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[1]，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。

最后的实验表明 BERT 模型的有效性，并在 11 项 NLP 任务中夺得 SOTA 结果。

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。

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。

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：

1. 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
2. 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。

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 。

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。

BERT 的三个 Embedding 直接相加会对语义有影响吗?

Embedding 的数学本质，就是以 one hot 为输入的单层全连接。

也就是说，世界上本没什么 Embedding，有的只是 one hot。

现在我们将 token, position, segment 三者都用 one hot 表示，然后 concat 起来，然后才去过一个单层全连接，等价的效果就是三个 Embedding 相加。

因此，BERT 的三个 Embedding 相加，其实可以理解为 token, position, segment 三个用 one hot 表示的特征的 concat，而特征的 concat 在深度学习领域是很常规的操作了。

用一个例子理解一下：

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]，再经过维度为 [4+3+2,768] = [9, 768] 的 Embedding 矩阵，得到的 word Embedding 其实就是一样的。

Embedding 就是以 one hot 为输入的单层全连接。

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：1）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；2）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。

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。

BERT深度双向的特点，双向体现在哪儿?

BERT的预训练模型中，预训练任务是一个mask LM ，通过随机的把句子中的单词替换成mask标签， 然后对单词进行预测。

这里注意到，对于模型，输入的是一个被挖了空的句子， 而由于Transformer的特性（Self-attention机制）， 它是会注意到所有的单词的，这就导致模型会根据挖空的上下文来进行预测， 这就实现了双向表示， 说明BERT是一个双向的语言模型。

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。

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。

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。

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W_Q、W_K、W_V线性变换为QKV，然后使用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。

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。

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。

BERT参数量计算

bert的参数主要可以分为四部分：embedding层的权重矩阵、multi-head attention、layer normalization、feed forward。

```
BertModel(vocab_size=30522,
hidden_size=768, max_position_embeddings=512,
token_type_embeddings=2)
```

一、embedding层

embedding层有三部分组成：token embedding、segment embedding和position embedding。

token embedding：词表大小词向量维度就是对应的参数了，也就是30522*768

segment embedding：主要用01来区分上下句子，那么参数就是2*768

position embedding：文本输入最长为512，那么参数为512*768

因此embedding层的参数为(30522+2+512)*768=23835648

二、multi-head attention

Q，K，V就是我们输入的三个句子词向量，从之前的词向量分析可知，输出向量大小从len -> len x hidden_size，即len x 768。如果是self-attention，Q=K=V，如果是普通的attention，Q !=K=V。但是，不管用的是self-attention还是普通的attention，参数计算并不影响。因为在输入单头head时，对QKV的向量均进行了不同的线性变换，引入了三个参数，W1，W2，W3。其维度均为：768 x 64。

为什么是64呢，从下图可知， Wi 的维度： dmodel x dk | dv | dq 而：dk | dv | dq = dmodle/h，h是头的数量，dmodel模型的大小，即h=12，dmodle=768； 所以：dk | dv | dq=768/12=64 得出：W1，W2，W3的维度为768 x 64

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,...,\text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

In this work we employ $h = 8$ parallel attention layers, or heads. For each of these we use $d_k = d_v = d_{\text{model}}/h = 64$. Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

每个head的参数为768*768/12，对应到QKV三个权重矩阵自然是768*768/12*3，12个head的参数就是768*768/12*3*12，拼接后经过一个线性变换，这个线性变换对应的权重为768*768。

因此1层multi-head attention部分的参数为`768*768/12*3*12+768*768=2359296`

12层自然是`12*2359296=28311552`

三、layer normalization

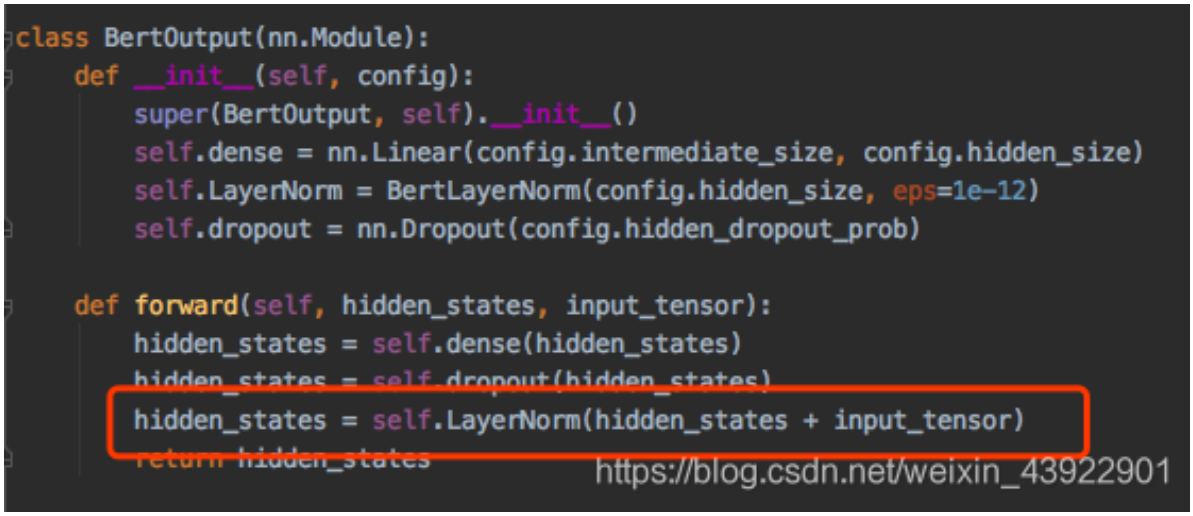
文章其实并没有写出layernorm层的参数，但是在代码中有，分别为gamma和beta。有三个地方用到了layer normalization，分别是embedding层后、multi-head attention后、feed forward后 ①词向量处

```python
class BertEmbeddings(nn.Module):
    """Construct the embeddings from word, position and token_type embeddings.
    """

    def __init__(self, config):
        super(BertEmbeddings, self).__init__()
        self.word_embeddings = nn.Embedding(config.vocab_size, config.hidden_size, padding_idx=0)
        self.position_embeddings = nn.Embedding(config.max_position_embeddings, config.hidden_size)
        self.token_type_embeddings = nn.Embedding(config.type_vocab_size, config.hidden_size)

        # self.LayerNorm is not snake-cased to stick with TensorFlow model variable name and be able to load
        # any TensorFlow checkpoint file
        self.LayerNorm = BertLayerNorm(config.hidden_size, eps=1e-12)
        self.dropout = nn.Dropout(config.hidden_dropout_prob)

    def forward(self, input_ids, token_type_ids=None, position_ids=None):

        seq_length = input_ids.size(1)
        if position_ids is None:
            position_ids = torch.arange(seq_length, dtype=torch.long, device=input_ids.device)
            position_ids = position_ids.unsqueeze(0).expand_as(input_ids)

        if token_type_ids is None:
            token_type_ids = torch.zeros_like(input_ids)

        words_embeddings = self.word_embeddings(input_ids)
        position_embeddings = self.position_embeddings(position_ids)
        token_type_embeddings = self.token_type_embeddings(token_type_ids)

        embeddings = words_embeddings + position_embeddings + token_type_embeddings
        embeddings = self.LayerNorm(embeddings)
        embeddings = self.dropout(embeddings)
        return embeddings
```


②多头注意力之后

```python
class BertSelfOutput(nn.Module):
    def __init__(self, config):
        super(BertSelfOutput, self).__init__()
        self.dense = nn.Linear(config.hidden_size, config.hidden_size)
        self.LayerNorm = BertLayerNorm(config.hidden_size, eps=1e-12)
        self.dropout = nn.Dropout(config.hidden_dropout_prob)

    def forward(self, hidden_states, input_tensor):
        hidden_states = self.dense(hidden_states)
        hidden_states = self.dropout(hidden_states)
        hidden_states = self.LayerNorm(hidden_states + input_tensor)
        return hidden_states
```


③最后的全连接层之后



但是参数都很少，gamma和beta的维度均为768。因此总参数为768 * 2 + 768 * 2 * 2 * 12（层数）

这三部分的参数为`768*2+12*(768*2+768*2)=38400`

四、feed forward

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

以上是论文中全连接层的公式，其中用到了两个参数W1和W2，Bert沿用了惯用的全连接层大小设置，即4 * dmodle，为3072，因此，W1，W2大小为768 * 3072，2个为 2 * 768 * 3072。

`12*（768*3072+3072*768）=56623104`

五、总结

总的参数=embedding+multi-head attention+layer normalization+feed forward

=23835648+28311552+38400+56623104

=108808704

≈110M

另一种算法

- 词向量参数（包括layernorm） + 12 * （Multi-Heads参数 + 全连接层参数 + layernorm参数）= （30522+512 + 2）* 768 + 768 * 2 + 12 * （768 * 768 / 12 * 3 * 12 + 768 * 768 + 768 * 3072 * 2 + 768 * 2 * 2） = 108808704.0 ≈ 110M

PS：这里介绍的参数仅仅是encoder的参数，基于encoder的两个任务next sentence prediction 和 MLM涉及的参数（分别是768 * 2，768 * 768，总共约0.5M）并未加入，此外涉及的bias由于参数很少，这里也并未加入。

参考

- https://zhuanlan.zhihu.com/p/357353536
- https://blog.csdn.net/weixin_43922901/article/details/102602557

- https://github.com/google-research/bert/issues/656

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?

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。

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。

BERT为什么意义重大

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。

BERT的“里程碑”意义在于：证明了一个非常深的模型可以显著提高NLP任务的准确率，而这个模型可以从无标记数据集中预训练得到。

既然NLP的很多任务都存在数据少的问题，那么要从无标注数据中挖潜就变得非常必要。在NLP中，一个最直接的有效利用无标注数据的任务就是语言模型，因此很多任务都使用了语言模型作为预训练任务。但是这些模型依然比较“浅”，比如上一个大杀器，AllenNLP的ELMO也就是三层的BiLSTM。

那么有没有可以胜任NLP任务的深层模型？有，就是transformer。这两年，transformer已经在机器翻译任务上取得了很大的成功，并且可以做的非常深。自然地，我们可以用transformer在语言模型上做预训练。因为transformer是encoder-decoder结构，语言模型就只需要decoder部分就够了。OpenAI的GPT就是这样。但decoder部分其实并不好。因为我们需要的是一个完整句子的encoder，而decoder的部分见到的都是不完整的句子。所以就有了BERT，利用transformer的encoder来进行预训练。但这个就比较“反直觉”，一般人想不到了。

我们再来看下BERT有哪些“反直觉”的设置?

ELMO的设置其实是最符合直觉的预训练套路，两个方向的语言模型刚好可以用来预训练一个BiLSTM，非常容易理解。但是受限于LSTM的能力，无法变深了。那如何用transformer在无标注数据行来做一个预训练模型呢？一个最容易想到的方式就是GPT的方式，事实证明效果也不错。那还有没有“更好”的方式？直观上是没有了。而BERT就用了两个反直觉的手段来找到了一个方法。

(1) 用比语言模型更简单的任务来做预训练。直觉上，要做更深的模型，需要设置一个比语言模型更难的任务，而BERT则选择了两个看起来更简单的任务：完形填空和句对预测。

(2) 完形填空任务在直观上很难作为其它任务的预训练任务。在完形填空任务中，需要mask掉一些词，这样预训练出来的模型是有缺陷的，因为在其它任务中不能mask掉这些词。而BERT通过随机的方式来解决了这个缺陷：80%加Mask，10%用其它词随机替换，10%保留原词。这样模型就具备了迁移能力。

感觉上，作者Jacob Devlin是拿着锤子找钉子。既然transformer已经证明了是可以handle大数据，那么就给它设计一种有大数据的任务，即使是“简单”任务也行。理论上BiLSTM也可以完成BERT里的两个任务，但是在大数据上BERT更有优势。

对NLP的影响

总体上，BERT模型的成功还在于是一种表示学习，即通过一个深层模型来学习到一个更好的文本特征。这种非RNN式的模型是非图灵完备的，无法单独完成NLP中推理、决策等计算问题。当然，一个好的表示会使得后续的任务更简单。

BERT能否像ResNet那样流行还取决于其使用的便利性，包括模型实现、训练、可迁移性等，可能有好的模型出现，但类似的预训练模型会成为NLP任务的标配，就像Word2vec，Glove那样。

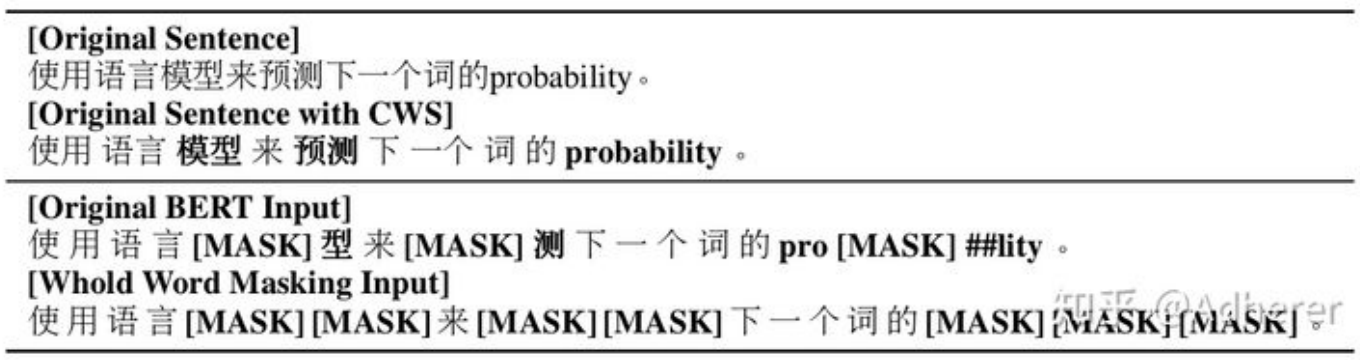
最后，BERT也打开了一个思路：可以继续在无标注数据上挖潜，而不仅仅限于语言模型。

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：



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。

BERT的优缺点

优点：

- 并行，解决长时依赖，双向特征表示，特征提取能力强，有效捕获上下文的全局信息，缓解梯度消失的问题等，BERT擅长解决的NLU任务。

缺点：

- 生成任务表现不佳：预训练过程和生成过程的不一致，导致在生成任务上效果不佳；
- 采取独立性假设：没有考虑预测[MASK]之间的相关性，是对语言模型联合概率的有偏估计（不是密度估计）；
- 输入噪声[MASK]，造成预训练-精调两阶段之间的差异；
- 无法文档级别的NLP任务，只适合于句子和段落级别的任务；

BERT模型的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：首先，在CBOW中，每个单词都会成为input 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。

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

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

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。

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：

(1).LSTM抽取特征的能力远弱于Transformer

(2).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)

(3).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

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。

elmo、GPT、bert三者之间有什么区别?

之前介绍词向量均是静态的词向量，无法解决一次多义等问题。下面介绍三种elmo、GPT、bert词向量，它们都是基于语言模型的动态词向量。下面从几个方面对这三者进行对比：

（1）**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采用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。

（2）**单/双向语言模型**：

- GPT采用单向语言模型，elmo和bert采用双向语言模型。但是elmo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。

BERT变体有哪些

XLNet

这个是融合了GPT和Bert两家的模型。

GPT是自编码模型， 通过双向LSTM编码提取语义。

Bert是自回归模型，不分前后，一坨扔进去，再用attention加强提取语义的效果。

XLNet融合了Bert的结构和GPT的有向预测，具体做法有点抽象，请耐心。

首先，XLNet觉得Bert给的任务太简单了。Bert是Mask language model，举例说明，假如Bert顺序输入abcd四个词，Bert会随机mask掉一部分词，假设mask成了ab[mask]d，接下来Bert要根据这个mask序列预测mask位置的词"c"。

XLNet认为不该一下子把所有词都告诉网络，可以先告诉网络第一个词是a，然后预测一下，再告诉网络一二位置的词是ab，再预测一下，最后告诉网络一二四位置的词是abd，再预测一下。这样力求网络能以最少的信息预测出结果。更有甚者，可能先告诉b，再告诉bd，再告诉abd.

为了实现上面的训练方式，XLNet这样做：

对于序列abcd，先给其加上position embedding. 文字不好表述，就是加了一维位置向量。

随机打散，可能变成bcda, 可能变成adbc. 这里以adbc为例。

根据a->d->b->c这个序列，我们预测a时什么都看不到，预测d时能看到a，预测b时能看到ad，预测c时能看到adb。我们调换一下顺序：a:None,b:ad,c:abd,d:a。这四种情况就对应了上述在分别mask掉a、b、c、d时的某一种情况。我们完全可以根据原始的输入"abcd"再加上01的掩码获得上述情况下的各种输入，即a位置掩盖所有，b位置掩盖c，c位置不掩盖，d位置掩盖bc。如果是自监督学习，就自己掩盖自己，如果是任务学习，就不掩盖自己。

总之，使用这种掩码机制可以等效为mask机制+序列预测的难度加强版。因为掩码机制使用矩阵，看着很像attention，就蹭个热度。又加之网络本身有自己的标准attention，故得名“双流attention”.

通过掩码机制，可以省略添加mask标记的过程，而众所周知mask标记会些许干扰语义模型的表示，所以XLNet算是解决了这个问题。

RoBERTa

RoBERTa 模型是BERT 的改进版(从其名字来看，A Robustly Optimized BERT，即简单粗暴称为强力优化的BERT方法)。 在模型规模、算力和数据上，与BERT相比主要有以下几点改进：

- 更大的模型参数量（论文提供的训练时间来看，模型使用 1024 块 V100 GPU 训练了 1 天的时间）
- 更大batch size。RoBERTa 在训练过程中使用了更大的bacth size。尝试过从 256 到 8000 不等的bacth size。
- 更多的训练数据（包括：CC-NEWS 等在内的 160GB 纯文本。而最初的BERT使用16GB BookCorpus数据集和英语维基百科进行训练）

另外，RoBERTa在训练方法上有以下改进：

- 去掉下一句预测(NSP)任务。
- 动态掩码。BERT 依赖随机掩码和预测 token。原版的 BERT 实现在数据预处理期间执行一次掩码，得到一个静态掩码。 而 RoBERTa 使用了动态掩码：每次向模型输入一个序列时都会生成新的掩码模式。这样，在大量数据不断输入的过程中，模型会逐渐适应不同的掩码策略，学习不同的语言表征。
- 文本编码。Byte-Pair Encoding（BPE）是字符级和词级别表征的混合，支持处理自然语言语料库中的众多常

见词汇。原版的 BERT 实现使用字符级别的 BPE 词汇，大小为 30K，是在利用启发式分词规则对输入进行预处理之后学得的。Facebook 研究者没有采用这种方式，而是考虑用更大的 byte 级别 BPE 词汇表来训练 BERT，这一词汇表包含 50K 的 subword 单元，且没有对输入作任何额外的预处理或分词。

ALBERT

ALBERT是紧跟着RoBERTa出来的，也是针对Bert的一些调整，重点在减少模型参数，对速度倒是没有特意优化。

提出了两种能够大幅减少预训练模型参数量的方法

- Factorized embedding parameterization。将embedding matrix分解为两个大小分别为V x E和E x H矩阵。这使得embedding matrix的维度从O(V x H)减小到O(V x E + E x H)。
- Cross-layer parameter sharing，即多个层使用相同的参数。参数共享有三种方式：只共享feed-forward network的参数、只共享attention的参数、共享全部参数。ALBERT默认是共享全部参数的。

使用Sentence-order prediction (SOP)来取代NSP。具体来说，其正例与NSP相同，但负例是通过选择一篇文档中的两个连续的句子并将它们的顺序交换构造的。这样两个句子就会有相同的话题，模型学习到的就更多是句子间的连贯性。

参考资料

BERT面试8问8答

关于BERT的若干问题整理记录

如何评价 BERT 模型?

transformer、bert、ViT常见面试题总结

BERT及其变种们