目录

[1 Attention 2](#_Toc14195729)

[1.1 Encoder-Decoder结构 2](#_Toc14195730)

[1.2 Soft Attention 2](#_Toc14195731)

[1.2.1 E-D结构的局限性 2](#_Toc14195732)

[1.2.2 Attention机制的应用与计算方式 2](#_Toc14195733)

[1.2.3 Attention机制的本质思想 3](#_Toc14195734)

[2 Self Attention 5](#_Toc14195735)

[2.1 Self Attention思想 5](#_Toc14195736)

[2.1.1 Self与Soft的差异 5](#_Toc14195737)

[2.1.2 与RNN相比的优势 5](#_Toc14195738)

[2.2 Self Attention的一种简单实现 5](#_Toc14195739)

[2.3 结合翻译任务分析Self Attention 6](#_Toc14195740)

[3 Transformer模型结构 7](#_Toc14195741)

[3.1 Scaled Dot-Product Attention 7](#_Toc14195742)

[3.2 Multi-HeadAttention 7](#_Toc14195743)

[3.3 Transformer结构 9](#_Toc14195744)

[3.4 Encoder结构与数据流 10](#_Toc14195745)

[3.5 Decoder结构与数据流 12](#_Toc14195746)

[3.6 Encoder和Decoder内部的残差连接 14](#_Toc14195747)

[3.7 最终的输出部分 15](#_Toc14195748)

# Attention

## Encoder-Decoder结构



图 1.1E-D结构

一般用于句子对<Source,Target>，我们的目标是给定输入句子Source，期待通过Encoder-Decoder框架来生成目标句子Target。Source和Target可以是同一种语言，也可以是两种不同的语言。而Source和Target分别由各自的单词序列构成：

计算公式：

编码器输出的中间编码

解码器输出的每一维

## Soft Attention

### E-D结构的局限性

由的计算方式可以看出，句子Source中任意单词对生成某个目标单词来说影响力都是相同的，都是根据相同的C计算出来的。而实际上，不同位置的对的影响都是不同的。

比如“Tom chase Jerry”—>“汤姆 追逐 杰瑞”，对的影响明显要比和大。

### Attention机制的应用与计算方式

计算每个的时候，都考虑到Source中每个单词的权重(注意力分配概率)信息，这时每个计算时的C就不再相同了，如图1.2所示。

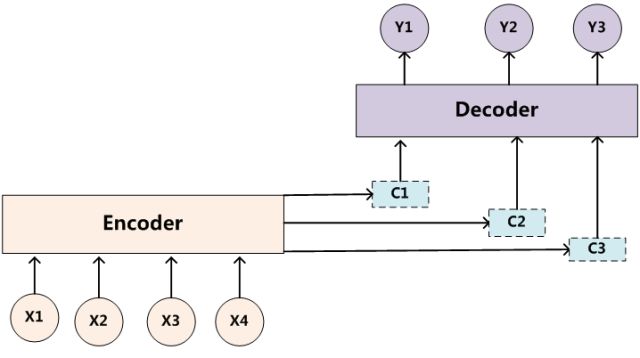


图 1.2加入Attention之后的E-D结构

计算公式：

代表相对与的权重，或者叫注意力分配系数

函数h是对输入的某种转换函数，比如Encoder用的RNN模型,h函数的结果往往是某个时刻输入后隐层节点的状态值。

在解码器中其实就是根据C和计算出来的状态值。

这里，所有相对于的权重向量的计算方式如下：

### Attention机制的本质思想

抛开Encoder-Decoder模型，Attention机制的本质思想如图1.3：

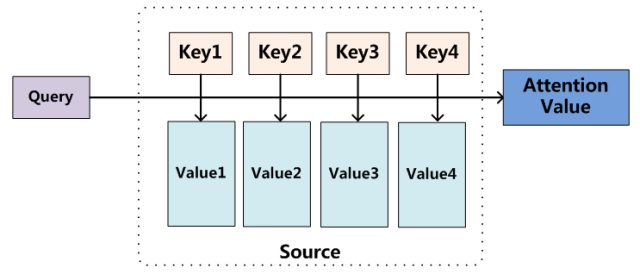


图 1.3Attention机制

其中，Source可以看成由一系列<Key,Value>数据对组成， Query为Target中要预测的元素，Attention的目的就在于通过Query和Key计算出相应Value对于Query而言的权重系数W，最后根据W对Vaule值加权求和就得到了Attention Value。

在很多E-D结构中，Key和Value是相同的，都使用的是词的编码表示。

# Self Attention

## Self Attention思想

### Self与Soft的差异

Soft Attention是为了计算Target中与Source中所有之间的关系

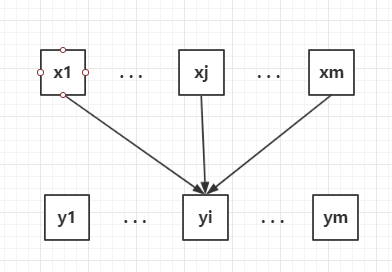


图 2.1 Soft Attention思想

Self Attention则是为了计算Source中所有之间的相互关系

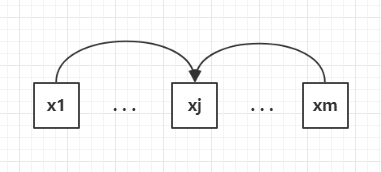


图 2.2Self Attention思想

### 与RNN相比的优势

RNN模型在计算时是不断的循环累积，距离越远的单词保留的信息就越少，在处理中长句时很难捕捉远距离单词之间的联系。

Self Attention计算的则是句中的每个单词与其他所有单词之间的关系，因此摆脱了远距离依赖问题。

## Self Attention的一种简单实现

《Attention is all you need》中提出了Self Attention的一种实现：Scaled Dot-Product Attention，结构如图2.3所示。



图 2.3Self Attention的一种实现

传统的Soft Attention中，K=V为Source中词的编码表示，Q为Target中词的编码表示。但在Self Attention中，Q=K=V都是Source中词的编码表示，学习的是Source内部词与词之间的联系。

## 结合翻译任务分析Self Attention

在翻译任务中，E-D结构计算出的中间语义编码C可以理解成语言在更高层次上的表达，对具体不同的语言都具有普适性。而Self Attention所学习的也正是句子内部词与词之间的联系，可以理解为句子的语法语义等高级信息，和中间语义编码C的性质就比较相似。



# Transformer模型结构

## Scaled Dot-Product Attention

一种特殊的Attention结构，输入三个矩阵：Q(query)、K(key)、V(value)。



图 3.1 Scaled Dot-Product Attention

**数学公式：**

计算公式：

其中：

1. Scale层是正规化，因为Q和K的点积会使得矩阵元素值都变得比较大，所以要正规化缩小一下，防止梯度爆炸
2. 的计算结果通常叫做Attention Score

## Multi-HeadAttention

并行多个Scaled Dot-Product Attention模块，然后将结果融合，也叫做自注意力(self-Attention)模型。

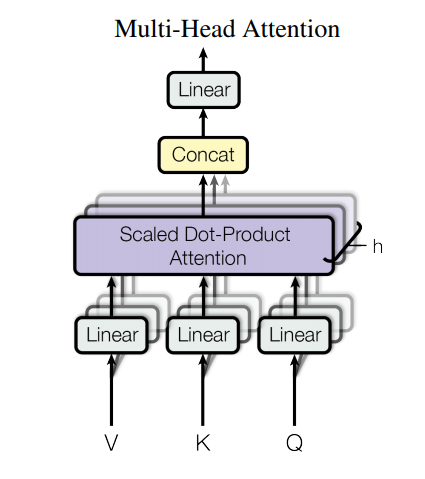


图 3.2Multi-HeadAttention

1. Q、K、V经过全连接层后的输出维度不变，然后在特征这个维度上均分为h份，分别进入到不同的Scaled Dot-Product Attention中进行计算
2. 每个head只学习元素数据中的部分特征维度的信息
3. 结果再使用Concat层进行拼接

具体计算方式：

原始维度如果是：

切分后就是：

可以只用一个Scaled Dot-Product Attention进行计算，然后将结果的后两个维度拼接在一起

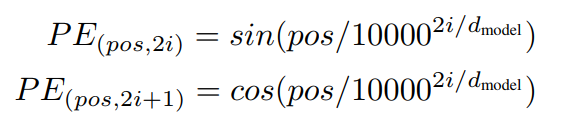
## Transformer结构

网络模型结构如图3.3所示。



图 3.3Transformer结构

1. Add: 矩阵加法 Norm:正规化
2. Positional Encoding：位置编码，引入上下文信息，本文使用三角函数进行计算，公式如下：



1. 左侧的Multi-HeadAttention+Feed Forward构成一个Encoder模块
2. 右侧的两个Multi-HeadAttention+Feed Forward构成一个Decoder模型

在论文中，N取值为6，因此Transformer结构的结构也可以表示成图3.4所示。

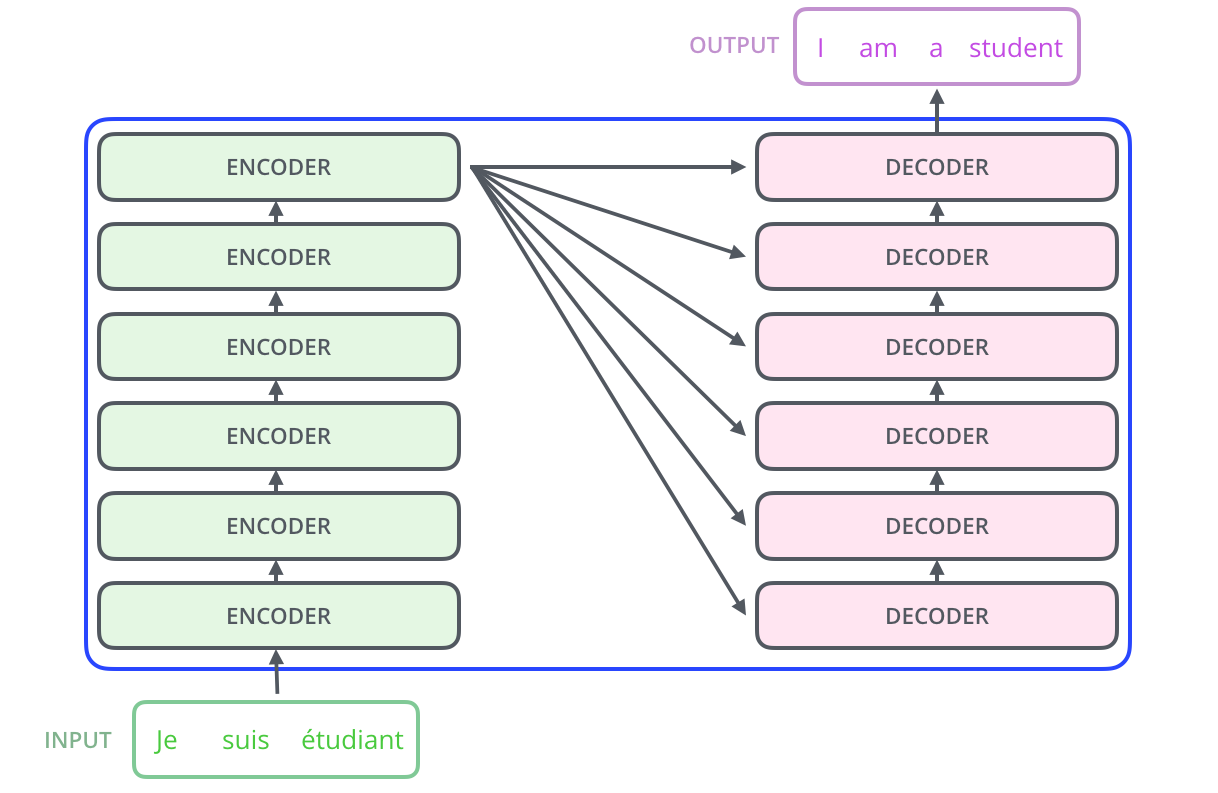


图 3.4Transformer的另一种表示

由上图可以看出，Encoder和Decoder都是一层层串联的，最后一层Encoder的结果会传给所有Decoder作为部分输入。

## Encoder结构与数据流

整个Encoder的数据可以分为三种，其中X为Encoder的输入，Z为Encoder内部Self-Attention的输出，R为整个Encoder最终的输出。

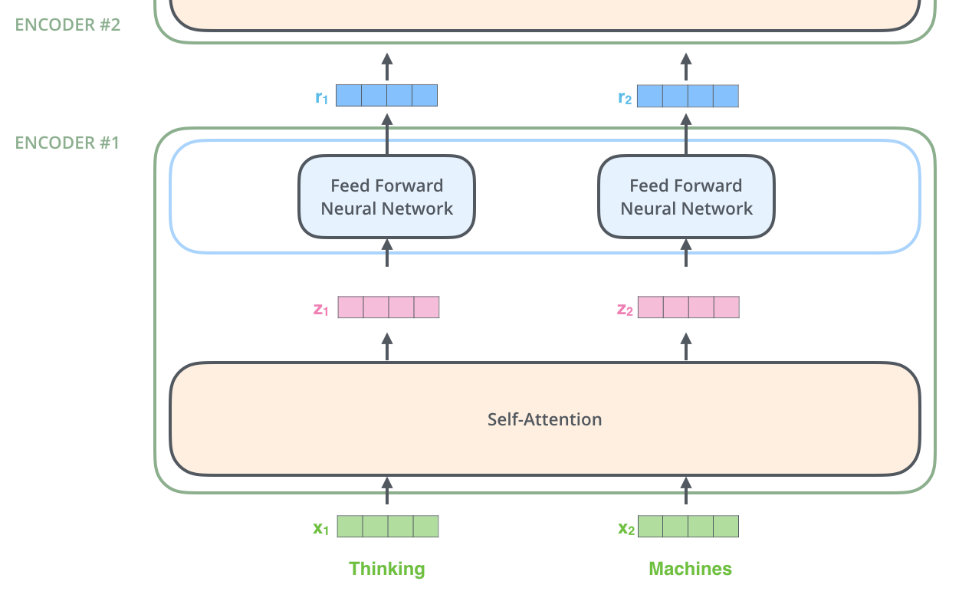


图 3.5 Encoder数据流

Z的具体计算方式如图3.7所示。

1. 输入矩阵X经过三个全连接层产生Q、K、V三个矩阵
2. 通过公式计算出V对应的权重矩阵W
3. V和W进行运算得到Z

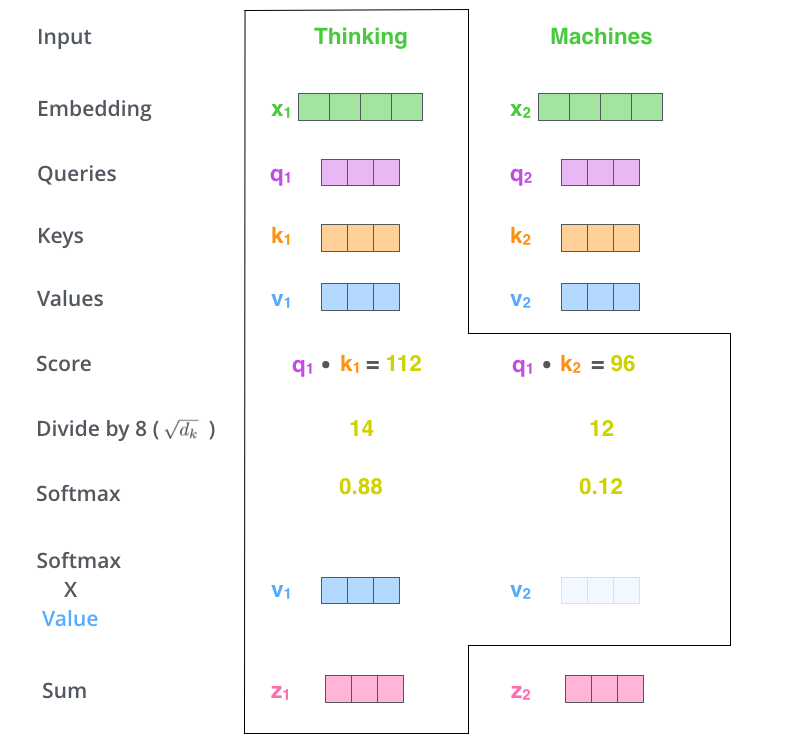


图 3.6单一Self-Attention的计算流程

上述只是单一Self-Attention的计算流程，在Transformer中使用的是多头Self-Attention，也就是会产生多个Z，Transformer中Z的个数为8，还需要对多个Z进行融合再输入到Feed Forward层，如图3.8所示。

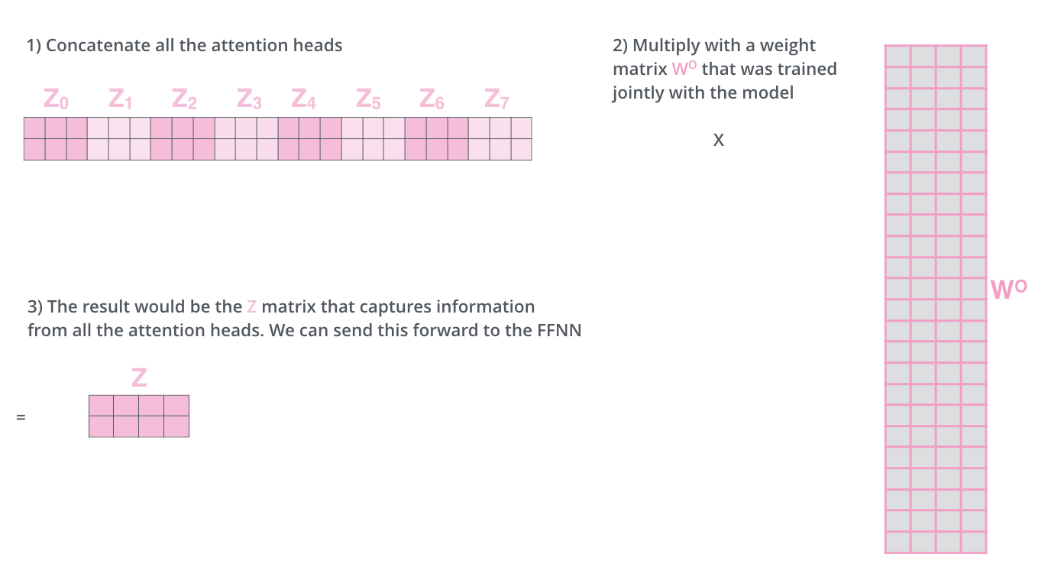


图 3.7多头Attention融合方式

## Decoder结构与数据流

经过6层Encoder模块的计算，最终得到了输入Source的中间编码表示C，然后将其经过两个全连接层就可以得到Decoder模块所需的K和V，如下图。

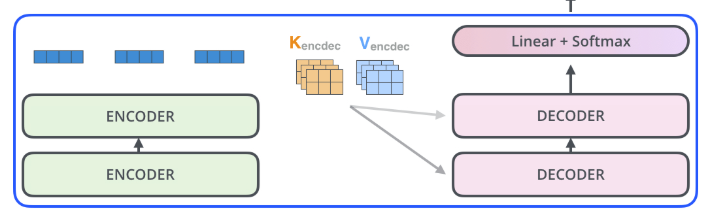


图 3.8 Encoder与Decoder的关联

其中，

1. Encoder产生的K、V传递给每个Decoder模块的第二个Self-Attention层
2. 第二个Self-Attention层的Q来自于第一个Self-Attention层的输出
3. 第一个Self-Attention层的Q、K、V则由上一个Decoder产生的结果生成
4. 第一个Decoder的第一个Self-Attention层的Q、K、V则由Target(也就是Output)的编码表示经过全连接层产生

如下图所示



图 3.9 Transformer结构

假设最终的，对于Transformer中第一个Decoder而言：

1. 一开始输入第一层Self-Attention的是(特殊的标志字段)的编码表示，输出
2. 第二层Self-Attention的输入是对应的中间编码和
3. 最终计算出的编码表示
4. 然后输入第一层Self-Attention的是的编码表示，输出
5. 第二层Self-Attention的输入是对应的中间编码和
6. 最终计算出的编码表示
7. 重复上述循环就能计算出预测的

## Encoder和Decoder内部的残差连接

Transformer的每个Encoder和Decode模块内部都是具有残差结构的，不过为了便于理解，前两节就未加以描述。

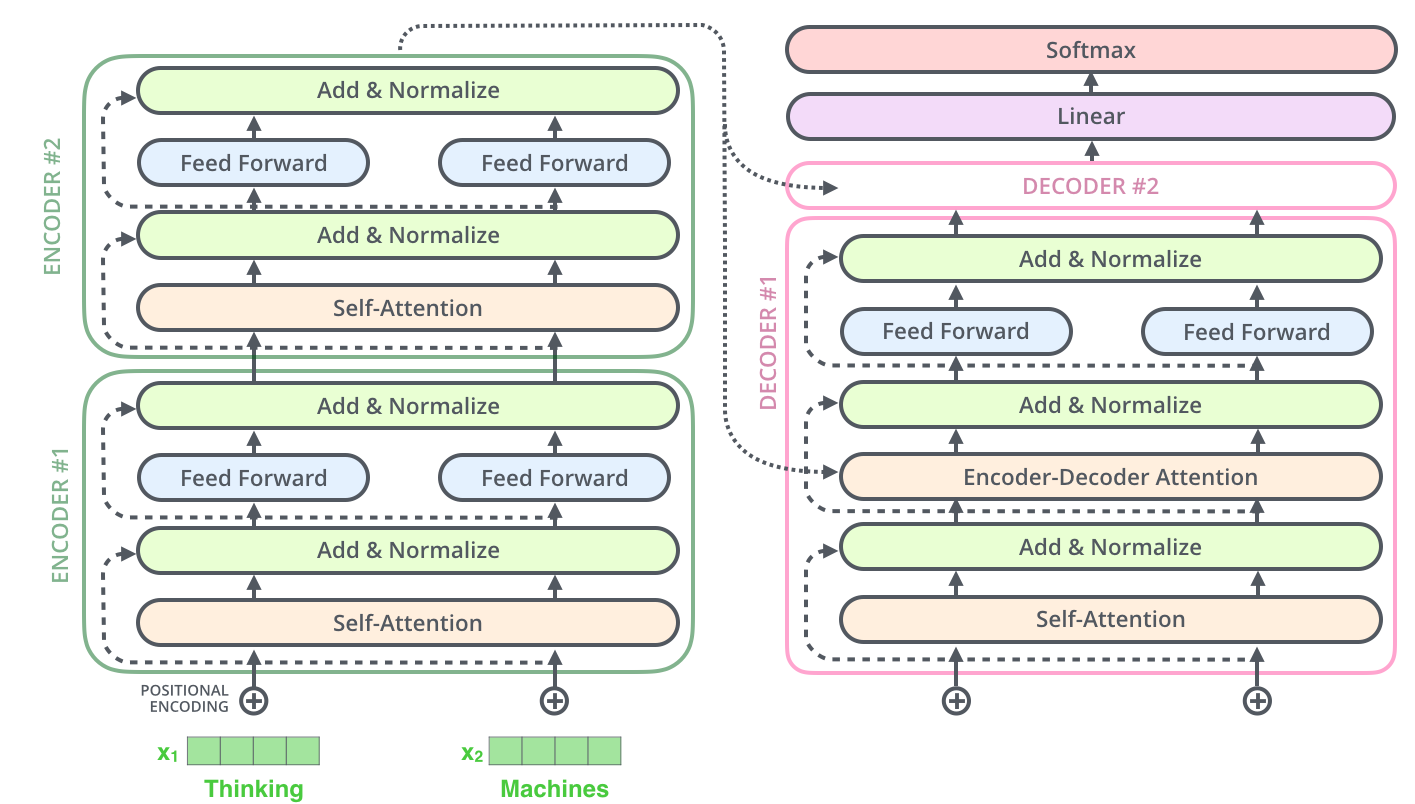


图 3.10E-D模块中的残差结构

其中，Add & Normalize就是残差层，它将上一层(Self-Attention/Feed Forward)的输入与上一层的输出进行有选择的融合，如下图所示。

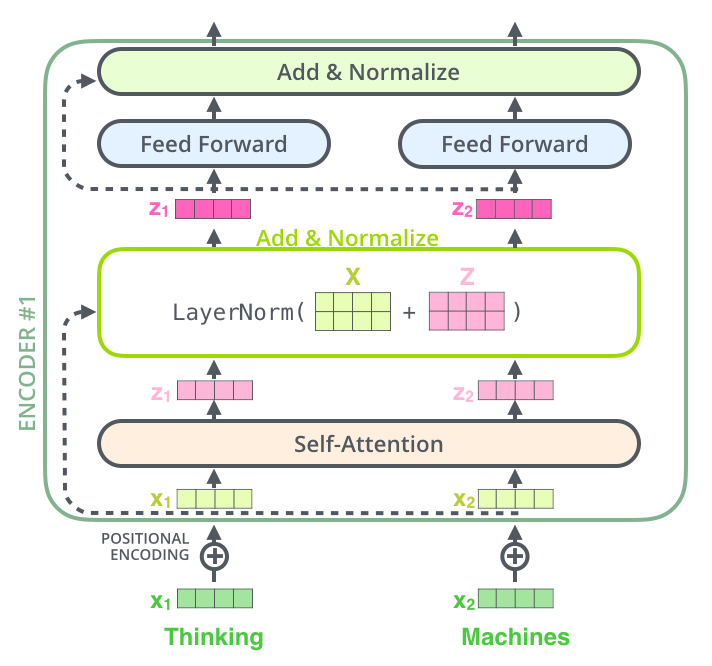


图 3.11Encoder模块内部的残差结构

## 最终的输出部分

前面已经介绍了Decoder每次只是用于生成一个预测单词，因此将其输出的最终向量经过全连接层和softmax层就可以知道预测的是哪一个词，如下图。

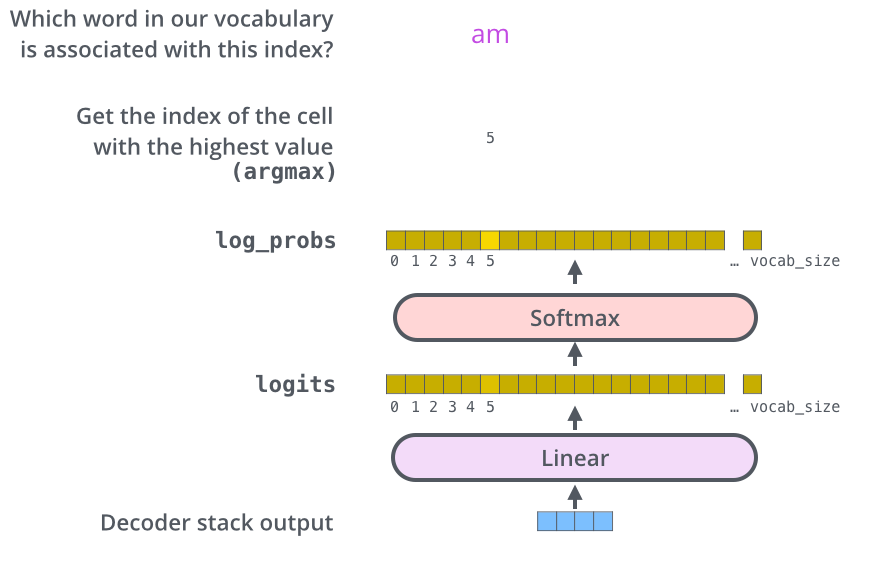


图 3.12最终的输出层

1. 假设有10000个单词，vocab\_size就等于10000
2. 经过softmax层就可以得到预测单词是这10000个单词中每个词的概率，取其最大值即可。

## 数据训练

对于Encoder，输入的是已知的Source(X)，因此不需要考虑Masked的问题，全部置1，代表直接计算，不进行影响。

词向量 Mask Encoder 输出

的词向量 的词向量

…

的词向量 的词向量

对于Decoder，第一层的第一个是Masked Attention，对于每个预测向量，它只能参考已知的

词向量 Mask Dcoder 输出

的词向量 的词向量

…

的词向量 的词向量

这里Attention内部的计算如下：

1. 先正常计算与所有之间的关系，与Encoder的Attention一样
2. 然后再将对应的向量全部清零(极小值)
3. 最后经过softmax算出权重，与V相乘，从Attention中输出

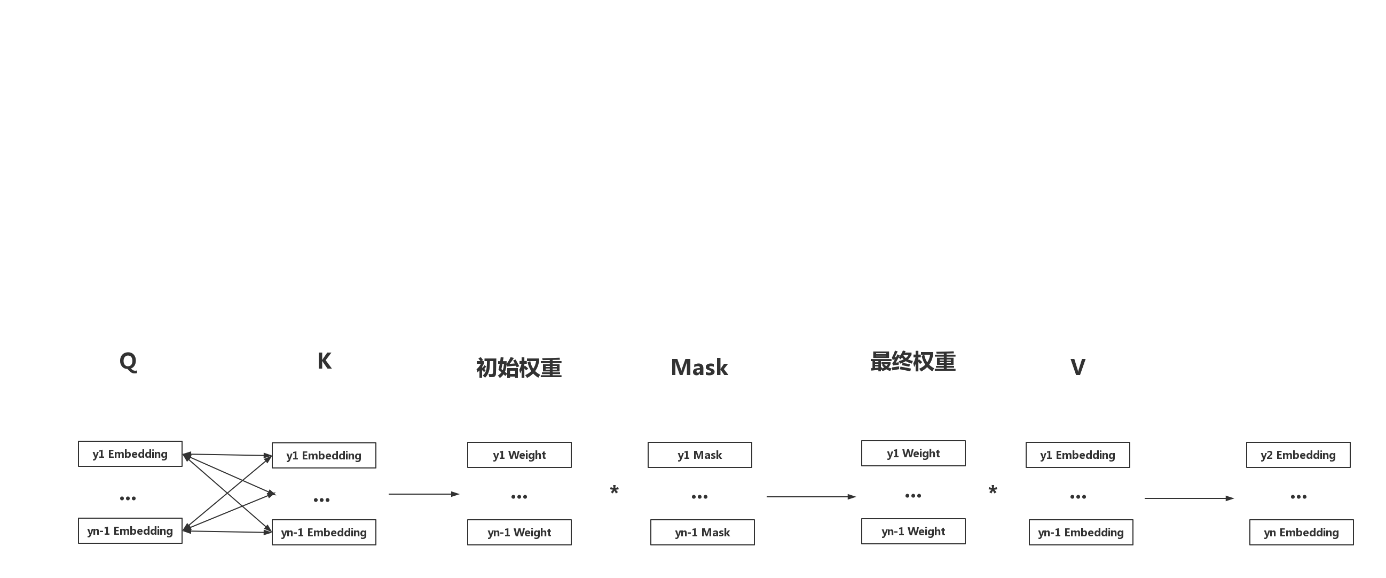


图 3.13Masked Attention计算流程

不管是Encoder还是Dcoder，都是在一次矩阵运算中就将所有结果输出。因为两者前面的计算是相同的，都是根据Q、K计算出权重，一次运算就可以。后面Decode只需将部分权重清零即可，这就是Transformer的并行性。