

第九讲 - Transformer

张建章

杭州师范大学

✉ jianzhang.zhang@foxmail.com

2025-02-01



1 Transformer架构概述

2 Attention

3 Transformer Blocks

4 矩阵并行计算

5 Transformer的输入：token嵌入+位置嵌入

6 语言模型建模

目录

- 1 Transformer架构概述
- 2 Attention
- 3 Transformer Blocks
- 4 矩阵并行计算
- 5 Transformer的输入：token嵌入+位置嵌入
- 6 语言模型建模

Transformer的组成

Transformer是一种基于多头自注意力机制的神经网络，广泛应用于大型语言模型的构建。Transformer通过并行处理输入序列，克服了传统RNN和LSTM模型的序列依赖问题，特别适用于处理长序列。

Transformer模型的核心由三个主要部分组成：

- 多头自注意力层 (**Multi-head Attention**)：负责捕捉输入序列中词与词之间的关系；
- 前馈神经网络层 (**Feedforward Networks**)：对每个位置的表示进行进一步处理；
- 层归一化 (**Layer Normalization**)：保证模型训练的稳定性。

1. Transformer架构概述

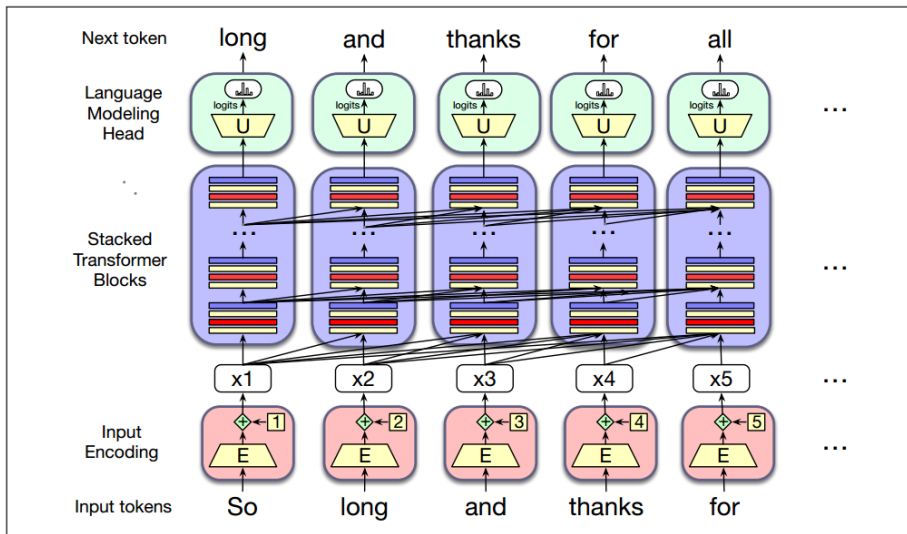


Figure 9.1 The architecture of a (left-to-right) transformer, showing how each input token get encoded, passed through a set of stacked transformer blocks, and then a language model head that predicts the next token.

2. Transformer的工作流程

- **输入与嵌入表示：**输入序列通过嵌入矩阵转换为向量表示，并且每个词的位置也会被编码。位置编码（Positional Encoding）用于处理输入序列中词语的顺序信息。

- **处理过程：**每个输入词向量通过一系列堆叠的Transformer块进行处理，每个块内含有多头自注意力层和前馈神经网络。最终，Transformer的输出会传递给语言建模头，用于生成下一个词的概率分布。

3. Transformer的优势与应用

- **非递归结构：**与传统的递归神经网络不同，Transformer不依赖于前一时刻的输出，因此可以并行计算，大大提高了训练效率。

- **强大的上下文建模能力：**通过多头自注意力机制，Transformer能够在不同层次上捕捉输入词之间的复杂依赖关系，特别适用于长文本的理解与生成任务。

目录

- 1 Transformer架构概述
- 2 Attention**
- 3 Transformer Blocks
- 4 矩阵并行计算
- 5 Transformer的输入：token嵌入+位置嵌入
- 6 语言模型建模

1. 背景：静态嵌入和上下文依赖

- 在早期的模型中，如word2vec等静态词嵌入，词的表示是固定的，不会随上下文的变化而变化。例如，词“chicken”的表示总是相同的。然而，在不同的上下文中，词的意义是动态变化的。举例来说，词“it”在不同句子中可以指代不同的对象，具体取决于上下文的内容。例如：

- 在句子“The chicken didn’t cross the road because it was too tired.”中，“it”指代的是“chicken”；

- 在句子“The chicken didn’t cross the road because it was too wide.”中，“it”指代的是“road”。

这些例子说明，词语的意义依赖于上下文，需要能够根据句子结构进行动态调整。

Transformer模型通过自注意力机制生成词语的上下文嵌入(**contextual embeddings**)。上下文嵌入是动态计算的，词语的表示不仅仅由词本身决定，还受其周围词语的影响，这使得模型能够更好地理解同一词在不同句子中的不同含义。

2. 注意力机制的引入

- **Transformer**中的注意力机制：Transformer使用自注意力机制来处理词语的上下文关系。通过这种机制，模型能够通过关注和整合上下文中的其他词语信息来构建词语的上下文化表示。

- **自注意力的核心思想**：对于每个词，Transformer模型会在不同的层次上计算该词与上下文中其他词的相关性，并据此调整词语的表示，使得词语的表示能够更加丰富和准确。

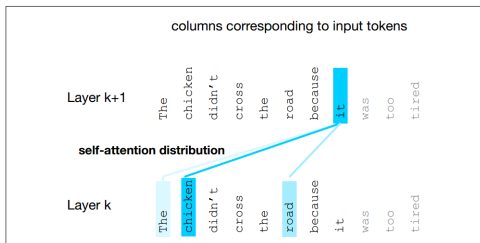


Figure 9.2 The self-attention weight distribution α that is part of the computation of the representation for the word *it* at layer $k + 1$. In computing the representation for *it*, we attend differently to the various words at layer k , with darker shades indicating higher self-attention values. Note that the transformer is attending highly to the columns corresponding to the tokens *chicken* and *road*, a sensible result, since at the point where *it* occurs, it could plausibly corefer with the chicken or the road, and hence we'd like the representation for *it* to draw on the representation for these earlier words. Figure adapted from [Uszkoreit \(2017\)](#).

3. 简化版的注意力计算

Attention 机制旨在为序列中每个位置 i 生成一个新的表示向量 a_i ，该向量通过对该位置与所有先前位置 $j \leq i$ 的表示 x_j 的加权求和得到：

$$a_i = \sum_{j \leq i} \alpha_{ij} x_j$$

其中权重 α_{ij} 反映了位置 j 对位置 i 的“相关性”或“注意力”强度，所有 α_{ij} 通过 softmax 归一化，确保其和为 1。这一机制允许模型根据上下文远程地集成不同位置的信息，以构建更丰富的上下文表示。

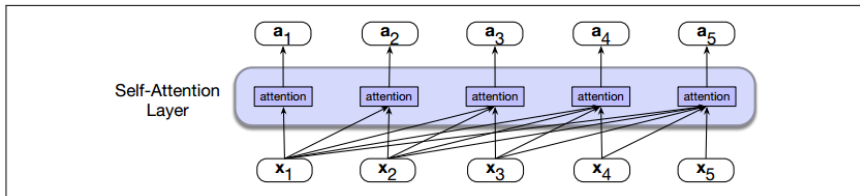


Figure 9.3 Information flow in causal self-attention. When processing each input x_i , the model attends to all the inputs up to, and including x_i .

① 相似度得分：最简单的做法是直接采用点积

$$\text{score}(x_i, x_j) = x_i \cdot x_j$$

② 归一化权重：对所有 $j \leq i$ 的得分做 softmax

$$\alpha_{ij} = \text{softmax}(\text{score}(x_i, x_j))$$

③ 加权求和：按权重对上下文向量求和，得到 a_i

$$a_i = \sum_{j \leq i} \alpha_{ij} x_j$$

4. 缩放点积注意力头 (Scaled dot-product attention head)

为了引入不同角色并控制数值稳定性，正式的 Attention 头 (Attention Head) 引入了三组投影矩阵 W^Q, W^K, W^V ，将输入 x_i 分别映射为查询 (query)、键 (key)、值 (value) 向量：

$$q_i = x_i W^Q, \quad k_j = x_j W^K, \quad v_j = x_j W^V$$

Transformer 通过将每个输入向量 x_i 投影为三类不同的子向量¹，分别扮演 Query、Key 和 Value 三种角色，其含义如下：

- Query (查询) 指当前正在计算注意力的元素所对应的向量，作为“提问”端，用来与所有先前输入进行相似度比较，决定关注哪些位置；
- Key (键) 指上下文中每个先前元素所对应的向量，作为“索引”端，与 Query 向量做点积来计算相似度得分，进而生成注意力权重；
- Value (值) 指每个先前元素所携带的信息向量，按照注意力权重加权后求和，生成当前元素的最终输出表示。

¹类似于Word2Vec模型为每个词语学习两种表示，即，target word representation和context word representation。

① 缩放相似度:

$$\text{score}(i, j) = \frac{q_i \cdot k_j}{\sqrt{d_k}}$$

通过除以 $\sqrt{d_k}$ (query/key向量的维度)防止点积值过大导致梯度消失或数值不稳定。

② 计算注意力权重:

$$\alpha_{ij} = \text{softmax}(\text{score}(i, j))$$

③ 生成头输出:

$$\text{head}_i = \sum_{j \leq i} \alpha_{ij} v_j, \quad a_i = \text{head}_i W^O$$

其中, W^O 用于将 Attention 头的输出映射回原始模型维度。

上述过程即为单一注意力头 (A single attention head)计算过程。

2. Attention

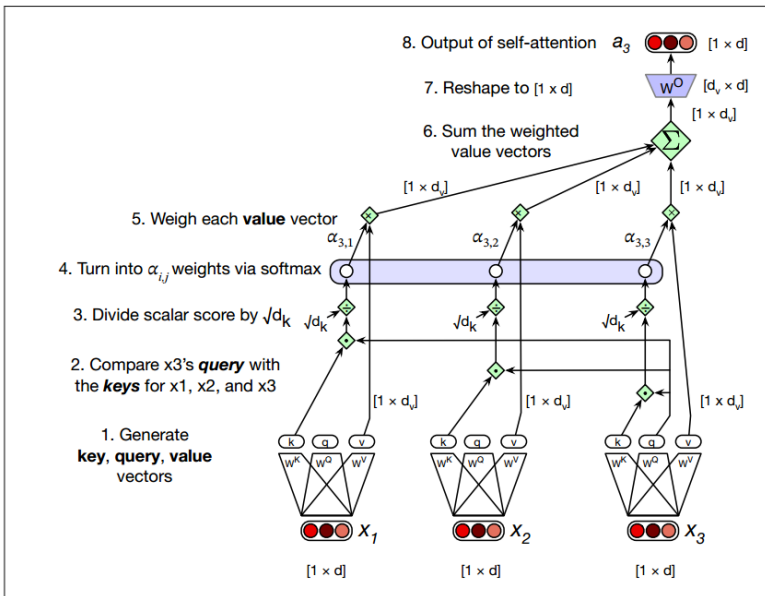


Figure 9.4 Calculating the value of a_3 , the third element of a sequence using causal (left-to-right) self-attention.

4. 多头注意力机制 (Multi-head Attention)

单个 Attention 头可能只捕捉一种关系模式，为增强表达能力，Transformer 并行使用 A 个头 (multi-head):

$$q_i^c = x_i W_Q^c, \quad k_j^c = x_j W_K^c, \quad v_j^c = x_j W_V^c \quad (1 \leq c \leq A)$$

$$\alpha_{ij}^c = \text{softmax}\left(\frac{q_i^c \cdot k_j^c}{\sqrt{d_k}}\right), \quad \text{head}_i^c = \sum_{j \leq i} \alpha_{ij}^c v_j^c$$

$$a_i = [\text{head}_i^1 \oplus \text{head}_i^2 \oplus \cdots \oplus \text{head}_i^A] W^O$$

各头在不同子空间独立学习注意力分布，最后拼接 (首尾相连，concatenation) 并线性变换得到最终输出 a_i ，可以等价地看作每个头的输出先分别做线性映射 (线性投影)，再相加。

通过上述机制，Transformer 在每个位置都能动态地从先前所有位置中选择性地“注意”并聚合信息，构建更强大的上下文表示，这正是其在大规模语言模型中取得突破性效果的核心所在。

2. Attention

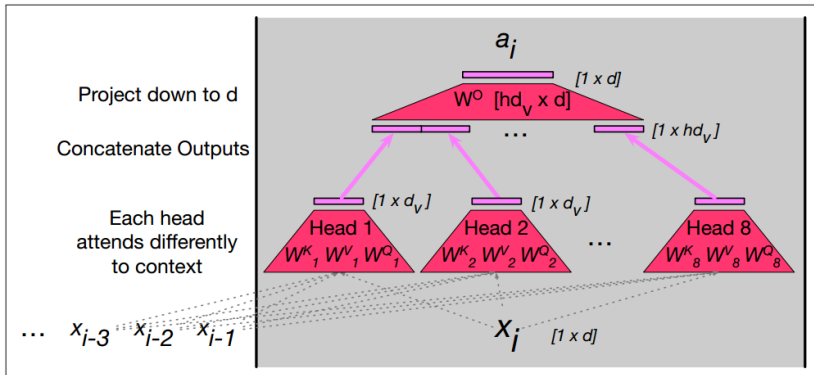


Figure 9.5 The multi-head attention computation for input x_i , producing output a_i . A multi-head attention layer has A heads, each with its own key, query and value weight matrices. The outputs from each of the heads are concatenated and then projected down to d , thus producing an output of the same size as the input.

目录

- 1 Transformer架构概述
- 2 Attention
- 3 Transformer Blocks**
- 4 矩阵并行计算
- 5 Transformer的输入：token嵌入+位置嵌入
- 6 语言模型建模

整体结构与残差流视图

Transformer Block 是 Transformer 架构的基本计算单元，其功能是将输入向量序列通过自注意力、多层感知机、残差连接和层归一化等操作，映射为同维度的输出向量序列。每一层的Transformer block的参数是共享的，不同层的Transformer block的参数不共享，参见第5页中的Transformer总体架构图。

残差流（Residual Stream）：对每个位置 i 的输入向量 x_i ，Transformer Block 维护一条“残差流”，依次将各组件的输出累加回流，保证信息在深层网络中顺畅传递。主要组件：

- ① 层归一化（LayerNorm）
- ② 多头自注意力（Multi-Head Attention）
- ③ 前馈网络（Feed-Forward Network, FFN）
- ④ 残差连接（Residual Connection）

多头自注意力是一个Transformer Block中的核心组件。

3. Transformer Blocks

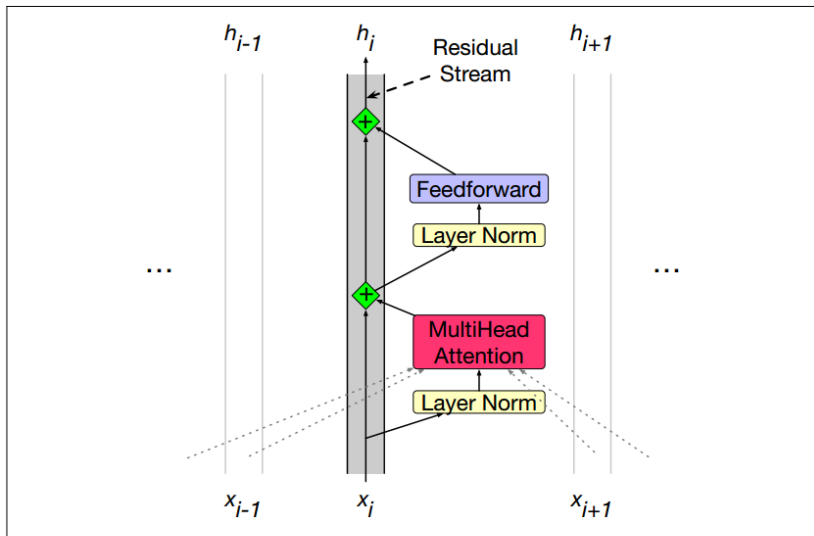


Figure 9.6 The architecture of a transformer block showing the residual stream. This figure shows the **prenorm** version of the architecture, in which the layer norms happen before the attention and feedforward layers rather than after.

各组件详解

1. 层归一化 (LayerNorm)

对单个向量 $x \in \mathbb{R}^d$ 做标准化, 旨在提升深度神经网络的训练性能:

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

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

其中 γ, β 为可学参数, 用以恢复表达能力。

2. 多头自注意力 (Multi-Head Attention)

输入先经层归一化得到 t_i^1 , 再与同层所有位置并行计算多头注意力:

$$t_i^2 = \text{MultiHeadAttention}(t_i^1, [t_1^1, \dots, t_N^1])$$

四个组件中唯一进行跨位置交互操作的组件, 用以聚合上下文信息。

3. 前馈网络 (FFN)

t_i^2 经过一次残差连接得到 t_i^3 ，在经过一次层归一化得到 t_i^4 ，逐位置、同参数地应用两层全连接与激活：

$$\text{FFN}(t_i^4) = (\text{ReLU}(t_i^4 W_1 + b_1)) W_2 + b_2$$

通常隐层维度 d_{ff} 大于模型维度 d ，以增强表达能力，每个位置 i 的权重 W_1 和 W_2 是相同的，但在不同的 Transformer block 之间权重各不相同。

4. 残差连接 (Residual Connection)

在注意力和前馈子层之外，均使用残差将子层输入与输出相加：

$$\begin{aligned} t_i^3 &= t_i^2 + x_i, \\ t_i^4 &= \text{LayerNorm}(t_i^3), \\ t_i^5 &= \text{FFN}(t_i^4), \\ h_i &= t_i^5 + t_i^3 \end{aligned}$$

最终输出 h_i 与输入 x_i 的维度均为 $[1 \times d]$ ，实现了模块的可堆叠性。

x_i 经过一个Transformer block后得到 h_i 的完整计算过程如下:

$$t_i^1 = \text{LayerNorm}(x_i),$$

$$t_i^2 = \text{MultiHeadAttention}(t_i^1, [t_1^1, \dots, t_N^1]),$$

$$t_i^3 = t_i^2 + x_i,$$

$$t_i^4 = \text{LayerNorm}(t_i^3),$$

$$t_i^5 = \text{FFN}(t_i^4),$$

$$h_i = t_i^5 + t_i^3$$

模块组合与可堆叠性

Prenorm 结构: 先归一化, 后计算子层, 较 Postnorm (先计算子层, 后归一化)更易训练;

输入输出匹配: 每一块输入输出维度均为 $[N \times d]$ (序列长度 \times 词嵌入维度), 可沿深度方向自由叠加, 形成从 12 层 (GPT-3 small)到数十层的深度网络;

信息流向:

1. $\underbrace{x_i}_{\text{输入}} \xrightarrow{\text{LayerNorm}} \underbrace{t_i^1}_{\text{Normed}}$
2. $\rightarrow \text{Attention} \rightarrow t_i^2 \rightarrow \text{Add}(x_i) \rightarrow t_i^3$
3. $\rightarrow \text{LayerNorm} \rightarrow t_i^4 \rightarrow \text{FFN} \rightarrow t_i^5 \rightarrow \text{Add}(t_i^3) \rightarrow h_i$

这一设计使模型既能保留局部输入信息, 又能灵活吸收全局上下文, 为后续层提供更丰富、深度的文本表示。

目录

- 1 Transformer架构概述
- 2 Attention
- 3 Transformer Blocks
- 4 矩阵并行计算**
- 5 Transformer的输入：token嵌入+位置嵌入
- 6 语言模型建模

将输入打包为矩阵 X

Transformer 中对每个位置 i 的自注意力 (self-attention) 计算彼此独立, 因此可通过矩阵运算将整条序列的注意力与后续子层计算一次性并行地完成, 极大提升计算效率。上一小节介绍的是一个词语 x_i 在一个 Transformer block 中的计算过程, 这一小节介绍一个包含 N 个词语的序列在 N 个 Transformer blocks 中的计算过程, 通过矩阵计算一次并行完成。

设序列长度为 N 、模型维度为 d (输入和输出的词嵌入维度), 将所有输入向量 x_1, \dots, x_N 按行堆叠, 构成矩阵

$$X \in \mathbb{R}^{N \times d}$$

每一行即对应位置 i 的输入表示 x_i (词嵌入向量)。

单头注意力的并行化

1. 生成 Q, K, V 矩阵

将 X 中的每个行向量与权重矩阵 W 相乘，得到每个 x_i 对应的query向量、key向量、value向量，按行堆叠后得到相应的矩阵表示：

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V,$$

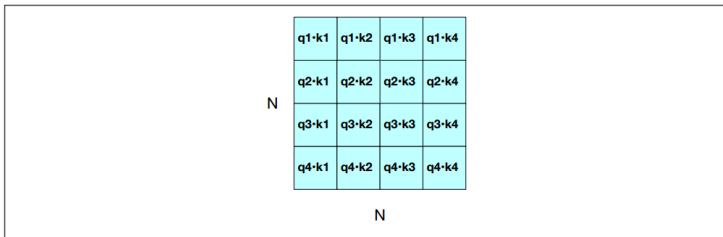
其中 $W^Q, W^K \in \mathbb{R}^{d \times d_k}$ 、 $W^V \in \mathbb{R}^{d \times d_v}$ ， $Q, K \in \mathbb{R}^{N \times d_k}$ ， $V \in \mathbb{R}^{N \times d_v}$ 。

2. 计算相似度矩阵并掩码

$$S = \frac{QK^T}{\sqrt{d_k}} \in \mathbb{R}^{N \times N},$$

矩阵乘法 QK^T 中，每个元素均为 Q 的行向量与 K 的列向量的点积，第 m 行的元素表示，第 m 个词语与其他全部词语（包括它自己在内）的点积相似度。

4. 矩阵并行计算



The diagram shows a large square representing an $N \times N$ matrix. Inside this square, a smaller 4x4 sub-matrix is highlighted with a cyan background. The elements of this sub-matrix are labeled as $q_i \cdot k_j$ for $i, j \in \{1, 2, 3, 4\}$. The label 'N' is placed to the left of the sub-matrix, and another 'N' is placed below it, indicating the dimensions of the overall matrix.

$q1 \cdot k1$	$q1 \cdot k2$	$q1 \cdot k3$	$q1 \cdot k4$
$q2 \cdot k1$	$q2 \cdot k2$	$q2 \cdot k3$	$q2 \cdot k4$
$q3 \cdot k1$	$q3 \cdot k2$	$q3 \cdot k3$	$q3 \cdot k4$
$q4 \cdot k1$	$q4 \cdot k2$	$q4 \cdot k3$	$q4 \cdot k4$

Figure 9.8 The $N \times N$ \mathbf{QK}^T matrix showing how it computes all $q_i \cdot k_j$ comparisons in a single matrix multiple.

在语言建模任务中，只能看到上文，任务是预测下一个词，因此，不需要计算当前词语与下文词语的点积相似度，即，针对当前词语，要把其下文遮盖 (**mask**) 起来。计算实现上，将上图矩阵中的上三角部分置为 $-\infty$ 即可，经过softmax计算， $-\infty$ 对应的元素 (权重) 变为0。

4. 矩阵并行计算

实践中，将矩阵上三角部分置为 $-\infty$ 通过掩码矩阵 (mask matrix) 实现，按维度缩放 QK^T 后，再与一个掩码矩阵 M 相加：

$$M_{ij} = \begin{cases} -\infty, & j > i \\ 0, & \text{otherwise} \end{cases}$$

N	q1·k1	$-\infty$	$-\infty$	$-\infty$
	q2·k1	q2·k2	$-\infty$	$-\infty$
	q3·k1	q3·k2	q3·k3	$-\infty$
	q4·k1	q4·k2	q4·k3	q4·k4
N				

Figure 9.9 The $N \times N$ QK^T matrix showing the $q_i \cdot k_j$ values, with the upper-triangle portion of the comparisons matrix zeroed out (set to $-\infty$, which the softmax will turn to zero).

3. 归一化与加权求和

$$head = \text{softmax} \left(\text{mask} \left(\frac{Q K^T}{\sqrt{d_k}} \right) \right) V$$

softmax 的结果为一个维度为 $N \times N$ 的权重矩阵，第 i 行表示第 i 个词语与其他词语的相似度 (权重)，与矩阵 V 相乘得到第 i 个词语的新的表示 (其他词语value向量的加权和)。 $head$ 的维度为 $N \times d_v$ ，每行对应一个词。

4. 输出映射

$$A = head W^O, \quad W^O \in \mathbb{R}^{d_v \times d}, A \in \mathbb{R}^{N \times d}.$$

上述以矩阵形式并行计算一个序列的一个Attention输出的过程如下图所示 (图中省略了点积缩放和 softmax 计算):

4. 矩阵并行计算

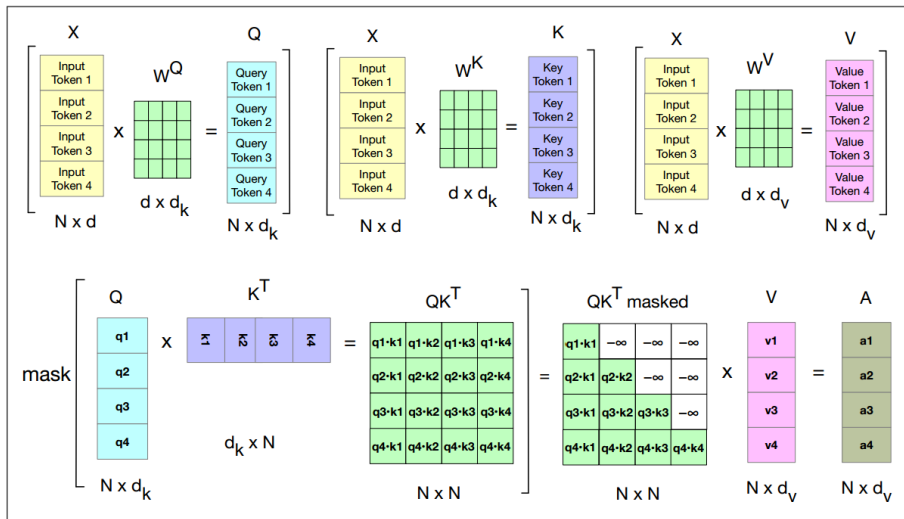


Figure 9.10 Schematic of the attention computation for a single attention head in parallel. The first row shows the computation of the Q , K , and V matrices. The second row shows the computation of QK^T , the masking (the softmax computation and the normalizing by dimensionality are not shown) and then the weighted sum of the value vectors to get the final attention vectors.

多头注意力的并行化

对每个头 $i = 1, \dots, A$ 重复上述步骤，得到 $\text{head}_i \in \mathbb{R}^{N \times d}$ ；将各头输出在列维度拼接 $[\text{head}_1 \oplus \dots \oplus \text{head}_A] \in \mathbb{R}^{N \times (A d_v)}$ ，再乘以投影矩阵 $W^O \in \mathbb{R}^{(A d_v) \times d}$ ，得到最终多头注意力输出：

$$Q^i = XW^{Q^i}; \quad K^i = XW^{K^i}; \quad V^i = XW^{V^i}$$

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

$$\text{MultiHeadAttention}(X) = (\text{head}_1 \oplus \text{head}_2 \dots \oplus \text{head}_A) W^O$$

整个Transformer Block 的并行形式

记输入为 $X \in \mathbb{R}^{N \times d}$, 可将一层 Transformer Block 概括为:

① $T_1 = \text{LayerNorm}(X)$

② $T_2 = \text{MultiHead}(T_1)$

③ $T_3 = T_2 + X$

④ $T_4 = \text{LayerNorm}(T_3)$

⑤ $T_5 = \text{FFN}(T_4)$

⑥ $H = T_5 + T_3$, 其中 $H \in \mathbb{R}^{N \times d}$ 即本层输出, 可作为下一层输入。

LayerNorm和FFN并行地作用于输入矩阵的每一行, 即每个词语。

并行化优势

- **高效利用硬件：**一次性调用大规模矩阵乘法与 Softmax，充分发挥 GPU/TPU 的并行吞吐能力；
- **编程简洁：**避免对每个位置的显式循环；
- **可扩展性强：**对序列长度 N 达数千、数万甚至更长的上下文窗口均可同样并行处理；
- **工程性能优化：**虽然注意力计算复杂度为 $O(N^2)$ ，但并行矩阵运算降低了实际执行时间，并可结合长度掩码和稀疏化等技术进一步优化。

目录

- 1 Transformer架构概述
- 2 Attention
- 3 Transformer Blocks
- 4 矩阵并行计算
- 5 Transformer的输入：token嵌入+位置嵌入**
- 6 语言模型建模

Transformer 的初始输入矩阵

$$X \in \mathbb{R}^{N \times d}$$

由两部分向量相加而成: Token 嵌入与 Position 嵌入。

1. Token 嵌入

嵌入矩阵

$$E \in \mathbb{R}^{|V| \times d}$$

包含词表中每个 token 的向量表示, 其中 $|V|$ 为词表大小, d 为模型维度。查表方式包括:

- ① **One-hot 乘法**: 将第 i 个 token 转为 one-hot 向量 $e_i \in \mathbb{R}^{1 \times |V|}$, 再计算 $e_i E \in \mathbb{R}^{1 \times d}$ 即为该 token 的嵌入向量。
- ② **索引切片**: 根据 token id 索引 E 的对应行, $E[\text{rows}, :]$, 效果等价。

$$\begin{matrix} & 5 & |V| \\ 1 & \boxed{0000100\dots0000} \end{matrix} \times \begin{matrix} & d \\ 5 & \boxed{} \\ |V| & \boxed{\text{E}} \end{matrix} = \begin{matrix} & d \\ 1 & \boxed{} \end{matrix}$$

Figure 9.11 Selecting the embedding vector for word V_5 by multiplying the embedding matrix \mathbf{E} with a one-hot vector with a 1 in index 5.

$$\begin{matrix} & |V| \\ \begin{matrix} \boxed{0000100\dots0000} \\ \boxed{0000000\dots0010} \\ \boxed{1000000\dots0000} \\ \dots \\ \boxed{0000100\dots0000} \end{matrix} \\ N & \end{matrix} \times \begin{matrix} & d \\ & \boxed{} \\ |V| & \boxed{\text{E}} \end{matrix} = \begin{matrix} & d \\ N & \boxed{} \end{matrix}$$

Figure 9.12 Selecting the embedding matrix for the input sequence of token ids W by multiplying a one-hot matrix corresponding to W by the embedding matrix \mathbf{E} .

2. Position 嵌入

绝对位置嵌入 (Absolute Position Embedding): 随机初始化一个位置嵌入矩阵

$$E_{\text{pos}} \in \mathbb{R}^{N \times d},$$

其中第 i 行 $E_{\text{pos}}[i]$ 对应序列中第 i 个位置。训练过程中, 该矩阵与其他参数一同学习。

静态正余弦嵌入 (Sinusoidal Embedding): 不依赖训练, 组合不同频率的正弦、余弦函数映射位置索引到 d 维空间, 可推广至任意长度。

相对位置嵌入 (Relative Position Embedding): 更复杂的方案, 直接在注意力计算中编码两词之间的相对位置, 而非一次性加在输入端。

3. 组合生成输入矩阵

单位置向量：对于序列中第 i 个 token，其最终输入向量为

$$x_i = E[\text{id}(w_i)] + E_{\text{pos}}[i] \in \mathbb{R}^{1 \times d}$$

整条序列：将所有 x_i 按行堆叠，得到

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \in \mathbb{R}^{N \times d}$$

作为后续所有 Transformer Block（自注意力等）的统一输入。

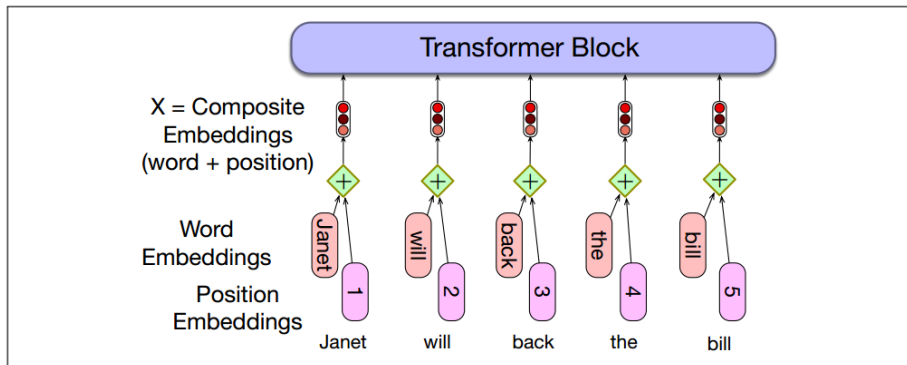


Figure 9.13 A simple way to model position: add an embedding of the absolute position to the token embedding to produce a new embedding of the same dimensionality.

通过这种“词汇嵌入 + 位置嵌入”的设计，Transformer 能在保持每个 token 固有语义的同时，将其在序列中的绝对或相对位置信息注入初始表示，为后续的自注意力层构建富含顺序感的上下文表示。

目录

- 1 Transformer架构概述
- 2 Attention
- 3 Transformer Blocks
- 4 矩阵并行计算
- 5 Transformer的输入：token嵌入+位置嵌入
- 6 语言模型建模**

语言建模头的定位与作用

Transformer 完成上下文表示后，还需在其顶端加入专门用于预测下一个词的模块—语言建模头（language modeling head），以实现自回归（causal）语言建模任务。

定位：位于最顶层 Transformer Block 之上，对每条输入序列的最后一个 token（位置 N ）的输出向量 h_N^L 进行处理。

作用：将维度为 $[1 \times d]$ 的输出向量映射为对词表中每个词的预测分数（logits），再经 softmax 归一化，得到下一位置 $N+1$ 的词分布。

回忆：语言模型本质上是词语预测器。在给定上下文词汇的情况下，它们会为每一个可能出现的后续词汇分配一个概率值。这种能力使得语言模型能够为所有潜在的后续词汇计算条件概率，从而生成覆盖整个词汇表的概率分布。

语言模型建模的结构与计算流程

1. 线性投影（Unembedding / Logits 计算）

输入：最后一层第 N 个位置的输出 $h_N^L \in \mathbb{R}^{1 \times d}$ 。投影矩阵：通常记作 $U \in \mathbb{R}^{d \times |V|}$ ，亦称“unembedding”矩阵。

计算：

$$u = h_N^L U \in \mathbb{R}^{1 \times |V|}$$

其中 $u = (u_1, \dots, u_{|V|})$ 为对词表中每个词的未归一化分数（logits）。

2. 权重绑定（Weight Tying）

为了减少参数并增强表示对应性，投影矩阵 U 常与输入嵌入矩阵 $E \in \mathbb{R}^{|V| \times d}$ 共享参数：

$$U = E^\top$$

即输入阶段用 E 将 one-hot 词向量映射到 embedding 空间，输出阶段用 E^\top 将输出向量映射回词表维度。

3. Softmax 归一化

将 logits 分数 u 经 softmax 转为概率分布:

$$y = \text{softmax}(u) \in \mathbb{R}^{1 \times |V|}, \sum_{k=1}^{|V|} y_k = 1$$

4. 生成与解码

在推理（generation）阶段，可对分布 y 进行贪心（greedy）或采样策略，选取词表索引 k 生成下一个 token w_{N+1} 。每个位置均反复此流程，实现自回归地逐词生成文本。针对单个 token 的整体堆叠架构见第44页。

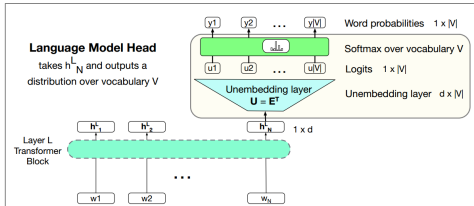


Figure 9.14 The language modeling head: the circuit at the top of a transformer that maps from the output embedding for token N from the last transformer layer (h_N^L) to a probability distribution over words in the vocabulary V .

6. 语言模型建模

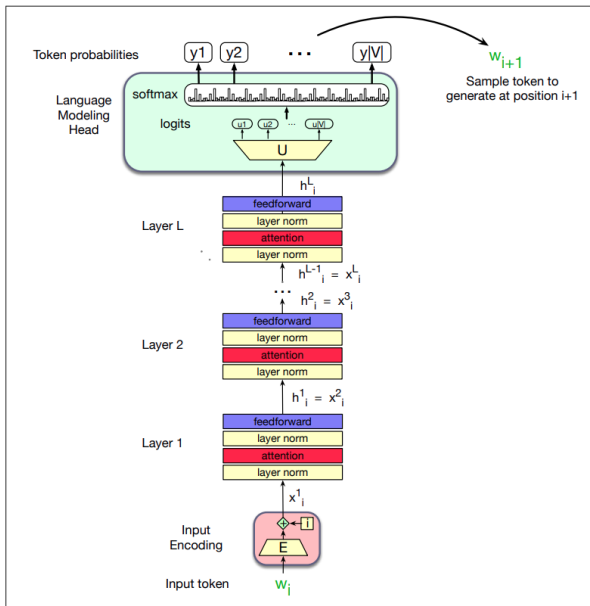


Figure 9.15 A transformer language model (decoder-only), stacking transformer blocks and mapping from an input token w_i to a predicted next token w_{i+1} .

THE END