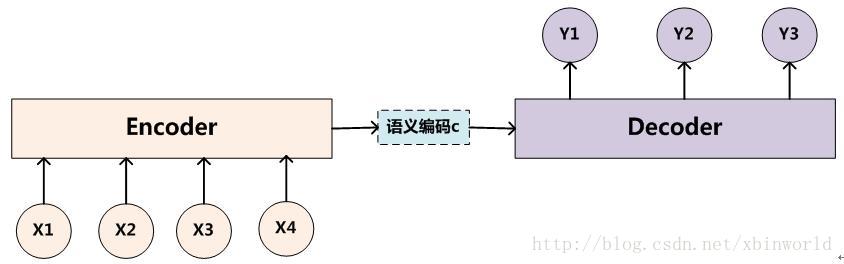
Attention and self-attention mechanism

## Encoder-decoder框架



在文本处理领域的Encoder-decoder框架，可以将其看做适合处理由一个句子（或篇章）生成另外一个句子（或篇章）的通用处理模型。针对句子对<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分别由各自的而Source和Target分别由各自的单词序列构成：

Source = {x1, x2 … xm}

Target = {y1, y2 … yn}

Encoder顾名思义就是对输入句子Source进行编码，将输入句子通过非线性变换转化为中间语义表示C：

C = F (x1, x2 … xm)

对于解码器Decoder来说，其任务是根据句子Source的中间语义表示C和之前已经生成的历史信息y1, y2, … yi-1来生成i时刻要生成的单词yi：

yi = G(C, y1 , y2 … yi-1)

每个yi都依次这么产生，那么看起来就是整个系统根据输入句子Source生成了目标句子Target。如果Source是中文句子，Target是英文句子，那么这就是解决机器翻译问题的Encoder-Decoder框架；如果Source是一篇文章，Target是概括性的几句描述语句，那么这是文本摘要的Encoder-Decoder框架；如果Source是一句问句，Target是一句回答，那么这是问答系统或者对话机器人的Encoder-Decoder框架。由此可见，在文本处理领域，Encoder-Decoder的应用领域相当广泛。

Encoder-Decoder框架不仅仅在文本领域广泛使用，在语音识别、图像处理等领域也经常使用。比如对于语音识别来说，图中所示的框架完全适用，区别无非是Encoder部分的输入是语音流，输出是对应的文本信息；而对于“图像描述”任务来说，Encoder部分的输入是一副图片，Decoder的输出是能够描述图片语义内容的一句描述语。一般而言，文本处理和语音识别的Encoder部分通常采用RNN模型，图像处理的Encoder一般采用CNN模型。

## Attention模型

Encoder-decoder框架是没有体现出“注意力模型”的，所以可以把它看做是注意力不集中的分心模型。因为目标句子target每个单词的生成过程如下：

y1 = f(c)

y2= f(c, y1)

y3 = f(c, y1, y2)

f是decoder的非线性变换函数。所以可以看出在生成目标句子的单词时，不论生成哪个单词，它们使用的输入句子Source的语义编码C都是一样的，没有任何区别。

而语义编码C是由句子Source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，其实句子Source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，这是为何说这个模型没有体现出注意力的缘由。这类似于人类看到眼前的画面，但是眼中却没有注意焦点一样。

如果拿机器翻译来解释这个分心模型的Encoder-Decoder框架更好理解，比如输入的是英文句子：Tom chase Jerry，Encoder-Decoder框架逐步生成中文单词：“汤姆”，“追逐”，“杰瑞”。

在翻译“杰瑞”这个中文单词的时候，分心模型里面的每个英文单词对于翻译目标单词“杰瑞”贡献是相同的，很明显这里不太合理，显然“Jerry”对于翻译成“杰瑞”更重要，但是分心模型是无法体现这一点的，这就是为何说它没有引入注意力的原因。

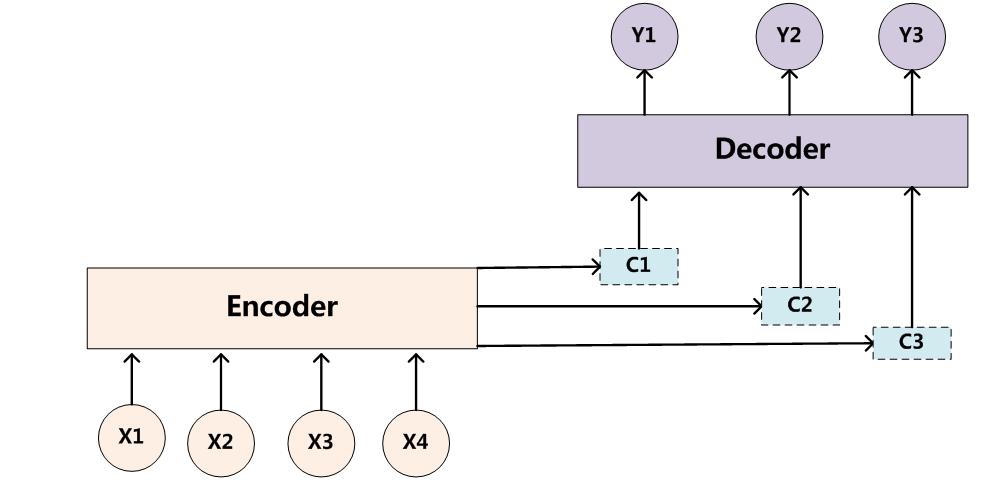
没有引入注意力的模型在输入句子比较短的时候问题不大，但是如果输入句子比较长，此时所有语义完全通过一个中间语义向量来表示，单词自身的信息已经消失，可想而知会丢失很多细节信息，这也是为何要引入注意力模型的重要原因。

上面的例子中，如果引入Attention模型的话，应该在翻译“杰瑞”的时候，体现出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：

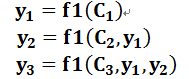
（Tom,0.3）(Chase,0.2) (Jerry,0.5)

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。这对于正确翻译目标语单词肯定是有帮助的，因为引入了新的信息。

同理，目标句子中的每个单词都应该学会其对应的源语句子中单词的注意力分配概率信息。这意味着在生成每个单词的时候，原先都是相同的中间语义表示C会被替换成根据当前生成单词而不断变化的。理解Attention模型的关键就是这里，即由固定的中间语义表示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。增加了注意力模型的Encoder-Decoder框架理解起来如图所示。



即生成目标句子单词的过程成了下面的形式：



而每个0?wx_fmt=png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：

上述内容就是经典的Soft Attention模型的基本思想，那么怎么理解Attention模型的物理含义呢？一般在自然语言处理应用里会把Attention模型看作是输出Target句子中某个单词和输入Source句子每个单词的对齐模型，这是非常有道理的。（对齐这个概念可以理解成为两个词的翻译对应关系，一般来说语料是句子级别对齐的，词对齐工具用来从句子对齐中学习到词对齐）

目标句子生成的每个单词对应输入句子单词的概率分布可以理解为输入句子单词和这个目标生成单词的对齐概率，这在机器翻译语境下是非常直观的：传统的统计机器翻译一般在做的过程中会专门有一个短语对齐的步骤，而注意力模型其实起的是相同的作用。

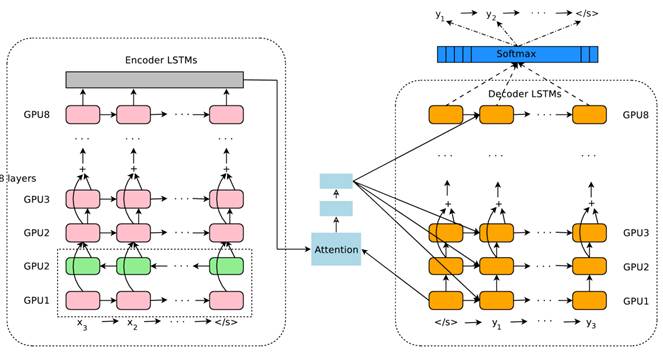


图8 Google 神经网络机器翻译系统结构图

图8所示即为Google于2016年部署到线上的基于神经网络的机器翻译系统，相对传统模型翻译效果有大幅提升，翻译错误率降低了60%，其架构就是上文所述的加上Attention机制的Encoder-Decoder框架，主要区别无非是其Encoder和Decoder使用了8层叠加的LSTM模型。

**Attention机制的本质思想**

如果把Attention机制从上文讲述例子中的Encoder-Decoder框架中剥离，并进一步做抽象，可以更容易看懂Attention机制的本质思想。

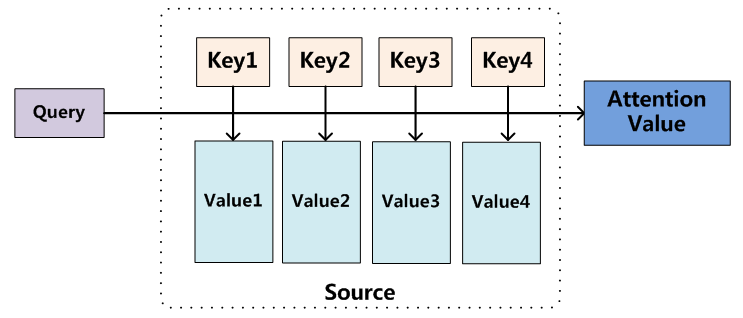


图9 Attention机制的本质思想

我们可以这样来看待Attention机制（参考图9）：将Source中的构成元素想象成是由一系列的<Key,Value>数据对构成，此时给定Target中的某个元素Query，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(一般情况下是求和)，即得到了最终的Attention数值。所以本质上Attention机制是对Source中元素的Value值进行加权求和，而Query和Key用来计算对应Value的权重系数。即可以将其本质思想改写为如下公式：

0?wx_fmt=png

其中，0?wx_fmt=png=||Source||代表Source的长度，公式含义即如上所述。上文所举的机器翻译的例子里，因为在计算Attention的过程中，Source中的Key和Value合二为一，指向的是同一个东西，也即输入句子中每个单词对应的语义编码，所以可能不容易看出这种能够体现本质思想的结构。

当然，从概念上理解，把Attention仍然理解为从大量信息中有选择地筛选出少量重要信息并聚焦到这些重要信息上，忽略大多不重要的信息，这种思路仍然成立。聚焦的过程体现在权重系数的计算上，权重越大越聚焦于其对应的Value值上，即权重代表了信息的重要性，而Value是其对应的信息。

从图9可以引出另外一种理解，也可以将Attention机制看作一种软寻址（Soft Addressing）:Source可以看作存储器内存储的内容，元素由地址Key和值Value组成，当前有个Key=Query的查询，目的是取出存储器中对应的Value值，即Attention数值。通过Query和存储器内元素Key的地址进行相似性比较来寻址，之所以说是软寻址，指的不像一般寻址只从存储内容里面找出一条内容，而是可能从每个Key地址都会取出内容，取出内容的重要性根据Query和Key的相似性来决定，之后对Value进行加权求和，这样就可以取出最终的Value值，也即Attention值。所以不少研究人员将Attention机制看作软寻址的一种特例，这也是非常有道理的。

至于Attention机制的具体计算过程，如果对目前大多数方法进行抽象的话，可以将其归纳为两个过程：第一个过程是根据Query和Key计算权重系数，第二个过程根据权重系数对Value进行加权求和。而第一个过程又可以细分为两个阶段：第一个阶段根据Query和Key计算两者的相似性或者相关性；第二个阶段对第一阶段的原始分值进行归一化处理；这样，可以将Attention的计算过程抽象为如图10展示的三个阶段。

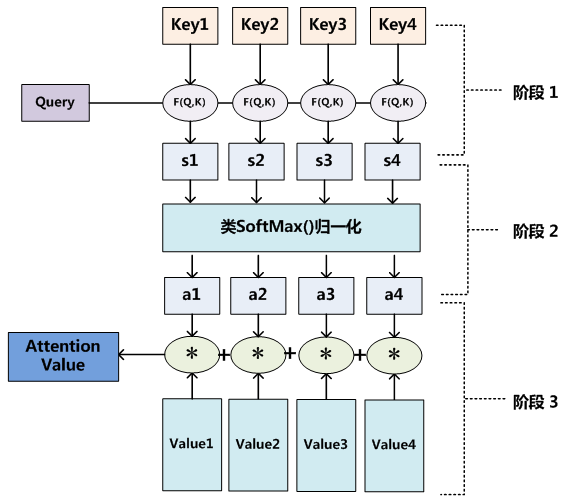
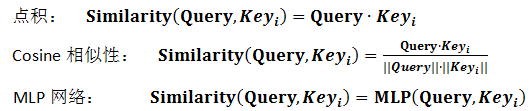


图10 三阶段计算Attention过程

在第一个阶段，可以引入不同的函数和计算机制，根据Query和某个0?wx_fmt=png，计算两者的相似性或者相关性，最常见的方法包括：求两者的向量点积、求两者的向量Cosine相似性或者通过再引入额外的神经网络来求值，即如下方式：



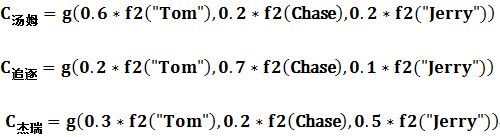
第一阶段产生的分值根据具体产生的方法不同其数值取值范围也不一样，第二阶段引入类似SoftMax的计算方式对第一阶段的得分进行数值转换，一方面可以进行归一化，将原始计算分值整理成所有元素权重之和为1的概率分布；另一方面也可以通过SoftMax的内在机制更加突出重要元素的权重。即一般采用如下公式计算：

0?wx_fmt=png

第二阶段的计算结果0?wx_fmt=png即为0?wx_fmt=png对应的权重系数，然后进行加权求和即可得到Attention数值：

0?wx_fmt=png

通过如上三个阶段的计算，即可求出针对Query的Attention数值，目前绝大多数具体的注意力机制计算方法都符合上述的三阶段抽象计算过程。



其中，f2函数代表Encoder对输入英文单词的某种变换函数，比如如果Encoder是用的RNN模型的话，这个f2函数的结果往往是某个时刻输入0?wx_fmt=png后隐层节点的状态值；g代表Encoder根据单词的中间表示合成整个句子中间语义表示的变换函数，一般的做法中，g函数就是对构成元素加权求和，即下列公式：

0?wx_fmt=png

其中，0?wx_fmt=png代表输入句子Source的长度，0?wx_fmt=png代表在Target输出第i个单词时Source输入句子中第j个单词的注意力分配系数，而0?wx_fmt=png则是Source输入句子中第j个单词的语义编码。假设0?wx_fmt=png下标i就是上面例子所说的“ 汤姆” ，那么0?wx_fmt=png就是3，h1=f(“Tom”)，h2=f(“Chase”),h3=f(“Jerry”)分别是输入句子每个单词的语义编码，对应的注意力模型权值则分别是0.6,0.2,0.2，所以g函数本质上就是个加权求和函数。如果形象表示的话，翻译中文单词“汤姆”的时候，数学公式对应的中间语义表示0?wx_fmt=png的形成过程类似图4。

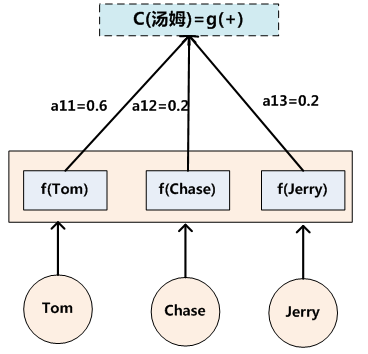


图4 Attention的形成过程

这里还有一个问题：生成目标句子某个单词，比如“汤姆”的时候，如何知道Attention模型所需要的输入句子单词注意力分配概率分布值呢？就是说“汤姆”对应的输入句子Source中各个单词的概率分布：(Tom,0.6)(Chase,0.2) (Jerry,0.2) 是如何得到的呢？

为了便于说明，我们假设对图2的非Attention模型的Encoder-Decoder框架进行细化，Encoder采用RNN模型，Decoder也采用RNN模型，这是比较常见的一种模型配置，则图2的框架转换为图5。

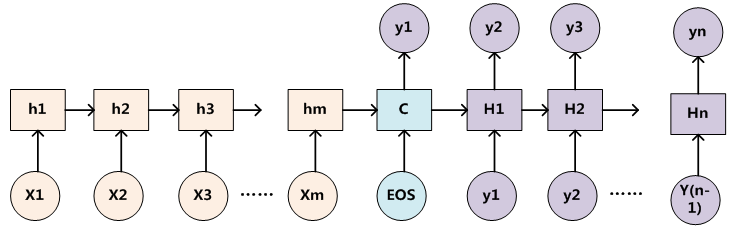


图5 RNN作为具体模型的Encoder-Decoder框架

那么用图6可以较为便捷地说明注意力分配概率分布值的通用计算过程。

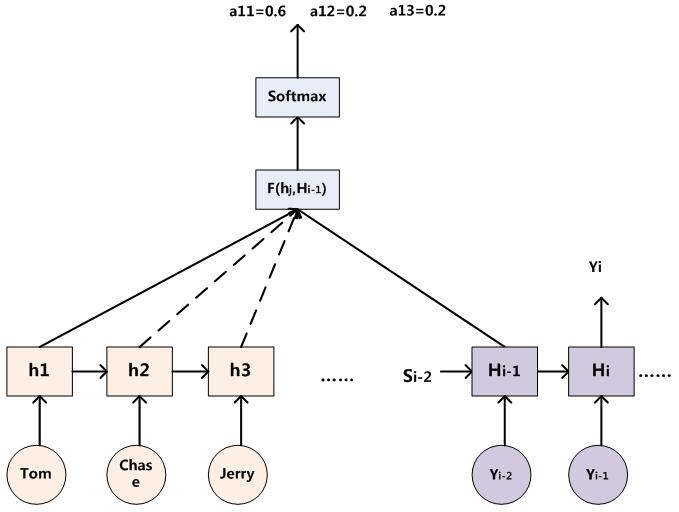


图6 注意力分配概率计算

对于采用RNN的Decoder来说，在时刻i，如果要生成yi单词，我们是可以知道Target在生成0?wx_fmt=png之前的时刻i-1时，隐层节点i-1时刻的输出值0?wx_fmt=png的，而我们的目的是要计算生成0?wx_fmt=png时输入句子中的单词“Tom”、“Chase”、“Jerry”对0?wx_fmt=png来说的注意力分配概率分布，那么可以用Target输出句子i-1时刻的隐层节点状态0?wx_fmt=png去一一和输入句子Source中每个单词对应的RNN隐层节点状态hj进行对比，即通过函数F(0?wx_fmt=png,0?wx_fmt=png)来获得目标单词0?wx_fmt=png和每个输入单词对应的对齐可能性，这个F函数在不同论文里可能会采取不同的方法，然后函数F的输出经过Softmax进行归一化就得到了符合概率分布取值区间的注意力分配概率分布数值。

**Self Attention模型介绍**

通过上述对Attention本质思想的梳理，我们可以更容易理解本节介绍的Self Attention模型。Self Attention也经常被称为intra Attention（内部Attention），最近一年也获得了比较广泛的使用，比如Google最新的机器翻译模型内部大量采用了Self Attention模型。

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Query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，所以此处不再赘述其计算过程细节。

如果是常规的Target不等于Source情形下的注意力计算，其物理含义正如上文所讲，比如对于机器翻译来说，本质上是目标语单词和源语单词之间的一种单词对齐机制。那么如果是Self Attention机制，一个很自然的问题是：通过Self Attention到底学到了哪些规律或者抽取出了哪些特征呢？或者说引入Self Attention有什么增益或者好处呢？我们仍然以机器翻译中的Self Attention来说明，图11和图12是可视化地表示Self Attention在同一个英语句子内单词间产生的联系。

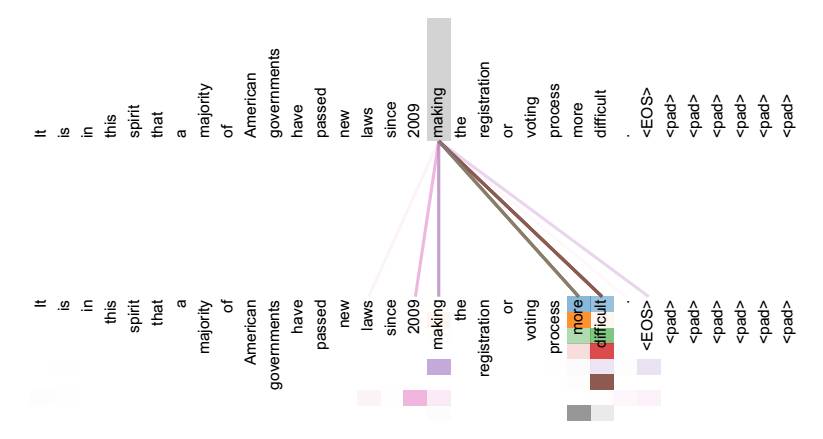


图11 可视化Self Attention实例

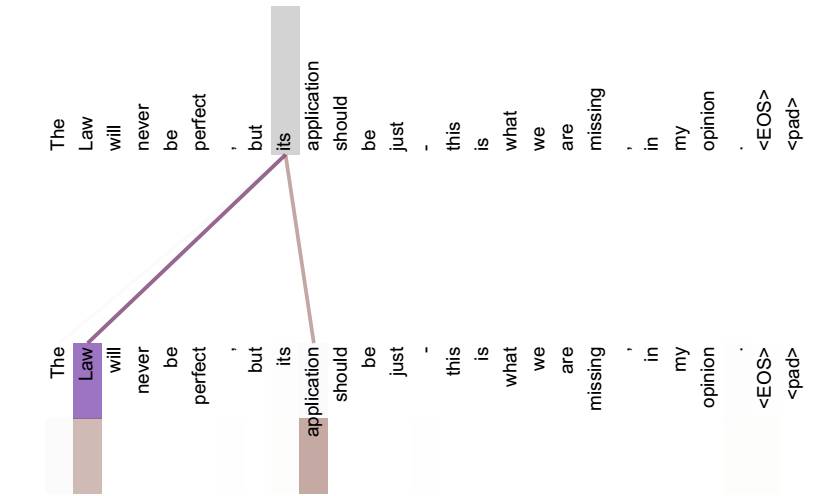


图12 可视化Self Attention实例

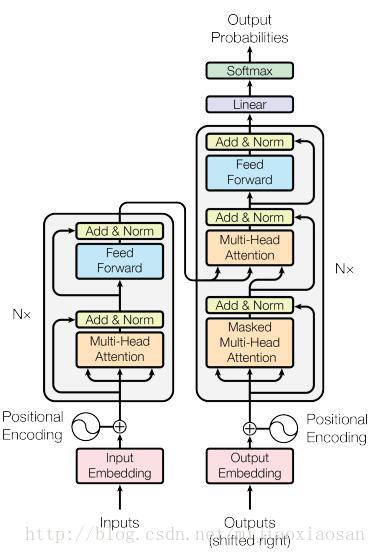
从两张图（图11、图12）可以看出，Self Attention可以捕获同一个句子中单词之间的一些句法特征（比如图11展示的有一定距离的短语结构）或者语义特征（比如图12展示的its的指代对象Law）。

很明显，引入Self Attention后会更容易捕获句子中长距离的相互依赖的特征，因为如果是RNN或者LSTM，需要依次序序列计算，对于远距离的相互依赖的特征，要经过若干时间步步骤的信息累积才能将两者联系起来，而距离越远，有效捕获的可能性越小。

但是Self Attention在计算过程中会直接将句子中任意两个单词的联系通过一个计算步骤直接联系起来，所以远距离依赖特征之间的距离被极大缩短，有利于有效地利用这些特征。除此外，Self Attention对于增加计算的并行性也有直接帮助作用。这是为何Self Attention逐渐被广泛使用的主要原因。

**Self Attention模型详解**

Attention Is All You Need提出了一个只基于attention的结构来处理序列模型相关的问题，比如机器翻译。传统的神经机器翻译大都是利用RNN或者CNN来作为encoder-decoder的模型基础，而谷歌最新的只基于Attention的Transformer模型摒弃了固有的定式，并没有用任何CNN或者RNN的结构。该模型可以高度并行地工作，所以在提升翻译性能的同时训练速度也特别快。

以下是谷歌Transformer的结构示意图。   


模型分为编码器和解码器两个部分，编码器由6个相同的层堆叠在一起，每一层又有两个支层。第一个支层是一个多头的自注意机制，第二个支层是一个简单的全连接前馈网络。在两个支层外面都添加了一个residual的连接，然后进行了layer nomalization的操作。模型所有的支层以及embedding层的输出维度都是dmodel=512dmodel=512. 模型的解码器也是堆叠了六个相同的层。不过每层除了编码器中那两个支层，解码器还加入了第三个支层，如图中所示同样也用了residual以及layer normalization。具体的细节后面再讲。

论文是从attention结构开始介绍Transformer的，这里为了方便，从模型的输入开始介绍。

模型输入。   
编码器和解码器的输入就是利用学习好的embeddings将tokens（一般应该是词或者字符）转化为dmodeldmodel 维向量。对解码器来说，利用线性变换以及softmax函数将解码的输出转化为一个预测下一个token的概率。

位置编码。   
由于模型没有任何循环或者卷积，为了使用序列的顺序信息，需要将tokens的相对以及绝对位置信息注入到模型中去。论文在输入embeddings的基础上加了一个“位置编码”。位置编码和embeddings由同样的维度都是dmodeldmodel 所以两者可以直接相加。有很多位置编码的选择，既有学习到的也有固定不变的。本文中用了正弦和余弦函数进行编码。

PE(pos,2i)=sin(pos/100002i/dmodel)PE(pos,2i)=sin(pos/100002i/dmodel)

PE(pos,2i+1)=cos(pos/100002i/dmodel)PE(pos,2i+1)=cos(pos/100002i/dmodel)

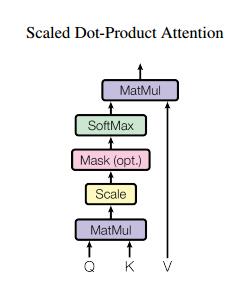
这里的pos是位置，i是维度。

下面介绍我对本文最重要的一部分注意力模型的理解。   
论文首先给attentioin进行了一个定义。attention函数可以看作将一个query和一系列key-value对映射为一个输出（output）的过程。这里的query，keys，values以及output都是向量。输出是由带权的values加起来得到的，而每个value的权重是根据query和相应的key通过一个函数计算出来的。

scaled 的点乘attention   
论文中用的attention是基本的点乘的方式，就是多了一个所谓的scale。输入包括维度为dkdk 的queries以及keys，还有维度为dvdv 的values。计算query和所有keys的点乘，然后每个都除以dk−−√dk（这个操作就是所谓的Scaled）。之后利用一个softmax函数来获取values的权重。   
实际操作中，attention函数是在一些列queries上同时进行的，将这些queries并在一起形成一个矩阵QQ 同时keys以及values也并在一起形成了矩阵KK 以及VV 。则attention的输出矩阵可以按照下述公式计算：

Attention(Q,K,V)=softmax(QKTdk−−√)VAttention(Q,K,V)=softmax(QKTdk)V

文中还讨论了点乘和additive attention的区别，additive的attention用了一个单层的前馈网络，理论上二者的复杂度是差不多的，但是实际上点乘更快而且空间上更有效率，因为点乘可以利用高度优化了的矩阵乘法代码来实现。   
当dkdk 比较小时，两个机制的效果几乎差不多，而在dkdk 比较大的时候，additive的attention效果比没有scale操作的点乘attention效果好。作者怀疑这是因为the dot products growing too large in magnitude to result in useful gradients after applying the softmax function（不知道怎么翻译好），所以作者用了scale这个操作来规模化点乘。   
下图为作者scale 点乘attention的结构图。

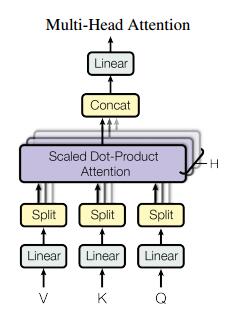


多头的attention （Multi-Head Attention）   
这个应该是本文最核心的部分。本文结构中的Attention并不是简简单单将一个点乘的attention应用进去。作者发现先对queries，keys以及values进行hh 次不同的线性映射效果特别好。学习到的线性映射分别映射到dkdk , dkdk 以及dvdv维。分别对每一个映射之后的得到的queries，keys以及values进行attention函数的并行操作，生成dvdv 维的output值。具体操作细节如以下公式。

MultiHead(Q,K,V)=Concat(head1,...,headh)MultiHead(Q,K,V)=Concat(head1,...,headh)

where:headi=Attention(QWiQ,KWiK,VWiV)where:headi=Attention(QWiQ,KWiK,VWiV)

这里映射的参数矩阵，WiQ∈Rdmodel∗dkWiQ∈Rdmodel∗dk ,WiK∈Rdmodel∗dkWiK∈Rdmodel∗dk ,WiV∈Rdmodel∗dvWiV∈Rdmodel∗dv。   
本文中对并行的attention层（或者成为头）使用了h=8h=8 的设定。其中每层都设置为dk=dv=dmodel/h=64dk=dv=dmodel/h=64. 由于每个头都减少了维度，所以总的计算代价和在所有维度下单头的attention是差不多的。

把这些attention头的output拼接起来作为最终值，就像下图中multihead attention结构示意图所描述的一样。   


在本模型中如何使用attention   
本文提出的模型Transformer以三种不同的方式使用了多头attention。   
1. 在encoder-decoder的attention层，queries来自于之前的decoder层，而keys和values都来自于encoder的输出。这个类似于很多已经提出的seq-to-seq模型所使用的attention机制。   
2. 在encoder含有self-attention层。在一个self-attention层中，所有的keys, values以及queries都来自于同一个地方，本例中即encoder之前一层的的输出。   
3. 类似的，decoder中的self-attention层也是一样。不同的是在scaled点乘attention操作中加了一个mask的操作（设置为负无穷），这个操作是保证softmax操作之后不会将非法的values连到attention中（个人理解，比如你这一位置queryattention的values不能有这一位置之后的values的信息，只能有该位置前面的values，本人菜鸟欢迎拍砖）。

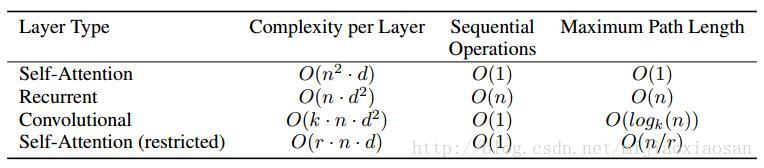
之前说模型由堆叠在一起的六个层组成，每层由两个支层，attention层就是其中一个，而attention之后的另一个支层就是一个前馈的网络。公式描述如下。

FFN(x)=max(0,xW1+b1)W2+b2FFN(x)=max(0,xW1+b1)W2+b2

该网络是两个线性变换，中间加了一个ReLU激活函数。每个位置（position）上的线性变换是一样的，但是不同层与层的参数是不一样的。该网络的输入和输出维度都是dmodeldmodel ，不过中间层的维度是2048.

模型的整体框架基本介绍完了，其最重要的创新应该就是Self-Attention的使用级联的多头attention架构。

作者在文中深入讨论了为什么选择使用self attention这一结构。主要从三个方面来谈。第一是每层的总计算复杂性，其次是能并行计算的数量，这点论文用所要求的序列操作的最小值来量化。第三点是针对网络长距离依赖的路径长度。为了提升在长序列上的计算性能，self-attention应该改被限制（restrict）在一个size大小为rr 的邻域内，这样可以将最大路径长度增加到O(n/r)O(n/r) ，至于具体如何去做作者说在未来工作中会研究。



上图为self attention结构与cnn以及rnn的性能比较，可以看出self attention还是具有优势的。

论文中的东西就说这么多了，下面说我的感想。   
自我感觉谷歌这次放大招了，完全抛弃了CNN以及RNN这几年做神经机器翻译的固有思维，仅用了attention来对句子进行编码和解码。从论文后面的附图可以看出该模型已经在很大程度上“理解”了句子的意思。也许过一段时间，各种层叠attention及其变种的模型就会满天飞了，但谷歌无疑开了这样一个好头，就是我们不必对固有的模型抱有很深的执念，尝试各种可能也许会有意想不到的效果。不过话说回来，谁知道这个模型是失败了多少次之后才有的成功呢？