第 8 章：非凸优化算法

部分 1：PyTorch 入门

授课教师：曹语

课程主页：https://yucaoyc.github.io/math3806

目录

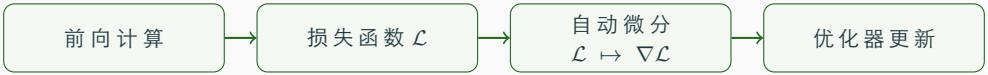
为什么先学一点 PyTorch?

前面章节中，我们已经用 cvxpy 求解过不少凸优化问题。

到了非凸和机器学习模型中，很多目标函数不再适合直接交给凸优化求解器。

本节先用 Logistic 回归这个熟悉的凸模型练习 PyTorch；真正的非凸性可通过使用神经网络模型出现。

通常的做法是把训练拆成四件事：



重复以上四步，参数逐步变化

学习目标

完成本节后，希望能够：

- 说出 tensor、Dataset、DataLoader 分别承担什么角色；
- 用 PyTorch 写出一个 logistic 回归模型；
- 使用二分类交叉熵作为训练目标；
- 读懂训练循环中的四步：清零梯度、前向计算、反向传播、更新参数；
- 观察学习率如何影响 full-batch 训练过程。

目录

第 4 章回顾：Logistic 回归的训练图景

第 4 章中，Logistic 回归把二分类问题写成一个可优化问题：

$$(\boldsymbol{x}_i, y_i) \to z_i = \boldsymbol{a}^\top \boldsymbol{x}_i + b \to p_i = \sigma(z_i),$$

$$\mathcal{L}(\boldsymbol{a}, b) = \frac{1}{n} \sum_{i=1}^{n} \ell_i, \ \ell_i = -\big[y_i \log p_i + (1 - y_i) \log(1 - p_i)\big].$$

关键元素：

数学对象	在 PyTorch 中对应什么
数据 $(\boldsymbol{x}_i, y_i)$	组织成 batch：`Dataset`、`DataLoader`
模型参数 $(\boldsymbol{a}, b)$	模型与可训练参数
损失 ℓ_i	衡量预测和标签差距：损失函数
梯度与更新	改进参数：优化器

本节主线： 把这些数学对象对应到 PyTorch 训练循环。

模块 1：Tensor — 创建张量

PyTorch 的基本数据结构是 tensor。可以先把它理解成类似 NumPy array 的对象。

创建 tensor 时，先想清楚：数据是什么，形状是多少，以及是否需要被优化。

语法	含义
`torch.tensor([1., 2., 3.])`	从列表创建一维 tensor
`torch.zeros(2, 3)`	创建 2×3 的全零 tensor
`torch.randn(4, 5)`	创建 4×5 的随机 tensor
`torch.tensor(X, dtype=torch.float32)`	指定数据类型

模块 2：Dataset 与 DataLoader

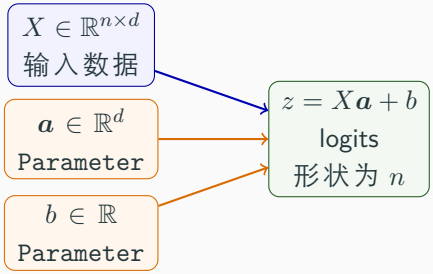
训练数据通常不是每次手工切片，而是交给 `Dataset` 和 `DataLoader` 管理。

对象	作用
`Dataset`	回答“第 i 个样本 $(\boldsymbol{x}_i, y_i)$ 是什么?”
`DataLoader`	把若干样本组成 batch，并控制是否打乱顺序
batch (X, y)	训练循环每次读入的统一格式

本节先看 full-batch 训练：把全部训练样本组成一个 batch。这样可以先专注理解训练循环本身。

模块 3：参数声明

```
import torch
from torch import nn
X = torch.tensor([[1., 2.], [3., 4.]])
a = nn.Parameter(torch.zeros(2))
b = nn.Parameter(torch.zeros(()))
```



关键点： 训练时只更新 `nn.Parameter`，数据 tensor 通常不需要梯度。

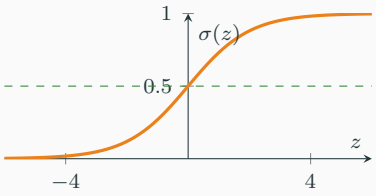
模块 4：模型输出与损失函数

Logistic 回归先给出 logit：

$$z_i = \boldsymbol{a}^\top \boldsymbol{x}_i + b, \qquad p_i = \sigma(z_i).$$

在代码中，模型直接输出 logit：

$$\texttt{logits = model(X)} = X\boldsymbol{a} + b.$$



单个样本的二分类交叉熵为

$$\ell_i = -\left[y_i \log p_i + (1 - y_i) \log(1 - p_i)\right].$$

因此使用 `BCEWithLogitsLoss`：

$$\texttt{criterion(logits,y)} = \frac{1}{n} \sum_i \ell(z_i, y_i),$$

$$\ell(z_i, y_i) = \mathrm{BCE}\big(\sigma(z_i), y_i\big).$$

它默认返回平均损失，并把 Sigmoid 和 BCE 合在一起，数值上更稳定。注意：标签 y 需是与 logits 同形状的浮点张量。

模块 5：梯度计算和参数更新

有了损失函数后，训练就是不断降低

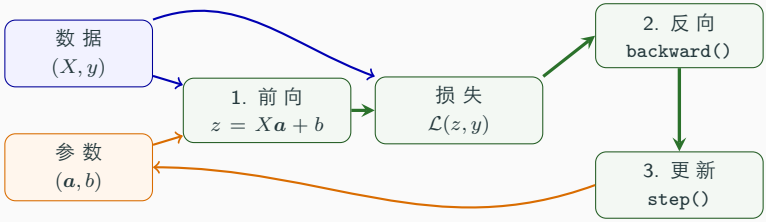
$$\mathcal{L}(\boldsymbol{a}, b) = \frac{1}{n}\sum_i \ell_i, \qquad \boldsymbol{a} \leftarrow \boldsymbol{a} - \eta \nabla_{\boldsymbol{a}} \mathcal{L}, \quad b \leftarrow b - \eta \frac{\partial \mathcal{L}}{\partial b}.$$

```
criterion = nn.BCEWithLogitsLoss()
optimizer = torch.optim.SGD([a, b], lr=0.1)
optimizer.zero_grad() # clear old gradients
loss = criterion(X @ a + b, y)
loss.backward()     # compute gradients
optimizer.step()    # update parameters
```

关键提醒： `loss.backward()` 计算梯度，`optimizer.step()` 才真正改变参数。

训练循环：模块关系图

full-batch 中，数据固定；每轮计算当前梯度并更新参数。



提醒： 每轮最初时前先调用 `zero_grad()`；否则 `.grad` 会累加旧梯度。

训练循环：四个动作

每次迭代中的训练循环基本固定：

0. `optimizer.zero_grad()`：清除上一轮梯度；
1. `logits = model(X)`，`loss = criterion(logits, y)`：前向计算并得到损失；
2. `loss.backward()`：反向传播并计算梯度；
3. `optimizer.step()`：用优化器更新参数。

目录

训练一个 Logistic 模型

课堂演示中固定相同初始模型，比较不同学习率下的 full-batch 训练：

设置	观察重点
学习率较小	稳定但下降较慢
学习率适中	速度与稳定较平衡
学习率较大	可能更快，也可能震荡

参与探究：改变学习率

建议只改一个参数，再观察变化：

- 固定数据和模型，只改变学习率：过小、适中、过大；
- 记录训练损失、测试准确率、曲线波动程度；
- 思考为什么学习率过大会造成不稳定。

问题：如果训练损失上下剧烈震荡，你会如何调整学习率？

课堂讨论：学习率控制每一步走多远，是训练稳定性的核心参数之一。

目录

快速检查

1. `Dataset` 和 `DataLoader` 分别负责什么?
2. `loss.backward()` 和 `optimizer.step()` 分别做什么?
3. 如果学习率过大，训练损失可能出现什么现象?

快速检查

1. `Dataset` 和 `DataLoader` 分别负责什么?
2. `loss.backward()` 和 `optimizer.step()` 分别做什么?
3. 如果学习率过大，训练损失可能出现什么现象?

答案： 1. 前者定义样本，后者组织 batch；2. 前者计算梯度，后者更新参数；3. 可能下降不稳定，甚至出现震荡或发散。

总结与阅读材料

本节课建立了后续 SGD 理论需要的计算图景：

- 模型与损失：输入和参数给出预测，损失 $\mathcal{L}$ 衡量误差；
- 梯度与更新：`loss.backward()` 计算梯度，`optimizer.step()` 更新参数；
- 数据组织：`Dataset` 定义样本，`DataLoader` 组织 batch。

阅读材料： PyTorch Quickstart Tutorial

下一部分： 为什么只用一小批样本的梯度也能指导优化?