第 5 章

转换器网络

转换器网络是比较流行和先进的深度学习架构之一，主要用于自然语言处理任务。自从转换器网络出现以来，它已经取代了循环神经网络和长短期记忆网络用于各种任务。一些新的自然语言处理模型，如 BERT、GPT 和 T5，都是基于转换器网络的。本章将详细研究转换器网络并了解其工作原理。

本章将从了解转换器的基本概念开始，然后将学习转换器如何使用编码器到解码器架构来完成语言翻译任务。在此之后，将通过探索每个编码器组件来详细检查转换器的编码器是如何工作的。在理解了编码器之后，将深入研究解码器并详细研究每个解码器组件。本章的最后会将编码器和解码器放在一起，看看转换器网络是如何作为一个整体工作的。

5.1 转换器介绍

循环神经网络由于其递归结构，能够对序列数据进行处理，是一种流行的模型用于生成文本序列任务。然而，循环模型的主要挑战之一是捕获长期依赖性。当循环神经网络在处理长序列数据时，模型需要在许多时间步长上记住之前的输入，以便正确预测当前输出。但是，由于梯度消失或爆炸的问题，长序列中的信息在传递到后续时间步长时会丢失或被模糊化，这就是所谓的长期依赖性问题。这会导致模型在长序列中表现不佳，无法捕捉到序列中的长期依赖性，从而影响了模型的性能。为了解决这个问题，出现了一些变种的循环神经网络模型，如长短时记忆网络和门控循环单元，它们通过引入不同的机制来控制信息在时间步长之间的流动。这些模型可以更好地处理长期依赖性，因此在生成文本序列任务中表现更好，然而循环神经网络的局限依然存在。

2017 年 12 月，Vaswani 等发表了他们的开创性论文 *Attention is All You Need*，在文中提出了新颖的转换器网络。转换器网络可以用于生成文本序列任务，而不需要使用循环结构，可以更好地处理长序列。转换器目前是多项自然语言处理任务的最先进模型。转换器的问世创造了自然语言处理领域的重大突破，也为 BERT、GPT-3、T5 等新的革命性架构铺平了道路。

转换器网络是一种由编码器(encoder)和解码器(decoder)组成的架构，用于序列到序列的任务，如机器翻译、对话生成等。在这种网络中，编码器将源序列映射到一个高维表示空间，而解码器则从该空间中解码目标序列。具体来说，在编码器中，输入序列中的每个单词都会被转换成一个向量表示，这些向量表示将经过一系列自注意力机制和前馈神经网络层进行编码。这些编码向量将包含源序列的上下文信息，并且可以用于生成目标序列。解码器在接收到编码器学习的表示后，使用类似的机制进行解码，逐步生成目标序列中的每个

单词。在生成每个单词时，解码器会在自注意力机制和编码器-解码器注意力机制之间进行交互，以获得源序列的上下文信息。在整个过程中，编码器和解码器之间的信息传递都是通过自注意力机制和编码器-解码器注意力机制进行的。自注意力机制可以帮助模型找到输入序列中不同位置的相关性，而编码器-解码器注意力机制可以帮助模型将目标序列中的每个单词与源序列中的相关单词进行对齐。总之，转换器网络是一种强大的序列到序列模型，它已被广泛应用于自然语言处理任务，并在很多任务中取得了很好的表现。假设需要将一个句子从英语转换为法语，如图 5-1 所示，将英语句子作为输入提供给编码器。编码器学习给定英语句子的表征并将该表征提供给解码器。解码器将编码器的表征作为输入并生成法语句子作为输出。

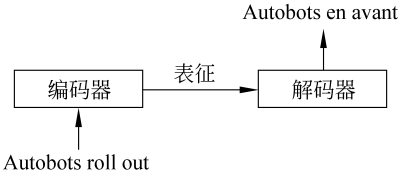

图 5-1　转换器的编码器和解码器

本章将把论文中描述的模型称为原始转换器网络。本章先从外部查看转换器网络的构建，在接下来的部分中，本章将探索模型的每个组件内部的内容。原始的转换器网络是由 6 个堆叠的编码器层和 6 个堆叠的解码器层组成的。每个编码器层和解码器层都有多头自注意力机制(multi-head self-attention)和前馈神经网络组成，这些层都使用残差连接和层归一化(add & norm)来帮助训练。在编码器中，第一个(底层)输入序列被馈送到多头自注意力机制，生成自注意力向量，再经过前馈神经网络。然后，这些向量被传递到下一个(更高层)编码器中，如此往复，直到到达顶部(最高层)编码器，输出最终的编码表示。在解码器中，第一个(底层)输入序列和编码器的输出编码表示被馈送到多头自注意力机制和多头编码器-解码器注意力机制。然后，这些向量被传递到下一个(更高层)解码器中，如此往复，直到到达顶部(最高层)解码器，生成最终的输出序列。最终预测是由输出层计算的，它将解码器的最后一层的输出向量映射到预测空间。如图 5-2 所示，左边部分是 6 层编码器栈，右边部分是 6 层解码器栈。

转换器网络是一种完全基于自注意力机制的神经网络架构，它完全摆脱了递归结构，因此可以高效地处理序列数据。在转换器网络中，每个输入标记都与序列中的所有其他标记进行交互，以生成上下文表示。这种交互基于自注意力机制，其中每个标记通过与自己和其他标记的关系来计算其在上下文中的重要性。这使得转换器网络在处理长序列时能够更好地捕捉全局依赖关系，并且在处理各种自然语言处理任务时表现出色。这一点将在接下来的部分中被详细介绍。

在图 5-2 左侧编码器中，输入序列通过自注意力子层和前馈网络子层进行编码，生成上下文表示；在图 5-2 右侧解码器中，目标输出序列通过两个注意力子层和一个前馈网络子层进行解码，生成最终的输出。在转换器网络中，没有 RNN、LSTM 或 CNN 这样的递归结构，而是使用自注意力机制来捕捉序列中的依赖关系。在生成文本序列任务中，自注意力机制可以替代传统的递归结构，以更好地处理长期依赖性问题。自注意力机制是一种“单词对单词”的操作。它通过对同一序列中不同位置的单词之间的关系进行计算来获取上下文信息，而不像传统的循环神经网络一样需要按顺序处理整个序列。在自注意力机制中，每个单词都与序列中的其他单词进行交互，并计算其与其他单词之间的相似度得分，然后将这些得分用于计算加权平均值以获取单词的表示。这种方法避免了循环神经网络中存在的梯度消

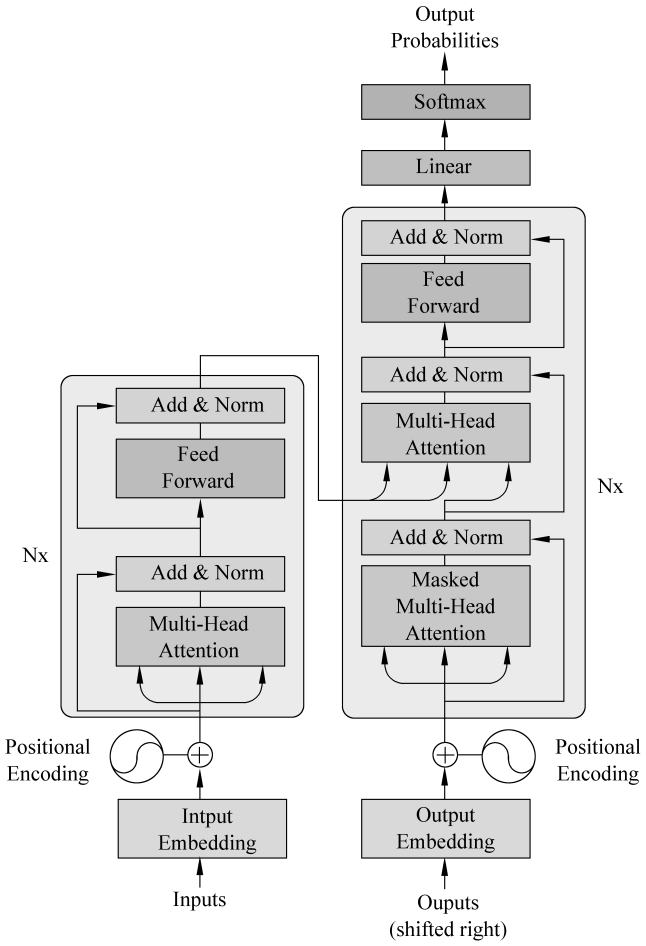

图 5-2　转换器网络

失问题，并且在处理长序列时具有更好的效果。例如，当机器翻译任务中，自注意力机制可以使模型更加关注源语言句子中与当前目标语言单词相关的部分。

来看以下序列："The cat sat on the mat."，注意力机制将在单词向量之间运行点积，并确定单词在所有其他单词中的最强关系，包括与自身相关的关系(cat 和 mat)。注意力机制将在单词向量之间运行点积，并计算每个单词与所有其他单词之间的相关性分数，这些分数可以被视为该单词对于序列中其他单词的重要性。在这个例子中，注意力机制将为每个单词计算一个注意力权重向量，其中每个元素表示该单词与序列中其他单词之间的相关性分数。例如，当处理单词 cat 时，注意力机制会计算该单词与序列中其他单词的相关性分数，包括与自身相关的分数。这些分数将被用于计算一个注意力权重向量，该向量将指示模型在处理该单词时应该关注哪些其他单词。注意力机制的输出将被用于计算每个单词的上下文向量，这个上下文向量将包含该单词与序列中其他单词的相关信息。在这个例子中，注意力机制将帮助模型捕捉到 cat 与 mat 之间的联系，从而更准确地理解这个句子的含义。

自注意力机制将提供更深层次的词与词之间的关系并产生更好的结果。对于每个自注意力子层，原始的转换器网络不是并行运行一个而是 8 个自注意力机制以加快计算速度，将

在5.2节“理解编码器”中探讨这种架构，这个过程被命名为“多头自注意力”。多头自注意力是一种注意力机制的变体，它将注意力机制分解为多个头部(也称为子空间)，每个头部计算一组查询、键和值的相似性，然后将这些头部的结果拼接在一起形成最终的注意力表示。在多头注意力中，模型学习多组查询、键和值的线性投影，然后分别计算它们的注意力表示。这些头部可以并行计算，因此多头注意力可以更快地处理大规模数据。此外，多头注意力还可以帮助模型学习多个不同的表示，因为不同的头部可能关注不同的信息。多头注意力在自然语言处理任务中非常有用，如语言建模、机器翻译和文本分类。在这些任务中，多头自注意力可以帮助模型更好地捕捉不同层次的语义信息，从而提高模型的性能。例如，在机器翻译任务中，多头自注意力可以帮助模型更好地关注不同的单词和短语，从而提高翻译的准确性。在转换器中的编码器和解码器是如何将英文句子转换为法语句子的呢？编码器和解码器内部发生了什么？首先，详细研究一下编码器。

5.2　理解编码器

转换器网络的编码器由一系列相同的层组成，每个层都包含两个子层，分别是多头自注意力机制层(multi-head self-attention layer)和全连接前馈神经网络层(feed-forward neural network layer)。

多头自注意力机制层可以同时将一个输入序列中的所有单词作为输入，并在保留单词之间关系的同时，计算出每个单词的“注意力权重”，并使用这些权重来重新组合输入序列中的单词向量，得到新的表示。这个新的表示是基于注意力机制的，可以更好地反映输入序列中不同单词之间的关系。全连接前馈神经网络层会对经过多头自注意力机制层得到的新的表示进行进一步处理，以获得更高层次的抽象表示。该层通常由两个全连接层组成，分别通过一个ReLU激活函数进行激活，从而将输入映射到新的空间中。在编码器的每层中，都会在多头自注意力机制层和全连接前馈神经网络层之间添加一个残差连接(residual connection)，并使用一个标准化层(normalization layer)来归一化输出。残差连接可以使网络更容易训练，并且可以防止信息在传递过程中丢失。标准化层可以帮助网络更好地处理梯度，并提高训练速度和效果。编码器的多层结构使得网络可以对输入序列进行多次处理，从而逐步捕捉输入序列中的各个层次的语义信息，并提高整个网络对序列建模的能力。

转换器由一堆 N 个编码器组成。一个编码器的输出作为输入发送到其上面的编码器。如图5-3所示，图中有一堆 N 个编码器，每个编码器将其输出发送到其上方的编码器。最终编码器返回给定源语句的表示作为输出。将源句作为编码器的输入，并将源句的表征作为输出，如图5-3所示。

在论文 *Attention is All You Need* 中，作者使用了 $N=6$，这意味着框架将6个编码器一个接一个地堆叠起来，可以尝试不同的 N 值。但是，为了简单和更好地理解，我们简化编码器测堆叠的层数为 $N=2$。现在的问题是编码器究竟是如何工作的？如何将源语句生成表征的？为了理解这一点，深入了解编码器并查看其组件。如图5-4所示，这里显示了两个编码器的堆叠，Encoder_1被详细扩展，从图中我们了解到以下4点。

(1) 输入序列中的每个单词将被转换为一个嵌入向量，并加上位置编码，以表示其在序列中的位置。这些嵌入向量和位置编码将被用作输入序列的表示，然后被输入最底部的编

码器(Encoder_1)。

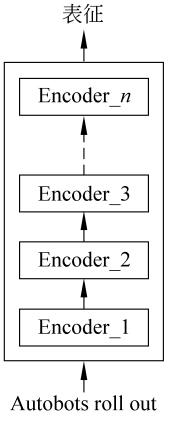

图 5-3 *N* 个编码器的堆栈

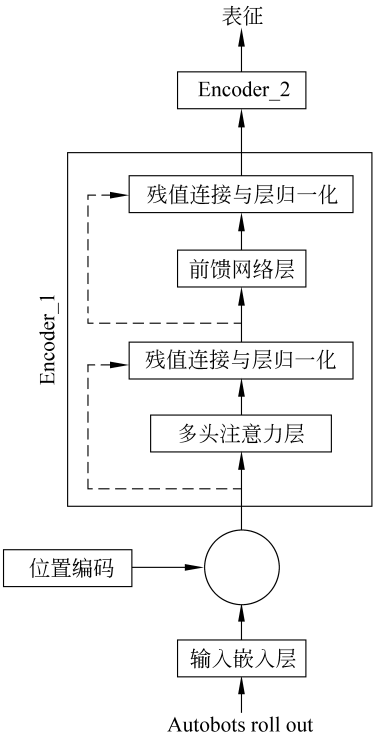

图 5-4 展开 Encoder_1 编码器

(2) Encoder_1 使用一个多头自注意力机制对输入进行处理。自注意力机制利用输入序列中所有单词的嵌入向量计算注意力分数,以此来确定每个单词与其他单词的相对重要性,并生成一个注意力矩阵作为输出。

(3) 将注意力矩阵作为输入提供给 Encoder_1 中的下一个子层,即前馈神经网络。前馈神经网络是一个简单的全连接前馈网络,用于处理自注意力机制的输出,生成 Encoder_1 的最终表征。

(4) 将 Encoder_1 的输出作为输入提供给 Encoder_2,并重复步骤(3)~(4),直到达到所需的编码器层数。

对于一般的转换器网络,可以将多个编码器层堆叠在一起以构建更强大的模型,其中每个编码器层都可以逐步学习输入序列的不同特征和语义信息。最终编码器的输出将是给定输入句子的表征,也称为编码器表征。当使用转换器网络进行序列到序列的任务时,可以将从最终编码器获得的编码器表示作为解码器的输入,并尝试生成目标句子。

从图 5-4 中可以了解到,所有的编码器块都是一样的,可以观察到每个编码器块都由两个子层组成。

- 多头自注意力层。
- 前馈网络层。

在转换器网络中,每个主要子层的输出都包含了残差连接和层归一化函数。残差连接的作用是将子层未处理的输入和子层的输出相加,保留了输入的信息,避免了信息的丢失。

层归一化函数则用于规范化子层输出的值域，以避免过度的梯度消失或爆炸。因此，每层的归一化输出可以表示为

$$\text{LayerNorm}(x+\text{Sublayer}(x))$$

其中，Sublayer(x)表示主要子层的输出；x 表示子层的输入。LayerNorm 函数对输出进行规范化处理。

在转换器网络中，编码器层的每层都具有相同的结构，但是每层的内容与前一层并不完全相同。这是因为每层都需要处理不同的输入，因此每层的内容可能不同。例如，在编码器的第一层，需要将输入转换为输入嵌入，并添加位置编码。这个子层仅存在于堆栈的底层，而在其他五层中并不包含这个子层。这可以确保编码器输入在所有层中都是稳定的，因为在每层中都遵循相同的输入表示。此外，在注意力子层中，每层都需要计算输入序列中所有单词的注意力分数，因此每层的注意力分数矩阵可能不同。因此，尽管每一层的结构相同，但是每一层的具体内容和输出都不完全相同。每一层都可以看作从前一层学习的表示的进一步提炼，因此每一层都在探索输入序列的不同方面。多头注意力机制在每一层中的作用是找到与每个单词最相关的其他单词，从而为输入序列中的每个位置提供更全面的信息。因此，在每一层中，多头注意力机制都可以理解为执行类似的任务，但是由于输入序列的不同部分可能具有不同的关联模式，因此每一层都需要独立地探索和学习这些关联。这种层层逐步提炼的过程，以及每一层的不同探索和学习方式，使得转换器网络能够对输入序列进行深入的理解和建模。

转换器网络的设计者引入了一个非常有效的约束。模型的每个子层的输出都有一个恒定的维度，包括嵌入层和残差连接。这个维度是 d_{model} 并且可以根据目标设置为另一个值，在原始的转换器网络中 $d_{\text{model}}=512$。实际上所有的关键操作都是点积，维度保持稳定减少了计算操作的数量，减少了资源的消耗，并且更容易在信息流经模型时对其进行跟踪。这个编码器的全局视图(图 5-4)显示了转换器高度优化的架构。在以下部分中，将详细介绍每个子层和机制。下面将从嵌入子层开始。

5.2.1 输入嵌入层

在转换器网络中，输入嵌入子层使用学习的嵌入将输入标记转换为维度 $d_{\text{model}}=512$ 的向量。嵌入子层的工作方式与其他标准转换模型类似。标记器将句子转换为标记，每个标记器都有自己的方法，但结果是相似的。例如，应用于序列“Transformer is an innovative NLP model!”的分词器将在一种模型中生成以下分词。

```
['the', 'transform', 'er', 'is', 'an', 'innovative','n', 'l', 'p', 'model', '!']
```

注意到此分词器将字符串规范化为小写并将其截断为子部分。分词器通常会提供将用于嵌入过程的整数表示。例如

```
Text="The cat slept on the couch. It was too tired to get up."
tokenized text=[1996, 4937, 7771, 2006, 1996, 6411, 1012, 2009, 2001, 2205, 5458,
               2000, 2131, 2039, 1012]
```

本书选择 Word2Vec 嵌入方法的 Skip-gram 架构来说明转换器的嵌入子层。Skip-gram 将专注于单词窗口中的中心单词并预测上下文单词。例如，word(i)是窗口大小为 2

中心词，Skip-gram 模型将分析词 word($i-2$)、word($i-1$)、word($i+1$)和 word($i+2$)，然后窗口将滑动并重复该过程。Skip-gram 模型通常包含输入层、权重、隐含层和包含词嵌入的输出。假设需要对以下句子进行嵌入。

```
The black cat sat on the couch and the brown dog slept on the rug.
```

本书将专注于两个词“black”和“brown”，这两个词的词嵌入向量应该是相似的。由于必须为每个单词生成一个大小为 $d_{\text{model}}=512$ 的向量，因此每个单词获得一个大小为 512 的向量嵌入。

为了验证为这两个词生成的词嵌入，可以使用余弦相似度来查看“black”和“brown”这两个词的词嵌入是否相似。余弦相似度使用欧几里得范数(L^2 范数)在单位球体中创建向量。正在比较的向量的点积是这两个向量的点之间的余弦。示例嵌入中大小为 $d_{\text{model}}=512$ 的黑色向量与大小为 $d_{\text{model}}=512$ 的棕色向量之间的余弦相似度为

```
cosine_similarity(black, brown)=[[0.9998901]]
```

Skip-gram 模型学习到的词嵌入中，黑色和棕色单词向量在向量空间中靠近彼此，从而被检测到是词典中的颜色子集，这些词嵌入向量可以作为输入数据被传递给转换器网络的后续层，从而提供有用的信息。然而，由于没有额外的向量或信息指示单词在序列中的位置，因此丢失了大量信息。转换器网络的设计者提出了另一个创新功能：位置编码。看看位置编码是如何工作的。

5.2.2 位置编码

如果假定输入句为：“Autobots roll out”，对于循环神经网络，每个输入(如单词)都会依次传递给网络，并在网络的隐藏状态中建立一种记忆。这种记忆可以捕捉到先前输入的信息，并将其用于下一个输入的处理。在处理完整个句子后，网络会输出一个结果，表示对整个句子的理解。对于转换器网络，所有输入单词都会同时传递给网络，每个单词都会与句子中的所有其他单词一起处理。这是通过注意力机制实现的，网络会自动学习哪些单词在给定的上下文中最重要。

因此，对于输入句子“Autobots roll out”，循环神经网络会逐字处理句子，而转换器网络会将整个句子并行处理。这种差异在处理长句子时尤为明显，因为循环神经网络需要记住先前的信息，因此处理长句子时可能会出现梯度消失或梯度爆炸的问题。而转换器可以更好地处理长句子，因为它不需要记住先前的信息，而是通过注意力机制选择要关注的单词。并行输入单词有助于减少训练时间，也有助于学习长期依赖，然而问题在于既然将单词并行地输入网络，那么如果不保留单词顺序网络如何理解句子的意思呢？了解单词在句子中的位置很重要，有助于理解句子的意思，所以应该给转换器一些关于词序的信息，这样它才能理解句子。现在更详细地探讨一下。

对于给定的句子“Autobots roll out”。首先得到句子中每个单词的嵌入，表示为 d_{model}。如果嵌入维度 d_{model} 是 4，那么输入矩阵的维度将是[句子长度×嵌入维度]=[3×4]。输入矩阵 $\boldsymbol{X}$(嵌入矩阵)表示输入句子“Autobots roll out”，设输入矩阵 $\boldsymbol{X}$ 如下：

$$\boldsymbol{X}=\begin{bmatrix} 1.769 & 2.22 & 3.4 & 5.8 \\ 7.3 & 9.9 & 8.5 & 7.1 \\ 9.1 & 7.1 & 0.85 & 10.1 \end{bmatrix}$$

现在，如果输入矩阵 $\boldsymbol{X}$ 直接传递给网络，将无法理解词序，因此需要添加一些指示词序（词的位置）的信息。为此，在转换器网络中这种信息是通过引入位置编码（positional encoding）来实现的，就是表示单词在句子中的位置（词序）的编码。位置编码的维数与输入矩阵 $\boldsymbol{X}$ 的维数相同。现在，在将输入矩阵（嵌入矩阵）直接传递到网络之前，我们包含位置编码，因此只需要将位置编码矩阵 $\boldsymbol{PE}$ 添加到嵌入矩阵 $\boldsymbol{X}$，然后将其作为输入提供给网络，所以现在输入矩阵不仅有单词的嵌入，还有单词在句子中的位置。

$$\boldsymbol{X}=\boldsymbol{X}+\boldsymbol{PE}=\begin{bmatrix}1.769 & 2.22 & 3.4 & 5.8\\ 7.3 & 9.9 & 8.5 & 7.1\\ 9.1 & 7.1 & 0.85 & 10.1\end{bmatrix}+\begin{bmatrix}0 & 1 & 0 & 1\\ 0.841 & 0.54 & 0.01 & 0.99\\ 0.909 & -0.416 & 0.02 & 0.99\end{bmatrix}$$

$$=\begin{bmatrix}1.769 & 3.22 & 3.4 & 6.8\\ 8.14 & 10.44 & 8.51 & 8.09\\ 10.0 & 6.68 & 0.87 & 11.09\end{bmatrix}$$

那么，位置编码矩阵 $\boldsymbol{PE}$ 是怎么获得呢？给转换器网络一些关于词序的信息非常重要，因为句子中单词的顺序对于理解句子的意思至关重要。位置编码是一个向量，它与每个单词的词向量相加，以提供有关单词在句子中位置的信息。这个向量的形式是根据正弦和余弦函数计算的，公式如下：

$$\boldsymbol{PE}_{(\text{pos},2i)}=\sin\left(\frac{\text{pos}}{10\ 000^{2i/d_{\text{model}}}}\right)\quad \boldsymbol{PE}_{(\text{pos},2i+1)}=\cos\left(\frac{\text{pos}}{10\ 000^{2i/d_{\text{model}}}}\right)$$

在上面的等式中，pos 表示单词在句子中的位置；i 表示词嵌入的位置（$0\leqslant i<d_{\text{model}}/2$）；$d_{\text{model}}$ 是词向量的维度。因此，对于输入句子“Autobots roll out”，每个单词都将与其位置编码相加，然后并行传递给转换器网络。这使得网络能够同时捕捉到单词之间的关系和单词在句子中的位置，从而更好地理解整个句子的意思。通过使用这个等式，可以写出以下内容：

$$\boldsymbol{PE}=\begin{bmatrix}\sin\left(\frac{\text{pos}}{10\ 000^{\circ}}\right) & \cos\left(\frac{\text{pos}}{10\ 000^{\circ}}\right) & \sin\left(\frac{\text{pos}}{10\ 000^{2/4}}\right) & \cos\left(\frac{\text{pos}}{10\ 000^{2/4}}\right)\\ \sin\left(\frac{\text{pos}}{10\ 000^{\circ}}\right) & \cos\left(\frac{\text{pos}}{10\ 000^{\circ}}\right) & \sin\left(\frac{\text{pos}}{10\ 000^{2/4}}\right) & \cos\left(\frac{\text{pos}}{10\ 000^{2/4}}\right)\\ \sin\left(\frac{\text{pos}}{10\ 000^{\circ}}\right) & \cos\left(\frac{\text{pos}}{10\ 000^{\circ}}\right) & \sin\left(\frac{\text{pos}}{10\ 000^{2/4}}\right) & \cos\left(\frac{\text{pos}}{10\ 000^{2/4}}\right)\end{bmatrix}$$

从这个矩阵可以看出，在位置编码中当 i 为偶数时使用 sin 函数，当 i 为奇数时使用 cos 函数，简化的矩阵可以写成下面的形式。

$$\boldsymbol{PE}=\begin{bmatrix}\sin(\text{pos}) & \cos(\text{pos}) & \sin\left(\frac{\text{pos}}{100}\right) & \cos\left(\frac{\text{pos}}{100}\right)\\ \sin(\text{pos}) & \cos(\text{pos}) & \sin\left(\frac{\text{pos}}{100}\right) & \cos\left(\frac{\text{pos}}{100}\right)\\ \sin(\text{pos}) & \cos(\text{pos}) & \sin\left(\frac{\text{pos}}{100}\right) & \cos\left(\frac{\text{pos}}{100}\right)\end{bmatrix}$$

在输入的句子中，单词“Autobots”在 0 的位置，“roll”在 1 的位置，“out”在 2 的位置，替换 pos 值，最终的位置编码矩阵 $\boldsymbol{P}$ 如下：

$$\boldsymbol{PE}=\begin{bmatrix}\sin(0) & \cos(0) & \sin\left(\frac{0}{100}\right) & \cos\left(\frac{0}{100}\right)\\ \sin(1) & \cos(1) & \sin\left(\frac{1}{100}\right) & \cos\left(\frac{1}{100}\right)\\ \sin(2) & \cos(2) & \sin\left(\frac{2}{100}\right) & \cos\left(\frac{2}{100}\right)\end{bmatrix}=\begin{bmatrix}0 & 1 & 0 & 1\\ 0.841 & 0.54 & 0.01 & 0.99\\ 0.909 & -0.416 & 0.02 & 0.99\end{bmatrix}$$

在计算出位置编码 $\boldsymbol{PE}$ 之后，只须对嵌入矩阵 $\boldsymbol{X}$ 执行逐元素加法，并将修改后的输入矩阵提供给编码器。现在，重新审视编码器架构。如图 5-5 所示，这里显示了单个编码器块。正如所观察到的，在将输入直接发送到编码器之前，首先得到输入嵌入矩阵，然后将位置编码添加到其中。

例如，在句子“*The black cat sat on the couch and the brown dog slept on the rug*.”中，*Skip-gram* 产生了两个非常接近的向量，检测到 *black* 和 *brown* 构成了词典的颜色子集，但是由于没有额外的向量或信息指示单词在序列中的位置。在句子中，可以看到 *black* 在位置 $pos=2$，*brown* 在位置 $pos=10$。如果对这两个字进行位置编码，会得到一个大小为 512 的位置编码向量，现在想检查一下这个结果是否有意义，用于词嵌入的余弦相似度函数可以方便地更好地可视化位置的接近度：$cosine\_similarity(pos(2)), pos(10)=[[0.860\ 001\ 3]]$。与词嵌入的相似度比较，位置的编码显示出更低的值，所以位置编码已经把这些词区别开了。现在的问题是如何将位置编码添加到词嵌入向量。如果以 *black* 词嵌入为例，将其定义为 $\boldsymbol{y}_1$，就可以将其添加到通过位置编码函数获得的位置向量 $\boldsymbol{pe}(2)$中（表示 pos=2 位置向量），将获得输入词 black 的位置编码，如图 5-6 所示。

$$\boldsymbol{pc}(\text{black})=\boldsymbol{y}_1+\mathbf{pe}(2)$$

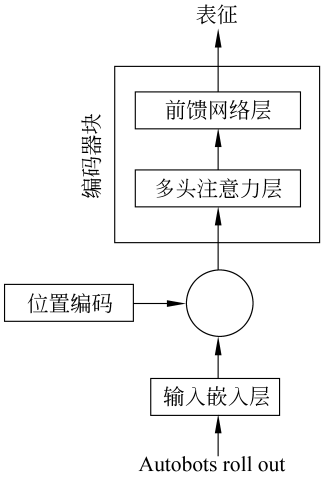

图 5-5 单个编码器块

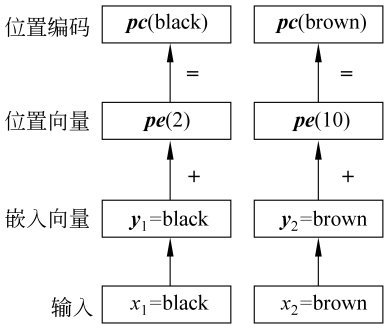

图 5-6 black 和 brown 的位置编码

这个解决方案很简单，但是这种方法可能会丢失一些有关输入单词的信息，因为位置编码向量的值通常比嵌入向量的值小很多，这可能会减少输入单词的贡献。因此，一些研究人员提出了一些替代的位置编码方案。例如，使用基于时间的位置编码或可学习的位置编码，以更好地捕捉输入单词的信息并提高模型性能。

另外，有些方法可以增加 $\boldsymbol{y}_1$ 的值以确保词嵌入层的信息可以在后续层中有效地使用，

其中之一是将任意值添加到 $\boldsymbol{y}_1$，如 $\boldsymbol{y}_1 * \sqrt[2]{d_{\text{model}}}$。将位置编码矩阵与嵌入矩阵相加是实现位置编码的一种简单而有效的方法，但是这种方法可能会导致位置编码矩阵覆盖嵌入矩阵的部分信息，从而可能丢失一些有用的信息。为了解决这个问题，可以采用可学习的位置编码方法，如之前所述。这种方法允许网络学习如何将位置编码向量与词向量结合起来，以便更好地捕捉单词之间的关系和顺序，同时保留词嵌入层的信息。在这种方法中，位置编码向量被初始化为任意值，并在训练过程中进行优化。这使得网络可以自己学习如何结合位置编码向量和词向量，以便更好地理解句子中单词的顺序和含义。因此，通过使用可学习的位置编码方法，转换器网络可以更好地利用词嵌入层的信息，并且在保留单词顺序的同时，更好地理解整个句子的含义。

相同的操作适用于单词 brown 和序列中的所有其他单词。该算法的输出不是基于规则的，在每次运行期间可能会略有不同。可以将余弦相似度函数应用于 black 和 brown 的位置编码向量：cosine_similarity(***pc***(black)),***pc***(brown)=[[0.962 709 4]]。词嵌入的相似度非常高为 0.99，位置编码向量将这两个词分开，相似度值较低为 0.86。最后将每个词的词嵌入向量添加到其各自的位置编码向量，这使两个词的余弦相似度达到了 0.96，现在每个词的位置编码现在包含初始词嵌入信息和位置编码值。

5.2.3　多头自注意力层

多头自注意力层是一种深度学习中常用的层，用于处理序列数据，如自然语言文本。它是基于注意力机制的一种变种。在自注意力机制中，输入序列中的每个元素都会被编码为一个向量表示。这些向量将被用于计算与其他元素的相似度，从而决定每个元素对其他元素的关注程度，最终生成一个加权的表示。多头自注意力层则通过将输入向量分别映射到多个子空间，并在每个子空间中执行独立的自注意力计算来增强模型的表现能力。具体来说，多头自注意力层将输入序列中的每个元素分别映射到多个子空间，每个子空间都有自己的注意力计算过程。这些子空间的输出被串联起来，并通过一个全连接层进行线性变换，得到最终的输出。通过多头自注意力层的操作，模型可以更好地学习输入序列中的关系和模式，提高模型的表示能力和泛化能力，使得模型在自然语言处理等任务中表现更加出色。

要了解多头注意力的工作原理，首先需要了解自注意力机制。通过一个例子来理解自注意力机制，考虑句子：A dog ate the food because it was hungry。在句子中，代词 it 可以表示 dog 或 food。通过阅读句子，可以很容易地理解它所暗示的代词是狗而不是食物，但是模型如何理解在给定的句子中，代词 it 暗示 dog 而不是 food，这就是自注意力机制可以帮助地方。

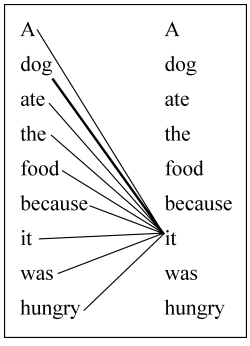

图 5-7　自注意力示例

在给定的句子中，首先模型计算单词 A 的表示，然后计算单词 dog 的表示，然后计算单词 ate 的表示，以此类推。在计算每个单词的表示时，它会将每个单词与句子中的所有其他单词相关联，以便更多地了解该单词。例如，当计算单词 it 的表示时，模型将单词 it 与句子中的所有单词联系起来，以更多地了解单词 it。如图 5-7 所示，为了计算单词 it 的表示，模型将单词 it 与句子中的所有单词相关联。通过将单词 it 与句子中的所有单词相关联，模型可以理解单词 it 与 dog 和 food 之间的相关性。正如所观察到的，连接 it