Transformer面试题答案解析

公众号: NLP从入门到放弃

知乎: DASOU

NLP从入门到放弃

微信扫描二维码，关注我的公众号

1. 史上最全Transformer面试题

1. Transformer为何使用多头注意力机制？（为什么不使用一个头）
2. Transformer为什么Q和K使用不同的权重矩阵生成，为何不能使用同一个值进行自身的点乘？（注意和第一个问题的区别）
3. Transformer计算attention的时候为何选择点乘而不是加法？两者计算复杂度和效果上有什么区别?
4. 为什么在进行softmax之前需要对attention进行scaled（为什么除以dk的平方根），并使用公式推导进行讲解
5. 在计算attention score的时候如何对padding做mask操作?
6. 为什么在进行多头注意力的时候需要对每个head进行降维？（可以参考上面一个问题）
7. 大概讲一下Transformer的Encoder模块?

8. 为何在获取输入词向量之后需要对矩阵乘以embedding size的开方？意义是什么？
9. 简单介绍一下Transformer的位置编码？有什么意义和优缺点？
10. 你还了解哪些关于位置编码的技术，各自的优缺点是什么？
11. 简单讲一下Transformer中的残差结构以及意义。
12. 为什么transformer块使用LayerNorm而不是BatchNorm？LayerNorm 在Transformer的位置是哪里？
13. 简答讲一下BatchNorm技术，以及它的优缺点。
14. 简单描述一下Transformer中的前馈神经网络？使用了什么激活函数？相关优缺点？
15. Encoder端和Decoder端是如何进行交互的？（在这里可以问一下关于seq2seq的attention知识）
16. Decoder阶段的多头自注意力和encoder的多头自注意力有什么区别？（为什么需要decoder自注意力需要进行 sequence mask)
17. Transformer的并行化提现在哪个地方？Decoder端可以做并行化吗？
18. 简单描述一下wordpiece model 和 byte pair encoding，有实际应用过吗？
19. Transformer训练的时候学习率是如何设定的？Dropout是如何设定的，位置在哪里？Dropout 在测试的需要有什么需要注意的吗？
20. 引申一个关于bert问题，bert的mask为何不学习transformer在attention处进行屏蔽score的技巧？

---

2. 3分钟从零解读Transformer的Encoder

Transformer 分为两个部分，encoder 侧 和 decoder 侧。今天，我们聊一下 encoder 侧。这部分由 N 个完全相同的大模块堆叠而成（原论文N=6）。

这个结构怎么理解？这个构造就需要我们确保每一个模块的输入和输出维度是相同的，在实现代码的时候，我们只需要完成一个模块的代码的构造就可以。

注解：你可以把这个过程想象成 RNN 竖过来的一个流程，是不是就很好理解（当然这样想只是帮助你理解）。

其次对于这每一个大的模块，又分为两个模块，分别是多头注意力层和前馈神经网络层。进一步拆分，多头注意力层可以分为注意力层和 Add&Norm 层。前馈神经网络可以分为 Linear 层和 Add&Norm 层。

多头注意力层，核心点在于 Q/K/V 三个矩阵，其中 Q/K 矩阵生成权重矩阵(经由softmax)，随后和V矩阵得到加权和。

这个过程重复了 n_heads 次，这个 n_heads 代表的就是头的数目，这里需要注意的是我们需要确保 hidden_size/n_heads 需要为一个整数，不然代码会报错。

Add 代表一个残差结构。对于残差结构，可以使得信息前后向传播更加顺畅，缓解了梯度破碎问题。在 NLP 角度来看，残差结构一定程度上促进了 NLP 网络结构向窄而深的方向发展。

我们可以把 Transformer 和之前的模型对比一下，比如 RNN 模型，一般来说，我们会选择 单层RNN 或者 一个 Bilstm，对于这些比较传统的模型，只是在时间长度上进行了延展，并没有在深度上做的太深。

所以说，残差结构是有助于网路变深的。

顺便联想一下 Elmo，使用的是 双层双向lstm，训练起来已经非常慢了，所以对于RNN这种比较传统的模型，做深太难了，GNMT也是用了很多的 tricks 进行加速训练。

Norm 代表的是 Layer Normalization。为什么这里使用 Layer Normalization，而不是BN，这个后面有文章说，这里直白的回答就是，BN的效果差，所以不用。

随后多头注意力层的输出经过前馈神经网络。对前馈神经网络，比较简单，我们需要注意的是它分为两个 Linear 层，第一层的激活函数为 Relu，第二层没有使用激活函数。

最后我们谈一下整个encoder的输入和输出。

先说输入，分为两个部分：word embedding 和 position encoding

word embedding 没什么可说的，初始化后跟着训练或者使用word2vec这种已经有的看具体任务的效果。

position encoding 这里 transformer 使用的是 正余弦函数进行表达。其实这里进行初始化然后进行训练也是可以的，论文原作者的实验表明效果基本没区别。

对于 position encoding 表示的绝对位置，这点大家都没异议，那么 position encoding 究竟有没有表达相对位置信息，之后会有个文章专门讲讲这个知识点。

然后说一下 encoder的输出，感觉很少有人谈到这里。

encoder 的输出需要注意的细节点在于它需要和 decoder做交互，所以它的输出为 K/V 矩阵，记住这个细节点，**Q 矩阵来自decoder模块，K/V矩阵来自encoder**。

写到这里，我估摸这三分钟差不多能看完，现在没有留言功能，有问题大家在公众号对话框发送，我后台能看见。

能点个在看，老铁们！！鞠躬感谢！！

原版Transformer的位置编码究竟有没有包含相对位置信息

简单概述

Transformer 原版的位置编码也就是正余弦函数编码，表达的是绝对位置信息，同时包含相对位置信息。但是经过线性变化，相对位置信息消失。基于此，需要对位置编码进行优化。

正文

原版位置编码使用的是正余弦函数，通过三角函数，可以得出一个结论就是：PE_{pos+k}可以被PE_{pos}线性表示。

从这一点来说，原版位置编码可以反应一定的相对位置信息。

接下来，我们来看，经过注意力层，这个相对位置信息还在不在?

很简单，把词向量和位置向量作为输入，经过注意力层，然后因式分解，得到四个部分，我们重点关注包含两个不同位置编码的公式部分，形式如下：

$$PE_{pos}^{T}W_{q}^{T}W_{k}PE_{pos+k} \qquad (1)$$

我们想要证明，这个公式能不能反应相对位置信息。

为了解决这个问题，我们化繁为简，先从下面这个公式入手：

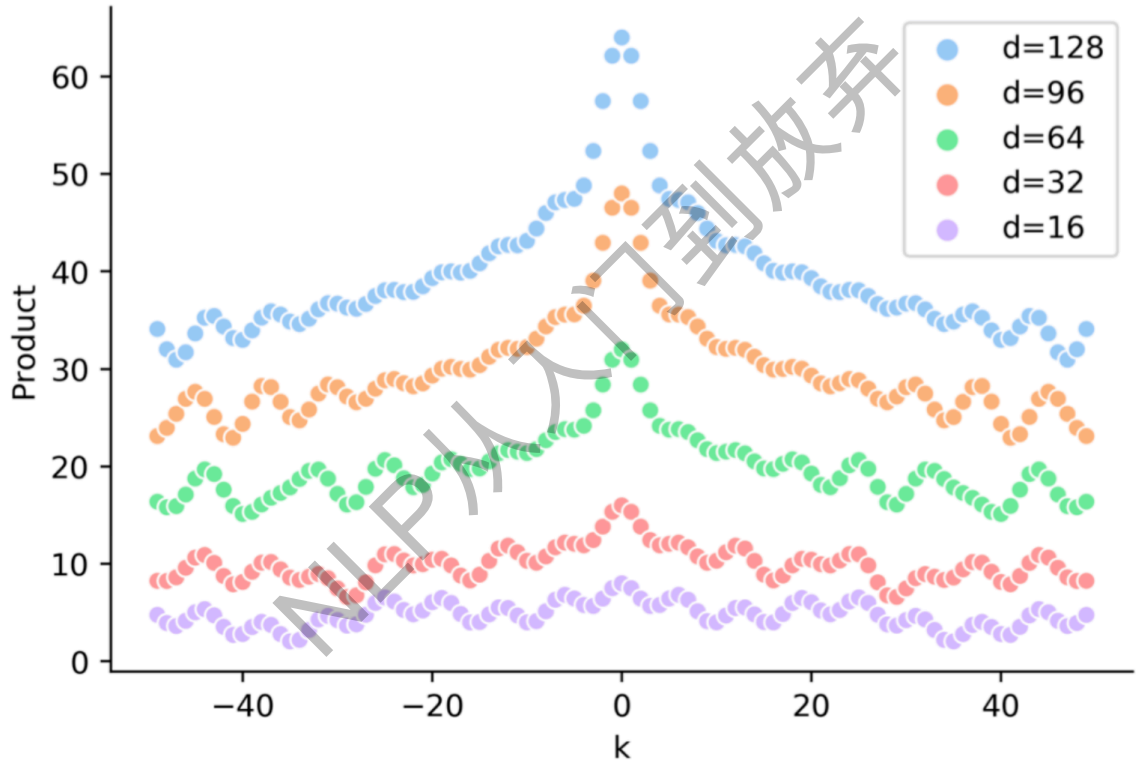
$$PE_{pos}^{T}PE_{pos+k} \qquad (2)$$

注意看公式(1)和公式(2)的区别，在中间多了两个矩阵相乘，这两个矩阵，是我们的Q/K矩阵，可以看做是一个线性变化，记住这个细节点。

经过公式推导，我们很容易知道公式(2)最后的结果只和两个位置的相对位置 k 相关，这个结果是包含相对位置信息。也就是说两个不同位置PE的点积的结果可以反映相对距离。

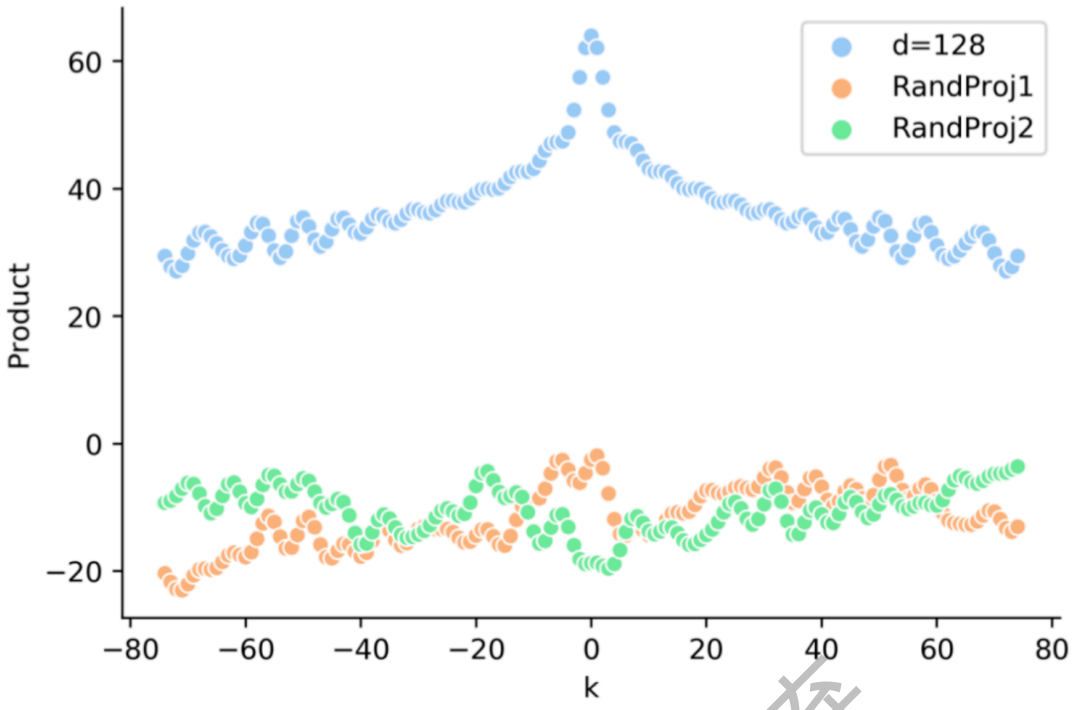
通过实验我们知道这个结果大小随着相对距离的增大而减小，值得注意的是它并不能反映相对位置的方向，因为他是一个对称的。

具体的我们可以看下面这个图：



很好，接下来，我们就是证明本来可以反映相对位置信息的公式(2)，在加上中间这个线性变化之后，相对位置信息还在不在。

直接看效果图：



这个图需要重点看的下面两个，也就是加了线性变化之后，变化趋势从最上面蓝色图标的线变成了下面两条线，也就是趋势已经完全没有了。

也就是说，实验结果显示，公式(1)的结果随着 k 的变化没有明显的趋势变化，也就是说相对位置信息消失了。

上面这些内容，估计5分钟左右吧，本来想加上相对位置编码，不过内容也挺多的，下回再发吧。

同学们，如果觉写的还行，给个在看。

---

4. 谈一下相对位置编码

经过线性变化之后，正余弦函数表示的相对位置信息消失，所以需要优化。

一般来讲，谈到优化，三种比较有名：RPR； Transformer-XL；complex embeddings；

我在这个文章简单讲一下RPR。

老样子，不涉及到公式推导，尽量把我的理解讲出来。

RPR思路

RPR思路很简单，原始正余弦函数，是在输入的的时候与词向量相加作为输入，在attention丢失相对位置信息，改进的话就是不在输入的时候进行位置编码，而是在attention中显示把相对位置信息加入进去。

如何理解相对位置

绝对位置编码是在每个位置都对应一个唯一的位置编码信息，RPR把这一部分去掉去学习一个相对位置编码。

首先我们需要知道相对位置是有方向的的。

举个例子：”我/爱/中国/共产党“

”我“对”爱“的相对位置就是 -1， ”中国“对”爱“的相对位置就是 1。

RPR修改思想

作者认为在相对位置小于4的时候，attention对相对位置比较敏感，在大于4之后，相对位置不敏感。所以窗口设置为4。

需要注意的是，窗口设为4，代表的当前位置左边4个，右边也有4个，再加上自己，就是一共9个位置，也就是：

$[i-4, i-3, i-2, i-1, i, i+1, i+2, i+3, i+4]$

当你的attention进行到哪个单词的时候，你的 i 就对应的是哪个位置。

还是上面那句话举例子。

如果此时的输入是“我”，那么用到的相对位置编码就是$[i, i+1, i+2, i+3]$

如果此时输入的是“爱”，那么这个时候用到的相对位置编码就是$[i-1, i, i+1, i+2]$

了解了这个，我们再谈一下这个相对位置信息是怎么显示加入进去的。

这个显示的加入分为两个部分。

第一个部分是在计算e_{ij}的时候，涉及到RPR的一个表征:a_{ij}^K，表示对 Q/K/V三者中的K做了修改。

第二个部分就是在计算z_i的时候，涉及到另一个RPR的表征:a_{ij}^V，表示对Q/K/V三者中的V做了修改。

两个部分的修改都是使用加法。

BN踩坑记--谈一下Batch Normalization的优缺点和适用场景

这个问题没有定论，很多人都在探索，所以只是聊一下我自己的理解，顺便为讲 layer-norm做个引子。

BN的理解重点在于它是针对整个Batch中的样本在同一维度特征在做处理。

在MLP中，比如我们有10行5列数据。5列代表特征，10行代表10个样本。是对第一个特征这一列（对应10个样本）做一次处理，第二个特征（同样是一列）做一次处理，依次类推。

在CNN中扩展，我们的数据是N·C·H·W。其中N为样本数量也就是batch_size，C为通道数，H为高，W为宽，BN保留C通道数，在N,H,W上做操作。比如说把第一个样本的第一个通道的数据，第二个样本第一个通道的数据.....第N个样本第一个通道的数据作为原始数据，处理得到相应的均值和方差。

BN有两个优点。

第一个就是可以解决内部协变量偏移，简单来说训练过程中，各层分布不同，增大了学习难度，BN缓解了这个问题。当然后来也有论文证明BN有作用和这个没关系，而是可以使损失平面更加的平滑，从而加快的收敛速度。

第二个优点就是缓解了梯度饱和问题（如果使用sigmoid激活函数的话），加快收敛。

BN的缺点：

第一个，batch_size较小的时候，效果差。这一点很容易理解。BN的过程，使用 整个batch中样本的均值和方差来模拟全部数据的均值和方差，在batch_size 较小的时候，效果肯定不好。

第二个缺点就是 BN 在RNN中效果比较差。这一点和第一点原因很类似，不过我单挑出来说。

首先我们要意识到一点，就是RNN的输入是长度是动态的，就是说每个样本的长度是不一样的。

举个最简单的例子，比如 batch_size 为10，也就是我有10个样本，其中9个样本长度为5，第10个样本长度为20。

那么问题来了，前五个单词的均值和方差都可以在这个batch中求出来从而模型真实均值和方差。但是第6个单词到底20个单词怎么办?

只用这一个样本进行模型的话，不就是回到了第一点，batch太小，导致效果很差。

第三个缺点就是在测试阶段的问题，分三部分说。

首先测试的时候，我们可以在队列里拉一个batch进去进行计算，但是也有情况是来一个必须尽快出来一个，也就是batch为1，这个时候均值和方差怎么办?

这个一般是在训练的时候就把均值和方差保存下来，测试的时候直接用就可以。那么选取效果好的均值和方差就是个问题。

其次在测试的时候，遇到一个样本长度为1000的样本，在训练的时候最大长度为600，那么后面400个单词的均值和方差在训练数据没碰到过，这个时候怎么办?

这个问题我们一般是在数据处理的时候就会做截断。

还有一个问题就是就是训练集和测试集的均值和方差相差比较大，那么训练集的均值和方差就不能很好的反应你测试数据特性，效果就回差。这个时候就和你的数据处理有关系了。

BN使用场景

对于使用场景来说，BN在MLP和CNN上使用的效果都比较好，在RNN这种动态文本模型上使用的比较差。至于为啥NLP领域BN效果会差，Layer norm 效果会好，下一个文章会详细聊聊我的理解。

---

6. NLP任务中-layer-norm比BatchNorm好在哪里

本文主要是讲一下，为什么NLP任务中，比如Transformer，使用LayerNorm而不是使用BatchNorm

这个问题其实很有意思，理解的最核心的点在于：为什么LayerNorm单独对一个样本的所有单词做缩放可以起到效果。

大家往下慢慢看，我说一下我自己的理解，欢迎大佬拍砖，如果觉得我说的还行，点个在看鼓励一下。

为啥BN在NLP中效果差

上一个文章有说 BN的使用场景，不适合 RNN这种动态文本模型，有一个原因是因为batch中的长度不一致，导致有的靠后面的特征的均值和方差不能估算。

这个问题其实不是个大问题，可以缓解。我们可以在数据处理的时候，使句子长度相近的在一个batch，就可以了。所以这不是为啥NLP不用BN的核心原因。

回忆一下上个文章中，BN在MLP中的应用。 BN是对每个特征在batch_size上求的均值和方差。记住，是每个特征。比如说身高，比如说体重等等。这些特征都有明确的含义。

但是我们想象一下，如果BN应用到NLP任务中，对应的是对什么做处理?

是对每一个单词!

也就是说，我现在的每一个单词是对应到了MLP中的每一个特征。

也就是默认了在同一个位置的单词对应的是同一种特征，比如:“我/爱/中国/共产党”和“今天/天气/真/不错”

如何使用BN，代表着认为 "我"和“今天”是对应的同一个维度特征，这样才可以去做BN。

大家想一下，这样做BN，会有效果吗?

不会有效果的，每个单词表达的特征是不一样的，所以按照位置对单词特征进行缩放，是违背直觉的。

layner-norm 的特点

layner-norm 的特点是什么？layner-norm 做的是针对每一个样本，做特征的缩放。换句话讲，保留了N维度，在C/H/W维度上做缩放。

也就是，它认为“我/爱/中国/共产党”这四个词在同一个特征之下，所以基于此而做归一化。

这样做，和BN的区别在于，一句话中的每个单词都可以归到一个名字叫做“语义信息”的一个特征中（我自己瞎起的名字，大家懂就好），也就是说，layner-norm也是在对同一个特征下的元素做归一化，只不过这里不再是对应N（或者说batch size），而是对应的文本长度。

上面这个解释，有一个细节点，就是，为什么每个单词都可以归到“语义信息”这个特征中。大家这么想，如果让你表达一个句子的语义信息，你怎么做?

最简单的方法就是词语向量的加权求和来表示句子向量，这一点没问题吧。（当然你也可以自己基于自己的任务去训练语义向量，这里只是说最直觉的办法）

上面这个方法就是出于每个单词都是语义信息的一部分这个insight。

引申-为啥BN在CNN可以而在NLP不可以

但是，我还想问一个问题，CNN中证明BN效果是很好的，NLP中的文本可以类比为图像，为什么BN在图像中效果好，在文本上效果差。

我是这样理解的。还是回到刚才，BN是对单词做缩放，在NLP中，单词由词向量来表达，本质上是对词向量进行缩放。词向量是什么？是我们学习出来的参数来表示词语语义的参数，不是真实存在的。

这就是NLP和图像的一个区别，图像的像素是真实存在的，像素中包含固有的信息。比如说，一张图像，最上面的一行像素，可以归为背景这个特征（这里只是为了理解，CNN做BN是基于整个feature map，而不是单独某一行像素）。

这个理解不确保正确，只是我自己的理解（记得是从一个知乎答案看到的，改天好好找一找）

简答说一下

写到这里，我写文章不是为了推导公式，因为这种推导文章太多了，而是想让大家看了我的文章之后再去看这些推导公式能够更加容易理解。

然后大家有问题的话，私信和我说，我也知道我自己写的哪里有问题，好改进。

点个在看再走呗，老弟

---

7. 谈一谈Decoder模块

本文主要是谈一些比较容易误解的细节点，说实话，把自己的理解用文字表达出来真是个细致活。

如果觉得对您有点帮助，帮忙点个在看或者赞。

一个小小的问题

我先说一个自己花了点时间才琢磨出来的东西，其实不难，就是当时没转过弯来。

我们都知道，decoder的交互层，Q矩阵来自本身，K/V矩阵来自整个Encoder层输出。

但是对于每个单词都会有一个encoder的输出，那么K/V矩阵是用的其中哪个输出计算过来的?

我这个问题的问法其实是错误的。

我当时的理解背景是认为这个交互的过程很类似seq2seq的attention，它一般是使用最后一个时刻的隐层输出作为context vector。

我基于此产生了上面这个问题，这个K/V矩阵应该由哪个位置单词（对比RNN就是哪个时刻的单词）的输出生成。

后来看了一下代码，才明白自己错在哪里?

K/V矩阵的计算不是来自于某一个单词的输出，而是所有单词的输出汇总计算K/V矩阵。这个过程和在Encoder中计算K/V矩阵是一样的，只不过放在了交互层，一时没想明白。

正文

与Encoder很类似，Decoder同样由完全相同的N个大模块堆叠而成，原论文中N为6。

每个大的模块分为三部分：多头注意力层，交互层，前馈神经层；每个层内部尾端都含有Add&Norm。

和Encoder重复的内容我就跳过了，之前讲过，没看过的同学可以去看那个文章。

多头自注意力层

首先谈一下多头自注意力层，这里需要注意的细节点是，需要对当前单词和之后的单词做mask。

为什么需要mask?

最常规的解释就是在预测阶段，你的模型看不见当前时刻的输出以及未来时刻单词。

这句话其实有点绕，如果读的不仔细会让人误解为mask的时候需要把当前时刻的单词也mask掉...(拖出去斩了吧)。

从代码角度讲，你只需要把当前时刻之后所有单词mask掉就好了。

我自己对这句话的理解是我们需要确保模型在训练和测试的时候没有GAP。

举个简单的例子来理解，如果做机器翻译，你需要翻译出来的句子是 "我/爱/吃/苹果"。

当前时刻是”爱“这个单词作为输入的一部分，另一部分是上一个时刻”我“作为输入的时候的输出值。

当然在机器翻译中，我们一般使用 teacher forcing加速收敛，所以这里就使用”我“作为当前时刻输入的另一个部分。

所以这个时候，输入就是”我“的编码信息和”爱“的编码信息（当然还有位置编码）。

我要预测的是”吃“这个单词。

如果我们没有mask，模型也是可以运行的，也就说此时”吃“和”苹果“两个词对”爱“这个时刻的输出是有贡献的。

那么问题来了，测试数据中你根本没有ground truth，你怎么办?

也就说，训练的时候，你的模型是基于知道这个时刻后面的单词进行的训练，但是测试的时候，做机器翻译，你不知道自己应该翻译出来什么东西。

这就是问题的核心。

你训练模型的时候，一部分精力花在了”吃“和”苹果“两个词上，这不就是无用功吗?

所以，确保模型在训练和测试的时候没有GAP，我们需要mask掉”吃“和”苹果“两个词。

交互模块

这一块需要注意的就是之前文章提到的，Q矩阵来自本身，K/V矩阵来自encoder的输出。

还有一个细节点是，K/V矩阵对应的是来自整个encoder的输出。

如果看transformer那个经典图的话，初期很容易理解为encoder和decode对应的每一层互相交互，这是不对的。

是整个输出与decoder做交互。

---

8. Transformer的并行化

本文主要谈一下关于 Transformer的并行化。文章比较短，适合大家碎片化阅读。

Decoder不用多说，没有并行，只能一个一个的解码，很类似于RNN，这个时刻的输入依赖于上一个时刻的输出。

对于Encoder侧：

首先，6个大的模块之间是串行的，一个模块计算的结果做为下一个模块的输入，互相之前有依赖关系。

从每个模块的角度来说，注意力层和前馈神经层这两个子模块单独来看都是可以并行的，不同单词之间是没有依赖关系的。

当然对于注意力层在做attention的时候会依赖别的时刻的输入，不过这个只需要在计算之前就可以提供。

然后注意力层和前馈神经层之间是串行，必须先完成注意力层计算再做前馈神经层。

有点绕，不知道有没有讲清楚。

简单讲，就是6个encoder之间是串行，每个encoder中的两个子模块之间是串行，子模块自身是可以并行的。

---

9. 谈一下 Transformer为何使用多头注意力机制?

答案解析参考这里：为什么Transformer 需要进行 Multi-head Attention?
https://www.zhihu.com/question/341222779

注解：简单回答就是，多头保证了transformer可以注意到不同子空间的信息，捕捉到更加丰富的特征信息。其实本质上是论文原作者发现这样效果确实好，我把作者的实验图发在下面：

	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

10. Transformer为什么Q和K使用不同的权重矩阵生成，为何不能使用同一个值进行自身的点乘?

答案解析参考这里：transformer中为什么使用不同的K 和 Q， 为什么不能使用同一个值？ - 知乎
https://www.zhihu.com/question/319339652

注解：简单回答就是，使用Q/K/V不相同可以保证在不同空间进行投影，增强了表达能力，提高了泛化能力。

11. Transformer计算attention的时候为何选择点乘而不是加法？两者计算复杂度和效果上有什么区别?

答案解析：为了计算更快。矩阵加法在加法这一块的计算量确实简单，但是作为一个整体计算attention的时候相当于一个隐层，整体计算量和点积相似。在效果上来说，从实验分析，两者的效果和dk相关，dk越大，加法的效果越显著。更具体的结果，大家可以看一下实验图(从莲子同学那里看到的，专门去看了一下论文)：

Attention	newstest2013	Params
Mul-128	22.03 ± 0.08 (22.14)	65.73M
Mul-256	22.33 ± 0.28 (22.64)	65.93M
Mul-512	21.78 ± 0.05 (21.83)	66.32M
Mul-1024	18.22 ± 0.03 (18.26)	67.11M
Add-128	22.23 ± 0.11 (22.38)	65.73M
Add-256	22.33 ± 0.04 (22.39)	65.93M
Add-512	**22.47** ± 0.27 (22.79)	66.33M
Add-1028	22.10 ± 0.18 (22.36)	67.11M
None-State	9.98 ± 0.28 (10.25)	64.23M
None-Input	11.57 ± 0.30 (11.85)	64.49M

Table 5: BLEU scores on newstest2013, varying the type of attention mechanism.

---

12. 为什么在进行softmax之前需要对attention进行scaled（为什么除以dk的平方根），并使用公式推导进行讲解

答案解析参考这里：transformer中的attention为什么scaled? - LinT的回答 - 知乎
https://www.zhihu.com/question/339723385/answer/782509914

注解：针对大佬回答的第二个问题，也就是方差的问题，我简单的写了一个代码验证了一下，不愿意看公式推导的同学直接看代码结果就可以。代码如下:

```
import numpy as np
arr1=np.random.normal(size=(3,1000))
arr2=np.random.normal(size=(3,1000))
result=np.dot(arr1.T,arr2)
arr_var=np.var(result)
print(arr_var) #result: 2.9 (基本上就是3，和就是我们设定的维度)
```

13. 计算attention score的时候如何对padding做mask操作?

答案解析：padding位置置为负无穷(一般来说-1000就可以)。对于这一点，涉及到batch_size之类的，具体的大家可以看一下抱抱脸实现的源代码，点击在这里

这个是最新版，比较老版本的实现地址我也罗列一下，应该没啥区别，我没细看，一直用的老版本的，点击这里

系列总结

整个Transformer这一块基本就是讲完了，基本上可以解决之前那个关于transformer面试题百分之八十的题目。

至于剩下的题目会放在之后别的模块去讲，比如 wordpiece model 会在总结机器翻译知识点的时候写一下，然后 GPT 会在总结词向量知识点的时候写一下。

欢迎大家关注微信公众号: NLP从入门到放弃