

PR #3083 完整报告

radixark/miles

fix(megatron): respect --no-save-optim when saving checkpoints

合并时间: 2026-09-15 10:48

原文链接: <http://prhub.com.cn/radixark/miles/pull/3083>

执行摘要

- 一句话: 修复 LoRA 保存忽略 `--no-save-optim` 致恢复静默重启
- 推荐动作: 值得精读, 尤其是 commit 历史中的方案修正过程: 从“跳过整个训练状态文件”改为“只丢弃 optimizer 条目”, 展示了数据契约对 flag 语义的约束。关注点包括: 单文件三元组 (iteration / scheduler / optimizer) 的解耦设计、`getattr(args, "no_save_optim", False)` 的兼容写法、以及测试中伪造 `megatron.bridge` 模块让 HF PEFT 导出快速失败的技巧。建议与 #3245 一起阅读, 形成保存 / 加载闭环。

功能与动机

PR body 明确指出: `--no-save-optim` 在 LoRA 运行保存 checkpoint 时被忽略。LoRA 训练状态是单一文件, 同时承载 iteration、LR scheduler 与 optimizer; 若因 `--no-save-optim` 跳过整个文件, 三者会全部丢失, 而恢复路径没有其他来源: 没有 `training_state_rank*.pt` 时, `load_lora_adapter` 返回 `iteration=None`, HF base 报告 0, adapter 权重能恢复但 step 计数器与 LR schedule 静默重启。

实现拆解

1. 保存侧核心改造 (`miles/backends/megatron_utils/lora_utils.py` 的 `save_lora_checkpoint`): 在训练状态写入处新增 `save_optimizer = not getattr(args, "no_save_optim", False)`, `--no-save-optim` 时 optimizer 键写 None, 而 iteration 与 `opt_param_scheduler` 照常序列化; 日志随之区分 optimizer/scheduler 与 scheduler 两种文案。`getattr` 带默认值是为了兼容未声明该字段的调用方 (如单测中的 Namespace)。
2. 加载侧配套 (`_load_training_state`): 在调用 `optimizer.load_state_dict` 前增加 `training_state.get("optimizer") is not None` 守卫, 避免对新格式 checkpoint (optimizer 为 None) 崩溃, 同时保证 iteration 返回值与 scheduler 恢复逻辑不受影响。
3. 设计权衡: 早期 commit (d8536e65) 尝试直接 gate 文件写入并删除调用点传参, 让 flag 单一所有者; 后续 commit (97702fd) 修正为“仍写文件、只丢 optimizer”——因为 LoRA 训练状态是单文件三元组, 跳过文件等于丢掉恢复路径唯一可靠的 step 与 LR 来源。这属于对“flag 语义”的澄清: `--no-save-optim` 意味着“不保存 optimizer 状态”而非“不写训练状态文件”。
4. 测试与验证配套: 在 `tests/fast/backends/megatron_utils/test_lora_utils.py` 新增 `TestSaveLoraCheckpointTrainingState` 与 `TestLoadTrainingState`, 通过 monkeypatch 伪造 `get_parallel_state` 与 `megatron.bridge` (让 HF PEFT 导出快速失败), 分别验证:

默认写入完整状态；`--no-save-optim` 时 `optimizer` 为 `None` 且 `resume` 元数据保留；无 `optimizer` 的 `checkpoint` 仍能恢复 `step` 与 `scheduler`。PR body 另附 4x H200 端到端对比，修复后恢复日志出现 `Resuming LoRA training from iteration 1` 与 `Restored LRScheduler state`。

关键文件：

- `miles/backends/megatron_utils/lora_utils.py`（模块 LoRA 检查点；类别 `source`；类型 `core-logic`；符号 `save_lora_checkpoint`, `_load_training_state`）：核心修复文件：
`save_lora_checkpoint` 在 `--no-save-optim` 下将 `optimizer` 条目置 `None` 而保留 `iteration` 与 `LR scheduler`；`_load_training_state` 增加 `optimizer` 为 `None` 的守卫，保证新旧 `checkpoint` 均可正确恢复。
- `tests/fast/backends/megatron_utils/test_lora_utils.py`（模块 LoRA 检查点；类别 `test`；类型 `test-coverage`；符号 `TestSaveLoraCheckpointTrainingState`, `test_no_save_optim_drops_the_optimizer_and_keeps_the_resume_metadata`, `TestLoadTrainingState`, `test_an_optimizer_free_checkpoint_still_restores_the_step_and_the_schedule`）：新增两个测试类，分别覆盖训练状态文件的保存契约（默认写入、`--no-save-optim` 只丢 `optimizer`）与加载契约（无 `optimizer` 的 `checkpoint` 仍恢复 `step` 与 `scheduler`），是该行为的回归保护。

关键符号：`save_lora_checkpoint`, `_load_training_state`

关键源码片段

`miles/backends/megatron_utils/lora_utils.py`

核心修复文件：`save_lora_checkpoint` 在 `--no-save-optim` 下将 `optimizer` 条目置 `None` 而保留 `iteration` 与 `LR scheduler`；`_load_training_state` 增加 `optimizer` 为 `None` 的守卫，保证新旧 `checkpoint` 均可正确恢复。

```
# save_lora_checkpoint 中的训练状态写入段（位于 native adapter 与 HF PEFT 导出之后）。
# LoRA 训练状态是单文件三元组：iteration + LR scheduler + optimizer。
# --no-save-optim 只允许丢弃 optimizer 条目；iteration 与 scheduler 是恢复路径
# 获取 step 计数与学习率曲线的唯一来源，连它们一起丢掉会让续训静默重启。
if optimizer is not None:
    # getattr 带默认值：兼容未声明 no_save_optim 的调用方（如单测 Namespace）
    save_optimizer = not getattr(args, "no_save_optim", False)
    rank = dist.get_rank() if dist.is_initialized() else 0
    torch.save(
        {
            "iteration": iteration,
            "optimizer": optimizer.state_dict() if save_optimizer else None,
            "opt_param_scheduler": opt_param_scheduler.state_dict() if opt_param_scheduler else None,
        },
        save_path / f"training_state_rank{rank}.pt",
    )
    logger.info(f"Saved {'optimizer/scheduler' if save_optimizer else 'scheduler'} state to {save_path}")
```

```

def _load_training_state(
    adapter_dir: Path,
    optimizer: Any | None,
    opt_param_scheduler: Any | None,
) -> int | None:
    """从 LoRA adapter checkpoint 目录恢复训练状态。

    与保存侧契约对应: --no-save-optim 写出的 checkpoint 中 optimizer 键为 None,
    这里必须跳过 optimizer 恢复, 但 iteration 与 LR scheduler 仍照常恢复,
    保证续训从正确的 step 与学习率曲线继续。
    """
    if optimizer is None:
        return None

    rank = dist.get_rank() if dist.is_initialized() else 0
    state_path = adapter_dir / f"training_state_rank{rank}.pt"
    if not state_path.exists():
        return None

    # optimizer 状态字典可能含非张量对象 (step 计数、param group 元数据),
    # 因此这里必须完整反序列化 (weights_only=False)
    training_state = torch.load(state_path, map_location="cpu", weights_only=False)

    if training_state.get("optimizer") is not None:
        optimizer.load_state_dict(training_state["optimizer"])
        logger.info("Restored optimizer state from LoRA checkpoint")

    if opt_param_scheduler is not None and training_state.get("opt_param_scheduler") is not None:
        opt_param_scheduler.load_state_dict(training_state["opt_param_scheduler"])
        logger.info("Restored LR scheduler state from LoRA checkpoint")

    iteration = training_state.get("iteration")
    if iteration is not None:
        logger.info(f"Resuming LoRA training from iteration {iteration}")
    return iteration

```

tests/fast/backends/megatron_utils/test_lora_utils.py

新增两个测试类, 分别覆盖训练状态文件的保存契约 (默认写入、`--no-save-optim` 只丢 optimizer) 与加载契约 (无 optimizer 的 checkpoint 仍恢复 step 与 scheduler), 是该行为的回归保护。

```

class TestSaveLoraCheckpointTrainingState:
    def _save(self, tmp_path, monkeypatch, *, no_save_optim, scheduler=None):
        # 用 SimpleNamespace 伪造并行状态, 避免单测里初始化真实分布式环境
        rank0 = SimpleNamespace(rank=0)
        monkeypatch.setattr(
            lora_utils,
            "get_parallel_state",

```

```

        lambda: SimpleNamespace(effective_dp=rank0, cp=rank0, tp=rank0, pp=rank0),
    )
    # HF PEFT 导出依赖真实 Megatron bridge; 这里挂一个快速失败的假模块,
    # 让测试只聚焦 native adapter 与训练状态文件的写入行为
    bridge = types.ModuleType("megatron.bridge")
    bridge.AutoBridge = SimpleNamespace(
        from_hf_pretrained=Mock(side_effect=RuntimeError("no bridge in this test"))
    )
    monkeypatch.setitem(sys.modules, "megatron.bridge", bridge)

    adapter = torch.nn.Parameter(torch.ones(2))
    model = [
        SimpleNamespace(
            named_parameters=lambda: [("layers.0.self_attention.lora_A.weight", adapter)]
        )
    ]
    args = Namespace(
        hf_checkpoint="/nonexistent",
        target_modules=None,
        lora_rank=8,
        lora_alpha=16,
        lora_dropout=0.0,
        no_save_optim=no_save_optim,
    )
    optimizer = SimpleNamespace(state_dict=lambda: {"step": 7})
    save_lora_checkpoint(
        model, args, str(tmp_path), optimizer=optimizer,
        opt_param_scheduler=scheduler, iteration=3,
    )
    return sorted(path.name for path in tmp_path.iterdir())

```

```

def test_no_save_optim_drops_the_optimizer_and_keeps_the_resume_metadata(self, tmp_path,
monkeypatch):

```

```

    # --no-save-optim 语义是“不保存 optimizer 状态”; 若把 iteration 与
    # LR schedule 一起丢掉, 续训会从 iteration 0 静默重启且学习率曲线错乱
    scheduler = SimpleNamespace(state_dict=lambda: {"lr": 0.5})
    files = self._save(tmp_path, monkeypatch, no_save_optim=True, scheduler=scheduler)

    assert files == ["adapter_megatron_rank0.pt", "training_state_rank0.pt"]
    state = self._state(tmp_path)
    assert state["optimizer"] is None
    assert state["opt_param_scheduler"] == {"lr": 0.5}
    assert state["iteration"] == 3

```

评论区精华

该 PR 无实质 review 评论 (comments 与 review_comments 均为 0) , 真正的设计讨论沉淀在 commit 消息中。

claude[bot]: 该 PR 来自 fork, 自动化 Claude Code Review 已禁用; 维护者可评论 @claude review 手动触发一次评审。

commit d8536e65 (Zhichenzzz): Megatron 的 save_checkpoint 在 --no-save-optim 下已跳过 optimizer 与 scheduler 状态; save_lora_checkpoint 此前无条件写 training_state_rank*.pt, 应在写状态的位置统一 gate, 让 flag 有单一所有者。

commit 97702fd (Zhichenzzz): LoRA 训练状态是单文件 (iteration + LR scheduler + optimizer), 跳过整个文件会连带丢掉前两者; 恢复路径没有其他来源, load_lora_adapter 返回 iteration=None, adapter 权重恢复而 step 计数与 LR schedule 静默重启。

Zhichenzzz 直接 APPROVED (无评论), 批准依据大概率是 PR body 中的 4x H200 端到端对比表。

- fork PR 的自动 review 策略 (other): 未触发 bot 评审; Zhichenzzz 直接 APPROVED, 批准依据为 PR body 中的 4x H200 端到端验证数据。
- --no-save-optim 的语义边界: 跳过整个文件还是只丢 optimizer (design): 最终语义澄清: --no-save-optim 表示“不保存 optimizer 状态”而非“不写训练状态文件”; 加载侧同步增加 optimizer 为 None 的守卫。

风险与影响

- 风险:
 1. 行为变更: --no-save-optim 下 training_state_rank*.pt 仍会写出 (仅 optimizer 为 None), 任何依赖“文件不存在”来判断冷启动的外部脚本会观察到行为差异; 仓库内加载路径已同步适配, 但仓库外的消费方需要知晓。
 2. 数据契约兼容: _load_training_state 使用 weights_only=False 完整反序列化, 读取旧 checkpoint (optimizer 存在) 与新 checkpoint (optimizer 为 None) 时都能正确分支, 单测覆盖两个方向, 回归风险较低。
 3. 与 #3245 的耦合: 本 PR 处理保存侧, --no-load-optim 是加载侧的另一半 (#3245), 两者需配套合入, 否则会出现“保存时 drop、恢复时仍尝试 load”的不对称。
 4. 磁盘占用: 训练状态文件照写, --no-save-optim 不再带来文件级节省, 但这是有意取舍, 换取恢复正确性。
 - 影响: 用户影响: LoRA + Megatron 后端 + --no-save-optim 组合下, 断点恢复不再静默丢失 step 计数与 LR schedule, 避免学习率曲线错乱带来的隐性收敛与质量下降。系统影响: checkpoint 目录结构约定保持 (optimizer 键可为 None), 单测覆盖保存与加载两个方向, 为后续 checkpoint 行为提供回归保护。团队影响: 与 #3245 共同确立 LoRA checkpoint 中 optimizer 状态保存 / 加载的完整语义, 为 Megatron 主路径与 LoRA 副路径的 flag 行为对齐提供了参照。
 - 风险标记: 行为变更: --no-save-optim 仍写训练状态文件, 与 #3245 需配套合入, 恢复路径依赖 optimizer None 守卫

关联脉络

- PR #3245 fix(megatron): respect --no-load-optim when resuming a LoRA adapter: PR body 明确标注为本 PR 的另一半 (--no-load-optim)；改动同一批文件 (lora_utils.py、test_lora_utils.py、checkpoint.py)，两者合起来才构成 LoRA checkpoint 优化器状态保存 / 加载的完整语义闭环。