「课代表来了」跟李沐读论文之——Transformer

原创 郭必扬 SimpleAI 2021-12-19 13:22

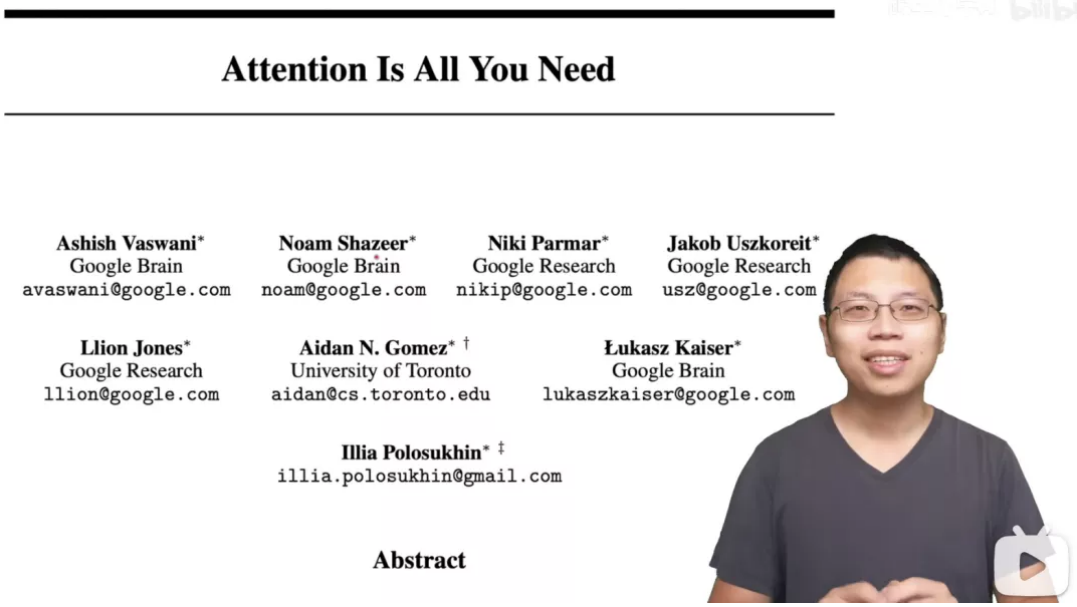
收录于话题
#Nice NLP Notes 19 #不一样的论文解读 16

「课代表来了」跟李沐读论文之——Transformer

墙裂推荐沐神在B站上开的一系列公开课，尤其是最近出的“论文精读”系列，真的讲得太好了。可能很多经典论文咱们之前也读过，但是听李沐老师再讲一遍，我们又可以收获很多新知识，尤其是李沐老师结合自己丰富的经验做出的很多精彩评论。

我也慕名去听了课，第一堂课我选择听NLP经典论文"Attention Is All You Need"，边听边整理了如下笔记 📒，方便自己日后查阅，也分享给大家。

李沐老师读论文的顺序一般是这样的：**标题+作者 > 摘要 > 导言 > 结论 > 相关工作 > 模型设计 > 实验，最后做一个总评**。我这里也是按照这个顺序进行记录的。



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

标题+作者

- 标题：Attention Is All You Need
- 发表：NIPS-2017
- 机构：Google

*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.

> Comments by Li Mu: 8个作者都是共同贡献，这在计算机论文中很少见。但是为了解释为什么有这么多共同一作，论文在脚注中清楚了写明了每个人在这个工作中的贡献，确实大家都担得上“共同贡献”这一说。

摘要

背景是机器翻译、encoder-decoder框架，而且一般都会在中间使用Attention机制。

本文提出了一个新的简单的网络架构——Transformer，完全基于attention机制。

> Comments by Li Mu: 这是个好名字，跟“变形金刚”表达相同

贡献：网络简单，且跟之前的网络结构都不一样，不使用rnn或cnn的单元。并行度更好，训练快很多。在机器翻译上取得了更好的效果。

> Comments by Li Mu: 最开始Transformer是在机器翻译的背景上提出的，这是个小领域，因此可能最初并没有收到广泛关注。后面Transformer开始在各种其他领域大显神通，到最近在vision上面也表现优异，才真正火出圈了

结论

本文提出的Transformer是第一个纯基于attention的序列转录模型，使用multi-head self-attention替代了之前的rnn结构。

在机器翻译上，比RNN和CNN都要快，还取得了新的SOTA。

作者们对这种纯attention的模型能取得这么好的效果感到非常激动。因此作者们期待Transformer能在其他领域（CV，audio等）大放异彩。

> Comments by Li Mu: 作者在2017年的时候就一定程度上预测了未来。如今Transformer不光横扫了NLP的各种任务，还在CV上也取得了惊人的效果。

最后代码开源在tensor2tensor库里面。

> Comments by Li Mu: 其实代码最好是放在摘要的最后一句话，让读者可以第一时间看到代码。

Intro（导言）

介绍了传统的RNN，CNN以及encoder-decoder架构。分析了RNN的缺点：1. 难以并行 2. 容易遗忘。再介绍了attention机制。最后提出了一个全新的架构Transformer。

> Comments by Li Mu: 这个intro很短，主要是内容比较多，而NIPS的篇幅较短。

Background（相关工作）

为了提高对序列数据的计算效率，很多工作都使用卷积神经网络作为基础的building block来进行模型构建，从而实现并行的计算。然而，CNN是通过滑动窗口来提取特征，所以对于长距离的关系较难捕捉。但CNN还有一个优点——**多通道机制**，使得模型可以从多个角度去提取数据的特征。

所以**Transformer借用了多通道的思想，设计了多头的注意力机制**。

另外，self-attention不是本工作提出了，而是在曾经的很多工作中都被成功应用了。

模型

序列模型中较好的是encoder-decoder架构。

要点：

1. encoder把输入序列处理得到中间表示，然后decoder读入这个中间表示，处理后得到输出序列；
2. 输入序列和输出序列不一定一样长；
3. decoder是一种auto-regressive的方式来输出的，即每一步都会读入上一步的输出。

Transformer依然是一个encoder-decoder的架构，但它主要组成是**self-attention**和**point-wise fully connected layer**，结构如下：

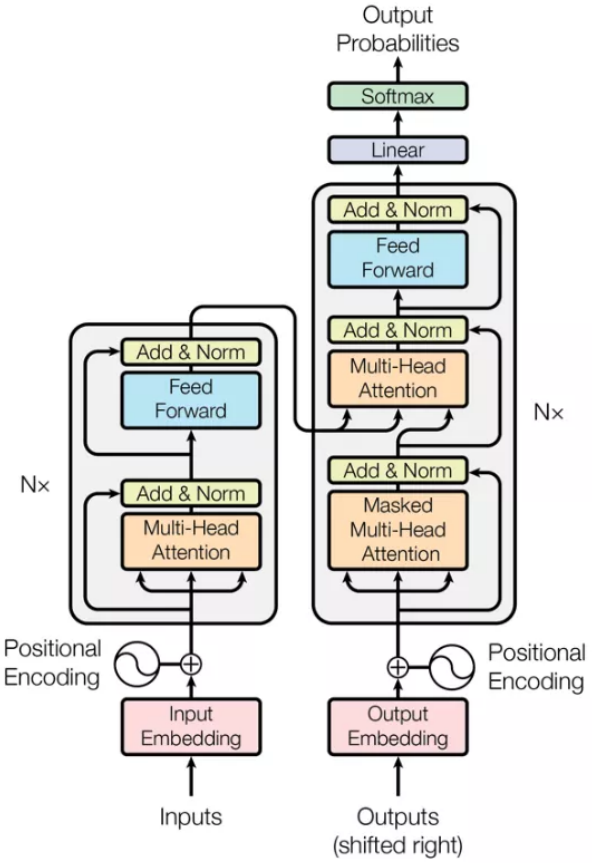


Figure 1: The Transformer - model architecture.

> Comments by Li Mu: 这个图画的很好，在神经网络的时代，画图是一个重要的技能。然而这个图属于那种“不明觉厉”的图，很好看，但是不容易看懂。

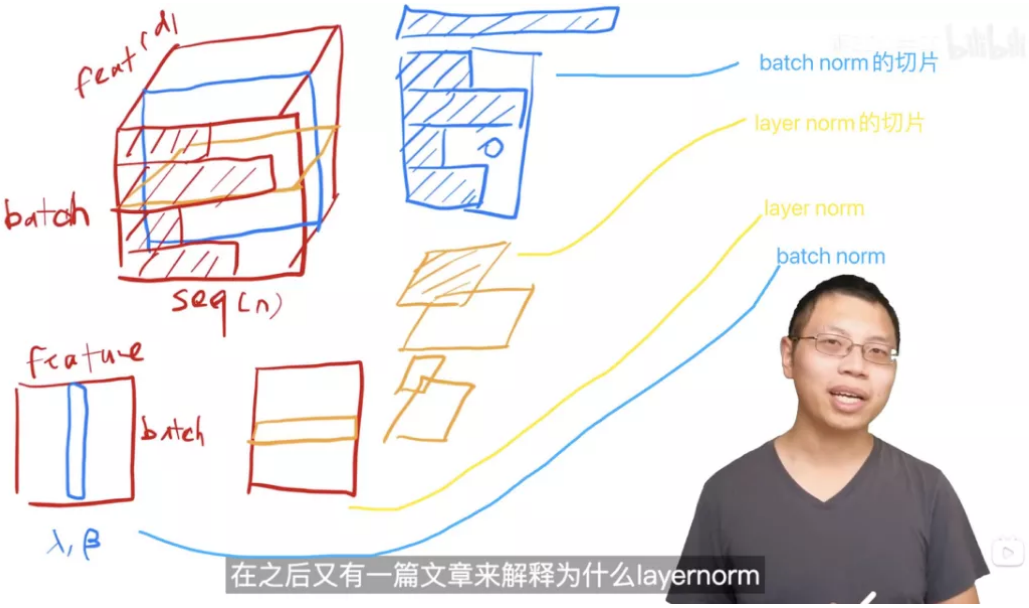
Encoder

- Encoder由N=6个一模一样的层组成；
- 每个层，包含2个子层：①multi-head self-attention layer，②position-wise fully connected feed-forward network（就是个MLP）；
- 每个子层，都会使用residual connection和layer norm来处理，子层的输出都可以表示为：$LayerNorm(x + Sublayer(X))$；
- 为了方便残差连接，上面所有的层、包括embedding层，都使用d=512作为输出维度。

> Comments by Li Mu: 总之，Encoder就俩超参数：N和d。这种设计直接影响了后面各种基于Transformer的模型设计，比如BERT，GPT等等，都主要调节这两个参数。

「插播」沐神小课堂——什么是Layer Norm：

- Batch Norm就是把一个batch的tensor，按照feature的每个维度（即按照列）去进行规范化（均值0方差1）
- Layer Norm则是在batch内逐个样本去做规范化



我们在序列化数据中更常使用的是Layer Norm，因为序列的长度会变化，如果使用batch norm的话，可能导致均值方差波动很大，从而影响效果，而layer norm则是逐个样本去进行的，就不会受影响。

Decoder

- 跟Encoder一样由N=6个一模一样的层构成；
- 每个层，包含3个子层，相比于Encoder中的设计，多了一个multi-head attention layer；
- 为了防止Decoder在处理时看到未来的信息，这里对self-attention做了进一步的处理，即使用了一个**mask**机制，在t时刻时把后面的单元都mask掉，从而不会attend到未来的信息。

逐个看看每个sub-layer：

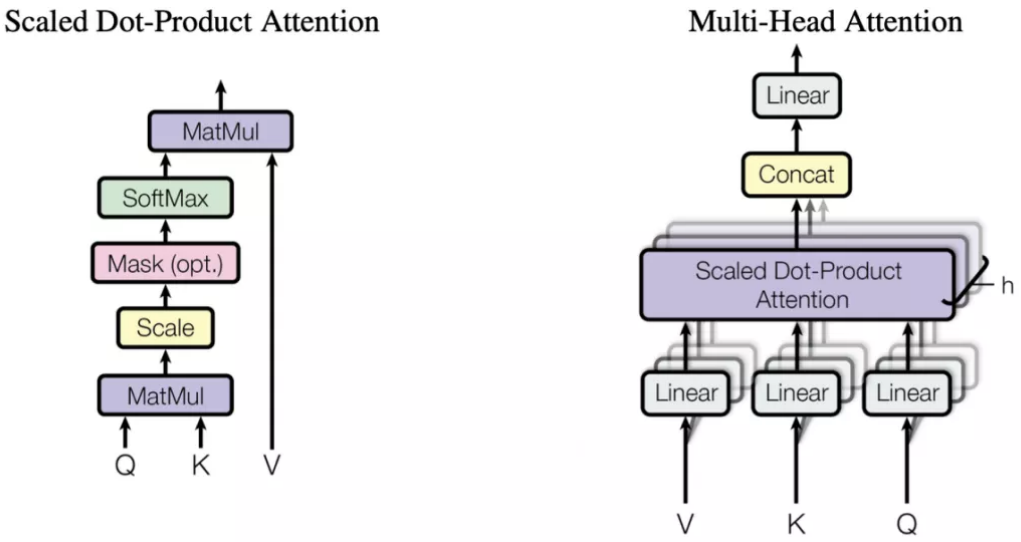
Scaled Dot-product Attention

在Transformer中我们使用的attention机制是Scaled Dot-product Attention，下图中的d_k代表的Q，K，V的维度：

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

这里的attention机制，相比于经典的Dot-product Attention其实就是多了一个scale项。这里的作用是啥呢？当d比较小的时候，要不要scale都无所谓，但是当d比较大时，内积的值的范围就会变得很大，不同的内积的差距也会拉大，这样的话，再经过softmax进一步的扩大差距，就会使得得到的attention分布很接近one-hot，这样会导致梯度下降困难，模型难以训练。在Transformer中，d=512，算比较大了，因此需要进行scaling。

下图很清晰地展示了scaled Dot-product Attention是如何进行的：



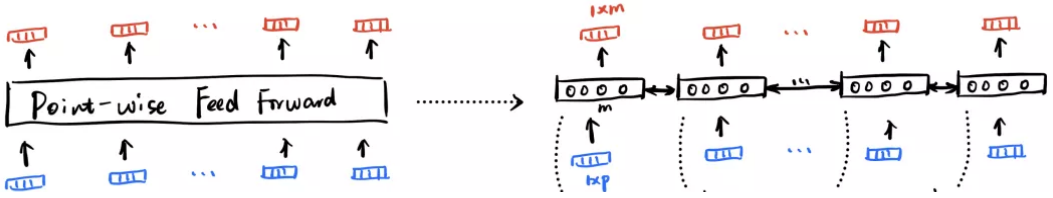
Multi-Head Attention

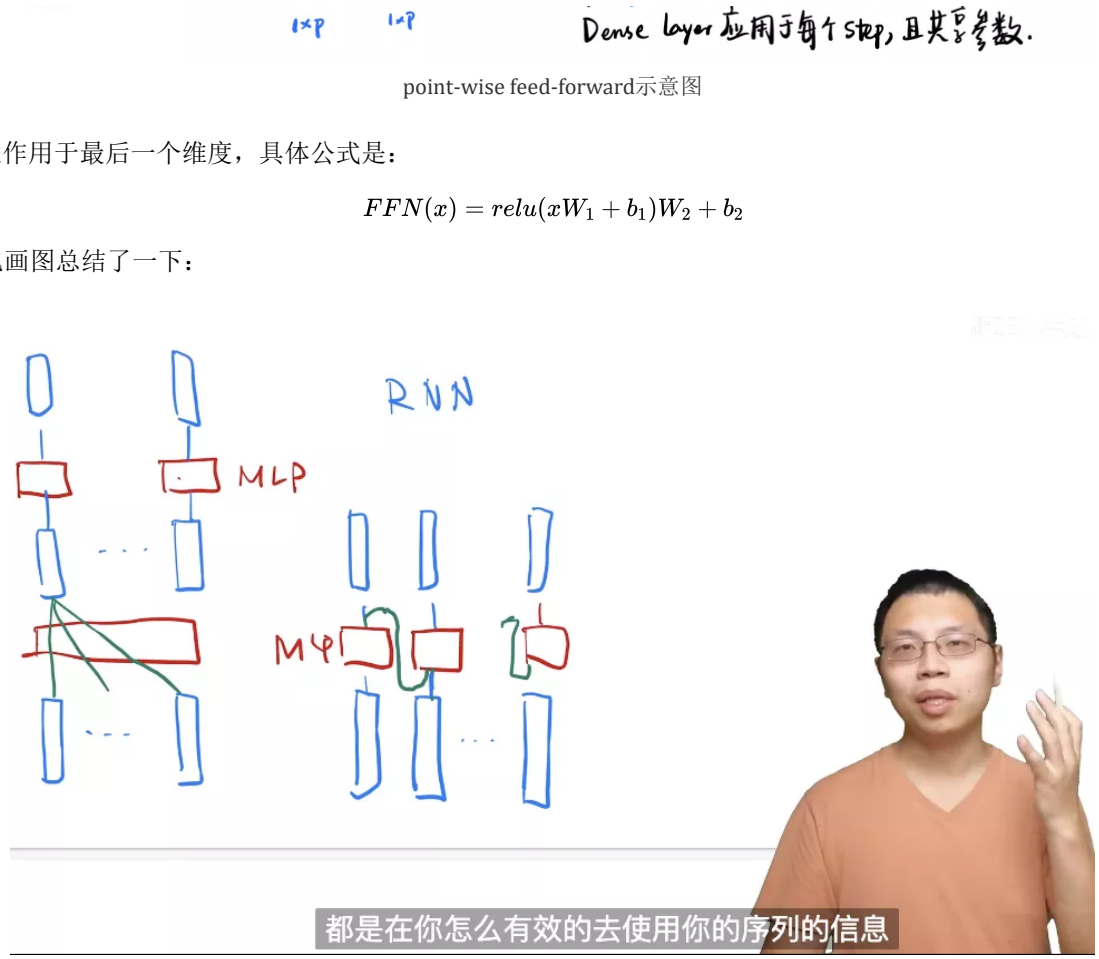
原本的SDP Attention，没什么可学习的参数，作者发现，我们可以先把原本的向量，通过线性层隐射到多个低维的空间，然后再并行地进行SDP Attention操作，在concat起来，可以取得更好的效果。这类似于CNN中的多通道机制。一个向量先隐射成多个更低维的向量，相当于分成了多个视角，然后每个视角都去进行Attention，这样模型的学习能力和潜力就会大大提升，另外由于这里的降维都是参数化的，所以让模型可以根据数据来学习最有用的视角。

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

Point-wise Feed-forward Networks

这里其实就是名字比较花哨，实际上就是简单的MLP。唯一需要注意的是这个MLP的修饰词——Point-wise，它的意思是它是对每个position（词）都分开、独立地处理。我之前在文章「一个小问题：深度学习模型如何处理大小可变的输入」中画过一个图：





point-wise feed-forward示意图

即MLP只是作用于最后一个维度，具体公式是：

$$FFN(x) = relu(xW_1 + b_1)W_2 + b_2$$

沐神后面也画图总结了一下：

这个图左边是Transformer示意图，右边是RNN示意图（其中对RNN单元简化成一个MLP，本质上类似）。

- Transformer是通过attention来全局地聚合序列的信息，然后通过MLP进行语义空间的转换；
- RNN则是通过把上一时刻的信息传入下一时刻的单元，来使用序列信息，但也是通过MLP进行语义空间转换。所以二者本质区别在于如何使用序列的信息。

Embedding和softmax层

Transformer中使用了三处embedding：对input和output的token进行embedding，以及在softmax前面的Linear transformation中也使用跟embedding相同的权重（这样是为了能够把decoder的output通过相同的embedding给转换回token的概率，因为embedding的作用就是做token跟vector之间的转换）。三处的embedding都是同样的权重。

另外值得注意的点就是，作者把embedding都乘上了$\sqrt{d_{model}}$。这是为了在后面跟position embedding相乘的时候能够保持差不多的scale。

Position Encoding

由于self-attention实际上是不考虑序列顺序的，只是单纯把各个position的信息进行聚合，无论顺序怎么改变，对于self-attention来说都是一样的。因此，这里特意设计了position encoding这种东西，来添加位置信息。

具体的，文章使用的是周期不同的sin和cos函数来计算得到每个position的Embedding：

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

其中pos是具体位置的index，i则是具体的dimension。总之，这里给每个位置，都构造了一个长尾$d_{model} = 512$的向量，来作为该position也就是某个具体token的位置表示。

最后，这个position encoding是直接跟embedding相加，输入到模型中。

为啥使用Self-Attention

> Comments by Li Mu: 整个文章实际上对模型的解释是比较欠缺的。

作者主要通过下面这个表来对比self-attention和其他结构的差别（restricted self-attention不用管，不怎么用）：

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)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

上图中，sequential operations衡量的是在处理序列的时候的并行复杂度，越小说明并行度越高；max path length则表示序列中任意两个点传递信息的最大距离。

Self-attention的主要优势在于并行度高（相比RNN）、信息距离短（相比RNN和CNN）。而在复杂度方面，其实没有明显优势：

- self-attention每层的复杂度是$O(n^2 \cdot d)$，是因为有n个position，每个position都要计算n次attention，在计算attention的时候，是进行维度为d的内积，所以复杂度是n * n * d；
- RNN层的复杂度是$O(n \cdot d^2)$，因为有n个position，每个position会经过一个线性变化（矩阵乘法），变换的复杂度是d * d，因此最终复杂度是n * d * d；
- CNN实际上跟RNN类似，但因为往往涉及到多个kernel，所以多乘了一个k。但一般在NLP中k也不大，所以没有大差别。

在NLP中，往往n和d都会比较大，所以这三者的计算复杂度没有质的差别。

> Comments by Li Mu: 看起来应该Transformer在计算效率上会比CNN、RNN更快，但我们现在体会到的Transformer的模型却不是这样的，为啥呢？实际上是因为self-attention对整个模型的假设更少，所以我们往往需要更大量的数量、更大的模型才能够训练出跟CNN、RNN同样的效果来。这就导致现在基于Transformer的模型都特别大特别贵。

这里就涉及到inductive bias这个概念了，在 什么是Inductive bias（归纳偏置）？ 文章中，我们知道合理的inductive bias可以让模型的训练加速，这里由于self-attention的inductive bias相比于CNN、RNN更少，所以训练起来也更费劲。但我猜想，对于大规模预训练来说，少一点inductive bias是不是更好？

训练细节&超参数

英语翻译德语，使用BPE分词法构造英语德语共用的词典，使用8个P100 GPU，每个batch大概0.4s，总共训练了12小时，其实时间成本相对来说还是可承受的。

学习率使用了warmup，先增强后减。

使用了两种正则化：

1. Residual dropout，对每个sub-layer的输出都使用了dropout，还对embedding层也使用dropout，dropout rate=0.1
2. Label Smoothing，使用了程度为0.1的smoothing，这会损害一点的perplexity，但是nuisance提高accuracy和BLEU得分

下表则是展示了不同的模型结构（超参数）的性能差别：

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

实际上可修改的参数不多，主要就是层数（N）、向量维度（d_{model}）、头数（h）。像d_k, d_v都是根据d_{model}和h算出来的。

沐神点评

写作上

这篇文章写的非常简洁，没有太多的介绍和解释，属于大佬的写作风格。不过对于我们大多数研究者来说，还是需要尽可能把背景解释清楚，在正文花足够的篇幅在把故事讲清楚，从而让读者更好理解、认识的更深入。

Transformer

这个模型的最大意义在于给NLP届乃至CV、audio等其他模态的数据提供了统一的模型架构，有点类似于CNN刚提出时对CV领域的巨大变革。有了Transformer，以及现在各种基于Transformer的预训练模型、其他模态的成功运用，Transformer对于多模态学习的进步有着深远的意义。

然后，纵使Transformer已经提出了多年，我们对Transformer的真正原理依然缺乏理解。例如，文章的标题Attention is all you need实际上也是不对的，后续的研究者已经证明Transformer的成功对于MLP、residual connection等其他组件也是缺一不可的，attention的作用就是对序列的信息做了聚合，并不是attention一个人起了全部作用。

另外，self-attention相比如RNN、CNN等的优势，可能在于它所作的归纳偏置（inductive bias）更加一般化，所以经过大规模的训练，可以学习到更丰富、更一般化的知识。但代价就是它对数据的关键信息的抓取能力就下降了，我们需要更

多的数据更大的模型才能训练出理想的效果。

但Transformer的出现，给了整个AI届新的活力，让我们发现在CNN、RNN统治的时代，我们依然可以设计出新的网络结构发光发热，因此也带动了一系列新的网络架构设计，比如纯MLP的模型等等。

以上就是课代表带来的听课笔记啦！笔记虽然简单，但毕竟是二手的知识，最推荐的还是大家直接看视频，品味原汁原味的沐神の味道。下一期我们继续听沐神来讲解大名鼎鼎的BERT，咱们不见不散~

其他原创好文

什么是Inductive bias（归纳偏置）？

一个小问题：深度学习模型如何处理大小可变的输入

使用Huggingface+PyTorch来微调模型「初级教程完结撒花丶(°▿°)ノ」

何时能懂你的心——图卷积神经网络（GCN）

从此明白了卷积神经网络（CNN）

Hello NLP(2)——关于word2vec你想知道的一切

整理12小时，只为让你20分钟搞懂Seq2seq

博一结束后的一些反思

SimpleAI

追求用简单、有趣的方式来分享AI知识。

77篇原创内容

公众号

收录于话题 #Nice NLP Notes 19

上一篇 · 一条龙搞定情感分析：文本预处理、加载词向量、搭建RNN

文章已于2021/12/19修改

喜欢此内容的人还喜欢

[源码解析] PyTorch 分布式(8) -------- DistributedDataParallel 之 论文篇

罗西的思考

DeepLabv3+论文精读

Geek部落格

从顶会论文看多模态预训练研究进展

python遇见NLP