PR #2152 完整报告

THUDM/slime

Optimize memory usage for _VocabParallelLogProbEntropy

合并时间：2026-06-30 18:13

原文链接：http://prhub.com.cn/THUDM/slime/pull/2152

执行摘要

- 一句话：优化 vocab 并行 logprob/entropy 计算，减少内存分配
- 推荐动作：建议精读 slime/utils/ppo_utils.py 中的 in-place 改造和 sum_softmax_logits 设计，是典型的 CUDA 显存优化模式。调用侧 loss.py 的改动简单，但测试容差变化值得关注。

功能与动机

原实现中 softmax 计算会分配多个 [seq_len, vocab] 中间张量（logits_max, normalized_logits, exp_logits, softmax），熵计算使用 (softmax * logits).sum 也产生大乘积张量，导致训练显存压力。PR 通过 in-place 操作和更紧凑的 CUDA einsum 减少中间缓冲。

实现拆解

1. 新增 with_entropy_grad 参数及条件反向：在 _VocabParallelLogProbEntropy.forward 新增布尔参数 with_entropy_grad，并在入口 ctx.set_materialize_grads(False) 避免不必要的梯度张量物化；只有当 with_entropy=True 且 with_entropy_grad=True 时才保留熵反向计算图。
2. 重构 vocab_parallel_softmax 为 in-place 单缓冲模式：新增 inplace 开关，开启时利用 sub_ / exp_ / div_ 覆盖输入张量，只需一次 [seq_len, vocab] 缓冲区。同时前移 predicted_logits 的 gather 操作，在 inplace 破坏原始值之前复制目标位置的 logit。
3. 引入 sum_softmax_logits 函数：针对 CUDA 张量使用 torch.einsum("ij,ij->i", softmax, logits) 避免显式分配 (softmax * logits) 中间张量，仅保留一维结果。
4. 调用端适配：policy_loss_function 传递 with_entropy_grad=args.entropy_coef != 0，当熵系数为零时跳过熵梯度计算。
5. 测试容差放宽：为熵前向和反向设置更宽松的绝对容差（1e-4 / 1e-6），并新增 entropy.requires_grad 断言验证梯度条件。

关键文件：

- slime/utils/ppo_utils.py（模块 训练工具；类别 source；类型 core-logic；符号 _VocabParallelLogProbEntropy.forward, vocab_parallel_softmax, sum_softmax_logits）：核心修改：重构 _VocabParallelLogProbEntropy 前向逻辑，引入 inplace softmax 和 einsum 优化。

- slime/backends/megatron_utils/loss.py（模块 损失计算；类别 source；类型 core-logic；符号 policy_loss_function）：调用侧传入 with_entropy_grad 参数，根据 entropy_coef 是否为零决定是否保留熵梯度。
- tests/test_ppo_logprob_entropy_gpu.py（模块 GPU 测试；类别 test；类型 test-coverage；符号 _assert_legacy_parity）：测试适配：传递 with_entropy_grad 参数，并放宽熵前向 / 反向的绝对容差。

关键符号：_VocabParallelLogProbEntropy.forward, vocab_parallel_softmax, sum_softmax_logits, policy_loss_function, _assert_legacy_parity

关键源码片段

slime/utils/ppo_utils.py

核心修改：重构 _VocabParallelLogProbEntropy 前向逻辑，引入 inplace softmax 和 einsum 优化。

```
class _VocabParallelLogProbEntropy(torch.autograd.Function):
    @staticmethod
    def forward(
        ctx,
        vocab_parallel_logits: torch.Tensor,
        target: torch.Tensor,
        log_prob_keep_mask: torch.Tensor | None,
        process_group,
        with_entropy: bool,
        with_entropy_grad: bool, # 新增：控制是否计算熵梯度
    ) -> tuple[torch.Tensor, torch.Tensor]:
        ctx.set_materialize_grads(False) # 避免不必要的张量物化
        with_entropy_grad = with_entropy and with_entropy_grad
        vocab_parallel_logits = vocab_parallel_logits.float()
        seq_len, vocab_parallel_size = vocab_parallel_logits.shape
        rank, _world_size = _get_vocab_parallel_rank_size(process_group)
        vocab_start_index = rank * vocab_parallel_size
        vocab_end_index = vocab_start_index + vocab_parallel_size

        target_mask = (target < vocab_start_index) | (target >= vocab_end_index)
        masked_target_1d = (target - vocab_start_index).clone()
        masked_target_1d[target_mask] = 0
        arange_1d = torch.arange(seq_len, device=vocab_parallel_logits.device)

        def vocab_parallel_softmax(
            logits: torch.Tensor,
            inplace: bool = False, # 控制是否复用输入缓冲区
        ) -> tuple[torch.Tensor, torch.Tensor, torch.Tensor, torch.Tensor]:
            logits_max = logits.max(dim=-1, keepdim=True).values
            _maybe_all_reduce(logits_max, dist.ReduceOp.MAX, process_group)
            # 当 inplace=True 时直接修改输入张量，避免分配新 buffer
            normalized_logits = logits.sub_(logits_max) if inplace else logits - logits_max
```

```
        # 必须在 inplace 破坏前收集目标位置的 logit（小量拷贝）
        predicted_logits = normalized_logits.view(-1, vocab_parallel_size)[arange_1d, masked_
        target_1d]
        # 复用 normalized_logits 的存储，只需一个 [seq, vocab] 缓冲区
        exp_logits = normalized_logits.exp_()
        sum_exp_logits = exp_logits.sum(dim=-1, keepdim=True)
        _maybe_all_reduce(sum_exp_logits, dist.ReduceOp.SUM, process_group)
        softmax = exp_logits.div_(sum_exp_logits) # 再覆盖为 softmax
        return predicted_logits, sum_exp_logits, softmax, logits_max

    # ... 后续逻辑使用 inplace=True 调用 vocab_parallel_softmax ...

    def sum_softmax_logits(softmax: torch.Tensor, logits: torch.Tensor) -> torch.Tensor:
        # CUDA 上使用 einsum 避免分配 [seq, vocab] 乘积张量
        if softmax.is_cuda:
            return torch.einsum("ij,ij->i", softmax, logits).unsqueeze(-1)
        # CPU 回退
        return (softmax * logits).sum(dim=-1, keepdim=True)
```

评论区精华

无 review 评论，PR 由作者自行合并。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 精度风险：in-place 操作改变了计算顺序，可能导致前向熵值和反向梯度与旧实现存在毫量级差异。测试已放宽容差（ENTROPY_FORWARD_ATOL=1e-4, ENTROPY_BACKWARD_ATOL=1e-6），但若模型对熵项敏感可能出现训练曲线偏移。
  2. einsum 回退：sum_softmax_logits 对非 CUDA 张量回退到逐元素乘法，仅在 CUDA 上走 einsum 通路，CPU 回退路径未测试。
  3. 梯度条件覆盖：with_entropy_grad=False 时熵输出 requires_grad=False，若上层代码错误地依赖其梯度会导致静默错误；测试虽覆盖了 entropy.requires_grad 断言，但只针对 parity 场景。 - 影响：降低所有使用 _VocabParallelLogProbEntropy 的 PPO 训练作业的显存占用（约减少 2-3 个 [seq, vocab] 缓冲区，每个 buffer 大小约 seq_len * vocab_size * 4 byte）。启用熵项但 entropy_coef=0 的场景可节省额外 backward 显存。影响范围限于 PPO 训练路径，不影响推理或 rollout。 - 风险标记：in-place 操作精度风险，einsum 仅在 CUDA 使用，entropy 梯度条件可能遗漏

关联脉络

- PR #2144 perf: fuse PPO logprob entropy computation: 同一功能线的性能优化，修改了相同文件（ppo_utils.py, loss.py, test 文件），且本 PR 在 #2144 基础上进一步做内存优化。

- PR #2153 bugfix: 下移 with_entropy_grad 判断，与本 PR 的 with_entropy_grad 引入直接相关，修改了 loss.py 和 ppo_utils.py。