



上海交通大学
SHANGHAI JIAO TONG UNIVERSITY



学生创新中心
Student Innovation Center

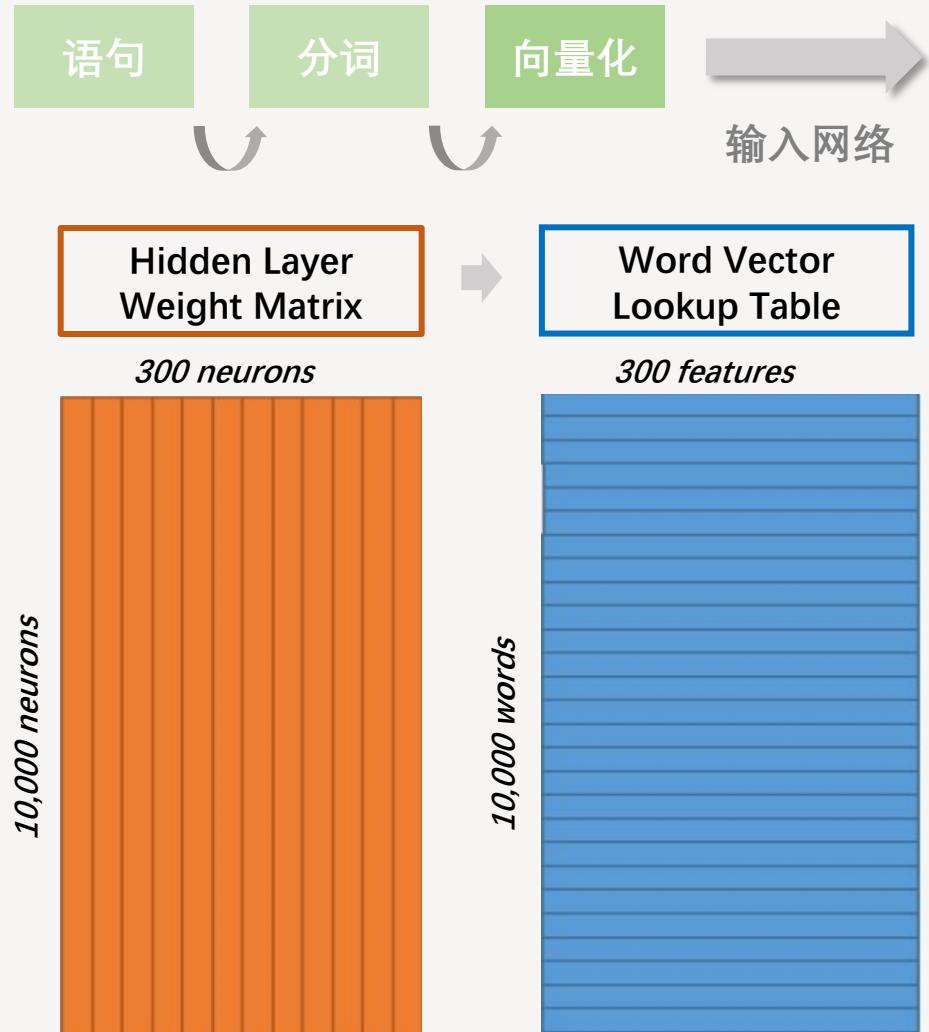


Transformer “Attention is all you need”

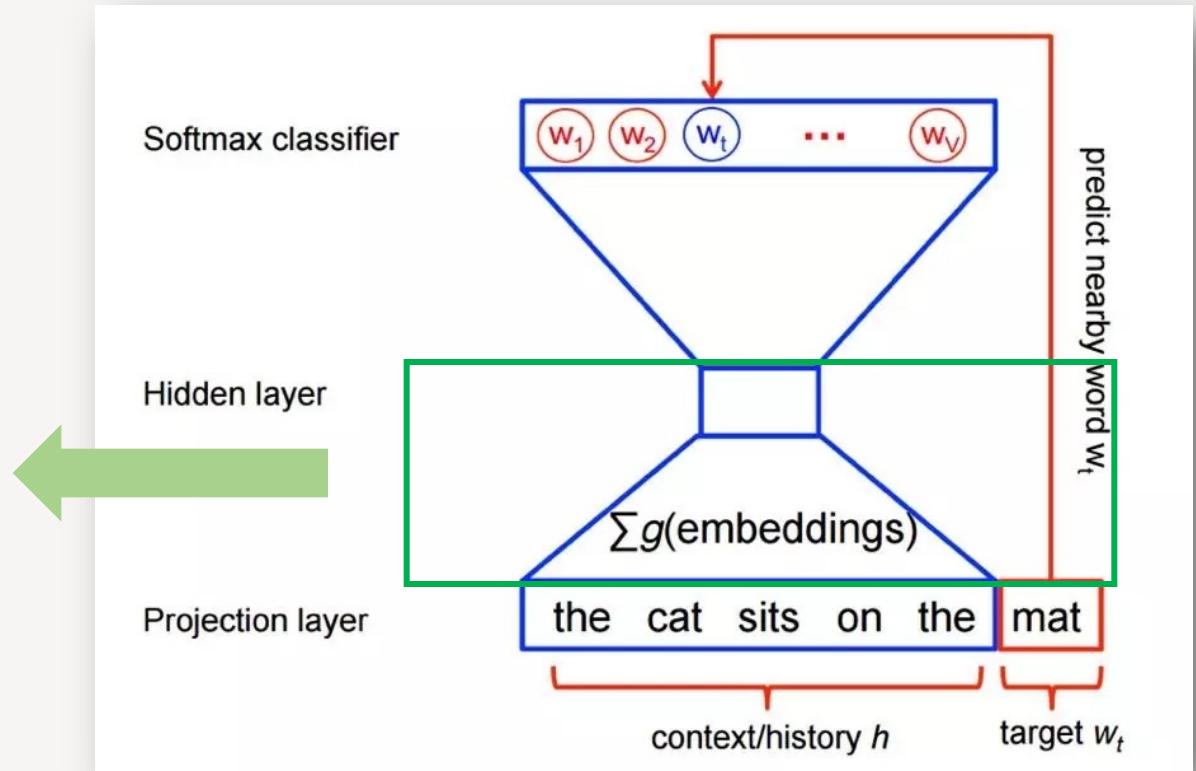
学生创新中心：肖雄子彦



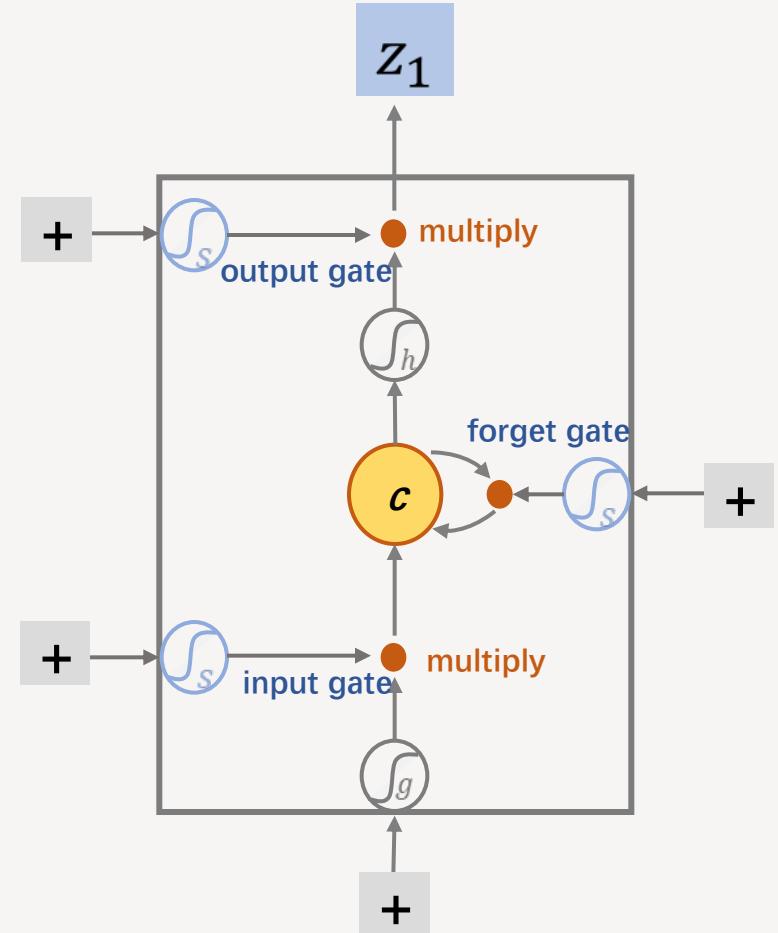
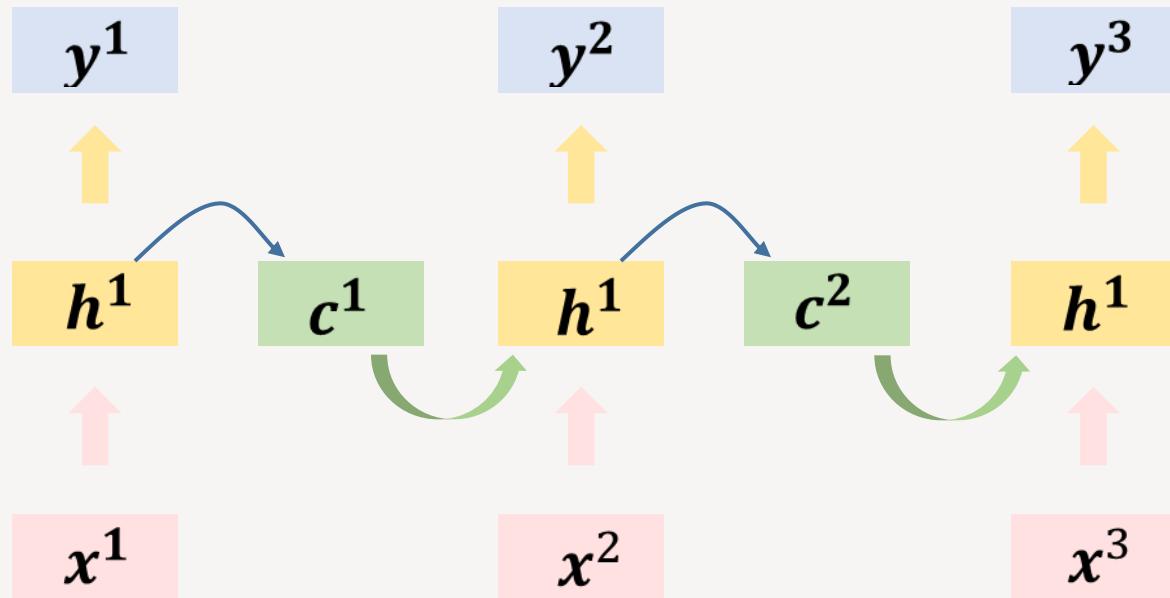
1、回顾输入方式：Word2Vec



不同语境下的词义无法得到不同的embedding
思考：能否在词向量中加入上下文的考虑？



2、回顾网络结构：RNN/LSTM



- RNN/LSTM按顺序迭代（看完前一个词再进入下一个）
- 长文下，捕捉信息间依赖关系能力越来越弱
- 不同词的重要性无法体现

We want

- 并行训练，提升计算效率
- 长距离间的信息捕捉
- 关注重点词语内容

Transformer!

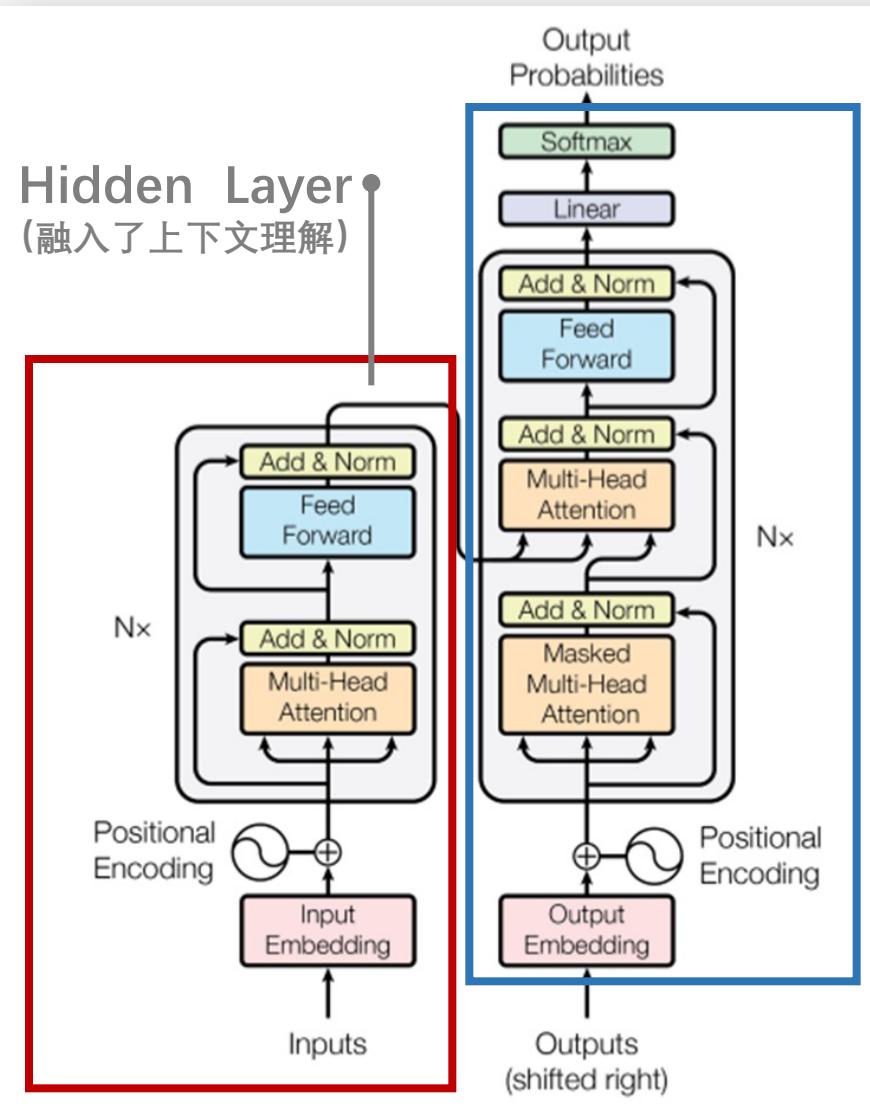
Transformer模型

2017 Seq2Seq Model
proposed by Google

上游任务
Encoder

预训练语言模型
(语言表征)

然后适配给下游
完成不同的语言任务



下游任务 Decoder

机器翻译

阅读理解

智能问答

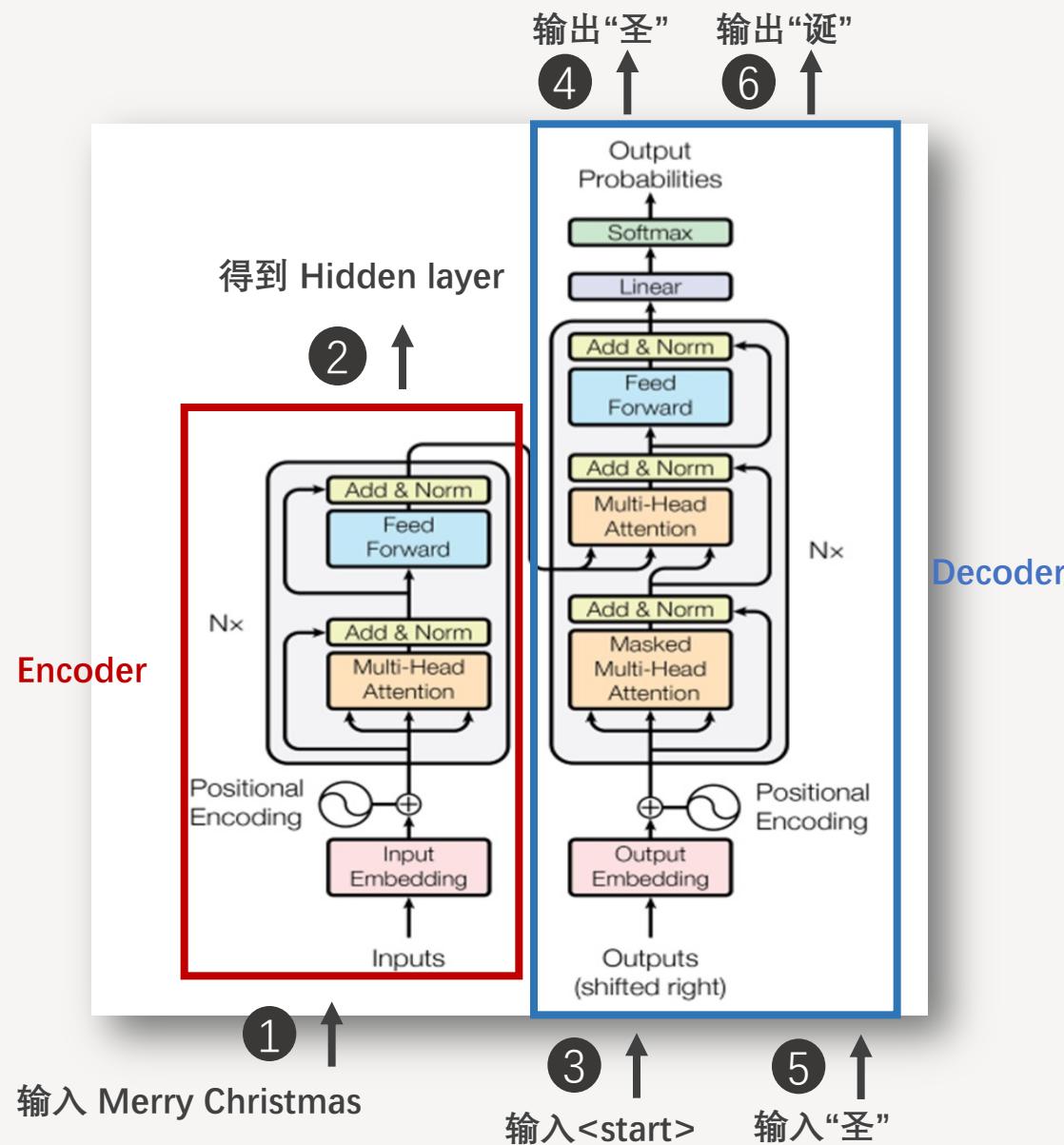
情感分析

NER命名实体识别 ...



预训练模型非常重要
其性能直接影响下游任务表现
BERT由Transformer衍生出来

For example



机器翻译任务：

- ① 输入一个语言序列到“编码器”
Merry Christmas (圣诞快乐~)
- ② 得到“编码器”的输出（隐藏层）
再输入到“解码器”
- ③ 输入 <*start*> 符到“解码器”
- ④ 得到“解码器”第一个输出 “圣”字
- ⑤ “圣” 再成为“解码器”的第二个输入
- ⑥ 得到第二个输出“诞”
重复步骤，直到“解码器”输出 <*end*>
序列生成完成。

Transformer模型

重点内容：

- Positional-encoding位置嵌入
- Self-Attention自注意力机制
- Multi-head Attention多头注意力机制
- Layer Normalization层标准化

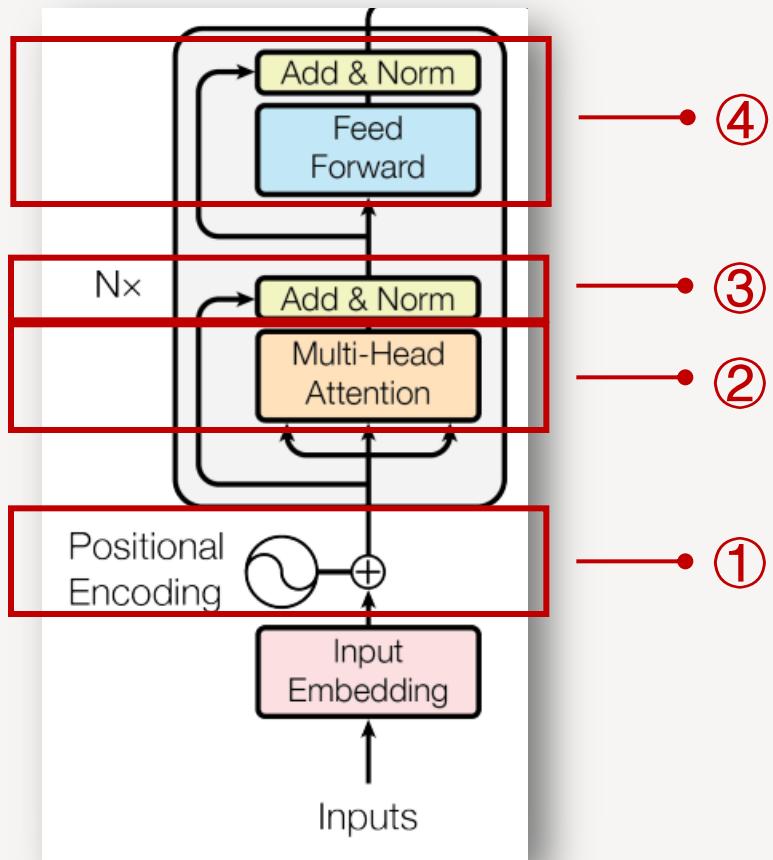


上海交通大学
SHANGHAI JIAO TONG UNIVERSITY



学生创新中心
Student Innovation Center

BERT 模型 即 编码器

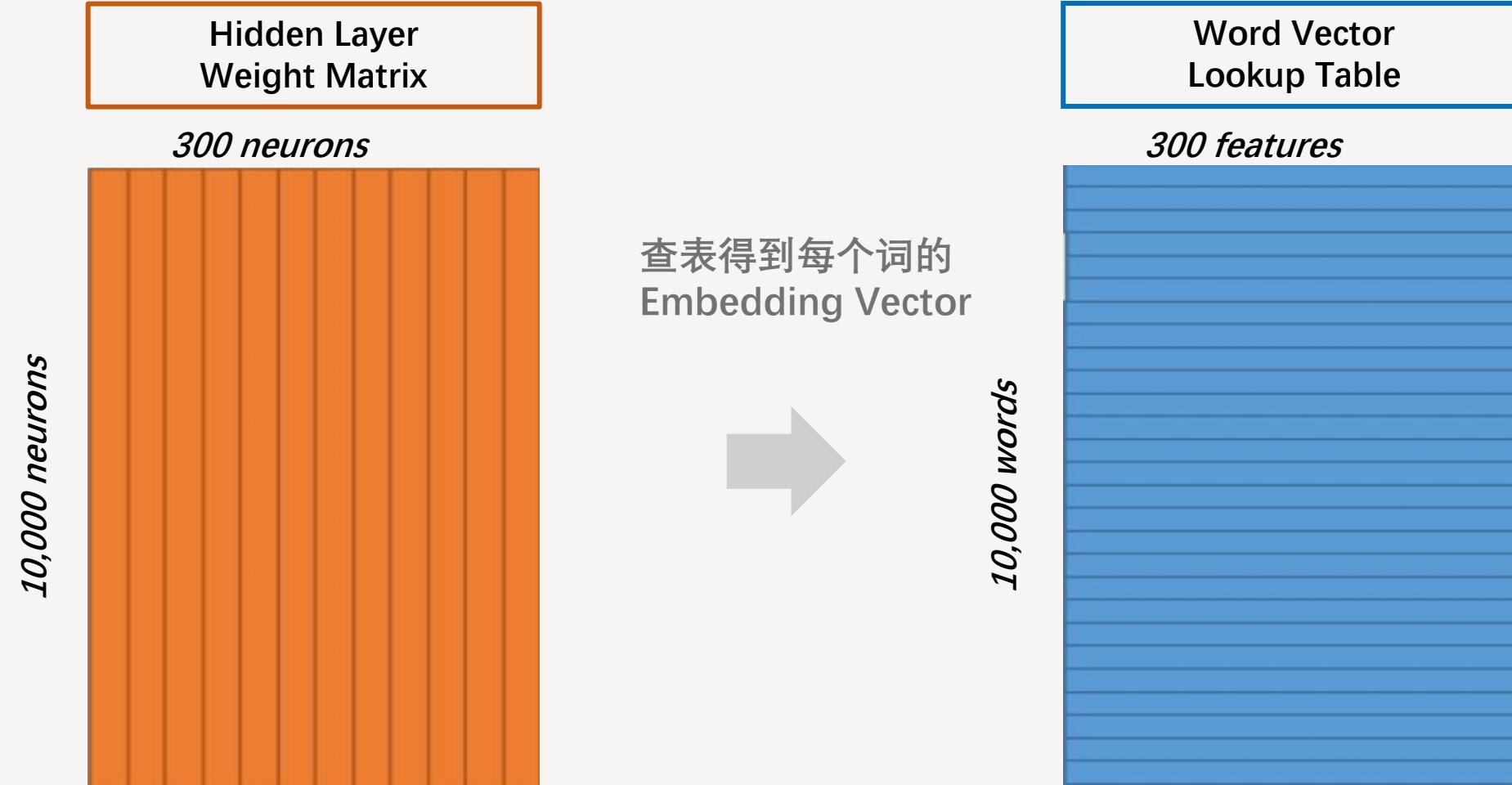


Transformer编码器

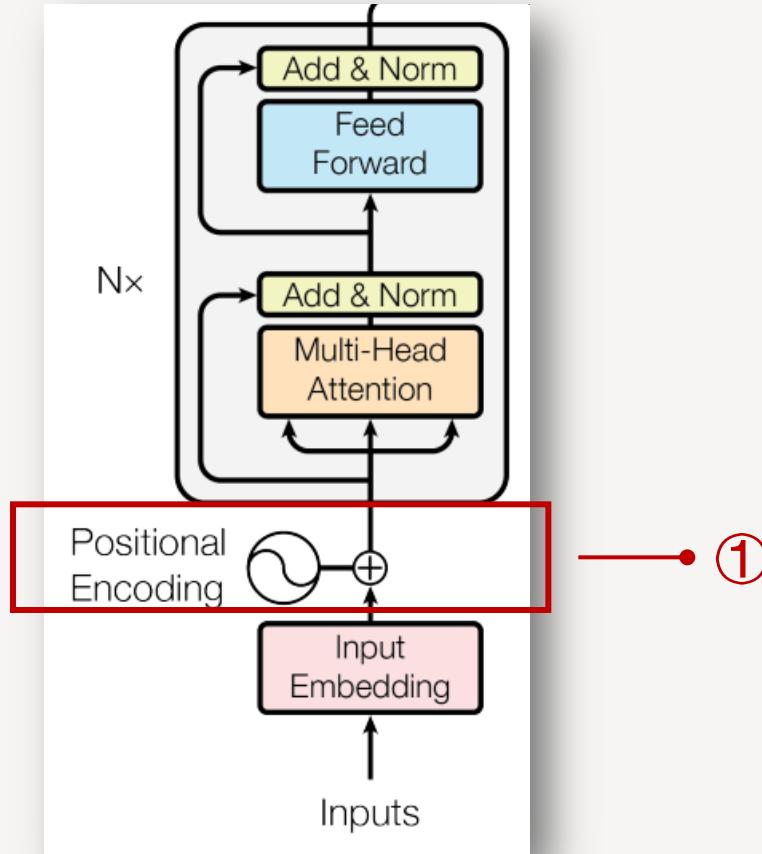
—BERT

- ① 位置嵌入 Positional Encoding
理解句子里词的顺序
- ② 多头注意力机制 Multi-Head Attention
可以真正实现上下文理解的方法
- ③ 残差连接与归一化 Add & Norm
- ④ 全连接层 Feed Forward

Input Embedding



① 位置嵌入 Positional Encoding



Transformer编码器

——BERT

- ① 位置嵌入 Positional Encoding
理解句子里词的顺序
- ② 多头注意力机制 Multi-Head Attention
可以真正实现上下文理解的方法
- ③ 残差连接与归一化 Add & Norm
- ④ 全连接层 Feed Forward

① 位置嵌入 Positional Encoding

位置嵌入 *positional encoding* [Max Sequence Length, Embedding Dimension]

Transformer没有RNN\LSTM的迭代操作, 所以我们需要提供每个字的位置信息给它, Transformer才能识别出语言中的顺序关系。

论文中使用了 \sin 和 \cos 的线性变换来提供位置信息:

$$\text{偶数序列 } PE_{(pos,2i)} = \sin(pos / 10000^{2i/d_{model}})$$

$$\text{奇数序列 } PE_{(pos,2i+1)} = \cos(pos / 10000^{2i/d_{model}})$$

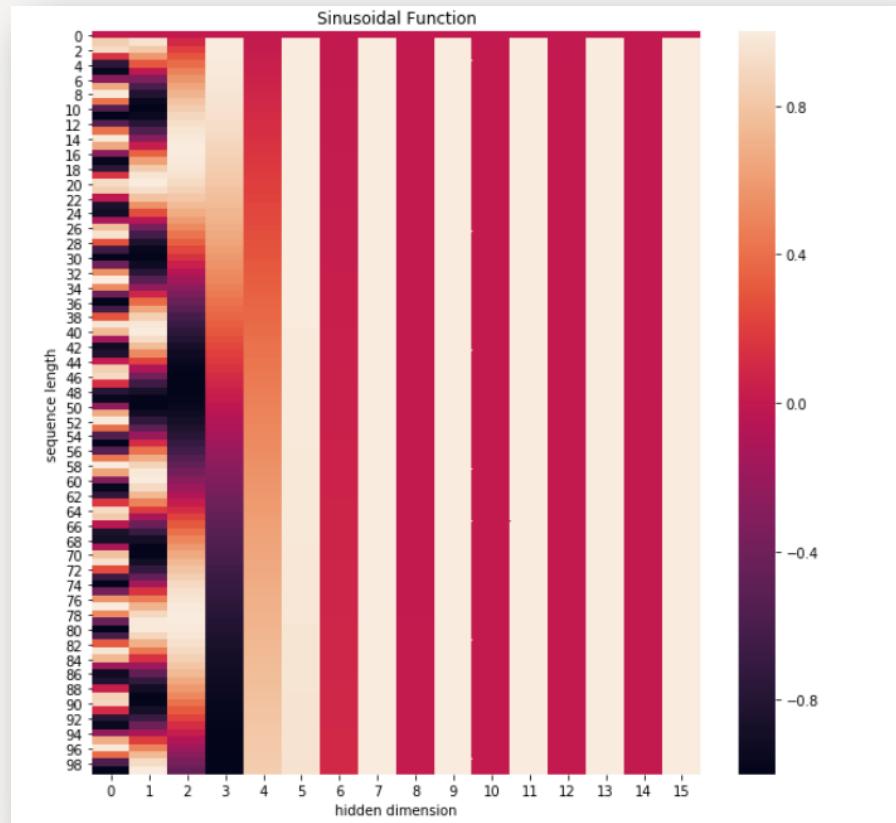
字的位置[0,maxlength)

[0, Embedding dimension)

位置嵌入计算得到的维度是:

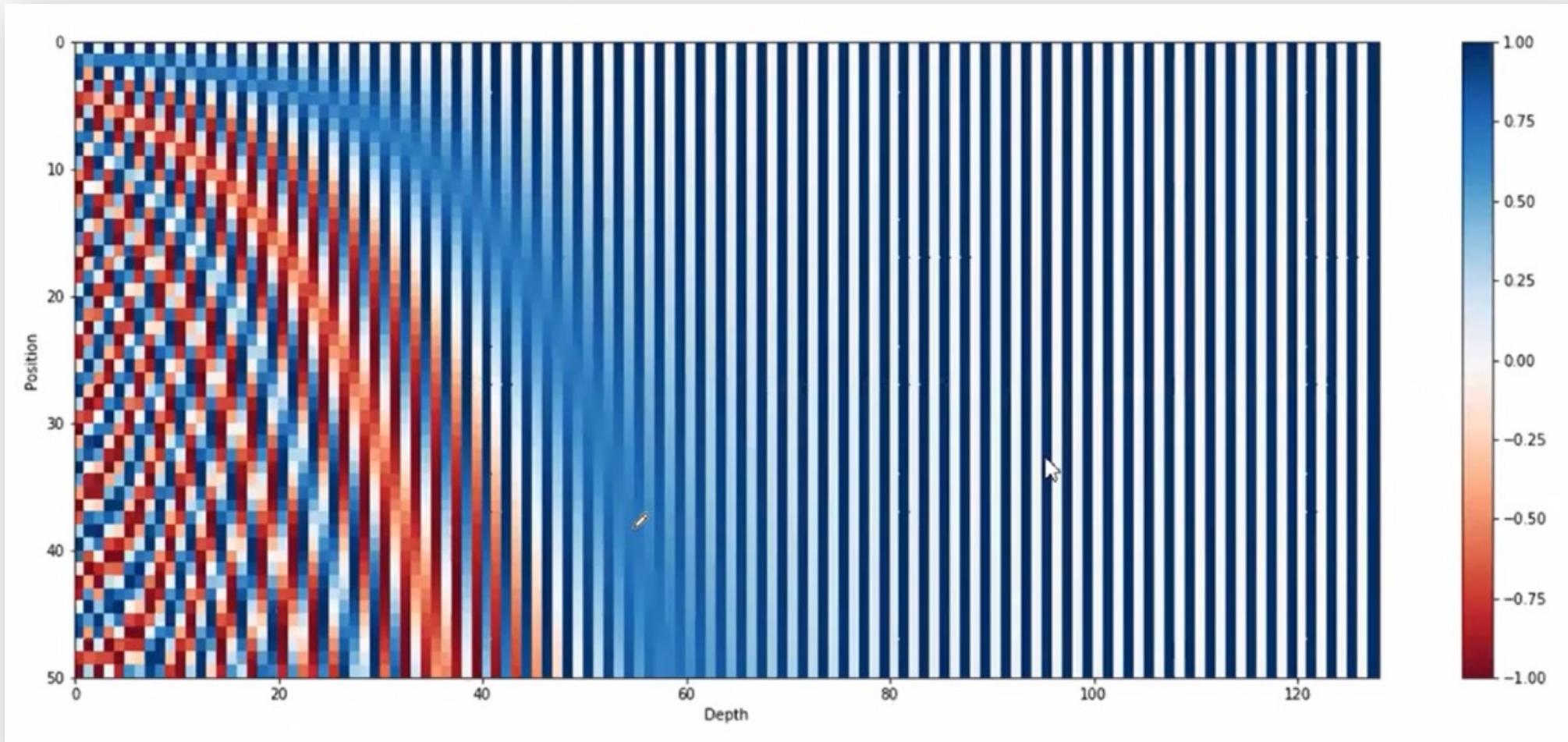
[Max Sequence Length, Embedding Dimension]

位置嵌入和字向量元素相加



① 位置嵌入 Positional Encoding

句子长度为50，维度为128的Positional Encoding

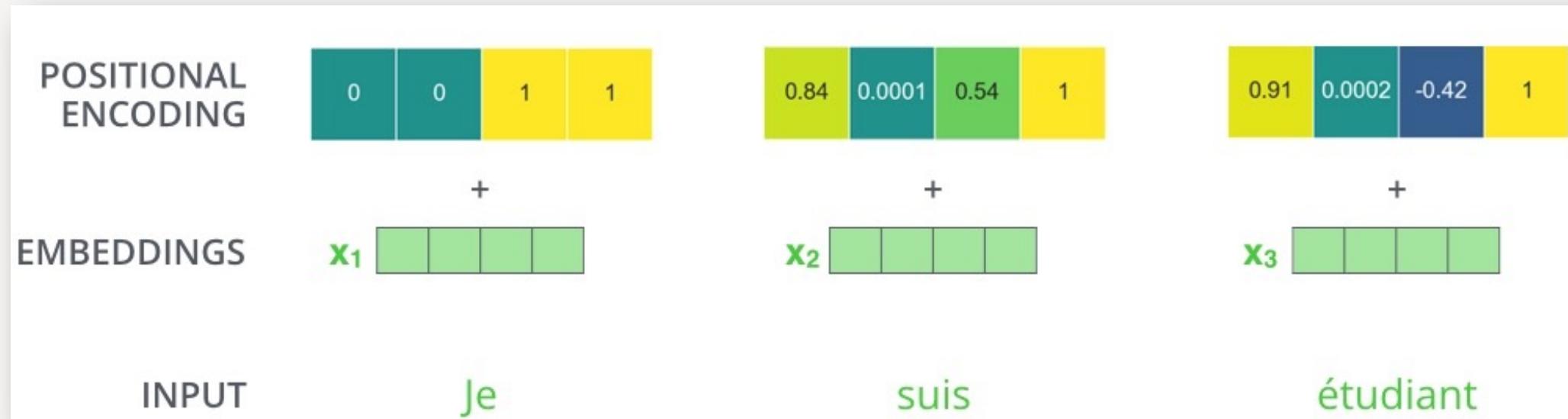


① 位置嵌入 Positional Encoding

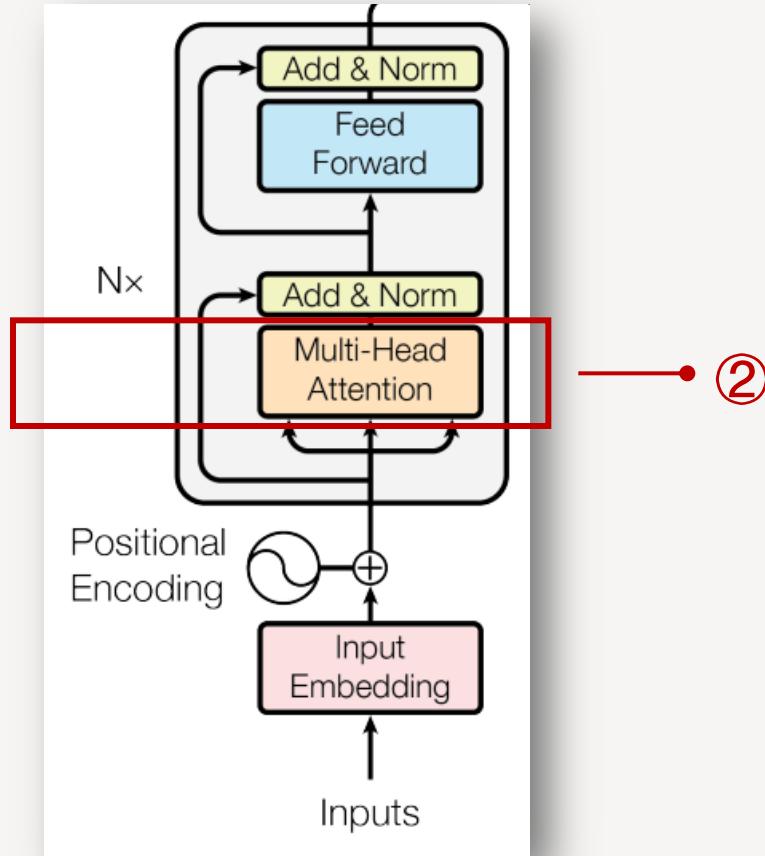
如前面所说

位置嵌入的维度为 $[max\ sequence\ length, embedding\ dimension]$

初始化字/词向量后，与Positional Encoding进行元素相加



② Multi-Head Attention



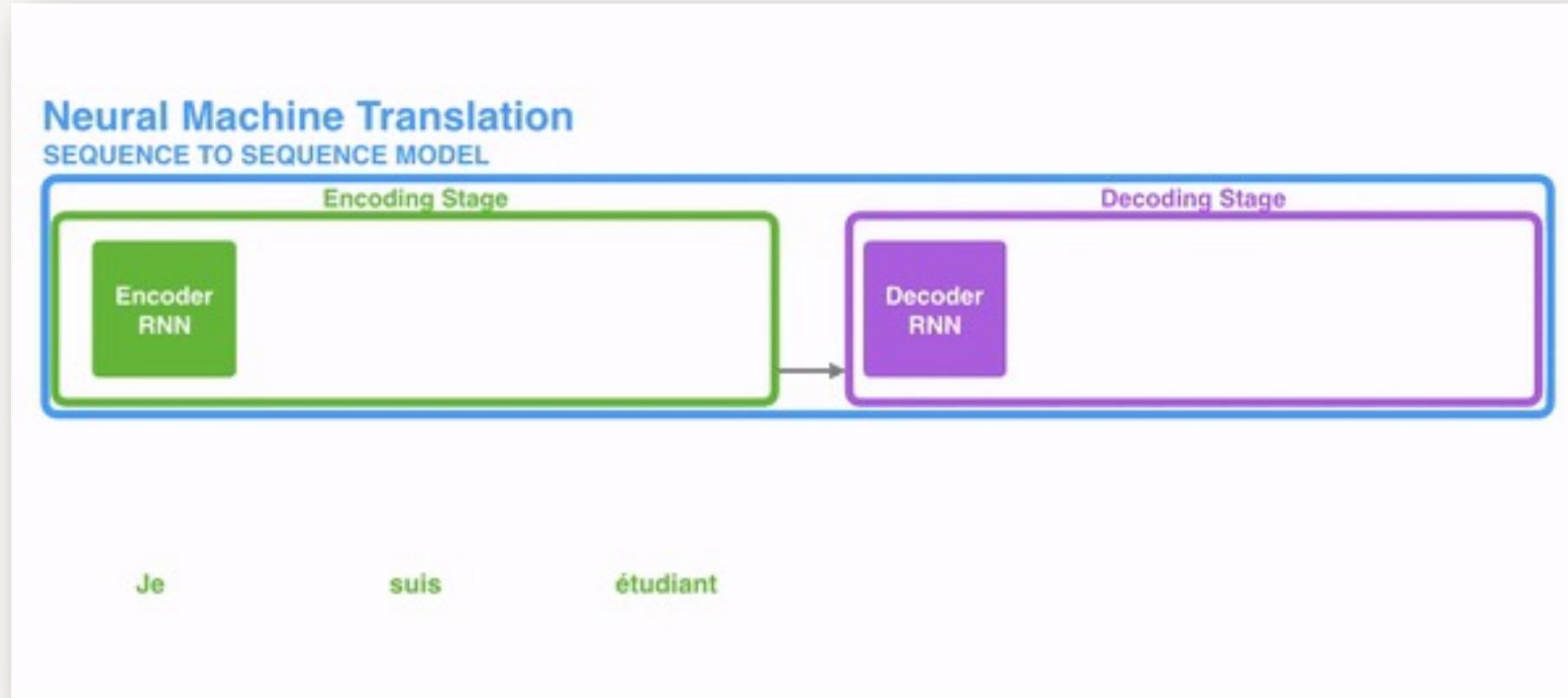
Transformer编码器

——BERT

- ① 位置嵌入 Positional Encoding
理解句子里词的顺序
- ② 多头注意力机制 Multi-Head Attention
可以真正实现上下文理解的方法
- ③ 残差连接与归一化 Add & Norm
- ④ 全连接层 Feed Forward

Attention Mechanism?

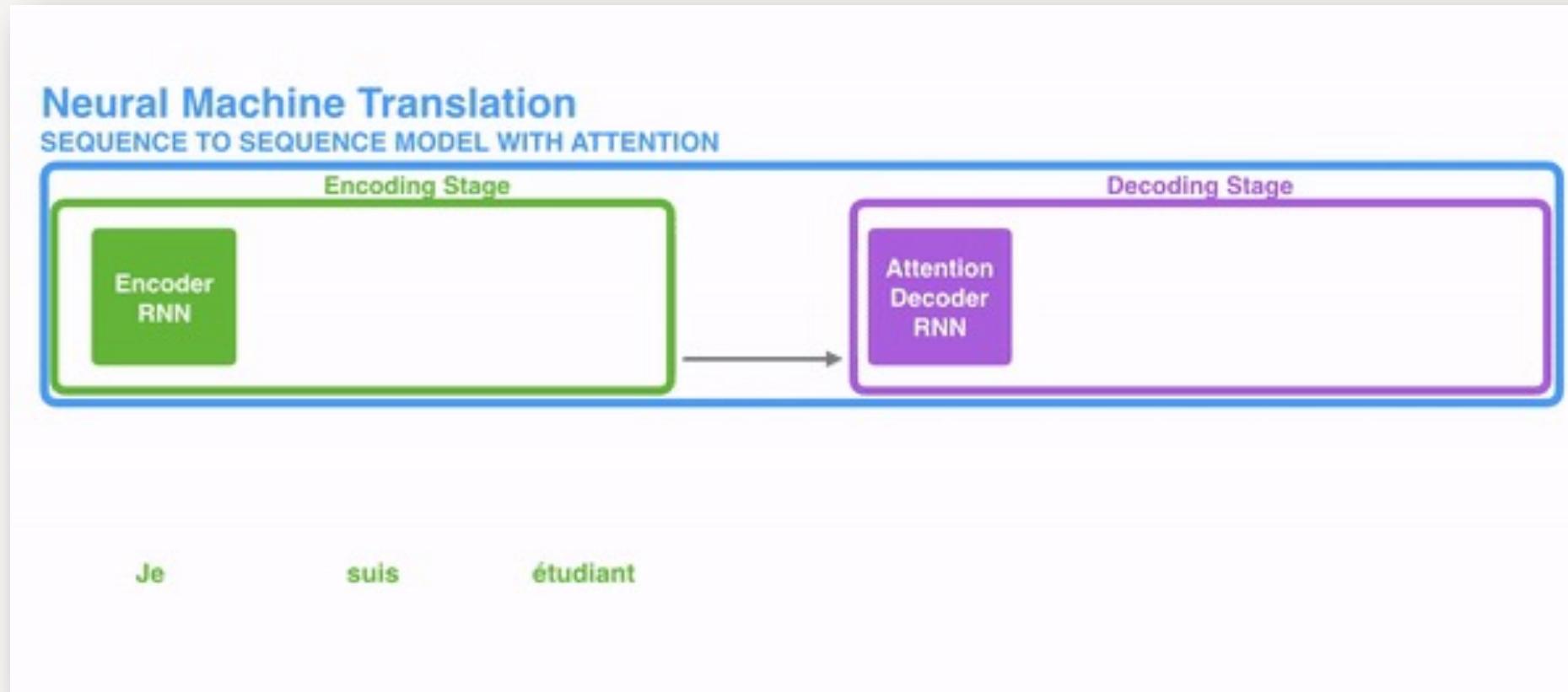
回顾Seq2Seq模型，传统的机器翻译基本都是基于Seq2Seq模型来做的，模型分为Encoder与Decoder，主要单元：RNN、LSTM。



如果文本稍长，就很容易丢信息。为了解决这个问题，Attention应运而生。

Attention Mechanism?

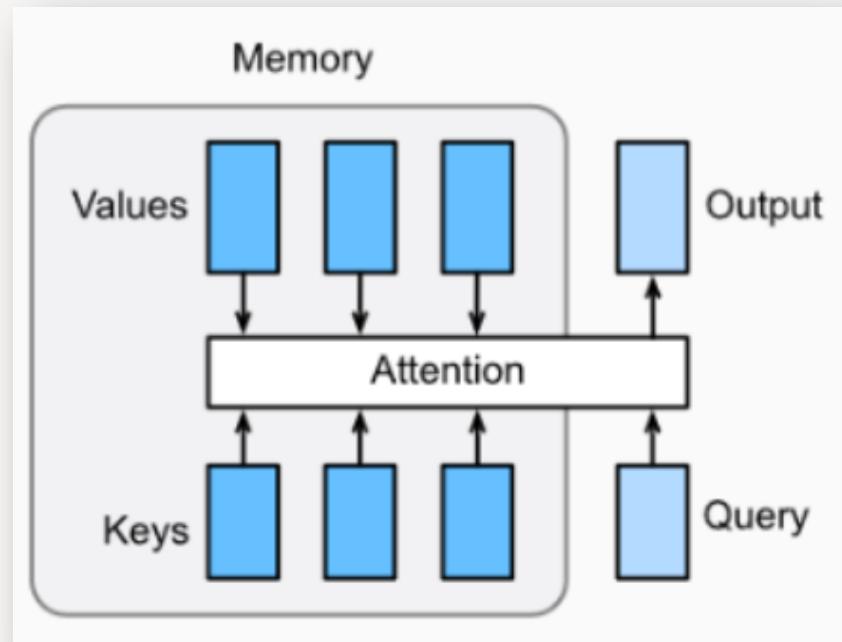
对Encoder所有节点的Hidden State进行加权求和，提供给Decode。
而不仅仅只是Encoder最后一个节点的Hidden state。



能让相关性高的hidden state的分数值更大，相关性低的hidden state的分数值更低。

What is the Attention Mechanism

Attention是一种让模型对重要信息重点关注并充分学习的技术，能作用于任何序列模型中。



Attention 注意力机制

- Bahdanau等人发表《Neural Machine Translation by Jointly Learning to Align and Translate》，第一个将Attention机制应用到NLP领域中（机器翻译）。
- 2017年Google团队发表《Attention is all you need》，大量使用了自注意力（self-attention）机制来学习文本表示。

Attention机制，其实就是建立隐层的权重分配（注意力要放在哪里）

Attention Mechanism?

普通RNN

$$s_i = f(s_{i-1}, y_{i-1})$$

f 代表RNN中一步的运算

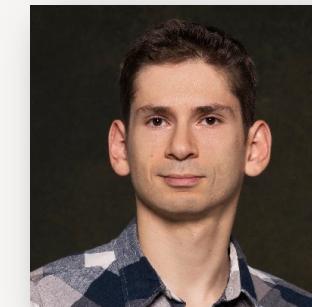


RNN with Attention

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

$$c_i = \sum_{j=1}^N \alpha_{ij} h_j$$

$$\alpha_{ij} = \text{softmax} \frac{\exp(e_{ij})}{\sum_{k=1}^N \exp(e_{ik})}$$



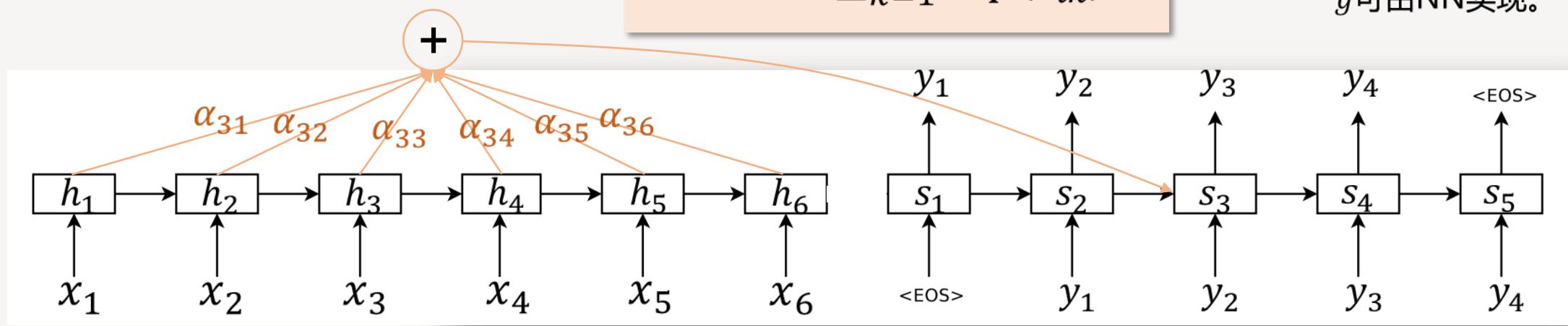
Dzmitry Bahdanau



Yoshua Bengio

$$e_{ij} = g(s_{i-1}, h_i)$$

g 可由NN实现。



Encoder

机器翻译 (Dzmitry et al. ICLR 2014)

Decoder



② Self-Attention 自注意力机制

sentence

X 的维度
[batch size, seq length]



$$X \in \mathbb{R}^{\text{batch size} * \text{seq len}}$$

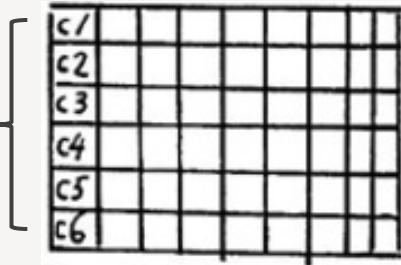
$$X_{\text{embed}} = \text{EmbeddingLookup}(X) + \text{PositionalEncoding}$$

$$X_{\text{embed}} \in \mathbb{R}^{\text{batch size} * \text{seq len} * \text{embed dim}}$$

Embedding Lookup + Positional Encoding

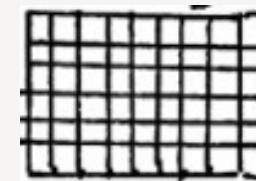
$X_{\text{embedding}}$

Seq length

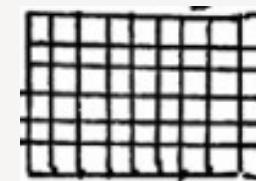


[batch size, seq length, embed dim]

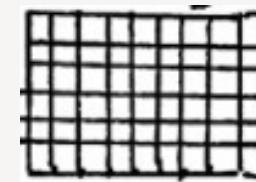
$$Q = X_{\text{embed}} W_Q$$



$$K = X_{\text{embed}} W_K$$



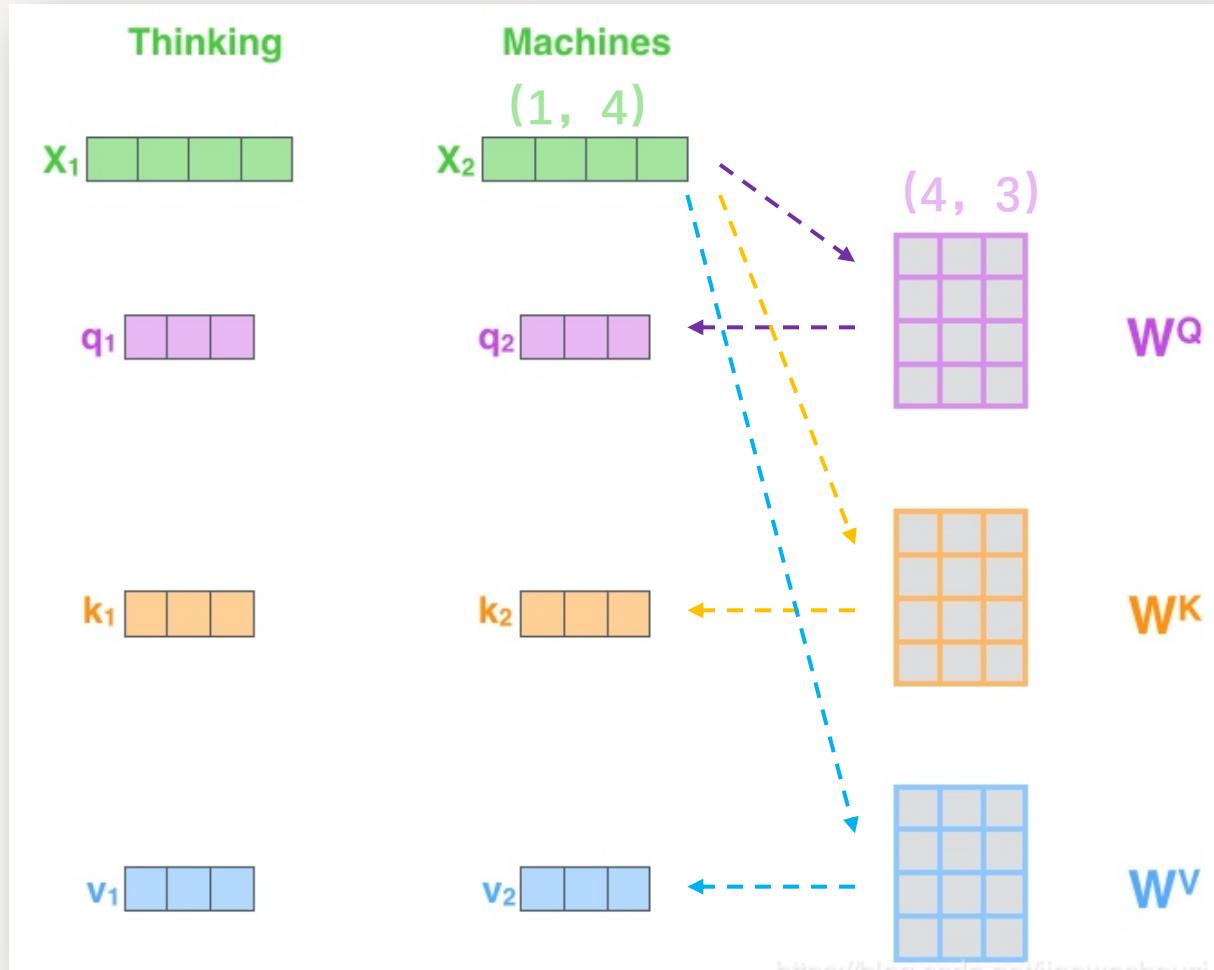
$$V = X_{\text{embed}} W_V$$



分别进行三次线性变换，
得到Q, K, V三个矩阵

② Self-Attention 自注意力机制

1. 对Embedding Layer的输出进行三次线性变换



矩阵表达

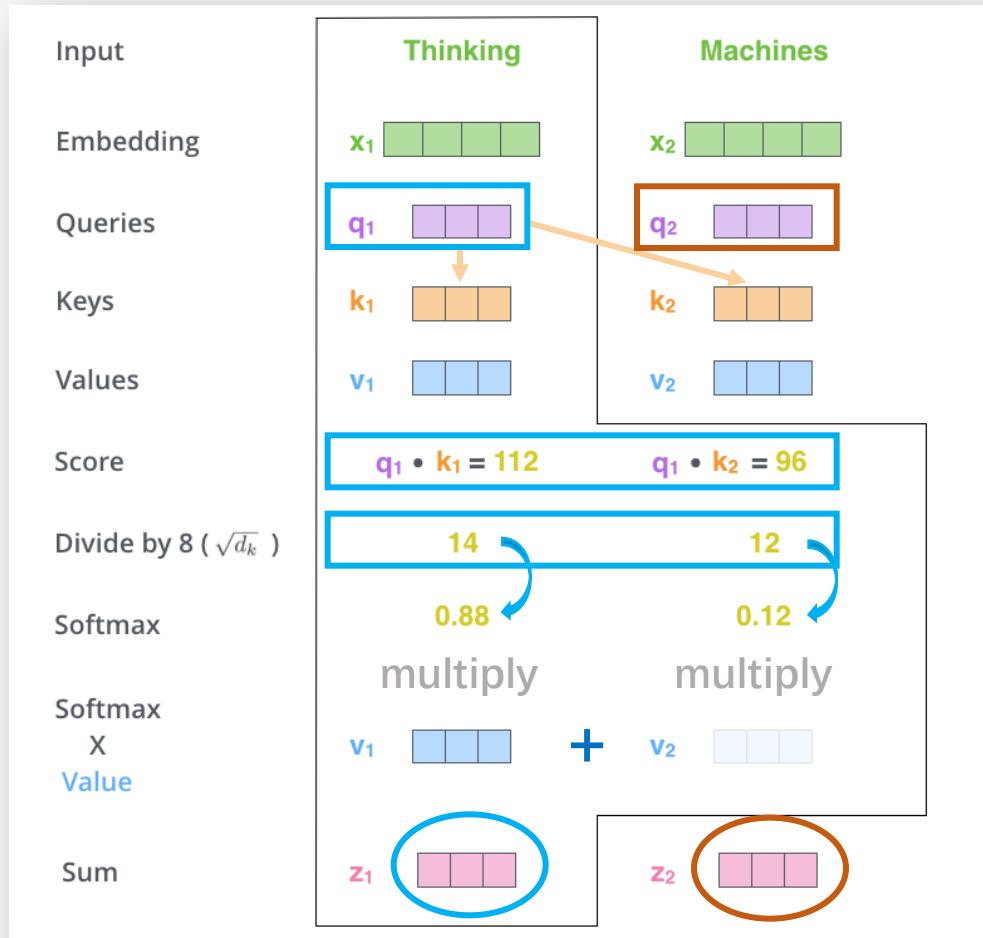
The diagram shows the matrix expression for the computation of Query (Q), Key (K), and Value (V) vectors from input matrix X :

$$X \times W^Q = Q$$
$$X \times W^K = K$$
$$X \times W^V = V$$

② Self-Attention 自注意力机制

2. QK进行点积，拉回原分布后，进行Softmax，得到的结果与V相乘

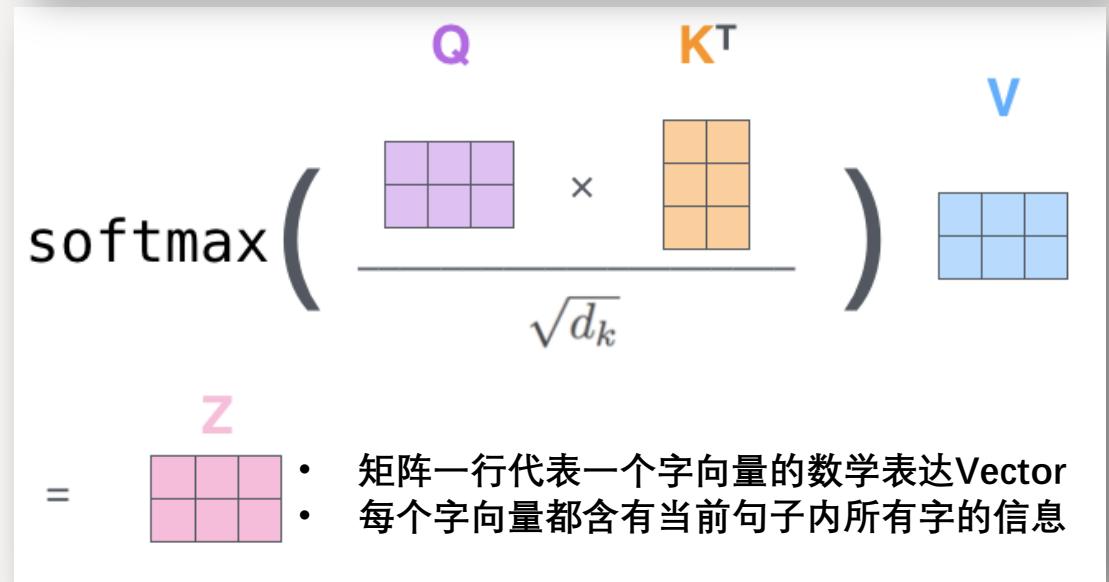
两个向量越相似
点积就越大



新的vector里就融入了Thinking和Machines一整句的信息

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

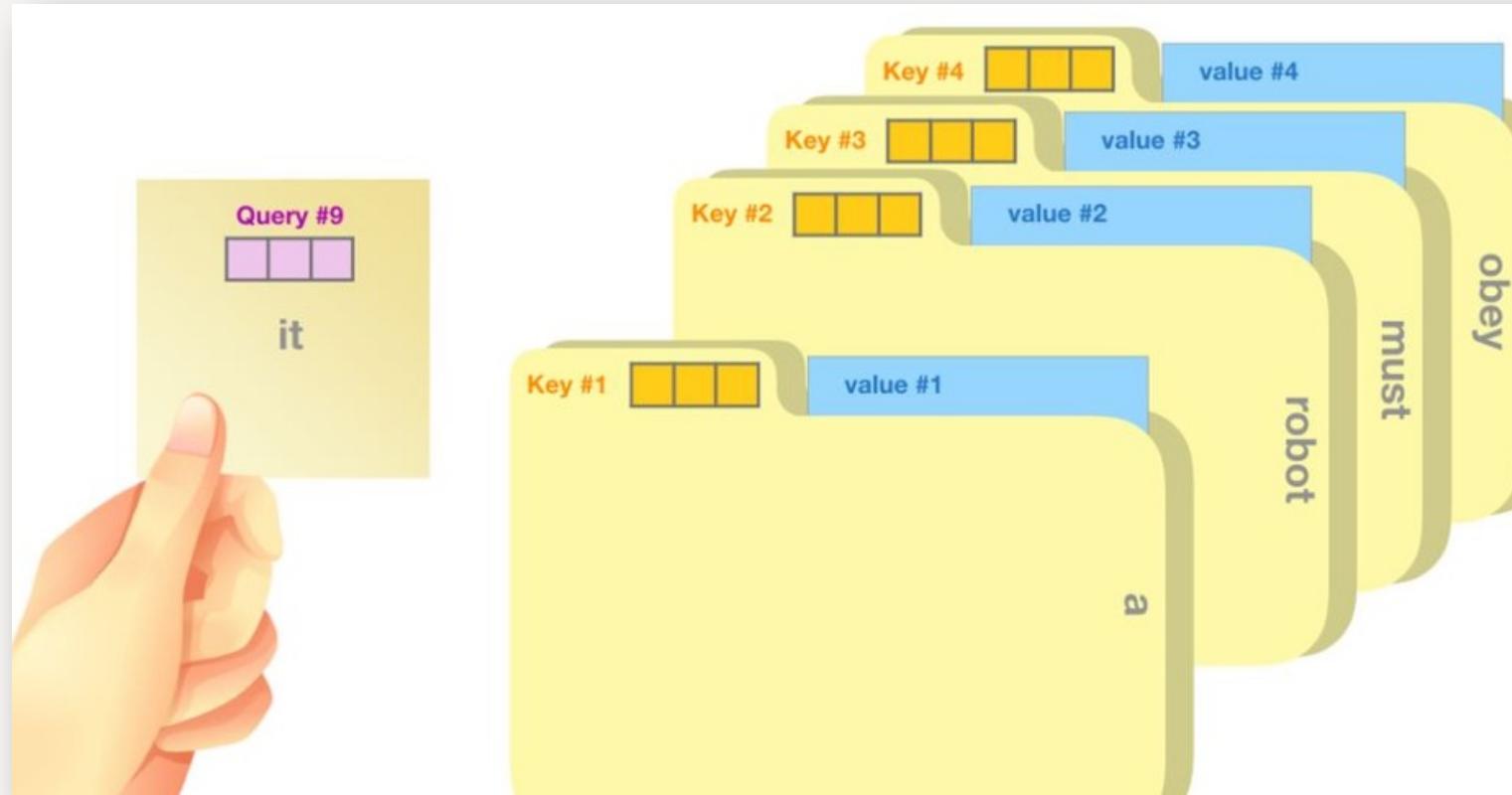
⁴To illustrate why the dot products get large, assume that the components of q and k are independent random variables with mean 0 and variance 1. Then their dot product, $q \cdot k = \sum_{i=1}^{d_k} q_i k_i$, has mean 0 and variance d_k .



关于Attention的QKV

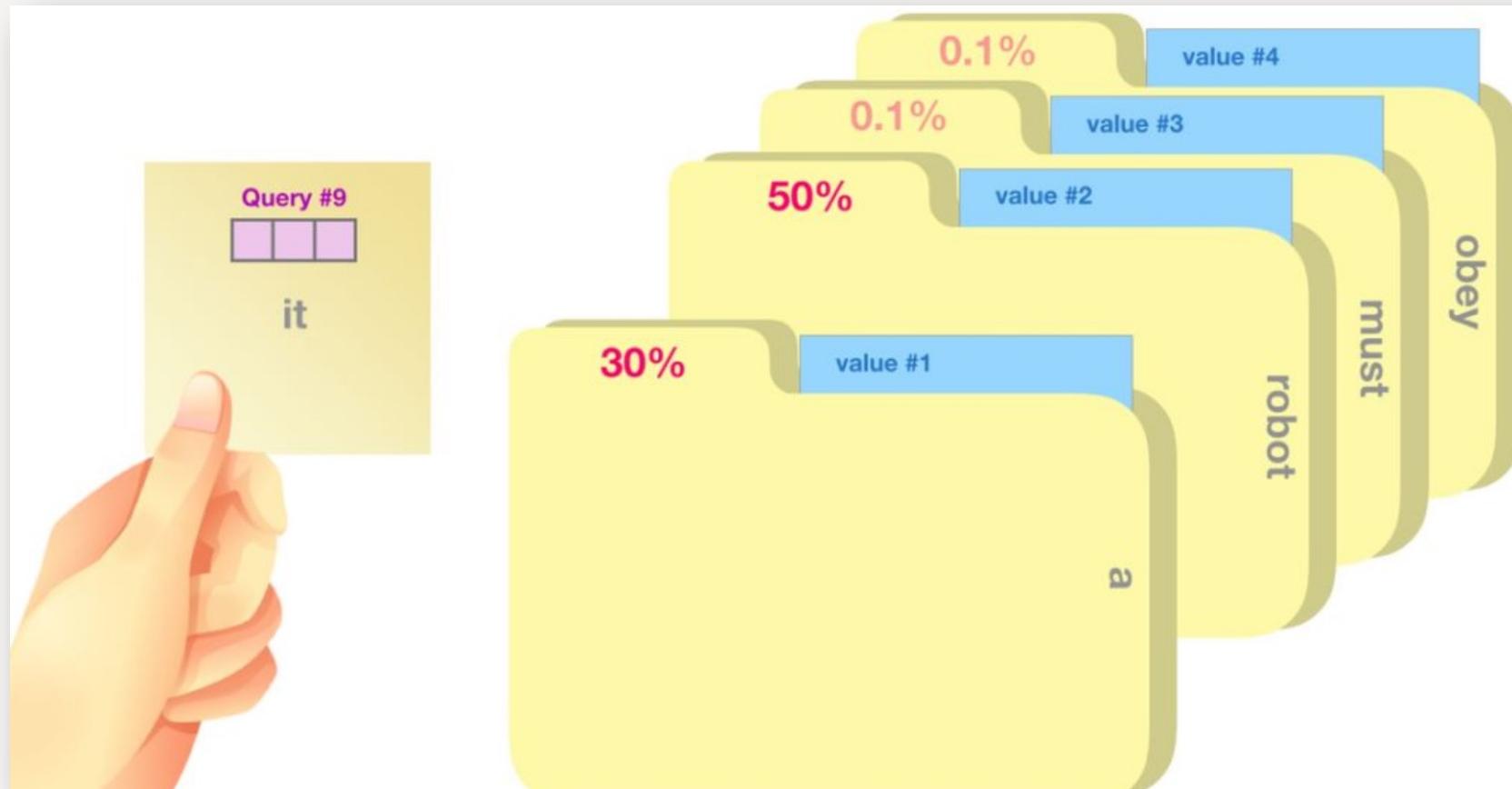
Attention本质上是为序列中每个元素都分配权重，可以理解为软寻址。

如果序列中每一个元素都以{K,V}形式存储，那么Attention则通过计算Q和K的相似度来完成寻址。Q和K计算出来的相似度反映了取出来的V值的重要程度。



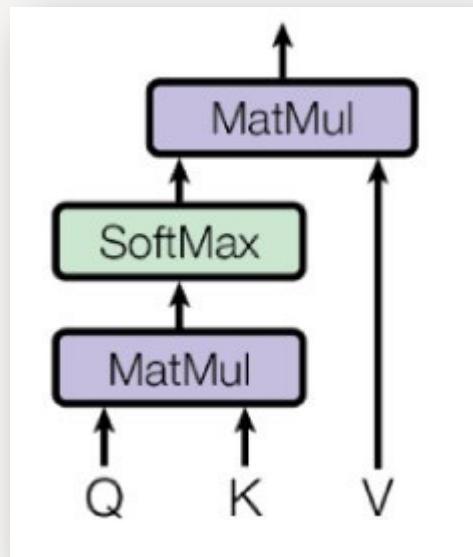
关于Attention的QKV

Self-attention机制在【KQV模型】中的特殊点在于Q、K、V都是“自己”，
因为是文本和文本自己求相似度，再和自身相乘计算得来。



② Self-Attention 自注意力机制

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



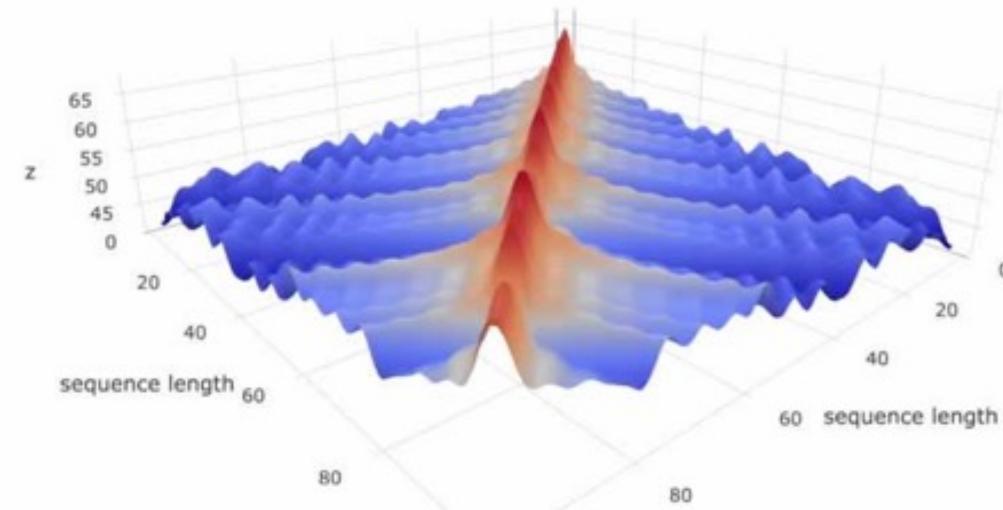
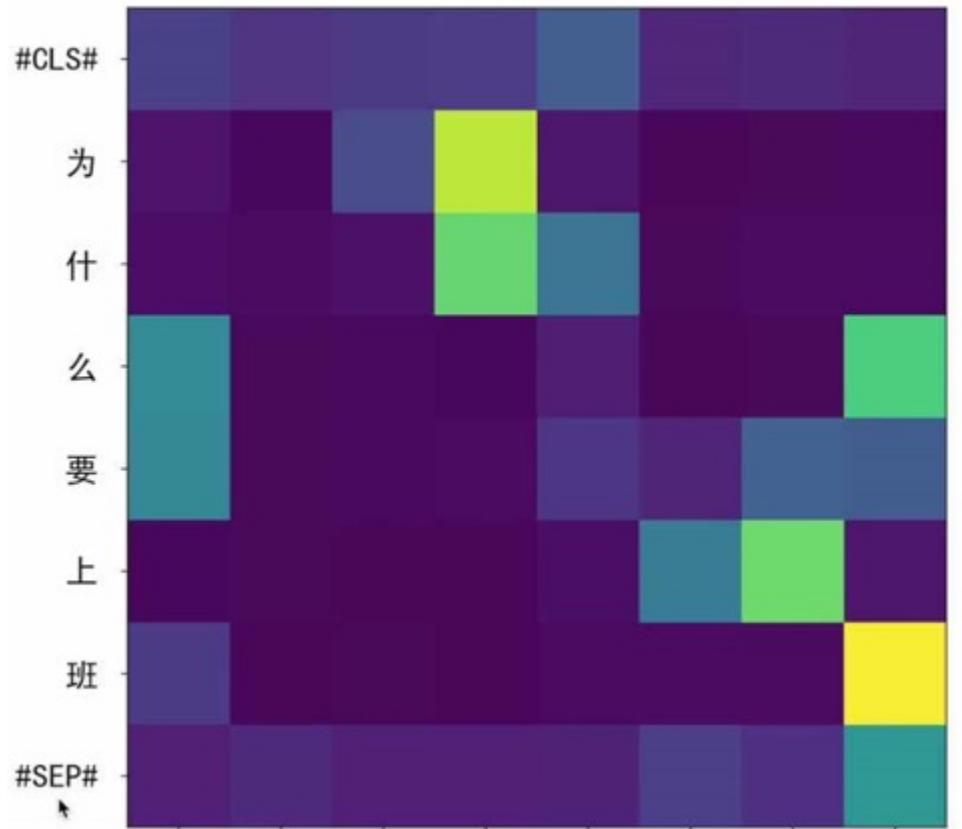
计算过程

- ① 将query和每个key进行相似度（点积）计算，得到权重
- ② 使用softmax函数对权重进行归一化
- ③ 将权重和相应的value进行加权求和得到最后的Attention。

所以Attention机制是对Value值进行加权求和，而Query和Key用来计算对应Value的权重系数。

② Self-Attention 自注意力机制

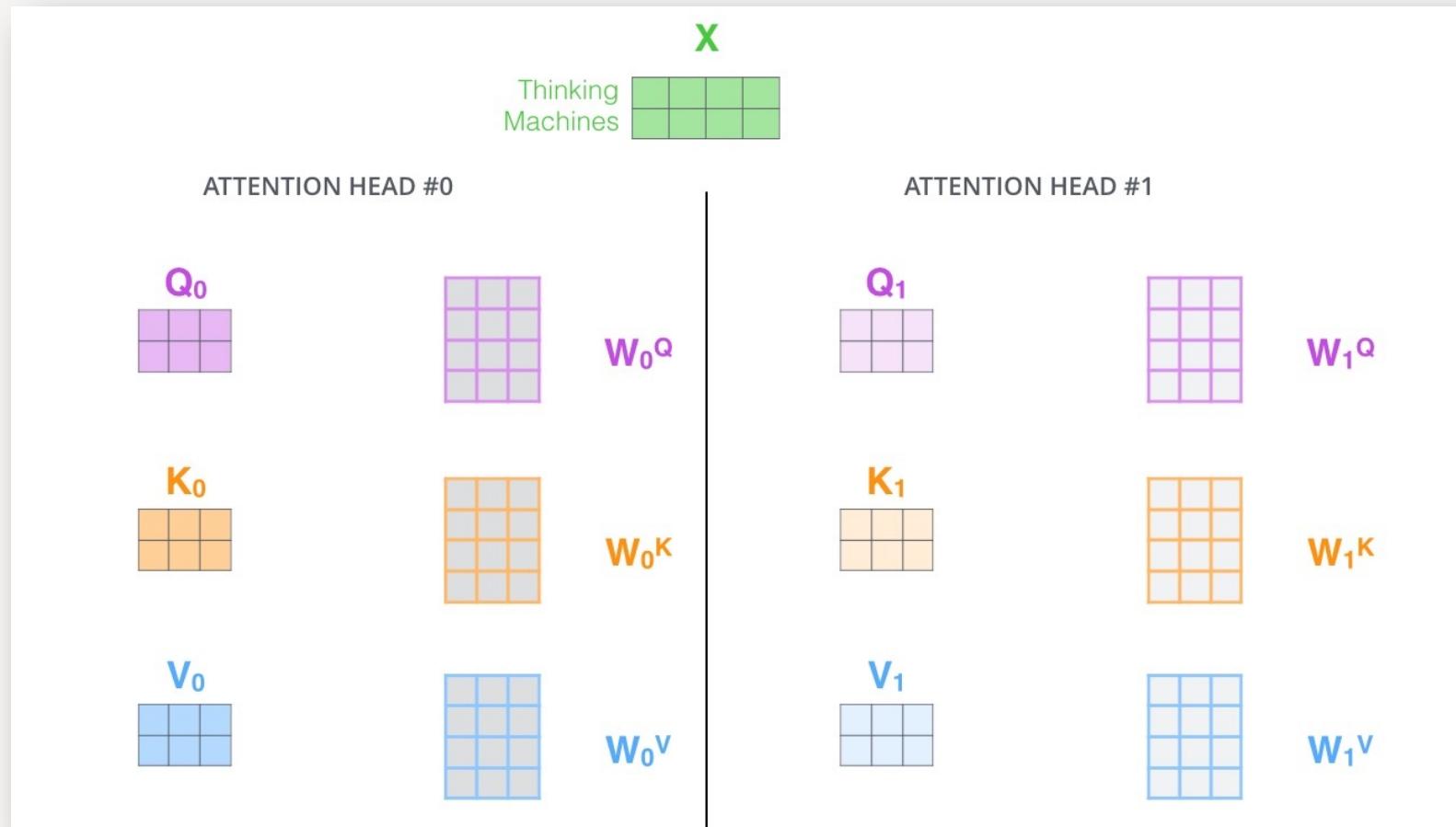
注意力机制：让每句话中每一个字对应的那一条向量里，都融入这句话所有字的信息。



https://github.com/aespresso/a_journey_into_math_of_ml/tree/master/04_transformer_tutorial_2nd_part

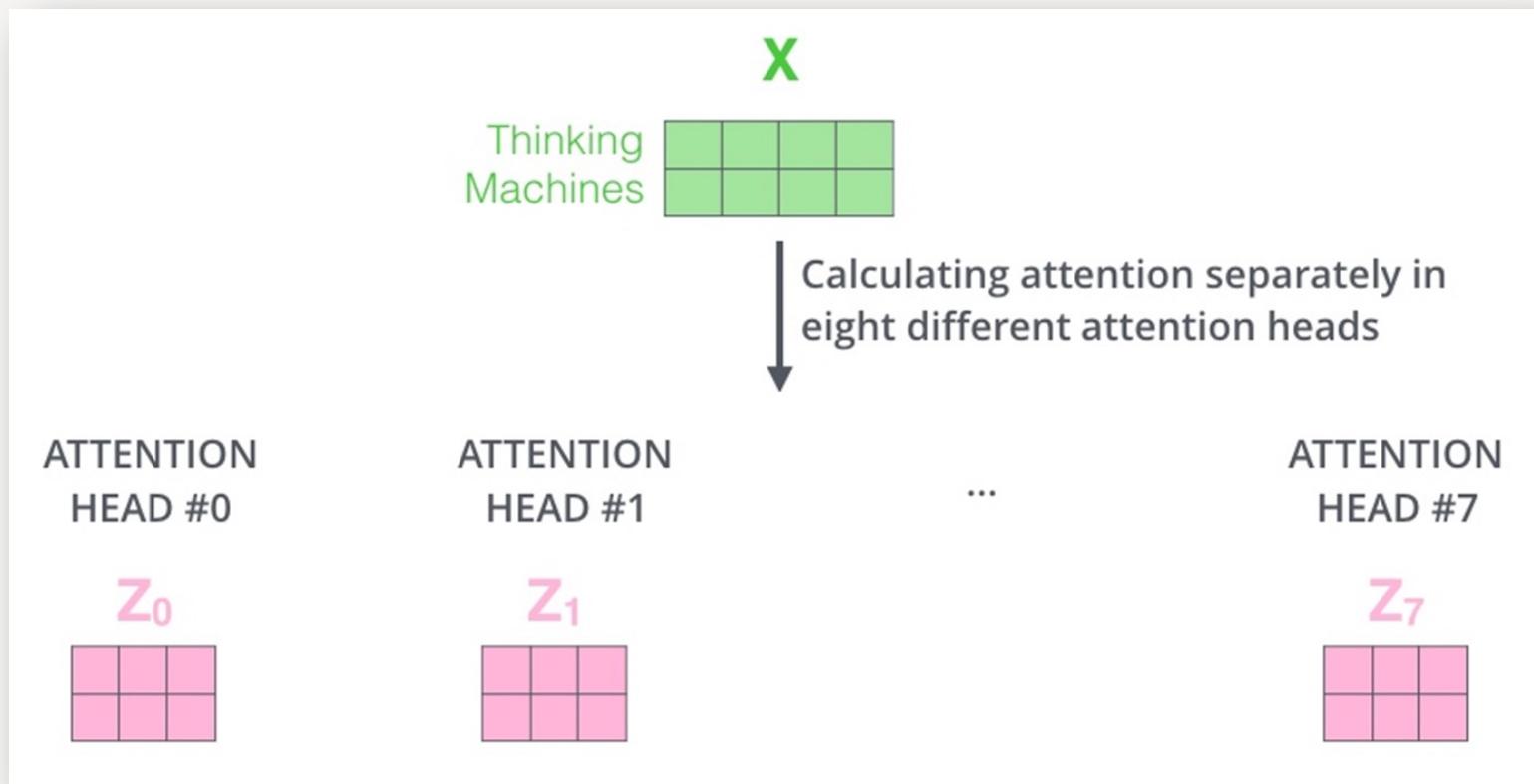
② Multi-head Attention

Multi-head Attention 用不同的线性变换分成不同的head



② Multi-head Attention

每一个head经过Attention，计算得到一个output



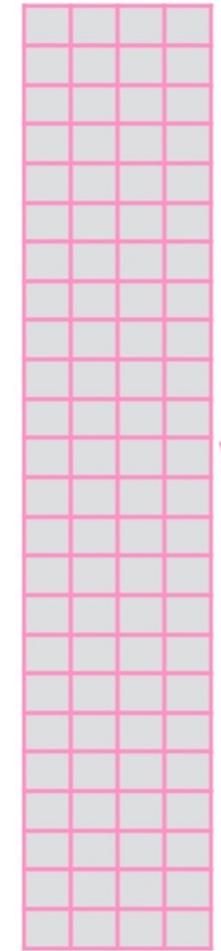
② Multi-head Attention

1) Concatenate all the attention heads



2) Multiply with a weight matrix W^o that was trained jointly with the model

x



Why Multi-head? 用注意力机制提取多重语义

3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

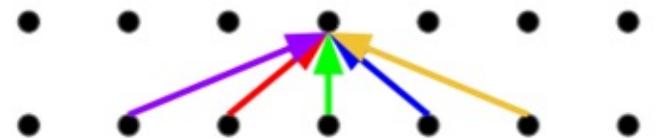
$$= \begin{matrix} Z \\ \hline \end{matrix}$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

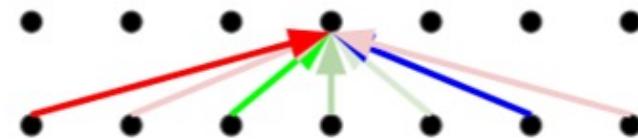
where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

CNN vs Self-attention

Convolution



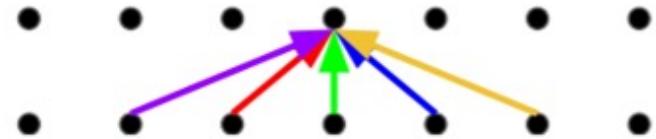
Multi-Head Attention



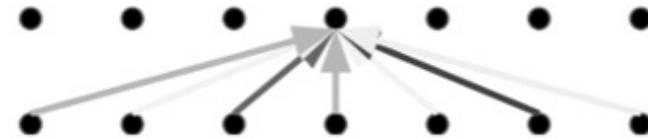
Multi-head Attention

- 类比于CNN的Multi-kernel，Self-attention产生了Multi-head Attention
不同的head可以学到不同的语义、层次关系。
- CNN: Fixed-size Perceptive → Self-attention: Variable-sized Perceptive
一次计算得到文本序列中任意两个元素的相似度，以整个文本作为观察范围。

Convolution



Self-Attention

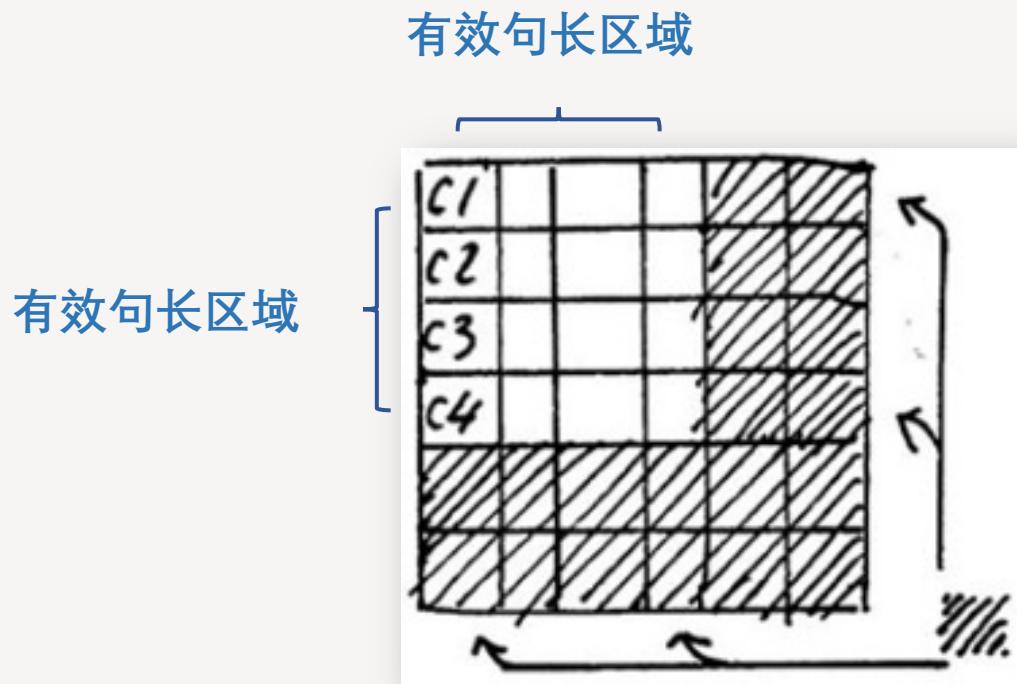


细节：最大句长和有效句长

self attention 的计算过程中, 我们通常使用 *mini batch*。

X 的维度是 [*batch size*, *sequence length*] , 而一个 *mini batch* 是由多个不等长的句子组成的, 我们需要按照 *mini batch* 中最大的句长对剩余的句子进行补齐长度

一般用 0 来进行填充 (*padding*)



但这时在进行 *softmax* 的时候就会产生问题

$$\sigma(z_j) = \frac{e^{z_j}}{\sum_{j=1}^k e^{z_j}} \quad e^0 = 1$$

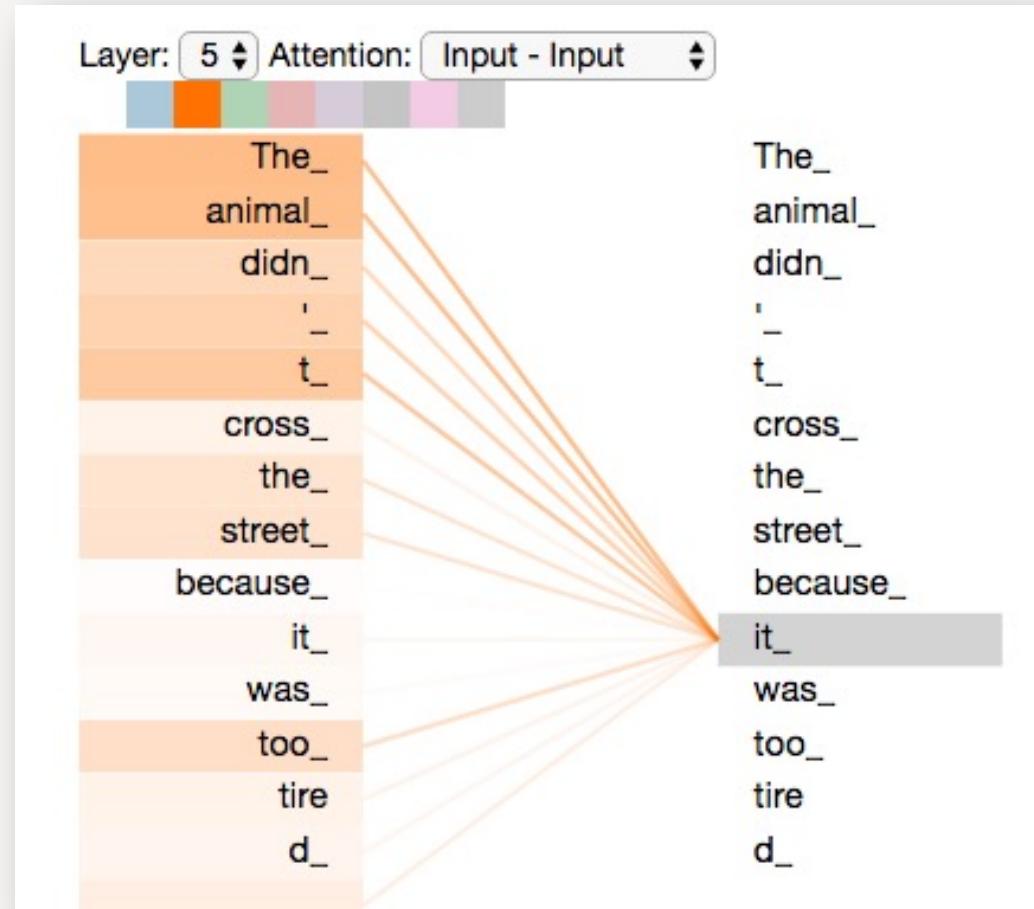
超出句长的无效区域需要 Mask
加一个很大的负数偏置项
这样使无效区域经过运算后几乎还是0

$$Z_{mask} = Z_{mask} + bias_{mask}$$

$$bias_{mask} \rightarrow -\infty$$

$$e^{z_{mask}} \rightarrow 0$$

Attention 效果

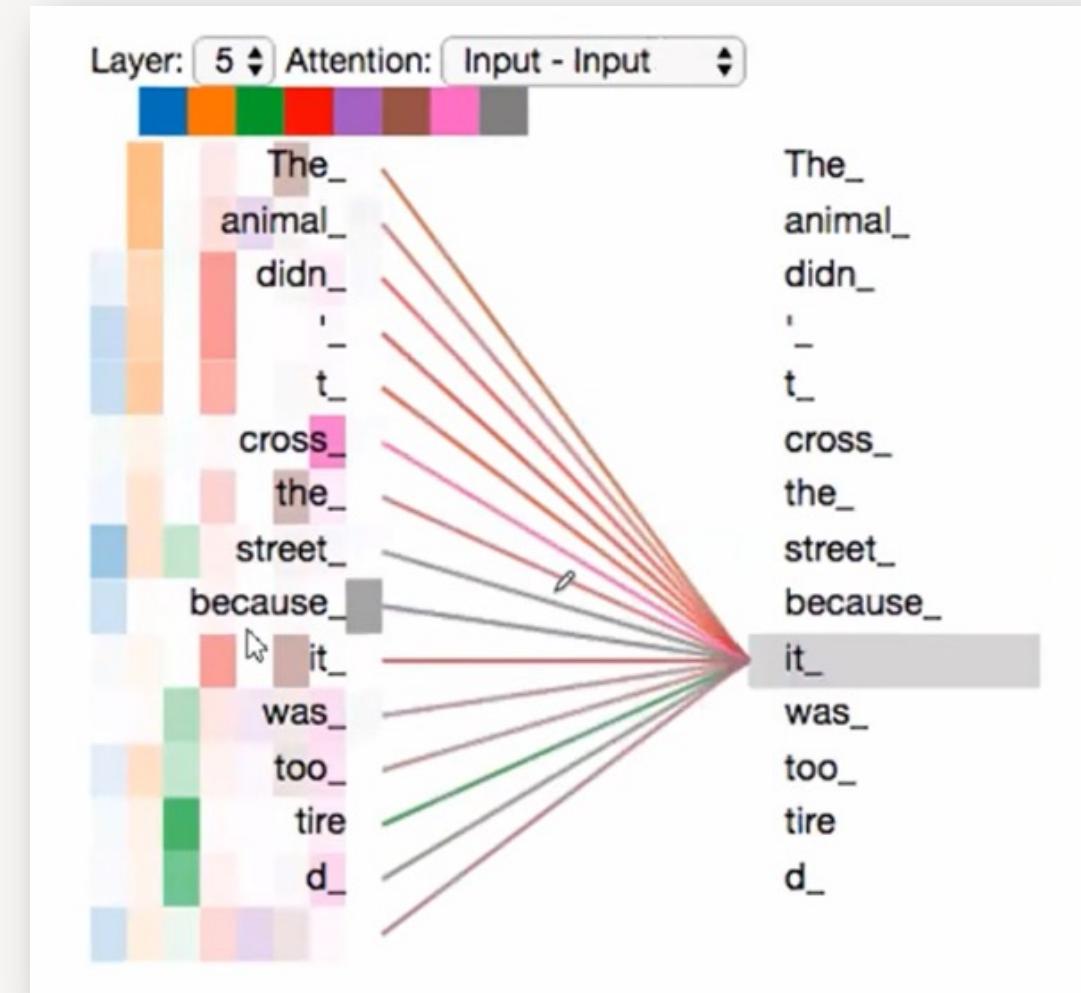
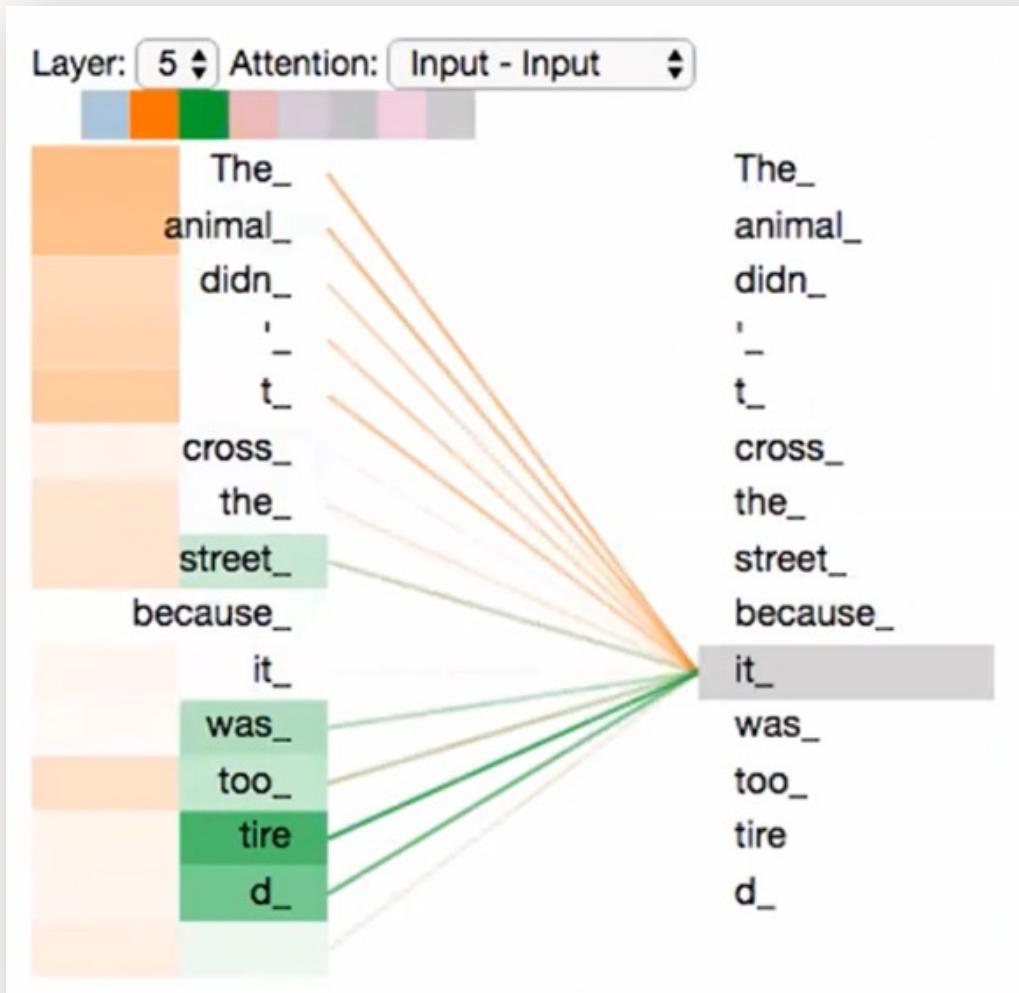


**“The animal didn't cross the street,
because it was too tired.”**

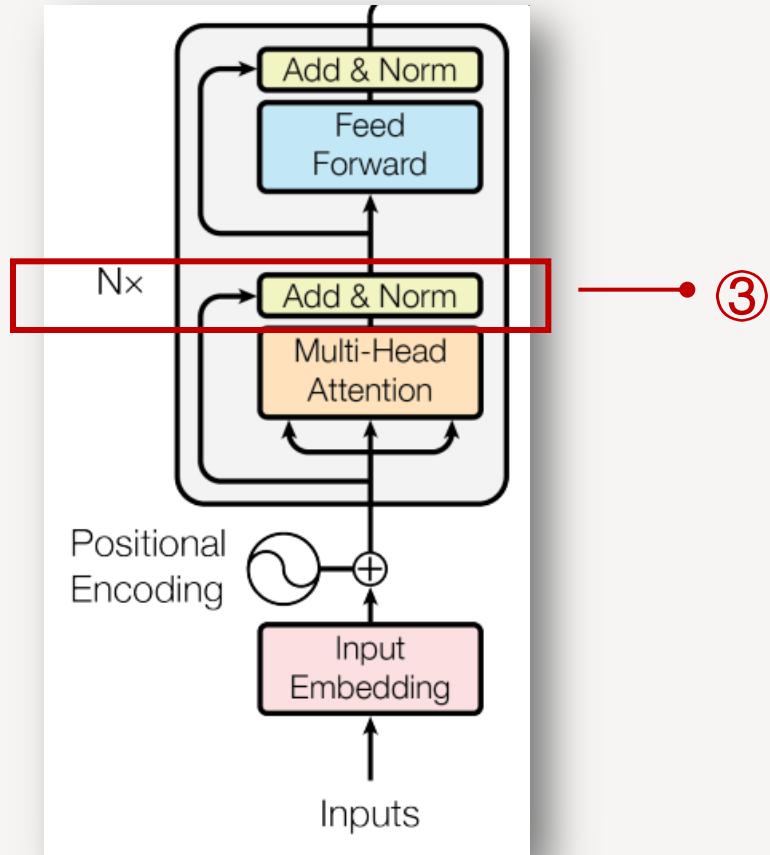
这里的it到底代表的是animal还是street?
对于我们来说能很简单的判断出来，
但是对于机器来说，是很难判断的。

Self-attention就能够让机器把it和animal联系起来

Multi-head Attention 效果



③ Add & Normalization



Transformer编码器

——BERT

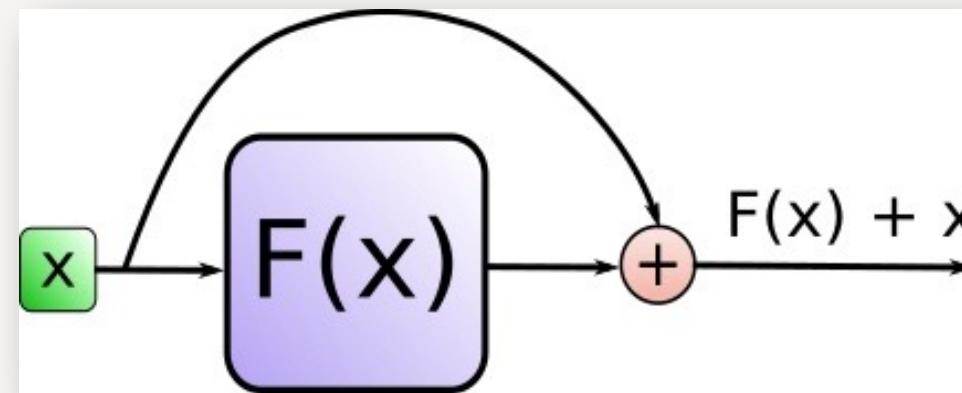
- ① 位置嵌入 Positional Encoding
理解句子里词的顺序
- ② 多头注意力机制 Multi-Head Attention
可以真正实现上下文理解的方法
- ③ 残差连接与归一化 Add & Norm
- ④ 全连接层 Feed Forward

③ Add & Normalization

Add代表了Residual Connection，是为了解决深层网络训练困难的问题。
(通过将前一层的信息无差的传递到下一层，可以有效的仅关注差异部分)

[batch size,
sequence length,
embedding dimension]

$X_{embedding}$



$Attention(Q, K, V)$

$X_{embedding}$
+
 $Attention(Q, K, V)$

③ Add & Normalization

Batch Normalization

1	3	5
2	2	2
0	1	5
4	6	2
4	2	3
1	0	2

3	2.8
2	0
3	2.8
4	2.8
3	1.4
1	1.4

Batch

Mean Std

1	3	4
2	5	2
0	1	5
4	6	2
4	3	3
1	0	2

Mean Std

2 3 3
3.7 5.1 2.8

Layer Normalization

Batch

$$\mu_i = \frac{1}{m} \sum_{i=1}^m x_{ij}$$

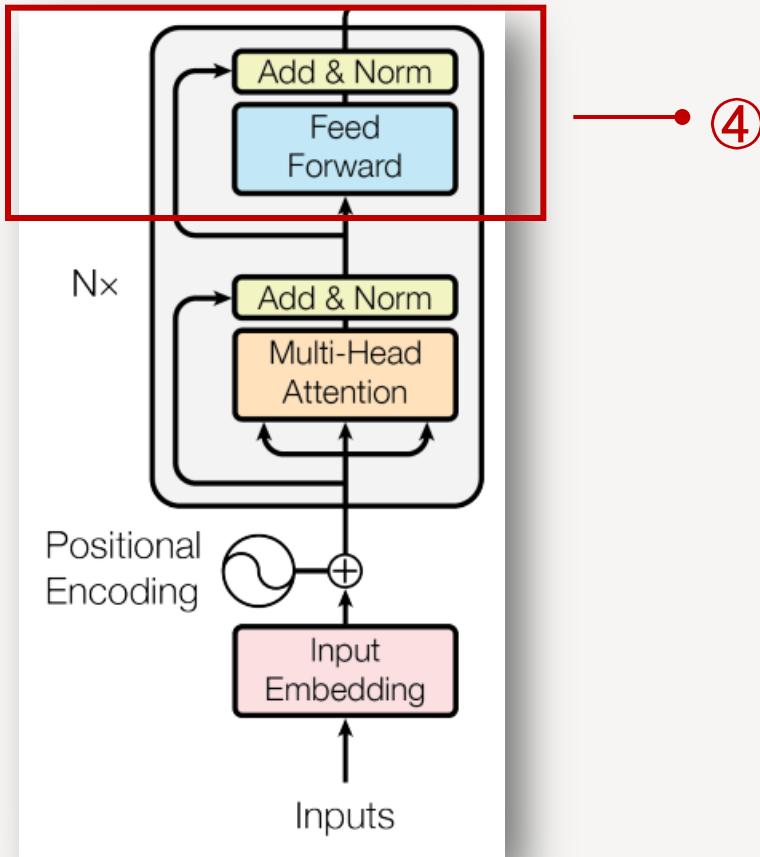
$$\sigma_i^2 = \frac{1}{m} \sum_{i=1}^m (x_{ij} - \mu_i)^2$$

$$LayerNorm(x) = \alpha \odot \frac{x_{ij} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} + \beta$$

α, β 用来弥补归一化过程中损失的信息
一般初始化 α 为全1, β 为全0



④ Feed Forward



Transformer 编码器

——BERT

- ① 位置嵌入 Positional Encoding
理解句子里词的顺序
- ② 多头注意力机制 Multi-Head Attention
可以真正实现上下文理解的方法
- ③ 残差连接与归一化 Add & Norm
- ④ 全连接层 Feed Forward

transformer encoder 整体结构

① 字向量与位置编码

$$X = EmbeddingLookup(X) + Positional Encoding$$

$$X \in \mathbb{R}^{batch.size * seq.len * embed.dim}$$

② 自注意力机制

$$Q = Linear(X) = XW_Q$$

$$K = Linear(X) = XW_K$$

$$V = Linear(X) = XW_V$$

$$X_{attention} = SelfAttention(Q, K, V)$$

③ 残差连接与Layer Normalization

$$X_{attention} = X + X_{attention}$$

$$X_{attention} = LayerNorm(X_{attention})$$

④ Feed Forward 两层线性映射并激活，如ReLU

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

$$X_{hidden} \in \mathbb{R}^{batch.size * seq.len * embed.dim}$$



transformer encoder 整体结构

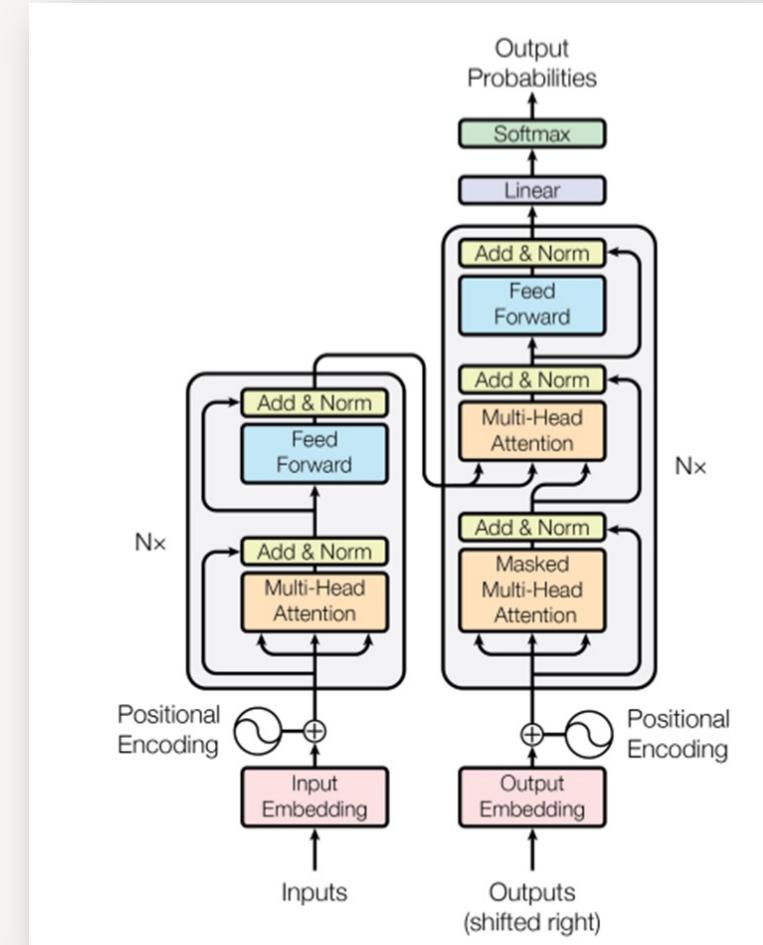
经过自注意力机制

一句话中的每个字都含有这句话中其他字的信息。

那么，我们可不可以添加一个空白字符到句子最前面，
让句子中的所有信息向这个空白字符汇总，
然后再映射成想要分的类别呢？

这就是BERT的预训练目标之一

在BERT的预训练中，
我们给每句话的句头加一个特殊字符，
用句头的特殊字符的 *hidden state* 完成分类任务。



• T h a n k s •

学生创新中心：肖雄子彥



上海交通大学
SHANGHAI JIAO TONG UNIVERSITY



学生创新中心
Student Innovation Center