2. BERT常见问题★

1．BERT 的基本原理是什么?

BERT 来自 Google 的论文Pre-training of Deep Bidirectional Transformers for Language Understanding，BERT 是“Bidirectional Encoder Representations from Transformers”的首字母缩写，整体是一个自编码语言模型（Autoencoder LM），并且其设计了两个任务来预训练该模型。

第一个任务是采用 MaskLM 的方式来训练语言模型，通俗地说就是在输入一句话的时候，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。

第二个任务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，引入这个任务可以更好地让模型学到连续的文本片段之间的关系。

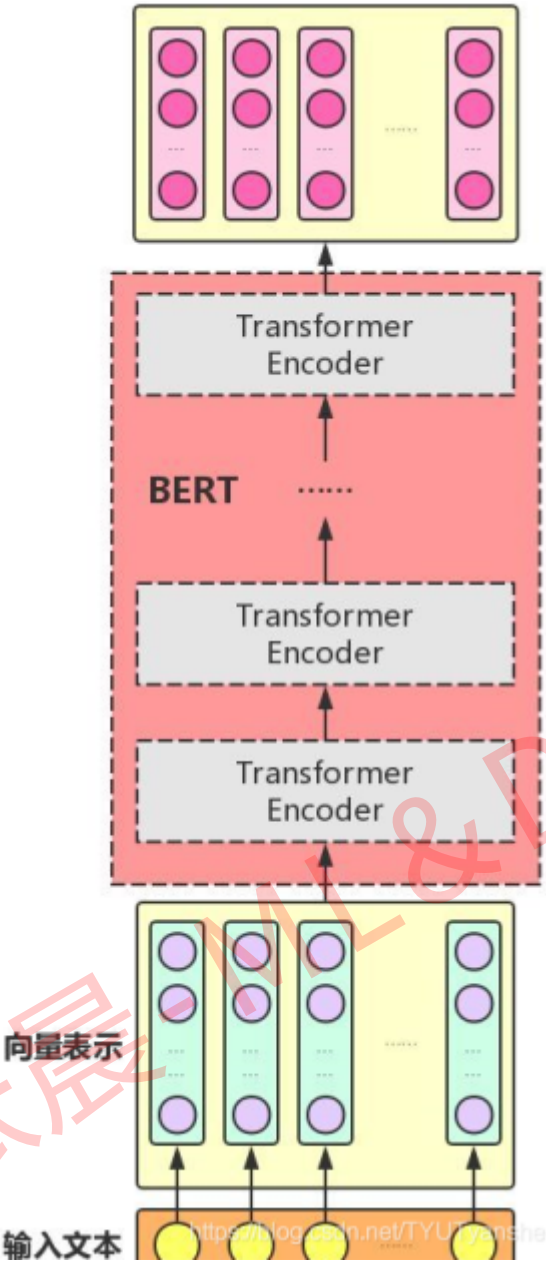
BERT 相较于原来的 RNN、LSTM 可以做到并发执行，同时提取词在句子中的关系特征，并且能在多个不同层次提取关系特征，进而更全面反映句子语义。相较于 word2vec，其又能根据句子上下文获取词义，从而避免歧义出现。同时缺点也是显而易见的，模型参数太多，而且模型太大，少量数据训练时，容易过拟合。

2．BERT 是怎么用 Transformer 的?

BERT 只使用了 Transformer 的 Encoder 模块，原论文中，作者分别用 12 层和 24 层 Transformer Encoder 组装了两套 BERT 模型，分别是：

- $BERT_{BASE}: L=12, H=768, A=12, TotalParameters=110M$
- $BERT_{LARGE}: L=24, H=1024, A=16, TotalParameters=340M$

其中层的数量(即，Transformer Encoder 块的数量)为 L，隐藏层的维度为H，自注意头的个数为A 。在所有例子中，我们将前馈/过滤器(Transformer Encoder 端的feed-forward 层)的维度设置为4H ，即当 H=768时是3072；当H=1024是4096。图示如下：

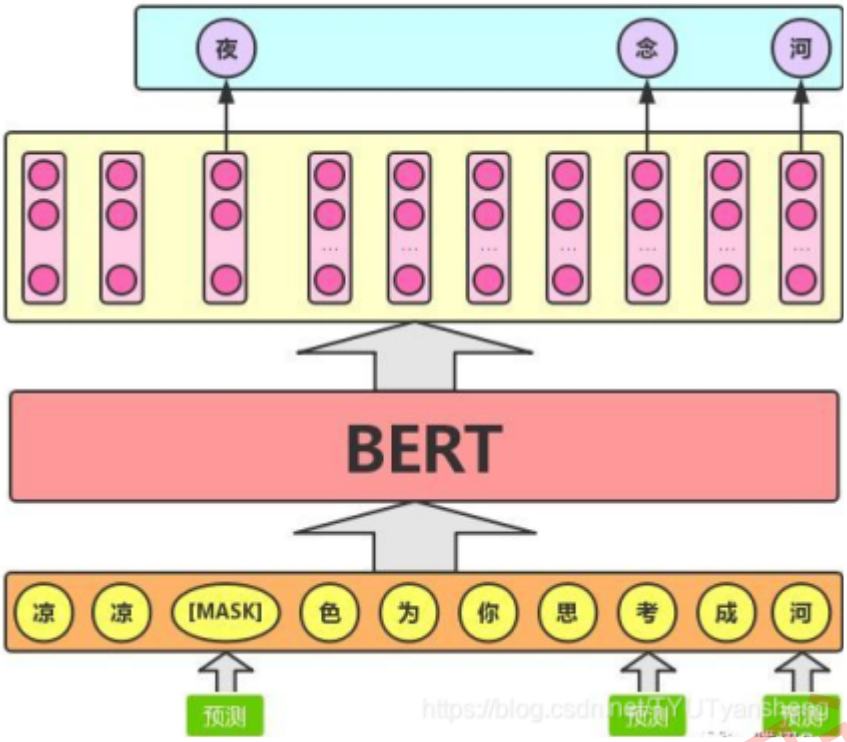


需要注意的是，与 Transformer 本身的 Encoder 端相比，BERT 的 Transformer Encoder 端输入的向量表示，多了 Segment Embeddings(全文信息)。

3．BERT 的训练过程是怎么样的?

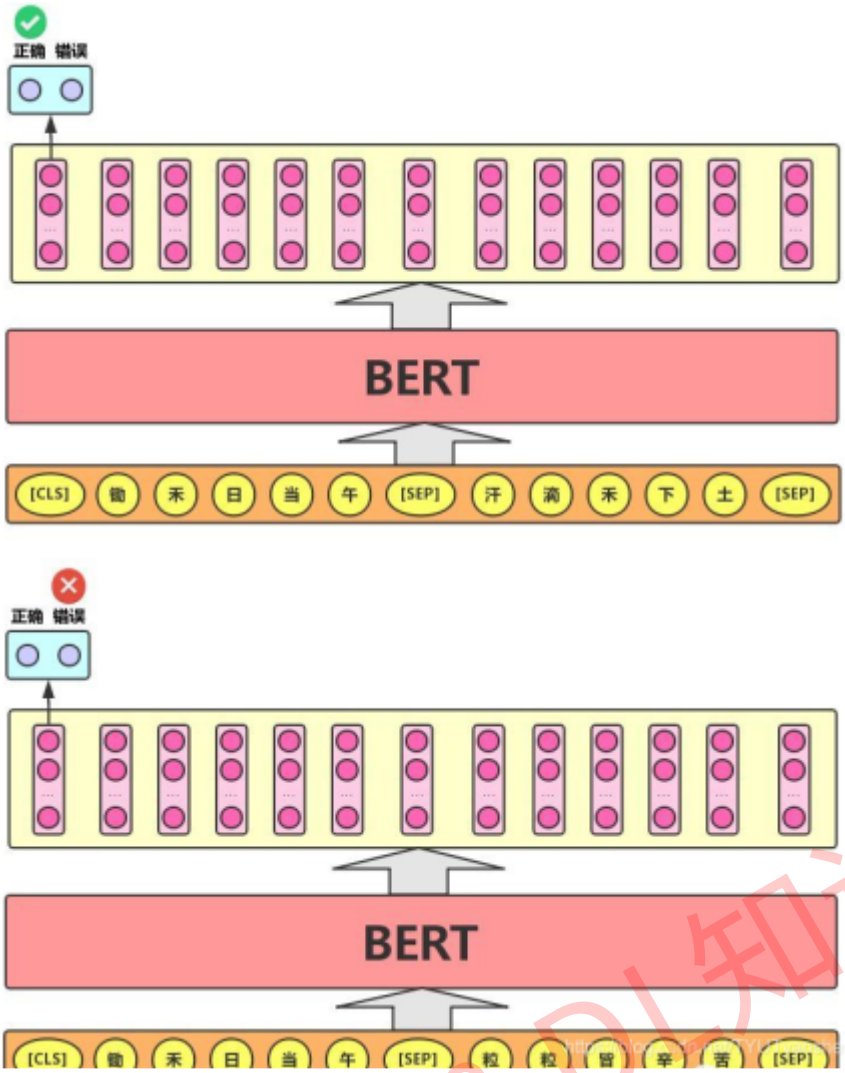
在论文原文中，作者提出了两个预训练任务：Masked LM 和 Next Sentence Prediction。

Masked LM 的任务描述为：给定一句话，随机抹去这句话中的一个或几个词，要求根据剩余词汇预测被抹去的几个词分别是什么，如下图所示。



BERT 模型的这个预训练过程其实就是在模仿我们学语言的过程，思想来源于完形填空的任务。具体来说，文章作者在一句话中随机选择 15% 的词汇用于预测。对于在原句中被抹去的词汇， 80% 情况下采用一个特殊符号 [MASK] 替换， 10% 情况下采用一个任意词替换，剩余 10% 情况下保持原词汇不变。这么做的主要原因是：在后续微调任务中语句中并不会出现 [MASK] 标记，而且这么做的另一个好处是：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。上述提到了这样做的一个缺点，其实这样做还有另外一个缺点，就是每批次数据中只有 15% 的标记被预测，这意味着模型可能需要更多的预训练步骤来收敛。

Next Sentence Prediction 的任务描述为：给定一篇文章中的两句话，判断第二句话在文本中是否紧跟在第一句话之后，如下图所示。



这个类似于段落重排序的任务，即：将一篇文章的各段打乱，让我们通过重新排序把原文还原出来，这其实需要我们对全文大意有充分、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，文章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。

BERT 模型通过对 Masked LM 任务和 Next Sentence Prediction 任务进行联合训练，使模型输出的每个字 / 词的向量表示都能尽可能全面、准确地刻画输入文本（单句或语句对）的整体信息，为后续的微调任务提供更好的模型参数初始值。

4．为什么 BERT 比 ELMo 效果好？ELMo 和 BERT 的区别是什么？

4.1为什么 BERT 比 ELMo 效果好？

从网络结构以及最后的实验效果来看，BERT 比 ELMo 效果好主要集中在以下几点原因：

- LSTM 抽取特征的能力远弱于 Transformer
- 拼接方式双向融合的特征融合能力偏弱(没有具体实验验证，只是推测)
- 其实还有一点，BERT 的训练数据以及模型参数均多余 ELMo，这也是比较重要的一点

4.2 ELMo 和 BERT 的区别是什么?

ELMo 模型是通过语言模型任务得到句子中单词的 embedding 表示，以此作为补充的新特征给下游任务使用。因为 ELMO 给下游提供的是每个单词的特征形式，所以这一类预训练的方法被称为“Feature-based Pre-Training”。而 BERT 模型是“基于 Fine-tuning 的模式”，这种做法和图像领域基于 Fine-tuning 的方式基本一致，下游任务需要将模型改造成 BERT 模型，才可利用 BERT 模型预训练好的参数。

5．BERT 有什么局限性?

从 XLNet 论文中，提到了 BERT 的两个缺点，分别如下：

- BERT 在第一个预训练阶段，假设句子中多个单词被 Mask 掉，这些被 Mask 掉的单词之间没有任何关系，是条件独立的，然而有时候这些单词之间是有关系的，比如”New York is a city”，假设我们 Mask 住”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。
- 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。

- 另外还有一个缺点，是 BERT 在分词后做[MASK]会产生的一个问题，为了解决 OOV 的问题，我们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，例如：

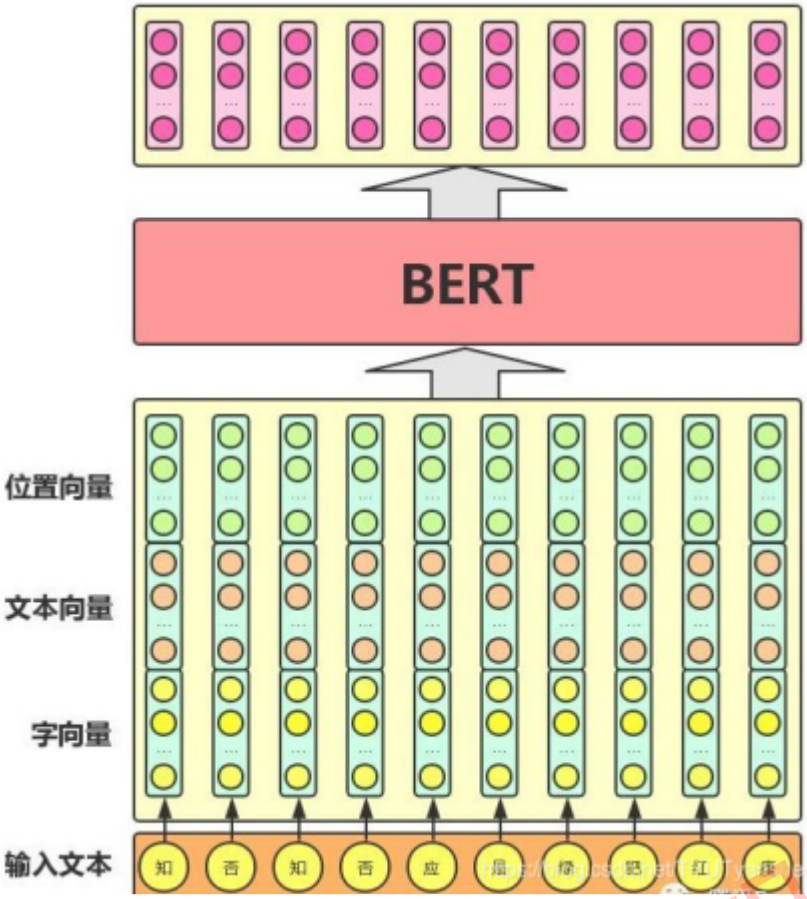
[Original Sentence] 使用语言模型来预测下一个词的probability。 **[Original Sentence with CWS]** 使用 语言 **模型** 来 **预测** 下 一个 词 的 **probability** 。
[Original BERT Input] 使 用 语 言 **[MASK]** 型 来 **[MASK]** 测 下 一 个 词 的 **pro [MASK] ##lity** 。 **[Whold Word Masking Input]** 使 用 语 言 **[MASK] [MASK]** 来 **[MASK] [MASK]** 下 一 个 词 的 **[MASK] [MASK] [MASK]** 。

probability 这个词被切分成”pro”、”#babi”和”#lity”3 个 WordPiece。有可能出现的一种随机 Mask 是把”#babi” Mask 住，但是”pro”和”#lity”没有被 Mask。这样的预测任务就变得容易了，因为在”pro”和”#lity”之间基本上只能是”#babi”了。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。

6．BERT 的输入和输出分别是什么?

BERT 模型的主要输入是文本中各个字/词(或者称为 token)的原始词向量，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；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，如下图所示（为方便描述且与 BERT 模型的当前中文版本保持一致，统一以「字向量」作为输入）：



从上图中可以看出，BERT 模型通过查询字向量表将文本中的每个字转换为一维向量，作为模型输入；模型输出则是输入各字对应的融合全文语义信息后的向量表示。 此外，模型输入除了字向量(英文中对应的是 Token Embeddings)，还包含另外两个部分：

- 文本向量(英文中对应的是 Segment Embeddings)：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
- 位置向量(英文中对应的是 Position Embeddings)：由于出现在文本不同位置的字/词所携带的语义信息存在差异（比如：“我爱你”和“你爱我”），因此，BERT 模型对不同位置的字/词分别附加一个不同的向量以作区分
- 最后，BERT 模型将字向量、文本向量和位置向量的加和作为模型输入。特别地，在目前的 BERT 模型中，文章作者还将英文词汇作进一步切割，划分为更细粒度的语义单位（WordPiece），例如：将 playing 分割为 play 和##ing；此外，对于中文，目前作者未对输入文本进行分词，而是直接将单字作为构成文本的基本单位。

需要注意的是，上图中只是简单介绍了单个句子输入 BERT 模型中的表示，实际上，在做 Next Sentence Prediction 任务时，在第一个句子的首部会加上一个[CLS] token，在两个句子中间以及最后一个句子的尾部会加上一个[SEP] token。

7．针对句子语义相似度/多标签分类/机器翻译/文本生成的任务，利用 BERT 结构怎么做 fine-tuning?

- 针对句子语义相似度的任务：

实际操作时，上述最后一句话之后还会加一个[SEP] token，语义相似度任务将两个句子按照上述方式输入即可，之后与论文中的分类任务一样，将[CLS] token 位置对应的输出，接上 softmax 做分类即可(实际上 GLUE 任务中就有很多语义相似度的数据集)。

- 针对多标签分类的任务：

多标签分类任务，即 MultiLabel，指的是一个样本可能同时属于多个类，即有多个标签。以商品为例，一件 L 尺寸的棉服，则该样本就有至少两个标签——型号：L，类型：冬装。

对于多标签分类任务，显而易见的朴素做法就是不管样本属于几个类，就给它训练几个分类模型即可，然后再一一判断在该类别中，其属于那个子类别，但是这样做未免太暴力了，而多标签分类任务，其实是可以「只用一个模型」来解决的。利用 BERT 模型解决多标签分类问题时，其输入与普通单标签分类问题一致，得到其 embedding 表示之后(也就是 BERT 输出层的 embedding)，有几个 label 就连接到几个全连接层(也可以称为 projection layer)，然后再分别接上 softmax 分类层，这样的话会得到 loss1，loss2, … ,lossn，最后再将所有的 loss 相加起来即可。这种做法就相当于将 n 个分类模型的特征提取层参数共享，得到一个共享的表示(其维度可以视任务而定，由于是多标签分类任务，因此其维度可以适当增大一些)，最后再做多标签分类任务。

- 针对翻译的任务

针对翻译的任务，我自己想到一种做法，因为 BERT 本身会产生 embedding 这样的“副产品”，因此可以直接利用 BERT 输出层得到的 embedding，然后在做机器翻译任务时，将其作为输入/输出的 embedding 表示，这样做的话，可能会遇到 UNK 的问题，为了解决 UNK 的问题，可以将

得到的词向量 embedding 拼接字向量的 embedding 得到输入/输出的表示(对应到英文就是 token embedding 拼接经过 charcnn 的 embedding 的表示)。

8．BERT 应用于有空格丢失或者单词拼写错误等数据是否还是有效？有什么改进的方法？

8.1 BERT 应用于有空格丢失的数据是否还是有效？

按照常理推断可能会无效了，因为空格都没有的话，那么便成为了一长段文本，但是具体还是有待验证。而对于有空格丢失的数据要如何处理呢？一种方式是利用 Bi-LSTM + CRF 做分词处理，待其处理成正常文本之后，再将其输入 BERT 做下游任务。

8.2 BERT 应用于单词拼写错误的数据是否还是有效？

如果有少量的单词拼写错误，那么造成的影响应该不会太大，因为 BERT 预训练的语料非常丰富，而且很多语料也不够干净，其中肯定也还是会含有不少单词拼写错误这样的情况。但是如果单词拼写错误的比例比较大，比如达到了 30%、50%这种比例，那么需要通过人工特征工程的方式，以中文中的同义词替换为例，将不同的错字/别字都替换成同样的词语，这样减少错别字带来的影响。例如花被、花琪、花呗、花呗、花钡均替换成花呗。

9．BERT 的 embedding 向量如何得来的？

以中文为例，「BERT 模型通过查询字向量表将文本中的每个字转换为一维向量，作为模型输入(还有 position embedding 和 segment embedding)；模型输出则是输入各字对应的融合全文语义信息后的向量表示。」

而对于输入的 token embedding、segment embedding、position embedding 都是随机生成的，需要注意的是在 Transformer 论文中的 position embedding 由 sin/cos 函数生成的固定的值，而在这里代码实现中是跟普通 word embedding 一样随机生成的，可以训练的。作者这里这样选择的原因可能是 BERT 训练的数据比 Transformer 那篇大很多，完全可以让模型自己去学习。

10．BERT 模型为什么要用 mask？它是如何做 mask 的？其 mask 相对于 CBOW 有什么异同点？

10.1 BERT 模型为什么要用 mask？

BERT 通过在输入 X 中随机 Mask 掉一部分单词，然后预训练过程的主要任务之一是根据上下文单词来预测这些被 Mask 掉的单词。其实这个就是典型

的 Denosing Autoencoder（去噪自编码器） 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。

关于 DAE LM 预训练模式，优点是它能比较自然地融入双向语言模型，同时看到被预测单词的上文和下文，然而缺点也很明显，主要在输入侧引入[Mask]标记，导致预训练阶段和 Fine-tuning 阶段不一致的问题。

10.2 它是如何做 mask 的?

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，如同上述 10.1 所述，在输入侧引入[Mask]标记，会导致预训练阶段和 Fine-tuning 阶段不一致的问题，因此在论文中为了缓解这一问题，采取了如下措施：

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

这样做的好处是，BERT 并不知道[MASK]替换的是这 15%个 Token 中的哪一个词(「注意：这里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。

10.3 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 output 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」：

- 首先，在 CBOW 中，每个单词都会成为 output word，而 BERT 不是这么做的，原因是这样做的话，训练数据就太大了，而且训练时间也会非常长。
- 其次，对于输入数据部分，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，也就是说 BERT 在输入端将待预测的 input word 用[MASK] token 代替了。
- 另外，通过 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。

10.4 为什么 BERT 中输入数据的[mask]标记为什么不能直接留空或者直接输入原始数据，在 self-attention 的 Q K V 计算中，不与待预测的单词做 Q K V 交互计算?

这个问题还要补充一点细节，就是数据可以像 CBOW 那样，每一条数据只留一个“空”，这样的话，之后在预测的时候，就可以将待预测单词之外的所有单词的表示融合起来(均值融合或者最大值融合等方式)，然后再接上 softmax 做分类。

乍一看，感觉这个 idea 确实有可能可行，而且也没有看到什么不合理之处，但是需要注意的是，这样做的话，需要每预测一个单词，就要计算一套 Q、K、V。就算不每次都计算，那么保存每次得到的 Q、K、V 也需要耗费大量的空间。总而言之，这种做法确实可能也是可行，但是实际操作难度却很大，从计算量来说，就是预训练 BERT 模型的好几倍(至少)，而且要保存中间状态也并非易事。其实还有挺重要的一点，如果像 CBOW 那样做，那么文章的“创新”在哪呢~

11．BERT 的两个预训练任务对应的损失函数是什么(用公式形式展示)?

BERT 的损失函数由两部分组成，第一部分是来自 Mask-LM 的「单词级别分类任务」，另一部分是「句子级别的分类任务」。通过这两个任务的联合学习，可以使得 BERT 学习到的表征既有 token 级别信息，同时也包含了句子级别的语义信息。具体损失函数如下：

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

其中 θ是 BERT 中 Encoder 部分的参数， θ1是 Mask-LM 任务中在 Encoder 上所接的输出层中的参数， θ2则是句子预测任务中在 Encoder 接上的分类器参数。因此，在第一部分的损失函数中，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，那么具体说来有：

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

在句子预测任务中，也是一个分类问题的损失函数：

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

因此，两个任务联合学习的损失函数是：

$$L(\theta, \theta_1, \theta_2) = -\sum_{i=1}^{M} \log p(m = m_i|\theta, \theta_1) - \sum_{j=1}^{N} \log p(n = n_i|\theta, \theta_2)$$

具体的预训练工程实现细节方面，BERT 还利用了一系列策略，使得模型更易于训练，比如对于学习率的 warm-up 策略，使用的激活函数不再是普通的 ReLu，而是 GeLu，也使用了 dropout 等常见的训练技巧。

12. 词袋模型到 word2vec 改进了什么？word2vec 到 BERT 又改进了什么？

12.1 词袋模型到 word2vec 改进了什么？

词袋模型(Bag-of-words model)是将一段文本（比如一个句子或是一个文档）用一个“装着这些词的袋子”来表示，这种表示方式不考虑文法以及词的顺序。「而在用词袋模型时，文档的向量表示直接将各词的词频向量表示加和」。通过上述描述，可以得出词袋模型的两个缺点：

- 词向量化后，词与词之间是有权重大小关系的，不一定词出现的越多，权重越大。
- 词与词之间是没有顺序关系的。

而 word2vec 是考虑词语位置关系的一种模型。通过大量语料的训练，将每一个词语映射成一个低维稠密向量，通过求余弦的方式，可以判断两个词语

之间的关系，word2vec 其底层主要采用基于 CBOW 和 Skip-Gram 算法的神经网络模型。

因此，综上所述，词袋模型到 word2vec 的改进主要集中于以下两点：

- 考虑了词与词之间的顺序，引入了上下文的信息
- 得到了词更加准确的表示，其表达的信息更为丰富

12.2 word2vec 到 BERT 又改进了什么?

word2vec 到 BERT 的改进之处其实没有很明确的答案，如同上面的问题所述，BERT 的思想其实很大程度上来源于 CBOW 模型，如果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。

13．BERT 的MASK方式的优缺点?

BERT的mask方式：在选择mask的15%的词当中，80%情况下使用mask掉这个词，10%情况下采用一个任意词替换，剩余10%情况下保持原词汇不变。

优点：

- 被随机选择15%的词当中以10%的概率用任意词替换去预测正确的词，相当于文本纠错任务，为BERT模型赋予了一定的文本纠错能力；
- 被随机选择15%的词当中以10%的概率保持不变，**缓解**了finetune时候与预训练时候输入不匹配的问题（预训练时候输入句子当中有mask，而finetune时候输入是完整无缺的句子，即为输入不匹配问题）。

缺点：

- 针对有两个及两个以上连续字组成的词，随机mask字割裂了连续字之间的相关性，使模型不太容易学习到词的语义信息。主要针对这一短板，因此google此后发表了BERT-WWM，国内的哈工大联合讯飞发表了中文版的BERT-WWM。

14．BERT中的NSP任务是否有必要?

在此后的研究（论文《Crosslingual language model pretraining》等）中发现，NSP任务可能并不是必要的，消除NSP损失在下游任务的性能上能够与原始BERT持平或略有提高。这可能是由于Bert以单句子为单位输入，模型无法学习到词之间的远程依赖关系。针对这一点，后续的RoBERTa、ALBERT、spanBERT都移去了NSP任务。

15．BERT深度双向的特点，双向体现在哪儿?

BERT使用Transformer-encoder来编码输入，encoder中的Self-attention机制在编码一个token的时候同时利用了其上下文的token，其中‘同时利用上下文’即为双向的体现，而并非向Bi-LSTM那样把句子倒序输入一遍。

16．BERT深度双向的特点，深度体现在哪儿?

针对特征提取器，Transformer只用了self-attention，没有使用RNN、CNN，并且使用了残差连接有效防止了梯度消失的问题，使之可以构建更深层的网络，所以BERT构建了多层深度Transformer来提高模型性能。

17．BERT中并行计算体现在哪儿?

不同于RNN计算当前词的特征要依赖于前文计算，有时序这个概念，是按照时序计算的，而BERT的Transformer-encoder中的self-attention计算当前词的特征时候，没有时序这个概念，是同时利用上下文信息来计算的，一句话的token特征是通过矩阵并行‘瞬间’完成运算的，故，并行就体现在self-attention。

18．BERT中Transformer中的Q、K、V存在的意义?

在使用self-attention通过上下文词语计算当前词特征的时候，X先通过WQ、WK、WV线性变换为QKV，然后如下式右边部分使用QK计算得分，最后与V计算加权和而得。

倘若不变换为QKV，直接使用每个token的向量表示点积计算重要性得分，那在softmax后的加权平均中，该词本身所占的比重将会是最大的，使得其他词的比重很少，无法有效利用上下文信息来增强当前词的语义表示。而变换为QKV再进行计算，能有效利用上下文信息，很大程度上减轻上述的影响。

19．BERT中Transformer中Self-attention后为什么要加前馈网络?

由于self-attention中的计算都是线性了，为了提高模型的非线性拟合能力，需要在其后接上前馈网络。

20．BERT中Transformer中的Self-attention多个头的作用?

类似于cnn中多个卷积核的作用，使用多头注意力，能够从不同角度提取信息，提高信息提取的全面性。

21．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12×12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。

22．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。

```
class BertLayerNorm(nn.Module):
    def __init__(self, hidden_size, eps=1e-5):
        super(BertLayerNorm, self).__init__()
        self.weight = nn.Parameter(torch.ones(hidden_size))
        self.bias = nn.Parameter(torch.zeros(hidden_size))
```

```
        self.variance_epsilon = eps
    def forward(self, x):
        u = x.mean(-1, keepdim=True)
        s = (x - u).pow(2).mean(-1, keepdim=True)
        x = (x - u) / torch.sqrt(s + self.variance_epsilon)
        return self.weight * x + self.bias
```

帖一个LayerNorm的实现，可以看到module中有weight和bias参数，以Sigmoid激活函数为例，批量归一化之后数据整体处于函数的非饱和区域， 只包含线性变换，破坏了之前学习到的特征分布。为了恢复原始数据分布，具体实现中引入了变换重构以及可学习参数w和b ，也就是上面的weight和bias，简而言之，规范化后的隐层表示将输入数据限制到了一个全局统一的确定范围，为了保证模型的表达能力不因为规范化而下降，引入了b是再平移参数，w是再缩放参数。（过激活函数前规范化，之后还原）

23．wordpiece的作用

wordpiece其核心思想是将单词打散为字符，然后根据片段的组合频率，最后单词切分成片段处理。和原有的分词相比，能够极大的降低OOV的情况，例如cosplayer, 使用分词的话如果出现频率较低则是UNK，但bpe可以把它切分成cos play er, 模型可以根据词根以及前缀等信息，学习到这个词的大致信息，而不是一个OOV。

wordpiece与BPE(Byte Pair Encoding)算法类似，也是每次从词表中选出两个子词合并成新的子词。与BPE的最大区别在于，如何选择两个子词进行合并：BPE选择频数最高的相邻子词合并，而WordPiece选择能够提升语言模型概率最大的相邻子词加入词表。

24．如何优化BERT效果

- 感觉最有效的方式还是数据。
- 把现有的大模型ERNIE_2.0_large, Roberta，roberta_wwm_ext_large、roberta-pair-large等进行ensemble，然后蒸馏原始的bert模型，这是能有效提高的，只是操作代价比较大。
- BERT上面加一些网络结构，比如attention，rcnn等，个人得到的结果感觉和直接在上面加一层transformer layer的效果差不多。
- 改进预训练，在特定的大规模数据上预训练，相比于开源的用百科，知

道等数据训练的更适合你的任务，以及在训练后续mask的时候去mask低频词或者实体词（听说过有人这么做有收益，但没具体验证）。

- 文本对抗，作者了解的不多。感兴趣可以看看BERT-ATTACK: Adversarial Attack Against BERT Using BERT

25．如何优化BERT性能

- 压缩层数，然后蒸馏，直接复用12层bert的前4层或者前6层，效果能和12层基本持平，如果不蒸馏会差一些。
- 双塔模型（短文本匹配任务），将bert作为一个encoder，输入query编码成向量，输入title编码成向量，最后加一个DNN网络计算打分即可。
- int8预估，在保证模型精度的前提下，将Float32的模型转换成Int8的模型。
- 提前结束，大致思想是简单的case前面几层就可以输出分类结果，比较难区分的case走完12层，但这个在batch里面计算应该怎么优化还没看明白，有的提前结束有的最后结束，如果在一个batch里面的话就不太好弄。感兴趣的可以看看BERT Loses Patience:Fast and Robust

26．elmo、GPT和bert在单双向语言模型处理上的不同之处?

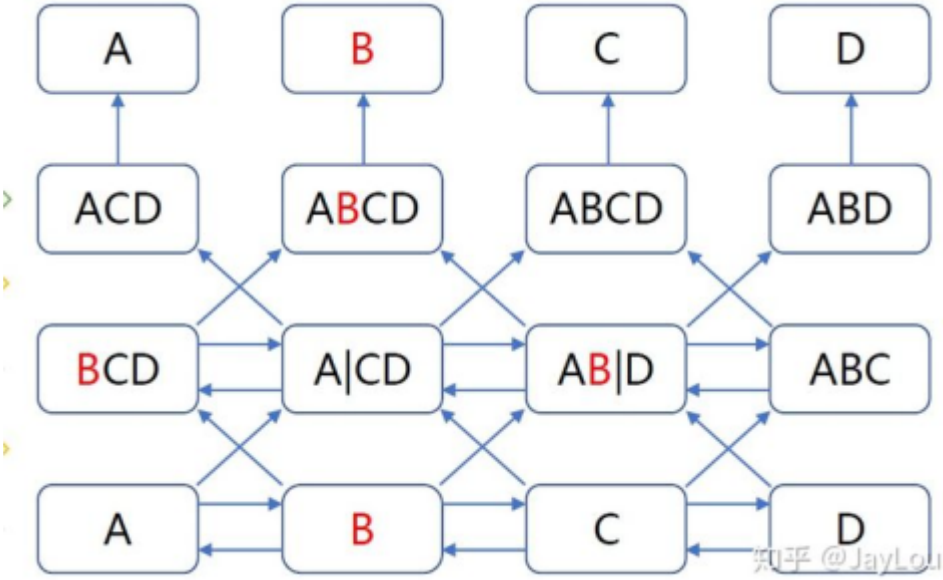
在上述3个模型中，只有bert共同依赖于左右上下文。那elmo不是双向吗？实际上elmo使用的是经过独立训练的从左到右和从右到左LSTM的串联拼接起来的。而GPT使用从左到右的Transformer，实际就是“Transformer decoder”。

27．bert构建双向语言模型不是很简单吗？不也可以直接像elmo拼接Transformer decoder吗?

BERT 的作者认为，这种拼接式的bi-directional 仍然不能完整地理解整个语句的语义。更好的办法是用上下文全向来预测[mask]，也就是用 “能/实现/语言/表征/../的/模型”，来预测[mask]。BERT 作者把上下文全向的预测方法，称之为 deep bi-directional。

29．bert为什么要采取Marked LM，而不直接应用Transformer Encoder?

我们知道向Transformer这样深度越深，学习效果会越好。可是为什么不直接应用双向模型呢？因为随着网络深度增加会导致标签泄露。如下图：



深度双向模型比left-to-right 模型或left-to-right and right-to-left模型的浅层连接更强大。遗憾的是，标准条件语言模型只能从左到右或从右到左进行训练，因为双向条件作用将允许每个单词在多层上下文中间接地“see itself”。

为了训练一个深度双向表示（deep bidirectional representation），研究团队采用了一种简单的方法，即随机屏蔽（masking）部分输入token，然后只预测那些被屏蔽的token。论文将这个过程称为“masked LM”(MLM)。

30．bert为什么并不总是用实际的[MASK]token替换被“masked”的词汇?

首先，预训练和finetuning之间不匹配，因为在finetuning期间从未看到[MASK]token。为了解决这个问题，团队并不总是用实际的[MASK]token替换被“masked”的词汇。相反，训练数据生成器随机选择15%的token。Transformer encoder不知道它将被要求预测哪些单词或哪些单词已被随机单词替换，因此它被迫保持每个输入token的分布式上下文表示。

此外，因为随机替换只发生在所有token的1.5%（即15%的10%），这似乎不会损害模型的语言理解能力。使用MLM的第二个缺点是每个batch只预测了15%的token，这表明模型可能需要更多的预训练步骤才能收敛。团队证明MLM的收敛速度略慢于 left-to-right的模型（预测每个token），但MLM模型在实验上获得的提升远远超过增加的训练成本。

https://zhuanlan.zhihu.com/p/56382372
<https://zhuanlan.zhihu.com/p/56382372>