BERT

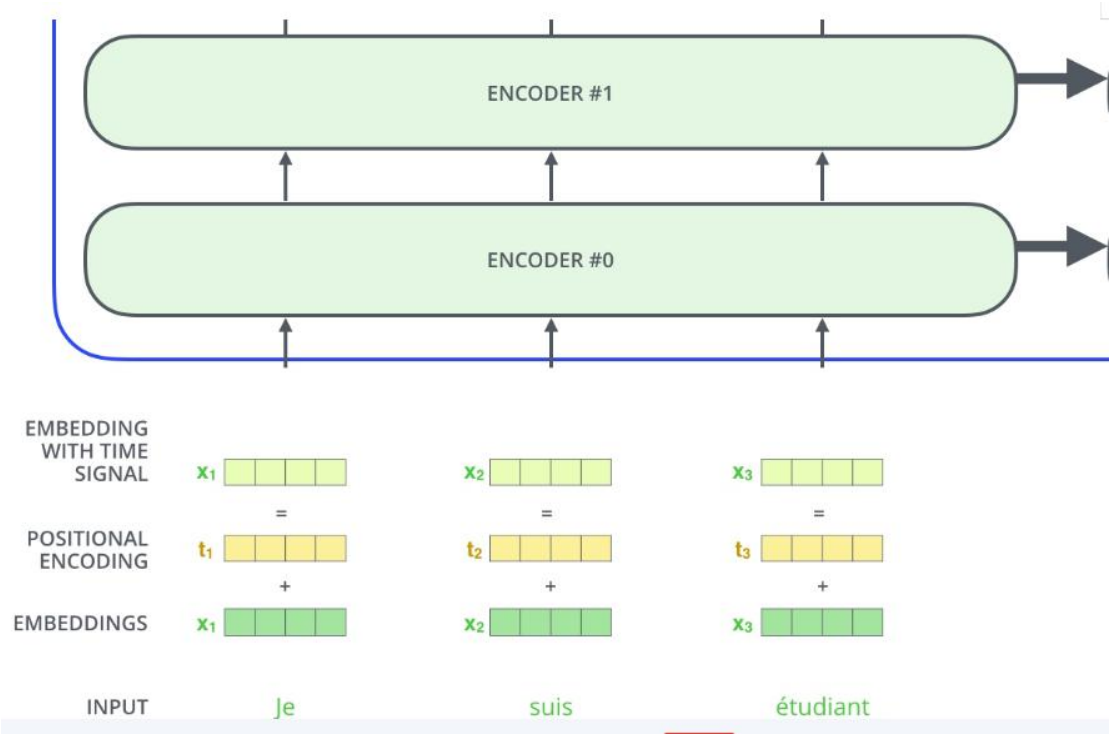
Positional Encoding

到目前为止，transformer 模型中还缺少一种解释输入序列中单词顺序的方法。为了处理这个问题，transformer 给 encoder 层和 decoder 层的输入添加了一个额外的向量 Positional Encoding，维度和 embedding 的维度一样，这个向量采用了一种很独特的方法来让模型学习到这个值，这个向量能决定当前词的位置，或者说在一个句子中不同的词之间的距离。这个位置向量的具体计算方法有很多种，计算方法如下：

$$PE(pos, 2i) = sin(pos/10000^{2i}/d_model)$$

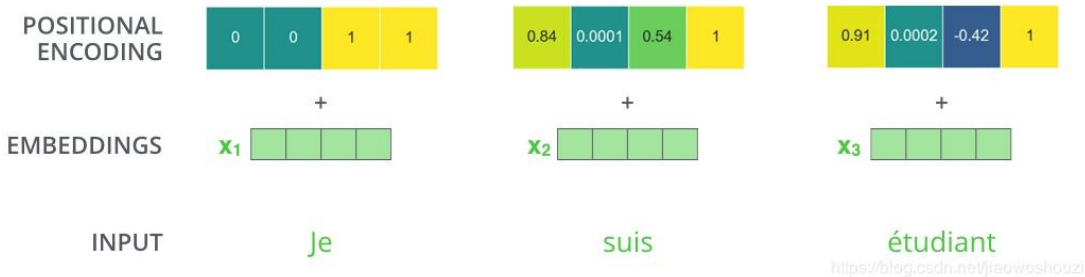
$$PE(pos, 2i+1) = cos(pos/10000^{2i}/d_model)$$

其中 pos 是指当前词在句子中的位置，i 是指向量中每个值的 index，可以看出，在偶数位置，使用正弦编码，在奇数位置，使用余弦编码。最后把这个 Positional Encoding 与 embedding 的值相加，作为输入送到下一层。



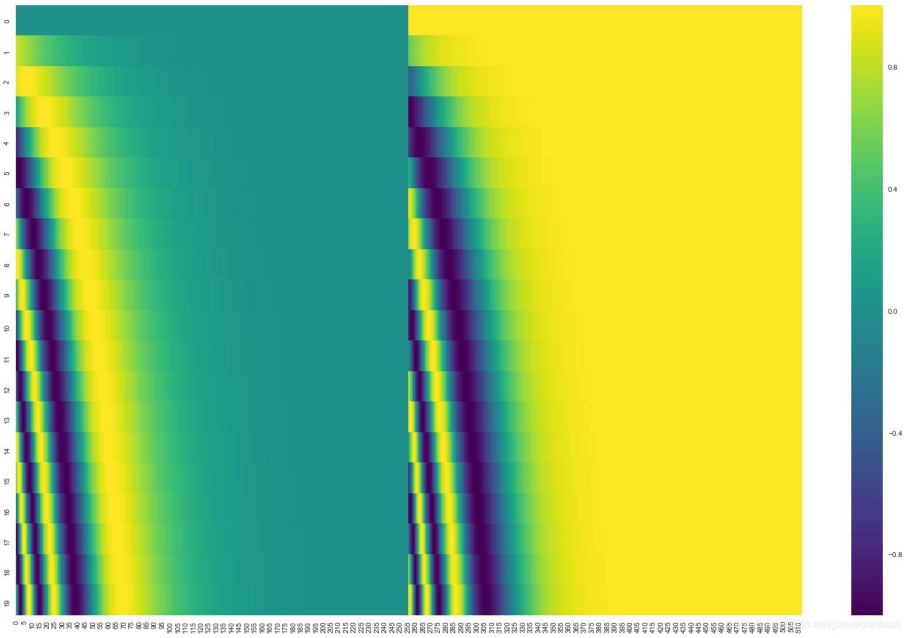
为了让模型捕捉到单词的顺序信息，我们添加位置编码向量信息（POSITIONAL ENCODING），位置编码向量不需要训练，它有一个规则的产生方式（上图公式）。

如果我们的嵌入维度为 4，那么实际上的位置编码就如下图所示：



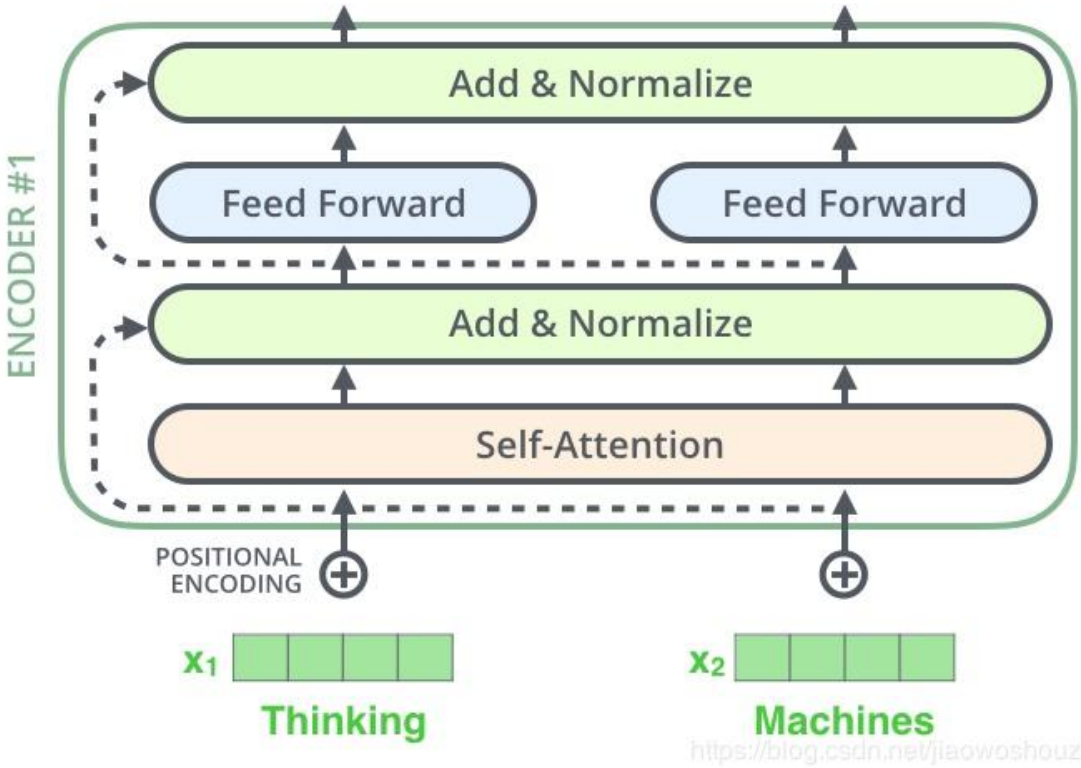
那么生成位置向量需要遵循怎样的规则呢?

观察下面的图形，每一行都代表着对一个矢量的位置编码。因此第一行就是我们输入序列中第一个字的嵌入向量，每行都包含 512 个值，每个值介于 1 和-1 之间。我们用颜色来表示 1，-1 之间的值，这样方便可视化的方式表现出来:

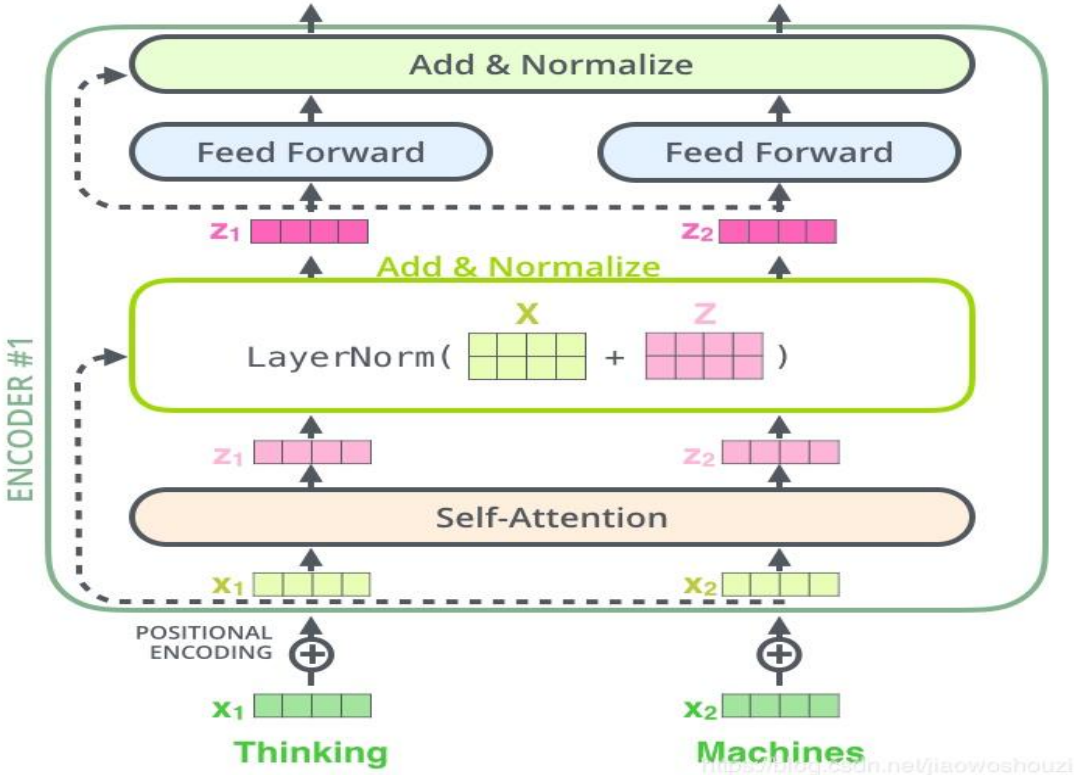


这是一个 20 个字（行）的（512）列位置编码示例。你会发现它咋中心位置被分为了 2 半，这是因为左半部分的值是一由一个正弦函数生成的，而右半部分是由另一个函数（余弦）生成。然后将它们连接起来形成每个位置编码矢量。

Layer normalization
在 transformer 中，每一个子层（self-attetion，ffnn）之后都会接一个残差模块，并且有一个 Layer normalization



在进一步探索其内部计算方式，我们可以将上面图层可视化为下图:

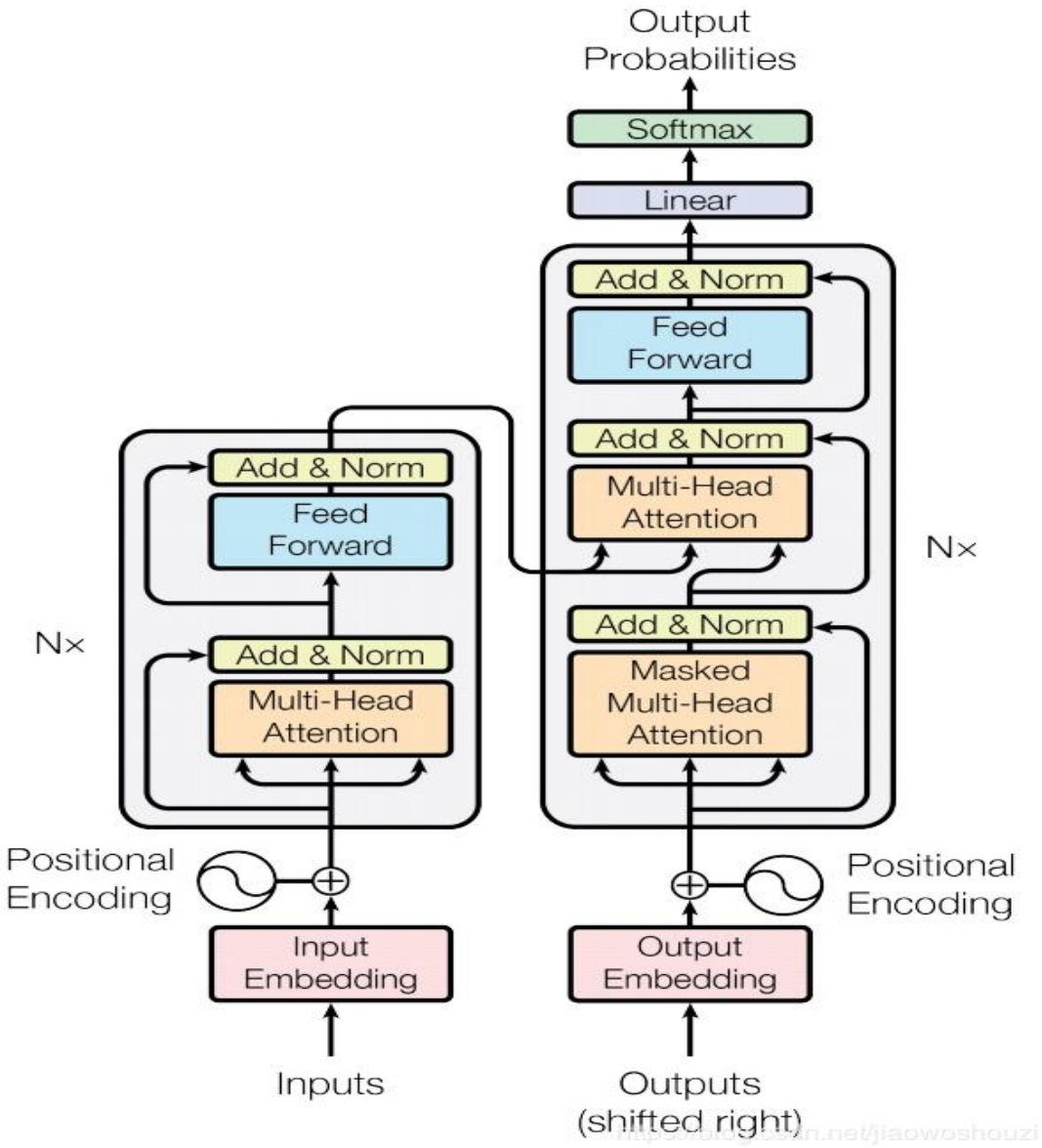


残差模块相信大家都很清楚了，这里不再讲解，主要讲解下 Layer normalization。Normalization 有很多种，但是它们都有一个共同的目的，那就是把输入转化成均值为 0 方差为 1 的数据。我们在把数据送入激活函数之前进行 normalization（归一化），因为我们不

希望输入数据落在激活函数的饱和区。

说到 normalization，那就肯定得提到 Batch Normalization。BN 的主要思想就是：在每一层的每一批数据上进行归一化。我们可能会对输入数据进行归一化，但是经过该网络层的作用后，我们的数据已经不再是归一化的了。随着这种情况的发展，数据的偏差越来越大，我的反向传播需要考虑到这些大的偏差，这就迫使我们只能使用较小的学习率来防止梯度消失或者梯度爆炸。

Decoder 层



上图是 transformer 的一个详细结构，相比本文一开始结束的结构图会更详细些，接下来，我们会按照这个结构图讲解下 decoder 部分。

可以看到 decoder 部分其实和 encoder 部分大同小异，不过在最下面额外多了一个 masked mutil-head attetion，这里的 mask 也是 transformer 一个很关键的技术，我们一起来看一下。

Mask

mask 表示掩码，它对某些值进行掩盖，使其在参数更新时不产生效果。Transformer 模型里面涉及两种 mask，分别是 padding mask 和 sequence mask。

其中，padding mask 在所有的 scaled dot-product attention 里面都需要用到，而 sequence mask 只有在 decoder 的 self-attention 里面用到。

打卡完毕

参考文档：

https://blog.csdn.net/sunhua93/article/details/102764783
https://zhuanlan.zhihu.com/p/43493999
https://blog.csdn.net/jiaowoshouzi/article/details/89073944
https://github.com/Separius/BERT-keras