1. Transformer常见问题★

- https://mp.weixin.qq.com/s?__biz=MjM5ODkzMzMwMQ==&mid=2650412561&idx=2&sn=ef7a88ca7acfb4d666d51f1c0a7cdb7f&chksm=becd904b89ba195d9c07e5aca6fcf483cce3aee0378049d4abb977d5ce2a98095efc963151a2&scene=0&xtrack=1&key=9291a57e0a5baeebbaf718932e83da5ee54d07d88fae8558c9e5dbafbef324e337ce72a4d9dea2ff35af7fa44dd4b08cf6e034e8764cdf8bac6932ec6617a6aaca6450f685df2182adaebaf975b80322&ascene=14&uin=MjUwMzAyMzA2Mg%3D%3D&devicetype=Windows+10&version=62070158&lang=zh_CN&exportkey=AW5Q0tRPBzJeksxeUx0Tgl8%3D&pass_ticket=hfc4Vp3922C3naIdEO223HgA3viz%2Betribul7sgjA3lyyLBZFE%2FSmZ3Jf1%2BwTH0a <https://mp.weixin.qq.com/s?__biz=MjM5ODkzMzMwMQ==&mid=2650412561&idx=2&sn=ef7a88ca7acfb4d666d51f1c0a7cdb7f&chksm=becd904b89ba195d9c07e5aca6fcf483cce3aee0378049d4abb977d5ce2a98095efc963151a2&scene=0&xtrack=1&key=9291a57e0a5baeebbaf718932e83da5ee54d07d88fae8558c9e5dbafbef324e337ce72a4d9dea2ff35af7fa44dd4b08cf6e034e8764cdf8bac6932ec6617a6aaca6450f685df2182adaebaf975b80322&ascene=14&uin=MjUwMzAyMzA2Mg%3D%3D&devicetype=Windows+10&version=62070158&lang=zh_CN&exportkey=AW5Q0tRPBzJeksxeUx0Tgl8%3D&pass_ticket=hfc4Vp3922C3naIdEO223HgA3viz%2Betribul7sgjA3lyyLBZFE%2FSmZ3Jf1%2BwTH0a>
- https://mp.weixin.qq.com/s/lUqpCae3TPkZlgT7gUatpg <https://mp.weixin.qq.com/s/lUqpCae3TPkZlgT7gUatpg>
- https://blog.csdn.net/zandaoguang/article/details/115713550 <https://blog.csdn.net/zandaoguang/article/details/115713550>

<https://mp.weixin.qq.com/s?__biz=MjM5ODkzMzMwMQ==&mid=2650412561&idx=2&sn=ef7a88ca7acfb4d666d51f1c0a7cdb7f&chksm=becd904b89ba195d9c07e5aca6fcf483cce3aee0378049d4abb977d5ce2a98095efc963151a2&scene=0&xtrack=1&key=9291a57e0a5baeebbaf718932e83da5ee54d07d88fae8558c9e5dbafbef324e337ce72a4d9dea2ff35af7fa44dd4b08cf6e034e8764cdf8bac6932ec6617a6aaca6450f685df2182adaebaf975b80322&ascene=14&uin=MjUwMzAyMzA2Mg%3D%3D&devicetype=Windows+10&version=62070158&lang=zh_CN&exportkey=AW5Q0tRPBzJeksxeUx0Tgl8%3D&pass_ticket=hfc4Vp3922C3naIdEO223HgA3viz%2Betribul7sgjA3lyyLBZFE%2FSmZ3Jf1%2BwTH0a>

「1.3.2 多头 Encoder-Decoder attention 交互模块」

多头 Encoder-Decoder attention 交互模块的形式与多头 self-attention 模块一致，唯一不同的是其Q,K,V矩阵的来源，其Q矩阵来源于下面子模块的输出(对应到图中即为 masked 多头 self-attention 模块经过 Add & Norm 后的输出)，而K,V矩阵则来源于整个 Encoder 端的输出，仔细想想其实可以发现，这里的交互模块就跟 seq2seq with attention 中的机制一样，目的就在于让 Decoder 端的单词(token)给予 Encoder 端对应的单词(token)“更多的关注(attention weight)”

「1.4.1 Add & Norm 模块」

Add & Norm 模块接在 Encoder 端和 Decoder 端每个子模块的后面，其中 Add 表示残差连接，Norm 表示 LayerNorm，残差连接来源于论文Deep Residual Learning for Image Recognition[1]，LayerNorm 来源于论文Layer Normalization[2]，因此 Encoder 端和 Decoder 端每个子模块实际的输出为：LayerNorm $(x+\text{Sublayer}(x))$，其中Sublayer $(x))$为子模块的输出。

具体做法是使用不同频率的正弦和余弦函数，公式如下：

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

其中pos为位置，i为维度，之所以选择这个函数，是因为任意位置PE_{pos+k}可以表示为PE_{pos}的线性函数，这个主要是三角函数的特性：

$$\sin(\alpha+\beta)=\sin(\alpha)\cos(\beta)+\cos(\alpha)\sin(\beta)$$
$$\cos(\alpha+\beta)=\cos(\alpha)\cos(\beta)-\sin(\alpha)\sin(\beta)$$

需要注意的是，Transformer 中的 Positional Encoding 不是通过网络学习得来的，而是直接通过上述公式计算而来的，论文中也实验了利用网络学习 Positional Encoding，发现结果与上述基本一致，但是论文中选择了正弦和余弦函数版本，**「因为三角公式不受序列长度的限制，也就是可以对 比所遇到序列的更长的序列 进行表示。」**

3.1 self-attention是什么?

「self-attention」，也叫 「intra-attention」，是一种通过自身和自身相关联的 attention 机制，从而得到一个更好的 representation 来表达自身，self-attention 可以看成一般 attention 的一种特殊情况。在 self-attention 中，$Q=K=V$，序列中的每个单词(token)和该序列中其余单词(token)进行 attention 计算。self-attention 的特点在于**「无视词(token)之间的距离直接计算依赖关系，从而能够学习到序列的内部结构」**，实现起来也比较简单，值得注意的是，在后续一些论文中，self-attention 可以当成一个层和 RNN，CNN 等配合使用，并且成功应用到其他 NLP 任务。

很明显，引入 Self Attention 后会更容易捕获句子中长距离的相互依赖的特征，因为如果是 RNN 或者 LSTM，需要依次序序列计算，对于远距离的相互依赖的特征，要经过若干时间步步骤的信息累积才能将两者联系起来，而距离越远，有效捕获的可能性越小。

4.1 Why Multi-head Attention

原论文中说到进行 Multi-head Attention 的原因是将模型分为多个头，形成多个子空间，可以让模型去关注不同方面的信息，最后再将各个方面的信息综合起来。其实直观上也可以想到，如果自己设计这样的一个模型，必然也不会只做一次 attention，多次 attention 综合的结果至少能够起到增强模型的作用，也可以类比 CNN 中同时使用**「多个卷积核」**的作用，直观上讲，多头的注意力**「有助于网络捕捉到更丰富的特征/信息」**。

5.Transformer 相比于 RNN/LSTM，有什么优势？为什么？

5.1 RNN 系列的模型，并行计算能力很差

RNN 系列的模型T时刻隐层状态的计算，依赖两个输入，一个是T时刻的句子输入单词X_t，另一个是$T-1$时刻的隐层状态的输出S_{t-1}，这是最能体现 RNN 本质特征的一点，RNN 的历史信息是通过这个信息传输渠道往后传输的。而 RNN 并行计算的问题就出在这里，因为t时刻的计算依赖$t-1$时刻的隐层计算结果，而$t-1$时刻的计算依赖$t-2$时刻的隐层计算结果，如此下去就形成了所谓的序列依赖关系。

5.2 Transformer 的特征抽取能力比 RNN 系列的模型要好

上述结论是通过一些主流的实验来说明的，并不是严格的理论证明，具体实验对比可以参见：

放弃幻想，全面拥抱 Transformer：自然语言处理三大特征抽取器（CNN/RNN/TF）比较[10]

但是值得注意的是，并不是说 Transformer 就能够完全替代 RNN 系列的模型了，任何模型都有其适用范围，同样的，RNN 系列模型在很多任务上还是首选，熟悉各种模型的内部原理，知其然且知其所以然，才能遇到新任务时，快速分析这时候该用什么样的模型，该怎么做好。

6.1 训练

Transformer 训练过程与 seq2seq 类似，首先 Encoder 端得到输入的 encoding 表示，并将其输入到 Decoder 端做交互式 attention，之后在 Decoder 端接收其相应的输入(见 1 中有详细分析)，经过多头 self-attention 模块之后，结合 Encoder 端的输出，再经过 FFN，得到 Decoder 端的输出之后，最后经过一个线性全连接层，就可以通过 softmax 来预测下一个单词(token)，然后根据 softmax 多分类的损失函数，将 loss 反向传播即可，所以从整体上来说，Transformer 训练过程就相当于一个有监督的多分类问题。

- 需要注意的是，「**Encoder 端可以并行计算，一次性将输入序列全部 encoding 出来，但 Decoder 端不是一次性把所有单词(token)预测出来的，而是像 seq2seq 一样一个接着一个预测出来的。**」

8.为什么说 Transformer 可以代替 seq2seq?

这里用代替这个词略显不妥当，seq2seq 虽已老，但始终还是有其用武之地，seq2seq 最大的问题在于「将 Encoder 端的所有信息压缩到一个固定长度的向量中」，并将其作为 Decoder 端首个隐藏状态的输入，来预测 Decoder 端第一个单词(token)的隐藏状态。在输入序列比较长的时候，这样做显然会损失 Encoder 端的很多信息，而且这样一股脑的把该固定向量送入 Decoder 端，Decoder 端不能够关注到其想要关注的信息。

上述两点都是 seq2seq 模型的缺点，后续论文对这两点有所改进，如著名的Neural Machine Translation by Jointly Learning to Align and Translate[11]，虽然确确实实对 seq2seq 模型有了实质性的改进，但是由于主体模型仍然为 RNN(LSTM)系列的模型，因此模型的并行能力还是受限，而 transformer 不但对 seq2seq 模型这两点缺点有了实质性的改进(多头交互式 attention 模块)，而且还引入了 self-attention 模块让源序列和目标序列首先“自关联”起来，这样的话，源序列和目标序列自身的 embedding 表示所蕴含的信息更加丰富，而且后续的 FFN 层也增强了模型的表达能力(ACL 2018 会议上有论文对 Self-Attention 和 FFN 等模块都有实验分析，见论文：How Much Attention Do You Need?A Granular Analysis of Neural Machine Translation Architectures[12])，并且 Transformer 并行计算的能力是远远超过 seq2seq 系列的模型，因此我认为这是 transformer 优于 seq2seq 模型的地方。

编码器-解码器模型虽然非常经典，但是局限性也很大。最大的局限性是编码器和解码器之间唯一的联系是一个固定长度的语义向量context vector。也就是说，编码器要将整个序列的信息压缩成一个固定长度的向量。它存在三个弊端：

（1）对于编码器来说，语义向量context vector可能无法完全表示整个序列的信息。
（2）对于编码器来说，先输入到网络的序列携带的信息可能会被后输入的序列覆盖掉，输入的序列越长，这种现象就越严重。
（3）对于解码器来说，在解码的时候，对于输入的每个单词的权重是不一致的。

以上三个弊端使得在解码的时候解码器一开始可能就没有获得输入序列足够多的信息，自然解码的准确率也就不高。所以，在NMT（Neural Machine Translation，神经机器翻译）任务上，还添加了attention的机制。

10.Transformer 如何并行化的?

Transformer 的并行化我认为主要体现在 self-attention 模块，在 Encoder 端 Transformer 可以并行处理整个序列，并得到整个输入序列经过 Encoder 端的输出，在 self-attention 模块，对于某个序列$x_1, x_2, \ldots, x_n$，self-attention 模块可以直接计算$\boldsymbol{x}_i, \boldsymbol{x}_j$的点乘结果，而 RNN 系列的模型就必须按照顺序从x_1计算到x_n。

11.self-attention 公式中的归一化有什么作用?

首先说明做归一化的原因，随着d_k的增大，$q \cdot k$点积后的结果也随之增大，这样会将 softmax 函数推入梯度非常小的区域，使得收敛困难(可能出现梯度消失的情况)

为了说明点积变大的原因，假设q和k的分量是具有均值 0 和方差 1 的独立随机变量，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，因此为了抵消这种影响，我们将点积缩放$\frac{1}{\sqrt{d_k}}$，对于更详细的分析，参见(有空再来总结，哈哈~): transformer 中的 attention 为什么 scaled?[13]

mask表示掩码，它对某些值进行掩盖，使其在参数更新时不产生效果。 Transformer 模型里面涉及两种 mask，分别是padding mask 和 sequence mask。其中，padding mask 在所有的 scaled dot-product attention 里面都需要用到，而 sequence mask 只有在 Decoder 的 self-attention 里面用到。

（1）padding mask

什么是 padding mask 呢？因为每个批次输入序列长度是不一样的也就是说，我们要对输入序列进行对齐。具体来说，就是给在较短的序列后面填充 0。但是如果输入的序列太长，则是截取左边的内容，把多余的直接舍弃。因为这些填充的位置，其实是没什么意义的，所以我们的attention机制不应该把注意力放在这些位置上，所以我们需要进行一些处理。

具体的做法是，把这些位置的值加上一个非常大的负数(负无穷)，这样的话，经过softmax，这些位置的概率就会接近0！

而我们的 padding mask 实际上是一个张量，每个值都是一个Boolean，值为 false 的地方就是我们要进行处理的地方。

（2）Sequence mask

sequence mask 是为了使得 decoder 不能看见未来的信息。也就是对于一个序列，在 time_step 为 t 的时刻，我们的解码输出应该只能依赖于 t 时刻之前的输出，而不能依赖 t 之后的输出。因此我们需要想一个办法，把 t 之后的信息给隐藏起来。

那么具体怎么做呢？也很简单：产生一个上三角矩阵，上三角的值全为0。把这个矩阵作用在每一个序列上，就可以达到我们的目的。

- 对于 decoder 的 self-attention，里面使用到的 scaled dot-product attention，同时需要 padding mask 和 sequence mask 作为 attn_mask，具体实现就是两个mask相加作为 attn_mask。
- 其他情况，attn_mask 一律等于 padding mask。

现在，因为模型每步只产生一组输出，假设模型选择最高概率，扔掉其他的部分，这是种产生预测结果的方法，叫做greedy 解码。另外一种方法是beam search，每一步仅保留最头部高概率的两个输出，根据这俩输出再预测下一步，再保留头部高概率的两个输出，重复直到预测结束。top_beams是超参可试验调整。
