# 卷积的序列到序列学习[[1]](#footnote-1)

赵淼译

(北京交通大学电子与信息工程学院，电子科学与技术，15650761201)

## 摘要

最先进的S2S的学习映射将输入序列映射为一个可变长度的输出序列是通过RNN。这里作者引进了一个完全基于CNN的结构。相对于循环模型，这种模型可以在训练过程中完全并联运行更适合在GPU上运行并且优化也更加容易。我们使用门控线性单元简化了梯度传播，为每个解码器加了一个单独的注意力模块。这种模型准确率提升，并且速度更快。

## 1 介绍

序列到序列的学习在许多任务上取得了成功。主流的编码输入序列的方法是对输入序列进行编码使用双向RNN和通过另一个解码RNN获得可变长度的输出。这两者可以通过一个软注意力机制接入。

卷积网络由几个优点但并不常用。相对与循环网络卷积网络结构会获得固定大小的上下文向量表示，这种上下文可以通过多个层叠加使上下文向量覆盖范围逐渐增大。可以精确的控制建模依赖关系的最大长度。卷积网络不依赖于过去时间步的计算，因此允许一个并行的计算。这与RNN形成对比，RNN通过隐藏层的状态保存获得过去的信息，由此难以实现序列的并行计算。

多层卷积神经网络在输入序列上创建了层次化地表示，在这个序列中，附近的输入元素在较低的层次上相互作用，远处的元素在更高层次上相互作用。层次化的结构提供了一个更短的路径捕捉长期的依赖关系相对于RNN的链式结构。我们获得一个特征表示捕捉关系在一个n个单词的窗口上通过应用一个O（n/k）的卷积操作，核的宽度为k，相对于一个线性的数目O（n）。输入到卷积网络通过固定数目的卷积核和非线性操作，而RNN对使用了高达n个操作和非线性操作对于第一个单词和最后一个单词只是一个孤立的子集。固定数量的非线性应用也可以使学习得到放松。

最近的工作在卷积网络上包括引入循环池化在一系列卷积层或是使用不带有注意力的翻译机制。过去的研究集中在小型数据集上，在大数据集上，解码器使用RNN。

这里应用架构是完全的卷积的，使用了门控线性单元和残差连接。我们也使用注意力机制在每个解码层和证明了增加注意力层的开销是微不足道的。

以往的序列转换依靠复杂的循环和卷积神经网络作为编解码器，这里提出了一个简单

## 2 循环的序列到序列学习

LSTM GRU 注意力机制

在多层的时候使用短期连接或是残差连接。双向RNN模拟过去和未来的细

过去的序列模型难以并行的处理，将位置与时间步对齐，限制了使用。注意力机制已经成为各种任务中引人注目的序列建模和转导模型的一个组成部分，允许对依赖进行建模，而不考虑它们在输入或输出序列中的距离。然而，除了少数情况之外，这种关注机制与一个经常出现的网络一起使用。

这里提出了一个转换器，取代了循环结构，使用注意力机制捕捉输入和输出的全局依赖性，这种转换器允许并行处理。

## 2 背景

减少顺序计算的目标也构成扩展的神经网络GPU [16]、ByteNet [18]和ConvS2S [9]的基础，它们都使用卷积神经网络作为基本构建模块、并行计算所有输入和输出位置的隐藏表示。 在这些模型中，关联任意两个输入和输出位置的信号所需的操作次数会随着位置之间的距离而增加。

Self-attention，有时称为intra-attention，是一种attention机制，它关联单个序列的不同位置以计算序列的表示。

端到端的内存网络基于循环attention机制，而不是序列对齐的循环，并且已被证明在简单语言的问题回答和语言建模任务中表现良好

## 3 模型架构

大多数有竞争力神经序列转换模型都是有一个编解码的架构。这里，编码器映射一个用符号表示的输入序列(x1,...,xn) 到一个连续的表示z = (z1,...,zn)。 根据z，解码器生成符号的一个输出序列(y1,...,ym) ，一次一个元素。 在每一步中，模型都是自回归的[10]，当生成下一个时，消耗先前生成的符号作为附加输入。

转换器跟随了这种模块，将自注意力，点积和全连接层堆叠起来获得编解码器。



编码器：由N为6的完全相同的堆叠而成。第一层是一个multi-head self-attention机制，第二层是一个简单的、位置完全连接的前馈网络。我们对每个子层再采用一个残差连接[11] ，接着进行层标准化[1]。 也就是说，每个子层的输出是LayerNorm(x + Sublayer(x))，其中Sublayer(x) 是由子层本身实现的函数。 为了方便这些残差连接，模型中的所有子层以及嵌入层产生的输出维度都为dmodel = 512。

解码器：解码器同样由N = 6 个完全相同的层堆叠而成。 除了每个编码器层中的两个子层之外，解码器还插入第三个子层，该层对编码器堆栈的输出执行multi-head attention。 与编码器类似，我们在每个子层再采用残差连接，然后进行层标准化。 我们还修改解码器堆栈中的self-attention子层，以防止位置关注到后面的位置。 这种掩码结合将输出嵌入偏移一个位置，确保对位置的预测 i 只能依赖小于i 的已知输出。

应用博客：<https://blog.csdn.net/tg229dvt5i93mxaq5a6u/article/details/78422216>

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。

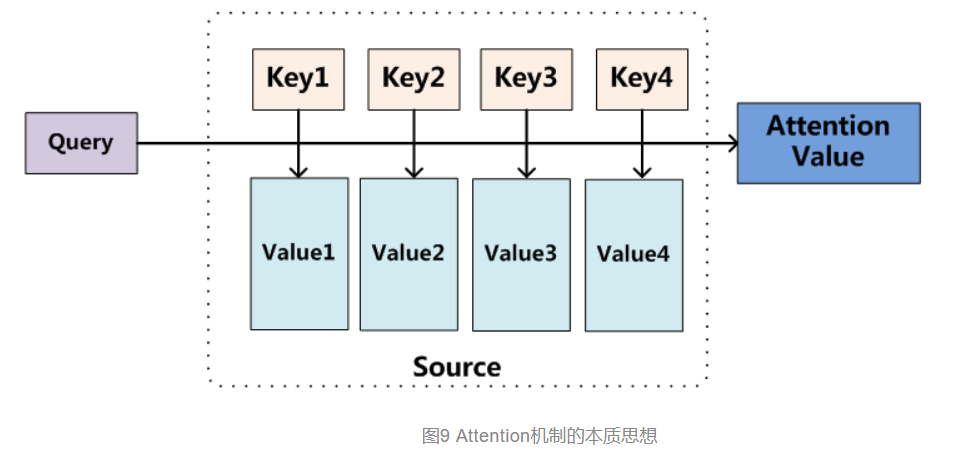
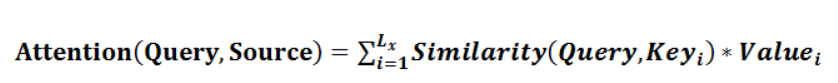


图9 Attention机制的本质思想

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：



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。

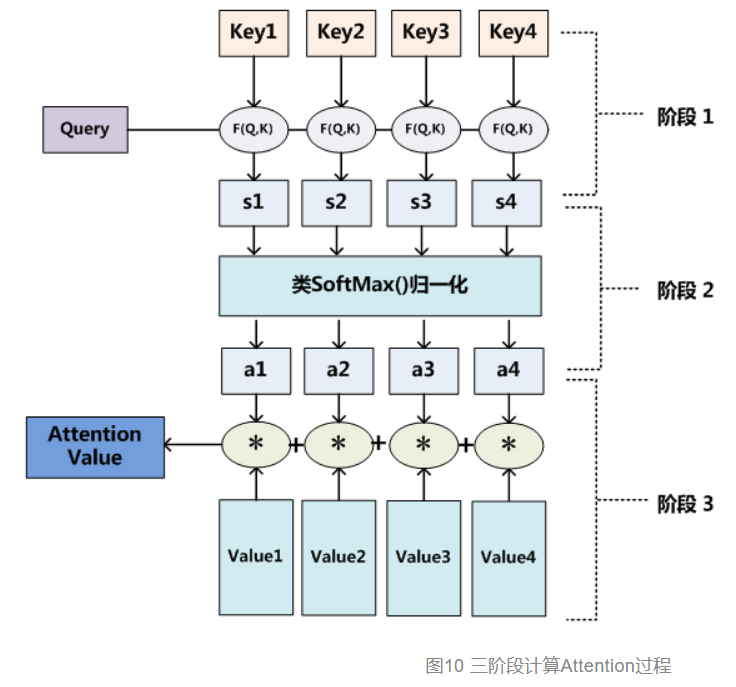
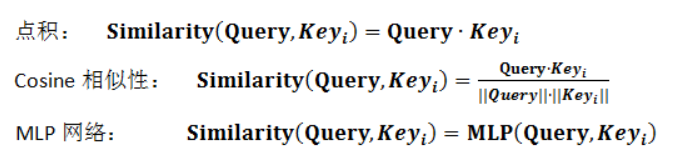
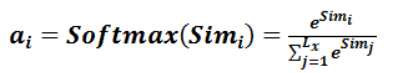


图10 三阶段计算Attention过程

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，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：



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：



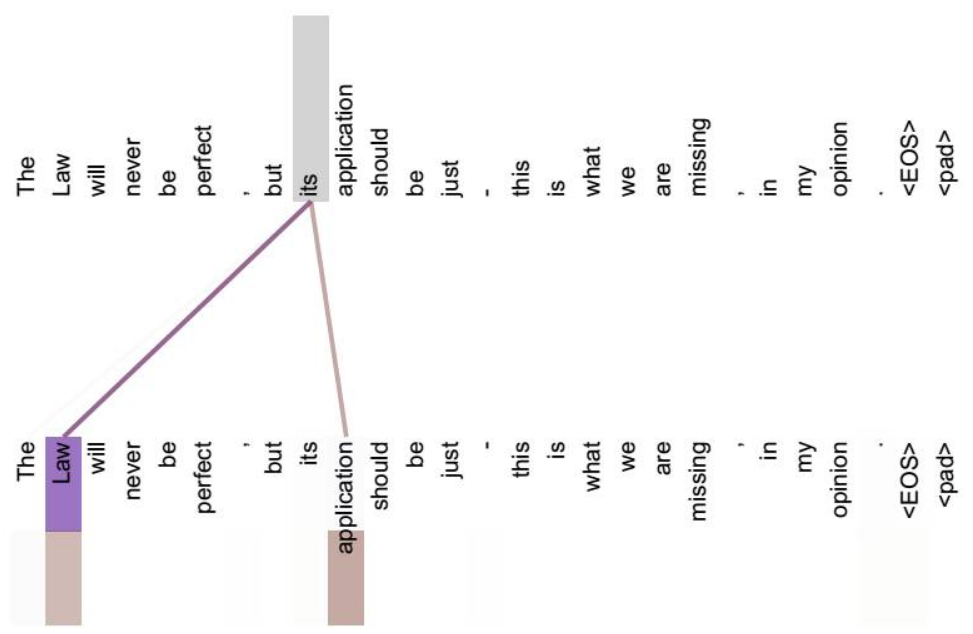
通过如上三个阶段的计算，即可求出针对Query的Attention数值，目前绝大多数具体的注意力机制计算方法都符合上述的三阶段抽象计算过程。

Self Attention模型

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。



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。

注意力机制可以描述为对于将查询与一组关键字-值对之间的映射到一个输出上，这里查询，关键字，值都是向量。输出被表示为值的一个加权和，对于每个值的权重通过查询和相应关键字之间的相关性确定。

### 按比例缩放的注意力机制

这里的注意力机制称为按比例缩放的注意力机制，输入包含dk维的关键字和查询值向量，值的维度为dv。计算查询值和关键字之间的点积，除以根号dk，然后应用softmax函数获得值的权重。

|  |  |
| --- | --- |
|  | 1 |

这里提到了两种注意力模型，一种是点积的，一种是加法，点击速度快，节省空间，在dk比较小时两者性能较接近，在dk比较大时，加法比点积性能好，这力认为对于较大的dk值，点积在幅度上增长过快，使softmax函数进入梯度极小的区域，因此使用缩放因子进行调节。

### 多抽头的注意力模型

这里做着发现将query、key和value分别用不同的，可学习的线性映射h倍到到相应的维度而不是使用单一的注意力函数更有效。这里并行的进行多个注意力函数，产生dv维的输出值，然后进行拼接得到最后的值。

多抽头结构允许模型的不同表示子空间联合关注不同位置的信息。 如果只有一个attention head，它的平均值会削弱这个信息。

|  |  |
| --- | --- |
|  | 1 |

其中，映射为参数矩阵WiQ ∈ ℝdmodel×dk , WiK ∈ ℝdmodel×dk , WiV ∈ ℝdmodel×dv 及W O ∈ ℝhdv×dmodel。

在这项工作中，我们采用h = 8 个并行attention层或head。 对每个head，我们使用dk =dv =dmodel ∕ h = 64。 由于每个head的大小减小，总的计算成本与具有全部维度的单个head attention相似。

### 我们注意力模型的应用

主要有三种方式：

1.编解码attention层，query来自前一个解码层，key和value来自解码器输出，允许解码器中的每个位置能关注到输入序列中的所有位置。这模仿序列到序列模型中典型的编码器—解码器的attention机制。

2.自编码注意力层。在self-attention层中，所有key，value和query来自相同的空间，这里来自与编码器浅层的输出，编码器的每个位置都能关注到编码器每一层的所有位置。

3.相似的，相似的解码器每个允许关注到包括这个位置的各个位置，因此需要避免向左的信息流保持自回归的属性。这里对softmax层的不合法连接进行屏蔽，设置为负无穷大。

### 点积前向传播网络

除了attention子层之外，我们的编码器和解码器中的每个层都包含一个完全连接的前馈网络，该前馈网络单独且相同地应用于每个位置。 它由两个线性变换组成，之间有一个ReLU激活。

尽管线性变换在不同位置上是相同的，但它们使用层与层之间的不同参数。 它的另一种描述方式是两个内核大小为1的卷积。 输入和输出的维度为dmodel = 512，内部层的维度为dff = 2048。

### 嵌入和softmax

在这里使用了词嵌入技术，softmax预测下一个词的概率。在我们的模型中，两个嵌入层之间和pre-softmax线性变换共享相同的权重矩阵，类似于[30]。 在嵌入层中，我们将这些权重乘以â -----
  dmodel。

### 位置编码

这里的attetion网络不包含循环和卷积，没有位置的信息，因此需要注入字符的相对和绝对位置的信息。这里位置编码和嵌入模型具有相同的维度，因此可以作加法。这里有多种位置编码。

这里使用正、余弦函数作为位置编码。

|  |  |
| --- | --- |
|  | 1 |

这种方法位置编码的每个维度对应一个正弦曲线，波长形成集合级数，从2pi到10000\*2pi。容易学习到相对位置，对于固定偏移k，PEpos+k可以表示为PEpos的线性函数。

这里还允许模型推断比训练期间遇到的更长的序列。

## 4 为什么自注意力

这里对比了自注意力层和循环层，卷积层对于将可变长度的符号序列映射为等长度的序列。例如典型的编解码网络中的隐藏层。

这里关心的使总计算量。能够并行计算的计算量。第三是路径长度。

这里attention层明显降低了输入到输出的最短路径，更容易学习长期依赖性。



当序列长度n小于维度d时计算复杂度更好。对于长序列可以考虑使用受限的自注意力。这回增加最大的路径长度。

卷积层计算复杂度较高，相对路径会比attetion稍长，但会考虑位置关系。自注意力模型可以产生更可解释的模型。

## 训练

残差丢弃 我们将丢弃[33]应用到每个子层的输出，在将它与子层的输入相加和规范化之前。 此外，在编码器和解码器堆栈中，我们将丢弃应用到嵌入和位置编码的和。 对于基本模型，我们使用Pdrop = 0.1丢弃率。

Label Smoothing 在训练过程中，我们使用的label smoothing的值为ϵls = 0.1[36]。 这让模型不易理解，因为模型学得更加不确定，但提高了准确性和BLEU得分。

在表3的行（A）中，我们改变attention head的数量和attention key和value的维度，保持计算量不变，如3.2.2节所述。 虽然只有一个head attention比最佳设置差0.9 BLEU，但质量也随着head太多而下降。

### 变体对比



在表3行（B）中，我们观察到减小key的大小dk会有损模型质量。 这表明确定兼容性并不容易，并且比点积更复杂的兼容性函数可能更有用。 我们在行（C）和（D）中进一步观察到，如预期的那样，更大的模型更好，并且丢弃对避免过度拟合非常有帮助。 在行（E）中，我们用学习到的位置嵌入[9]来替换我们的正弦位置编码，并观察到与基本模型几乎相同的结果。

我们对基于attention的模型的未来感到兴奋，并计划将它们应用于其他任务。 我们计划将Transformer扩展到除文本之外的涉及输入和输出模式的问题，并调查局部的、受限attention机制以有效处理大型输入和输出，如图像、音频和视频。 让生成具有更少的顺序性是我们的另一个研究目标。

1. 赵淼 电信研1706班 电子科学与技术 17120036 15650761201 [↑](#footnote-ref-1)