Transformer

Sequence to Sequence Model的一种

编码器解码器架构作用：编码器全面理解输入序列的语义，并将其压缩为高阶表示（**Context**），解码器则基于此上下文信息，逐步生成目标序列

输入部分细节

1. **Word Embedding**（词向量嵌入）

- 把输入的每个词（一个ID）转换成一个向量，比如 512维。
- 使用可学习的 `nn.Embedding` 层实现。

2. **Positional Encoding**（位置编码）

- 因为 Transformer 不像 **RNN** 有顺序结构，所以必须显式加入位置信息。
- 分两种方式：
  - 原始论文用的固定的正余弦函数
  - 现在大多数用的可学习的位置向量

Encoder模块细节

一个 Encoder 包括多个重复的子层，即**block**块（通常 6 层）：

每层（个**block**）包含两个子模块：

1. 多头注意力机制（**Multi-Head Self Attention**）
   - 输入之间相互看 → 比如“我 爱 学习”，每个词都看整个句子
2. 前馈神经网络（**Feed Forward Network**）
   - 每个词单独处理，升维、激活、降维，类似 MLP
   - 小型的全连接网络

每个子模块后都有：

- 残差连接
- **Layer Normalization**

Decoder部分细节

作用：产生输出

- 会把上一个时间节点的输出当作当前时间节点的输入
- **Auto-regressive**（自回归）类型

基本构成

每一层 Decoder 包含 **3** 个子层 + 残差连接 **+ LayerNorm**：

- 已生成的词作为带掩码自注意力的输入，要进行位置编码和词向量生成，且解码器的输入是随着解码器的输出不断变化的
- 经过编码器处理过的输入和带掩码自注意力的输出作为多头注意力的输入

1. **Masked Multi-Head Self-Attention**（带掩码的自注意力）
   - 作用：让每个位置的词只能“看到自己和前面的词”
   - 用法：防止 Decoder 在训练时“看到未来词”，屏蔽未来信息
2. **Encoder-Decoder Attention**（跨模块注意力）
   - 作用：让 Decoder 能看到 Encoder 编码过的输入序列

    - **Query** 来自 **Decoder**, **Key** 和 **Value** 来自 **Encoder** 的输出。

    - 让 Decoder 能“参考”输入句子的语义信息，这样就可以用注意力机制让 Decoder“参考”输入句子，在生成翻译/回答/续写时更合理

3. **Feed Forward Network**（前馈神经网络）

    - 结构：两个全连接层 + 激活函数（**ReLU/GeLU**）、

4. 残差连接 **+ LayerNorm**

   每个子层后都加：

    - 残差连接：`output = input + Sublayer(input)`
    - LayerNorm：保持训练稳定、收敛更快

- 最后输出的矩阵只有第**n**行会用来预测下一个词

image-20250702225431058

Train（训练细节）

1. **Encoder**:

    - 接收输入序列（如英文句子），编码成一系列上下文相关的向量

    - 每个向量代表一个词的语义信息（包含上下文）

2. **Decoder**:

    - 输入目标序列（如中文句子）中前面的真实词（即 label 中已知的部分）

    - 每一步预测下一个词（比如预测“我 爱 ____”里的“你”）

3. **Teacher Forcing**:

    - 在训练阶段，模型每一步的输入直接使用真实目标序列中的**token**（即**“**正确答案**”**），而不□模型自己生成的token
    - 训练时，Decoder 不用自己的输出作为下一步输入
    - 而□用真实的上一个词，快速学习，避免误差累积

Attention细节

- **Decoder** 内部的 **Self-Attention**: **Mask** 住后面的词，防止模型看到答案（实现自回归）
- **Encoder-Decoder Attention**: Decoder 的每一层都会“参考” Encoder 输出的语义向量，来帮助自己理解输入句子的含义

损失函数

- 每个位置的输出 → **softmax** → 得到一个词的概率分布
- 与真实词的 one-hot 编码做对比 → 使用 **Cross Entropy Loss**

优化目标

使所有预测位置的交叉熵损失最小化

即：模型学会尽可能接近地预测出目标句子中的每一个词。

Teaching Forcing

训练时，**Decoder** 是可以看到**“**前面的正确答案**”**的，但不能看到**“**当前或未来的词**”**。这个技巧叫做 `Teacher Forcing`（教师强制）。

训练 Decoder 的时候：

- 模型生成第一个词的时候，输入 `<BOS>`（开始符）
- 第二个词用 真实的第一个词（比如“我”）作为输入
- 第三个词用真实的“我 爱”
- …直到句尾

而 不是 用模型上一步自己预测的词作为下一步的输入。

这种做法就叫 **Teacher Forcing**。

Residual Connection（残差连接）

基本原理

它将层的输入直接加到该层的输出上，形成"捷径"或"跳跃连接"。如果一个层的输入□x，输出□F(x)，则残差连接后的最终输出□x + F(x)

缓解梯度消失

- 如果一个层的输出□ $y = F(x) + x$（残差连接）
- 那么反向传播时，梯度 ∂L/∂x 可以分解为两部分：

$$ \frac{\partial{L}}{\partial{y}} ·\frac{\partial{y}}{\partial{x}} = \frac{\partial{L}}{\partial{y}} · (\frac{\partial{F(x)}}{\partial{x}} + 1) $$

- 即使 $\frac{\partial{F(x)}}{\partial{x}}$ 很小，加上1后仍能保证有效的梯度传递

Layer Normalization

它□做标准化的，避免不同样本间分布不稳定。

- 与 **BatchNorm** 不同，它对的□一个样本内部的所有特征归一化，而不□整批样本。
- 在 NLP 序列建模中比 BatchNorm 更合适（因为样本长度不固定、batch 大小可能很小）
- **Layer Normalization**□对同一个**feature**不同的**dimension**进行归一化，**Batch Normalization**□对不同的**feature**的同一个**dimension**进行归一化

Explanation

给定四个词，下面展示**self-attention**的计算过程

单头注意力

image-20250427121123206

> 1. □输入进行□嵌入，加上位置编码得到a^1,a^2,a^3,a^4
> 2. 计算查询向量、键向量、值向量：
>
> $$ \begin{aligned} Q = \begin{bmatrix}q^1 \ q^2 \ q^3 \ q^4\end{bmatrix} &= \begin{bmatrix}a^1 \ a^2 \ a^3 \ a^4\end{bmatrix}W^q \[1em] K = \begin{bmatrix}k^1 \ k^2 \ k^3 \ k^4\end{bmatrix} &= \begin{bmatrix}a^1 \ a^2 \ a^3 \ a^4\end{bmatrix}W^k \[1em] V = \begin{bmatrix}v^1 \ v^2 \ v^3 \ v^4\end{bmatrix} &= \begin{bmatrix}a^1 \ a^2 \ a^3 \ a^4\end{bmatrix}W^v \ \end{aligned} $$
>
> 2. 计算**attention score** $$ \begin{aligned} A = \begin{bmatrix}\alpha_{1,1} & \alpha_{1,2} & \alpha_{1,3} & \alpha_{1,4}\ \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} & \alpha_{2,4} \ \alpha_{3,1} & \alpha_{3,2} & \alpha_{3,3} & \alpha_{3,4} \ \alpha_{4,1} & \alpha_{4,2} & \alpha_{4,3} & \alpha_{4,4}\end{bmatrix} &=Q \cdot K^T\ &=

> \begin{bmatrix}q^1 \ q^2 \ q^3 \ q^4\end{bmatrix} \cdot \begin{bmatrix}k^1 & k^2 & k^3 & k^4\end{bmatrix} \end{aligned} $$
>
> 3. 经过$\sqrt{d_k}$放缩作**softmax**，d_k为每个$key/querey$向量的维度大小
>
> $$ \begin{aligned} &\quad\quad\quad\quad\quad\quad\text{输入矩阵 } A \xrightarrow{\text{softmax}} \text{输出矩阵 } A^{'} \ &\frac{1}{\sqrt{d_k}}\begin{bmatrix}\alpha_{1,1} & \alpha_{1,2} & \alpha_{1,3} & \alpha_{1,4}\ \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} & \alpha_{2,4} \ \alpha_{3,1} & \alpha_{3,2} & \alpha_{3,3} & \alpha_{3,4} \ \alpha_{4,1} & \alpha_{4,2} & \alpha_{4,3} & \alpha_{4,4}\end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix}\alpha^{'}{1,1} & \alpha^{'}{1,2} & \alpha^{'}{1,3} & \alpha^{'}{1,4}\ \alpha^{'}{2,1} & \alpha^{'}{2,2} & \alpha^{'}{2,3} & \alpha^{'}{2,4} \ \alpha^{'}{3,1} & \alpha^{'}{3,2} & \alpha^{'}{3,3} & \alpha^{'}{3,4} \ \alpha^{'}{4,1} & \alpha^{'}{4,2} & \alpha^{'}{4,3} & \alpha^{'}{4,4}\end{bmatrix} \end{aligned} $$
>
> 4. 计算与值向量加权求和的值
>
> $$ [b^1,b^2,b^3,b^4] = \begin{bmatrix}\alpha^{'}{1,1} & \alpha^{'}{1,2} & \alpha^{'}{1,3} & \alpha^{'}{1,4}\ \alpha^{'}{2,1} & \alpha^{'}{2,2} & \alpha^{'}{2,3} & \alpha^{'}{2,4} \ \alpha^{'}{3,1} & \alpha^{'}{3,2} & \alpha^{'}{3,3} & \alpha^{'}{3,4} \ \alpha^{'}{4,1} & \alpha^{'}{4,2} & \alpha^{'}{4,3} & \alpha^{'}{4,4}\end{bmatrix} \cdot \begin{bmatrix}v^1 \ v^2 \ v^3 \ v^4\end{bmatrix} $$
>
> 即 $$ Attention(Q.K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V $$

多头注意力

如下图所示：会使用多的矩阵去作变换，如$W^{q,1},W^{q,2}$

image-20250510194255934

以四个词，两个头为例，下面来展示计算过程

- 对于四个词向量a^i,a^j,a^m,a^n，可以用$W^q、W^k、W^v$先计算出全局查询的查询、键、值向量，$Q、K、V$
- 其中：$W^q、W^k、W^v$为全局权重矩阵

$$ Q = \begin{bmatrix} a^i\ a^j\a^m\a^n \end{bmatrix}W^q\ K = \begin{bmatrix} a^i\ a^j\a^m\a^n \end{bmatrix}W^k\ V = \begin{bmatrix} a^i\ a^j\a^m\a^n \end{bmatrix}W^v $$

- 然后进行多头拆分：

！！要注意的□多头拆分也可以直接通过全局矩阵**$Q、K、V$**进行分割 $$ \begin{align} Q_1 = QW^{q,1} \quad Q_2 = QW^{q,2} \ K_1 = KW^{q,1} \quad K_2 = KW^{q,2}\ V_1 = VW^{q,1} \quad V_2 = VW^{q,2} \end{align} $$

- 可得出表达式

$$ \begin{align} Q_1 &= \begin{bmatrix} q^{i,1}\ q^{j,1}\q^{m,1}\q^{n,1} \end{bmatrix} Q_2 = \begin{bmatrix} q^{i,2}\ q^{j,2}\q^{m,2}\q^{n,2} \end{bmatrix} \

\end{align} $$

$$ K_1 = \begin{bmatrix} k^{i,1}\ k^{j,1}\k^{m,1}\k^{n,1} \end{bmatrix} K_2 = \begin{bmatrix} k^{i,2}\ k^{j,2}\k^{m,2}\k^{n,2} \end{bmatrix} $$

$$ V_1 = \begin{bmatrix} v^{i,1}\ v^{j,1}\v^{m,1}\v^{n,1} \end{bmatrix} V_2 = \begin{bmatrix} v^{i,2}\ v^{j,2}\v^{m,2}\v^{n,2} \end{bmatrix} $$

- 接着计算每个头的注意力

$$ head_1= softmax(\frac{Q_1K_1^T}{\sqrt{d_k}})V_1\ head_2= softmax(\frac{Q_2K_2^T}{\sqrt{d_k}})V_2 $$

- 合并多头输出

$$ multihead = \begin{bmatrix} head_1 & head_2 \end{bmatrix} $$

- 最后进行投影

$$ output = multihead\cdot W^O $$

Self-attention

适用于多向量输入的情形，且输入向量之间是有联系的

因此不能用**FC**作为训练模型，**FC**忽略了向量之间的联系，训练效果会很差

Sequence Labeling （输出输入一对一）

工作示例图：

- 自注意力机制考虑了所有输入向量，然后把整个考虑的结果输出成一个向量给到**FC**进行训练

image-20250418130432286

工作原理

基本思想：自注意力允许模型在处理序列数据时，计算序列中每个位置与所有其他位置之间的关联性

三个关键向量：

- 查询向量$(Query, Q)$
- 键向量(Key, K)
- 值向量$(Value, V)$

计算步骤：

- 对输入序列中的每个元素，通过三个不同的权重矩阵生成$Q、K、V$向量
  - 使用三个不同的权重矩阵进行线性变换：
    - $Q = X × W^Q$
    - $K = X × W^K$
    - $V = X × W^V$

    其中$W^Q、W^K、W^V$□可训练的参数矩阵

- 计算每个位置的查询向量(Q)与所有位置的键向量(K)的点积，获得**attention score**（注意力分数）
  - dot product本质上□测量两个向量之间相似度的方法。当两个向量方向相似时，点积值较大；方向相反时，点积为负；方向垂直时，点积为零
    - 查询向量(Q)相当于**"我想找什么信息"**
    - 键向量(K)相当于**"各个位置提供的信息类型"**

        - 点积结果表示**"这个位置提供的信息与我需要的匹配程度"**

- 对注意力分数进行缩放（除以键向量维度的平方根），主要是方式后续的**softmax**被推入梯度极小的区域，防止梯度消失

- 应用softmax函数（如归一化RELU也可以），将分数转换为概率分布

- 用这些概率加权求和所有位置的值向量(V)，最后算出来的值会被attention score最高的输入所主导
    - 值向量(V)决定位置包含的实际信息内容
    - Q-K点积：确定**"我应该关注哪里"**（计算相关性）
    - V的加权求和：确定**"我应该提取什么信息"（获取内容）**

Multi-head Self-attention（多头注意力）

要有多个查询向量Q，不同的查询向量负责不同种类的相关性

- 计算a^i与其它输入的关联性
    - 计算$b^{i,1}$
        - 根据$q^{i,1}、k^{i,1}、k^{j,1}$计算出$b^{i,1}$
    - 计算$b^{i,2}$
        - 根据$q^{i,2}、k^{i,2}、k^{j,2}$计算出$b^{i,2}$
    - 计算b^{i}
        - 使用$b^{i,1}$与$b^{i,2}$再乘上一个权重矩阵得到b^{i}，得到**attention score**
    - 后面的处理与前面类似

image-20250420112828589

Masked Multi-Head Self-Attention(多头掩码注意力)

自回归模型□一个词一个词生成的，也就□说模型在做推理任务的时候是无法看到后面的词的。

相应地，在训练的过程中，我们会给予模型正确的输出，但□我们并不希望模型看到它还未生成的词，也就□说假设训练数据□句子 `"A B C D"`，模型会一次性看到全部token，但通过掩码限制每个位置只能注意左侧。

自回归掩码（上三角掩码）：通常用于K、V矩阵 $$ \begin{bmatrix} 1 & masked &masked&masked\ 1 & 1 &masked & masked\ 1 &1& 1 &masked\ 1&1&1&1 \end{bmatrix} $$ image-20250702191639250

Word Embedding（词嵌入）

传统文本处理尝使用单热编码（**One-Hot Encoding**）来表示单词。但□这种表达方式无法捕捉单词之间的语义关系，不适合用来词的编码

- 因为每个单词表示的向量都□正交的
- 同时维度会非常大，会耗费计算资源

Positional Encoding

- **Self-attention**的局限：自注意力机制□"置换不变的"，即打乱输入序列顺序后结果不变，这对序列建模□不利的
- 序列顺序的重要性：在语言和其他序列数据中，单词或令牌的顺序包含重要信息，影响意义

绝对位置编码的工作流程

1. 生成位置向量（**positional vector**），每个位置有唯一的位置向量
2. 把这个位置向量直接加到对应位置的输入词嵌入向量上

> ***Final_embedding = Token_embedding + Positional_encoding***

绝对位置编码

给每个位置分配一个单独的向量

常用：可学习式、三角式

三角式

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

1. pos为绝对位置
2. $2i$为维度下标，$2i\le d_{model}$
3. d_{model}为模型维度，也就□每个词或者位置会被编码为d_{model}维的向量

- 比如I am a kid
  - 绝对位置
    - I的pos为0
    - am的pos为1，其它以此类推
  - 维度下标（对于am，pos=1）
    - i=0
      - 第0维：$PE_{(1,0)} = sin(1/10000^{0/d_{model}})$
      - 第1维：$PE_{(1,1)}=cos(1/10000^{0/d_{model}})$
    - i=1
      - 第2维：$PE_{(1,2)}=sin(1/10000^{2/d_{model}})$
      - 第3维：$PE_{(1,3)}=cos(1/10000^{2/d_{model}})$

为每个位置分配一个向量，通过一个二维旋转矩阵引入相对位置信息

- 编码因子：$w_i = 10000^{\frac{2i}{d_{model}}}$
  - 指数级频率变化
    - 在表达式中，pos/w_i为频率，而编码因子以10000为底数，不同维度的波长呈现至少级变化，允许位置编码在非常宽的频率范围内分布，因为编码因子可以看作□指数函数（后续给出证明）
    - 这种特性允许模型捕捉长距离的和短距离的依赖关系（词和词之间），对于任意长度的序列都适用，有效解决了LSTM里面长序列遗忘的问题
  - 平滑频率变化
    - 模型需要处理高维特征，d_{model}越大，说明w_i的增长就越慢，从而导致$10000^{2i/d_{model}}$增长缓慢，因而相对于位置pos变化，$ pos/10000^{2i/d_{model}}$变化得更慢，导致频率变化变慢
    - 这种情况可以导致相邻位置编码差异较小，让模型能够捕捉到位置连续性和顺序性

三角式编码的特性

$$ \underbrace{\begin{bmatrix} PE_{(pos + \Delta,2i)} \ PE_{(pos + \Delta,2i+1)} \end{bmatrix}}_{位置(pos + \Delta)处的编码}

\underbrace{ \begin{bmatrix} cos(\Delta \theta_i) & sin(\Delta \theta_i)\ -sin(\Delta \theta_i) & cos(\Delta \theta_i) \end{bmatrix}}{相□位置信息} \underbrace{ \begin{bmatrix} PE{(pos ,2i)} \ PE_{(pos ,2i+1)} \end{bmatrix}}_{位置pos处的编码} $$

其中: Δ为绝对位置之差, $\theta_i = \frac{1}{10000^{\frac{2i}{d_{model}}}}$

而 $$ \underbrace{ \begin{bmatrix} cos(\Delta \theta_i) & sin(\Delta \theta_i)\ -sin(\Delta \theta_i) & cos(\Delta \theta_i) \end{bmatrix}}_{顺时针旋转\Delta\theta_i} $$

□一个旋转矩阵，表示顺时针旋转$\Delta\theta_i$（与具体的位置**pos**无关）

image-20250701224305593

下面我们来证明一下 $$ \begin{align} \begin{bmatrix} PE_{(pos + \Delta,2i)} \ PE_{(pos + \Delta,2i+1)} \end{bmatrix}&=\begin{bmatrix} sin((pos+\Delta)\cdot\theta_i) \ cos((pos+\Delta)\cdot\theta_i) \end{bmatrix}\ &=\begin{bmatrix} sin(pos\cdot\theta_i)cos(\Delta\cdot\theta_i)+cos(pos\cdot\theta_i)sin(\Delta\cdot\theta_i) \ cos(pos\cdot\theta_i)cos(\Delta\cdot\theta_i)-sin(pos\cdot\theta_i)sin(\Delta\cdot\theta_i) \end{bmatrix}\ &=\begin{bmatrix} cos(\Delta \theta_i) & sin(\Delta \theta_i)\ -sin(\Delta \theta_i) & cos(\Delta \theta_i) \end{bmatrix} \begin{bmatrix} sin(pos\cdot\theta_i)\ cos(pos\cdot\theta_i) \end{bmatrix}\ &=\begin{bmatrix} cos(\Delta \theta_i) & sin(\Delta \theta_i)\ -sin(\Delta \theta_i) & cos(\Delta \theta_i) \end{bmatrix} \begin{bmatrix} PE_{(pos,2i)}\ PE_{(pos,2i+1)} \end{bmatrix}

\end{align} $$

Transformer能够捕捉长短距离依赖的数学原因

已知: $$ \begin{align}

w_i&= 10000^{2i/d_{model}} \ \theta_i &= pos/10000^{2i/d_{model}} \ &= pos / w_i

\end{align} $$

- i小: w_i小，频率高，对应词向量低维度，θ_i对pos变化敏感，捕捉短距离依赖
- i大: w_i大，频率低，对应词向量高维度，θ_i对pos变化不敏感，对于长序列的词，位置编码的值不会变成0，例如$\Delta pos=1000$，$\frac{1000}{10000} = 0.1$，能捕捉长距离依赖
- 如果w_i为10000，$sin(pos/10000)$需pos变化20000π才重复周期，出现相同编码值，混淆文档开头和结尾的词，但□在日常使用并不会出现如此长序列的词（这种方式也决定了它的使用上限）

相对位置编码

- 传统注意力分数的计算□通过绝对位置编码注入到输入嵌入中，QK^T仅仅计算词的内容相关性，而忽略了位置信息，长序列中，绝对位置编码可能因周期性重复导致混淆

- 相对位置编码不考虑绝对位置，在内积中融入相对位置**(query和key的位置差)信息，□注意力得分直接**感知到Q和K的位置差，考虑位置信息

相对位置编码来源于绝对位置编码，我们先来推导下绝对位置编码下Q和K的表达形式 $$ q_i = W_q(x_i+p_i)\ k_j=W_k(x_j+p_j) $$ 二者做内积 $$ q_i^Tk_j = (x_i+p_i)^TW_q^TW_k(x_j+p_j)\ =\underbrace{x_i^TW_q^TW_kx_j}{输入向量内积}+\underbrace{x_i^TW_q^TW_kp_j+p_i^TW_q^TW_kx_j}{输入-位置交互项}+\underbrace{p_i^TW_q^TW_kp_j}_{位置编码内积} $$ 假设位置信息和输入信息相互独立，上式可化成

$$ q_i^Tk_j =x_i^TW_q^TW_kx_j + \underbrace{p_i^TW_q^TW_kp_j}{相□位置项} \ = x_i^TW_q^TW_kx_j +\underbrace{\beta{i-j}}{相□位置项} $$ 而$\beta{i-j}$□偏置项，也就□相对位置编码，在多头注意力中，每个注意力头$head_h$都会分配一组偏置项β_{i-j}^h(why??/)

T5（可学习偏置）

引入分桶处理的思想：不同的位置差按照大小会被分配到不同的桶内

写代码再细学

ALibi（无需训练的线性偏置）

直接在QK内积上加上一个不用训练的偏置项，这个偏置项□位置差矩阵乘以m

image-20250702192315616

其中：m□每个注意力头的斜率，$m_h=2^{-\frac{8\times h}{n_head}}$，$n_{head}$为多头注意力的头数 $$ m=\begin{bmatrix}2^{-\frac{8\times 1}{n_head}}\2^{-\frac{8\times 2}{n_head}}\ \cdots\2^{-\frac{8\times n_{head}}{n_head}}\end{bmatrix} $$ 位置差矩阵□一个下三角矩阵，位置差矩阵$D_{ij} = -(i - j)$

最后效果□ $$ softmax(q_iK^T+m\cdot[-(i-1),\dots,-2,-1,0]) $$

RoPE（旋转位置编码）

结合了旋转位置编码和相对位置编码，对每个**Q**和**K**左乘一个旋转矩阵

在正常的注意力分数计算过程当中，我们□这么计算的 $$ Attention_score = q_m\cdot k_n^T $$ 但□为了融入相对位置信息，我们让q和k分别乘上一个旋转矩阵 $$ q^{rot} = q_m\cdot R_m \ k^{rot} = k_n \cdot R_n $$ 所以：$$ \begin{aligned} q^{rot}\cdot k^{rot} &= q_mR_mR_n^Tk_n^T \ &= q_mR_mR_{-n}k_n^T \ &= q_mR_{m-n}k_n^T \end{aligned} $$ 可见其将位置差信息融入$Q、K$矩阵，同时由于**RoPE**的特性，模型可以外推自己的上下文能力并且还保持一定精度

1. 旋转矩阵周期性

   对于过大的相对位置模型可能从未见过，但□由于旋转矩阵具有周期性，可以使这个相对位置落在训练过的范围内，从而增强泛化能力

2. 线性位置差

无论绝对位置 m,n 多大，注意力分数仅依赖相对位置$m-n$

旋转矩阵的构造

- 对于维度为d的向量，RoPE将向量分成$d/2$组，每组应用一个二维旋转矩阵，**d**一般都是偶数

$$ R_{\theta,m} = \begin{bmatrix} \cos m\theta_{1} & -\sin m\theta_{1} & 0 & \cdots & 0 \ \sin m\theta_{1} & \cos m\theta_{1} & 0 & \cdots & 0 \ 0 & 0 & \cos m\theta_{2} & -\sin m\theta_{2} & 0 \ 0 & 0 & \sin m\theta_{2} & \cos m\theta_{2} & 0 \ \vdots & \vdots & \vdots & \ddots & \vdots \ \end{bmatrix} $$

- 每组二维向量独立乘以一个二维旋转矩阵

$$

\begin{bmatrix} x_{i}' \ x_{i+1}' \end{bmatrix}

\begin{bmatrix} \cos m\theta_i & -\sin m\theta_i \ \sin m\theta_i & \cos m\theta_i \end{bmatrix} \begin{bmatrix} x_i \ x_{i+1} \end{bmatrix}

$$

已知位置差$(m-n)$的旋转矩阵：

这里$\theta_i = \frac{1}{10000^{\frac{2i}{d_{model}}}}$ $$ R_{m-n} = \begin{bmatrix} cos((m-n) \theta_i) & sin((m-n) \theta_i)\ -sin((m-n) \theta_i) & cos((m-n) \theta_i) \end{bmatrix} $$ 可以分解 $$ R_n^TR_m = R_{m-n} $$ 由旋转矩阵性质可知：$$ R_{-m} = R^T_m $$

再以内积形式呈现 $$ (R_nq)^TR_mk=q^TR_{m-n}k $$ 所以有 $$ q_n^{rot} = R_nq\ k_m^{rot} = R_mk $$ 可见其将位置差信息融入$Q、K$矩阵

KV Cache

- 主要应用于推理阶段
- 只存在于解码器中
- 目的□为了加速$Q、K、V$相乘速度
- 但也会加大内存占用

出现这个技术的原因□在自回归模型中，模型生成的□一个接一个的token。模型每次都要把预测输出的文字序列重新丢到模型里面计算，那么就要重新计算$K、V$，浪费计算资源。

比如：

- 我□
- 我□中
- 我□中国
- 我□中国人

如果不使用KV Cache进行缓存，模型就要重复计算“我□”这两个词的$K、V$

不适用**KV Cache**

image-20250702191639250

使用**KV Cache**

image-20250702191137599

Pre-Norm与Post-Norm

Pre-Norm（前归一化）和**Post-Norm**（后归一化）□Transformer模型中两种不同的归一化策略。他们的主要区别在于LN（Layer Normalization）的位置不同。

- 前归一化：在自注意力模块或者前馈网络之前进行层归一化，这种结构在训练时更容易且较为稳定
- 后归一化：在自注意力模块或者前馈网络之后进行层归一化。原始的Transformer使用的□这个，可以帮助稳定梯度，但需要更加谨慎地调节参数，如**学习率预热(warm-up)**。
- 目前比较主流的方法□前归一化，因为其训练起来比较稳定