llama3-RoPE源码

核心函数：precompute_freqs_cis

reshape_for_broadcast

apply_rotary_emb

precompute_freqs_cis:

```
def precompute_freqs_cis(dim: int, end: int, theta: float = 10000.0):
    freqs = 1.0 / (theta ** (torch.arange(0, dim, 2)[: (dim // 2)].float() / dim))#计算theta_i
    t = torch.arange(end, device=freqs.device) #创建seq_len长度的序号并和freqs放在同一gpu/cpu上
    freqs = torch.outer(t, freqs).float()  #对t和freqs作外积，结果矩阵形状为（seq_len,head_dim/2）也就是下图矩阵
    freqs_cis = torch.polar(torch.ones_like(freqs), freqs)  # 传入幅度（1）和角度（freqs）参数， torch.polar函数将极坐标表示的复数转换为直角坐标表示的复数
    return freqs_cis #复数形式

"================================================================="

#使用freq_cis时：
 #从预先计算好的freqs_cis中取出现在正在计算的位置对应的freqs_cis
        self.freqs_cis = self.freqs_cis.to(h.device)
        freqs_cis = self.freqs_cis[start_pos : start_pos + seqlen]
```

$$\boldsymbol{R}^d_{\Theta,m} = \underbrace{\begin{pmatrix} \cos m\theta_0 & -\sin m\theta_0 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_0 & \cos m\theta_0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_1 & -\sin m\theta_1 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_1 & \cos m\theta_1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2-1} & -\sin m\theta_{d/2-1} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2-1} & \cos m\theta_{d/2-1} \end{pmatrix}}_{\boldsymbol{W}_m}$$

$$\Theta = \left\{\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, \ldots, d/2]\right\}$$

reshape_for_broadcas:

```
#就是为了让freqs_cis和xq，xk维度匹配，能进行广播操作，结果形状变为
[batch_size,seq_len,1,head_dim/2]
def reshape_for_broadcast(freqs_cis: torch.Tensor, x: torch.Tensor):
    ndim = x.ndim#获取目标张量 x 的维度数。
    assert 0 <= 1 < ndim#确保 x 至少有2个维度（因为 freqs_cis 需要与 x 的第二维和最后一维匹配）
    assert freqs_cis.shape == (x.shape[1], x.shape[-1])#确保 freqs_cis 的形状与 x 的第二维和最后一维相匹配。这是为了确保 freqs_cis 可以与 x 在这些维度上进行广播。
    shape = [d if i == 1 or i == ndim - 1 else 1 for i, d in enumerate(x.shape)]#创建一个新的形状列表，其中 x 的第二维和最后一维保持不变，其他维度都是1。这样做是为了确保 freqs_cis 在重塑后可以在这些维度上与 x 广播。
    return freqs_cis.view(*shape)#使用 view 方法将 freqs_cis 重塑为新的形状，并返回。
```

apply_rotary_emb:

```
def apply_rotary_emb(
    xq: torch.Tensor,
    xk: torch.Tensor,
    freqs_cis: torch.Tensor,
) -> Tuple[torch.Tensor, torch.Tensor]:
    """
此函数使用提供的频率张量 `freqs_cis` 对给定的查询 `xq` 和键 `xk` 张量应用旋转嵌入。输入张量被重塑为复数，并且频率张量被重塑以兼容广播。生成的张量包含旋转嵌入，并且作为实数张量返回。
返回：
- Tuple[torch.Tensor, torch.Tensor]：包含旋转嵌入的修改后的查询张量和键张量的元组。
    """
    #torch.view_as_complex将实数张量转换为复数张量，*xq.shape[:-1]将列表拆开，比如[2,4,6]->2,4,6
    #-1, 2)，将原本最后一维拆成两维，（dim/2,2）
    #reshape完维度增加了一维，并且最后一维的大小为2
    #torch.view_as_complex将最后一维中两个元素分别作为实部和虚部转换成复数
    xq_ = torch.view_as_complex(xq.float().reshape(*xq.shape[:-1], -1, 2))
    xk_ = torch.view_as_complex(xk.float().reshape(*xk.shape[:-1], -1, 2))
    #对角度矩阵进行扩维，为了匹配xq，xk矩阵,形状变为[batch_size,seq_len,1,head_dim/2]
    freqs_cis = reshape_for_broadcast(freqs_cis, xq_)
    xq_out = torch.view_as_real(xq_ * freqs_cis).flatten(3)#torch.view_as_real将复数转换为实属矩阵形式，最后一维度含两个元素分别是原来复数的实部和虚部
    """
    tensor([(0.4737-0.3839j), (-0.2098-0.6699j), (0.3470-0.9451j), (-0.5174-1.3136j)])
>>> torch.view_as_real(x)
tensor([[ 0.4737, -0.3839],
        [-0.2098, -0.6699],
        [ 0.3470, -0.9451],
        [-0.5174, -1.3136]])"""
    #(xk_ * freqs_cis)通过乘旋转矩阵进行旋转，由于freqs_cis的模是1，所以总模不变，只改变角度，角度改变大小为m*theta_i
    xk_out = torch.view_as_real(xk_ * freqs_cis).flatten(3)
    return xq_out.type_as(xq), xk_out.type_as(xk)
```