

位置编码

【通俗易懂-大模型的关键技术之一:旋转位置编码rope (3) -哔哩哔哩】 <https://b23.tv/H6gSsae>

绝对位置编码-transformer

$$p_{i,2t} = \sin\left(\frac{i}{10000^{\frac{2t}{d}}}\right)$$
$$p_{i,2t+1} = \cos\left(\frac{i}{10000^{\frac{2t}{d}}}\right)$$
$$x'_i = (x_i + p_i)$$

```
import torch
import math

def get_position_encoding(seq_len, d_model):
    position_encoding = torch.zeros(seq_len, d_model)
    position = torch.arange(0, seq_len, dtype=torch.float).unsqueeze(1)

    div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-math.log(10000.0) /
d_model))

    position_encoding[:, 0::2] = torch.sin(position * div_term)
    position_encoding[:, 1::2] = torch.cos(position * div_term)

    return position_encoding
```

可学习位置编码-Bert

$$x'_i = (x_i + p_{wi}) \quad PW$$

512 × 768

bert 512

512个位置
每个位置用768维的向量表示

旋转位置编码ROPE

优缺点

优点：RoPE以绝对位置编码的方式实现了相对位置编码，使得能够在不破坏注意力形式的情况下，以“加性编码”的方式让模型学习相对位置。

①相比其他相对位置编码来说，实现简单，计算量少。

②可以应用于线性注意力。

③RoPE具有**远程衰减**的特性，使得每个位置天然能够更关注到附近的信息。

缺点：RoPE相比训练式的绝对位置编码具有一定的外推能力，如可以在2k数据长度训练的模型进行略长于2k的推理。但是相比于Alibi等位置编码，其直接外推能力并不算特别好，需要通过**线性插值**、**NTK插值**、**YaRN**等方式来优化外推能力。

公式

以token级别的公式来表示：

x'_i 表示加上位置编码之后的 x ：

$$x'_i = f(x, i)$$

$$x'_m = W_q x_m e^{im\theta} = (W_q x_m) e^{im\theta} = q_m e^{im\theta}$$

$$x'_n = W_k x_n e^{in\theta} = (W_k x_n) e^{in\theta} = k_n e^{in\theta}$$

假设词向量只有2维

$$x_m^T x'_n = (q_m^1 \ q_m^2) \begin{pmatrix} \cos((m-n)\theta) & -\sin((m-n)\theta) \\ \sin((m-n)\theta) & \cos((m-n)\theta) \end{pmatrix} \begin{pmatrix} k_n^1 \\ k_n^2 \end{pmatrix}$$

证明过程可以先不管（因为我也没看懂）

代码



望舒同学

旋转位置编码Rope - 代码实现详解

哔哩哔哩

$$\theta_i = 10000^{-2i/d}, i \in [1, 2, \dots, \frac{d}{2}]$$

```
theta = 1. / (self.base ** (torch.arange(0, self.d, 2).float() / self.d)).to(x.device)
10000
```

获得一句话
中每个字的顺序

```
[0, 1, ..., seq_len - 1]
```

```
seq_idx = torch.arange(seq_len, device=x.device).float().to(x.device)
```

将 θ_i 与顺序融合



$m\theta_i$

```
idx_theta = torch.einsum('m,d->md', seq_idx, theta)
```

对于第 m 个字,

$m\theta_0, m\theta_1 \dots m\theta_{\frac{d}{2}}, m\theta_0, m\theta_1 \dots m\theta_{\frac{d}{2}}$

```
idx_theta2 = torch.cat([idx_theta, idx_theta], dim=1)
```

```
cos_cached = idx_theta2.cos()[:, None, None, :]
sin_cached = idx_theta2.sin()[:, None, None, :]
```

一定需要按维度顺序两两组合
旋转么？

不一定要严格按照相邻两两组合旋转
神经元是无序的，不依赖维度顺序

$$\begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ \vdots \\ q_{d/2-1} \end{pmatrix} \otimes \begin{pmatrix} \cos m\theta_0 \\ \cos m\theta_0 \\ \cos m\theta_1 \\ \cos m\theta_1 \\ \vdots \\ \cos m\theta_{d/2-1} \\ \cos m\theta_{d/2-1} \end{pmatrix} + \begin{pmatrix} -q_1 \\ -q_3 \\ q_2 \\ \vdots \\ -q_{d-1} \\ q_{d-2} \end{pmatrix} \otimes \begin{pmatrix} \sin m\theta_0 \\ \sin m\theta_0 \\ \sin m\theta_1 \\ \sin m\theta_1 \\ \vdots \\ \sin m\theta_{d/2-1} \\ \sin m\theta_{d/2-1} \end{pmatrix}$$

Chatglm 中

```
def rotate_half(x):
    """Rotates half the hidden dims of the input."""
    x1 = x[::2, : x.shape[-1] // 2]
    x2 = x[1::2, : x.shape[-1] // 2]
    return torch.cat((-x2, x1), dim=-1)
```

```
x_rope, x_pass = x[::2, : self.d], x[1::2, : self.d]
```

```
neg_half_x = rotate_half(x_rope)
```

```
x_rope = (x_rope * cos_cached[:x.shape[0]]) + (neg_half_x * sin_cached[:x.shape[0]])
```

$$\begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ \vdots \\ -q_{d/2} \\ \vdots \\ -q_{d-2} \\ -q_{d-1} \end{pmatrix} \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ \vdots \\ q_{d/2} \\ \vdots \\ q_{d-1} \\ q_{d-2} \end{pmatrix}$$

BY1M421R7JQ 04:43/04:54

生成旋转矩阵

```
def precompute_freqs_cis(dim: int, seq_len: int, theta: float = 10000.0):
```

计算词向量元素两两分组之后，每组元素对应的旋转角度\theta_a_i

```
freqs = 1.0 / (theta ** (torch.arange(0, dim, 2)[: (dim // 2)].float() / dim))
```

生成 token 序列索引 t = [0, 1, ..., seq_len-1]

```
t = torch.arange(seq_len, device=freqs.device)
```

freqs.shape = [seq_len, dim // 2]

```
freqs = torch.outer(t, freqs).float() # 计算m * \theta_a
```

计算结果是个复数向量

```

# 假设 freqs = [x, y]
# 则 freqs_cis = [cos(x) + sin(x)i, cos(y) + sin(y)i]
freqs_cis = torch.polar(torch.ones_like(freqs), freqs)
return freqs_cis

# 旋转位置编码计算
def apply_rotary_emb(
    xq: torch.Tensor,
    xk: torch.Tensor,
    freqs_cis: torch.Tensor,
) -> Tuple[torch.Tensor, torch.Tensor]:
    # xq.shape = [batch_size, seq_len, dim]
    # xq_.shape = [batch_size, seq_len, dim // 2, 2]
    xq_ = xq.float().reshape(*xq.shape[:-1], -1, 2)
    xk_ = xk.float().reshape(*xk.shape[:-1], -1, 2)

    # 转为复数域
    xq_ = torch.view_as_complex(xq_)
    xk_ = torch.view_as_complex(xk_)

    # 应用旋转操作, 然后将结果转回实数域
    # xq_out.shape = [batch_size, seq_len, dim]
    xq_out = torch.view_as_real(xq_ * freqs_cis).flatten(2)
    xk_out = torch.view_as_real(xk_ * freqs_cis).flatten(2)
    return xq_out.type_as(xq), xk_out.type_as(xk)

```

Alibi

使用于百川7B（我也没搞懂）