基于Transformer模型的机器翻译

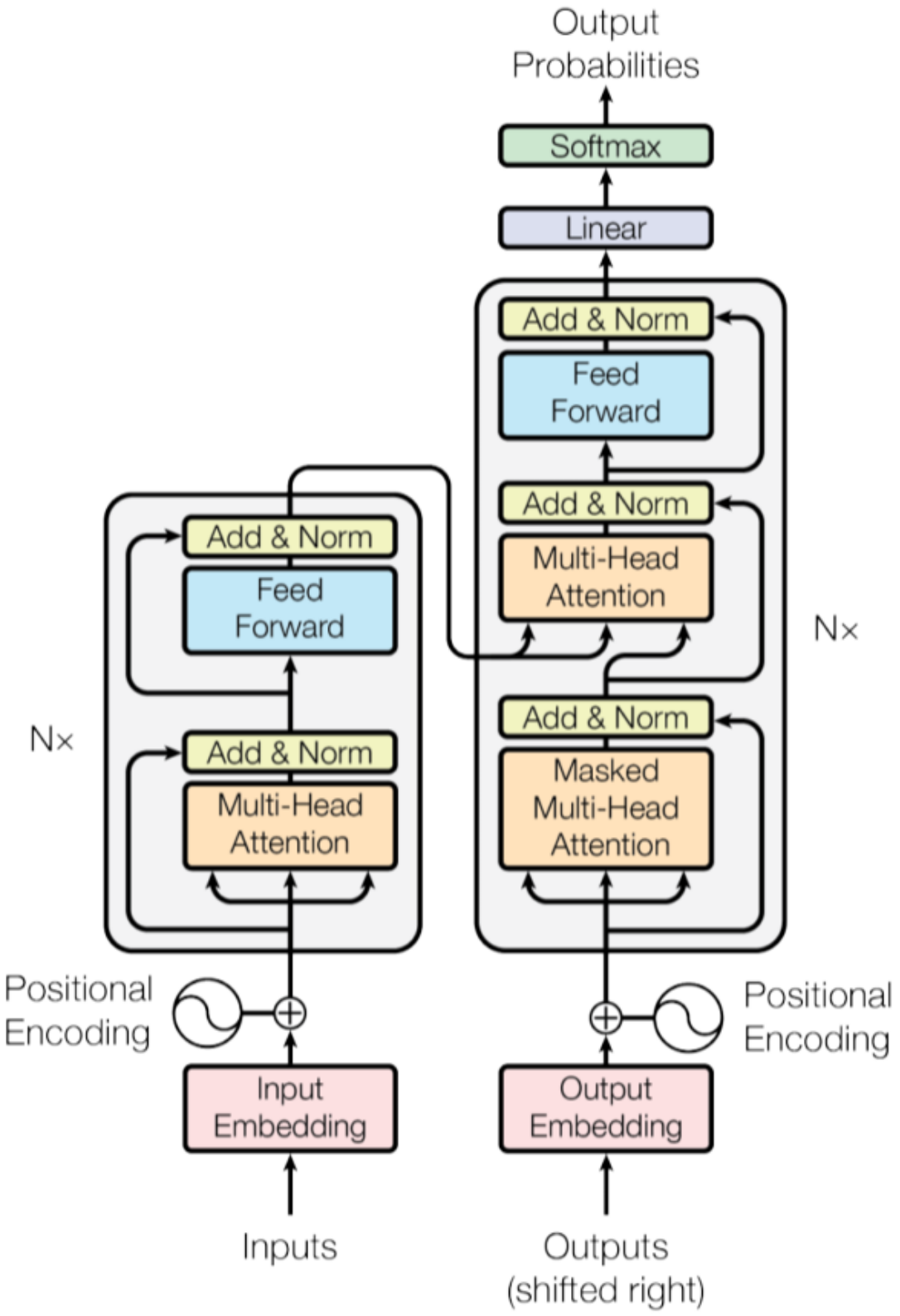
一、任务描述

目标是使用预处理好的IWSLT'14 De-En数据集，实现基于Transformer的机器翻译。

二、Transformer模型简介

以下部分为我学习后总结的内容，可能有理解错误的地方。

Transformer模型结构图



结构图引用自《*Attention Is All You Need*》论文。

Transformer模型主要包括Encoder和Decoder两部分，下面分Encoder和Decoder分别进行介绍。

Encoder部分

输入部分

1、Embedding词嵌入

一个单词对应一个one-hot向量，一句话构成一个矩阵，通过与词嵌入矩阵做矩阵乘法获得为维度为512的词嵌入向量，即这句话的分布式表示。下图以四个字（一个汉字一个单词）为例，生成了4*512的矩阵。

2、位置编码

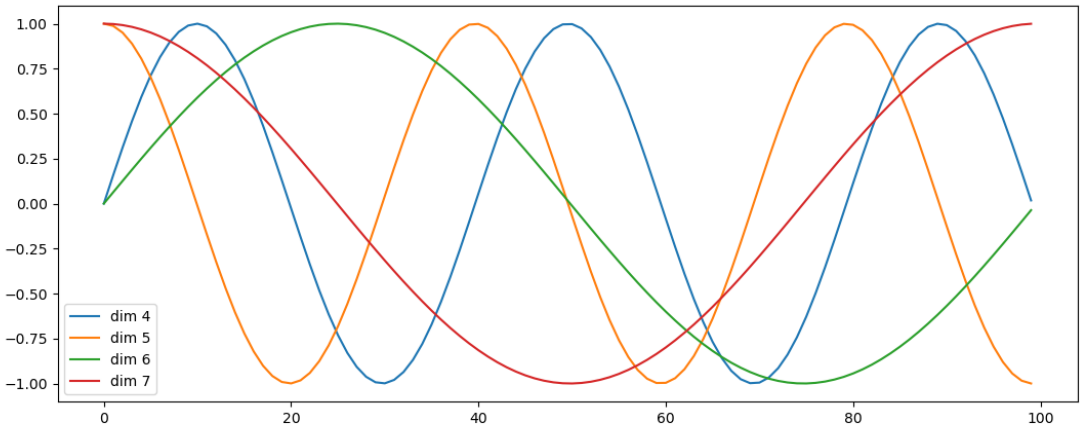
位置编码公式如下：

$$PE_{(pos,2i)} = \sin(\frac{pos}{10000^{\frac{2i}{d_{model}}}}) \tag{1}$$

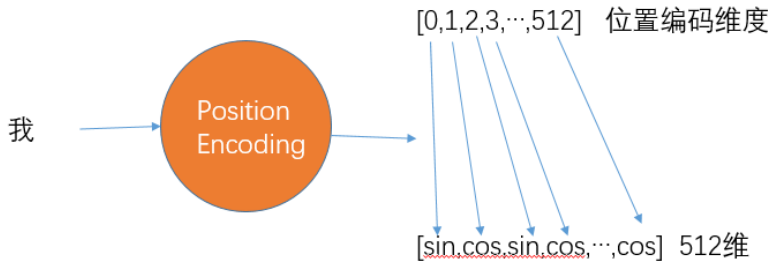
$$PE_{(pos,2i+1)} = \cos(\frac{pos}{10000^{\frac{2i}{d_{model}}}}) \tag{2}$$

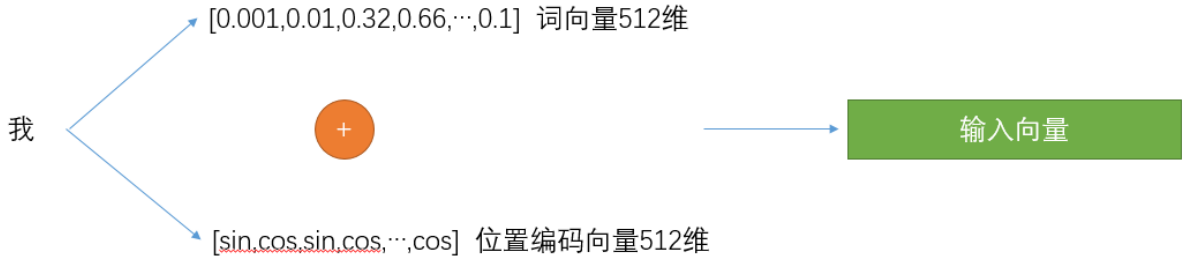
pos：句子中的第几个单词
i：词向量的第几维
d_{model}：词嵌入维度

随着i 的增大，正余弦函数的周期不断变大，以此将位置信息编码，输入给模型。图像大致如下：



继续以上图“我爱中国”为例，“我”这个字的$pos=0$，它有$d_{model}=512$维，i的取值范围是$[0,511]$，其中偶数维度使用式（1），奇数维度使用式（2），获得位置编码向量。再用位置编码向量与词向量按位相加，得到最终的输入向量。注意，一个词是一个向量，如果是一句话，则输入是一个矩阵。





注意力部分

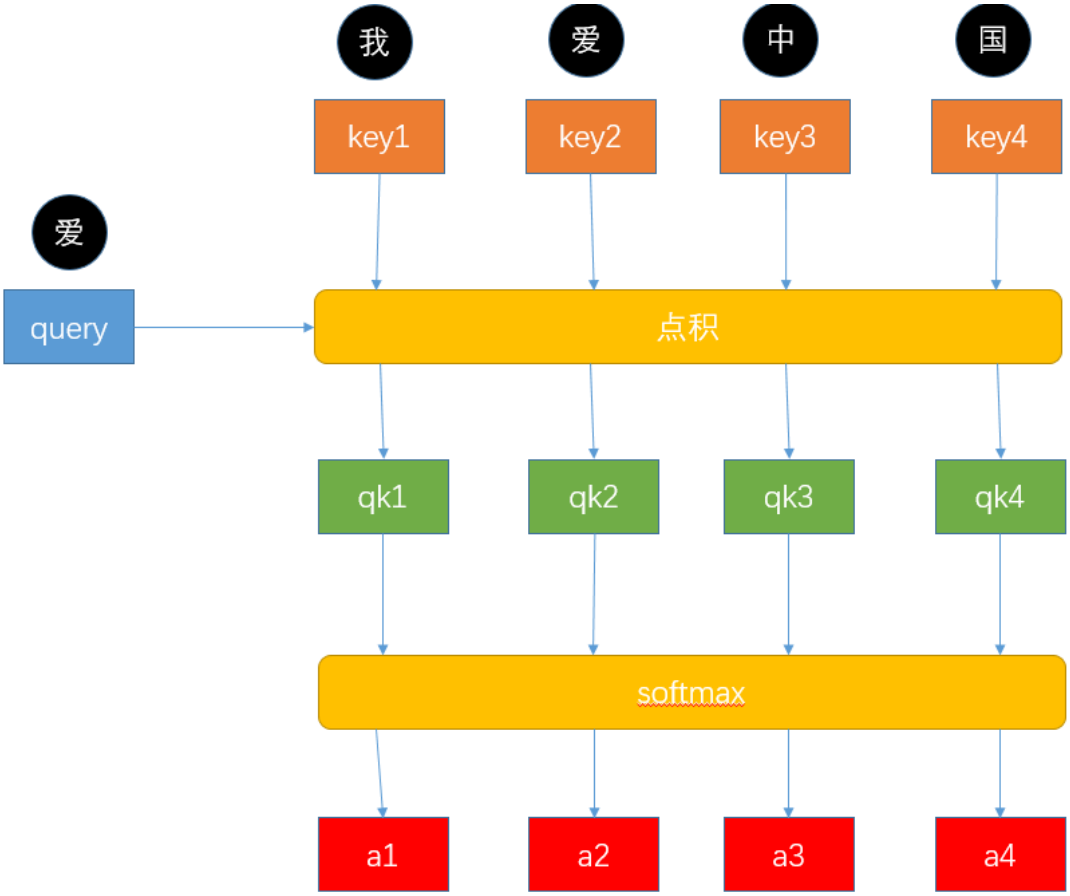
1、注意力机制

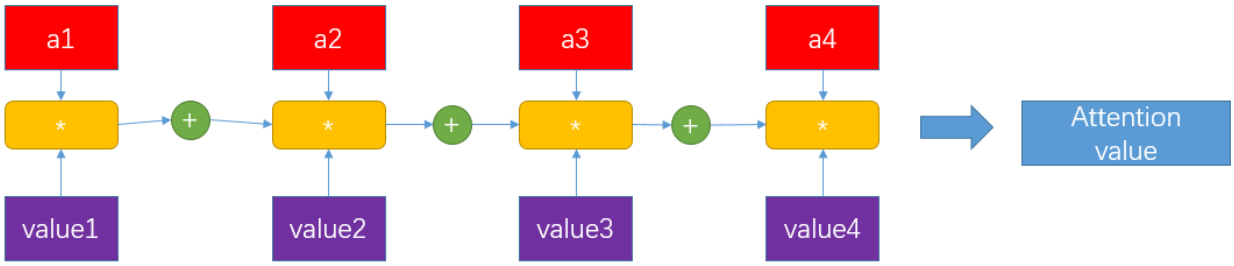
举例来说，人们看到一张图片时，注意力不会均匀分布在图片的每个部分，只有鲜明的部分会引起人们的注意。与此类似，对于一句话，人们的关注的也不是每一词，有些词能鲜明表达整句话含义，而有些词没什么意义。因此需要把更有意义的词凸显出来，上下文向量由此而来，强化含义丰富的词，弱化没有意义的虚词，使神经网络更好理解句子含义。

2、自注意力机制

计算当前词向量query（Q）与句子中所有词向量key（K）的相似程度，得到权重（QK^T），归一化后，再对value（V）加权求和（$softmax(QK^T)V$），得到当前词对整句话的表示。

如下图所示。由“爱”这个词与这句话中每一个词的的词向量作点积，经过softmax层归一化处理再与value加权求和，得到最终的Attention值。



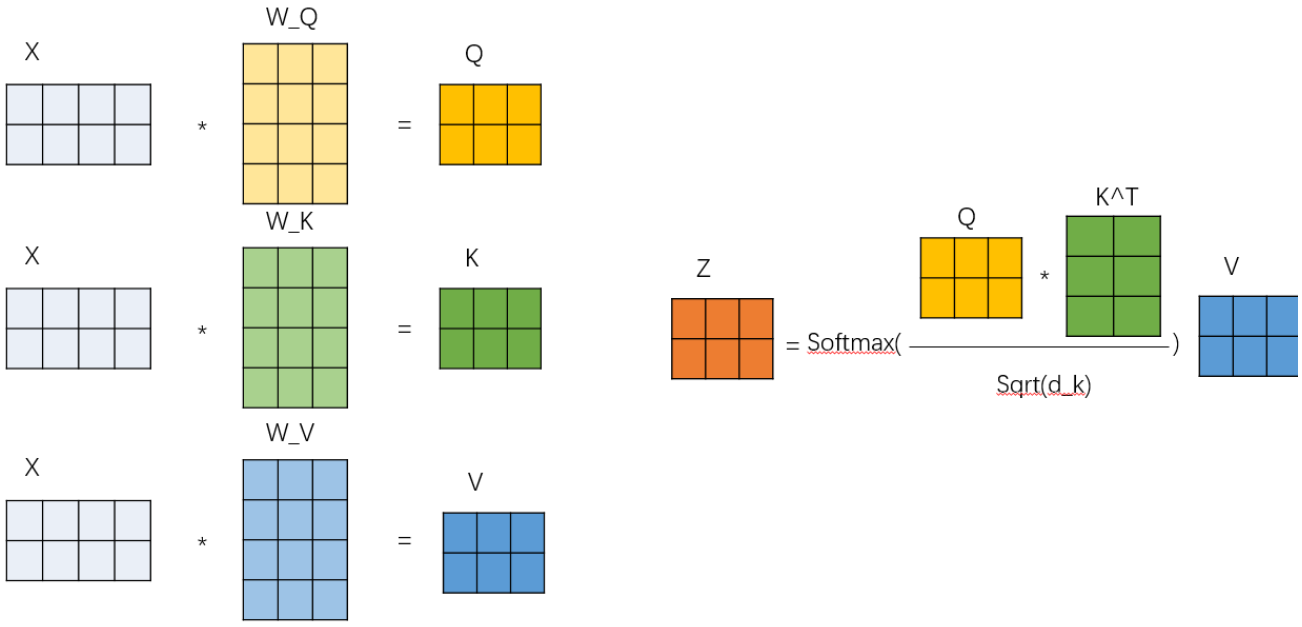


3、Transformer的自注意力

计算公式如下：

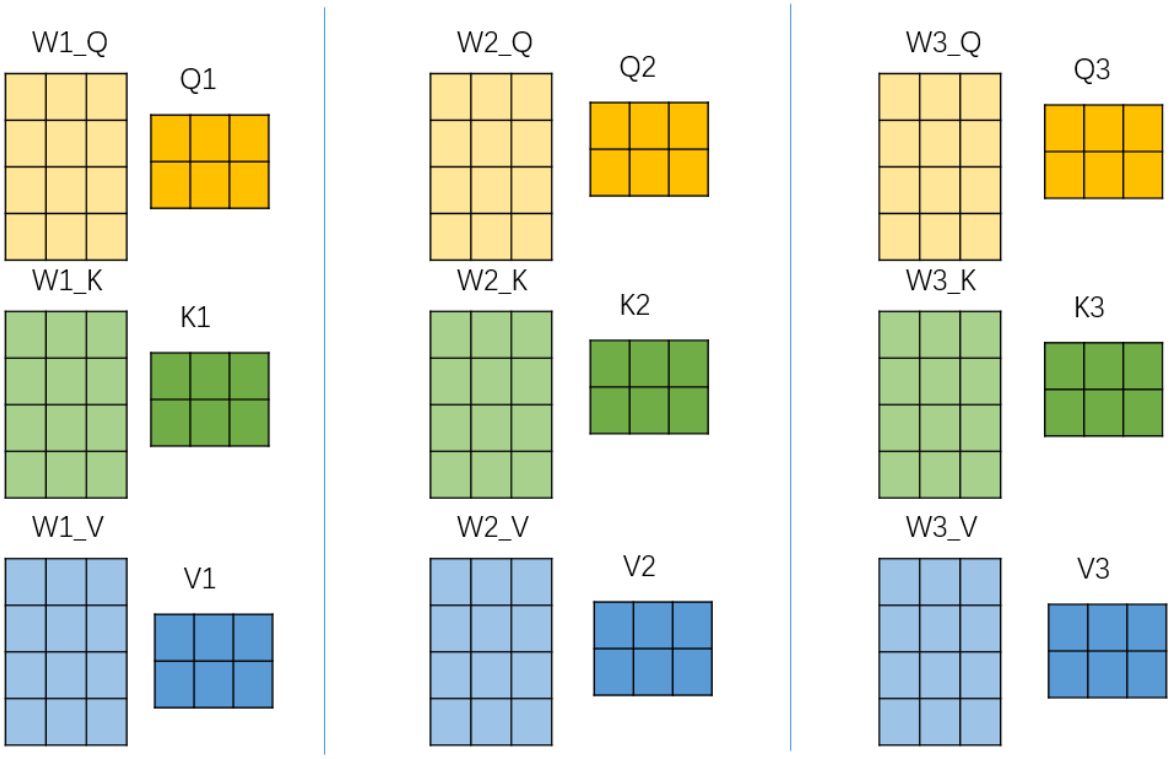
$$Attention(Q,K,V)=softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{3}$$

Transformer的自注意力与上节提到的自注意力机制类似，区别在于使用$\frac{QK^T}{\sqrt{d_K}}$，但多除了一个$\sqrt{d_k}$。原因在$softmax$上，如果某个值过大，再过一遍$softmax$会把这个值放得更大，基本接近1了，这样很容易产生梯度消失。除以$\sqrt{d_k}$，可以弱化这种差异，还可以使q·k均值为0，方差为1。



4、多头注意力机制

Transformer的自注意力机制设置了多套参数，一套参数对应一个头。Transformer使用多头注意力原因：每一个头可以关注到词之间的不同的联系，采用多头可以避免采用一个头时只关注词之间的一方面的联系，而忽略了其他方面的联系，多头相当于把原始信息映射到了不同的空间。采用8头的原因：在《Attention is All You Need》这篇论文中提到使用多头（8头）注意力效果更好；下图中，三套参数（省略了X），对应三个空间，这样可以使Transformer捕捉到不同子空间的信息，提取到更丰富的特征信息。

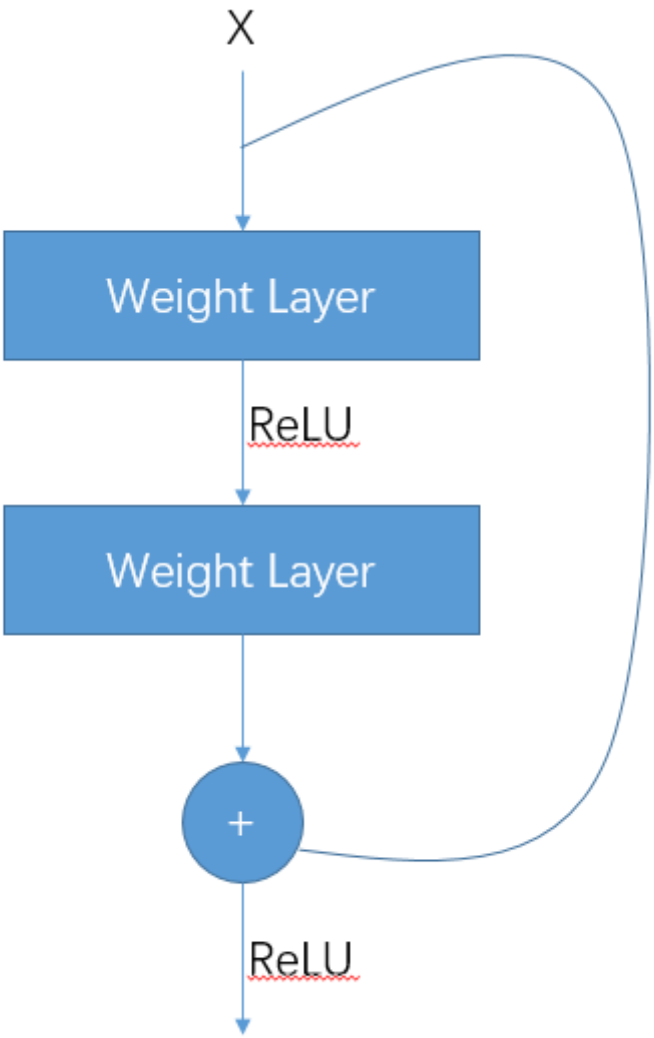


多头会有多个输出，最后的结果需要合并。

残差连接与Layer Normaliztion

1、残差连接结构

神经网络层数越多，连乘越多，梯度消失一般情况下是因为连乘。残差连接确保即使连乘再多，也有一个1，梯度不会为0，缓解了梯度消失。



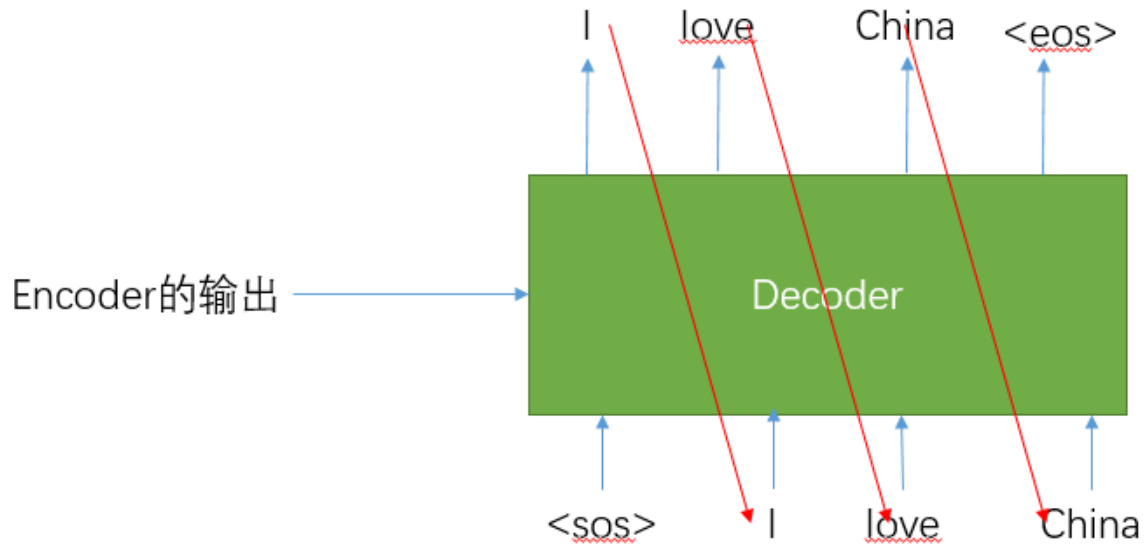
2、Layer Normalization

Normalization可以使模型收敛更快。Transformer使用LN而不适用BN，原因在于BN是把一个batch_size大小的一组样本的同一个维度进行标准化。而在机器翻译模型中，输入的样本是一个个句子，对几句话的第一个词进行标准化显然没什么意义，可能还把重要信息模糊了。所以BN不适用，使用LN。LN是对一个样本即一句话进行标准化，符合预期。

Decoder部分

1、掩盖的多头自注意力

观察Decoder和Encoder的多头自注意力模块，差别在于Decoder多了一个“Masked”。在Decoder的注意力部分Masked主要是遮盖未来时刻的单词。因为在实际预测的时候，我们是得不到当前时刻之后的单词的。以下图为例，当Decoder根据Encoder的输出和“$< sos >$”输出“I”后，把“I”作为下一个时刻的输入，再据此预测下一个单词。此时Decoder并不知道后面的“love”和“China”两个单词。但是如果不对这两个单词进行遮盖，“I love China”这几个单词都会为生成“love”提供信息。但这不符合实际，我们要翻译一句话，不可能事先知道译文，所以必须进行遮盖，才能正确预测。



2、与Encoder的交互

Encoder的总输出要给到每个Decoder层。Encoder生成K，V矩阵，Decoder生成Q矩阵。Decoder的Q来自于本身，K，V来自于Encoder。

三、实验过程遇到问题

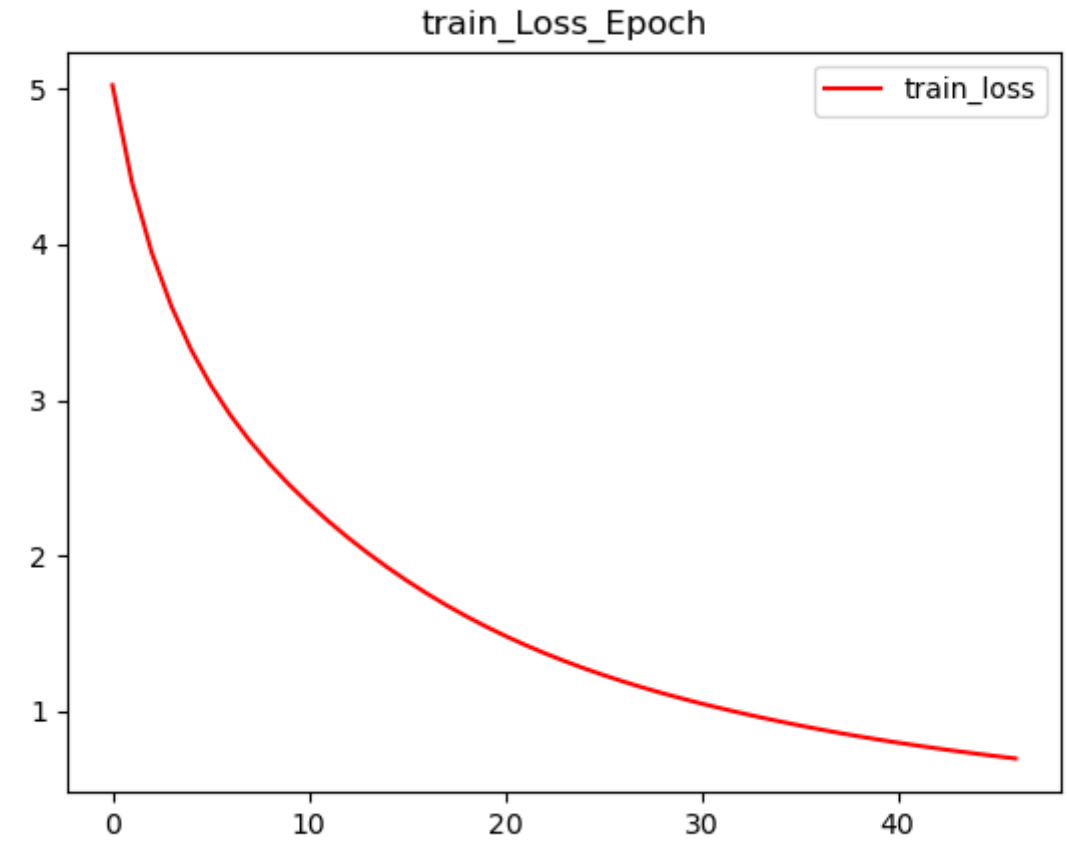
1、内存不足

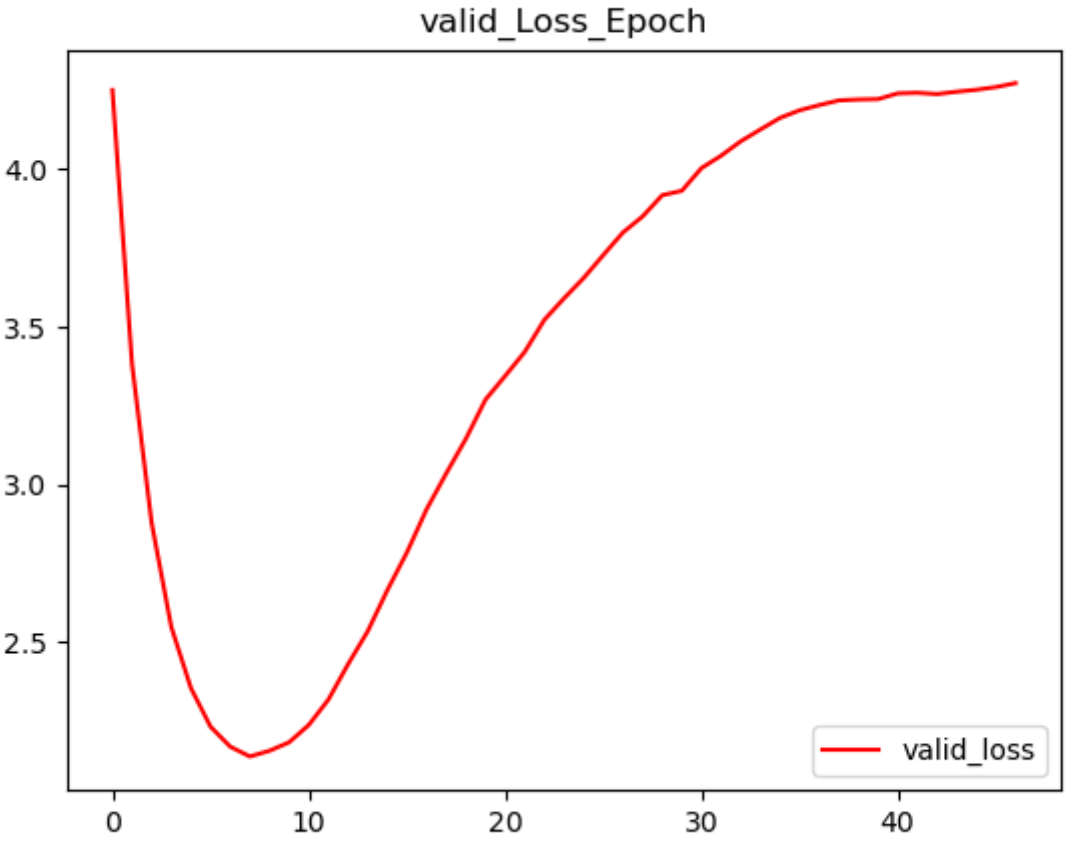
由于我自己的电脑没有GPU，根本无法运行这么大的数据集，只能使用课上提供的服务器进行训练。但是代码提上去运行就发生了下面的错误，一开始，我以为是服务器GPU也不能运算这么大的数据集，后来分析了一下代码，猜测可能是我每次训练的时候都把输入按最长的那个句子补齐，导致输入维度特别大，GPU运算也有负担。于是我在读取数据的时候，限制了句子长度，在服务器上顺利运行了，也许我的猜测是正确的。但是这样处理了之后，我又产生了疑问。如果输入的就是长句子，要怎样运算？一般都是怎么处理的？

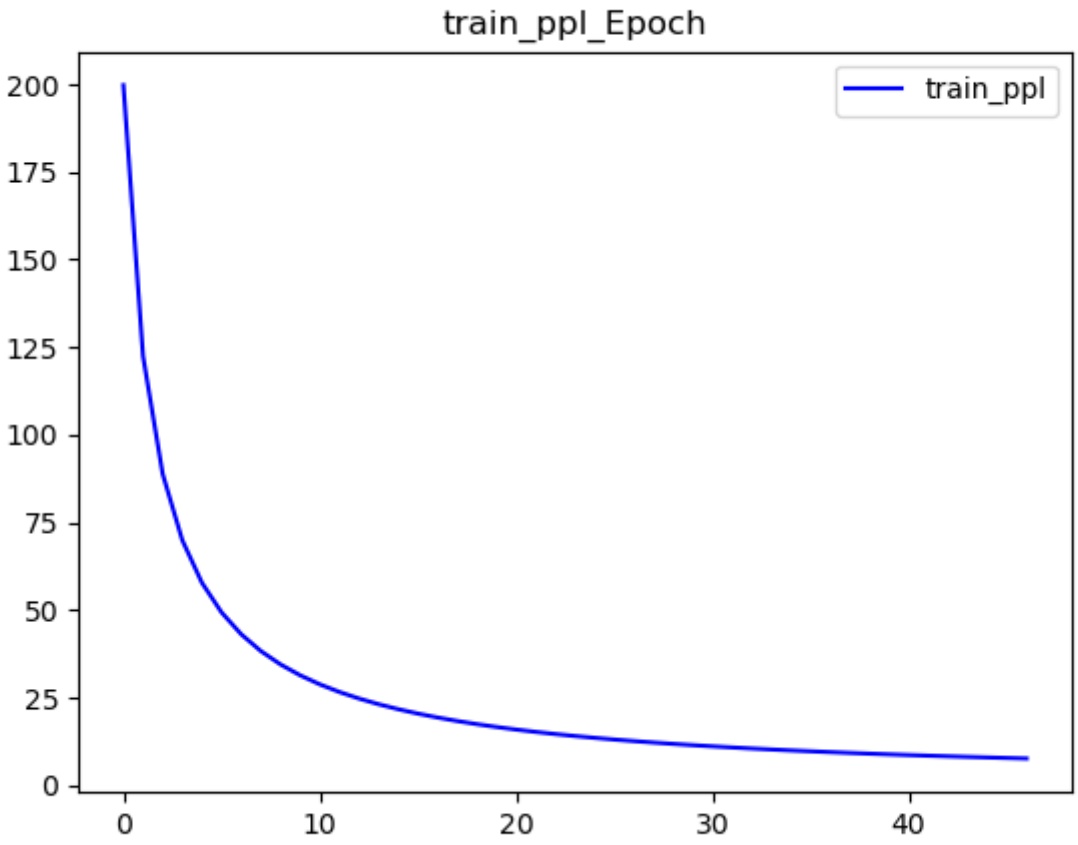
```
RuntimeError: CUDA out of memory. Tried to allocate 280.00 MiB (GPU 0; 11.91
GiB total capacity; 9.12 GiB already allocated; 170.56 MiB free; 11.02 GiB
reserved in total by PyTorch)
```

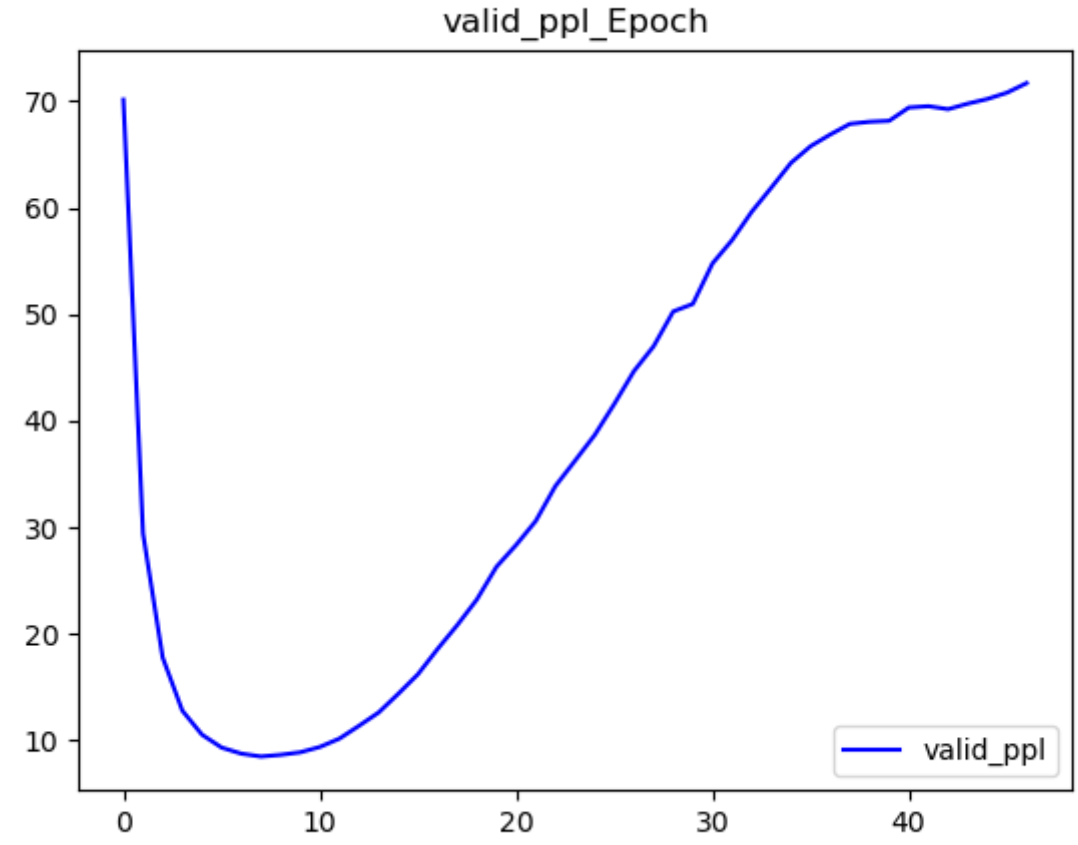
2、运行缓慢

代码在服务器上运行也十分缓慢，不知道是什么原因。最终只出了47代的训练结果。曲线如下图所示，一眼看上去非常像是过拟合，但我分析可能是训练的代数太小了导致的。机器翻译模型训练代数应该在几千以上，在验证集上会出现loss和ppl的波动也十分正常的。


train_Loss_Epoch
train_loss
5
4
3
2
1
0
10
20
30
40


valid_Loss_Epoch
valid_loss
4.0
3.5
3.0
2.5
0
10
20
30
40





3、batch_size反常

由于整个数据集无法使用，我只能随机抽取了几百个句子进行训练，发现一个现象batch_size越小，取32，16等，loss值计算出来更小，而当batch_size取64，128时，loss变得很大。在我查的资料里面提出在GPU承受范围内batch_size的值越大时训练的越好。也有可能是数据量太小引起的。

四、总结

结果是我只把Transformer的理论部分摸清了，实践却什么都没有训练出来。虽然显性Bug都解决了，但可能有一些隐藏Bug导致整个程序运行缓慢。对于这种运行缓慢和内存问题没什么经验，无从下手。整个实验过程非常坎坷，一步一个坎。前期学习过程花费了太多时间，也导致后期实验过程投入的时间不足，没什么产出，效率低下，也许我应该选择更简单一些的任务。整个过程虽然坎坷，但是我对于Transformer理解更加深刻了，而且对于PyTorch的使用更加熟练了，最频繁访问的网站变成了Pytorch官网帮助文档。
说明：由于个人能力限制，Transformer模型部分的代码并非我原创，学习引入了许多人的代码，形成了整个代码。也许正是因为这样才产生了对我来说的隐藏Bug。

五、下一步计划

原创写一份Transformer模型代码，在自己电脑上运行小数据集测试debug。

六、心得体会

虽然结果不好，但是过程还是学到了很多东西，我想应该能为我的科研之路奠定一些基础。从零开始学，内心很焦灼，越焦灼越想拖延，强迫自己一点一点学起来，每学到一点东西，焦虑就少几分，到最终克服这种心理。觉得自己抗压能力很强，以后也不会拖延了，我相信我能做到。

学习参考资源如下：
（有一些找不到了，没贴出来）
60分钟入门PyTorch（一）——Tensors - 知乎 (zhihu.com)
PyTorch深度学习快速入门教程（绝对通俗易懂！）【小土堆】_哔哩哔哩_bilibili
Transformers from scratch | peterbloem.nl
一文搞懂BPE分词算法 - 知乎 (zhihu.com)
深度学习网络调参技巧 - 知乎 (zhihu.com)
Transformer代码完全解读！-技术圈 (proginn.com)
【实战教程】用Pytorch实现Transformer - 简书 (jianshu.com)
手把手教你用Pytorch代码实现Transformer模型（超详细的代码解读）_白马金羁侠少年的博客-CSDN博客_transformer模型代码pytorch
https://github.com/jadore801120/attention-is-all-you-need-pytorch