

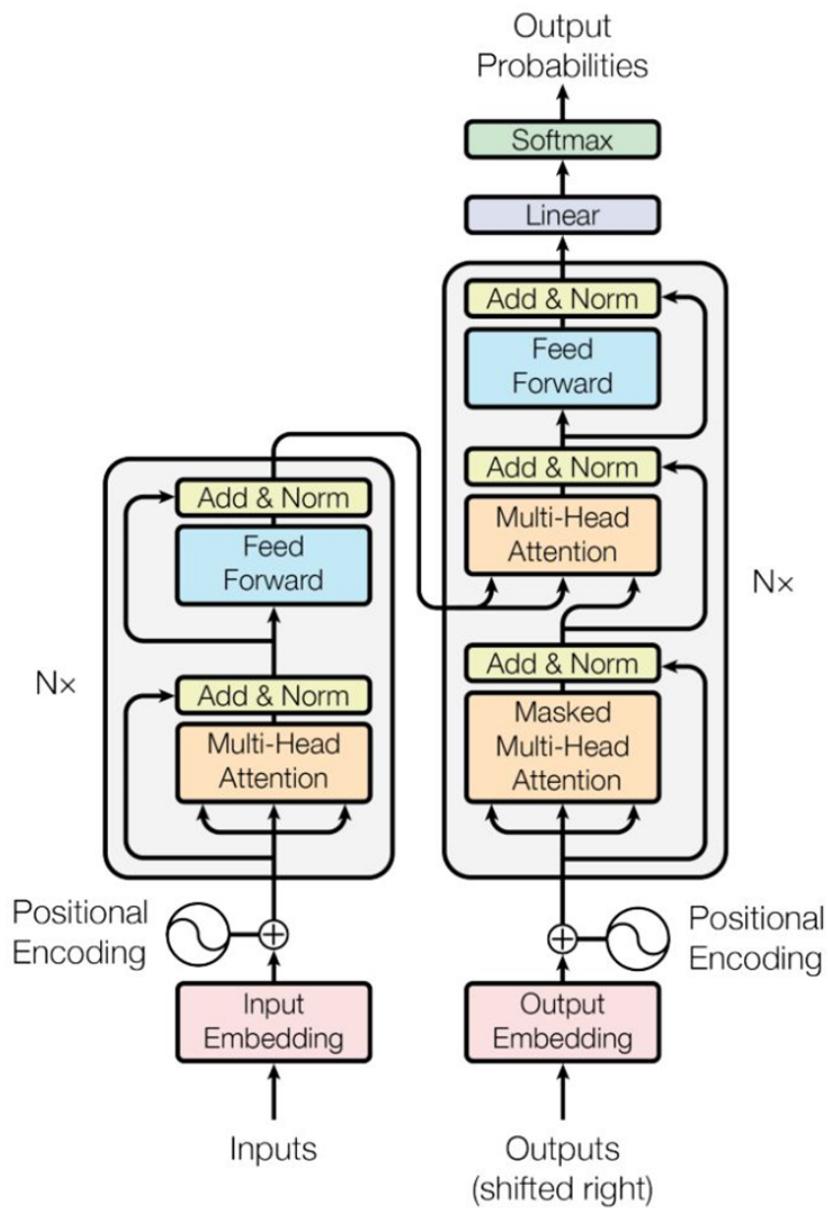
Transformer/Bert基础知识及面试点

基础回顾

- G:\OneDrive\5 学习\00. 自己做的slides\00. 机器学习活动 Slides - Xianjun

Transformer

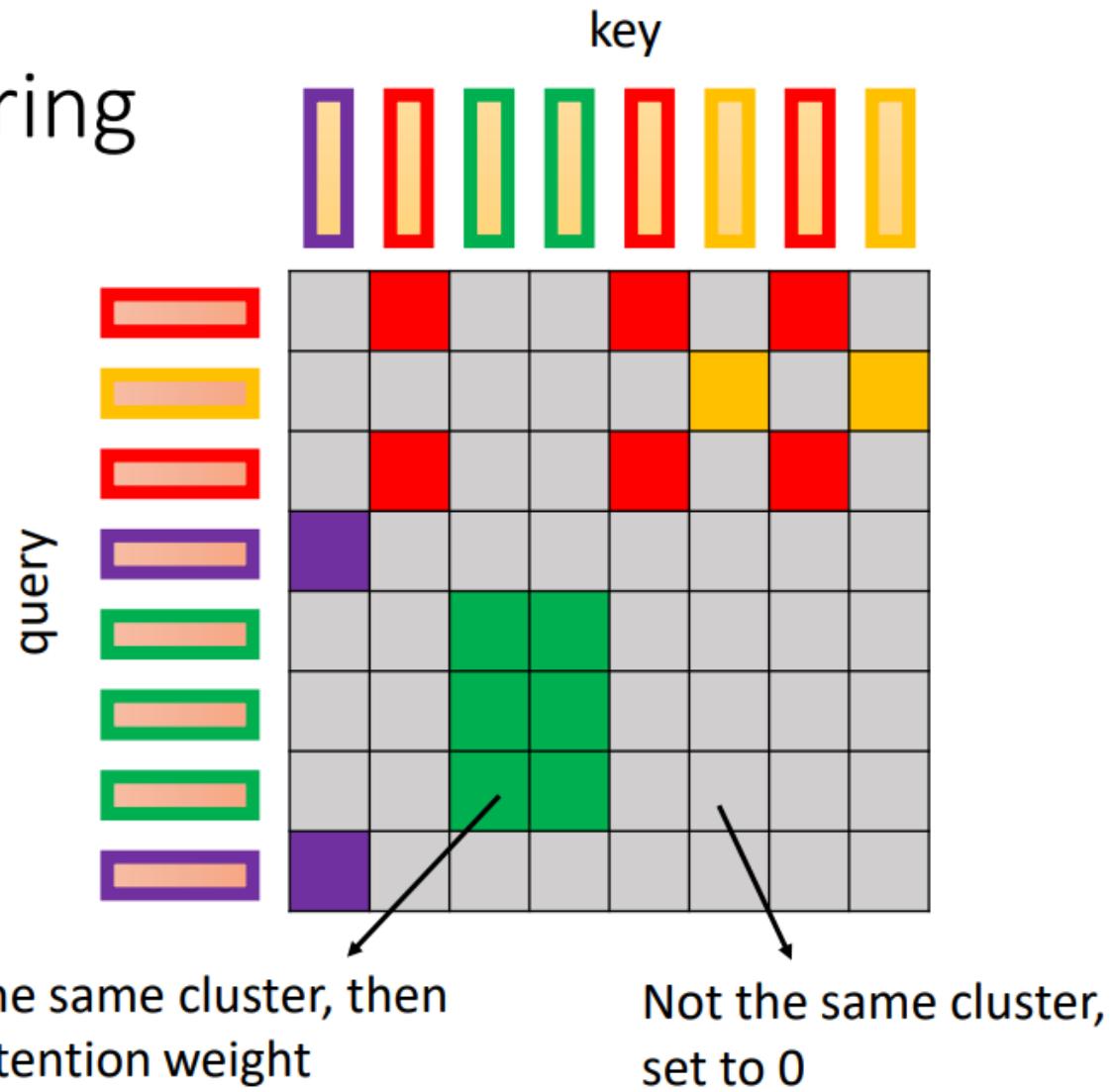
- 架构



- 拆分stack encoder -> BERT, stack decoder -> GPT
- Transformer decoder与encoder不同支持: masked multihead attention, cross attention
- Self-attention的问题是计算量大 (N^2) , 为了改进计算量, 产生了一些其他形式的attention
 - Reformer
 - 尝试预测哪些attention是重要的(attention结果大), 哪些不重要。利用clustering的方法对query和key快速聚类, 只有query和key在相同的cluster才做计算

Clustering

Step 2



Belong to the same cluster, then
calculate attention weight

Not the same cluster,
set to 0

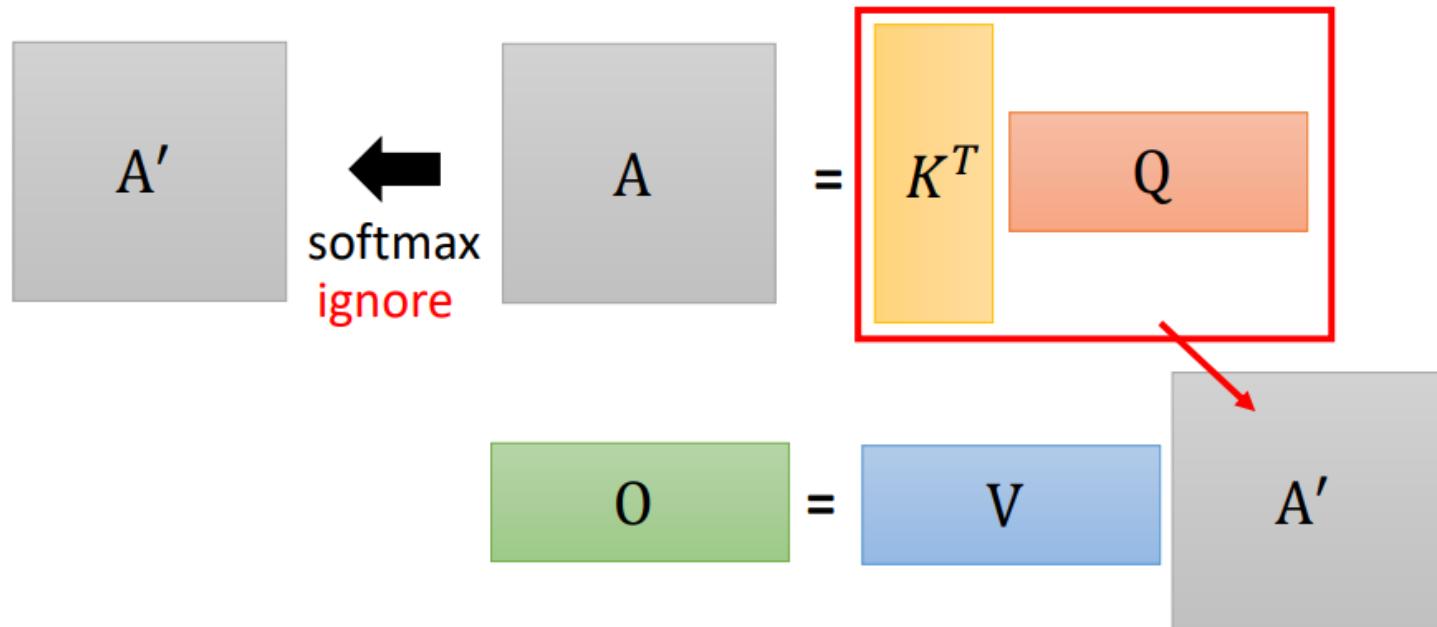
- Linformer

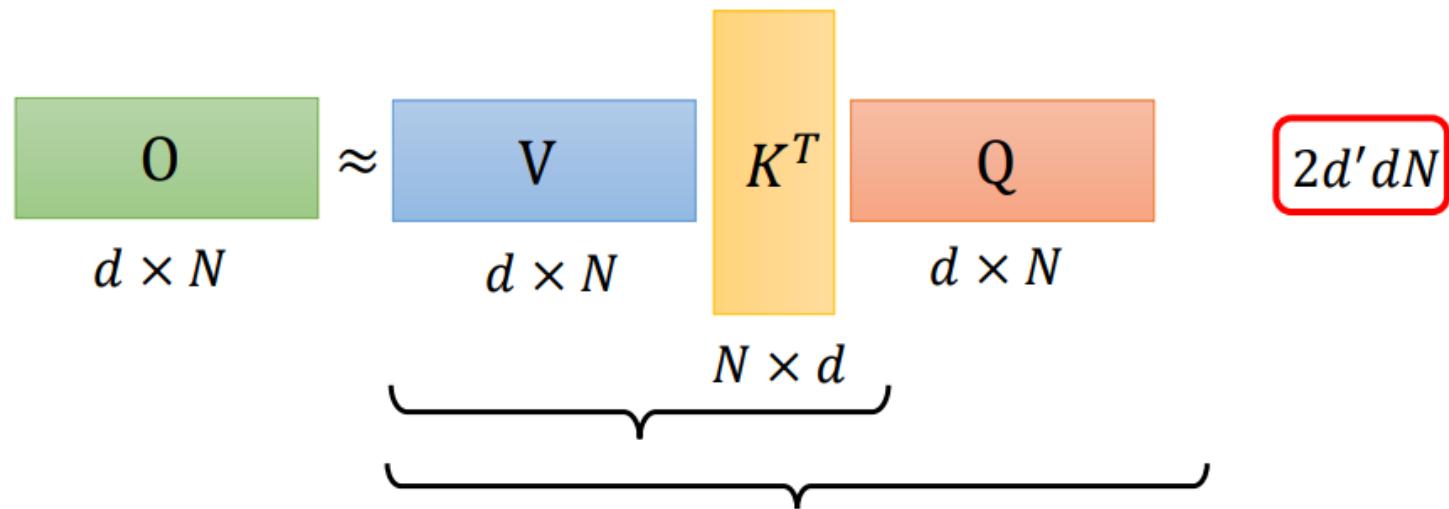
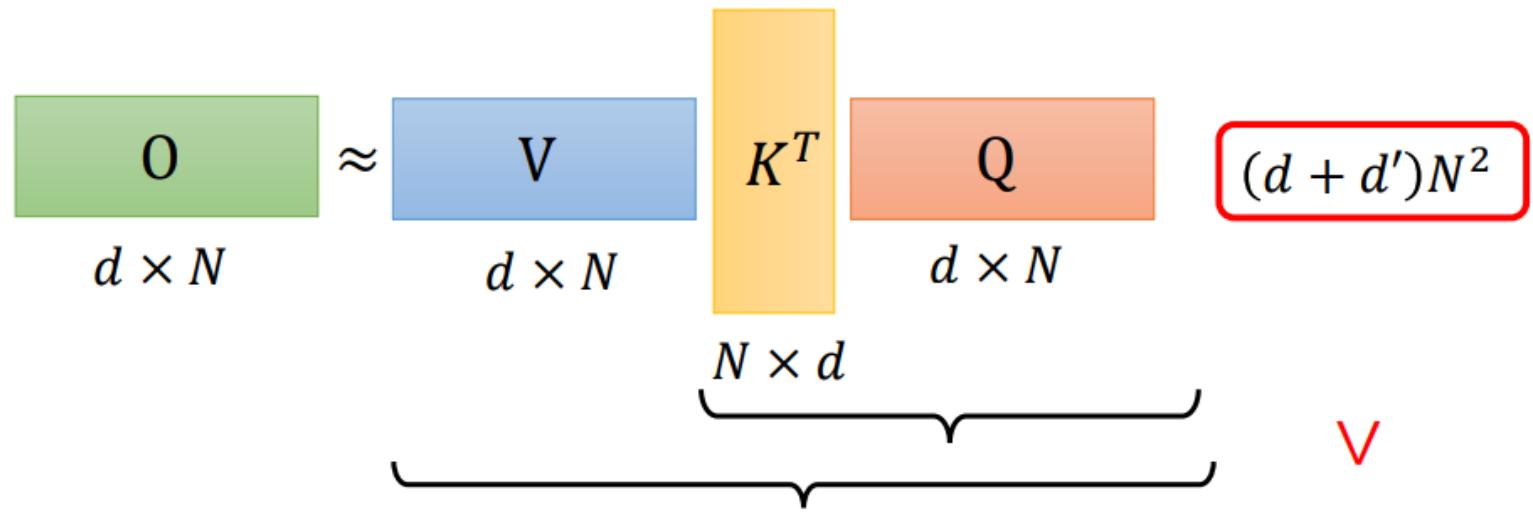
- Do we need full attention matrix? Attention map里面很多column都是duplicated, 所以attention matrix是low rank的，所以拿掉重复的，只计算low rank的matrics就可以简化计算。
 - 也做了q,k,v先计算kv的优化
- Performer (主要做计算加速，见下方Self-attention加速和summary section)
- Self-attention计算加速
 - N是sequence长度，一般都远大于d和d'
 - 本图假设没有softmax操作，加上softmax后也可以数学推导做类似优化

Attention Mechanism is three-matrix Multiplication

Review

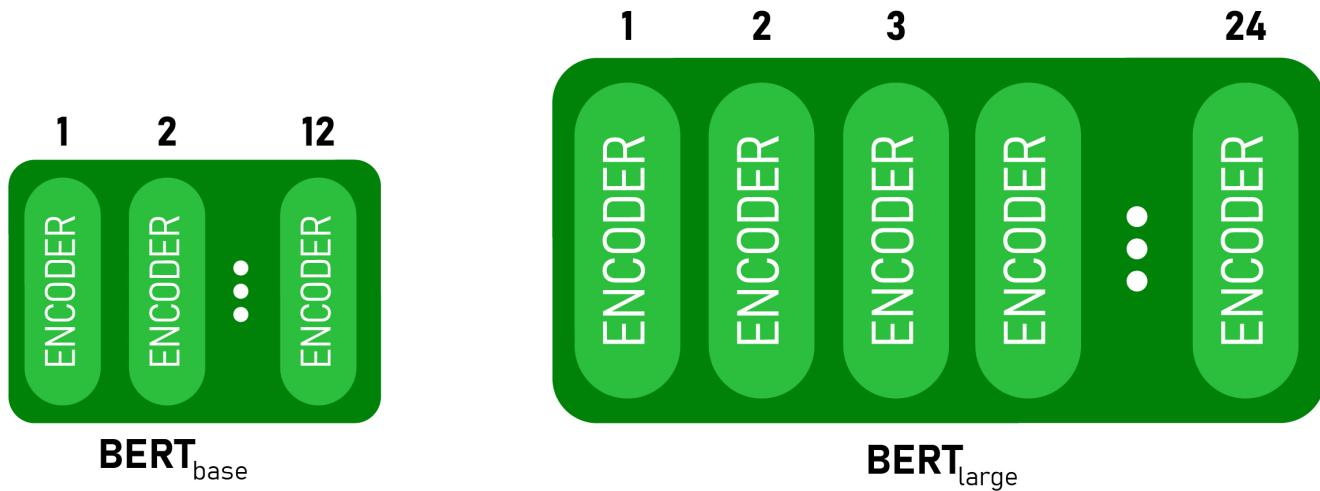
$$d \times N \quad \begin{matrix} Q \\ K \\ V \end{matrix} = \begin{matrix} W^q \\ W^k \\ W^v \end{matrix} \begin{matrix} I \\ I \\ I \end{matrix}$$





BERT

- Arch



- BERT基础
 - Self-supervised learning, 采用的是autoencoder的方法
 - Bert embedding是一种contextualized word embedding
 - bert中的special token:
 - [mask]: 主要是为了MLM任务, Bert的embedding向量中也有专门存放[mask]这个token的embedding向量
 - [cls]: special classification embedding, 句首符号, 单输出head
 - [sep]: 放在句尾, 用来区分句子, 因为bert中有个NSP的任务
 - [pad]: 占位符, 用来做padding
 - [unk]: 更多是为了预测服务的, 如果为了input的句子里有embedding矩阵没有的token, 分token的时候, 这类token都会变成unk
- Bert training

- Task 1: Masking input / Mask language Modeling (MLM)
 - Mask有两种方法: 换成special token, 随机替换为其他token. Bert里面先决定哪些被mask, 然后决定是怎么mask (换成special token或者随机换成其他token)

Masking Input

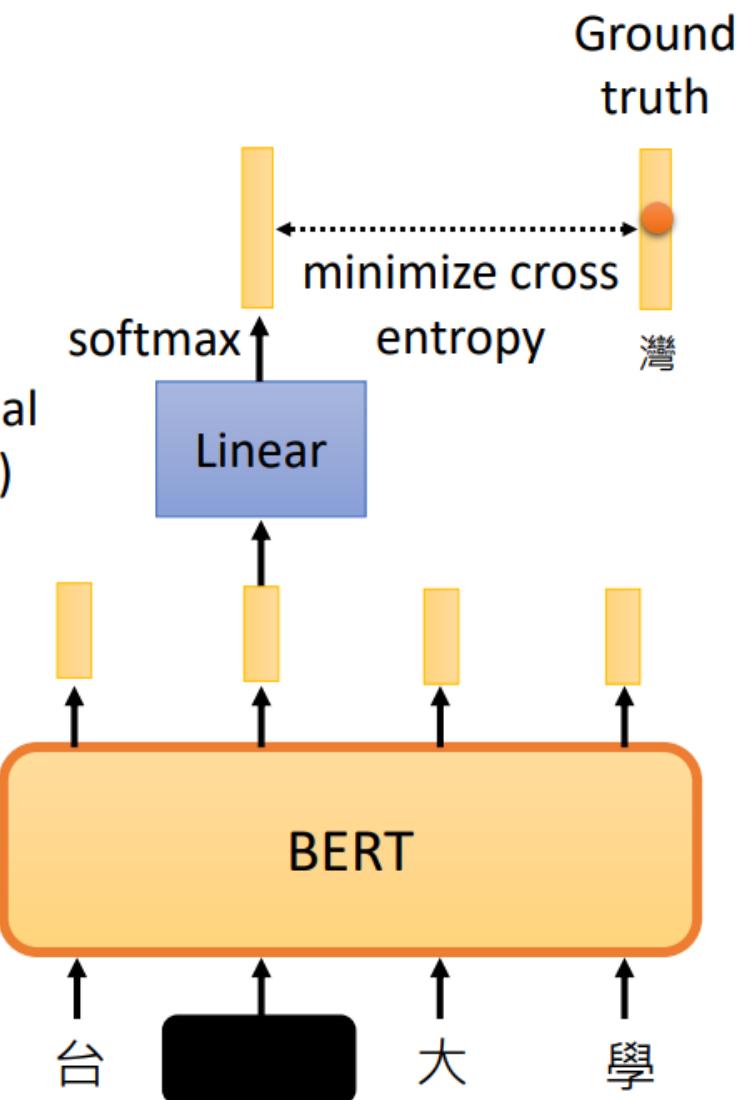
<https://arxiv.org/abs/1810.04805>

= MASK (special token)
or

= Random
一、天、大、小 ...

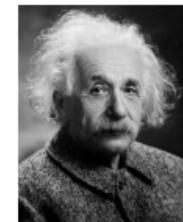
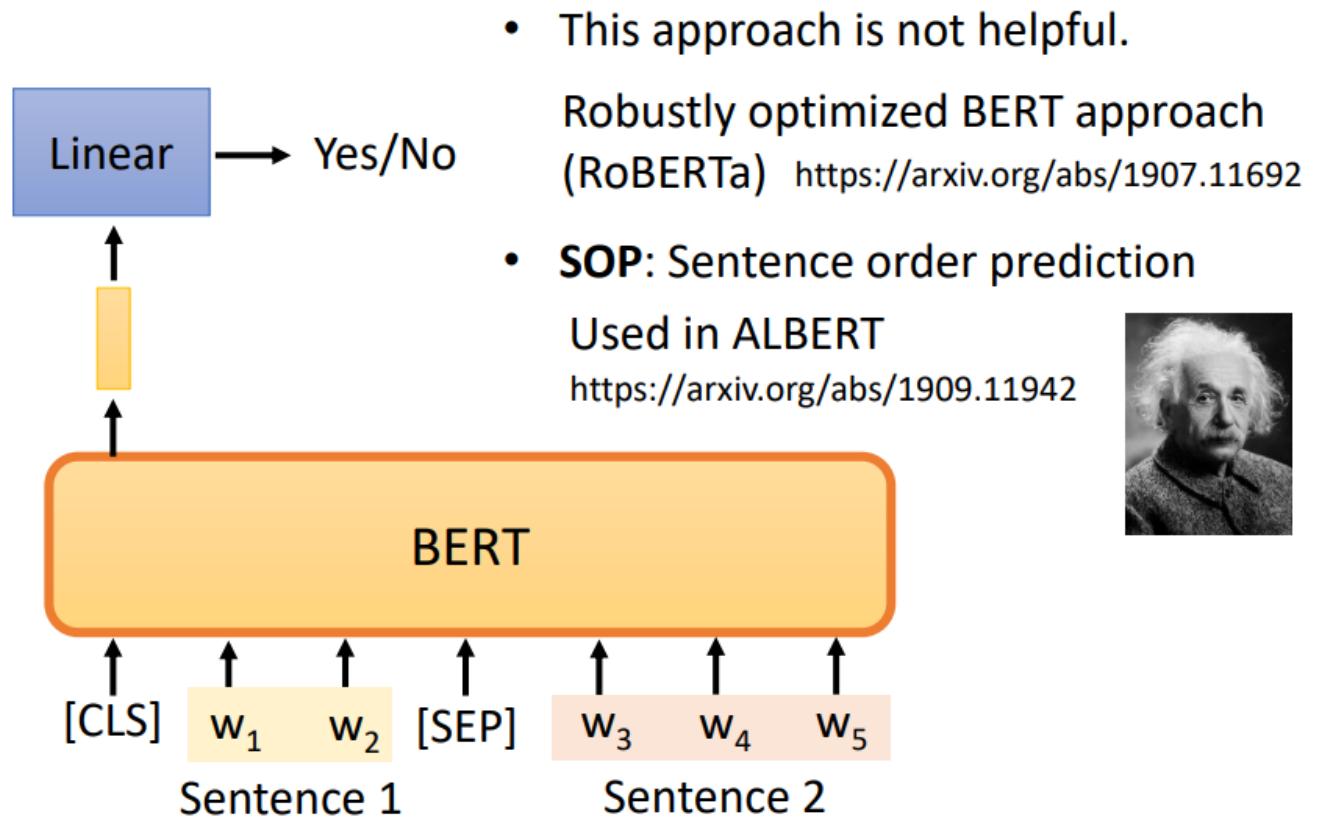
Transformer Encoder

Randomly masking some tokens

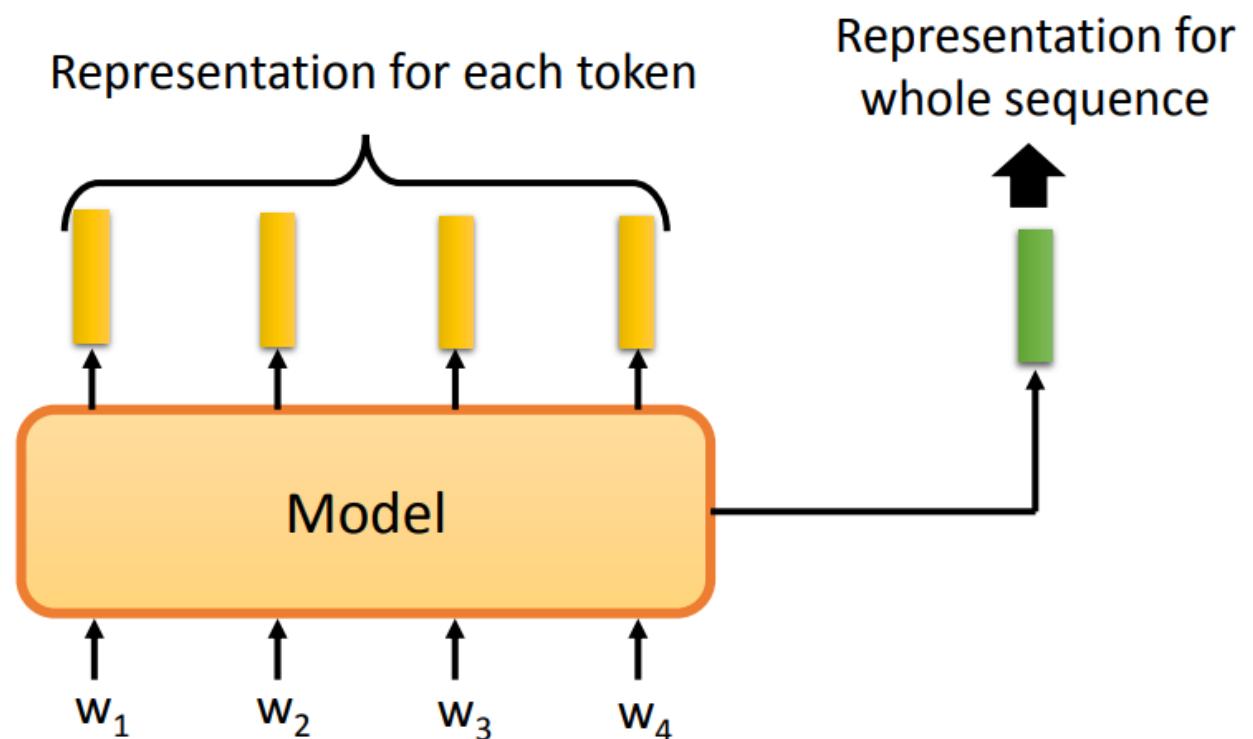


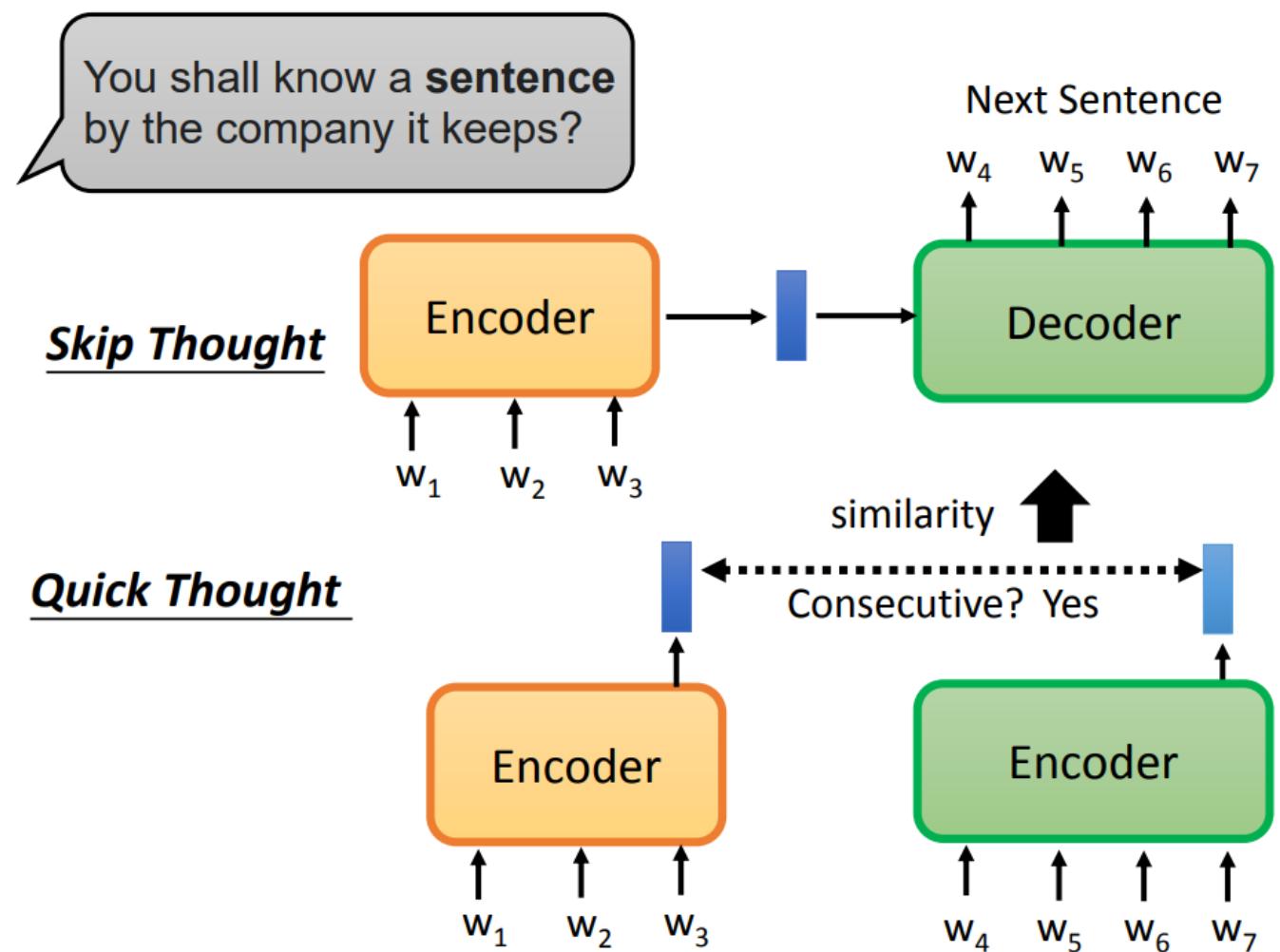
- Task 2: Next Sentence Prediction (NSP)
 - 后来有人提出NSP过于简单，实际不会对model有太多作用

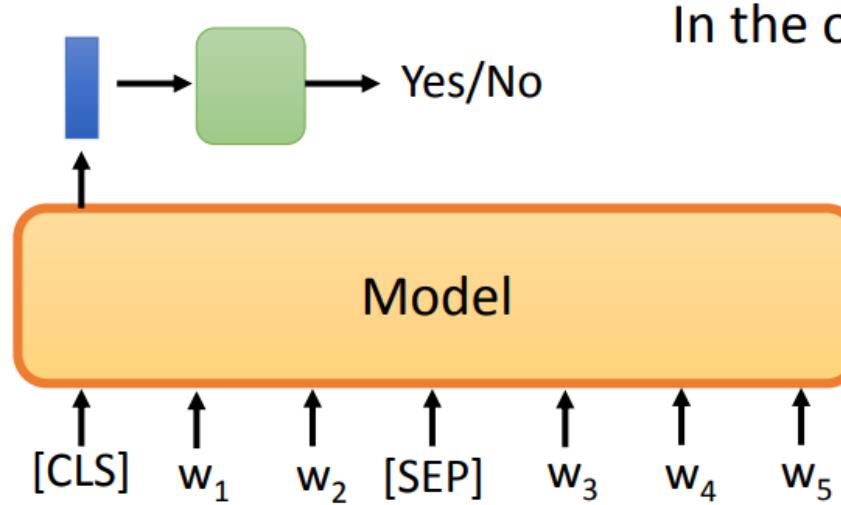
- RoBerta/ALBert用Sentence order prediction (SOP), 其正例与NSP相同, 但负例是通过选择一篇文档中的两个连续的句子并将它们的顺序交换构造的。这样两个句子就会有相同的话题, 模型学到的就更多是句子间的连贯性。
- NSP和SOP都可以用来做sentence embedding, 给出输入[CLS], 输出后链接其他layer(比如一层NN)获得sentence embedding



Sentence Level







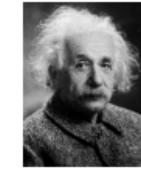
NSP: Next sentence prediction

Robustly optimized BERT approach (RoBERTa)

[Liu, et al., arXiv'19]

SOP: Sentence order prediction

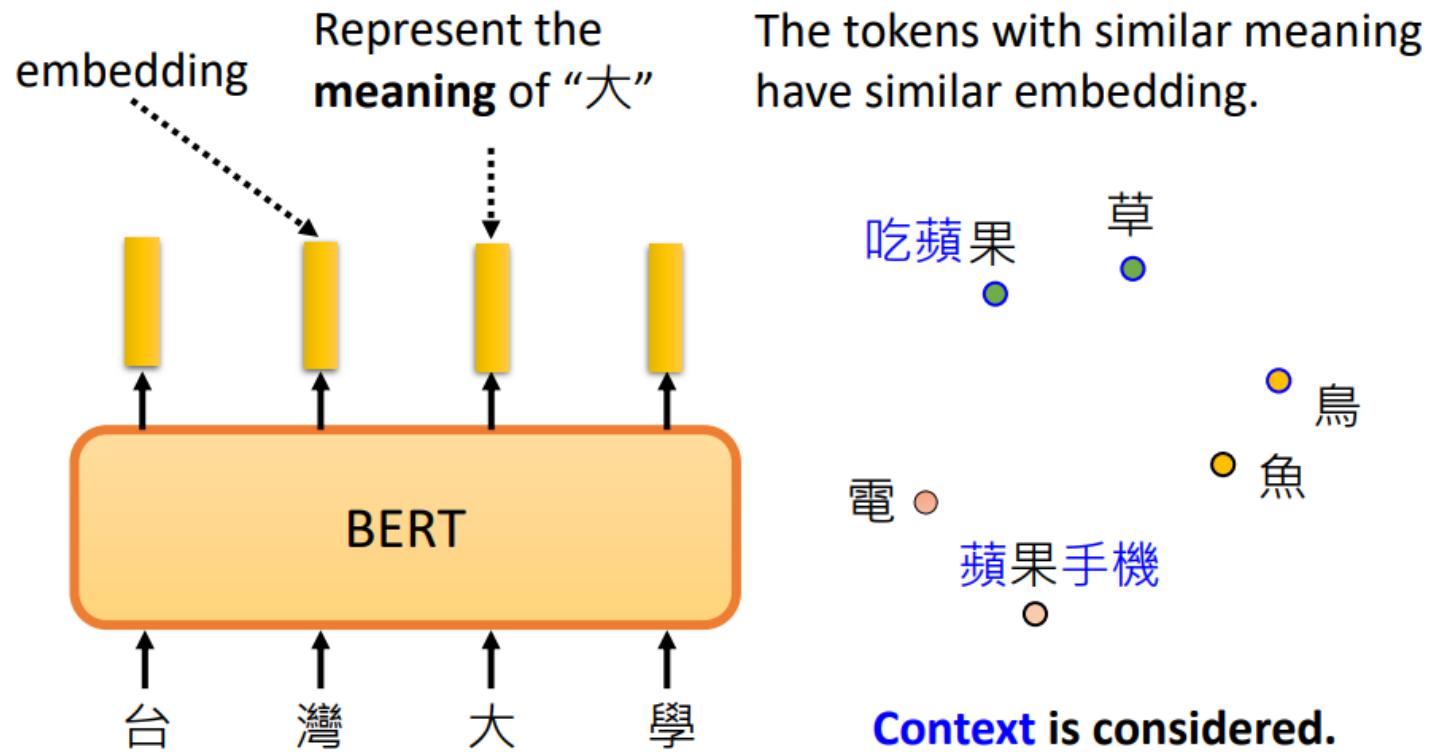
Used in ALBERT



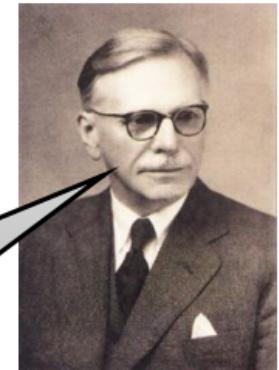
structBERT (Alice) [Want, et al., ICLR'20]

- 1.
- Why Bert works

- 语言学认为词的意义是跟经常和它一起出现的词有关。Bert的self-attention充分学习了词的上下文context信息，能catch住多义词在不同context下的语义



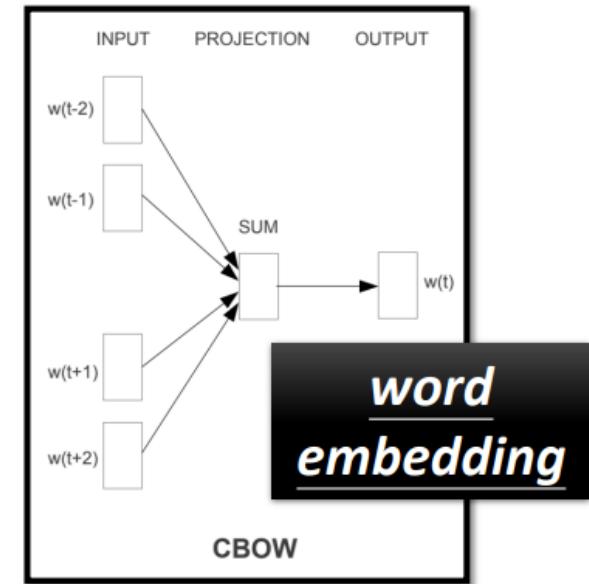
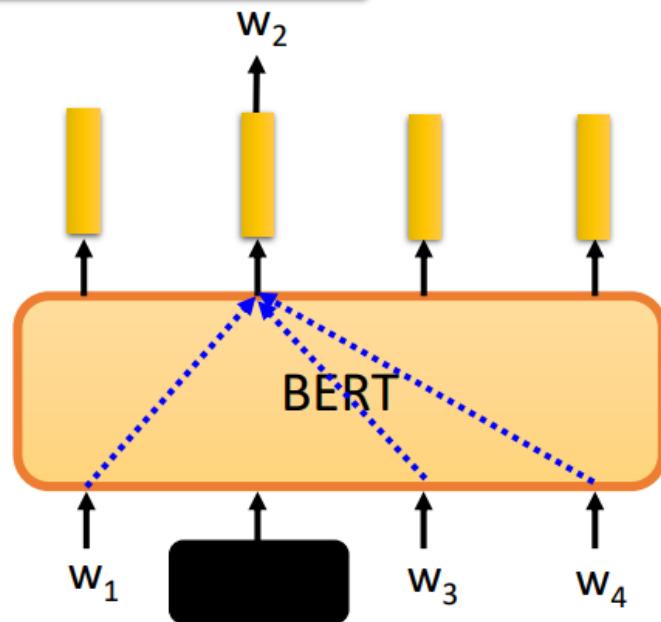
Why does BERT work?



You shall know a word by
the company it keeps

John Rupert Firth

Contextualized
word embedding



- Multi-lingual BERT
 - 为什么多语言在各自语言上学习能catch到跨语言的信息?

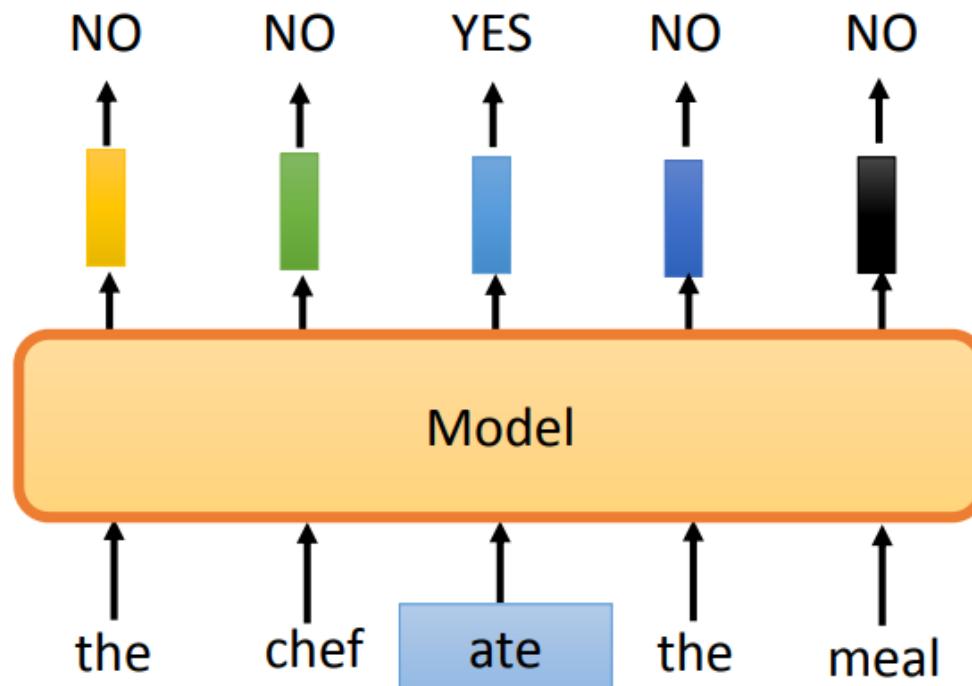
- 我的理解：每个语言有自己的pattern，语言之间pattern比较类似，导致不同语言可能有一定的mapping关系，视频里做了简单实验验证，把不同语言的所有embedding做average，算出差值x，在A语言的embedding加上差值x map到B语言，结果是有很大的相关性。另外多语言预训练（XLM）还可以加translation language model task.
- ELECTRA
 - 任务：给出某些词被替换过的input，预测每个词是否被替换掉。RTD（replaced token detection）任务。
 - 用一个small bert的预测mask的输出作为被替换的词，small bert效果不能太好，避免输出是完全正确的，这样ELECTRA才能从预测哪些词是替换的任务里面学到信息

Replace or Not?

Efficiently Learning an Encoder that Classifies Token Replacements Accurately (ELECTRA)

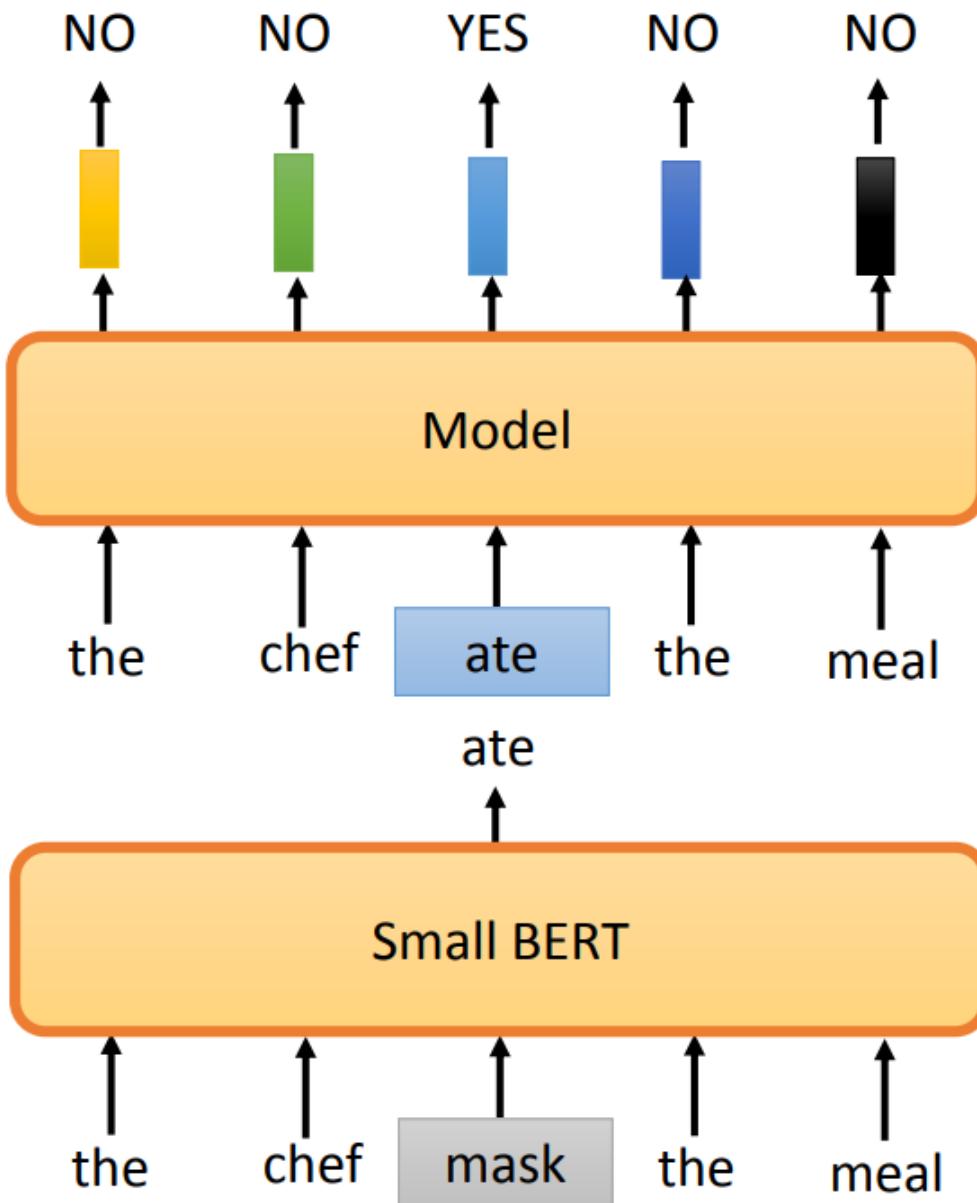


ELECTRA



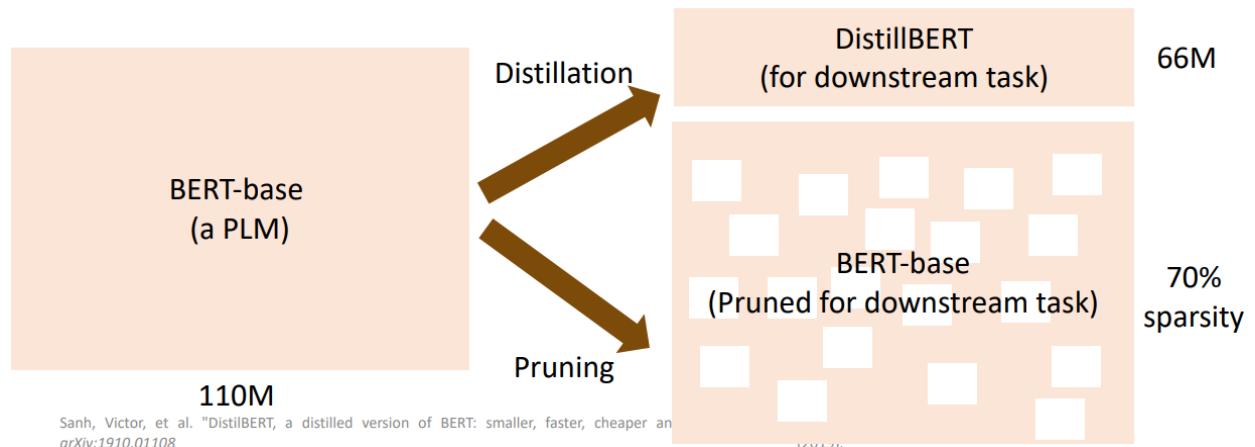
Predicting yes/not
is easier than
reconstruction.

Every output
position is used.

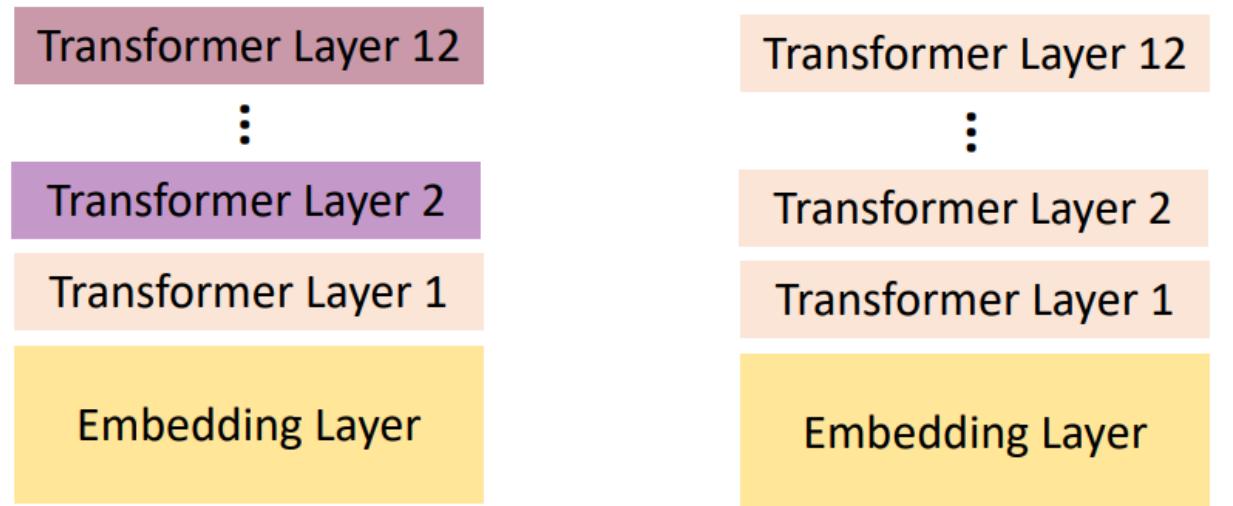


Note: This is
not GAN.

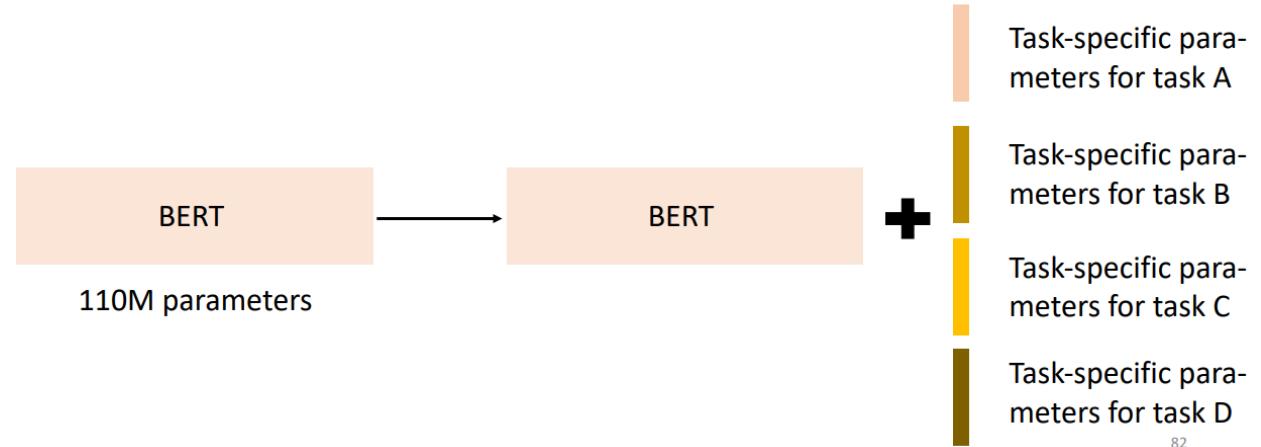
- PLM (Pre-trained Language Models)
 - PLM的问题
 - Data scarcity in downstream tasks
 - The PLM is too big, and they are still getting bigger
 - Need a copy for each downstream task, consume too much space
 - Inference takes too long
 - Data scarcity in downstream tasks 解决方案
 - Data-Efficient Fine-tuning, use natural language prompts and add scenario-specific designs
 - Prompt Tuning, QA设计问题, 怎么构造training data
 - Few-shot/Zero-shot Learning
 - Semi-supervised Learning
 - The PLM is too big, and they are still getting bigger 解决方案
 - 方法1: Reducing the Number of Parameters
 - Pre-train a large model, but use a smaller model for downstream tasks. Distillation or pruning



- Share the parameters among the transformer layers, 比如ALBERT



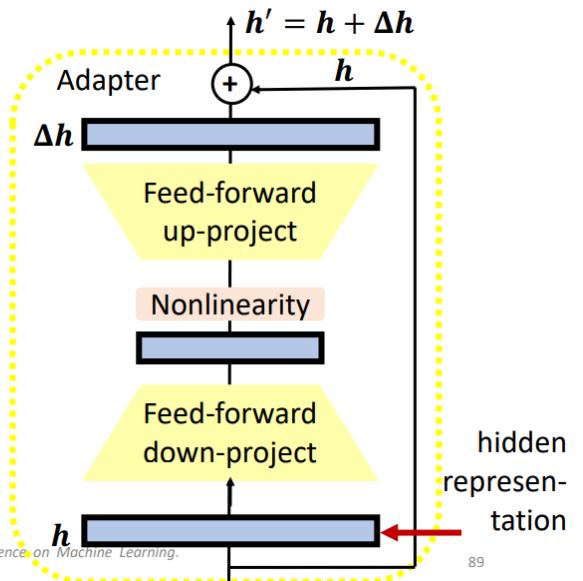
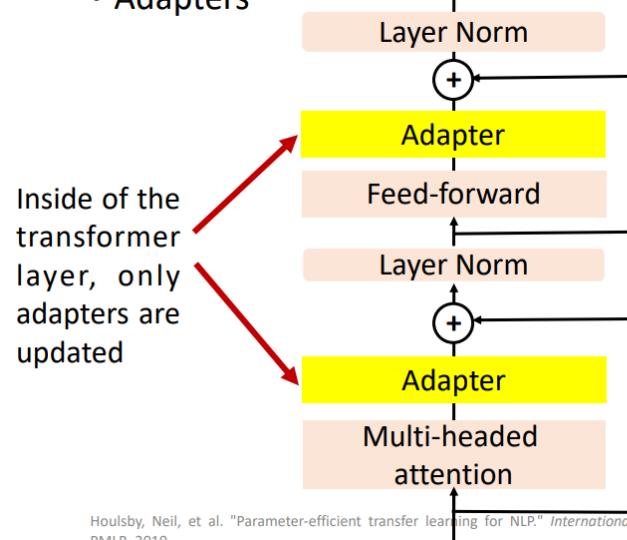
- 方法2: Parameter-Efficient Fine-tuning - Reduce the task-specific parameters in downstream task
 - Parameter-Efficient Fine-tuning Benefits
 - Drastically decreases the task-specific parameters
 - Less easier to overfit on training data; better out-of-domain performance
 - Fewer parameters to fine-tune; a good candidate when training with small dataset
 - 方法2.1: Use a small amount of parameters for each downstream task, 只 fine tuning task specific layers



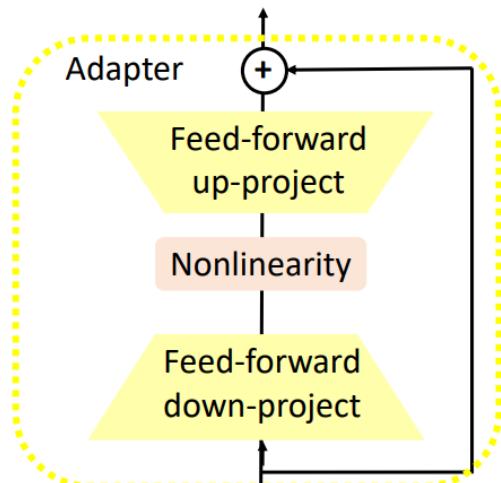
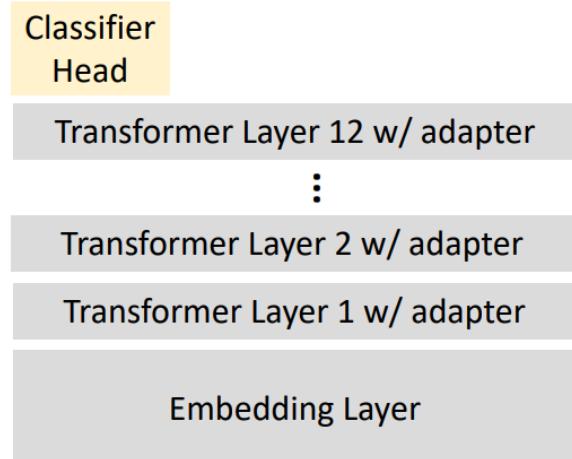
- 方法2.2: Adapter: Use special submodules to modify hidden representations

在pretrain model里面插入小的adapter module, 比如bert, 在每个transfomer里面加入小型的FFN结构。Fine tuning的时候, 只更新task specific head和adapter module的参数, pretrain model原始部分的参数保持不变。

- Adapters



- Adapters: During fine-tuning, only update the adapters and the classifier head



- 方法2.3: LoRA, Prefix Tuning, Soft Prompting: Insert trainable module in each layer. 个人感觉还是类似于上面的adapter的方法

- Benefit 1: Drastically decreases the task-specific parameters

	Adapter	LoRA	Prefix Tuning	Soft Prompt
Task-specific parameters*	$\Theta(d_{model}rL)$	$\Theta(d_{model}rL)$	$\Theta(d_{model}nL)$	$\Theta(d_{model}n)$
Percent Trainable	<5%	<0.1%	<0.1%	<0.05%
Illustration			n : Prefix length $k_{p_1} v_{p_1}$... $k_{p_n} v_{p_n}$	n : Prefix length

*not including the classifier head

109

◦

面试题讨论 Transfomer

1. Self-Attention 复杂度

1. Self-Attention时间复杂度: \$\$\$O(n^2 \cdot d)
2. \$\$\$, 其中n是input序列(sentence)的长度, d是token embedding的维度。包括三个步骤: 相似度计算, softmax 和加权平均。它们分别的时间复杂度是:

- 相似度计算可以看作大小为(n,d)和(d,n)的两个矩阵相乘: $\text{similarity}(n,d) * (d,n) = (n^2 \cdot d)$
 - softmax , 得到一个(n,n)的矩阵
 - softmax就是直接计算了, 时间复杂度为: $\text{softmax}(n^2)$
 - softmax
 - 加权平均可以看作大小为(n,n)和(n,d)的两个矩阵相乘: $\text{weighted average}(n,n) * (n,d) = (n^2 \cdot d)$
 - softmax , 得到一个(n,d)的矩阵
2. Transformer为何使用multi-head注意力机制（为什么不使用一个头）？为什么在进行多头注意力的时候需要对每个head进行降维？
1. 多头使参数矩阵形成多个子空间，保证了transformer可以注意到不同子空间的信息，捕捉到更加丰富的特征信息。
 3. 768×768 : 1套权重组合对 $V \in \mathbb{R}^{768 \times 768}$
 softmax 的最后一个维度做加权和
 $768 \times 64 \times 12$: 12套权重组合对 $V \in \mathbb{R}^{768 \times 64}$
 softmax 的最后一个维度做加权和，再把12套结果沿最后一个维度拼起来。因此不同的64维度的value向量得到了不同的组合方式。
将原有的高维空间转化为多个低维空间并再最后进行拼接，形成同样维度的输出，借此丰富特性信息，降低了计算量。
这里有一个容易引起误解的地方：多头”不是“加头”，它实际上是“分头”.用 12 组独立的 64 维的 attention，代替原来的单组 768 维的 attention。在全部计算完成后，得到的 12 个输出向量再拼起来，恢复回 768 的维度。
在体量上，768 维的“单头”完全可以看作是 12 组 64 维的“多头”。只不过，每一组的 attention score 都是一模一样的(单头相当于12组64维向量用了完全相同的attention score, 对应multi-head则是12组64维向量用了12个不同的attention score)。然而，在multi-head模式下，这 12 组就相当于 12 个特征空间，彼此分化，各自的 attention score 都是独有的(12组attention score)。“multi-head $768 \times 64 \times 12$ 与直接使用 768×768 矩阵统一计算的区别”，就在那里。虽然，输入向量、输出向量、隐层向量 (QKV) 都是 768 维的，计算过程也没有本质区别，但模型容量在 attention-score 处增加了。
 4. Transformer为什么Q和K使用不同的权重矩阵生成，为何不能使用同一个值进行自身的点乘？V的作用是什么？

1. 使用Q/K不相同可以保证在不同空间进行投影，增强了表达能力，提高了泛化能力。Q,K主要用来计算attention score, K和Q使用了不同的W_k, W_Q来计算，可以理解为是在不同空间上的投影。正因为有了这种不同空间的投影，增加了表达能力，这样计算得到的attention score矩阵的泛化能力更高。

从graph的角度来讲。attention score实际上可以表示edge上的权重。如果 Q K 是同一个，也就是self attention，然后W也是对称的，那么其实生成的attention score是无向图的邻接矩阵，但是WW如果不对称的话可以生成有向图的邻接矩阵，表达能力更强了。

如果不用Q，直接拿K和K点乘的话，你会发现attention score 矩阵是一个对称矩阵。因为是同样一个矩阵，都投影到了同样一个空间，所以泛化能力很差。这样的矩阵导致对V进行提纯的时候，效果也不会好。

2. V还代表着原来的句子，维度可以与Q,K 不同。
5. Transformer计算attention的时候为何选择点乘而不是加法？两者计算复杂度和效果上有什么区别？
6. Attention score目的是对V进行weighted sum, 计算attention score的方式有additive and multiplicative, 点乘计算更快，同时效果与Q,K的向量维度\$\$\$\$d_k
\$\$\$\$有关。
7. 为什么在进行softmax之前需要对attention进行scaled（除以\$\$\$\$\sqrt{d_k}
8. \$\$\$\$），并使用公式推导进行讲解
9. 假设 Q 和 K 的均值为0，方差为1。它们的矩阵乘积将有均值为0，方差为\$\$\$\$d_k
10. \$\$\$\$，因此除以\$\$\$\$\sqrt{d_k}

\$\$\$\$用于缩放来保证方差为1，原因是Q 和 K 的矩阵乘积的均值本应该为 0，方差本应该为1，这样可以获得更平缓的softmax。当维度很大时，点积结果会很大，会导致softmax的梯度很小。为了减轻这个影响，对点积进行缩放。

$$\text{Attention}(Q, K, V) = \text{softmax}_k \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

知乎 @Necther

11. 简单介绍一下Transformer的位置编码？有什么意义和优缺点？

因为self-attention是位置无关的，无论句子的顺序是什么样的，通过self-attention计算的token的hidden embedding都是一样的，这显然不符合人类的思维。因此要有一个办法能够在模型中表达出一个token的位置信息，transformer使用了固定的positional encoding来表示token在句子中的绝对位置信息。

12. 你还了解哪些关于位置编码 positional embedding 的技术，各自的优缺点是什么？

1. 绝对位置编码

- Learnable Positional Embedding 编码绝对位置: 直接对不同的位置随机初始化一个position embedding, 加到word embedding上输入模型, 作为参数进行训练。

2. 相对位置编码 (相对位置编码, RPE)

- Sinusoidal Position Encoding，使用正余弦函数表示绝对位置，通过两者乘积得到相对位置：这样设计的好处是位置的postional encoding可以被位置线性表示，反应其相对位置关系。

- \$\$\$\$\$PE_{\{(pos, 2i)\}} = \sin(pos/10000^{2i/d_{model}}))
 - \$\$\$\$\$, \$\$\$PE_{\{(pos, 2i+1)\}} = \cos(pos/10000^{2i/d_{model}}))

\$\$\$\$

- T5 (Text-to-Text Transfer Transformer)

- 在Attention矩阵的基础上加一个可训练的偏置项 $\beta_{i,j} = \text{softmax}(x_i)W_QW_K^T + \text{color{green}{\beta_{i,j}}}$
 - 不同与常规位置编码对将 $\beta_{i,j}$ 视为 $i-j$ 的函数并进行截断的做法，T5对相对位置进行了一个“分桶”处理，即相对位置是 $i-j$ 的位置实际上对应的是 $f(i-j)$ 位置，映射关系如下：

- \$\$\$\$\begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c|c|c} & & & & & & & & & & & & & & \end{array}
 - \hline
 - i - j & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
 - \hline
 - f(i-j) & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 8 & 8 & 8 & 9 & 9 & 9 & 9 & 9 \\
 - \hline

- $i - j \& 16 \& 17 \& 18 \& 19 \& 20 \& 21 \& 22 \& 23 \& 24 \& 25 \& 26 \& 27 \& 28 \& 29 \& 30 \& \cdots \\$
 - $\backslash hline$
 - $f(i-j) \& 10 \& 10 \& 10 \& 10 \& 10 \& 10 \& 11 \& 11 \& 11 \& 11 \& 11 \& 11 \& 11 \& \cdots \\$
 - $\backslash hline \backslash end\{array\}$
- \$\$\$\$

13. 简单讲一下Transformer中的残差结构以及意义

Encoder和decoder的self-attention层和ffn层都有残差连接。反向传播的时候mitigate梯度消失。

14. 为什么transformer块使用LayerNorm而不是BatchNorm? LayerNorm 在Transformer的位置是哪里?

1. 每个批次内句子的长度要一致，对于短的句子要进行padding，padding到该批次中最长句子的长度，导致有些嵌入是没有意义的。不应该让padding的内容影响到这一维度的分布。
2. layernorm抹掉了不同token间的分布差异，但是保留了一个token内不同特征之间的差异。nlp关注的是一个一个的token，自然不同token间的分布相同好。

LayerNorm用在multi-head attention layer和激活函数层之间。

CV使用BN是认为channel维度的信息对cv方面有重要意义，如果对channel维度也归一化会造成不同通道信息一定的损失。而同理nlp领域认为句子长度不一致，并且各个batch的信息没什么关系，因此只考虑句子内信息的归一化，也就是LN。同时multi-head attention之后会造成vector维度上数值的变化

15. Encoder端和Decoder端是如何进行交互的？（在这里可以问一下关于seq2seq的attention知识）

Decoder第二个multi-head attention模块，做cross attention (Encoder-Decoder Attention), Q from decoder, K,V from encode

16. Decoder阶段的多头自注意力和encoder的多头自注意力有什么区别？（为什么需要decoder自注意力需要进行 sequence mask）

Decoder有两层multi-head attention, encoder有一层multi-head attention, Decoder的第二层multi-head attention是为了转化输入与输出句长，Decoder的q可以与自身的k,v的倒数第二个维度可以不一样，但是需要和encoder的qk维度一样

17. Transformer的并行化体现在哪个地方？Decoder端可以做并行化吗？

训练的时候可以，但是inference的时候不可以

18. 简单描述一下wordpiece model 和 byte pair encoding

1. Word level (比如word2vec)

将句子拆分为词，即word-level.

- 优点
 - 能够保存较为完整的语义信息
- 缺点
 - 词汇表会非常大，大的词汇表对应模型需要使用很大的embedding层，这既增加了内存，又增加了时间复杂度。通常，transformer模型的词汇量很少会超过50,000，特别是如果仅使用一种语言进行预训练的话，而transformerxl使用了常规的分词方式，词汇表高达267735
 - word-level级别的分词略显粗糙，无法发现更加细节的语义信息，例如模型学到的“old”，“older”，and “oldest”之间的关系无法泛化到“smart”，“smarter”，and “smartest”
 - word-level级别的分词对于拼写错误等情况的鲁棒性不好
 - oov问题不好解决

2. Char level (比如dssm)

- 优点
 - 这可以大大降低embedding部分计算的内存和时间复杂度，以英文为例，英文字母总共就26个。中文常用字也就几千个。
 - char-level的文本中蕴含了一些word-level的文本所难以描述的模式，因此一方面出现了可以学习到char-level特征的词向量FastText，另一方面在有监督任务中开始通过浅层CNN、HighwayNet、RNN等网络引入char-level文本的表示；
- 缺点

- 任务的难度大大增加了，毕竟使用字符大大扭曲了词的意义，一个字母或者一个单中文字实际上并没有任何语义意义，单纯使用char-level往往伴随着模型性能的下降；
- 增加了输入的计算压力，原本”I love you“是3个embedding进入后面的cnn、rnn之类的网络结构，而进行char-level拆分之后则变成8个embedding进入后面的cnn或者rnn之类的网络结构，这样计算起来非常慢

3. Sub-word level

subword-level的分词方式遵循的原则是：尽量不分解常用词，而是将不常用词分解为常用的子词。

subword的分词往往包含了两个阶段，一个是encode阶段，形成subword的vocabulary dict，一个是decode阶段，将原始的文本通过subword的vocabulary dict转化为token的index然后进入embedding层。

▪ Byte-Pair Encoding (BPE)

- 对每个词进行词频统计，并且对每个词末尾的字符转化为末尾字符和</w>两个字符，停止符"</w>"的意义在于表示subword是词后缀。
- 统计每一个连续字节对的出现频率，选择最高频者合并成新的subword
- 继续上述过程，继续迭代直到达到人工预设的subword词表大小或下一个最高频的字节对出现频率为1
 - 优点
 - 可以有效地平衡词汇表大小和步数(编码句子所需的token数量)。
 - 缺点
 - 基于greedy和确定的符号替换，不能提供带概率的多个分片结果(相对于unigram来说)，最终会导致decode的时候面临含糊不清的问题。

NUMBER	SUBWORDS (_ means space)	TEXT
1	de/ep/_le/ar/n/ing	deep learning
2	d/ee/p/_le/ar/n/ing	deep learning
3	d/e/ep/_le/ar/n/ing	Deep learning

▪ Wordpiece

- wordpiece是一种介于BPE和unigram中的一种分token的算法,整体的计算思路和BPE类似,仅仅在生成subword的时候不同
 - Bpe中, 统计每一个连续字节对的出现频率, 选择最高频者合并成新的subword
 - Wordpiece则使用了概率相除的方法, 这里和决策树的分裂过层非常类似, 两个两个相邻字符或subword之间进行分裂判断分裂增益是否增大, 增大则合并。\$\$\$\$ $\log P(t_{\text{merge}}) - (\log P(t_x) + \log P(t_y)) = \log(\frac{P(t_{\text{merge}})}{P(t_x)P(t_y)})$
 - \$\$\$)

19. Transformer 是如何训练的? 测试阶段如何进行测试呢?

1. Training

Transformer 训练过程与 seq2seq 类似, 首先 Encoder 端得到输入的 encoding 表示, 并将其输入到 Decoder 端做交互式 attention, 之后在 Decoder 端接收其相应的输入, 经过多头 self-attention 模块之后, 结合 Encoder 端的输出, 再经过 FFN, 得到Decoder 端的输出之后, 最后经过一个线性全连接层, 就可以通过 softmax 来预测下一个单词(token), 然后根据 softmax 多分类的损失函数, 将 loss 反向传播即可, 所以从整体上来说, Transformer 训练过程就相当于一个有监督的多分类问题。

需要注意的是, 「Encoder 端可以并行计算, 一次性将输入序列全部 encoding 出来, 但 Decoder 端不是一次性把所有单词(token)预测出来的, 而是像 seq2seq 一样一个接着一个预测出来的。」

2. Testing

- 在测试的时候, 是sequential的生成第一个位置的输出, 然后有了这个之后, 第二次预测时, 再将其加入输入序列, 以此类推直至预测结束。

20. Transformer 如何并行化的?

Transformer 的并行化主要体现在 self-attention 模块, 在Encoder端Transformer可以并行处理整个序列, 并得到整个输入序列经过Encoder端的输出。

21. Transformer中Self-attention后为什么要加前馈网络FFN

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。

类比cnn网络中, cnn block和fc交替连接, 效果更好。相比于单独的multi-head attention, 在后面加一个ffn, 可以提高整个block的非线性变换的能力。

22. Transformer 在哪里做了parameter/weight sharing

1. Encoder 和 Decoder 间的token Embedding层权重共享 (BPE/Wordpiece 对应词表的embedding)
2. Decoder 中 token Embedding 层和 Full Connect(最后反向预测单词) 层权重共享

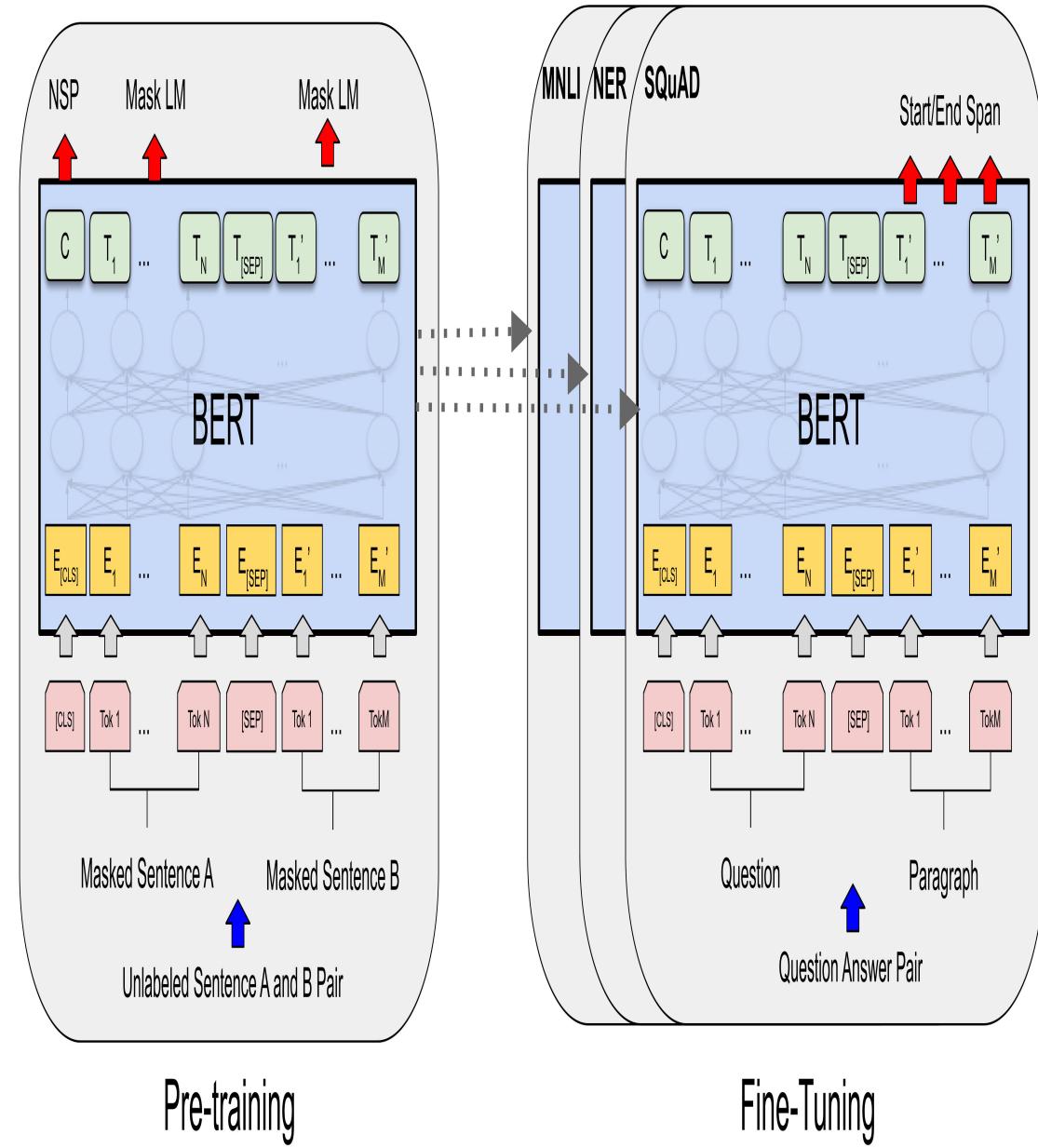
BERT

1. BERT的模型参数量

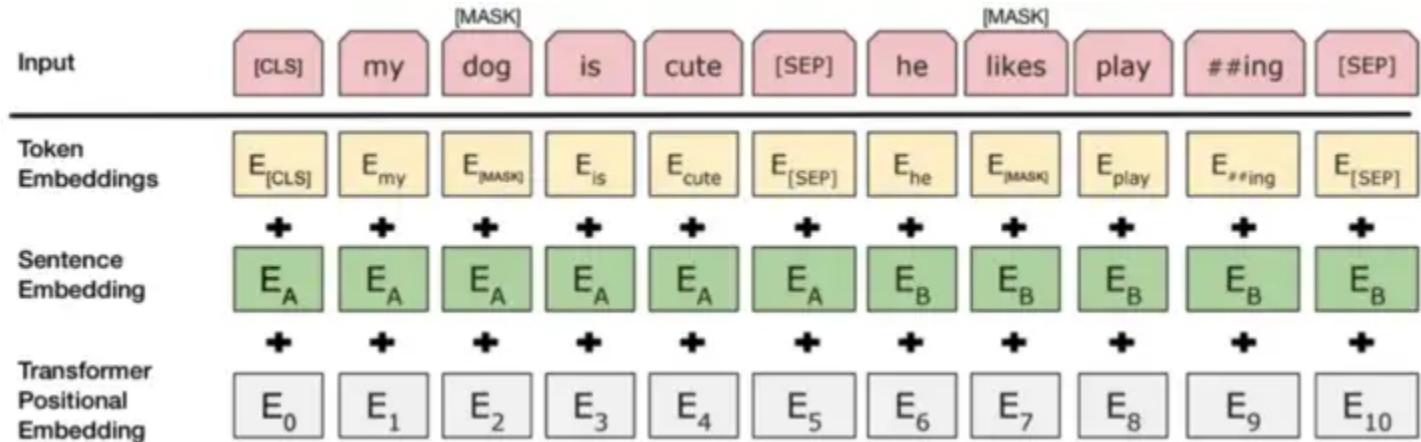
1. BERT-Base, 12层 transformer layer(only encoder), 768个隐单元, 12个Attention head, 110M参数
 - vocab_size=30522, hidden_size=768, max_position_embeddings=512, token_type_embeddings=2
 - Token Embeddings: 总词汇是30522每个输出维度都是768, 参数量是 30522×768
 - Position Embeddings: transformer中位置信息是通过sincos生成, 但是在bert中是学出来了 (原文中说的应该是的数据量足, 能学出来) 最大长度是512所以这里参数量是 512×768
 - Segment Embeddings: 用1和0表示, 所以参数是 2×768
2. BERT-Large, 24层 transformer layer(only encoder), 1024个隐单元, 16个head, 340M参数

2. Bert model架构

1. 整体pretrain + fine tuning



2. Input embedding



3. 为什么bert有三个嵌入层, 它们是如何实现的

Token Embeddings, Segment Embeddings, Position Embeddings

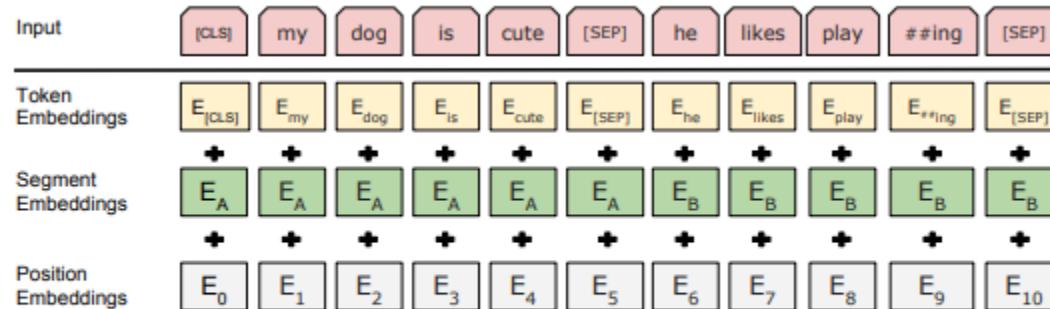


Figure 2: BERT input representation. The input embeddings is the sum of the token embeddings, the segmentation embeddings and the position embeddings.

4. Bert 的是怎样预训练的

1. Task 1: MLM (mask language modeling): 将完整句子中的部分字 mask, 预测该 mask 词

2. Task 2: NSP (Next Sentence Prediction): 为每个训练前的例子选择句子 A 和 B 时，50% 的情况下 B 是真的在 A 后面的下一个句子，50% 的情况下是来自语料库的随机句子，进行二分预测是否为真实下一句
5. Bert 为什么做mask，具体是怎么做的，这么做的原因和意义是什么
 1. 在数据中随机选择 15% 的标记，其中 80% 被换为 [MASK]，10% 不变、10% 随机替换其他单词
 - 80% 的 token 被替换为 [MASK]。意义：在不泄露 label 的情况下，通过被 mask 的单词的上下文，预测该单词，训练模型和词向量。问题：下游任务中不会出现 [MASK]，导致预训练和微调不一致
 - 10% 的 token 会替换为随机的 token。意义：减弱了预训练和微调不一致给模型带来的影响。同时让模型知道，当单词不是 [MASK] 标记时，仍然需要输出，就迫使模型尽量在每一个 token（即使该词不是 mask）上都学习基于上下文的表示，这也是解决一词多义的关键。也类似加 noise 使模型具有根据上下文理解纠错的能力，更 robust。问题：模型可能会认为要预测的词永远不会是该位置原本的词。
 - 10% 的 tokens 会保持不变但需要被预测：意义：让模型知道要预测的词有可能是该位置原本的词，而不是永远都是随机单词。
 - 如果没有 mask，直接用交叉熵损失的训练数据预测输入序列的每个单词，学习任务是微不足道的。该网络事先知道它需要预测什么，因此它可以很容易地学习权值。（知道输入什么词，还要预测该词）
 - 加入 [MASK] 可以在不泄露 label 的情况下融合真双向语义信息。另外在后续微调任务中语句中并不会出现 [MASK] 标记，造成 train test 不一致。
 - 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。真的有 10% 的情况下是泄密的（占所有词的比例为 $15\% * 10\% = 1.5\%$ ），这样能够给模型一定的 bias，相当于额外的奖励，将模型对于词的表征能够拉向词的真实表征。
 - 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。
2. Decoder 的 mask

Transformer 模型的 decoder 层存在 mask，这个 mask 的作用是在翻译预测的时候 mask 掉句子后面的部分，防止模型看到“答案”
6. Bert 的 loss function?

1. Bert loss function 组成:

- 第一部分是来自 Mask-LM 的单词级别分类任务
 - 被 mask 的词集合为 M, 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:

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

关于NLP那些你不知道的事

- 另一部分是句子级别的分类任务, NSP(Next Sentence Prediction)分类任务

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

关于NLP那些你不知道的事

- 总loss function为

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

关于NLP那些你不知道的事

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

关于NLP那些你不知道的事

注: 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

2. 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。

7. Bert Tokenizer

1. BasicTokenizer: 根据空格等进行普通的分词

- 包括了一些预处理的方法：去除无意义词，跳过\t这些词，unicode变换，中文字符筛选等等

2. WordpieceTokenizer: 前者的结果再细粒度的切分为WordPiece

8. 长文本预测如何构造Tokens?

1. head-only: 保存前 510 个 token (留两个位置给 [CLS] 和 [SEP])
2. tail-only: 保存最后 510 个token
3. head + tail : 选择前128个 token 和最后382个 token (文本在800以内) 或者前256个token+后254个token (文本大于800tokens)
4. Longformer

9. bert流程是怎么样的

1. 首先定义处理好输入的tokens的对应的id作为input_id,因为不是训练所以input_mask和segment_id都是采取默认的1即可
2. 在通过embedding_lookup把input_id向量化，如果存在句子之间的位置差异则需要对segment_id进行处理，否则无操作；再进行position_embedding操作
3. 进入Transform模块，后循环调用transformer的前向过程，次数为隐藏层个数，每次前向过程都包含self_attention_layer、add_and_norm、feed_forward和add_and_norm四个步骤
4. 输出结果为句向量则取[cls]对应的向量（需要处理成embedding_size），否则也可以取最后一层的输出作为每个词的向量组合all_encoder_layers[-1]

10. 为什么要在Attention后使用残差结构

Residual network结构能够很好的消除层数加深所带来的gradient vanishing问题, 还可以平滑error surface, 使优化变得容易

11. BERT 的embedding向量如何得来的

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，加上position embedding 和 segment embedding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。

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，而在里代码实现中是跟普通 word embedding 一样随机生成的，可以训练的。

12. BERT mask 相对于 CBOW 有什么异同点

1. 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。

2. 不同点

- 在 CBOW 中，每个单词都会成为 input word，并且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。
- 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

13. BERT训练时使用的学习率 warm-up 策略是怎样的

相当于对学习率自适应的一个过程。因为模型一开始参数迭代的方向比较重要，所以在开始训练的时候，避免部分噪声样本把参数更新方向带偏，所以开始训练的时候会设置一个warm-up步数，当前步的学习率和当前步数成正比，即学习率为 $(\text{current_step}/\text{warm_up_step}) * \text{learning_rate}$

刚开始模型对数据的“分布”理解为零，或者是说“均匀分布”（当然这取决于你的初始化）；在第一轮训练的时候，每个数据点对模型来说都是新的，模型会很快地进行数据分布修正，如果这时候学习率就很大，极有可能导致开始的时候就对该数据“过拟合”，后面要通过多轮训练才能拉回来，浪费时间。当训练了一段时间（比如两轮、三轮）后，模型已经对每个数据点看过几遍了，或者说对当前的batch而言有了一些正确的先验，较大的学习率就不那么容易会使模型学偏，所以可以适当调大学习率。这个过程就可以看做是warmup。那么为什么之后还要decay呢？当模型训到一定阶段后（比如十个epoch），模型的分布就已经比较固定了，或者说能学到的新东西就比较少了。如果还沿用较大的学习率，就会破坏这种稳定性，用我们通常的话说，就是已经接近loss的local optimal了，为了靠近这个point，我们就要慢慢来。

14. BERT的优缺点

1. 优点

- Bert通过self attention真正的学到了bidirectional的信息，解决长时依赖问题，特征提取能力强，有效捕获上下文的全局信息

- BERT加入了Next Sentence Prediction来和Masked-LM一起做联合训练，可以获取比词更高级别的句子级别的语义表征。
- 可并行计算
- BERT 模型是将预训练模型和下游任务模型结合在一起的，做下游任务时仍然是用BERT模型，不需要对模型做修改，微调成本小
- 可以解决一词多义问题：同一个词在转换为BERT的输入之后，embedding的向量是一样的，但是通过BERT中的多层transformer encoder之后，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。

2. 缺点

- 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，只将80%的词替换为[mask]，但这也只是缓解、不能解决。另外也没有考虑预测[MASK]之间的相关性，是对语言模型联合概率的有偏估计
- 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。
- 静态MASK问题。BERT的MASK方式为静态MASK，即15%的token一旦选择就不再改变，也就是说一开始随机选择了15%的token进行MASK处理，在接下来的N个epoch阶段中不再改变（以后的轮次一直预测这些MASK）。
- NSP任务过于简单，取另一个文章的句子作为第二句，只需要分清主题即可，而不需要知道语义，对模型训练帮助不大。多SP之所以没用是因为这个任务不仅包含了句间关系预测，也包含了主题预测（“topic prediction”和“coherence prediction”），而主题预测显然更简单些（比如一句话来自新闻财经，一句话来自文学小说），模型会倾向于通过主题的关联去预测。
- 生成任务表现不佳，因为只用了transformer的encoder，并且预训练过程和生成过程的不一致
- self-attention计算量较大，复杂度为\$\$\$\$d_k^2
- \$\$\$

15. 如何优化BERT性能

1. Model compression by knowledge distillation
2. Albert优化 [Google ALBERT原理讲解](#)

1. Factorized Embedding Parameterization

在BERT、XLNet、RoBERTa中，词表的embedding size(E)和transformer层的hidden size(H)都是相等的，这个选择有两方面缺点：

1. 从建模角度来讲，wordpiece向量应该是不依赖于当前内容的(context-independent)，而transformer所学习到的表示应该是依赖内容的。所以把E和H分开可以更高效地利用参数，因为理论上存储了context信息的H要远大于E。
2. 从实践角度来讲，NLP任务中的vocab size本来就很大，如果E=H的话，模型参数量就容易很大，而且embedding在实际的训练中更新地也比较稀疏。

作者认为没有这个必要，因为预训练模型主要的捕获目标是H所代表的“上下文相关信息”而不是E所代表的“上下文无关信息”。Factorized就是在词表V到隐层H的中间，插入一个小维度E，多做一次尺度变换： $\text{$$$$O(V \times E + E \times H)}$
\$\$\$\$，减少了参数量

2. Cross-layer parameter sharing

跨层参数共享，就是不管12层还是24层transformer，它们都共享transformer所有参数

3. Sentence Order Prediction (SOP) Inter-sentence coherence loss

很多研究(XLNet、RoBERTa)都发现next sentence prediction没什么用处，所以作者也审视了一下这个问题，认为NSP之所以没用是因为这个任务不仅包含了句间关系预测，也包含了主题预测(“topic prediction”和“coherence prediction”)，而主题预测显然更简单些(比如一句话来自新闻财经，一句话来自文学小说)，模型会倾向于通过主题的关联去预测。因此换成了SOP(sentence order prediction)，预测两句话有没有被交换过顺序，这样模型可以学到更多的信息量。

4. ELECTRA RTD(Replaced token detection)任务

16. 为什么BERT在第一句前会加一个[CLS]标志？

1. 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。
2. The first token of every sequence is always a special classification token [CLS]. The final hidden state corresponding to this token is used as the aggregate sequence representation for classification tasks. (用来预测NSP next sentence)

prediction的分类)。无明显语义信息的符号会更“公平”地融合文本中各个词的语义信息，从而更好的表示整句话的语义。

具体来说，self-attention是用文本中的其它词来增强目标词的语义表示，但是目标词本身的语义还是会占主要部分的，因此，经过BERT的12层，每次词的embedding融合了所有词的信息，可以去更好的表示自己的语义。而[CLS]位本身没有语义，经过12层，得到的是attention后所有词的加权平均，相比其他正常词，可以更好的表征句子语义。当然，也可以通过对最后一层所有词的embedding做pooling去表征句子语义。

17. Bert encoder和transformer encoder的区别

1. 在输入端增加segment embedding和position embedding
2. 不再使用三角函数的方法生成，而是采用learnable embedding table

18. Bert 的 CLS 能够有效的表达 Sentence Embeddings 吗？

1. CLS本为NSP而生，在用于非NSP任务上表现可能不好
2. 实际应用中，Averaging all tokens is better than only using [CLS] vector. (Paper: WhiteningBERT: An Easy Unsupervised Sentence Embedding Approach)

19. 为什么BERT选择mask掉15%这个比例的词，可以是其他的比例吗？

20. 从CBOW的角度，这里 \$\$\$\$p=15\%
21. \$\$\$\$有一个比较好的解释是：在一个大小为 \$\$\$\$frac{1}{p}=\frac{100}{15}\approx7
\$\$\$\$的窗口中随机选一个词，类似CBOW中滑动窗口的中心词，区别是这里的滑动窗口是非重叠的。从CBOW的滑动窗口角度 10%~20%都是还ok的比例。

22. 使用BERT预训练模型为什么最多只能输入512个词，最多只能两个句子合成一句？

1. 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。在BERT config中: "max_position_embeddings" : 512, "type_vocab_size": 2, 因此，在直接使用Google的BERT预训练模型时，输入最多512个词（还要除掉[CLS]和[SEP]），最多两个句子合成一句。这之外的词和句子会没有对应的embedding。
2. 当然，如果有足够的硬件资源自己重新训练BERT，可以更改 BERT config，设置更大的 max_position_embeddings 和 type_vocab_size值去满足自己的需求。

23. 在BERT应用中，如何解决长文本问题？

1. 这个问题现在的解决方法是用Sliding Window（划窗），主要见于诸阅读理解任务（如Stanford的SQuAD）。
Sliding Window即把文档分成有重叠的若干段，然后每一段都当作独立的文档送入BERT进行处理。最后再对于这些独立文档得到的结果进行整合。Sliding Window可以只用在Training中。因为Test之时不需要Back Propagation，亦不需要large batchsize，因而总有手段将长文本塞进显存中（如torch.no_grad, batchsize=1）
2. 抽取重要片段。抽取长文本的关键句子作为摘要，然后进入BERT。
3. 直接截断。从长文本中截取一部分，具体截取哪些片段需要观察数据，如新闻数据一般第一段比较重要就可以截取前边部分；
4. Transformer-xl 等模型

24. BERT是如何区分一词多义的

同一个词在转换为BERT的输入之后，embedding的向量是一样的，但是通过BERT中的多层transformer encoder之后，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。

25. BERT中的词表是怎么来的

Wordpiece算法构建的，BERT也用该算法进行分词，WordPiece算法可以看作是BPE的变种。不同点在于，WordPiece基于概率生成新的subword而不是最高频字节对。