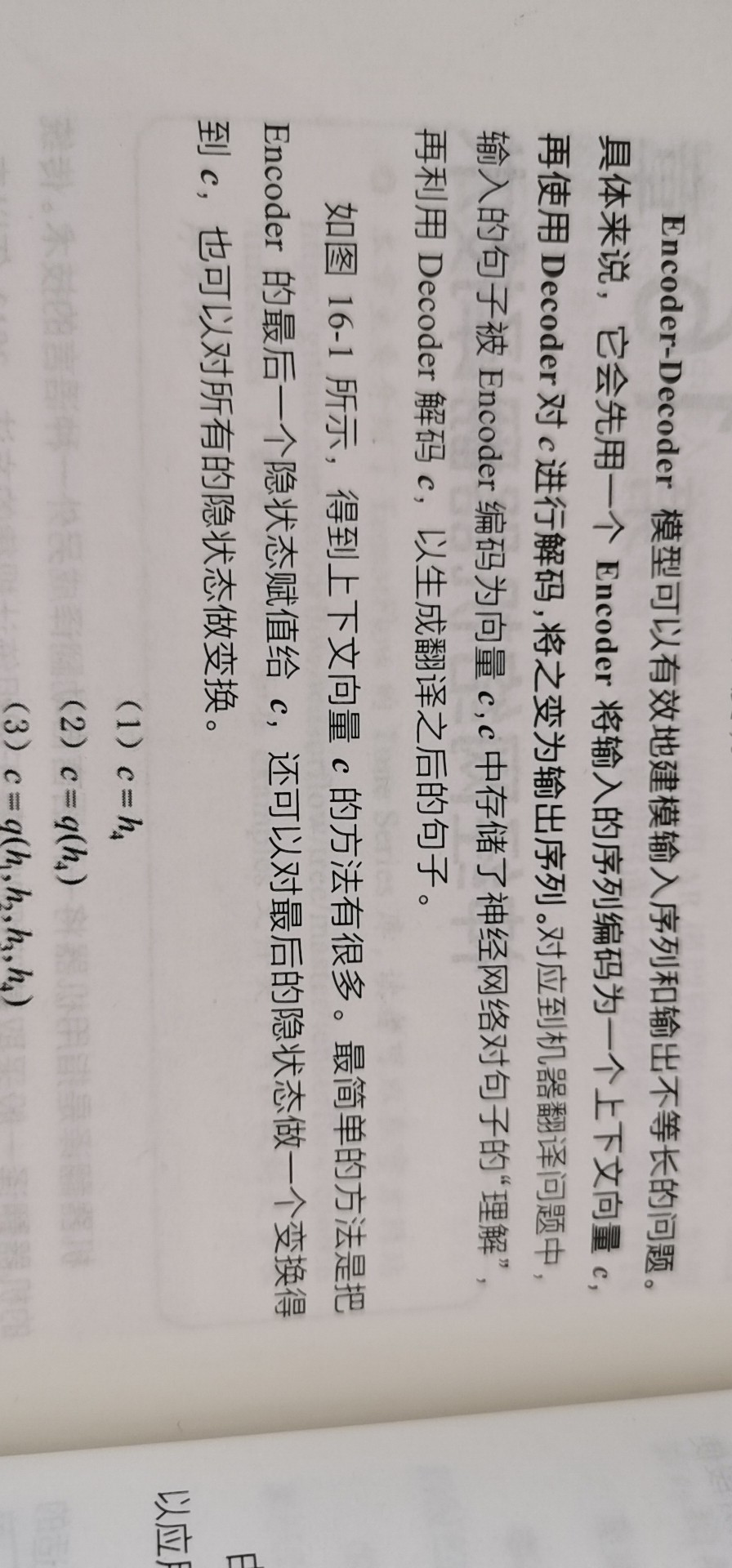
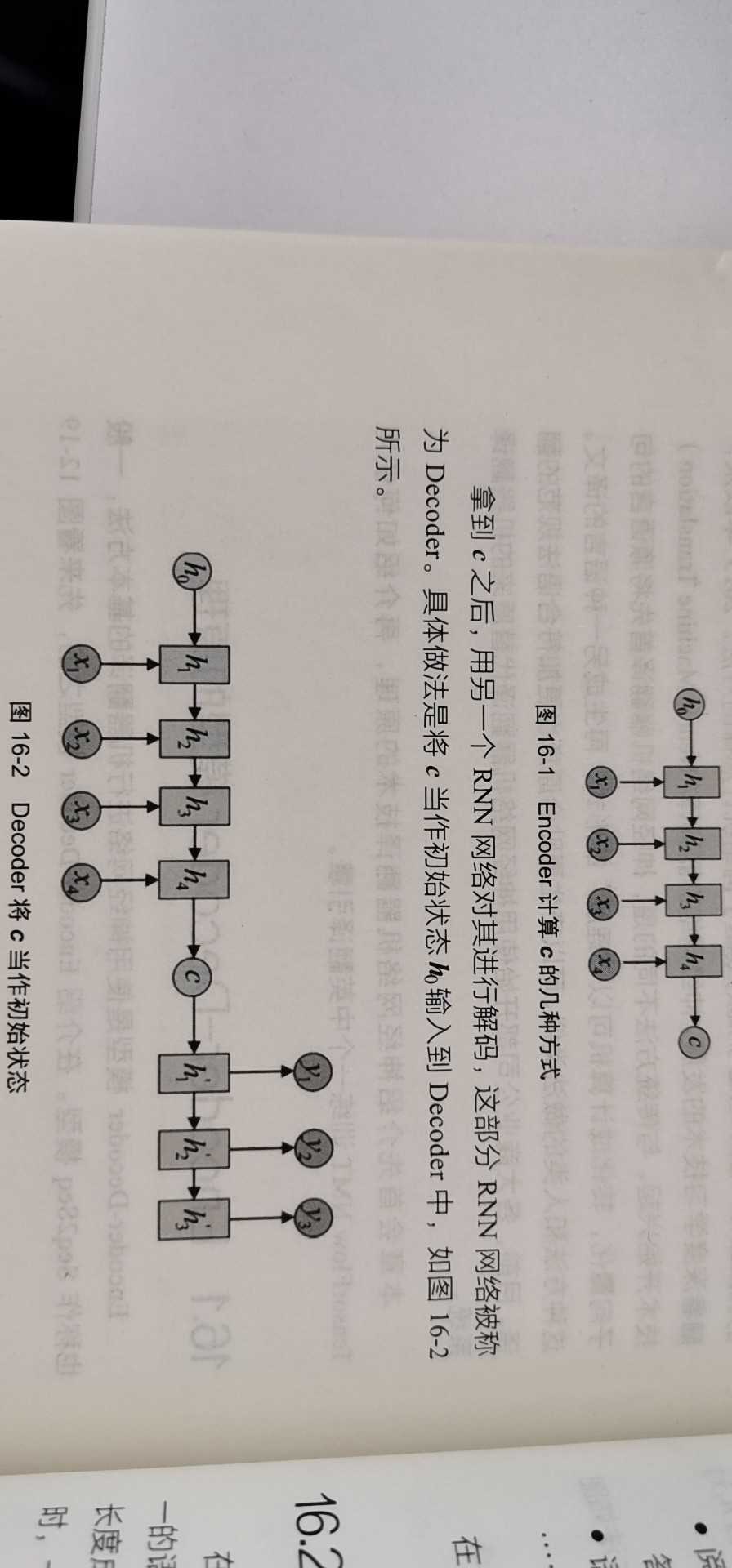
# Encoder-Decoder框架

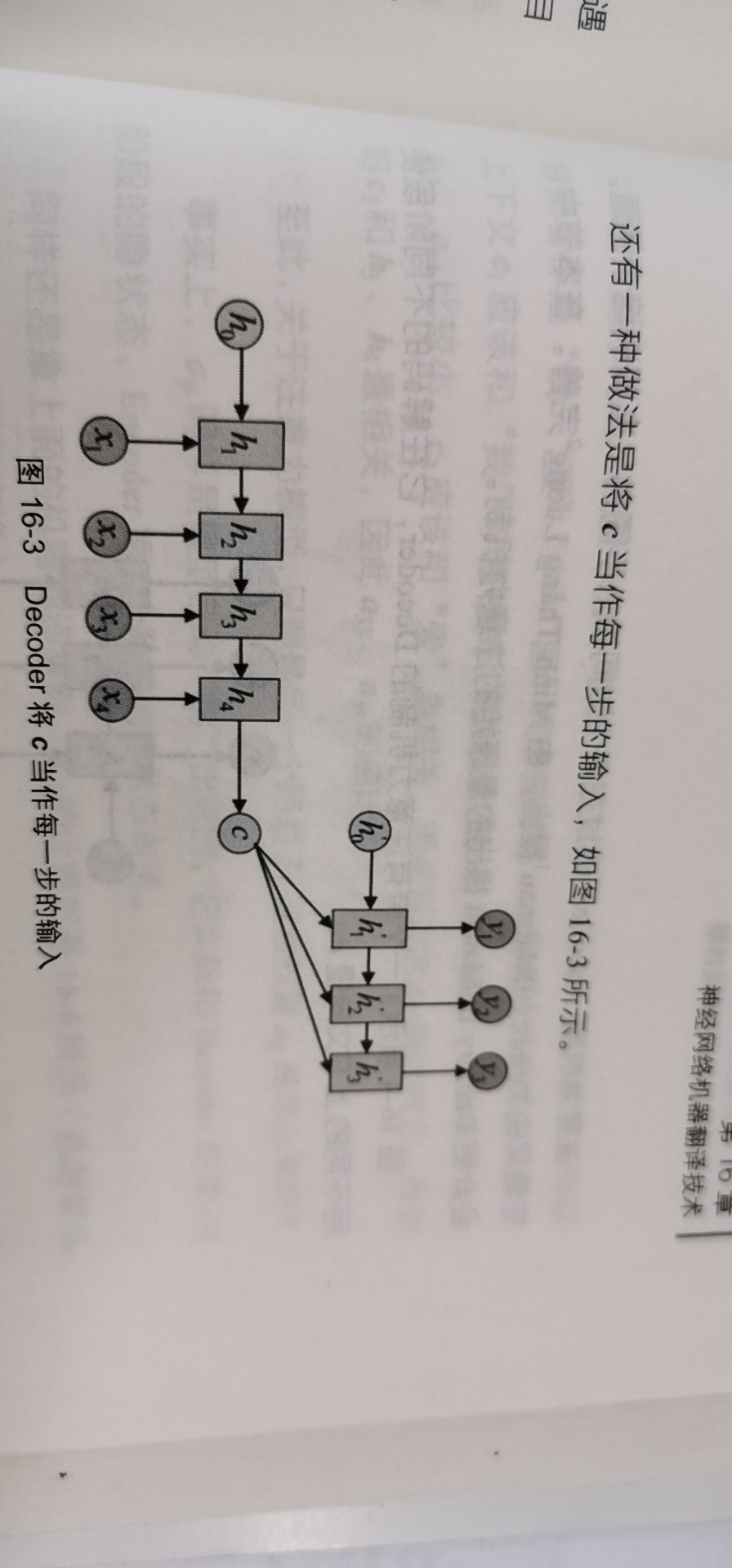
目前大多数注意力模型附着在Encoder-Decoder框架下，当然，其实注意力模型可以看作一种通用的思想，本身并不依赖于特定框架。

Encoder-Decoder框架可以看作是一种深度学习领域的研究模式，应用场景异常广泛。下图是文本处理领域里常用的Encoder-Decoder框架最抽象的一种表示。









文本处理领域的Encoder-Decoder框架可以这么直观地去理解：可以把它看作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。对于句子对<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：

https://pic1.zhimg.com/80/v2-4ab3a2d834a45581c32ea62aec28e428_hd.jpg

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：

https://pic3.zhimg.com/80/v2-749be2e4a08d8e8b3874d23b0f6c3f22_hd.jpg

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

https://pic3.zhimg.com/80/v2-18682b9de059b0ea2d1a25a45bea708a_hd.jpg

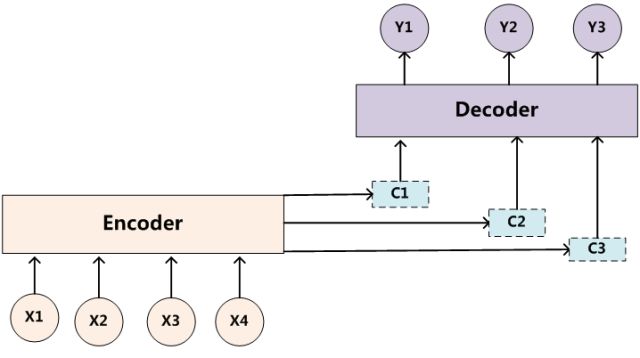
来生成i时刻要生成的单词https://pic2.zhimg.com/80/v2-3b9fde6372a3124bf0dfbedf5b9e354d_hd.jpg：

https://pic2.zhimg.com/80/v2-0036edc4b2888e0df612585dd5f28ecd_hd.jpg

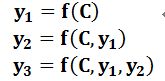
每个yi都依次这么产生，那么看起来就是整个系统根据输入句子Source生成了目标句子Target。如果Source是中文句子，Target是英文句子，那么这就是解决机器翻译问题的Encoder-Decoder框架；如果Source是一篇文章，Target是概括性的几句描述语句，那么这是文本摘要的Encoder-Decoder框架；如果Source是一句问句，Target是一句回答，那么这是问答系统或者对话机器人的Encoder-Decoder框架。由此可见，在文本处理领域，Encoder-Decoder的应用领域相当广泛。

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2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



传统的Encoder-Decoder框架是没有体现出“注意力模型”的，所以可以把它看作是注意力不集中的分心模型。为什么说它注意力不集中呢？请观察下目标句子Target中每个单词的生成过程如下：



其中f是Decoder的非线性变换函数。从这里可以看出，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。

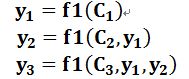
上面的例子中，如果引入Attention模型的话，应该在翻译“杰瑞”的时候，体现出英文单词对于翻译当前中文单词不同的影响程度，比如给出类似下面一个概率分布值：

（Tom,0.3）(Chase,0.2) (Jerry,0.5)

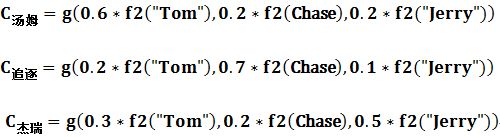
每个英文单词的概率代表了翻译当前单词“杰瑞”时，注意力分配模型分配给不同英文单词的注意力大小。这对于正确翻译目标语单词肯定是有帮助的，因为引入了新的信息。

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yi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Ci。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。

即生成目标句子单词的过程成了下面的形式：



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：



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xi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：

https://pic3.zhimg.com/80/v2-a204b9a5817be9a7fbc0a1abfa6b9ab2_hd.jpg

其中，Lx代表输入句子Source的长度，aij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hj则是Source输入句子中第j个单词的语义编码。假设下标i就是上面例子所说的“汤姆” ，那么Lx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Ci的形成过程类似图4。

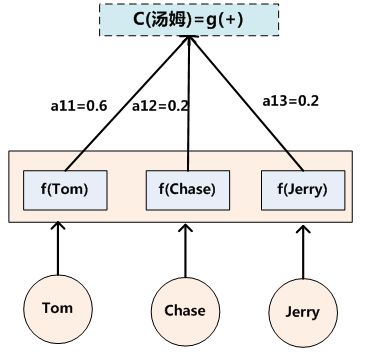


图4 Attention的形成过程

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。

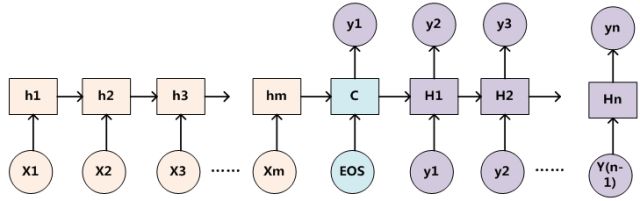


图5 RNN作为具体模型的Encoder-Decoder框架

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。

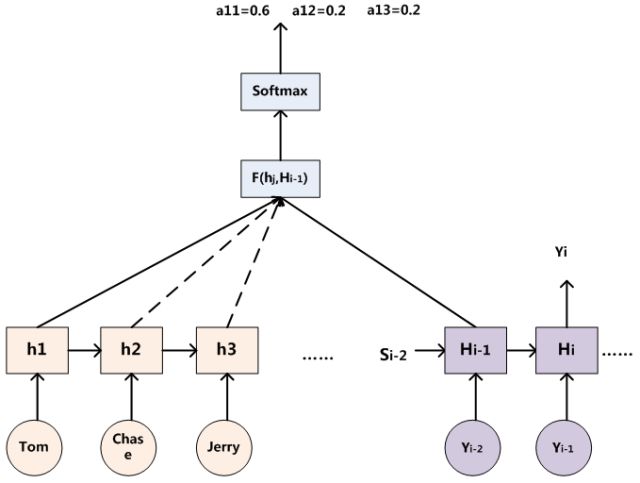


图6 注意力分配概率计算

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Yi之前的时刻i-1时，隐层节点i-1时刻的输出值Hi-1的，而我们的目的是要计算生成Yi时输入句子中的单词“Tom”、“Chase”、“Jerry”对Yi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Hi-1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(hj,Hi-1)来获得目标单词yi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。

绝大多数Attention模型都是采取上述的计算框架来计算注意力分配概率分布信息，区别只是在F的定义上可能有所不同。

# Attention模型的物理含义

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。

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。

# 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。

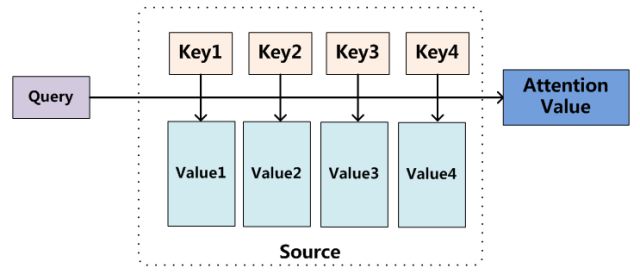


图9 Attention机制的本质思想

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，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：

https://pic2.zhimg.com/80/v2-76cac5c196e43afc8338712b6a41d491_hd.jpg

其中，Lx=||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。

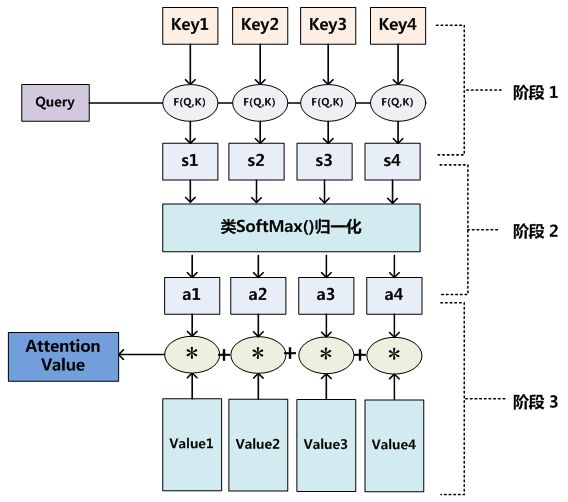
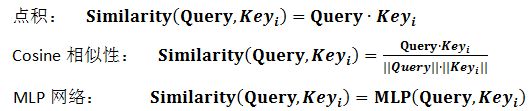


图10 三阶段计算Attention过程

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Key\_i，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：



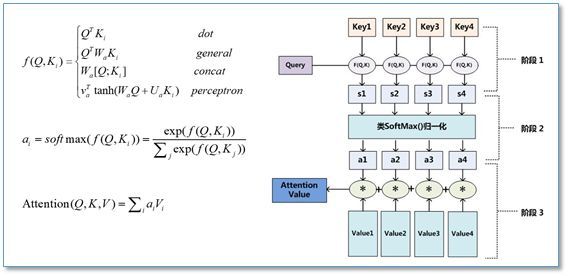
第一阶段产生的分值根据具体产生的方法不同其数值取值范围也不一样，第二阶段引入类似SoftMax的计算方式对第一阶段的得分进行数值转换，一方面可以进行归一化，将原始计算分值整理成所有元素权重之和为1的概率分布；另一方面也可以通过SoftMax的内在机制更加突出重要元素的权重。即一般采用如下公式计算：

https://pic2.zhimg.com/80/v2-6b16e6271e83803448324e70ef5ac001_hd.jpg

第二阶段的计算结果a\_i即为value\_i对应的权重系数，然后进行加权求和即可得到Attention数值：

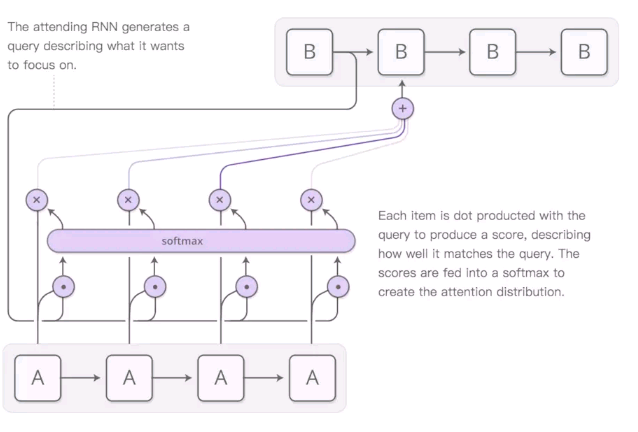
https://pic3.zhimg.com/80/v2-3dcf9d0ea703db3045d9a60d2e7eadb2_hd.jpg

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。



# Attention内部计算

先上图，再来解释：



上图中的A是我们的encoder， B是我们的decoder。  
可以想象，A网络接收了一个四个字的句子，对每个字都产生了一个输出（这些输出都是一个vector），我们称其为s1，s2，s3，s4。

我们看上图的B网络，在第一个B产生的hidden state（称其为h1）除了传给下一个cell外，还传到了A网络，这里就是Attention发挥作用的地方，我们来看看发生了什么。

**第一步**：  
h1 分别与s1，s2，s3，s4做点积，产生了四个数，称其为m1，m2，m3，m4（这些都是标量，不是向量了！）

**第二步**：  
m1，m2，m3，m4 传到一个softmax层，产生一个概率分布a1，a2，a3， a4。

**第三步**：  
将a1，a2，a3， a4 与s1，s2，s3，s4分别相乘，再相加，得到得到一个vector，称其为Attention vector。

**第四步**：

Attention vector 将作为输入传到B网络的第二个cell中，参与预测。

以上就是Attention机制的基本思想了。我们看到，Attention vector 实际上融合了s1，s2，s3，s4的信息，具体的融合是用一个概率分布来达到的，而这个概率分布又是通过B网络上一个cell的hidden state与s1，s2，s3，s4进行点乘得到的。  
Attention vector实际上达到了让B网络聚焦于A网络输出的某一部分的作用。

# Attention的通用定义

（1）给定一组向量集合values，以及一个向量query，attention机制是一种根据该query计算values的加权求和的机制。

（2）attention的重点就是这个集合values中的每个value的“权值”的计算方法。

有时候也把这种attention的机制叫做query的输出关注了（或者说叫考虑到了）原文的不同部分。（Query attends to the values）

# Attention的计算变体

首先从大的概念来讲，针对attention的变体主要有两种方式：

（1）一种是在attention向量的加权求和计算方式上进行创新

（2）另一种是在attention score（匹配度或者叫权值）的计算方式上进行创新

（3）当然还有一种就是把二者都有改变的结合性创新，或者是迁移性创新，比如借鉴CNN的Inception思想等等。

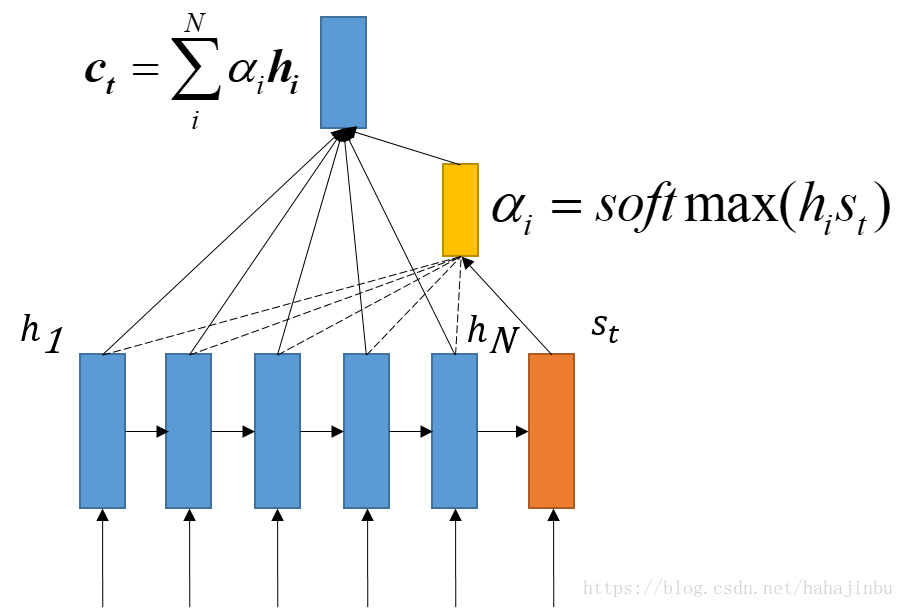
## Attention向量计算方式变体

大概分成这么几种：

* Soft attention、global attention、动态attention
* Hard attention
* local attention ：“半软半硬”的attention
* 静态attention
* 强制前向attention

### Soft Attention模型，global attention，动态attention

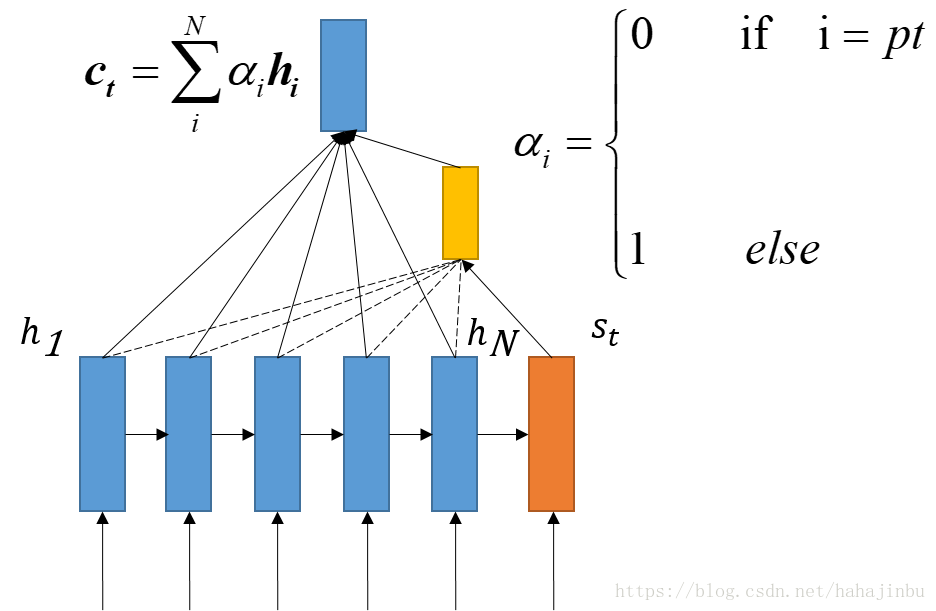
这三个其实就是Soft attention，也就是我们上面讲过的那种最常见的attention，是在求注意力分配概率分布的时候，对于输入句子X中任意一个单词都给出个概率，是个概率分布，把attention变量（context vecor）用ct表示，attention得分在经过了softmax过后的权值用alpha alpha表示。



论文：Neural machine translation by jointly learning to align and translate

### Hard attention

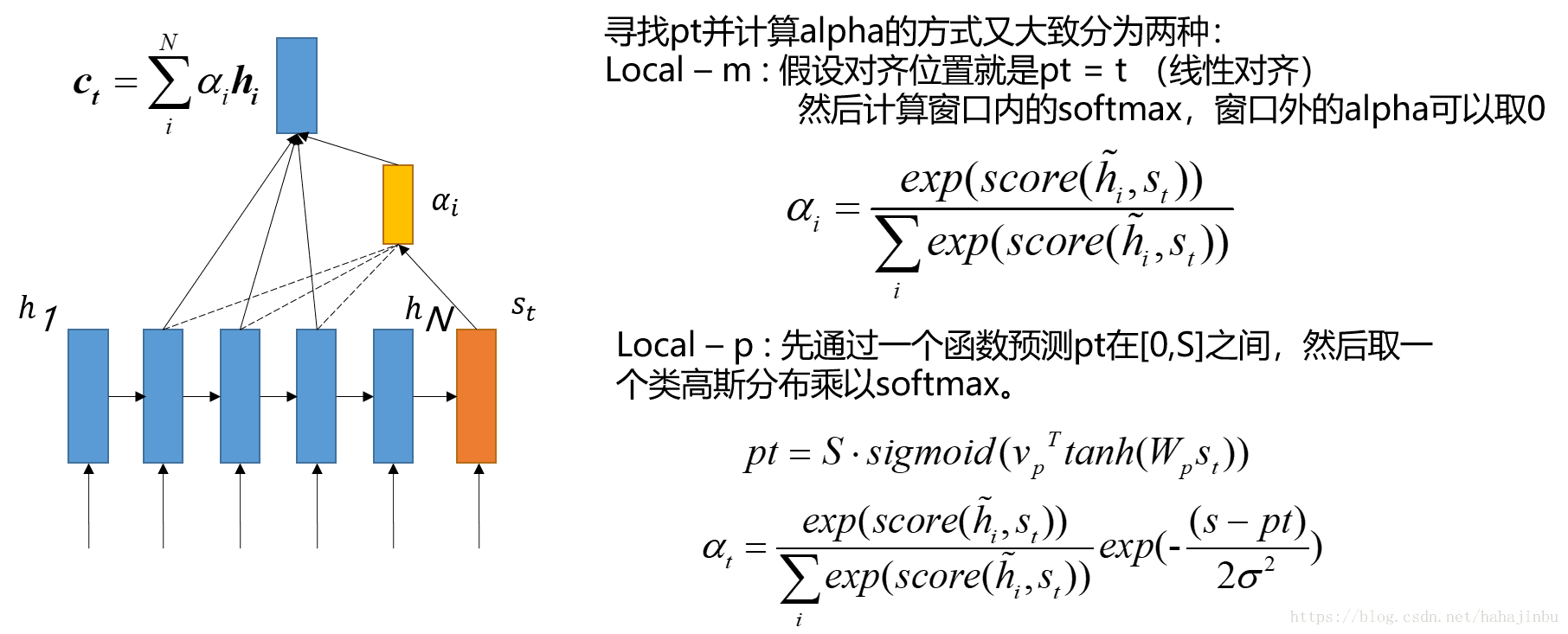
Soft是给每个单词都赋予一个单词match概率，那么如果不这样做，直接从输入句子里面找到某个特定的单词，然后把目标句子单词和这个单词对齐，而其它输入句子中的单词硬性地认为对齐概率为0，这就是Hard Attention Model的思想。



Hard attention 一般用在图像里面，当图像区域被选中时，权重为1，剩下时候为0。  
<https://www.cnblogs.com/Determined22/p/6914926.html>

### local attention （半软半硬attention）

Soft attention 每次对齐的时候都要考虑前面的encoder的所有hi，所以计算量会很大，因此一种朴素的思想是只考虑部分窗口内的encoder隐藏输出，其余部分为0，在窗口内使用softmax的方式转换为概率。这个local attention相反概念的是global attention，global attention其实就是softmax attention，这里不多赘述global attention了。



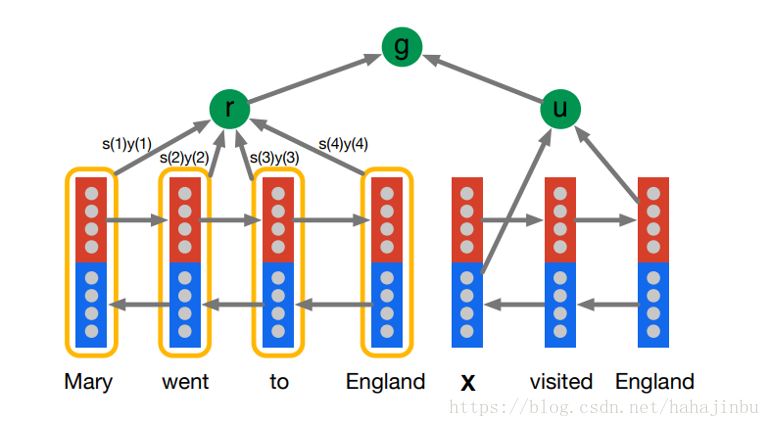
论文：Effective Approaches to Attention-based Neural Machine Translation

在这个模型中，对于是时刻t的每一个目标词汇，模型首先产生一个对齐的位置 pt（aligned position），context vector 由编码器中一个集合的隐藏层状态计算得到，编码器中的隐藏层包含在窗口[pt-D,pt+D]中，D的大小通过经验选择。

上式之中，大S指的是源句子的长度，Wp和vp是指的模型的参数，通过训练得到，为了支持pt附近的对齐点，设置一个围绕pt的高斯分布，其中小s是在以pt为中心的窗口中的整数，pt是一个在[0，S]之间的实数。小Sigma σ 一般取窗口大小的一半。

### 静态attention

静态attention：对输出句子共用一个St的attention就够了，一般用在Bilstm的首位hidden state输出拼接起来作为st（在图所示中为u）



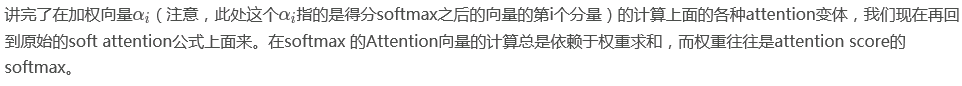
论文：Teaching Machines to Read and Comprehend 以及

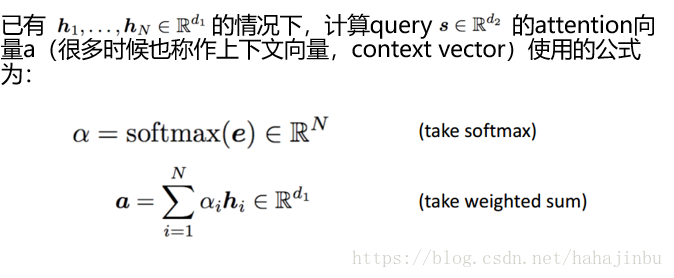
Supervised Sequence Labelling with Recurrent Neural Networks

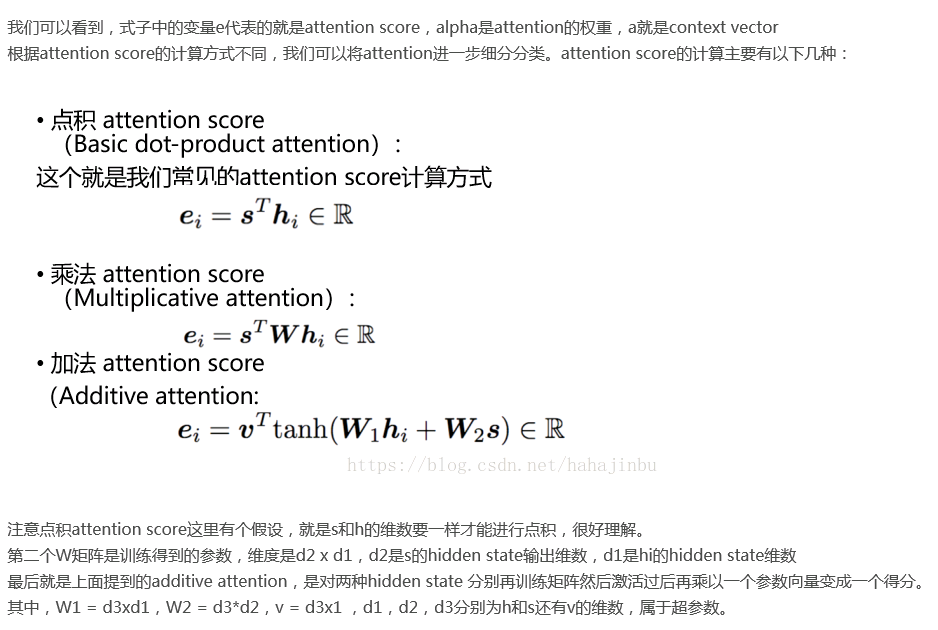
这个前面的每个hidden state 不应该都和这里的 u 算一次attention score吗，怎么这里只有一个r和u进行了交互？

其实他这里的r表示的是加权平均的self attention，这个权就是attention ct向量，这个图里面把attention ct的计算过程省略了。直接跳到了ct和st计算真正的s’t的部分。他这个里面用的实际的attention score的计算并不是用点积，是additive attention，什么是additive attention呢？这个下面就会讲根据按照attention score计算的不同的attention model变体。

## Attention score的计算方式变体







# Self Attention模型

## Self Attention简介

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。

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SelfAttention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。

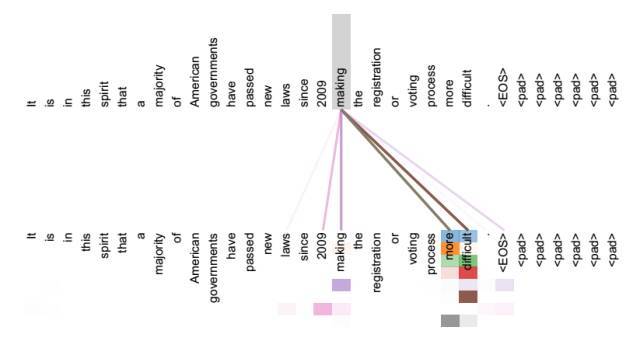


图11 可视化Self Attention实例

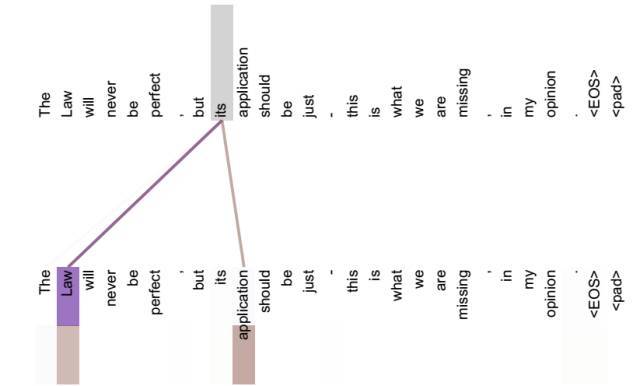


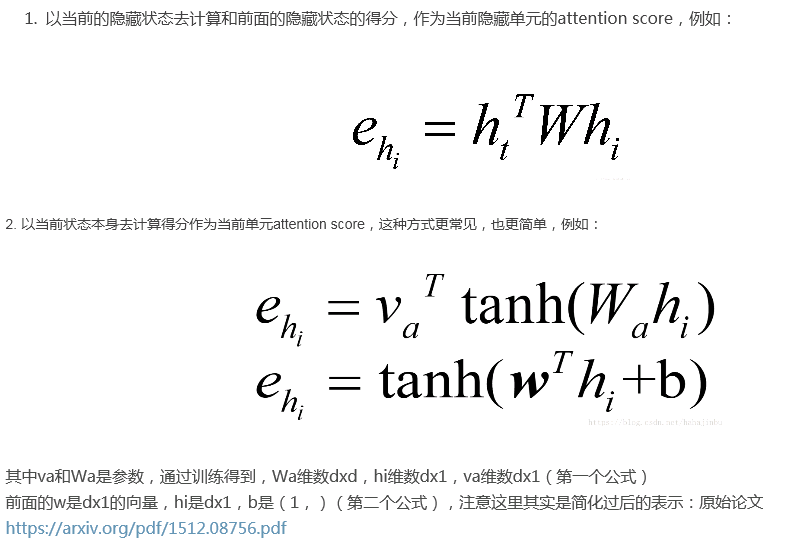
图12 可视化Self Attention实例

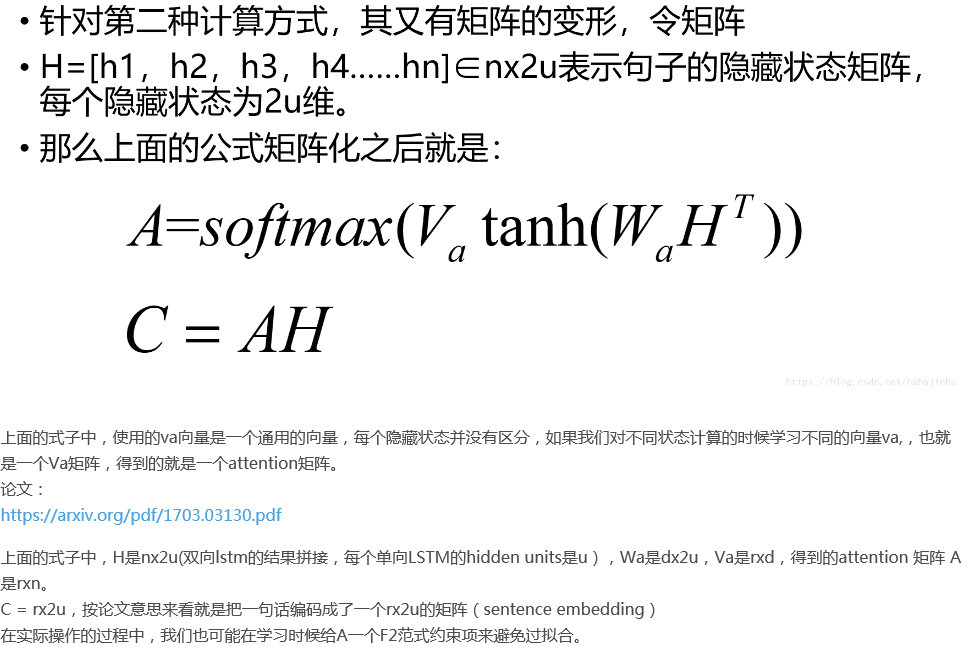
从两张图（图11、图12）可以看出，Self Attention可以捕获同一个句子中单词之间的一些句法特征（比如图11展示的有一定距离的短语结构）或者语义特征（比如图12展示的its的指代对象Law）。

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SelfAttention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，SelfAttention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计算方式

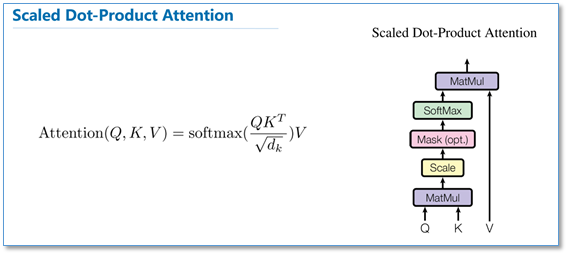




# Scaled Dot-Product attention

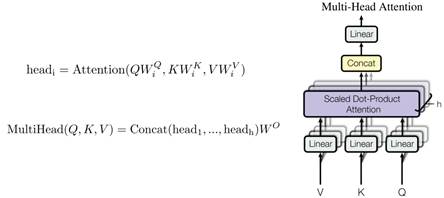
Scaled Dot-Product attention：缩放点积attention，在使用点积运算进行相似度计算的基础上，缩小了http://ir.dlut.edu.cn/Uploads/ue/image/20180415/6365939389766816457808992.jpg倍（dk为词向量的维度）。其目的在于调节的作用，使得内积不易过大。

http://ir.dlut.edu.cn/Uploads/ue/image/20180415/6365939389776176468780808.jpg

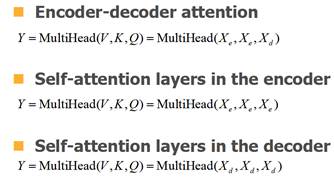


# Multi-Head Attention

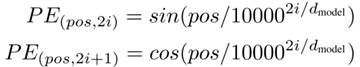
 Google 在 attention is all you need 中发明了一种叫transformer的网络结构，其中用到了multi-head attention。多头attention（Multi-head attention）的结构贺公式如图所示。首先，需要对query、key和value进行一个线性变换；然后输入到缩放点积attention机制，重复做h次，每次的输入为线性变换后的原始输入，这里，多头就是指做多次attention之后进行拼接，每一次算一个头，每次Q、K和V的线性变换参数W是不一样的；最后，将拼接后的模型做一次线性变换，得到的值为多头attention的结果。可以看出，多头attention与传统的attention区别在于计算了h次，这样可以从不同的维度和表示子空间里学习到相关的信息，可通过attention可视化机制来验证。



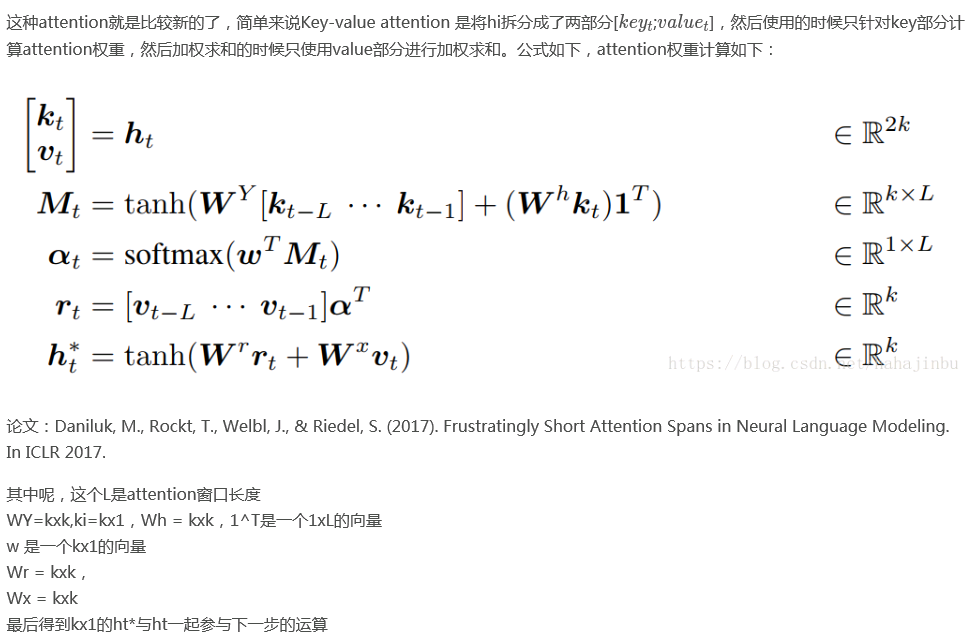
在整个模型中，从编码器到解码器的地方中使用了多头attention进行连接，K、V和Q分别是编码器的层输出（这里K=V）和解码器中多头attention的输入，这其实跟主流的机器翻译模型中的attention一样，进行传统的翻译对齐任务。然后，在编码器和解码器中都使用了多头自注意力self-attention来学习文本的表示，K=V=Q，即里面的每个词都要和该句子中的所有词进行attention计算，其主要目的是学习句子内部的词依赖关系，捕获句子中的内部结构。



这里，要着重说一下位置编码。因为该模型没有使用RNN等序列模型，不能考虑到时序信息，因此，这里拟合了一个位置编码函数，来模拟词语的顺序。实验结果表明，这种方式是合理有效的。



# key-value attention



# 总结

总的来说，attention的机制就是一个加权求和的机制，只要我们使用了加权求和，不管你是怎么花式加权，花式求和，只要你是根据了已有信息计算的隐藏状态的加权和求和，那么就是使用了attention，而所谓的self attention就是仅仅在句子内部做加权求和（区别与seq2seq里面的decoder对encoder的隐藏状态做的加权求和）。

self attention作用范围更大一点，而key-value其实是对attention进行了一个更广泛的定义罢了，我们前面的attention都可以套上key-value attention，比如很多时候我们是把k和v都当成一样的算来，做self的时候还可能是quey=key=value。

# 参考资料

[Attention is all you need](https://link.jianshu.com/?t=https://arxiv.org/abs/1706.03762)

<https://blog.csdn.net/hahajinbu/article/details/81940355>

<https://zhuanlan.zhihu.com/p/37601161>

<http://ir.dlut.edu.cn/news/detail/486>

<https://www.jianshu.com/p/c94909b835d6>