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：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
- word2vec：其又能根据句子上下文获取词义，从而避免歧义出现。
- ELMO：elmo 是伪双向，只是将左到右，右到左的信息加起来，而且用的是 `lstm`，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。

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，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 根据上下文动态调整 word embedding，可以解决多义词的问题**。
- **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

只是使用了 transformer 的 encoder

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。**

网络层数 L，隐藏层维度 H，Attention 多头个数 A

- base：L=12, H=768, A=12, 110M,使用 GPU 内存：7G 多
- large: L=24,H=1024,A=16, 340M,使用 GPU 内存：32G 多
- transformer 是 512 维，encoder 是 6 个堆叠，8 个头，
- bert 是 12 个 transformer 叠加。每一个 transformer 由 6 个 encoder 叠加

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，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)

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

ERT 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. **位置嵌入（Position Embedding）**：位置嵌入是指将单词的位置信息编码成特征向量，位置嵌入是向模型中引入单词位置关系的至关重要的一环；
2. **WordPiece 嵌入**：WordPiece 是指将单词划分成一组有限的公共子词单元，能在单词的有效性和字符的灵活性之间取得一个折中的平衡。例如上图的示例中‘playing’被拆分成了‘play’和‘ing’；
3. **分割嵌入（Segment Embedding）**：用于区分两个句子，例如 B 是否是 A 的下文（对话场景，问答场景等）。对于句子对，第一个句子的特征值是 0，第二个句子的特征值是 1。」

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。

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”**！

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。

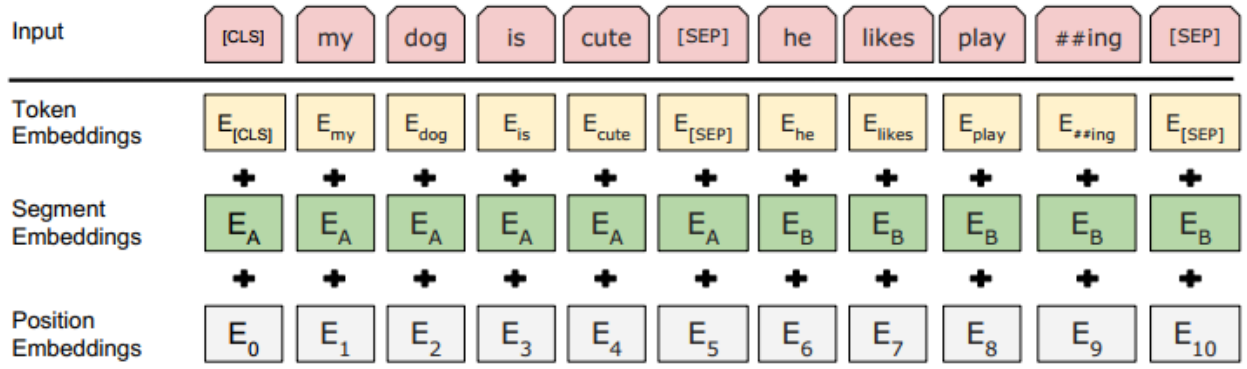
在这里想用一个例子再尝试解释一下：

- 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。

1.4 使用 BERT 预训练模型为什么最多只能输入 512 个词?

这是 Google BERT 预训练模型初始设置的原因，前者对应 Position Embeddings，后者对应 Segment Embeddings



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。

3.3 BERT 如何区分一词多义?

同一个字在转换为 bert 的输入之后（id），embedding 的向量是一样，但是通过 bert 中的多层 transformer encoder 之后，**attention 关注不同的上下文，就会导致不同句子输入到 bert 之后，相同字输出的字向量是不同的**，这样就解决了一词多义的问题。

3.4 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 infrence 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。

3.5 为什么说 ELMO 是伪双向，BERT 是真双向?

- ELMo 是伪双向，**只是将左到右，右到左的信息加起来**，而且用的是 lstm，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。

3.6 BERT 和 Transformer Encoder 的差异有哪些?

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` 。

加入 `Segment Embeddings` 的原因：Bert 会处理句对分类、问答等任务，这里会出现句对关系，而两个句子是有先后顺序关系的，如果不考虑，就会出现词袋子之类的问题（如：武松打虎 和 虎打武松 是一个意思了 ~），因此 Bert 加入了句子向量。

3.7 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）。

3.8 FFN 的作用?

- 增强模型的特征提取能力
- FFN 中的 ReLU 成为了一个主要的提供非线性变换的单元。

3.9 BERT 非线性的来源

- **前馈层的 GeLU 激活函数**
- **self-attention**：self-attention 是非线性的（来自 softmax）

> GeLU：在激活中引入了随机正则的思想，根据当前 input 大于其余 inputs 的概率进行随机正则化，即为在 mask 时依赖输入的数据分布，即 x 越小越有可能被 mask 掉，因此服从伯努利分布 $\text{Bernoulli}(\phi(x))$ ，其中， $\phi(x) = P(X \leq x)$
> ReLU：缺乏随机因素，只用 0 和 1

3.10 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。

3.11 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. 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。
2. 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**。
3. 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。

3.12 对于长度较长的语料，如何训练?

对于长文本，有两种处理方式，截断和切分。

- **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 损失函数组成：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；

优点：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。

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

- θ : BERT 中 Encoder 部分的参数；
- θ_1 : 是 Mask-LM 任务中在 Encoder 上所接的输出层中的参数；
- θ_2 :是句子预测任务中在 Encoder 接上的分类器参数；

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，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：

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

在第二部分的损失函数中，在句子预测任务中，也是一个分类问题的损失函数：

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

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。