3.Transformer

seq2seq传统缺点:seq2seq使用循环网络使用固有的顺序运行阻碍样本并行化，这在更长的序列长度上至关重要，因为有限的内存限制样本大小。

transformer的特征：

1. Self Attention、MultiHead Attention、Cross Attention、Masked Attention上节已讲过，
2. Positional Enconding:通过对位置进行embedding，提高模型序列特征的提取能力。
3. Residuals:残差模块化，防止模型退化。
4. Layer Norm：正则化模块，加快训练速度，防止模型过拟合。
5. FFN：两层全连接,中间串联ReLU函数。

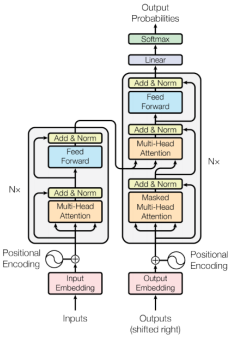


Figure 1: The Transformer - model architecture.

1.Encoder&Decoder

Encoder 端由 N(原论文中 N=6)个相同的大模块堆叠而成，其中每个大模块又由两个子模块构成，这两个子模块分别为多头 self-attention 模块、前馈神经网络模块；需要注意的是，Encoder 端每个大模块接收的输入是不一样的，第一个大模块(最底下的那个)接收的输入是源序列的 embedding(embedding 可以通过 word2vec 预训练得来)，其余大模块接收的是其前一个大模块的输出，最后一个模块的输出作为整个 Encoder 端的输出。

Decoder 端同样由 N(原论文中 N=6)个相同的大模块堆叠而成，其中每个大模块则由三个子模块构成，这三个子模块分别为多头 self-attention 模块、多头 Encoder-Decoder attention 、前馈神经网络模块；同样需要注意的是，Decoder 端每个大模块接收的输入也是不一样的，其中**第一个大模块(最底下的那个)训练时和测试时的接收的输入是不一样的**(也就是模型总览图示中的"shifted right"，**5.1会解释**)，其余大模块接收的是同样是其前一个大模块的输出，最后一个模块的输出作为整个 Decoder 端的输出。

2.Attention(上节中已解释)

3.Add&Norm

该结构接在Decoder和Encoder每个子模块后面，其中Add表示残差连接，Norm表示Layer Norm，因此每个子模块的输出可以表示为：$output_{sub} = LN(x + submodel(x))$

Layer Norm在每个子模块的最后出现，Layer Norm是一个通用的技术，其本质是规范优化空间，加速收敛。当我们使用梯度下降法做优化时，随着网络深度的增加，和迭代的进行，内部参数会发生偏移，数据在经过激活层的时候（如果激活函数是非饱和函数），容易陷入非饱和区，降低模型收敛速度。

Add对每个子模块进行残差连接，其目的是防止因深度过程而导致模型退化。

4.Position Encoding

位置编码层只在encoder端和decoder端的embedding之后，第一个block之前出现，它非常重要，没有这部分，Transformer模型就无法用。位置编码是Transformer框架中特有的组成部分，补充了Attention机制本身不能捕捉序列信息的缺陷，其构造方式如下：

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

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：

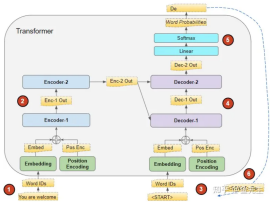
$$\sin(\alpha+\beta) = \sin(\alpha)\cos(\beta) + \cos(\alpha)\sin(\beta)$$
$$\cos(\alpha+\beta) = \cos(\alpha)\cos(\beta) - \sin(\alpha)\sin(\beta)$$

之所以选择这种构造方式：1.每个位置向量唯一（每个函数的频率足够小）。2.位置向量的值是有界的，且位于连续空间中。模型在处理位置向量时更容易泛化，即更好处理长度和训练数据分布不一致的序列（sin函数本身的性质）。3.不同的位置向量是可以通过线性转换得到的，这样，我们不仅能表示一个token的绝对位置，还可以表示一个token的相对位置。

5.相关问题思考

5.1 Decoder端的具体输入?

Decoder端的输入在每个大模块中都是不一样的，**第一个大模块**尤为特殊，它在训练过程中，接收的是由目标序列右移一位的带有位置编码的嵌入，而它在推理过程中，分不同时间步进行串行输入，在第一时间步，使用一个只有句首符号的空序列来代替训练过程中使用的目标序列。空序列转换为嵌入带有位置编码的嵌入，并被送入解码器，然后通过所有解码器的解码，产生第一个序列的最终解码信息，输出层预测出第一个目标单词，解码器将预测出的第一个目标单词放入第二时间步，第二时间步第一个大模块的输入序列包含句首符号产生的 token 和第一个时间步产生的目标单词.....以后每个时间步以此类推，直至结束。**其他大模块**的输入为上一个大模块的输出。



5.2 Transformer 相比于 RNN/LSTM，有什么优势？为什么？

1. **RNN系列网络，并行计算能力差。**
   RNN系列网络每个时刻的隐藏层计算依靠于两个输入，一个是当前时刻的单词输入，另一个是上一个时刻的隐藏层输出，RNN也是通过这样对序列进行建模的。但是这样同样会带来对计算上的影响，因为每个时刻的输入总是依赖于上一个时刻的输出，这样就形成了序列间的依赖关系，从而影响计算。
2. **transfomer的特征抽取能力要比RNN系列要好。**
   RNN系列具有长时依赖效应，如果输入的序列过长，RNN系列网络对于更远时刻的特征信息会出现更多的损失，提取的特征效果明显不如transformer，因为transformer的self-attentoin机制是通过将各个时刻的单词向量加权求和获得全文的语义特征，然后通过获得的全文语义特征来获得更能代表当前时刻当前单词的特征向量，不存在长距离信息损失的问题。

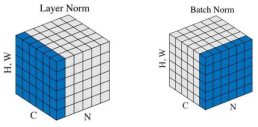
5.3 Transformer优于Seq2Seq的地方？

1. 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。
2. seq2seq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。
3. 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+positional encoding)，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，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。

5.4 self-attention 公式中的归一化有什么作用？

q·k的大小与向量长度d_k呈正相关，假设q和k的分量服从均值为0，方差为1的独立正态分布，那么q·k就服从均值为0，方差为d_k的分布，为了防止这种波动性增大，所以我们使用$\frac{q \cdot k}{\sqrt{d_k}}$来矫正方差。

5.5 为什么Transformer/RNN中采用LayerNorm，而不是采用BatchNorm?



上图左边是LayerNorm，右边是BatchNorm，在文本任务中，容易出现每个文本的长度不一致的问题，如果我们采用BN的话，我们需要计算每个序列的均值和方差，这样会导致，一方面如果遇到测试集中存在序列长度大于所有训练集文本的序列长度的文本的话，训练得到的参数就不能够对该文本多出来的序列进行匹配；另一方面在长短不一的情况下，文本中某些位置没有足够的batchsize的数据，使得计算出来的均值和方差的偏差较大。