Attention（注意力机制）

注意力机制：我们会把我们的焦点聚焦在比较重要的事物上

你会注意什么？

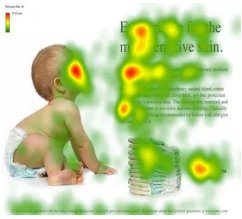
大数据（什么数据都有，重要的，不重要的）

对于重要的数据，我们要使用

对于不重要的数据，我们不太想使用

但是，对于一个模型而言（CNN、LSTM），很难决定什么重要，什么不重要

由此，注意力机制诞生了（有人发现了如何去在深度学习的模型上做注意力）



红色的是科学家们发现，如果给你一张这个图，你眼睛的重点会聚焦在红色区域

人--》看脸

文章看标题

段落看开头

后面的落款

这些红色区域可能包含更多的信息，更重要的信息

我（查询对象 Q），这张图（被查询对象 V）

我看这张图，第一眼，我就会去判断哪些东西对我而言更重要，哪些对我而言又更不重要（去计算 Q 和 V 里的事物的重要度）

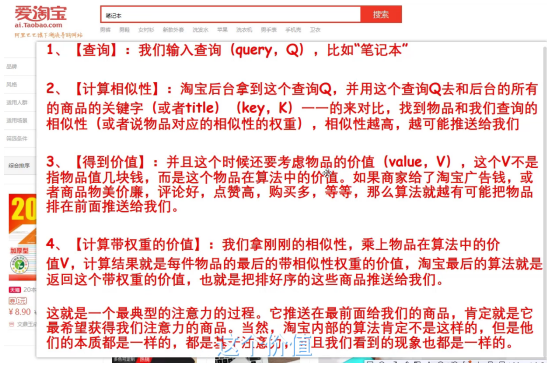
重要度计算，其实是不是就是相似度计算（更接近），点乘其实是求内积（不要关心为什么可以）

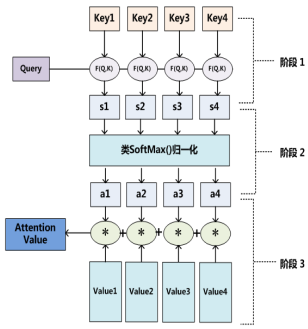
Q，$K = k_1, k_2, \cdots, k_n$，我们一般使用点乘的方式

通过点乘的方法计算Q 和 K 里的每一个事物的相似度，就可以拿到 Q 和k_1的相似值a_1，Q 和k_2的相似值a_2，Q 和k_n的相似值 a_n

做一层 $softmax(a_1, a_2, \cdots, a_n)$就可以得到概率

进而就可以找出哪个对Q 而言更重要了





我们还得进行一个汇总，当你使用 Q 查询结束了后，Q 已经失去了它的使用价值了，我们最终还是要拿到这张图片的，只不过现在的这张图片，它多了一些信息（多了于我而言更重要，更不重要的信息在这里）

V = $(v_1, v_2, \cdots, v_n)$

$(a_1, a_2, \cdots, a_n) * +(v_1, v_2, \cdots, v_n) = (a_1 * v_1 + a_2 * v_2 + \cdots + a_n * v_n)$ = V'

这样的话，就得到了一个新的 V'，这个新的 V' 就包含了，哪些更重要，哪些不重要的信息在里面，然后用 V' 代替 V

一般 K=V，在 Transformer 里，K!=V 可不可以，可以的，但是 K 和 V 之间一定具有某种联系，这样的 QK 点乘才能指导 V 哪些重要，哪些不重要

注意力机制是一个很宽泛（宏大）的一个概念，QKV 相乘就是注意力，但是他没有规定 QKV 是怎么来的

通过一个查询变量 Q，去找到 V 里面比较重要的东西

假设 K==V，然后 QK 相乘求相似度A，然后 AV 相乘得到注意力值Z，这个 Z 就是 V 的另外一种形式的表示

Q 可以是任何一个东西，V 也是任何一个东西， K往往是等同于 V 的（同源），K和 V 不同源不相等可不可以

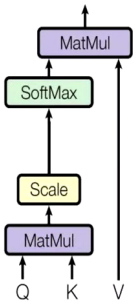
他没有规定 QKV 怎么来，他只规定 QKV 怎么做

不仅规定了 QKV 同源，而且固定了 QKV 的做法

本质上 QKV 可以看做是相等的 对于一个词向量（不一定准确），做的是空间上的对应，乘上了参数矩阵，依然代表 X

Self-Attention

自注意力机制，特别狭隘，属于注意力机制的，注意力机制包括自注意力机制的



torch.matmul(input, other) → Tensor
计算两个张量input和other的矩阵乘积

scale调整规模

QK 相乘求相似度，做一个 scale（未来做 softmax 的时候避免出现极端情况）

然后做 Softmax 得到概率

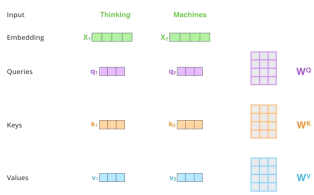
新的向量表示了K 和 V（K==V），然后这种表示还暗含了 Q 的信息（于 Q 而言，K 里面重要的信息），也就是说，挑出了 K 里面的关键点

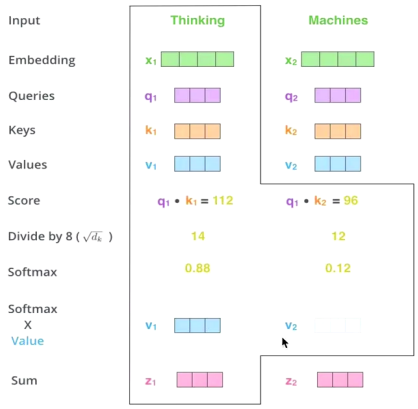
Self-Attention 的关键点再于，不仅仅是 K≈V≈Q 来源于同一个X，这三者是同源的

通过 X 找到 X 里面的关键点

并不是 K=V=Q=X，而是通过三个参数 W_Q, W_K, W_V

1. Q、K、V的获取
   1.



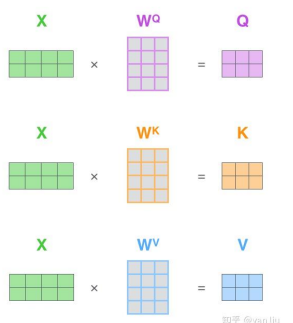


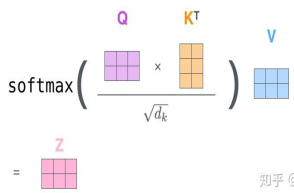
z_1表示的就是 thinking 的新的向量表示

对于 thinking，初始词向量为x_1

现在我通过 thinking machines 这句话去查询这句话里的每一个单词和 thinking 之间的相似度

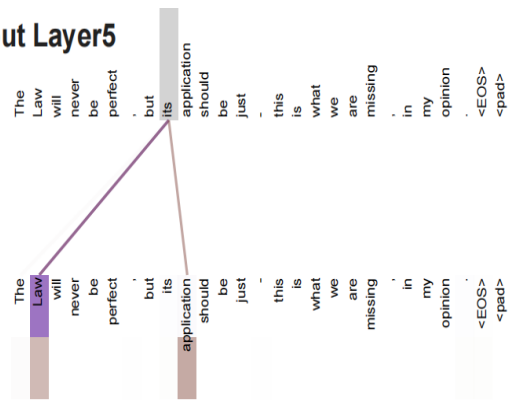
新的z_1依然是 thinking 的词向量表示，只不过这个词向量的表示蕴含了 thinking machines 这句话对于 thinking 而言哪个更重要的信息

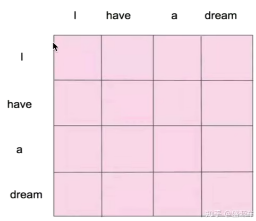




知乎 @yan liu

Input-Input Layer5





不做注意力，its 的词向量就是单纯的 its，没有任何附加信息

也就是说 its 有 law 这层意思，而通过自注意力机制得到新的 its 的词向量，则会包含一定的 laws 和 application 的信息

交叉注意力机制

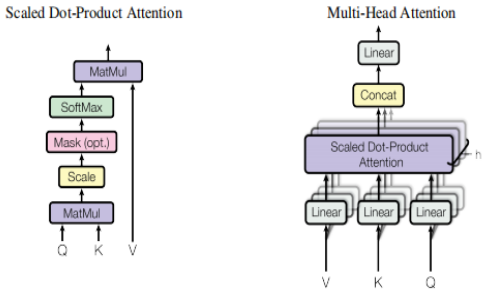
Q 和 V 不同源，但是 K 和 V 同源

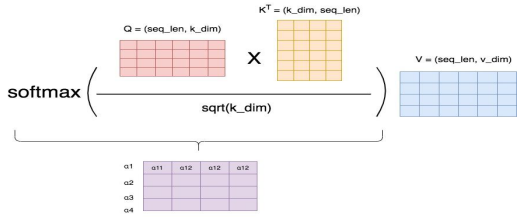
cyd 注意力机制

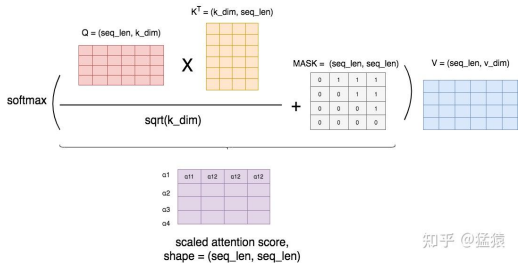
Q 和 V 同源，Q 和 K 不同源

xxx 注意力机制

Q 必须为 1，K 和 V 不同源







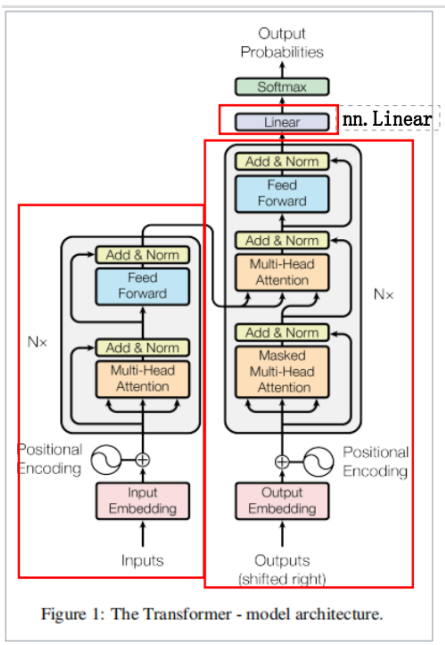
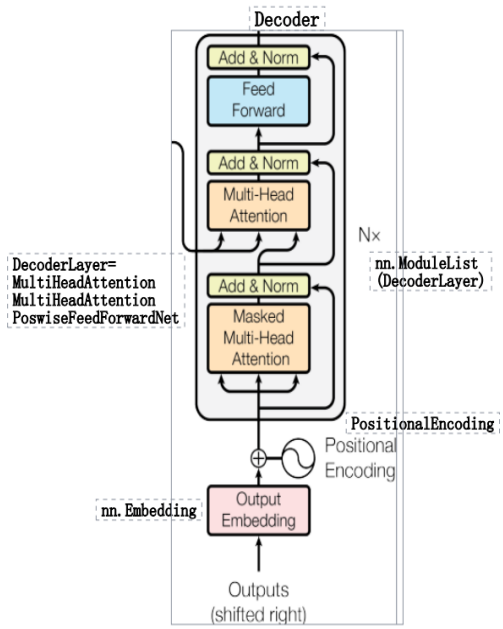
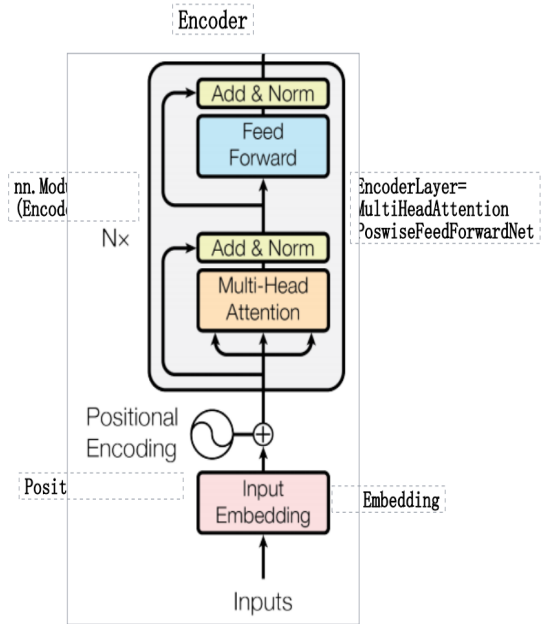


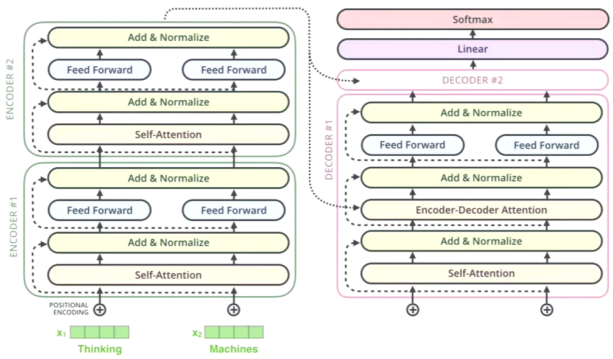
Figure 1: The Transformer - model architecture.





https://jalammar.github.io/illustrated-transformer/

https://www.tensorflow.org/tutorials/text/transformer?hl=zh-cn#%E6%8D%9F%E5%A4%B1%E5%87%BD%E6%95%B0%E4%B8%8E%E6%8C%87%E6%A0%87%EF%BC%88loss_and_metrics%EF%BC%89



建立新单词包含和源句子的关系

自注意力

新生成句子	源句子				
	ich	mochte	ein	bier	P
ich	FALSE	FALSE	FALSE	FALSE	TRUE
mochte	FALSE	FALSE	FALSE	FALSE	TRUE
ein	FALSE	FALSE	FALSE	FALSE	TRUE
bier	FALSE	FALSE	FALSE	FALSE	TRUE
P	FALSE	FALSE	FALSE	FALSE	TRUE

竖着的这些都是阿巴阿巴：
阿巴阿巴包含着源句子的信息

新生成句子	源句子				
	S	i	want	a	Pbeer
S	FALSE	FALSE	FALSE	FALSE	FALSE
i	FALSE	FALSE	FALSE	FALSE	FALSE
want	FALSE	FALSE	FALSE	FALSE	FALSE
a	FALSE	FALSE	FALSE	FALSE	FALSE
Pbeer	FALSE	FALSE	FALSE	FALSE	FALSE

建立新单词包含和源句子关系

掩码+自注意力

新生成句子	源句子				
	S	i	want	a	Pbeer
S	FALSE	TRUE	TRUE	TRUE	TRUE
i	FALSE	FALSE	TRUE	TRUE	TRUE
want	FALSE	FALSE	FALSE	TRUE	TRUE
a	FALSE	FALSE	FALSE	FALSE	TRUE
Pbeer	FALSE	FALSE	FALSE	FALSE	FALSE

新生成句子	源句子				
	S	i	want	a	Pbeer
S	FALSE	TRUE	TRUE	TRUE	TRUE
i	FALSE	FALSE	TRUE	TRUE	TRUE
want	FALSE	FALSE	FALSE	TRUE	TRUE
a	FALSE	FALSE	FALSE	FALSE	TRUE
Pbeer	FALSE	FALSE	FALSE	FALSE	FALSE

模糊，注意力就是让我们建立联系而已，方便计算机更注意哪些东西，某个相似度高，计算机就更注意哪个，然后黑盒子出结果。

新句子包含源语句信息

互注意力

新生成句子	源句子				
	ich	mochte	ein	bier	P
S	FALSE	FALSE	FALSE	FALSE	TRUE
i	FALSE	FALSE	FALSE	FALSE	TRUE
want	FALSE	FALSE	FALSE	FALSE	TRUE
a	FALSE	FALSE	FALSE	FALSE	TRUE
Pbeer	FALSE	FALSE	FALSE	FALSE	TRUE

Q来自decoder_inputs经掩码+自相关后

查找包含源语句S i want a Pbeer的阿巴阿巴1单词

K,V来自encoder_outputs
也就是包含源语句ich mochte ein bier P的阿巴阿巴2单词被查找

建立起阿巴阿巴1和阿巴阿巴2联系
有这些联系我们黑盒子预测就行

这里我们建立了一些联系，包含了某些关系

中间这些其实都是位置，我们只不过用注意力机制搞些关系而已
输出的是什么不知道，我们最后liner+softmax是要转换语种对应单词个数为输出就可以了，然后梯度下降，不断训练。
--->预测值与真实值不断比较，训练，使yloss不断下降即可。

什么联系，不知道，位置上有联系，语义上有联系？
黑盒子，搞不明白

Attention Is All You Need

Ashish Vaswani[*]
Google Brain
avaswani@google.com

Noam Shazeer[*]
Google Brain
noam@google.com

Niki Parmar[*]
Google Research
nikip@google.com

Jakob Uszkoreit[*]
Google Research
usz@google.com

Llion Jones[*]
Google Research
llion@google.com

Aidan N. Gomez[*] [†]
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser[*]
Google Brain
lukaszkaiser@google.com

Illia Polosukhin[*] [‡]
illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

主流的序列转换模型都是基于复杂的循环神经网络或卷积神经网络，且都包含一个encoder和一个decoder

我们提出了一个新的、简单的网络架构，Transformer，它只基于单独的attention机制，完全避免使用循环和卷积

在两个翻译任务上表明，我们的模型在质量上更好，同时具有更高的并行性，且训练所需要的时间更少

以前的方法：
主流序列转导模型基于复杂的CNN或RNN，包括编码器和解码器。
有的模型使用注意力机制连接编码器和解码器，达到了最优性能。

缺点：
①难以并行
②时序中过早的信息容易被丢弃
③内存开销大

本文方法
本文提出的Transformer完全摒弃了之前的循环和卷积操作，完全基于注意力机制，拥有更强的并行能力，训练效率也得到较高提升

主流的序列转换模型都是基于复杂的循环神经网络或卷积神经网络，且都包含一个encoder和一个decoder。表现最好的模型还通过attention机制把encoder和decoder联接起来。我们提出了一个新的、简单的网络架构，Transformer，它只基于单独的attention机制，完全避免使用循环和卷积。在两个翻译任务上表明，我们的模型在质量上更好，同时具有更高的并行性，且训练所需要的时间更少。我们的模型在 WMT2014 英语-德语的翻译任务上取得了28.4的BLEU评分。在现有的表现最好模型的基础上，包括整合模型，提高了2个BLEU评分。在WMT2014英语-德语的翻译任务上，我们的模型在8个GPU上训练了3.5天（这个时间只是目前文献中记载的最好的模型训练成本的一小部分），创造了单模型的SOTA结果，BLEU分数为41.8，通过在大量和少量训练数据上所做的英语选区分析工作的成功，表明Transformer能很好的适应于其它任务。

[*]Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

[†]Work performed while at Google Brain.

[‡]Work performed while at Google Research.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

1 Introduction

之前语言模型和机器翻译的方法和不足
方法： RNN、LSTM、GRU、Encoder-Decoder
不足：
①从左到右一步步计算，因此难以并行计算
②过早的历史信息可能被丢弃，时序信息一步一步向后传递
③内存开销大，训练时间慢

近期工作和问题
近期一些工作通过分解技巧和条件计算提高了计算效率，但是顺序计算的本质问题依然存在

RNN,LSTM,GRU,Gated Recurrent Neural Networks 在序列建模和转换任务上，比如语言模型和机器翻译，已经是大家公认的取得SOTA结果的方法。自此，无数的努力继续推动递归语言模型和encoder-decoder体系结构的界限。

Recurrent neural networks, long short-term memory [13] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and transduction problems such as language modeling and machine translation [35, 2, 5]. Numerous efforts have since continued to push the boundaries of recurrent language models and encoder-decoder architectures [38, 24, 15].

递归模型通常沿输入和输出序列的符号位置进行因子计算。在计算时将位置与步骤对齐，它们生成一系列隐藏状态ht，t位置的ht使用它的前驱ht?1和当前的输入生成。这种内部的固有顺阻碍了训练样本的并行化，在序列较长时，这个问题变得更加严重，因为内存的限制限制了样本之间的批处理。最近的工作通过因子分解技巧[21]和条件计算[32]在计算效率方面取得了显著的提高，同时也提高了后者的模型性能。然而，顺序计算的基本约束仍然存在。

Recurrent models typically factor computation along the symbol positions of the input and output sequences. Aligning the positions to steps in computation time, they generate a sequence of hidden states h_t, as a function of the previous hidden state h_{t-1} and the input for position t. This inherently sequential nature precludes parallelization within training examples, which becomes critical at longer sequence lengths, as memory constraints limit batching across examples. Recent work has achieved significant improvements in computational efficiency through factorization tricks [21] and conditional computation [32], while also improving model performance in case of the latter. The fundamental constraint of sequential computation, however, remains.

本文改进

（1）引入注意力机制： 注意力机制可以在RNN上使用，通过注意力机制把encoder的信息传给decoder，可以允许不考虑输入输出序列的距离建模。

（2）提出Transformer ： 本文的Transformer 完全不用RNN，这是一种避免使用循环的模型架构，完全依赖于注意机制来绘制输入和输出之间的全局依赖关系，并行度高，计算时间短

在各种各样的任务中，注意力机制已经成为各种引人注目的序列模型和转换模型中的不可或缺的组成部分，它允许对依赖关系建模，而不需要考虑它们在输入或输出序列中的距离。然而，在除少数情况外的所有情况下[27]，这种注意机制都与一个递归网络结合使用

Attention mechanisms have become an integral part of compelling sequence modeling and transduction models in various tasks, allowing modeling of dependencies without regard to their distance in the input or output sequences [2, 19]. In all but a few cases [27], however, such attention mechanisms are used in conjunction with a recurrent network.

在这项工作中，我们提出了Transformer，这是一种避免使用循环的模型架构，完全依赖于注意机制来绘制输入和输出之间的全局依赖关系。Transformer允许更显著的并行化，使用8个P100 gpu只训练了12小时，在翻译质量上就可以达到一个新的SOTA

In this work we propose the Transformer, a model architecture eschewing recurrence and instead relying entirely on an attention mechanism to draw global dependencies between input and output. The Transformer allows for significantly more parallelization and can reach a new state of the art in translation quality after being trained for as little as twelve hours on eight P100 GPUs.

2 Background

CNN代替RNN
优点：
①减小时序计算
②可以输出多通道

卷积的感受野是一定的，距离间隔较远的话就需要多次卷积才能将两个远距离的像素结合起来，所以对长时序来讲比较难

减少序列计算的目标也成就了 Extended Neural GPU [16], ByteNet [18] 和 ConvS2S [9]的基础，它们都使用了卷积神经网络作为基础模块，并行计算所有输入和输出位置的隐藏表示。在这些模型中，将来自两个任意输入或输出位置的信号关联起来所需的操作数，随位置间的距离而增长，ConvS2S为线性增长，ByteNet为对数增长。这使得学习远距离位置之间的依赖性变得更加困难[12]. 在Transformer中，这种情况被减少到了常数次操作，虽然代价是由于平均注意力加权位置信息降低了有效分辨率，如第3.2节所述，我们用多头注意力抵消这种影响

The goal of reducing sequential computation also forms the foundation of the Extended Neural GPU [16], ByteNet [18] and ConvS2S [9], all of which use convolutional neural networks as basic building block, computing hidden representations in parallel for all input and output positions. In these models, the number of operations required to relate signals from two arbitrary input or output positions grows in the distance between positions, linearly for ConvS2S and logarithmically for ByteNet. This makes it more difficult to learn dependencies between distant positions [12]. In the Transformer this is reduced to a constant number of operations, albeit at the cost of reduced effective resolution due to averaging attention-weighted positions, an effect we counteract with Multi-Head Attention as described in section 3.2.

自注意力
有时也称为内部注意力，是一种将单个序列的不同位置关联起来以计算序列表示的注意机制。 自注意力已成功用于各种任务，包括阅读理解、抽象摘要、文本蕴涵和学习与任务无关的句子表示

self-attention,有时也叫做内部注意力，是一种注意力机制，它将一个序列的不同位置联系起来，以计算序列的表示。self-attention 已经成功的运用到了很多任务上，包括阅读理解、抽象摘要、语篇蕴涵和学习任务无关的句子表征等

Self-attention, sometimes called intra-attention is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence. Self-attention has been used successfully in a variety of tasks including reading comprehension, abstractive summarization, textual entailment and learning task-independent sentence representations [4, 27, 28, 22].

端到端记忆网络
基于循环注意机制而不是序列对齐循环，并且已被证明在简单语言问答和语言建模任务中表现良好

已经被证明，端到端的记忆网络使用循环attention机制替代序列对齐的循环，在简单的语言问答和语言建模任务中表现良好

End-to-end memory networks are based on a recurrent attention mechanism instead of sequence-aligned recurrence and have been shown to perform well on simple-language question answering and language modeling tasks [34].

Transformer优点
用注意力机制可以直接看一层的数据，就规避了CNN的那个缺点

然而，据我们所知，Transformer是第一个完全依赖于self-attetion来计算其输入和输出表示而不使用序列对齐的RNN或卷积的转换模型，在下面的章节中，我们将描述Transformer，motivate ，self-attention，并讨论它相对于[17,18]和[9]

To the best of our knowledge, however, the Transformer is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence-aligned RNNs or convolution. In the following sections, we will describe the Transformer, motivate self-attention and discuss its advantages over models such as [17, 18] and [9].

3 Model Architecture

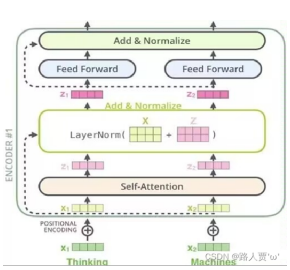
大多数有竞争力的序列转换模型都有encoder-decoder结构构。这里，encoder将符号表示的输入序列(x 1 , . . . , x n)映射成一个连续表示的序列z = (z 1 , . . . , z n)。给定z，解码器以一次生成一个字符的方式生成输出序列(y 1 , . . . , y m)。在每一步，模型都是自回归的[10]，在生成下一个字符时，将先前生成的符号作为附加输入

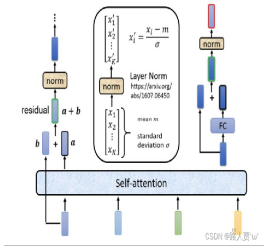
Most competitive neural sequence transduction models have an encoder-decoder structure [5, 2, 35]. Here, the encoder maps an input sequence of symbol representations $(x_1, ..., x_n)$ to a sequence of continuous representations $\mathbf{z} = (z_1, ..., z_n)$. Given $\mathbf{z}$, the decoder then generates an output sequence $(y_1, ..., y_m)$ of symbols one element at a time. At each step the model is auto-regressive [10], consuming the previously generated symbols as additional input when generating the next.

Transformer遵循这个总体架构，使用堆叠的self-attention层、point-wise和全连接层，分别用于encoder和decoder，如图1的左半部分和右半部分所示

编码器 encoder
将一个长为n的输入（如句子），序列(x1, x2, … xn)映射为(z1, z2, …, zn)（机器学习可以理解的向量）
encoder由n个相同层组成，重复6个layers，每个layers会有两个sub-layers，每个sub-layers里第一个layer是multi-head attention，第二个layer是 simple，position-wise fully connected feed-forward network，简称 MLP。
每个sub-layer的输出都做一个残差连接和layerNorm。计算公式：LayerNorm(x + Sublayer(x))，Sublayer(x) 指 self-attention 或者 MLP。
残差连接需要输入和输出的维度一致，所以每一层的输出维度在transformer里都是固定的，都是512维。
与CNN不一样的是，MLP的空间维度是逐层下降，CNN是空间维度下降，channel维度上升

解码器 decoder
decoder 拿到 encoder 的输出，会生成一个长为 m 的序列(y1, y2, … , ym)。n 和 m 可以一样长、也可以不一样长，编码时可以一次性生成，解码时只能一个个生成（auto-regressive 自回归模型）
decoder同样由n个相同层组成。
除了encoder中的两个子层外，decoder还增加了一个子层：对encoder层的输出执行多头注意力。
另外对自注意力子层进行修改(Mask)，防止某个position受后续的position的影响。确保位置i的预测只依赖于小于i的位置的已知输出。
输出就是标准的 Linear+softmax





LayerNorm模块
LayerNorm是层标准化，和 BatchNorm 在很多时候几乎一样，除了实现方法不同。
BN取的是不同样本的同一个特征，而LN取的是同一个样本的不同特征。在BN和LN都能使用的场景中，BN的效果一般优于LN，原因是基于不同数据，同一特征得到的归一化特征更不容易损失信息。
但是有些场景是不能使用BN的，例如batchsize较小或者在RNN中，这时候可以选择使用LN。LN得到的模型更稳定且起到正则化的作用。RNN能应用到小批量和RNN中是因为LN的归一化统计量的计算是和batchsize没有关系的。

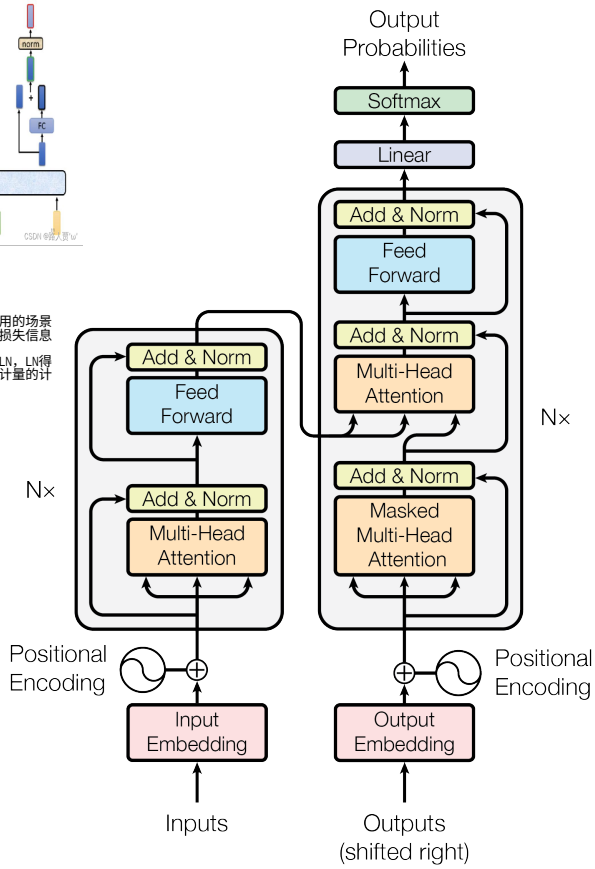


Figure 1: The Transformer - model architecture.

The Transformer follows this overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder, shown in the left and right halves of Figure 1, respectively.

3.1 Encoder and Decoder Stacks

Encoder: The encoder is composed of a stack of $N = 6$ identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. We employ a residual connection [11] around each of the two sub-layers, followed by layer normalization [1]. That is, the output of each sub-layer is $\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$, where $\mathrm{Sublayer}(x)$ is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension $d_{\text{model}} = 512$.

Encoder:
encoder由N(N=6)个完全相同的layer堆叠而成.每层有两个子层。
第一层是multi-head self-attention机制，第二层是一个简单的、位置全连接的前馈神经网络。
我们在两个子层的每一层后采用残差连接[11]，接着进行layer normalization[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embedding层产生的输出维度都为dmodel=512。

Decoder: The decoder is also composed of a stack of $N = 6$ identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization. We also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i.

Decoder:
decoder也由N(N=6)个完全相同的layer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，该子层对编码器堆栈的输出执行multi-head attention操作，与encoder相似，我们在每个子层的后面使用了残差连接，之后采用了layer normalization。
我们也修改了decoder stack中的 self-attention 子层，以防止当前位置信息中被添加进后续的位置信息。这种掩码与偏移一个位置的输出embedding相结合， 确保对第i个位置的预测 只能依赖小于i的已知输出。

3.2 Attention

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum

Attention机制可以描述为将一个query和一组key-value对映射到一个输出，其中query，keys，values和输出均是向量。输出是values的加权求和，其中每个value的权重 通过query与相应key的兼容函数来计算。

注意力机制是对每个 Q 和 K做内积，将它作为相似度。

当两个向量做内积时，如果他俩的 d 相同，向量内积越大，余弦值越大，相似度越高。

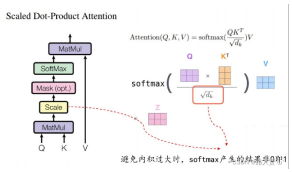
如果内积值为0，他们是正交的，相似度也为0。

Q： query(查询)
K： key(键)
V： value(值)

Q就在目标 target区域，就是decoder那块，K和V都在源头 ，就是encoder区域。
自注意力则是QKV都在一个区域，要么都在decoder要么都在encoder。 目的就是为了能够发现一句话内部或者一个时序内部的关联信息。

Scaled Dot-Product Attention是特殊attention，输入包括查询Q和键K的维度dk 以及值V的维度dv?。计算查询和键的点积，将每个结果除 ，然后用 softmax() 函数来获得值的权重。

在实际使用中，我们同时计算一组查询的注意力函数，并一起打包成矩阵 Q。键和值也一起打包成矩阵 K 和 V。



加性attention和点积(乘性)attention区别
①点积attention与我们的算法一致，除了缩放因子
②加性attention使用带一个单隐层的前馈网络计算兼容函数

尽管这两种attention在原理复杂度上相似，但点积attention在实践中更快、空间效率更高，因为它可以使用高度优化的矩阵乘法代码。

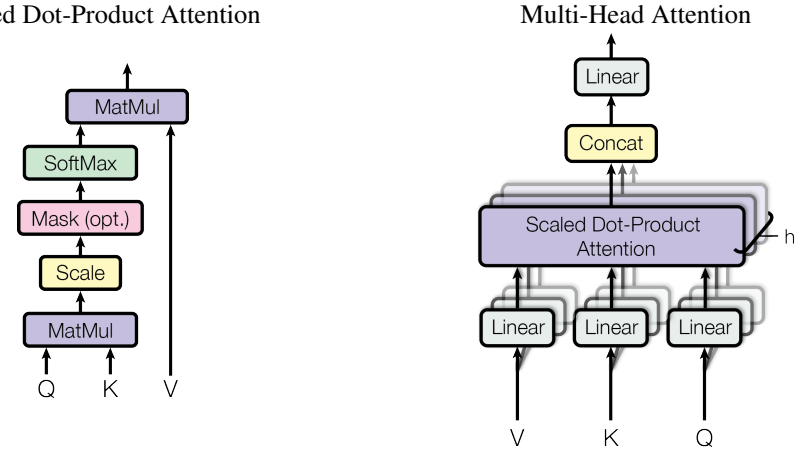


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

3.2.1 Scaled Dot-Product Attention

We call our particular attention "Scaled Dot-Product Attention" (Figure 2). The input consists of queries and keys of dimension d_k, and values of dimension d_v. We compute the dot products of the query with all keys, divide each by $\sqrt{d_k}$, and apply a softmax function to obtain the weights on the values.

我们称我们的特殊attention为Scaled Dot-Product Attention(Figure 2)。输入由query、d_{k}的key和d_{v}的value组成。我们计算query和所有key的点积，再除以\sqrt{d_{k}}然后再通过softmax函数来获取values的权重

In practice, we compute the attention function on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V. We compute the matrix of outputs as:

在实际应用中，我们把一组query转换成一个矩阵Q，同时应用attention函数。key和valuue也同样被转换成矩阵K和矩阵V。我们按照如下方式计算输出矩阵：

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{1}$$

The two most commonly used attention functions are additive attention [2], and dot-product (multi-plicative) attention. Dot-product attention is identical to our algorithm, except for the scaling factor of $\frac{1}{\sqrt{d_k}}$. Additive attention computes the compatibility function using a feed-forward network with a single hidden layer. While the two are similar in theoretical complexity, dot-product attention is much faster and more space-efficient in practice, since it can be implemented using highly optimized matrix multiplication code.

additive attention和dot-product (multi-plicative) attention是最常用的两个attention 函数。dot-product attention除了没有使用缩放因子外，与我们的算法相同。Additive attention使用一个具有单隐层的前馈神经网络来计算兼容性函数。尽管在理论上两者的复杂度相似，但是在实践中dot-product attention要快得多，而且空间效率更高，这是因为它可以使用高度优化的矩阵乘法代码来实现。

While for small values of d_k the two mechanisms perform similarly, additive attention outperforms dot product attention without scaling for larger values of d_k [3]. We suspect that for large values of d_k, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients [4]. To counteract this effect, we scale the dot products by $\frac{1}{\sqrt{d_k}}$.

当d_{k}的值较小时，这两种方法性能表现的相近，当d_{k}比较大时，addtitive attention表现优于 dot-product attention。我们认为对于大的d_{k}?，点积在数量级上增长的幅度大，将softmax函数推向具有极小梯度的区域。为了抵消这种影响，我们对点积扩展\frac{1}{\sqrt{d_{k}}}倍。

3.2.2 Multi-Head Attention

Instead of performing a single attention function with d_{model}-dimensional keys, values and queries, we found it beneficial to linearly project the queries, keys and values h times with different, learned linear projections to d_k, d_k and d_v dimensions, respectively. On each of these projected versions of queries, keys and values we then perform the attention function in parallel, yielding d_v-dimensional

Scaled Dot-Product Attention是特殊attention，输入包括查询Q和键K的维度dk 以及值V的维度dv。计算查询和键的点积，将每个结果除sqrt{d_{k}} ，然后用 softmax() 函数来获得值的权重。

在实际使用中，我们同时计算一组查询的注意力函数，并一起打包成矩阵 Q。键和值也一起打包成矩阵 K 和 V

[4] To illustrate why the dot products get large, assume that the components of q and k are independent random variables with mean 0 and variance 1. Then their dot product, $q \cdot k = \sum_{i=1}^{d_k} q_i k_i$, has mean 0 and variance d_k.

不再使用一个attention函数，而是使用不同的学习到的线性映射将queries，keys和values分别线性投影到 dq、dk 和 dv 维度 h 次。

然后在queries，keys和values的这些投影版本中的每一个上并行执行注意力功能，产生h个注意力函数。

最后将这些注意力函数拼接并再次投影，产生最终输出值。

一个 dot product 的注意力里面，没有什么可以学的参数。具体函数就是内积，为了识别不一样的模式，希望有不一样的计算相似度的办法。
本文的点积注意力先进行了投影，而投影的权重w是可学习的。多头注意力给h次机会学习不一样的投影方法，使得在投影进去的度量空间里面能够去匹配不同模式需要的一些相似函数，然后把 h 个头拼接起来，最后再做一次投影。这种做法有一些像卷积网络 CNN的多输出通道
多头注意力的输入还是 Q 、K 、V，但是输出是将不同的注意力头的输出合并，在投影到WO 里，每个头hi把 Q,K,V 通过 可以学习的 Wq ， Wk ， Wv投影到 dv上，再通过注意力函数，得到head。

output values. These are concatenated and once again projected, resulting in the final values, as depicted in Figure 2.

Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this.

相比于使dmodel维度的queries,keys,values执行一个attention函数，我们发现使用不同的学习到的线性映射把queries，keys 和 values线性映射到dk，dk和dv维度h次是有益的。在queries,keys和values的每个映射版本上，我们并行的执行attention函数，生成维dv输出值。它们被拼接起来再次映射，生成一个最终值，如 Figure 2 中所示

Multi-head attention允许模型把不同位置子序列的表示都整合到一个信息中。如果只有一个attention head，它的平均值会削弱这个信息

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

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 d_{k} = d_{v} = d_{model} / h = 64 d_{k}= d_{v} = d_{model}/h = 64 。 由于每个head尺寸上的减小，总的计算成本与具有全部维度的单个head attention相似。

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

In this work we employ $h = 8$ parallel attention layers, or heads. For each of these we use $d_k = d_v = d_{\text{model}}/h = 64$. Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

3.2.3 Applications of Attention in our Model

The Transformer uses multi-head attention in three different ways:

multi-head attention在Transformer中有三种不同的使用方式：

在encoder-decoder attention层中，queries来自前面的decoder层，而keys和values来自encoder的输出。这使得decoder中的每个位置都能关注到输入序列中的所有位置。 这是模仿序列到序列模型中典型的编码器－解码器的attention机制，例如[38, 2, 9]。
encoder包含self-attention层。 在self-attention层中，所有的key、value和query来自同一个地方，在这里是encoder中前一层的输出。 encoder中的每个位置都可以关注到encoder上一层的所有位置。
类似地，decoder中的self-attention层允许decoder中的每个位置都关注decoder层中当前位置之前的所有位置（包括当前位置）。 为了保持解码器的自回归特性，需要防止解码器中的信息向左流动。我们在scaled dot-product attention的内部 ，通过屏蔽softmax输入中所有的非法连接值（设置为 -∞）实现了这一点。

- In "encoder-decoder attention" layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence. This mimics the typical encoder-decoder attention mechanisms in sequence-to-sequence models such as [38, 2, 9].
- The encoder contains self-attention layers. In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder.
- Similarly, self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position. We need to prevent leftward information flow in the decoder to preserve the auto-regressive property. We implement this inside of scaled dot-product attention by masking out (setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections. See Figure 2.

3.3 Position-wise Feed-Forward Networks

除了encoder子层之外，我们的encder和decoder中的每个层还包含一个全连接的前馈网络，该网络分别单独应用于每一个位置。这包括两个线性转换，中间有一个ReLU激活。

尽管线性变换在不同位置上是相同的，但它们在层与层之间使用不同的参数。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dff = 2048。

In addition to attention sub-layers, each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2}$$

While the linear transformations are the same across different positions, they use different parameters from layer to layer. Another way of describing this is as two convolutions with kernel size 1. The dimensionality of input and output is $d_{\text{model}} = 512$, and the inner-layer has dimensionality $d_{ff} = 2048$.

3.4 Embeddings and Softmax

与其他序列转换模型类似，我们使用学习到的嵌入词向量 将输入字符和输出字符转换为维度为dmodel的向量。我们还使用普通的线性变换和softmax函数将decoder输出转换为预测的下一个词符的概率。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sqrt(dmodel)

Similarly to other sequence transduction models, we use learned embeddings to convert the input tokens and output tokens to vectors of dimension d_{model}. We also use the usual learned linear transformation and softmax function to convert the decoder output to predicted next-token probabilities. In our model, we share the same weight matrix between the two embedding layers and the pre-softmax linear transformation, similar to [30]. In the embedding layers, we multiply those weights by $\sqrt{d_{\text{model}}}$.

Transformer用了三种不同的注意力头：

（1）Encoder 的注意力层： 输入数据在经过 Embedding+位置encoding 后，复制成了三份一样的东西，分别表示 K Q V。同时这个数据既做 key 也做 query 也做value，其实就是一个东西，所以叫自注意力机制。输入了n个 Q，每个Q会有一个输出，那么总共也有n个输出，输出是 V 加权和（权重是 Q 与 K 的相似度）。

（2）Decoder 的注意力层： 这个注意力层就不是自注意力了，其中 K 和 V 来自 Encoder的输出，Q来自掩码多头注意力输出

（3）Decoder 的掩码注意力层： 掩码注意力层就是将t时刻后的数据权重设置为 0，该层还是自注意力的

除了attention子层，我们encoder-decoder框架中每一层都包含一个全连接的前馈网络，它分别相同地应用于每个位置。它由两个线性变换和中间的一个ReLU激活函数组成

Embedding： 特征嵌入，embedding是可以简单理解为通过某种方式将词向量化，即输入一个词输出该词对应的一个向量。（embedding可以采用训练好的模型如GLOVE等进行处理，也可以直接利用深度学习模型直接学习一个embedding层，Transformer模型的embedding方式是第二种，即自己去学习的一个embedding层。）

embeddings将输入和输出tokens转换为向量，线性变换和softmax函数将decoder输出转换为预测的写一个token概率

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

3.5 Positional Encoding

由于我们的模型不包含循环或卷积，为了让模型利用序列的顺序信息，我们必须加入序列中关于字符相对或者绝对位置的一些信息。 为此，我们在encoder和decoder堆栈底部的输入嵌入中添加“位置编码”。 位置编码和嵌入的维度dmodel相同，所以它们两个可以相加。有多种位置编码可以选择，例如通过学习得到的位置编码和固定的位置编码[9]

Since our model contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, we must inject some information about the relative or absolute position of the tokens in the sequence. To this end, we add "positional encodings" to the input embeddings at the bottoms of the encoder and decoder stacks. The positional encodings have the same dimension d_{model} as the embeddings, so that the two can be summed. There are many choices of positional encodings, learned and fixed [9].

在这项工作中，我们使用不同频率的正弦和余弦函数：

In this work, we use sine and cosine functions of different frequencies:

因为transformer模型不包含循环或卷积，输出是V的加权和（权重是 Q与K的相似度，与序列信息无关），对于任意的K-V，将其打乱后，经过注意力机制的结果都一样

但是它顺序变化而值不变，在处理时序数据的时候，一个序列如果完全被打乱，那么语义肯定发生改变，而注意力机制却不会处理这种情况。

其中pos 是位置，i 是维度。也就是说，位置编码的每个维度对应于一个正弦曲线。波长形成了从2π到10000 · 2π的几何数列。我们之所以选择这个函数，是因为我们假设它可以让模型很容易地通过相对位置来学习，因为对任意确定的偏移k，PEpos+k可以表示为PEpos的线性函数

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

在注意力机制的输入中加入时序信息，位置在encoder端和decoder端的embedding之后，用于补充Attention机制本身不能捕捉位置信息的缺陷。

我们还尝试使用预先学习的positional embeddings[9]来代替正弦波，发现这两个版本产生了几乎相同的结果（see Table 3 row (E)）。我们之所以选择正弦曲线，是因为它允许模型扩展到比训练中遇到的序列长度更长的序列。

一次词在嵌入层表示成一个512维的向量，用另一个512维的向量表示位置数字信息的值。用周期不一样的sin和cos函数计算。

where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}.

We also experimented with using learned positional embeddings [9] instead, and found that the two versions produced nearly identical results (see Table 3 row (E)). We chose the sinusoidal version because it may allow the model to extrapolate to sequence lengths longer than the ones encountered during training.

4 Why Self-Attention

一是每层的总计算复杂度

二是可以并行化的计算量，以所需的最小序列操作数衡量

三是网络中长距离依赖关系之间的路径长度

计算远距离依赖一直是序列转换任务中的关键挑战，其中的一个关键因素就是其路径长度。

路径距离越短，学习长距离依赖越容易。

In this section we compare various aspects of self-attention layers to the recurrent and convolutional layers commonly used for mapping one variable-length sequence of symbol representations $(x_1, ..., x_n)$ to another sequence of equal length $(z_1, ..., z_n)$, with $x_i, z_i \in \mathbb{R}^d$, such as a hidden layer in a typical sequence transduction encoder or decoder. Motivating our use of self-attention we consider three desiderata.

One is the total computational complexity per layer. Another is the amount of computation that can be parallelized, as measured by the minimum number of sequential operations required.

The third is the path length between long-range dependencies in the network. Learning long-range dependencies is a key challenge in many sequence transduction tasks. One key factor affecting the ability to learn such dependencies is the length of the paths forward and backward signals have to traverse in the network. The shorter these paths between any combination of positions in the input and output sequences, the easier it is to learn long-range dependencies [12]. Hence we also compare the maximum path length between any two input and output positions in networks composed of the different layer types.

As noted in Table 1, a self-attention layer connects all positions with a constant number of sequentially executed operations, whereas a recurrent layer requires $O(n)$ sequential operations. In terms of computational complexity, self-attention layers are faster than recurrent layers when the sequence

在这一节中，我们将self-attention layers与常用的recurrent layers和convolutional layers进行各方面的比较，比较的方式是 将一个可变长度的符号表示序列（x1，. . . ，xn）映射到另一个等长序列（z1，. . . ，zn），用 xi,zi∈Rd，比如在典型的序列转换的encoder或decoder中的隐藏层。我们考虑三个方面，最后促使我们使用self-attention。

一是每层的总计算复杂度。另一个是可以并行化的计算量，以所需的最小序列操作数衡量。

第三个是网络中长距离依赖关系之间的路径长度。在许多序列转换任务中，学习长距离依赖性是一个关键的挑战。影响学习这种依赖关系能力的一个关键因素是网络中向前和向后信号必须经过的路径的长度。输入和输出序列中任意位置组合之间的这些路径越短，越容易学习长距离依赖。因此，我们还比较了在由different layer types组成的网络 中的任意两个输入和输出位置之间的最大的路径长度

如表1所示，self-attention layer用常数次(O(1))的操作连接所有位置，而recurrent layer需要O(n)顺序操作。在计算复杂度方面，当序列长度N小于表示维度D时，self-attention layers比recurrent layers更快，这是使用最先进的机器翻译模型表示句子时的常见情况，例如word-piece [38] 和byte-pair [31] 表示。为了提高包含很长序列的任务的计算性能，可以仅在以输出位置为中心，半径为r的的领域内使用self-attention。这将使最大路径长度增长到O（n/r）。我们计划在今后的工作中进一步研究这种方法。

核宽度为k<n的单层卷积不会连接每一对输入和输出的位置。要这么做，在相邻的内核情况下，需要一个n个卷积层的堆栈， 在扩展卷积的情况下需要O(logk(n)) 层[18]，它们增加了网络中任意两个位置之间的最长路径的长度。 卷积层通常比循环层代价更昂贵，这与因子k有关。然而，可分卷积[6]大幅减少复杂度到O(k-n-d+n-d-2)。然而，即使k=n，可分离卷积的复杂度等于self-attention layer和point-wise feed-forward layer的组合，这是我们在模型中采用的方法。

一个随之而来的好处是，self-attention可以产生更多可解释的模型。我们从我们的模型中研究attention的分布，并在附录中展示和讨论示例。每个attention head不仅清楚地学习到执行不同的任务，还表现出了许多和句子的句法和语义结构相关的行为

length n is smaller than the representation dimensionality d, which is most often the case with sentence representations used by state-of-the-art models in machine translations, such as word-piece [38] and byte-pair [31] representations. To improve computational performance for tasks involving very long sequences, self-attention could be restricted to considering only a neighborhood of size r in the input sequence centered around the respective output position. This would increase the maximum path length to $O(n/r)$. We plan to investigate this approach further in future work.

A single convolutional layer with kernel width $k < n$ does not connect all pairs of input and output positions. Doing so requires a stack of $O(n/k)$ convolutional layers in the case of contiguous kernels, or $O(log_k(n))$ in the case of dilated convolutions [18], increasing the length of the longest paths between any two positions in the network. Convolutional layers are generally more expensive than recurrent layers, by a factor of k. Separable convolutions [6], however, decrease the complexity considerably, to $O(k \cdot n \cdot d + n \cdot d^2)$. Even with $k = n$, however, the complexity of a separable convolution is equal to the combination of a self-attention layer and a point-wise feed-forward layer, the approach we take in our model.

As side benefit, self-attention could yield more interpretable models. We inspect attention distributions from our models and present and discuss examples in the appendix. Not only do individual attention heads clearly learn to perform different tasks, many appear to exhibit behavior related to the syntactic and semantic structure of the sentences.

5 Training

This section describes the training regime for our models.

5.1 Training Data and Batching

We trained on the standard WMT 2014 English-German dataset consisting of about 4.5 million sentence pairs. Sentences were encoded using byte-pair encoding [3], which has a shared source-target vocabulary of about 37000 tokens. For English-French, we used the significantly larger WMT 2014 English-French dataset consisting of 36M sentences and split tokens into a 32000 word-piece vocabulary [38]. Sentence pairs were batched together by approximate sequence length. Each training batch contained a set of sentence pairs containing approximately 25000 source tokens and 25000 target tokens.

5.2 Hardware and Schedule

We trained our models on one machine with 8 NVIDIA P100 GPUs. For our base models using the hyperparameters described throughout the paper, each training step took about 0.4 seconds. We trained the base models for a total of 100,000 steps or 12 hours. For our big models,(described on the bottom line of table 3), step time was 1.0 seconds. The big models were trained for 300,000 steps (3.5 days).

5.3 Optimizer

We used the Adam optimizer [20] with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$. We varied the learning rate over the course of training, according to the formula:

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5}) \tag{3}$$

This corresponds to increasing the learning rate linearly for the first $warmup\_steps$ training steps, and decreasing it thereafter proportionally to the inverse square root of the step number. We used $warmup\_steps = 4000$.

5.4 Regularization

We employ three types of regularization during training:

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Residual Dropout We apply dropout [33] to the output of each sub-layer, before it is added to the sub-layer input and normalized. In addition, we apply dropout to the sums of the embeddings and the positional encodings in both the encoder and decoder stacks. For the base model, we use a rate of $P_{drop} = 0.1$.

Label Smoothing During training, we employed label smoothing of value $\epsilon_{ls} = 0.1$ [36]. This hurts perplexity, as the model learns to be more unsure, but improves accuracy and BLEU score.

6 Results

6.1 Machine Translation

On the WMT 2014 English-to-German translation task, the big transformer model (Transformer (big) in Table 2) outperforms the best previously reported models (including ensembles) by more than 2.0 BLEU, establishing a new state-of-the-art BLEU score of 28.4. The configuration of this model is listed in the bottom line of Table 3. Training took 3.5 days on 8 P100 GPUs. Even our base model surpasses all previously published models and ensembles, at a fraction of the training cost of any of the competitive models.

On the WMT 2014 English-to-French translation task, our big model achieves a BLEU score of 41.0, outperforming all of the previously published single models, at less than $1/4$ the training cost of the previous state-of-the-art model. The Transformer (big) model trained for English-to-French used dropout rate $P_{drop} = 0.1$, instead of 0.3.

For the base models, we used a single model obtained by averaging the last 5 checkpoints, which were written at 10-minute intervals. For the big models, we averaged the last 20 checkpoints. We used beam search with a beam size of 4 and length penalty $\alpha = 0.6$ [38]. These hyperparameters were chosen after experimentation on the development set. We set the maximum output length during inference to input length + 50, but terminate early when possible [38].

Table 2 summarizes our results and compares our translation quality and training costs to other model architectures from the literature. We estimate the number of floating point operations used to train a model by multiplying the training time, the number of GPUs used, and an estimate of the sustained single-precision floating-point capacity of each GPU [5].

6.2 Model Variations

To evaluate the importance of different components of the Transformer, we varied our base model in different ways, measuring the change in performance on English-to-German translation on the

[5]We used values of 2.8, 3.7, 6.0 and 9.5 TFLOPS for K80, K40, M40 and P100, respectively.

Table 3: Variations on the Transformer architecture. Unlisted values are identical to those of the base model. All metrics are on the English-to-German translation development set, newstest2013. Listed perplexities are per-wordpiece, according to our byte-pair encoding, and should not be compared to per-word perplexities.

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

development set, newstest2013. We used beam search as described in the previous section, but no checkpoint averaging. We present these results in Table 3.

In Table 3 rows (A), we vary the number of attention heads and the attention key and value dimensions, keeping the amount of computation constant, as described in Section 3.2.2. While single-head attention is 0.9 BLEU worse than the best setting, quality also drops off with too many heads.

In Table 3 rows (B), we observe that reducing the attention key size d_k hurts model quality. This suggests that determining compatibility is not easy and that a more sophisticated compatibility function than dot product may be beneficial. We further observe in rows (C) and (D) that, as expected, bigger models are better, and dropout is very helpful in avoiding over-fitting. In row (E) we replace our sinusoidal positional encoding with learned positional embeddings [9], and observe nearly identical results to the base model.

6.3 English Constituency Parsing

To evaluate if the Transformer can generalize to other tasks we performed experiments on English constituency parsing. This task presents specific challenges: the output is subject to strong structural constraints and is significantly longer than the input. Furthermore, RNN sequence-to-sequence models have not been able to attain state-of-the-art results in small-data regimes [37].

We trained a 4-layer transformer with $d_{model} = 1024$ on the Wall Street Journal (WSJ) portion of the Penn Treebank [25], about 40K training sentences. We also trained it in a semi-supervised setting, using the larger high-confidence and BerkleyParser corpora from with approximately 17M sentences [37]. We used a vocabulary of 16K tokens for the WSJ only setting and a vocabulary of 32K tokens for the semi-supervised setting.

We performed only a small number of experiments to select the dropout, both attention and residual (section 5.4), learning rates and beam size on the Section 22 development set, all other parameters remained unchanged from the English-to-German base translation model. During inference, we

Table 4: The Transformer generalizes well to English constituency parsing (Results are on Section 23 of WSJ)

Parser	Training	WSJ 23 F1
Vinyals & Kaiser el al. (2014) [37]	WSJ only, discriminative	88.3
Petrov et al. (2006) [29]	WSJ only, discriminative	90.4
Zhu et al. (2013) [40]	WSJ only, discriminative	90.4
Dyer et al. (2016) [8]	WSJ only, discriminative	91.7
Transformer (4 layers)	WSJ only, discriminative	91.3
Zhu et al. (2013) [40]	semi-supervised	91.3
Huang & Harper (2009) [14]	semi-supervised	91.3
McClosky et al. (2006) [26]	semi-supervised	92.1
Vinyals & Kaiser el al. (2014) [37]	semi-supervised	92.1
Transformer (4 layers)	semi-supervised	92.7
Luong et al. (2015) [23]	multi-task	93.0
Dyer et al. (2016) [8]	generative	93.3

increased the maximum output length to input length + 300. We used a beam size of 21 and $\alpha = 0.3$ for both WSJ only and the semi-supervised setting.

Our results in Table 4 show that despite the lack of task-specific tuning our model performs surprisingly well, yielding better results than all previously reported models with the exception of the Recurrent Neural Network Grammar [8].

In contrast to RNN sequence-to-sequence models [37], the Transformer outperforms the Berkeley-Parser [29] even when training only on the WSJ training set of 40K sentences.

7 Conclusion

In this work, we presented the Transformer, the first sequence transduction model based entirely on attention, replacing the recurrent layers most commonly used in encoder-decoder architectures with multi-headed self-attention.

在这项工作中，我们提出了Transformer，第一个完全基于attention的序列转换模型，用multi-headed self-attention取代了encoder-decoder架构中最常用的recurrent layers。

For translation tasks, the Transformer can be trained significantly faster than architectures based on recurrent or convolutional layers. On both WMT 2014 English-to-German and WMT 2014 English-to-French translation tasks, we achieve a new state of the art. In the former task our best model outperforms even all previously reported ensembles.

对于翻译任务，Transformer比基于循环或卷积层的体系结构训练更快。 在WMT 2014英语-德语和WMT 2014英语-法语翻译任务中，我们取得了最好的结果。 在前面的任务中，我们最好的模型甚至胜过以前报道过的所有整合模型。

We are excited about the future of attention-based models and plan to apply them to other tasks. We plan to extend the Transformer to problems involving input and output modalities other than text and to investigate local, restricted attention mechanisms to efficiently handle large inputs and outputs such as images, audio and video. Making generation less sequential is another research goals of ours.

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，并研究局部的、受限的attention机制，以有效地处理图像、音频和视频等大型输入和输出。 让生成具有更少的顺序性是我们的另一个研究目标。

The code we used to train and evaluate our models is available at `https://github.com/tensorflow/tensor2tensor`.

Acknowledgements We are grateful to Nal Kalchbrenner and Stephan Gouws for their fruitful comments, corrections and inspiration.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[3] Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc V. Le. Massive exploration of neural machine translation architectures. *CoRR*, abs/1703.03906, 2017.

[4] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733*, 2016.

[5] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[6] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357*, 2016.

[7] Junyoung Chung, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[8] Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. Recurrent neural network grammars. In *Proc. of NAACL*, 2016.

[9] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122v2*, 2017.

[10] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[12] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[14] Zhongqiang Huang and Mary Harper. Self-training PCFG grammars with latent annotations across languages. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 832–841. ACL, August 2009.

[15] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

[16] Łukasz Kaiser and Samy Bengio. Can active memory replace attention? In *Advances in Neural Information Processing Systems, (NIPS)*, 2016.

[17] Łukasz Kaiser and Ilya Sutskever. Neural GPUs learn algorithms. In *International Conference on Learning Representations (ICLR)*, 2016.

[18] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099v2*, 2017.

[19] Yoon Kim, Carl Denton, Luong Hoang, and Alexander M. Rush. Structured attention networks. In *International Conference on Learning Representations*, 2017.

[20] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[21] Oleksii Kuchaiev and Boris Ginsburg. Factorization tricks for LSTM networks. *arXiv preprint arXiv:1703.10722*, 2017.

[22] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*, 2017.

[23] Minh-Thang Luong, Quoc V. Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114*, 2015.

[24] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[25] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[26] David McClosky, Eugene Charniak, and Mark Johnson. Effective self-training for parsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 152–159. ACL, June 2006.

[27] Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model. In *Empirical Methods in Natural Language Processing*, 2016.

[28] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*, 2017.

[29] Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the ACL*, pages 433–440. ACL, July 2006.

[30] Ofir Press and Lior Wolf. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*, 2016.

[31] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[32] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[33] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[34] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2440–2448. Curran Associates, Inc., 2015.

[35] Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.

[36] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[37] Vinyals & Kaiser, Koo, Petrov, Sutskever, and Hinton. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, 2015.

[38] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[39] Jie Zhou, Ying Cao, Xuguang Wang, Peng Li, and Wei Xu. Deep recurrent models with fast-forward connections for neural machine translation. *CoRR*, abs/1606.04199, 2016.

[40] Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. Fast and accurate shift-reduce constituent parsing. In *Proceedings of the 51st Annual Meeting of the ACL (Volume 1: Long Papers)*, pages 434–443. ACL, August 2013.

Attention Visualizations

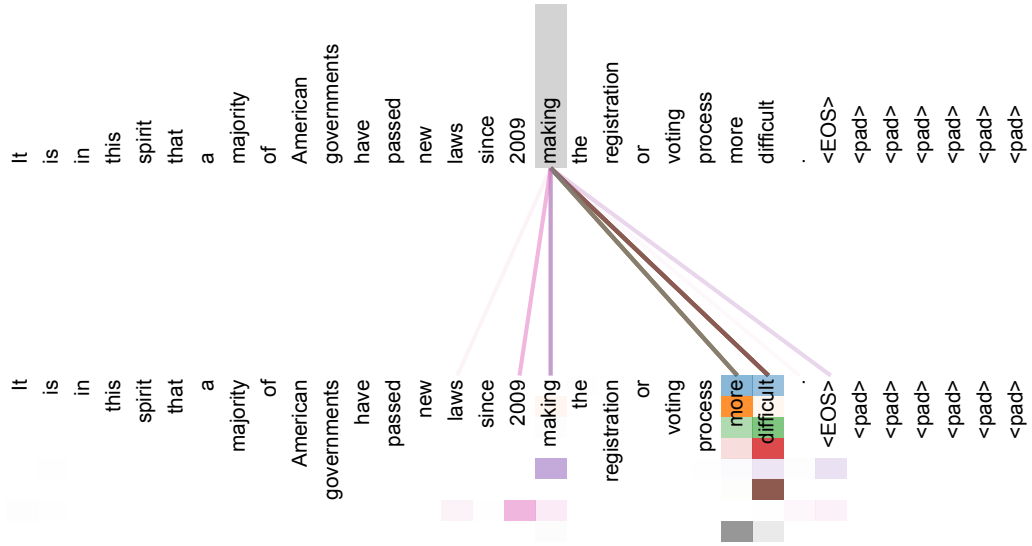


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb 'making', completing the phrase 'making...more difficult'. Attentions here shown only for the word 'making'. Different colors represent different heads. Best viewed in color.

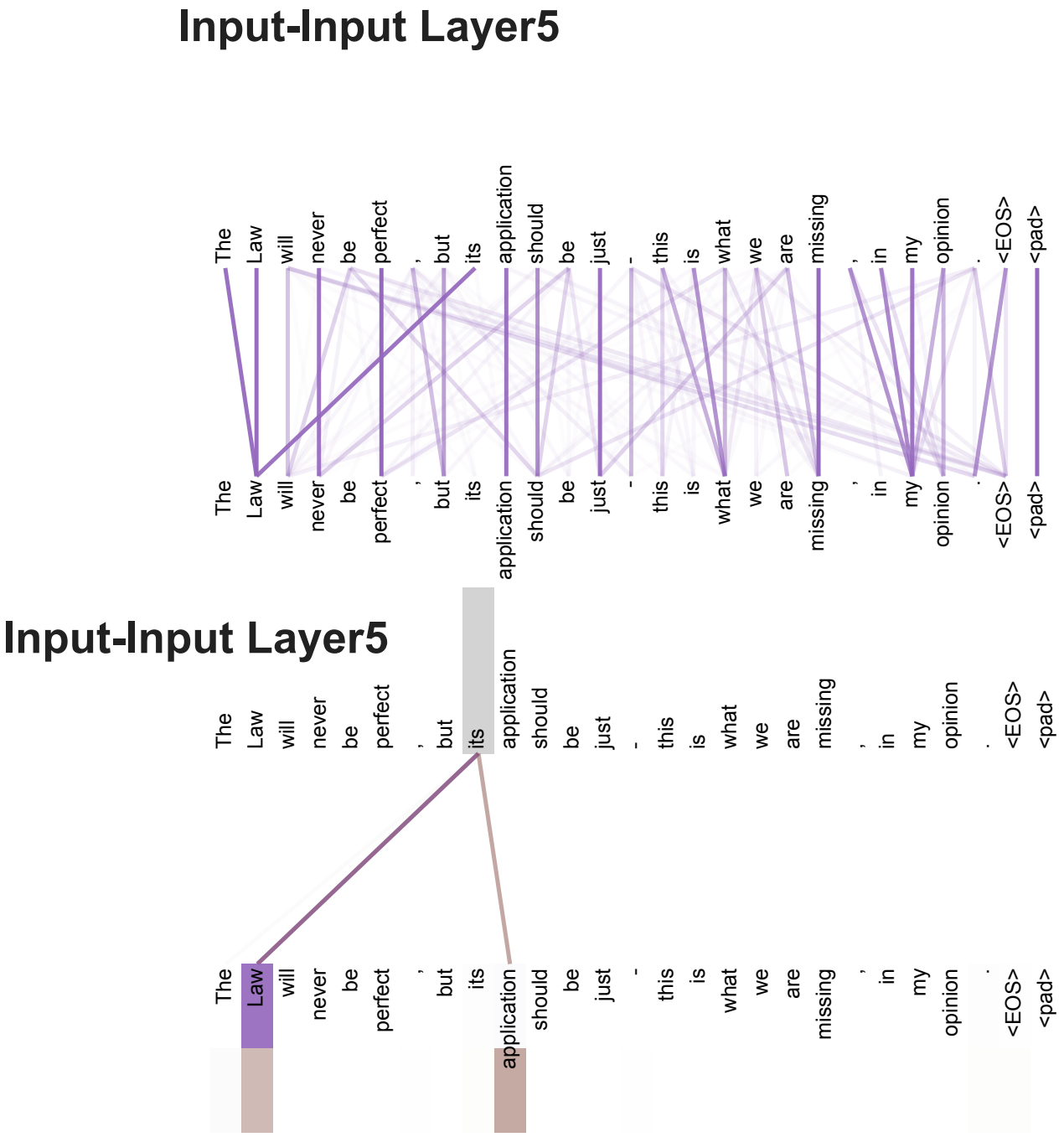


Figure 4: Two attention heads, also in layer 5 of 6, apparently involved in anaphora resolution. Top: Full attentions for head 5. Bottom: Isolated attentions from just the word 'its' for attention heads 5 and 6. Note that the attentions are very sharp for this word.

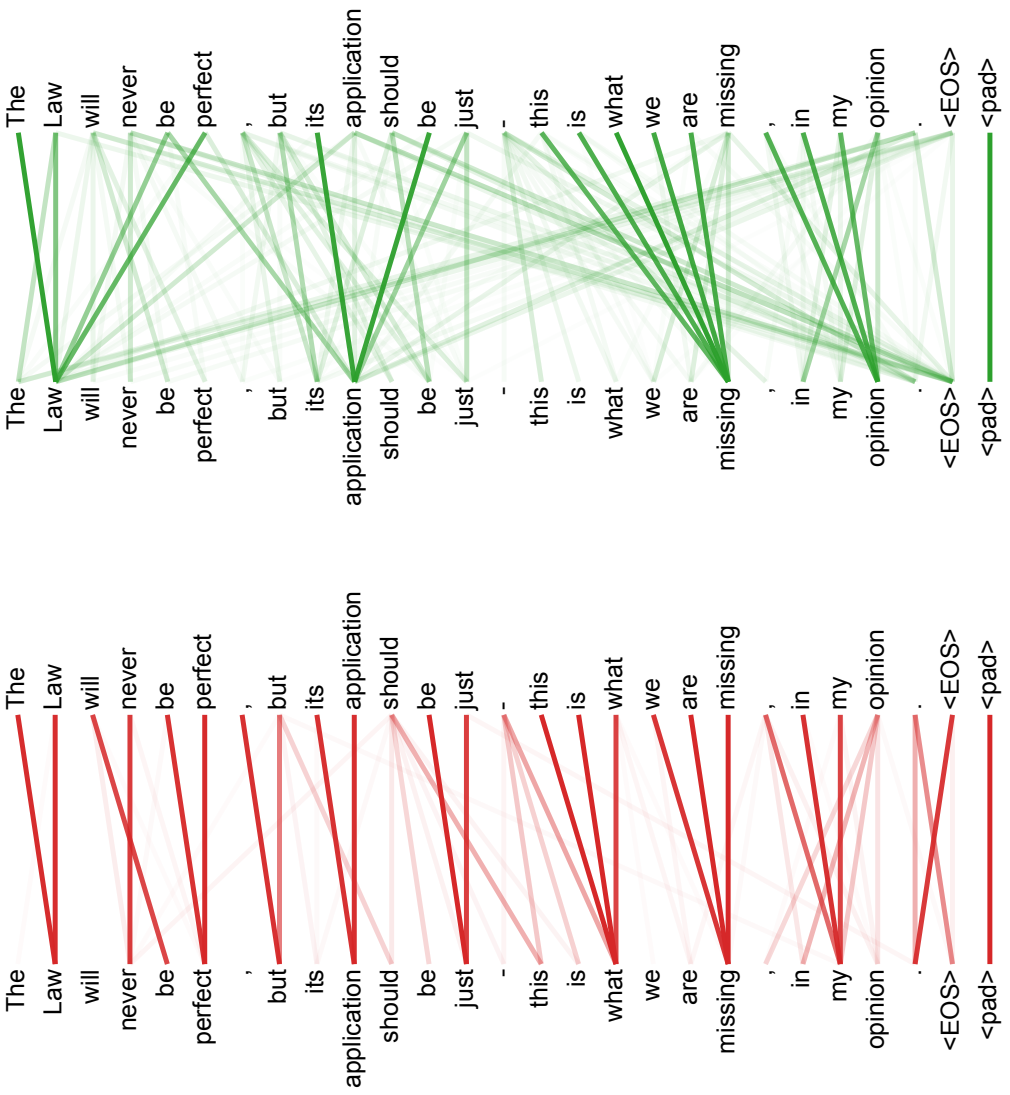


Figure 5: Many of the attention heads exhibit behaviour that seems related to the structure of the sentence. We give two such examples above, from two different heads from the encoder self-attention at layer 5 of 6. The heads clearly learned to perform different tasks.