

# 深度学习优化器详解

从零开始理解SGD、RMSprop等优化算法

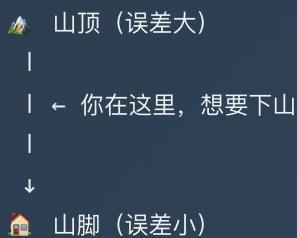
# 目录

1. 什么是优化器
  2. 核心概念：梯度下降
  3. SGD – 随机梯度下降
  4. Momentum – 动量优化
  5. RMSprop – 均方根传播
  6. Adam – 自适应矩估计
  7. 如何选择优化器
  8. 实战代码示例

## 什么是优化器

## 生活中的例子

想象你在山上迷路了，需要下山回家。但是现在雾很大，你只能看到脚下的地面坡度。**优化器就是你下山的策略。**



深度学习中：

- 山 = 误差函数 (Loss Function)
  - 下山 = 减小误差

- 你的位置 = 模型参数（权重）
- 下山策略 = 优化器

## 优化器的目标

找到让模型预测最准确的参数（权重）。

## 核心概念：梯度下降

### 什么是梯度？

梯度 = 斜坡的陡峭程度和方向



### 数学表达式（简化版）



**比喻：**

- 学习率 = 每步的步长
- 梯度 = 告诉你哪个方向最陡

## 可视化示例

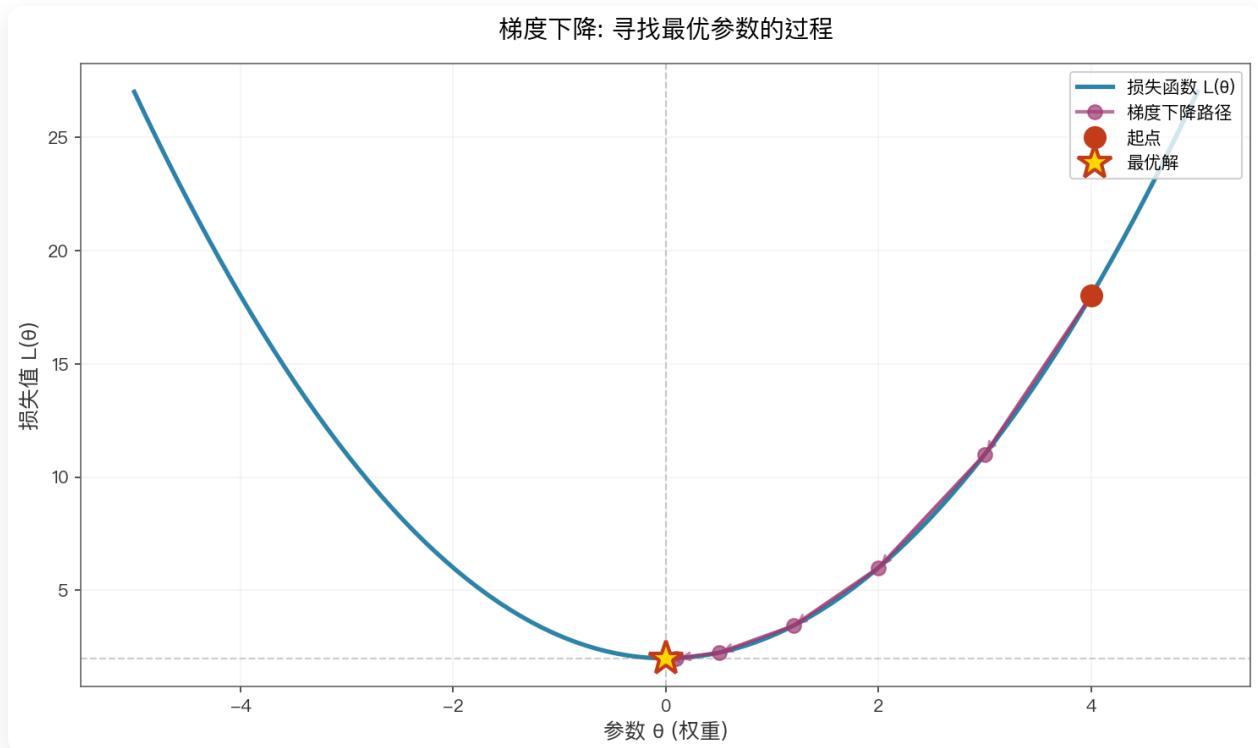


图1: 梯度下降过程 – 从起点逐步向最优解移动,每次迭代都沿着梯度反方向前进

## SGD: 随机梯度下降

Stochastic Gradient Descent

### 核心思想

最基础、最朴素的优化方法：**看到斜坡就往下走。**

### 工作原理

1. 计算当前位置的梯度（斜坡方向）

2. 沿着反方向走一小步

3. 重复以上步骤

## 公式

$$\theta = \theta - \eta \times \nabla L$$

其中：

- $\theta$  (theta) = 参数 (权重)
- $\eta$  (eta) = 学习率
- $\nabla L$  (nabla L) = 梯度

## 用人话说

```
# 伪代码  
当前位置 = 当前位置 - 学习率 × 梯度
```

### 优点 ✓

- **简单直接**: 容易理解和实现
- **内存占用小**: 不需要记住历史信息

### 缺点 ✗

- **容易震荡**: 在峡谷地形会左右摇摆
- **速度慢**: 在平坦区域移动缓慢
- **容易卡住**: 可能困在局部最优点

## 图解：SGD的路径

起点   
|  
| ← 直上直下，路径简单  
|  
↓  
★ 终点

## 实战代码

```
# PyTorch示例
import torch.optim as optim

# 创建SGD优化器
optimizer = optim.SGD(
    model.parameters(), # 模型参数
    lr=0.01            # 学习率
)

# 训练循环
for data, target in dataloader:
    optimizer.zero_grad()      # 清空梯度
    output = model(data)       # 前向传播
    loss = criterion(output, target) # 计算损失
    loss.backward()             # 反向传播，计算梯度
    optimizer.step()            # 更新参数
```

## Momentum：动量优化

带冲劲的SGD

# 核心思想

像滚雪球一样下山 – 越滚越快，不容易停下来。

## 生活比喻

想象你推一个小球下山：

- **没有动量**: 小球走走停停，遇到小坑就停了
- **有动量**: 小球积累速度，能冲过小坑，滚得更快

## 工作原理

记住之前的移动方向，在当前方向上加上“惯性”。

$$\text{新速度} = \text{旧速度} \times \text{动量系数} + \text{当前梯度}$$

$$\text{新位置} = \text{旧位置} - \text{学习率} \times \text{新速度}$$

## 公式

$$v_t = \beta \times v_{t-1} + \nabla L$$

$$\theta = \theta - \eta \times v_t$$

其中：

- $v$  = 速度（累积的梯度）
- $\beta$  (beta) = 动量系数（通常0.9）

## 优点

- **加速收敛**: 在正确方向上越走越快
- **减少震荡**: 惯性让路径更平滑
- **冲过局部最优**: 动量可以帮助跳出小坑

## 缺点 ✗

- 可能冲过头：在接近最优点时可能震荡
- 需要调参：动量系数需要调整

## 图解:Momentum的路径

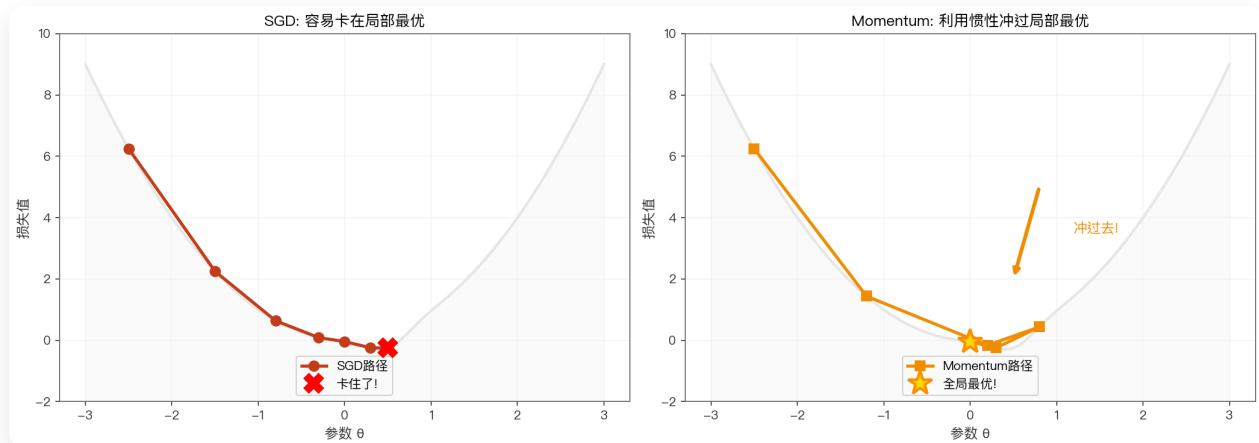


图2: Momentum效果 – 累积动量使优化路径更加平滑,加速收敛

## 对比图

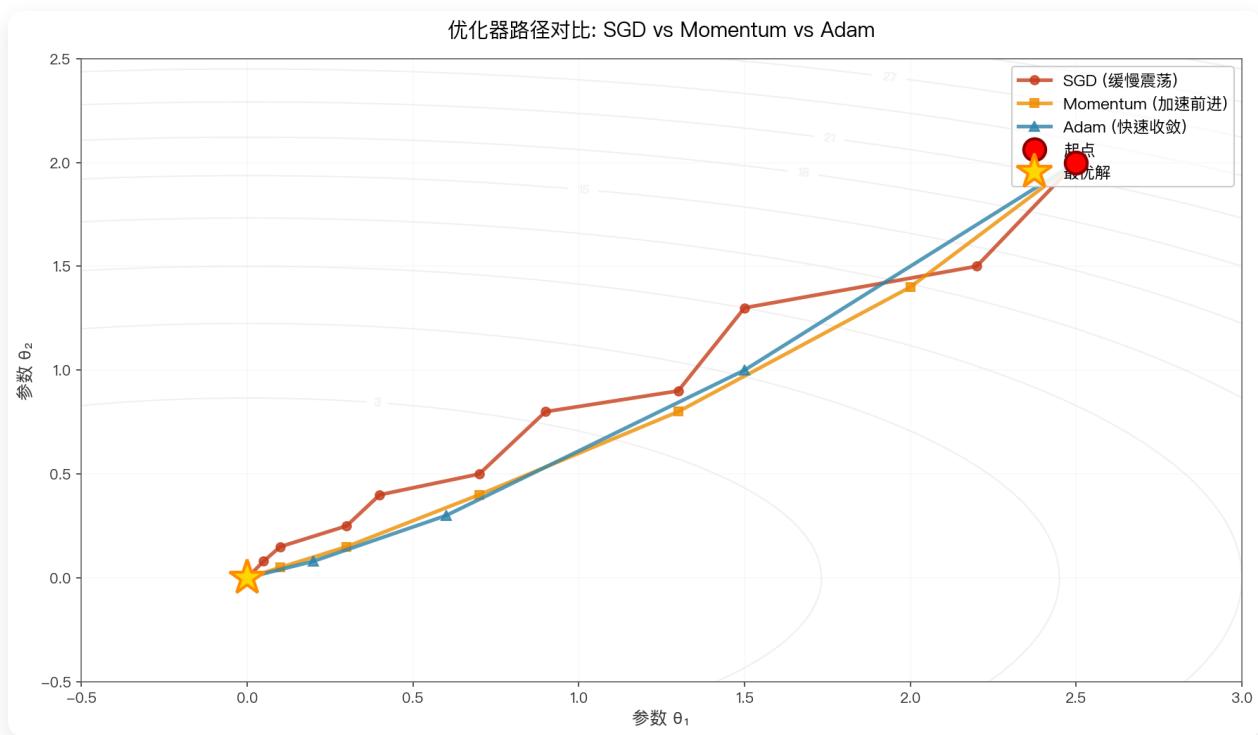


图3: SGD vs Momentum路径对比 – SGD路径呈锯齿状,Momentum路径更加平滑

## 实战代码

```
# PyTorch示例
optimizer = optim.SGD(
    model.parameters(),
    lr=0.01,
    momentum=0.9 # 动量系数
)
```

# RMSprop：均方根传播

---

Root Mean Square Propagation

## 核心思想

自适应调整每个参数的学习率 – 陡峭的地方小步走，平坦的地方大步走。

## 生活比喻

想象你在不同地形上行走：

- **陡坡**: 小心翼翼，小步前进（减小学习率）
- **平地**: 大步流星，快速前进（增大学习率）

## 为什么需要RMSprop?

在深度神经网络中，不同参数的梯度大小差异很大：

- 有的参数梯度很大（陡峭）
- 有的参数梯度很小（平坦）

SGD对所有参数用同一个学习率 → 不够灵活

## 工作原理

1. 计算梯度的**平方**
2. 用移动平均累积这些平方值
3. 用平方根来**归一化**学习率

## 公式

$$\begin{aligned} E[g^2]_t &= \rho \times E[g^2]_{t-1} + (1-\rho) \times g^2_t \\ \theta &= \theta - (\eta / \sqrt(E[g^2]_t + \epsilon)) \times g_t \end{aligned}$$

其中：

- $E[g^2]$  = 梯度平方的移动平均
- $\rho$  (rho) = 衰减率 (通常0.9)
- $\epsilon$  (epsilon) = 防止除零的小数 (如 $1e-8$ )

## 用人话说

```
# 伪代码
梯度平方累积 = 0.9 × 旧累积 + 0.1 × 当前梯度2
调整后的学习率 = 学习率 / √(梯度平方累积)
新位置 = 旧位置 - 调整后的学习率 × 梯度
```

## 优点 ✓

- **自适应学习率**: 不同参数不同速度
- **适合处理稀疏梯度**: 在RNN等模型中表现好
- **减少震荡**: 在峡谷地形表现好

## 缺点 ✗

- **学习率可能衰减过快**: 后期学习率可能太小
- **需要调整超参数**: 衰减率 $\rho$ 需要调整

## 图解: RMSprop的自适应

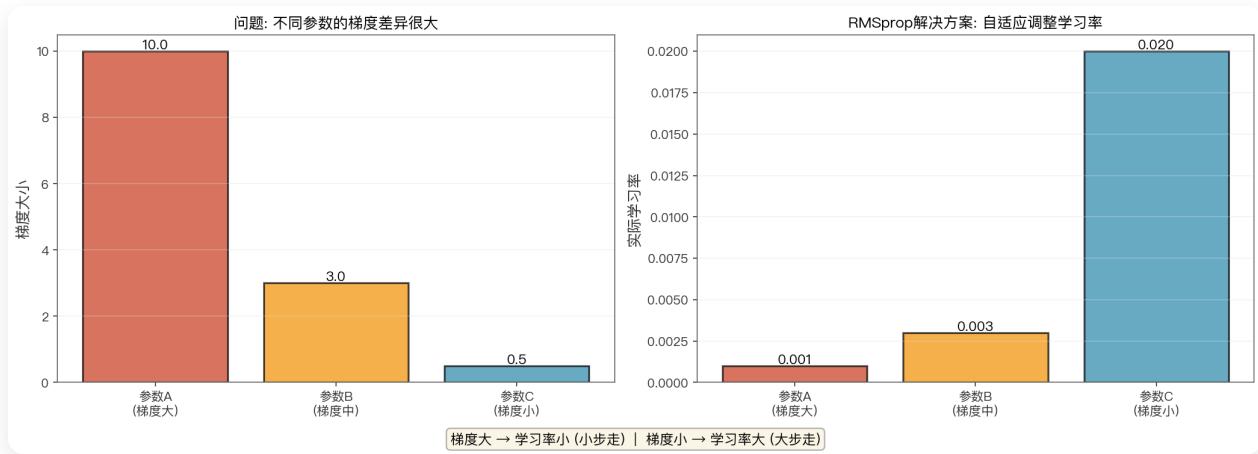


图4: RMSprop自适应学习率 – 梯度大的参数用小步长,梯度小的参数用大步长

## 实战代码

```
# PyTorch示例
optimizer = optim.RMSprop(
    model.parameters(),
    lr=0.01,          # 初始学习率
    alpha=0.9,        # 衰减率 (对应公式中的ρ)
    eps=1e-8          # 防止除零
)
```

## Adam: 自适应矩估计

Adaptive Moment Estimation

# 核心思想

Momentum + RMSprop = Adam ↗

结合了：

- Momentum的动量（记住方向）
- RMSprop的自适应学习率（调整步长）

## 为什么叫Adam?

Adaptive Moment – 自适应的一阶矩（均值）和二阶矩（方差）估计。

## 工作原理

Adam维护两个移动平均：

1.  $m_t$ : 梯度的移动平均（一阶矩，类似Momentum）
2.  $v_t$ : 梯度平方的移动平均（二阶矩，类似RMSprop）

## 公式（完整版）

```
# 第一步：计算移动平均  
 $m_t = \beta_1 \times m_{t-1} + (1-\beta_1) \times g_t$   
 $v_t = \beta_2 \times v_{t-1} + (1-\beta_2) \times g^2_t$   
  
# 第二步：偏差修正  
 $\hat{m}_t = m_t / (1 - \beta_1^t)$   
 $\hat{v}_t = v_t / (1 - \beta_2^t)$   
  
# 第三步：更新参数  
 $\theta = \theta - \eta \times \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ 
```

其中：

- $\beta_1 = 0.9$  (一阶矩衰减率)
- $\beta_2 = 0.999$  (二阶矩衰减率)
- $\epsilon = 1e-8$  (防止除零)

## 用人话说

```
# 伪代码  
# 1. 记住方向 (像Momentum)  
方向累积 = 0.9 × 旧方向 + 0.1 × 当前梯度  
  
# 2. 记住梯度大小 (像RMSprop)  
大小累积 = 0.999 × 旧大小 + 0.001 × 当前梯度2  
  
# 3. 修正偏差 (重要! )  
修正后的方向 = 方向累积 / (1 - 0.9步数)  
修正后的大小 = 大小累积 / (1 - 0.999步数)  
  
# 4. 更新参数  
新位置 = 旧位置 - 学习率 × 修正后的方向 / √修正后的大小
```

## 为什么需要偏差修正?

在训练初期，移动平均m和v都接近0，会导致：

- 一开始更新太慢
- 偏差修正可以加速初期训练

## 优点 ✅

- **最常用**: 在大多数任务中表现最好
- **鲁棒性强**: 对超参数不敏感
- **快速收敛**: 结合了Momentum和RMSprop的优点
- **适合大多数场景**: 默认选择

## 缺点 ✗

- **可能过拟合**: 在某些小数据集上
- **不一定最优**: 某些特定任务其他优化器更好

## 图解:Adam的优势

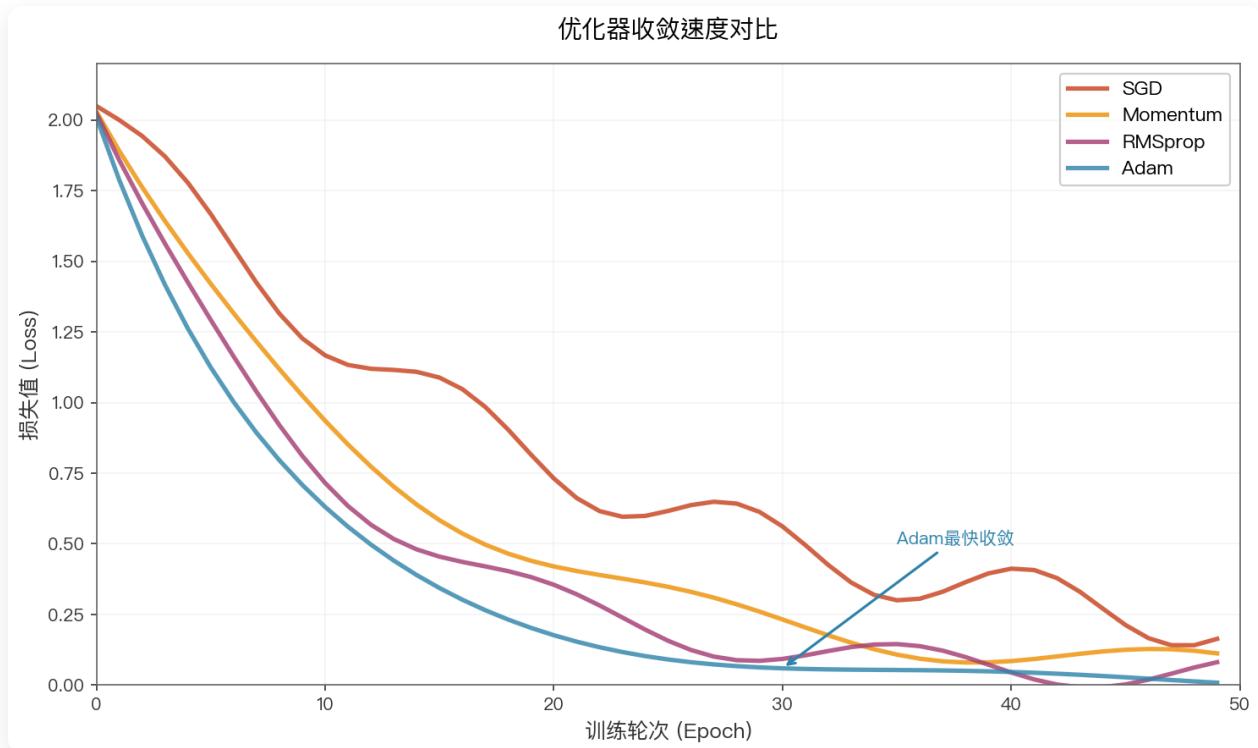


图5: 优化器收敛对比 – Adam结合了Momentum和RMSprop的优点, 收敛最快且路径最平滑

## 实战代码

```
# PyTorch示例
optimizer = optim.Adam(
    model.parameters(),
    lr=0.001,           # 学习率 (Adam通常用较小值)
    betas=(0.9, 0.999), # (β₁, β₂)
    eps=1e-8,          # ε
    weight_decay=0      # L2正则化 (可选)
)
```

# 如何选择优化器

## 快速决策树



## 详细对比表

优化器	速度	稳定性	内存	适用场景	推荐度
SGD	★★	★★★★★	★★★★★	小模型、简单任务	★★★★★
SGD+Momentum	★★★★	★★★★★	★★★★★	计算机视觉、图像分类	★★★★★
RMSprop	★★★★★	★★★★	★★★★	RNN、非平稳数据	★★★★★
Adam	★★★★★★	★★★★★	★★★★	大多数任务	★★★★★★

## 具体建议

⌚ 入门建议：直接用Adam

```
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

🖼 图像分类 (CNN)：SGD + Momentum

```
optimizer = optim.SGD(model.parameters(), lr=0.1, momentum=0.9)
```

📝 自然语言处理 (Transformer)：Adam或AdamW

```
optimizer = optim.AdamW(model.parameters(), lr=1e-4)
```

⌚ 循环神经网络 (RNN/LSTM)：RMSprop或Adam

```
optimizer = optim.RMSprop(model.parameters(), lr=0.001)
```

# 实战代码示例

## 完整训练循环

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader

# 1. 定义模型
class SimpleNet(nn.Module):
    def __init__(self):
        super(SimpleNet, self).__init__()
        self.fc1 = nn.Linear(784, 128)
        self.fc2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# 2. 初始化模型
model = SimpleNet()

# 3. 选择优化器（试试不同的优化器！）
# 方案A: Adam (推荐新手)
optimizer = optim.Adam(model.parameters(), lr=0.001)

# 方案B: SGD + Momentum
# optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)

# 方案C: RMSprop
# optimizer = optim.RMSprop(model.parameters(), lr=0.001)

# 4. 定义损失函数
criterion = nn.CrossEntropyLoss()

# 5. 训练循环
def train(model, dataloader, optimizer, criterion, epochs=10):
    model.train()

    for epoch in range(epochs):
```

```
total_loss = 0

for batch_idx, (data, target) in enumerate(dataloader):
    # 5.1 清空梯度
    optimizer.zero_grad()

    # 5.2 前向传播
    output = model(data)

    # 5.3 计算损失
    loss = criterion(output, target)

    # 5.4 反向传播（计算梯度）
    loss.backward()

    # 5.5 更新参数（优化器的核心作用！）
    optimizer.step()

    total_loss += loss.item()

    # 打印每个epoch的平均损失
    avg_loss = total_loss / len(dataloader)
    print(f'Epoch {epoch+1}/{epochs}, Loss: {avg_loss:.4f}')

# 6. 开始训练
# train(model, train_loader, optimizer, criterion)
```

## 对比不同优化器

```
import matplotlib.pyplot as plt

# 训练同一个模型，用不同优化器
def compare_optimizers(model, train_loader, epochs=20):
    optimizers = {
        'SGD': optim.SGD(model.parameters(), lr=0.01),
        'SGD+Momentum': optim.SGD(model.parameters(), lr=0.01, momentum=0.9),
        'RMSprop': optim.RMSprop(model.parameters(), lr=0.001),
        'Adam': optim.Adam(model.parameters(), lr=0.001)
    }

    results = {}

    for name, optimizer in optimizers.items():
        print(f'\n训练使用 {name}...')
        model_copy = SimpleNet() # 每次用新模型

        loss_history = []
        for epoch in range(epochs):
            # ... 训练代码 ...
            loss_history.append(avg_loss)

        results[name] = loss_history

    # 绘制对比图
    plt.figure(figsize=(10, 6))
    for name, losses in results.items():
        plt.plot(losses, label=name)

    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('不同优化器的收敛速度对比')
    plt.legend()
    plt.grid(True)
    plt.show()
```

# 学习率调度器

## 为什么需要调整学习率?

想象下山:

- **开始**: 大步走 (学习率大) → 快速接近最优点
- **接近山脚**: 小步走 (学习率小) → 精确找到最优点

## 常用策略

```
# 1. 阶梯式衰减 (StepLR)
# 每30个epoch, 学习率乘以0.1
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=30, gamma=0.1)

# 2. 余弦退火 (CosineAnnealingLR)
# 学习率按余弦曲线变化
scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=100)

# 3. 指数衰减 (ExponentialLR)
# 每个epoch学习率乘以0.95
scheduler = optim.lr_scheduler.ExponentialLR(optimizer, gamma=0.95)

# 使用方法
for epoch in range(epochs):
    train(...)
    scheduler.step() # 更新学习率
```

# 常见问题解答

## Q1: 为什么我的loss不下降?

可能原因:

1. 学习率太大 → 减小10倍试试
2. 学习率太小 → 增大10倍试试
3. 梯度消失 → 检查激活函数, 用BatchNorm
4. 数据问题 → 检查数据预处理

```
# 解决方案: 使用学习率查找器
def find_lr(model, optimizer, dataloader, start_lr=1e-7, end_lr=10):
    lrs = []
    losses = []

    for lr in np.logspace(np.log10(start_lr), np.log10(end_lr), 100):
        # 设置学习率
        for param_group in optimizer.param_groups:
            param_group['lr'] = lr

        # 训练一步
        loss = train_one_step(model, dataloader)

        lrs.append(lr)
        losses.append(loss)

    # 绘图找到最佳学习率
    plt.plot(lrs, losses)
    plt.xscale('log')
    plt.xlabel('Learning Rate')
    plt.ylabel('Loss')
    plt.show()
```

## Q2: Adam vs SGD, 到底选哪个?

简单规则:

- 刚开始学习 → 用Adam
- 要刷榜/论文 → 试试SGD+Momentum

- 时间紧迫 → 用Adam (收敛快)
- 追求最优 → 两个都试试

## Q3: 如何调整超参数?

**推荐顺序:**

### 1. 学习率 (最重要! )

- Adam: 从0.001开始
- SGD: 从0.1开始

### 2. Batch Size

- 小batch (32–128) → 训练更稳定
- 大batch (256–1024) → 训练更快

### 3. 其他参数 (通常用默认值就好)

- Momentum: 0.9
- Beta1/Beta2: 0.9/0.999
- Weight decay: 0 或  $1e-4$

---

## 总结

---

## 核心概念回顾

- ⌚ 优化器 = 下山策略
  - └ SGD: 最基础, 直接往下走
  - └ Momentum: 加速, 有惯性
  - └ RMSprop: 自适应步长
  - └ Adam: 集大成者 (推荐)

## 新手推荐配置

```
# 直接复制这个配置，90%的任务都能用！  
model = YourModel()  
optimizer = optim.Adam(  
    model.parameters(),  
    lr=0.001,  
    betas=(0.9, 0.999),  
    eps=1e-8  
)  
scheduler = optim.lr_scheduler.CosineAnnealingLR(  
    optimizer,  
    T_max=100  
)
```

## 记忆口诀

选优化器，记住这几点：

1. 新手入门，直接Adam
2. 图像分类，SGD+动量
3. 自然语言，Adam或AdamW
4. 学习率，从小开始试
5. 不收敛，先查学习率

## 进一步学习资源

### 1. 论文原文：

- Adam: [Kingma & Ba, 2014]
- RMSprop: [Hinton's Coursera课程]
- Momentum: [Qian, 1999]

### 2. 可视化工具：

- [Optimizer Visualization](#)
- [Neural Network Playground](#)

### 3. PyTorch官方文档：

- [torch.optim](#)

## 附录：数学符号说明

符号	含义	读音
$\theta$	参数（权重）	theta
$\eta$	学习率	eta
$\nabla L$	梯度	nabla L
$\beta$	动量系数	beta
$\rho$	衰减率	rho
$\epsilon$	极小值	epsilon
$\alpha$	RMSprop衰减率	alpha

## 结语

希望这份文档能帮助你理解优化器的原理！

### 记住最重要的一点：

优化器就是帮助模型找到最佳参数的“向导”。不同的向导有不同的策略，但目标都是一样的——**让模型变得更好。**

### 实践建议：

1. 先用Adam，跑通整个流程
2. 观察训练曲线，调整学习率
3. 如果有时间，试试其他优化器
4. 多做实验，积累经验

祝学习顺利！🚀

本文档由AI生成，用于深度学习教学  
如有疑问，欢迎讨论交流