背景

循环神经网络（RNN）、长短期记忆人工神经网络以及 Transformer 都是自然语言处理领域的相关算法模型。其中 RNN 能将每个单词同它所处的语句中上下文部分都联系起来，不仅仅是对单独的词做处理，而是将其所处的语境也考虑进来，从而让整个模型更好地去理解自然语言。下图就是其循环核提取特征的过程。

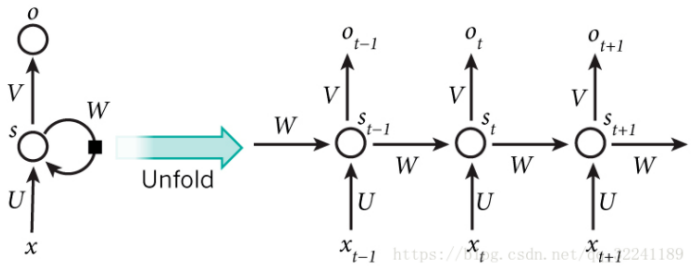

图 1 RNN

其中$\{\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}\}$是参数矩阵，输入数据 $\boldsymbol{x}$ 是时间序列$\{x_1,\cdots,x_{t-1},x_t,x_{t+1}\}$，状态信息 $\boldsymbol{s}$ 储存了不同时刻的状态$\{s_1,\cdots,s_{t-1},s_t,s_{t+1}\}$，$\{o_1,\cdots,o_{t-1},o_t,o_{t+1}\}$为输出特征。其中：

$$s_t = tanh(x_t * U + s_{t-1} + B_s) \tag{1}$$

$$o_t = softmax(s_t * V + B_o) \tag{2}$$

通过训练对参数进行优化。但是，这有一个很明显的缺陷，当数据变得较长的时候，会导致序列展开过长，在反向传播时会出现由于长期依赖问题导致的梯度消失现象。后来也提出了 LSTM 来解决 RNN 中存在的问题，但是这两种网络都存在同样的问题就是，无法并行化处理，必须先处理上一时刻的数据才能处理当前时刻的数据，这就导致了训练效率低。

Transformer 提出了一种注意力机制，来对输入输出数据之间的依赖关系进行建模分析。

方法

Positional Encoding

从结构框图中可以看出，其对输入做了 input Embedding 将输入数据映射为

d_{model} 维的向量后，又进行了 Positional Encoding，即给每个向量添加位置编码（因为单词放置的顺序不同，句子表达的意思不同）

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

位置编码计算公式如上所示，通过使用不同频率的正弦和余弦函数进行编码，然后再和对应位置的词向量进行相加。式中 pos 便是单词在句子中的绝对位置，d_{model} 表示词向量的维度，$2i$ 和 $2i+1$ 表示奇偶性，i 表示词向量的第 i 维。

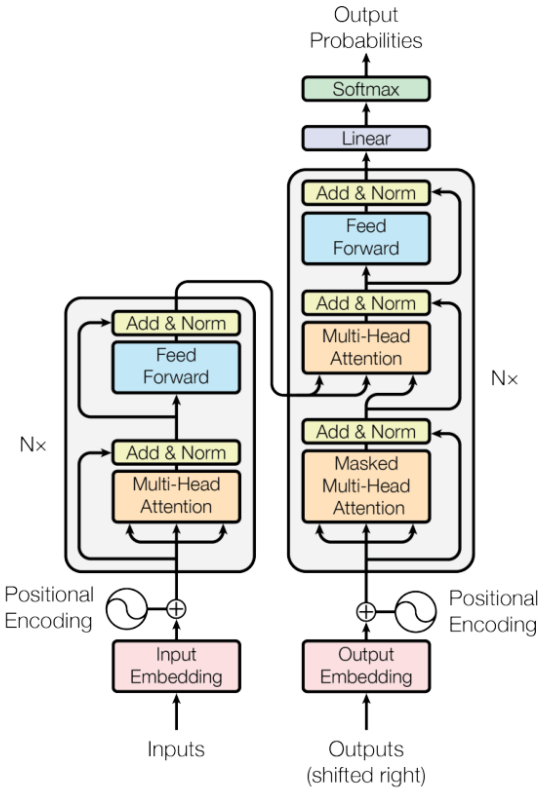

图 2 Transformer 结构框图

Q：为什么通过三角函数就可以引入位置信息了？

A：词序信息的表示方法很丰富，但究其根本，需要的是对不同维度的不同位置生成合理的数值表示。这里的合理，理解为不同位置的同一维度的位置向量之间，含有相对位置信息，而相对位置信息可以通过函数的周期性实现。例如，求 $pos+k$ 位置的位置向量 $PE_{(pos+k,2i)}$ 和 $PE_{(pos+k,2i+1)}$：

$$\begin{cases} PE(pos+k,2i) = PE(pos,2i) \times PE(k,2i+1) + PE(pos,2i+1) \times PE(k,2i) \\ PE(pos+k,2i+1) = PE(pos,2i+1) \times PE(k,2i+1) - PE(pos,2i) \\ \qquad\qquad\qquad\qquad \times PE(k,2i) \end{cases}$$

通过上式可以看出，对于 *pos*+*k* 位置的位置向量某一维 2*i* 或 2*i*+1 而言，可以表示为，*pos* 位置与 *k* 位置的位置向量第 2*i* 维与第 2*i*+1 维的线性组合，这样的线性组合意味着位置向量中蕴含了相对位置信息。

（参考文章 https://www.zhihu.com/question/347678607）

Self-Attention

Transformer 是用 Self-Attention Layer 来替代 RNN 的 sequence-to-sequence 模型。

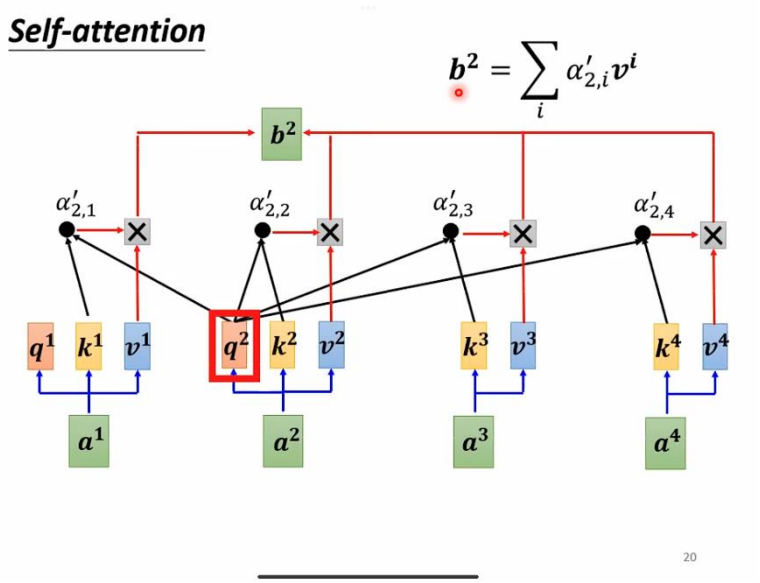

图 3 Self-Attention 结构示意图

假设输入时序向量 $\boldsymbol{x}=\{x_1,x_2,x_3,x_4\}$经过 input Embedding 映射到高纬度向量 $\boldsymbol{a}=\{a_1,a_2,a_3,a_4\}$，然后向量 $\boldsymbol{a}$ 通过共享的参数矩阵$\{\mathbf{W_q},\mathbf{W_k},\mathbf{W_v}\}$生成各自的 $\{q^i,k^i,v^i\}$，其中 *q* 表示 query，用来匹配每一个 key，*k* 表示 key，用来被 query 匹配的，*v* 表示从 *a* 中提取出来的信息。

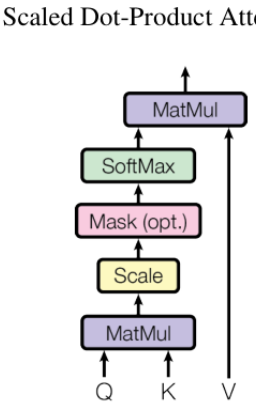

图 4 Scale Dot-Product Attention

上述所有的$\{q^i,k^i,v^i\}$集合起来构成了下图中的{Q,K,V}，随后 Q 和 K 再进行点乘操作。如图 3 所示，将q^2与每一个k^i进行点乘得到$\{\alpha_{2,1},\alpha_{2,2},\alpha_{2,3},\alpha_{2,4}\}$，由于点乘后的数值很大，经过 Softmax 后梯度变得很小，所以将点乘后的结果再进行一个 Scale 操作，即除以 k 向量维度的算术平方根$\sqrt{d_k}$，再通过 Softmax 后得到$\{\alpha'_{2,1},\alpha'_{2,2},\alpha'_{2,3},\alpha'_{2,4}\}$，该值即表示对于每一个提取到的特征信息$v^i$的权重大小，所以再将得到的$\alpha'_{2,i}$和对应的$v^i$相乘再相加得到$b_2$。其余$b_i$向量的计算原理相同。由此就可以得到时序向量中每个词的重要程度，即实现了下式的功能：

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

和 RNN 相比最大的优势就是以上所有操作可以并行化处理，提高了计算效率。

Multi-Head Attention

自注意力机制的缺陷是在对当前位置进行编码时，会过度将注意力集中于自身位置，多头注意力相当于多个不同的自注意力的集成，扩展了模型专注于不同位置的能力，将模型在不同的表示子空间里学习到相关的信息进行融合，这样每个头可能关注输入的不同部分。

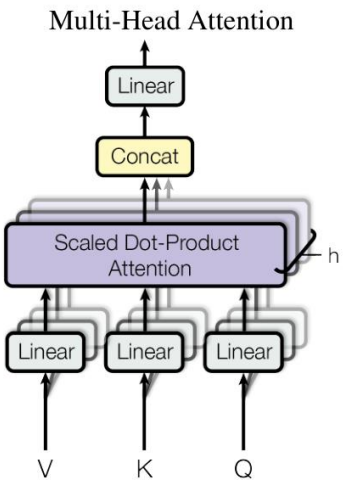

图 5 多头注意力机制

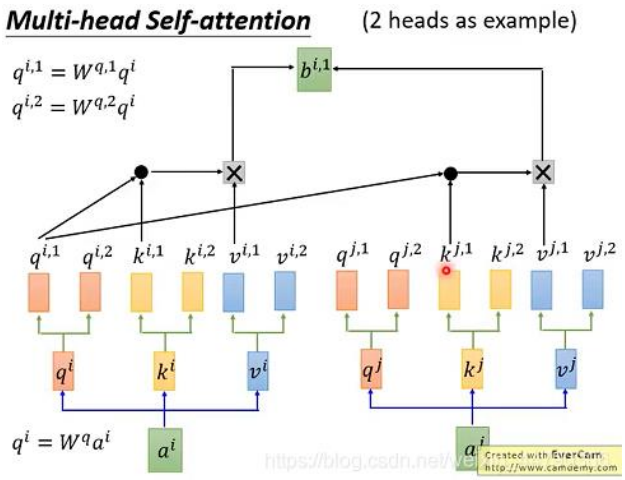

图 6 多头注意力机制说明

假设分为两个注意力头，那么将原来的 $\{q^i,k^i,v^i\}$ 分为 $\{q^{i,1},q^{i,2};k^{i,1},k^{i,2};v^{i,1},v^{i,2}\}$，每个头的 q 只能和对应头的 k 进行运算，比如 $q^{i,1}$ 只能与 $k^{i,1}$ 进行计算。最后如图 5 所示，将多头结果 $b^{i,1}$ 和 $b^{i,2}$ 进行 concat 拼接，再进行线性运算得到 b_1 和 b_2。

除此之外，Transformer 使用的不是 BN 而是层归一化，因为其是用于 NLP

领域的，单独分析词向量的每一维是没有意义的，因此选择层归一化，对词向量的所有维度归一化。

结构分析

Transformer 解码器结构中第一个层结构是 Masked Multi-Head Attention，和 Multi-Head Attention 的计算原理一样，只是多了掩码操作。一个是 padding mask，即对每个批次的序列长度进行对齐，较短的补 0，较长的截取左边的内容；二是 sequence mask，即解码器的输入是之前解码器的预测输出信息，如图 7 所示，也就是求输出向量 b^2 时只考虑输入向量 a^1 和 a^2 。

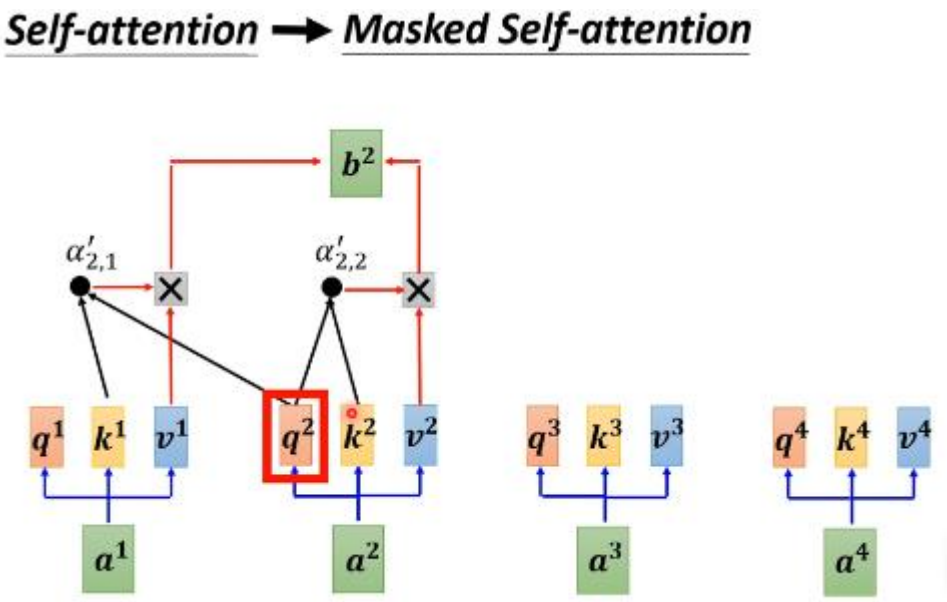

图 7 Masked Multi-Head Attention

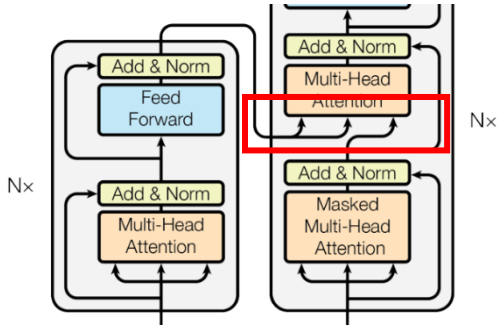

图 8 交叉注意力

从图 8 中的红色框中可以看出，解码器第二个多头注意力层的输入一部分来自于与之相关的编码器的输出 k，v，另一部分来自于 Masked Multi-Head Attention

输出的向量 q。

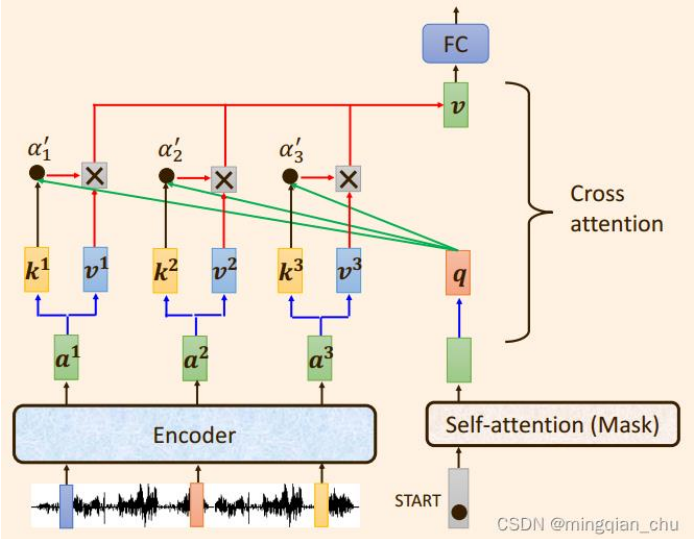

图 9 交叉注意力示意图

总结

多头注意力机制更像是卷积中的分组卷积操作，比如说早期的 AlexNet 分组 GPU 计算。从特征映射角度考虑多头注意力融合了多个自注意力的结果，不是简单的加和平均，增加了非线性。

像编码器和解码器之间的连接感觉和 DenseNet 里的稠密连接一样，利用多层语义信息进行预测。