BERT算法又是基于Transformer，Transformer又是基于attention机制

Encoder-Decoder->Attention->Transformer

2.Encoder-Decoder

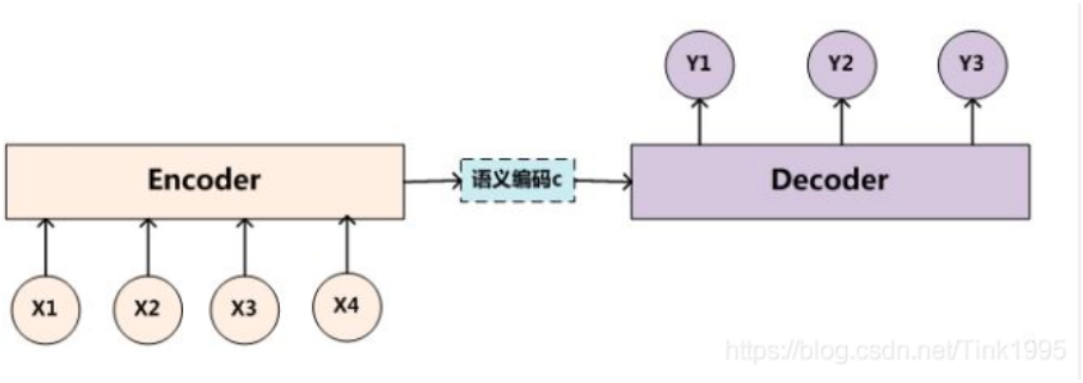
2.1Encoder-Decoder简介

编码-解码框架，目前大部分attention模型都是依附于Encoder-Decoder框架进行实现，在NLP中Encoder-Decoder框架主要被用来处理序列-序列问题。也就是输入一个序列，生成一个序列的问题。这两个序列可以分别是任意长度。

具体到NLP中的任务比如：
文本摘要，输入一篇文章(序列数据)，生成文章的摘要(序列数据)
文本翻译，输入一句或一篇英文(序列数据)，生成翻译后的中文(序列数据)
问答系统，输入一个question（序列数据），生成一个answer（序列数据）

基于Encoder-Decoder框架具体使用什么模型实现，用的较多的应该就是seq2seq模型和Transformer了。

2.2Encoder-Decoder结构原理



上图就是Encoder-Decoder框架在NLP领域中抽象后的最简单的结构图。

Encoder：编码器，对于输入的序列<x1,x2,x3...xn>进行**编码**，使其转化为一个语义编码C，这个C中就储存了序列<x1,x2,x3...xn>的信息。

编码方式有很多种，在文本处理领域主要有**RNN**/LSTM/GRU/BiRNN/BiLSTM/BiGRU。

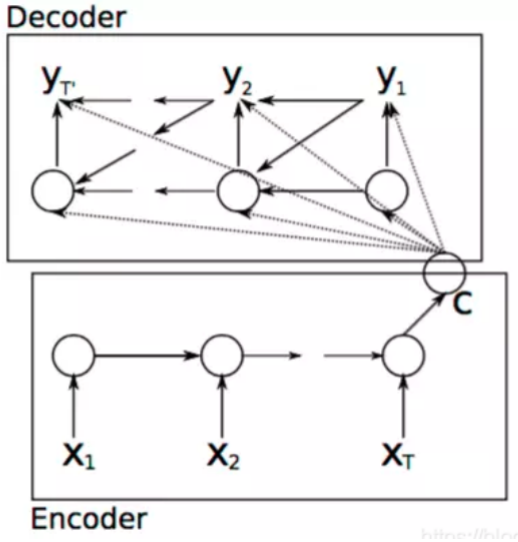
以RNN为例说明一下：
以上图为例，输入<x1,x2,x3,x4>，通过RNN生成隐藏层的状态值<h1,h2,h3,h4>，如何确定语义编码C呢？最简单的办法直接用最后时刻输出的ht作为C的状态值，这里也就是可以用h4直接作为语义编码C的值，也可以将所有时刻的隐藏层的值进行汇总，然后生成语义编码C的值，这里就是C=q(h1,h2,h3,h4)，q是非线性激活函数。

得到了语义编码C之后，接下来就是要在Decoder中对语义编码C进行解码了。

Decoder：解码器，根据输入的语义编码C，然后将其解码成序列数据，解码方式也可以采用RNN/LSTM/GRU/BiRNN/BiLSTM/BiGRU。Decoder和Encoder的编码解码方式可以任意组合，并不是说Encoder使用了RNN，Decoder就一定也需要使用RNN才能解码，Decoder可以使用LSTM，BiRNN这些。
Decoder如何解码：

基于seq2seq模型的解码方式：

[论文1]Cho et al., 2014 . Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation.



论文中指出，因为语义编码C包含了整个输入序列的信息，所以在解码的每一步都引入C。

普通的Encoder-Decoder结构存在很多问题，

在生成目标句子的单词时，不论生成哪个单词，他们使用的语义编码C都是一样的，没有任何区别。而语义编码C是由输入序列X的每个单词经过Encoder 编码产生的，这意味着不论是生成哪个单词，y1,y2还是y3，其实输入序列X中任意单词对生成某个目标单词yi来说影响力都是相同的，没有任何区别，其实如果Encoder是RNN的话，理论上越是后输入的单词影响越大，并非等权的，

用一个语义编码C来记录整个序列的信息，序列较短还行，如果序列是长序列，会出现很多问题。

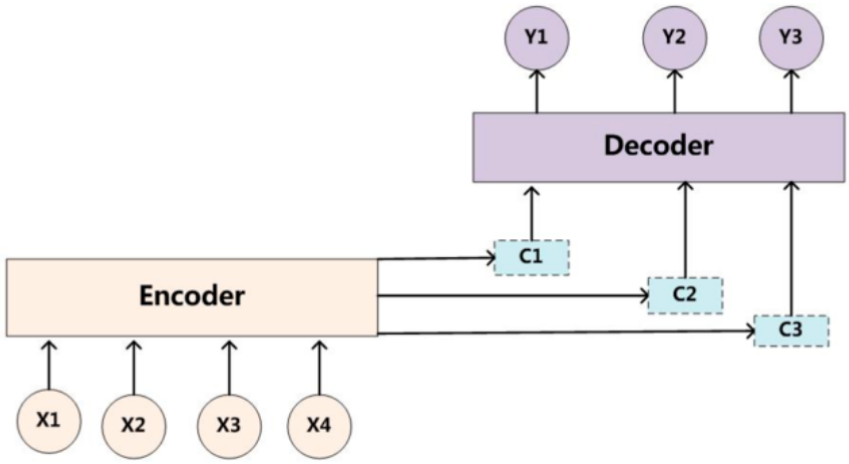
既然一个C不行，就用多个C，诶~这个时候基于Encoder-Decoder的attention model就出现了。

3.Attention Model

3.1Attention 注意力机制简介

同一张图片，不同的人观察注意到的可能是不同的地方，这就是人的注意力机制。attention 就是模仿人的注意力机制设计地。

3.2Attention 原理



上图就是引入了Attention 机制的Encoder-Decoder框架。

上图不再只有一个单一的语义编码C，而是有多个C1,C2,C3这样的编码。当我们在预测Y1时，可能Y1的注意力是放在C1上，就用C1作为语义编码，当预测Y2时，Y2的注意力集中在C2上，就用C2作为语义编码，以此类推，就模拟了人类的注意力机制。

怎么计算出C1，C2，C3...Cn呢?

以机器翻译例子"Tom chase Jerry" - "汤姆追逐杰瑞"来说明:
当我们在翻译"杰瑞"的时候，为了体现出输入序列中英文单词对于翻译当前中文单词不同的影响程度，比如给出类似下面一个概率分布值:
（Tom,0.3） （Chase,0.2） （Jerry,0.5）
每个英文单词的概率代表了翻译当前单词“杰瑞”时，注意力分配模型分配给不同英文单词的注意力大小。
生成每个单词Yi的时候，原理固定的C换成了根据当前输出单词来调整成加入注意力模型的变化的Ci。

而每个**Ci**可能对应着不同的源语句子单词的注意力分配概率分布，比如对于上面的英汉翻译来说，其对应的信息可能如下:

$$C_{汤姆} = g(0.6 * f2("Tom"), 0.2 * f2(Chase), 0.2 * f2("Jerry"))$$

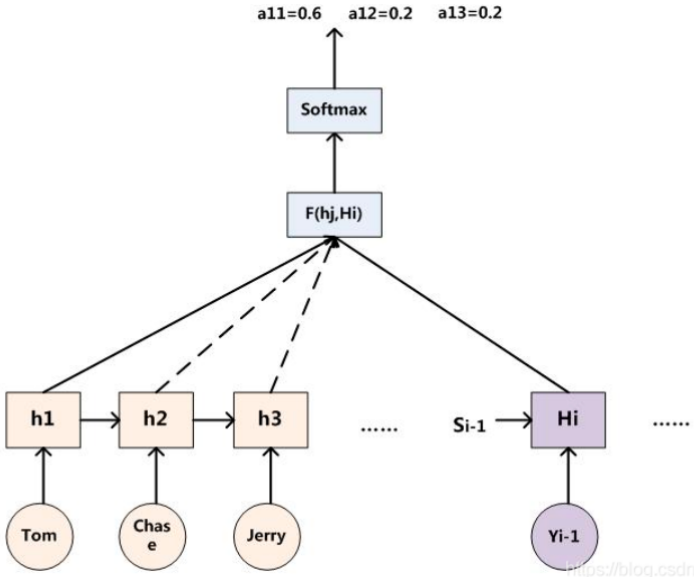
$$C_{追逐} = g(0.2 * f2("Tom"), 0.7 * f2(Chase), 0.1 * f2("Jerry"))$$

$$C_{杰瑞} = g(0.3 * f2("Tom"), 0.2 * f2(Chase), 0.5 * f2("Jerry"))$$

f2就对应着隐藏层的值，g函数就是加权求和。αi表示权值分布

$$C_i = \sum_{j=1}^{n} \alpha_{ij} h_j$$

怎么知道attention模型所需要的输入句子单词注意力分配概率分布值呢？也就是αij?



decoder上一时刻的输出值Yi-1与上一时刻传入的隐藏层的值Si-1通过RNN生成Hi，然后计算Hi与h1，h2，h3...hm的相关性，得到相关性评分[f1,f2,f3...fm]，然后对Fi进行softmax就得到注意力分配αij。然后将encoder的输出值h与对应的概率分布αij进行点乘求和，就能得到注意力attention值了。
缺点:
1.只能在Decoder阶段实现并行运算，Encoder部分依旧采用的是RNN，LSTM这些按照顺序编码的模型，Encoder部分还是无法实现并行运算，不够完美。

2.就是因为Encoder部分目前仍旧依赖于RNN，所以对于中长距离之间，两个词相互之间的关系没有办法很好的获取。

为了改进上面两个缺点，更加完美的Self-Attention出现了。

3.3Attention 机制的本质思想

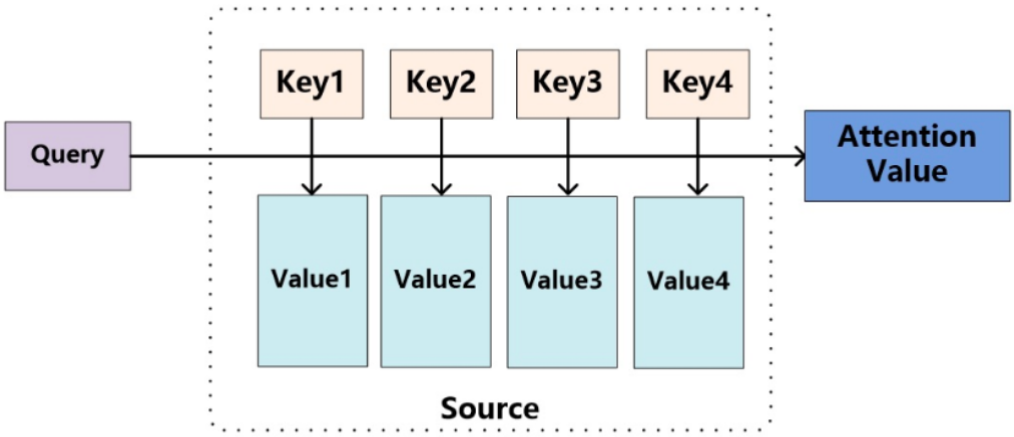


图9 Attention机制的本质思想

将Source中的构成元素想象成是由一系列的<Key,Value>数据对构成（对应到咱们上里面的例子，key和value是相等地，都是encoder的输出值h），此时给定Target中的某个元素Query（对应到上面的例子也就是decoder中的hi），通过计算Query和各个Key的相似性或者相关性，得到每个Key对应Value的权重系数，然后对Value进行加权求和，即得到了最终的Attention数值。

4.Self-Attention

在一般任务的Encoder-Decoder框架中，输入Source和输出Target内容是不一样的，比如对于英-中机器翻译来说，Source是英文句子，Target是对应的翻译出的中文句子，Attention机制发生在Target的元素和Source中的所有元素之间。而Self Attention顾名思义，指的不是Target和Source之间的Attention机制，而是Source内部元素之间或者Target内部元素之间发生的Attention机制，也可以理解为Target=Source这种特殊情况下的注意力计算机制。其具体计算过程是一样的，只是计算对象发生了变化而已。

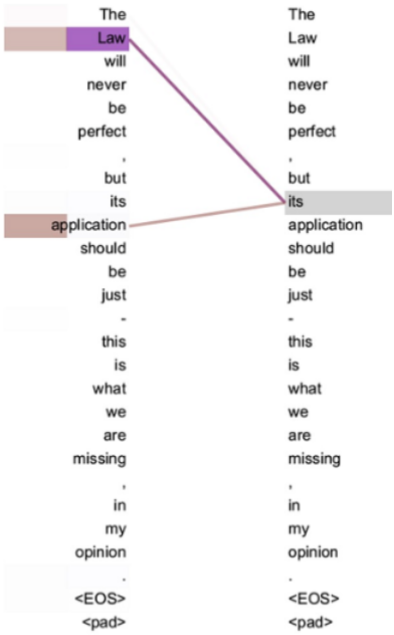


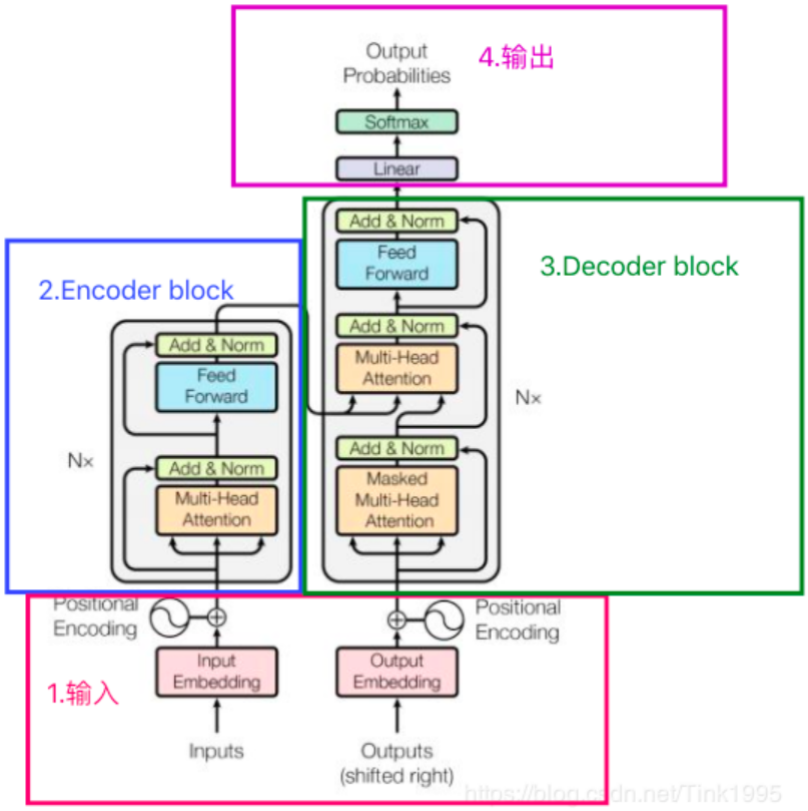
图12 可视化Self Attention实例

我们想知道这句话中的its，在这句话里its指代的是什么，与哪一些单词相关，那么就可以将its作为Query，然后将这一句话作为Key和Value来计算attention值，找到与这句话中its最相关的单词。通过self-attention我们发现its在这句话中与之最相关的是Law和application，通过我们分析语句意思也十分吻合。

https://www.cnblogs.com/miners/p/15101283.html self-attention的代码实现

2.Transformer 原理

2.1 Transformer整体结构



https://blog.csdn.net/Tink1995/article/details/105080033

Transformer是一个基于Encoder-Decoder框架的模型

2.2 Transformer的inputs 输入

Transformer输入是一个序列数据

以"Tom chase Jerry" 翻译成中文"汤姆追逐杰瑞"为例，

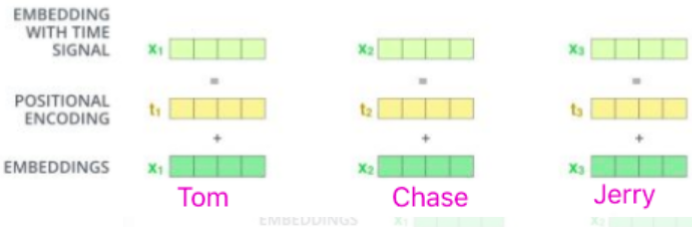
Encoder 的 inputs就是"Tom chase Jerry" 分词后的词向量。可以是任意形式的词向量，如word2vec，GloVe，one-hot编码。

输入inputs embedding后需要给每个word的词向量添加位置编码positional encoding，为什么需要添加位置编码？

一句话中同一个词，如果词语出现位置不同，意思可能发生翻天覆地的变化，就比如：我欠他100W 和他欠我100W。可见获取词语出现在句子中的位置信息是一件很重要的事情。

Transformer 的是完全基于self-Attention地，而self-attention是不能获取词语位置信息。所以在我们输入的时候需要给每一个词向量添加位置编码。

这个positional encoding怎么获取？可以通过数据训练学习得到positional encoding，类似于训练学习词向量，goole在之后的bert中的positional encoding便是由训练得到。



positional encoding与词向量相加，拼接相加都可以，只是本身词向量的维度512维就已经蛮大了，再拼接一个512维的位置向量，变成1024维，这样训练起来会相对慢一些，影响效率。

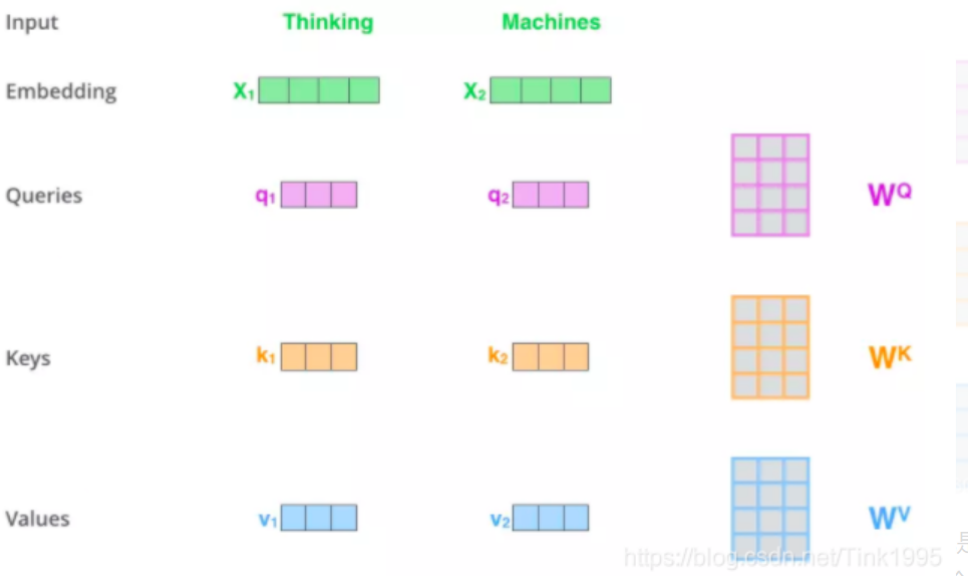
Transformer 的 Decoder的输入，对应到上面例子里面就是：Encoder接受英文"Tom chase Jerry"，Decoder接受中文"汤姆追逐杰瑞"。

2.2 Transformer的Encoder

图第2部分 Encoder block，由6个encoder堆叠而成，Nx=6。

灰框部分就是一个encoder的内部结构，从图中可以看出一个encoder由Multi-Head Attention 和 全连接神经网络Feed Forward Network构成。

Multi-Head Attention：



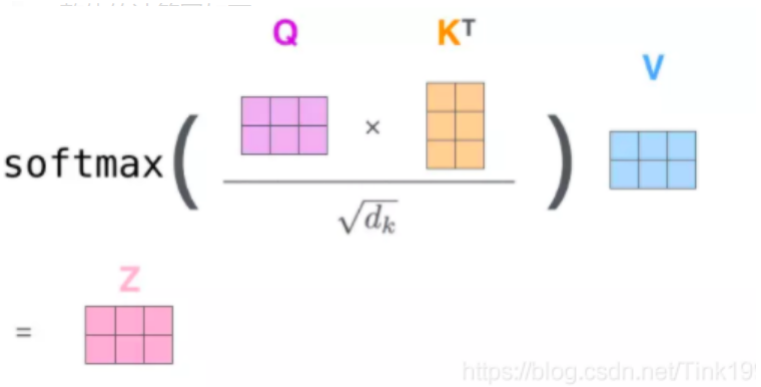
先回顾一下self-attention，假如输入序列是"Thinking Machines"，x1，x2就是对应地"Thinking"和"Machines"添加过位置编码之后的词向量，然后词向量通过三个权值矩阵W Q , W K , W V ，转变成为计算Attention值所需的Query，Keys，Values向量。

步骤1： 输入序列中每个单词之间的相关性得分，计算相关性得分可以使用点积法，就是用Q中每一个向量与K中每一个向量计算点积。

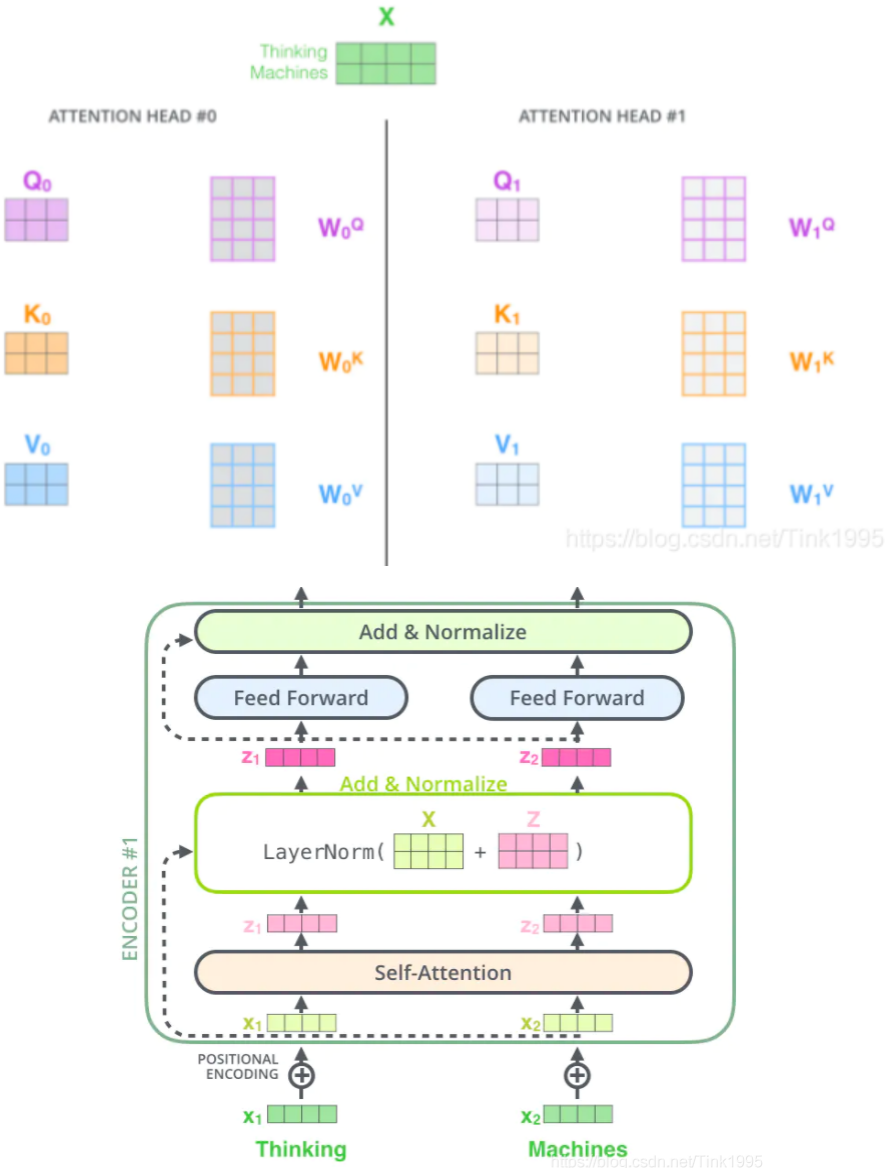
步骤2： 对于输入序列中每个单词之间的相关性得分进行归一化，归一化的目的主要是为了训练时梯度能够稳定。

步骤3： 通过softmax函数，将每个单词之间的得分向量转换成[0,1]之间的概率分布，

步骤4： 根据每个单词之间的概率分布，然后乘上对应的Values值，α与V进行点积，得到矩阵Z。



bedding矩阵，self-attention只使用了一组W Q , W K , W V ， 来进行变换得到Query，Keys，Values。而Multi-Head Attention使用多组W Q , W K , W V ， 得到多组Query，Keys，Values，然后每组分别计算得到一个Z矩阵，最后将得到的多个Z矩阵进行拼接。Transformer里面是使用了8组不同的W Q , W K , W V 。



在经过Multi-Head Attention得到矩阵Z之后，并没有直接传入全连接神经网络FNN，而是经过了一步：Add＆Normalize。

Add＆Normalize：

Add

Add，就是**在Z的基础上加了一个残差块X**，加入残差块X的目的是为了防止在深度神经网络训练中发生退化问题。**退化**的意思就是深度神经网络通过增加网络的层数，Loss逐渐减小，然后趋于稳定达到饱和，然后再继续增加网络层数，Loss反而增大。

ResNet 残差神经网络：

为什么深度神经网络会发生退化?

假如某个神经网络的最优网络层数是18层，但是我们在设计的时候并不知道到底多少层是最优解，本着层数越深越好的理念，我们设计了32层，那么32层神经网络中有14层其实是多余地，我们要想达到18层神经网络的最优效果，必须保证这多出来的14层网络必须进行**恒等映射** （恒等映射的意思就是说，输入什么，输出就是什么，可以理解成F(x)=x这样的函数，因为只有进行了这样的恒等映射咱们才能保证这多出来的14层神经网络不会影响我们最优的效果。）

但现实是神经网络的参数都是训练出来地，要想保证训练出来地参数能够很精确的完成F(x)=x的恒等映射其实是很困难地。多余的层数较少还好，对效果不会有很大影响，但多余的层数一多，可能结果就不是很理想了。这个时候就提出了**ResNet 残差神经网络来解决神经网络退化**的问题。

残差块是什么?

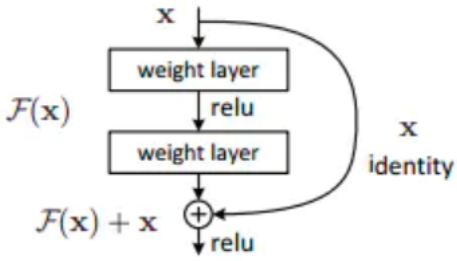


Figure 2. Residual learning: a building block.

为什么添加了残差块能防止神经网络退化问题?

添加了残差块后，要完成恒等映射的函数变成h(X)=F(X)+X，我们要让h(X)=X，相当于只需要让F(X)=0就可以了。

神经网络通过训练变成0是比变成X容易，一般初始化神经网络的参数的时候就是设置的[0,1]之间的随机数，所以经过网络变换后很容易接近于0。

Transformer中加上的X也就是Multi-Head Attention的输入，X矩阵。

Normalize

为什么要进行Normalize呢?

在神经网络进行训练之前，都需要对于输入数据进行Normalize归一化，目的有二：1，能够加快训练的速度。2.提高训练的稳定性。

Feed-Forward Networks

这里是一个两层的神经网络，先线性变换，然后ReLU非线性，再线性变换。

然后经过Add＆Normalize，输入下一个encoder中，经过6个encoder后输入到decoder中。

2.3 Transformer的Decoder

Decoder block也是由6个decoder堆叠而成，Nx=6。

一个decoder由Masked Multi-Head Attention，Multi-Head Attention 和 FNN构成。

Transformer Decoder的输入：

Decoder的输入分为两类：
一种是训练时的输入，一种是预测时的输入。
训练时的输入就是对应的target数据。例如翻译任务，Encoder输入"Tom chase Jerry"，Decoder输入"汤姆追逐杰瑞"。
预测时的输入，一开始输入的是起始符，然后每次输入是上一时刻Transformer的输出。例如，输入""，输出"汤姆"，输入"汤姆"，输出"汤姆追逐"，输入"汤姆追逐"，输出"汤姆追逐杰瑞"，输入"汤姆追逐杰瑞"，输出"汤姆追逐杰瑞"结束。
Masked Multi-Head Attention

与Encoder的Multi-Head Attention计算原理一样，只是多加了一个mask码。mask 表示掩码，它对某些值进行掩盖，使其在参数更新时不产生效果。

Transformer 模型里面涉及两种 mask，分别是 padding mask 和 sequence mask。

1.padding mask

因为每个批次输入序列长度是不一样的也就是说，要对输入序列进行对齐。具体来说，就是给在较短的序列后面填充 0。但是如果输入的序列太长，则是截取左边的内容，把多余的直接舍弃。

2.sequence mask

sequence mask 是为了使得 decoder 不能看见未来的信息。对于一个序列，在 time_step 为 t 的时刻，解码输出应该只能依赖于 t 时刻之前的输出，而不能依赖 t 之后的输出。需要想一个办法，把 t 之后的信息给隐藏起来。

这在训练的时候需要，因为训练的时候每次我们是将target数据完整输入进decoder中地，预测时不需要，预测的时候我们只能得到前一时刻预测出的输出。

Decoder中的第二个Multi-Head Attention就只是基于Attention，它的输入Quer来自于Masked Multi-Head Attention的输出，Keys和Values来自于Encoder中最后一层的输出。

2.4 Transformer的输出

首先经过一次线性变换，然后Softmax得到输出的概率分布，然后通过词典，输出概率最大的对应的单词作为我们的预测输出。