深度学习中学习率策略的研究（实践篇）

朱梦

初稿于 2025-08-15，修改于 2025-08-15

1. 基于 PyTorch 实现 batch 细粒度的学习率策略

PyTorch 官方推荐将优化器和学习率调度器解耦合，有助于代码维护，各自职责更清晰，且推荐先优化器更新（optimizer.step()）后学习率调度器更新（scheduler.step()）。另外，PyTorch 官方提供的学习率调度器是 epoch 细粒度的。综上，基于 PyTorch 实现 batch 细粒度的学习率策略，需注意如下三点：

（1） 初始学习率的设置、最大学习率的设置。

（2） epoch 细粒度的学习率调度器如何修改为 batch 细粒度的学习率调度器。

（3） 学习率调度器自动维护的计数器是从零计数吗？

```python
from torch.optim.lr_scheduler import LRScheduler

class BatchScheduler(LRScheduler):
    """batch细粒度学习率调度器"""

    def __init__(self, optimizer, lr_lambda, last_epoch=-1):
        self.lr_lambda = lr_lambda
        super().__init__(optimizer, last_epoch)

    def get_lr(self):
        step = self.last_epoch
        return [self.lr_lambda(step)] * len(self.base_lrs)

    def step(self, epoch=None):
        # 更新计数器
        if epoch is None:
            self.last_epoch += 1
        else:
            self.last_epoch = epoch

        # 计算并应用新学习率
        values = self.get_lr()
        self._update_lr(values)

    def _update_lr(self, lrs):
        for param_group, lr in zip(self.optimizer.param_groups, lrs):
            param_group['lr'] = lr
        self._last_lr = lrs
```

学习率策略模板示例代码：

```python
class MyLRLambda:
    def __init__(self, max_lr: float, t1: int, r_min: float = 0.01) -> None:
        self.max_lr = max_lr
        self.t1 = t1
        self.r_min = r_min

    def __call__(self, batch_idx: int) -> float:
        if batch_idx > self.t1:
            r_t = self.r_min
        else:
            r_t = 1 + (self.r_min - 1) / self.t1 * batch_idx

        return self.max_lr * r_t

if __name__ == '__main__':
    lr_lambda = MyLRLambda(0.1, 100)
    init_lr = lr_lambda(0)
    print(init_lr)
```

2. 基于 JAX 实现 batch 细粒度的学习率策略