《机器翻译（二）》讲义

## 一、机器翻译模型评价

## 1.1 人工评价方法

人工评价是由人类评审员对机器翻译结果进行评估。评审员通常是双语专家，他们根据一定的标准对翻译质量进行评分。评价标准包括：

* 忠实度（Fidelity）：译文内容与原文是否一致，是否准确传达了原文的意思。
* 流利度（Fluency）：译文是否符合目标语言的语法规则，是否自然流畅。

评价的方法是给定系统A和B在相同测试集上的译文，双语专家逐一对比A和B的译文，判断A是否优于B。具体步骤如下：

1. 评审员逐个对比每个源语言句子的两个译文，判断哪个更好。
2. 若A优于B，则A得一票；若B优于A，则B得一票；若无法区分，则不计分。
3. 根据得票数判断系统A和B之间的优劣。

基于人工评价方法的不能能够比较准确地捕捉细微的翻译质量差异，还具有可靠性高，尤其是由经验丰富的双语专家进行评价。缺点是主观性强，不同评审员可能有不同的评价标准和偏好。此外，差异性大，不同评审员的评分可能不一致。再者需要投入大量人力资源且效率低下。

## 1.2 自动评价方法

自动评价方法使用计算机程序对机器翻译结果进行评估，通过比较机器译文与参考译文的相似度来计算翻译质量得分。

1. BLEU（Bilingual Evaluation Understudy）

BLEU是一种常用的自动评价指标，通过统计机器译文与参考译文中n元文法（n-gram）的匹配情况来衡量翻译质量。其计算步骤如下：

1. 统计机器译文和参考译文中的*n*元文法匹配数。
2. 计算*n*元文法的精确率（Precision），即匹配的*n*元文法数量占系统译文中所有*n*元文法数量的比率。
3. 引入长度惩罚项（Brevity Penalty, BP），防止机器译文过短。
4. 计算BLEU得分：



其中，*pn*是*n*元文法的精确率，*wn*是权重（通常为均等分配），BP是长度惩罚项。

（2）METEOR（Metric for Evaluation of Translation with Explicit ORdering）

METEOR结合了词级别的精确率和召回率，同时考虑同义词匹配和词序信息。计算公式如下：



其中，*P*为精确率，*R*为召回率，*α*为权重参数。

该评价标准考虑了词形变化和同义词匹配，提高了评价的准确性，对单词顺序的考虑使得其评价更为细致。

（3）ROUGE（Recall-Oriented Understudy for Gisting Evaluation）

ROUGE指标最初用于评估自动摘要生成，但也可用于机器翻译评价。主要通过计算机器译文与参考译文之间的覆盖率来衡量质量。常用的变体有ROUGE-N、ROUGE-L等，其计算方式如下：







其中，是长度为个词的真实答案，是长度为个词的预测答案，表示和的最长公共子序列的长度,所以和分别表示召回率和准确率，用于控制指标的准确率和召回率的重要程度。

## 二、Transformer介绍

Transformer模型是由Vaswani等人在2017年提出的，其论文《Attention is All You Need》对这一模型进行了详细介绍。Transformer模型在自然语言处理（NLP）任务中取得了巨大的成功，特别是在机器翻译任务中。与传统的循环神经网络（RNN）和卷积神经网络（CNN）不同，Transformer完全基于注意力机制（Attention Mechanism），不依赖于序列的顺序处理，具有高度的并行化能力。



图1 Transformer与CNN、RNN以及Attention机制的联系

在本文中，我们将试图把模型简化一点，并逐一介绍里面的核心概念，希望让普通读者也能轻易理解。首先介绍Transformer的整体结构，下图是Transformer用于中英文翻译的整体结构：

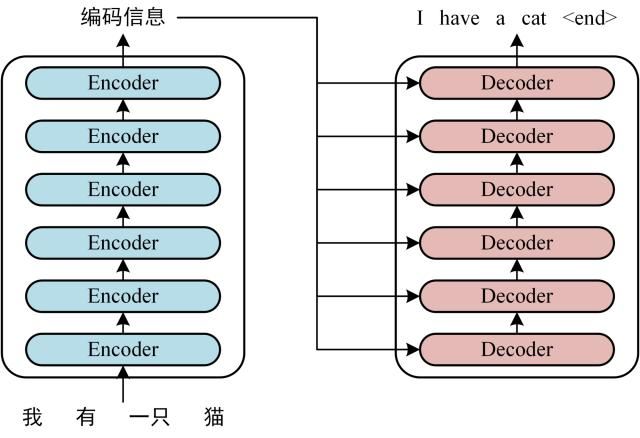


图2 Transformer的整体结构

可以看到Transformer由Encoder和Decoder两个部分组成，每个部分由多个相同的子层（层堆叠）组成。encoder：输入源语言序列，输出中间表示。Decoder：输入中间表示和目标语言的部分序列，输出目标语言序列。Encoder和Decoder 都包含6个block。Transformer 的工作流程大体如下：

第一步：获取输入句子的每一个单词的表示向量，由单词的Embedding（Embedding就是从原始数据提取出来的Feature）和单词位置的Embedding相加得到。

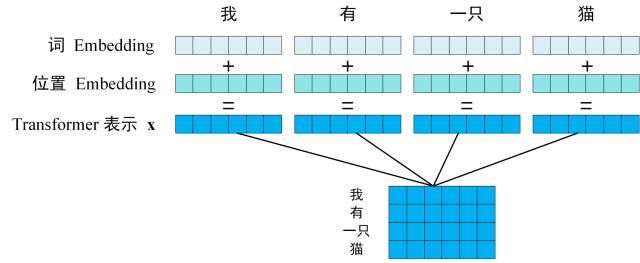


图3 Transformer的输入表示

第二步：将得到的单词表示向量矩阵 (如上图所示，每一行是一个单词的表示 x) 传入 Encoder 中，经过 6 个 Encoder block 后可以得到句子所有单词的编码信息矩阵 C，如下图。单词向量矩阵用表示，是句子中单词个数，是表示向量的维度 (论文中 d=512)。每一个Encoder block 输出的矩阵维度与输入完全一致。

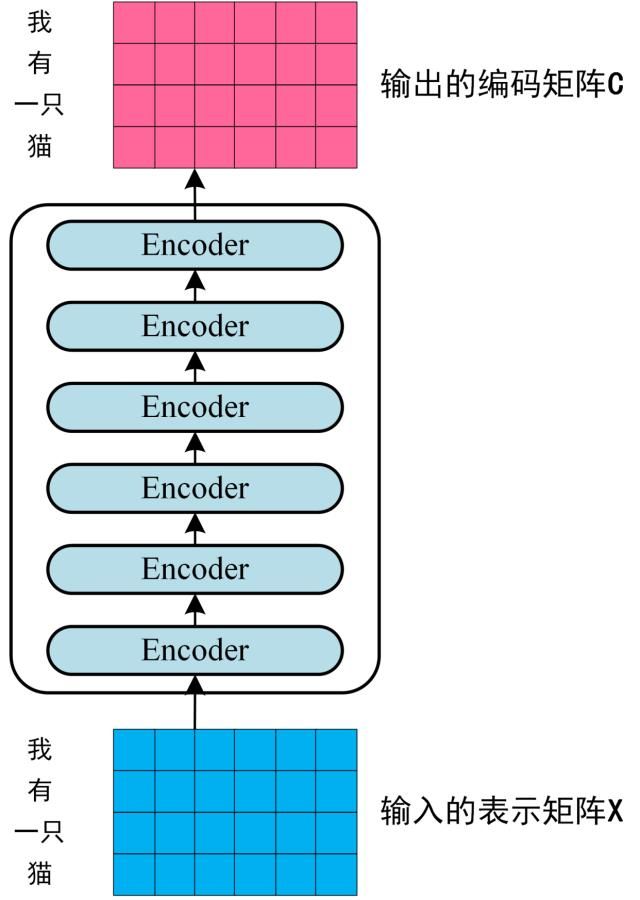


图4 Transformer的编码句子信息

第三步：将 Encoder 输出的编码信息矩阵传递到Decoder 中，Decoder 依次会根据当前翻译过的单词翻译下一个单词，如下图所示。在使用的过程中，翻译到单词的时候需要通过 Mask (掩盖) 操作遮盖住之后的单词。

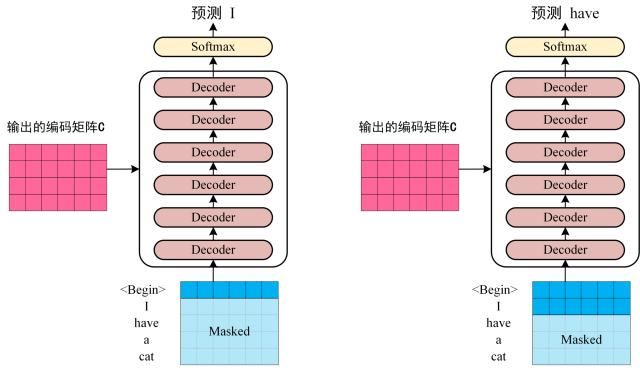


图5 Transformer的Decode预测

上图 Decoder 接收了 Encoder 的编码矩阵 C，然后首先输入一个翻译开始符 “<Begin>”，预测第一个单词“I”；然后输入翻译开始符“<Begin>”和单词“I”，预测单词“have”，以此类推。这是Transformer 使用时候的大致流程。

## 三、基于Transformer的机器翻译模型

### 3.1 Transformer 的输入

Transformer中单词的输入表示*x*由单词Embedding和位置Embedding（Positional Encoding）相加得到。**单词的Embedding**有很多种方式可以获取，例如可以采用Word2Vec、Glove 等算法预训练得到，也可以在Transformer 中训练得到。

**位置Embedding**表示单词出现在句子中的位置。因为Transformer不采用RNN的结构，而是使用全局信息，不能利用单词的顺序信息，而这部分信息对于NLP来说非常重要。所以Transformer中使用位置Embedding保存单词在序列中的相对或绝对位置。

位置Embedding用PE表示，PE的维度与单词Embedding是一样的。PE可以通过训练得到，也可以使用某种公式计算得到。在Transformer中采用了后者，计算公式如下：

|  |  |
| --- | --- |
|  | （10） |
|  | （11） |

其中，*pos*表示单词在句子中的位置，*d*表示PE的维度（与词Embedding一样），2*i* 表示偶数的维度，2*i*+1表示奇数维度（）。使用这种公式计算PE有以下的好处：

使PE能够适应比训练集里面所有句子更长的句子，假设训练集里面最长的句子是有20个单词，突然来了一个长度为21的句子，则使用公式计算的方法可以计算出第21位的Embedding。

可以让模型容易地计算出相对位置，对于固定长度的间距k，PE(*pos*+*k*)可以用 PE(*pos*)计算得到。因为Sin(A+B) = Sin(A)Cos(B) + Cos(A)Sin(B)，Cos(A+B) = Cos(A)Cos(B) - Sin(A)Sin(B)。

将单词的词Embedding和位置Embedding相加，就可以得到单词的表示向量*x*，*x*就是Transformer的输入。

### 3.2 多头注意力

在第四节，我们已经知道怎么通过Self-Attention 计算得到输出矩阵Z，而Multi-Head Attention 是由多个 Self-Attention 组合形成的，下图是论文中 Multi-Head Attention 的结构图。

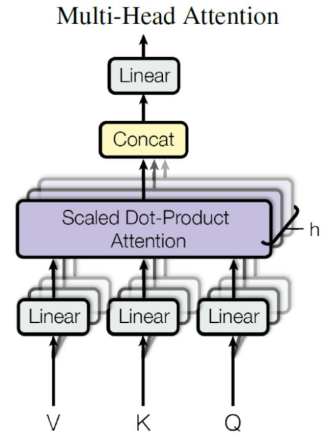


图6 多头注意力机制示意图

从上图可以看到 Multi-Head Attention 包含多个 Self-Attention 层，首先将输入*x*分别传递到*h*个不同的 Self-Attention 中，计算得到*h*个输出矩阵Z。下图是*h*=8时候的情况，此时会得到8个输出矩阵Z。

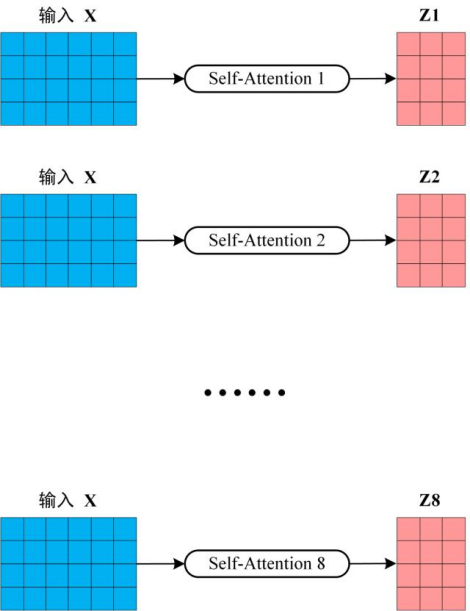


图7 8个注意力头计算示意图

得到 8 个输出矩阵到之后，Multi-Head Attention 将它们拼接在一起 （Concat），然后传入一个Linear层，得到 Multi-Head Attention 最终的输出***Z***。

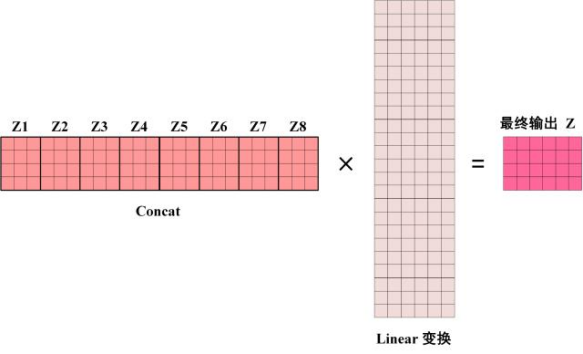
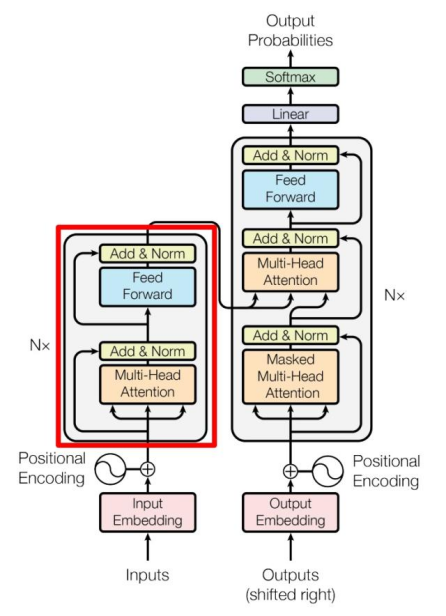


图8 8个注意力头计算结果的拼接

可以看到 Multi-Head Attention 输出的矩阵***Z***与其输入的矩阵*X*的维度是一样的。

### 3.3 Encoder 结构



上图红色部分是 Transformer 的 Encoder block 结构，可以看到是由 Multi-Head Attention, Add & Norm, Feed Forward, Add & Norm 组成的。刚刚已经了解了 Multi-Head Attention 的计算过程，现在了解一下 Add & Norm 和 Feed Forward 部分。

Add & Norm 层由 Add 和 Norm 两部分组成，其计算公式如下：

|  |  |
| --- | --- |
|  | （12） |
|  | （13） |

其中*X*表示Multi-Head Attention或者Feed Forward的输入，MultiHeadAttention(*X*) 和 FeedForward(*X*) 表示输出（输出与输入*X*维度是一样的，所以可以相加）。

Add指 *X*+MultiHeadAttention(*X*)，是一种残差连接，通常用于解决多层网络训练的问题，可以让网络只关注当前差异的部分，在 ResNet 中经常用到：



图9 残差连接

Norm指 Layer Normalization，通常用于RNN结构，Layer Normalization 会将每一层神经元的输入都转成均值方差都一样的，这样可以加快收敛。

Feed Forward 层比较简单，是一个两层的全连接层，第一层的激活函数为Relu，第二层不使用激活函数，对应的公式如下。

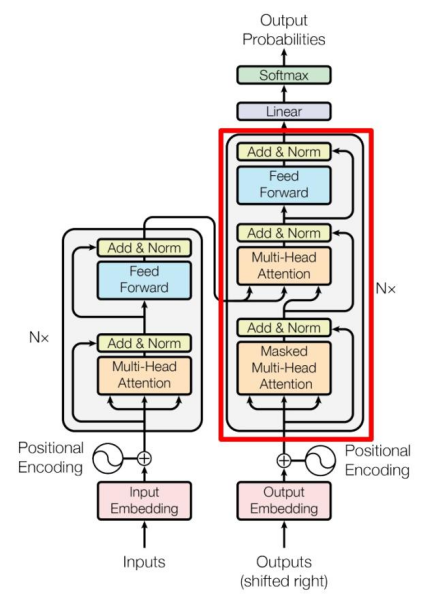
|  |  |
| --- | --- |
|  | （14） |

*X*是输入，Feed Forward 最终得到的输出矩阵的维度与*X*一致。

通过上面描述的 Multi-Head Attention, Feed Forward, Add & Norm 就可以构造出一个 Encoder block，Encoder block 接收输入矩阵，并输出一个矩阵。通过多个 Encoder block 叠加就可以组成 Encoder。

第一个 Encoder block 的输入为句子单词的表示向量矩阵，后续 Encoder block 的输入是前一个 Encoder block 的输出，最后一个 Encoder block 输出的矩阵就是编码信息矩阵C，这一矩阵后续会用到Decoder中。

### 3.4 Decoder 结构



上图红色部分为Transformer的Decoder block结构，与Encoder block相似，但是存在一些区别：

* 包含两个Multi-Head Attention 层。
* 第一个 Multi-Head Attention 层采用了 Masked 操作。
* 第二个 Multi-Head Attention 层的K, V矩阵使用 Encoder 的编码信息矩阵C进行计算，而Q使用上一个 Decoder block 的输出计算。
* 最后有一个 Softmax 层计算下一个翻译单词的概率。

Decoder block的第一个Multi-Head Attention采用了Masked操作，因为在翻译的过程中是顺序翻译的，即翻译完第 *i* 个单词，才可以翻译第*i*+1个单词。通过 Masked 操作可以防止第 *i* 个单词知道 *i*+1 个单词之后的信息。下面以“我有一只猫”翻译成 “I have a cat”为例，了解一下 Masked 操作。

在 Decoder 的时候，是需要根据之前的翻译，求解当前最有可能的翻译，如下图所示。首先根据输入 "<Begin>" 预测出第一个单词为 "I"，然后根据输入 "<Begin> I" 预测下一个单词 "have"。

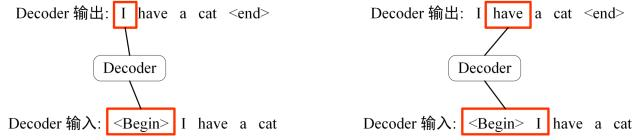


图10 Decoder输入输出示意图

Decoder 可以在训练的过程中使用 Teacher Forcing 并且并行化训练，即将正确的单词序列 (<Begin> I have a cat) 和对应输出 (I have a cat <end>) 传递到 Decoder。那么在预测第 *i* 个输出时，就要将第 *i*+1 之后的单词掩盖住，注意 Mask 操作是在 Self-Attention 的 Softmax 之前使用的，下面用 0 1 2 3 4 5 分别表示“<Begin> I have a cat <end>”。

第一步：是 Decoder 的输入矩阵和 Mask 矩阵，输入矩阵包含“<Begin> I have a cat”(0, 1, 2, 3, 4) 五个单词的表示向量，Mask 是一个 5×5 的矩阵。从 Mask 可以发现单词 0 只能使用单词 0 的信息，而单词 1 可以使用单词 0, 1 的信息，即只能使用之前的信息。

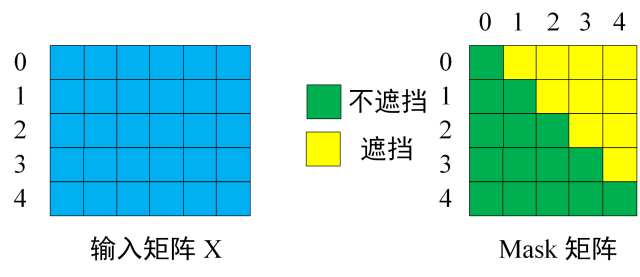


图11 Decoder的输入矩阵和 Mask 矩阵

第二步：接下来的操作和之前的 Self-Attention 一样，通过输入矩阵*X*计算得到Q、K、V矩阵。然后计算*Q*和的乘积。

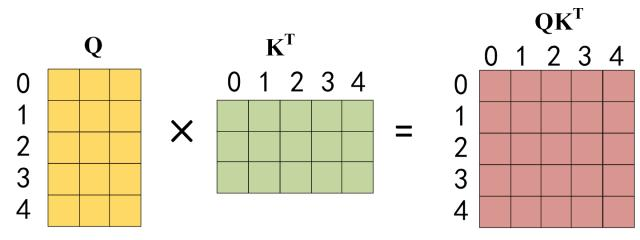


图12 Decoder的Self-Attention

第三步：在得到之后需要进行Softmax，计算attention score，我们在 Softmax 之前需要使用Mask矩阵遮挡住每一个单词之后的信息，遮挡操作如下：

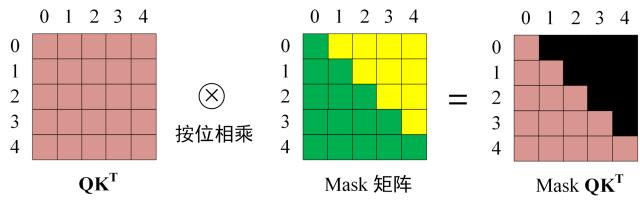


图13 Decoder的attention score计算

得到 Mask 之后在 Mask 上进行Softmax，每一行的和都为1。但是单词 0 在单词 1、2、3、4上的attention score都为0。

第四步：使用 Mask 与矩阵*V*相乘，得到输出*Z*，则单词1的输出向量是只包含单词 1 信息的。

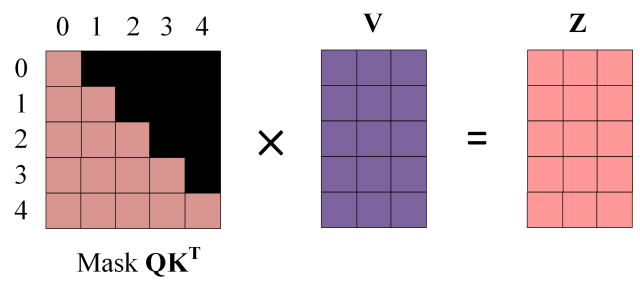


图14 输出向量计算

第五步：通过上述步骤就可以得到一个 Mask Self-Attention 的输出矩阵，然后和 Encoder 类似，通过 Multi-Head Attention 拼接多个输出，然后计算得到第一个 Multi-Head Attention 的输出***Z***，***Z***与输入***X***维度一样。

Decoder block 第二个 Multi-Head Attention 变化不大，主要的区别在于其中Self-Attention的K、V矩阵不是使用上一个Decoder block的输出计算的，而是使用Encoder 的编码信息矩阵C计算的。

根据 Encoder 的输出C计算得到K, V，根据上一个Decoder block 的输出Z计算Q（如果是第一个 Decoder block 则使用输入矩阵***X***进行计算），后续的计算方法与之前描述的一致。这样做的好处是在Decoder的时候，每一位单词都可以利用到Encoder所有单词的信息（这些信息无需 Mask）。

Decoder block 最后的部分是利用 Softmax 预测下一个单词，在之前的网络层我们可以得到一个最终的输出***Z***，因为Mask的存在，使得单词0的输出只包含单词0的信息，如下：

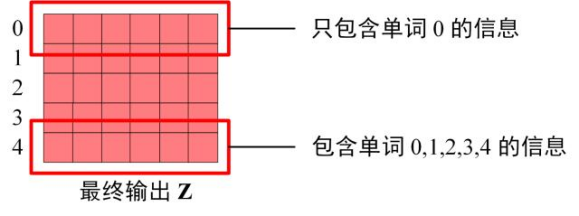


图15 Decoder的输出

Softmax根据输出矩阵的每一行预测下一个单词：

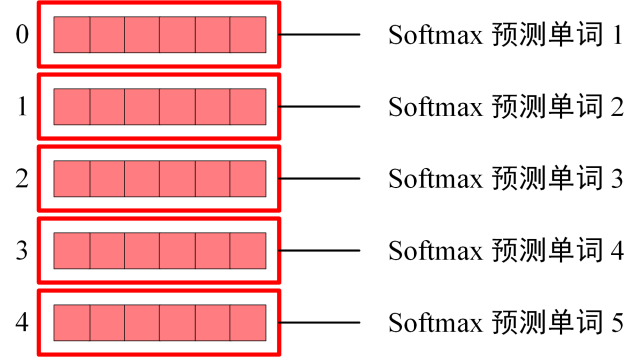


图16 Softmax预测每一个单词

这就是 Decoder block 的定义，与 Encoder 一样，Decoder 是由多个 Decoder block 组合而成。

# 参考文献

[1] [基于transformers的自然语言处理(NLP)\_\_入门图解transformer](https://datawhalechina.github.io/learn-nlp-with-transformers/#/./%E7%AF%87%E7%AB%A02-Transformer%E7%9B%B8%E5%85%B3%E5%8E%9F%E7%90%86/2.2-%E5%9B%BE%E8%A7%A3transformer)

[2] [翻译: 详细图解Transformer多头自注意力机制 Attention Is All You Need](https://blog.csdn.net/zgpeace/article/details/126635650)