations from Transformers）是谷歌提出，作为一个Word2Vec的替代者，其在NLP领域的11个方向大幅刷新了精度，可以说是近年来自残差网络最优突破性的一项技术了。论文的主要特点以下几点：

1. **使用了双向Transformer作为算法的主要框架**，之前的模型是从左向右输入一个文本序列，或者将left-to-right 和 right-to-left 的训练结合起来，实验的结果表明，双向训练的语言模型对语境的理解会比单向的语言模型更深刻；
2. 使用了Mask Language Model(**MLM**) 和 Next Sentence Prediction(**NSP**) 的多任务训练目标；
3. 使用更强大的机器训练**更大规模的数据**，使BERT的结果达到了全新的高度，并且Google开源了BERT模型，用户可以直接使用BERT作为Word2Vec的转换矩阵并高效的将其应用到自己的任务中。

BERT **只利用了 Transformer 的 encoder 部分。**因为BERT 的目标是生成语言模型，所以只需要 encoder 机制。

1.2 BERT与其他模型相比

- **RNN/LSTM**：与传统的 **RNN/LSTM** 相比，BERT 能够实现并行化计算，不仅能够同时捕捉词在句子中的上下文关系，还能在多个层次上提取词语之间的语义关联，从而更全面地反映句子的整体语义。
- **word2vec**：相较于 **Word2Vec** 这类静态词向量模型，BERT 的优势在于可以根据具体的上下文动态生成词向量，有效解决了词义歧义的问题。例如，“苹果”在不同语境下可以指水果或公司，BERT 能根据上下文自动判断其具体含义。
- **ELMO**：与 **ELMo** 相比，BERT 是真正意义上的双向语言模型。ELMo 虽然通过拼接从左到右和从右到左的两个 LSTM 表示实现了某种程度上的“双向”建模，但本质上仍是浅层融合，并未充分建模上下文之间的深层交互。此外，ELMo 使用 LSTM 结构导致模型参数较多、体积较大，在小规模数据集上训练时容易出现过拟合。

其次bert在多方面的nlp任务表现来看效果都较好，具备较强的泛化能力，对于特定的任务只需要添加一个输出层来进行fine-tuning即可

1.3 BERT，GPT，ELMo

BERT, GPT, ELMo之间的不同点

关于特征提取器:

- `ELMo`采用两部分**双层双向LSTM**进行特征提取，然后再进行特征拼接来融合语义信息。
- `GPT`和`BERT`采用**Transformer**进行特征提取。BERT采用的是Transformer架构中的Encoder模块；GPT采用的是Transformer架构中的Decoder模块.
- 很多NLP任务表明Transformer的特征提取能力强于LSTM, 对于ELMo而言, 采用1层静态token embedding + 2层LSTM，LSTM 在长距离依赖建模上不如 Transformer 强，且无法并行化，训练效率低，所以提取特征的能力有限。

单/双向语言模型:

- 三者之中, 只有 `GPT` 采用单向语言模型, 而 `ELMo` 和 `BERT` 都采用双向语言模型.
- ELMo虽然被认为采用了双向语言模型，但实际上是左右两个单向语言模型分别提取特征，然后进行特征拼接， 这种融合特征的能力比BERT一体化的融合特征方式弱。
- 三者之中, 只有ELMo没有采用Transformer。GPT和BERT都源于Transformer架构，**GPT的单向语言模型采用了经过修改后的Decoder模块**，Decoder采用了look-ahead mask，只能看到context before上文信息，未来的信息都被mask掉了。**而BERT的双向语言模型采用了Encoder模块**，Encoder只采用了padding mask，可以同时看到context before上文信息, 以及context after下文信息。

BERT, GPT, ELMo各自的优点和缺点

ELMo

- **优点**：从早期的Word2Vec预训练模型的最大缺点出发, 进行改进, 这一缺点就是无法解决多义词的问题。ELMo 通过上下文相关的词向量有效解决了多义词问题，使同一个词在不同语境下能生成不同的嵌入表示。它基于双向 LSTM 建模上下文信息，并以特征插件的形式灵活应用于各类下游任务，无需微调。

> **Note**
>
> ELMo 能看到整个句子的信息，但它并不是像 BERT 那样通过自注意力机制全局建模，而是通过 LSTM 的序列结构逐步建模上下文关系。

- **缺点**：ELMo使用**LSTM提取特征的能力弱于Transformer**；ELMo使用向量拼接的方式融合上下文特征的能力弱于Transformer.

GPT

- **优点**：GPT使用了Transformer提取特征, 使得模型能力大幅提升.
- **缺点**：GPT只使用了单向Decoder，无法融合未来的信息.

BERT

- **优点**：BERT使用了双向Transformer提取特征，使得模型能力大幅提升；添加了两个预训练任务, MLM + NSP的多任务方式进行模型预训练.
- **缺点**：**模型过于庞大, 参数量太多**, 需要的数据和算力要求过高, 训练好的模型应用场景要求高；更适合用于语言嵌入表达，语言理解方面的任务，**不适合用于生成式的任务。**

1.4 与Transformer区别

1.BERT 是基于 Transformer 架构改进而来的模型，但它**只使用了 Transformer 的 Encoder 部分** ，并未使用 Decoder。

2.与Transformer本身的Encoder端相比，BERT的Transformer Encoder端输入的向量表示增加了Segment Embeddings。Segment Embedding 是 BERT 中用来区分不同句子的一种嵌入表示，通过将“句子身份信息”加入输入中，使模型能够有效处理句子对任务。

参数规模对比

模型类型	层数（L）	隐藏层维度（H）	注意力头数（A）	总参数量	GPU 显存占用
BERT Base	12	768	12	~1.1 亿	约 7GB+
BERT Large	24	1024	16	~3.4 亿	约 32GB+

模型类型	层数（L）	隐藏层维度（H）	注意力头数（A）	总参数量	GPU 显存占用
标准 Transformer	Encoder 6 层 Decoder 6 层	512	8	——	

BERT 是基于 Transformer 架构改进而来的预训练语言模型，它只使用了 Transformer 的 Encoder 部分，并没有包含 Decoder。在具体结构上，BERT Base 版本由 12 个堆叠的 Transformer Encoder Block 组成（即总共有 12 层），**每一层都是一个完整的 Encoder Block，而不是在一个 Block 内部再叠加多个 Encoder 层。**这些 Encoder Block 依次对输入序列进行多头自注意力计算和前馈神经网络处理，从而逐层提取上下文语义信息。

1.5 word2vec到BERT改进了什么

word2vec到BERT的改进之处其实没有很明确的答案，BERT的思想其实很大程度上来源于CBOW模型如果。CBOW 是通过上下文预测中心词，MLM 是通过遮盖部分词，让模型根据上下文预测被遮盖的词，本质上是一个“上下文预测词”的任务。从准确率上说改进的话，BERT利用更深的模型，以及海量的语料，得到的embedding表示，来做下游任务时的准确率是要比word2vec高不少的。实际上，这也离不开模型的“加码”以及数据的“巨大加码”。再从方法的意义角度来说，**BERT的重要意义在于给大量的NLP任务提供了一个泛化能力很强的预训练模型，而仅仅使用word2vec产生的词向量表示，不仅能够完成的任务比BERT少了很多，而且很多时候直接利用word2vec产生的词向量表示给下游任务提供信息，下游任务的表现不一定会很好，甚至会比较差。**

2.模型结构

2.1两个任务

（1）Masked LM (MLM)

在将单词序列输入给 BERT 之前，每个序列中有 15% 的单词被 `[MASK]` token 替换。 然后模型尝试基于序列中其他未被 mask 的单词的上下文来预测被掩盖的原单词。在BERT的实验中，15%的WordPiece Token会被随机Mask掉。**在训练模型时，一个句子会被多次喂到模型中用于参数学习，在每次训练迭代中，都对输入句子重新进行随机 Mask，也就是说，同一个句子在不同训练轮次中被 mask 的词可能是不同的。但是Google并没有在每次都mask掉这些单词，而是在确定要Mask掉的单词之后，80%的概率会直接替换为 `[Mask]`，10%的概率将其替换为其它任意单词，10%的概率会保留原始Token。**

1. **80% 的 tokens 会被替换为 [MASK] token**：**是 Masked LM 中的主要部分，可以在不泄露 label 的情况下融合真双向语义信息**；
2. **10% 的 tokens 会称替换为随机的 token** ： 因为需要在最后一层随机替换的这个 token 位去预测它真实的词，而模型并不知道这个 token 位是被随机替换的，就迫使模型尽量在每一个词上都学习到一个 全局语境下的表征，因而也能够让 BERT 获得更好的语境相关的词向量（**这正是解决一词多义的最重要特性**）；
3. **10% 的 tokens 会保持不变但需要被预测** ： 这样能够给模型一定的 bias ， 相当于是额外的奖励，将模型对于词的表征能够拉向词的 真实表征；

（2）Next Sentence Prediction (NSP)

BERT 在预训练阶段使用句子对进行 NSP（Next Sentence Prediction）任务的学习，50% 的样本是真实连续句子，50% 是随机拼接的。输入中 `[CLS]` 用于分类任务的聚合表示，`[SEP]` 用于标识句子边界，这种设计使 BERT 更好地建模句子级语义关系。

在 BERT 的训练过程中，模型接收成对的句子作为输入，并且预测其中第二个句子是否在原始文档中也是后续句子。

1. 在训练期间，50% 的输入对在原始文档中是前后关系，另外 50% 中是从语料库中随机组成的，并且是与第一句断开的。
2. 在第一个句子的开头插入 `[CLS]` 标记，表示该特征用于分类模型，对非分类模型，该符号可以省去，在每个句子的末尾插入 `[SEP]` 标记，表示分句符号，用于断开输入语料中的两个句子。

2.2 Embedding

BERT的输入的编码向量（长度是512）是3个嵌入特征的单位和，这三个词嵌入特征是：

1. Token Embedding（词元嵌入 / WordPiece 嵌入）

Token Embedding 是 BERT 输入中最基础的部分，它表示每个输入词或子词的语义信息。BERT 使用 **WordPiece 分词技术**，将文本切分为有意义的子词单元（如“playing” → “play” + “ing”），从而在保留常见词完整性的同时，也能有效处理未登录词和罕见词。每个词元都会被映射为一个固定维度的向量，作为模型理解语言的基础。

2. Segment Embedding（句子分割嵌入）

Segment Embedding 用于区分输入中的不同句子，帮助模型识别哪些词属于第一个句子，哪些属于第二个句子。例如，在问答任务或句子对分类任务中，**BERT 接收两个句子作为输入，Segment Embedding 会分别为它们分配标识符（0 和 1）。如果属于第一个句子（A句），则加上 Segment ID 为 0 的向量。如果属于第二个句子（B句），则加上 Segment ID 为 1 的向量。**通过这种方式，模型能够理解句子之间的关系，并支持诸如自然语言推理、文本蕴含等需要句子间建模的任务。

3. Position Embedding（位置嵌入）

Position Embedding 的作用是为模型引入序列的位置信息。由于 BERT 基于 Transformer 架构，而 Transformer 本身不具有顺序感知能力，因此必须通过位置嵌入来告诉模型每个词在句子中的位置。这些位置向量是可学习的，长度与最大序列长度一致（通常为 512），并与 Token Embedding 和 Segment Embedding 相加，使模型能够感知词语的顺序和结构信息。

> **Note**
>
> Token Embedding是分词器映射得来的，而Segment Embedding 和 Position Embedding 都是可学习的参数矩阵，它们在训练过程中被不断优化，训练完成后就成为模型的一部分，在推理和部署阶段直接使用即可，无需重新训练。

3.模型细节

3.1 BERT在第一句前会加一个[CLS]标志

BERT在第一句前会加一个[CLS]标志，最后一层该位**对应向量可以作为整句话的语义表示**，从而用于下游的分类任务等。

3.2 BERT的三个Embedding直接相加会对语义有影响吗

BERT的三个Embedding相加，**本质可以看作一个特征的融合**，强大如 BERT 应该可以学到融合后特征的语义信息的。

> **Note**
>
> Embedding的本质：**Embedding层就是以one hot为输入、中间层节点为字向量维数的全连接层！而这个全连接层的参数，就是一个“字向量表”**！

Embedding 层的本质：**就是一个查表工具，背后对应着一个“字向量表”** 。你可以把它想象成一本词典，每个词（或字、子词）都对应一个向量。当你输入一个词的时候，模型就根据这个词的编号去这个“词典”里查出对应的向量，这就是所谓的 Embedding 操作。

从数学上看，Embedding 层等价于一个以 one-hot 向量为输入、输出为词向量的全连接层。举个例子，如果你有一个词汇表大小是 10,000，想把每个词映射成 768 维的词向量，那这个 Embedding 层本质上就是一个 `10000 × 768` 的矩阵。每个词的 one-hot 向量乘上这个矩阵，就会得到一个 768 维的词向量。

但问题来了：one-hot 向量几乎全是 0，只有一个位置是 1。所以这种乘法运算实际上就是在矩阵中取出某一行，也就是我们常说的“查表”。既然如此，就没有必要真的去做矩阵乘法了，直接根据词的编号去查找对应的行就可以了。这样一来，计算速度大大提升，内存占用也减少了，这就是为什么我们会用“查表”来代替矩阵相乘的原因。

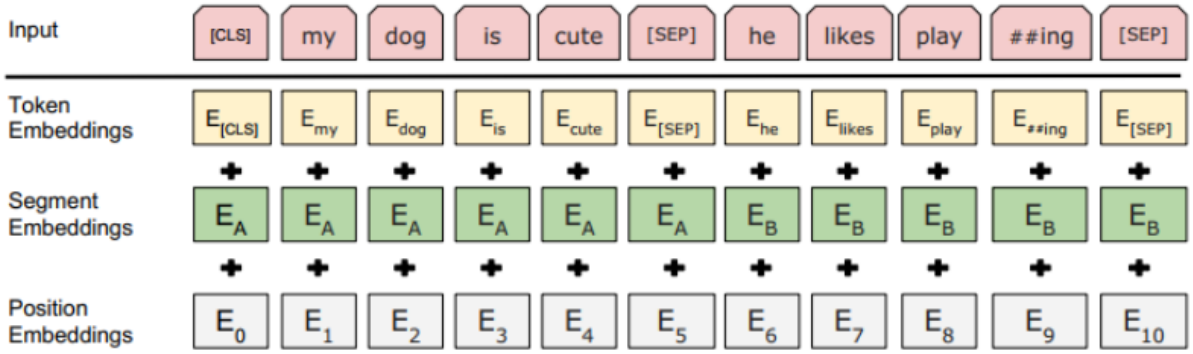
从运算上来看，one hot型的矩阵相乘，就像是相当于查表，于是它直接用查表作为操作，而不写成矩阵再运算，这大大降低了运算量。**再次强调，降低了运算量不是因为词向量的出现，而是因为把one hot型的矩阵运算简化为了查表操作。**

在这里想用一个例子再尝试解释一下：

- 假设 token Embedding 矩阵维度是 `[4,768]`；position Embedding 矩阵维度是 `[3,768]`；segment Embedding 矩阵维度是 `[2,768]`。
- 对于一个字，假设它的 token one-hot 是 `[1,0,0,0]`；它的 position one-hot 是 `[1,0,0]`；它的 segment one-hot 是 `[1,0]`。
- 那这个字最后的 word Embedding，就是上面三种 Embedding 的加和。
- 如此得到的 word Embedding，和concat后的特征：`[1,0,0,0,1,0,0,1,0]`，再过维度为 `[4+3+2,768] = [9, 768]` 的全连接层，得到的向量其实就是一样的。

3.3 使用BERT预训练模型为什么最多只能输入512个词?

BERT 的输入由三部分组成：Token Embedding 表示词的语义信息，Position Embedding 表示词的位置信息，Segment Embedding 区分句子身份。**由于 Position Embedding 的最大长度被设计为 512，所以 BERT 最多只能处理 512 个词的输入**；Segment Embedding 则用于区分两个句子，帮助模型理解句子间的语义关系。



BERT输入：

- `token embedding`：词向量表示，该向量既可以随机初始化，也可以利用Word2Vector等算法进行预训练以作为初始值，使用WordPiece tokenization让BERT在处理英文文本的时候仅需要存储30,522 个词，而且很少遇到oov的词，token embedding是必须的；
- `position embedding`：和Transformer的sin、cos函数编码不同，直接去训练了一个position embedding。给每个位置词一个随机初始化的词向量，再训练；
- `segment embedding`：该向量的取值在模型训练过程中自动学习，用于刻画文本的全局语义信息，并与单字/词的语义信息相融合。

输出是文本中各个字/词融合了全文语义信息后的向量表示。

3.4 BERT如何区分一词多义?

在传统的词向量模型（如 Word2Vec 或 GloVe）中，每个词都有一个固定的向量表示。也就是说，无论这个词出现在什么语境下，它的词向量都是一样的。这种"静态"的方式无法很好地处理**一词多义** 的问题。例如，"苹果"既可以指水果，也可以指公司，但在这些模型中，它只有一个向量。

而 **BERT 解决这个问题的核心机制是：上下文相关（contextualized）的词向量表示** 。

具体来说：

- 在输入阶段，同一个字或词虽然会被映射为相同的 token ID，对应的初始 Embedding 向量也是一样的；
- 但随着输入进入 BERT 的多层 Transformer Encoder，模型会通过 **自注意力机制（Self-Attention）** 去关注这个词在当前句子中的上下文信息；
- 不同的上下文会导致不同的 attention 权重分布，从而使得模型对这个词的理解发生变化；
- 经过多层编码后，最终输出的词向量就会根据其所在语境产生不同的表示。

3.5 BERT中Normalization结构：LayerNorm

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**，Batch Normalization 是对这批样本的同一维度特征做归一化
- **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。Layer Normalization 是对这单个样本的所有维度特征做归一化。

BatchNorm的缺点：

1. 需要较大的batch以体现整体数据分布
2. 训练阶段需要保存每个batch的均值和方差，以求出整体均值和方差在推理阶段使用
3. 不适用于可变长序列的训练，如RNN

Layer Normalization：一个独立于batch size的算法，所以无论一个batch样本数多少都不会影响参与LN计算的数据量，从而解决BN的两个问题。LN的做法是根据样本的特征数做归一化。Layer Normalization不依赖于batch的大小和输入sequence的深度，因此可以用于batch-size为1和RNN中对边长的输入sequence的normalize操作。但在大批量的样本训练时，效果没BN好。

实践证明，LN用于RNN进行Normalization时，取得了比BN更好的效果。但用于CNN时，效果并不如BN明显。

3.6 为什么说ELMO是伪双向，BERT是真双向?

> **Note**
>
> ELMo 虽然号称“双向”，但本质上只是两个单向模型的拼接，缺乏真正的上下文交互机制，因此被称为“伪双向”；而 BERT 利用自注意力机制，在训练中同时考虑词的前后上下文，实现了“真双向”建模，大幅提升了模型的表达能力和泛化性能。

- ELMo 使用两个方向的 LSTM（从左到右和从右到左）分别建模上下文信息。最终词向量是这两个方向输出的拼接或加权，并未实现真正的上下文交互。因此，它只是形式上的“双向”，被称为“伪双向”。同时缺点也是显而易见的，模型参数太多，而且模型太大，少量数据训练时，容易过拟合。
- BERT 通过 Masked Language Model（MLM）任务，利用整个句子的上下文预测被遮盖的词。模型在预测一个词时，能同时看到它左右两边的信息，实现了真正意义上的双向建模。结合 Transformer 的自注意力机制，BERT 能动态融合上下文语义，被称为“真双向”。

3.7 BERT和Transformer Encoder的差异有哪些?

与原始 Transformer 的 Encoder 相比，BERT 在输入中加入了 `Segment Embeddings`。这一设计是为了处理涉及句对的任务，如句对分类、问答等，帮助模型区分不同的句子来源。

由于两个句子之间存在顺序和语义关系，如果不做区分，模型可能会混淆句子顺序，导致语义错误（例如“武松打虎”和“虎打武松”被误认为是同一个意思）。因此，BERT 引入 `Segment Embeddings` 来标识每个词所属的句子，从而更好地建模句间关系。

3.8 Scaled Dot Product:为什么是缩放点积，而不是点积模型?

当**输入信息的维度 d 比较高，点积模型的值通常有比较大方差**，从而**导致 softmax函数的梯度会比较小**。因此，缩放点积模型可以较好地解决这一问题。

常用的Attention机制为加性模型和点积模型，理论上加性模型和点积模型的复杂度差不多，但是**点积模型在实现上可以更好地利用矩阵乘积，从而计算效率更高**（实际上，随着维度d的增大，加性模型会明显好于点积模型）。

3.9 FFN的作用?

FFN（前馈神经网络）在 Transformer 结构中起到了**增强模型非线性表达能力和特征提取能力的关键作用** 。它由两个全连接层组成，中间通过 ReLU（或 GELU）激活函数引入非线性变换，使模型能够更好地捕捉复杂的语义特征。

3.10 BERT非线性的来源

BERT 模型的非线性主要来源于以下两个部分：

1.前馈层（Feed-Forward Network）中的 GeLU 激活函数。GeLU 激活函数在引入非线性的同时，也带来了随机正则的思想。它是根据当前输入大于其他输入的概率来进行一种类似“mask”的随机正则化操作。

2.Self-Attention 的非线性 来自于 softmax 函数的应用，它本身是一个非线性操作

> **Note**
>
> 具体来说，GeLU 是基于输入值在整个分布中的位置来决定其是否被保留或抑制的：
>
> $$\text{Bernoulli}(\phi(x))，\text{其中，}\phi(x) = P(X \leq x)$$

> 这里的 $\phi(x)$ 表示标准正态分布下输入值 x 的累积分布概率。也就是说，输入值越小，它被“抑制”（即输出为0）的概率越高，这符合伯努利分布的思想。
>
> 相比 ReLU：
>
> - **ReLU 缺乏这种随机性**，它的行为是确定性的：输入大于 0 就保留（1），否则置零（0）；
> - 而 GeLU 则更具概率性和鲁棒性，使模型在训练过程中具有一定的正则化效果。

3.11 MLM任务，对于在数据中随机选择 15% 的标记，其中80%被换位[mask]，10%不变、10%随机替换其他单词，原因是什么?

典型的Denosing Autoencoder的思路，**那些被Mask掉的单词就是在输入侧加入的所谓噪音**。类似BERT这种预训练模式，被称为DAE LM。因此总结来说BERT模型 `[Mask]` 标记就是**引入噪音**的手段。预测一个词汇时，模型并不知道输入对应位置的词汇是否为正确的词汇（ 10%概率），这就迫使模型更多地依赖于上下文信息去预测词汇，并且赋予了模型一定的纠错能力。

两个缺点：

1. 因为Bert用于下游任务微调时， `[MASK]` 标记不会出现，它只出现在预训练任务中。这就造成了预训练和微调之间的不匹配，微调不出现 `[MASK]` 这个标记，模型好像就没有了着力点、不知从哪入手。所以只将80%的替换为 `[mask]`，但这也只是缓解、不能解决。
2. 相较于传统语言模型，Bert的每批次训练数据中只有 15% 的标记被预测，这导致模型需要更多的训练步骤来收敛。

3.12其mask相对于CBOW有什么异同点?

相同点：

- CBOW的核心思想是：给定上下文，根据它的上文 Context-Before 和下文 Context-after 去预测 input word。
- 而BERT本质上也是这么做的，但是BERT的做法是给定一个句子，会随机Mask 15%的词，然后让BERT来预测这些Mask的词。

不同点：

1. **训练方式的不同**

   在 CBOW 模型中，每个单词都会作为目标词（input word）进行预测，也就是说，每个词都会成为训练样本中的“被预测词”。而 BERT 并没有采用这种方式，因为如果对每一个词都进行完整的上下文建模和预测，会导致训练数据量极大、训练时间非常长。因此，BERT 采用了“随机遮盖”策略（Masked Language Model），每次只预测一小部分被遮盖的词（通常是 15% 的 token），从而降低计算成本。

2. **输入数据的形式不同**

   CBOW 的输入是“局部”的：对于每一个要预测的目标词，CBOW 只使用其上下文中一定窗口内的词作为输入；
   BERT 的输入是“完整”的句子：在输入时保留整个句子的信息，并通过将部分词语替换为 `[MASK]` 标记来告诉模型“你要预测这些位置上的词”。所以可以说，**BERT 是在一个更完整、更丰富的上下文环境中进行建模的 。**

3. **词向量表示的能力不同**

CBOW 训练完成后，每个词都会对应一个固定的词向量（word embedding），无论这个词出现在什么语境下，它的向量表示都不会改变。这导致 CBOW 很难处理一词多义的问题（如“苹果”在水果和公司两个语境下的含义完全不同）。而 BERT 输出的词向量（token embedding）是上下文相关的（contextualized），它会根据当前句子的整体语义动态调整表示。所以即使是同一个词，在不同的上下文中也会得到不同的词向量，从而更好地解决一词多义问题。

3.13 对于长度较长的语料，如何训练?

对于长文本的处理，BERT 通常采用两种方式：**截断** 和 **切分** 。由于 BERT 的最大输入长度限制为 512 个 token，其中 `[CLS]` 占据 1 个位置，`[SEP]` 至少占用 1 个位置，因此留给实际文本内容的最大长度为 **510 个 token** 。

- 截断：一般来说文本中最重要的信息是开始和结尾，因此文中对于长文本做了截断处理。
    1. head-only：保留前510个字符
    2. tail-only：保留后510个字符
    3. head+tail：保留前128个和后382个字符
- 切分: 将文本分成k段，每段的输入和Bert常规输入相同，第一个字符是[CLS]表示这段的加权信息。文中使用了Max-pooling, Average pooling和self-attention结合这些片段的表示。

4. BERT 损失函数

BERT 的损失函数由两部分组成： 第一部分是来自 **Masked Language Model（MLM）** 的单词级别分类任务； 第二部分是 **Next Sentence Prediction（NSP）** 的句子级别分类任务。

优点：

通过这两个任务的联合学习，可以让 BERT 学习到既包含 token 级别的语义信息，也包含句子级别的语义信息。
损失函数形式如下：

$$L(\theta, \theta_1, \theta_2) = L_1(\theta, \theta_1) + L_2(\theta, \theta_2)$$

其中：

- θ：BERT 中 Encoder 部分的参数（也就是整个 Transformer 的主干网络参数）；
- θ_1：是 Mask-LM 任务中在 Encoder 上所接的输出层中的参数（用于预测被 mask 掉的词）；
- θ_2：是句子预测任务中在 Encoder 上接的分类器参数（用于判断两个句子是否连续）。

第一部分：Masked Language Model 损失函数

这个损失函数衡量的是：BERT 在预测被遮盖词时的准确度，预测得越差，损失越大。如果被 mask 的词集合为 M，因为它是一个词典大小 $|V|$ 上的多分类问题，所用的损失函数叫做负对数似然函数（且是最小化，等价于最大化对数似然函数），那么具体说来有：

$$L_1(\theta, \theta_1) = -\sum_{i=1}^{M} \log p(m = m_i \mid \theta, \theta_1), \quad m_i \in [1, 2, \ldots, |V|]$$

第二部分：Next Sentence Prediction 损失函数

这个损失函数衡量的是：BERT 判断两个句子是否为连续句子的准确性，判断错误越多，损失越大。在句子预测任务中，也是一个分类问题的损失函数：

$$L_2(\theta, \theta_2) = -\sum_{j=1}^{N} \log p(n = n_i \mid \theta, \theta_2), \quad n_i \in [\mathrm{IsNext}, \mathrm{NotNext}]$$

5.模型优缺点和局限性

5.1 BERT优点

1. Transformer Encoder因为有Self-attention机制，因此BERT自带双向功能
2. 计算可并行化
3. 微调成本小
4. 因为双向功能以及多层Self-attention机制的影响，使得BERT必须使用Cloze版（完形填空版本）的语言模型Masked-LM来完成token级别的预训练
5. 为了获取比词更高级别的句子级别的语义表征，BERT加入了Next Sentence Prediction来和Masked-LM一起做联合训练
6. 为了适配多任务下的迁移学习，BERT设计了更通用的输入层和输出层

5.2 BERT缺点

1. `[MASK]`标记在实际预测中不会出现，训练时用过多`[MASK]`影响模型表现
2. 每个batch只有15%的token被预测，所以BERT收敛得比left-to-right模型要慢（它们会预测每个token）
3. task1的随机遮挡策略略显粗犷，推荐阅读《Data Nosing As Smoothing In Neural Network Language Models》
4. BERT对硬件资源的消耗巨大（大模型需要16个tpu，历时四天；更大的模型需要64个tpu，历时四天。

5.3 BERT局限性

从XLNet论文中，提到了BERT的两个缺点，分别如下

1. 被**mask掉的单词之间是有关系的**，比如”New York is a city”，”New”和”York”两个词，那么给定”is a city”的条件下”New”和”York”并不独立，因为”New York”是一个实体，看到”New”则后面出现”York”的概率要比看到”Old”后面出现”York”概率要大得多。 但是需要注意的是，这个问题并不是什么大问题，甚至可以说对最后的结果并没有多大的影响，因为本身BERT预训练的语料就是海量的(动辄几十个G)，所以如果训练数据足够大，其实不靠当前这个例子，靠其它例子，也能弥补被Mask单词直接的相互关系问题，因为总有其它例子能够学会这些单词的相互依赖关系。
2. BERT的在预训练时会出现特殊的`[MASK]`，但是它在下游的fine-tune中不会出现，这就出现了**预训练阶段和fine-tune阶段不一致的问题**。其实这个问题对最后结果产生多大的影响也是不够明确的，因为后续有许多BERT相关的预训练模型仍然保持了`[MASK]`标记，也取得了很大的结果，而且很多数据集上的结果也比BERT要好。但是确确实实引入`[MASK]`标记，也是为了构造自编码语言模型而采用的一种折中方式。
3. BERT在分词后做`[MASK]`会产生的一个问题，为了解决OOV的问题，通常会把一个词切分成更细粒度的WordPiece。BERT在Pretraining的时候是随机Mask这些WordPiece的，这就可能出现**只Mask一个词的一部分的情况**，这样它只需要记住一些词(WordPiece的序列)就可以完成这个任务，而不是根据上下文的语义关系来预测出来的。类似的中文的词”模型”也可能被Mask部分(其实用”琵琶”的例子可能更好，因为这两个字只能一起出现而不能单独出现)，这也会让预测变得容易。为了解决这个问题，很自然的想法就是词作为一个整体要么都Mask要么都不Mask，这就是所谓的Whole Word Masking。这是一个很简单的想法，对于BERT的代码修改也非常少，只是修改一些Mask的那段代码。

> **Note**

> BERT 的 MLM 预训练方式虽然强大，但也存在几个局限性：
>
> 1. 被 mask 的词之间缺乏联合建模；
> 2. 引入了 `[MASK]` 导致预训练与微调不一致；
> 3. WordPiece 分词下可能出现“局部 mask”，削弱了任务难度；