

KV Cache初探 从MHA、MQA到GQA

东川路第一可爱猫猫虫



主要内容

- 追根溯源: 什么是KV Cache
- 如何优化KV Cache: MHA → MQA → GQA



LLM的本质： next token prediction

- 自回归模型

AR autoregressive $x_{t+1} = f(x_t, x_{t-1}, \dots, x_{t-p+1})$

根据前一段时间的数据预测下一时刻的数据

- 自回归LLM

each token conditions on its past

因果掩码 causal mask

屏蔽未来的词

这样的attention称为casual attention

举例说明计算过程

- 先来看单头注意力是怎么做next token prediction的
- 我们用 x_1, \dots, x_t 生成 x_{t+1}

最后一个已知token x_t 经过矩阵运算生成 q_t , 前 t 个 x_i 经过矩阵运算生成 k_i, v_i , q_t 与每个 k_i 分别求内积, 算出权重, 对 v_i 加权求和, 这就是单头注意力学习到的上下文

将其输出记作 o_t

- 再来看多头注意力

h 个注意力头的多头注意力就是重复上述操作 h 次, 再拼接

多头注意力的输出为 $o_t = [o_t^{(1)}, o_t^{(1)}, \dots, o_t^{(h)}]$

KV Cache

- 如果生成一个长度为1000的token序列，当我们生成第1000个token时，第一个token的k和v已经被计算了999次
这造成了巨大的重复计算浪费
解决办法就是把以前计算好的k和v缓存起来，以供后续调用
这就是**KV Cache**
- 为什么不缓存Q?
Q没有被重复计算，每次只算当前的最后一个已知token的Q

KV Cache遇到了问题

- LLM用GPU
 - GPU每张卡的显存**有限**
 - 因此要对KV Cache进行优化，减小KV Cache
- MQA
 - 《Fast Transformer Decoding: One Write-Head is All You Need》
 - 所有注意力头共享一套K和V
 - 理论上，MQA直接将KV Cache减少到了原来的 $\frac{1}{h}$
 - KV Cache 大幅降低，效果损失也比较有限

折中之选

- GQA

《GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints》

本质上就是把MHA和MQA做了一个折中

将所有头分为 g 个组 (g 可以整除 h)，每组共享同一对K、V

$g=h$ MHA $g=1$ MQA

- Llama3、DeepSeek-V1用的就是GQA

g 取的是8 这考虑到了推理效率

每张卡负责计算一个组内的attention head

