9 The Transformer

> “The true art of memory is the art of attention ”
>
> ——Samuel Johnson, Idler #74, September 1759

在本章中，我们介绍了Transformer，这是构建大型语言模型的标准架构。基于Transformer的大型语言模型彻底改变了语音和语言处理领域。事实上，本教材中的每一章都会使用这些模型。目前我们将重点放在**从左到右（有时称为因果或自回归）**语言建模上，其中给定一系列输入标记，模型通过条件化在先前上下文上逐个预测输出标记。

Transformer是一种具有特定结构的神经网络，包括一种称为自注意力（self-attention）或多头注意力（multi-head attention）的机制。注意力可以理解为通过关注并整合周围标记的信息来构建标记含义的上下文表示，从而帮助模型学习标记之间在较大范围内的关系。

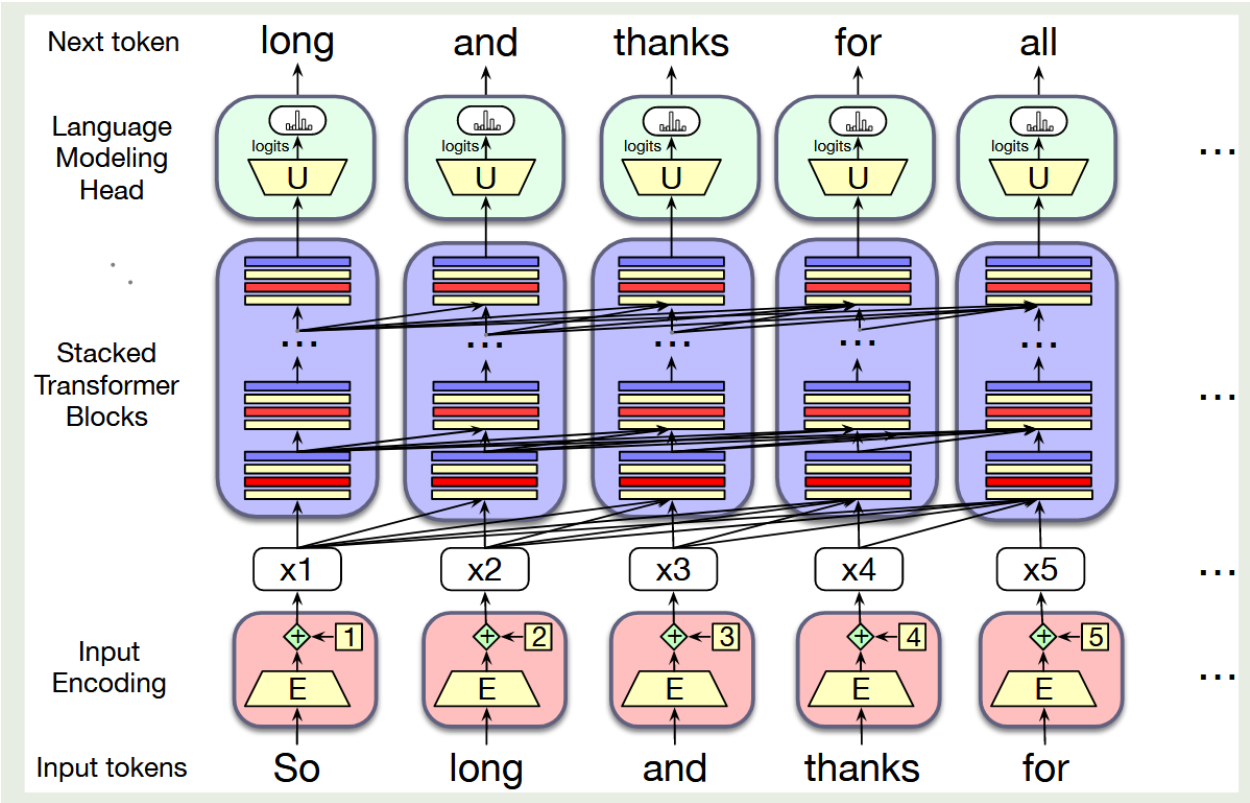

图 9.1 一个（从左到右）转换器的架构，显示每个输入令牌如何被编码，通过一系列堆叠的转换器块，然后是一个语言模型头部，用于预测下一个令牌。

图9.1概述了Transformer架构。Transformer有三个主要组成部分。中心部分是多层Transformer块。每个块是一个多层网络（包括多头注意力层、前馈网络和层归一化步骤），将列 i 中的输入向量 x_i（对应于输入标记i）映射到输出向量 h_i。n 个块的集合将整个上下文窗口的输入向量 $(x_1, ..., x_n)$ 映射到长度相同的输出向量窗口 $(h_1, ..., h_n)$。一列中可以包含从12到96个或更多的堆叠块。

Transformer块的列之前是输入编码组件，该组件将输入标记（例如单词“thanks”）处理为上下文向量表示，使用嵌入矩阵 $\mathbf{E}$ 以及一种标记位置编码机制。每一列之后是一个语言建模头部，该头部接收最终Transformer块生成的嵌入，经过反嵌入矩阵 $\mathbf{U}$ 和词汇表上的softmax处理以生成该列的单个标记。

基于Transformer的语言模型非常复杂，因此其细节将在接下来的五章中逐步展开。在接下来的章节中，我们将介绍多头注意力、Transformer块的其他部分，以及输入编码和语言建模头部组件。第10章讨论语言模型的预训练方式，以及如何通过采样生成标记。第11章介绍了掩蔽语言建模和双向Transformer编码模型BERT系列。第12章展示了如何通过提供指令和示例来提示LLM执行NLP任务，以及如何使模型与人类偏好对齐。第13章将介绍具有编码器-解码器架构的机器翻译。

9.1 注意力机制

回忆第六章中的内容，对于**word2vec**和其他静态嵌入，词语的含义表示始终是相同的向量，与上下文无关。例如，单词“chicken”总是由相同的固定向量表示。因此，表示“it”这样的代词的静态向量可能会在某种程度上编码出这是用于动物和无生命实体的代词。但在具体上下文中，它具有更丰富的含义。考虑它在以下两个句子中的使用：

(9.1) **The chicken** didn’t cross the road because **it** was too tired.

(9.2) **The chicken** didn’t cross the road because **it** was too wide.

在(9.1)中，“it”指的是鸡（即，读者知道鸡很累），而在(9.2)中，“it”指的是道路（读者知道道路太宽）。也就是说，如果我们要计算这个句子的含义，我们需要让“it”的含义在第一句中与鸡相关联，而在第二句中与道路相关，依赖于上下文。

此外，考虑像因果语言模型那样从左到右地阅读，处理句子直到“it”这个词：

(9.3) **The chicken** didn’t cross the road because **it**

在此时，我们还不知道“it”最终会指代什么！因此，针对此时的“it”的表示可能同时包含“chicken”和“road”的信息，因为读者正在试图猜测接下来会发生什么。

这种词语与远距离词语之间存在丰富语言关系的现象普遍存在于语言中。再考虑两个例子：

(9.4) The **keys** to the cabinet **are** on the table.

(9.5) I walked along the **pond**, and noticed one of the trees along the **bank**.

在(9.4)中，短语“The keys”是句子的主语，在英语和许多语言中，主语必须与动词在人称和数上保持一致；在这里，两者都是复数。在英语中，我们不能用像“is”这样单数动词搭配“keys”这样的复数主语（我们将在第十八章进一步讨论一致性问题）。在(9.5)中，我们知道“bank”指的是池塘或河流的边而不是金融机构，因为上下文包括“pond”这样的词（我们将在第十一章进一步讨论词义）。

这些例子的关键在于，这些帮助我们在上下文中计算词义的上下文词语在句子或段落中可能相隔甚远。Transformer可以通过整合这些有用的上下文词语的含义来构建词语含义的上下文表示，即**上下文嵌入**。在Transformer中，通过层层叠加，我们逐渐构建出更加丰富的输入标记含义的上下文化表示。在每一层，我们通过结合来自前一层的标记 i 的信息和邻近标记的信息来计算标记 i 的表示，为每个位置的每个词语生成上下文化表示。

注意力机制是Transformer中的一个机制，用来对上下文中适当的其他标记的表示进行加权和组合，从而在第 $k-1$ 层中构建出第 k 层的标记表示。

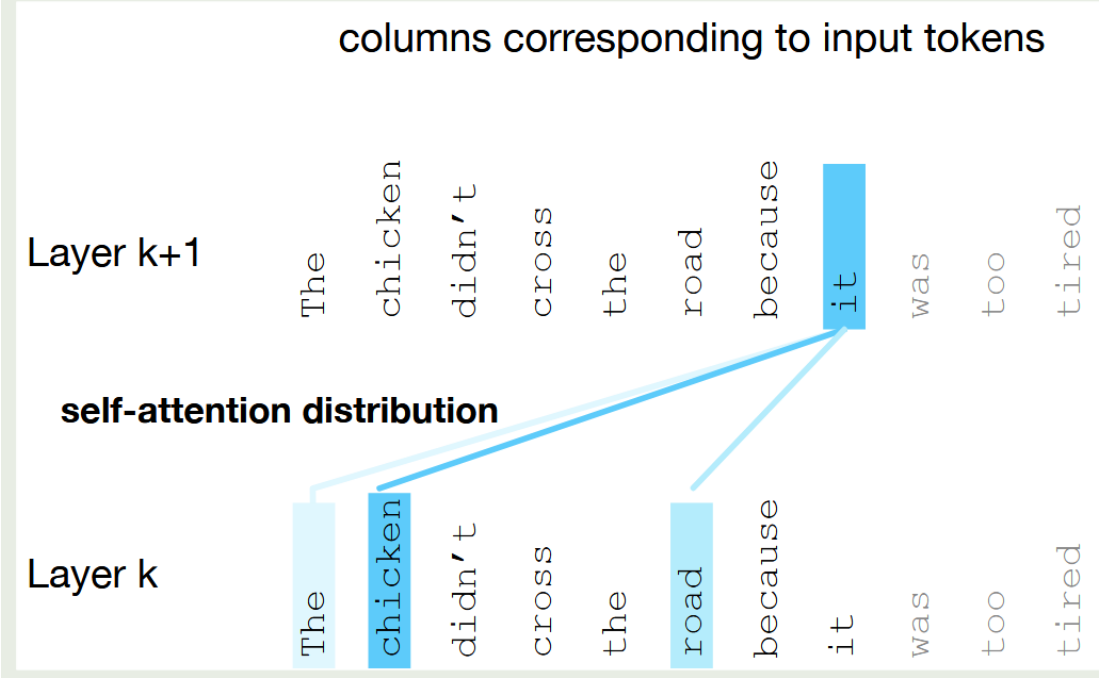

图 9.2 第 k+1 层中用于计算单词 it 表示的表示的自我注意力权重分布 α。在计算表示时，我们以不同的方式关注第 l 层中的各个单词，较深的阴影表示较高的自我注意力值。请注意，转换器高度关注与“鸡肉”和“道路”对应的列，这是一个合理的结果，因为在它出现的点，它可能与鸡肉或道路进行核心关联，因此我们希望它所表示的特征能够利用这些较早单词的表示。此图改编自 Uszkoreit (2017)。

图9.2展示了一个简化的Transformer示例（Uszkoreit, 2017）。图中描述了当当前标记为“it”时的情景，我们需要在Transformer的第k +1层为该标记计算上下文化表示，引用第k层的每个前置标记的表示。图中使用颜色来表示上下文词语的注意力分布：标记“chicken”和“road”都具有较高的注意力权重，这意味着在计算“it”的表示时，我们将主要参考“chicken”和“road”的表示。这在构建“it”的最终表示时很有用，因为它最终将与“chicken”或“road”形成核心指代关系。

现在让我们来探讨这种注意力分布是如何表示和计算的。

9.1.1 注意力机制的正式定义

如我们所述，注意力计算是一种在Transformer的特定层中为一个标记计算向量表示的方法，通过有选择性地关注并整合前一层的先前标记信息。注意力机制接收对应于位置 i 的输入标记的表示 x_i 以及先前输入标记的上下文窗口 x_1 到 x_{i-1} ，生成输出 a_i 。

在因果、从左到右的语言模型中，上下文包括任何先前的词语。也就是说，在处理 x_i 时，模型可以访问 x_i 以及上下文窗口中所有之前的标记表示（上下文窗口可以包含数千个标记），但不能访问 i 之后的标记。（相比之下，在第11章中我们将推广注意力机制，使其可以向前看未来的词语。）

图9.3展示了整个因果自注意力层中的信息流，在该层中，每个标记位置 i 上均并行进行相同的注意力计算。因此，自注意力层将输入序列 $(x_1, ..., x_n)$ 映射为长度相同的输出序列 $(a_1, ..., a_n)$ 。

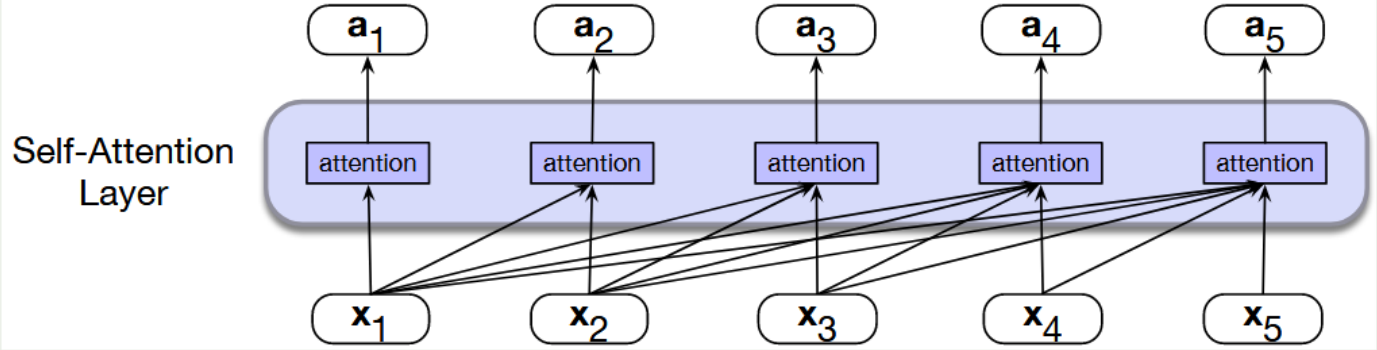

图 9.3 因果自注意力中的信息流。在处理每个输入 xi 时，模型关注所有直到并包括 xi 的输入。

简化的注意力定义

本质上，注意力实际上就是一个上下文向量的加权和，只是在计算权重和求和内容上增加了许多复杂性。为了便于理解，我们先描述一种简化的注意力，其中在标记位置 i 的注意力输出 a_i 只是所有满足 $j \leq i$ 的表示 x_j 的加权和；我们用 α_{ij} 表示 x_i 在计算 a_j 时的贡献程度:

$$\text{Simplified version:} \quad \mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j$$

每个 α_{ij} 是一个标量，用于在求和输入以计算 a_i 时为输入 x_j 赋予权重。在注意力机制中，我们根据当前标记 i 与每个先前嵌入的相似程度对其进行加权。因此，注意力的输出是先前标记嵌入的加权和，权重与这些嵌入与当前标记嵌入的相似度成比例。我们通过点积计算相似度得分，将两个向量映射为一个从 $-\infty$ 到 ∞ 的标量值。得分越大，表示对比向量越相似。然后我们使用softmax将这些得分标准化，生成权重向量 α_{ij} ，其中 $j \leq i$:

$$\begin{aligned} \textit{Simplified Version:} \quad \text{score}(\mathbf{x}_i, \mathbf{x}_j) &= \mathbf{x}_i \cdot \mathbf{x}_j \\ \alpha_{ij} &= \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \forall j \leq i \end{aligned}$$

因此，在图9.3中，我们通过计算三个得分来求出 a_3 ： $x_3 \cdot x_1$ ， $x_3 \cdot x_2$ 和 $x_3 \cdot x_3$ ，并用softmax对它们进行标准化，得到的概率作为各个标记相对于当前位置i的比例相关性权重。当然，由于 x_i 与其自身非常相似，softmax权重通常在 x_i 上最高，从而产生较高的点积。然而，其他上下文词语也可能与 i 相似，softmax会对这些词语赋予一定的权重。然后，我们在方程9.6中使用这些权重作为 α 值，计算出加权和即为 a_3 。

方程9.6至9.8中的简化注意力演示了基于注意力的 a_i 计算方法：将 x_i 与先前向量对比，将这些得分归一化为概率分布，用来加权先前向量的和。但现在我们已准备好去除这些简化条件。

单个注意力头：查询、键和值矩阵

现在我们已经了解了注意力的基本直觉，让我们介绍Transformer中实际使用的注意力头（attention head）。(在Transformer中，头通常用于指代特定的结构层)。注意力头使我们能够清晰地区分每个输入嵌入在注意力过程中扮演的三种不同角色：

- 作为当前元素，与前面的输入进行比较。我们称这个角色为查询（query）。
- 作为被比较的前置输入，以确定相似性权重。我们称这个角色为键（key）。
- 最后，作为被加权并求和的前置元素的值（value），用于计算当前元素的输出。

为了捕捉这三种不同的角色，Transformer引入了权重矩阵 W^Q 、 W^K 和 W^V 。通过这些权重，我们将每个输入向量 x_i 投影到其作为键、查询或值的表示:

$$\mathbf{q}_i = \mathbf{x}_i\mathbf{W}^{\mathbb{Q}}; \quad \mathbf{k}_i = \mathbf{x}_i\mathbf{W}^{\mathbb{K}}; \quad \mathbf{v}_i = \mathbf{x}_i\mathbf{W}^{\mathbb{V}}$$

在这些投影的基础上，当我们计算当前元素 x_i 与某个先前元素 x_j 的相似性时，我们使用当前元素的查询向量 q_i 与前置元素的键向量 k_j 的点积。此外，点积的结果可以是任意大的正值或负值，对大值进行指数运算可能会导致数值问题和训练过程中梯度的损失。为避免这种情况，我们通过将点积除以查询和键向量维度 d_k 的平方根来进行缩放，从而在简化方程9.7的基础上得到了方程9.11。接下来的softmax计算生成 α_{ij} 保持不变，但现在的 a_i 输出计算基于值向量 v 的加权和（见方程9.13）。

以下是一组最终的自注意力计算方程，用于从单个输入向量 x_i 计算单个自注意力输出向量 a_i 。该注意力版本通过对前置元素的值求和来计算 a_i ，每个值的权重由其键与当前元素的查询的相似性决定:

$$\begin{aligned}
\mathbf{q}_i = \mathbf{x}_i\mathbf{W}^{\mathbb{Q}}; \quad \mathbf{k}_j = -\mathbf{x}_j\mathbf{W}^{\mathbb{K}}; \quad \mathbf{v}_j = \mathbf{x}_j\mathbf{W}^{\mathbb{V}} \\
\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}} \\
\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \forall j \leq i \\
\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j
\end{aligned}$$

我们在图9.4中展示了计算序列中第三个输出 a_3 的情况。

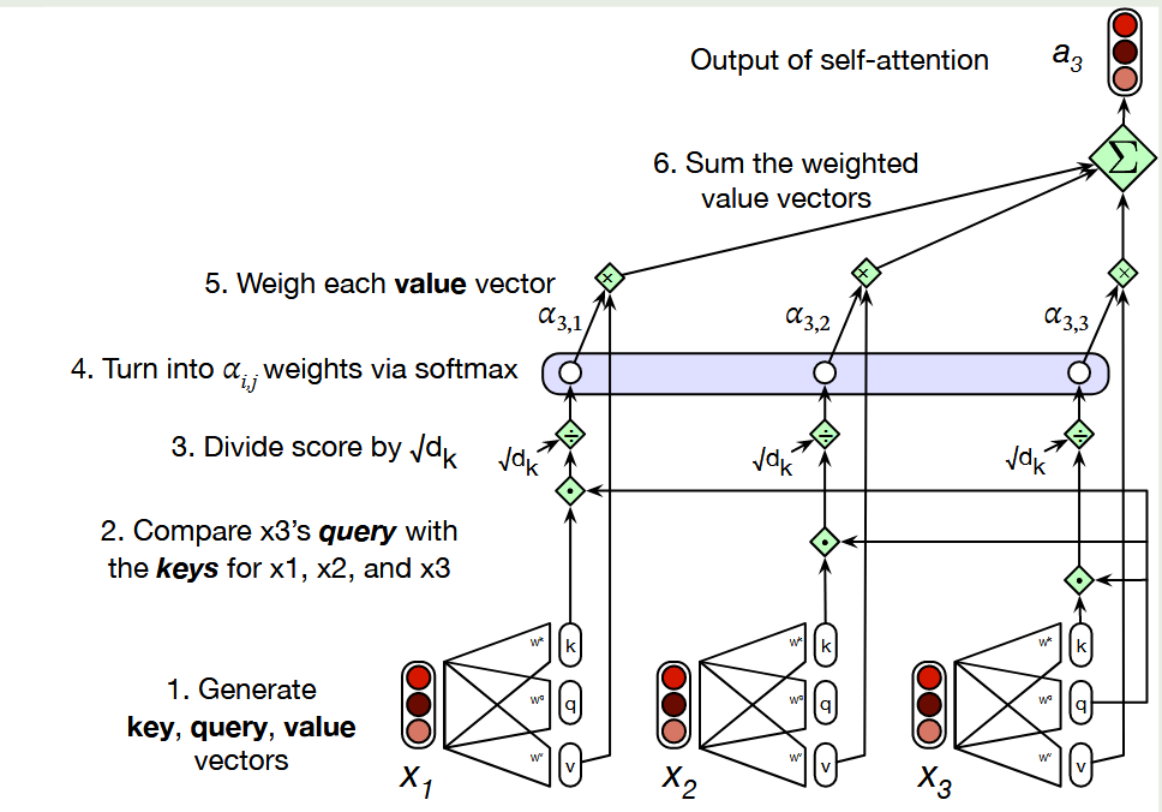

图 9.4 计算序列中第三个元素 a3 的值，使用因果（从左到右）自我注意力。

关于维度 注意力的输入 x_i 和注意力的输出 a_i 都具有相同的维度 $1 \times d$ （我们通常称 d 为模型维度，并且正如我们将在9.2节讨论的那样，每个Transformer块的输出 h_i 以及Transformer块中的中间向量也具有相同的维度 $1 \times d$ ）。键和查询向量的维度为 d_k 。查询向量和键向量的维度都是 $1 \times d_k$ ，因此

可以对它们进行点积 $q_i \cdot k_j$ 。值向量的维度为 d_v 。变换矩阵 W^Q 的形状为 $[d \times d_k]$ ， W^K 为 $[d \times d_k]$ ， W^V 为 $[d \times d_v]$ 。在原始的Transformer工作中（Vaswani等，2017年）， d 为512， d_k 和 d_v 均为64。

多头注意力

方程9.11-9.13描述了单个注意力头。实际上，Transformer使用多个注意力头。直觉上，每个头可能会基于不同的目的关注上下文：某些头可能专门用于表示上下文元素与当前标记之间的不同语言关系，或者寻找上下文中的特定模式。

在多头注意力中，我们在模型相同深度的并行层中设置了 h 个独立的注意力头，每个头有自己的一组参数，使该头能够建模输入之间关系的不同方面。因此，自注意力层中的每个头i都有自己的一组键、查询和值矩阵： W^{Ki} 、 W^{Qi} 和 W^{Vi} 。它们用于将输入分别投影为每个头的键、值和查询嵌入。

使用多个头时，模型维度 d 仍用于输入和输出，键和查询嵌入维度为 d_k ，值嵌入的维度为 d_v （在原始Transformer论文中， d_k = d_v =64， h =8， d =512）。因此，对于每个头 i ，我们有形状为 $[d \times d_k]$ 的权重层 W^{Qi} ，形状为 $[d \times d_k]$ 的 W^{Ki} ，以及形状为 $[d \times d_v]$ 的 W^{Vi} 。

下面是多头注意力的计算方程；图9.5展示了直观说明:

$$
\begin{aligned}
\mathbf{q}_i^c = \mathbf{x}_i \mathbf{W^{Qc}}; \mathbf{k}_j^c = \mathbf{x}_j \mathbf{W^{Kc}}; \mathbf{v}_j^c = \ \mathrm{x}_j \mathrm{W}^{\mathrm{Vc}}; \quad \forall\, c \quad 1 \leq c \leq h \\
\mathrm{score}^c(\mathbf{x}_i, \mathbf{x}_j) = -\frac{\mathbf{q}_i^c \cdot \mathbf{k}_j^c}{\sqrt{d_k}} \\
\alpha_{ij}^c = \ \mathrm{softmax}(\mathrm{score}^c(\mathbf{x}_i, \mathbf{x}_j)) \ \forall j \leq i \\
\mathsf{head}_i^c = \sum_{j \leq i} \alpha_{ij}^c \mathbf{v}_j^c \\
\mathbf{a}_i = (\mathsf{head}^1 \oplus \mathsf{head}^2 \ldots \oplus \mathsf{head}^h) \mathrm{W}^O \\
\mathrm{MultiHeadAttention}(\mathbf{x}_i, [\mathbf{x}_1, \cdots, \mathbf{x}_N]) = \mathbf{a}_i
\end{aligned}
$$

每个 h 个头的输出维度为 $1 \times d_v$ ，因此具有 h 个头的多头层输出由 h 个维度为 $1 \times d_v$ 的向量组成。这些向量连接在一起，生成一个维度为 $1 \times hd_v$ 的单一输出。然后我们使用另一个线性投影矩阵 $\mathbf{W}^O \in \mathbb{R}^{hd_v \times d}$ 对其重新整形，最终在每个输入 i 上得到具有正确输出形状 $[1 \times d]$ 的多头注意力向量 a_i 。

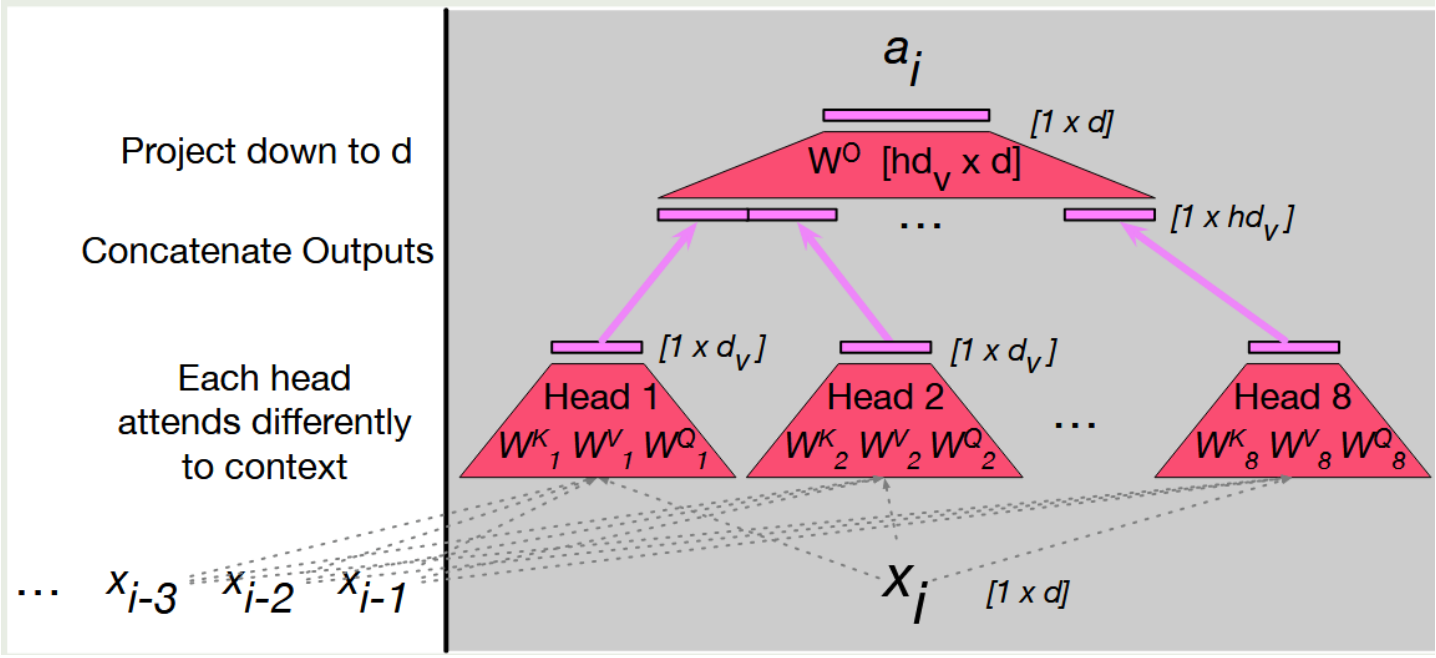

图 9.5 输入 xi 的多头注意力计算，生成输出 ai。一个多头注意力层有 h 个头，每个头都有自己的键、查询和值权重矩阵。每个头的输出被连接起来，然后投影到 d，从而生成与输入相同大小的输出。

9.2 Transformer块

自注意力计算是所谓Transformer块的核心，除了自注意力层外，还包括三种其他类型的层：(1) 前馈层，(2) 残差连接，(3) 归一化层（俗称“层归一化”）。

图9.6展示了一个Transformer块，简要说明了通常被称为残差流（Elhage等人，2021）的Transformer块的思维方式。在残差流视角中，我们将单个标记 i 在Transformer块中的处理视为标记位置 i 的一个 d 维表示流。这个残差流从最初的输入向量开始，各种组件从残差流中读取输入并将输出添加回流中。

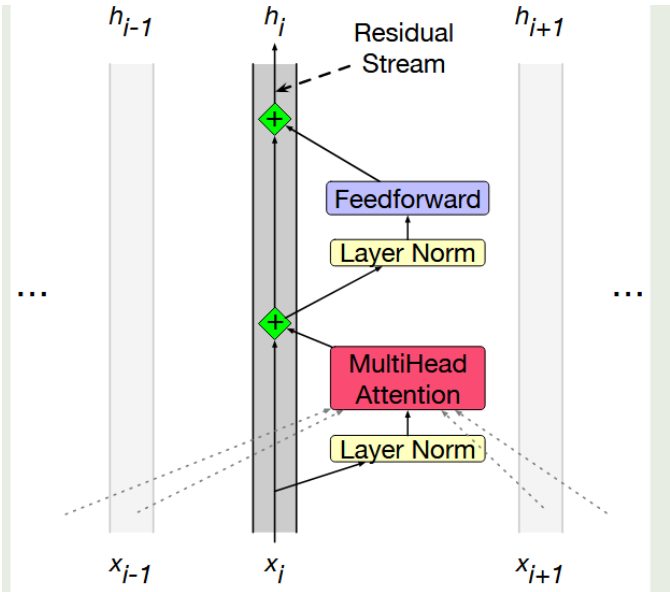

图 9.6 转换器块的架构图，显示了残差流。此图展示了架构的预规范化版本，在注意力和前馈层之前进行层规范化，而不是之后。

残差流底部的输入是一个标记的嵌入，其维度为 d 。该初始嵌入通过残差连接向上传递，并由Transformer的其他组件逐步添加，包括我们已经看到的注意力层以及接下来要介绍的前馈层。在注意力层和前馈层之前，有一个计算称为层归一化。

因此，初始向量先通过层归一化和注意力层，结果被添加回流中，通常是添加到初始输入向量 x_i 中。然后，这个求和向量再次通过另一层归一化和一个前馈层，这些输出再次添加回残差流中。我们用 h_i 来表示标记i的Transformer块最终输出。（在早期描述中，残差流常被视为残差连接的另一种隐喻，即将组件的输入添加到其输出中，而残差流是一种更清晰的方式来可视化Transformer。）

我们已经了解了注意力层，接下来让我们在处理单个输入 x_i 时介绍前馈层和层归一化计算。

前馈层

前馈层是一个全连接的两层网络，即包含一个隐藏层和两个权重矩阵（详见第七章）。每个标记位置i的权重相同，但在不同层之间不同。通常，前馈网络的隐藏层维度 d_{ff} 比模型维度 d 更大（例如在原始Transformer模型中， $d = 512$ ， $d_{ff} = 2048$ ）。

$$\text{FFN}(\mathbf{x}_i) = \text{ReLU}(\mathbf{x}_i\mathbf{W}_1 + b_1)\mathbf{W}_2 + b_2$$

层归一化

在Transformer块的两个阶段中，我们对向量进行归一化（Ba等人，2016年）。这个过程称为层归一化（layer normalization），是多种归一化形式中的一种，可以通过保持隐藏层的值在适合梯度训练的范围内来提高深度神经网络的训练性能。

层归一化是统计学中**z-score**的一种变体，应用于隐藏层中的单个向量。换句话说，“层归一化”一词有些容易混淆；层归一化不是应用于整个Transformer层，而是仅应用于单个标记的嵌入向量。因此，层归一化的输入是一个维度为 d 的单个向量，输出也是该向量的归一化形式，维度仍然是 d 。层归一化的第一步是计算要归一化向量的均值 μ 和标准差 σ 。给定一个维度为 d 的嵌入向量 x ，这些值按如下方式计算:

$$\mu = \frac{1}{d}\sum_{i=1}^{d} x_i$$

$$\sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2}$$

计算得出这些值后，通过从每个元素中减去均值并除以标准差，对向量的各个分量进行归一化。这一计算结果是一个均值为零、标准差为一的新向量。

$$\hat{\mathbf{x}} = \frac{(\mathbf{x} - \mu)}{\sigma}$$

最后，在层归一化的标准实现中，引入了两个可学习的参数 γ 和 β ，分别代表增益和偏移值。

$$\text{LayerNorm}(\mathbf{x}) = \gamma\frac{(\mathbf{x} - \mu)}{\sigma} + \beta$$

综合以上

一个Transformer块计算的函数可以通过将每个组件的计算分解成一个方程来表达，用 t （形状为 $[1 \times d]$ ）代表Transformer，并用上标表示块中的每次计算:

$$
\begin{aligned}
\mathbf{t}_i^1 &= \text{LayerNorm}(\mathbf{x}_i) \\
\mathbf{t}_i^2 &= \text{MultiHeadAttention}(\mathbf{t}_i^1, [\mathbf{x}_1^1, \cdots, \mathbf{x}_N^1]) \\
\mathbf{t}_i^3 &= \mathbf{t}_i^2 + \mathbf{x}_i \\
\mathbf{t}_i^4 &= \text{LayerNorm}(\mathbf{t}_i^3) \\
\mathbf{t}_i^5 &= \text{FFN}(\mathbf{t}_i^4) \\
\mathbf{h}_i &= \mathbf{t}_i^5 + \mathbf{t}_i^3
\end{aligned}
$$

请注意，唯一一个从其他标记（其他残差流）中获取信息的组件是多头注意力，它（如方程9.27所示）会查看上下文中的所有相邻标记。然而，注意力的输出会被添加到该标记的嵌入流中。实际上，Elhage等人（2021年）表明我们可以将注意力头视为字面上将邻近标记的残差流信息移入当前流中。因此，每个位置的高维嵌入空间包含关于当前标记及其邻近标记的信息，尽管它们存在于向量空间的不同子空间中。图9.7展示了这种信息流动的可视化。

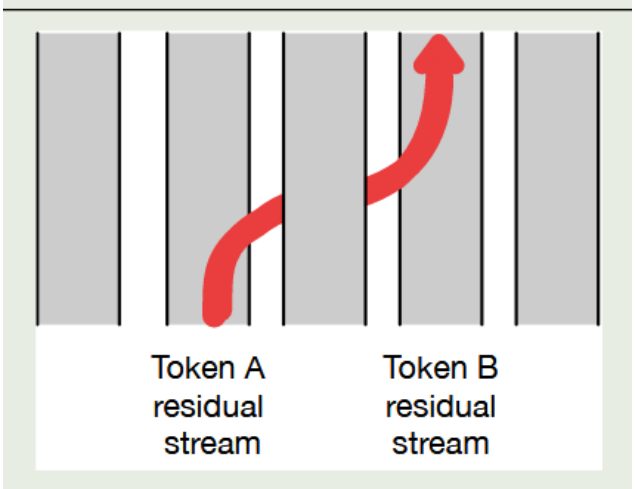

图 9.7 注意力头可以从令牌 A 的残差流中移动信息到令牌 B 的残差流中。

关键是，Transformer块的输入和输出维度是匹配的，因此它们可以堆叠。块输入的每个标记向量 x_i 具有维度 d ，输出 h_i 也具有维度 d 。大型语言模型的Transformer堆叠了多个这样的块，从12层（用于T5或GPT-3-small模型）到96层（用于GPT-3 large），甚至更多层，用于更新的模型。稍后我们将进一步探讨堆叠问题。

方程（9.27）及随后的方程仅适用于单个Transformer块，但残差流隐喻贯穿于所有Transformer层，从第一个Transformer块到12层的Transformer。在较早的Transformer块中，残差流表示当前标记。在最高的Transformer块中，残差流通常表示接下来的标记，因为最终它会被训练来预测下一个标记。

在堆叠多个块时，还有一个额外的要求：在最后（最高）Transformer块的最后，每个标记流的最后 h_i 上会有一个额外的层归一化（位于我们将很快定义的语言模型头层之下）。

9.3 使用单个矩阵 X 并行计算

关于多头注意力和Transformer块其余部分的描述，之前主要是从在单个残差流中计算单个时间步 i 的单个输出的角度出发的。但正如我们先前指出的那样，为每个标记计算的注意力计算 a_i 与其他标记的计算彼此独立，并且在Transformer块中从输入 x_i 计算 h_i 的所有计算也是独立的。这意味着我们可以轻松地并行化整个计算，利用高效的矩阵乘法算法。

为此，我们将输入序列中 N 个标记的嵌入打包到一个尺寸为 $[N \times d]$ 的矩阵X中。 X 的每一行代表一个输入标记的嵌入。大型语言模型的Transformer通常使用 N = 1K、2K，甚至多达32K个标记（或更多）的输入长度，因此X通常有1K到32K行，每行的维度为嵌入的维度 d （即模型的维度）。

并行化注意力计算

首先我们来看单个注意力头的情况，然后再介绍多头注意力以及Transformer块中的其他组件。对于单个头，我们将矩阵 X 分别与键、查询和值矩阵 W^Q （形状为 $[d \times d_k]$ ）、 W^K （形状为 $[d \times d_k]$ ）和 W^V （形状为 $[d \times d_v]$ ）相乘，生成矩阵Q（形状 $[N \times d_k]$ ）、K（形状 $[N \times d_k]$ ）和V（形状 $[N \times d_v]$ ），其中包含所有的键、查询和值向量:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{\mathbf{Q}};\ \mathrm{K} = \mathbf{X}\mathbf{W}^{\mathbf{K}};\ \mathrm{V} = \mathbf{X}\mathbf{W}^{\mathbf{V}}$$

有了这些矩阵后，我们可以通过将Q与K的转置矩阵 K^T 相乘来同时计算所有所需的查询-键比较。乘积的形状为 $[N \times N]$ ，如图9.8所示。

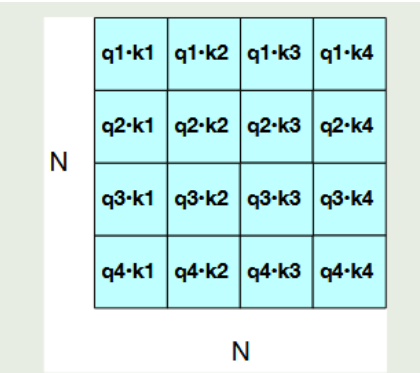

图 9.8 N × N QKT 矩阵，显示它如何在一个矩阵乘法中计算所有 qi · k j 比较。

获得 QK^T 矩阵后，我们可以非常高效地对这些得分进行缩放、取softmax，然后将结果与V相乘，得到一个形状为 $[N \times d]$ 的矩阵：即输入中每个标记的向量嵌入表示。对于单个头，我们将整个序列N个标记的自注意力步骤简化为如下计算:

$$\mathbf{A} = \text{ softmax}\left(\text{mask}\left(\frac{\mathbf{Q}\mathbf{K}^{\intercal}}{\sqrt{d_k}}\right)\right)\mathbf{v}$$

屏蔽未来的标记

您可能注意到我们在上面的方程9.32中引入了一个掩码函数。这是因为我们描述的自注意力计算有一个问题：在 QK^T 的计算中，每个查询值都获得了与每个键值的分数，包括那些在查询之后的键值。在语言建模的环境中，这不合适：如果已经知道了下一个词，预测它就非常简单了！为了解决这个问题，矩阵的上三角部分的元素被置为零（设为 $-\infty$ ），从而消除任何后续词的知识。这实际上是通过添加一个掩码矩阵 M 实现的，其中对所有 $j > i$ 的 $M_{ij} = -\infty$ （即上三角部分），否则 $M_{ij} = 0$ 。

图9.9展示了经过掩码处理的 QK^T 矩阵。（第11章将讨论如何利用未来的词来完成需要这些词的任务。）

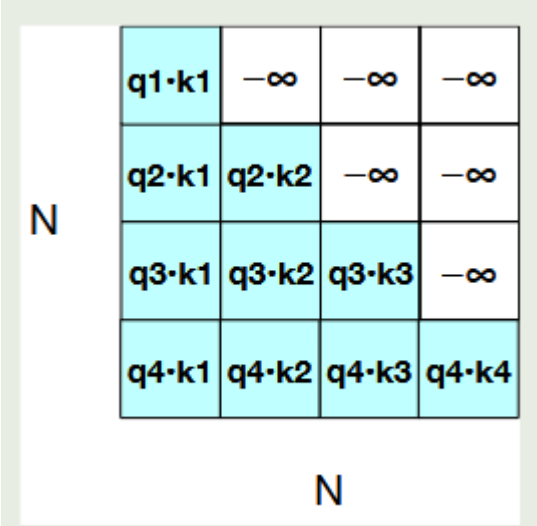

图 9.9 N × N QKT 矩阵，显示 qi · k j 值，比较矩阵的上三角部分被清零（设置为-∞，softmax 将将其转换为零）。

图9.10展示了并行化的矩阵形式中单个注意力头的所有计算过程。

图9.8和图9.9还表明，注意力的复杂度在输入长度上是平方关系的，因为在每一层都需要计算输入中每一对标记的点积。这使得对非常长的文档（如整本小说）计算注意力变得昂贵。然而，现代大型语言模型能够使用包含数千甚至数万个标记的上下文。

并行化多头注意力

在多头注意力中，与自注意力相同，输入和输出具有模型维度 d ，键和查询嵌入维度为d_k，值嵌入的维度为d_v（在原始Transformer论文中， $d_k = d_v = 64$ ， $h = 8$ ， $d = 512$ ）。因此，对于每个头 i ，我们有权重层 $\mathbf{W}_i^{\mathbf{Q}} \in \mathbb{R}^{d \times d_k}$ ， $\mathbf{W}_i^{\mathbf{Q}} \in \mathbb{R}^{d \times d_k}$ 和 $\mathbf{W}_i^{\mathbf{V}} \in \mathbb{R}^{d \times d_v}$ ，这些与打包到 X 中的输入相乘，得到 $\mathbf{Q} \in \mathbb{R}^{N \times d_k}$ 、 $\mathrm{K} \in \mathbb{R}^{N \times d_k}$ 和 $\mathbf{V} \in \mathbb{R}^{N \times d_v}$ 。每个 h 个头的输出形状为 $N \times d_v$ ，因此具有 h 个头的多头层输出由 h 个形状为 $N \times d_v$ 的矩阵组成。为了在进一步处理中使用这些矩阵，它们被连接在一起，生成一个维度为 $N \times hd_v$ 的单一输出。最后，我们使用另一个线性投影矩阵WO ∈ Rhdv×d，对其重新整形为每个标记的原始输出维度。将连接的N × hd_v矩阵输出与 $\mathbf{W}^0 \in \mathbb{R}^{hd_v \times d}$ 相乘，得到形状为 $[N \times d]$ 的自注意力输出 A 。

$$\mathbf{Q^i} = \mathbf{X}\mathbf{W}^{\mathbf{Qi}};\ \mathbf{K^i} = \ \mathrm{X}\mathrm{W}^{\mathrm{Ki}}\ ;\ \mathrm{V}^{\mathrm{i}} = \mathrm{X}\mathrm{W}^{\mathrm{Vi}}$$

$$\mathsf{head}_i = \mathsf{SelfAttention}(\mathbf{Q^i}, \mathbf{K^i}, \mathbf{V^i}) = \ \mathrm{softmax}\left(\frac{\mathbf{Q^i}\mathbf{K^{iT}}}{\sqrt{d_k}}\right)\mathbf{V^i}$$

$$\mathrm{MultiHeadAttention}(\mathrm{X}) = (\mathsf{head}_1 \oplus \mathsf{head}_2 \ldots \oplus \mathsf{head}_h)\mathrm{W}^0$$

将所有组件与并行输入矩阵X整合

整个 N 个输入标记的 N 层Transformer块并行计算的函数可以表示如下:

$$\begin{aligned} \mathrm{O} &= \mathrm{LayerNorm}(\mathbf{X} + \mathrm{MultiHeadAttention}(\mathbf{X})) \\ \mathrm{H} &= \mathrm{LayerNorm}(\mathbf{O} + \mathrm{FFN}(\mathbf{O})) \end{aligned}$$

或者，我们可以将每个组件的计算分解成一个方程，使用 T （形状为 $[N \times d]$ ）代表Transformer，并用上标标示块中的每个计算:

$$
\begin{aligned}
\mathbf{T^1} &= \text{MultiHeadAttention}(\mathbf{X}) \\
\mathbf{T^2} &= \mathbf{X} + \mathbf{T^1} \\
\mathbf{T^3} &= \text{LayerNorm}(\mathbf{T^2}) \\
\mathbf{T^4} &= \text{FFN}(\mathbf{T^3}) \\
\mathbf{T^5} &= \mathbf{T^4} + \mathbf{T^3} \\
\mathbf{H} &= \text{LayerNorm}(\mathbf{T^5})
\end{aligned}
$$

在这里，当我们使用类似 $\text{FFN}(\mathbf{T}^3)$ 的符号时，意味着同一个FFN被并行应用到窗口中的每个 N 个嵌入向量。同样，每个 N 个标记在层归一化中并行归一化。关键是，Transformer块的输入和输出维度是匹配的，因此它们可以堆叠。由于块输入的每个标记 x_i 具有维度 d ，这意味着输入 X 和输出 H 的形状都是 $[N \times d]$ 。

9.4 输入：标记和位置的嵌入

让我们讨论输入矩阵 X 的来源。给定一个包含 N 个标记的序列（N 为上下文长度），形状为 $[N \times d]$ 的矩阵 X 包含上下文中每个词的嵌入。Transformer通过分别计算两个嵌入来实现这一点：输入标记嵌入和输入位置嵌入。

标记嵌入 是一个维度为 d 的向量，作为输入标记的初始表示（第7章和第8章中已介绍）。在Transformer层中通过残差流传递向量时，这种嵌入表示会随着上下文的增加而变化，并在构建不同类型的语言模型时发挥不同的作用。这些初始嵌入存储在嵌入矩阵 E 中，每个词汇中的标记 $|V|$ 占据一行。因此，每个单词都是一个 d 维行向量， E 的形状为 $[|V| \times d]$ 。

给定一个输入标记字符串如 "Thanks for all the"，我们首先将这些标记转换为词汇索引（使用BPE或SentencePiece在首次进行标记化时创建的）。例如，"thanks for all the" 的表示可能是 $w = [5, 4000, 10532, 2224]$ 。然后，我们通过索引选择 E 中的相应行（第5行、第4000行、第10532行、第2224行）。

选择嵌入矩阵中的标记嵌入的另一种方式是将标记表示为形状为 $[1 \times |V|]$ 的单热向量（one-hot vector），即词汇中的每个词对应一个维度。回忆一下，在独热向量中，除一个元素外，所有元素都为0，该元素所在的维度是词汇中该单词的索引，其值为1。例如，如果词汇中的"thanks"的索引为5，那么 $x_5 = 1$ ，而对 $i \neq 5$ 的所有 $x_i = 0$ ，如下所示：

$$
\begin{array}{cccccccccccc}
[0 & 0 & 0 & 0 & 1 & 0 & 0 & \dots & 0 & 0 & 0 & 0] \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & \dots & \dots & \dots & \dots & |\text{V}|
\end{array}
$$

乘以一个仅有一个非零元素 $x_i = 1$ 的单热向量实际上是选择出单词 i 的相关行向量，从而得到单词 i 的嵌入，如图9.11所示。

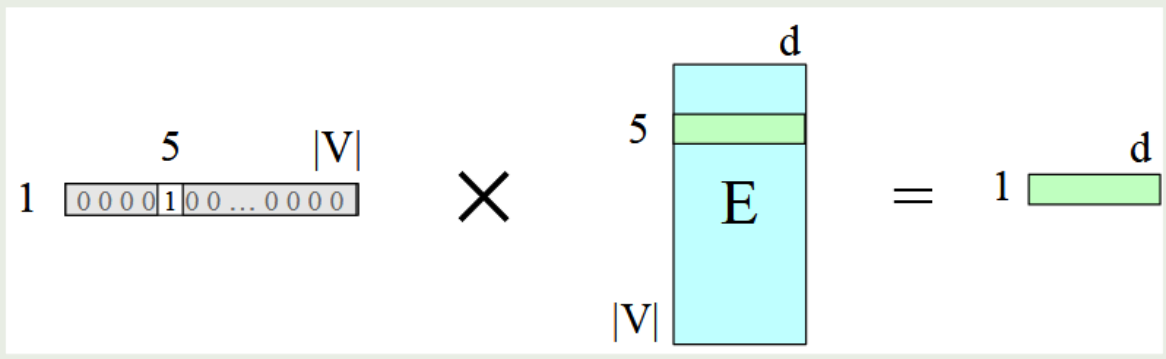

图 9.11 选择单词 V5 的嵌入向量，通过将嵌入矩阵 E 与索引为 5 的单热向量相乘来实现。

我们可以将这个想法扩展，将整个标记序列表示为一个矩阵，该矩阵包含 transformer 上下文窗口中的 N 个位置的独热向量，如图 9.12 所示。

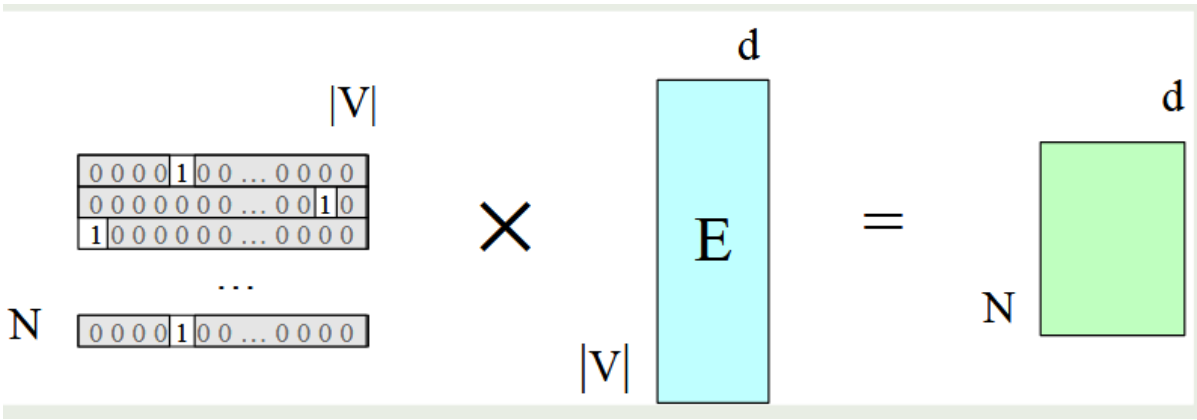

图 9.12 通过将对应于 W 的一热矩阵与嵌入矩阵 E 相乘，选择用于标记 ID 序列 W 的嵌入矩阵。

这些标记嵌入与位置无关。为了表示序列中每个标记的位置，我们将这些标记嵌入与序列中特定位置的**位置嵌入**结合。

这些位置嵌入来自哪里？最简单的方法称为**绝对位置**，它从随机初始化的嵌入开始，嵌入对应于到某一最大长度的每个可能输入位置。例如，就像我们为单词 "fish" 设置一个嵌入一样，我们也可以为位置3设置一个嵌入。与词嵌入一样，这些位置嵌入在训练过程中与其他参数一起学习。我们可以将它们存储在形状为 $[1 \times N]$ 的矩阵 E_{pos} 中。

为了生成捕获位置信息的输入嵌入，我们只需将每个输入词的嵌入与其相应位置嵌入相加。每个标记和位置嵌入的大小均为 $[1 \times d]$，因此它们的和也为 $[1 \times d]$。这一新嵌入作为进一步处理的输入。图9.13展示了这一概念。

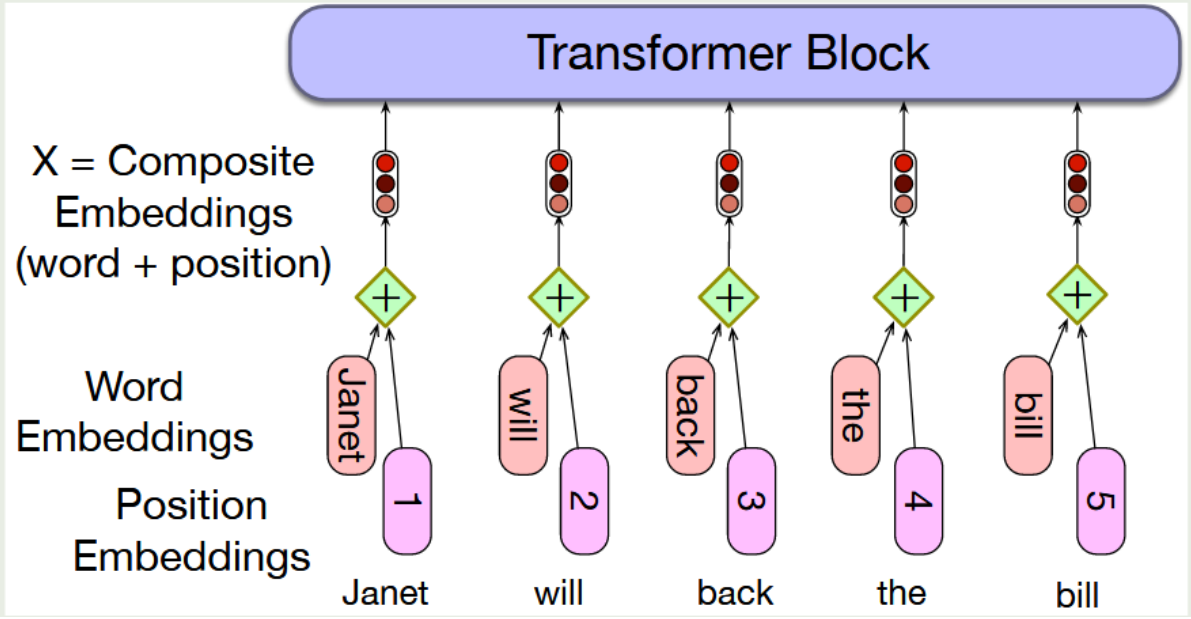

图 9.13 一个简单的方式来建模位置：将绝对位置的嵌入添加到标记嵌入中，以产生相同维度的新的嵌入。

输入的最终表示，矩阵 X，是一个 $[N \times d]$ 矩阵，其中每行 i 是输入中第 i 个标记的表示，通过将 $E[id(i)]$（即出现在位置 i 的标记的 id 的嵌入）与 $P[i]$（即位置 i 的位置嵌入）相加得到。

这种绝对位置嵌入方法的一个潜在问题是，在输入的初始位置会有大量的训练样本，而在序列的最大长度位置会相应较少。因此，这些后者嵌入可能训练不充分，并且在测试期间泛化能力不佳。绝对位置嵌入的替代方法是选择一个静态函数，将整数输入映射为实值向量，以捕捉位置间的内在关系。也就是说，它能体现输入中位置4比位置17更接近位置5的事实。原始的Transformer工作中使用了具有不同频率的正弦和余弦函数的组合。还有更复杂的位置嵌入方法，比如相对位置嵌入，这种方法表示相对位置而非绝对位置，通常在每一层的注意力机制中实现，而不是在初始输入时添加。

9.5 语言建模头(The Language Modeling Head)

Transformer的最后一个组件是**语言建模头**。在这里，我们使用“头”一词来指我们在基础Transformer架构上添加的神经网络部分，当我们将预训练的Transformer模型应用于各种任务时需要使用这一组件。语言建模头是我们进行语言建模所需的电路。

回忆一下，从第3章的简单n-gram模型到第7章和第8章的前馈和RNN语言模型，语言模型都是词预测器。给定词语的上下文，它们会为每个可能的下一个词分配一个概率。例如，如果前面的上下文是“Thanks for all the”，我们希望知道下一个词是“fish”的可能性:

$$P(fish|Thanksforallthe)$$

语言模型能够对每一个可能的下一个词分配条件概率，提供整个词汇表上的分布。第3章中的n-gram语言模型通过计算一个词与其前n − 1个词一起出现的次数来确定该词的概率。上下文大小为n − 1。而对于Transformer语言模型，其上下文窗口的大小可以非常大：对于非常大的模型，可以达到2K、4K，甚至32K个标记。

语言建模头的任务 是从最后一个标记 N 的最终Transformer层的输出中预测位置 $N+1$ 的下一个词。图9.14展示了如何完成该任务，即获取最后一层中最后一个标记的输出（形状为 $[1 \times d]$ 的 d 维输出嵌入），并生成词语上的概率分布（我们将从中选择一个词进行生成）：

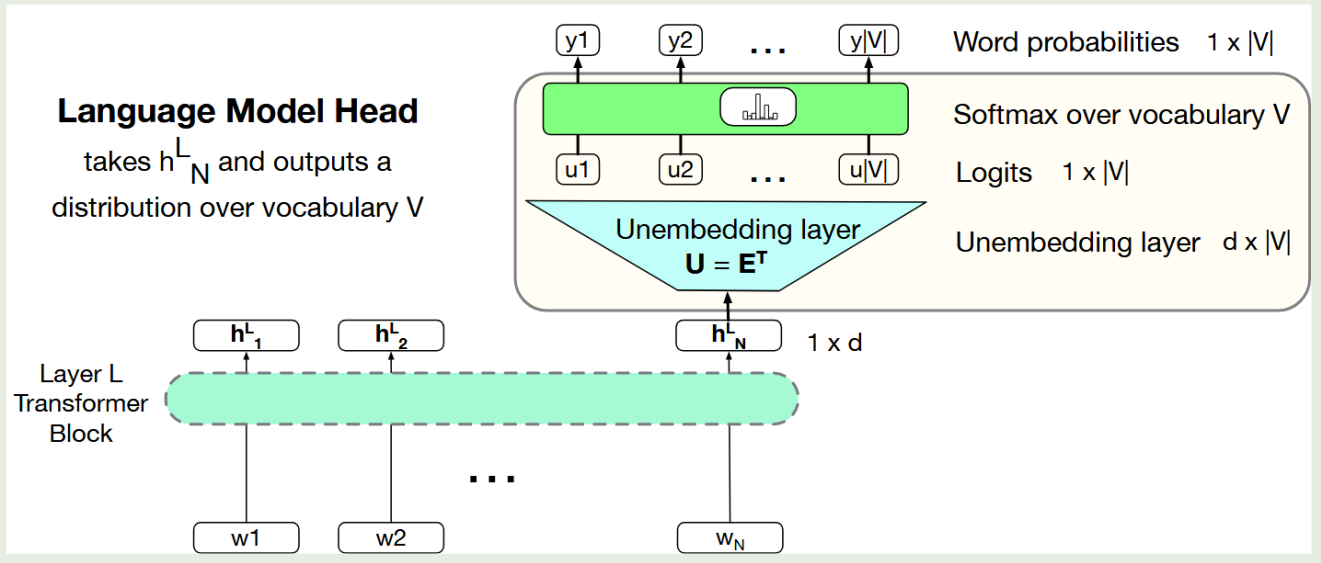

图 9.14 语言模型头部：Transformer顶部的电路，将最后一个Transformer层（hL N）的标记 N 的输出嵌入映射到词汇表 V 中的单词的概率分布。

图9.14中的第一个模块是线性层，它的任务是从表示最后一个块 L 中位置 N 的输出标记嵌入 h_N^L（形状为 $[1 \times d]$）投影到对词汇表中每个可能词的单一分数的logit向量（或得分向量）。因此，logit向量 u 的维度为 $[1 \times |V|]$。

这个线性层可以学习，但通常我们将该矩阵绑定到嵌入矩阵 E 的转置。回忆一下，在权重绑定中，我们在模型中的两个不同矩阵上使用相同的权重。因此，在Transformer的输入阶段，嵌入矩阵（形状 $[|V| \times d]$）用于从词汇表上的单热向量（形状 $[1 \times |V|]$）映射到嵌入（形状 $[1 \times d]$）。在语言模型头中，嵌入矩阵的转置 E^T（形状 $[d \times |V|]$）用于从嵌入（形状 $[1 \times d]$）映射回词汇表上的向量（形状 $[1 \times |V|]$）。在学习过程中，E 会被优化，以便在这两种映射中表现良好。因此，**我们有时将转置 E^T 称为“解嵌入层”（unembedding layer）**，因为它执行这种反向映射。

一个softmax层将logits u 转换为词汇表上的概率分布 y。

$$\mathbf{u} = \mathrm{h}_{\mathrm{N}}^{\mathrm{L}}\,\mathrm{E}^{\mathrm{T}}$$
$$\mathbf{y} = \mathrm{softmax}(\mathbf{u})$$

我们可以使用这些概率分布来给定文本分配概率。最重要的用途是生成文本，这可以通过从这些概率 y 中采样一个词来实现。我们可以采样最高概率的词（即“贪心”解码），或使用其他采样方法（将在第??节介绍）。无论使用何种方法，从概率向量 y 中选择的条目 y_k 对应的词就是我们生成的词，该词的索引为 k。

图9.15展示了单个标记 i 的完整堆叠架构。请注意，每一层Transformer的输入 x_i 与前一层的输出 h_i^- 相同。

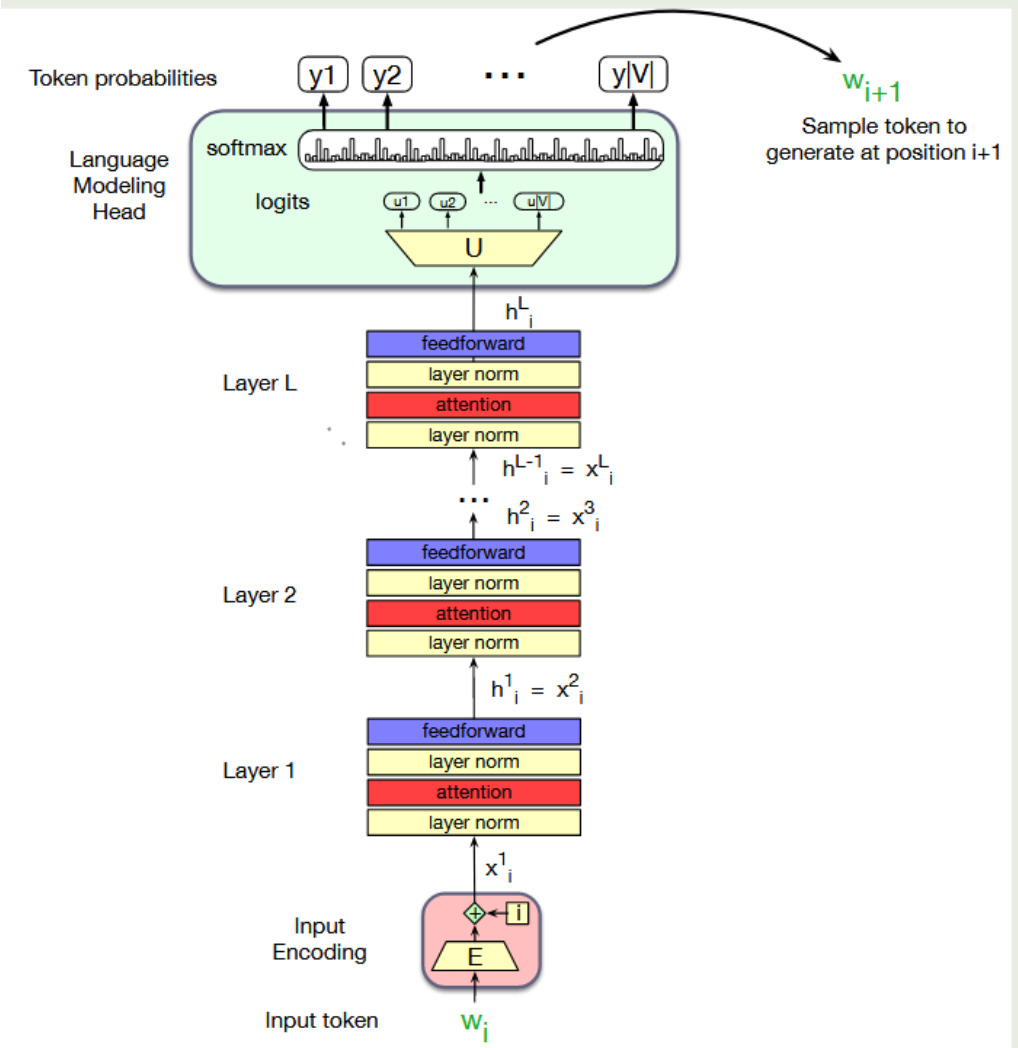

图 9.15 转换器语言模型（仅解码器），堆叠转换器块并将输入令牌 wi 映射到预测的下一个令牌 wi+1。

现在看到所有这些Transformer层展开在页面上，我们可以指出解嵌入层的另一个有用特性：作为一个称为**logit lens**的工具，用于解释Transformer内部（Nostalgebraist, 2020）。我们可以从Transformer的任何层中获取一个向量，假设它是预输出嵌入，仅通过与解嵌入层相乘即可获得logits，并计算softmax以查看该向量可能表示的词分布。这可以作为模型内部表示的有用窗口。由于网络没有被训练为以这种方式操作内部表示，因此logit lens并不总是完美工作，但它仍然是一个有用的技巧。

在结束之前，有一个术语注释：您有时会看到用于这种单向因果语言模型的Transformer被称为“仅解码器模型”（decoder-only model）。这是因为该模型大致构成了我们将在第13章中用于机器翻译的Transformer的编码器-解码器模型的一半。（令人困惑的是，Transformer的最初引入采用的是编码器-解码器架构，直到后来，因果语言模型的标准范式才是使用该原始架构的仅解码器部分来定义的）。

9.6 总结

本章介绍了Transformer及其用于语言建模的各个组件。我们将在下一章继续讨论语言建模任务，包括训练和采样等问题。

以下是本章主要内容的总结：

- **Transformer是一种基于多头注意力的非递归网络**，即一种自注意力。多头注意力计算接收输入向量 x_i 并通过加入来自前置标记的向量映射到输出 a_i，这些向量根据它们对于当前词处理的相关性进行加权。
- **Transformer块由残差流组成**，其中前一层的输入传递到下一层，并与不同组件的输出相加。这些组件包括多头注意力层和后续的前馈层，每个层之前都有层归一化。通过堆叠多个Transformer块，可以构建更深、更强大的网络。
- **Transformer的输入通过将嵌入（使用嵌入矩阵计算）与位置编码相加来生成**，位置编码表示标记在窗口中的顺序位置。
- **语言模型可以由堆叠的Transformer块构成，顶部添加语言模型头部**，该头部将顶层的输出 H 应用解嵌入矩阵以生成logits，然后通过softmax生成词概率。
- **基于Transformer的语言模型具有广泛的上下文窗口**（对于非常大的模型，窗口可达32768个标记），使它们能够利用大量上下文来预测即将出现的词语。