课程作业

课程名称 ： **计算机语音技术**
作业名称 ： **文献阅读报告**
学号 ： **21281280**
姓名 ： **柯劲帆**
班级 ： **物联网2101班**
指导老师 ： **朱维彬**
修改日期 ： **2023年12月24日**

> 文献名称：**Attention Is All You Need**
>
> Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[J]. Advances in neural information processing systems, 2017, 30.

1. 文献解读

1.1. 引言

引言首先在第一段叙述了当时（2017年）的深度学习主流架构：RNN架构，以LSTM和GRU为代表的RNN在多项序列任务中取得了SOTA的结果，主要的研究趋势是不断提升递归语言模型和“encoder-decoder”架构的能力上限。

接着第二段叙述了RNN的不足，主要是其必须使用串行计算，必须依照序列的顺序性，并行计算困难。

而注意力机制的应用可以无视序列的先后顺序，捕捉序列间的关系。

因此文章提出一种新架构Transformer，完全使用注意力机制来捕捉输入输出序列之间的依赖关系，规避RNN的使用。实验证明，这种新架构在GPU上训练大幅加快，且达到了新的SOTA水平。

总结：

- 论文主题：提出一种新架构
- 解决问题：提出能够并行计算的架构替代RNN
- 技术思路：仅使用注意力机制处理输入输出序列

1.2. 算法

1.2.1 整体架构

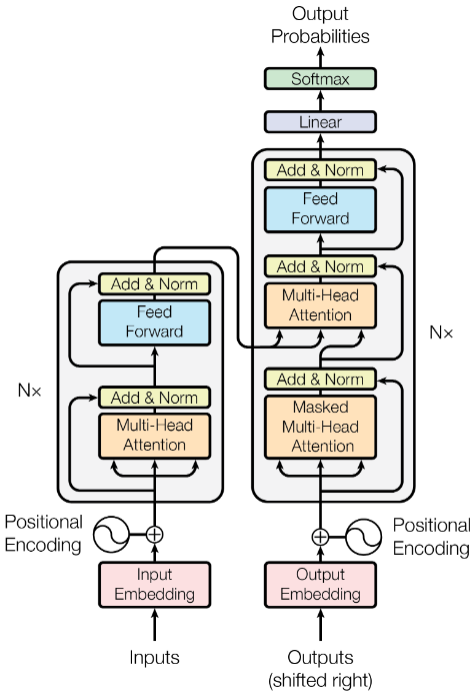

图1 子编码器架构

上图是Transformer架构的模型网络结构。

整个Transformer模型由$N = 6$个这样的Encoder-Decoder对组成，如下图所示：

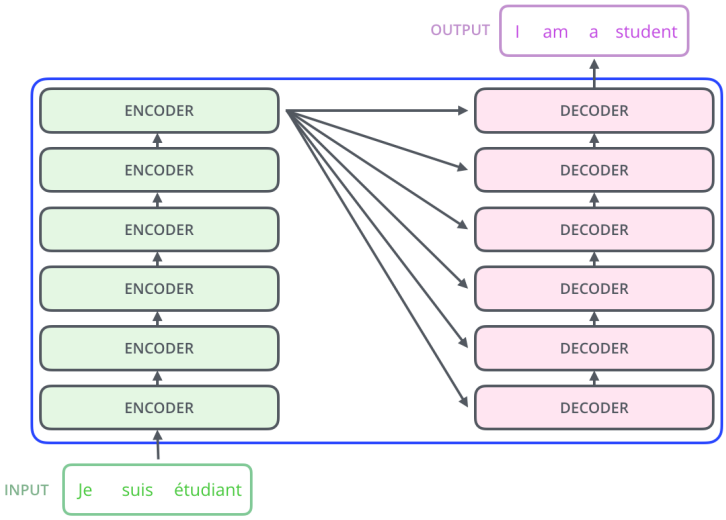

图2 编码-解码器架构

每个子编码器的架构相同，为图1所示，但是各个子编码器的模型权重不同；同理，每个子解码器的架构相同，权重不同。

因此，模型的数据处理过程如下：

1. 对输入数据进行文本嵌入；
2. 对输入数据进行位置编码；
3. 将输入数据依次输入$N = 6$层子编码器；
4. 将最后一层子编码器的输出分别传入每层子解码器；
5. 对目标数据进行文本嵌入；
6. 对目标数据进行位置编码；
7. 将目标数据依次输入$N = 6$层子解码器；
8. 使用全连接层和Softmax层将解码器的输出转换为预测值并输出。

1.2.2. 缩放点乘注意力

这一部分包含于多头注意力模块之中。

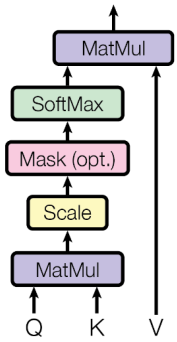

图3 缩放点乘注意力计算描述

计算公式如下：

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

其中，输入的Q为查询（Query）向量，K为关键字（Key）向量，V为值（Value）向量，由输入token序列X计算得到：

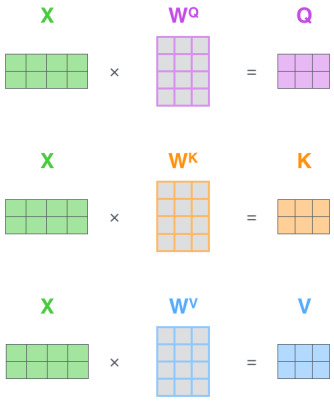

图4 自注意力矩阵计算描述

其中，W^Q、W^K和W^V都是待学习的权重。

接着，QK^T表示将查询向量Q与关键字向量K做内积，意在计算两者的相关性。查询与对应token的关键字的相关性越高，得到的结果矩阵中的对应值越高，受到注意力就会越高。将结果矩阵除以$\sqrt{d_k}$（避免较高值造成Softmax层的梯度消失），送进Softmax层进行概率计算。

得到概率矩阵后，将概率矩阵与值向量V相乘，作为对应token的值的权重。

最后，将加权的值向量输出。

注意力机制算法的核心思想是，对于每个token的键值对，查询与一个token的键越相似，就对该token的值越关注。

1.2.3. 多头注意力

这部分对应于图1中的`Multi-Head Attention`。

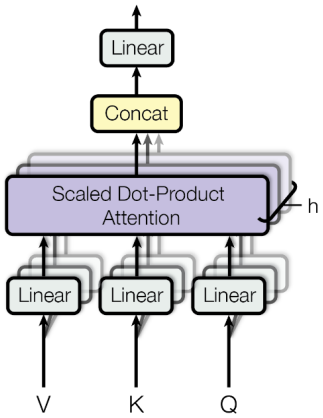

图5 多头注意力计算描述

计算公式如下：

$$\begin{aligned}\text{MultiHead}(Q,K,V) &= \text{Concat}\left(\text{head}_1,\ldots,\text{head}_\text{h}\right)W^O \\ \text{where head} &= \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)\end{aligned}$$

多头注意力实际上是多个自注意力的叠加，类似卷积神经网络中多个通道特征，这样可以有效学习到多个不同的特征。

将查询向量Q、关键字向量K和值向量V复制$h=8$份（h即为“多头”的头数），分别放入多个自注意力计算模块中计算特征。最后将得到的特征堆叠，送入全连接层融合。

1.2.4. 应用注意力机制

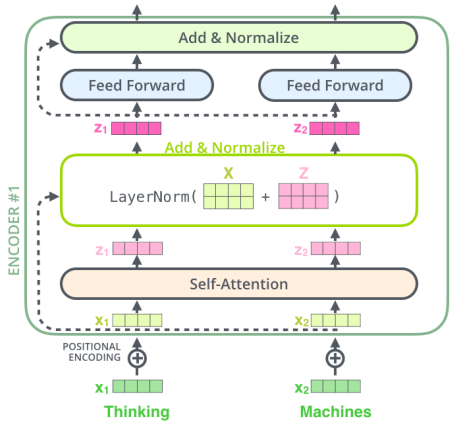

图6 子编码器计算流程

在编码器中，每一个子编码器都由以下计算流程依次组成：

1. 输入token序列X进行多头注意力计算；
2. 将h个自注意力模块计算结果使用全连接层整合；
3. 与原输入序列X相加，构成残差网络，并进行层归一化；
4. 将结果送入前馈网络；
5. 进行残差计算，随后归一化，最后输出。

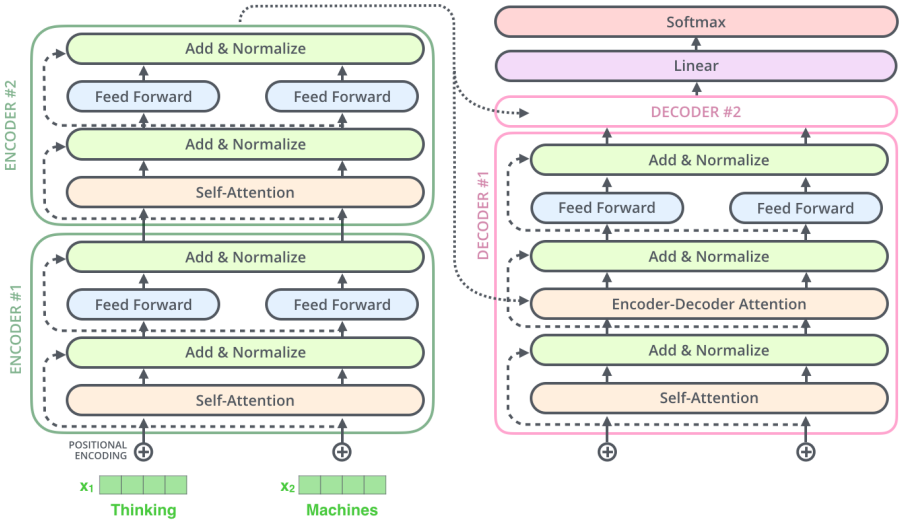

图7 编码解码器计算流程

在解码器中，每一个子解码器都由以下计算流程依次组成：

1. 对目标token序列进行掩码处理；
2. 送入多头注意力模块计算；
3. 与输入的目标序列求和计算残差并归一化；
4. 将编码器输出结果作为查询向量Q，从目标序列计算出关键字向量K和值向量V，输入自注意力模块计算；
5. 与输入序列求和计算残差并归一化；
6. 将结果送入前馈网络；
7. 与输入序列求和计算残差并归一化，最后输出。

1.2.5. 前馈网络

计算公式如下：

$$FFN\left(x\right)=max\left(0,xW_1+b_1\right)W_2+b_2$$

两层全连接层之间使用RELU作为激活层。

1.2.6. 掩码多头注意力

由于Transformer架构支持并行计算，当预测$Output_t$需要在时，不能向模型提供$Output_{t+1}$及之后的子序列，因此要对目标序列输入进行掩码。

比如一组数据为

```
{
    Input: "i love computer speech technology",
    Target: "我爱计算机语音技术"
}
```

希望预测Target[4]“机”时，只能向模型提供Target[0:4]“我爱计算”，而不能输入后面的序列，因此需要将其遮掩。

遮掩方法是让注意力公式的Softmax的输入为$-\infty$，那么得到的后面的token的注意力权重就几乎为0，达到遮住后面的token的效果。

即希望预测Target[4]“机”时，Softmax的输入相当于["我", "爱", "计", "算", "$-\infty$", "$-\infty$", "$-\infty$", "$-\infty$", "$-\infty$"]，其中文字代表其对应的计算得到的注意力权重。

1.2.7. 位置编码

计算公式为：

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

其中pos为token在序列中的下标，$2i$或$2i+1$为词向量的维度序号。即词向量维度为偶数时使用正弦函数，为奇数时使用余弦函数。

该函数满足以下性质：

- 对于一个词嵌入向量的不同元素，编码各不相同；
- 对于向量的同一个维度处，不同pos的编码不同。且pos间满足相对关系：

  $$\begin{cases}PE(\text{ pos }+k,2i)=PE(\text{ pos },2i)\times PE(k,2i+1)+PE(\text{ pos},2i+1)\times PE(k,2i)\\PE(\text{ pos }+k,2i+1)=PE(\text{ pos },2i+1)\times PE(k,2i+1)-PE(\text{ pos },2i)\times PE(k,2i)\end{cases}$$

  从实际意义上看，即例如Target[4]“机”的位置编码可以被Target[1]“爱”和Target[3]“算”的位置编码线性表示。

1.3. 结论

1.3.1. 实验结果

研究测试了“英语-德语”和“英语-法语”两项翻译任务。使用论文的默认模型配置，在8张P100上只需12小时就能把模型训练完。

研究使用了Adam优化器，并对学习率调度有一定的优化。模型有两种正则化方式：

1. 每个子层后面有Dropout，丢弃概率0.1；
2. 标签平滑。

Model	BLEU		Training Cost (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	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [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 Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

图8 翻译任务实验结果

实验表明，Transformer在翻译任务上胜过了所有其他模型，且训练时间大幅缩短。

论文同样展示了不同配置下Transformer的消融实验结果。

	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

图9 消融实验结果

实验A表明，计算量不变的前提下，需要谨慎地调节h和d_k、d_v的比例，太大太小都不好。这些实验也说明，多头注意力比单头是要好的。实验B表明，d_k增加可以提升模型性能。作者认为，这说明计算key、value相关性是比较困难的，如果用更精巧的计算方式来代替点乘，可能可以提升性能。实验C, D表明，大模型是更优的，且dropout是必要的。如正文所写，实验E探究了可学习的位置编码。可学习的位置编码的效果和三角函数几乎一致。

1.3.2. 研究结论

Transformer这一仅由注意力机制构成的模型。Transformer的效果非常出色，不仅训练速度快了，还在两项翻译任务上胜过其他模型。

Transformer在未来还可能被应用到图像、音频或视频等的处理任务中。

1.4. 个人见解

在接触过的研究中，我使用过很多基于Transformer架构的模型。

在图片Caption任务中，我将Transformer与CNN和RNN结合的编码-解码器进行效果比较，后者的效果非常差，而Transformer可以达到较好的效果。

另外，基于Transformer架构的各类大模型可以达到很好的效果，其中我体验过商用的ChatGPT，也复现过开源的Llama2、LLaVA等VQA任务模型，使用过RoBERTa等BERT-base模型完成token分类任务，使用ViT为CLIP等图像分类任务模型进行图像特征提取等。

由于学识不足，个人无法独立给出对Transformer架构本身的评价。因此，本人查阅了近年来对Transformer进行改进或替代的研究，搜集到如下几点问题或改进方法：

- Transformer模型中自注意力机制的计算量会随着上下文长度的增加呈平方级增长，计算效率非常低。最近一项研究Mamba: Linear-Time Sequence Modeling with Selective State Spaces针对长文本有更好的效果；
- SIMPLIFYING TRANSFORMER BLOCKS发现可以移除一些Transformer模块的部分，比如残差连接、归一化层和值参数以及MLP序列化子块（有利于并行布局），以简化类似 GPT 的解码器架构以及编码器式BERT模型。

2. 参考和引用资料

- https://zhuanlan.zhihu.com/p/569527564
- https://jalammar.github.io/illustrated-transformer/