最优学习率的搜索算法

朱梦

初稿于 2025 年 6 月 29 日，修改于 2025-06-29

1. 问题定义及其分析

对于深层神经网络模型，学习率过大容易导致训练震荡甚至发散，过小则收敛缓慢浪费算力资源。因此，选取一个合适的学习率对模型的训练是至关重要的。基于批处理大小和学习率的缩放规律、模型宽度和学习率的缩放规律、模型深度和学习率的缩放规律，因此可以在某个较小的批处理大小且较合适的模型尺寸下对学习率进行搜索，搜出一个最优的学习率，然后基于缩放规律进行尺度迁移。

2. 问题解决

预期目标：在某个较小的批处理大小且较合适的模型尺寸下对学习率进行搜索，搜出一个最优的学习率。

2.1 基于网格搜索的最优学习率搜索

```python
import torch
from torch.utils.data import DataLoader

def grid_search_lr(model, train_loader, val_loader, lr_list, epochs=3):
    results = {}
    for lr in lr_list:
        optimizer = torch.optim.Adam(model.parameters(), lr=lr)
        val_loss = train_eval(model, train_loader, val_loader, optimizer, epochs)
        results[lr] = val_loss
        print(f"LR: {lr:.1e} | Val Loss: {val_loss:.4f}")
    return results

def train_eval(model, train_loader, val_loader, optimizer, epochs):
    model.train()
    for epoch in range(epochs):
        for x, y in train_loader:
            optimizer.zero_grad()
            loss = model(x, y)  # 替换为实际损失计算
            loss.backward()
            optimizer.step()

```

```
    # 验证评估
    model.eval()
    total_loss = 0
    with torch.no_grad():
        for x, y in val_loader:
            loss = model(x, y)
            total_loss += loss.item()
    return total_loss / len(val_loader)

# 使用示例
lr_list = [1e-5, 3e-5, 1e-4, 3e-4, 1e-3]
best_lr = min(grid_search_lr(model, train_loader, val_loader, lr_list), key=
    grid_search_lr.get)
```

2.2 基于学习率扫描器的最优学习率搜索

学习率扫描器（Learning Rate Finder）的核心原理：

（1） 从极小的学习率开始，指数级增加学习率（通常每批次增加 1.02-1.05 倍）。

（2） 监控损失函数的变化曲线。

（3） 识别损失下降最快且稳定的学习率区间。

```
def lr_finder(model, train_loader, start_lr=1e-7, end_lr=10,
              num_iter=100, beta=0.98):
    optimizer = torch.optim.SGD(model.parameters(), lr=start_lr)
    lr_mult = (end_lr / start_lr) ** (1/num_iter)
    lrs, losses = [], []
    avg_loss, best_loss = 0, float('inf')

    for i, (inputs, targets) in enumerate(train_loader):
        if i >= num_iter: break

        # 更新学习率
        current_lr = start_lr * (lr_mult ** i)
        for param_group in optimizer.param_groups:
        param_group['lr'] = current_lr

        # 训练步骤
        outputs = model(inputs)
        loss = criterion(outputs, targets)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # 指数平滑损失
        avg_loss = beta * avg_loss + (1-beta) * loss.item()
        smoothed_loss = avg_loss / (1 - beta**(i+1))

```

```
        # 记录结果
        lrs.append(current_lr)
        losses.append(smoothed_loss)

        # 早停：当损失开始显著增加
        if smoothed_loss > 4 * best_loss:
            break
        if smoothed_loss < best_loss:
            best_loss = smoothed_loss

    return lrs, losses

import matplotlib.pyplot as plt

lrs, losses = lr_finder(model, train_loader)

plt.figure(figsize=(10,6))
plt.plot(np.log10(lrs), losses)
plt.xlabel('log10(Learning Rate)')
plt.ylabel('Loss')
plt.title('LR Finder Curve')

# 识别最优区间：损失下降最陡峭的中间位置
min_loss_idx = np.argmin(losses)
steepest_slope_idx = np.argmin(np.gradient(losses[:min_loss_idx]))
optimal_lr = lrs[steepest_slope_idx]

plt.axvline(x=np.log10(optimal_lr), color='r', linestyle='--')
plt.show()
```

2.3 基于贝叶斯优化的最优学习率搜索

核心原理：

（1） 通过高斯过程建模损失函数。

（2） 智能选择下一个评估点（学习率）。

```
import optuna

def objective(trial):
    # 在log空间采样
    lr = trial.suggest_float('lr', 1e-6, 1e-1, log=True)

    model = create_model()
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)

    # 短周期训练
    for epoch in range(3):
        train_epoch(model, train_loader, optimizer)
```

```
13
14      # 验证评估
15      val_loss = evaluate(model, val_loader)
16      return val_loss
17
18  study = optuna.create_study(direction='minimize')
19  study.optimize(objective, n_trials=15)
20
21  # 获取最优结果
22  best_lr = study.best_params['lr']
23  print(f"Optimal LR: {best_lr:.2e}")
```

2.4 基于动态搜索算法的最优学习率搜索

```
1   def adaptive_lr_search(model, train_loader, init_lr=1e-3):
2       lr = init_lr
3       optimizer = torch.optim.SGD(model.parameters(), lr=lr)
4       best_loss = float('inf')
5
6       for epoch in range(10):
7           # 监控梯度统计量
8           grad_norms = []
9
10          for batch in train_loader:
11              loss = train_step(model, batch, optimizer)
12
13              # 收集梯度范数
14              total_norm = 0
15              for p in model.parameters():
16                  if p.grad is not None:
17                      param_norm = p.grad.data.norm(2)
18                      total_norm += param_norm.item() ** 2
19              grad_norms.append(total_norm ** 0.5)
20
21          # 分析梯度特征
22          grad_mean = np.mean(grad_norms)
23          grad_std = np.std(grad_norms)
24
25          # 调整规则
26          if grad_std / grad_mean > 0.5:  # 高震荡
27              lr *= 0.8
28          elif grad_mean < 1e-4:          # 更新不足
29              lr *= 1.5
30          else:                           # 稳定状态
31              lr *= 0.98
32
33          # 更新优化器
34          for param_group in optimizer.param_groups:
```

```
            param_group['lr'] = lr

        print(f"Epoch {epoch}: LR={lr:.2e}, Grad={grad_mean:.2e}±{grad_std:.2e}")

    return lr
```

3. 实验结果及其分析

4. 结论及其反思

组合策略实践分析：

（1） 初始探索：使用 LR Finder 确定大致范围（10-20 分钟）。

（2） 精搜阶段：贝叶斯优化或并行搜索（1-2 小时）。

（3） 最终验证：在最优值附近进行完整训练验证。

（4） 持续优化：部署后监控模型性能，动态调整学习率。