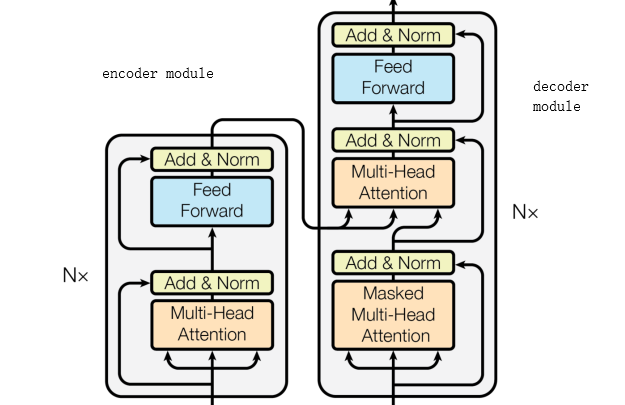
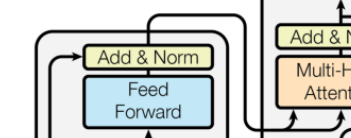
搭载transformer模型的基本代码思路：

一：

依据transformer的框架图，搭载出Tranformer函数：



重点理解其中encoder第二层传入decoder

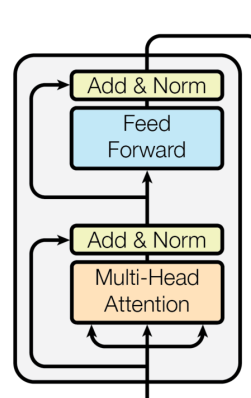


decode(self, memory, src\_mask, tgt, tgt\_mask):

代码中memory是encoder的运算结果传入decoder

二：

依据框架图构件出encoder函数：



在encoder module中

Encoder函数实现6词循环的神经网络大框架（该循环次数和自行定义）实现外部的框架结构，在EncoderLayer中具体定义了muli-head，及feed forward，具体实现函数是

#搭建框架中的attention层和feedforward层  
self.sublayer = clones(SublayerConnection(size, dropout), 2)

在该函数中实现了norm

x = self.sublayer[0](x, lambda x: self.self\_attn(x, x, x, mask))  
# 注意到attn得到的结果x直接作为了下一层的输入  
return self.sublayer[1](x, self.feed\_forward)

三：

定义了sublayer module

分别实现框架图中的 add（残差连接）和norm，及dropout操作。

对应代码分别是

return x + self.dropout(sublayer(self.norm(x)))

LayerNorm(nn.Module):

四：

定义decoder函数

Decoder函数除了mask 和多一层外，同时有一层是通过encoder传入的，具体思路同encoder相同。该处的mask是形成一个三角矩阵，在做train 过程中，不会将attention注意到未求出的之后语句之中。

（decoder的神经网络模块层数）self.sublayer = clones(SublayerConnection(size, dropout), 3)

（encoder的神经网络模块层数）self.sublayer = clones(SublayerConnection(size, dropout), 2)

五：

定义attention module：

Attention 代码中多头注意力不太好理解，具体讲解见李宏毅课程，对应的实现代码是：

query, key, value = \  
 [l(x).view(nbatches, -1, self.h, self.d\_k).transpose(1, 2)

这里，相当于将一个语句，拆分为多个块。分别做attention。该处具体原因是，实测效果较好，因为能够捕获到语句中各自单词见的关系，具体讲解见李宏毅课程中，一笔带过。没有严格的数学理论支撑。

#此处执行多头注意力模型将数据从[14, 5, 256]变为[14, 5, 8, 32]

这里通过调用

subsequent\_mask

来实现mask，其中的调用较为复杂，层层嵌套。可从run中一步步用找函数的调用。

六：

定义embedding 这里transformer中通过

pe[:, 0::2] = torch.sin(position \* div\_term)  
pe[:, 1::2] = torch.cos(position \* div\_term)

x = x + Variable(self.pe[:, :x.size(1)], requires\_grad=False)

增加了位置对训练的影响，实现，具体讲解内容见李宏毅课程中的讲解。

七：

Generator函数实现将训练好的词向量映射为单词，选择概率最大的单词作为结果，使用greed search

八：

evaluate函数

通过调用训练好的模型，直接生成翻译结果。

参考：

李宏毅课程：

<https://www.youtube.com/watch?v=ugWDIIOHtPA&t=2034s>

哈佛NLP论坛文章及代码：

<https://nlp.seas.harvard.edu/2018/04/03/attention.html>

Gready search

https://blog.csdn.net/weixin\_42615068/article/details/93767781 ，

该步骤每步解码后，找出最大的概率再找下一个最大的。