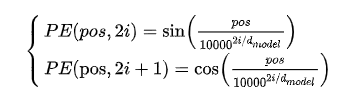
transformer答疑

1. 如何理解transformer的位置向量？



pos代表一个句子中的一个词。

如：我 想 谈恋爱

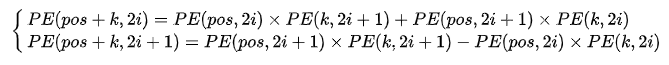
PE(0,2i) 代表位置为0的词我这个词的词向量的第2i个位置

PE(0,2i+1) 代表位置为0的词我这个词的词向量的第2i+1个位置

位置向量的目的是：希望模型学习到词与词之间的相对位置。

为啥这个公式可以表示词与词之间的位置信息呢？



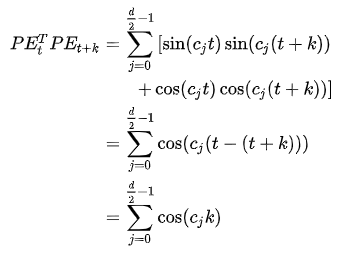


这个公式的含义是：

句子中的第pos+k个位置，某个词之后的第k个位置的词，2i是这个词对应的词向量的第2i个值，可以表示为第pos个位置的词向量的线性表示。

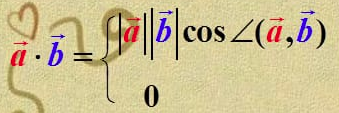
但是这个线性表示引入的东西比较多，包括第k个位置词向量的2i+1个值和2i个值，还有第pos个位置词的2i和2i+1的位置。

这样做一个线性表示就能表示词与词之间的相对位置吗？



怎样计算两个向量之间的距离：

因为每个位置向量的模都是一样的大小。跟词向量维度有关，如果维度为512，那么模的大小为256^0.5



衡量距离可以用向量和向量之间的角度来衡量。因此点积就可以衡量向量之间的距离。

可以看出第k个位置词和第t+k个位置之间的距离是k的函数。

因此那种表示是可以衡量相对位置的。

但是，思考另外一个问题：



这个公式意味着词与词之间的距离一致时便无法分清先后顺序。

例如：我 想 谈恋爱。针对想这个词，我 和 谈恋爱这两个词是无法定位其先后顺序的。

同时，位置向量需要通过长途跋涉才会到达decoder。

在第一个encoder中就出现了



这个意味着中间加了个矩阵作为乘法，虽然说最终的结果是一个标量，但是w是一个可以学习得到的参数，是具有可变性的，可变的东西意味着不再是一个仅仅是关于k的一个自变量的函数，而是更加复杂，更加没有规律性。

这可能就是transformer中位置信息做的不够细致的一个原因吧。后面会有很多人进行改进。

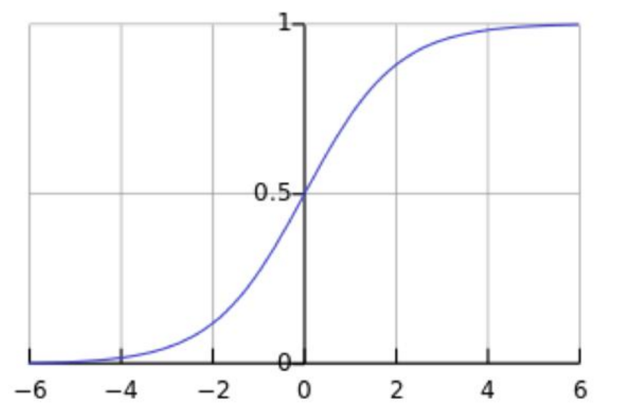
**Batch normalization**

1. 理解layer norm之前需要先理解BN

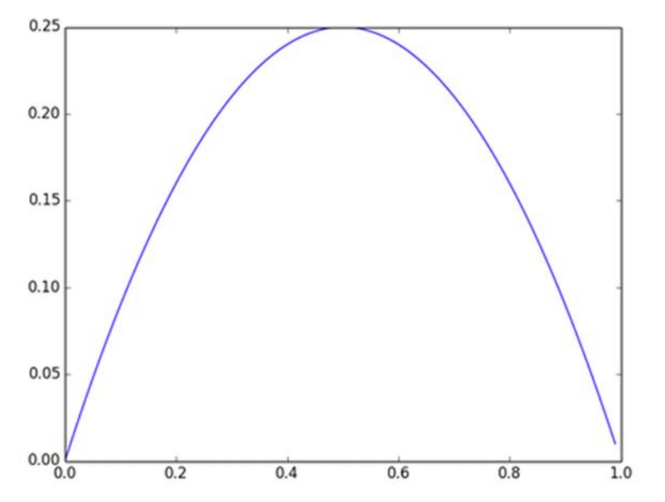
iid是统计学习中非常重要的假设，如果通过一批数据训练好了一个模型，上线了之后n天之后发现效果越来越差了，可能是因为数据的分布发生了变化。bn就是用到深度学习中的保持iid的一个东西。

1. 假如说深度学习中激活函数都用sigmoid

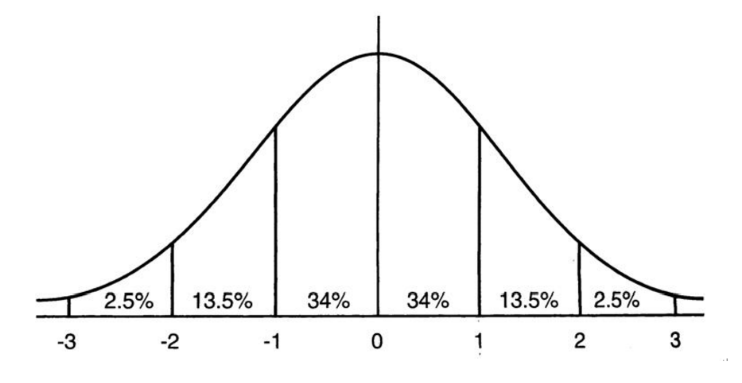
如下图可以看到当x属于-2到2之间的时候几乎非线性激活函数激活效果如同线性函数，意味着这一块区域的导数最大，激活函数看的是激活函数的导数，但是越接近两边导数越接近0. 在求深度学习的参数的时候，需要对w进行迭代，迭代的时候我们希望导数越大越好，越大收敛越快。



sigmoid函数的导数：



标准正态分布。



正态分布是一个特别普遍的分布，根据中心极限定理定理可知：

任意k个变量的均值组成的符合变量的分布都服从正态分布。

所以可以假设训练样本服从正态分布，还有深度学习中每一层隐函数往后面激活的时候，激活前的那些每个维度的变量也服从正态分布。

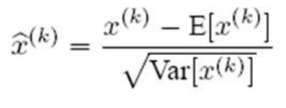
将正态分布和sigmoid函数联合起来看：

当x属于标准正态分布时，x在-2和2之间的比例达到27%+68%=95%。

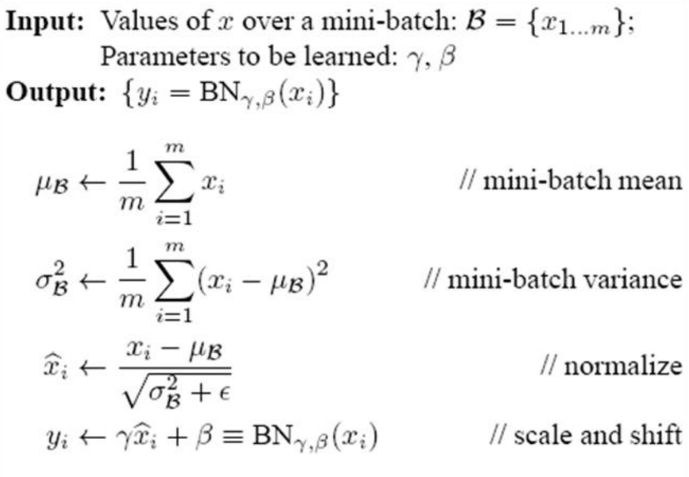
因此如果我们令输入深度学习中，进行激活函数之前的值都尽量保证在-2到2之间，那么损失函数的收敛速度是非常快的。

如何做呢？

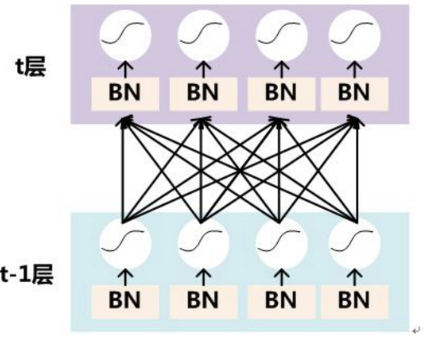
深度学习训练中最常用的就是minibatch sgd，在minibatch中强制将每个维度的数据都转化为标准正态分布。



然鹅，即使全部转化为标准正态分布，还是有问题，并不能保证每个数据都能落在-2到2之间。这个时候可以再做一个过分的事情，对minibatch中的样本再进行加乘，因为这样可以将x拉到线性变化区，有点玄幻的感觉，但是这就是深度学习的精髓啊。其中r和b都是待学习参数。咋学习的？暂时没研究。可见加了bn之后模型收敛速度可是非常的快。



bn操作的位置是激活之前



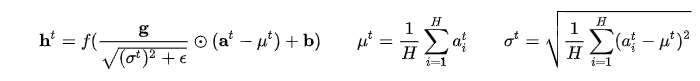
模型训练好了，之后需要在线预测

在线预测需要几个每个隐含层激活前的那些每个维度的均值方差信息，训练的时候保存下来，预测的时候利用他们求全局的均值方差即可。同时scale参数和shift参数也是训练好的，直接拿来用即可。

**Layer norm**

1. DNN中可以使用BN，RNN没有minibatch的概念，怎样BN呢？

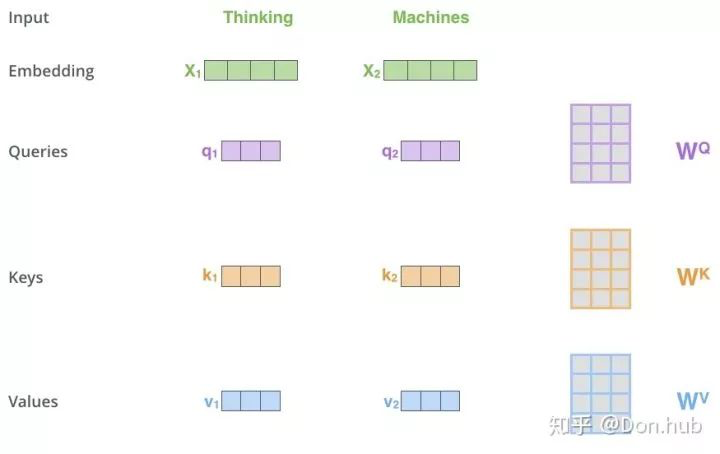
真的是对不同对网络结构遍历同样的想法，不行就改个方法，转个置就可以了，可行就是创新。。



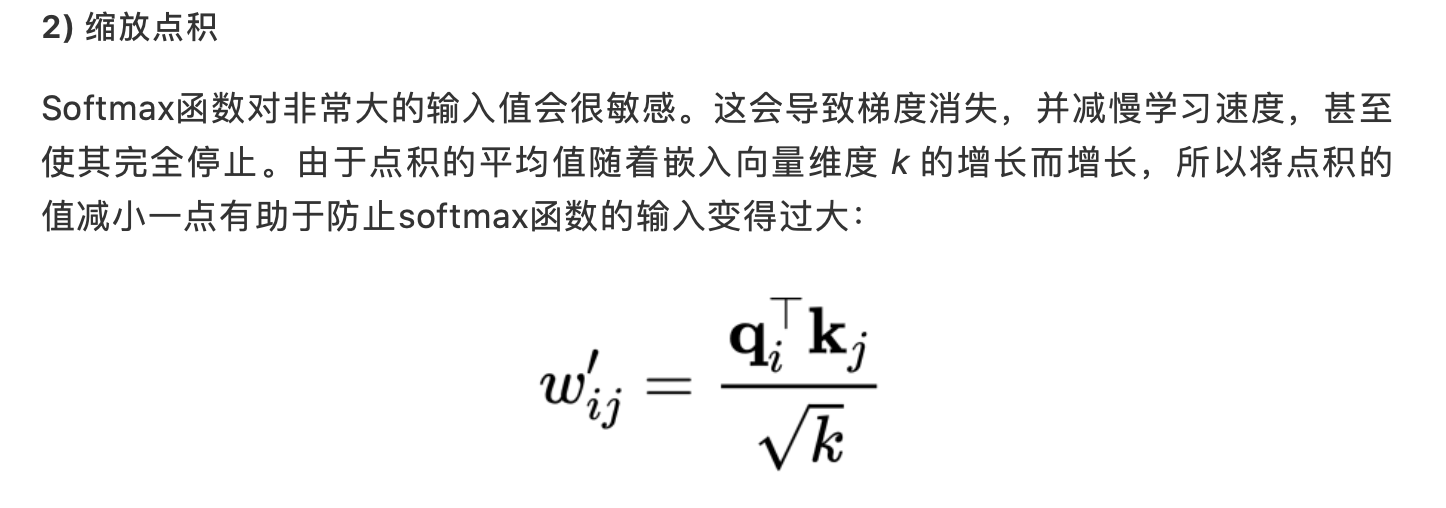
rnn中的LN就是在net(t)上做，net(t)是一个向量。对这个向量求均值和方差，同BN还需要加入shift参数和scale参数。同理，rnn模型训练完成之后，可以利用历史均值和方差求出一个总的均值和方差进行标准化

1. self-attention是什么东西？

self-attention和seq2seq中的attention是不一样的，seq2seq中attention的计算是将decoder中的隐含层hi包括了进去。而self attention重要的self这个东西，意思是自己跟自己计算，并不是encoder是自己decoder也是自己，仅仅是x向量跟x向量自己计算。其中WQ WK WV是通过decoder逐层反向传播计算出来的，并不是输入是自己输出是自己训练出来的。



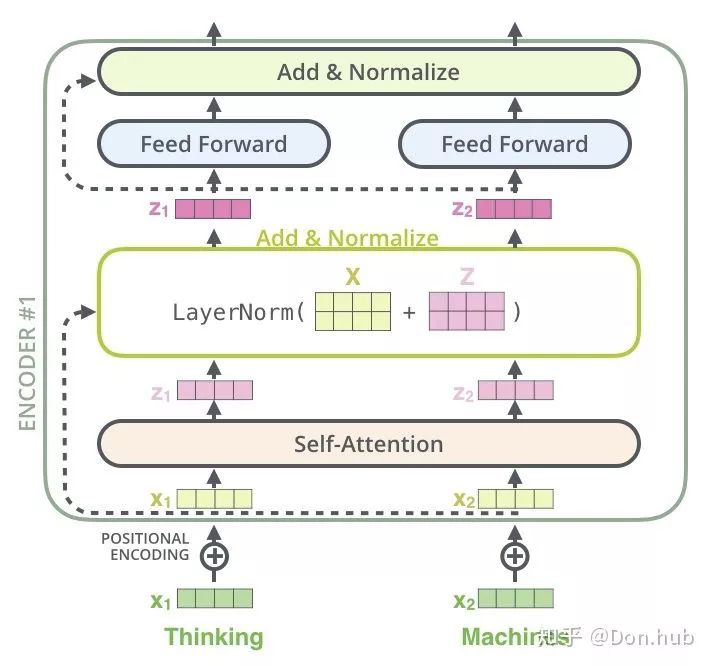
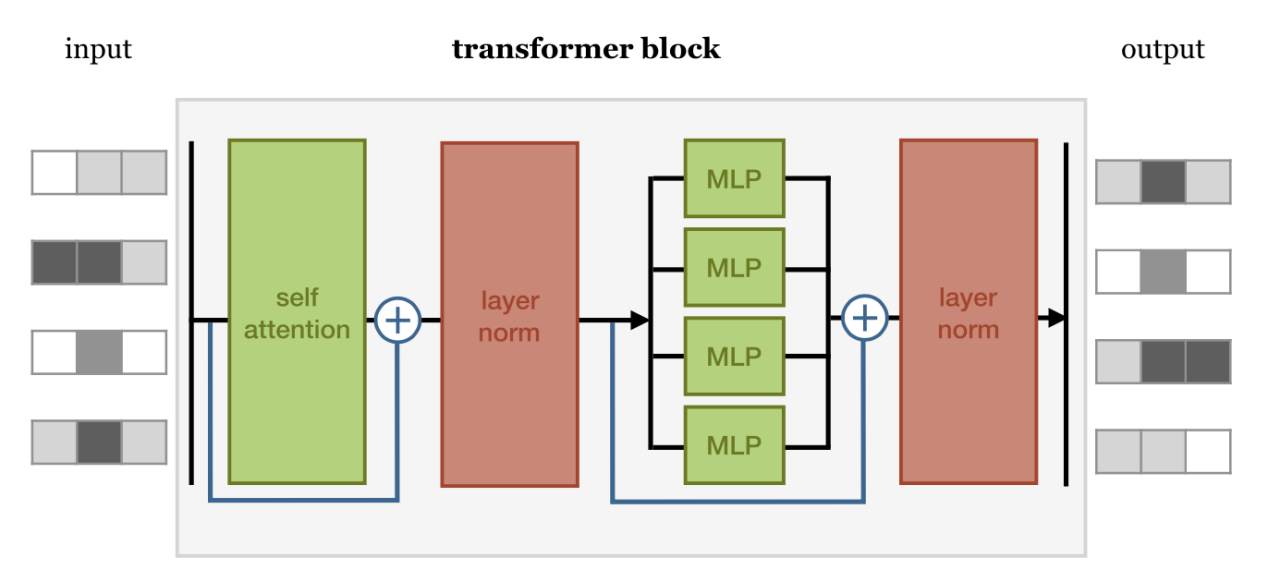
1. 缩放点积



1. 彻底理解transformer中的一个encoder。

1）一个句子由多个词组成，每个词由词向量，每个词向量分别乘以WQ WK WV就可以得到这个词向量对应的 查询向量、键向量、值向量，然后经过一波self attention操作，就得到了下图所示的v矩阵。然后残差连接，在做layer norm。

1. mlp就是做乘以一个矩阵，然后接激活函数
2. 然后再做加和和layer norm



1. 怎样理解一个decoder？

下图左边是encoder，右边下方是decoder。decoder是相对于encoder多了一个encoder-decoder attention。下图仅仅用了2个encoder和2个decoder。论文中的transformer是用了6个。用多少是超参数，可以自己选。

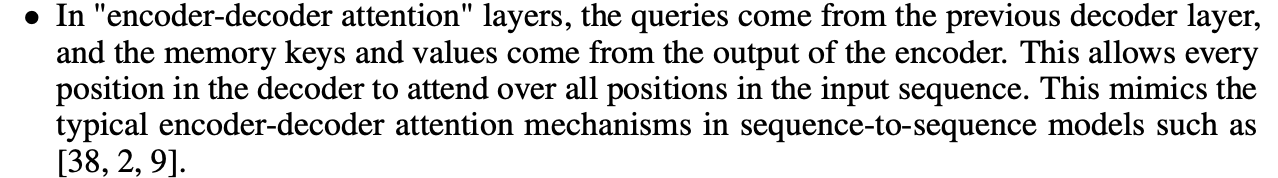
理解decoder中最重要的一部分就是理解encoder-decoder attention。这东西和self-attention大致是一样的，都是用多头 self-attention。不同的是：

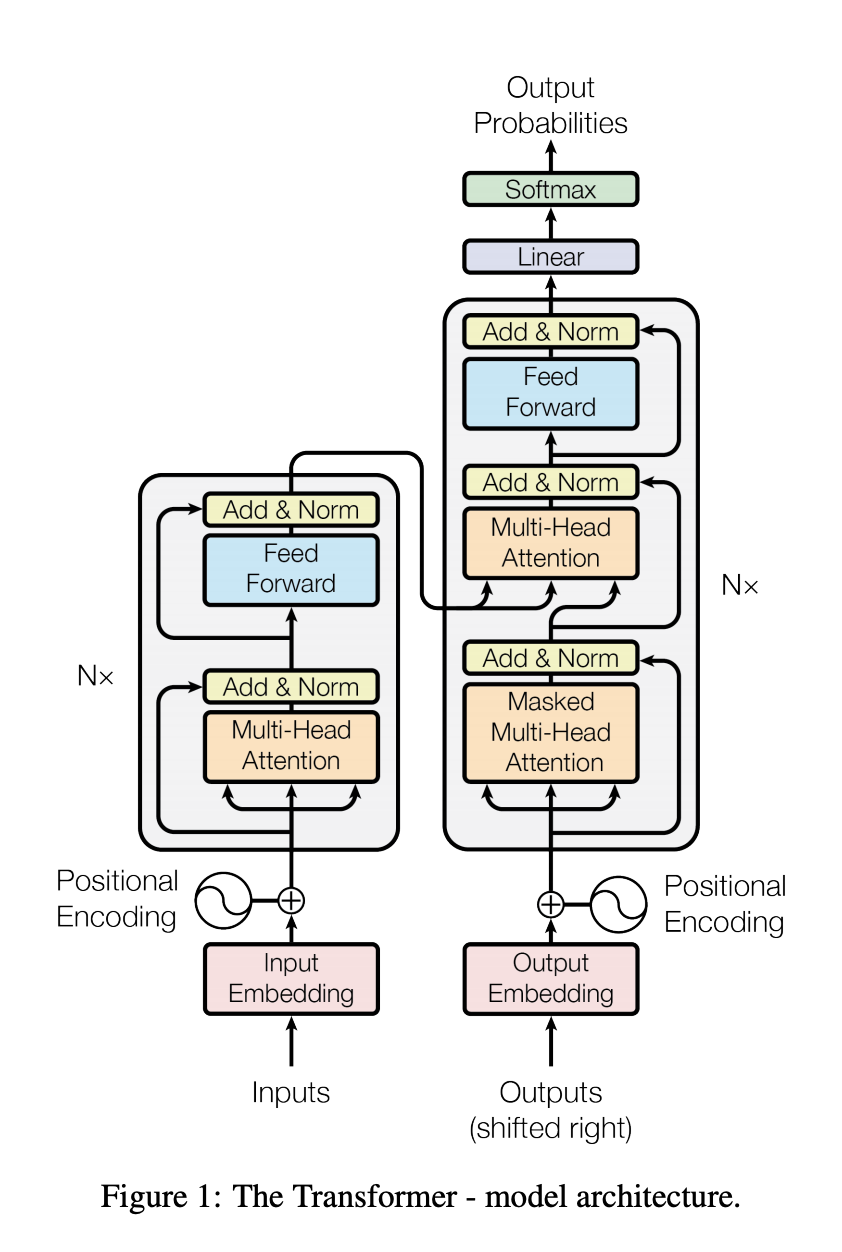
Q来自于decoder，K、V来自于最后一个encoder中的self-attention.

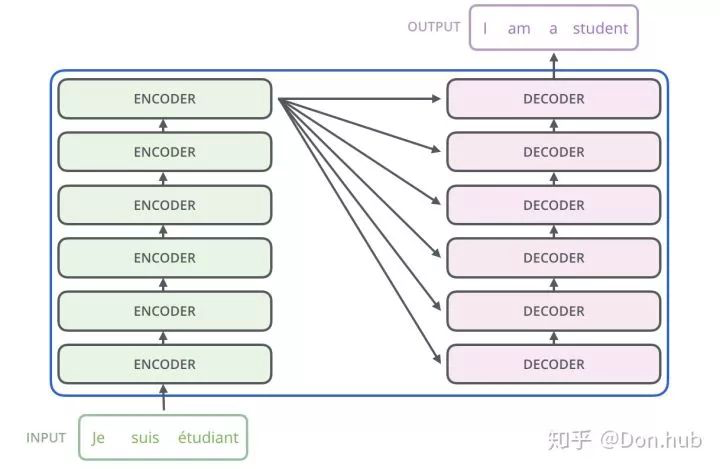
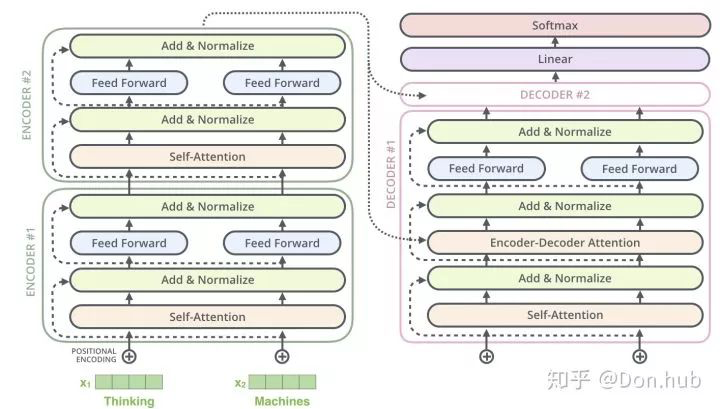
这里有个小细节：

encoder中每个self attention要学习的参数都包括:

WQ、WK、WV。最后一个encoder中的self-attention会得到K、V，这是将输入矩阵与WK、WV相乘之后的结果。而论文里说Q来自于decoder，那就意味着decoder-encoder attention中需要学习一个WQ矩阵，与decoder中的self-attention输出做一个乘法得到Q而不是用了

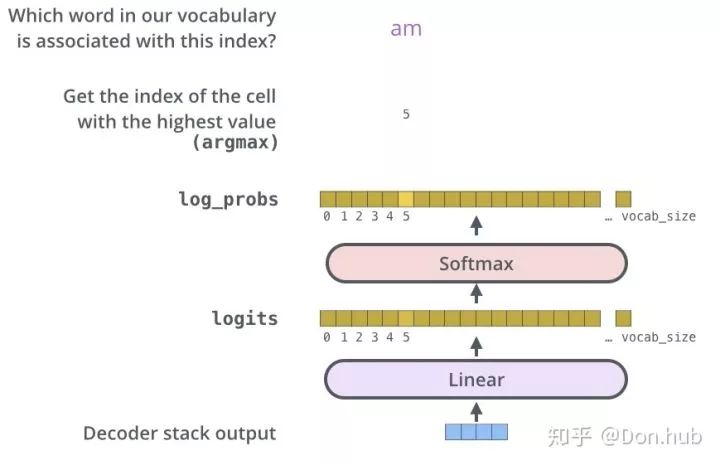






1. 怎样理解output layer

decoder的输入是一个矩阵，包含位置向量，输出的应该也是矩阵，为什么decoder之后的结果是一个向量呢？对向量乘以矩阵，然后做softmax就可以分类



1. transformer中的mask

首先为了并行化训练，decoder中的输入会做mask。因为模拟训练的过程中不应该知道该解码的下一个词是什么。

1. transformer如何并行？

rnn被诟病的是rnn速度慢。速度慢的原因是不支持并行。

1）rnn训练的时候是可以并行训练的，只是预测的时候是串行的

1. transformer训练的时候跟rnn一样也是可以并行的，不过解码的时候只能串行。

训练的时候怎样并行？decoder中待预测部分用mask

11. transformer的应用