【训练方向 2025 冬季训练营】学习率调度器实现

1 背景与意义

在大模型训练框架中，**学习率调度器（LR Scheduler）**是一个极其核心、但又经常被低估的组件。在工程实践中，学习率调度器控制着学习率随着训练进程推进而亦步亦趋的变化，并且**直接决定了模型是否能够稳定收敛、是否能够复现实验结果、以及训练过程是否具备可控性与可恢复性**。

在 InfiniTrain 当前的开发阶段中，优化器 Optimizer 或其衍生的分布式优化器 DistributedOptimizer 已经具备完整、稳定的执行逻辑；但学习率的演化策略仍然是一个可以被独立抽象、独立实现、并且极具工程价值的模块。

同时，随着 InfiniTrain 对于多维并行、多机分布式以及异构训练等功能的支持与完善，后续亦将逐步开展针对更大规模模型、在更大规模集群上的实际训练实践，而为了追求更优的训练效果，学习率调度器的实现不仅重要，也将**直接影响 InfiniTrain 在大规模训练场景下的可扩展性与可维护性**。

因此，本训练营项目将围绕**学习率调度器的设计与实现**展开，要求在不修改现有优化器与分布式执行逻辑的前提下，完成一个**工程化、可扩展、可恢复**的学习率调度模块。

2 问题描述

2.1 目标

实现一个**学习率调度器模块**，用于在训练过程中按照预定策略动态调整优化器使用的学习率。

该调度器需要满足以下目标：

- 能够在训练过程中按 step 或 epoch 推进自身状态
- 能够向 Optimizer 提供当前应使用的学习率
- 能够在 checkpoint / resume 场景下保持行为一致
  - *注：InfiniTrain 主分支并未实现 checkpoint 功能，这里要求给出基础实现并留好接口*
- 能够与 PyTorch 中等价的学习率调度策略在数值与行为上保持一致

注意，我们将提供与 InfiniTrain 中 GPT-2/LLaMA-3 模型等价的 PyTorch 脚本，作为正确性验证用途。PyTorch 脚本**需要学员自行修改，插入 torch.optim.LRScheduler 的相关逻辑**。学员在 **InfiniTrain 中实现等价的逻辑**，并将二者 **loss 进行对比**。由于这部分修改后的 python 代码**用于对齐用途，也需要作为项目成果的一部分进行提交**。

2.2 原则

- **职责单一**：LRScheduler 只负责学习率的生成与状态演化，不参与参数更新、不感知梯度、不干涉优化器的执行逻辑。
- **最小耦合**：要求最小修改 Optimizer 的内部执行流程，仅通过公开接口设置学习率。
- **状态可恢复**：LRScheduler 必须具备明确、可序列化的内部状态，用于 checkpoint / resume。
- **接口稳定、易扩展**：接口与 PyTorch 对齐；同时，新增调度策略时，不应破坏已有 Scheduler 接口。

2.3 核心功能要求

训练营规则上，项目的实现并无具体技术路线要求，只要遵循上述原则并达成功能目标，通过标准即可认定为完成。下述“核心功能要求”仅作为上述原则的深入展开解释，并提供可能的实现思路供学员参考，以减轻学员理解项目任务的负担。

2.3.1 Scheduler 的基本抽象

2.3.1.1 LRScheduler 类设计

需要为学习率调度器设计一个统一的抽象接口，并在接口/使用方式上与 PyTorch 对齐（形式不做强制要求，以下仅为参考）：

代码块

```
class LRScheduler {
public:
    virtual void Step() = 0;                    // 推进调度器状态
    virtual float GetLR() const = 0;            // 获取当前学习率
    virtual StateDict State() const = 0;        // 导出状态
    virtual void LoadState(const StateDict&) = 0;
    virtual ~LRScheduler() = default;
};
```

2.3.1.2 与 Optimizer 的交互关系

在 PyTorch 中，LRScheduler 的 `step()` **语义上等价于一次 optimizer step**，而非一次 forward / backward。

Scheduler 不应直接参与参数更新，其典型调用顺序如下：

代码块

```
scheduler.Step();     // 更新学习率
optimizer.Step();     // 使用当前学习率更新参数
```

Scheduler 与 Optimizer 的唯一交互点应为**学习率的设置接口**，例如：`optimizer.SetLearningRate(lr)`；**不允许**在 Scheduler 中引入任何分布式通信、梯度访问或参数切分相关逻辑。

2.3.2 基础调度策略支持

不同调度策略各自表现为 LRScheduler 的派生类。

项目要求至少实现以下几种常见学习率调度策略：

1. **ConstantLR**：学习率始终保持不变
2. **StepLR**：每隔 `step_size` 个 step，将学习率乘以 `gamma`
3. **Linear Warmup**：在前 `warmup_steps` 内，学习率从 0 或初始值线性增长到目标学习率

要求上述调度器在给定相同初始参数时，与 PyTorch 中等价实现的数值行为保持一致。

2.3.3 组合调度策略支持

从抽象上看，学习率调度器本来就不是“只能有一种策略”，而是一个“随 step / epoch 演化的函数族”，组合只是函数复合或分段定义。PyTorch 原生支持调度策略的组合，类似 torch.nn.Module 一样可以嵌套、叠加的方式，组合调度策略仍然表现为 LRScheduler 的派生类。

项目要求至少实现以下几种学习率调度策略的组合手段：

1. **SequentialLR**：按顺序分段组合
2. **ChainedScheduler**：多种策略叠加，在每一步都同时执行
3. **LambdaLR**：自由定义学习率策略

要求上述调度器组合策略在给定相同初始参数时，与 PyTorch 中等价实现的数值行为保持一致。

2.3.4 Checkpoint / Resume 支持

InfiniTrain 主分支并未实现 checkpoint 功能，这里仅要求给出基础实现并留好接口。

Scheduler 必须支持状态保存与加载，至少包括：

- 当前 step / epoch
- 当前学习率（或可由状态推导出的等价信息）

恢复训练后，Scheduler 的行为应与不中断训练时完全一致。

2.3.4 交互接口设计

这部分要求调度器的使用方式应尽量简洁、直观。所有相关参数应在 `examples/` 下的 `main.cc` 里面以 gflags 的形式定义，使用上例如：

代码块

```
--lr-scheduler=steplr
--warmup-steps=1000
--total-steps=100000
```

3 提交要求及评判标准

3.1 提交要求

请提交以下内容：

1. 项目完成后，将代码**提交 PR**，命名：`【训练营】学习率调度器实现`
   a. PR 中不包含用于对齐的 PyTorch 代码
2. **提交项目报告**，包含：
   a. **设计文档部分**：如设计思路、关键模块架构、实现流程等
   b. **结果展示部分**：与 PyTorch 等价实现的对比验证结果
      i. 测例至少包括：针对 GPT-2/LLaMA-3 两个模型，在八卡 DDP（DP=8） 下，分别使用 `StepLR` 以及 `ChainedScheduler(StepLR, LinearLR)` 两种情况的 loss 结果比较，共 4 个测例。
      ii. `num_iterations` 至少为 10（至少训练 10 步），将上述四种测例下 InfiniTrain 和 PyTorch 的运行命令与 log 输出一同进行截图（四个测例分别使用两个框架跑，一共 8 张截图）。
   c. **模块使用说明**：简要说明功能使用方式（比如通过哪些命令行参数进行哪些配置）。
3. 将用于对齐的 **PyTorch 代码与上述报告**发送至 zhangbolun@qiyuanlab.com。

3.2 通过标准

项目需满足以下条件方可通过：

1. **功能正确性**：Scheduler 数值行为与 PyTorch 等价实现对齐
   - 在报告中结果展示的基础上，我们仍将在所支持的调度策略中，随机选取 **3 个黑盒测例**进行测试，记录正确性
2. **工程合理性**：满足 2.2 中叙述的实现原则
   - 不修改 Optimizer 的执行逻辑
   - 接口清晰、职责边界明确
   - 新增调度策略不破坏已有实现，且易扩展

3.3 优秀标准

1. 拓展功能实现，包括：
    a. 更多的默认调度策略，诸如 ExponentialLR、CosineAnnealingLR 等
    b. 目前 InfiniTrain 的 master 分支已实现多线程分布式的 DDP、TP、PP（带vPP）以及 SP 的组合，并且支持基于 DistributedOptimizer 的 ZeRO-1 显存优化策略。我们将基于 scripts/test_config.json 里面涉及的所有测例场景，对学习率调度器的实现进行测试，实现应在所有场景下均具有通用正确性
2. 在通过的基础上，功能完善、易扩展，且代码达到合入标准
    a. 即经过仓库维护者进行 pr review 且开发者根据 review comment 完成修改，最后由仓库维护者 approve 该 pr。

4 参考资料

- PyTorch 学习率调度器文档说明
- PyTorch 学习率调度器代码实现