Transformer 之模型细解

背景

在 2018 年，要说自然语言处理领域较为轰动的成果话，莫非 Google 的 Bert 模型，霸占各个竞赛排名，这就犹如当年的霆锋一首《谢谢你的爱 1999》霸占各大音乐排行榜一样。下图是斯坦福大学问答测试语料的排行。从图可一窥究竟，Bert 模型是何等的风行天下。

1 Mar 20, 2019	BERT + DAE + AoA (ensemble) *Joint Laboratory of HIT and iFLYTEK Research*	**87.147**	**89.474**
2 Mar 15, 2019	BERT + ConvLSTM + MTL + Verifier (ensemble) *Layer 6 AI*	86.730	89.286
3 Mar 05, 2019	BERT + N-Gram Masking + Synthetic Self-Training (ensemble) *Google AI Language* https://github.com/google-research/bert	86.673	89.147
4 Apr 13, 2019	SemBERT(ensemble) *Shanghai Jiao Tong University*	86.166	88.886
5 Mar 16, 2019	BERT + DAE + AoA (single model) *Joint Laboratory of HIT and iFLYTEK Research*	85.884	88.621
6 Mar 05, 2019	BERT + N-Gram Masking + Synthetic Self-Training (single model) *Google AI Language* https://github.com/google-research/bert	85.150	87.715
7 Jan 15, 2019	BERT + MMFT + ADA (ensemble) *Microsoft Research Asia*	85.082	87.615

那么，我们一直要说的是 Transformer，为啥要提到 Bert 呢？Bert 简单来说就是 Google 基于 Transformer 提供的预训练模型。那各位看官可能会问，啥是预训练模型呢？预训练模型还真是一句两句不能说的清楚，举个例子吧，一般大公司根据自己的垄断地位，掌握了大量的语料，小公司市场所限，掌握的语料相对较少，特别是在深度神经网络流行的今天，没有巨大的语料库，想训练一个好深度的网络模型，

基本不可能的。那么，有人就提出了，针对相似的应用场景，大公司把训练好的深度网络模型结构及参数开放出来，小公司根据自己需要进行微调结构与参数，这样，小公司也能分享深度网络带来的好处，那么这样的方式就叫迁移学习，这模型也就是预训练模型。

本文受篇幅所限，需要看官准备以下知识，神经网络、LSTM、注意力等知识。本文在编写过程中，采用了大量的网络资料，主要感觉自己文字能力有限，加上有些资料写的确实已经无比清晰，所采用的资料已经在文章最后进行注明。

Transformer **模型**

先直接给出 Transformer 模型图。

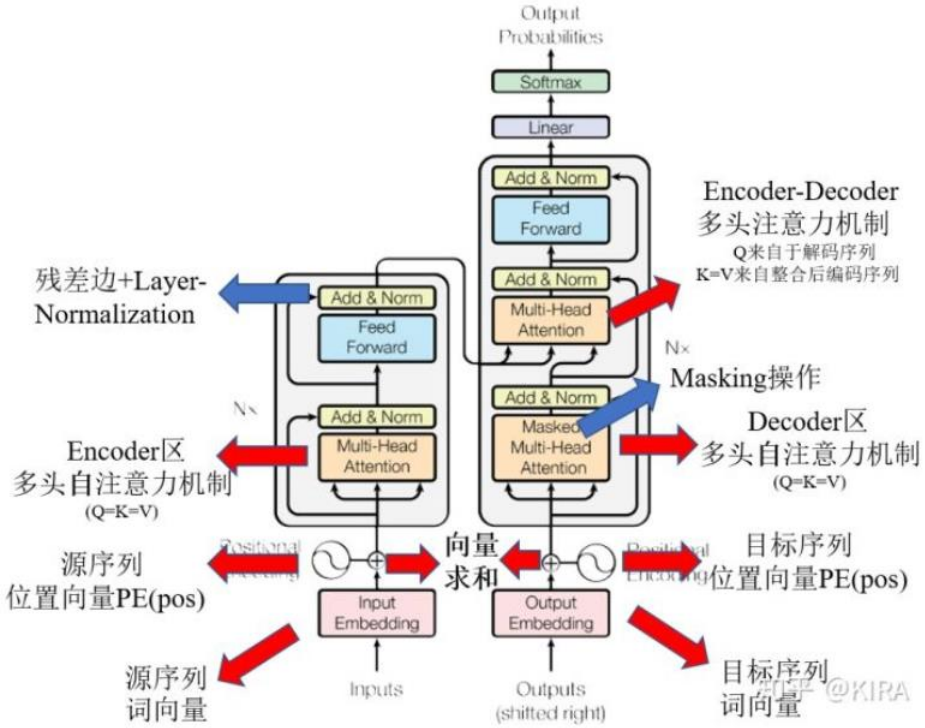


图2 Transformer示意

我相信，这图一看，就跟我第一次看到这个结构一样，一脸懵圈。和经典的 seq2seq 模型一样，Transformer 模型中也采用了 encoer-decoder 架构。上图的，左半边用 **NX** 框出来的，就代表一层 encoder，

其中论文里面的 encoder 一共有 6 层这样的结构。上图的右半边用 **NX** 框出来的，则代表一层 decoder，同样也有 6 层。定义输入序列首先经过 word embedding，再和 positional encoding 相加后，输入到 encoder 中。输出序列经过的处理和输入序列一样，然后输入到 decoder。最后，decoder 的输出经过一个线性层，再接 Softmax。这里，简述下 Encoder、Decoder 的各个组成部分。

Encoder 由 6 层相同的层组成，每一层分别由两部分组成：

第一部分是 multi-head self-attention

第二部分是 position-wise feed-forward network，是一个全连接层

两个部分，都有一个残差连接(residual connection)，然后接着一个 Layer Normalization。

Decoder 也是由 6 个相同的层组成，每一个层包括以下 3 个部分:

第一个部分是 multi-head self-attention mechanism

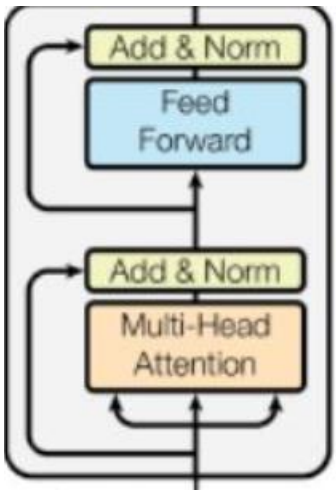
第二部分是 multi-head context-attention mechanism

第三部分是一个 position-wise feed-forward network

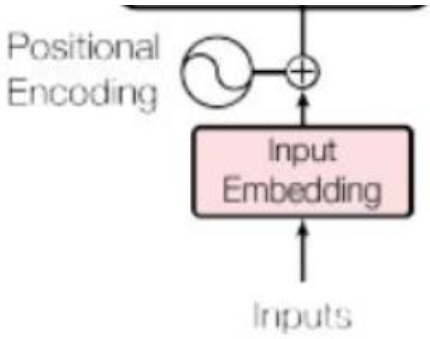
和 encoder 一样，上面三个部分的每一个部分，都有一个残差连接，后接一个 **Layer Normalization。**

decoder 和 encoder 不同的地方在 multi-head context-attention mechanism。

但从图可以看出，存在一个共同的结构，那就是



那么，我们就一步一步解答这个结构的计算方式。但是，在说明这个结构之前，我们还是要先说明下输入部分的结构，即下图



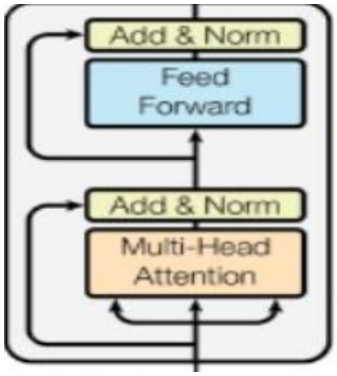
首先，输入的是词向量，这个部分已经在上文中进行了说明，就不再介绍。这里说明的是 Positional Encoding 部分。Positional Encoding 其实是对词向量加入了位置信息。

对于句子中的每一个字，其位置 pos∈[0,1,2,...,9](每句话 10 个字),每个字是 N（512）维向量，维度 i （i∈[0,1,2,3,4,..N]）带入函数

$$f(pos,i)=\frac{pos}{10000^{\frac{i}{N}}}$$

对于 i 为偶数，用 sin 函数进行计算，如果 i 为奇数，用 cos 函数进行计算。把一个字的词向量中的所有数值，进行该计算，计算后的数值输入到下一层中。

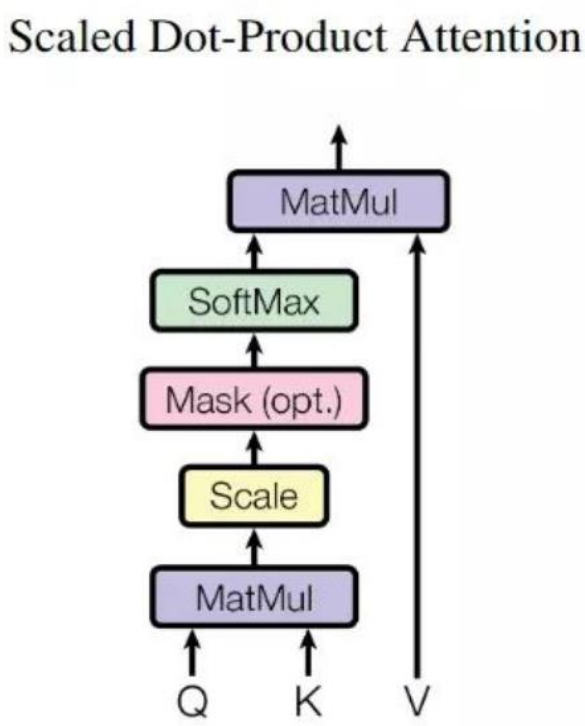
接下来就展开说明下面的结构了。



这个结构体一共包括四个部分 Multi-head attention、add&Norm、

Feed Forward、add&norm，其中两个 add&norm 是一样的。

Multi-head attention 从结构上，又可以进行细分，是不是已经有点晕了。结构图中有三个输入，分别是 K、Q、V，这里需要先解释下这几个参数。



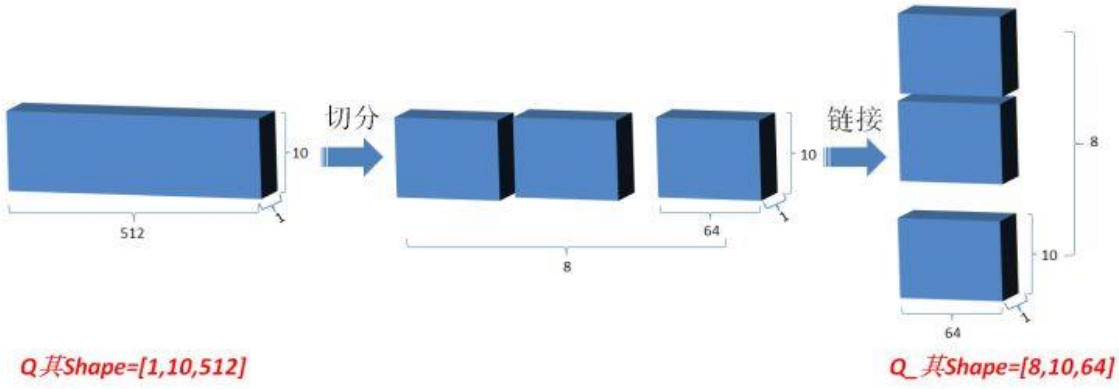
1. 在 encoder 的 self-attention 中，Q、K、V 都来自同一个地方，它们是上一层 encoder 的输出。对于第一层 encoder，它们就是 word embedding 和 positional encoding 相加得到的输入。
2. 在 decoder 的 self-attention 中，Q、K、V 也是自于同一个地方，它们是上一层 decoder 的输出。对于第一层 decoder，同样也是 word embedding 和 positional encoding 相加得到的输入。但是对于 decoder，我们不希望它能获得下一个 time step (即将来的信息，不想让他看到它要预测的信息)，因此我们需要进行 sequence masking。
3. 在 encoder-decoder attention 中，Q 来自于 decoder 的上一层的输出，K 和 V 来自于 encoder 的输出，K 和 V 是一样的。

目前可能描述有点抽象，不容易理解。结合一些应用来说，比如，

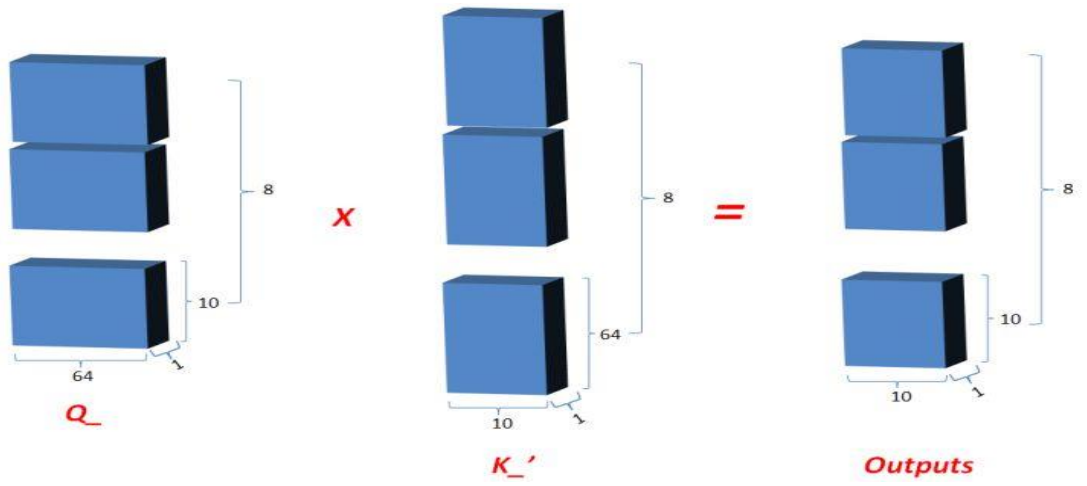
如果是在自动问答任务中的话，Q 可以代表答案的词向量序列，取 K = V 为问题的词向量序列，那么输出就是所谓的 Aligned Question Embedding。

首先分别对 V，K，Q 三者分别进行线性变换，即将三者分别输入到三个单层神经网络层，激活函数选择 relu，输出新的 V，K，Q（三者 shape 都和原来 shape 相同，即经过线性变换时输出维度和输入维度相同）。

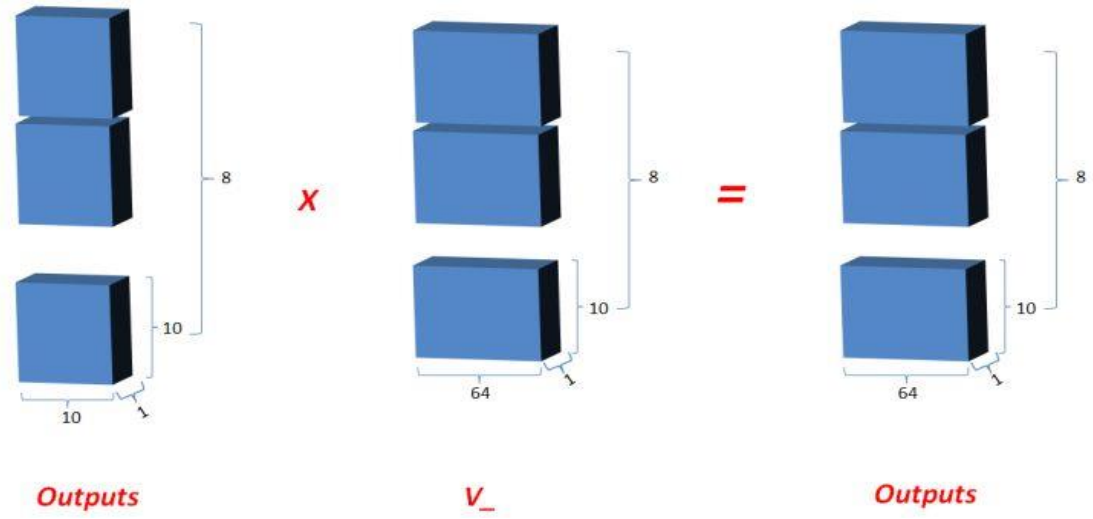
然后将 Q 在最后一维上进行切分为 num_heads(假设为 8)段，然后对切分完的矩阵在 axis=0 维上进行 concat 链接起来；对 V 和 K 都进行和 Q 一样的操作；操作后的矩阵记为 Q_,K_,V_；



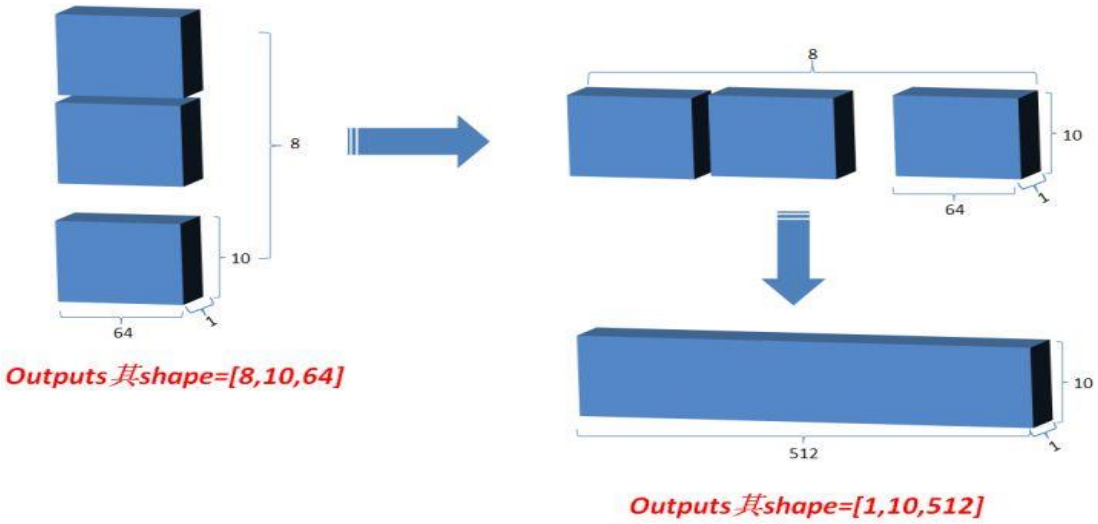
Q_矩阵相乘 K_的转置（对最后 2 维），生成结果记为 outputs，然后对 outputs 进行 scale 一次更新为 outputs；此次矩阵相乘是计算词与词的相关性，切成多个 num_heads 进行计算是为了实现对词与词之间深层次相关性进行计算。



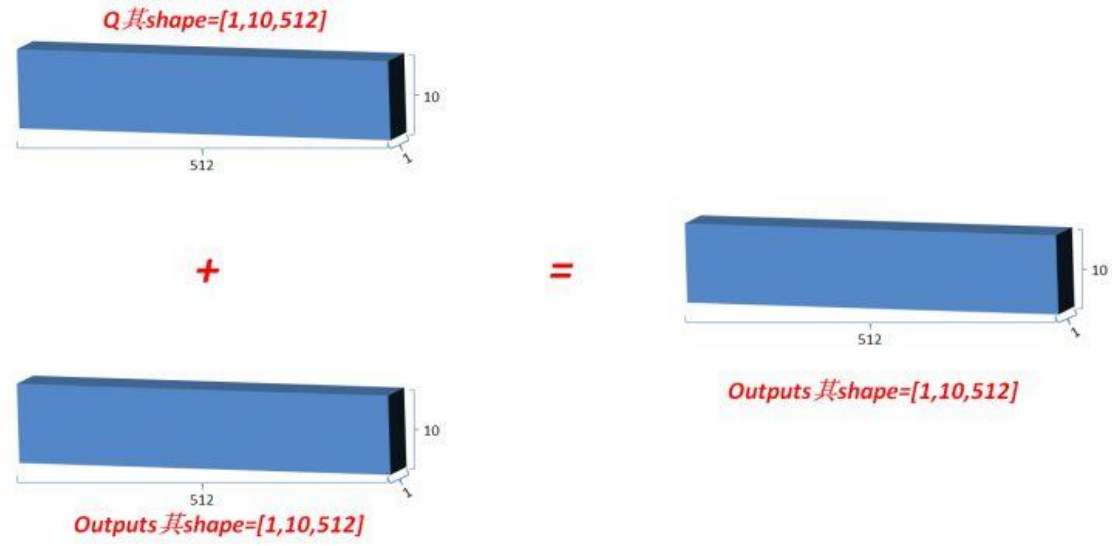
对 outputs 进行 softmax 运算，更新 outputs，即 outputs=softmax(outputs)。最新的 outputs（即 K 和 Q 的相关性，而 K 和 V 一直相等） 矩阵相乘 V_， 其值更新为 outputs；



最后将 outputs 在 axis=0 维上切分为 num_heads 段，然后在 axis=2 维上合并， 恢复原来 Q 的维度。

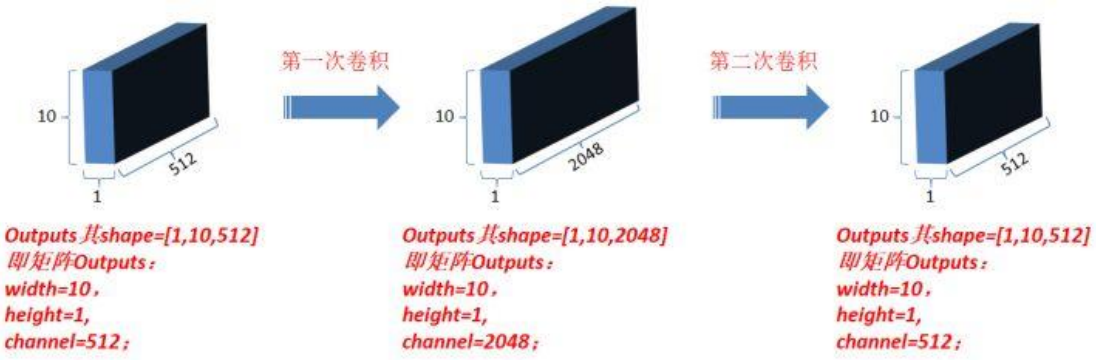


Add&norm 类似 ResNet，将最初的输入与其对应的输出叠加一次，即 outputs=outputs+Q， 使网络有效叠加，避免梯度消失；

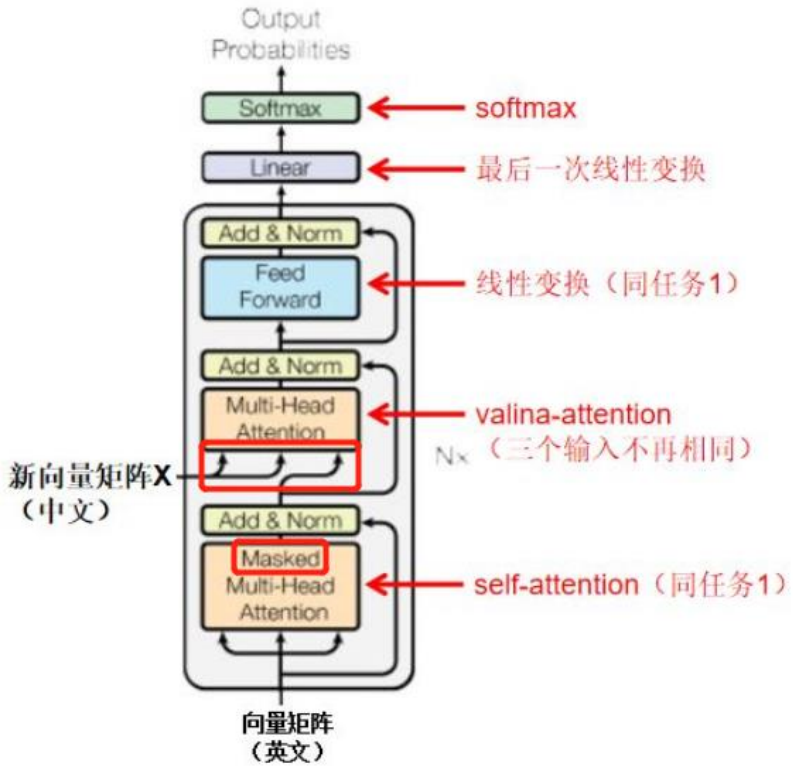


norm 标准化矫正一次，在 outputs 对最后一维计算均值和方差，。用 outputs 减去均值除以方差+spsilon 得值更新为 outputs，然后变量 gamma*outputs+变量 beta。

feedForward 对 outputs 进行第一次卷积操作，结果更新为 outputs（卷积核为 1*1，每一次卷积操作的计算发生在一个词对应的向量元素上，卷积核数目即最后一维向量长度，也就是一个词对应的向量维数）；对最新 outputs 进行第二次卷积操作，卷积核仍然为 1*1，卷积核数目为 N。



令 matEnc=outputs，完成一次循环，然后返回到 3.2 开始第二次循环；共循环 Nx（自定义；每一次循环其结构相同，但对应的参数是不同的，即是独立训练的）；完成 Nx 次后，模型的编码部分完成，仍然令 matEnc=outputs，准备进入解码部分。



解码部分，主要在于一个 Masked、并且在第二次 Attention 采用的是 valina-attention，这个在本文前半部分已经有阐述。

参考资料

https://zhuanlan.zhihu.com/p/47812375

https://zhuanlan.zhihu.com/p/44731789

https://juejin.im/post/5b9f1af0e51d450e425eb32d#heading-13

https://jalammar.github.io/illustrated-transformer/

https://arxiv.org/abs/1706.03762

https://www.github.com/kyubyong/transformer