NeurIPS 2017 这篇文章提出了Transformer模型，抛弃RNN，只用Attention。

# 注意力是你所需要的一切

**Ashish Vaswani*** 谷歌大脑 avaswani@google.com

**Noam Shazeer*** 谷歌大脑 noam@google.com

**Niki Parmar*** 谷歌研究nikip@google.com

**Jakob Uszkoreit*** 谷歌研究 usz@google.com

**Llion Jones*** 谷歌研究llion@google.com

**Aidan N. Gomez*** [†] 多伦多大学 aidan@cs.toronto.edu

**Łukasz Kaiser*** 谷歌大脑 lukaszkaiser@google.com

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

## 摘要

占主导地位的序列转导模型是基于复杂的递归或卷积神经网络，包括一个编码器和一个解码器。性能最好的模型还通过注意机制将编码器和解码器连接起来。我们提出了一个新的简单的网络结构--Transformer，它只基于注意力机制，完全不需要递归和卷积。在两个机器翻译任务上的实验表明，这些模型在质量上更胜一筹，同时也更容易并行化，需要的训练时间也大大减少。我们的模型在WMT 2014英德翻译任务中取得了28.4 BLEU的成绩，比现有的最佳结果（包括集合）提高了2 BLEU以上。在WMT 2014英法翻译任务中，我们的模型在8个GPU上训练了3.5天后，建立了新的单模型最先进的BLEU得分，即41.0分，这只是文献中最佳模型训练成本的一小部分。

2017年谷歌在一篇名为《Attention Is All You Need》的论文中,提出了一个基于attention(自注意力机制)结构来处理序列相关的问题的模型，名为Transformer。

Transformer在很多不同nlp任务中获得了成功，例如：文本分类、机器翻译、阅读理解等。在解决这类问题时，Transformer模型摒弃了固有的定式，并没有用任何CNN或者RNN的结构，而是使用了Attention注意力机制，自动捕捉输入序列不同位置处的相对关联，善于处理较长文本，并且该模型可以高度并行地工作，训练速度很快。

## 1 简介

递归神经网络，特别是长短时记忆[12]和门控递归[7]神经网络，已经被牢固地确立为序列建模和转换问题的最先进方法，如语言建模和机器翻译[29, 2, 5]。此后，众多的努力不断推动着递归语言模型和编码器-解码器架构的发展[31, 21, 13]。

*贡献相等。列表顺序是随机的。Jakob提议用自我关注取代RNN，并开始努力评估这一想法。Ashish和Illia一起设计并实现了第一个Transformer模型和在这项工作的每一个方面都有关键性的参与。Noam提出了缩放点积注意力、多头注意力和无参数位置表示，并成为参与几乎所有细节的另一个人。Niki在我们的原始代码库和tensor2tensor中设计、实现、调整和评估了无数的模型变体。Llion也尝试了新的模型变体，负责我们的初始代码库，以及高效的推理和可视化。Lukasz和Aidan花了无数个漫长的日子来设计和实现tensor2tensor的各个部分，取代了我们早期的代码库，大大改善了结果，并大规模地加速了我们的研究。

[†]在谷歌大脑时进行的工作。

[‡]在谷歌研究院工作时进行的工作。

第31届神经信息处理系统会议（NIPS 2017），美国加州长滩。

递归模型通常沿着输入和输出序列的符号位置进行计算。将位置与计算时间的步骤相一致，它们产生一个隐藏状态的序列$h_t$，作为前一个隐藏状态$h_{t-1}$和位置$t$的输入的函数。这种固有的顺序性排除了训练实例内的并行化，这在较长的序列长度上变得至关重要，因为内存限制了跨实例的批处理。最近的工作通过因式分解技巧[18]和条件计算[26]在计算效率方面取得了重大改进，同时在后者的情况下也提高了模型性能。然而，顺序计算的基本约束仍然存在。

注意机制已经成为各种任务中引人注目的序列建模和转换模型的一个组成部分，允许对依赖关系进行建模而不考虑它们在输入或输出序列中的距离[2, 16]。然而，除了少数情况外[22]，这种注意机制是与递归网络一起使用的。

在这项工作中，我们提出了Transformer，这是一个避免递归的模型架构，而是完全依靠注意力机制来得出输入和输出之间的全局依赖关系。Transformer允许显著提高并行化程度，并且在8个P100 GPU上经过短短12个小时的训练，就可以达到翻译质量的新水平。

## 2 背景介绍

减少顺序计算的目标也构成了扩展神经GPU[20]、ByteNet[15]和ConvS2S[8]的基础，它们都使用卷积神经网络作为基本构建模块，对所有输入和输出位置并行计算隐藏的表征。在这些模型中，将来自两个任意输入或输出位置的信号联系起来所需的操作数量随着位置之间的距离增长，对于ConvS2S是线性增长，对于ByteNet是对数增长。这使得学习遥远位置之间的依赖关系更加困难[11]。在Transformer中，这被减少到一个恒定的操作数，尽管代价是由于注意力加权位置的平均化而降低了有效的分辨率，如第3.2节所述，我们用多头注意力抵消了这种影响。

自我注意，有时也被称为内部注意，是一种与单个序列的不同位置相关的注意机制，以计算该序列的表示。自我注意已被成功地用于各种任务，包括阅读理解、抽象概括、文本连带和学习与任务无关的句子表示[4, 22, 23, 19]。

端到端记忆网络是基于递归注意机制，而不是序列对齐的递归，并已被证明在简单语言问题回答和语言建模任务中表现良好[28]。

然而，据我们所知，Transformer是第一个完全依靠自我注意来计算其输入和输出的表征而不使用序列对齐的RNN或卷积的转导模型。在下面的章节中，我们将描述Transformer，激励自我注意，并讨论它与[14, 15]和[8]等模型相比的优势。

## 3 模型结构

大多数有竞争力的神经序列转导模型有一个编码器-解码器结构[5, 2, 29]。这里，编码器将输入的符号表征序列（$x_1, ..., x_n$）映射为连续表征序列$\mathbf{z} = (z_1, ..., z_n)$。给定$\mathbf{z}$后，解码器每次生成一个元素的符号输出序列（$y_1, ..., y_m$）。在每一步，该模型是自动回归的[9]，在生成下一步时，消耗先前生成的符号作为额外输入。

Transformer遵循这一整体架构，在编码器和解码器中都使用了堆叠的自注意和点状的全连接层，分别在图1的左半部和右半部中显示。

### 3.1 编码器和解码器堆栈

**编码器。** 编码器是由$N$=6个相同的层堆叠而成。每层有两个子层。第一层是一个多头的自我关注机制，第二层是一个简单的，位置的

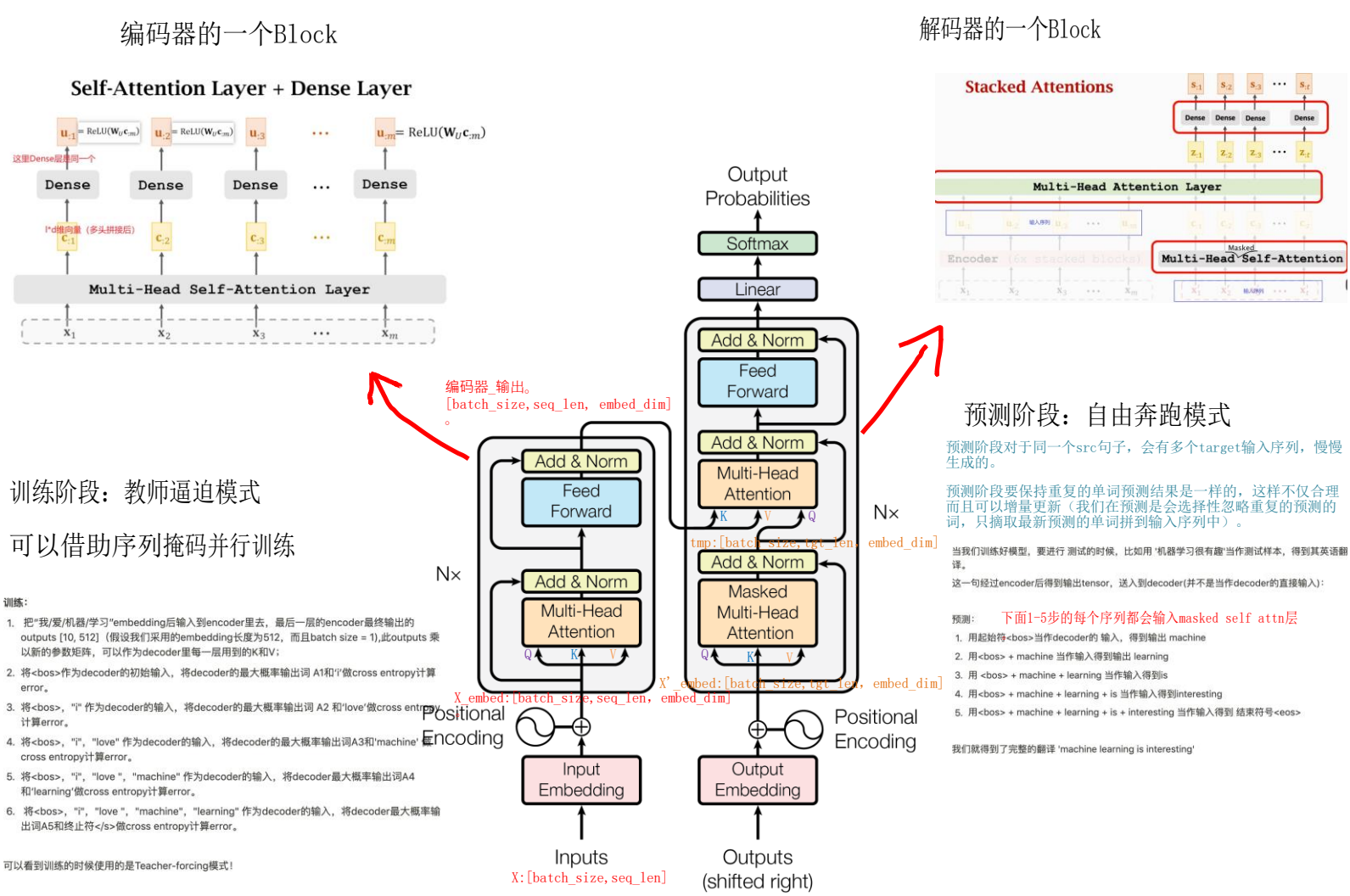



图1：变压器--模型结构。

聪明的全连接前馈网络。我们在两个子层的每一个周围都采用了一个残差连接[10]，然后进行层规范化[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。为了方便这些剩余连接，模型中的所有子层以及嵌入层都会产生维数为$d_{\text{model}}$ = 512 的 输出。

```
residual = inputs
output = self.fc(inputs)
return nn.LayerNorm(d_model).cuda()(output + residual)
```

**解码器。**解码器也是由$N$=6 个相同层的堆栈组成。除了每个编码器层的两个子层之外，解码器还插入了第三个子层，它对编码器堆栈的输出进行多头关注。与编码器类似，我们在每个子层周围采用剩余连接，然后进行层的归一化。我们还修改了自我注意
解码器堆栈中的子层，以防止位置出现在后续位置上。这种屏蔽，再加上输出嵌入偏移一个位置的事实，确保对位置$i$的预测只取决于小于$i$的位置的已知输出。

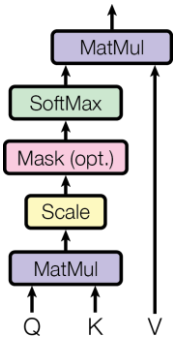
### 3.2 注意

注意力函数可以被描述为将一个查询和一组键值对映射到一个输出，其中查询、键、值和输出都是向量。输出被计算为数值的加权和，其中分配给每个数值的权重是由查询与相应的键的兼容性函数计算的。

#### 3.2.1 缩放点积注意

我们称我们的特别关注为 "缩放点积关注"（图2）。输入包括查询和维度为$d_k$的键，以及维度为$d_v$ 的值。我们计算出


缩放点积注意




多头注意力机制代码实现：
https://paste.ubuntu.com/p/ymmMTqNsCC/

多头关注

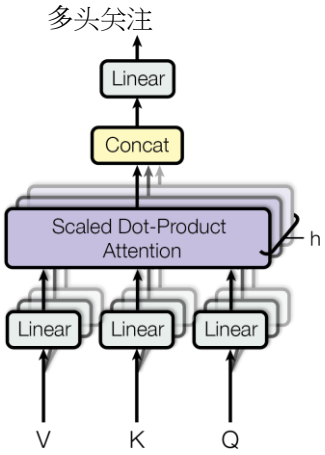



图2：（左）按比例的点乘法注意力。(右)多头注意由几个平行运行的注意层组成。

用所有的键进行查询，将每个键除以$\sqrt{d_k}$，并应用softmax函数来获得值的权重。

在实践中，我们同时计算一组查询的注意函数，并将其打包成矩阵$Q$，键和值也被打包成矩阵$K$和$V$。我们计算输出的矩阵为。

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

两种最常用的注意力函数是加法注意力[2]，以及点积（多参数）注意力。点积式注意力与我们的算法相同，除了$\frac{1}{\sqrt{d_k}}$的比例系数。加法注意力使用一个具有以下特征的前馈网络来计算相容性函数
一个单一的隐藏层。虽然两者在理论上的复杂度相似，但点积关注在实践中要快得多，而且空间效率更高，因为它可以用高度优化的矩阵乘法代码来实现。

虽然对于小的$d_k$值，这两种机制的表现相似，但对于较大的$d_k$值，加性注意优于点积注意，而没有缩放功能[3]。我们怀疑，对于大的$d_k$值，点积的大小会越来越大，把softmax函数推到它的梯度极小的区域[4]。为了抵消这种影响，我们把点乘的比例定为$\frac{1}{\sqrt{d_k}}$。

### 3.2.2 多头关注

我们发现，与其用$d_{\text{model}}$维的键、值和查询来执行一个单一的注意函数，不如用不同的、学习过的线性投影将查询、键和值分别投影到$d_k$、$d_k$和$d_v$维，这样做是有益的。在这些预测的查询、键和值的每个版本上，我们再平行地执行注意函数，产生$d_v$-dimensional输出值。这些值被串联起来，并再次进行投影，从而得到最终的数值，如图2所示。

多头注意允许模型在不同的位置上共同关注来自不同表征子空间的信息。在单头注意的情况下，平均化抑制了这一点。

[4]为了说明为什么点积会变大，假设$q$和$k$的成分是独立的随机的

变量，均值为0，方差为1。那么它们的点积，$q \cdot k = \sum_{i=1}^{d_k} q_i k_i$，均值为0，方差为$d_k$。

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h) W^O$$
$$\text{其中头部}_i = \text{注意}(QW_i^Q, KW_i^K, VW_i^V)$$

其中，投影是参数矩阵$W_i^Q \in \mathbb{R}^{d_{\text{模型}} \times dk}$，$W_i^K \in \mathbb{R}^{d_{\text{模型}} \times dk}$，$W_i^V \in \mathbb{R}^{d_{\text{模型}} \times dv}$和$W^O \in Rhdv \times dmodel$。

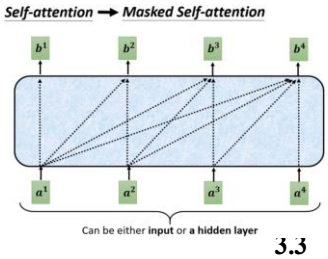
在这项工作中，我们采用了$h$=8个平行注意层，或称头。对于每个头，我们使用$d_k = d_v = d_{\text{model}}/h = 64$。由于每个头的维度减少，总的计算成本与全维度的单头注意相似。

### 3.2.3 注意力在我们模型中的应用

变形金刚以三种不同方式使用多头关注。

- 在"编码器-解码器关注"层中，查询来自前一个解码器层，而记忆键和值来自编码器的输出。这使得解码器中的每个位置都能关注到输入序列中的所有位置。这模拟了序列到序列模型中典型的编码器-解码器注意力机制，如[31, 2, 8]。
- 编码器包含自我关注层。在自我关注层中，所有的键、值和查询都来自同一个地方，在这种情况下，就是编码器中前一层的输出。编码器中的每个位置都可以关注到编码器前一层的所有位置。
- 同样，解码器中的自我关注层允许解码器中的每个位置关注解码器中的所有位置，直到并包括该位置。我们需要防止解码器中的信息向左流动，以保持自动回归的特性。我们通过屏蔽掉（设置为）softmax输入中对应于非法连接的所有值，在缩放点乘法注意力中实现这一点。见图2。 $-\infty$

蒙蔽的自我关注的神经元连线是这样的，可以通过序列蒙蔽矩阵来实现




解码器需要用Maksed Self Attention

解码器必须用屏蔽的，否则只用自己注意的话语，你训练时的损失=f(y1)....yt)，里面的yi是会有右侧的信息（你想预测的词也在里面），这样训练时作弊的。

### 3.3 基于位置的前馈网络

除了注意子层之外，我们的编码器和解码器中的每一层都包含一个全连接的前馈网络，该网络分别适用于每个位置，并且完全相同。这包括两个线性变换，中间有一个ReLU激活。

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

虽然线性变换在不同的位置上是相同的，但它们在不同的层上使用不同的参数。另一种描述方式是将其作为两个内核大小为1的卷积。输入和输出的维度是$d_{\text{model}}$ = 512，内层的维度是$d_{ff}$ = 2048。

### 3.4 嵌入和Softmax

与其他序列转换模型类似，我们使用学习的嵌入将输入标记和输出标记转换为维度为$d$的向量$_{\text{model}}$。我们还使用通常学习的线性转换和softmax函数将解码器输出转换为预测的下一个标记概率。在
在我们的模型中，我们在两个嵌入层和pre-√softmax之间共享相同的权重矩阵。
线性变换，类似于[24]。在嵌入层中，我们把这些权重乘以 $dmodel$。

### 3.5 位置编码

由于我们的模型不包含递归和卷积，为了使模型能够利用序列的顺序，我们必须注入一些关于序列中标记的相对或绝对位置的信息。为此，我们将 "位置编码 "添加到输入嵌入中的

表1：不同层类型的最大路径长度、每层复杂度和最小序列操作数。$n$是序列长度，$d$是表示维度，$k$是卷积的内核大小，$r$是限制性自我关注的邻域大小。

| 层类型每层复杂度 | 顺序的最大路径长度 | 业务 | |
|---|---|---|---|
| 自我关注 | $O(n^2 \cdot d)$ | $O(1)$ | $O(1)$ |
| 循环性 | $O(n \cdot d^2)$ | $O(n)$ | $O(n)$ |
| 卷积法 | $O(k \cdot n \cdot d^2)$ | $O(1)$ | $O(\log_k(n))$ |
| 自我关注(限制性) | $O(r \cdot n \cdot d)$ | $O(1)$ | $O(n/r)$ |

编码器和解码器堆栈的底部。位置编码与嵌入具有相同的维度$d_{\text{model}}$，因此两者可以相加。位置编码有很多选择，有学习的和固定的[8]。

在这项工作中，我们使用不同频率的正弦和余弦函数。

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

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波。波长形成一个从$2\pi$到$10000 \cdot 2\pi$的几何级数。我们选择这个函数是因为我们假设它能让模型很容易地学会通过相对位置来关注，因为对于任何固定的偏移量$k$，$PE_{pos+k}$可以表示为$PE_{pos}$的线性函数。

Learned Positional Embedding，这个是绝对位置编码，即直接对不同的位置随机初始化一个postion embedding，这个postion embedding作为参数进行训练。

我们还试验了用学习的位置嵌入[8]来代替，发现这两个版本产生了几乎相同的结果（见表3行（E））。我们选择正弦波版本是因为它可能允许模型推断出比训练期间遇到的序列长度更长的序列。

## 4 为什么自我关注

在本节中，我们将自我注意层的各个方面与通常用于将一个可变长度的符号表示序列（$x_1, ..., x_n$）映射到另一个等长的序列（$z_1, ..., z_n$）的递归和卷积层进行比较，$x_i, z_i \in \mathbb{R}^d$，如典型序列转导编码器或解码器的隐藏层。我们使用自我注意的动机是考虑三个要求。

一个是每层的总计算复杂性。另一个是可以并行化的计算量，以所需的最小顺序操作数来衡量。

第三是网络中长距离依赖关系之间的路径长度。学习长距离的依赖关系是许多序列转导任务中的一个关键挑战。影响学习这种依赖关系能力的一个关键因素是前向和后向信号在网络中必须穿越的路径的长度。在输入和输出序列的任何位置组合之间的这些路径越短，就越容易学习长距离的依赖关系[11]。因此，我们也比较了由不同层类型组成的网络中任何两个输入和输出位置之间的最大路径长度。

如表1所示，自留层以恒定数量的顺序执行操作连接所有位置，而递归层需要$O(n)$顺序操作。就计算复杂度而言，当序列长度$n$小于表示维数$d$时，自注意层比递归层快，而这是最常见的情况。
机器翻译中最先进的模型所使用的句子表征，如单词-片断
[31]和字节对[25]表示。为了提高涉及很长序列的任务的计算性能，自我注意可以被限制在只考虑一个大小为$r$的邻域，在

以各自的输出位置为中心的输入序列。这将使最大路径长度增加到$O(n/r)$。我们计划在未来的工作中进一步研究这种方法。

一个内核宽度为$k<n$的单一卷积层并不能连接所有的输入和输出位置对。在连续核的情况下，这样做需要堆叠$O(n/k)$个卷积层，在扩张卷积的情况下，需要堆叠$O(log_k(n))$，增加网络中任何两个位置之间最长路径的长度。卷积层通常比递归层更昂贵，是$k$的倍数。然而，可分离卷积[6]大大降低了复杂性，为$O(k \cdot n \cdot d + n \cdot d^2)$。然而，即使在$k=n$的情况下，可分离卷积的复杂度也等于自我注意层和点状前馈层的组合，也就是我们模型中采用的方法。

作为副作用，自我注意可以产生更多可解释的模型。我们从我们的模型中检查了注意力的分布，并在附录中提出和讨论了一些例子。不仅个别注意力头明显学会了执行不同的任务，许多人似乎表现出与句子的句法和语义结构有关的行为。

## 5 培训

本节描述了我们模型的训练制度。

### 5.1 训练数据和批处理

我们在由大约450万个句子对组成的标准WMT 2014英德数据集上进行训练。句子使用字节对编码[3]，它有一个共享的源-目标词汇，约37000个标记。对于英语-法语，我们使用了规模大得多的WMT 2014英语-法语数据集，包括3600万个句子，并将代币分成32000个词件词汇[31]。句子对按照大致的序列长度被分到一起。每个训练批次包含一组句子对，其中包含大约25000个源标记和25000个目标标记。

### 5.2 硬件和时间表

我们在一台有8个NVIDIA P100 GPU的机器上训练我们的模型。对于我们的基础模型，使用本文所述的超参数，每个训练步骤大约需要0.4秒。我们总共训练了100,000步或12小时的基础模型。对于我们的大模型，（在表3的最下面一行描述），步骤时间为1.0秒。大模型被训练了30万步（3.5天）。

### 5.3 优化器

我们使用亚当优化器[17]，$\beta_1 = 0.9$，$\beta_2 = 0.98$，$E = 10^{-9}$。在训练过程中，我们改变了学习率，根据公式。

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

这相当于在第一个$warmup_steps$训练步骤中线性增加学习率，此后按步骤数的反平方根比例减少。我们使用$warmup_steps = 4000$。

### 5.4 正规化

在训练过程中，我们采用了三种类型的正则化。

**残余剔除**
我们对每个子层的输出进行剔除[27]，然后再将其添加到子层的输入并进行归一化处理。此外，我们对编码器和解码器堆栈中的嵌入和位置编码的总和进行剔除。对于基础模型，我们使用$P_{drop} = 0.1$的比率。

表2：在英译德和英译法的newstest2014测试中，Transformer取得了比以前最先进的模型更好的BLEU分数，而训练成本只有一小部分。

| 模型 | BLEUT | | 培训成本（FLOPs） | |
|---|---|---|---|---|
| | EN-DE | EN-FR | EN-DE | EN-FR |
| ByteNet [15] | 23.75 | | | |
| Deep-Att + PosUnk[32] | | 39.2 | | $1.0 \cdot 10^{20}$ |
| GNMT + RL [31] | 24.6 | 39.92 | $2.3 \cdot 10^{19}$ | $1.4 \cdot 10^{20}$ |
| ConvS2S [8] | 25.16 | 40.46 | $9.6 \cdot 10^{18}$ | $1.5 \cdot 10^{20}$ |
| 教育部[26] | 26.03 | 40.56 | $2.0 \cdot 10^{19}$ | $1.2 \cdot 10^{20}$ |
| Deep-Att + PosUnk Ensemble [32] | | 40.4 | | $8.0 \cdot 10^{20}$ |
| GNMT + RL合集[31] | 26.30 | 41.16 | $1.8 \cdot 10^{20}$ | $1.1 \cdot 10^{21}$ |
| ConvS2S合集[8] | 26.36 | **41.29** | $7.7 \cdot 10^{19}$ | $1.2 \cdot 10^{21}$ |
| 变压器（基本型号） | 27.3 | 38.1 | **$3.3 \cdot 10^{18}$** | |
| 变压器（大） | **28.4** | **41.0** | $2.3 \cdot 10^{19}$ | |

**标签平滑** 在训练期间，我们采用了标签平滑值$\epsilon_{ls} = 0.1$ [30]。这损害了迷惑性，因为模型学会了更多的不确定因素，但提高了准确性和BLEU得分。

## 6 结果

### 6.1 机器翻译

在WMT 2014的英德翻译任务中，大转化器模型（表2中的转化器（big））比之前报道的最佳模型（包括合集）高出2.0 BLEU以上，建立了一个新的最先进的BLEU分数，即28.4。这个模型的配置列在表3的最下面一行。训练在8 个 P100 GPU上花了3.5天。即使是我们的基本模型，也超过了以前发表的所有模型和组合，而训练成本只是任何竞争模型的一小部分。

在WMT 2014的英法翻译任务中，我们的大模型取得了41.0的BLEU分数，超过了之前公布的所有单一模型，而训练成本还不到之前最先进模型的1/4。为英译法训练的Transformer（大）模型使用了辍学率$P_{drop} = 0.1$，而不是0.3。

对于基本模型，我们使用了通过平均最后5个检查点得到的单一模型，这些检查点是以10分钟的时间间隔写入的。对于大模型，我们对最后的20个检查点进行了平均。我们使用波束搜索，波束大小为4，长度惩罚$\alpha$=0.6[31]。这些超参数是在开发集上进行实验后选择的。我们将推理过程中的最大输出长度设置为输入长度+50，但尽可能提前终止[31]。

表2总结了我们的结果，并将我们的翻译质量和训练成本与文献中的其他模型架构进行了比较。我们通过将训练时间、使用的GPU数量和每个GPU的持续单精度浮点能力的估计值相乘来估计训练一个模型所使用的浮点运算的数量[5]。

### 6.2 模型变化

为了评估Transformer不同组件的重要性，我们以不同的方式改变了我们的基础模型，在开发集newstest2013上测量英德翻译的性能变化。我们使用了上一节所述的波束搜索，但没有使用检查点平均法。我们在表3中展示了这些结果。

在表3行（A）中，我们改变了注意头的数量以及注意键和值的维度，保持计算量不变，如3.2.2节所述。虽然单头关注比最佳设置差0.9 BLEU，但随着头数的增加，质量也会下降。

[5]我们对K80、K40、M40和P100的数值分别为2.8、3.7、6.0和9.5 TFLOPS。

表3：变压器结构的变化。未列出的数值与基本模型的数值相同。所有指标都是在英德翻译开发集newstest2013上得到的。根据我们的字节对编码，列出的困惑是按字数计算的，不应该与按字数计算的困惑进行比较。

| | $N$ | $d_{model}$ | $d_{ff}$ | $h$ | $d_k$ | $d_v$ | $P_{drop}$ | $\epsilon_{ls}$ | 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) | 位置嵌入而不是正弦波 | | | | | | | | | 4.92 | 25.7 | |
| big | 6 | 1024 | 4096 | 16 | | | 0.3 | | 300K | **4.33** | **26.4** | 213 |

在表3行（B）中，我们观察到，减少注意键大小$d_k$，会伤害模型质量。这表明，确定兼容性并不容易，比点积更复杂的兼容性函数可能是有益的。我们在(C)和(D)行中进一步观察到，正如预期的那样，更大的模型更好，而且dropout对于避免过度拟合很有帮助。在(E)行中，我们用学习过的位置嵌入[8]来取代我们的正弦波位置编码，观察到的结果与基础模型几乎相同。

## 7 总结

在这项工作中，我们提出了Transformer，这是第一个完全基于注意力的序列转换模型，用多头的自我注意力取代了编码器-解码器架构中最常用的递归层。

对于翻译任务，Transformer的训练速度明显快于基于递归或卷积层的架构。在WMT 2014英译德和WMT 2014英译法的翻译任务中，我们实现了新的技术状态。在前一项任务中，我们的最佳模型甚至超过了之前报道的所有组合。

我们对基于注意力的模型的未来感到兴奋，并计划将其应用于其他任务。我们计划将Transformer扩展到涉及文本以外的输入和输出模式的问题，并研究局部的、受限的注意力机制，以有效地处理图像、音频和视频等大型输入和输出。使生成的顺序性降低是我们的另一个研究目标。

我们用来训练和评估模型的代码可在https://github.com/ tensorflow/tensor2tensor。

**鸣谢** 我们感谢Nal Kalchbrenner和Stephan Gouws的富有成效的评论、纠正和启发。

## 参考文献


[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.通过联合学习对准和翻译的神经机器翻译。*CoRR*，abs/1409.0473，2014。

[3] Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc V. Le.神经机器翻译架构的大规模探索。*CoRR*, abs/1703.03906, 2017.

[4] Jianpeng Cheng, Li Dong, and Mirella Lapata.*arXiv预印本arXiv:1601.06733*，2016。

[5] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio.使用rnn编码器-解码器学习短语表征，用于统计机器翻译。*CoRR*，abs/1406.1078，2014。

[6] Francois Chollet.Xception:深度学习与深度可分离卷积。*arXiv预印本arXiv:1610.02357*，2016。

[7] Junyoung Chung, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio.关于序列建模的门控递归神经网络的经验性评估。*CoRR*, abs/1412.3555, 2014.

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

[9] Alex Graves.用递归神经网络生成序列。*arXiv预印本arXiv:1308.0850*, 2013。

[10] 何开明, 张翔宇, 任少卿, 和孙健.用于图像年龄识别的深度残差学习。在*IEEE计算机视觉和模式识别会议论文集中*，第770-778页，2016。

[11] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber.递归网中的梯度流：学习长期依赖关系的困难，2001年。

[12] Sepp Hochreiter和Jürgen Schmidhuber.长短期记忆.*神经计算*，9（8）：1735-1780，1997。

[13] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu.探索语言建模的极限。*arXiv预印本arXiv:1602.02410*，2016。

[14] Łukasz Kaiser和Ilya Sutskever.神经GPU学习算法。In *International Conference on Learning Representations (ICLR)*, 2016.

[15] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Ko-Ray Kavukcuoglu.线性时间内的神经机器翻译。*arXiv预印本arXiv:1610.10099v2*, 2017。

[16] Yoon Kim, Carl Denton, Luong Hoang, and Alexander M. Rush.结构化注意力网络。在*国际学习代表会议上*，2017年。

[17] Diederik Kingma 和 Jimmy Ba.Adam: 一种随机优化的方法.在*ICLR*, 2015.

[18] Oleksii Kuchaiev 和 Boris Ginsburg.LSTM网络的因式分解技巧。*arXiv预印本arXiv:1703.10722*, 2017。

[19] 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.

[20] 萨米-本吉奥 Łukasz Kaiser.主动记忆能取代注意力吗？In *Advances in Neural Information Processing Systems, (NIPS)*, 2016.

[21] Minh-Thang Luong, Hieu Pham, and Christopher D Manning.基于注意力的神经机器翻译的有效方法。
*arXiv预印本arXiv:1508.04025*，2015。

[22] Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit.一个可分解的注意力模型。In *Empirical Methods in Natural Language Processing*, 2016.

[23] Romain Paulus, Caiming Xiong, and Richard Socher.一个用于抽象化总结的深度强化模型。 *arXiv预印本arXiv:1705.04304*，2017。

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

[25] Rico Sennrich, Barry Haddow, and Alexandra Birch.带有子词单元的稀有词的神经机器翻译。 *arXiv预印本arXiv:1508.07909*，2015。

[26] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean.大得离谱的神经网络。*arXiv preprint arXiv:1701.06538*, 2017.

[27] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdi-
nov.Dropout：防止神经网络过拟合的简单方法。*机器学习研究杂志*，15（1）：1929-1958，2014。

[28] Sainbayar Sukhbaatar, arthur szlam, Jason Weston, and Rob Fergus.端到端内存网络。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.

[29] Ilya Sutskever, Oriol Vinyals, and Quoc VV Le.用神经网络进行序列到序列的学习。In *Advances in Neural Information Processing Systems*, pages 3104-3112, 2014.

[30] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna.重新思考计算机视觉的初始架构。*CoRR*, abs/1512.00567, 2015.

[31] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al.谷歌的神经机器翻译系统。弥合人类和机器翻译之间的差距。
*arXiv预印本arXiv:1609.08144*，2016。

[32] 周杰，曹颖，王旭光，李鹏，和徐伟。带有快进连接的深度递归模型用于神经机器翻译。*CoRR*, abs/1606.04199, 2016.