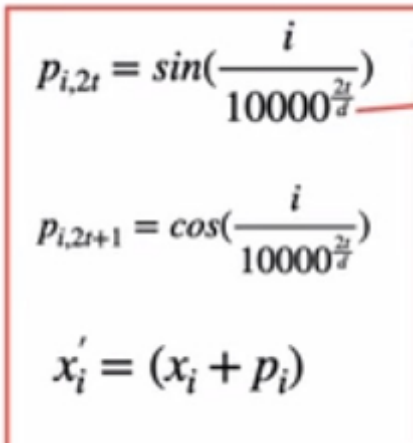
位置编码

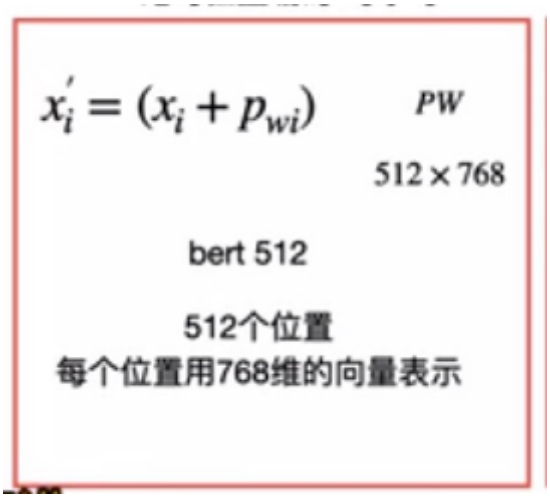
【通俗易懂-大模型的关键技术之一:旋转位置编码rope （3）-哔哩哔哩】 https://b23.tv/H6gSsae

绝对位置编码-transformer




```
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



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：

$$x_i^{'} = f(x, i)$$

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

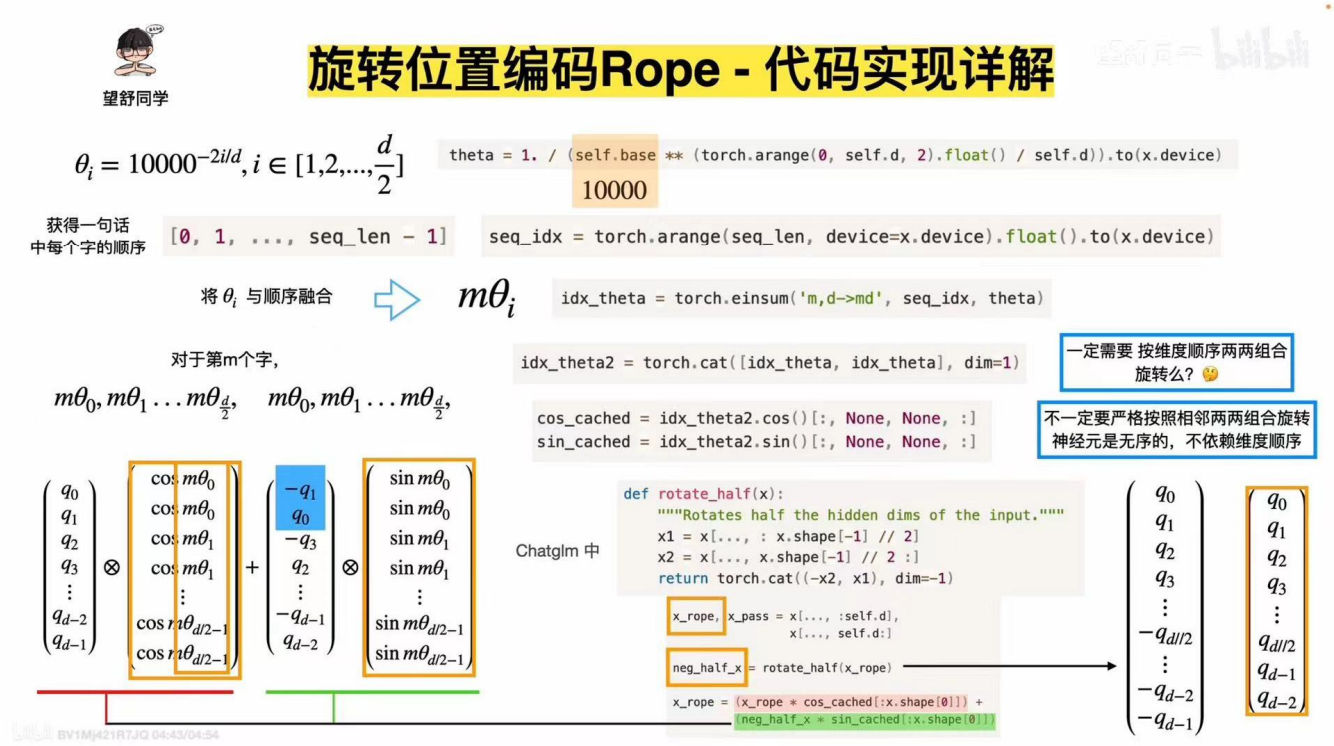
$$x_n' = W_k x_n e^{in\theta} = (W_k x_n) e^{in\theta} = k_n e^{in\theta}$$

假设词向量只有2维

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

证明过程可以先不管（因为我也没看懂）

代码




```python
# 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_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

    # 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）