llama3源码-transformer

在已经构造好了Attention类，RMSNorm类，以及RoPE所需的各种函数后，可以开始进行transformer模型的搭建

在搭建之前，我们还需要FeedForward类，TransformerBlock类

FeedForward：

作用：引入非线性，增加模型深度，提高表达能力等

```
#前馈神经网络层（全连接），用了三个linear层
class FeedForward(nn.Module):
    def __init__(
        self,
        dim: int,# 输入维度
        hidden_dim: int, # 前馈层的隐藏维度
        multiple_of: int,# 确保隐藏维度是这个值的倍数
        ffn_dim_multiplier: Optional[float],#隐藏维度的自定义乘数。默认为 None
    ):
        #调用父类init就是为了注册模块，来确保所有层的统一性(参数管理、梯度计算)
        super().__init__()
        #缩小hidden_dim大小为2/3
        hidden_dim = int(2 * hidden_dim / 3)
        # 用户自定义hidden_dim的乘数
        if ffn_dim_multiplier is not None:
            hidden_dim = int(ffn_dim_multiplier * hidden_dim)
        #要确保自定义的hidden_dim大小是multiple_of的倍数，而且不能比之前的hidden_dim小
        hidden_dim = multiple_of * ((hidden_dim + multiple_of - 1) //
multiple_of)

        #定义三层linear，按列还是按行并行就是取决于是升维还是降维映射
        #ColumnParallelLinear升维映射
        self.w1 = ColumnParallelLinear(
            dim, hidden_dim, bias=False, gather_output=False, init_method=lambda
x: x
        )
        #RowParallelLinear降维映射
        self.w2 = RowParallelLinear(
            hidden_dim, dim, bias=False, input_is_parallel=True,
init_method=lambda x: x
        )
        self.w3 = ColumnParallelLinear(
            dim, hidden_dim, bias=False, gather_output=False, init_method=lambda
x: x
        )
    #用F.silu激活函数，silu(x)=x*sigmoid(x),此处F.silu(self.w1(x))相当于是self.w3(x)
的缩放因子
    def forward(self, x):
        return self.w2(F.silu(self.w1(x)) * self.w3(x))
```

TransformerBlock:

transformerblock作为构造transformer模型的基本块，包括了RMSNorm归一化（pre-norm），attention计算（包含RoPE），feedforward模块

```
class TransformerBlock(nn.Module):
    #接受layer_id(第几层)和ModelArgs
    def __init__(self, layer_id: int, args: ModelArgs):

        super().__init__()
        #总query头数
        self.n_heads = args.n_heads
        #model_dim
        self.dim = args.dim
        #每个头的dim
        self.head_dim = args.dim // args.n_heads
        #实例化Attention类
        self.attention = Attention(args)
        #实例化FeedForward类
        self.feed_forward = FeedForward(
            dim=args.dim,
            hidden_dim=4 * args.dim,
            multiple_of=args.multiple_of,
            ffn_dim_multiplier=args.ffn_dim_multiplier,
        )
        self.layer_id = layer_id
        #实例化两种RMSNorm类
        # 注意力输出的层归一化
        self.attention_norm = RMSNorm(args.dim, eps=args.norm_eps)
        # 前馈输出的层归一化
        self.ffn_norm = RMSNorm(args.dim, eps=args.norm_eps)
    def forward(
        self,
        x: torch.Tensor,#输入词向量
        start_pos: int,#开始推理位置
        freqs_cis: torch.Tensor,#位置编码角度矩阵
        mask: Optional[torch.Tensor],#注意力的mask矩阵（可选），默认为None
    ):
        #将x进行RMSnorm之后再进行attention（pre-Norm），然后进行残差连接
        h = x + self.attention(
            self.attention_norm(x), start_pos, freqs_cis, mask
        )
        #ffn时也会进行RMSnorm归一化（pre-Norm）然后再和h残差连接，ffn层是一个两头窄中间宽的形式(hidden_dim大)，
        out = h + self.feed_forward(self.ffn_norm(h))
        return out
```

最终可以开始搭建模型：

Transformer:

```
class Transformer(nn.Module):
    def __init__(self, params: ModelArgs):
```

```
        super().__init__()
        #把超参数赋值给self.params
        self.params = params
        #词典大小
        self.vocab_size = params.vocab_size
        #transformer层数(transformerblock数量)
        self.n_layers = params.n_layers
        #词嵌入向量，用ParallelEmbedding保证可以并行，从单词表大小映射到model_dim
        self.tok_embeddings = ParallelEmbedding(
            params.vocab_size, params.dim, init_method=lambda x: x
        )
        #定义一个torch.nn.ModuleList()容器，存放层，存在里面的层的参数都被当作可训练的
        self.layers = torch.nn.ModuleList()
        #向self.layers中加入params.n_layers个层(transformerblock)
        for layer_id in range(params.n_layers):
            self.layers.append(TransformerBlock(layer_id, params))

        #实例化RMSNorm类
        self.norm = RMSNorm(params.dim, eps=params.norm_eps)
        #定义一个输出层，把transformer的输出映射到单词表空间上，ColumnParallelLinear按列并行的全连接输出层
        #从model_dim映射到单词表
        self.output = ColumnParallelLinear(
            params.dim, params.vocab_size, bias=False, init_method=lambda x: x
        )
        #
        self.freqs_cis = precompute_freqs_cis(
            # Note that self.params.max_seq_len is multiplied by 2 because the token limit for the Llama 2 generation of models is 4096.
            # Adding this multiplier instead of using 4096 directly allows for dynamism of token lengths while training or fine-tuning.
            self.params.dim // self.params.n_heads, self.params.max_seq_len * 2 #具体传参值：dim：模型维度除以头的数量，end:4096*2，self.params.max_seq_len * 2是为了可扩展性
        )
    #forward是真正来做推理的，tokens输入的token，start_pos推理到第几步(为了KVcache和RoPE)
    @torch.inference_mode()
    def forward(self, tokens: torch.Tensor, start_pos: int):
        #历史已经有的tokens，提取他的形状
        _bsz, seqlen = tokens.shape
        #进行词嵌入
        h = self.tok_embeddings(tokens)
        #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)
        #位置相关，取[start_pos : start_pos + seqlen]部分
        freqs_cis = self.freqs_cis[start_pos : start_pos + seqlen]

        mask = None
        #如果这一步forward只推理一个token，就不用mask，如果一次推理多个（seqlen个）则需要mask(下三角)
        if seqlen > 1:
            #创建一个大小为(seqlen, seqlen)，填充值为float("-inf")的mask矩阵
            mask = torch.full(
                (seqlen, seqlen), float("-inf"), device=tokens.device
            )
```

```
            #用于创建一个上三角矩阵。input是输入的矩阵，diagonal参数指定了对角线的偏移量，diagonal=1意味着对角线以上的元素保持不变，对角线及以下的元素被设置为零
            mask = torch.triu(mask, diagonal=1)

            #对一个掩码矩阵进行水平堆叠（horizontal stack）操作（沿着第二维），以扩展掩码矩阵的大小，并确保新的掩码矩阵与特定的数据类型一致
            #torch.zeros((seqlen, start_pos)构造全零矩阵，来和之前构造的mask矩阵水平拼接
            #拼接结果形状(seqlen,start_pos+seqlen),其中前start_pos列值都为0，表示不mask掉之前生成的，从start_pos到seqlen列值为一个下三角矩阵表示要mask，因为要一次生成seqlen个token
            mask = torch.hstack([
                torch.zeros((seqlen, start_pos), device=tokens.device),
                mask
            ]).type_as(h)
        #不断forloop层，不断传递h, start_pos, freqs_cis, mask
        for layer in self.layers:
            h = layer(h, start_pos, freqs_cis, mask)
        #最后的全连接层之前进行RMSnorm
        h = self.norm(h)
        output = self.output(h).float()
        return output
```