date: 2025年1月5日
bg：在两年前，有人问我说self attention是怎么算的，我一点都不知道，但我假装会，匆匆去看大名鼎鼎的attention is all u need, 然后惊呼：这tm都是些什么？惊觉每个单词都能看懂，但组合就看不懂了。只好借助知乎和b站（不愧是学习网站）等博主的科普，了解个大概。

大致印象是：每一个token会对其他（包括自己），用自己的query去问其他token的key，来计算内积（两个向量的相似度），得到该token对其他token的相似度，然后作为权重乘以对应token的value，然后加和，就是该token经过self attention的输出；类推接下来每个token。

然后这周上四休三，周末居然一点都不觉得累，刚好看到3b1b的视频，就想着一次性梳理一遍过程。结果却花了不少时间，3b1b的视频里面的过程貌似，行列有误（猜测，不敢质疑大佬，错的话就怪我）。所以自己推了一下过程，大致梳理如下，和以往一样，以大白话为主，主打一个接地气。

首先，输入是一堆token，token是什么呢？可以简单理解成一个个单词，更严谨的可以理解为词加词根，例如smallest会被拆分为small和est。然后token会经过（词嵌入）word embedding，也就是该token的一个向量表示（通常是高维），用向量来表示一个token。

这里插入一个小话题，word embedding有一个很有意思的理解：
假设咱们有一个包罗几乎所有单词的向量化表示的矩阵，咱们把它称为embedding dict（具体叫啥我忘了），然后每一行对应一个token的向量化表示。

$$EmbeddingDic = \begin{bmatrix} 0.1 & 0.3 & ... & 0.2 \\ -0.3 & 0.5 & ... & 0.1 \\ ... & ... & ... & \\ 0.88 & 0.3 & ... & -0.5 \end{bmatrix}$$

然后，一个句子：a cat is running to you，如何获得a, cat,...各自的向量表示（embedding）呢？其实，它就是a, cat,...各自的token，在这个embedding dict的行序列号。根据序列号，取出来对应的行，就是它的embedding（向量化表示）。为什么呢？

可以这样看，咱们先对每个token做one hot编码，比如
a， cat的编码分别是[1,0,...,0], [0,1,0,...,0]，那么该句子的one hot编码乘以embedding dict,就得到a和cat对应索引的行向量.

$$\begin{bmatrix} 1 & 0 & ... & 0 \\ 0 & 1 & ... & 0 \\ ... & ... & ... & \end{bmatrix} \times \begin{bmatrix} 0.1 & 0.3 & ... & 0.2 \\ -0.3 & 0.5 & ... & 0.1 \\ ... & ... & ... & \\ 0.88 & 0.3 & ... & -0.5 \end{bmatrix} = \begin{bmatrix} 0.1 & 0.3 & ... & 0.2 \\ -0.3 & 0.5 & ... & 0.1 \\ ... & ... & ... & \end{bmatrix}$$

是不是有点神奇,而且,这样就很巧妙的,将原本one hot矩阵的稀疏问题,以及无法表示相关token之间关系的问题给解决了.因为它将从一个绝对正交,映射到一个相对低维的空间去. 然后随着训练, 每个token的关系,就可以逐渐由不同token的向量表达所捕捉, 可以用向量内积表示出token之间的相关性.

然后, 以上每个token都会有一个向量表示,假设其特征维度为m, 假设一个句子,有n个token,那么该次输入,就可以表示为一个n*m的矩阵,咱们把它叫做Embedding(E), 每一个token的嵌入向量写作e_i,

$$E = \begin{bmatrix} e_1 \\ ... \\ e_n \end{bmatrix}_{n*m},$$

$e_i = [e_{i1}, e_{i2}, ..., e_{im}]_{1*m}$,m为特征维度

第一部分: Q, K

前置内容部分就是这样,下面是其计算过程.前面每个token的embedding,表示为该token的向量空间表达,但同一个token,对于不同的上下文,可能有不同的含义,也就是不同的向量表达,例如3b1b举得例子,model, 可以是模型, 也可以是模特,这两个涵义在不同语境中是差别很大的.那么具体是如何实现的呢?就是用attention实现的.

还记得前面说的,大致思路是,每一个token,会有q,k,v三个向量, 分别叫做query(问询), key(键), value(值); 每个token是怎么捕捉该token与其他token的相关性的呢? 是通过q去问询其他(包括自己)token的k(key), 即这个token, 去每个token那边的key那都问一遍, 咱俩关系好吗? 然后算一个权重(内积)出来, 然后用该权重(内积)作为系数,乘以对方token的v(value向量), 就是该token在考虑了其他token(包括自己)的关系变化后的向量表达.

这里的q,k,v可以是不同的. 这里只将自注意力(self attention),所以这里的q,k,v都是token自己的, 变化而来的.
那么是怎么变化来的,其实也很简单, 因为对于token来说,其嵌入维度是m, 可能是太多而不必要的,所以一般会将其压缩到低一点的维度, 也就是从$e_i = [e_{i1}, e_{i2}, ..., e_{im}]_{1*m}$ 会降低到$q_i = [q_{i1}, q_{i2}, ..., e_{id}]_{1*d}$的d维度.具体是将$$e_i$乘以一个矩阵$W_Q$,如下:

$$e_i = [e_{i1}, e_{i2}, ..., e_{im}]_{1*m} \times \begin{bmatrix} w_{11} & ... & w_{1d} \\ ... & ... & ... \\ w_{m1} & ... & w_{md} \end{bmatrix}_{W_Q:m*d} = q_i = [q_{i1}, q_{i2}, ..., e_{id}]_{1*d}$$

也就是$q_i = e_i * W_Q, shape: 1*m*m*d = 1*d$
将n个token,都做这样的操作,表示为成一个矩阵Q,如下:

$$Q = \begin{bmatrix} q_1 \\ ... \\ q_n \end{bmatrix}_{n*d} = \begin{bmatrix} e_1 \\ ... \\ e_n \end{bmatrix}_{n*m} \times W_Q = E \times W_Q$$

同样的, k_i也是一样的,
$k_i = [k_{i1}, k_{i2}, ..., k_{id}]_{1*d} = e_i * W_K$

$$K = \begin{bmatrix} k_1 \\ ... \\ k_n \end{bmatrix}_{n*d} = \begin{bmatrix} e_1 \\ ... \\ e_n \end{bmatrix}_{n*m} \times W_K = E \times W_K$$

这里的W_Q, W_k都是m*d维度的.

然后token e_i 用query q_i 去问询其他token e_j的key,计算内积
$q_i = [q_{i1}, ..., q_{id}]; k_j = [k_{j1}, ..., k_{jd}]$

内积为: $q_i * k_j^T = \sum_{z=1}^{z=d} q_{iz} * k_{jz}$

表示为矩阵为:

$$Q * K^T = \begin{bmatrix} q_1 \\ ... \\ q_n \end{bmatrix}_{n*d} \times \begin{bmatrix} k_1, ..., k_n \end{bmatrix}_{d*n} = \begin{bmatrix} q_1 * k_1^T, ..., q_1 * k_n^T \\ ..., ..., ... \\ q_n * k_1^T, ..., q_n * k_n^T \end{bmatrix}_{n*n}$$

这里,论文里面做了一个scale操作,也就是将$Q * K^T$的每个值除以$\sqrt{d}$, 因为每个值是d个相乘再加和,为避免d过大时导致的数值计算问题,所以这里除以d使得其范围变小. 下面暂时忽略,因为表达式上没什么关系. 同时,也忽略**Softmax操作**

观察这个$Q * K^T$可以知道,i行代表的是第i个token的embedding 的query对所有token的key问询,计算后的权重向量.
那么下一步,咱们要做的就是,将每个token问询所得到的权重向量(对应的$Q * K^T$的行向量),各自乘以对应token的value, 然后加总就得到该token的输出.
这里暂时将这个token问询的权重向量,写为s_i, 那么:

$$QK^T = \begin{bmatrix} s_1 \\ ... \\ s_n \end{bmatrix}_{n*n}, s_{ij} = [s_{i1}, ..., s_{in}]$$

第二部分 V

注意,V与Q, K的维度有点不一样.

W_V的维度为n*v, v为输出的向量的维度,一般而言, 例如transformer,输入与输出的维度是相同的,这里以embedding为输入的话, 那么这里的v也是会等于m. 而且一般会使用resnet, 那么也要求输入输出的维度是相同的. 对于resnet简单可以理解为$output_i = e_i + \sum_{j=1}^{j=n} s_{ij} * v_j$, 这里的$v_j$与输入$e_j$维度相同,都为1*m,即v=m

W_v为m*v维度, 将m维度压缩到v维度, 因为resnet,这里是m=v

$$V = \begin{bmatrix} v_1 \\ ... \\ v_n \end{bmatrix}_{n*v} = \begin{bmatrix} e_1 \\ ... \\ e_n \end{bmatrix}_{n*m} \times W_V = E \times W_V, v = m$$

那么,对于token e_i的问询权重s_{ij}乘以v_j对于j从1到n, 然后加和,也就是token e_i的输出.

resnet的话,就是$output_i = e_i + \sum_{j=1}^{j=n} s_{ij} * v_j$, 拓展到n个token的话,用矩阵表达,即

$$O = E + \begin{bmatrix} o_1 \\ ... \\ o_n \end{bmatrix}_{n*v, v=m} = E + \begin{bmatrix} \sum_{j=1}^{j=n} s_{1j} * v_j \\ ... \\ \sum_{j=1}^{j=n} s_{nj} * v_j \end{bmatrix}_{n*v, v=m} = E + QK^TV$$

所以论文里的公式得证:
$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d}})V$

第三部分 多头注意力机制

上面所说的,都是单头注意力机制, 那么多头注意力机制是什么呢?

作者说: 直接把一个embedding的m维同时输入, 用一个注意力,似乎不如, 把m维的embedding, 通过一个MLP, 变成h个dv维度, 也就是m=h*dv, 然后分别用h个注意力分别处理每个dv维度的输入,
然后得到h个dv维度输出(因为输入和输出维度相同),然后concat回来成m维度的向量.

3.2 关于masked attention

论文里面有个masked多头注意力,这个又是什么呢? 这个是为了让每个token在问询的时候,只问询自己位置前面的token,而不问后面的token.

具体是怎么实现的呢?还记得QK^T这个n*n的大矩阵吗? 每行代表的是该token对所有token问询的权重向量, 然后除以$\sqrt{d}$, 再softmax, 那么要避免该token问询,它位置后面的token,只需要对右上角矩阵赋值负无穷,那么在softmax的时候,它的值就会变成0. 这就是所谓的masked attention.

到此, 终于结束了.一句话总结就是,每个token的向量会计算关于其他token的相关度, 作为权重,再甲醛得到该token,根据上下文调整后的向量.以此来得到token之间的相关情况.应该足够白话,过程应该足够清晰吧,不清晰那也没办法了.