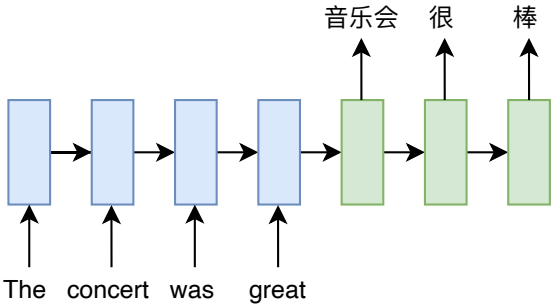
1. 注意力机制

1.1 背景与动机

（1）序列建模

序列建模是针对具有时间或顺序依赖关系的序列数据（如文本、时间序列），通过模型捕捉数据内在顺序关联，以实现预测、分类、生成等任务的机器学习任务。

序列建模任务（如机器翻译、语音识别、文本生成）中，早期主流方法是基于 **RNN/LSTM** 的编码器-解码器架构。



假设序列模型的输出和输出分别为：

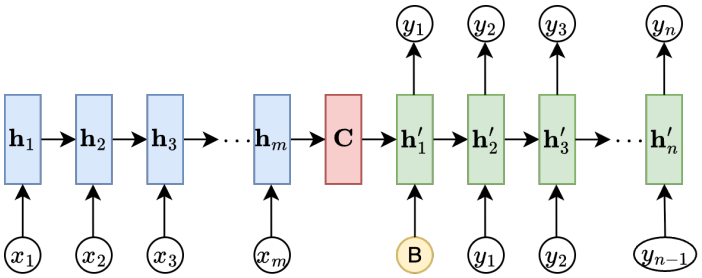
$$\begin{aligned}\text{Source} &= \langle \mathbf{x}_1, \mathbf{x}_2 \dots \mathbf{x_m} \rangle \\ \text{Target} &= \langle \mathbf{y}_1, \mathbf{y}_2 \dots \mathbf{y_n} \rangle\end{aligned}$$

则编码器-解码器架构可表示为：

$$\begin{aligned}\mathbf{C} &= E\left(\mathbf{x}_1, \mathbf{x}_2 \dots \mathbf{x_m}\right) \\ \mathbf{y_i} &= D\left(\mathbf{C}, \mathbf{y}_1, \mathbf{y}_2 \dots \mathbf{y_{i-1}}\right)\end{aligned}$$

其中：

- **源序列（Source）**：$\langle \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m \rangle$，这是输入序列，例如一段文本中的单词或令牌。
- **目标序列（Target）**：$\langle \mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n \rangle$，这是输出序列，例如翻译后的文本或生成的响应。
- **编码器（Encoder）**：$\mathbf{C} = E(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m)$，编码器 E 将整个源序列编码为一个固定大小的上下文向量 $\mathbf{C}$。这个向量旨在捕获输入序列的语义信息。
- **解码器（Decoder）**：$\mathbf{y}_i = D(\mathbf{C}, \mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_{i-1})$，解码器 D 使用上下文向量 $\mathbf{C}$ 和之前生成的所有目标令牌（即 $\mathbf{y}_1$ 到 $\mathbf{y}_{i-1}$）来生成下一个令牌 $\mathbf{y}_i$。这体现了自回归（autoregressive）生成过程，即每个步骤的输出依赖于之前的输出。



（2）简单编码器-解码器架构的问题

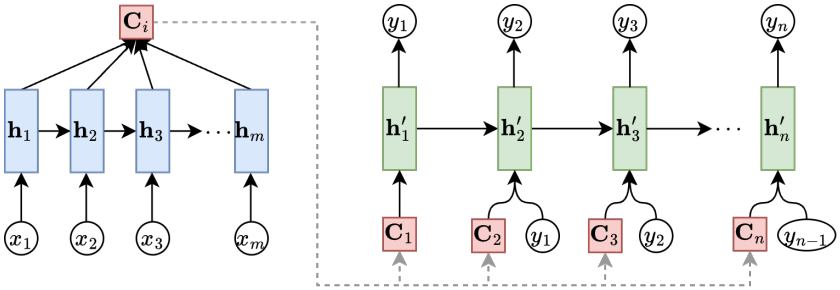
- 长距离依赖难以建模
  - RNN 在理论上可以捕捉任意长距离依赖，但在实际中，由于梯度消失或梯度爆炸，模型更倾向于依赖最近的输入。即使 LSTM/GRU 在一定程度上缓解了问题，但对超长文本的捕捉能力仍有限。
- 信息压缩瓶颈
  - 解码器需要依赖编码器输出的一个固定长度向量（context vector）来生成翻译。这意味着整个输入序列的信息被“压缩”到一个向量中，导致信息丢失。
- 选择性不足
  - 不同的输入对预测结果的重要性不同，但固定表示无法动态反映这种差异。

解决思路：引入一种机制，让模型在生成输出的每一步时，**能够“选择性地”关注输入序列中最相关的部分**。这就是注意力机制（Attention Mechanism）的核心动机。

1.2 基本原理

（1）包含注意力机制的编码器-解码器架构

注意力机制的核心思想是：**通过计算输入中不同部分的重要性分数，进行加权组合，从而动态提取信息。**



编码器（通常是一个双向RNN）处理源序列，为每个输入 token 生成一个包含上下文信息的隐藏状态：

$$\mathbf{h}_j = \mathrm{E}\left(\mathbf{x}_j, \mathbf{h}_{j-1}\right), \quad j \in 1, \ldots m$$

解码器的当前状态 $\mathbf{h}'_i$ 由其前一个状态 $\mathbf{h}'_{i-1}$、前一个输出 $\mathbf{y}_{i-1}$ 和**上一个时间步的上下文向量** $\mathbf{C}_{i-1}$ 共同决定：

$$\mathbf{h}'_i = \mathrm{D}\left(\mathbf{y}_{i-1}, \mathbf{h}'_{i-1}, \mathbf{C}_{i-1}\right)$$

上下文向量 $\mathbf{C}_i$ 是**所有编码器隐藏状态的加权和**，权重表示为 α_{ij}（注意力权重）。

$$\mathbf{C}_i = \sum_{j=1}^{m} \alpha_{ij} \mathbf{h}_j$$

（2）注意力的计算

考虑当前解码器状态 $\mathbf{h}'_i$，它与每一个编码器状态 $\mathbf{h}_j$ 的相关性得分为 e_{ij}：

$$e_{ij} = \text{score}(\mathbf{h}'_i, \mathbf{h}_j)$$

常见的评分函数有加性（Additive）、点积（Dot-Product）、缩放点积（Scaled Dot-Product）等。

将得分通过 softmax 函数归一化，得到**注意力权重** α_{ij}，它表示在生成第 i 个输出时，模型对第 j 个输入 token 的“关注”程度。

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{m} \exp(e_{ik})}$$

注意力机制有多种变体，主要区别在于 **score 函数** 的设计。

- 加性注意力（Additive Attention, Bahdanau Attention）

$$\text{score}(\mathbf{h}', \mathbf{h}) = \mathbf{h}'^{\top} \tanh(\mathbf{W}_q \mathbf{h}' + \mathbf{W}_k \mathbf{h})$$

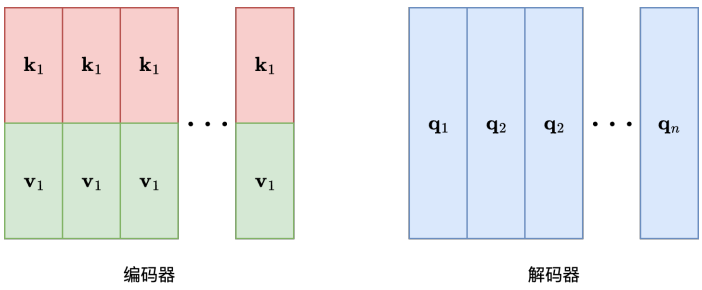
- 乘性注意力

$$\text{score}(\mathbf{h}', \mathbf{h}) = \mathbf{h}'^{\top} \mathbf{h}$$

（3）注意力机制的解读

注意力机制通常也被理解为一种“查询”的过程，包括三个核心组件：

- **Query (Q - 查询)**：代表当前的需要或“问题”。在注意力机制中，它通常是解码器在某一时间步的隐藏状态 $\mathbf{h}'_i$。表示在当前时刻，需要从编码器中寻找什么信息。
- **Key (K - 键)**：代表输入序列中每个元素的“标识”。通常是编码器隐藏状态集合 $\mathbf{H} = \{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_m\}$。每个 Key $\mathbf{h}_j$ 是其所对应输入 token 的特征表示，用于与 Query 进行匹配，计算相关性。
- **Value (V - 值)**：代表输入序列中每个元素真正的“语义内容”或“信息”。在许多基础架构中，Value 与 Key 是相同的，即 $\mathbf{v}_j = \mathbf{h}_j$。但在更复杂的模型（如 Transformer）中，Value 可以是编码器状态的另一个线性变换，从而允许模型存储和提取与匹配过程不同的信息。



注意力权重计：

$$e_{ij} = \text{score}(\mathbf{q}_i, \mathbf{k}_j)$$
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{m} \exp(e_{ik})}$$

上下文向量 (Context Vector)：

$$\mathbf{C}_i = \sum_{j=1}^{m} \alpha_{ij} \mathbf{v}_j$$

2. 自注意力机制（Self-Attention）

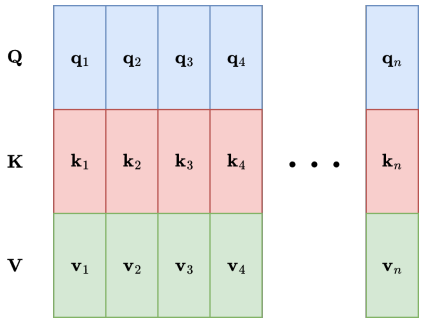
2.1 背景与动机

在传统的注意力机制（Attention）中，**查询向量（Query）通常来自解码器，而键（Key）和值（Value）来自编码器**。这种设计适合输入序列和输出序列不同的场景，例如机器翻译任务：输入是源语言句子，输出是目标语言句子。

然而，在许多任务中，我们希望在**同一个序列内部**建模不同位置之间的依赖关系。例如，在句子表示学习中，某个词的含义往往取决于句中其他词。**自注意力机制（Self-Attention）**的核心思想是：

- Query、Key、Value 全部来自同一个输入序列；
- 每个位置的表示通过与序列中所有位置的交互来更新。

这样，每个位置的表示能够整合全局信息，而不再局限于卷积神经网络（CNN）中的局部感受野，或循环神经网络（RNN）中的逐步信息传递。这一特性为长距离依赖建模提供了强有力的工具。



2.2 计算过程

设输入序列长度为 n，每个元素的维度为 d_{model}，则输入矩阵可表示为

$$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n] \in \mathbb{R}^{n \times d_{\text{model}}}.$$

自注意力的计算过程可分为以下几个步骤。

(1) 线性映射得到 Q, K, V

通过三个不同的线性变换矩阵 $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{d_{\text{model}} \times d_k}$，分别得到查询、键和值矩阵：

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}^K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}^V.$$

其中，$\mathbf{Q} \in \mathbb{R}^{n \times d_k}$，$\mathbf{K} \in \mathbb{R}^{n \times d_k}$，$\mathbf{V} \in \mathbb{R}^{n \times d_v}$。

(2) 注意力分数的计算

对于序列中第 i 个位置，其查询向量 $\mathbf{q}_i$ 与所有键向量 $\mathbf{k}_j$ 计算点积相似度：

$$\text{score}(\mathbf{q}_i, \mathbf{k}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{\mathbf{d}_k}}.$$

分母的 $\sqrt{d_k}$ 是缩放因子（scaling factor），用于缓解当 d_k 较大时点积数值过大、导致 softmax 梯度过小的问题。

(3) softmax 归一化

将上述分数转换为概率分布，得到注意力权重：

$$\alpha_{ij} = \frac{\exp(\text{score}(\mathbf{q}_i, \mathbf{k}_j))}{\sum_{j'=1}^{n} \exp(\text{score}(\mathbf{q}_i, \mathbf{k}_{j'}))}.$$

其中，α_{ij} 表示位置 i 对位置 j 的依赖程度。

(4) 加权求和得到输出

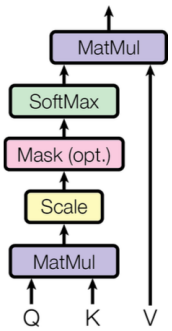
基于注意力权重，更新后的表示为：

$$\mathbf{z}_i = \sum_{j=1}^{n} \alpha_{ij} \mathbf{v}_j.$$

将所有位置的结果组合，可以写成矩阵形式：

$$\mathbf{Z} = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}$$

直观上，每个位置的输出向量是序列中所有位置值向量的加权和，而权重由对应的注意力分数决定。



2.3 多头自注意力（Multi-Head Self-Attention）

单一的注意力机制往往只能捕捉一种关系模式，例如语义相似性。但自然语言中的依赖关系十分复杂，既包括语法上的主谓关系，也包括语义上的同义或上下位关系。为了增强模型的表达能力，Transformer 引入了**多头注意力机制**。

在多头注意力中，采用多个不同的线性变换矩阵：

$$\mathbf{W}_i^Q, \mathbf{W}_i^K, \mathbf{W}_i^V, \quad i = 1, \ldots, h$$

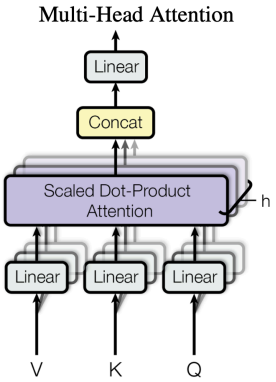
并行计算 h 个注意力头，每个注意力头在不同的子空间中学习依赖关系。其结果拼接后再经过一次线性映射，得到最终输出：

$$\text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V),$$

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W}^O,$$

其中 $\mathbf{W}^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$ 是输出映射矩阵。

直观地，可以将不同的注意力头看作不同的“专家”：某些头可能更擅长捕捉句法关系，而另一些头则专注于语义关联。多个头的并行建模，使得模型能够在多个角度综合理解输入序列。



2.4 特点

自注意力机制相较于传统序列建模方法具有显著优势。首先，它能够**并行计算**，因为所有位置的表示可以同时更新，不再依赖顺序展开；其次，它能够自然地建模**长距离依赖**，任意两个位置都能直接交互，而不会像 RNN 那样受到梯度消失的困扰；最后，通过多头注意力机制，模型具备了在不同子空间中建模多种依赖关系的**灵活性**。

然而，自注意力机制也存在不足。由于需要计算一个 $n \times n$ 的注意力矩阵，其时间和空间复杂度均为 $O(n^2)$，当输入序列过长时，计算和存储开销十分巨大。此外，自注意力本身不包含位置信息，因此必须额外引入**位置编码（Positional Encoding）**来为模型提供顺序感知能力。

3. Transformer

3.1 背景与动机

在深度学习早期，自然语言处理主要依赖循环神经网络（RNN）及其变体长短期记忆网络（LSTM）。RNN 通过顺序结构捕捉上下文依赖，但在建模长距离依赖时容易遭遇梯度消失或梯度爆炸的问题。卷积神经网络（CNN）也曾被用于序列建模，但卷积操作本质上依赖局部感受野，需要堆叠多层卷积才能扩大感受野，效率不高。

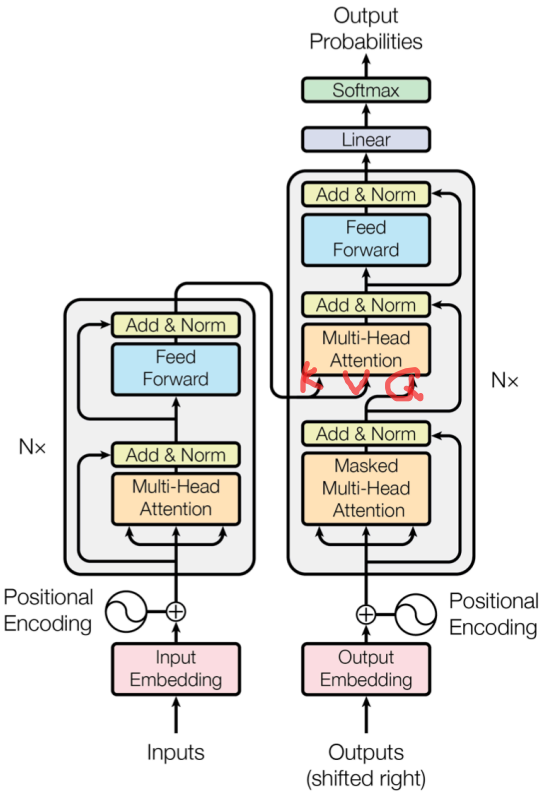
2017 年，Vaswani 等人提出的 Transformer 模型彻底改变了这一格局。Transformer 完全抛弃了循环和卷积结构，转而使用基于注意力的机制，尤其是自注意力（Self-Attention），来捕捉序列中的依赖关系。这一设计不仅大幅提升了并行计算效率，还在长距离依赖建模上表现优越。自提出以来，Transformer 已成为自然语言处理和人工智能领域的核心架构，并推动了大规模预训练语言模型（如 BERT、GPT 系列）的发展。

3.2 总体架构

Transformer 的总体架构由**编码器（Encoder）**和**解码器（Decoder）**两个部分组成，二者通过注意力机制进行交互。整体可以看作一个堆叠的模块化框架：编码器由若干相同的编码器层堆叠而成，解码器由若干相同的解码器层堆叠而成。

设输入序列为 $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]$，其经过嵌入层（Embedding）和位置编码（Positional Encoding）后得到连续向量表示。编码器将其转化为一系列上下文表示；解码器在自回归生成时接收已生成的序列作为输入，并通过交叉注意力机制获取编码器的上下文信息，逐步生成目标序列$\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_m]$。

Transformer 的核心单元是**多头自注意力机制**与**前馈神经网络**的组合。每个编码器层和解码器层都包含这些核心模块，并辅以残差连接和层归一化（Layer Normalization），以保证训练稳定性。



3.3 编码器

编码器的输入是经过嵌入和位置编码的序列表示。每个编码器层由两部分组成：多头自注意力子层（Multi-Head Self-Attention）和前馈全连接子层（Position-wise Feed-Forward Network）。

首先，多头自注意力子层根据输入序列 $\mathbf{X}$ 计算 Query、Key 和 Value，然后得到上下文表示：

$$\mathbf{Z} = \text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}).$$

这使得每个位置的表示能够直接整合全局信息。随后，编码器对子层输出施加残差连接与层归一化：

$$\tilde{\mathbf{Z}} = \text{LayerNorm}(\mathbf{X} + \mathbf{Z}).$$

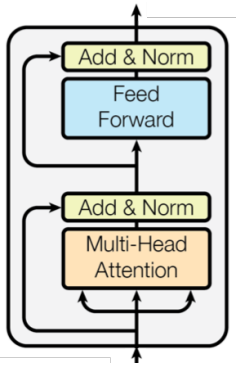
接下来是逐位前馈神经网络，其结构通常为两层全连接层，中间带有非线性激活函数（如 ReLU 或 GELU）：

$$\text{FFN}(\mathbf{z}) = \max(0, \mathbf{z}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2.$$

将其应用于序列的每个位置，并再次施加残差连接与归一化：

$$\mathbf{H} = \mathrm{LayerNorm}(\tilde{\mathbf{Z}} + \mathrm{FFN}(\tilde{\mathbf{Z}})).$$

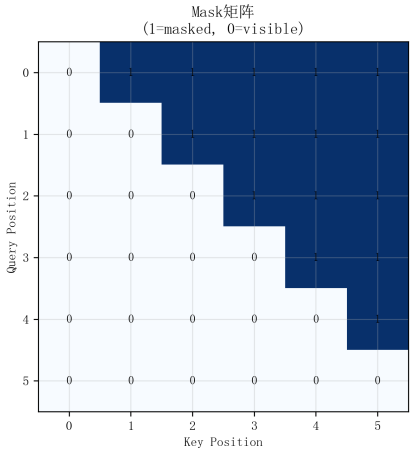
通过堆叠 N 个这样的编码器层，可以逐步增强输入序列的表征能力。

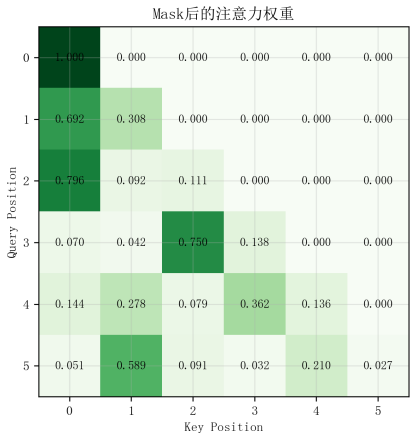


3.4 解码器

解码器的结构与编码器类似，但为了保证生成的因果性，解码器在自注意力子层中引入了**掩码（mask）机制**。具体而言，在计算注意力时，仅允许当前位置关注先前位置，从而保证序列的自回归性质。

一个解码器层由三部分组成。第一部分是带掩码的多头自注意力机制，其计算方式与编码器类似，但在 softmax 之前将未来位置的得分设为负无穷大，以确保信息流方向正确。





第二部分是与编码器交互的多头注意力机制，称为**编码器—解码器注意力（Encoder–Decoder Attention）**。在这一部分，解码器的查询向量来自解码器的隐状态，而键和值来自编码器的输出，从而使解码器能够利用输入序列的上下文信息。

最后，仍然是逐位置前馈神经网络以及残差连接与归一化。

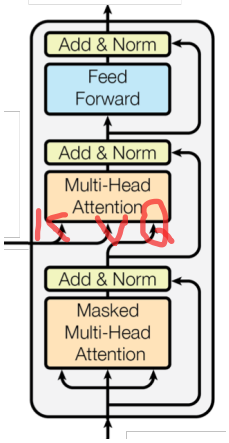
设解码器某层的输入为 $\mathbf{Y}^{(l-1)}$，其三个子层依次计算：

$$\tilde{\mathbf{Y}}^{(l)} = \mathrm{LayerNorm}\big(\mathbf{Y}^{(l-1)} + \mathrm{MaskedMultiHead}(\mathbf{Y}^{(l-1)}, \mathbf{Y}^{(l-1)}, \mathbf{Y}^{(l-1)})\big),$$

$$\hat{\mathbf{Y}}^{(l)} = \mathrm{LayerNorm}\big(\tilde{\mathbf{Y}}^{(l)} + \mathrm{MultiHead}(\tilde{\mathbf{Y}}^{(l)}, \mathbf{H}^{(N)}, \mathbf{H}^{(N)})\big),$$

$$\mathbf{Y}^{(l)} = \mathrm{LayerNorm}\big(\hat{\mathbf{Y}}^{(l)} + \mathrm{FFN}(\hat{\mathbf{Y}}^{(l)})\big),$$

其中 $\mathbf{H}^{(N)}$ 表示编码器最后一层的输出。



3.5 层归一化

层归一化（Layer Normalization）的作用是稳定训练过程并加快收敛。在 Transformer 中，它通常与残差连接一起出现（即 Add & Norm 结构）。

假设序列表示矩阵为$\mathbf{H} \in \mathbb{R}^{n \times d}$，其中 n 为序列长度，d 为特征维度。层归一化在每个位置（矩阵的每一行）上独立进行，即对向量

$$\mathbf{h} = [h_1, h_2, \ldots, h_d] \in \mathbb{R}^d$$

执行以下操作：

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

$$\hat{h}_i = \frac{h_i - \mu}{\sqrt{\sigma^2 + \epsilon}}, \quad y_i = \gamma \hat{h}_i + \beta,$$

其中 $\gamma, \beta \in \mathbb{R}^d$ 是可学习参数。

与批归一化（BatchNorm）不同，LayerNorm 的均值和方差在**特征维度**上计算，而不是在 batch 维度上，因此与 batch 大小无关，更适合自然语言处理等序列任务。

3.6 位置编码

自注意力机制本身不包含序列的位置信息，因此 Transformer 需要额外引入**位置编码（Positional Encoding）**。位置编码将序列中每个位置的顺序信息显式地加入到嵌入向量中。

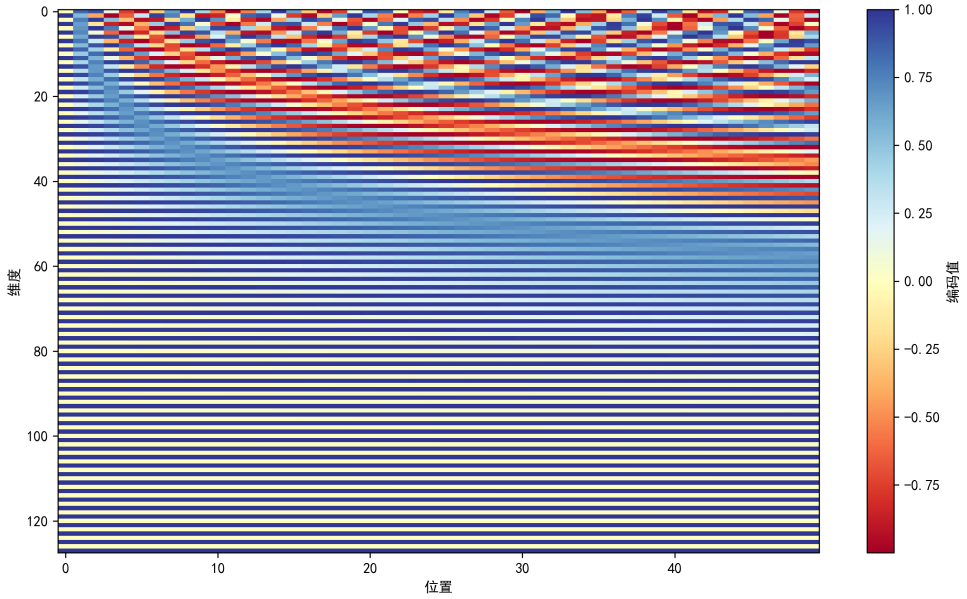
Vaswani 等人提出了一种基于三角函数的固定位置编码，其定义为

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

其中 pos 表示位置，i 表示维度索引。这样的位置编码能够为不同位置提供独特的表示，同时保证相对位置信息可以通过线性组合表达。

在实践中，也有使用**可学习的位置编码**，即将位置索引作为参数进行训练。这种方法能够在数据驱动下自动优化位置表示。

如下图所示，最大长度为50的句子的128维位置编码。每一列代表一个token的嵌入向量。



3.7 Transformer 训练

Transformer 的训练目标通常是最大化条件概率分布，即给定输入序列 $\mathbf{X}$，预测目标序列 $\mathbf{Y}$ 的概率：

$$P(\mathbf{Y} \mid \mathbf{X}) = \prod_{t=1}^{m} P(\mathbf{y}_t \mid \mathbf{y}_{<t}, \mathbf{X})$$

在实现上，解码器通过 softmax 输出在词表上的概率分布。设解码器输出的隐藏状态为 h_t，则预测分布为

$$P(\mathbf{y}_t \mid \mathbf{y}_{<t}, \mathbf{X}) = \text{softmax}(\mathbf{h}_t \mathbf{W}^O + \mathbf{b}^O),$$

其中 $\mathbf{W}^O$ 和 $\mathbf{b}^O$ 为输出层参数。

训练时采用**交叉熵损失函数**，目标是最小化预测分布与真实标签分布之间的差异：

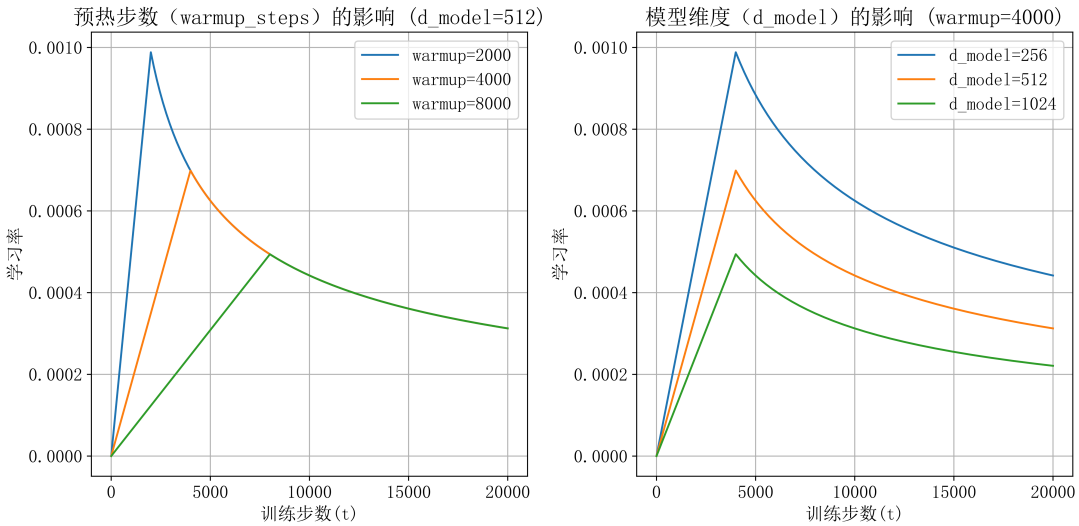
$$\mathcal{L} = -\sum_{t=1}^{m} \log P(\mathbf{y}_t^* \mid \mathbf{y}_{<t}^*, \mathbf{X}),$$

其中 y_t^* 表示目标序列的真实第 t 个词。

为了提高训练稳定性，Transformer 中还采用了**残差连接、层归一化和 Dropout** 等技巧。优化器方面，论文提出了带有学习率预热（warmup）的 Adam 优化器，其学习率随训练步数 t 的变化形式为

$$\text{lr}(t) = d_{\text{model}}^{-\frac{1}{2}} \cdot \min\left(t^{-\frac{1}{2}},\, t \cdot \text{warmup\_steps}^{-\frac{3}{2}}\right).$$

这种设计在训练初期逐渐增加学习率，以避免模型过早陷入不良局部最优，而在后期逐步减小学习率，以保证收敛的稳定性。



Transformer以自注意力机制为核心，通过编码器–解码器结构实现信息交互，并辅以位置编码补充序列信息。在训练过程中，优化方法与归一化技巧进一步保证了模型的高效学习。凭借这些特性，Transformer 成为现代深度学习尤其是自然语言处理的主流框架。

参考文献

- 注意力机制：Neural Machine Translation by Jointly Learning to Align and Translate, 2015.
- **Transformer：Attention Is All You Need, 2017.**