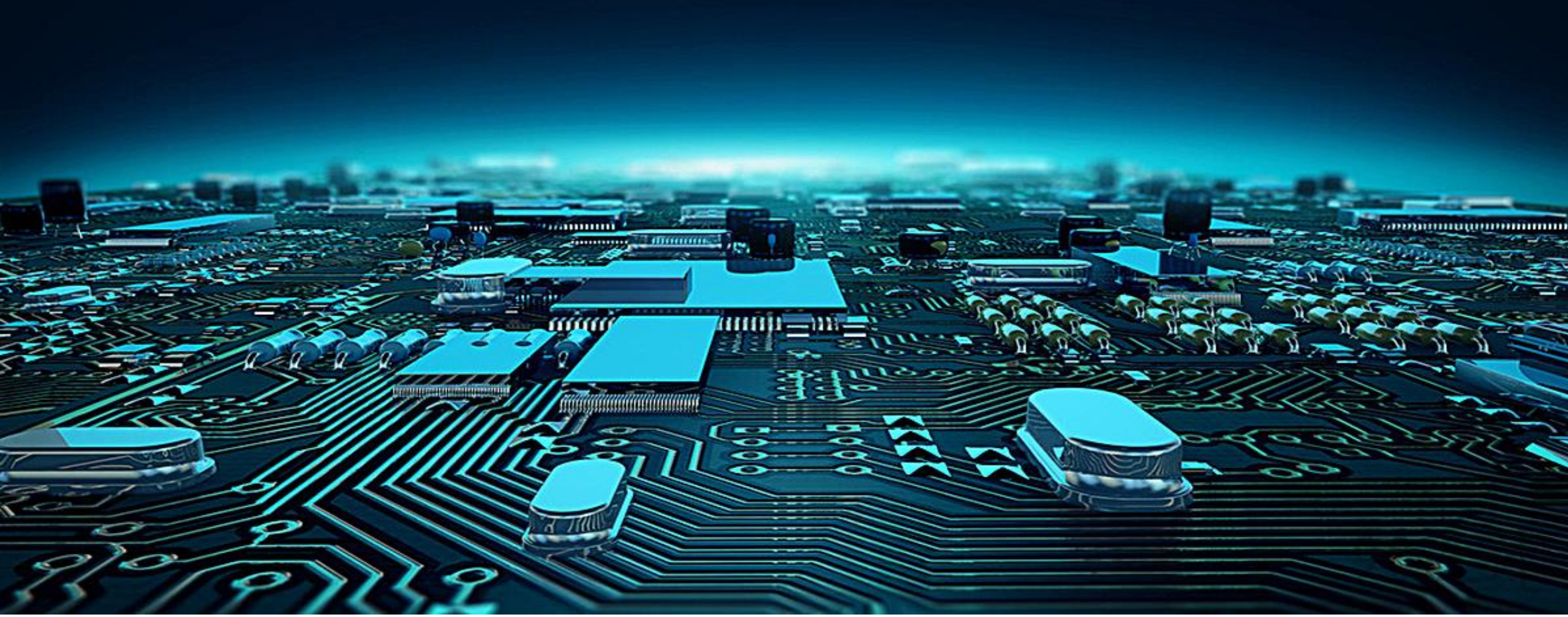
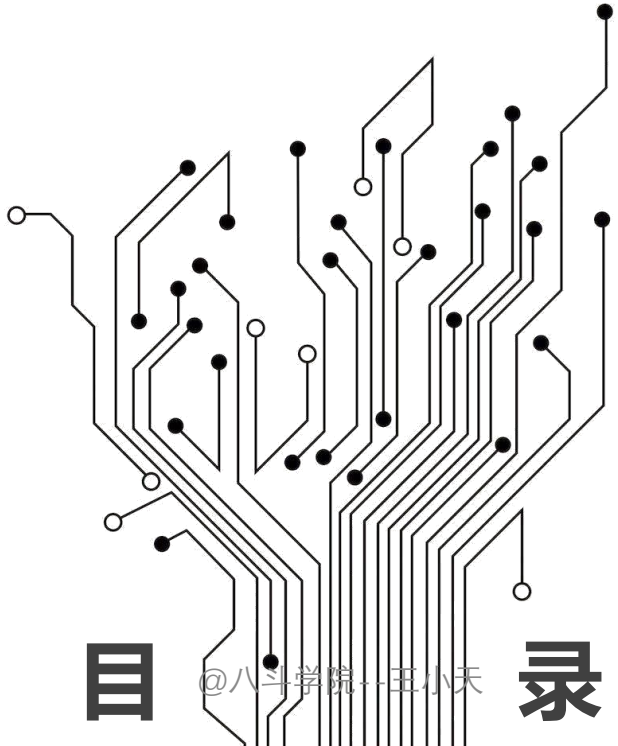


Transformer&Bert

@八斗学院--王小天(Michael)

2020/5/24

1. Transformer
2. Bert



目　录

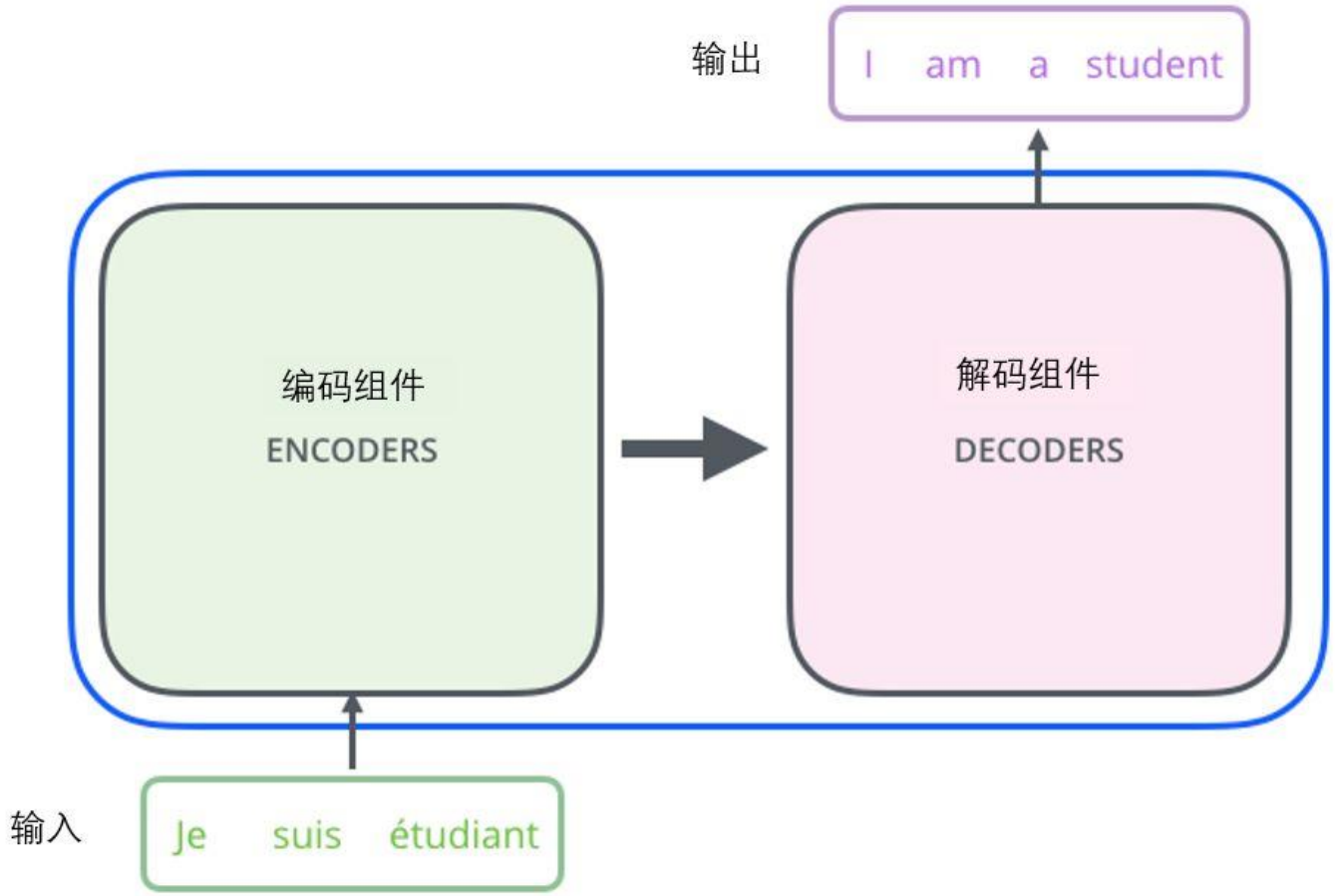
Transformer

Transformer由论文《Attention is All You Need》提出，现在是谷歌云TPU推荐的参考模型。

Transformer改进了RNN最被人诟病的训练慢的缺点，利用self-attention机制实现快速并行。并且Transformer可以增加到非常深的深度，充分发掘DNN模型的特性，提升模型准确率。

首先将这个模型看成是一个黑箱操作。在机器翻译中，就是输入一种语言，输出另一种语言。

那么拆开这个黑箱，它是由编码组件、解码组件和它们之间的连接组成。



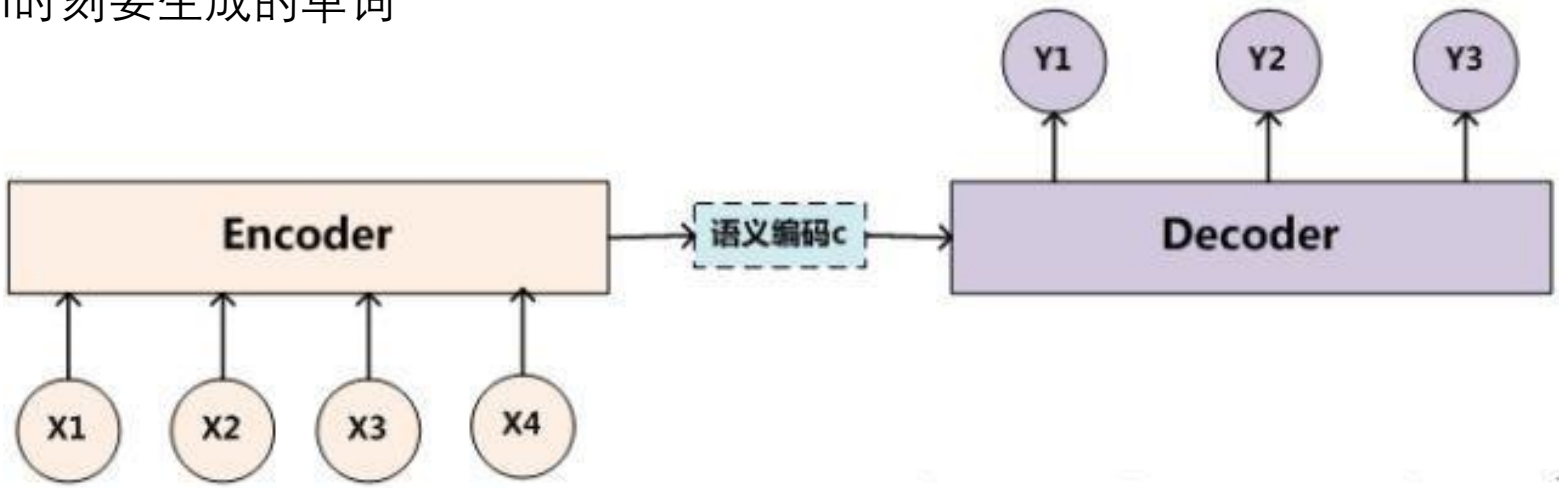
Transformer

文本处理领域的Encoder-Decoder框架可以这么直观地去理解：可以把它看作适合处理由一个句子（或篇章）生成另外一个句子（或篇章）的通用处理模型。

对于句子对<Source，Target>，我们的目标是给定输入句子Source，期待通过Encoder-Decoder框架来生成目标句子Target。Source和Target可以是同一种语言，也可以是两种不同的语言。

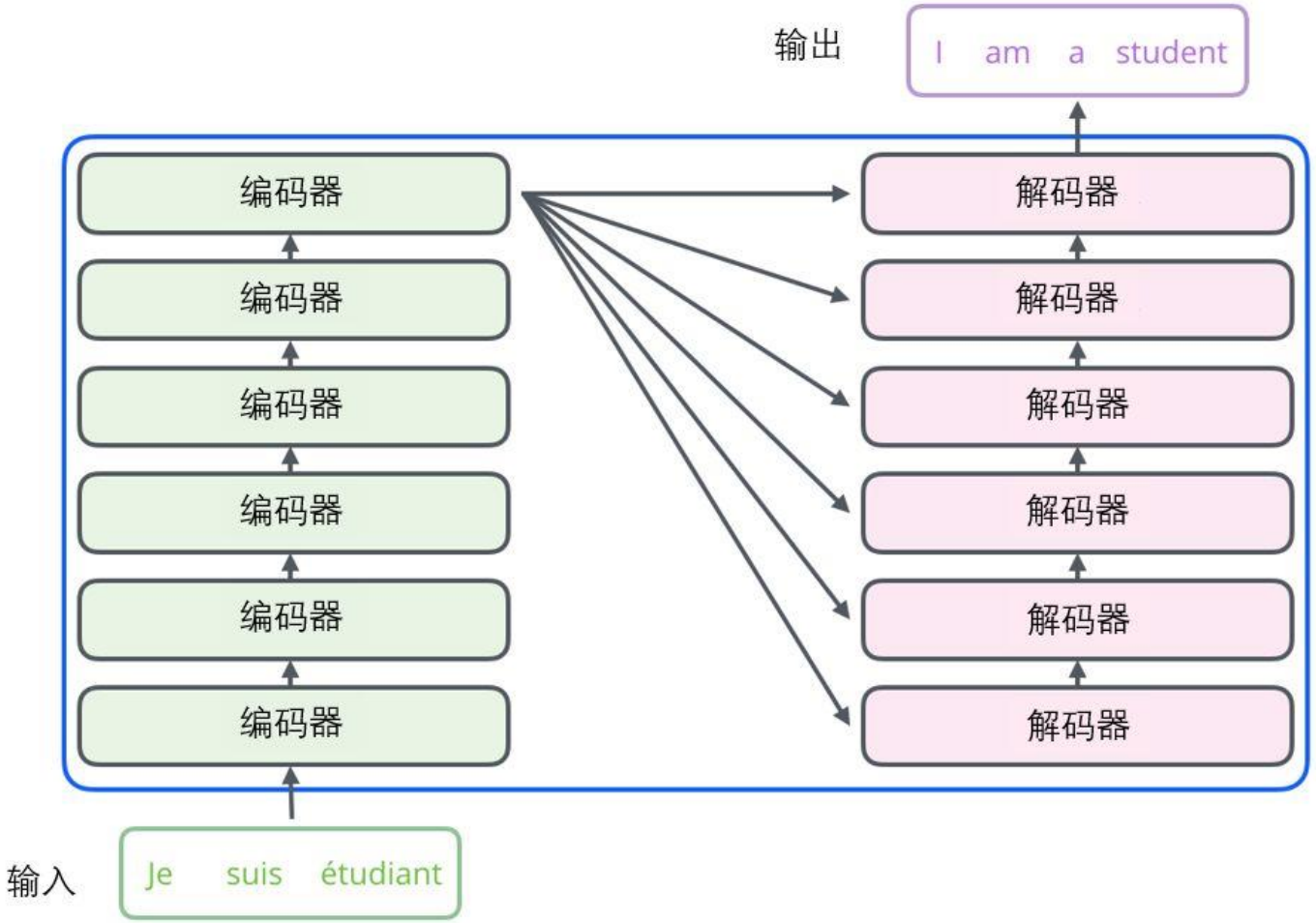
- Encoder顾名思义就是对输入句子Source进行编码，将输入句子通过非线性变换转化为中间语义表示C。

- 对于解码器Decoder来说，其任务是根据句子Source的中间语义表示C和之前已经生成的历史信息来生成i时刻要生成的单词



Transformer

编码组件部分由一堆编码器（encoder）构成。解码组件部分也是由相同数量（与编码器对应）的解码器（decoder）组成的。

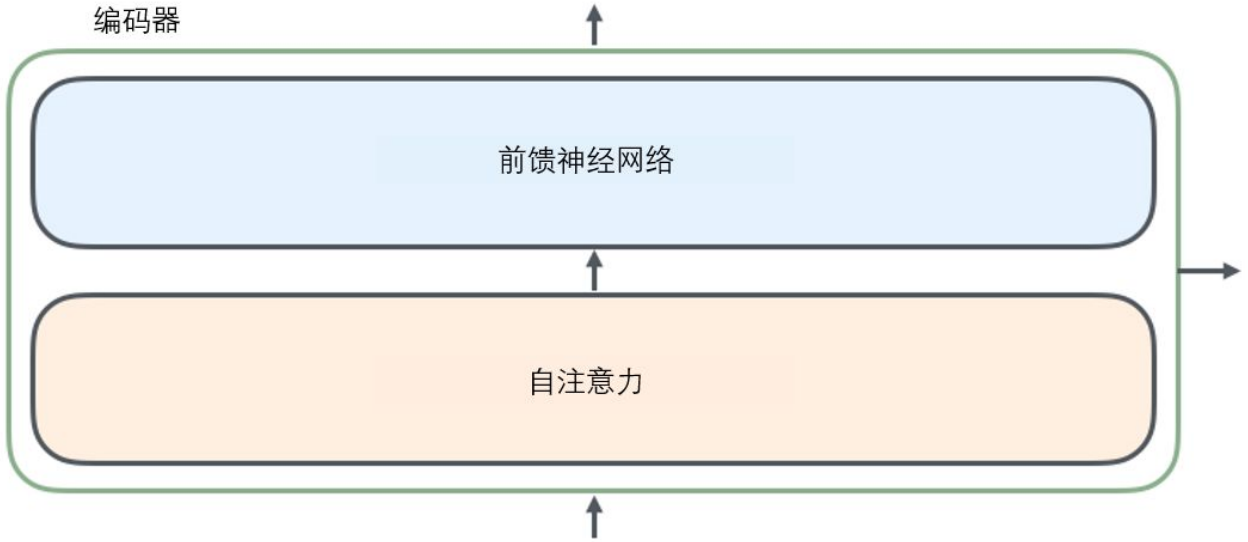


Transformer

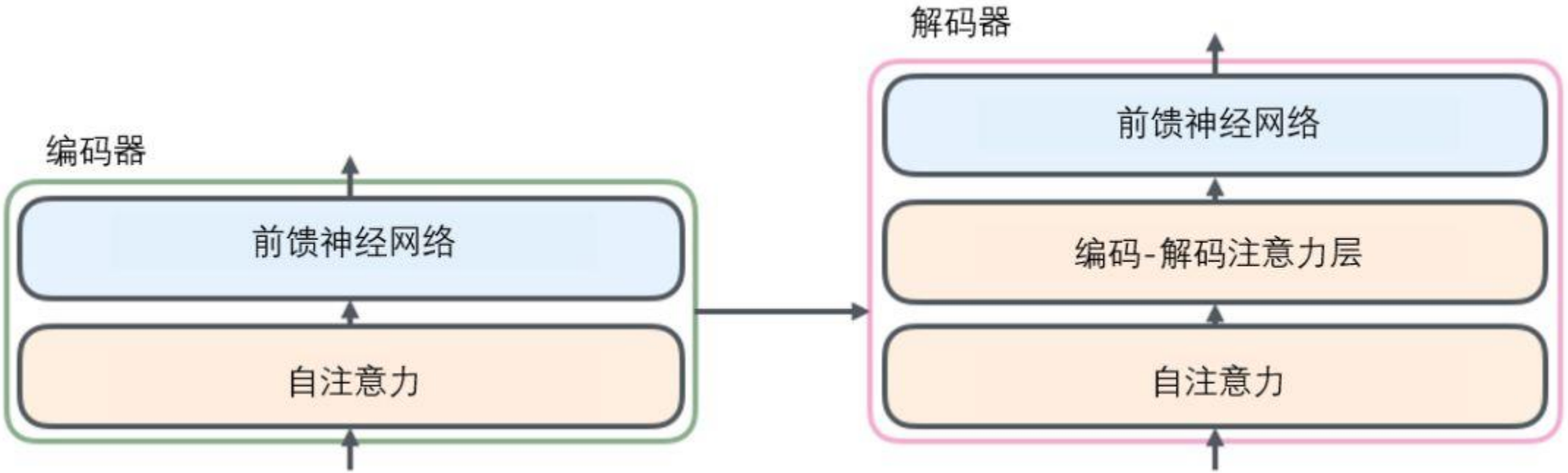
所有的编码器在结构上都是相同的，但它们没有共享参数。每个编码器都可以分解成两个子层。

从编码器输入的句子首先会经过一个自注意力（self-attention）层，这层帮助编码器在对每个单词编码时关注输入句子的其他单词。

自注意力层的输出会传递到前馈（feed-forward）神经网络中。每个位置的单词对应的前馈神经网络都完全一样。

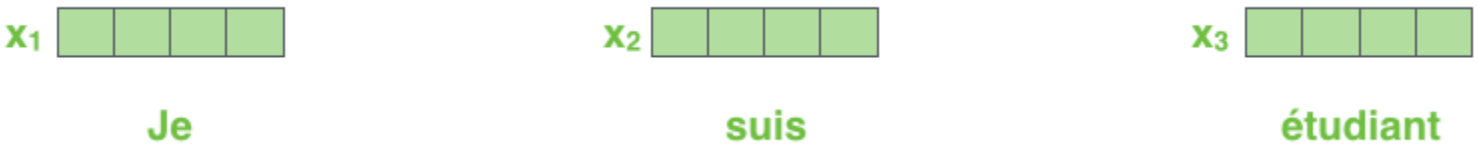


解码器中也有编码器的自注意力（self-attention）层和前馈（feed-forward）层。除此之外，这两个层之间还有一个注意力层，用来关注输入句子的相关部分。

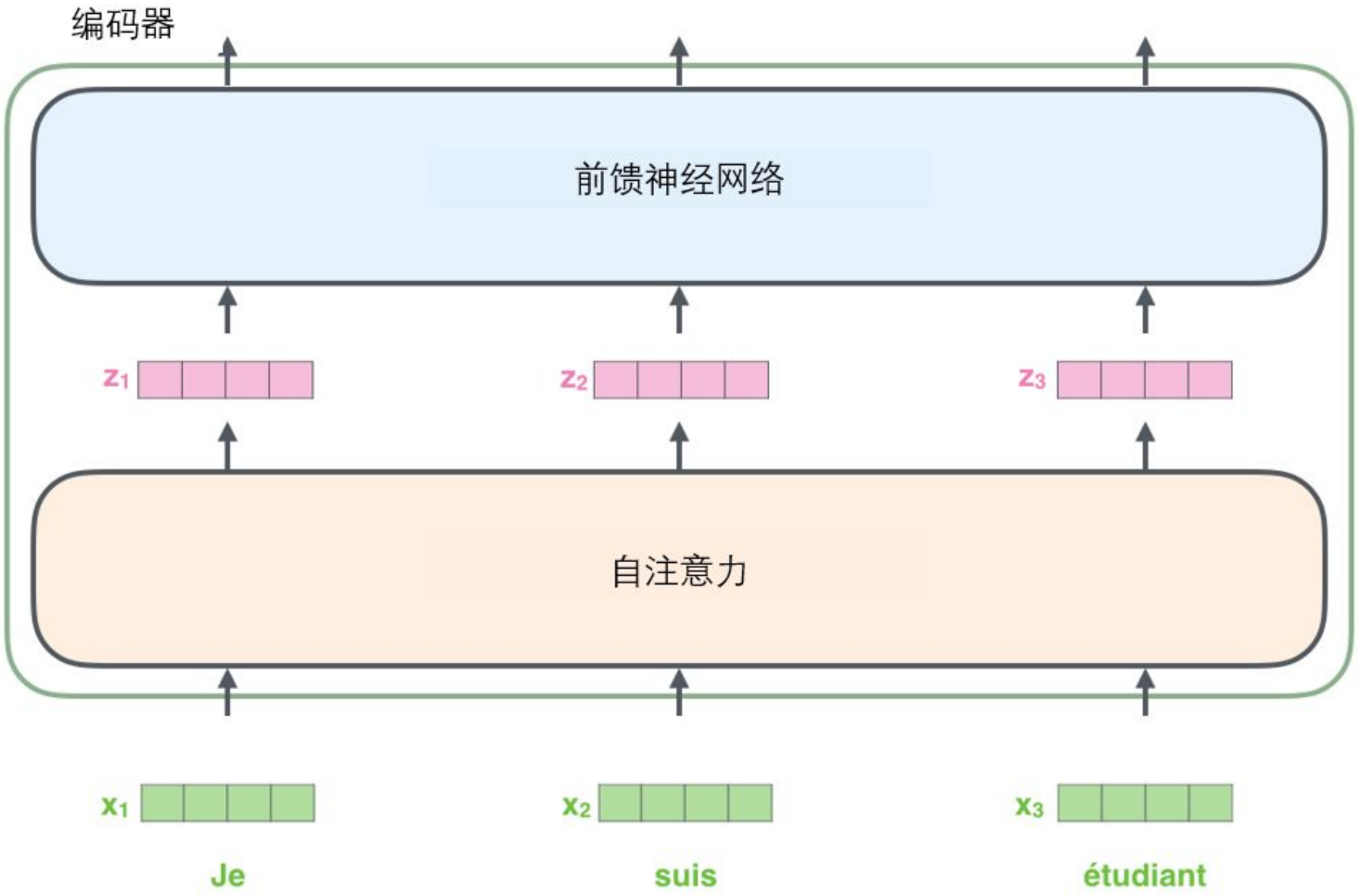


将张量引入图景

- 像大部分NLP应用一样，我们首先将每个输入单词通过词嵌入算法转换为词向量。
- 每个单词都被嵌入为512维的向量，我们用这些简单的方框来表示这些向量。
- 词嵌入过程只发生在最底层的编码器中。所有的编码器都有一个相同的特点，即它们接收一个向量列表，列表中的每个向量大小为512维。在底层（最开始）编码器中它就是词向量，但是在其他编码器中，它就是上一个编码器的输出（也是一个向量列表）。向量列表大小是我们可以设置的超参数——一般是我们训练集中最长句子的长度。

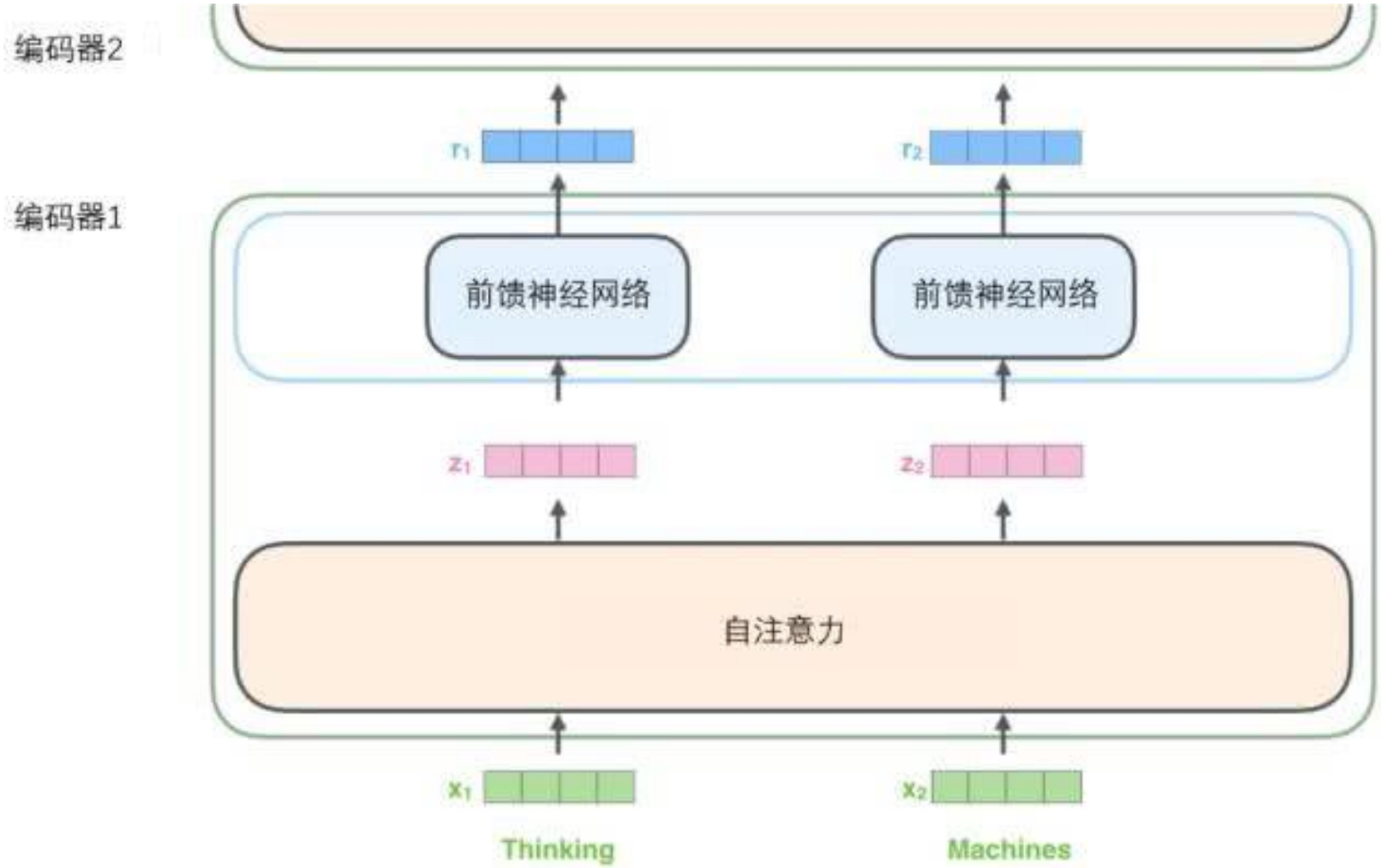


将输入序列进行词嵌入之后，每个单词都会流经编码器中的两个子层。



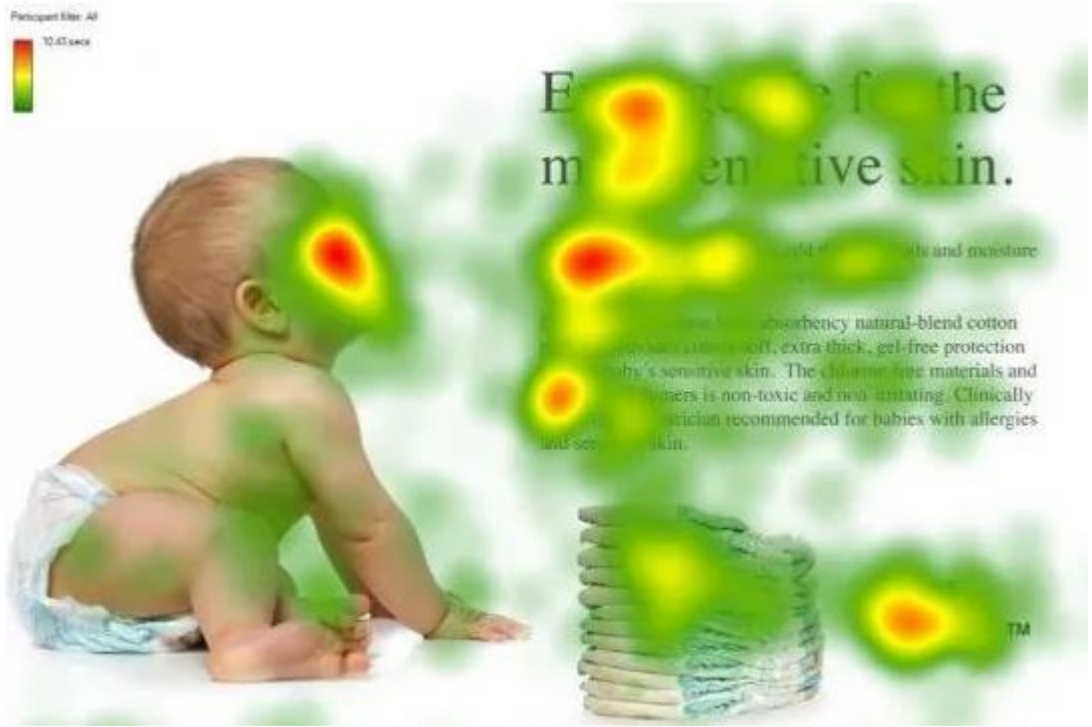
Transformer

输入序列的每个单词都经过自编码过程。然后，他们各自通过前向传播神经网络——完全相同的网络，而每个向量都分别通过它。



Transformer-自注意力机制

- 视觉注意力机制是人类视觉所特有的大脑信号处理机制。人类视觉通过快速扫描全局图像，获得需要重点关注的目标区域，也就是一般所说的注意力焦点，而后对这一区域投入更多注意力资源，以获取更多所需要关注目标的细节信息，而抑制其他无用信息。
- 这是人类利用有限的注意力资源从大量信息中快速筛选出高价值信息的手段，是人类在长期进化中形成的一种生存机制，人类视觉注意力机制极大地提高了视觉信息处理的效率与准确性。
- 深度学习中的注意力机制从本质上讲和人类的选择性视觉注意力机制类似，核心目标也是从众多信息中选择出对当前任务目标更关键的信息。



Tom chase Jerry

在翻译“杰瑞”这个中文单词的时候，分心模型（没有注意力机制）里面的每个英文单词对于翻译目标单词“杰瑞”贡献是相同的，很明显这里不太合理，显然“Jerry”对于翻译成“杰瑞”更重要，但是分心模型是无法体现这一点的。

没有引入注意力的模型在输入句子比较短的时候问题不大，但是如果输入句子比较长，此时所有语义完全通过一个中间语义向量来表示，单词自身的信息已经消失，可想而知会丢失很多细节信息，这也是为何要引入注意力模型的重要原因。

上面的例子中，如果引入Attention模型的话，应该在翻译“杰瑞”的时候，体现出英文单词对于翻译当前中文单词不同的影响程度，比如给出类似下面一个概率分布值：

(Tom,0.3) (Chase,0.2) (Jerry,0.5)

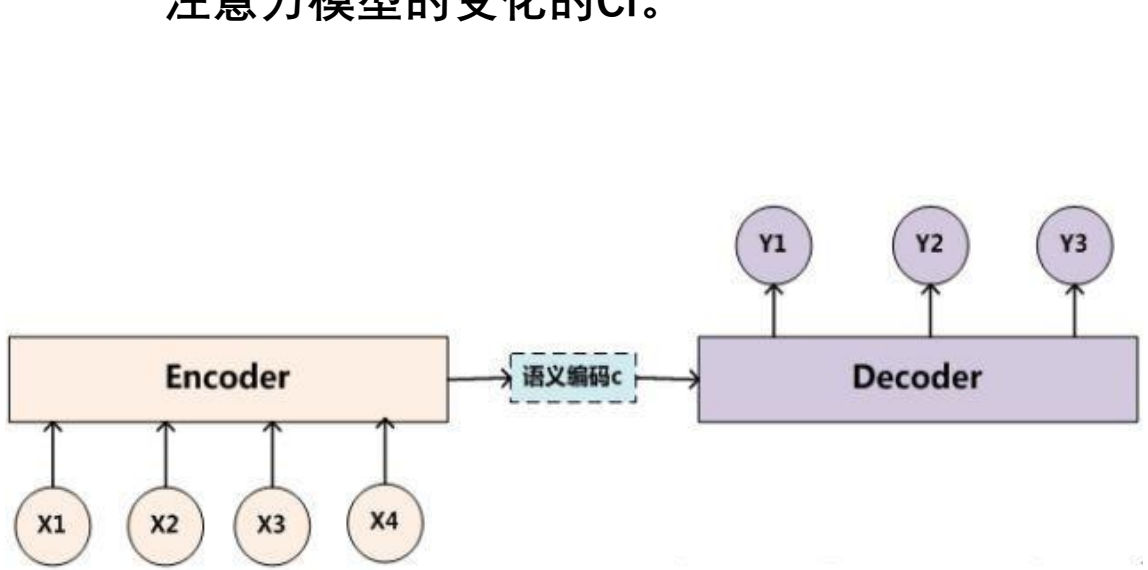
每个英文单词的概率代表了翻译当前单词“杰瑞”时，注意力分配模型分配给不同英文单词的注意力大小。这对于正确翻译目标语单词肯定是有帮助的，因为引入了新的信息。

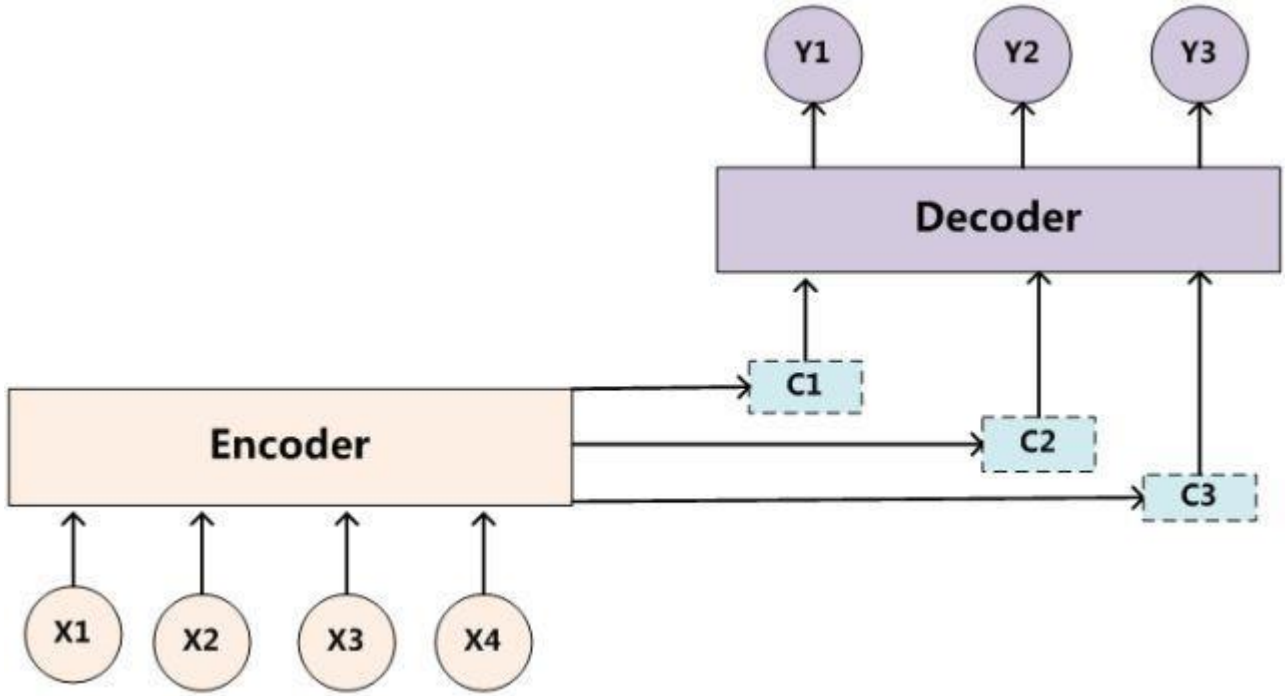
Transformer-自注意力机制

同理，目标句子中的每个单词都应该学会其对应的源语句子中单词的注意力分配概率信息。

这意味着在生成每个单词的时候，原先都是相同的中间语义表示C会被替换成根据当前生成单词而不断变化的Ci。

理解Attention模型的关键就是这里，**即由固定的中间语义表示C换成了根据当前输出单词来调整成加入注意力模型的变化的Ci。**





Transformer-自注意力机制

而每个Ci可能对应着不同的源语句子单词的注意力分配概率分布，比如对于上面的英汉翻译来说，其对应的信息可能如下：

$$C_{汤姆} = g(0.6 * f2("Tom"), 0.2 * f2(Chase), 0.2 * f2("Jerry"))$$

$$C_{追逐} = g(0.2 * f2("Tom"), 0.7 * f2(Chase), 0.1 * f2("Jerry"))$$

$$C_{杰瑞} = g(0.3 * f2("Tom"), 0.2 * f2(Chase), 0.5 * f2("Jerry"))$$

The animal didn’ t cross the street because it was too tired

这个“it”在这个句子是指什么呢？它指的是street还是这个animal呢？这对于人类来说是一个简单的问题，但是对于算法则不是。

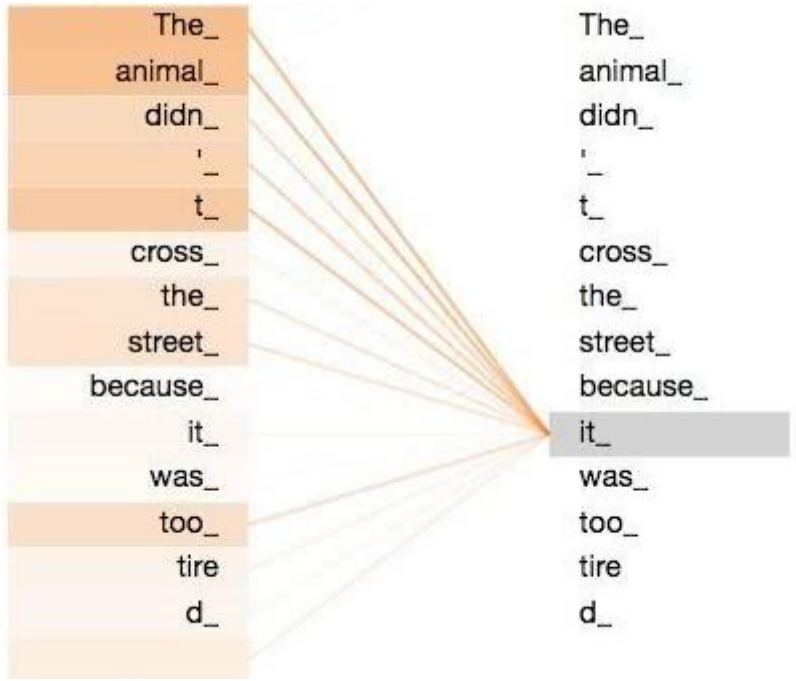
当模型处理这个单词“it”的时候，自注意力机制会允许“it”与“animal”建立联系。

随着模型处理输入序列的每个单词，自注意力会关注整个输入序列的所有单词，帮助模型对本单词更好地进行编码。

RNN会将它已经处理过的前面的所有单词/向量的表示与它正在处理的当前单词/向量结合起来。而自注意力机制会将所有相关单词的理解融入到我们正在处理的单词中。

Transformer-自注意力机制

当我们在编码器中编码“it”这个单词的时，注意力机制的部分会去关注“The Animal”，将它的表示的一部分编入“it”的编码中。



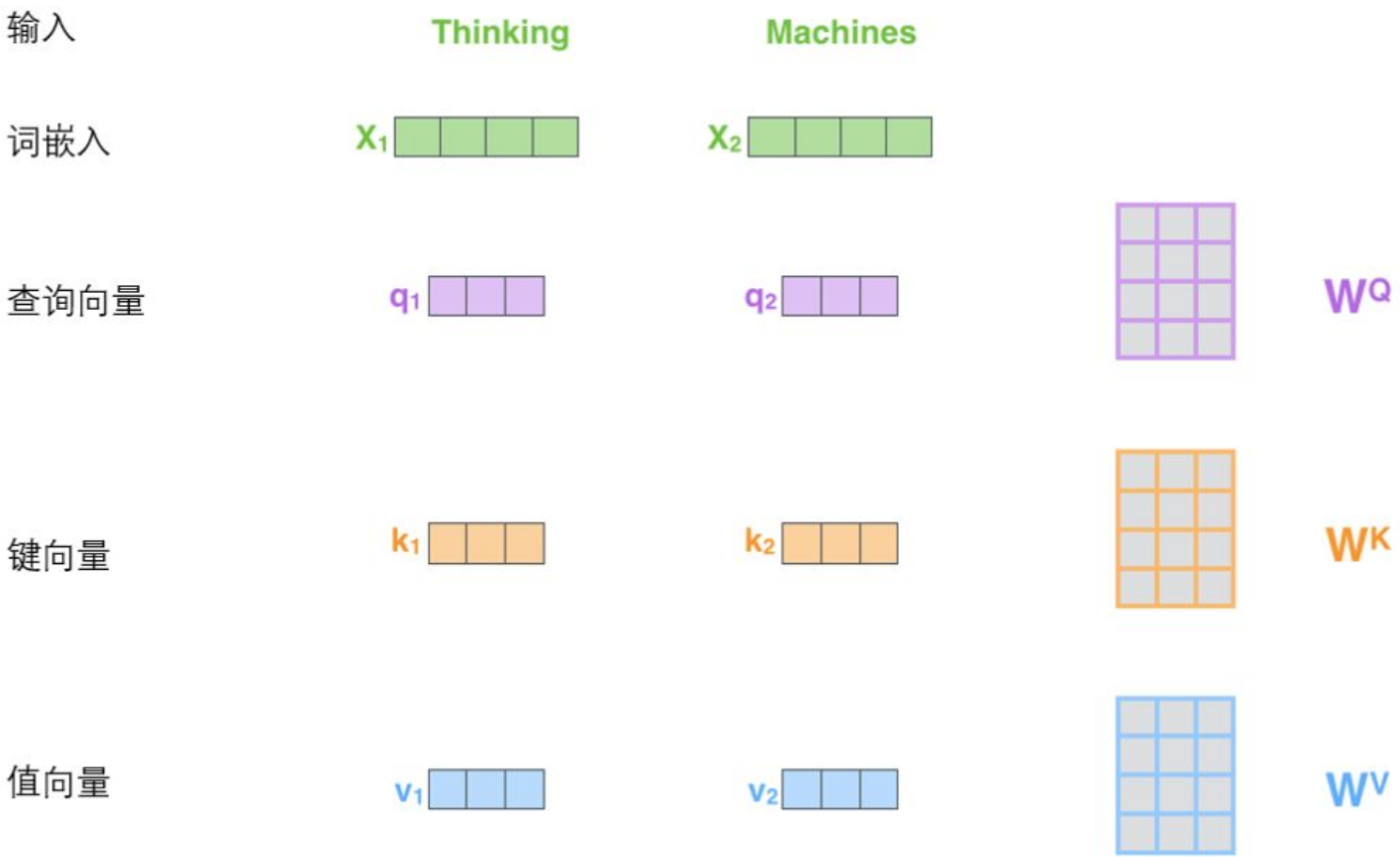
计算自注意力的第一步就是从每个编码器的输入向量（每个单词的词向量）中生成三个向量。也就是说对于每个单词，我们创造**一个查询向量、一个键向量和一个值向量**。

这三个向量是通过词嵌入与三个权重矩阵相乘后创建的。

可以发现这些新向量在维度上比词嵌入向量更低。他们的维度是64，而词嵌入和编码器的输入/输出向量的维度是512。但实际上不强求维度更小，这只是一种基于架构上的选择。

Transformer-自注意力机制

X1与WQ权重矩阵相乘得到q1，就是与这个单词相关的查询向量。最终使得输入序列的每个单词创建一个查询向量、一个键向量和一个值向量。



Transformer-自注意力机制

什么是查询向量、键向量和值向量？

它们都是有助于计算和理解注意力机制的抽象概念：

给一个Q(query)，通过Q和一组K（key）的关系，决定哪些V(value)重要

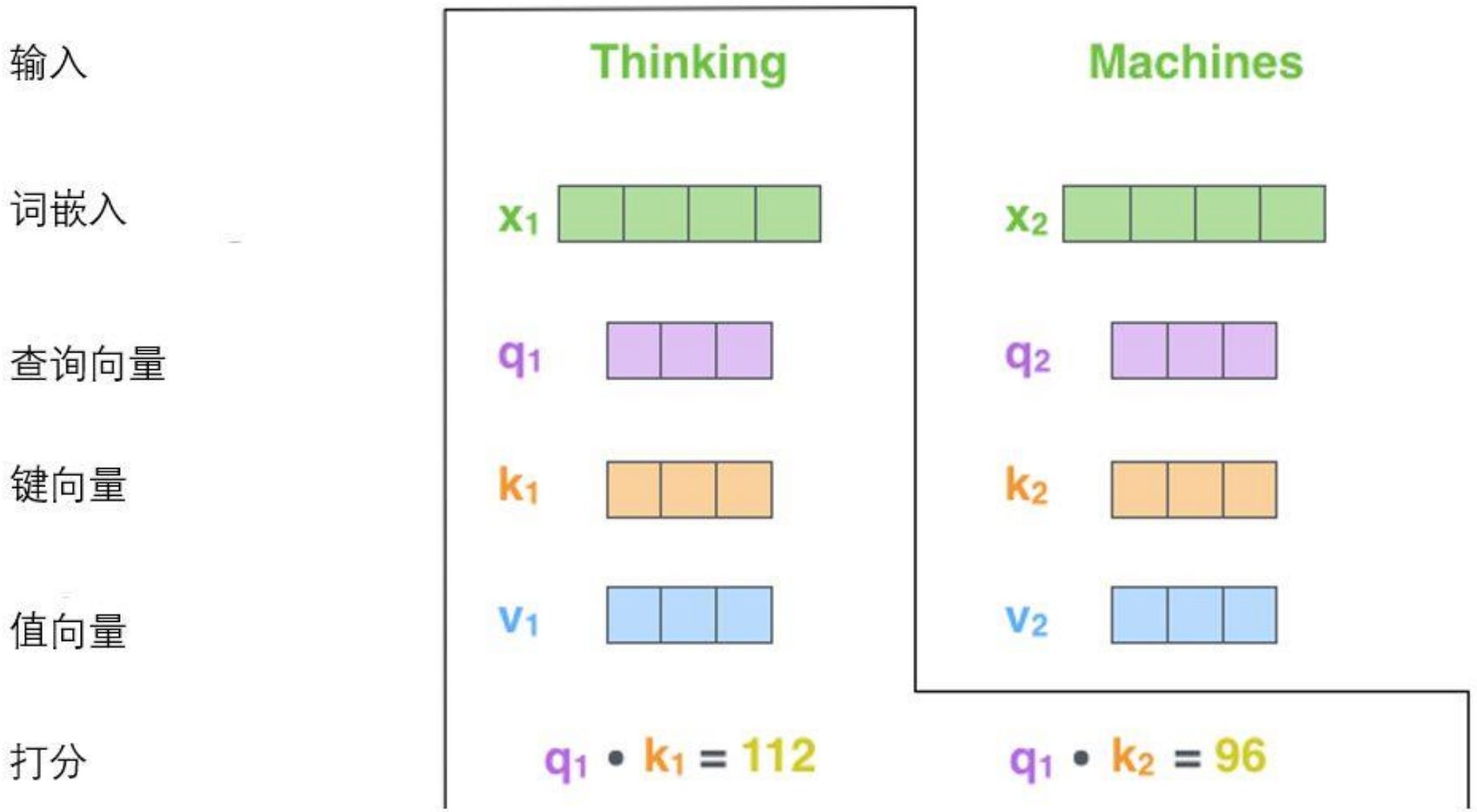
计算自注意力的第二步是计算得分。

假设我们在为这个例子中的第一个词“Thinking”计算自注意力向量，我们需要拿输入句子中的每个单词对“Thinking”打分。

这些分数决定了在编码单词“Thinking”的过程中有多重视句子的其它部分。

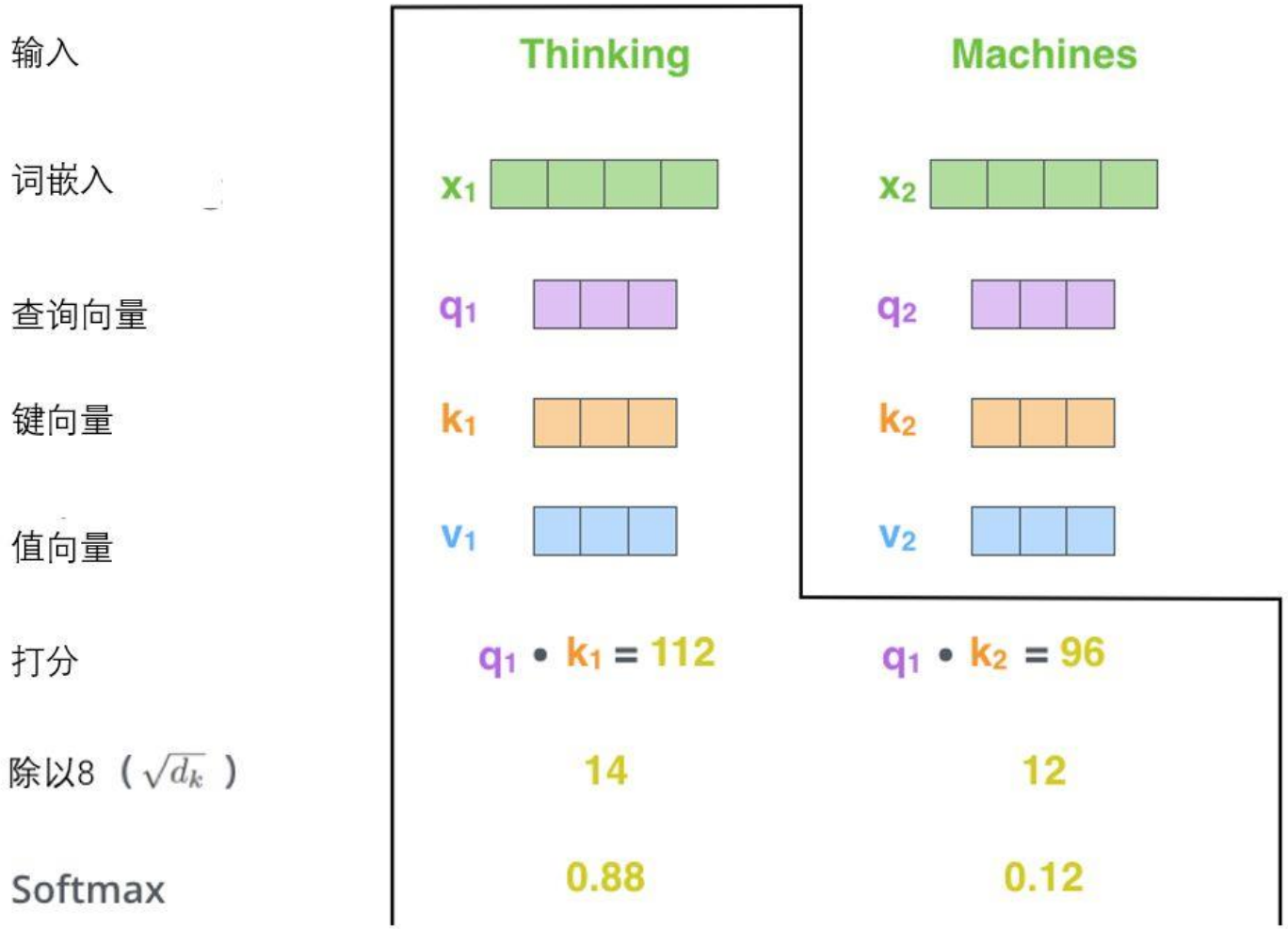
Transformer-自注意力机制

这些分数是通过打分单词（所有输入句子的单词）的键向量与“Thinking”的查询向量相点积来计算的。所以如果我们是处理位置最靠前的词的自注意力的话，第一个分数是q1和k1的点积，第二个分数是q1和k2的点积。



Transformer-自注意力机制

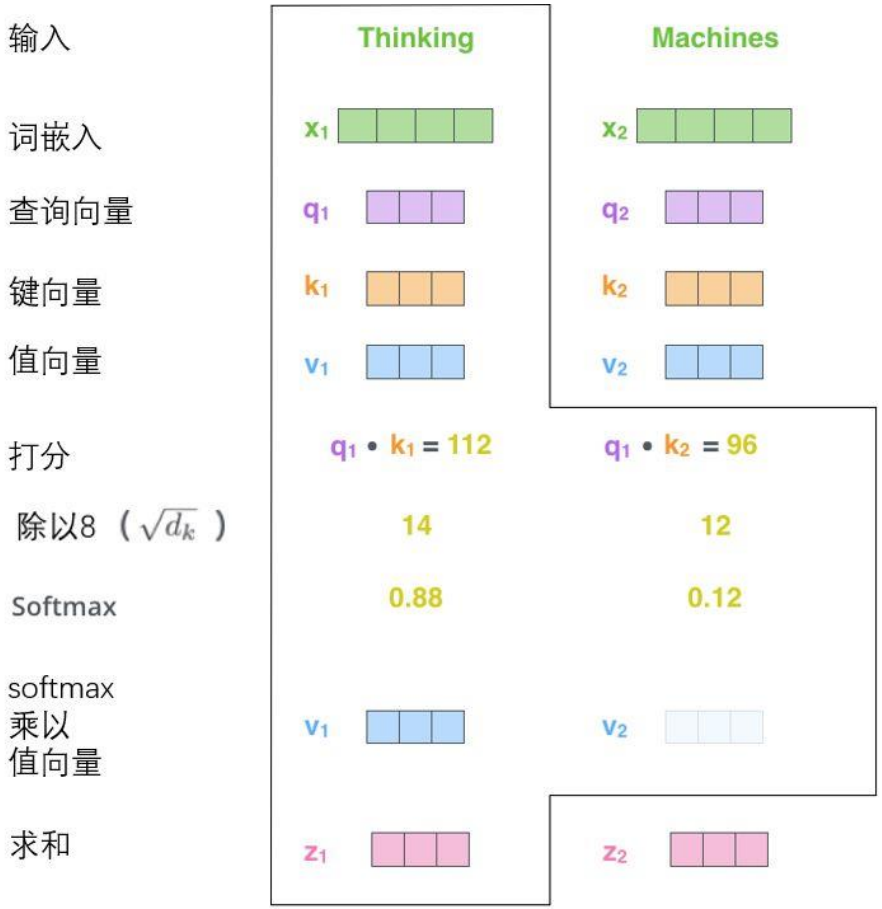
第三步和第四步是将分数除以8(8是论文中使用的键向量的维数64的平方根，这会让梯度更稳定。这里也可以使用其它值，8只是默认值)，然后通过softmax传递结果。softmax的作用是使所有单词的分数归一化，得到的分数都是正值且和为1。



Transformer-自注意力机制

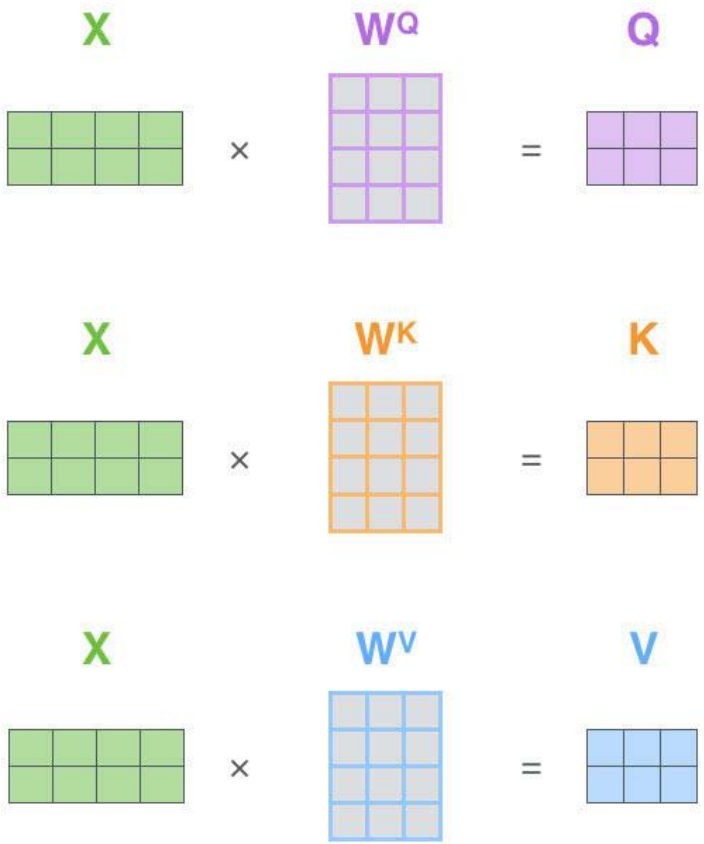
第五步是将每个值向量乘以softmax分数(这是为了准备之后将它们求和)。这里的直觉是希望关注语义上相关的单词，并弱化不相关的单词(例如，让它们乘以0.001这样的小数)。

第六步是对加权值向量求和（自注意力的另一种解释就是在编码某个单词时，就是将所有单词的表示（值向量）进行加权求和，而权重是通过该词的表示（键向量）与被编码词表示（查询向量）的点积并通过softmax得到。），然后即得到自注意力层在该位置的输出(在我们的例子中是对于第一个单词)。



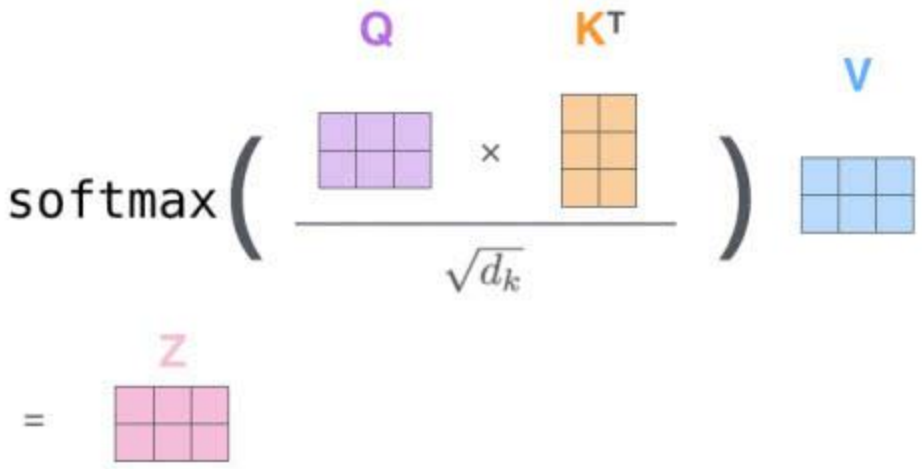
通过矩阵运算实现自注意力机制

第一步是计算查询矩阵、键矩阵和值矩阵。为此，我们将输入句子的词嵌入装进矩阵X中，将其乘以我们训练的权重矩阵(WQ，WK，WV)。



Transformer-自注意力机制

由于我们处理的是矩阵，我们可以将步骤2到步骤6合并为一个公式来计算自注意力层的输出



Transformer-多头注意力机制

通过增加一种叫做“多头”注意力（“multi-headed” attention）的机制，论文进一步完善了自注意力层，并在两方面提高了注意力层的性能：

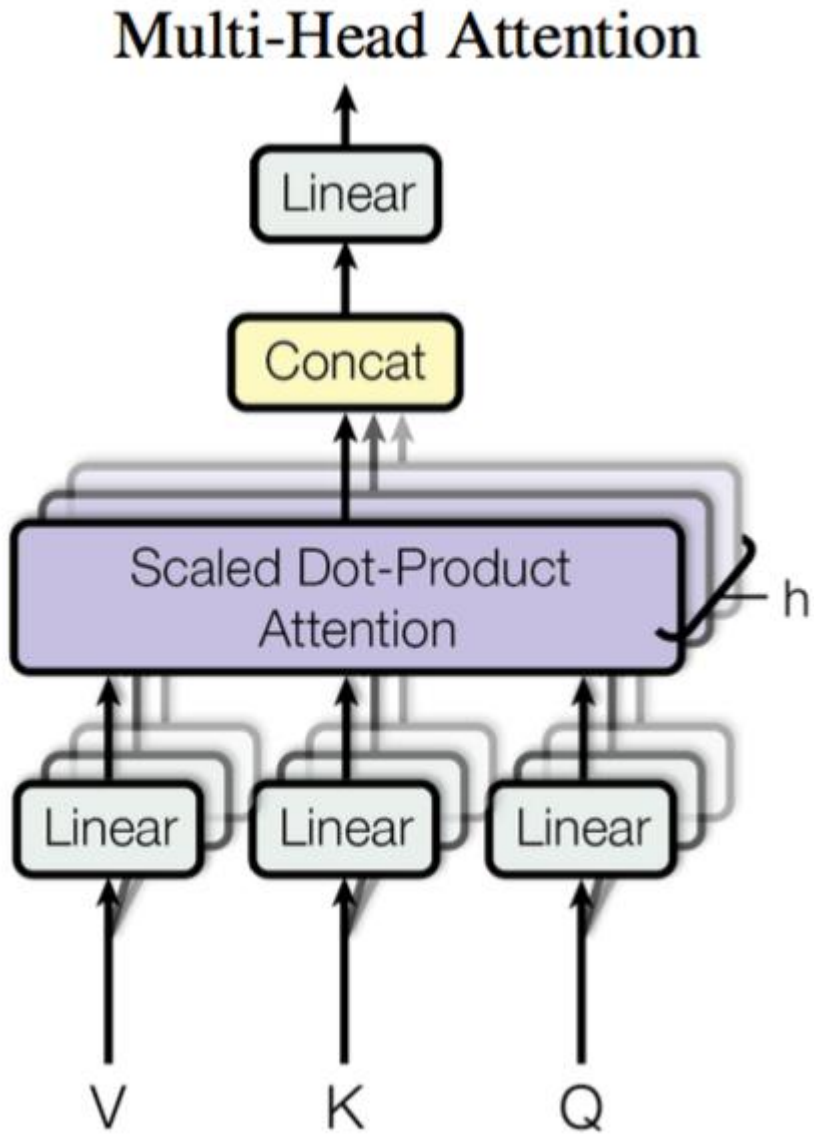
1.它扩展了模型专注于不同位置的能力。在上面的例子中，虽然每个编码都在z1中有或多或少的体现，但是它可能被实际的单词本身所支配。如果我们翻译一个句子，比如“The animal didn’t cross the street because it was too tired”，我们会想知道“it”指的是哪个词，这时模型的“多头”注意机制会起到作用。

2.它给出了注意力层的多个“表示子空间”（representation subspaces）。接下来我们将看到，对于“多头”注意机制，我们有多个查询/键/值权重矩阵集(Transformer使用八个注意力头，因此我们对于每个编码器/解码器有八个矩阵集合)。这些集合中的每一个都是随机初始化的，在训练之后，每个集合都被用来将输入词嵌入(或来自较低编码器/解码器的向量)投影到不同的表示子空间中。

Transformer-多头注意力机制

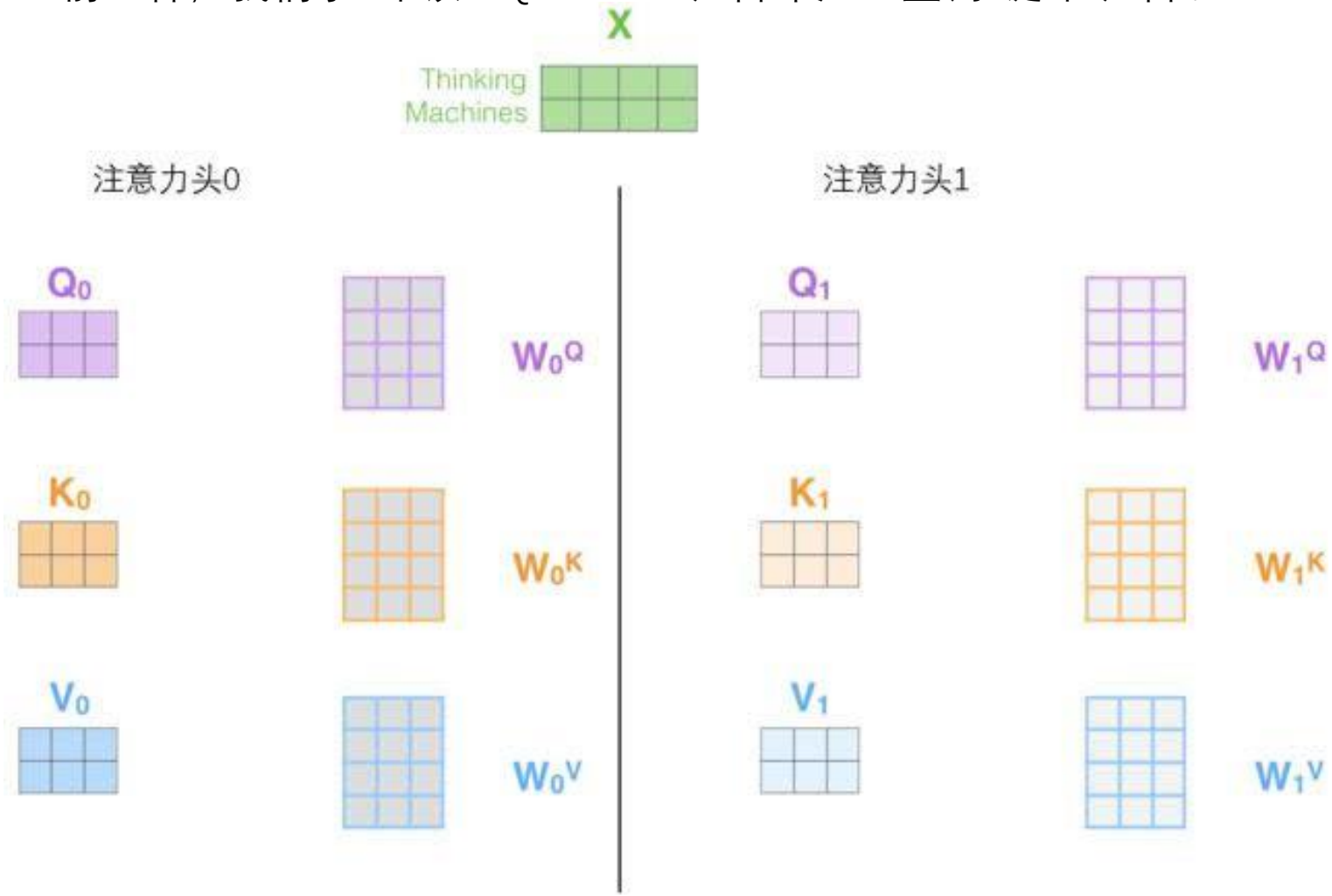
这里的multi-head的含义是，每个Transformer结构会由多层结构完全一样的，但权重矩阵不同的Attention组成，也就是我们上面提到的Attention结构。

这么做的目的是为了防止模型只关注到模型的一部分特征，却忽略了其他特征，所以增加模型的厚度，让模型拥有多层结构相同，但是权重不同的Attention，每一个head都关注到了不同的特征，那么模型整体就会关注到更多的特征。



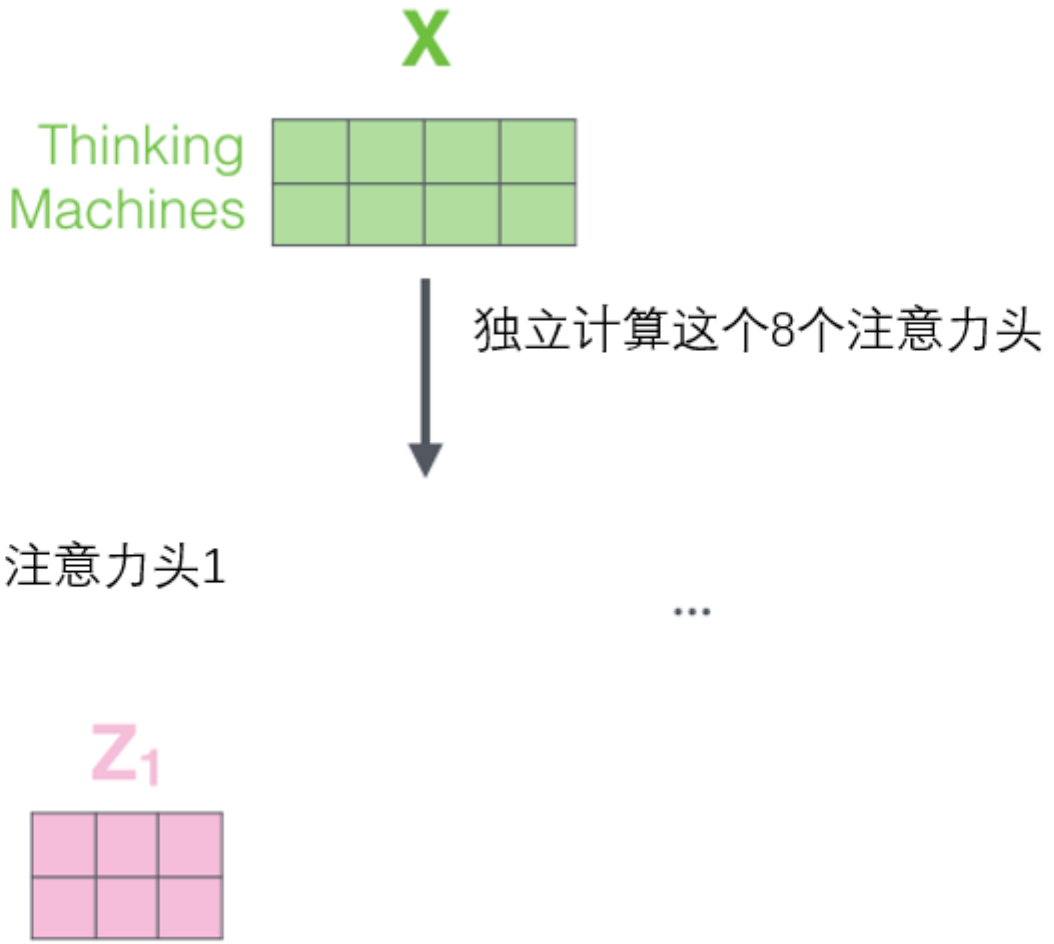
Transformer-多头注意力机制

在“多头”注意机制下，我们为每个头保持独立的查询/键/值权重矩阵，从而产生不同的查询/键/值矩阵。和之前一样，我们拿X乘以WQ/WK/WV矩阵来产生查询/键/值矩阵。



Transformer-多头注意力机制

如果我们做与上述相同的自注意力计算，需八次不同的权重矩阵运算，我们就会得到八个不同的Z矩阵。



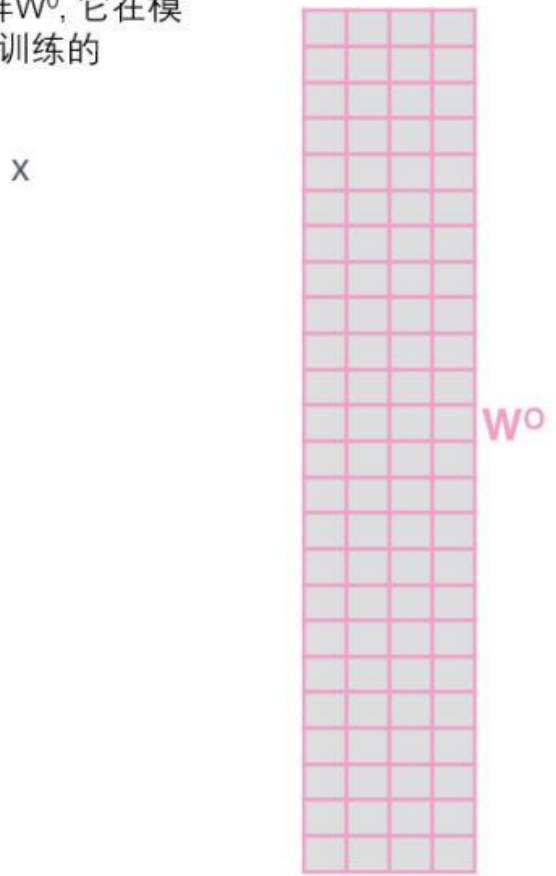
Transformer-多头注意力机制

这给我们带来了一点挑战。前馈层不需要8个矩阵，它只需要一个矩阵(由每一个单词的表示向量组成)。所以我们需要一种方法把这八个矩阵压缩成一个矩阵。

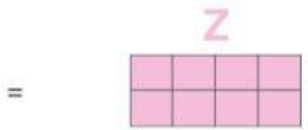
那该怎么做？其实可以直接把这些矩阵拼接在一起，然后用一个附加的权重矩阵WO与它们相乘

1）将所有注意力头拼接起来

2）乘以矩阵W^0，它在模型中是联合训练的



3）结果是一个融合所有注意力头信息的矩阵Z，我们可以将其送到前馈神经网络



Transformer-多头注意力机制

1）这是我们的输入句子*

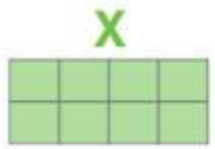
2）编码每一个单词

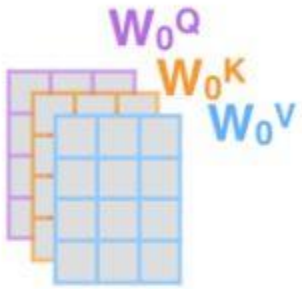
3）将其分为8个头，将矩阵X或R乘以各个权重矩阵

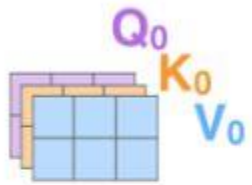
4）通过输出的查询/键/值（Q/K/V）矩阵计算注意力

5）将所有注意力头拼接起来，乘以权重矩阵W^0

Thinking
Machines



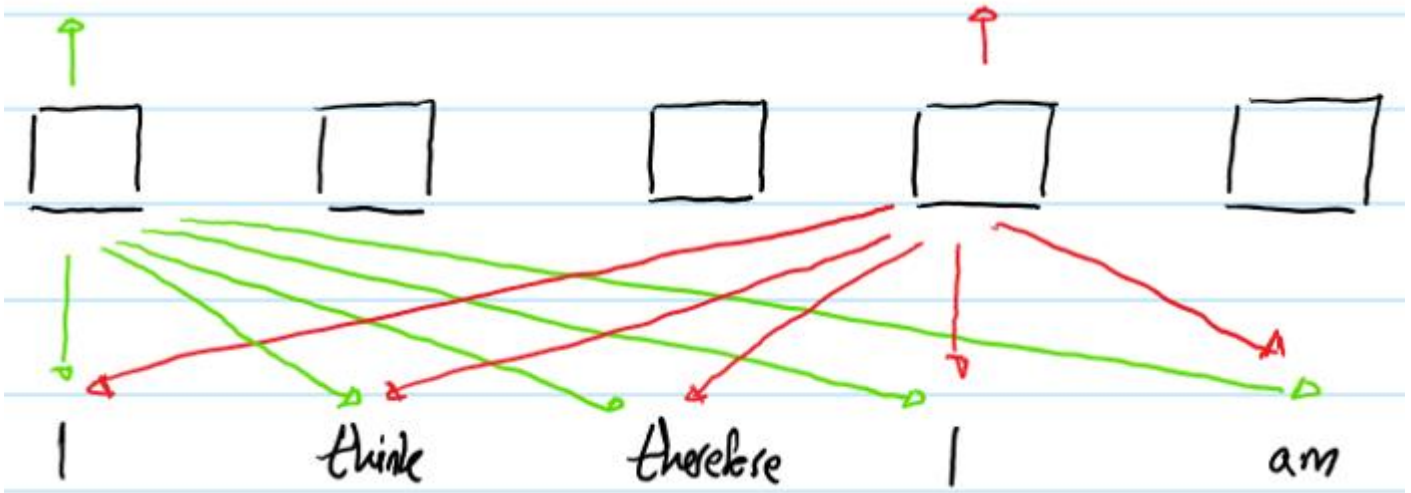




*注：除了第0个编码器，其他编码器都不需要进行词嵌入。它可以直接讲前面一层编码器的输出作为输入（矩阵R）。

Transformer-位置编码

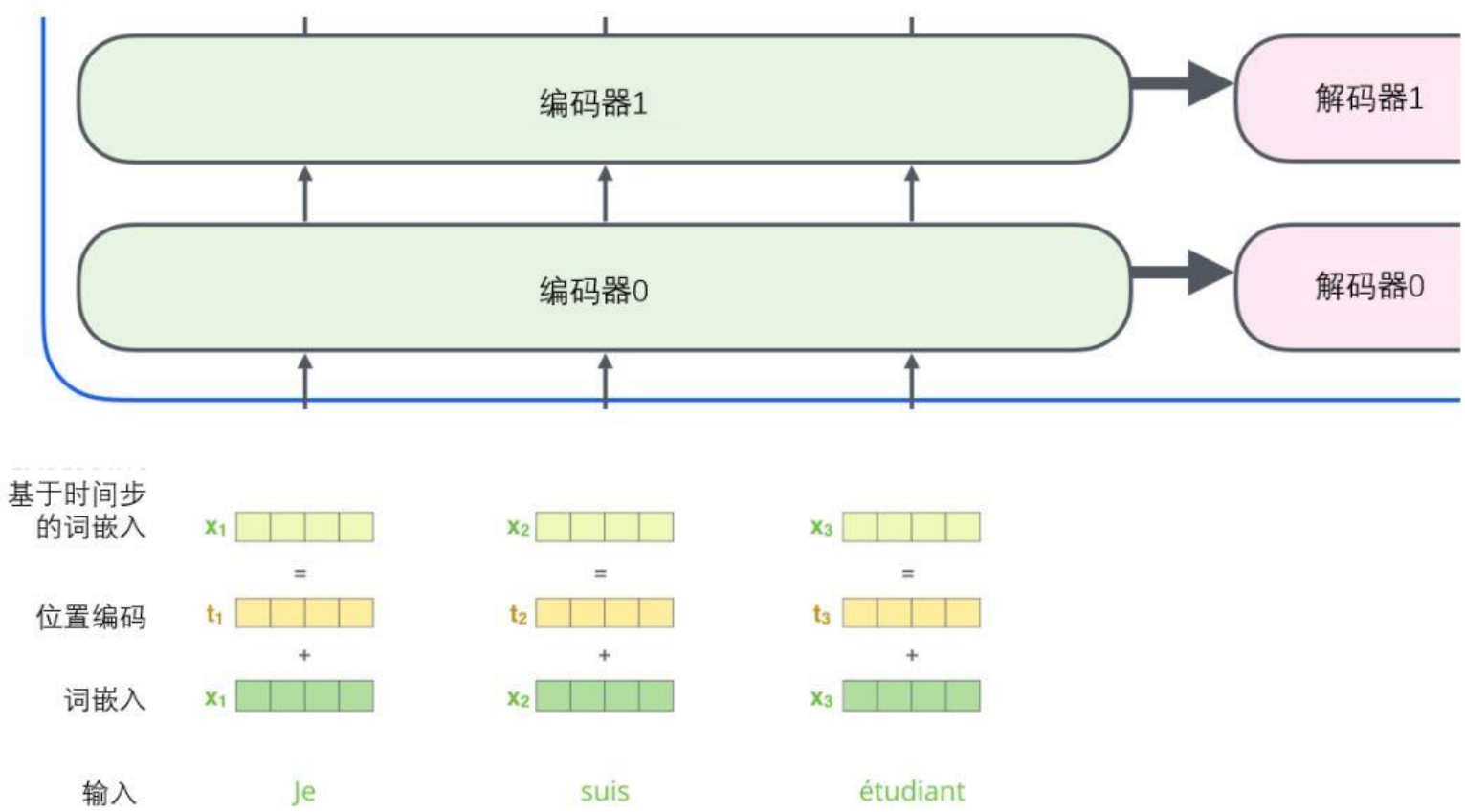
在self-attention中，第一个“I”与第二个“I”的输出将完全相同。因为它们用于产生输出的“input”是完全相同的。即在同一个输入序列中，不同位置的相同的单词的output representation完全相同，这样就不能体现单词之间的时序关系。--所以要对单词的时序位置进行编码表征。



Transformer-位置编码

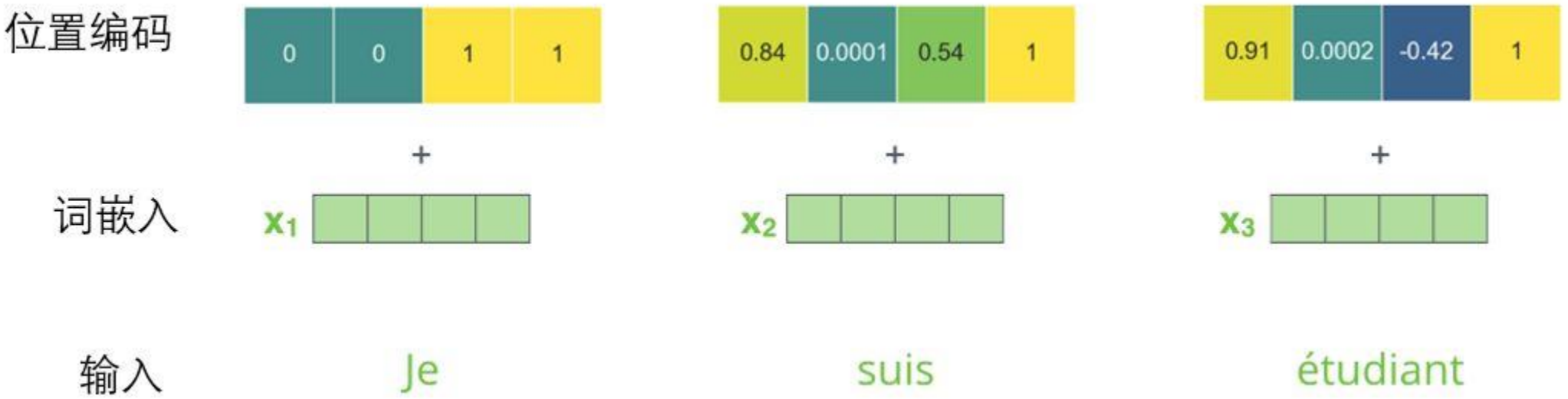
到目前为止，我们对模型的描述还缺少一种理解输入单词顺序的方法。

为了解决这个问题，Transformer为每个输入的词嵌入添加了一个向量。这些向量遵循模型学习到的特定模式，这有助于确定每个单词的位置，或序列中不同单词之间的距离。这里的直觉是，将位置向量添加到词嵌入中使得它们在接下来的运算中，能够更好地表达的词与词之间的距离。



Transformer-位置编码

为了让模型理解单词的顺序，我们添加了位置编码向量，这些向量的值遵循特定的模式。
如果我们假设词嵌入的维数为4，则实际的位置编码如下：



Transformer-位置编码

Position Encoding（PE）的公式：

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Pos：句中字的位置，取值范围是 [0, 句子长度]
i：词向量的维度，取值范围是 [0，词嵌入长度]
dmodel: 词嵌入长度

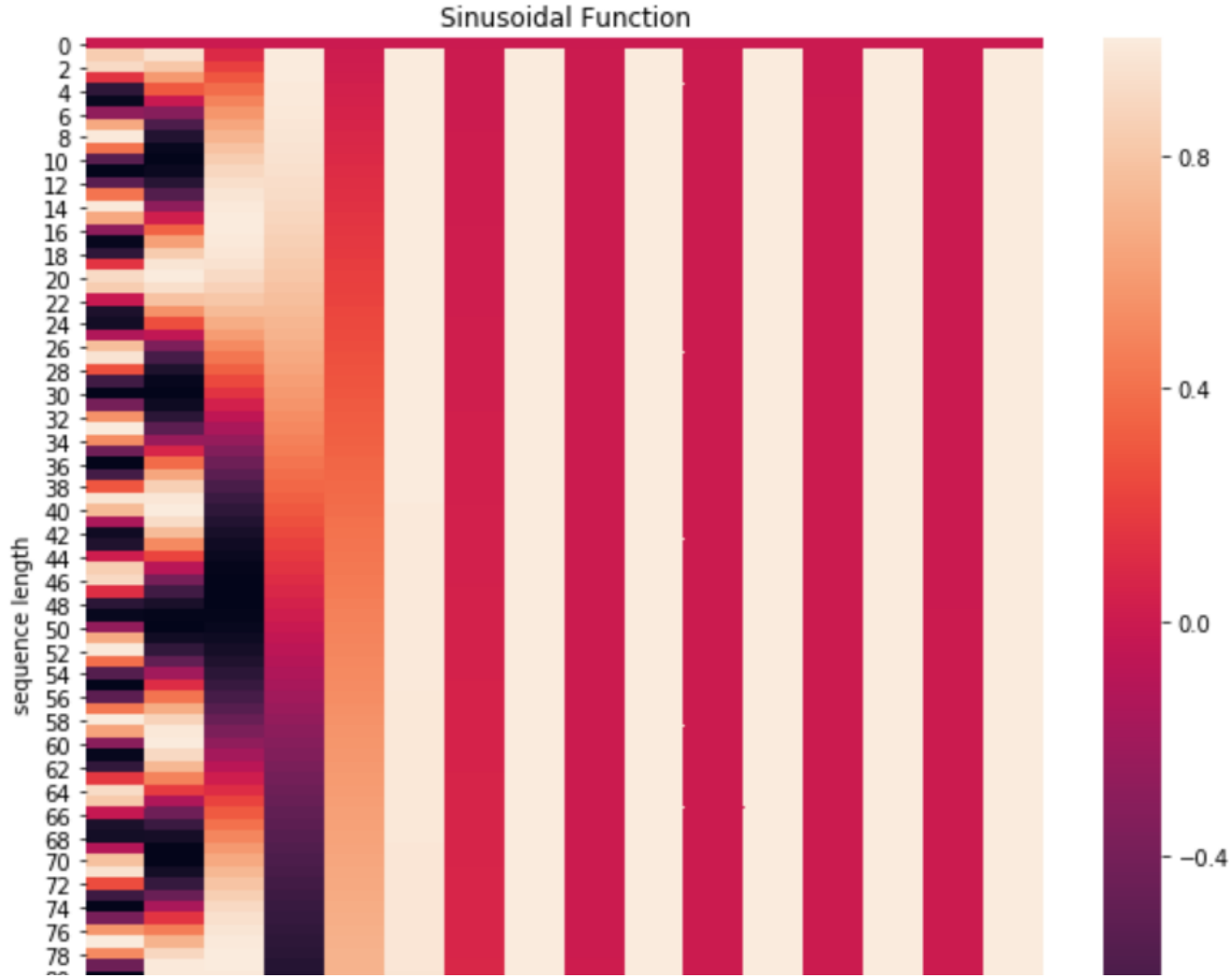
举个例子，如pos=3，d(model)=128，那么3对应的位置向量如下：
[$\sin(3/10000^{0/128})$, $\cos(3/10000^{1/128})$, $\sin(3/10000^{2/128})$,$\cos(3/10000^{3/128})$, ...]

Transformer-位置编码

每一行对应一个词向量的位置编码，所以第一行对应着输入序列的第一个词。每行包含n个值，每个值介于1和-1之间。我们已经对它们进行了颜色编码，所以图案是可见的。

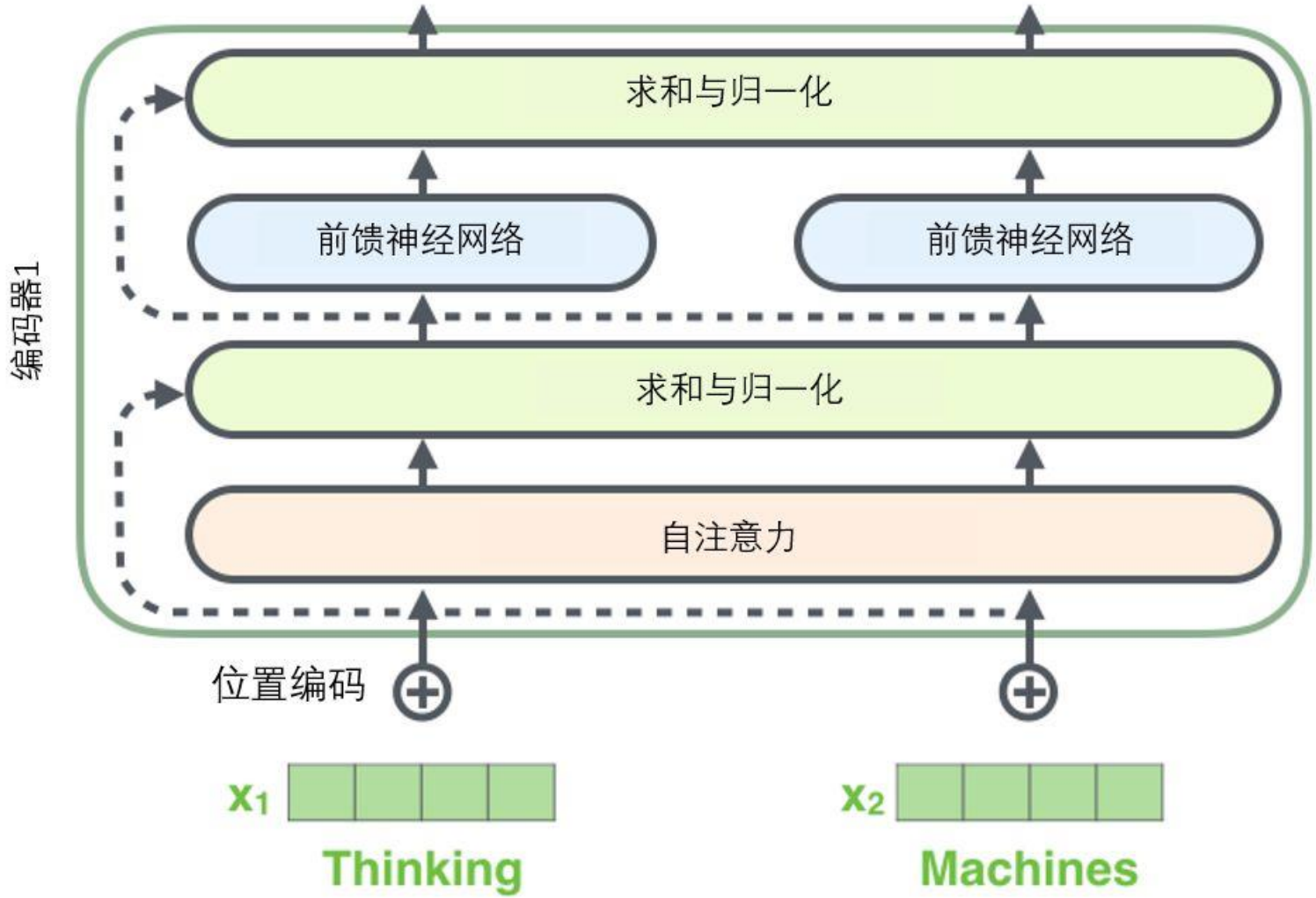
用三角函数的原因：

- 过大的位置embedding，跟词嵌入相加，很容易导致词向量本身含义的丢失；
- 纵向观察，随着embedding dimension增大，位置嵌入函数呈现不同的周期变化。



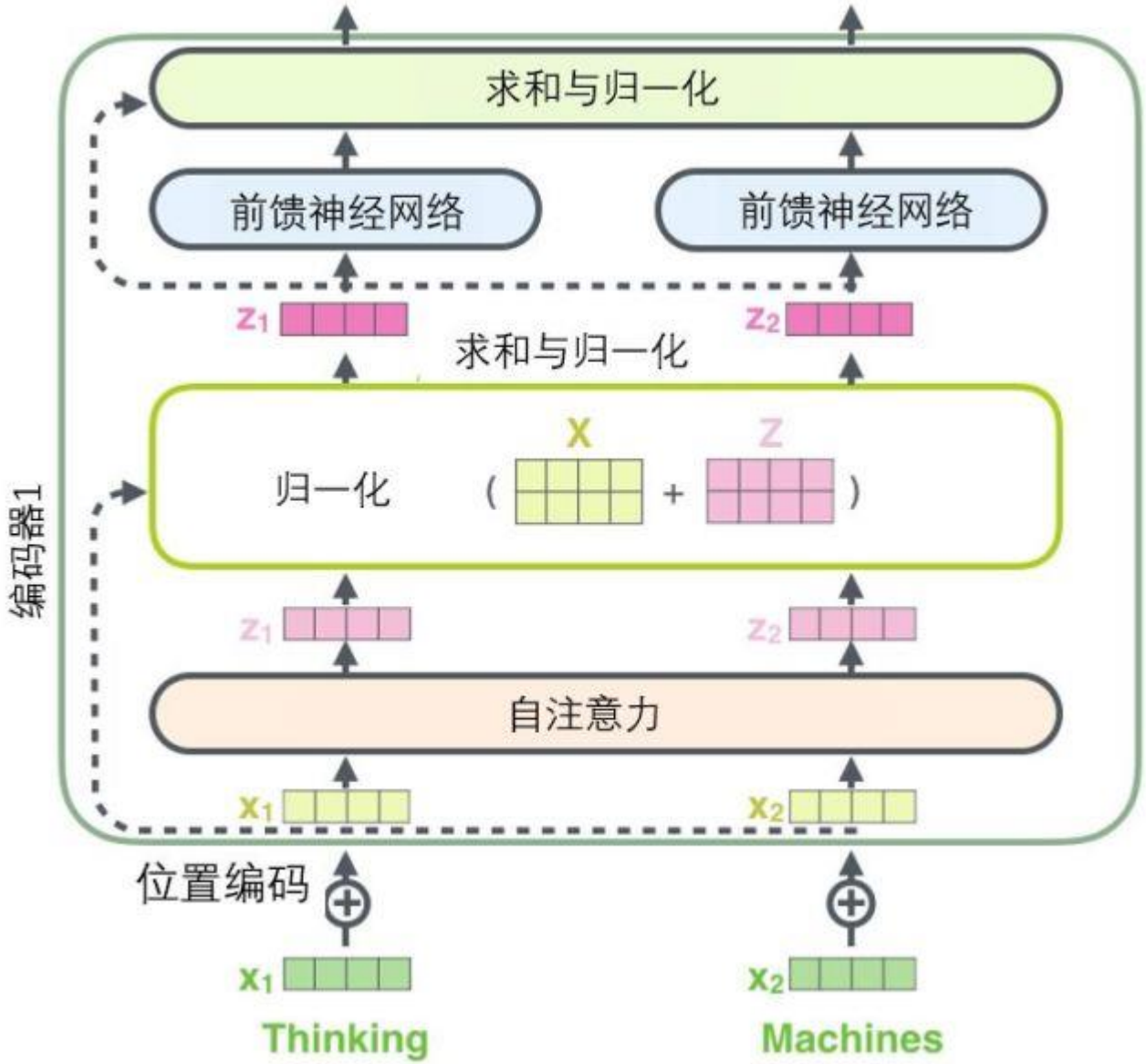
Transformer-残差模块

在每个编码器中的每个子层（自注意力、前馈网络）的周围都有一个残差连接，并且都跟随着一个“层-归一化”步骤。



Transformer-残差模块

可视化这些向量以及这个和自注意力相关联的层-归一化操作,

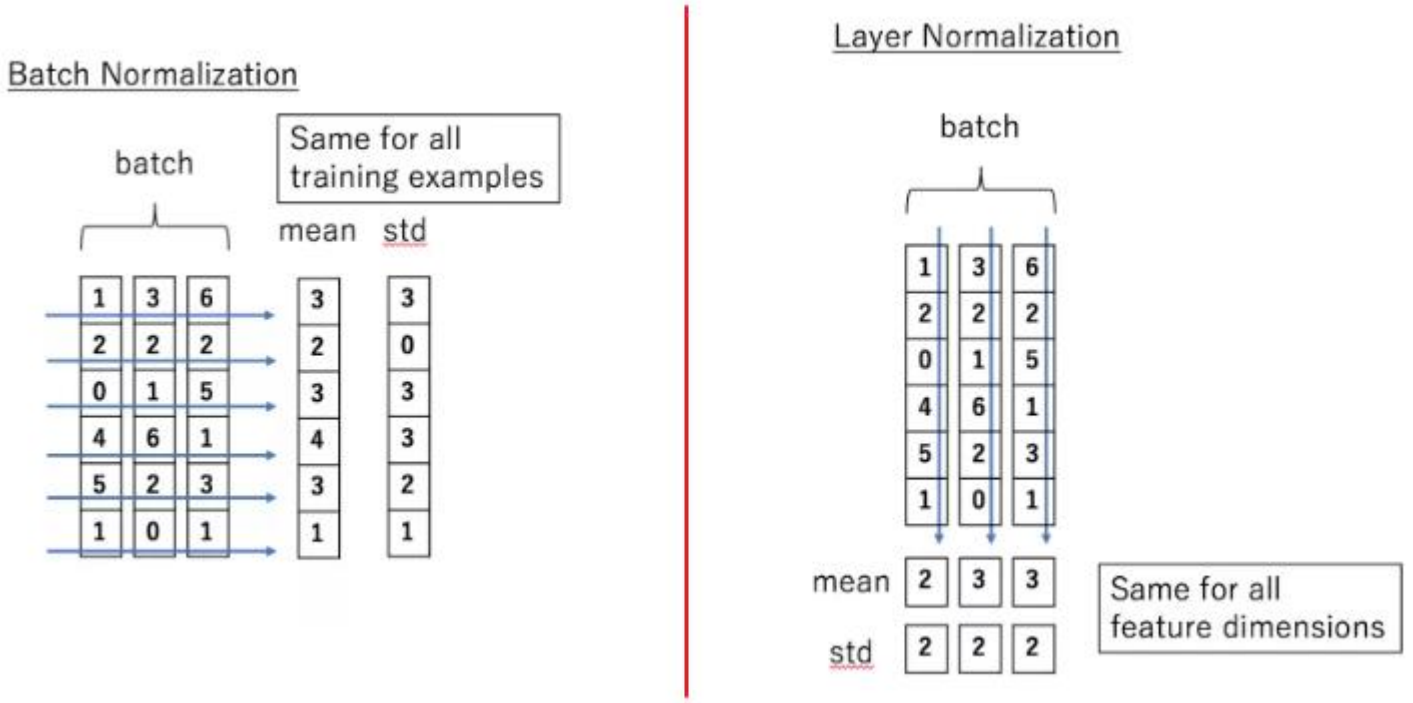


Transformer-归一化

Layer Normalization

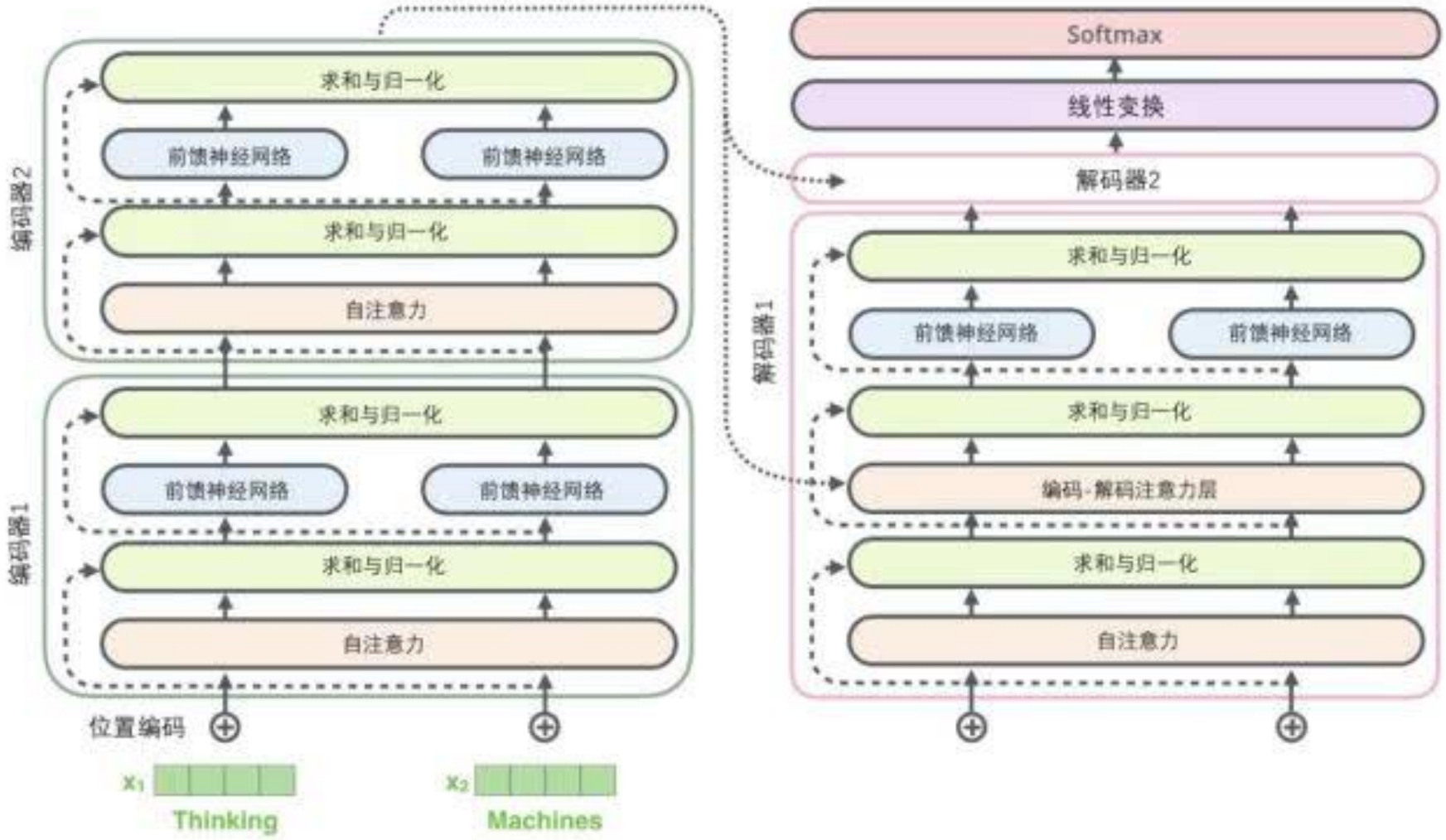
我们比较熟悉batch normalization，是将一个batch中所有相同的feature进行normalization，
而layer normalization是指用对每一个样本的feature进行normalization。

$$LN(x_i) = \alpha * \frac{x_i - \mu_L}{\sqrt{\sigma_L^2 + \varepsilon}} + \beta$$



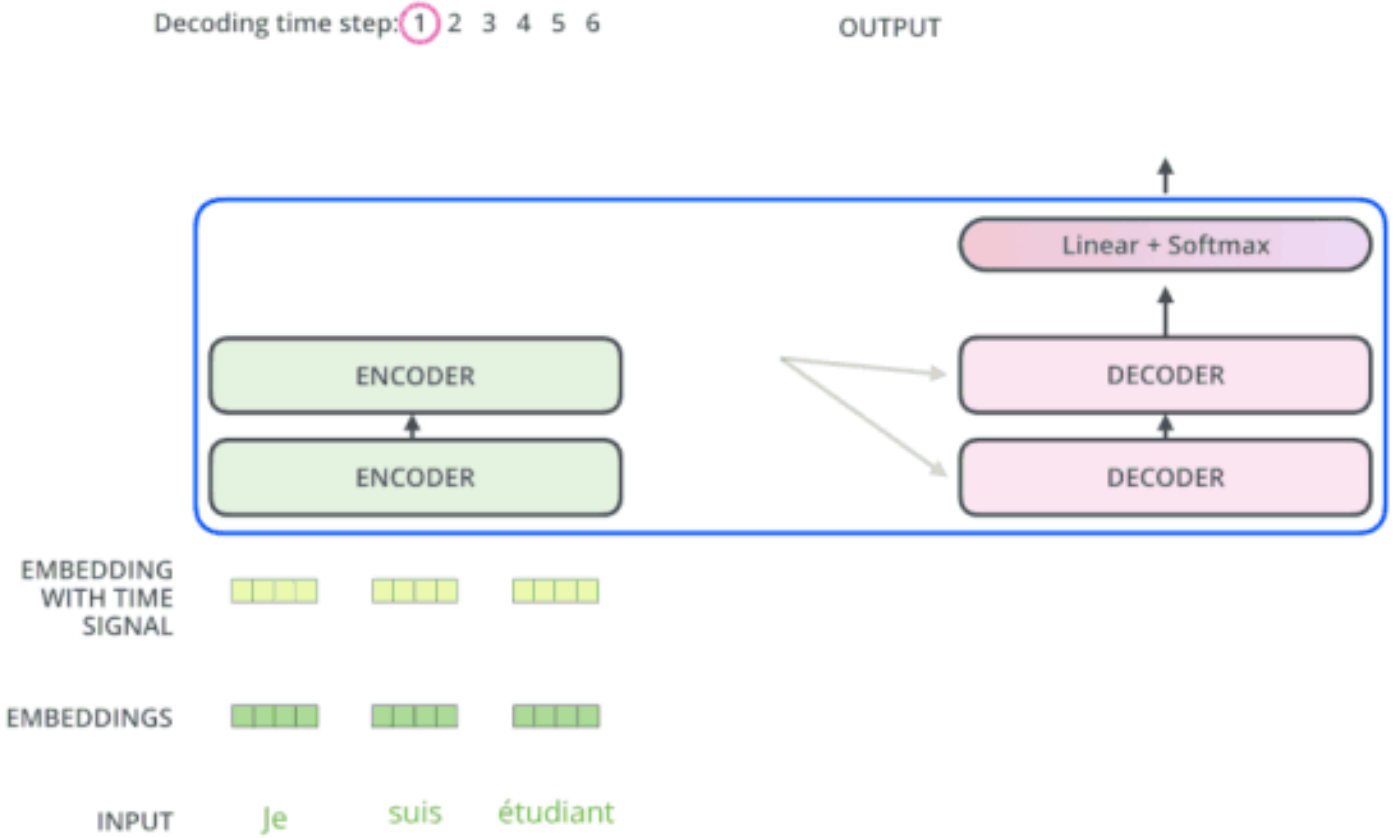
Transformer-残差模块

解码器的子层也是这样的。如果我们想象一个2 层编码-解码结构的transformer，它看起来会像下面这张图一样：



Transformer-解码组件

编码器通过处理输入序列开启工作。顶端编码器的输出之后会变转化为一个包含向量K（键向量）和V（值向量）的注意力向量集 。这些向量将被每个解码器用于自身的“编码-解码注意力层”，而这些层可以帮助解码器关注输入序列哪些位置合适：



Transformer-解码组件

在完成编码阶段后，则开始解码阶段。解码阶段的每个步骤都会输出一个输出序列（在这个例子里，是英语翻译的句子）的元素。

接下来的步骤重复了这个过程，直到到达一个特殊的终止符号，它表示transformer的解码器已经完成了它的输出。

每个步骤的输出在下一个时间步被提供给底端解码器，并且就像编码器之前做的那样，这些解码器会输出它们的解码结果 。另外，就像我们对编码器的输入所做的那样，我们会嵌入并添加位置编码给那些解码器，来表示每个单词的位置。

而那些解码器中的自注意力层表现的模式与编码器不同：在解码器中，自注意力层只被允许处理输出序列中更靠前的那些位置。在softmax步骤前，它会把后面的位置给隐去（把它们设为-inf）。

这个“编码-解码注意力层”工作方式基本就像多头自注意力层一样，只不过它是通过在它前面的层来创造查询矩阵，并且从编码器的输出中取得键/值矩阵。

Transformer-解码组件

Masked Self-Attention

在decoder中的attention layer中的attention 部分，和encoder中的attention不同。

输入序列进入到一个masked self-attention中，因为我们在prediction的时候decoder是从左往右地逐个predict，所以在我们做attention的时候，每一个时间步的输入应该只attention到之前的输入，因此我们要像前文所说的那样，通过在attention的系数矩阵的对应位置加上负无穷然后经过softmax函数，来将某些位置的权重mask掉。

Transformer-最终的线性变换和Softmax层

解码组件最后会输出一个实数向量。我们如何把浮点数变成一个单词？这便是线性变换层要做的工作，它之后就是Softmax层。

线性变换层是一个简单的全连接神经网络，它可以把解码组件产生的向量投射到一个比它大得多的、被称作对数几率（logits）的向量里。

不妨假设我们的模型从训练集中学习一万个不同的英语单词（我们模型的“输出词表”）。因此对数几率向量为一万个单元格长度的向量——每个单元格对应某一个单词的分数。

接下来的Softmax 层便会把那些分数变成概率（都为正数、上限1.0）。概率最高的单元格被选中，并且它对应的单词被作为这个时间步的输出。

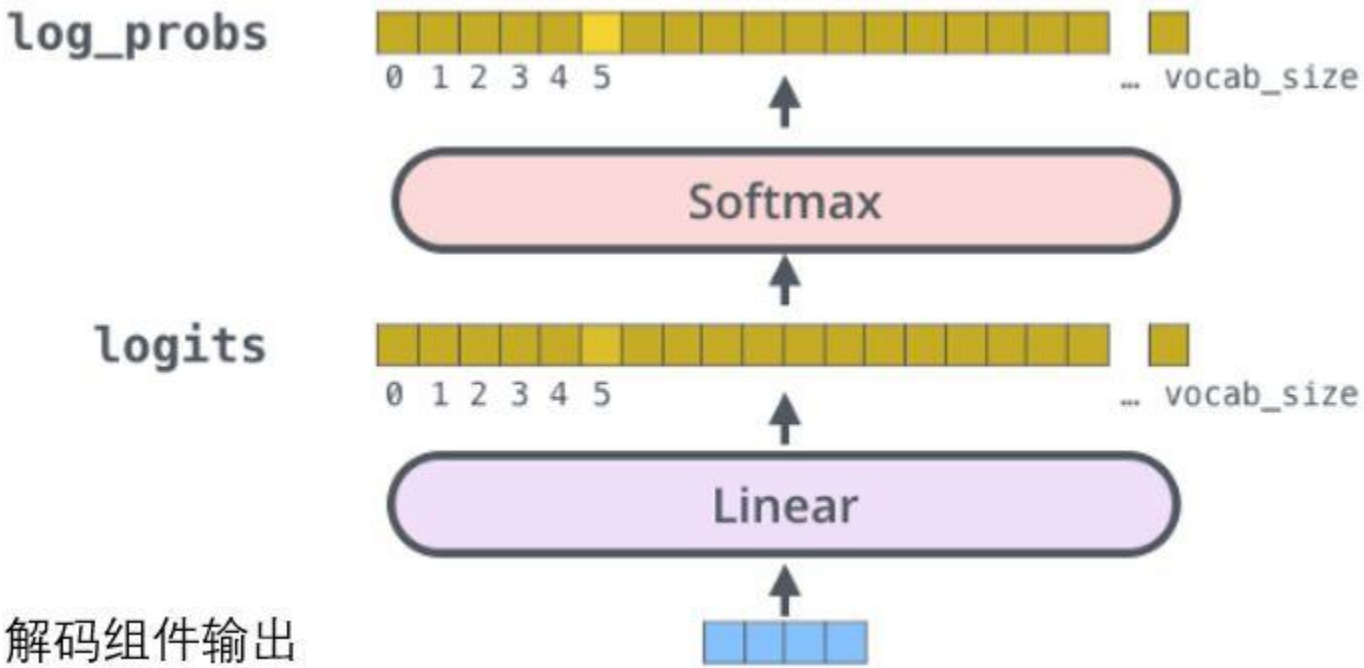
Transformer-最终的线性变换和Softmax层

查找词表中对应该索引的单词

am

获取最高概率的索引
(argmax)

5



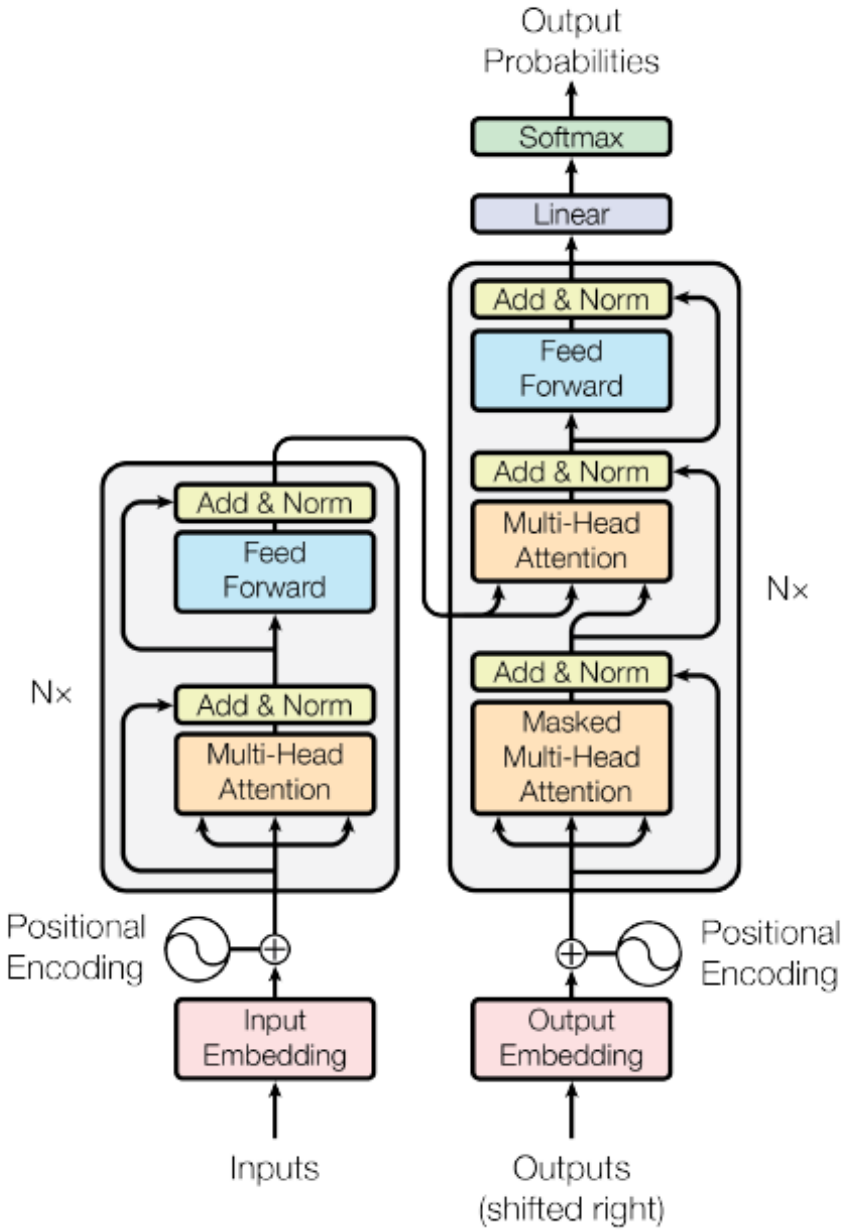


Figure 1: The Transformer - model architecture.

Transformer—Feed Forward

Transformer中的feed forward网络可以理解为两个连续的线性变换，这两个变换中间是一个ReLU激活函数：

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$$

Transformer的优势：

- 相较于RNN的最大优势在于其速度上的优势；
- 相较于CNN，其能直接获取全局信息。

Bert

自google在2018年10月底公布BERT在11项nlp任务中的卓越表现后，**BERT（Bidirectional Encoder Representation from Transformers)** 就成为NLP领域大火、整个ML界都有耳闻的模型。

一句话概括，BERT的出现，彻底改变了预训练产生词向量和下游具体NLP任务的关系，提出龙骨级的训练词向量概念。

主要是将之前的Transform加上更为泛化的预训练，得到了很好的语言表达模型。

Bert-bert刷新的11项纪录

1. MultiNLI（multi-genre natural language inference，文本蕴含识别）

文本间的推理关系，又称为文本蕴含关系。样本都是文本对，第一个文本M作为前提，如果能够从文本M推理出第二个文本N，即可说M蕴含N，M->N。两个文本关系一共有三种entailment（蕴含）、contradiction（矛盾）、neutral（中立）

2. QQP（quora question pairs，文本匹配）

判断两个问题是不是同一个意思，即是不是等价的。属于分类任务

3. QNLI（question natural language inference，自然语言问题推理）

是一个二分类任务。正样本为（question，sentence），包含正确的answer；负样本为（question，sentence），不包含正确的answer。

4. SST-2（the stanford sentiment treebank，斯坦福情感分类树）

   分类任务。

5. CoLA（the corpus of linguistic acceptability，语言可接受性语料库）

   分类任务，预测一个句子是否是acceptable。

6. STS-B（the semantic textual similarity benchmark，语义文本相似度数据集）

   样本为文本对，分数为1-5，用来评判两个文本语义信息的相似度。

Bert-bert刷新的11项纪录

7. MRPC（microsoft research paraphrase corpus，微软研究释义语料库）

   样本为文本对，判断两个文本对语音信息是否是等价的。

8. RTE（recognizing textual entailment，识别文本蕴含关系）

   与MNLI相似，只不过数据集更少

9. WNLI（winograd NLI，自然语言推理）

10. SQuAD（the standFord question answering dataset，斯坦福问答数据集）

11. NER（named entity recognition，命名实体识别）

12. SWAG（the situations with adversarial generations dataset）

    从四个句子中选择最可能为前句下文的那个

Bert

BERT是一个预训练的模型，那么什么是预训练呢?

举例子进行简单的介绍

假设已有A训练集，先用A对网络进行预训练，在A任务上学会网络参数，然后保存以备后用，当来一个新的任务B，采取相同的网络结构，网络参数初始化的时候可以加载A学习好的参数，其他的高层参数随机初始化，之后用B任务的训练数据来训练网络，当加载的参数随着B任务的训练进行不断的改变，称为“fine-tuning”，即更好地把参数进行调整使得更适合当前的B任务

优点：当任务B的训练数据较少时，很难很好的训练网络，但是获得了A训练的参数，会比仅仅使用B训练的参数更优。

目前最好的自然语言预训练方法无疑是BERT。它的工作流程分为两步：

1. 使用大量未标记的数据，也就是无监督的方式学习语言表达。

2. 使用少量经过标记的训练数据对模型进行fine-tune，以监督学习的方式，执行多种监督任务。

Bert—Masked Language Model（MLM）

原本叫cloze test，是完形填空的意思。

随机mask语料中15%的字，然后将掩盖的字的位置输出的最终隐层向量送入softmax，来预测masked token。

这样输入一个句子，每次只预测句子中大概15%的词，所以BERT训练很慢。（但是google设备NB。）

而对于盖住词的特殊标记，在下游NLP任务中不存在。因此，为了和后续任务保持一致，作者按一定的比例在需要预测的词位置上输入原词或者输入某个随机的词。

如：my dog is hairy
有80%的概率用“[mask]”标记来替换——my dog is [MASK]
有10%的概率用随机采样的一个单词来替换——my dog is apple
有10%的概率不做替换——my dog is hairy

Input: The man went to the $[MASK]_1$. He bought a $[MASK]_2$ of milk .
Labels: $[MASK]_1$ = store; $[MASK]_2$ = gallon

Mask的目的：

Transformer encoder不知道它将被要求预测哪些单词或哪些单词已被随机单词替换，因此它被迫保持每个输入token的分布式上下文表示。

此外，因为随机替换只发生在所有token的1.5％（即15％的10％），这**不会损害模型的语言理解能力**。

Bert-Next Sentence Prediction（NSP）

下一句预测

为了让模型捕捉两个句子的联系，这里增加了Next Sentence Prediction的预训练方法，这一任务可以从任何语料库中生成。

具体地说，当选择句子A和B作为预训练样本时，B有50%的可能是A的下一个句子，也有50%的可能是来自语料库的随机句子。

完全随机地选择了NotNext语句，使模型具备理解长序列上下文的联系的能力，最终的预训练模型在此任务上实现了97%-98%的准确率。

Sentence A = The man went to the store.
Sentence B = He bought a gallon of milk.
Label = IsNextSentence

Sentence A = The man went to the store.
Sentence B = Penguins are flightless.
Label = NotNextSentence

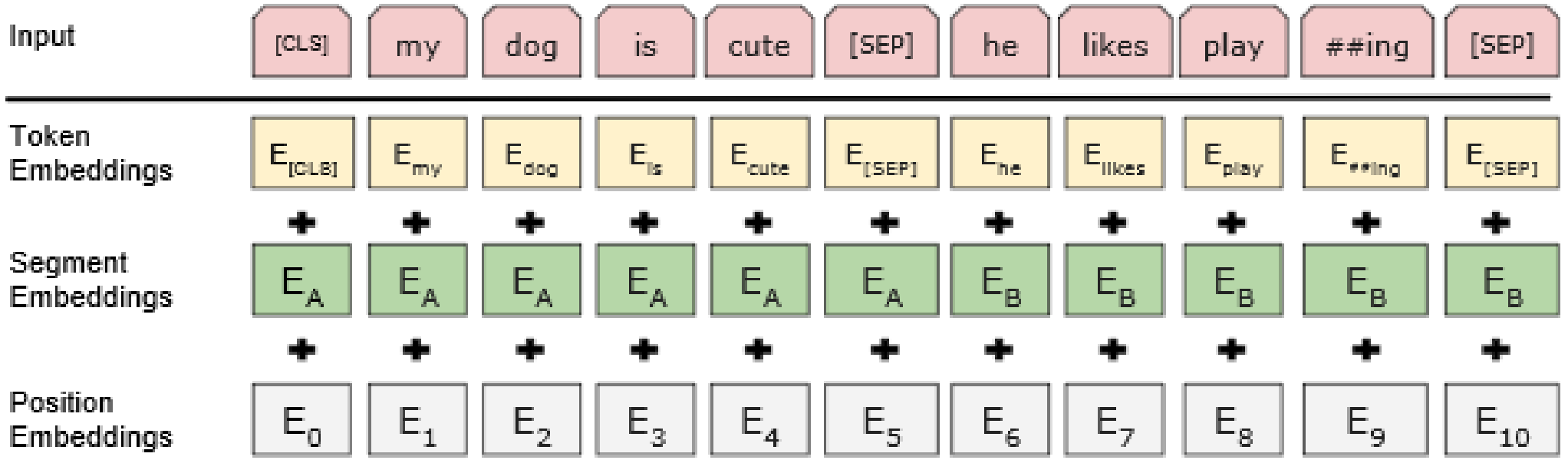
Bert-对输入的处理

bert的输入部分是个线性序列，两个句子通过分隔符分割，最前面和最后增加两个标识符号。CLS/SEP

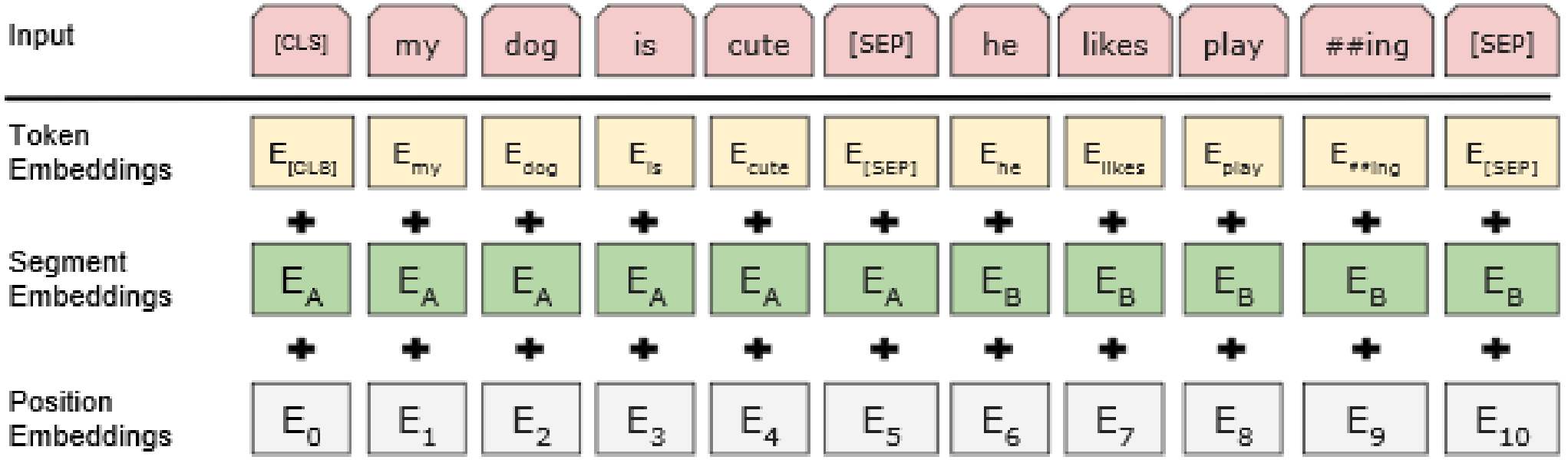
每个单词有三个embedding:

1. 位置信息embedding，这是因为NLP中单词顺序是很重要的特征，需要在这里对位置信息进行编码;
2. 单词embedding，词嵌入;
3. 句子embedding，因为前面提到训练数据都是由两个句子构成的，那么每个句子有个句子整体的embedding项对应给每个单词。

把单词对应的三个embedding叠加，就形成了Bert的输入。



Bert-对输入的处理



（1）token embeddings表示的是词向量，第一个单词是CLS，可以用于之后的分类任务
（2）segment embeddings用来区别两种句子，因为预训练还要做以两个句子为输入的分类任务
（3）position embeddings表示位置信息

优点：

BERT通过预训练和精调可以解决11项NLP的任务。使用的是Transformer，相对于rnn而言更加高效、能捕捉更长距离的依赖。

缺点：

作者在文中主要提到的就是MLM预训练时的mask问题：

1)[MASK]标记在实际预测中不会出现，训练时用过多[MASK]影响模型表现;

2)每个batch只有15%的token被预测，所以BERT收敛得比其他模型要慢（它们会预测每个token）