Attention is all you need 论文精读笔记 Transformer

文章标签： transformer 深度学习 人工智能

目录

- 前言
- 正文
  - Relative works
  - 模型架构
    - 注意力 Attention
    - Position-wise Feed Forward Network
    - Embeddings and softmax
    - Positional Encoding
  - 为什么使用自注意力
  - Evaluation
  - 论文讨论与总结

前言

本文主要内容来自李沐老师的论文精读视频。文章需要部分相关基础。

> Transformer论文逐段精读【论文精读】_哔哩哔哩_bilibili
>
> 更多请见：https://github.com/mli/paper-reading
>
> https://www.bilibili.com/video/BV1pu411o7BE

其实 Transformer 的机制架构在 李宏毅老师 2021 机器学习课程中学过一遍了，但是当时没有做笔记，于是现在已经忘的只记得图长什么样了，Multi-head attention之类已然忘得无影无踪，所以这次计划多做一些笔记。

感觉李沐老师的论文精读系列更适合研究生等有论文写作需要的，会讲很多论文写作的事项，会剖析作者的写法，而且讲的很细，会把所以然讲清楚。

李宏毅老师的机器学习系列是对目前最流行的深度学习的各个领域的各个模型有一个很快的概览，当然包括模型的基本思路原理都会说清楚，很通俗易懂，课程很有意思，PPT的制作更是我辈楷模。上他的课经常会有：“原来这个是这样的 ”的感叹。

此外最近也计划看李沐老师的 d2l 动手学深度学习v2，感觉有很多落实到代码的介绍，沐神YYDS就完事了，之前直播课的时候就有关注可惜自己太菜了，没跟住，就看了最基本的一部分。之后希望有机会看一波做个笔记。

> 课程安排 - 动手学深度学习课程
>
> 免费深度学习在线课程

https://courses.d2l.ai/zh-v2/

正文

Relative works

CNN 有一个固定的问题，就是较难解决距离较远之间的元素的关系，因为CNN是局部一层一层，要到较高的层之后才能看到两个距离较远的元素。而这篇文章还看中 CNN 的一个很好的特性，就是 Multi-Channel 的输出，因为 CNN 可以有多个 Filter 分别检测或者识别不同的 Pattern，作者也想要类似这个的功能，作者用了 Multi-Head Attention。

模型架构

因为是看到中间26min处开始记笔记的，开头可能一部分记的不全。

Batch Normalization（Batch Norm）：一个 Batch 的 Feature Vector 其实就相当于是一个矩阵，我们将同一个 Feature， 不同样本的值做正则，对于一个Batch的数据，同一个 Feature 下不同样本的数值，均值变成0，方差变成1。（方法就是每个样本减去均值，除以标准差）在训练时，会每个Batch做这个正则，在预测的时候，我们会把之前全局的均值和标准差计算出来保存，对于新输入的要预测的样本，做类似的正则。

Layer Normalization（Layer Norm）：对于一个样本，把所有的Feature值拿出来做正则。比如前文提到的一个 Batch 的 Feature Vector 其实就相当于是一个矩阵，这个矩阵做转置、Batch Norm、再转置，就是 Layer Norm。相当于是 Batch Norm 对每列做处理，让其均值为0，方差为1，Layer Norm 对每行做处理。

然后因为我们对文本序列做处理，这里就需要注意了。因为文本相当于单词的序列，也就是一个样本是一个序列，而序列的每个元素（token）做了嵌入之后也是个向量，就相当于一个 Batch 是一个 3D 的表示，有样本，样本内的词，词向量中元素的值。但是做Norm还是一样的。Batch Norm，还是对于同一个特征（词向量某个位置的元素），不同的样本，不同的词，都要放在一起，把截出来的二维矩阵拉成向量作 Norm。

为什么我们这个场景用 Layer Norm 而不是 Batch Norm？因为其实对于文本来说，长度不确定，后面可能填很多零。同一个词向量的位置，可能对于很多样本，都是填充的0。有用的部分是不确定的，抖动是比较大的。但是做 Layer Norm，就是同一个句子内部做这个操作。每个样本做操作，也不需要存全局的均值和方差。

自回归Auto Regressive：当前输出的输入集是上面一些时刻的输出。预测的时候是不能看到后面的输出。

Decoder：因为在预测一个词的时候，应该不能看到后面的输出。做法就是带掩码的注意力 机制。

注意力 Attention

注意力的思路就是加权求和，之前很多地方都介绍过。就是有一组 Key，和对应的 Value。每次都有一个 Query 进来，Query会和Key算相似度，和哪个key相似度大，那么对应的Value在求和的时候权重就高。那么怎么算相似度呢？各有各的算法。本文用的叫做 Scaled dot product Attention。

Scaled dot product Attention: 沐神表示名字长但是最简单hhh。方法就是 Query 和 Key 做内积，假设 Query 和 Key 的维度是 dk，那么内积之后的结果除以 根号dk ，再做 Softmax 就得到每个 value 求和的权重。补充说明一下下面的公式，Q是Query，大小 n*dk，K是key，大小m*dk，所以Q与K转置的乘积

矩阵就是 n * m，而V是一个m*dv的矩阵，所以最后的输出就是一个 n*dv 的矩阵。每一行代表一个 query，每一列代表对应 value 的权重（当然公式中还要除以根号dk，然后做 softmax）

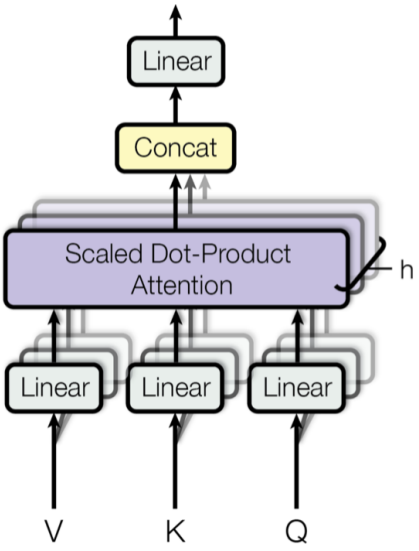
$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

PS：根号 Latex \sqrt{}

作者表示流行的注意力有 additive attention 和 dot-product attention，前面的可以处理 Query 和 Key 不等长的情况。但是点积注意力比较简单好实现，所以作者用了这个，但是为什么需要除以维度的平方根呢？因为如果维度大了以后，因为之后要做softmax，很有可能最后点积的值会很大，导致只有一个softmax后接近1，其他都接近0，值更加向两端靠拢，导致梯度非常小，train不起来。

Mask 是说，对于 qt，不去管 kt 以后的，qt 应该只能看到 kt 以前的 k1，k2 等等。具体的方法就是把 qt 和 kt+1 及之后的乘积换成一个很小的数（比如-1e10），最后 softmax就会等于 0。

Multihead-Head Attention 是说，作者发现，与其只做一次前面说的那种 Attention，不如把原来的 QKV 都各自投影（线性变换）到一堆低维向量，让这些低维向量做 Attention，做完以后拼接起来再做一个线性变换得到结果。一个原因是原来的 Attention 其实是没有什么参数能学习的，但是在 Multi-head Attention 中线性变换的参数是可以学习的，另外一个就是这样的 Multi-head 很像 CNN 的多卷积核，可以识别不同模式。此外就是因为残差连接要求输入输出的维度不变，这里QKV映射到低维就是 原维度除以 head 数目，比如论文中就是 512 维除以 8 个 head，实际注意力中计算qkv的维度是 64 维。

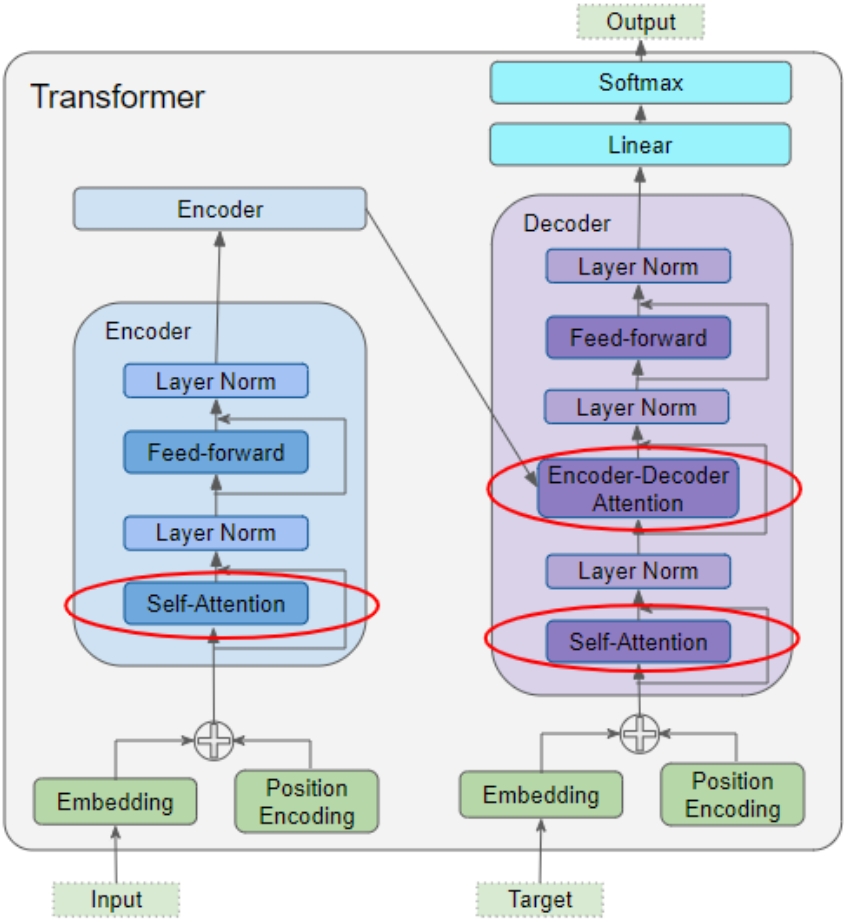


自注意力　本身就是说 QKV 都是一个东西，那么很明显，对于一个句子来说，序列中每个元素是一个 token，token自己有一个 Embedding向量。那么一句话做自注意力，就是一个单词，和其他所有单词算相似度，相似度越高，权重越大。最后这个权重与它自己（同时是value）计算加权和。这样就会出现权重其实还是集中在自身，因为自己和自己的相似度肯定是最高的，然后与这个单词相似的单词权重也会高一点。当然多头注意力机制下，有线性映射，情况会复杂很多。关于 self-attention 还有一个质量很高的英文材料这里推荐一下，我简单过了一下，非常清晰：

https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

然后对于 Transformer 架构下的三处 Attention 做分别介绍说明，以对整个模型流程有一个更好的了解。下面是架构图（其实和作者的图一样的，图源可见连接，by Ketan Doshi）



第一处 Attention 是 Self Attention 在 Encoder 中，输入就是上一层 encoder 的输出，是 self-attention ，所以 QKV 都是从输入进行线性变换得来。然后可以 Attend 输入的全部部分。

第二处看 Decoder 的最下面输入，也是 self-attention，但是注意这里是 masked self-attention，对于 qt 是看不到 kt 之后的 k 的，计算的结果会被直接设置为负无穷，然后权重就是0了。

第三处看 Encoder 指向 Decoder 的那个 Encoder-Decoder Attention。

Position-wise Feed Forward Network

Feed Forward Network 就是最朴素大家最开始学的全连接前馈神经网络，MLP。但是这里有一个 Position-wise，指的是这里的神经网络是将一个序列的每个元素（句子的每个单词）作为输入，且每次计算的神经网络是一样的（就是说拿同一个神经网络，权重不变，对每个单词做一次运算）。

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

看论文中的公式，这个 max 函数的部分其实就是一个 relu，里面是一个普通的权重加偏置，所以就是正常的一层神经网络加上relu激活。然后外面又是一层神经网络。所以其实就是单隐藏层的神经网络，另外这里，本来输入的是每个token 512 维的输入，W1 将其投影到了 2048 维的空间，然后由于残差需要输入输出维度一直， W2又将其投影回了 512 维。

这里与 RNN 做了一个对比，Transformer 与 RNN 都有 MLP 完成对语义空间的转换，但是区别在于如何传递序列信息，Transformer 使用 Attention 在第一步就将序列元素相互之间的信息提取出来放到了输出，

所以之后只要用一样的 MLP 也不需要其他数据，每个元素单独处理做语义空间转换。RNN 则是通过每次运算时拿前一个元素的输出的隐含信息来完成序列信息的抽取。

Embeddings and softmax

架构中有3处Embedding，编码器和解码器各自的输入需要，以及softmax前面的 Linear 层也需要一个 Embedding，三个 Embedding 是一样的权重。

然后就是在 Embedding 层，把权重乘了 根号d ，d是512。这里的操作是因为，Embedding 在学习的过程中，通常会把 L2-Norm 学成 1，而无关维度长度，如果维度长了，很明显部分元素相对来说就会比较小，而 Positional Encoding （后面会说）和维度 d 是无关的，所以出现不匹配，这里就要把权重乘 更号 d。

Positional Encoding

Attention 中，这些操作只检查向量之间的相似度，没有考虑他们的顺序信息。也就是说一句话打乱之后得到的结果是一样的。RNN 天然保留了顺序信息，所以之类

为什么使用自注意力

最容易注意的是论文中的 Table 1，其中的 Maximum Path Length 指的是一个点的信息要跳到另一个点要走多少步。

时间复杂度的话，Self-Attention 这里是 Query 和 Key 转置的矩阵乘法，两个矩阵大小都是 n*d，然后和 Value 矩阵相乘，Value 矩阵也是 n*d，最后的复杂度其实就是 O(n^2d) * 2 还是 O(n^2d)。RNN 是做 n 次运算，每次运算内部就是一个 d 维输入和 d 维参数。所以是 nd^2 。CNN中k是卷积核，一般3或5所以可以当成常数项。

PS：n 和 d 在当前的大部分模型中差不多一个数量集比较接近。

这里沐神做的评论是，看上去 Attention 可以对长数据首先并行度高算得快，其次可以看到全局数据，对全局表现好。但是实际上，Attention对于模型的假设要少很多，所以需要更多数据和更大的模型来训练。

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

Evaluation

这里因为 Transformer 做的机器翻译，没有做过学习，这里就不说太多了。

文中说 step time 其实就是一个 batch 的 time。

Optimizer 使用了 Adam，这里 β2 用 0.98 不太常见，学习率也是公式计算，有warm up，和步数衰减，总的还要乘模型维度的 -0.5 次方。也就是说模型要学习的向量越长的时候，学习率会降低。

正则使用了 Dropout，Dropout 率为 0.1 并不高，但是使用了大量的 Dropout 层，所以正则还是比较狠的。另一个正则是 label smoothing，就是说因为 softmax很难逼近正常标签中的 0 和 1，因为 1 需要无穷大的输入，才能输出为 1。所以就降低一点要求，比如 0.9，就容易达到很多。但是 Transformer 这篇文章比较狠，直接降到 0.1 。这里文中说到的 perplexity ，就是 loss 做指数，模型的不确信度。

论文讨论与总结

这里直接引用沐神的评价了。

文章简短，而且没有使用太多写作技巧。基本就是提出一个东西，和CNN，RNN比怎么样，最后的实验结果。但是好的文章应该是讲好一个故事，因为这篇文章提出来的东西比较多，所以篇幅受限也是一个原因。好的写法应该是讲好一个故事，把自己的提出的东西，不重要的部分放到附录，正文讲好故事，为什么要做这个事情，设计的理念是什么，对整个文章的思考是什么。

Transformer 后续的相关工作证明，其不只是在 机器翻译 这个任务上，在其他 NLP 任务上表现都很好。就像是 CNN 在计算机视觉上的突破，使得研究者不需要再像以前一样，对于每个任务都要去学习相关的预处理方法和模型，而是直接在 CNN 这个架构内就可以做出很好的成功。此外 Transformer 也不只是在 NLP，现在发现，也在图片，语音，Video 上取得了很好的进展。而这，和人类认知世界很像，人类就是通过多模态认知世界，图像，语音，文字等等。而Transformer能在这些模态都取得不错的效果，抽取出的特征能在同一个语义空间，对于训练多模态，更大的模型也有帮助。

再一个是说，题目说只需要 Attention 其实不完全正确，架构中的 MLP 和 残差连接，这些都是不可或缺的，只有 Attention 是训练不出来什么的。

还有就是前面所说过的，Transformer 对模型和数据的假设小，所以需要更大的模型更多的数据。

Finally, Li Mu is all you need.