工程设计课程讲义系列

课程代码 46050107 · 深度学习与图像识别 · 2 学分 / 32 学时

人工智能与图像识别

——产教融合项目式课程讲义——

第一篇　理论篇：十二讲（第 6–18 周）

第二篇　项目背景篇：三个真实项目到底在做什么

第三篇　任务书篇：15 个小组的任务、实验与验收标准

主讲：张洪进　工程学院

授课班级：2024 机器人 1 班、2024 机器人 2 班

2026–2027 学年第一学期　2026 年 10 月

人工智能与图像识别——产教融合项目式课程讲义

主编：张洪进

2026 年 10 月第一版

本讲义配套课程《深度学习与图像识别》（课程代码 46050107，工程学院，专业选修，2 学分 32 学时：讲授 24 学时＋实验 8 学时）。教材：许庆阳《深度学习模型与算法基础》（清华大学出版社，2023 年 12 月第 1 版）；参考书：Ian Goodfellow、Yoshua Bengio、Aaron Courville《深度学习》（人民邮电出版社，2017 年）。本讲义为教学内部资料，其中引用的网络图片与论文插图仅用于课堂教学，版权归属原作者，来源均已逐图标注。

前言

这本讲义和你以前见过的教材不一样

如果你翻开这本讲义时想的是“又是一本背概念应付考试的教材”，那你很快会发现不对劲——这本讲义里没有“期末划重点”，取而代之的是三份货真价实的工程任务书：今年秋天，你们中的绝大多数人将加入三个真实项目组，和研究生、毕业设计学长一起，把人工智能技术用到海洋石油平台的全景监控、水下锚链检查和海量工程数据治理上。

这就是我们这门课的教学改革方案，我们叫它**“产校验方案”**——真实生产项目直接进入学校课堂，用真实项目检验课堂所学：*12* 讲理论课打底，*4* 次现场实验搭桥，*15* 个小组认领真实课题，第 *19* 周周五（2027 年 1 月 22 日）全部成果提交验收。你在这门课里写下的每一行代码、跑出的每一个数字，都不是“作业题”，而是要被真实系统采用的工程结果。

怎么使用这本讲义

全书分三篇，对应你在课程里的三个阶段：

- **第一篇 · 理论篇（第 6–18 周随堂讲授）**：十二讲，从最古老的感知器一路讲到今天的大模型。完全从零基础写起：需要的数学，现场补（蓝色“知识补齐”盒子）；看不懂的地方，给生活化的类比（青色“零基础扫盲”盒子）；每一讲都告诉你它和三个项目的关系（橙色“项目连线”盒子）。每讲最后一道课后编程题，要求你**指挥大模型（AI）替你编程**——这是本课程最核心的技能训练：你不需要从零手写代码，但你必须能看懂、能改动、能讲清楚 AI 给你写了什么。
- **第二篇 · 项目背景篇（第 5–8 周自行阅读）**：三个项目各一章，讲清楚它们“是什么、为什么做、为谁做”。读完这一篇，你拿到任务书时不会迷茫。
- **第三篇 · 任务书篇（贯穿全学期）**：15 个小组每组一份任务书——总体任务、第 19 周周五的提交清单、哪些活 AI 干、哪些活必须你自己干、防糊弄指南、满分验收标准，以及四个实验各自的具体要求。这一篇是你这学期的“作战地图”。

关于 AI 的承诺与底线

这门课不禁止 AI，恰恰相反——**这门课要求你用 AI**。四次现场实验的第一课，就是配置你自己的大模型编程助手。但有一条底线贯穿全书：**AI 可以用，结果你自己扛**。实验数据不许直接抄 AI 给的数字；训练精度不许虚构；每一个交付物都要有“命令＋日志＋产物”三件套证据。工程世界里，说谎的代价不是扣分，是事故。

致谢

本讲义参考了国内外大量经典教材、论文与开源资料，包括许庆阳《深度学习模型与算法基础》、Goodfellow 等《深度学习》、*Stanford CS231n/CS224n* 课程讲义、以及 *Fast-FoundationStereo*、*Depth*

Anything V2、*R-GCN* 等论文与开源项目；三个项目的背景材料来自真实工程资料。所有引用在书中逐处标注，并在附录集中列出。感谢参与三个项目的全体研究生助教与工程师。

张洪进

2026 年 10 月

目录

前言 i

第一部分　第一篇　理论篇：十二讲，从感知器到大模型 1

第 0 讲　开课导引：这门课怎么上、怎么学、怎么考 2
0.1 这门课讲什么：从一个神经元到大模型 . . . 2
0.1.1 先排座次：人工智能、机器学习、神经网络与深度学习 . . . 4
0.2 三个真实项目一页速览 . . . 4
0.3 你需要准备什么 . . . 6
0.4 如何用大模型帮你学编程（本书最重要的方法论） . . . 7
0.4.1 AI 编程助手是什么 . . . 7
0.4.2 提示词怎么写：四个要素 . . . 8
0.4.3 agent 循环：AI 是怎么“自己干活”的 . . . 8
0.5 零基础数学包：够用就好的三小块 . . . 9
0.6 零基础 Python 包：从装环境到会读代码 . . . 11
0.7 全书地图：三篇怎么配合用 . . . 12

第 1 讲　感知器、线性单元与学习规则 14
1.1 生物神经元：大脑的原始零件 . . . 14
1.2 感知器：一颗人工神经元 . . . 15
1.3 决策边界：感知器在平面上画一条线 . . . 16
1.4 它怎么学习：感知器学习规则 . . . 18
1.5 从 0/1 到连续：线性单元与最小均方 . . . 20
1.6 天花板与寒冬：一本书按下的暂停键 . . . 21
1.7 从这一讲到 2026 . . . 23

第 2 讲　BP 神经网络、反向传播与梯度检查 25
2.1 多层感知器：自动造特征的流水线 . . . 25
2.2 激活函数：没有它，叠一百层也白叠 . . . 26
2.3 梯度下降：在山地里找最低谷 . . . 28
2.4 反向传播：把责任一层层倒推回去 . . . 29
2.5 梯度检查：给你的反向传播“对账” . . . 30
2.5.1 三种求导范式：从手推公式到 autograd . . . 31
2.6 完整训练一次：把循环跑起来 . . . 32
2.7 从这一讲到 2026 . . . 34

第 3 讲　超参数、权重初始化、模型训练与评估 **37**
3.1 损失面与梯度下降：先看清脚下是什么地形 . . . 37
3.2 学习率：整张清单上最重要的一个数 . . . 38
3.3 权重初始化：第一步就走对 . . . 40
3.4 batch、epoch、iteration：分清训练的三兄弟 . . . 41
3.5 训练集、验证集、测试集：背题与会做题 . . . 42
3.6 向量化编程：为什么一行 numpy 顶一百行 for . . . 43
3.7 优化器家族一分钟：从 SGD 到 Adam . . . 44
3.8 从这一讲到 2026 . . . 45

第 4 讲　自编码器 **48**
4.1 无监督学习登场：没有标签的世界 . . . 48
4.2 自编码器结构：自己压缩自己 . . . 49
4.3 隐空间：压缩后的向量是“本质坐标” . . . 50
4.4 去噪自编码器：先弄脏，再学会擦 . . . 51
4.5 变体一分钟与“AE 的当代后代” . . . 52
4.6 从这一讲到 2026 . . . 54

第 5 讲　卷积神经网络原理与前向计算 **57**
5.1 全连接看图：一场参数灾难 . . . 57
5.2 卷积 = 滑动窗口做点积 . . . 58
5.3 padding 与 stride：滑窗的两个旋钮 . . . 59
5.4 池化：只留最强的响应 . . . 60
5.5 特征图与层级：从边缘到猫脸 . . . 60
5.6 完整前向流水线：从像素到类别得分 . . . 63
5.7 从这一讲到 2026 . . . 65

第 6 讲　卷积神经网络的训练与计算模式 **67**
6.1 练手房：MNIST 与 CIFAR-10 . . . 67
6.2 LeNet-5 精读：第一栋“经典公寓”的户型图 . . . 68
6.3 数据增强：白捡的“新数据” . . . 70
6.4 训练循环：五个动作的摩天轮 . . . 72
6.5 批归一化 BN：给每一层的数据“扶正” . . . 73
6.6 评估：准确率会骗人 . . . 74
6.7 过拟合对策清单 . . . 75
6.8 从这一讲到 2026 . . . 76

第 7 讲　卷积神经网络的演化：加宽、加深与残差 **78**
7.1 ImageNet：军备竞赛的“高考考场” . . . 78
7.2 AlexNet：一鸣惊人的三件法宝 . . . 79
7.3 VGG：极简主义——只堆 3×3 . . . 80
7.4 GoogLeNet：多尺度并行与 1×1 省钱术 . . . 82
7.5 深化的障碍：56 层为什么输给 20 层 . . . 83

7.6 残差革命：给梯度修一条高速公路 . . . 83
7.7 设计经验法则：深度、宽度、分辨率的三角 . . . 84
7.8 从这一讲到 2026 . . . 86

第 8 讲 轻量化网络、全卷积网络与卷积操作模式设计 88
8.1 模型压缩四板斧 . . . 88
8.2 深度可分离卷积：轻量化的核心积木 . . . 89
8.3 轻量家族一分钟点名 . . . 91
8.4 全卷积网络 FCN：从“是什么”到“在哪儿” . . . 91
8.5 卷积操作模式工具箱 . . . 94
8.6 端侧部署三件套：ONNX、推理引擎与 INT8 . . . 95
8.7 从这一讲到 2026 . . . 97

第 9 讲 循环神经网络 RNN 99
9.1 序列数据无处不在：世界是按帧来的 . . . 99
9.2 RNN 的核心思想：一个网络用到底，一张便签传下去 . . . 100
9.3 用 RNN 做语言模型：文字接龙 . . . 102
9.4 BPTT：把展开图当普通网络训练 . . . 104
9.5 梯度消失与爆炸：记忆的大敌 . . . 104
9.6 RNN 的应用与局限，以及 2026 年为什么还学它 . . . 105
9.7 从这一讲到 2026 . . . 107

第 10 讲 长短时记忆网络 LSTM 与 GRU 109
10.1 细胞状态：一条直通远方的传送带 . . . 109
10.2 三扇门：该忘的忘，该记的记，该说的说 . . . 111
10.3 LSTM 逐步计算：六行公式，六句人话 . . . 112
10.4 GRU：两扇门的精简版 . . . 113
10.5 实战对比：一张图看懂差距 . . . 115
10.6 2026 年的选型决策：什么时候还用 RNN 家族 . . . 116
10.7 从这一讲到 2026 . . . 117

第 11 讲 注意力模型 119
11.1 为什么要注意力：从 Seq2Seq 的“失忆症”说起 . . . 119
11.2 Q/K/V 三兄弟：一次“查字典”式的信息检索 . . . 120
11.3 自注意力：一句话内部的“圆桌会议” . . . 123
11.4 多头注意力与位置编码：一支足球队而非一个球星 . . . 124
11.4.1 多头注意力：八支荧光笔各画各的重点 . . . 124
11.4.2 位置编码：给天生的“脸盲”贴工位号 . . . 124
11.5 Transformer 全家福：用注意力搭一整栋楼 . . . 125
11.6 规模定律：从 GPT-1 到 2026 年的大模型 . . . 126
11.7 Agent：注意力 + 工具 = 会干活的模型 . . . 128
11.8 从这一讲到 2026 . . . 129

第 12 讲　生成对抗模型 **132**
12.1 生成模型与判别模型：无中生有 vs 看图说话 . . . 132
12.2 GAN：造假者与鉴定师的对决 . . . 133
12.3 训练：一场没有裁判的拉锯战 . . . 134
12.4 GAN 演进长廊：从模糊噪点到以假乱真 . . . 135
12.5 另一条路：变分自编码器（VAE） . . . 136
12.6 后浪：扩散模型 . . . 137
12.7 深度伪造与鉴伪：攻防共生（工程伦理专场） . . . 138
12.8 从这一讲到 2026 . . . 140

第二部分　第二篇　项目背景篇：三个真实项目到底在做什么 **143**

第二篇导读：三个项目全景 **144**
0.9 三个项目：先来一张速览表 . . . 144
0.10 共同的课程框架：节点、证据与底线 . . . 145
0.11 15 个小组地图：找到你自己的位置 . . . 145
0.12 怎么读这一篇 . . . 146

项目一　六目拼接：给六只眼睛一个全景世界 **147**
1.1 需求从哪来：一台相机看不住一座平台 . . . 147
1.2 全景拼接五步曲：从六张歪图到一幅长卷 . . . 151
1.2.1 第一步：畸变校正——先把“哈哈镜”掰直 . . . 151
1.2.2 第二步：特征提取与匹配——在两幅画面里找到“同一个点” . . . 151
1.2.3 第三步：几何对齐——一张 3×3 表格把画面“扶正” . . . 152
1.2.4 第四步：缝合线——让“刀口”藏在最不显眼的地方 . . . 153
1.2.5 第五步：融合——把接缝“羽化”到看不见 . . . 153
1.3 360Stitcher 长什么样：六个模块与一串硬数字 . . . 153
1.4 为什么要“可学习”的参数：九个参数头 . . . 156
1.5 双路线对照：逐个学头，还是整网端到端 . . . 158
1.6 八个组怎么分工：一条流水线上的八个工位 . . . 159
1.7 你将用到第一篇的哪些讲 . . . 160

项目二　水下双目视觉自动标记：让 AI 看懂海底世界 **163**
2.1 场景与需求：甲方要“AI 看一眼就知道有多远” . . . 165
2.2 水下为什么难：光、匹配、泛化、真值四道坎 . . . 166
2.2.1 第一道坎：光被水“吃掉” . . . 167
2.2.2 第二道坎：立体匹配的前提被破坏 . . . 168
2.2.3 第三道坎：深度网络泛化差 . . . 168
2.2.4 第四道坎：没有深度真值 . . . 168
2.3 双目立体视觉：两只眼睛如何“看出”距离 . . . 169
2.4 没有深度真值怎么办：自动打标签（本章高潮） . . . 171
2.4.1 标签本体：一张逐像素视差图 . . . 171
2.4.2 三步流水线：教师打标、多重过滤、学生迭代 . . . 171

2.4.3 最大的先例：NVIDIA 用 140 万对伪标签蒸出快 10 倍的学生 . . . 172
2.4.4 五条路线与前沿模型 . . . 174
2.5 我们的 500 对数据集：一次真实的翻车与自救 . . . 176
2.5.1 翻车：八个“典型场景”全军覆没 . . . 176
2.5.2 自救：两条硬指标把数据“挖”出来 . . . 176
2.5.3 没有标定板的自标定 . . . 176
2.5.4 一个价值 7 倍的工程决策 . . . 176
2.5.5 验收：数字说话 . . . 177
2.6 四个组怎么分工：四条并行赛道，实验四会师 . . . 179
2.7 回头看：你将用到第一篇的哪些本事 . . . 180

项目三 数据基座治理：让海量工程数据开口说话 183
3.1 数据的洪水与资料的迷宫 . . . 184
3.2 让机器“按结构读懂”：数据结构化 . . . 185
3.3 一句话问数：智能体与可追溯查询 . . . 188
3.4 知识门控：AI 抽的知识入库前先过检查 . . . 190
3.5 图谱查关系：知识图谱与图检索 . . . 192
3.6 假内容攻防战（本章高潮） . . . 193
3.7 0.88 的谎言与 0.74 的诚实 . . . 196
3.8 三层身份与验收文化 . . . 198
3.9 回头看：你将用到第一篇的哪些本事 . . . 199

第三部分 第三篇 任务书篇：15 个小组的任务、实验与验收 202

第三篇使用说明：任务书怎么读、怎么交、怎么评 203
0.10 任务书怎么读：固定七节，先通则后本组 . . . 203
0.11 证据怎么留：三件套、交付契约与求助通道 . . . 203
0.12 AI 怎么用：分工通则与两条红线 . . . 204
0.13 成绩怎么算：建议方案（教师可调） . . . 205
0.14 第 19 周周五交什么：总清单模板 . . . 205
0.15 15 个小组速查表 . . . 205

第 1 组 六目拼接 · 匹配决策组（九头之一：匹配与几何） 207
1.1 组档案 . . . 207
1.2 项目背景与你的位置 . . . 207
1.3 核心任务 . . . 208
1.4 第 19 周周五提交清单 . . . 209
1.5 AI 分工与人工必做 . . . 209
1.6 防糊弄指南与满分标准 . . . 209
1.7 四次实验安排 . . . 210
1.7.1 实验一（10 月 12 日）：让大模型替你读代码，你来当裁判 . . . 210
1.7.2 实验二（11 月 2 日）：本机部署，亲眼看见九个头上线 . . . 211
1.7.3 实验三（11 月 23 日）：mc 头真训练、真交付、真对比 . . . 211

1.7.4 实验四（12 月 21 日）：补齐两头，并用新标签复训 mc 211

第 2 组 六目拼接 · 特征质量组（九头之二：特征与时序） 213

2.1 组档案 213
2.2 项目背景与你的位置 213
2.3 核心任务 214
2.4 第 19 周周五提交清单 215
2.5 AI 分工与人工必做 215
2.6 防糊弄指南与满分标准 216
2.7 四次实验安排 216
2.7.1 实验一（10 月 12 日）：读懂规格书，写出规格书 217
2.7.2 实验二（11 月 2 日）：本机部署，外加一次“占位模型”对照 217
2.7.3 实验三（11 月 23 日）：temporal_state 真训练交付，图像头交设计 217
2.7.4 实验四（12 月 21 日）：真机 patch 联调，图像头迁移转真 218

第 3 组 六目拼接 · 缝合融合组（九头之三：缝合与融合） 219

3.1 组档案 219
3.2 项目背景与你的位置 219
3.3 核心任务 220
3.4 第 19 周周五提交清单 221
3.5 AI 分工与人工必做 221
3.6 防糊弄指南与满分标准 221
3.7 四次实验安排 222
3.7.1 实验一（10 月 12 日）：让大模型替你读 C++，你来当裁判 222
3.7.2 实验二（11 月 2 日）：本机部署，顺手记下基线数字 223
3.7.3 实验三（11 月 23 日）：gain_field 真训练交付，seam 头交设计 223
3.7.4 实验四（12 月 21 日）：八头齐，fusion 正式版压轴上场（合并联调） 223

第 4 组 六目拼接 · 数据与标签组 224

4.1 组档案 224
4.2 项目背景与你的位置 224
4.3 核心任务 225
4.4 第 19 周周五提交清单 226
4.5 AI 分工与人工必做 226
4.6 防糊弄指南与满分标准 226
4.7 四次实验安排 227
4.7.1 实验一（10 月 12 日）：读懂标签生产线，画出数据流图 227
4.7.2 实验二（11 月 2 日）：先在基准图上跑通全流程 228
4.7.3 实验三（11 月 23 日）：真机场地采集，交出第一条验收线 228
4.7.4 实验四（12 月 21 日）：扩量与质检，交出最终数据家底 228

第 5 组　六目拼接 · 测试集与评测组 **229**
5.1 组档案 . . . 229
5.2 项目背景与你的位置 . . . 229
5.3 核心任务 . . . 230
5.4 第 19 周周五提交清单 . . . 231
5.5 AI 分工与人工必做 . . . 231
5.6 防糊弄指南与满分标准 . . . 232
5.7 四次实验安排 . . . 233
5.7.1 实验一（10 月 12 日）：让 agent 替你读评测工具，你来写说明书 . . . 233
5.7.2 实验二（11 月 2 日）：两条路线都部署，两种探针都点亮 . . . 233
5.7.3 实验三（11 月 23 日）：冻结测试集 v1，基线入账，回归全绿 . . . 234
5.7.4 实验四（12 月 21 日）：台账 v2 与困难样本，裁判的终场哨 . . . 234

第 6 组　六目拼接 · 端到端数据组 **235**
6.1 组档案 . . . 235
6.2 项目背景与你的位置 . . . 235
6.3 核心任务 . . . 236
6.4 第 19 周周五提交清单 . . . 237
6.5 AI 分工与人工必做 . . . 237
6.6 防糊弄指南与满分标准 . . . 238
6.7 四次实验安排 . . . 239
6.7.1 实验一（10 月 12 日）：读懂端到端的数据餐单，写两路线差异说明 . . . 239
6.7.2 实验二（11 月 2 日）：合成数据全流程，32+8 对先干跑一遍 . . . 240
6.7.3 实验三（11 月 23 日）：首批公开数据集（UDIS-D），50 对起步 . . . 240
6.7.4 实验四（12 月 21 日）：数据扩量与 benchmark 初版，双线收官 . . . 240

第 7 组　六目拼接 · 端到端训练组 **241**
7.1 组档案 . . . 241
7.2 项目背景与你的位置 . . . 241
7.3 核心任务 . . . 242
7.4 第 19 周周五提交清单 . . . 243
7.5 AI 分工与人工必做 . . . 243
7.6 防糊弄指南与满分标准 . . . 244
7.7 四次实验安排 . . . 244
7.7.1 实验一（10 月 12 日）：读懂你将要训练的东西，画出数据流 . . . 245
7.7.2 实验二（11 月 2 日）：本机 CPU 冒烟，三十分钟走通全链路 . . . 245
7.7.3 实验三（11 月 23 日）：4080 真训练，全书唯一的 200 epoch . . . 245
7.7.4 实验四（12 月 21 日）：全量真实数据微调，v2 收官 . . . 246

第 8 组　六目拼接 · 端到端优化组 **247**
8.1 组档案 . . . 247
8.2 项目背景与你的位置 . . . 247
8.3 核心任务 . . . 248

8.4 第 19 周周五提交清单 . . . 249
8.5 AI 分工与人工必做 . . . 249
8.6 防糊弄指南与满分标准 . . . 250
8.7 四次实验安排 . . . 250
8.7.1 实验一（10 月 12 日）：拆解一帧的时间都花在哪，写性能预算书 . . . 251
8.7.2 实验二（11 月 2 日）：引擎模式开起来，逐帧计时基线立起来 . . . 251
8.7.3 实验三（11 月 23 日）：复现第 7 组的公开版 v1，指标对得上账 . . . 251
8.7.4 实验四（12 月 21 日）：三档权衡表与 batch 方案，招牌交付收官 . . . 252

第 9 组 水下双目 · 数据组 253
9.1 组档案 . . . 253
9.2 项目背景与你的位置 . . . 253
9.3 核心任务 . . . 254
9.4 第 19 周周五提交清单 . . . 256
9.5 AI 分工与人工必做 . . . 256
9.6 防糊弄指南与满分标准 . . . 257
9.7 四次实验安排 . . . 257
9.7.1 实验一（10 月 12 日）：给视频开体检单，你亲自看 30 帧 . . . 258
9.7.2 实验二（11 月 2 日）：抽帧去冗一条龙，切分防住泄漏 . . . 258
9.7.3 实验三（11 月 23 日）：数据组也要懂训练——亲眼看 loss 下降 . . . 258
9.7.4 实验四（12 月 21 日）：换配比、调阈值，用验证 EPE 选出更好的一口粮 . . . 259

第 10 组 水下双目 · 部署组 260
10.1 组档案 . . . 260
10.2 项目背景与你的位置 . . . 260
10.3 核心任务 . . . 261
10.4 第 19 周周五提交清单 . . . 262
10.5 AI 分工与人工必做 . . . 263
10.6 防糊弄指南与满分标准 . . . 263
10.7 四次实验安排 . . . 264
10.7.1 实验一（10 月 12 日）：让 agent 开清单，你逐条当审计 . . . 264
10.7.2 实验二（11 月 2 日）：装上 4080，跑通第一对图（本组主力） . . . 265
10.7.3 实验三（11 月 23 日）：微调护航——你的环境要能“载训练” . . . 265
10.7.4 实验四（12 月 21 日）：三组旋钮 + ONNX 初探，给出权衡建议 . . . 265

第 11 组 水下双目 · 训练组 267
11.1 组档案 . . . 267
11.2 项目背景与你的位置 . . . 267
11.3 核心任务 . . . 268
11.4 第 19 周周五提交清单 . . . 269
11.5 AI 分工与人工必做 . . . 270
11.6 防糊弄指南与满分标准 . . . 270
11.7 四次实验安排 . . . 271

11.7.1 实验一（10 月 12 日）：把 yaml 读成人话 272
11.7.2 实验二（11 月 2 日）：让一个 batch 走完全程 272
11.7.3 实验三（11 月 23 日）：先立基线，再开微调（本组主力） 272
11.7.4 实验四（12 月 21 日）：一次只动一个旋钮 272

第 12 组 水下双目 · 评测组 274
12.1 组档案 274
12.2 项目背景与你的位置 274
12.3 核心任务 275
12.4 第 19 周周五提交清单 276
12.5 AI 分工与人工必做 276
12.6 防糊弄指南与满分标准 277
12.7 四次实验安排 278
12.7.1 实验一（10 月 12 日）：先把尺子造准 278
12.7.2 实验二（11 月 2 日）：一键打通“推理 → 视差 → 指标” 278
12.7.3 实验三（11 月 23 日）：两版权重同场比武，每模型不少于 50 对 279
12.7.4 实验四（12 月 21 日）：扫阈值、找最差桶、当个好裁判 279

第 13 组 数据基座 · 数据结构化组 280
13.1 组档案 280
13.2 项目背景与你的位置 281
13.3 核心任务 281
13.4 第 19 周周五提交清单 282
13.5 AI 分工与人工必做 283
13.6 防糊弄指南与满分标准 283
13.7 四次实验安排 284
13.7.1 实验一（10 月 12 日）：让大模型讲原理，你拿样例当裁判 284
13.7.2 实验二（11 月 2 日）：解析环境装通，10 页流水线跑通 285
13.7.3 实验三（11 月 23 日）：自己造数据，训一个“认版面”的小模型 285
13.7.4 实验四（12 月 21 日）：在线大模型救疑难表格，规则法当对照组 285

第 14 组 数据基座 · 智能问答与图检索组 287
14.1 组档案 287
14.2 项目背景与你的位置 288
14.3 核心任务 288
14.4 第 19 周周五提交清单 290
14.5 AI 分工与人工必做 290
14.6 防糊弄指南与满分标准 290
14.7 四次实验安排 291
14.7.1 实验一（10 月 12 日）：当一回 agent，走通“查询-回答-引用” 291
14.7.2 实验二（11 月 2 日）：本地小模型上岗，一句话问数跑通 292
14.7.3 实验三（11 月 23 日）：小图上真训练——两层 R-GCN 做链路预测 292
14.7.4 实验四（12 月 21 日）：本地 vs 在线——30 题上一张对比表 292

第 15 组　数据基座 · 虚假内容识别与对抗训练组 **294**
15.1 组档案 294
15.2 项目背景与你的位置 295
15.3 核心任务 295
15.4 第 19 周周五提交清单 297
15.5 AI 分工与人工必做 297
15.6 防糊弄指南与满分标准 298
15.7 四次实验安排 299
15.7.1 实验一（10 月 12 日）：先当一回受害者——AI 造假你来判 299
15.7.2 实验二（11 月 2 日）：规则当第一道闸，词袋小模型当基线 299
15.7.3 实验三（11 月 23 日）：主力会战——500 条数据集、两个模型、一份自查 . . . 299
15.7.4 实验四（12 月 21 日）：在线判官复核，蒸馏回流再上一分 300

附录：参考资料与图片来源 **301**

第一部分

第一篇　理论篇：十二讲，从感知器到大模型

第 0 讲　开课导引：这门课怎么上、怎么学、怎么考

欢迎来到《深度学习与图像识别》。在翻开第 1 讲之前，请先把这一章读完——它回答你此刻最关心的四个问题：这门课到底讲什么？我要加入的真实项目是什么？开学前该准备什么？以及，怎么让大模型帮你学编程。最后一个问题是全书最重要的方法论，没有之一：这门课不禁止 AI，恰恰相反，**这门课要求你用 AI**——但用的是门手艺，得先学。

本章地图

讲什么：课程全貌（12讲主线与 4 次实验）、三个真实项目速览、开学前的准备清单、与大模型协作学编程的方法、零基础数学包与零基础 Python 包、全书地图。**为什么值得读**：这门课一半的分数在项目里，项目的活儿全靠 AI 协作完成——先弄清“怎么用 AI”和“需要什么基础”，后面 12 讲才不会手忙脚乱。**学完你能**：装好环境、跑通第一行 Python、写出让 AI 听懂需求的提示词、拿着全书地图知道自己每一步在往哪儿走。

0.1　这门课讲什么：从一个神经元到大模型

如果把整门课压缩成一句话，那就是**从一个神经元，走到大模型**。12讲内容基本按历史顺序展开，恰好重演了人工智能七十多年的主线：1958 年的感知器（第 1 讲）长出了多层结构（第 2 讲），学会了正经的训练流程（第 3 讲）和自我压缩（第 4 讲）；1970 年代末卷积思路登场，网络第一次真正“看懂”图像（第 5–8 讲）；随后网络学会处理序列与语言（第 9–10 讲），最终靠注意力机制（第 11 讲）走到了今天的大模型，还学会了“无中生有”地生成图像（第 12 讲）。每讲都配有课后编程题——题目统一要求你**指挥 AI 实现**本讲核心内容的一个小应用。

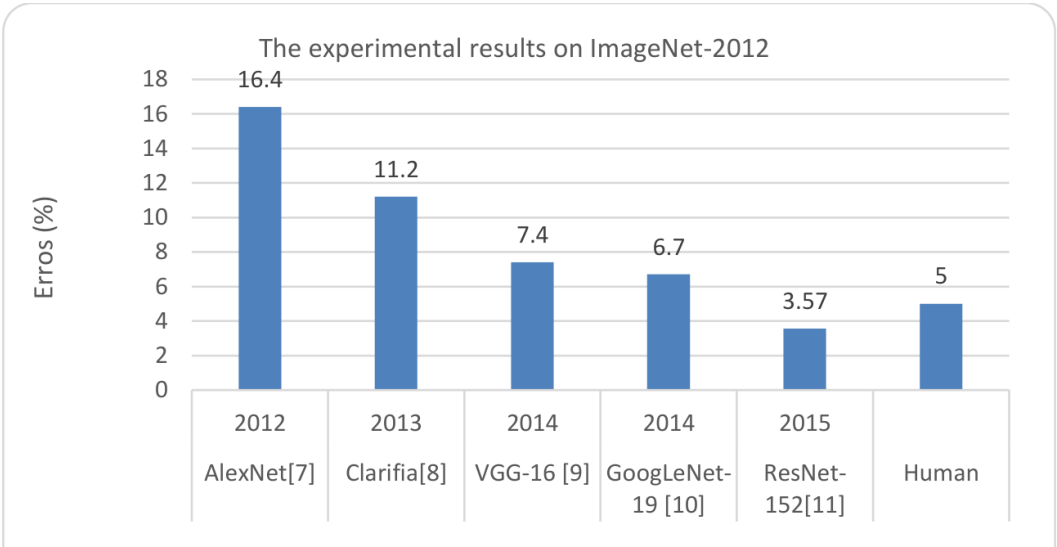


图 1　深度学习七十余年发展时间线：从 1943 年的神经元数学模型，到 2006 年“深度学习”命名，再到 2020 年代的大模型

图片来源：维基百科 History of artificial neural networks 词条

下表是 12 讲的路线图，右列是配套教材（许庆阳《深度学习模型与算法基础》，清华大学出版社 2023 年 12 月版）的课后阅读页码——每讲上课前请把对应页码过一遍，带着问题来听。

表 1　12 讲路线图：主题、一句话与教材页码

讲次	主题	一句话	教材页码
1	感知器、线性单元与学习规则	一个神经元如何从错误中学习	11–19
2	BP神经网络、反向传播与梯度检查	把神经元叠起来，误差“责任倒推”	20–30
3	超参数、权重初始化、模型训练与评估	训练的“火候”与考试的“卷子”	31–35、48–50
4	自编码器	没有标签，网络自己学会压缩	35–48
5	卷积神经网络原理与前向计算	用滑动窗口看图像	50–70
6	卷积神经网络的训练与计算模式	数据、增强与训练技巧	71–85
7	卷积神经网络的演化	从 LeNet 到 ResNet 的三十年	86–100
8	轻量化网络、全卷积网络	把网络塞进手机，看清每个像素	101–115
9	循环神经网络 RNN	会顺着序列思考的网络	167–186
10	长短时记忆网络 LSTM 与 GRU	给网络装上带闸门的记忆	187–203
11	注意力模型	大模型时代的地基	204–229
12	生成对抗模型	AI 如何“无中生有”画图	230–248

再说教学形式。本课程 2 学分、32 学时，拆成两半：**24 学时随堂讲授**（就是第一篇的 12 讲，每次 2 学时讲一讲），以及 **8 学时的 4 次现场实验**。实验不是“照着指导书按按钮”的验证作业：第一次实验你就把大模型编程助手配置进自己的编辑器，此后所有实验和项目作业都默认你在与 AI 协作。四次实验的时间与内容如下。

表 2　四次现场实验一览（实验二至四各 10 分共 30 分，实验一不计分；平时作业 20 分、项目成果 50 分，总评满分 100 分）

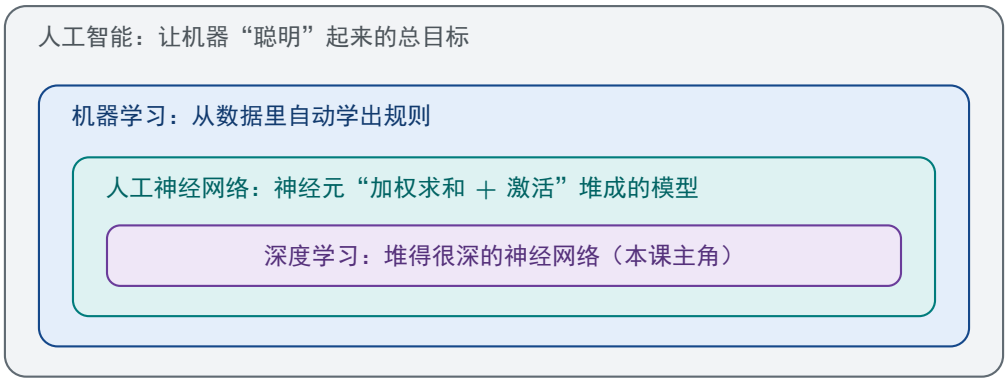
序号	时间	内容
实验一	10 月 12 日	大模型 agent 模式配置及应用：在 VS Code 中安装 CLINE 编程助手，连接 GLM 或 DeepSeek 大模型 API
实验二	11 月 2 日	匹配合适程序并执行本地部署：用 Ollama 在自己电脑上运行开源大模型、编译项目程序
实验三	11 月 23 日	深度网络的模型训练实践：亲手跑通第 2–3 讲的训练循环
实验四	12 月 21 日	模型优化训练实践：把训练跑得又快又稳

除了课堂与实验，你的每周预算里还有 **6–8 小时的小组项目时间**：全班按 5 人一组分成 15 个小组，分属三个真实工程项目（下一节就介绍）。第 19 周周五（2027 年 1 月 22 日）全部成果提交验收。注意这个时间结构：**项目不是课程结束后的“大作业”，而是与课程同步进行的“实习”**——开学头几周你就会进入项目组、拿到任务书；此后每次课上学的每一样东西，几周内都会在你的项目里用到。

0.1.1　先排座次：人工智能、机器学习、神经网络与深度学习

正文开始之前，先把四个最出镜的词排好座次。它们不是四门并列的学科，而是**一圈套一圈的包含关系**：人工智能 ⊃ 机器学习 ⊃ 人工神经网络 ⊃ 深度学习。各用一句话认个脸：

- **人工智能（AI）**：最大的圈——让机器做出类似人类智能行为的一切尝试的总称，这个名字是 1956 年达特茅斯会议起的；
- **机器学习（ML）**：人工智能里最成功的一个流派——不靠工程师手写规则，让机器从数据里自己把规则学出来；
- **人工神经网络（NN）**：机器学习里的一种模型——用第 1 讲那种“加权求和、过阈值激活”的神经元堆叠连接而成；
- **深度学习（DL）**：把神经网络堆得很深（很多层）——本课程的主角，也是 2026 年所有大模型的底座。



零基础扫盲：两个教材高频词——“表示学习”与“端到端学习”

配套教材开篇会反复出现这两个词，这里提前用人话讲一遍，全书统一按这个意思使用。**表示学习（representation learning）**：同样一堆东西，“怎么记”差别巨大。早期的 One-hot 表示像一本超长点名册：一万个类别就造一个一万维向量，只有代表自己的那个位置是 1、其余全是 0——占地方，而且任何两个类别之间“谁跟谁像”的信息量为零。**分布式表示**换成另一条路：用一小段稠密数值（比如 64 个数）拼出万物，每个维度都在说话，含义相近的东西向量也挨得近。把“点名册”学成这种稠密向量的过程，就叫**特征嵌入**（embedding）——大模型里“词向量”三个字的来历。一句话：**好的表示不是人定的，是学出来的**；第 4 讲整讲的训练，练的就是这门功课。**端到端学习（end-to-end learning）**：老办法把一个任务切成好几段，每段手工设计一个算法、各自调参、层层接力；端到端学习把整条流水线换成一个网络——原始数据从这头进，最终答案从那头出，中间全部交给训练自动打磨。一句话：从“层层外包”到“一家全包”。你们六目拼接项目里那条“端到端路线”（第 一 章），名字就是这么来的。

0.2　三个真实项目一页速览

这门课最大的不同，是你们要干的活儿来自三个**正在运行的真实工程项目**，而不是教科书后面的习题。下图是整门课 16 周的全景时间轴：绿色是 12 讲理论课，橙色是 4 次实验，蓝色是项目关键节点，红色是第 19 周的提交截止线——四条线并行推进，这张图值得贴在宿舍墙上。



图 2　课程 16 周全景时间轴：12 讲理论课（绿）、4 次现场实验（橙）、项目关键节点（蓝）与第 19 周成果提交（红）

三个项目的速览如下表，随后每个项目用一段人话介绍。更完整的背景（甲方是谁、现场长什么样、难点在哪）请阅读第二篇各章，那里每个项目有一整章。

表 3　三个真实项目速览（15 个小组，每组 5 人）

项目	甲方 / 背景	做什么	组数	关键词
六目拼接	360Stitcher 系统总体组	六路相机实时 360 度全景拼接：把九类人工调参升级为九个可学习的“参数头”神经网络	8	实时全景、ONNX、回退保护
水下双目自动标记	中海油 HYSY123 FPSO	为 ROV 水下检查视频建立双目立体感知：大模型“教师”打伪标签，训练轻量“学生”模型	4	双目立体、伪标签、锚链
数据基座治理	海洋石油平台	治理 126 份 CSV 共 3.1 亿行运行数据与 1001 页工程手册：结构化、一句话问数、虚假内容识别与对抗训练	3	数据治理、智能体、R-GCN

项目一：六目拼接。想象六只“眼睛”围成一圈盯着海上平台四周，各自拍一路视频，系统要把六路画面实时拼成一幅无缝的 360 度全景。传统拼接算法里有九类参数（畸变校正、特征匹配、对齐、融合……），过去全靠工程师一把一把手工调，换个相机、换个光照就要重调一遍。这个项目做的事，是把这九类参数变成九个**可学习的神经网络“参数头”**：模型自己算参数，系统负责拼接，算完还要转成 ONNX 格式交付，并配一道“回退保护”——模型万一失手，系统自动退回老算法，现场一分一秒都不能停。项目里另有一条端到端路线作对照，用 RTX 4080 训练。详见第 一 章。

项目二：水下双目视觉自动标记。HYSY123 是一座漂浮在海上、排水量十几万吨的浮式生产储卸油装置（FPSO），它靠几条巨大的系泊锚链“锚”在油田上方。锚链要靠水下机器人（ROV）定期下水检查，但 ROV 拍回的视频浩如烟海，人工打标签慢、贵、还不稳定。这个项目用**双目相机**的左右两幅画面算“视差”、反推深度，让机器自动识别锚链的每一环；再用 FoundationStereo 这样的大模型当“教师”给画面打伪标签，过滤掉不可靠的，拿来训练轻量的“学生”模型——你们还要自己搭建 500 对锚链立体图对作为家底。详见第 二 章。

项目三：数据基座治理。某海洋石油平台的运行数据是 126 份 CSV 文件、共 3.1 亿行，配上 1001 页工程手册——靠人翻，一辈子翻不完。这个项目干三件事：把散乱的数据和手册**结构化**；搭一个**“一句话问数”**的智能体（工程师说人话，Gemma 大模型加 DuckDB 数据库加 R-GCN 图检索负

责听懂并查数）；以及**虚假内容识别与对抗训练**——万一有人往系统里喂假数据怎么办？那就训练一个“监考员”模型来抓，造假方与监考方反复过招、共同变强。详见第 三 章。

项目连线：理论 12 讲与三个项目的关系

不要小看速览表里那几行字，这门课的每一讲都有一条暗线通向它们：第 1–3 讲的神经网络基础，是九个参数头、学生模型、监考员模型的共同骨架；第 5–8 讲的卷积网络，负责六目拼接的图像特征与水下画面理解；第 9–11 讲的序列模型与注意力，是“一句话问数”智能体的引擎；第 12 讲的对抗思想，正是数据基座“造假者 vs 监考员”拉锯战的理论原型。**第一篇每讲末尾的“项目连线”盒子会反复提醒你这条暗线**——学理论时知道自己为什么学。

0.3　你需要准备什么

工欲善其事，必先利其器。第一次实验（10月 12 日，配置大模型编程助手）不允许空手进场，请在开学两周内对照下面的清单备齐装备。总共四样：**一台能装 Python 的电脑**、**一个大模型 API 账号**、**20 GB 硬盘空间**，以及最重要的——**好奇心**。

表 4　电脑最低配置与账号清单

项目	最低要求	说明
CPU	4核	无特殊要求
内存	8 GB	16 GB 更从容
硬盘可用空间	20 GB	放 Python 环境、模型文件与项目数据
显卡	不强制	有 NVIDIA 显卡（8 GB 显存以上）跑本地模型更快；没有也完全能完成本课
操作系统	Windows 10/11 或 macOS	实验机房为 Windows
大模型 API 账号	GLM 或 DeepSeek 任一	GLM 开放平台（open.bigmodel.cn）或 DeepSeek 开放平台（platform.deepseek.com），注册即有免费额度，实验一要用

这里解释一下“API 账号”是什么。大模型跑在厂商的机房里，你的程序通过 **API**（Application Programming Interface，应用程序接口）把问题发过去、把答案收回来，就像点外卖：厨房在远处，你在窗口下单取餐。注册后你会拿到一串“钥匙”（API Key），实验一里把它填进编程助手，AI 就在你的编辑器里上岗了。除了云端点餐，课程后半段还会学**本地部署**——用 Ollama 把开源模型整个搬到自己电脑上跑（实验二），那时你的笔记本就是厨房。

准备好了硬件，接下来干第一件实事。

动手时间：第一件事，跑通你的第一行 Python

目标：在一台全新电脑上，看到你自己跑出来的第一行程序输出。全程允许（鼓励）问 AI。

步骤：(1)按下一节 0.6 的四步装好 Anaconda 与 VS Code；(2) 打开 VS Code，新建文件 `hello_ai.py`，敲入下面一行；(3) 点右上角“运行”，在下方终端里看到输出。

```
1  print("Hello AI")   # print 是"打印": 把括号里的内容显示到屏幕
```

如果卡住了，把这句话发给你的 AI 助手（没装助手就先在网页版大模型里问）：

> “我是完全零基础的学生，电脑是 Windows 11，从没装过 Python。请一步一步教我安装 Anaconda

和 VS Code，并运行我的第一个程序 `print("Hello AI")`。每一步告诉我点什么按钮，出任何报错我会把原文发给你，你要先用一句话告诉我错误原因，再给修改办法。”

验收：屏幕上出现 `Hello AI`。就这么简单——但请注意，这句话已经被你“指挥 AI 从零搭环境”跑通了，这正是四次实验和全部课后题的标准工作方式。

0.4　如何用大模型帮你学编程（本书最重要的方法论）

先说结论：这门课里，**“会写提示词”比“会背语法”值钱**。你不需要从零手写每一行代码，但你必须能看懂 AI 写了什么、能改动它、能向别人讲清楚它为什么这么写。本节把这套方法拆成四件事：工具、提示词、agent 循环、以及不可逾越的底线。

0.4.1　AI 编程助手是什么

VS Code 是微软出品的免费代码编辑器，是目前 AI 编程生态的主战场；**Cline** 是运行在 VS Code 里的开源 AI 编程助手插件（官网 cline.bot），背后连接你配置的大模型（GLM 或 DeepSeek）。它在对话框里听你说人话，然后替你新建文件、写代码、保存、运行、看报错、自己修改、再运行——直到任务完成。你在整个过程中的角色，从“打字员”变成了“提需求、做验收的甲方”。

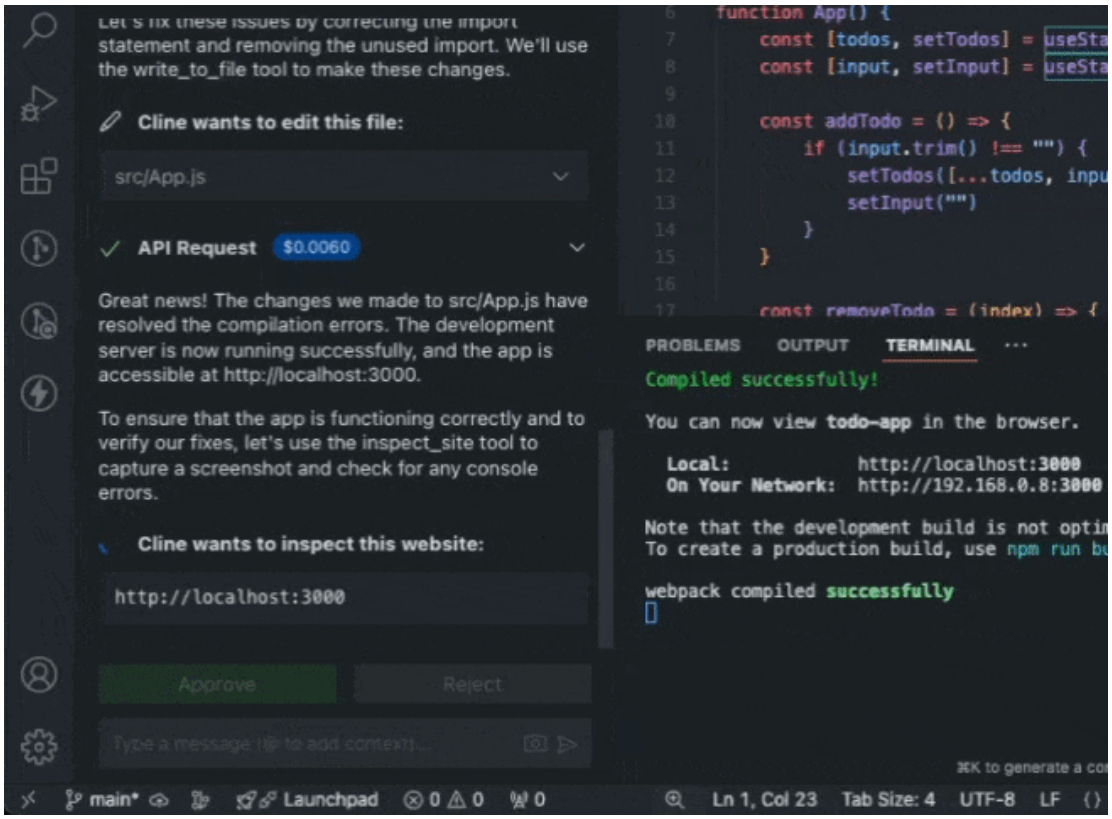


图 3　VS Code 与 Cline AI 编程助手：左侧对话框下达任务，助手自动创建并修改右侧的代码文件，还能代为执行命令

图片来源：CLINE 开源项目官方文档 cline.bot

0.4.2　提示词怎么写：四个要素

提示词（prompt）就是你发给 AI 的指令。写得含糊，AI 就用最平庸的方式糊弄你；写得具体，它就是顶尖助教。四个要素，配一个“反例 / 正例”对照表：

1. **具体**：说清语言、库、输入输出、参数，别让 AI 猜；
2. **给上下文**：你的系统、已装的环境、学过什么、报错原文；
3. **要解释**：要求逐行中文注释、解释每步在干嘛——这是你“看得懂”的保险；
4. **迭代追问**：第一次不满意就继续调，像带实习生一样带它。

表 5　提示词四要素的“反例 / 正例”对照

要素	反面示例	正面示例
具体	“帮我写个神经网络”	“用 numpy 写一个 2-2-1 全连接网络，tanh 激活，平方损失，学习率 0.1”
给上下文	（什么都不说）	“我的电脑是 Win11 + Anaconda，只学过 C 语言，numpy 没装过”
要解释	“直接给我代码”	“每行加中文注释，写完用三句话讲清整体思路”
迭代追问	不满意就放弃	“把学习率改成 0.01 再跑一遍，并解释为什么这次收敛更稳”

0.4.3　agent 循环：AI 是怎么“自己干活”的

你可能会奇怪：AI又看不到你的屏幕，它怎么帮你把程序跑通？答案是它运行在一个**循环**里：**思考**（分析当前局面）→ **行动**（写文件、执行命令）→ **观察**（读取运行结果和报错）→ 再思考……这个“思考–行动–观察”的循环（下图）就是 2026 年所有 AI 编程助手的核心机制，学术源头是 2022 年的 ReAct 论文。Cline 把这个循环用在了你的项目目录上：它改一行代码就编译一次，看见报错自己分析自己改，跑通了才向你汇报。第一次实验（10 月 12 日）你要亲手把这套循环配置起来。

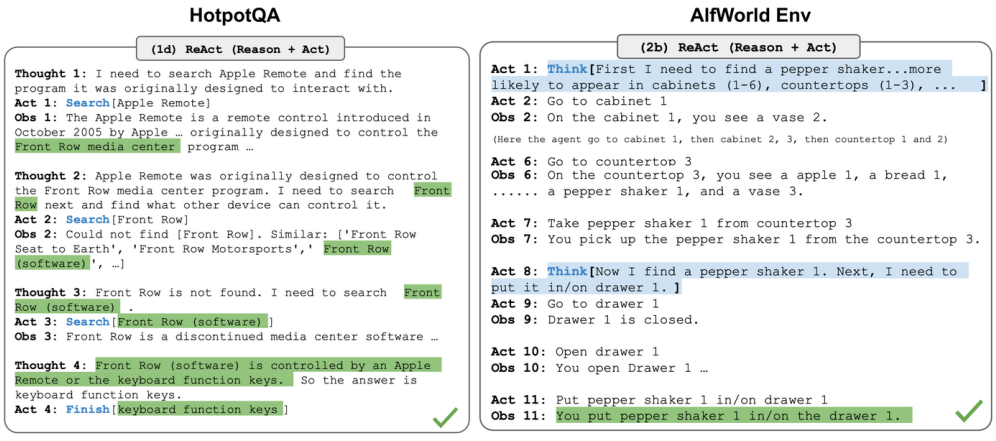


图 4　大模型智能体（Agent）的工作循环：思考（Reasoning）→ 行动（Acting）→ 观察（Observing），循环往复直到完成任务

图片来源：ReAct: Synergizing Reasoning and Acting in Language Models, Yao et al., 2022

前沿速递：从“补全下一行”到“自己干活”（2024–2026）

如果你在 2023 年用过 AI 写代码，印象大概是“高级自动补全”：打半行，它猜半行。2024 年起，风向变了——编程助手普遍进化为 **agent 模式**：接下完整任务，自己拆解、自己改文件、自己跑测试。2025 年，“氛围编程”（vibe coding）一词被造出来并迅速当选各大词典年度热词：人只描述意图和验收标准，AI 写代码，人负责把关——这正是本课要求你的工作方式。衡量这种能力的权威基准（在真实软件仓库里修复真实 bug 的 SWE-bench 等）成绩在 2024–2026 年间逐年跳涨，头部大模型在标准工程任务上的修复率已从两成上下爬到与熟练工程师相当的水平。你在实验一里配置的 agent，就是这条演化路线的最新形态。顺便说一句：本讲义的部分代码示例与全部插图脚本，也是作者与大模型协作完成的——这本身就是教学内容。

底线必须先讲清楚。

AI 可以用，结果自己扛

三条纪律，违反任何一条，视同作业未完成：

1. **不许只抄不看**。每个交付物都可能被随机抽问，讲不清楚就是没做。代码是 AI 写的没关系，**讲不清楚就是没做**。
2. **实验数字不许抄 AI 给的**。准确率、误差、运行时间，必须是你自己的代码在你的数据上跑出来的，交付要有“命令 + 日志 + 产物”三件套证据。
3. **AI 会一本正经地胡说**。这叫幻觉（hallucination）：它可能给你不存在的函数、虚构的结论、“99% 的精度”。它说什么不算数，你跑出来才算数。

工程世界里，说谎的代价不是扣分，是事故。这门课从第一行代码起就按工程标准要求你。

最后，既然整本书都建立在“大模型”上，那大模型本身是什么？这里先用三句话扫个盲，原理在第 11 讲细讲。

零基础扫盲：大模型到底是个啥？三句话

第一句：它本质是一台**超大规模的文字接龙机器**——你说上半句，它计算“下一个字接什么最合适”，一个字一个字往外蹦。**第二句**：它“读”过互联网上的海量文本，训练过程就是反复调整内部几千亿个**参数**（可以想象成几千亿个“重视程度”旋钮），把“接得像”磨成“接得对”。**第三句**：它靠**注意力机制**在长文本里挑重点（哪句话和当前问题有关就多看哪句），这是它能力的关键，也是第 11 讲的主角。就这些——不神秘，也没有“意识”，但它足够好用，好用到你这门课的成绩里有五十要靠它。

0.5　零基础数学包：够用就好的三小块

你们学过高等数学、线性代数、概率论，但多半已经还给老师了。没关系——这本书用到的数学就三小块，而且每一块都只需要“会算、能理解”的程度。以下三个蓝色盒子现场补课，后文用到时会回指。

知识补齐：向量与点积——“合拍程度”打分器

向量（vector）就是一串有序的数，例如 $\mathbf{a}=(1,2)$，几何上画成一支带方向的箭头。**点积（dot product）**：两个向量对应位置相乘再相加：

$$\mathbf{a}\cdot\mathbf{b} \;=\; a_1b_1+a_2b_2+\cdots+a_nb_n. \tag{1}$$

例如 $\mathbf{a}=(1,2)$、$\mathbf{b}=(2,1)$，点积 $=1\times2+2\times1=4$。**人话翻译：**点积是给两个向量的“合拍程度”打分——方向越一致分越高，互相垂直得 0 分，方向相反得负分。下图里两支箭头的夹角越小，点积越大。第 1 讲的感知器，干的就是这件事：拿“权重向量”给“样本向量”打合拍分。

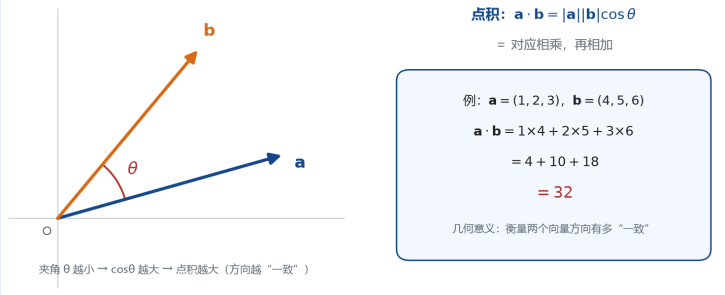


图 5　向量与点积示意：两个向量做“对应相乘再相加”；夹角越小（方向越一致），点积得分越高

知识补齐：导数与链式法则——“变化率”与“变化率相乘”

导数（derivative）：函数此刻的变化率，几何上是曲线在该点的斜率，记作 $f'(x)$ 或 $\frac{\mathrm{d}y}{\mathrm{d}x}$。导数大，意味着 x 动一点、y 就动很多。例如 $f(x)=x^2$ 的导数是 $f'(x)=2x$：在 $x=3$ 处斜率为 6。**链式法则（chain rule）**：如果 y 随 u 变、u 又随 x 变（复合函数），那么总变化率等于两段变化率相乘：

$$\frac{\mathrm{d}y}{\mathrm{d}x}=\frac{\mathrm{d}y}{\mathrm{d}u}\cdot\frac{\mathrm{d}u}{\mathrm{d}x}. \tag{2}$$

人话翻译：像汇率换算——人民币兑美元一段汇率，美元兑日元一段汇率，两段相乘就是人民币兑日元的总汇率。下图里 x 先影响 u、u 再影响 y，导数就沿着箭头一路乘过去。第 2 讲的反向传播，本质就是链式法则在网络上的规模化使用。

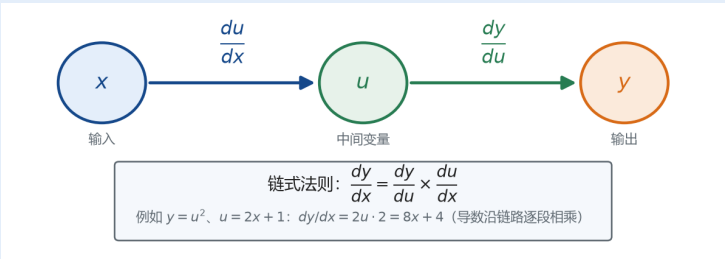


图 6　链式法则示意：x 先影响 u、u 再影响 y，总变化率 = 两段变化率相乘，导数沿箭头传递

知识补齐：概率三句话

第一句：概率（probability）是可能性的度量，取值 0 到 1；所有可能情况的概率加起来等于 1。**第二句：**条件概率 $P(A\mid B)$ 读作“在 B 已经发生的前提下，A 发生的概率”。比如“阴天时下雨”的概率就比“平时下雨”大——前提改变了判断。模型说“这张图是猫的概率 0.97”，说的就是一

个条件概率（前提是这张图的像素）。**第三句：**期望（expectation）是加权平均：$E[X]=\sum_i p_i x_i$，把每种结果按发生的可能性加权平均，回答“平均而言会怎样”。第 3 讲评估模型时，“平均误差”就是期望。

0.6　零基础 Python 包：从装环境到会读代码

为什么这门课用 Python，而不是你们学过的 C++？一张表说清分工：

表 6　Python、C/C++ 与 AI 工具链在本课中的分工

工具	它是什么	在本课的角色
Python	语法接近自然语言的解释型语言	**主力**：写模型、调库、画图；也是大模型最擅长写的语言
C / C++	编译型语言，你们已学过	**底层**：PyTorch、ONNX运行时的内核是 C++，被 Python 调用，本课不必直接写
Anaconda	Python 的“全家桶”发行版 + 环境管理器	**地基**：统一全班环境，避免“我电脑上装不上”
VS Code + Cline	编辑器 + AI 编程助手插件	**结对程序员**：你说需求，它写代码、跑程序
Ollama	本地大模型运行器	**实验二**：把开源模型搬回自己电脑跑

环境安装四步走，跟着下图从左到右：(1) 官网下载并安装 Anaconda（一路下一步）；(2) 安装 VS Code，在插件市场装 Python 和 Cline 两个插件，并在 Cline 里填入你的 API Key；(3) 新建一个 `.py` 文件；(4) 点运行按钮。哪一步卡住，就把报错原文发给 AI（见 0.3 节的求助提示词）。

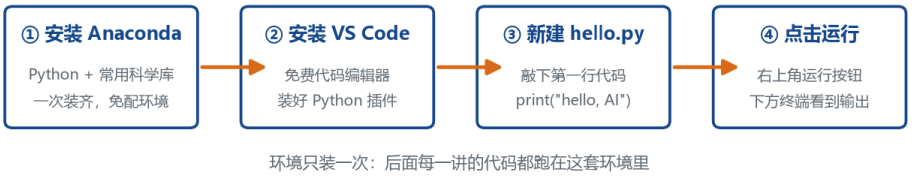


图 7　Python 环境安装四步：安装 Anaconda → 安装 VS Code 与插件 → 新建 `.py` 文件 → 一键运行

环境有了，最后补一条最薄的底：读代码。你不需要会写，但需要看懂。下面 13 行覆盖了本课代码九成以上的语法，每行都值得读三遍：

```
x = 3                       # 变量：直接起名字赋值，不用声明类型
pi = 3.14159                # 小数叫浮点数
name = "robot"              # 引号包起来的是字符串
scores = [90, 85, 77]       # 列表 list：一串值排好队，下标从 0 开始
scores[0]                   # 取第 1 个：90（注意是 0 不是 1！）
info = {"name": "g01", "n": 5}  # 字典 dict：键值对，像查表
info["n"]                   # 按键取值：5
if x > 0:                   # if 判断：注意行末冒号
    print("正数")            # 缩进 4 格表示"属于 if"——缩进就是语法！
for s in scores:            # for 循环：逐个取出列表里的元素
    print(s)                # 会打印 90、85、77 三行
def add(a, b):              # def 定义函数：输入 a、b，返回 a+b
    return a + b
```

```
import math                 # import 引入现成的工具库
math.sqrt(16)               # 调用库函数：4.0
```

三条铁律：**冒号 + 缩进**定归属（缩进错了程序就错）；**下标从 0 开始**；**报错先读最后一行**。除此之外的语法，遇到就问 AI，现学现用完全来得及。

零基础扫盲：报错了怎么办？

初学者看见满屏红字的第一反应是“完了”。其实报错是程序在跟你说话，三步应对：**(1) 倒着读：**Python 报错（Traceback）最后一行才是病因——`NameError` 多半是名字拼错，`IndentationError` 是缩进不对，`TypeError` 是拿数字当字符串用了。**(2) 整段复制给 AI：**把报错原文和你的代码一起发过去，问“哪错了、为什么、怎么改”，这是最高效的排错方式。**(3) 实在不行就搜索：**把报错信息原样粘进搜索引擎，九成错误别人都犯过。记住：**报错不是失败，是免费的纠错教练。**

0.7　全书地图：三篇怎么配合用

最后交代全书的结构。全书三篇对应课程三个阶段，用法各不相同：**第一篇（理论篇，12 讲）**随堂推进，是每天的主粮——每讲有故事、有公式、有人话翻译、有防糊弄提醒，讲末有一道“用 AI 实现”的课后编程题和一张术语表；**第二篇（项目背景篇）**三个项目各一章，建议开学前两周读完，你拿到任务书时才不会两眼一抹黑；**第三篇（任务书篇）**是 15 个小组的作战地图，含每组任务、提交清单、AI 分工与验收标准，以及四次实验的命令级步骤——这一篇是工具书，不必通读，但每个节点都要对照执行。

正文之前，还要统一三个词的日常用法，全书不再重复解释。**“网络”**（或“神经网络”）：一个由大量简单计算单元组成的函数——吃进数字，吐出数字，中间全是乘法、加法和一点点非线性，它不是生物意义上的网络。**“模型”**：训练好参数的网络，物理上就是一个几 MB 到几百 GB 的文件；平时说“下载一个模型”，就是下载这份装满参数的文件。**“训练”**：用数据自动调整网络参数的过程——第 1–2 讲你会亲手做这件事，做完你就再也不会觉得这个词玄乎了。其余术语在各讲首次出现时都会给“中文（English）”和一句人话，章末术语表可随时回查。

大模型（Large Language Model, LLM）
参数规模巨大、在海量文本上训练的语言模型，能接龙式地生成文字，也能被改造去写代码、看图。

参数（parameter）
网络内部可调整的数值，多数是“权重”与“偏置”。训练就是调参数；参数量是模型规模的第一指标。

训练（training）
用数据自动调整参数、让网络输出越来越接近正确答案的过程。

推理（inference）
训练完成后用网络做预测的过程——“跑模型”通常指推理。

数据集（dataset）
一批带标注（或不含标注）的数据，训练与评估的原料。

API
应用程序接口；这里特指通过网络调用云端大模型服务的方式，需要注册账号并持有密钥。

提示词（prompt）

发给大模型的指令文本。提示词质量直接决定产出质量，0.4 节的四要素是本书的通用写法。

智能体（Agent）

让大模型进入“思考–行动–观察”循环、能使用工具（写文件、跑命令）自主完成任务的系统，如 Cline。

幻觉（hallucination）

大模型一本正经地生成错误内容的现象。对策：验收、复算、要证据。

本地部署（local deployment）

把开源模型下载到自己电脑上运行（工具如 Ollama），与调云端 API 相对；实验二的内容。

第 1 讲　感知器、线性单元与学习规则

1958年夏天，美国海军在华盛顿召开发布会，介绍一台“会自己学习的机器”。《纽约时报》的报道读起来激动得不像话：海军造出了一种电子计算机的雏形，**预计它有朝一日能走路、说话、看见、写字、自我复制，并意识到自己的存在**。这台机器叫“感知器”（Perceptron），主持建造的是康奈尔航空实验室的心理学家罗森布拉特（Frank Rosenblatt）。它当然没有这么神——事实上它只会做一件小事：盯着一张小图，分辨左右。但把夸张的新闻稿放到一边，这台机器身体里的核心想法——**加权和、阈值、从错误中调整权重**——一路演化了六十多年，变成了你今天可能天天在用的大模型。本讲的任务，就是拆开这台被吹上天的机器：看看它身体里装了什么想法，是什么让它栽了大跟头，又是什么让它笑到了最后。

那台机器后来被称作 Mark I 感知器，是一台你听得见“学习声音”的实体设备：400 个光导管负责“看”像素（这就是它的输入），每个输入连着一个由小马达带动的电位器——一个会自动旋转的旋钮，旋转角度就存着**权重**；训练时马达真的会转动旋钮，满屋子咔哒作响。1962 年罗森布拉特在书里写下的一句话，放在今天依然成立：“感知器的产生，不是为了模仿大脑的每一个细节，而是为了回答一个问题：机器能不能从自己的错误中学到东西？”

本讲地图

讲什么：生物神经元的工作方式、感知器的数学形式与决策边界、感知器学习规则、从 0/1 输出到连续输出的线性单元与最小均方、以及 1969 年的 XOR 危机与第一次 AI 寒冬。**为什么值得学**：感知器是所有神经网络的最小单元——本课三个项目里的每一个模型，剥到最里层都是它。**学完你能**：徒手算出一个感知器的输出；解释学习规则里每一项的含义；说清平方损失为什么长那样；给同学讲明白“第一次 AI 寒冬”的来龙去脉。

1.1　生物神经元：大脑的原始零件

要造人工神经元，先看真品长什么样。人脑约有 860 亿个神经元（neuron），每个神经元平均与几千个同伴相连。它的结构可以粗略想成三段：**树突**（dendrite）像无数根天线，负责从别的神经元接收电化学信号；**胞体**（soma）负责汇总所有收到的信号；当累计信号**超过某个阈值**时，胞体“点火”，沿着**轴突**（axon）——一根细长的输出线——发出一个电脉冲，把信号传给下一批神经元的树突。注意这个机制里最关键的两点：第一，多个输入要**加在一起**跟阈值比；第二，超阈值才放电，不超就沉默——“全或无”。

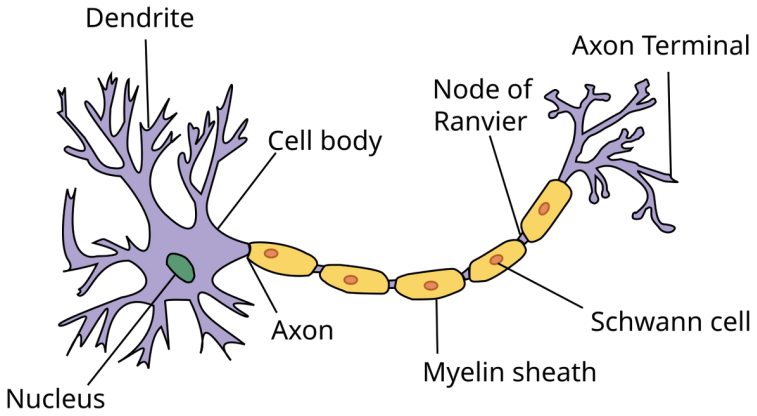


图 1.1　显微镜下的生物神经元：从胞体向外伸展的大量细长突起是接收信号的树突，那根最长的输出突起是轴突

图片来源：维基百科 Neuron 词条

那“学习”发生在哪里？生物层面，学习主要是调整**突触**——神经元之间的连接——的**强度**：常用的一条通路越用越通畅，不用的逐渐萎缩。1943年，神经生理学家麦卡洛克（McCulloch）与逻辑学家皮茨（Pitts）把上面这套机制抽象成了一个纯数学对象：**把每个输入乘上一个“强度”，加总，再和一个阈值比**——这就是神经元的第一个数学模型。下图把生物神经元和它的数学替身并排摆在一起，一一对应；本讲的余下部分，就是把右边这个数学替身讲透。

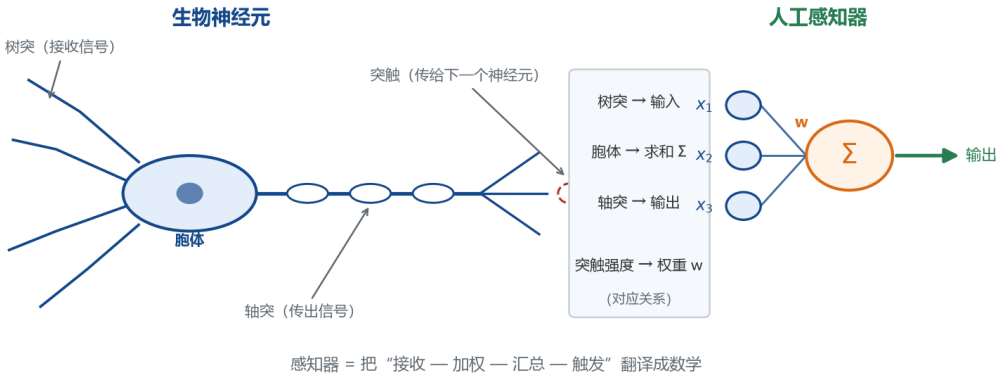


图 1.2　生物神经元与感知器对照：树突接收多路信号 ↔ 多路输入；胞体加权汇总 ↔ 求和节点；超过阈值轴突放电 ↔ 阶跃激活输出 0 或 1；突触连接强度可变 ↔ 权重可学习

1.2　感知器：一颗人工神经元

罗森布拉特做的，就是把 1943 年的抽象模型变成一台能学习的机器。感知器的全部计算只有一行：

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right), \qquad f(z) = \begin{cases} 1, & z \geq 0, \\ 0, & z < 0. \end{cases} \tag{1.1}$$

人话翻译：收 n 路输入 $x_1, \ldots, x_n$，每路乘上自己的权重（weight）w_i，加总后再加一个偏置（bias）b，最后过一道“及格线” f——信号总和不低于 0 就输出 1，否则输出 0。函数 f 叫激活函数（activation function），感知器用的是**阶跃函数**（step function），直译就是“过线就响”。Σ（大写西格玛）是“连加”的数学记号，$\sum_{i=1}^{n} w_i x_i$ 读作“$w_i x_i$ 从 $i=1$ 一直加到 $i=n$”。

权重和偏置各有一个贴身的理解：**权重 = 重视程度**。$w_2 = 0.8$ 表示第二路输入很受重视，$w_3 = -0.2$ 表示第三路是“反向证据”——它越强你越不信。**偏置 = 门槛高低**：b 越大，门槛越低，越容易被触发为 1。权重与偏置合称**参数**（parameter）——感知器的全部参数就这 $n+1$ 个数，所谓**训练**，就是调它们。这个定义先记住，全书还会用几十次。

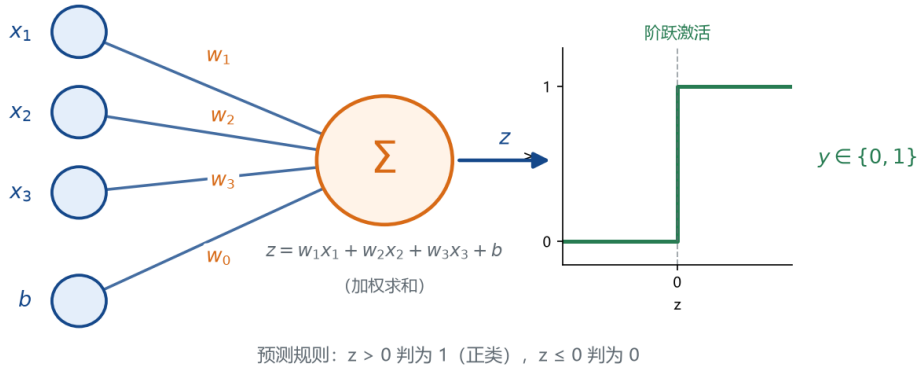


图 1.3　感知器结构：三路输入 x_1, x_2, x_3 与偏置 b 各自乘上权重后进入求和节点 Σ，总和经阶跃激活输出 0 或 1

动手算一遍，比看十遍都管用。

动手时间：徒手算一个感知器

设某感知器有三路输入，权重 $w = (0.6, -0.2, 0.8)$，偏置 $b = -1.0$。**第一步（示例）**：输入 $x = (1, 0, 1)$。加权和 $z = 0.6 \times 1 + (-0.2) \times 0 + 0.8 \times 1 + (-1.0) = 0.4 \geq 0$，输出 $y = 1$。注意：权重是负的那一路，乘的还是 0，没帮上忙也没拖后腿。**第二步（换你）**：分别算 $x = (0, 1, 1)$ 和 $x = (0, 0, 1)$ 的输出。（答案：$-0.4 \to 0$；$-0.2 \to 0$。都算对了吗？）**第三步（想一想）**：把 w_2 从 -0.2 改成 0.9，刚才 $x = (0, 1, 1)$ 的输出变吗？（$0.9 + 0.8 - 1.0 = 0.7 \geq 0$，输出变成 1。）——恭喜，你刚才**手工完成了一次“训练”**：为了纠正一个输出，改了一个参数。下一节要讲的，就是让机器自己干这件事。

1.3　决策边界：感知器在平面上画一条线

感知器的判断，在几何上有一个漂亮的形象。以两路输入为例，把每个样本看成平面上的一个点 (x_1, x_2)，输出由下式决定：

$$w_1 x_1 + w_2 x_2 + b = 0. \tag{1.2}$$

学过直线的你一眼就能认出来：这是**一条直线**。直线一侧的所有点，加权和都大于等于 0，感知器统一输出 1；另一侧统一输出 0。这条线就叫**决策边界**（decision boundary）——感知器并不“理解”你的问题，它只是在平面上找一条线，把两类点分开。下图里，误分类的点会在训练中把直线往正确的方向“推”（怎么推的，1.4 节见分晓）。

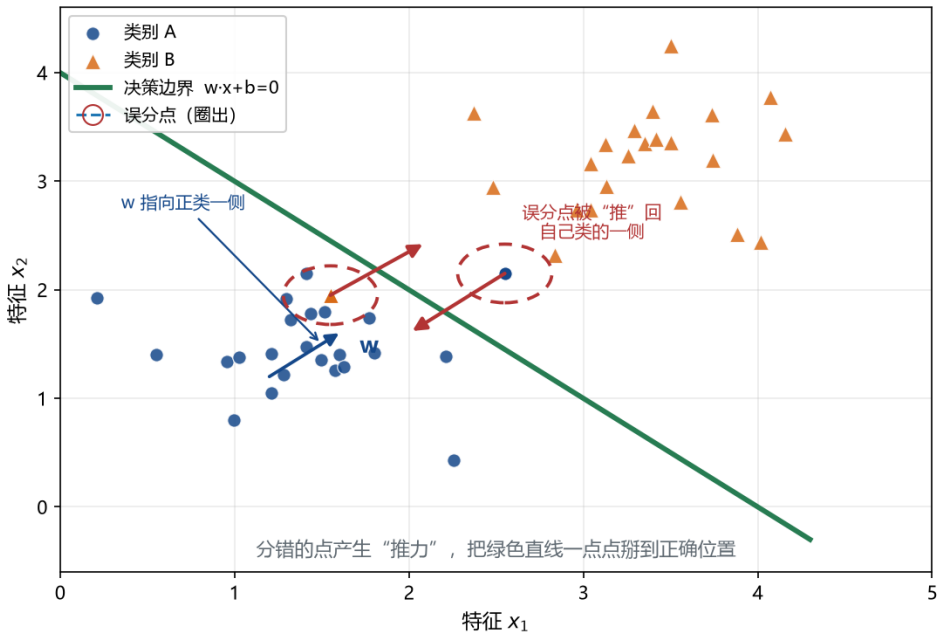


图 1.4　二维两类样本与一条决策直线：直线一侧判为正类（圆点）、另一侧判为负类（方块）；箭头示意误分样本会把直线往正确方向推

哪些问题能被一条线分开？最简单的逻辑运算就能说明问题。看下面的真值表：“与”（AND）、“或”（OR）都能一条直线分开（这类问题称为**线性可分**，linearly separable）；而“异或”（XOR，两输入**不同**才输出 1）却怎么摆直线都分不开——正类两点在对角线上，负类两点在另一条对角线上，一条直线永远顾此失彼。记住这个小小的“分不开”，它是 1.6 节一场大雪崩的引信。

表 1.1　简单逻辑运算的真值表：与、或线性可分，异或不可分

输入	$(0,0)$	$(0,1)$	$(1,0)$	$(1,1)$	一条直线分得开吗
与 AND（同时为 1 才输出 1）	0	0	0	1	能
或 OR（有 1 就输出 1）	0	1	1	1	能
异或 XOR（不同才输出 1）	0	1	1	0	**不能！**

“一条直线分开两类”并不只是逻辑游戏，它是大量真实任务的雏形。统计学家费希尔 1936 年引入的鸢尾花数据集是机器学习界最著名的“入门数据”：三种鸢尾花，测量花瓣与花萼的长宽。下图里可以看到，其中两种花（setosa 与 versicolor）只用“花瓣长 × 花瓣宽”两个量画在平面上，几乎一条直线就能分开——本讲课后编程题就是它。至于分不开的场合怎么办，答案要到下一讲：把感知器叠起来。



图 1.5　鸢尾花数据集涉及的三种鸢尾花之一：本讲课后题用“刚毛鸢尾（setosa）”与“变色鸢尾（versicolor）”两类，花瓣长宽两个特征在平面上几乎一条直线即可分开

图片来源：维基百科 Iris flower data set 词条

1.4　它怎么学习：感知器学习规则

1958年那台满屋咔哒声的机器，是怎么“学习”的？答案朴素得惊人：**错了才改**。对每个带标准答案的样本，感知器先猜一个输出，猜对了就什么都不动；猜错了，就把各路权重朝着“更接近正确”的方向挪一小步。写成公式就是**感知器学习规则**（perceptron learning rule）：

$$w_i \leftarrow w_i + \eta\,(t - y)\,x_i, \qquad b \leftarrow b + \eta\,(t - y). \tag{1.3}$$

人话翻译：箭头“$\leftarrow$”读作“更新为”；t 是标准答案（教师信号，teacher），y 是感知器刚才的输出，η（读“伊塔”，希腊字母 eta）是**学习率**（learning rate）——每次修正的步子迈多大。分三种情形看：**猜对**（$t = y$，修正量为 0）——什么都不动；**把正类错判成 0**（$t - y = 1$）——该样本哪些输入是 1，对应权重就加上一小步，让下次更“重视”它们；**把负类错判成 1**（$t - y = -1$）——反向挪回来。偏置跟着同样修，相当于“顺便调门槛起伏”。下图是“一次错误如何推动直线”的形象版本。

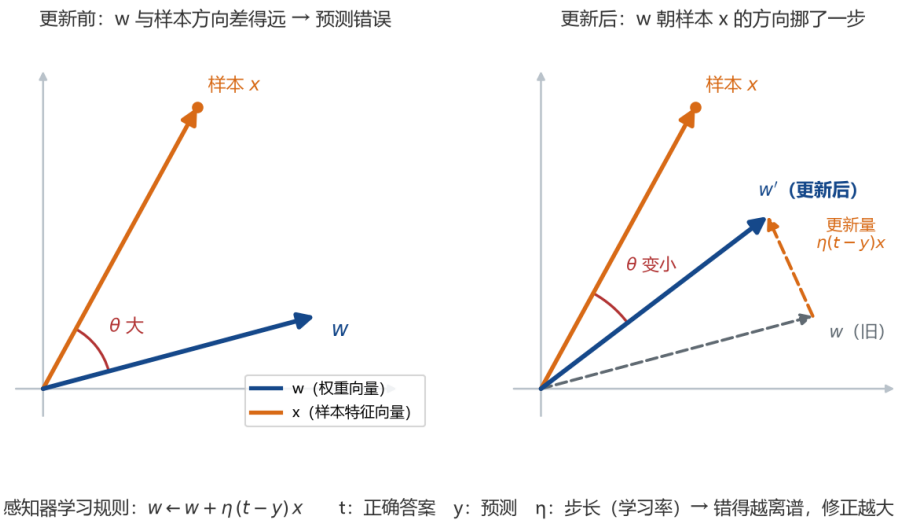


图 1.6　感知器学习规则的几何直觉：预测错了才更新——权重向量朝正确答案的方向挪一小步，学习率 η 控制步长；直线随之转动与平移

为什么这个“挪一小步”总能把直线挪到正确的位置？这里要用到开课导引 0.5 节数学包里的点积（回看第 0 讲）：感知器的加权和 $z = \mathbf{w}\cdot\mathbf{x} + b$ 本质是给“样本 $\mathbf{x}$ 与权重 $\mathbf{w}$ 有多合拍”打分——夹角越小分越高。错把正类判成负类，说明这支“权重箭头”和样本“夹角太大”，修正量 $+\eta\,(t-y)\,\mathbf{x}$ 恰好把箭头**朝样本方向**扳近一点；反之扳远。一来二去，箭头指向正类那一片，直线就立起来了。

知识补齐：向量与点积——感知器的打分器（回指第　讲）

把输入写成向量 $\mathbf{x} = (x_1, \ldots, x_n)$、权重写成 $\mathbf{w} = (w_1, \ldots, w_n)$，加权和就是**点积**：

$$\mathbf{w}\cdot\mathbf{x} = w_1x_1 + w_2x_2 + \cdots + w_nx_n. \tag{1.4}$$

三个要点：**(1)** 点积 = 对应相乘再相加，正是感知器求和那一步；**(2)** 点积是“合拍程度”打分——两个向量方向越一致分越高，垂直为零，相反为负；**(3)** 决策边界 $\mathbf{w}\cdot\mathbf{x} + b = 0$ 就是“合拍分恰好抵消门槛”的等高线：直线始终与 $\mathbf{w}$ 垂直，b 决定它离原点多远。后面各讲凡是出现“打分”“匹配度”“相似度”，底下多半是一次点积。

完整的训练流程是一个循环：拿一批带答案的样本，逐个**预测**；预测错了就按规则**更新**权重；整批过完称为一轮（epoch），再从头来第二轮、第三轮……直到某一整轮一个错都没犯。罗森布拉特在 1960 年证明了一条漂亮的定理：**只要两类确实线性可分，这个循环保证在有限步内停下来**——“能学会”是有数学保证的，不是玄学。至于“分不开”的那种情况，规则会永远改下去、永远学不会——这也正是下一节要谈的天花板。

听完感知器规则的全部故事，往回倒九年，去见一见它的祖师爷。1949 年，心理学家赫布（Donald Hebb）在《行为的组织》一书中提出：学习发生在**连接**上——如果神经元 A 反复参与点亮神经元 B，它们之间的突触连接就该变强。后人把这条假说浓缩成一句英文顺口溜：“Fire together, wire together”——**一起放电的神经元，连在一起**。

知识补齐：Hebb 学习规则（1949）——学习规则的祖师爷

赫布讲的是生物脑里的突触，搬到人工神经元上，就成了历史上第一条学习规则，一行写完：

$$w_i \leftarrow w_i + \eta\, x_i\, y. \tag{1.5}$$

人话：输入 x_i 和输出 y 同时活跃，权重就涨，涨多少正比于两者的“共同活跃度”——**被连接的两个神经元同时放电，连接就变强**。全程不需要标准答案：只要两路信号相关就连，因此它又叫**相关学习**。它和本节感知器规则的分工，一张表看清：

	Hebb 规则（1949）	感知器规则（1958）
要标准答案吗	不要——无监督	要（教师信号 t）——有监督
拿什么改权重	两路信号**同时活跃**（相关性）	预测与答案的**误差** $(t-y)$
一句话形象	“总在一起玩的，加连接”	“错了才改，改在参与出错处”

请记住这位祖师爷：从本讲的感知器规则、下一节的 LMS，到第 2 讲的反向传播，再到第 4 讲的无监督学习，所有“调权重”的方法血脉里都流着赫布的思想——**活动留下连接，经验塑造网络**。

零基础扫盲：把“学习率”想成油门

η 这个数为什么叫“学习率”？设想你在驾校练习倒车入库：方向盘打太猛（η 太大），车在车位里荡来荡去停不稳；打太轻（η 太小），挪半天也进不了库。学习率控制的就是每次修正的幅度：太大，训练在正确答案附近来回震荡；太小，训练慢得让人怀疑人生。选多合适？第 3 讲会用整整一节回答。现在只需记住：**学习率是你拧的第一个“训练旋钮”**。

1.5　从 0/1 到连续：线性单元与最小均方

阶跃函数有个先天限制：输出只有 0 和 1，只能回答“是 / 否”。可现实里大量问题要的是**数字**：明天气温多少度？这块板材的厚度是多少微米？而且 0/1 输出下，“差一点”和“错很多”没有区别——猜 0.49 判 0，猜 0.51 判 1，训练信号一样糙。改良方案简单到令人发笑：**把阶跃函数换成一条直线** $f(z)=z$（恒等函数），输出就是加权和本身，一个连续值。这样的神经元叫**线性单元**（linear unit）。1960 年前后，斯坦福的维德罗（Widrow）与霍夫（Hoff）做出了一颗叫 ADALINE 的自适应线性单元，用来自动滤除电话线路里的噪声——这是神经网络第一次在真实通信工程里上岗。

输出连续了，“错多少”也就能算了。定义**平方误差损失**（squared error loss）：

$$E = \frac{1}{2}(t-y)^2. \tag{1.6}$$

人话翻译：E 衡量“这次错得多离谱”——预测与真值的差，平方，再乘 $\frac{1}{2}$。三个设计动机：**大错重罚**（差 3 罚 4.5，差 1 只罚 0.5，平方放大大错）；**处处可导、曲线光滑**（下一讲求导要用，阶跃函数除跳点外导数全为零，等于告诉训练“往哪都不坡”，没法用）；**正负误差不抵消**（差 $+2$ 和差 -2 不会互相掩盖）。至于那个 $\frac{1}{2}$：纯粹为了平方求导后冒出来的 2 被约掉，公式干净些。

维德罗与霍夫证明，直接对 E 求导并沿负梯度方向修参数，得到的更新规则与感知器规则**一模一样**（$w_i \leftarrow w_i + \eta\,(t-y)\,x_i$），但含义升级了：每一批样本**每一步都更新**，不再“错才动”。这套方法叫 **LMS 最小均方规则**（Least Mean Square）。请留意这件事的分量：上一节的规则是“错了才动”的经验智慧，这一节的规则是“沿坡往下走”的数学必然——同一个公式，两次发明，而后者脚下的

那条路（沿负梯度更新参数）正是第 2 讲梯度下降与反向传播的地基。顺便，你的词汇表里从此多了一员大将：凡是“衡量输出与正确答案差距的函数”，一律称**损失函数**（loss function），全书反复出现。

1.6　天花板与寒冬：一本书按下的暂停键

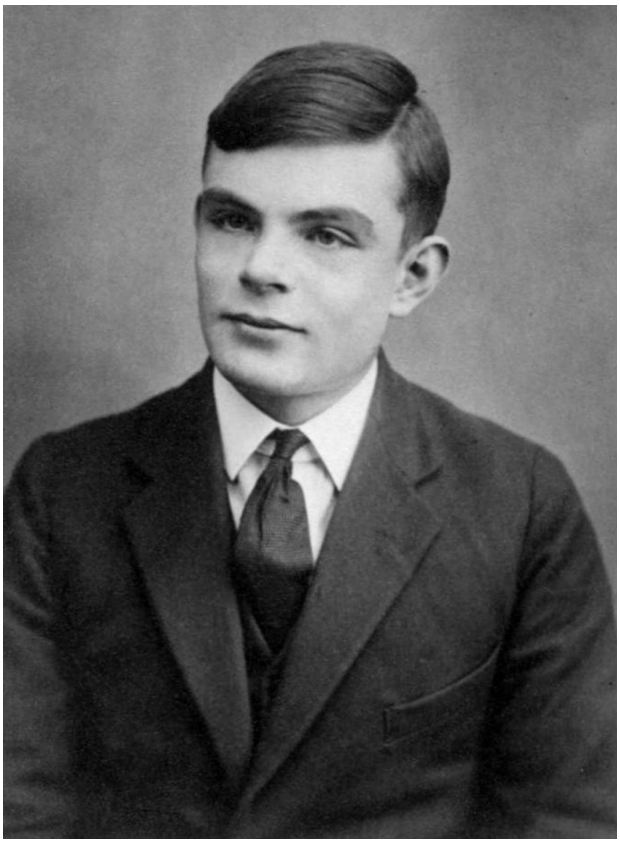


图 1.7　艾伦·图灵（1912–1954）：1950 年发表《计算机器与智能》，讨论“机器能否思考”并提出模仿游戏（图灵测试）

图片来源：维基百科 Turing 词条

感知器的报道之所以敢写“会自我复制、有自我意识”，是因为当时整个学界对“机器智能”期待爆棚——图灵 1950 年那篇《计算机器与智能》已经把“机器能思考吗”吵成了显学。期望被抬得越高，摔下来就越响。1969 年，MIT 的明斯基（Minsky）与佩珀特（Papert）出版了一本薄薄的书，书名就叫《Perceptrons》。书里没有谩骂，只有数学：他们严格证明，**单层感知器连异或（XOR）这样简单的问题都解决不了**——正是 1.3 节真值表里那个“分不开”。更扎心的还有一系列别的局限。书的结尾其实暗示：多层网络也许能绕过这些限制——但当时没人知道**怎么训练**多层网络：隐藏层接触不到标准答案，“错了才改”的规则对它失灵。

后果来得又快又狠。资助机构觉得这条路被“证伪”了，经费断崖式撤离；研究者的兴趣转向其他路线；神经网络研究进入长达十几年的冰期，史称**第一次 AI 寒冬**。罗森布拉特本人 1971 年在一次划船事故中去世，没能看到自己的感知器“复活”。回头看，这场寒冬令人唏嘘的地方在于：**解药其实早已写出**——1974 年，哈佛博士生韦伯斯（Werbos）在论文里给出了把误差责任从输出层倒推回每一层的方法；1986 年，鲁梅尔哈特（Rumelhart）、辛顿（Hinton）与威廉姆斯（Williams）让它广为人知。那套方法叫**反向传播**，是第 2 讲的主角。感知器死而复生的故事还留了一个教训，也是本

书反复念叨的道理：把牛皮吹破，买单的是整个领域——所以 **AI 可以用，结果自己扛**。

把镜头拉远，七十多年的恩怨情仇可以装进一张表。五个阶段——起步、高举、寒冬、复苏、爆发——注意它和你手里这本讲义的顺序几乎重合：**历史顺序就是学习顺序**，后面 11 讲就是把这张表逐行展开。

表 1.2　神经网络编年史：五个阶段一张表（教材第 1 章编年史的人话版）

阶段	年份	事件	一句话人话
起步	1943	M-P 神经元模型	神经元第一次被写成数学：加权求和，过阈值才放电（1.1 节）
	1949	Hebb 学习规则	“一起放电，连在一起”：学习规则的祖师爷（1.4 节）
	1957	感知器	第一台能自己学习的机器，本讲主角
	1960	Adaline	线性单元加最小均方：神经网络第一次工程上岗（1.5 节）
高举	1956	达特茅斯会议	“人工智能”正式得名，全行业期望值拉满
寒冬	1969	《Perceptrons》	XOR 按死单层感知器，经费撤离：第一次 AI 寒冬（上文）
复苏	1974 / 1986	反向传播两次问世	韦伯斯先写出解药，鲁梅尔哈特与辛顿让它家喻户晓：多层网络终于能训了（第 2 讲）
	1982	Hopfield 网络	物理学家用“能量地形”讲联想记忆，神经网络回魂
	1989	卷积网络（CNN）	杨立昆（Yann LeCun）把“看局部”装进网络，机器识别真正开始“看图”（第 5 讲）
	1995	SVM	核方法如日中天，神经网络再坐冷板凳
	1998	LeNet-5	卷积网络读邮编、认支票：在美国银行系统真刀真枪商用
爆发	2006	Hinton 深度学习	逐层预训练破局，“深度学习”正式得名（第 4 讲）
	2012	AlexNet 横扫 ImageNet	GPU 加深网络，识别错误率一口气砍下约十个百分点，深度学习时代开幕（第 5–7 讲）
	2016	AlphaGo	深度网络加强化学习赢下围棋，全民上了一堂 AI 公开课
	2022	ChatGPT 与大模型	预训练加注意力：从一根人造神经元长成万亿参数（第 11 讲）

前沿速递：从 400 个旋钮到万亿参数——“权重即知识”

把 Mark I 感知器与 2026 年的大模型放上天平：一边是 400 个马达旋钮，另一边是万亿级别的参数（第 11 讲细讲），多了约九个数量级。有趣的是，参数的本质六十八年**没有变过**：每一个依然是“重视程度”旋钮——大模型回答你一句话，就是几千亿个旋钮对文字片段做加权打分、层层接力。所谓“知识存在模型里”，更准确的说法是**知识存在参数的取值组合里**。2024–2026 年大模型的“军备竞赛”——参数从千亿冲向万亿、又反过来研究怎么用更少参数干更多活（第 8 讲的轻量化）——本质仍是感知器时代的三个老问题：**旋钮怎么布置、怎么调、调多少个才够用**。你在这一讲学会的“加权和 + 从错误中调权重”，就是这一切的原点。

项目连线：第 1 讲 × 三个项目——人人都是感知器的后代

别以为感知器只是历史课，它此刻就活在三个项目里。**六目拼接**：九个可学习“参数头”网络里有一个 policy（策略）头，它的输出本质是“这一帧要不要用这个特征、这条匹配边要不要信”的 0/1 决策——一颗放大了几十亿倍的感知器。**水下双目**：学生模型里“左右两幅图像块是否匹配”的打分头，干的正是“加权重合拍程度”的活儿，只是把 400 个像素换成了图像块特征（详见第 二 章）。**数据基座**：监考员模型判断“这条内容是真数据还是伪造的”，是教科书式的两类分类——与本讲课后题的鸢尾花分类同一副骨架（详见第 三 章）。它们的差别只在：激活函数换了、层数堆了、参数量爆了。层数怎么堆是第 2 讲的事，怎么调好是第 3 讲的事。

1.7　从这一讲到 2026

本讲结束时，你已经拆开了那台 1958 年的机器。把它的核心想法——加权和加阈值、从错误中调权重——放到 2026 年的现场，你会发现它不但没被淘汰，反而站在舞台正中央：2026 年的大模型，剥到最里层仍然是**线性层堆出千亿、万亿个权重，靠梯度下降学习**的感知器放大版。今天这门手艺向两个极端延伸。一端是**稀疏巨模型**：参数大到任何显卡都装不下，于是 MoE 架构（Mixture of Experts，混合专家——把大网络拆成许多“专家”小网络，每个问题只唤醒相关的少数几个）成了主流：总参数上不封顶，单次推理却只动一小部分。另一端是**端侧小模型**：把能看图、会调函数的多模态模型整个塞进手机。夹在中间的，则是一场**开源权重**的公共化运动——权重（回忆本讲：就是那批“重视程度”旋钮的全部取值）被各家以宽松许可公开发布，人人可下载、可研究、可商用，“神经元加权重”从实验室机密变成了全人类的公共资产。

2026 现场：关键事实

- DeepSeek-V3（2024 年 12 月发布）：MoE 大模型，总参数 671B，每个词元（token，大模型处理文本的最小单位）仅激活 37B——不足总量的十八分之一；完整训练耗资 278.8 万 H800 GPU 小时，训练过程以“异常稳定”著称（来源：DeepSeek-V3 Technical Report，arXiv 2024，编号 2412.19437）。
- 阿里 Qwen3（2025 年 4 月 29 日开源）：旗舰 MoE 模型总参数 235B、每词元激活 22B，以 Apache 2.0 许可完全开放权重，并提供“深度思考 / 快速回答”双模式（来源：Qwen 官方博客“Qwen3: Think Deeper, Act Faster”，2025）。
- Google Gemma 3n（2025 年 6 月）：Gemma 家族首个多模态端侧小模型，面向手机等资源受限设备，支持文本、图像输入与函数调用——大模型第一次“住进”口袋（来源：Google Developers Blog，2025）。
- 2026 年开源格局：DeepSeek、Qwen、Kimi、GLM 等中国团队与 Meta Llama、Google Gemma、Mistral 共同主导，新世代模型以“原生多模态加多尺寸档位”快速迭代（来源：Featherless.ai 年度开源模型盘点，2026）。

一句话收束：**你在本讲徒手算过的那颗 $\mathbf{w}\cdot\mathbf{x}+b$ 神经元，如今被复制了万亿倍，装进了机房和手机——感知器从未过时，它只是长成了大模型。**

本讲课后题要求你指挥 AI 写一个真感知器，在真实数据集上把它训练出来、画出它学到的那条决策直线。数据就是 1.3 节见过的鸢尾花——五要素如下。

第 1 讲课后编程题（用 AI 实现）

题目：指挥 AI 辅助实现一个感知器分类器，在 sklearn 自带的鸢尾花数据集上完成两类分类：只用**花瓣长度、花瓣宽度**两个特征，只取 `setosa` 与 `versicolor` 两类；训练后打印准确率，并用 matplotlib 把样本散点和你学到的决策边界画在一张图上；然后把数据换成另外两类（`versicolor` 与 `virginica`）再跑一遍，观察并解释结果为什么变了。

交付物：(1) 代码文件 `perceptron_iris.py`；(2) 两次运行的截图（各含准确率数字与决策边界图）；(3) 一段 50 字以上的说明，写清“我改了 AI 代码里的什么、为什么这么改”。

提示词建议（可直接抄）：“我是零基础 Python 学生。请用 numpy 手写一个感知器分类器（不要直接调 sklearn 的 Perceptron 类），使用 sklearn 自带的 load_iris 数据集，取花瓣长度、花瓣宽度两个特征，只保留 setosa 和 versicolor 两类。要求：学习率 0.1，训练 10 轮；训练后打印准确率；用 matplotlib 画散点图，并把决策边界直线叠画上去；每一行代码加中文注释；最后告诉我怎么运行这个文件。”

验收线：代码在你自己的电脑上能跑通；随机抽问时能不看书讲清楚 w、b、η 各自的作用和“什么时候更新”；至少有一处自己的修改（换特征、换学习率、换类别都算）。**代码是 AI 写的没关系，讲不清楚就是没做。**

加分项：`versicolor` 与 `virginica` 在花瓣特征平面上并不线性可分——把阶跃激活换成线性单元加平方损失（1.5 节），预测值以 0.5 为界判类，看准确率会不会更好，并用两三句话解释原因。

感知器（Perceptron）

罗森布拉特 1958 年提出的人工神经元：输入加权求和加偏置，过阶跃函数输出 0 或 1。

权重（weight）

每路输入前的系数，表示对该输入的“重视程度”，可正可负，是可学习的主要参数。

偏置（bias）

加在加权和上的常数，相当于触发门槛的高低调节，也是可学习参数。

参数（parameter）

权重与偏置的统称；网络中所有可被训练调整的数。

激活函数（activation function）

加在加和结果上的非线性函数；本讲的阶跃是最古老的一种，第 2 讲会换更平滑的几种。

决策边界（decision boundary）

网络判类的“分界线”；二维输入时感知器的决策边界是一条直线。

线性可分（linearly separable）

两类样本能被一条直线（高维下是一个超平面）完全分开；感知器只对这类问题有理论保证。

学习率（learning rate）

每次参数修正的步长 η；训练中最先要拧的旋钮，第 3 讲专题讨论。

损失函数（loss function）

量化“输出与正确答案差多少”的函数；本讲的平方误差 $E = \frac{1}{2}(t-y)^2$ 是最经典的入门款。

异或问题（XOR problem）

异或运算四点无法用一条直线分开，单层感知器因此失效；正是它逼出了多层网络与反向传播。

第 2 讲　BP 神经网络、反向传播与梯度检查

上一讲结尾，感知器倒在了异或面前：四个点，一条直线怎么也分不开。战胜它的思路朴素得出奇——**一条线不够，就用两条**。把几个感知器“叠”起来：第一层各自画线，第二层对画线的结果再做判断，四点即可分开。感知器时代最大的死结就此解开了一半。为什么只算一半？因为新的麻烦马上到来：第一层神经元藏在网络内部，看不见标准答案，“错了才改”的学习规则对它**失灵**——第一层的权重到底该怎么改？这一讲的主角**反向传播**（Back Propagation，简称 BP），就是那套“把误差责任从输出层一层层倒推回去”的记账方法。它 1986 年成名，直到 2026 年的今天，全世界所有大模型的训练核心仍然是它。

本讲地图

讲什么：多层感知器（MLP）的结构与“自动造特征”直觉、四种激活函数、梯度下降、反向传播四步公式、以及训练代码的第一道安检——梯度检查。**为什么值得学**：反向传播是三个项目里一切模型训练的引擎；实验三（11月 23 日）你亲手跑的就是本讲的训练循环。**学完你能**：画出 2-2-1 网络并讲清误差怎么倒推；把四步公式翻译成人话；用数值梯度检查抓住手写 BP 的 bug；看懂 PyTorch 的 `loss.backward()` 背后发生了什么。

2.1　多层感知器：自动造特征的流水线

把感知器排成队，就得到**多层感知器**（Multi-Layer Perceptron，MLP）。最典型的结构分三种角色：**输入层**（输入层神经元不计算，只负责把数字摆上台面，比如花瓣的长宽）；一个或多个**隐藏层**（hidden layer，真正干活的中间层，外界看不见所以叫“隐藏”）；以及**输出层**（给出最终答案）。相邻两层之间两两全连接（fully connected）——前一层每个输出都连到后一层每个神经元，每个连接上有一个可学习的权重。下图是一个 4-5-2 的三层网络：4 个输入、5 个隐藏神经元、2 个输出，图中每一根箭头都是一个参数。

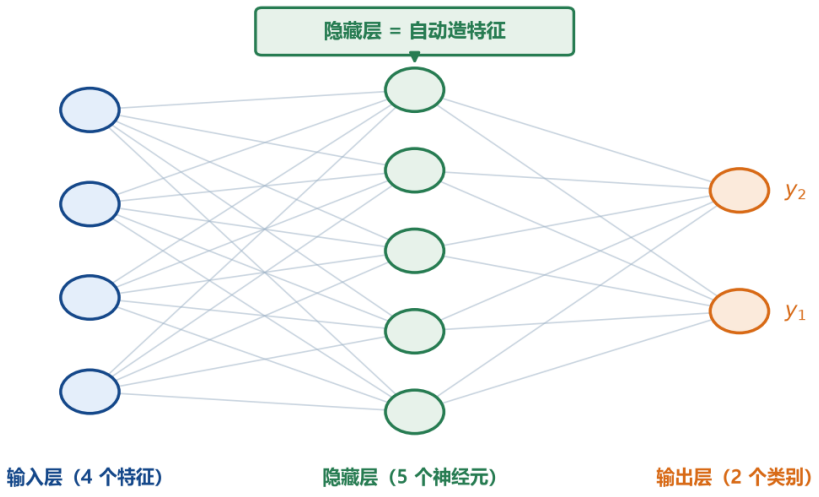


图 2.1　一个 4-5-2 的三层全连接网络：输入层只摆数，隐藏层负责“自动造特征”，输出层给出最终判断；每根箭头都是一个可学习的权重

隐藏层到底在干什么？一句人话：**自动造特征的车间**。原始特征（像素值、温度、花瓣长度）进去，加工出来的新特征出来——至于新特征长什么样，没人手工设计，是训练“逼”出来的：哪一种中间表示能让最后那层感知器把直线画开，网络就往哪种表示变形。1989年数学家证明过一条“通用近似定理”：带一个足够宽的隐藏层，MLP 原则上能逼近任意连续函数——“神经网络什么都能学”这句话的理论底气就在这里（当然，“原则上能”和“实际学得动”是两回事，那是第 3 讲的话题）。

现在回到异或。下图左边是那个死局：四个点卡在两条对角线上，一条直线无解。右边是两层网络的解法：第一个隐藏神经元画一条线，学出“至少一边是 1”（或）；第二个隐藏神经元画另一条线，学出“两边同时是 1”（与）；输出神经元对两个隐藏结果做一次“或且非与”——恰好就是异或。两条线分别单独都不行，**合作**起来就分开了。这就是“叠起来”的全部含义：**每一层单独看都是简单的感知器，叠起来才有复杂的本事**。

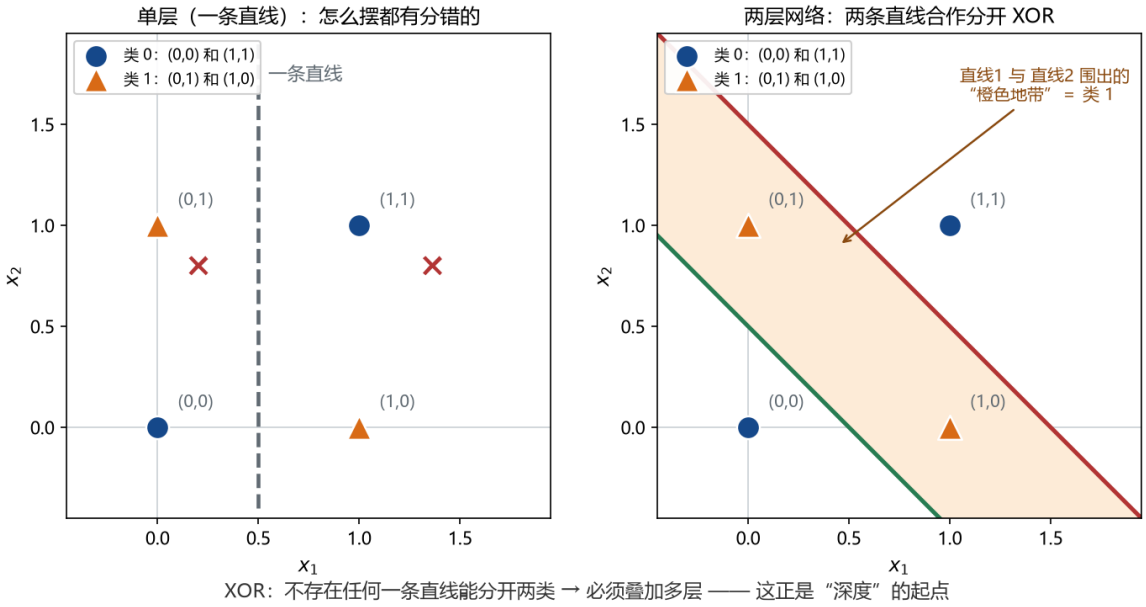


图 2.2　XOR 问题：左——四个点无法用一条直线分开；右——两个隐藏神经元各画一条直线，输出层把两条线的判断组合起来，即可分开四点

2.2　激活函数：没有它，叠一百层也白叠

且慢，先堵一个漏洞。如果每个神经元还是上一讲的阶跃函数，或者干脆什么函数都不加（输出等于加权和），会发生什么？代数一算就露馅：两层线性变换叠起来还是线性——

$$\mathbf{W}_2(\mathbf{W}_1\mathbf{x}) = (\mathbf{W}_2\mathbf{W}_1)\,\mathbf{x}, \tag{2.1}$$

两个矩阵乘完是一个矩阵，一百层等价于一层。所以**层与层之间必须夹一道“把直线掰弯”的非线性变换**，网络才有资格学复杂的东西。这道变换就是激活函数，它是每颗神经元的“性格”。五款最常用的如下（下图是其中前四款的曲线）：

- **sigmoid**：$\sigma(z) = 1/(1+\mathrm{e}^{-z})$，把任意数压进 $(0,1)$ 区间，像“概率化的开关”；如今多用于输出层给概率，隐藏层已少见。
- **tanh**：把任意数压进 $(-1,1)$，输出以 0 为中心，梯度表现通常比 sigmoid 顺滑，隐藏层的老将。
- **ReLU**：$\max(0,\, z)$，负数掐灭、正数放行；计算极快、正区不打弯，现代网络的默认首选。

- **LeakyReLU**：ReLU 的改良，负区不留死寂、只留一条斜率 0.01 的小缝。专治 **dying ReLU**（神经元“装死”）：若某个神经元的输入几乎总为负，ReLU 让它输出恒为 0、导数也恒为 0，权重从此收不到任何学习信号；那条小缝保证它总有被唤醒的机会。
- **Maxout**：可学习的激活函数——神经元自带一小片参数，每次取几条候选直线里最高的那个值（分段线性，但“折点”由训练自己学出来）。表达力更强，代价是参数翻倍，工程里不如 ReLU 普及。

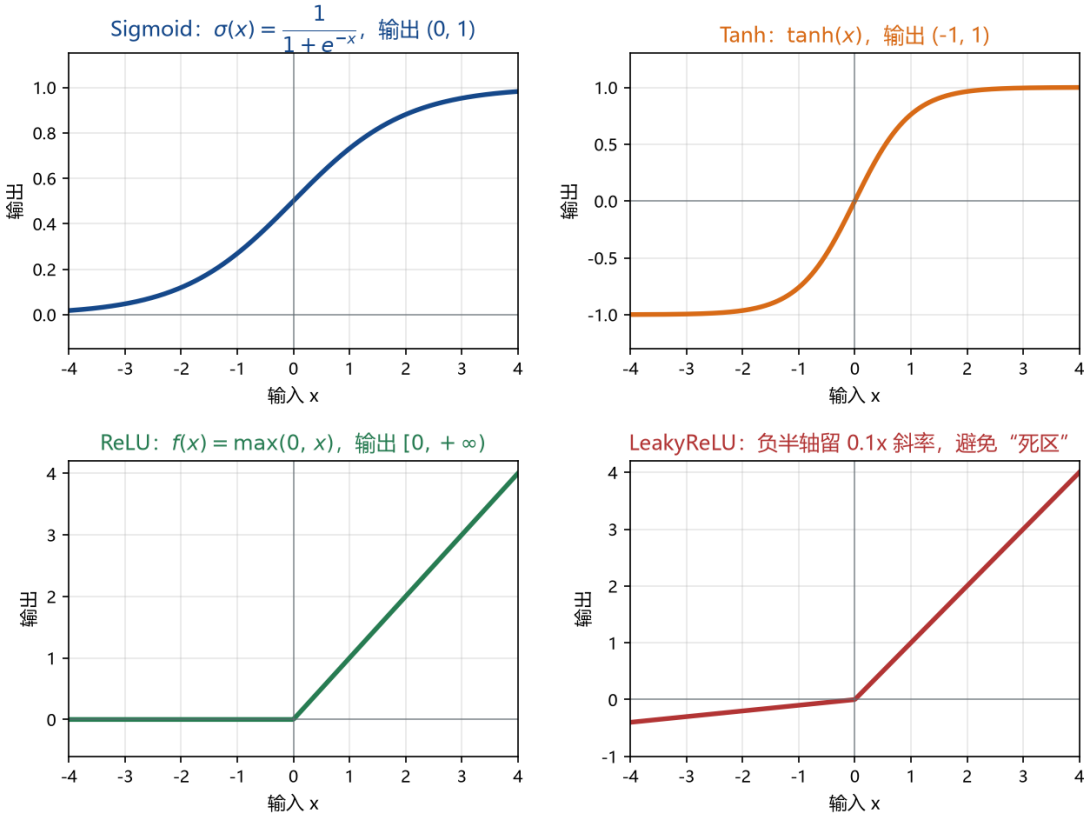


图 2.3　四种常用激活函数曲线（2×2 子图）：sigmoid 与 tanh 两端趋于水平（容易饱和），ReLU 与 LeakyReLU 正区斜率恒为 1

表 2.1　五款激活函数速查：一句人话与常用位置

函数	一句人话	常用位置	点评
sigmoid	压到 0–1 的“概率开关”	输出层（二分类概率）	两端易饱和，隐藏层已让位
tanh	压到 −1 到 1，零中心	隐藏层（经典网络）	比 sigmoid 收敛更顺
ReLU	负数掐灭，正数放行	隐藏层（现代默认）	快、稳，缺点是可能“神经元死亡”
LeakyReLU	负区留 0.01 小缝	隐藏层（ReLU 替代）	专治“死透”的神经元
Maxout	取几条候选里的最大者	隐藏层（不常见）	可学习的激活函数：折点由训练决定，参数翻倍

零基础扫盲：饱和区——学不动的“平原”

看上图 sigmoid 和 tanh 的两端：曲线几乎水平。数学上，**越平的地方导数越接近 0**。导数是什么？回忆开课导引数学包（第 0 讲）：导数是“变化率”，在训练里它就是“学习信号的大小”——导数为 0，意味着无论误差多大，这个神经元都“纹丝不动”：**饱和**（saturation）。埋个伏

笔：下一节反向传播会把误差信号乘着导数一层层往回传，路上经过几段饱和区，信号就被乘得所剩无几——这就是让人闻风丧胆的“梯度消失”的根源之一，也是 ReLU 能上位的理由（它的正区斜率恒为 1，信号路过不打折）。第 3 讲训练调参时还会和它打交道。

Softmax 与交叉熵：分类的黄金搭档

感知器只回答“是 / 否”，现实里更多问题是**多选一**：这张图是猫、狗还是鸟？两件套专门伺候这类问题。**Softmax** 站在输出层，把一排打分变成一排**概率**：

$$y_i = \frac{\mathrm{e}^{z_i}}{\sum_j \mathrm{e}^{z_j}}, \tag{2.2}$$

人话：谁的分数高谁的概率大，且所有概率加起来恰好等于 *1*——“票数”从此变成“得票率”。**交叉熵损失**（cross-entropy）则站在损失的位置，衡量**预测的概率分布与真值差多远**：

$$L = -\sum_i t_i \log y_i, \tag{2.3}$$

人话：真值 t 是一张只在正确类别处为 *1* 的选票；正确类的得票率越低，罚得越狠——预测满分（$y=1$）罚 *0*，接近零票则罚出天文数字。两件套在现代框架里常被熔成一个损失使用，熔合后的梯度形式简洁得漂亮：

$$\frac{\partial L}{\partial z_i} = y_i - t_i, \tag{2.4}$$

恰好等于“预测概率减真实标签”。推导这里不给（框架也根本不需要你推），但这个结论值得记：误差信号就是“你猜的”减“标准答案”，朴素得像设计好的。第 3 讲的课后题就用这套黄金搭档。

2.3　梯度下降：在山地里找最低谷

现在假设网络结构定了、激活函数选了，还剩最后一个问题——参数怎么调？先给“调得好不好”定个尺子：**损失函数**（第 1 讲已登场，比如平方误差）。所有参数的一组取值，对应一个损失值；想象把“每一组参数取值”都摆成平面上的一个点，把损失值当海拔，参数空间就成了一片**山地**：山谷是损失低的好参数，山头是损失高的坏参数。**训练的目标，就是找到最低谷**。

怎么下山？微积分给了一个笨但可靠的答案：站在任何一点，**梯度**（gradient）指向**最陡的上坡**方向——梯度就是“全部偏导数组成的向量”，每个偏导数回答“这一维参数动一丁点，损失变多少”。那么最陡的下山方向就是它的反方向。写成公式：

$$w \;\leftarrow\; w - \eta \frac{\partial L}{\partial w}. \tag{2.5}$$

人话翻译：把每个参数 w 朝“损失变小的方向”挪一小步——$\partial L/\partial w$（读“L 对 w 的偏导数”）是这一点上 w 方向的坡度，前面加负号就是下山；η 还是老朋友**学习率**，步子迈多大。如此反复，“沿坡下山、走到谷底”，就叫**梯度下降**（gradient descent）。

但请注意这片山地的规模：一个现代模型的参数以百万、亿、万亿计——这是**亿维**的山地，每走一步都要算出**每一个维度的坡度**。靠谁来算？手算一两个参数还行，十亿个不可能。下一节的反向传播，就是那个把这件事变可能的算法。另外先预告一张图：不同学习率下山会走出截然不同的路径（震荡、平稳、龟速），损失曲面的立体长什么样，第 3 讲有专门的等高线图等着你。

2.4　反向传播：把责任一层层倒推回去

来看这个场景：网络给出预测后，输出层“知道自己错多少”——预测值和标准答案的差明摆着。可隐藏层呢？它没有标准答案，上一讲“错了才改”的规则无从谈起。**每个权重该背多大的锅？**这就是反向传播回答的问题。

思想可以用一句话讲完：**责任按贡献分配，从右往左传**。输出层的误差是一笔“总账”，往回分账时，谁对这笔误差**影响大**（谁权重大、参与多），谁就多背；每个神经元收到分来的账，再往它上游分……直到每个参数都拿到自己的那份责任，按责任大小修自己。下图里蓝色是数据从左往右的前向流动，红色是责任从右往左的倒推流动——一去一回，正好构成一次完整的训练呼吸。

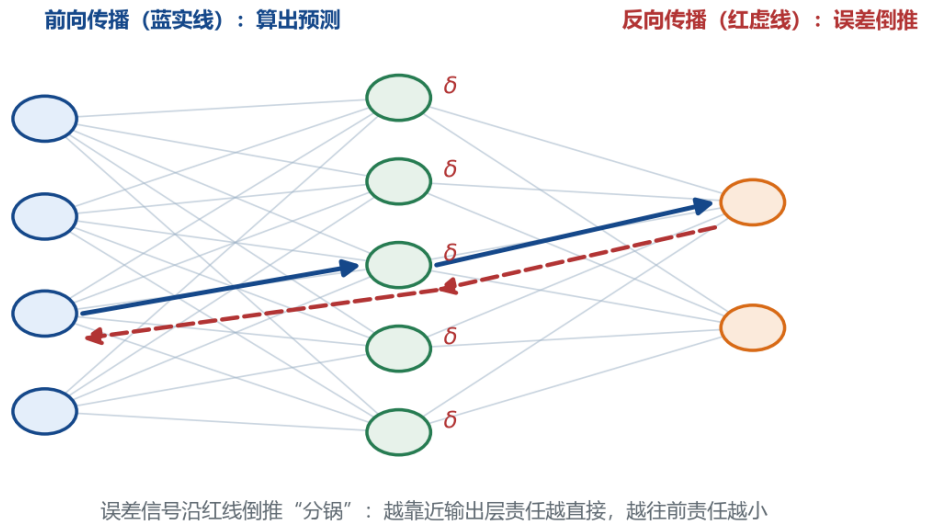


图 2.4　反向传播示意：蓝色箭头为前向计算流（输入 → 输出），红色箭头为误差倒推流（输出 → 输入）；每个权重按贡献分得自己的“责任”δ

把“分账”变成数学的钥匙，是开课导引 0.5 节埋好的——链式法则。

知识补齐：链式法则——反向传播的全部数学（回指第　讲）

若 y 随 u 变、u 随 x 变，则总变化率相乘：

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{\mathrm{d}y}{\mathrm{d}u} \cdot \frac{\mathrm{d}u}{\mathrm{d}x}. \tag{2.6}$$

一个多层网络，本质是一条很长的复合函数链：输入变成第一层输出，第一层输出变成第二层输入……最后变成损失。要问“某个早层参数动一丁点，损失动多少”，就把沿途每一段的变化率**乘起来**。反向传播没有任何新数学——它只是链式法则的工程化：**沿链条从右往左，把已经算好的导数逐段乘回来**，中间结果全程复用，绝不重复计算。

下面是反向传播的四步公式。设定：共 L 层，第 l 层的加权和记 $\mathbf{z}^{(l)}$，激活后的输出记 $\mathbf{a}^{(l)}$；$\mathbf{W}^{(l)}$、$\mathbf{b}^{(l)}$ 是该层参数；$\boldsymbol{\delta}^{(l)}$（粗体德尔塔 delta）是第 l 层每个神经元的“责任清单”；$\odot$ 表示“对应位置相乘”。每条公式下面都配了人话。

第一步（前向）：信号从左到右算一遍，途中存下每层结果：

$$\mathbf{a}^{(l)} = \sigma\big(\mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}\big). \tag{2.7}$$

人话：输入逐层加工到输出，顺手把每层的“中间产物”记在账本上——回传时要用。

第二步（输出层责任）：

$$\boldsymbol{\delta}^{(L)} = \nabla_{\mathbf{a}^{(L)}} L \;\odot\; \sigma'\big(\mathbf{z}^{(L)}\big). \tag{2.8}$$

人话：答案错多少（损失对输出的导数），乘上激活函数在该点的斜率——斜率是 *0* 的神经元（回忆饱和区）暂时学不动，这是原文照抄的物理规律，不是 *bug*。

第三步（隐藏层责任）：

$$\boldsymbol{\delta}^{(l)} = \big(\mathbf{W}^{(l+1)}\big)^{\mathsf{T}} \boldsymbol{\delta}^{(l+1)} \;\odot\; \sigma'\big(\mathbf{z}^{(l)}\big). \tag{2.9}$$

人话：下游的责任，按权重比例“退货”给上游——转置 $\mathbf{W}^{\mathsf{T}}$ 就是“按贡献反着分账”；再乘上本神经元自己的斜率。一层层往左推，直到第一隐层。

第四步（更新）：

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta\, \boldsymbol{\delta}^{(l)} \big(\mathbf{a}^{(l-1)}\big)^{\mathsf{T}}, \qquad \mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \eta\, \boldsymbol{\delta}^{(l)}. \tag{2.10}$$

人话：谁参与犯错谁改——责任大（δ 大）、参与多（输入 $\mathbf{a}$ 大）的权重改得多；方向与梯度相反（负号），步子由 η 定。

至此可以回答本讲开头的问题了：隐藏层看不见标准答案怎么办？**不用看见——它的“答案”就是下游退回来的责任**。再品味一次这件事的效率：一次反向扫描，就把**全部**参数的梯度都算了出来，计算量大约只有两次前向传播。“一次错误，全体对账”，如此便宜，这就是反向传播统治深度学习六十年的原因。

2.5　梯度检查：给你的反向传播“对账”

坏消息：上面第三步里那个转置，是手写 BP 代码时最容易写错的地方之一——漏转置、转错轴、导数写错号，程序**不报错**，只是“学得特别慢”或者“根本学不动”。这是深度学习里最难抓的一类 bug：不崩溃，只摆烂。好消息：有一个笨办法能当场抓住它。

办法来自导数的定义本身。想知道 w 处的坡度？把 w 往两边各挪一丁点 ε（读“艾普西龙”，取 10^{-5} 这个量级），量一量海拔变化：

$$\frac{\partial L}{\partial w} \;\approx\; \frac{L(w+\varepsilon) - L(w-\varepsilon)}{2\varepsilon}. \tag{2.11}$$

这叫**数值梯度**（numerical gradient）：下图里，两端点连线的割线斜率，在该点附近就是切线斜率的好替身。它又慢（每查一个参数要跑两遍前向）又准——正好和你手推的**解析梯度**（analytic gradient，即上一节那些公式算出来的）互补：**训练前抽查一批参数，两相对账**，相对误差在 10^{-6} 量级以下即通过。几秒钟的对账，换整个训练的心安——**写训练代码，第一件事永远是梯度检查**，这个习惯从本讲的课后题开始养成，实验三还要用。

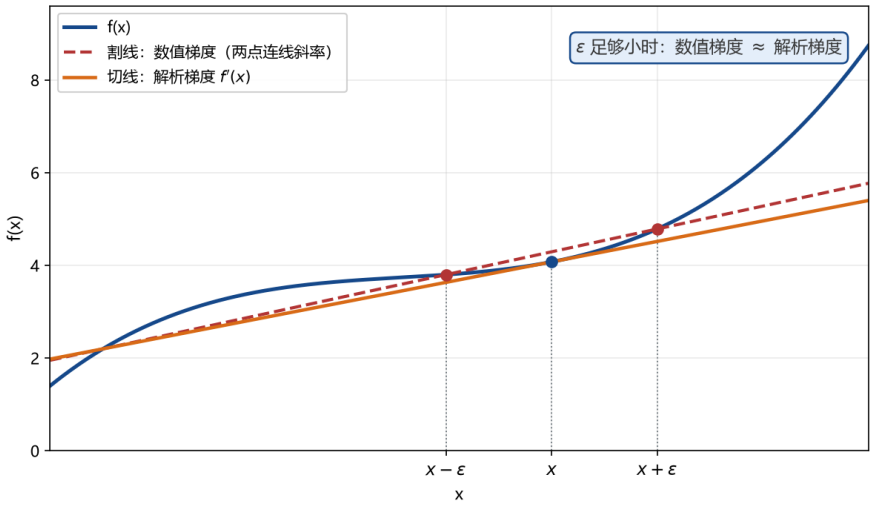


图 2.5　梯度检查的几何直觉：函数曲线上两端点连线的割线斜率（数值梯度）≈ 该点切线斜率（解析梯度），ε 越小越接近

先在纸上找一次感觉，再看代码。

动手时间：用手机计算器做一次梯度检查

取函数 $f(x) = x^2$，在 $x = 3$ 处。**解析梯度**（求导）：$f'(x) = 2x$，$f'(3) = 6$。**数值梯度**（定义）：取 $\varepsilon = 0.1$，$f(3.1) = 9.61$，$f(2.9) = 8.41$，于是

$$\frac{9.61 - 8.41}{2 \times 0.1} = 6.0.$$

两相对账，完全一致。再把 ε 换成 0.01 算一遍，看是不是更贴近 6——**割线在逼近切线**。整个过程只要一部手机计算器，你刚刚手工执行了一次“梯度检查”。

进阶版：把你手推的 BP 公式和数值定义各写一段代码，对每个参数比较（`w0` 取几个代表值）：

```
def num_grad(f, w, eps=1e-5):    # 数值梯度：定义式的笨办法
    return (f(w + eps) - f(w - eps)) / (2 * eps)

def ana_grad(w):                 # 解析梯度：你手推的公式 f'(x) = 2x
    return 2 * w

for w0 in [-1.0, 0.5, 3.0]:      # 抽查三个点
    print(w0, num_grad(lambda w: w**2, w0), ana_grad(w0))
# 输出两列几乎相同的数 => 你的"手推公式"对账通过
```

2.5.1　三种求导范式：从手推公式到 autograd

刚才的梯度检查，其实亮出了求导三大流派中的一派。三派同台看一眼，你就明白自己身在何处、框架在替你干什么：

表 2.2　三种求导范式：怎么算、好在哪、命运如何

范式	怎么算	优点	命运
数值微分	把参数挪一丁点、量海拔变化：只挪一边叫前向差分，两边都挪取平均叫中心差分——2.5节的梯度检查就是它	绝对可信，永远不错	太慢：每查一个参数要跑两遍前向，只配当“对账员”
手动微分	人推公式，把导数一行行写死进代码	快	易错：模型改一处、导数全部重推——早期神经网络库与 OpenCV 时代的常态
自动微分	把程序拆成加减乘除等基本算子，每个算子的导数已知，链式法则自动把全程串起来	又快又准又通用	现代框架标配

一句话记住第三行：**PyTorch 的 autograd 就是自动微分——你写前向，它送你反向**。至于这件事后来被工程界推到了什么程度，本讲章末的前沿盒子接着说。

2.6　完整训练一次：把循环跑起来

万事俱备。把前向、损失、反向、更新串成一个循环，就是全部深度学习训练的骨架。下面这段 10 行伪代码（numpy 语法，`@` 表示矩阵乘法）在 2-2-1 的 MLP 上拟合异或，正是课后题的骨架：

```
W1, b1, W2, b2 = init()                 # 1 初始化参数（怎么初始化见第3讲）
for step in range(20000):               # 2 训练循环开始
    A1 = tanh(X @ W1 + b1)              # 3 前向：算隐藏层（tanh）
    Yhat = A1 @ W2 + b2                 # 4 前向：算输出（线性）
    L = mean((Yhat - Y)**2)             # 5 损失：这一轮错多少
    # —— 训练前先做梯度检查（2.5 节），通过后再开跑 ——
    D2 = 2 * (Yhat - Y) / len(X)        # 6 反向：输出层责任（四步之第2步）
    D1 = (D2 @ W2.T) * (1 - A1**2)      # 7 反向：隐层责任（转置退货+tanh导数1-A1^2）
    W2 -= lr * A1.T @ D2                # 8 更新：谁参与犯错谁改（四步之第4步）
    W1 -= lr * X.T @ D1                 # 9 同上
    # 每 1000 步打印一次 L，盯着它下降
```

逐段对照：第 3–5 行是四步公式的第一步（前向）加损失；第 6–7 行是第二、三步（算责任，注意第 7 行里 `W2.T` 的转置退货和 `1 -A1**2`——tanh 的导数正好是 $1-\tanh^2$）；第 8–9 行是第四步（更新）。所谓“训练”，**不是人去拧几十亿个旋钮，而是误差沿着网络把每个旋钮该转多少自动算出来**——机器自己执行的、规模化的第 1 讲手工训练。等第 6 讲你见到 PyTorch 版的训练循环，会发现换的只是“中间三行”变成 `loss.backward()`，骨架一毫米没动。

BP 七步训练流程（教材经典流程）

把本讲的零件按顺序装起来，就是教材里那条经典的七步流程——它同时也是你今后任何训练代码的目录：

1. **初始化**参数（别用全 0——为什么，第 3 讲揭晓）；
2. **样本输入**：取一批训练样本喂进网络；
3. **前向**逐层计算到输出，对照答案算出误差；

4. **反向**传播：按四步公式逐层算出每个参数的误差信号（责任清单 $\boldsymbol{\delta}$）；
5. **更新权重**：沿负梯度方向挪一小步；
6. **循环**第 2–5 步，直到损失不再下降（收敛）；
7. 拿**测试集**检查精度——成绩合格，才算训成。

对照 2.6 节的十行代码：第 1 行是第 1 步，第 3–5 行是第 2、3 步，第 6–7 行是第 4 步，第 8–9 行是第 5 步，外层 for 循环是第 6 步，课后题最后打印的准确率是第 7 步——七步你其实已经全部见过了。

动手时间：在浏览器里“玩”一个神经网络

理解七步流程最好的方式，是亲眼看它转起来。打开 `playground.tensorflow.org`（TensorFlow Playground，谷歌出品的神经网络可视化沙盘，浏览器直接用、零安装）：左边一排是超参数旋钮（学习率、激活函数、层数、每层神经元数），右上角点“运行”，就能**亲眼看着**一个小网络在平面上把两色数据点分开——决策边界像水波一样慢慢变形，那就是梯度下降本尊。建议玩三局：(1) 选螺旋（spiral）数据集、只留 1 个隐藏神经元，看它怎么都分不开（重温 XOR 的死局）；(2) 加到 2 层 8 个神经元再试，看边界扭出花来；(3) 把激活函数从 tanh 换成 ReLU，对比收敛快慢。十分钟的沙盘，胜过十页文字。

胜者为王（WTA）竞争学习

前面所有训练都靠标准答案纠错，其实还有一条不用答案的路。**竞争学习**让输出神经元互相竞争：每来一个样本，大家各自用“与它的合拍程度”（回忆第 1 讲的点积打分）投标，**只让得分最高的赢家激活，其余全部沉默**——这就是**胜者为王**（Winner-Take-All，WTA）。赢家独享这次更新：把自己的权重往样本方向挪一点。一来二去，每个赢家的权向量停在数据某一片的“正中央”，**自动收敛为聚类中心**——一个标签不用，网络自己把数据分了堆。这是**自组织学习的入门**，第 4 讲的无监督自编码器，是这条路上的更远一站。

前沿速递：自动微分——你写前向，反向白送

本讲花最大力气讲的反向传播，在 2026 年的日常研发里几乎没人手写——不是因为它不重要，恰恰因为它太重要，被“焊”进了框架。PyTorch、JAX、MindSpore 等主流框架都内置**自动微分**（automatic differentiation，俗称 autograd）：你只写前向计算，框架沿计算图自动、精确地求出每个参数的导数——本质就是把你的 δ 倒推过程自动化到任意复杂的网络上，连梯度检查都替你留好了接口。2016 年前后它让深度学习研究从“推导一个网络三个月”变成“搭一个网络一下午”；2024–2026 年，大模型动辄千亿参数，手写反向早已物理上不可能，自动微分加分布式训练就是大模型时代的“纸和笔”。但框架会算不等于你不用懂：本讲的梯度检查思想至今仍是排查训练问题的第一件兵器——第 3 讲你会亲手用它抓住“学习率调错”与“初始化不当”两个现行犯，实验四（12 月 21 日）还有它的实战任务。

项目连线：反向传播——三个项目的共同引擎

本讲的训练循环，就是三个项目里所有“可学习”二字的底气。**六目拼接**：九个参数头的每一次微调，都是把拼接误差（错位的像素）经 BP 倒推到每个卷积核上；端到端路线在 RTX 4080 上烧掉的每一度电，都在算 δ（详见第 一 章）。**水下双目**：教师模型 FoundationStereo 参数全部冻结（不训练），学生模型的上亿参数，全靠反向传播从“与教师伪标签的差距”里一步步学出来——第二篇第 二 章那个“教师–过滤–学生”闭环的下半环，转的就是这个循环。**数据基座**：监考员真假分类器与 R-GCN 图检索模型，训练全是本讲的循环在转（详见第 三 章）。落到日程上：**实验三（11 月 23 日）你将亲手在真实数据上跑通这个循环；实验四（12 月 21 日）学把它跑得又快又稳**——那是第 3 讲的内容。

2.7　从这一讲到 2026

本讲的“前向—损失—反向—更新”，2026 年每天在全世界数以万计的显卡上运转，但它早已不是你手里的十行代码，形态发生了几重进化。第一重：**被编译**。PyTorch 2.5 起把整个反向传播捕获成一张计算图、交给编译器统一优化（Compiled Autograd）——反向传播从“运行时逐步递归求导”，变成“编译器手里可供全盘优化的图”。第二重：**被分布式**。模型大到一台机器装不下：中间激活存不下，就上“梯度检查点”（gradient checkpointing——注意别与本讲的梯度检查混淆：那是验算导数对不对，这是先扔掉中间结果、反向时再重算，拿时间换显存）；多卡之间梯度还要互相同步，DeepSeek-V3 干脆用 FP8（用更短的数字格式存数与运算，省显存省带宽）加双向流水线 DualPipe（前向与反向像双向车道交错通行），把反向传播的通信开销整个藏进计算里。第三重是悬念：**BP 会被取代吗**？辛顿本人提出的 Forward-Forward 方案（两次前向、不用反向梯度）2025 年已能训练卷积网络，是最有生物学味道的挑战者——但在大模型规模上仍全面落后。**反向传播活得很好，只是你越来越看不见它。**

2026 现场：关键事实

- PyTorch 2.5（2024 年 10 月）引入原型特性 Compiled Autograd：“捕获整个反向传播过程”为单一计算图交给 torch.compile 编译，且不受前向图断裂影响——反向传播正被纳入编译器流水线（来源：PyTorch v2.5.0 Release Notes，GitHub，2024）。
- 万亿参数训练的显存瓶颈靠“重算换显存”：梯度检查点 / 激活重算是标配；2025 年 8 月提出的 Adacc 框架把激活压缩与重计算统一调度，吞吐量较现有框架提升 1.01–1.37 倍（来源：Adacc，arXiv 2025，编号 2508.00806）。
- DeepSeek-V3 训练 671B MoE 模型时，核心矩阵运算用 FP8 精度，并以 DualPipe 双向流水线使反向传播中的 all-to-all 通信与另一路计算完全重叠，全部训练仅 278.8 万 H800 GPU 小时（来源：DeepSeek-V3 Technical Report，arXiv 2024，编号 2412.19437）。
- Forward-Forward（两次前向、无反向梯度）2025 年已扩展到卷积网络并正式刊出，被视为有生物学依据的 BP 替代路线；学界共识是其规模与性能尚不能取代 BP（来源：Scodellaro et al.，Training convolutional neural networks with the Forward-Forward algorithm，Scientific Reports，2025）。

一句话收束：**反向传播四十年没被推翻，但它正在被“编译”和“分布式化”——未来稀缺的不是会不会推链式法则，而是能不能在万卡集群上让梯度又快又省地流回去。**

本讲课后题不用任何框架，用 numpy 把上面的循环亲手跑通——这是全书唯一一次要求你手写反向传播，目的正是让你以后调用 `loss.backward()` 时心里有底。五要素如下。

第 2 讲课后编程题（用 AI 实现）

题目：指挥 AI 辅助，用 numpy 手写一个 2-2-1 的 MLP（2 输入、2 个隐藏神经元、tanh 激活、1 个线性输出）拟合 XOR 四个样本：$(0,0)\to 0$、$(0,1)\to 1$、$(1,0)\to 1$、$(1,1)\to 0$。要求顺序：先实现数值梯度检查并打印相对误差，通过后（$<10^{-5}$）才允许开训；用平方损失加全批量梯度下降，学习率 0.1，最多 2 万轮，训练到均方误差低于 0.01；用 matplotlib 画出“训练轮数 vs 损失”曲线并保存为 png；最后打印四个输入的预测值。

交付物：(1) 代码文件 `xor_mlp.py`；(2) 三张截图：梯度检查输出、损失下降曲线、四个输入的预测值；(3) 一段 50 字以上的说明，写清“我改了 AI 代码里的什么、为什么这么改”。

提示词建议（可直接抄）：“我是零基础学生，正在学反向传播。请用 numpy 实现 2-2-1 全连接网络：2 输入、2 个隐藏神经元（tanh 激活）、1 个线性输出，拟合 XOR 四个样本。要求：(1) 先实现数值梯度检查，打印每个参数解析梯度与数值梯度的相对误差；(2) 用均方误差损失和全批量梯度下降，学习率 0.1，最多 20000 轮，损失低于 0.01 就停；(3) 训练完打印四个输入的预测值；(4) 用 matplotlib 画损失随轮数下降的曲线并保存为 png；(5) 每行加中文注释，并注明哪几行对应‘前向、输出层 δ、隐层 δ、更新’四步。”

验收线：梯度检查通过且截图在；损失曲线单调下降并收敛到 0.01 以下；四个预测值与真值之差全部小于 0.1；抽问时能讲清 δ 如何从输出层倒推回隐藏层、$\mathbf{W}^{\mathsf{T}}$ 为什么会出现在隐层责任里；至少有一处自己的修改（换激活函数、换学习率、加一个隐藏神经元都算）。**代码是 AI 写的没关系，讲不清楚就是没做。**

加分项（任选其一）：(1) 把输出改成 sigmoid 激活加交叉熵损失（先问 AI“交叉熵是什么、为什么配 sigmoid”），对比收敛速度；(2) 把两个隐藏神经元学到的两条“直线”画在 XOR 四点散点图上，看看网络自己选了哪两条线；(3) 换三种参数初始化（全 0 / 太大 / Xavier）各训一次并对比曲线——第 3 讲的预告片。

多层感知器（MLP）

感知器排层叠成的全连接网络：输入层、一个或多个隐藏层、输出层。

隐藏层（hidden layer）

夹在输入与输出之间的计算层，负责“自动造特征”；其参数没有直接的标准答案，靠反向传播获得学习信号。

全连接（fully connected）

相邻两层神经元两两相连，每个连接带一个可学习权重。

激活函数（activation function）

层间必夹的非线性变换，如 sigmoid、tanh、ReLU；没有它，多层网络塌缩成一层。

饱和（saturation）

激活函数进入两端近水平区，导数接近 0，训练信号被“掐灭”的现象。

梯度（gradient）

损失对全部参数的偏导数组成的向量，指向最陡上坡方向；它的反方向就是下山方向。

梯度下降（gradient descent）

沿负梯度方向反复更新参数、走向损失最低点的迭代方法。

链式法则（chain rule）

复合函数的导数等于各段导数相乘；反向传播的全部数学。

反向传播（Back Propagation, BP）

利用链式法则把误差责任从输出层倒推回各层、一次算出全部参数梯度的算法，1986 年成名，至今仍是一切模型训练的引擎。

梯度检查（gradient check）

用数值定义（两点割线斜率）与手推解析梯度对账，抓手写 BP 的 bug；写训练代码的第一道安检。

第 3 讲　超参数、权重初始化、模型训练与评估

上一讲结尾，十行伪代码把训练循环跑了起来：前向、损失、反向、更新，一圈一圈地转。但只要你真去跑它，很快会撞见一件怪事：循环一个字没改，只把开头的几行“设置”换掉——学习率从 0.1 改成 10、权重全部从 0 起步、每批样本数从 100 改成 1——结果可能从“漂亮收敛”直接变成“数字爆炸成 nan”或“损失纹丝不动”。这些必须在开跑前拧好的旋钮，统称**超参数**（hyperparameter）——“参数之上的参数”：普通参数（权重、偏置）由反向传播自动学，超参数由你拧。调它们是一门手艺，而且像学做菜：**学习率是火候**，大火烧糊、小火夹生；**权重初始化是下料**，第一步就歪，后面步步歪；**验证集是尝味道**，菜好不好要边做边尝，不能等端上桌才发现咸了。本讲把这套“火候学”一次讲透：损失面长什么样、学习率怎么选、权重从哪开始、batch 三兄弟怎么分、数据怎么切、优化器家族谁是谁。第 2 讲埋的伏笔——用梯度检查抓住“学习率调错”与“初始化不当”两个现行犯、那张损失面等高线图——也都在这里兑现。

这一讲还是实验三（11 月 23 日，深度网络的模型训练实践）的直接理论装备：那天你敲下的每一条训练命令、盯着的每一条 loss 曲线，用的全是本讲的词汇。带着这个目的读，每一节都不抽象。

本讲地图

讲什么：损失面与梯度下降的“地形学”、学习率与衰减策略、权重初始化（Xavier/He）、batch/epoch/iteration三兄弟、训练—验证—测试三分与过拟合诊断、向量化编程、优化器家族（SGD、Momentum、RMSProp、Adam）。**为什么值得学**：超参数是三个项目里一切训练动作的把手；实验三（11 月 23 日）是本讲的实弹演习，实验四（12 月 21 日）要学“跑得又快又稳”，靠的也是这些旋钮。**学完你能**：读 loss 曲线诊断训练病症；给网络选对初始化；说清 batch size 的权衡；用 Adam 开训、用早停收工；看懂 `--epochs 200 --batch 8` 这类命令里每个词的含义。

3.1　损失面与梯度下降：先看清脚下是什么地形

第 2 讲说过，训练就是在参数空间这片“山地”里找最低谷。当时只给了一句话，现在把地图摊开。把一个只有两个参数的小模型的损失值画成海拔，就得到下面的等高线地形图：同一圈上海拔（损失）相同，中心是谷底。三条彩色路径是三次训练——起点相同、山是同一座，唯一区别是每一步迈多大。

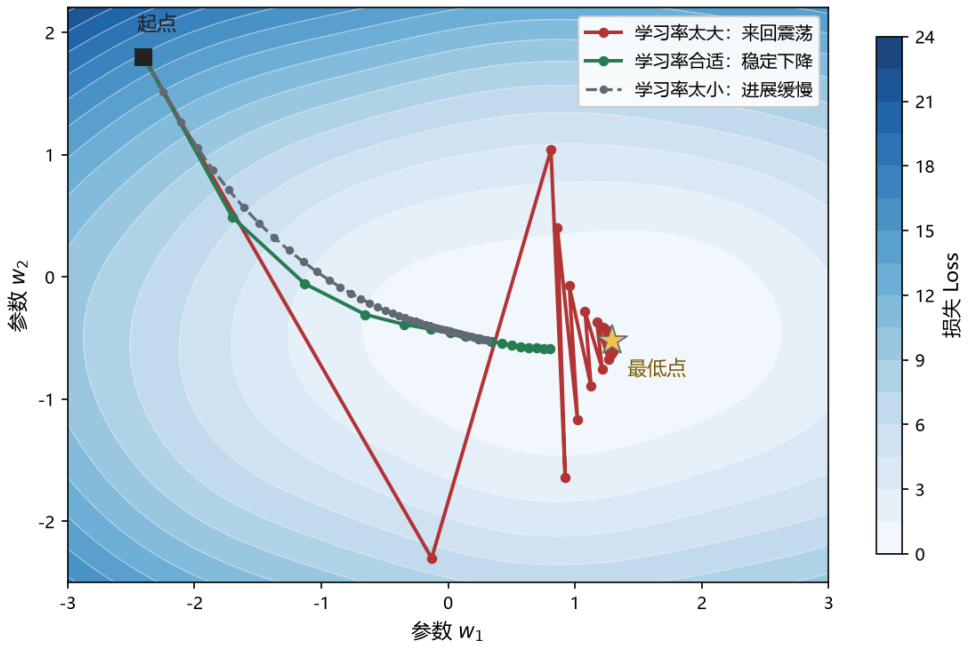


图 3.1　损失面等高线与三条下降路径：步长合适（稳步滑向谷底）、步长太大（在两壁间来回反弹甚至越弹越高）、步长太小（走到半路还没到）

读图要点：一条路径平滑转向、直奔谷底——步长合适；一条在谷底两侧来回横跳，甚至越跳越远离谷底——步长太大，每一步都跨过了头；还有一条方向对、步子碎，走出很远还在山腰——步长太小。三条路径的命运从第一步起就注定了，而决定步长的那个数，正是下一节的主角。

这片山地还有一个绕不开的地形问题：谷底不止一个。损失面上有**局部极小**（local minimum，四周都比它高的小坑），也有**鞍点**（saddle point，这个方向看是谷、换个方向看是山脊的“马鞍口”）。20 世纪 90 年代大家最怕局部极小——怕小球掉进半山腰的小坑再也出不来。但 2026 年的共识恰恰相反：现代模型的参数动辄百万上亿，在如此高维的山地里，**某一点要在所有方向上都恰好是坑底，概率极低；真正拦路的，是大量的鞍点和平坦高原**。这带来一个实用的副产物：训练“卡住不动”时，往往不是掉进坑里，而是正走在平地上——多等几步、换一批数据（加大批量内的噪声）或微调学习率，小球常常自己继续往前滚。下次你的曲线停住，先别绝望，也别乱调，等一等再看。

3.2　学习率：整张清单上最重要的一个数

学习率（learning rate）η 已经出场过两次：第 1 讲感知器挪线的步长、第 2 讲梯度下降的步伐。现在把它讲透。它是所有超参数里**影响力最大的一个**：同一份代码，只改它，能从“三分钟收敛”变成“永远训不动”。判断它合不合适，最直接的仪表是 loss 曲线。下图是同一个网络用三个不同学习率训练的曲线，三种病症一目了然。

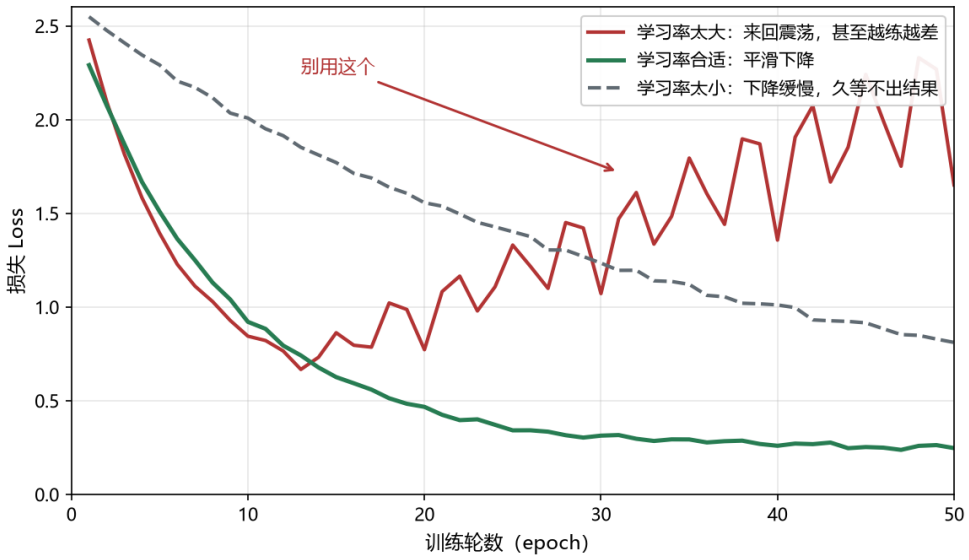


图 3.2　同一网络三个学习率的 loss 曲线：太大（剧烈震荡甚至上行）、合适（快速下降后趋平）、太小（缓慢下降，远未收敛）

逐条读图。**太大**：每步都跨过谷底，小球在两壁间弹来弹去，loss曲线剧烈震荡；再大一点，一步弹出场外，数值溢出，loss 变成 nan（not a number，非数）——上一讲说的现行犯一号在此归案：**训练打印出 nan，第一嫌疑人是学习率过大**。**太小**：曲线缓慢下降，看着有救，半天还在半山腰——GPU 烧着、曲线在动，你不敢停、停了又亏。**合适**：前段快速下降、后段趋于平坦，像一条滑梯，一路滑到谷底。

更进一步：好的训练往往不是“一个学习率用到底”，而是**先大后小**——前期大步赶路，后期小步精修。让学习率按计划变化的做法统称**学习率衰减**（learning rate schedule），三款常见配方混个脸熟即可：**阶梯衰减**（每过若干轮减半）、**余弦衰减**（沿余弦曲线从初值一路滑到近 0，2026 年视觉训练的常见默认）、**warmup**（开头几百步从极小值慢慢升到目标值——刚开训时参数还是“乱码”，太早迈大步容易当场翻车，大模型训练标配）。名词现在记住，实验四你会亲手拧。

先在纸上把“火候”的感觉找出来。

动手时间：用纸笔把三个学习率各“训”五步

取一维损失函数 $f(w) = (w-3)^2$（谷底在 $w=3$），它的梯度是 $2(w-3)$。用更新式 $w \leftarrow w - \eta \cdot 2(w-3)$，从 $w=0$ 出发，把三个学习率各手算三步：

	$\eta = 0.05$（小火）	$\eta = 0.9$（中火）	$\eta = 1.1$（大火）
第 1 步后	0.30	5.40	6.60
第 2 步后	0.57	1.08	−1.32
第 3 步后	0.81	4.54	8.18

对答案：大火每步跨过谷底越跳越远（发散，再算几步就“nan”）；中火在谷底两侧来回弹（震荡）；小火稳稳向 3 爬但一步只挪零点几。请你继续把小火算到第 10 步——离 3 还有约一步之遥，这就是“龟速”的体感。整个过程只要一张草稿纸，你刚刚手工执行了三种“火候”下的梯度下降。课后编程题会让你把这三条曲线用代码真画出来。

3.3　权重初始化：第一步就走对

训练开跑前，权重总得有个初值。最省事的念头：全设 0。这恰恰是灾难，而且灾得很有趣。想想第 2 讲的反向传播：同一隐层的所有神经元，输入相同、权重相同，前向算出的输出一模一样；反向分账时它们收到的责任也一模一样，更新之后依旧相同——**一百个神经元永远步调一致，等价于一个神经元**，网络白白胖胖，能力却是“独苗”。这叫**对称性没有被打破**（symmetry breaking）。好比一个班发一模一样的课本、做一模一样的作业，还要求答案分毫不差——永远长不出各有专长的专家。所以初值必须随机，且各个神经元各不相同。

随机，但随成什么样也大有讲究。方差太大：信号一层层放大，传到深处激活值爆表，sigmoid/tanh 全员进入饱和区（回忆第 2 讲那片“导数近乎为零的平原”），梯度消失，loss 当场 nan——现行犯二号归案。**方差太小**：信号层层萎缩，到深处激活值全趴在 0 附近，网络整体“昏迷”。于是“刚刚好”的标准非常清晰：**让信号穿过每一层后，方差不爆不缩**——文献里叫方差匹配（variance matching）。

按这个原则，有两个记名字就能用的经典配方：**Xavier 初始化**（Glorot 等，2010，适合 sigmoid/tanh）：权重方差取 $2/(n_{\text{in}}+n_{\text{out}})$；**He 初始化**（何恺明等，2015，适合 ReLU）：取 $2/n_{\text{in}}$。人话翻译：n_{in}、n_{out} 是该层进、出的连线数——**连线越多，每个权重就该越“谦虚”（越小）**，免得把一层的信号加爆。现代框架里通常一行代码自动完成，但出了问题你得知道它在防什么病。

Xavier、He 与 Glorot 条件——一句话版

为什么 Xavier 的方差是 $2/(n_{\text{in}}+n_{\text{out}})$，He 的却是 $2/n_{\text{in}}$？理想标准（文献称 **Glorot 条件**）其实有两条：信号**前向**穿过每层后激活方差不变，误差**反向**穿过每层后梯度方差不变。麻烦在于**两条不可兼得**——前向想把方差除以 n_{in}，反向想除以 n_{out}，于是各家做了不同取舍：

初始化	适用激活	方差条件（一句版）
Xavier	tanh / sigmoid	两条各让一步的折中：$2/(n_{\text{in}}+n_{\text{out}})$
He	ReLU	ReLU 掐掉一半信号，改为全力保前向：$2/n_{\text{in}}$
LeCun	Leaky ReLU / SELU	He 思路的对应保守版：$1/n_{\text{in}}$

人话：Xavier 是“和稀泥大师”，两个条件各让一步，适合对称激活；He 发现 ReLU 把负半边全掐了，干脆牺牲反向条件、保住前向激活方差不为零；LeCun 初始化则是同一思路在斜率较小的整流激活（Leaky ReLU 一族）上的对应版。框架里都是拼一个名字的事（`xavier_init`、`kaiming_init`、`lecun_normal`），但报出名字，你得知道它在保什么。

下图就是上一讲加分项里留下的“预告片”正式上映：同一网络、同一数据、同一学习率，只有初始化不同，三条曲线命运迥异。全 0：曲线平得像心电图停机——对称性没破，网络学了个寂寞；太大：前段剧烈震荡甚至起飞；Xavier：平滑下降，一路健康。至此第 2 讲承诺的两个现行犯全部抓捕归案。**训练出问题的排查顺序：先查学习率，再查初始化，最后才怀疑更深的病。**

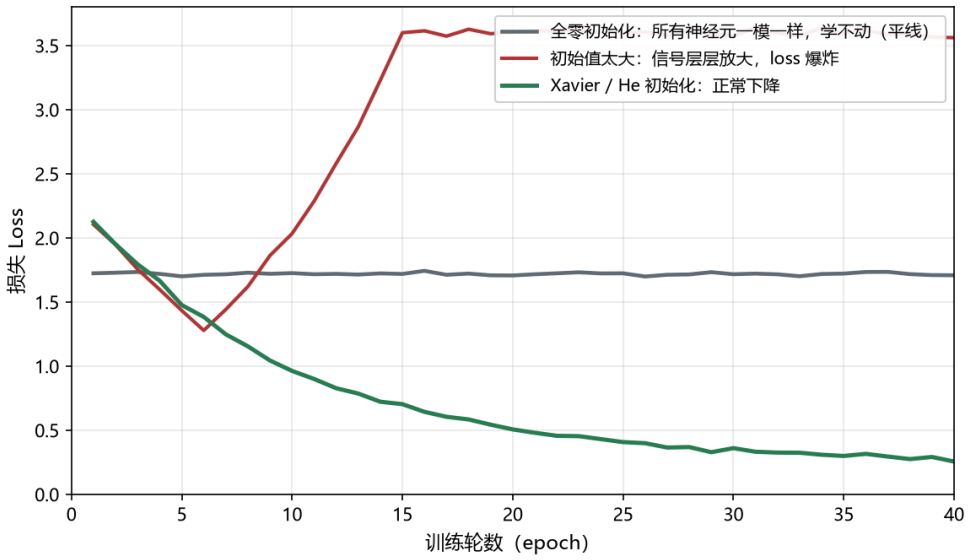


图 3.3　三种初始化的训练曲线对比：全 0（纹丝不动）、方差太大（震荡甚至爆炸）、Xavier（平滑收敛）

3.4　batch、epoch、iteration：分清训练的三兄弟

实操里还有三个容易搅浑的词，用吃饭比喻一次分清。训练数据是**全菜单**；**batch**（批）是一口吃的样本数，这一口有多大叫 batch size；**iteration**（迭代）是“咬一口、消化一次”——前向、算损失、反向、更新，完整一圈；**epoch**（轮）是**把整份菜单从头吃完一遍**。算笔账：1000个样本、batch size 取 100，一个 epoch 就是 10 次 iteration；训练 50 个 epoch，参数总共被更新了 500 次。

那“一口吃几个”怎么定？这是一笔典型的权衡账：

表 3.1　batch size 的权衡：大口稳但贵，小口省但吵

batch size	优点	代价
大（如 256、1024）	梯度估计稳、路线直；GPU 并行吃得满，单位时间效率高	显存开销大；口太大“路太顺”，泛化有时反而变差
小（如 8、32）	显存友好；噪声大，反而帮小球跳出鞍点，泛化常常更好	曲线抖、得配更小的学习率；单位样本效率低

两个极端各有名字：batch 取 1 叫随机梯度下降的“在线”模式，batch 取全量叫（全）批量梯度下降；夹在中间的叫 **mini-batch**（小批量）——现代训练的默认形态，以至于 2026 年大家说“SGD”时，指的几乎都是 mini-batch 版。它的更新式是本讲唯一的公式：

知识补齐：mini-batch SGD 的更新式——现代训练的底座

每一步，从训练集随机抽一小批 $\mathcal{B}$，用它估出的梯度下山：

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}} L(\mathcal{B}; \boldsymbol{\theta}_t), \tag{3.1}$$

人话：$\boldsymbol{\theta}$（粗体西塔）把全部权重和偏置打包装成一袋；不用全部数据算坡度（太贵），而是每步随机抽一小口当“抽样调查”，用抽样坡度代替真实坡度下山。噪声是特性不是缺陷：它让小球在鞍点附近也晃得动（呼应 3.1 节）。往第 2 讲的十行代码里加两行，就是标准的 mini-batch 训练：

```python
for epoch in range(E):              # 外圈：全菜单吃 E 遍
    np.random.shuffle(idx)          # 每轮先洗牌，吃的顺序随机
    for B in batches(idx, bs):      # 内圈：切成一口一口
        theta -= lr * grad(theta, B)   # 一口一更新
```

再补一条工程常识：同一份代码你跑两次、同学跑一次，曲线并不重合——不是玄学。初始化、洗牌都是随机的，由**随机数种子**（*random seed*）统一控制。做对比实验（比如本章课后题）前先把种子固定，否则你分不清差异来自超参数还是来自运气。

3.5　训练集、验证集、测试集：背题与会做题

到目前为止我们默认“loss降 = 学好了”。这里有个陷阱：那个 loss 是在**训练数据**上算的。一个记忆力超群的学生把例题全背下来，做原题可以满分——换一道新题立刻露馅。而训练要的是**会做题，不是背题**。所以必须把手头的数据切成三份，扮演三个角色：**训练集**（training set，课本例题，用来更新参数）、**验证集**（validation set，平时小测，用来调超参数、判断何时收手）、**测试集**（test set，高考，只许在最后动用一次，报告最终成绩）。比例常见 8:1:1 或 7:2:1，数据特别少时还有“交叉验证”这种轮流坐庄的切法（记住名词即可）。

一条铁律先立起来：**所有训练中的决定——换学习率、早停、选模型——只能看验证集**。原因看下图就懂：训练 loss 一路下降，验证 loss 却先降后升；两条线开始“张口”的位置，就是学生从“会做题”滑向“背题”的转折点，图中标注的最佳停止点，就是该收工的地方。这个白捡的超参数叫**早停**（early stopping）：验证 loss 最低的那一轮就停手，别恋战。

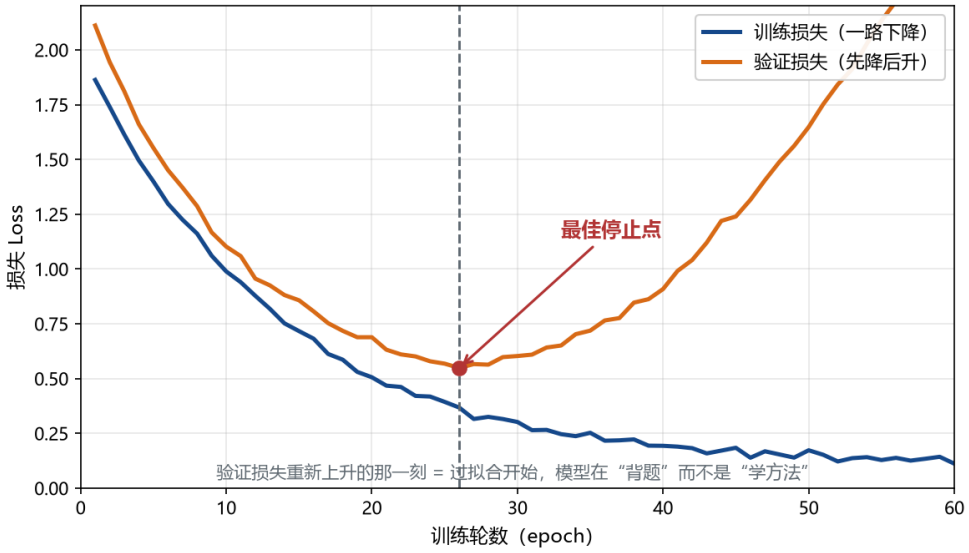


图 3.4　训练 loss（一路下降）与验证 loss（先降后升）双曲线：两线张口即过拟合起点，标注处为最佳停止点

两种典型病症总结成一张表，训练时对着自查：

表 3.2　训练病症速查：看两条 loss 曲线下诊断

病症	训练 loss	验证 loss	人话诊断	药方
过拟合	一直降	先降后升	背题了	更多数据；正则化；早停；dropout
欠拟合	高且平	高且平	压根没学会	加大网络；训练更久；换更好的特征
刚刚好	降后趋平	贴着训练线降后趋平	健康收工	记下超参数，出成绩

表里两味新药各一句话入门：**正则化**（regularization，给过大的权重“加税”，逼网络学简单的规律）；**dropout**（训练时每一步随机“掐灭”一部分神经元，逼网络不依赖任何单个“学霸”，测试时全员上岗）。详细的用法第 6 讲见。

防糊弄：高考卷一张纸都不许提前看

用测试集调参数，等于把高考真题当练习册刷——成绩必然虚高，而且你永远不知道自己虚高了多少。这不是洁癖，是工程伦理：2024–2026 年大模型评测里，“benchmark 污染”（考题混进了训练数据）是审稿人重点打假的对象，一个刷爆榜的模型若考题不干净，成色立刻存疑。落到你们的项目验收上：六目九头的 A/B 对比里，`metrics.json` 中的每个指标都必须来自验收方指定、你没在上面调过参的数据。切分数据的那一行代码，要在训练开始之前写好、锁死。

3.6　向量化编程：为什么一行 numpy 顶一百行 for

实验三你会在真实数据上训练，最先撞上的墙往往不是数学而是速度：同一份作业，同学的代码一分钟出结果，你的跑一晚上还没收敛到一半。九成情况下，差距不在模型，而在**向量化**（vectorization）——把“逐个元素的循环”改写成“对整个数组一次性运算”。

为什么能快这么多？人话版：Python 解释器执行 for 循环时，每一圈都要“重新读一遍代码再执行”，开销巨大；而 numpy 的数组运算在底层是编译好的 C/Fortran 线性代数库（BLAS），一次把整块数据算完，还自动用上 CPU 的向量指令；数据若在 GPU 上，更是几万个小核心同时开工——第 2 讲那行 `X @ W1`，底下是上百万次乘加同时发生。实测感受一下：

```
import numpy as np                  # 导入 numpy，社区习惯缩写为 np
n = 1_000_000
x = np.random.rand(n)               # 一百万个随机数

total = 0.0                         # 写法一：朴素 for 循环
for i in range(n):
    total += x[i] ** 2              # 一次只算一个数

total_v = np.sum(x ** 2)            # 写法二：向量化，一行顶一循环
# 本机实测（量级）：写法一约几百毫秒，写法二约几毫秒
```

下图是这类对比的量级示意：同样算一百万个数的平方和，循环版与向量化版的耗时约 100:1。今后写训练代码，凡是出现“`for i in range(样本数)` 然后逐个处理”的地方，先停下来想一想：能不能整块算？

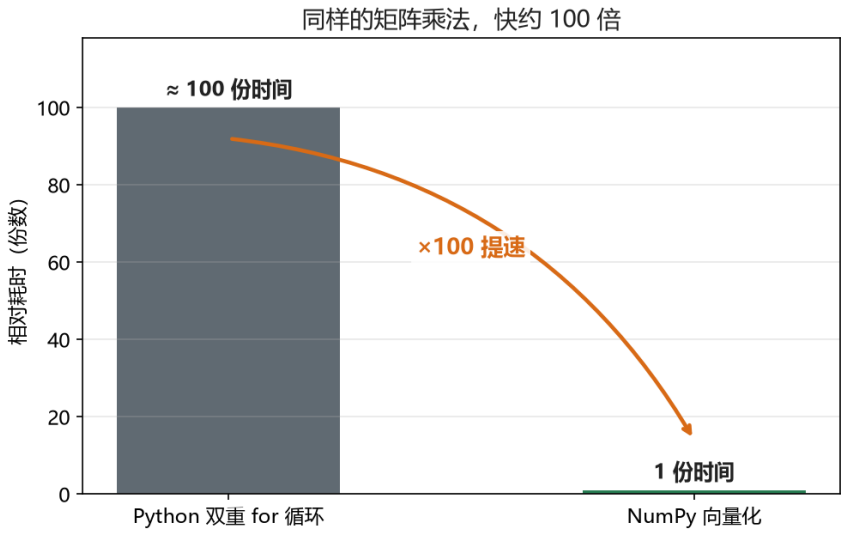


图 3.5　同样的计算，Python 循环与 numpy 向量化的耗时对比（量级示意约 100:1）

知识补齐：numpy 广播——形状不同的数组也能直接运算

广播（broadcasting）规则一句话：两个数组从最后一维往前对齐，每一维要么相等、要么其中一方是 1（是 1 的一方可自动“复制”扩展到匹配）。两个天天用的例子：

- `X -X.mean(axis=0)`：`X` 形状 $(100, 4)$，均值形状 $(4,)$，广播成 $(100, 4)$——一行完成“每个特征减去自己的均值”，这是数据标准化的核心动作。
- `W * s`：`W` 形状 $(3, 3)$，`s` 形状 $(3, 1)$，每行乘上不同的系数——一行完成“逐行缩放”，反向传播按行算梯度时天天用。

人话：广播就像自动复印粘贴，但并不真的占用那份内存。乘法、减法、比较都支持。

3.7　优化器家族一分钟：从 SGD 到 Adam

第 2 讲的更新式 $w \leftarrow w - \eta \partial L/\partial w$ 是最朴素的 **SGD**（stochastic gradient descent，随机梯度下降）：看到坡就迈步，所有参数共用同一个步长。七十年里，一批“改良版下山算法”陆续登场，本讲一分钟讲清最常用的四位——不推导，只记人话：

表 3.3　优化器家族速览：四位选手，各一句人话

优化器	一句人话	出生	点评
SGD	看坡下山，步长恒定	1951	朴素；调好了至今仍是 CNN 训练强手
Momentum	给小球加惯性，冲过小坑小坎	1964/1986	沟壑地形明显提速
RMSProp	每个参数自备步长：常走的路小步，少见的方向大步	2012	“自适应步长”代表作
Adam	Momentum + RMSProp：既带惯性又自适配步长	2015	2026 年的默认首选

用做菜的话收束：SGD 是死火候；Momentum 给锅装了个匀速转柄，冲劲带着它碾过小颠簸；RMSProp 给每根管子配了自己的阀门；Adam 说“我全都要”。实践口诀两条：**拿不准就用**

Adam，学习率 0.001 起步（Adam 对学习率不敏感，是它称王的主要原因）；卷积网络刷榜与论文复现里 SGD+Momentum 依然常见——它调好后泛化上限常常更高。四位在实验三的代码里都只是 `optimizer=` 后面换一个词的事，但你现在知道换的到底是什么。

前沿速递：训练一个大模型要烧多少电

把本讲的旋钮放大十亿倍，就是大模型的账单。GPT-3（2020 年，1750 亿参数）论文自报：一轮训练约 3.14×10^{23} 次浮点运算，用电约 129 万度——够一个三口之家用四百多年。GPT-4 级（2023 年）外界估算再高一到两个数量级：数千万度电，上万张 GPU 连转数月。于是 2024–2026 年上演了一场**效率竞赛**：**混合精度**（用低精度数字做大部分运算，显存与算力约省一半）、**FlashAttention**（2022 年，把注意力计算重新排布，又快又省显存）、**MoE 稀疏专家**（每次只唤醒一小部分参数干活）；2024 年底 DeepSeek-V3 公开报告用约 278.8 万 GPU 小时完成训练，把同级模型的账单砍低一个量级，引发全网对“训练成本”的重估。而“该烧多少”的理论依据是 **scaling law**（规模定律），三个词：**更大模型 + 更多数据 + 更多算力 = 可预测地更强**（2020 年提出；2022 年 Chinchilla 修正：数据要和参数同步加）。反过来看：正因为每次训练都贵得离谱，“用小实验先摸超参数、再放大”的调参手艺在 2026 年依然是稀缺技能——你每避免一次无效训练，省的都是几百万美元。本讲就是这门手艺的第一课。

项目连线：实验三命令里的每个词，都是本讲概念

实验三你会敲这样一条命令：`python train.py --epochs 200 --batch 8 --lr 0.001`。拆开看：`--epochs 200`，全菜单吃 200 遍；`--batch 8`，一口吃 8 个样本（显存小的机器跑得起的量级，回忆 batch 权衡表）；`--lr 0.001`，火候拧在 Adam 的常用起点。命令行就是往本讲的超参数字典里填空。**六目拼接**：九个参数头**逐头独立训练**——每个头一份自己的超参数字典（学习率、epoch、batch 各不相同，分类头和回归头脾气不同）；验收标准“逐头指标不劣于规则初值才允许替换”，意味着每个头都要做本讲的训练/验证切分、A/B 对比，并把成绩写进同名 `metrics.json`（第 一 章）。**水下双目**：学生模型从教师伪标签微调，采用“5 个 epoch 一道闸门”制度：每 5 个 epoch 在验证图对上测一次端点误差 EPE（end-point error，预测视差与真值视差的平均距离），不降即停——本讲的早停被写成了制度（第 二 章）。**数据基座**：问数系统里 Gemma 的**温度 0**（temperature 0：永远选最确定的词，不许自由发挥）与上下文长度 8192（一次能读进的 token——词元，大模型处理文本的最小单位——的数量）也是超参数：回答要“每次都一样”，靠的就是把温度拧到 0。它们与 `--lr` 本质同类：模型之外、人来定（第 三 章）。另外预告：实验四（12 月 21 日）的训练诊断环节，还有梯度检查的实战任务。

3.8　从这一讲到 2026

本讲的“训练集—验证集—测试集”三段论，正在被 2024–2026 年的实践改写，改写发生在三个方向。第一个方向：**数据有墙**。“更多数据就更强”的缩放定律（回忆章末前沿盒子里的 Chinchilla 修正）撞上了人类文本的储量上限——高质量公开文本被估计将在 2026 年前后“有效耗尽”，业界于是转向**合成数据**（让模型生成数据再来训模型）与多 epoch 重复利用。第二个方向：训练的重心从预训练（第 4 讲的“先学世界观”）移向**后训练**（post-training，基座模型训完之后的第二阶段，教它具体本事）——DeepSeek-R1 证明，不用人工示范答案、直接上大规模强化学习去“激励”，模型自己就能练出逐步推理的本事，这项工作 2025 年登上了《自然》封面。第三个方向最扎心：**评估会漏**。静态基准的考题会混进训练数据——正是本讲防糊弄盒子里那个“benchmark 污染”，它已严重到逼

出换代方案：每月换新题的**活基准**（live benchmark）成了主流，连顶级模型在其上的准确率也不到七成。数据有墙、评估会漏，唯有本讲教的训练方法本身，仍在快速进化。

2026 现场：关键事实

- Epoch AI 分析：按 Chinchilla 最优缩放，高质量人类公开文本将在 2026 年前后被有效耗尽，大模型必须转向合成数据、多 epoch 训练与推理时缩放（不加大模型、靠让它多想几步换性能）等新路径（来源：Epoch AI，“Will we run out of data to train LLMs?”，2024/2025）。
- DeepSeek-R1（2025 年 1 月 arXiv 预印本，2025 年 9 月发表于《自然》并登封面）：在大规模纯强化学习下，未经任何监督微调的 R1-Zero 自发出“自我反思、验证”等推理行为（来源：Guo et al.，Nature 645，633–638，2025）。
- LiveBench（ICLR 2025 Spotlight）：每月用最新数学竞赛、arXiv 论文与新闻出题、按客观真值自动评分，把基准污染压到最低；顶级模型在其六大类别上的准确率也低于 70%（来源：White et al.，ICLR 2025；livebench.ai）。
- 活基准已成体系：LiveNewsBench、LiveOIBench、DyCodeEval（ICML 2025）等相继出现，都用持续换题 / 动态生成对抗静态基准的数据泄漏（来源：各论文官方页面，2025）。

一句话收束：**你在这门课里学会的“留出测试集”，在 2026 年成了整个行业最难的事——测试题还没开考就可能已在训练集里，于是评测本身也必须“活”着。**

本讲的课后题，就是把上面所有概念放进同一份代码里做对照实验——这是“调参手艺”的第一次独立操练。五要素如下。

第 3 讲课后编程题（用 AI 实现）

题目：指挥 AI 辅助，做一个**超参数对比实验**。数据集用 sklearn 的双月牙 `make_moons`（或第 2 讲的 XOR，二选一）；训练同一个两层隐藏层 MLP 三次，**每次只改一个超参数**：三个学习率（如 0.001、0.1、10），或三种初始化（全 0、方差太大如标准差 1、Xavier）。其余设置完全相同（含随机种子）。用 matplotlib 把三条 loss 曲线画进同一张图（带图例），保存为 png；并写一段 50 字以上的结论：哪组最好、其余两组各得了本讲里的哪种“病”（用震荡、龟速、对称性、方差匹配这些词说）。

交付物：(1) 代码文件 `hyper_compare.py`；(2) 对比曲线 png 一张；(3) 50 字以上的实验结论；(4) 一段 50 字以上的“我改了 AI 代码里的什么、为什么”。

提示词建议（可直接抄）：“我是零基础学生，正在学超参数。请用 numpy（不用 PyTorch）实现：(1) 用 sklearn 的 make_moons 生成 200 个样本（噪声 0.2），按 8:1:1 切分训练/验证/测试集并标准化；(2) 一个 2-16-16-2 的全连接网络，ReLU 激活，输出层 softmax 配交叉熵损失（请用两句话给我解释 softmax 和交叉熵）；(3) 用普通 SGD（不用 Adam，这样才能看清学习率的威力），batch size 32，训练 500 个 epoch，随机种子固定 42；(4) 命令行参数 –lr，取 0.001、0.1、10 各训一次，其他完全相同；(5) 把三条训练 loss 曲线画进同一张图，带图例与中文标题，保存 png，并打印三组的最终验证准确率；(6) 每行中文注释，标明哪一行是学习率、哪里是初始化、哪里是 mini-batch 切分。”

验收线：代码能跑，三条曲线对比清晰、结论与曲线对得上（例如“`lr=10` 震荡甚至 nan，`lr=0.001` 龟速未收敛”）；抽问时能讲清 batch、epoch、iteration 三者的区别与 batch size 的权衡；至少一处自己的修改（换成三种初始化对比、加早停、改 batch size 都算）。**代码是 AI 写的没关**

系，讲不清楚就是没做。
加分项（任选其一）：(1) 给学习率加“阶梯衰减”（每 200 轮减半）重跑，对比曲线尾部形状；(2) 把数据集噪声从 0.2 加大到 0.4，看三种学习率的命运差距如何拉大；(3) 开训前先做第 2 讲的梯度检查，把“通过后再训练”写成流程。

课后阅读：教材《深度学习模型与算法基础》第 31–35、48–50 页，与本讲内容对应；读的时候带着本讲的“旋钮清单”去对照，看书上哪些词你已经能用人话复述。

超参数（hyperparameter）
训练开跑前由人设定的旋钮，如学习率、batch size、轮数；区别于训练中由反向传播自动学习的参数。

学习率（learning rate）
每步参数更新的步长，最敏感的超参数：太大震荡甚至爆炸成 nan，太小龟速；常配衰减策略（阶梯、余弦、warmup）。

权重初始化（weight initialization）
训练前给权重的随机初值：全 0 破坏不了对称性，太大梯度爆炸，太小梯度消失；经典配方 Xavier 与 He，原则是方差匹配。

batch / iteration / epoch
一口吃的样本数 / 咬一口消化一次（更新一次）/ 全菜单吃完一遍；batch size 是“稳与贵、噪与省”的权衡。

mini-batch SGD
每步随机抽一小批数据估梯度并更新的算法，现代训练的底座；噪声帮小球跳出鞍点。

训练集 / 验证集 / 测试集
课本例题 / 平时小测 / 高考；一切训练中的决定只看验证集，测试集只许最后用一次。

过拟合 / 欠拟合
背题了 / 没学会；用训练与验证 loss 的“张口”与“齐平”诊断，药方分别是更多数据与正则、加大网络。

早停（early stopping）
在验证 loss 最低处收工，白捡的超参数。

向量化（vectorization）
把逐元素循环改写成整块数组运算，numpy 与 GPU 提速百倍的根源。

Adam
动量与自适应步长的合体优化器，对学习率不敏感，2026 年的默认首选；拿不准就用它，0.001 起步。

第 4 讲　自编码器

把一张 3 MB 的照片发给朋友，你的第一反应是压成 zip——压缩程序的本事，是在数据里找重复规律，抠掉冗余。可如果压缩对象是一本 1001 页的设备手册呢？人类的压缩方式是写一页摘要：摘要没法一键还原出全文，却留住了最有用的信息。**这种压缩需要“理解”**。**自编码器**（autoencoder，常缩写为 AE）就是教神经网络学会“理解式压缩”的第一课：给它看大量数据，让它自己琢磨什么该留、什么该扔。

更妙的是，这一课**不需要任何标签**——数据自己就是自己的标准答案。这为理解 2026 年最重要的一件事埋下伏笔：GPT 这类大模型的看家本领，根基正是这种“无标签、自己出题”的学习方式。本讲从自编码器出发，一路走到“基础模型”时代的表示学习——你会看到，第 3 讲学的全套训练手艺，在一个没有标签的世界里照样通用。

本讲地图

讲什么：无监督与自监督学习为何重要；自编码器的沙漏结构与重建损失；隐空间与插值；去噪自编码器；稀疏 AE 与 VAE 一分钟；预训练—微调范式。**为什么值得学**：自编码器是“表示学习”的敲门砖；三个项目里 FoundationStereo 的隐空间、拼接头的“指纹”，玩法源头都在本讲。**学完你能**：画出沙漏结构并说清瓶颈的用意；解释隐空间插值为什么是“渐变的人脸”；在 sklearn 手写数字集上训一个自编码器，并画出隐空间的散点图。

4.1　无监督学习登场：没有标签的世界

前两讲的训练都长一个样：输入配标准答案（花瓣尺寸配花种、像素配数字），网络对着答案改错。这叫**监督学习**（supervised learning）——“监督”指答案像监考老师一样全程在场。好用，但贵：标签要人一条条标。ImageNet 为了给 1400 多万张图片打标签，动用了数万名众包标注员干了几年。而现实世界的处境恰恰相反：**数据泛滥、标签稀缺**——全网有几十亿张照片，没有哪张自带“这是猫”的标签；你们水下项目的 500 对锚链立体图对，在教师模型来打伪标签之前，也没有一个像素被标注过（第 二 章）。

于是换思路。**无监督学习**（unsupervised learning）不给答案，让模型从数据自身找结构：聚成几堆、压成几维、谁和谁像。自编码器就是其中最直观的一员——把输入喂进去，让网络把输入**重建**出来。听起来像左手抄右手？玄机全在中间那道“窄门”，下一节见。而在两者之间还有第三条路——**自监督学习**（self-supervised learning）：不请人来标，但把数据自己加工成“题目 + 答案”：遮住一个词让模型猜词、抠掉一块图让模型补图。题目和答案都从数据里来，一分钱标注费不花。

表 4.1　三种学习范式对比：谁出题、谁给答案

范式	谁出题	答案从哪来	2026年的代表
监督学习	人出题（配好标签）	人给的标签	图像分类、目标检测
无监督学习	数据自身	无答案，只找结构	聚类、自编码器、降维
自监督学习	数据自己加工出题	藏在数据里	GPT 预训练（猜下一词）、BERT（猜被遮的词）、MAE（补图块）

给 2026 年留个伏笔：大模型的“预训练”阶段吃下几乎整个互联网的文本，用的正是自监督出题；等它学完“世界观”，才轮到少量人工数据做“专业课”。下图是这条主线的历史全景——注意 2017 年之后最粗的那根线，就是自监督预训练。

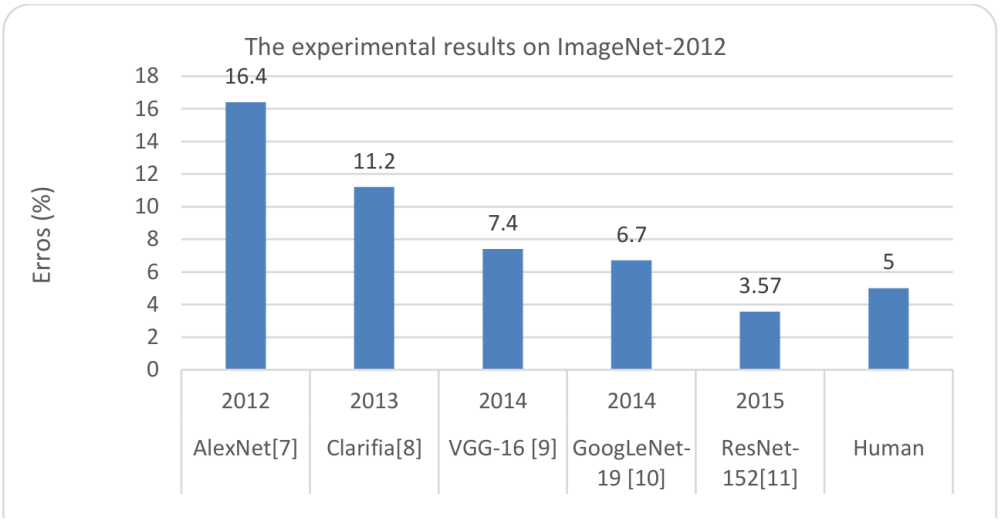


图 4.1　深度学习发展的里程碑时间线：从感知器到大模型，无监督与自监督是贯穿后半场的主线

图片来源：维基百科 Deep learning 词条

4.2　自编码器结构：自己压缩自己

结构看下图，一个“沙漏”：左边**编码器**（encoder）逐层变窄，负责压缩；正中间最窄处叫**瓶颈**（bottleneck，也叫隐层），信息必须过这道独木桥；右边**解码器**（decoder）逐层变宽，负责还原。训练目标只有一条：**输出尽量等于输入**。度量它的叫**重建损失**（reconstruction loss）：

$$L = \left\|\mathbf{x} - \hat{\mathbf{x}}\right\|^2, \tag{4.1}$$

人话：$\mathbf{x}$ 是输入，$\hat{\mathbf{x}}$（读“x 尖”）是重建输出，把两者逐个维度求差、平方、加总——重建得越像，损失越小。这就是第 *1* 讲的平方误差换了个舞台。

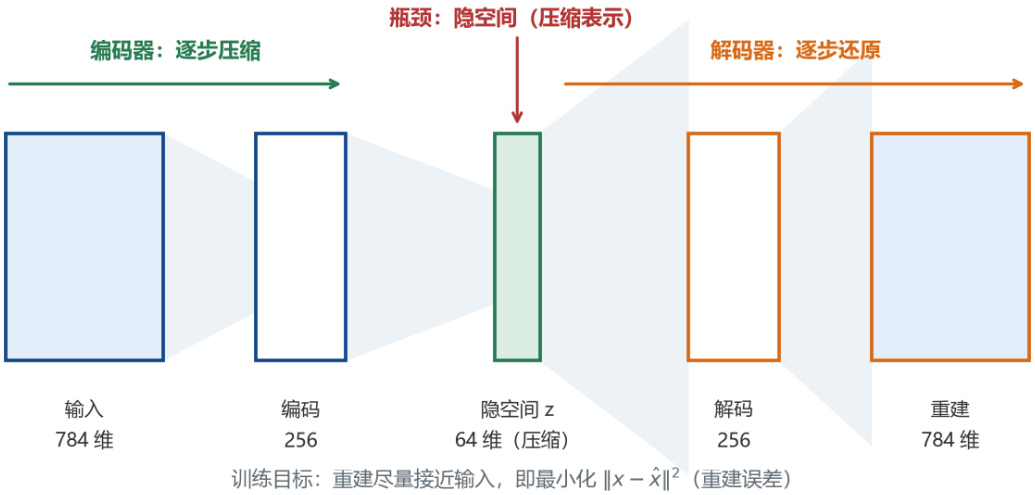


图 4.2　自编码器沙漏结构：784 维输入（28×28 手写数字）经编码压到 64 维隐码，再解码重建回 784 维；中间的窄门逼网络学会取舍

瓶颈为什么必须窄？做个思想实验：如果中间层比输入还宽，网络会学会最无耻的解法——恒等映射，把输入原样抄到输出，重建损失为 0，但什么也没学到，等于花一个亿盖了个复印机。**瓶颈是故意收的“信息税”**：64 维装不下 784 维的全部细节，网络被逼到墙角，只能学会分辨“哪些是本质、哪些是边角料”。压缩的意义由此产生。训练流程与第 3 讲毫无二致：前向、重建损失、反向传播、更新——唯一的区别是“标签”就是输入自己。一句话总结：**自编码器就是用反向传播训练出来的、可学习的压缩算法**。

生活化类比：zip 与自编码器——两种压缩的对话

zip 是无损压缩：逐字节找重复，能一字不差还原，但它“不懂内容”，对毫无重复规律的照片就无能为力。自编码器是有损压缩：丢掉它认为不重要的细节，还原出来的是“神似”而非像素级相同。这笔亏换来的赚头是：**隐码带着语义**——两张都像“8”的图，隐码也挨得近；zip 的压缩结果没有这种性质。前者像复印机，后者像“看完电影给朋友复述剧情”：说不全，但抓住了主线，而且两部电影若剧情相似，复述稿也相似。

4.3　隐空间：压缩后的向量是“本质坐标”

瓶颈处那个 64 维向量，叫**隐码**（latent code）。换个视角看它：每张图被映射成 64 维空间里的一个点，这个空间叫**隐空间**（latent space）。训练好的自编码器有一个漂亮性质：**长得像的输入，在隐空间里挨得近**——两张“8”的隐码距离，远小于“8”和“0”的距离。换句话说，隐空间把“像素的海洋”重新组织成了一张“按本质分类的地图”：下图中同类样本自然聚成簇，不需要任何标签。

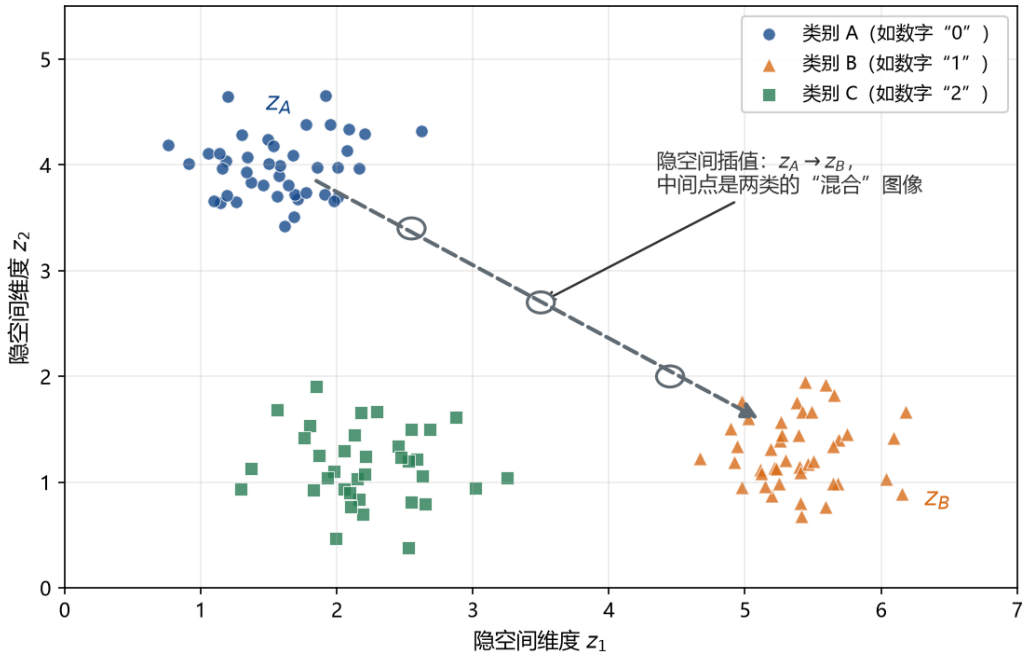


图 4.3　二维隐空间散点（按类别着色）：同类样本聚成簇；两簇之间的插值箭头表示隐码沿连线渐变时，解码出的结果平滑过渡

更神奇的是**插值**（interpolation）。取两张人脸的隐码 $\mathbf{a}$、$\mathbf{b}$，把连线上的点 $\mathbf{z} = (1-t)\,\mathbf{a} + t\,\mathbf{b}$（$t$ 从 0 到 1）逐个送进解码器，得到的是从一张脸平滑渐变到另一张脸的过渡序列：眉毛、鼻子、嘴巴逐个“变形”。注意这不是两张照片的像素叠加——叠加只会得到重影——而是每一点都由解码器重新“画”出一张完整合理的脸。人话翻译：**隐码不是照片的存档，而是脸的参数旋钮；旋钮连续转，脸**

就连续变。这说明网络学到的表示，抓住了“生成一张脸”的本质自由度。这个性质立刻能变现：高维数据降到二维看结构（可视化）、离群点检测（隐空间里的孤点）、以及最重要的——**表示**：既然隐码就是“本质特征”，就可以拿它去做分类、检索等其他任务，全程不用标签。这最后一步跨越，是 4.5 节和前沿盒子的主题。

先玩两个纸面游戏，把“瓶颈”和“插值”变成体感。

动手时间：当一回人肉自编码器，再手算一次插值

游戏一（人肉自编码器）：两人一组。甲看一幅简笔画 10 秒，然后用**不超过 10 个词**描述给乙（这 10 个词就是“瓶颈”——你被迫做取舍）；乙凭描述把画复原。复原结果当然与原图不同，但只要“是一只戴帽子的猫”这类本质还在，压缩就成功；若乙画出了一只没戴帽子的狗，说明瓶颈太窄，本质丢了。试三个瓶颈宽度：10 个词、5 个词、2 个词，看信息在哪一档崩掉。**游戏二（纸笔插值）**：取两个隐码 $\mathbf{a}=(1,0)$、$\mathbf{b}=(3,2)$，在方格纸上标出这两点，用 $\mathbf{z}=(1-t)\mathbf{a}+t\mathbf{b}$ 取 $t=0.25, 0.5, 0.75$ 算出三个中间点（$(1.5,0.5)$、$(2,1)$、$(2.5,1.5)$），标在连线上——四个点等距排列。想象每个点都由解码器还原成一幅图：这就是“渐变配方”。课后题里你会用真网络把它做出来。

知识补齐：TSNE——把高维点云拍扁到纸上

16 维的隐码没法直接画。**TSNE**（t-distributed Stochastic Neighbor Embedding，t 分布随机邻域嵌入）是最常用的“拍扁”工具，一句话原理：**尽量让高维空间里挨得近的点，在二维图上也挨得近，同时把不同的簇彼此拉开**。用法两行：

```
from sklearn.manifold import TSNE
Z2 = TSNE(n_components=2).fit_transform(Z)   # Z 是全部样本的隐码
```

注意：TSNE 只用来**看**，不用来训练；图上簇的形状、大小、簇间距都别过度解读，重点是“同类分不分得开”。

4.4　去噪自编码器：先弄脏，再学会擦

普通自编码器有个隐患：如果瓶颈不够窄，网络可能练成“死记硬背型复印机”——重建得很好，但没有学到结构。2008年提出的**去噪自编码器**（denoising autoencoder，DAE）的解法带着一股物理实验味：**先把输入弄脏，再要求还原出干净的**。具体说：给输入 $\mathbf{x}$ 加噪声得到脏版本 $\tilde{\mathbf{x}}$，网络看的是脏图，损失却对着干净原图算：

$$L = \left\| \mathbf{x} - D\big(E(\tilde{\mathbf{x}})\big) \right\|^2, \tag{4.2}$$

人话：E 是编码器、D 是解码器；输入是带噪声的 $\tilde{\mathbf{x}}$（读“x 波浪”），但评分标准是“离干净的原版 $\mathbf{x}$ 有多近”——只抄写不得分，必须把噪声擦掉。

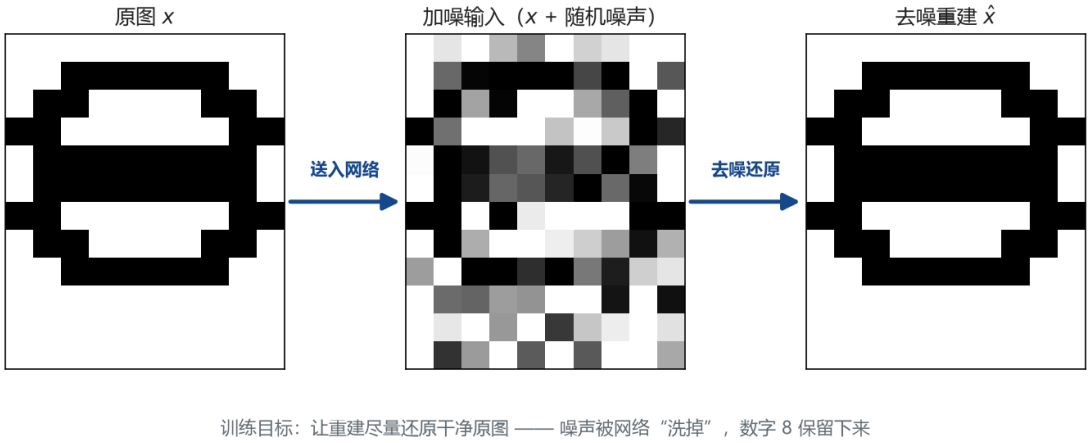


图 4.4　去噪自编码器三联图：干净原图（左）加噪变脏（中）送入网络，重建（右）不仅恢复笔画还顺手抹掉了噪声（以手写数字 8 为例）

为什么这样学到的特征更鲁棒？人话：如果输入带噪声而目标干净，网络就不可能靠“原样抄写”得高分——抄捷径的路被堵死了。它必须学到“这一笔大概是 8 的那个圆圈”级别的**结构性知识**，才能把噪声擦掉而不擦伤笔画。**强迫理解，不许复述**。这套“弄脏再还原”的自出题套路，是本讲最重要的思想遗产，2026 年它无处不在：BERT 遮词猜词是文本版去噪，MAE 抠图块补全是图像版去噪，第 12 讲扩散模型的“从纯噪声逐步去噪出图”是它的直系后代——大模型预训练的题目，多数是它的变体。

4.5　变体一分钟与“AE 的当代后代”

在上家族合影之前，先把教材里的两位正式成员请上台。**正则自编码器**（regularized autoencoder）：在重建损失之外给权重加 L1 / L2 惩罚（就是第 3 讲那句“给过大的权重加税”的同款手艺，也叫权重衰减）——相当于**给隐层加纪律**：不许死记训练集的细枝末节，想重建得像，就只能靠学到的结构。

稀疏自编码器：让大多数神经元保持沉默

稀疏 AE 给隐层加的不是“窄”，而是“闲”。**稀疏性约束**：给每个隐神经元定一条纪律——**平均激活度** ρ 必须很小（比如 0.05：一百次输入里平均只醒五次）；训练时统计每个隐单元的实际平均激活度 $\hat{\rho}$，谁偏离了目标 ρ，就用 **KL 散度**（KL divergence，衡量两个概率分布差多远的量，偏差越大罚得越重）写进损失里罚它。罚项把整个隐层压得**大部分时刻沉默**。人话：强迫网络用更少的活跃神经元表达同样的内容，逼出真正有用的特征——像整顿一支人浮于事的团队，每个人必须练出独当一面的本事。教材里“开关式特征、可解释性好”说的就是它：一个只在看到竖线时才亮的神经元，本身就是一条可读的规则。

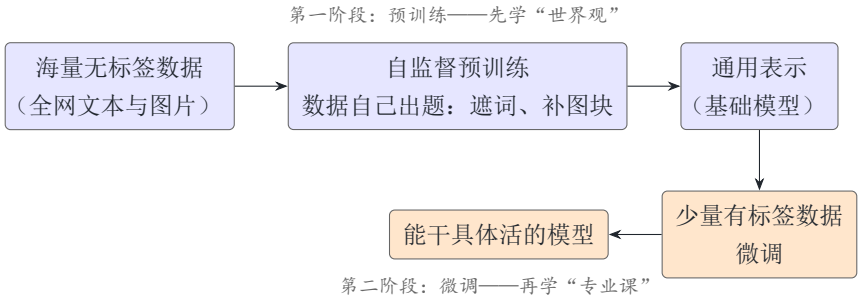
自编码器家族枝繁叶茂，本讲一分钟认全主要成员：

表 4.2　自编码器家族速览：各自加了什么、图什么

成员	加了什么	一句话用途
稀疏自编码器	隐层加“稀疏税”：多数隐单元须为 0	学出“开关式”特征，可解释性更好
变分自编码器（VAE）	隐码改成从一个分布采样，并对分布形状加正则	隐空间连续平滑，能采样生成新样本——生成模型的起点（第 12 讲）
卷积自编码器	编码、解码器换成卷积层	图像任务标配，参数更省、更懂像素的邻居关系
当代后代（BERT/MAE 等）	“遮住/弄脏，再还原”的自监督出题	大模型表示学习的雏形

表中再补两句人话。稀疏 AE 的用途刚才盒子里已经讲了，这里只打个比方：让每个隐单元专职检测一种特定模式——像给一百个员工各定一个岗位，不许扎堆。VAE：让隐码不再是一个点，而是一个“云团”（概率分布），并对云团的位置形状提要求，于是隐空间处处连续、随便取一点解码都是合理样本——自编码器由此从“压缩”跨进“生成”，第 12 讲还会遇到它的思想。

对 2026 年最重要的是最后一行。把“自出题、找结构”放大到极限，就是当代 AI 的生产流水线：**预训练**（pre-training）——先在海量无标签数据上做自监督训练（猜被遮的词、补被抠的图块），学到通用表示；**微调**（fine-tuning）——再用少量有标签数据，把这套表示适配到具体任务。两段式如下图：



一句话总结：**先无标签学“世界观”，再少量标签学“专业课”**。2026 年几乎所有语言与视觉模型都按此流水线生产。你们水下项目的教师模型 FoundationStereo 就是立体视觉界的基础模型：先在千万级立体数据上学出通用的立体匹配表示，下游小数据微调即用——你们做的“教师打伪标签、学生学着像教师”，本质上是把它的隐空间表示蒸给学生（第二篇第 二 章）。

动手时间（纸上）：逐层贪婪预训练——Hinton 2006 的破局操作

2006 年的死局是：网络一深，反向传播的学习信号到不了底层（第 2 讲饱和区的放大版），深层网络怎么训都输给浅层。辛顿团队的破局操作朴素得像搬砖——**一次只训一层**，学名**逐层贪婪预训练**（greedy layer-wise pre-training）：

1. 第一轮：只把输入接到第一个隐层，把它当**一台单层自编码器**训练到能重建输入，然后**固定**这层权重；
2. 第二轮：把第一层的输出当作新输入，训第二个隐层的自编码器，训完同样固定；
3. 如此逐层向上（堆叠 AE、堆叠稀疏 AE、堆叠去噪 AE，套路全同），最后把整座网络按第 3 讲的全套手艺整体**微调**一遍收工。

拿笔在纸上画一遍 784→196→49 的两层 AE：标出两轮训练里哪些权重在动、哪些被冻结——画完你就懂了“贪婪”二字：每层只顾自己重建得最好，从不保证对整座网络最优，所以叫逐层**贪婪**。人话总结：像盖楼，一层验收封顶再盖下一层，每层的起点都是上一层学到的有用表示而非“乱码”——**这是 2006 年深度学习复兴的钥匙操作**，“深度学习”这个名字也诞生于这几篇论文。学术争议也要如实交代：后来人们发现，有了 ReLU、批归一化和好的初始化（第 3 讲），逐层预训练在多数任务上**不再必要**——但把领域从寒冬里捞出来的历史转折点，非它莫属：没有 2006，就没有 2012。

前沿速递：表示学习——深度学习的灵魂

深度学习为什么叫“深”？表面看是层数，实质是**表示学习**（representation learning）：每一层把上一层的表示再抽象一步，边缘拼成纹理、纹理拼成部件、部件拼成物体（下一讲你会看到 CNN 的同款层级）。自编码器是这件事的最小完整闭环。2024–2026 年被称为“基础模型”（foundation model）时代：先在无标签海量数据上学通用表示，再小数据微调甚至零样本直接用——GPT-4、一键分割一切的 SAM、图文对齐的 CLIP 都是；视觉这边，你们项目里的 FoundationStereo（立体匹配）与 Depth Anything（单目深度）也是。两块里程碑各记一句话：simCLR（2020，对比学习——同一张图两次随机增强应“挨着”，不同图应“远离”，无标签学出好表示）；MAE（2021，遮住 75% 的图块让模型补全，把“去噪出题”推到极限）。最后送你一个检验任何模型“有没有表示学习的灵魂”的问题：**把标签全部摘掉，它还能从数据里学到什么？**——学不到东西的，只是台背题机器。

项目连线：压缩，但不丢关键信息

六目拼接：九个参数头里的 `descriptor_softening` 头，把 32×32 的图像块压成 256 维“指纹”（descriptor，描述子）——标准的自编码器式压缩：上千个像素值被压进 256 维，目的不是省存储，而是**让“像”的图像块在指纹空间里真的挨得近**，两路相机的图像块靠指纹比对完成匹配，比逐像素硬比稳得多（第 一 章）。**水下双目**：浑水弱光下，人工设计的匹配特征频频失效，项目改用深度基础模型的**隐空间表示**替代手工特征——教师打伪标签、学生模仿教师，本质是“把教师的表示蒸给学生”（第 二 章）。**数据基座**：把 1001 页手册压成结构化 JSON，同样是“压缩但不丢关键信息”：摘要式压缩、瓶颈式取舍，而“什么算关键”由问数场景定义——能支撑回答问题的字段一个不能丢，装饰性排版全可扔（第 三 章）。三条线共享同一句话：压缩的境界不在压得多小，而在丢得对不对。

4.6　从这一讲到 2026

自编码器“压缩再重建”的思想，在 2026 年无处不在，三条支线都还认得本讲的沙漏。第一条：**视频生成**。阿里 Wan、Sora 这类视频大模型，第一步都是先用 VAE 把整段视频压进低维的**时空潜空间**（时间和空间一起压：一段视频变成一小块三维“隐码”），再在潜空间里做扩散去噪——VAE 成了视频生成的“压缩器”，扩散模型的真身第 12 讲见。第二条：**自监督视觉骨干**。Meta 的 DINOv3 不用一张人工标签，在 17 亿张图片上自监督训练出 70 亿参数的通用视觉骨干（ViT，一种把图像切块当“字”来读的注意力网络，第 11 讲的主角），特征拿来就能分类、分割、检索——这是本讲前沿盒子那句“把标签全部摘掉，它还能学到什么”的满分答卷。第三条走得最远：**世界模型**。JEPA 一系把“在像素里重建”升级为“在嵌入空间里预测”——不逼模型补齐每个像素，只要求它预测“这幅

画面的抽象表示该长什么样”，V-JEPA 2 用超百万小时互联网视频学出对世界运动规律的直觉，只在约 62 小时机器人视频上后训练，就能零样本操控陌生实验室的机械臂。从 AE 到 VAE 到 MAE 再到扩散与 JEPA，主线一以贯之：**把世界压进好的向量**。

2026 现场：关键事实

- 阿里 Wan 团队 2025 年 3 月开源整套视频生成大模型（1.3B / 14B 两档，1.3B 版仅需约 8.19 GB 显存），采用扩散 Transformer 范式：自研 VAE 先把视频压进低维时空潜空间再行去噪——本讲的“编码器—潜空间—解码器”就是它的生产形态（来源：Team Wan，arXiv 2025，编号 2503.20314）。
- OpenAI 于 2025 年 9 月 30 日发布 Sora 2（旗舰级视频 + 音频生成模型，支持同步声音与更真实的物理表现）；Sora 类模型同样在 VAE 压缩的潜空间中做扩散生成（来源：OpenAI 官方博客“Sora 2 is here”，2025）。
- Meta 2025 年 8 月发布 DINOv3：纯自监督、零人工标注，在 17 亿张精选图像上训出最高 70 亿参数的 ViT 视觉骨干，并提出 Gram anchoring 技术防止长训练中的特征退化，特征可即插即用于分类、分割、检索（来源：Meta AI Blog 及 arXiv 2508.10104，2025）。
- Meta 的 V-JEPA 2（2025 年 6 月）在超 100 万小时互联网视频上自监督预训练；其动作条件版 V-JEPA 2-AC 仅用约 62 小时无标注机器人视频做后训练，即可在两个陌生实验室的 Franka 机械臂上零样本完成抓取与放置（来源：Assran, Bardes, LeCun et al.，arXiv 2025，编号 2506.09985）。

一句话收束：**从 1991 年的自编码器到 2025 年能想象世界的模型，演化逻辑从未变过：先学会把世界压进潜空间，才能在潜空间里想象世界。**

本讲课后题，是亲手把一台自编码器从压缩训到可视化——顺带把第 3 讲的训练手艺（切分、监控、早停）全用上。五要素如下。

第 4 讲课后编程题（用 AI 实现）

题目：指挥 AI 辅助，在 sklearn 自带的手写数字集 `load_digits`（1797 张 8×8 灰度图，共 64 维）上训练一个自编码器：结构 $64 \to 32 \to 16 \to 32 \to 64$（隐码 16 维，约为原维度的四分之一），训练完后交付两样可视化：(1) “原图 vs 重建”对比图——上下两排各 8 个数字，用 matplotlib 画成图像网格；(2) 把全部样本的 16 维隐码用 TSNE 拍扁到二维，画散点图并按 0–9 数字类别着色（十种颜色），看同类是否聚簇。

交付物：(1) 代码文件 `digits_ae.py`；(2) 两张 png（重建对比图、TSNE 散点图）；(3) 50 字以上的结论：16 维隐码够不够装下数字的“本质”？哪些数字的簇容易黏在一起、为什么；(4) 一段 50 字以上的“我改了 AI 代码里的什么、为什么”。

提示词建议（可直接抄）：“我是零基础学生，正在学自编码器。请用 Python 写一个自编码器：(1) 数据用 sklearn 的 load_digits，按 8:1:1 切分训练/验证/测试集，特征除以 16 归一化到 0–1；(2) 网络结构 64→32→16→32→64，隐码 16 维，ReLU 激活（输出层用 sigmoid），平方误差损失，小批量训练并监控验证损失做早停；(3) 训练完用 matplotlib 画‘原图 vs 重建’上下两排各 8 个数字的对比图，保存 png；(4) 用 sklearn 的 TSNE 把训练集全部隐码降到二维，画散点图，按数字类别 0–9 着色并带图例，保存 png；(5) 每行中文注释，标明哪一行是编码器、哪一行是解码器、哪一行取隐码。”

验收线：两张图都生成且能讲清——重建图能指出“丢了什么细节、保住了什么笔画”，TSNE 图上至少能辨认出 2–3 个明显分开的簇，并解释为什么 4 和 9、3 和 8 这类数字的簇容易粘连；能用自己的话说出瓶颈的作用；至少一处自己的修改（隐码改 8 维或 32 维对比、给输入加噪声改成去噪 AE、换激活函数都算）。**代码是 AI 写的没关系，讲不清楚就是没做。**

加分项（任选其一）：(1) 把隐码直接压到 2 维重新训练，对比“AE 二维隐码”与“AE 十六维隐码 + TSNE”两张散点图的形状差异，想想为什么不一样；(2) 训练时给输入加高斯噪声做成去噪自编码器，对比重建图的变化；(3) 拿训练好的隐码接一个 k 近邻分类器（digits 自带标签），报告“无监督特征 + 有监督分类”的准确率，与直接用原始 64 维像素对比。

课后阅读：教材《深度学习模型与算法基础》第 35–48 页，与本讲内容对应；重点看书中自编码器结构图与隐层的说法，与本讲沙漏图对照着读。

无监督学习（unsupervised learning）

不用人工标签、从数据自身找结构的学习方式；自编码器、聚类、降维都属于这一路。

自监督学习（self-supervised learning）

把数据自己加工成“题目 + 答案”（遮词、补图块）再学习；2026 年大模型预训练的标准形态。

自编码器（autoencoder）

编码器—瓶颈—解码器的沙漏网络，以“重建输入”为目标训练，学到压缩表示。

编码器 / 解码器（encoder / decoder）

沙漏的左半与右半：一个负责压缩、一个负责还原。

瓶颈（bottleneck）

沙漏最窄处的隐层；故意收窄的“信息税”，逼网络分辨本质与边角料。

隐码与隐空间（latent code / latent space）

瓶颈处的向量，及其所在的高维空间；相似输入在隐空间里彼此靠近，插值即渐变。

重建损失（reconstruction loss）

输出与输入的差距，常用平方误差；自编码器的“标准答案”就是输入自己。

去噪自编码器（denoising autoencoder）

输入加噪、目标干净的变体；堵死抄捷径的路，逼出鲁棒的结构性知识。

变分自编码器（VAE）

隐码改为分布采样并加正则的变体，隐空间连续、能采样生成新样本，是生成模型的起点之一。

表示学习（representation learning）

让模型自动学出数据的本质特征；深度学习之“深”的实质，基础模型时代的核心资产。

第 5 讲　卷积神经网络原理与前向计算

1968年，哈佛大学的休贝尔（Hubel）与维泽尔（Wiesel）把微电极插进麻醉猫的视皮层，向幕布投出各种线条，记录单个神经细胞的放电。他们发现了一个惊人的分工：**有些细胞只对特定方向的边缘放电**——竖直边缘经过时猛烈放电，线条一转就沉默；这类“边缘侦探”密密麻麻铺满视皮层的第一站。这项工作获得 1981 年诺贝尔生理学或医学奖，更重要的是它给出了一张工程蓝图：**视觉是从“找边缘”开始的，再一层层向上拼装出“认东西”的能力**。

1989 年，贝尔实验室的 LeCun 对着这张蓝图想：生物如此，机器为何不照抄？他把手写数字图片直接喂进一种带“局部小窗口 + 模板全图复用”机制的网络——**卷积神经网络**（Convolutional Neural Network，CNN），第一次让机器可靠地读出成千上万张支票与邮编上的手写数字。生物学出蓝图，反向传播来施工，这一讲讲它的原理与前向计算；怎么训练它，是下一讲（第 6 讲）的事。

本讲地图

讲什么：全连接看图为何参数爆炸；卷积 = 滑动窗口做点积；三大设计性质（局部连接、权值共享、平移等变）；padding 与 stride 及输出尺寸公式；池化；特征图与层级特征；完整前向流水线与通道。**为什么值得学**：三个项目里的视觉模型全是 CNN 或其直系亲属；实验三（11 月 23 日）训练的就是卷积网络。**学完你能**：手算一个 3×3 卷积；算出任意卷积层的输出尺寸；说清参数到底省在哪；看懂 CNN 结构图与特征图。

5.1　全连接看图：一场参数灾难

先弄清为什么不能把图片直接塞进第 2 讲的 MLP。全连接层要求输入是一维向量，所以图片得先“拉直”。一张 28×28 的手写数字拉直成 784 维向量，接 1000 个隐藏神经元——约 78 万个权重，还没开始认字就已经不少。那真实世界的照片呢？算给读者看：

表 5.1　全连接看图的参数账：图片越大越离谱

输入图	拉直后的维度	全连接到 1000 个隐单元	评论
手写数字 28×28（灰度）	784	约 78 万	勉强能接受
ImageNet 常用 224×224（RGB）	约 15 万	约 1.5 亿	开始失控
手机照片 1000×1000（RGB）	300 万	30 亿（float32 约 12 GB）	灾难

表格第三行值得掰开现场算一遍。一张 100 万像素的照片（约 1000×1000，手机厂商喊了十年的“百万像素”就是这个数），假设只接 1000 个隐藏神经元，光这一层的参数就是 $10^6 \times 10^3 = 10^9$——**十亿个**。什么体感？按 float32 存就是 4 GB，一张照片还没开始认，先吃掉半部手机的运行内存；训练时还得为每个参数另备一份梯度（第 3 讲），内存再翻一倍。而这，只是一张普通手机照、一层的账。

参数多还只是账面问题，更要命的是**拉直把位置信息丢了**。784 个数排成一队之后，“谁挨着谁”就没了：图里的笔画往右挪一个像素，784 维向量几乎每个分量都变了——人眼看来还是同一个数字，网络看来却是一行从没见过的全新输入。更别提第一层每个权重只认识“队伍里的第几个数”，根本不知道第 17 号和第 18 号在图上是左右邻居。空间结构，全连接层一出生就扔了。

于是只剩一条笨路：把一切都背下来。猫挪半步、镜头拉远、头歪 15 度，像素排布立刻全变，逐位对齐的模板马上失效；想靠蛮力兜住，就得为**每一种位置、每一种大小、每一种角度**各备一套模板——组合爆炸，永无出头之日。可视觉识别的基本要求恰恰相反，叫**形变不变性**：怎么挪、怎么缩、怎么转，都得认得是猫。

账算到这儿，病根齐了三条：**参数太多**（每个位置一套班子），**经验不共享**（左上角学出的“猫耳侦探”，到了右上角完全作废，得从头再学一遍——可猫耳不管长在哪个角落，长相是一样的呀），**位置与形变脆弱**（模板焊死在坐标和尺度上，物体一动就失明）。CNN 的全部设计，就是对症下药：**只看小块**（局部连接）、**模板复用**（权值共享），外加一剂**只留最强响应**（池化，5.4 节）兜住小幅位移。

5.2　卷积 = 滑动窗口做点积

卷积核（kernel，也叫 filter，滤波器）是一块小模板，比如 3×3 的九个数。用法看下图：把它按在图片左上角，模板盖住的几个像素与模板上对应的几个数**对应相乘再求和**，得到输出图左上角的一个数；然后向右滑一格，重复；一行滑完换下一行，直到扫遍全图。输出图叫**特征图**（feature map）——它标记的是“这块模板在图里最像的地点”。

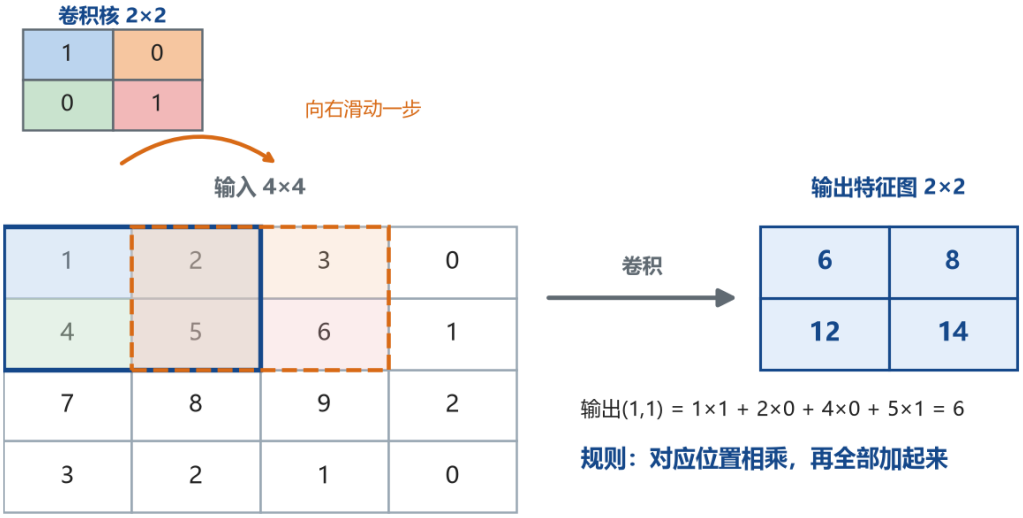


图 5.1　卷积的滑动窗口计算：2×2 的核在 4×4 输入上每次盖住 4 个像素，对应相乘再求和得到输出一个数；图中演示向右滑动两步

慢着——“对应相乘再求和”？这不就是开课导引里的**点积**（第 0 讲）、感知器里的**加权和**（第 1 讲）吗？完全正确，一分不差。**卷积核就是一个小感知器：只用九个权重，对图片的一小块打“相似度分”**——模板与那块图像越长越像，得分越高。卷积没有任何新数学，它就是点积的滑动应用；当年 LeCun 的突破，与其说是发明了新运算，不如说是发现了“把感知器按在图上滑”这件朴素的事恰好就是视觉需要的。

这个设计自带三条性质，而且条条都能在 1968 年的猫实验里找到原型：

- **局部连接**（local connectivity）：每个输出只依赖输入的一小块，这一小块叫**感受野**（receptive field）——视皮层的“边缘侦探”也只盯一小块视野。
- **权值共享**（weight sharing）：同一块模板扫遍全图，参数量与图像大小无关——猫耳模板在左上角、右上角通用，经验全图共享。

- **平移等变**（translation equivariance）：图里的物体挪个位置，特征图上的响应也挪同样的位置——猫在左在右，猫耳模板都照样点亮，只是亮的格子不同。

知识补齐：数学课的“卷积”与工程口的“卷积”不是一回事

严格数学定义的卷积，要把核先旋转 180 度再滑动（“翻面再扫”）；深度学习框架里叫 conv 的函数，实际做的是不翻转的**互相关**（cross-correlation）。为什么敢偷懒？因为核是**学出来的**——网络想要什么样的模板，它自己会学成需要的样子，翻不翻转根本不影响表达能力。名词之争留给数学课，工程口径记住一条即可：本书与主流框架一致，“卷积”都指不翻转的滑窗点积。

手机修图你天天在用卷积，只是不知道它叫这个名字。

动手时间：手机修图里的卷积核——手算一个角

打开手机修图 APP 的“锐化”“浮雕”“模糊”，它们就是三块现成的卷积核在图上滑动。锐化核：

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \qquad \text{模糊核：} \quad \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \text{浮雕核：} \quad \begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

取一张 4×4 的小图，像素随便填 0–9 的整数；把锐化核对准左上角的 3×3 区域，手算输出左上角那个数。自检：若该区域恰好是

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix},$$

则输出 $=0\cdot 1+(-1)\cdot 2+0\cdot 3+(-1)\cdot 4+5\cdot 5+(-1)\cdot 6+0\cdot 7+(-1)\cdot 8+0\cdot 9=5$。想想看：这块核（中心 5、四邻 −1）为什么能锐化？区域平坦时输出约等于原值；中心比四邻亮时，输出被进一步拉高——**中心与四邻的差被放大，对比度拉开，就是“锐”**。模糊核则相反，把九个数平均成一团。课后题会让你把这三块核真的跑在一张照片上。

5.3　padding 与 stride：滑窗的两个旋钮

滑动窗口上有两个官方旋钮。第一个：**padding**（填充），在图片外圈补一圈 0。动机有二：其一，3×3 的核每扫一遍，输出图就比输入缩一圈（4×4 进去 2×2 出来），网络几十层叠下来，图会缩没了——补 0 圈可以让输出尺寸保持不变；其二，让边角像素也有机会被模板盖到，不至于边缘信息没资格进特征图。第二个：**stride**（步幅），每次滑几格：stride $=1$ 逐格滑；stride $=2$ 跳着滑，输出尺寸约减半，相当于自带一步压缩。

两个旋钮拧完，输出尺寸由一条公式管死（建议背下来，实验三搭网络时天天用）：

$$O=\left\lfloor \frac{N+2p-f}{s} \right\rfloor +1, \tag{5.1}$$

人话：N 是输入边长，f 是核边长，p 是每侧补 0 的圈数，s 是步幅，O 是输出边长；除不尽的余数直接丢弃（$\lfloor\cdot\rfloor$ 是向下取整）。

三个算例检验理解：其一，32×32 输入、5×5 核、$p=0$、$s=1$：$O=(32-5)/1+1=28$——这正是下一讲 LeNet-5 的第一步；其二，同上但 $p=2$：$O=(32+4-5)/1+1=32$——补了两圈 0，尺寸保住了；其三，224×224 输入、3×3 核、$p=1$、$s=2$：$O=\lfloor(224+2-3)/2\rfloor+1=112$——尺寸减半，是现代网络常用的“一边卷一边缩”配置。下图把前两种情况画在了一起：

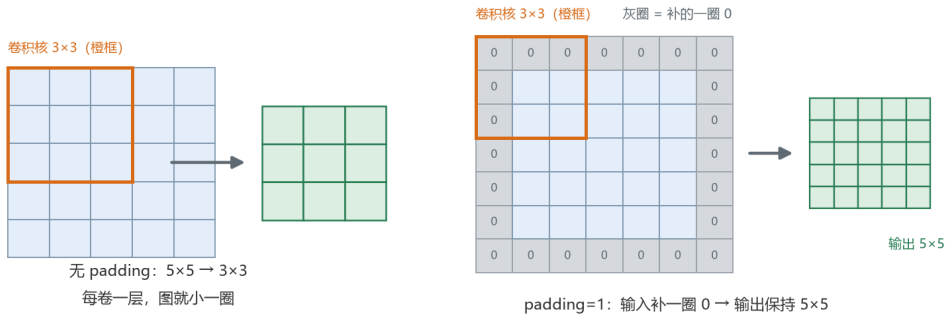


图 5.2　padding 与 stride 的效果：输入外圈补 0 后输出不再缩水；stride = 2 时窗口跳着滑，输出尺寸减半

5.4　池化：只留最强的响应

卷积之后常跟一步**池化**（pooling）：把特征图切成不重叠的小块，每块只留一个数。最常用的是 **max pooling**（最大池化）：2×2 区域取最大值，见下图。直觉是：模板在那一小块里“最像的地点”有一个就够了，具体压在左上角还是右下角不重要——**保留最强响应，丢弃精确位置**。顺带的收益：尺寸减半、计算减半，而且对小的位移不敏感（图案挪动一两个像素，最大值还是那一个）。

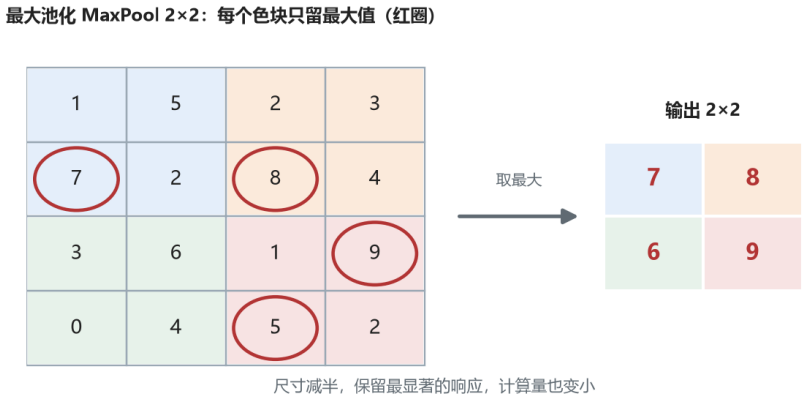


图 5.3　2×2 最大池化：4×4 特征图切成四块，每块取最大值得到 2×2 输出（彩色块标出每块的“冠军”）

另外两位池化成员一句话认识：**平均池化**（average pooling，取平均，手法更温和）；**全局平均池化**（global average pooling，GAP：整张特征图直接平均成一个数，现代分类网络常用它收尾，下一讲见）。注意池化自己**没有任何参数**——它是纯操作，不参与学习；网络要学的仍然只有卷积核和全连接的权重。

5.5　特征图与层级：从边缘到猫脸

一块模板扫一张图得一张特征图；多块模板并行扫，得到一叠特征图。层数加深，特征图的内容逐层升级：第一层检出边缘与斑点（正是 Hubel–Wiesel 的“简单细胞”干的事），中层拼出纹理与部件（条纹、转角、眼睛），深层直接响应整个物体（脸、车轮）。先看“一块模板长什么样”：下图里同一条斜边的输入，经过三个不同的核，得到三张“体检报告”——只在边缘经过的位置亮起，三张报告各亮各的。

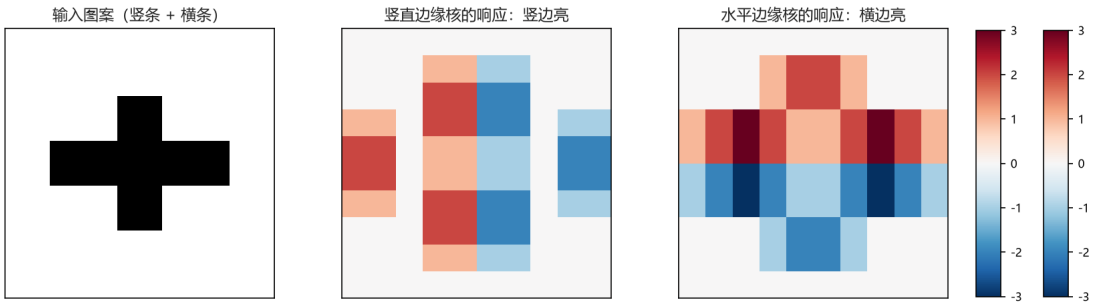


图 5.4　一条斜边（左）经过三个不同的卷积核得到三种响应图（中、右）：亮处即模板找到“同伙”的位置

关键问题是：每一层该放什么模板、层级该怎么长？答案：**没人设计，训练自己长出来的**——“认错东西的代价”经反向传播一层层压回去，网络自然把底层核逼成边缘侦探、把深层核逼成物体侦探。2014年 Zeiler 与 Fergus 的可视化工作第一次让人类看清了这套层级（下图，从左到右特征逐层抽象）：

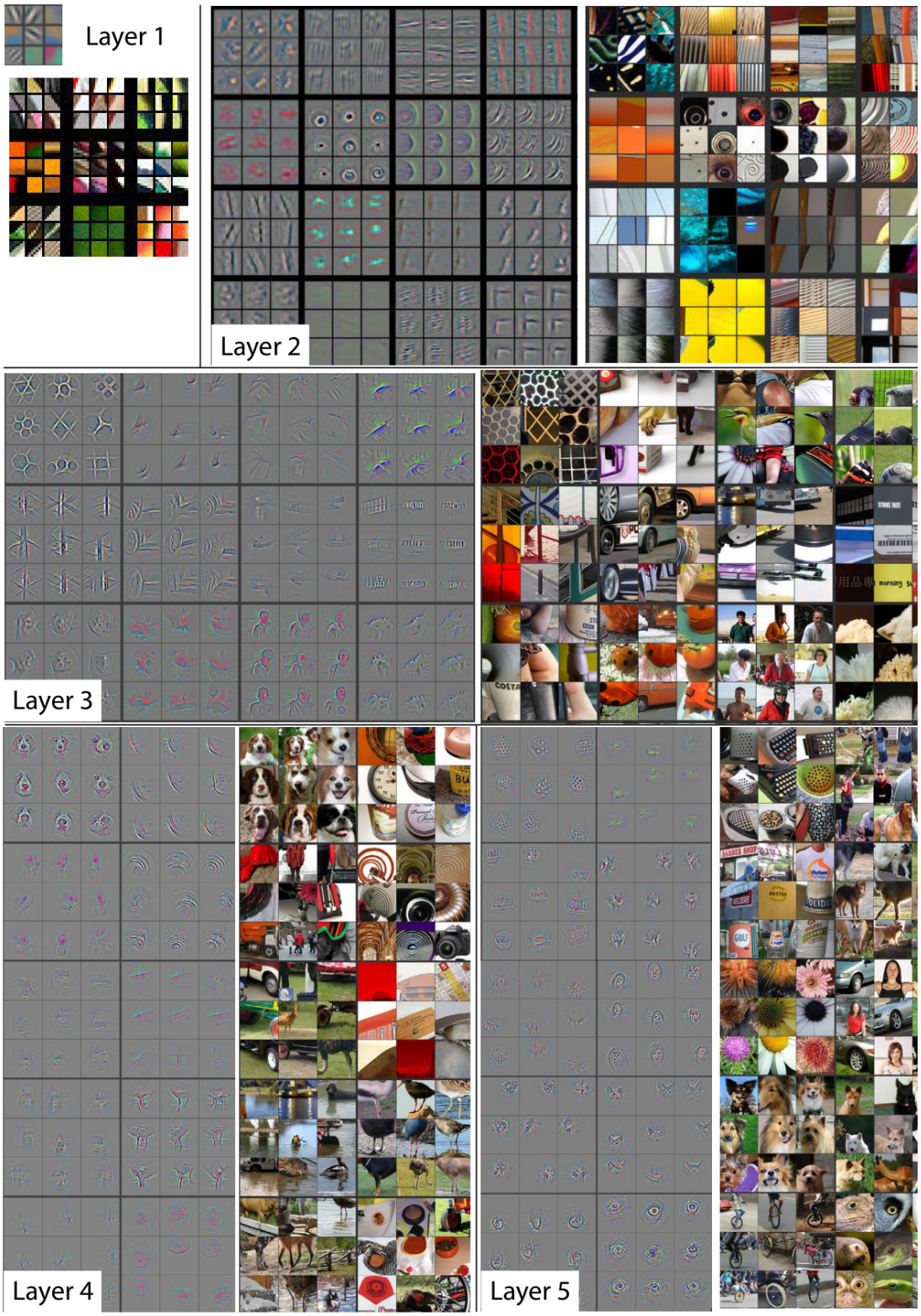


图 5.5　CNN 各层学到的特征层级可视化：底层是边缘与颜色块，中层拼出纹理与部件，高层已是物体部件与整体轮廓

图片来源：Zeiler & Fergus, "Visualizing and Understanding Convolutional Networks", 2014

这张图也回答了一个哲学问题：CNN 凭什么“像”视觉系统？因为它把 1968 年猫实验的发现——从边缘侦探到复杂细胞逐级组装——变成了可训练的数学结构。**生物学出蓝图，反向传播来施工，**

本讲开头那句话到这里落地。

5.6　完整前向流水线：从像素到类别得分

把本讲积木拼起来。一张图进入 CNN 的完整旅程见下图：图像先进“卷积—池化”单元（卷积提特征、池化缩尺寸），反复若干轮，特征越提越抽象、图越缩越小；然后**拉平**（flatten，把多张小特征图摊成一维向量），送进全连接层汇总判断；最后输出各类别得分。下一讲你会看到：得分经 softmax 变概率、与标签算交叉熵损失、再由反向传播逐层回传，把每个卷积核里的数修一点。

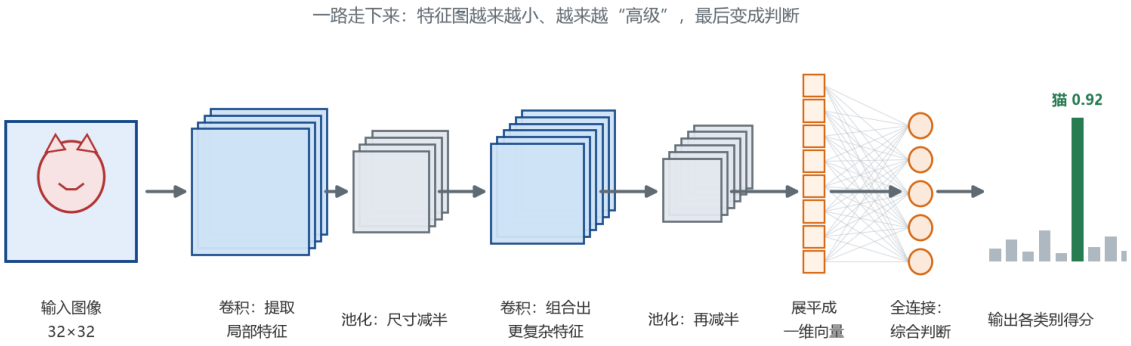


图 5.6　CNN 完整前向流水线：图像经“卷积—池化”反复提特征、缩尺寸，最后拉平送入全连接层输出类别得分

还有最后一个必懂名词：**通道**（channel）。其实前面已经埋好了——彩色图本身就是三叠灰度图（红、绿、蓝各一叠），这叫 3 通道输入；而每层卷积输出的一叠特征图（每个核一条），又是下一层的“多通道输入”。于是卷积核其实是个三维小方块：**边长 × 边长 × 通道数**——第一层的核是 $3 \times 3 \times 3$（吃彩色输入），深层的核可能是 $3 \times 3 \times 64$。通道就是“特征的平行宇宙”：每个通道一种模板视角，层层叠加越叠越厚。

生活化类比：张量——给数字们盖的高楼

向量是一排数（一维），矩阵是一张数表（二维），**张量**（tensor）是更高维的“数字楼”（三维及以上）：一张彩色图是 $3 \times H \times W$ 的三维张量——三层的楼，每层楼一个颜色通道；一批 32 张图打包就是四维张量 $32 \times 3 \times H \times W$。PyTorch 里一切数据都是张量；回忆第 3 讲的向量化——张量整体运算，正是“一行代码顶一百行循环”的舞台。

最后算总账，看参数到底省了几个数量级。同样处理 1000×1000 的 RGB 照片、提取 100 种模板：

表 5.2　参数总账：全连接 vs 卷积（同样 100 种模板的活儿）

方案	参数量	备注
全连接到 1000 个隐单元	约 30 亿	每个权重只服务一个位置，位置一动就作废
卷积：100 个 $3 \times 3 \times 3$ 的核	$100 \times 27 + 100 = 2800$	每块模板全图通用，参数与图像大小无关

30 亿对 2800——**百万倍的差距**，而且图像越大，省得越多。这就是 CNN 能把整个视觉世界装进一张显卡的记账原理。

梯度怎么流过卷积与池化？（人话版）

下一讲的主角反向传播，要从损失出发把“该怪谁”的梯度逐层传回去修参数。它在卷积和池化上怎么走？给四条直觉就够用，完整推导超出本讲义——思路就是链式法则加这四条。① **max 池化：梯度只认最大值**。前向时谁最大，反向时这股梯度就全额判给谁，同块其他位置一律为 0——像个路由器：前向顺手记住“冠军住在几号房”，反向只敲冠军的门，其余位置收到的信号是“与你无关”。②**平均池化：梯度均分**。前向把四个数平均成一个数，反向就把上游来的 1 份梯度平分成 4 份发回去，人人有份，童叟无欺。③**卷积核的梯度：还是点积**。某个核该修多少？把输入特征图与传回来的误差图（上游梯度铺成的图）对应相乘再求和——与前向卷积同款算法，还是点积思想。误差图在哪儿响应大，说明那个位置对这个核的得分影响大，核就朝那儿修。④**“求导时的卷积”不翻转，辨析一句话**。求卷积核自己的梯度（更新参数用的那份）时，做的就是不翻转的滑窗点积——跟前向一模一样；“翻转的核”只出现在把误差继续传给上一层的路上，那是下一讲推导里的细节。给初学者的结论：更新参数的那份梯度，仍是熟悉的点积，不必被“翻转”二字吓住。

前沿速递：CNN 还是 ViT？2026 年的版图

2020 年底，Google 的 Vision Transformer（ViT）把第 11 讲的注意力机制搬来看图：把图切成 16×16 的小块当“单词”，用自注意力让每块直接看到全图——不滑窗、不假设“东西长在隔壁”。在海量数据上 ViT 打败了 CNN，一时间“CNN 过时论”甚嚣尘上。但 2024–2026 年的共识冷静得多：小数据、端侧、低算力场景，CNN 仍占优——卷积自带的“局部 + 共享”先验让它在小数据上省样本、在手机上跑得快；超大数据与多模态场景，ViT 更强。两条路线也在合流：ConvNeXt 把 CNN 调校到能与 ViT 掰手腕的精度，各大视觉基础模型则多以 ViT 为骨干。你们项目的 FoundationStereo 就是“ViT 骨干 + 卷积细节修正”的混合体——所以本讲和第 11 讲，一讲都不能翘。

项目连线：从“人写的模板”到“学出来的模板”

六目拼接：传统方案的 ORB 特征（第 一 章），本质是手工设计的卷积核加点积匹配——FAST 找角点近似一块固定的边缘核，BRIEF 描述子近似一套预先写死的比较规则，全是第 1 讲点积思想的手工版；而九个参数头里的 `feature_uncertainty` 与 descriptor 头，输入正是 32×32 的图像块、网络主体正是几层卷积——把人写的模板升级成学出来的模板，是第 7 讲 CNN 三十年演化的微缩版在你组里上演。**水下双目**：立体匹配的底层逻辑，就是滑动窗口在左右两幅图之间找“长得一样”的地方——逐块打相似度分构成代价体（cost volume），而打分器本身就是卷积网络；FoundationStereo 内部同样在滑窗比对，只是模板全部预训练而来（第 二 章）。**数据基座**：B 类表格恢复里“扫描件图片与字段标签对上号”的环节用的也是视觉模型——小数据表格场景恰恰是本讲说的“CNN 仍占优”的地盘，你选型时的判断力就从这一讲来（第 三 章）。

本讲课后题，是把卷积核从手机修图 APP 里“请”出来，亲手滑一遍——顺便体感一下为什么工程实现绝不许用 Python 双重循环。五要素如下。

第 5 讲课后编程题（用 AI 实现）

题目：指挥 AI 辅助，写一个“卷积核体验器”：读入任意一张图片（自己手机拍的即可，先转灰度），对它分别应用三块 3×3 的核——锐化（中心 5、四邻 -1）、模糊（九个 1/9）、边缘检测（Laplace 核：中心 -4、上下左右为 1）——**滑动窗口必须自己写**：双重 for 循环逐位置“对应相乘再求和”，不许调用 `cv2.filter2D`、`scipy.signal.convolve2d`、`torch.nn.Conv2d` 这类现成卷积 API 来做这三个核。用 matplotlib 把“原图 + 三张结果”画成 2×2 四宫格，保存为 png。

交付物：(1) 代码文件 `conv_toy.py`；(2) 四宫格 png 一张；(3) 50 字以上的说明：三块核各把图片改成了什么样、为什么（用“点积 / 相似度”的语言说）；(4) 一段 50 字以上的“我改了 AI 代码里的什么、为什么”。

提示词建议（可直接抄）：“我是零基础学生，正在学卷积。请用 Python + numpy 写一个卷积核体验器：(1) 读入一张本地图片并转灰度（没有现成图就用 skimage 的示例图）；(2) 手写双重循环实现 3×3 卷积滑动窗口（对应相乘再求和），不许用 cv2.filter2D 等现成卷积函数；(3) 分别应用锐化核 [[0,-1,0],[-1,5,-1],[0,-1,0]]、模糊核（九个 1/9）、边缘检测 Laplace 核 [[0,1,0],[1,-4,1],[0,1,0]]；(4) 结果裁剪到 0–255 后，把原图与三张结果画成 2×2 四宫格，保存 png；(5) 先在一小块 4×4 数据上打印中间结果，帮我核对滑动计算对不对；(6) 每行中文注释，并用一句话回答：为什么循环版对大图很慢？工程上该怎么加速？”

验收线：四宫格生成且三张结果肉眼可辨（锐化更“脆”、模糊更“糊”、边缘图上轮廓发亮）；能对着图讲清“某个亮点 = 模板与该处图像块的点积得分高”；能当场回答两问——“3×3 核滑 1000×1000 的图要点多少次积”（约 100 万个位置、每个位置一次 9 项的点积）与“为什么工程实现不用 Python 循环”（第 3 讲向量化）；至少一处自己的修改（换 Sobel 核、给彩色图逐通道做、补 0 圈保尺寸都算）。**代码是 AI 写的没关系，讲不清楚就是没做。**

加分项（任选其一）：(1) 给边缘检测加上一圈 0 的 padding 重跑，对比四角的变化（呼应 5.3 节）；(2) 把 stride 改成 2 重跑，验证输出尺寸公式；(3) 把三块核串成流水线（先模糊再边缘），模拟“上一层的输出当下层的输入”，看看边缘图发生了什么。

课后阅读：教材《深度学习模型与算法基础》第 50–70 页，与本讲内容对应；看书上的卷积图示时，用本讲的“滑动点积”语言把每张图自己讲一遍。

5.7　从这一讲到 2026

本讲学的是卷积的“语法”。2026 年的生产现场里，这门语法被写在哪些地方？四个事实，比任何论战都更有说服力。

事实一：实时与边缘检测的工业默认，仍是卷积。2026 年 1 月正式发布的 YOLO26 依旧以卷积架构为主干，用“免 NMS 的一对一端到端检测头”把后处理环节直接取消，最小的 YOLO26n 在 CPU 上的 ONNX 推理较上一代最高快 43%，五档模型在 COCO 上做到 40.9–57.5 mAP——车间质检、无人机、车载视觉这些“快字当头”的场景，卷积仍是默认答案。

事实二：医疗现场的分工表。2024 年发表于 Journal of Medical Systems 的系统综述（汇总 36 项研究）给出冷静结论：ViT 要靠大规模预训练才见优势；在小数据、低分辨率的场景里，DenseNet-121 这类经典 CNN 依然能打平。“大模型用 ViT、小数据用 CNN”的分工，在医院里同样成立。

事实三：卷积从主角退成了关键零件。Stable Diffusion 3（2024 年 3 月）把去噪主力从卷积 U-Net 换成了 Transformer 家族的 MMDiT，但把像素压进潜空间、生成完再解压回来的 VAE 编解码器**仍是卷积网络**——你用文生图得到的每一张图，进出潜空间都得过一遍卷积。这位“压缩器”的真面目，

第 12 讲见。

事实四：卷积与注意力已经住进同一栋楼。MobileNetV4（ECCV 2024）把标准倒残差块、ConvNeXt块、前馈层与注意力统一进同一个搜索空间，让算法按目标芯片自动挑组合；它自带的 Mobile MQA 注意力更是专为移动加速器设计——早已不是“卷积还是注意力”的二选一，而是“每一层放哪个”的填空题。

所以 2026 年的正确提问不再是“卷积还是 Transformer”，而是**“卷积应该放在哪一层”**。学完本讲，再修完第 11 讲的注意力，你就同时握有回答这个问题的两支笔。

卷积 / 卷积核（convolution / kernel）

小模板在图上滑动、逐块做点积的运算；卷积核是要学习的参数。工程实现实为互相关，不翻转。

特征图（feature map）

一块核扫完整张图得到的“相似度地图”，亮处即模板命中之处。

局部连接与感受野（local connectivity / receptive field）

每个输出只依赖输入的一小块；那一小块叫感受野。

权值共享（weight sharing）

同一块模板扫遍全图，参数量与图像大小无关，经验全图复用。

平移等变（translation equivariance）

输入平移，特征图上的响应同步平移；物体在哪，响应就挪到哪。

padding（填充）

输入外圈补 0：既保住输出尺寸，又让边缘像素有机会被模板盖到。

stride（步幅）

滑窗每次跳几格；stride $= 2$ 输出约减半。输出尺寸公式：$O = \lfloor (N + 2p - f)/s \rfloor + 1$。

池化（pooling）

特征图切块留一个数（常用 2×2 取最大），压缩尺寸并换取位移不敏感性；自身没有参数。

通道（channel）

特征的平行宇宙：彩色输入 3 通道，每层卷积输出多通道；卷积核因此是三维小方块。

张量（tensor）

向量与矩阵的高维推广；一张 RGB 图是三维张量，一批图是四维张量，深度学习框架里的数据通通是张量。

第 6 讲　卷积神经网络的训练与计算模式

1998年前后，如果你在美国把一张支票存进银行，很可能它已经被一台机器“读过”了——不是柜员，而是一个卷积神经网络。贝尔实验室的燕乐纯（Yann LeCun）团队在 1989 年就教会了神经网络认手写数字，随后这套代号 LeNet 的系统一路迭代，到 20 世纪 90 年代末，**高峰期每天替全美银行读取 10%–20% 的支票**。这是卷积神经网络第一次大规模商用——比“深度学习”这个词火起来早了十几年。上一讲你已经看懂了卷积和池化怎么算（第 5 讲），也就是知道了这台机器**怎么往前走**；本讲要回答的是后半截问题：它**怎么学会**——训练循环怎么转、数据从哪来、Batch Normalization 为什么几乎无处不在，以及除了准确率，我们还能用什么指标评估它。学完本讲，你就可以亲手把一个 CNN“养大”。

本讲地图

讲什么：MNIST与 CIFAR-10 两个经典数据集、LeNet-5 逐层精读、数据增强、训练循环的五个动作与 PyTorch 代码、批归一化（BN）、混淆矩阵与精确率/召回率、过拟合对策清单。**为什么值得学**：三个项目的所有训练任务——九个参数头、FoundationStereo 微调、真假样本分类器——转的都是本讲这一个“循环”。**学完你能**：读懂并修改一个 PyTorch 训练脚本；解释 `model.train()` 与 `model.eval()` 的差别；用混淆矩阵判断一个分类器到底靠不靠谱。**课后阅读**：许庆阳《深度学习模型与算法基础》第 71–85 页。

6.1　练手房：MNIST 与 CIFAR-10

学开车不必先上高速，学训练也一样。深度学习为“第一次训练”准备了两个经典的“练手房”，小到你的笔记本几分钟就能跑完一次，又大到足以暴露训练的全部问题。

第一个是 **MNIST**（Modified National Institute of Standards and Technology）：7 万张 28×28 的灰度手写数字图片（0 到 9 共 10 类），其中 6 万张做训练、1 万张做测试。它诞生于 LeNet 团队之手——从美国人口普查局和高中生的原始表单中扫描、整理而来，1998 年起成为整个领域默认的入门考题。深度学习圈有句话：**手写数字识别是这个领域的 Hello World**——你以后每换一个新框架、新显卡，第一件事多半仍是在 MNIST 上跑一把。下图是它长什么样：每一张图就是 7 万分之一个样本，人类几乎张张能认，机器当年却要苦练许久。



图 6.1　MNIST 手写数字样本网格：每张为 28 × 28 的灰度图，同一个人写的“1”千姿百态，歪斜、粗细、断笔都可能是机器要克服的困难

图片来源：LeCun MNIST 数据库 / 维基百科 MNIST database 词条

第二个是 **CIFAR-10**：6 万张 32 × 32 的彩色小图，10 个类别（飞机、汽车、鸟、猫、鹿、狗、蛙、马、船、卡车），5 万训练、1 万测试。它是 MNIST 的“进阶版”——类别多了、彩色了、背景复杂了，同一条卷积路线照用不误；本篇第 7 讲和第 8 讲的课后题都会用到它。

扫盲：一张图片在机器眼里就是 784 个数

你眼里“一个手写的 8”，在机器眼里只是 28 × 28 = 784 个 0 到 255 之间的整数：背景接近 0（黑），笔画处接近 255（白）。所谓“训练一个分类器”，就是找一个函数：吃进这 784 个数，吐出 10 个类别各自的得分。这个视角会贯穿全书——第 1 讲的感知器吃 4 个数（鸢尾花的两长两宽），这里的 CNN 吃 784 个数，第 7 讲的 ImageNet 网络吃一百多万个（224 × 224 × 3）。**图像即矩阵**，是每一个初学者必须先建立的直觉。

6.2　LeNet-5 精读：第一栋“经典公寓”的户型图

把 LeNet-5 摊开在图纸上，就是下面这栋七层小楼。它不深也不宽，却奠定了此后二十六年所有图像网络的基本户型。

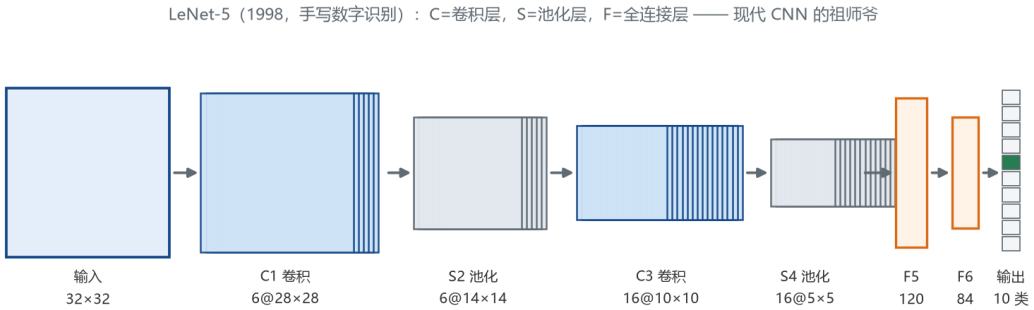


图 6.2　图题：LeNet-5 结构横向流程：32 × 32 输入经卷积（6 个 5 × 5 卷积核）、池化、卷积（16 个卷积核）、池化，再经全连接 120、84，输出 10 类得分

逐层数一遍它的“砖头”（下表同时给出每层参数量——请动手验算，公式在下面正文里）：

表 6.1　LeNet-5 逐层数字速查表（参数量为近似口径，总数约 6.1 万）

层	干什么	输出尺寸	参数量	一句话
输入	灰度图	32×32	0	原始支票局部照
C1	卷积，6 个 5×5 核	$6 \times 28 \times 28$	156	提取短线段等局部特征
S2	平均池化 2×2	$6 \times 14 \times 14$	12	压缩，抗小幅位移
C3	卷积，16 个 5×5 核	$16 \times 10 \times 10$	1516	把线段拼成弯钩、圆弧
S4	平均池化 2×2	$16 \times 5 \times 5$	32	再压缩
C5	全连接	120	48120	把 400 个数揉成整体判断
F6	全连接	84	10164	进一步提炼
输出	全连接	10	850	10 个数字各自的得分

参数量怎么算？以 C1 为例：6 个卷积核，每个核是 $5 \times 5 = 25$ 个权重再加 1 个偏置，共 $6 \times (25+1) = 156$ 个。请注意这个数字的分量：假如不用卷积、直接把 $32 \times 32 = 1024$ 个像素拉平接一层 100 个神经元的全连接，光这一层就要 $1024 \times 100 + 100$ 超过 10 万个参数——而参数一多，就需要更多数据、更多算力，还更容易过拟合（第 3 讲）。**卷积的本质优势在 1998 年就已经体现：用“局部连接 + 权重共享”花小钱办大事**。

再把两个“数值算例”做实。**其一，卷积核个数 = 输出特征图张数**：C1 有 6 个核，就输出 6 张 28×28 的特征图，每张对应一个模板的“相似度地图”——表里输出尺寸“$6 \times 28 \times 28$”开头的 6 就是这么来的；同理 C3 的 16 个核出 16 张图。看懂任何 CNN 结构图，先数卷积核个数。**其二，S2 平均池化为什么有 12 个参数？**平均池化本身是零参数的纯操作（第 5 讲），但 LeNet 给每个通道配了一对小旋钮：取完平均后先乘一个可学习的缩放系数、再加一个可学习的偏置，$6 \times 2 = 12$——相当于允许每张图“平均完再调调对比度和亮度”。

C3 这一层还藏着一个更精妙的手工设计：它有 16 个卷积核，按常规每个核都该看 S2 的全部 6 张特征图（$16 \times 6 = 96$ 条连接）；LeCun 却只保留了 60 条——**每张 C3 特征图只挑 S2 的几张做输入**，组合是精心挑选的（见下表）。省参数是一方面，更妙的是逼着 16 张特征图各有分工：前 6 张各看相邻 3 个通道，中间 6 张各看相邻 4 个，后 3 张故意“跳着挑”打破对称，最后 1 张六张全看。这种“手工艺接线”如今已被更通用的 1×1 卷积取代（第 7 讲），但“通道不必全连、可以省着连”的思想留了下来。

表 6.2　C3 的 16 张特征图各选 S2 的哪几张做输入（编号从 0 起，依 LeCun 1998 原表）：不是全连接，是精心挑选的组合

C3特征图	选用的 S2 通道	C3 特征图	选用的 S2 通道
0	0, 1, 2	8	2, 3, 4, 5
1	1, 2, 3	9	3, 4, 5, 0
2	2, 3, 4	10	4, 5, 0, 1
3	3, 4, 5	11	5, 0, 1, 2
4	4, 5, 0	12	0, 2, 3
5	5, 0, 1	13	1, 3, 4
6	0, 1, 2, 3	14	2, 4, 5
7	1, 2, 3, 4	15	全部 6 张

更重要的是户型本身。看图中从左到右的三段式：**卷积–池化堆叠两轮负责“提取特征”，全连接收尾负责“拍板分类”**。这个模式有个学名上的通俗版——特征提取器加分类头（backbone + head）。它强到什么程度？你将在第 7 讲看到，AlexNet、VGG 直到 ResNet 换了一茬又一茬，户型仍是它；

你项目里六目拼接的九个“参数头”，每个也都是“小特征提取 + 小分类头”的同款结构。1998 年定下的骨架，2026 年还在服役。

动手时间：纸笔算 LeNet-5

不看上表，自己算一遍（答案附后）。**(1)** C1 的输出尺寸：32×32 的图上滑 5×5 的核、步距 1、不加 padding，输出宽高各是多少？（提示：第 5 讲的公式 $(W-K+2P)/S+1$。）**(2)** C1 的参数量。**(3)** S2 池化后输出多大？**(4)** C3 在 14×14 上滑 5×5 的核，输出多大？**(5) 想一想：**若把 C1 的核从 5×5 换成 3×3 且不加 padding，输出变成多大？对网络“看到”的信息意味着什么？——答案：(1) 28；(2) 156；(3) $6\times14\times14$；(4) 10；(5) 30。核越小“看得越近”，要看得更远就得堆更多层——这正是第 7 讲 VGG 的出发点。

6.3　数据增强：白捡的“新数据”

MNIST 只有 6 万张训练图，够用吗？LeNet 时代的答案是不太够——网络记性好得很，几十个 epoch 之后它就开始**背答案**：训练集上几乎全对，没见过的字却认错，这正是第 3 讲讲过的过拟合。买新数据太贵，于是有了穷人的一百万：**数据增强**（data augmentation）——把已有的图随机变个样，标签却不变，等于白捡新数据。



图 6.3　图题：数据增强九宫格：中心为原图，四周是随机旋转、平移、翻转、加噪、亮度变化等增强结果——样子变了，标签仍然是同一个数字

为什么“变了样、标签不变”是成立的？因为**这些随机变换不触及类别的本质**：一个“8”略微歪一点、淡一点、抖一点，它还是 8；一张猫的照片左右镜像，它还是猫。增强没有创造新信息，但它逼着网络**见多识广**——不再依赖“训练集里的 8 都是端正的”这种投机取巧的捷径，而必须学会扭歪的 8、断笔的 8、墨淡的 8 的共性。用 PyTorch 实现只要一行：

```
from torchvision import transforms as T
# 训练时的变换：随机翻转 + 随机小角度旋转 + 转张量
train_tf = T.Compose([T.RandomHorizontalFlip(),
                      T.RandomRotation(10), T.ToTensor()])
# 测试时的变换：什么都不随机，原样评估
test_tf = T.Compose([T.ToTensor()])
```

注意两个细节。**第一**，随机性只属于训练：测试时必须关掉增强、用原图评估，否则成绩无法复

现。**第二**，“标签不变”依赖任务语义，不能想当然：手写数字**旋转 180 度会把 6 变成 9**，上下翻转则把它变得不像任何数字——对 MNIST 这两类变换就不合法；而字母“b”和“d”恰好互为镜像，做 OCR 时连水平翻转都不能用。动手选增强前，先问自己一句：变换之后，**我**还认得出它是几吗？

6.4　训练循环：五个动作的摩天轮

万事俱备：有数据（6.1节）、有结构（6.2 节）、有防过拟合手段（6.3 节），现在把第 2 讲的反向传播和第 3 讲的梯度下降装进一个循环。现代深度学习训练就是一个永远在转的摩天轮，每转一圈执行五个动作：

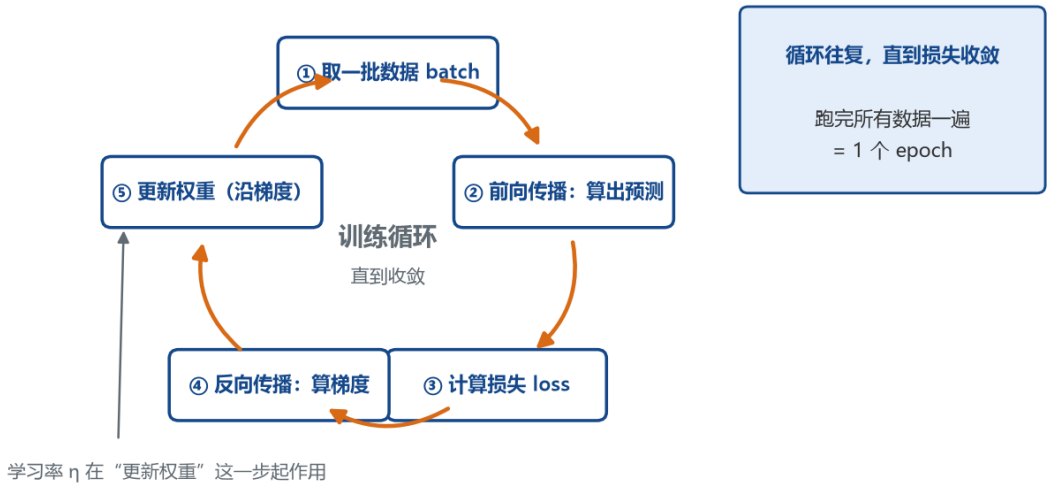


图 6.4　图题：训练循环流程环：取 batch → 前向 → 算 loss → 反向 → 更新参数，五步首尾相接循环往复，直到收敛或达到轮数上限

取 batch：不一次吃完 6 万张图，而是每次抓一小撮（如 64 张）——一小撮叫一个**批**（batch），整批前向加反向算一步；全部训练数据过完一遍叫一个**轮**（epoch），6 万张除以批大小 64，一个 epoch 约 938 步。**前向**：图像过网络，得到 10 个类别的得分。**算 loss**：用交叉熵损失把得分与标准答案的差距折成一个数（第 3 讲的损失函数家族在分类任务上的主力成员）。**反向**：`loss.backward()` 一行触发反向传播，算出每个参数该背多大责任（梯度）。**更新**：优化器让每个参数沿负梯度挪一小步。转，不停转，直到 loss 不再下降。PyTorch 版全文如下——第 2 讲那个手写循环，翻译成现代框架长这样，骨架一毫米没变：

```
import torch, torch.nn.functional as F

model = LeNetLike()                              # 6.2节那样的卷积网络
opt = torch.optim.Adam(model.parameters(), lr=1e-3)

model.train()                                    # 切到训练模式（见下方知识盒）
for epoch in range(5):                           # 整个训练集过5遍
    for x, y in train_loader:                    # 1) 取一批(batch)数据
        opt.zero_grad()                          #    清掉上一批的旧梯度
        pred = model(x)                          # 2) 前向：得到10类得分
        loss = F.cross_entropy(pred, y)          # 3) 算loss：差距多大
        loss.backward()                          # 4) 反向：算出所有梯度
```

```
        opt.step()                              # 5) 更新：沿负梯度挪一步
    print(f"epoch {epoch}, loss {loss.item():.4f}")
```

五步中只有第 1、2、3、5 行会因任务而变，第 4、5 行的“反向加更新”雷打不动——日后你在三个项目里训练任何模型，看到的都是这个摩天轮。

知识补齐：训练模式与推理模式——train/eval 一行之差

PyTorch 的网络对象自带一个开关：`model.train()` 与 `model.eval()`。它不改变参数，改变的是某些层的行为：**Dropout** 训练时随机“关掉”约一半神经元（逼网络不依赖个别神经元，防过拟合），推理时必须全部打开；**Batch Normalization**（下一节主角）训练时用当前批次的均值方差，推理时改用训练期间累计的滑动平均——因为推理时常常一张一张来，没有“批”可统计。忘记切模式是新手第一 bug：训练脚本忘了 `model.train()`，BN 统计错乱、loss 抖成心电图；评估忘了 `model.eval()`，随机关掉的神经元让准确率莫名掉一截，而曲线看起来一切正常。记住：**训练前 train()，评估前 eval()**，一行搞定。

6.5　批归一化 BN：给每一层的数据“扶正”

训练一个深网络为什么难？有一半的难处来自一个隐蔽的连锁反应：**每一层都在适应上一层的变化**。第 2 层的输入是第 1 层的输出——训练中第 1 层的参数每个 batch 都在更新，于是第 2 层脚下的大地一直在漂移；它刚适应好，上游又变了，只能跟着重学。层数一深，层层漂移叠加，训练就变得又慢又抖，学习率还不敢开大。2015 年 Ioffe 和 Szegedy 给出的解法朴素而漂亮：**既然每层的输入分布会漂，那就每一层都把它固定住**。这就是**批归一化**（Batch Normalization，简称 BN）。

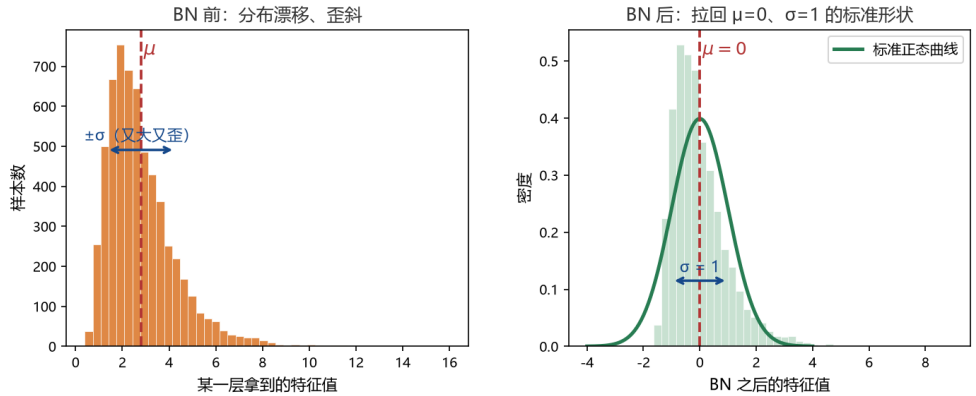


图 6.5　图题：BN 效果示意：同一批特征值归一化前（偏斜、范围杂乱）与归一化后（集中在 0 附近、方差稳定）的分布直方图对比

做法分两步。**第一步，拉回标准位**：对每个通道，用当前批次的统计量把数值强行归一到均值 0、方差 1——就是把歪的数据“扶正”（见上图前后对比）。**第二步，还网络一点自由**：再乘一个可学习的缩放系数 γ（伽马）、加一个可学习的平移 β（贝塔）——“扶正是原则，至于扶正后想再歪一点还是更正一点，网络自己决定”，万一原始分布本来就更好呢？在代码里，它的位置是约定俗成的三连：

```
self.net = nn.Sequential(
    nn.Conv2d(1, 16, 3),        # 1) 卷积：提取特征
    nn.BatchNorm2d(16),         # 2) BN：把分布扶正（卷积后、激活前）
    nn.ReLU(),                  # 3) 激活：非线性
```

```
)
```

为什么放在卷积后、激活前？因为激活函数各有“舒适区”（第 2 讲），BN 保证送进激活的数值分布稳定，避免大片数据落在饱和区。回报是立竿见影的三件套：**学习率可以开得更大、收敛显著更快、训练更稳**；顺带还有轻微的正则化效果。代价也有一条：训练时它依赖批统计，批太小（比如显存紧张只开到 4）统计就不准——工程上另有小批专用的变体，本课不展开。2015 年之后，几乎所有像样的卷积网络里都躺着一行 `BatchNorm2d`，包括你第 7 讲要见的主角们。

6.6　评估：准确率会骗人

训练跑完，测试集准确率 98.5%——可以交差了吗？先别急。**准确率（accuracy）只回答“总体判对的比例”，它会把所有错误一视同仁地藏起来**；要看清一个分类器，你需要一张它的“体检报告”。

表 6.3　MNIST 上的混淆矩阵示例（行＝真实类别，列＝预测类别；数字为一组真实训练的近似记录）。对角线是判对数量，其余全是不合格的“岔路”

真实 \ 预测	0	1	2	3	4	5	6	7	8	9
0	**956**	0	1	0	0	2	4	1	9	7
1	0	**1112**	3	1	1	1	3	4	7	3
2	12	18	**930**	18	8	3	8	12	18	5
3	3	6	26	**903**	2	25	4	7	23	11
4	5	3	8	1	**920**	1	10	4	8	22
5	13	4	4	24	4	**795**	13	3	26	6
6	15	7	7	1	6	10	**900**	1	9	2
7	7	12	12	2	6	3	1	**957**	8	20
8	12	8	14	18	9	17	12	6	**863**	15
9	8	4	5	17	26	9	3	17	14	**906**

这张表叫**混淆矩阵**（confusion matrix）：对角线上是各类判对的数量，**对角线以外的亮斑就是“张冠李戴”发生地**。读它的乐趣在于发现规律：上表里最亮的两块误判是 4 认成 9（22 次）、9 认成 4（26 次）——它们只差一笔的闭合；3 与 5 互认（25 与 24 次）——下弧线写得平一点就是对方。人看一眼就服气：“确实像”。所以混淆矩阵不只报成绩，还**告诉你该去补什么样的数据**（多收集连笔的 4 和 9）。

准确率之外，最常用的两个指标是**精确率**与**召回率**。以“找出 4”为例：**精确率**＝模型报出的“4”里有多少真是 4——衡量**报出来的准不准**；**召回率**＝所有真 4 里被模型抓出来多少——衡量**该找的找到没**。两者天然互相牵制：过滤条件卡得越严，报得越准，但漏掉的越多。把它们调和平均一下，就是综合分 **F1**。

扫盲：一张化验单读懂 TP / FP / FN / TN

把模型想成化验科医生：真的有病/没病，和化验单报了有病/没病，交叉出四种结局。**TP**（真阳性）＝有病且查出来了；**FP**（假阳性）＝没病却误报——虚惊一场；**FN**（假阴性）＝有病却漏检——**最危险的一种**；**TN**（真阴性）＝没病且没报。于是：精确率 $=\mathrm{TP}/(\mathrm{TP}+\mathrm{FP})$，“报告单的可信度”；召回率 $=\mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$，“漏检率的反面”。在锚链检查这种安全场景（第 二 章），一次 FN 就是漏掉一条缺陷链环——所以水下项目的评测组盯召回率比盯准确率更紧。

最后是一个能救命的提醒。**类别极不平衡时，准确率会撒弥天大谎**：产线质检里 99% 的图是良品、1% 是缺陷，一个把所有图都判“良品”的懒模型，准确率高达 99%——召回率却是 0%，一件缺陷都抓不到，在产线上一文不值。记住这条反例，你就永远不会再被“99% 准确率”单独忽悠：**先看**

混淆矩阵，再谈准确率。这正是你若担任项目评测组（第 一 章六目组 5、第 二 章水下组 4）时的日常职业。

6.7　过拟合对策清单

第 3 讲那张训练 loss 一路下降、验证 loss 先降后升的双曲线图，从今往后你几乎每周都会重逢：它一出现，就是在喊“过拟合了”。对策是成熟套路，列成清单逐条点名：

表 6.4　过拟合对策清单：按“性价比”排序，前三条几乎零成本

手段	一句话原理	PyTorch一行	成本
更多数据	见得多了就不会背	——（最难的一条）	高
数据增强	白捡新数据（6.3 节）	`T.RandomRotation(10)`	低
早停	验证 loss 回升就停	盯曲线手动停即可	低
L2 正则	惩罚过大的权重	`weight_decay=1e-4`	低
Dropout	随机关神经元防抱团	`nn.Dropout(0.5)`	低
批归一化	顺带轻微正则	`nn.BatchNorm2d(64)`	低

使用顺序也有讲究，老手的打法是：**先刻意把网络在小数据上过拟合（证明它有能力学），再逐条上对策把过拟合压下去**。如果连过拟合都过拟合不起来，说明网络太小或代码有 bug，加正则只会雪上加霜。这份清单请存进你的工具箱——实验三（11 月 23 日）训练曲线一旦翘尾，第一反应就该是翻它。

前沿速递：2026 年，“从零训练”已经是一件稀罕事

本讲教会你的“拿数据、定结构、转循环”从零养大一个 CNN，在 2024–2026 年的工业界反而成了**稀罕做法**。真实流程几乎都是**预训练–微调范式**（pretrain–finetune）：先在超大通用数据集上训练一个“底座”模型（这步通常由大厂完成并开源），你只需要在底座上用你自己的小数据微调最后几层——用 `timm` 或 HuggingFace 生态，十分钟就能得到一个像样的图像分类器，数据少到每类几十张也行。**从零训练如今主要发生在两类地方：课堂教学（比如本讲）和前沿研究**。这条范式的时间线值得记一笔：第 7 讲的 ImageNet 预训练是它的起点，2021 年起的 CLIP、DINOv2 把“底座”从卷积换成 Transformer，而你在水下项目里“微调 FoundationStereo”正是这一范式的现场直播。所以本讲的价值不在“复刻 1998”，而在让你**看懂整个范式最底层的那台发动机**——微调时转的，仍是本讲这五个动作。

项目连线：第 6 讲 × 三个项目——同一个摩天轮，三种坐法

实验三（11 月 23 日）是“模型训练实践”专场，三个项目的训练全是本讲循环的实例。六目拼接（第 一 章）：九个参数头各自在标注数据上用本讲循环训练，产出“头名字`.onnx` + 同名`metrics.json`”一对文件，拷入 `build/Release/models/` 生效——`metrics.json` 就是该头的“体检报告”，**逐头指标不劣于规则初值才允许替换**，否则系统自动回退保护。**水下双目**（第 二 章）：训练组的日常是微调 FoundationStereo 时盯着 loss/EPE 曲线下降——EPE 是立体匹配版的“平均错多少像素”，到了实验三的训练现场你天天会见到它。**数据基座**（第 三 章）：真假样本训练分类器，教科书式的两类分类——本讲的混淆矩阵和精确率/召回率就是它的验收语言。最后划重点：**六目组 5 与水下组 4（评测组）的整个职业，就是 6.6 节**——别人训练，你们审判。

本讲课后题：亲手把一个 LeNet 风格的 CNN 养到 98% 以上，并体验“改一个旋钮、看一次变化”的实验感。五要素如下。

第 6 讲课后编程题（用 AI 实现）

题目：指挥 AI 辅助实现一个 LeNet 风格的卷积神经网络（两层卷积–池化堆叠 + 全连接收尾，参考 6.2 节户型），在 MNIST 上训练并达到**测试准确率 $\geq$ 98%**；然后**自己改动一个超参数**（学习率、batch size、是否加数据增强三选一），用完全相同的方式再跑一遍，把两次的训练曲线（每轮的 loss 与测试准确率）各画成一张图，对比差异并解释原因。

交付物：(1) 代码文件 `lenet_mnist.py`；(2) 两张训练曲线截图（基线一次、修改一次，图上标注最终准确率）；(3) 一段 50 字以上的说明，写清“我改了什么、预期与实际效果是否一致”。

提示词建议（可直接抄）：“我是零基础 Python 学生。请用 PyTorch 写一个 LeNet 风格的 CNN：两层‘卷积 +ReLU+ 池化’后接全连接 120、84、10，结构对齐教材第 6 讲。在 MNIST 上训练：batch size 64、Adam、学习率 0.001、5 个 epoch；每个 epoch 结束记录测试集 loss 和准确率并存成列表，最后用 matplotlib 画出‘epoch–准确率’曲线并打印最终准确率；每行代码加中文注释；最后告诉我：若准确率到不了 98%，学习率、batch size、数据增强分别该怎么调？”

验收线：代码在你自己的电脑上跑通、最终准确率 $\geq$ 98%；随机抽问时能不看资料讲清训练循环五步各自干了什么、`model.train()` 与 `model.eval()` 的差别；至少有一处自己的修改（换超参数、加 BN、加增强都算）。**代码是 AI 写的没关系，讲不清楚就是没做。**

加分项：用 6.6 节的方法打印你模型在测试集上的混淆矩阵，找出最容易混淆的一对数字，用一两句话分析原因（它们长得像在哪？），并尝试用一种针对性的数据增强把这个误判次数压下去。

6.8　从这一讲到 2026

本讲这个五动作摩天轮，在 2026 年的训练现场转成了什么样？三个事实，替你把“怎么训”的地图更新到今年。

其一，数据增强已从“技巧”变成“默认配方”。2024 年的一篇综述把 MixUp、CutMix、RandAugment 等数百种增强方法纳入统一框架，梳理成“训练流水线中的标准模块”，覆盖视觉乃至多模态任务，并维护着持续更新的开源项目——今天训练视觉模型，增强列表与优化器、学习率一样，是开工就要填的默认三件套之一。你课后题里随手加的 `T.RandomRotation(10)`，就是这套配方里的最小一员；比它更凶的 MixUp 干脆把两张图按比例混成一张、标签也按比例混合，逼网络学“类别之间的过渡带”。

其二，“从零训练”进一步让位给“冻结底座 + 轻量探针”。本讲前沿速递盒子说过预训练–微调范式，2025 年 8 月 Meta 的 DINOv3 又往前推了一步：官方声称其自监督特征**无需微调**——把整个主干冻住、只训一个线性探针或小检测头，就能在广泛任务上超越此前专门训练的最优模型，并发布了覆盖不同算力档位的模型套件。训练的活儿，正从“调千万个参数”变成“选底座、接探针”。医疗影像那篇系统综述也站在同一边：ViT 能否胜过 CNN，关键变量就是有没有预训练。

其三，“怎么训”依旧是活跃战场。2026 年 1 月发布的 YOLO26 带来自研的 MuSGD 优化器、渐进损失（Progressive Loss）与小目标感知标签分配（STAL），五档规模在 COCO 上做到 40.9–57.5 mAP——连最成熟的工业工具链都还在迭代训练配方。与此同时，一条工程底线也已定型：公开配方、固定随机种子、可复现的成绩单，不再是美德，而是纪律。

给 2026 年的你三句话：**把预训练当地基，把增强当配方，把种子和日志当实验纪律**——而本讲的摩天轮，是这一切共同的地板。

MNIST

7 万张 28×28 灰度手写数字数据集（6 万训练 + 1 万测试）；深度学习的 Hello World。

CIFAR-10

6 万张 32×32 彩色小图、10 类；MNIST 的进阶版，第 7、8 讲课后题的战场。

批（batch）

一次前向/反向共用的一小撮样本；批大小是最先要定的训练旋钮之一。

轮（epoch）

全部训练数据完整过一遍；训练通常是若干个 epoch 的循环。

数据增强（data augmentation）

对样本施加不改变标签的随机变换（旋转、平移、翻转、噪声等），等效于免费扩充数据。

批归一化（Batch Normalization，BN）

在卷积后、激活前把每通道数值按批次统计扶正到均值 0 方差 1，再用可学习的 γ, β 微调；换更快收敛与更稳训练。

训练模式/推理模式

`model.train()` 与 `model.eval()` 切换的行为开关：Dropout 与 BN 在两种模式下行为不同。

混淆矩阵（confusion matrix）

行为真实类、列为预测类的矩阵；对角线判对，亮斑暴露易混类别对。

精确率（precision）

模型报出的正例中真正例的比例——“报出来的准不准”。

召回率（recall）

全部真实正例中被模型找出的比例——“该找的找到没”；安全场景的头号指标。

F1 分数

精确率与召回率的调和平均，两者的“一票综合”。

早停（early stopping）

验证集 loss 回升即停止训练，是过拟合清单里成本最低的一条。

第 7 讲　卷积神经网络的演化：加宽、加深与残差

2012年秋天，ImageNet 大规模视觉识别竞赛（ILSVRC）结果公布，排行榜让全场沉默了一瞬：一支几乎没人听说过的队伍，把 top-5 错误率从上一年的 25.8% 一刀砍到 15.3%，比第二名的 26.2% 整整低了近 11 个百分点——在往届每年只挪动一两个百分点的赛场上，这不是进步，是降维打击。冠军队伍是多伦多大学辛顿（Geoffrey Hinton）组的三名成员，训练设备是**两块打游戏用的 GTX 580 显卡**（他们不好意思申报“游戏显卡”，在论文里含糊地写了“GTX 580”）。此后十年，这条“加深、加宽、加残差”的军备竞赛一路把错误率压过人类水平，也把深度学习从冷宫一路抬进了你口袋里的手机。本讲就沿着这条时间线走一遍：每一代网络解决了什么瓶颈、又留下了什么新瓶颈——这是理解你项目里那些 backbone 的必经之路。

本讲地图

讲什么：ImageNet竞赛与错误率曲线、AlexNet 的三大法宝、VGG 的 3×3 极简主义、GoogLeNet 的多尺度并行与 1×1 降维、深层网络的“退化”难题、残差连接革命、设计经验法则。**为什么值得学：**六目端到端路线的 backbone 是 ResNet 家族，FoundationStereo 的 ViT 主干里也全是残差连接——你每天都在用 2012–2015 年的遗产。**学完你能：**说出五代网络的接力关系；解释“两个 3×3 抵一个 5×5”的算术；讲清残差为什么让百层网络能训练。**课后阅读：**许庆阳《深度学习模型与算法基础》第 86–100 页。

7.1　ImageNet：军备竞赛的“高考考场”

故事要从考场说起。2012 年之前的图像识别研究，各自为战：你用你的几百张图，我用我的几千张图，成绩无法比较，模型也无法拉开差距。斯坦福的李飞飞团队看中了这个死结，从 2007 年起发动一场笨功夫运动：借 WordNet 的概念树，靠众包平台雇了一百多个国家的标注员，一寸一寸给互联网图片打标签。到 2009 年，**ImageNet** 数据库初具规模——累计标注超过 1400 万张图片；从 2010 年起，每年用它举办 ILSVRC 竞赛，固定 1000 个类别，约 **140 万张**带标注图片（其中训练集 120 万张）。

为什么规模如此重要？因为第 6 讲你已经养过 CNN，知道它们的胃口：**参数越多，需要的燃料越多**。过去十几万参数的 LeNet 吃 6 万张 MNIST 尚可，想区分 1000 类猫猫狗狗花花世界，缺的从来不是模型点子——AlexNet 的卷积思想 LeCun 在 1989 年就有了——缺的是 140 万张图这样的燃料，和烧得起这些燃料的炉子。ImageNet 一举补齐了前一半，2012 年的显卡补齐了后一半，两头凑齐，深度学习点火成功。

考场的记分方式也要弄懂：**top-5 错误率**——模型对每张图给出 5 个候选类别，只要正确答案不在 5 个里才算错。给 5 次机会还算错，说明模型错得很离谱。下图是历年冠军的错误率曲线，注意三个刻痕：2012 年悬崖（AlexNet）、2014 年过半山腰（GoogLeNet 6.7%）、2015 年 **3.57%，首次低于人类水平（约 5%）**——此后考题对人类反而太难了，2017 年办完最后一届后竞赛功成身退。

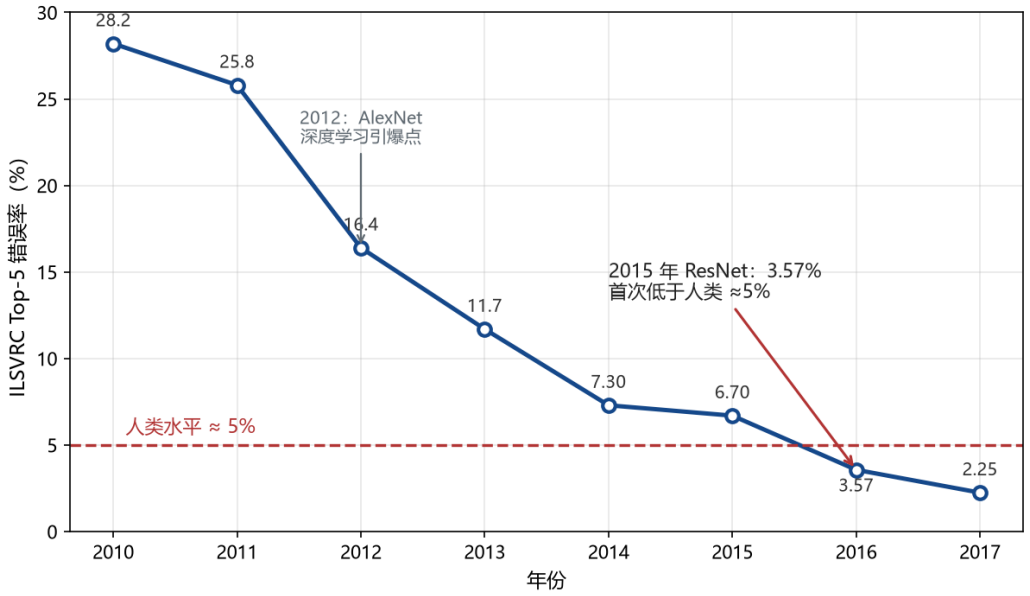


图 7.1　图题：ILSVRC 2010–2017 年冠军队伍 top-5 错误率逐年下降折线：2012 年 AlexNet 掀开悬崖，2015 年残差网络 3.57% 首次越过人类水平线（约 5%）

7.2　AlexNet：一鸣惊人的三件法宝

冠军网络后来得名 **AlexNet**（一作 Krizhevsky 的名字叫 Alex）。它的结构照抄 LeNet 的户型——卷积池化堆叠提取特征、全连接收尾——只是整栋楼放大了：**8 层**（5 层卷积 + 3 层全连接），6000 万参数，是 LeNet-5 的一千倍；输入也从 32×32 的小图换成 224×224 的彩照。结构上并无革命，**革命在训练配方**。三件法宝至今仍在你的代码里：

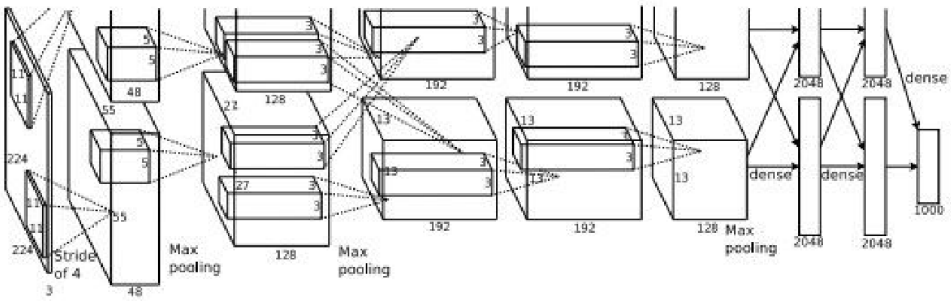


图 7.2　AlexNet 网络结构经典图：上下两半对应两块 GPU 各管一半通道、仅在特定层互通；5 段卷积逐级提特征，3 层全连接收尾分类

图片来源：Krizhevsky et al., ImageNet Classification with Deep Convolutional Neural Networks, NeurIPS 2012

法宝一：ReLU。第 2 讲见过它：$f(z) = \max(0, z)$，比 sigmoid/tanh 不饱和、梯度不断流。AlexNet 论文实测，同样网络换用 ReLU 后收敛快数倍——没有它，6000 万参数在 2012 年的硬件上根本练不出来。**法宝二：Dropout**。训练时随机丢弃约一半神经元，逼网络不把身家押在任何单个神经元上（第 6 讲过拟合清单里的老朋友，就是它发家的地方）。**法宝三：GPU 训练**。卷积是海量的小乘加，与显卡的本职工作完美同构（见下方扫盲盒）；两块 GTX 580 把六天的训练压到可接受，也把“买显卡炼模型”变成了此后十年的行业仪式。三件法宝合起来的启示，比网络本身重要得多：**当数据与算力到位，老想法也能爆发出新世界**。

法宝之外，论文里还躺着三个如今已有“盖棺定论”的细节，正适合当历史注脚读。**一是 LRN（局部响应归一化，Local Response Normalization）**：让相邻通道的响应互相“别苗头”，谁周围太兴奋就压谁一头——2012 年它是涨分利器，五年后 BN（第 6 讲）登场，干同样的活更简单更好，LRN 从此退居故纸堆：连“怎么归一化”都是演化中的部件。**二是重叠池化（overlapping pooling）**：池化窗口 3×3、步幅却只有 2——相邻窗口叠着滑，信息不至于一步丢太快，论文实测比标准的不重叠池化低约 0.4 个百分点的 top-1 错误率。**三是双 GPU 训练**：当年一块 GTX 580 只有 3 GB 显存，装不下 6000 万参数，于是两块卡各管一半神经元、仅在个别层交换数据——这本质是**模型并行**（一层拆两半）；今天更常用的是**数据并行**（每卡吃不同的批、各自算梯度再汇总）。注意时代背景：2012 年“并行训练”这个词本身还很新鲜，论文得专门画一张双卡结构图解释；如今你实验课上一行 `DataParallel`，就是这段历史的直系后代。

扫盲：为什么偏偏是游戏显卡改变了 AI

显卡（GPU）的本职是每秒在几百万个像素上做同一套乘加运算——把每个点的光照、贴图算出来。而卷积恰好就是“用同一个小核对几百万个位置做同样的乘加”。**一个天生并行、重复、规整的计算，撞上一个天生为它而生的硬件**。一块 RTX 4080 的 FP32 算力约 48 TFLOPS（每秒 48 万亿次浮点运算），相当于几万台 1998 年的 PC 同时开动。你项目六目组在 RTX 4080 上训练端到端模型，用的正是这笔红利。

7.3　VGG：极简主义——只堆 3×3

AlexNet 之后，各家开始“堆料”：核越用越大、层越加越多，结构花样百出。2014 年，牛津大学 VGG 组（Visual Geometry Group）的 Simonyan 与 Zisserman 反其道而行，交出一份极简主义宣言：**把所有花活全部删掉，只用最小的 3×3 卷积和 2×2 池化，像砌砖一样整齐地往上堆**，深度 11 到 19 层。这套毫无惊喜的“砖墙”拿到当年亚军（7.3%），却因为结构干净、易于改造，成了之后数年工程界的默认 backbone，也成了你今天运行 `torchvision.models.vgg16` 的由来。

它的关键洞察是一个算术题，需要先补一个概念——**感受野**（receptive field）：输出上某个神经元“看得见”输入图多大的范围（详细定义见下方知识盒）。两个 3×3 卷积层串起来，第二层的神经元第一层看 3×3、第二层再看第一层的 3×3，合计正好覆盖 5×5——**两层 3×3 抵得上一层 5×5 的视野**。那么为什么不直接用 5×5？算算参数（设通道数都是 C）：

$$5\times5:\ 25C^2 \qquad \text{versus} \qquad 3\times3\times2\text{层}:\ 2\times9C^2=18C^2. \tag{7.1}$$

人话翻译：同样的视野，两层 3×3 比 5×5 少 28% 的参数，还白赚一次 ReLU 非线性（两次弯折总比一次表达力强）。三层 3×3 抵一层 7×7，参数从 $49C^2$ 降到 $27C^2$，赚得更多。**用小核堆深度，比用大核更省、更强**——这一条算术直接把“深度”变成了值得军备的标的。

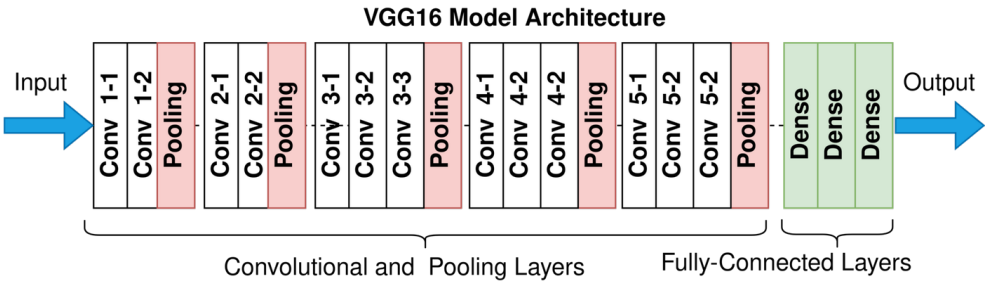


图 7.3　VGG16 结构图：整面“砖墙”全部由 3×3 卷积与 2×2 最大池化交替砌成，五段卷积逐级加厚通道，末端三层数万个单元的全连接收尾

图片来源：Simonyan & Zisserman, Very Deep Convolutional Networks (VGG), ICLR 2015

下图把这五代网络的深度摆在一起，你能看到军备竞赛的坡度。不过 VGG 也留下两个包袱：一是照这个砌法再往深处堆（20 层以上）就**越来越难练**（为什么？7.5 节揭晓）；二是那 1.38 亿参数、500 多 MB 的体格——末端三层全连接吞掉了绝大多数参数，妥妥的“虚胖”，这正是第 8 讲轻量化的第一动力。

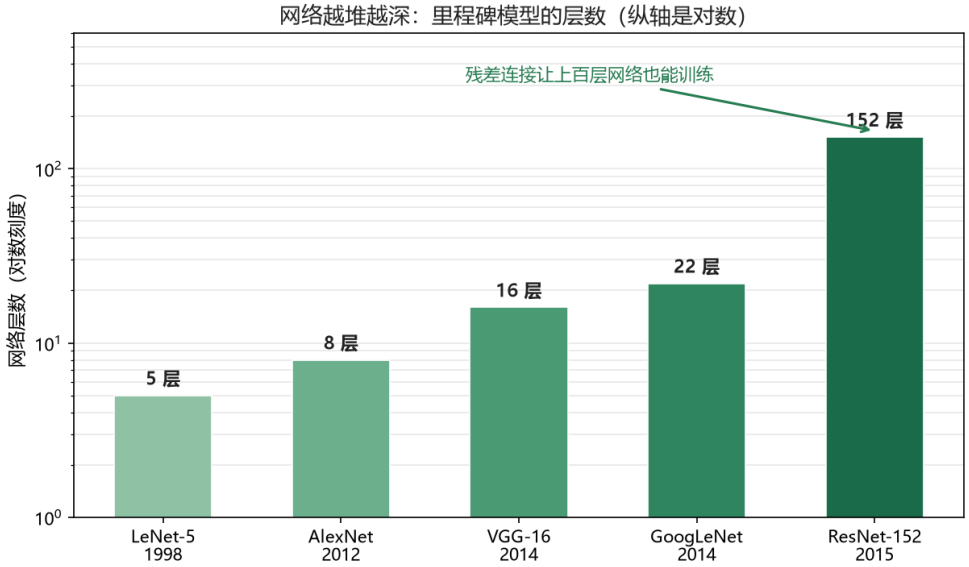


图 7.4　图题：网络深度里程碑条形图：LeNet-5（5 层卷积为主）、AlexNet 8 层、VGG16、GoogLeNet 22 层、ResNet 152 层——军备竞赛的“坡度”一目了然

知识补齐：感受野——神经元的一亩三分地

把卷积网络想成层层上报的情报系统：底层神经元只盯输入图上 3×3 的一小块（它的**感受野**），中层把底层的 3×3 情报再汇总成自己的 3×3，于是中层的感受野是 $(3+3-1)\times(3+3-1)=5\times5$。每加深一层，感受野按“$k$ + 上层的 -1”向外扩一圈。**感受野越大，神经元越能综合全局信息做判断**——识别一只猫需要看到耳朵、胡须和身子的组合，这就要求深层的感受野足够覆盖整个物体。深度之所以有价值，一半原因就是它用廉价的层层叠加换来了大感受野；8.5 节还会介绍不加深也能扩感受野的“空洞卷积”。

7.4　GoogLeNet：多尺度并行与 1×1 省钱术

同年（2014年）的冠军是谷歌的 **GoogLeNet**（名字致敬 LeNet），它提供了另一条军备思路：**不拼纵深拼花样**。它的基本积木叫 **Inception 模块**（名字来自电影《盗梦空间》的梗“we need to go deeper”）：在一层里，1×1、3×3、5×5 三种卷积**并排**开火，池化也并行一路，四路结果在通道维度拼接——同一层同时提取小、中、大三种尺度的特征，**让网络自己学该信谁**，而不是由设计者拍板选一种核。

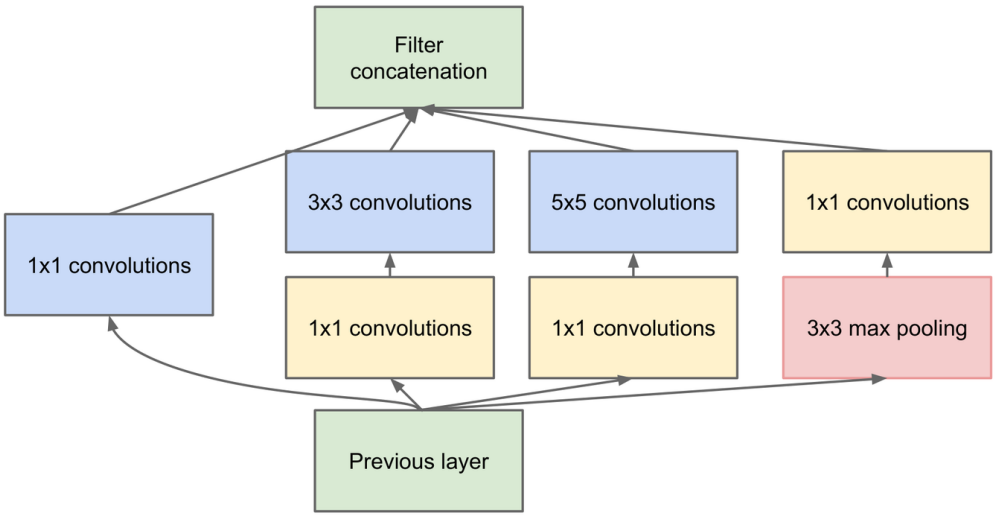


图 7.5　GoogLeNet / Inception 模块图：1×1、3×3、5×5 卷积与池化四路并行，各自接 1×1 瓶颈降维后拼接

图片来源：Szegedy et al., Going Deeper with Convolutions, CVPR 2015

但并行是有代价的：四路全开，计算量会爆炸，尤其 5×5 这种大核。救场的是图里那些前置的**1×1 卷积**——它不扫空间，只在**通道之间做加权组合**，可以理解为“通道维度的全连接”（每个像素位置上，把 256 个通道按权重混合出 64 个新通道）。把它当**瓶颈**（bottleneck）垫在 3×3、5×5 前面，先把 256 通道压到 64 再做大核卷积，该路的计算量立刻降到原来的三分之一以下。这招“1×1 降维省钱”此后无处不在，成为轻量化设计的标准动作。成绩单：GoogLeNet 深达 22 层，参数却只有约 680 万——不到 AlexNet 的十分之一，还更准。**“更深”与“更省”可以兼得**，前提是会设计。

Inception 家族并没有停在 2014 年。随后三年的迭代堪称一部小型演化史，一张表速览：

表 7.1　Inception v2–v4 演化速览：每一代只解决一个明确的瓶颈

版本	关键改动	一句话点评
Inception-v2（2015）	每层卷积后加 BN	达到同等精度，训练步数降到约 1/14——BN
Inception-v3（2015）	非对称卷积：1×3 接 3×1 替代 3×3	同样视野、更少参数，还白赚一次非线性
Inception-v4 / Inception-ResNet（2016）	模块里接入残差捷径	与何恺明的革命合流，训练稳定性再上一层

表里两处值得多说一句。**非对称卷积**：把一个 3×3 拆成先竖后横的两小条（1×3 再 3×1），参数从 9 个降到 6 个，表达能力反而更灵活——你在第 5 讲手算过卷积，可以自己验证“先竖后横”两步的视野确实仍是 3×3。**辅助分类器**：GoogLeNet 在网络中部额外接了两个小分类头、把它们的损失一并算进总损失，本意是给深层“递个梯度”、防梯度消失；后来的复现表明它主要起一点正则作用，对最终精度的帮助很有限，到 v4 时期已经可以放心去掉——又一个“当年灵、后来被更好的招式取代”的注脚，与上一节的 LRN 遥相呼应。

7.5　深化的障碍：56 层为什么输给 20 层

按 VGG 的算术，深度这么划算，那一路加深不就天下无敌了？2015 年，时任微软亚洲研究院研究员的何恺明团队做了一个直白的实验：在 CIFAR-10 上训练普通（plain）卷积网络，20 层对 56 层，其他条件全部相同。结果违反直觉——下图（复现何恺明论文的风格）里，**56 层的训练误差反而比 20 层还高**。

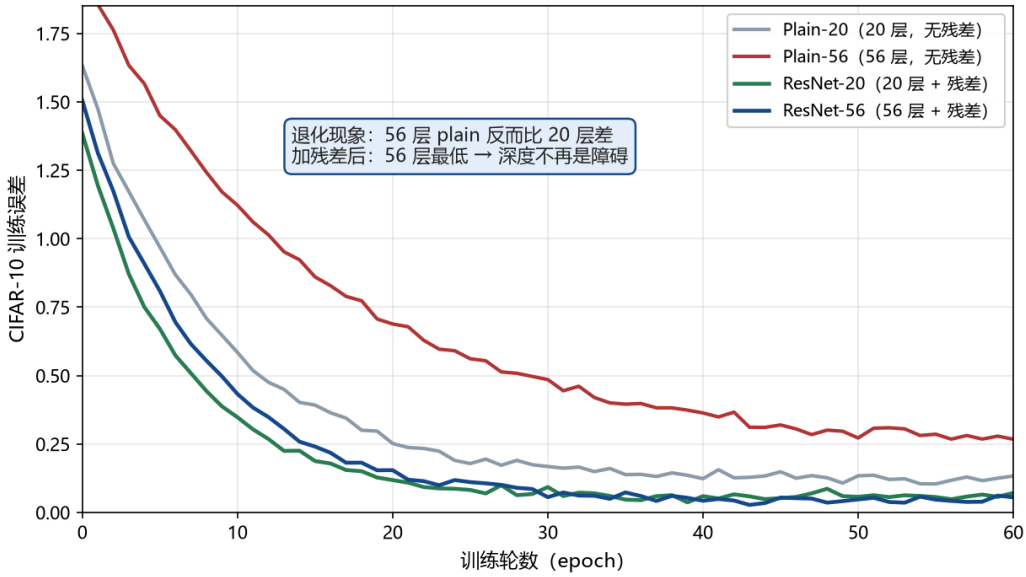


图 7.6　图题：复现 He 等人论文的四条曲线：普通网络 20 层与 56 层（56 层训练误差反而更高）、残差网络 20 层与 56 层（越深越好，且训练误差持续下降）

先把两个最常见的误判排除。**不是过拟合**：过拟合是训练误差低、测试误差高（第 3 讲），而这里是**训练误差本身**降不下去——网络连“背诵”都背不会。**也不是梯度消失**：彼时 ReLU 和归一化初始化（第 6 讲的 BN 也算同类）早已普及，20 层都能正常收敛。剩下的真相被称为**退化**（degradation）：**更深的理论上限不低，优化起来却更难**。想通这一点只需一个思想实验：假设 56 层网络的最优解就是“前 20 层照抄、后 36 层什么都不干”——只要后 36 层都学成恒等映射（输入原样输出），它就该和 20 层一样好。但优化器要费老大劲才能把普通卷积层调到“什么都不干”的精度；层层微小的偏差叠加，56 层反而学歪了。**深网络的问题不是能力不够，而是“维持现状”这件简单事对它来说太难。**

7.6　残差革命：给梯度修一条高速公路

何恺明的解法堪称优雅：**既然“什么都不干”难学，那就把它变成默认**。给网络加一条**恒等捷径**（identity shortcut）：把输入 x 原封不动抄送到输出端，与卷积层的输出相加：

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}, \tag{7.2}$$

人话翻译：$\mathbf{x}$ 是这一块的输入，$\mathcal{F}$ 是两三层卷积想学的东西，输出不再是卷积层单独的答案，而是“$\mathcal{F}$ 对 $\mathbf{x}$ 的修改量加在 $\mathbf{x}$ 上”。换个角度看，卷积层学的内容从完整答案 $H(\mathbf{x})$ 变成了修改量 $\mathcal{F}(\mathbf{x}) = H(\mathbf{x}) - \mathbf{x}$——所以叫**残差**（residual，即“剩余的修正量”）。**学“修改量”永远比学“完整答案”容易**：如果这层根本不需要干活，把 $\mathcal{F}$ 压向 0 即可（权重本来就被正则往小拽，见第 6 讲清单），恒等映射白送；如果需要干活，也只需学从“现状”到“目标”的那一小段差。

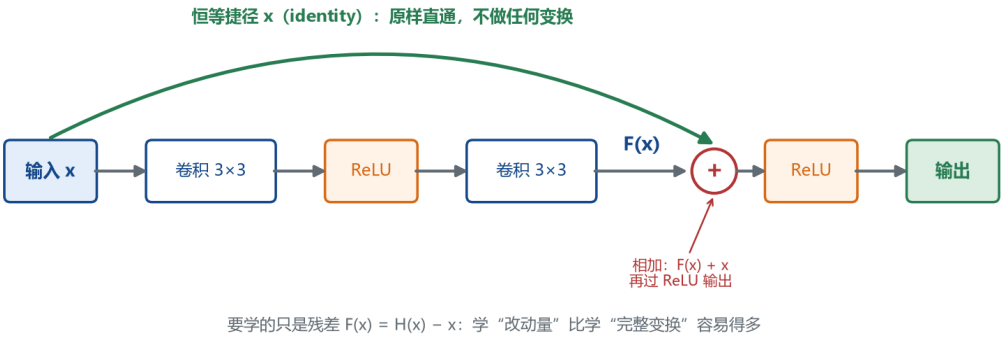


图 7.7　图题：残差块：输入 x 兵分两路——主路经两层卷积学修改量 $F(x)$，捷径路原样直通，两路相加后再过激活；标注“学习目标从 $H(x)$ 换成 $F(x)=H(x)-x$”

下图是 ResNet 论文原图，左列的两个小积木就是上图的实际尺寸（两种厚薄规格，深网络用带 1×1 瓶颈的右款）。别小看这条抄近道的加法边，它同时解决了两件事：**前向**，“什么都不干”成为零成本默认值，百层网络退化问题消失；**反向**，求导时梯度从输出端沿捷径一路“原样 +1”地流回输入端，中间任何一层都拦不住它——相当于给梯度修了一条不设收费站的高速公路，再深的网络也不会在训练初期“断粮”。组装起来就是 **ResNet**：把第 6 讲的卷积块换成残差块，一路砌到 **152 层**仍然稳定训练——2015 年，ResNet-152 以 **3.57%** 的 top-5 错误率夺冠，首次越过人类水平线，这篇论文拿下 CVPR 2016 最佳论文奖，也是中国学者（何恺明，清华本科、港中文博士）在深度学习史上最响亮的一笔。记住这个结构，它被称为深度学习史上最伟大的一个“加法”。

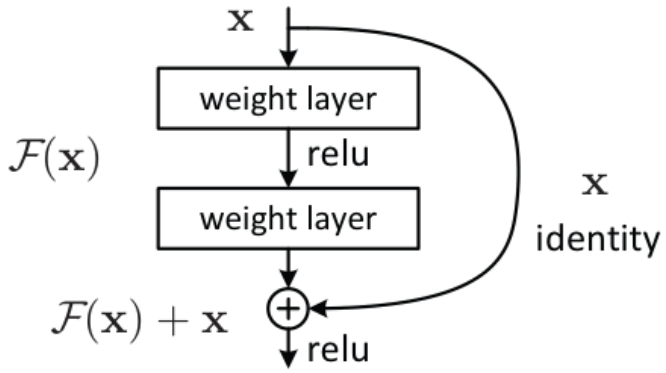


图 7.8　ResNet 论文原图：左侧两种残差块规格（普通款与 1×1 瓶颈款），右侧串联成 34 层网络与 VGG-19、普通 34 层的对比

图片来源：He et al., Deep Residual Learning for Image Recognition, CVPR 2016

7.7　设计经验法则：深度、宽度、分辨率的三角

军备竞赛打到 2015 年，兵法已经基本成型。把五代网络的成败浓缩成一张经验法则表，你日后自己选 backbone、改结构时照着想：

表 7.2　网络设计经验法则速查：想动哪一格，先看配什么药

想加什么	必须配的药	出处/代表
深度（层数）	残差连接，否则退化	ResNet（7.6节）
宽度（通道）	分组卷积省算力	AlexNet 双卡 / ResNeXt
感受野（看得远）	空洞卷积或多加层	DeepLab（第 8 讲）
省算力	1×1 瓶颈降维	GoogLeNet（7.4 节）
小参数量	小核堆深 + 压通道	VGG 算术 / MobileNet（第 8 讲）

最后一条全局观：**深度、宽度、分辨率是算力天平上的三个砝码**——输入图从 224×224 加到 448×448，计算量变 4 倍；通道翻倍，计算量翻 4 倍；层数翻倍，计算量翻约 1 倍。三者同比例放大收益最好（2019 年的 EfficientNet 把这条直觉总结成了“复合缩放”公式）。**结构决定上限，训练技巧（第 6 讲）决定你能不能摸到上限**——两讲合起来，才是完整的“养网络”手艺。

前沿速递：CNN 死了吗？——2024–2026 视觉骨干的真相

2017 年 Transformer 进军视觉（ViT）后，“CNN 已死”的标题年年都有。2022 年 ConvNeXt（FAIR，《A ConvNet for the 2020s》）狠狠回击：把 ResNet-50 按 2020 年代的训练配方和结构技巧逐项现代化，性能反超同期的 Swin Transformer——**CNN 不是不行，只是修炼多年的功课被 ViT 一步学走了**。2024–2026 年的现实格局是：**搞通用基础模型，ViT 系当家**——CLIP 图文对齐、SAM 分割、DINOv2 自监督特征，全是 ViT 骨干；**上端侧、跑实时，CNN 仍是主力**——手机、车载、工业相机里的检测分割模型至今多属 CNN/MobileNet/YOLO 家族（第 8 讲）。而对本讲最重要的是：**ResNet 的思想已经统治一切**——每个 Transformer 块写的都是“$x+$ 注意力 (x)”，捷径相加原封不动地搬了过去（第 11 讲细看）。学 ResNet 不是学古董，是学所有深度网络的通用语法。

项目连线：第 7 讲 × 三个项目——你的项目里全是这一讲的遗产

六目拼接（第 一 章）：端到端路线 `ropstitch.onnx` 的 backbone 正是 ResNet 家族——组 7 在 RTX 4080 上用第 6 讲的循环训练它；组 8 接手做量化压缩。而九头路线的“**逐头指标不劣于规则初值才允许替换、否则自动回退**”验收线，就是恒等捷径的**系统级翻版**：新参数头是“修改量”，规则初值是“恒等捷径”——改动只在证明不劣于现状时才被接纳，和 $H(x)=x+F(x)$ 保底不劣于 x 是同一个工程哲学。**水下双目**（第 二 章）：FoundationStereo 的 ViT 主干里，每个块都是“捷径 + 注意力”的残差结构——微调它，就是在 152 层那次革命的地基上盖楼。**数据基座**（第 三 章）：监考员、判官模型的骨干同样多为 ResNet/ViT 家族。一句话：**这一讲不是历史课，是你三个项目的零件说明书**。

军备竞赛的结论之一是“深就是好”，但你也该亲眼看一次**为什么深必须配残差**——课后题让两胞胎网络当场对决赛跑。五要素如下。

第 7 讲课后编程题（用 AI 实现）

题目：指挥 AI 辅助在 CIFAR-10 上做一场“有无残差”的对照实验：搭两个参数量接近的 6 层卷积网络——A 版为普通串接（plain），B 版在每两层卷积后加一条恒等捷径（shortcut）相加后再激活；两者用完全相同的训练设置（batch size 128、学习率 0.001、Adam、各训 20 个 epoch），把两条训练 loss 曲线与两条测试准确率曲线画在同一张图上对比，并写一段 50 字以上的结论：

残差版收敛是否更快、最终是否更高、差距大不大。

交付物：(1) 代码文件 `plain_vs_res.py`（内含两个网络定义与统一训练函数）；(2) 对比曲线截图（至少含训练 loss 与测试准确率两幅）；(3) 50 字以上的实验结论。

提示词建议（可直接抄）：“我是零基础 Python 学生。请用 PyTorch 在 CIFAR-10 上写两个 6 层卷积网络做对照：网络 A 为普通串接（卷积 +ReLU 堆 6 层，中间夹两次最大池化）；网络 B 参数量与 A 接近，但每两层卷积后加恒等捷径：输出 = ReLU(卷积输出 + 输入)，若通道数不一致用 1×1 卷积对齐。两者共用同一个训练函数：batch size 128、Adam、学习率 0.001、各训 20 个 epoch；每个 epoch 记录训练 loss 和测试准确率，最后用 matplotlib 把两条 loss 曲线画一张图、两条准确率曲线画一张图；每行加中文注释。”

验收线：两个网络都训完 20 个 epoch 且训练设置完全一致；能不看资料向同学解释“捷径相加为什么能让 56 层网络训练误差降下来”；至少有一处自己的修改（加深到 8 层、改学习率、换 batch size 都算）。**代码是 AI 写的没关系，讲不清楚就是没做。**

加分项：把两个网络都加深到 10 层再比一轮，观察“差距是否随深度拉大”；或在训练中记录每个 epoch 的梯度范数（提示：遍历 `model.parameters()` 取 `p.grad.norm()`），验证“捷径让梯度流得更通”。

7.8　从这一讲到 2026

“CNN 之死”的论战，到 2026 年有了判决书式的答案：**分疆而治，且互相渗透**。四条 2024–2026 年的硬事实，给本讲的军备竞赛续上最新一章。

ViT 阵营的三大地标。其一，Meta 的 DINOv3（2025 年 8 月）在 **17 亿张**精选图像上自监督训练出最高 70 亿参数的视觉骨干，靠“Gram anchoring”技术守住了长训练中稠密特征图的质量，官方称冻结后加个小探针即可超越各专门任务的此前最优——军备竞赛的“燃料”逻辑（7.1 节）在今天依然成立，只是考场换成了自监督。其二，SAM 2（2024 年 7 月开源，Apache 2.0）把一张图像当作单帧视频，用**流式记忆**模块统一了图像与视频的可提示分割：最小模型在 A100 上约 91.2 FPS，实时。其三，IDEA 的 DINO-X（2024 年 11 月）用超一亿样本的 Grounding-100M 支撑**开放词汇检测**：文本说得出名字的、乃至说不出名字的物体都能检，零样本 LVIS 稀有类 63.3 AP、超先前最优 5.8 AP——检测从“认训练集里那 1000 类”走向“万物皆可检”。

卷积阵营的反击同样有板有眼：2026 年 1 月发布的 YOLO26 免 NMS 端到端、检测头更轻，CPU 推理较上代最高快 43%，实时检测的工业默认仍然姓卷；2024 年 7 月的 Grounding DINO 1.5 Edge 为了冲开放词汇检测的边缘速度，干脆请 EfficientViT 来当图像编码器；ConvNeXt V2 的设计思想则被 MobileNetV4 吸收进搜索空间——连 Transformer 阵营的工具箱里都装着卷积零件。

所以 2026 年的真实图景不是绞刑架，而是**融合**：大模型看懂世界靠 ViT 基座，跑进车间和手机靠卷积，最强的系统往往两者都要。而本讲的残差捷径、1×1 省钱术、多尺度并行，依旧是这场融合里的通用语法——学它们不是学历史，是学现在进行时。

ImageNet

李飞飞团队 2009 年起构建的千万级标注图像数据库；ILSVRC 竞赛（2010–2017）用它 1000 类、约 140 万张图的子集，是深度学习的“高考考场”。

top-5 错误率

每张图给 5 个候选类、正确答案不在其中才算错的计分方式；ILSVRC 的官方指标。

AlexNet

2012 年冠军：8 层、6000 万参数，靠 ReLU、Dropout、GPU 训练三大法宝引爆深度学习革命。

ReLU

$f(z) = \max(0, z)$；不饱和、算得快，AlexNet 之后 CNN 的默认激活函数。

Dropout

训练时随机丢弃部分神经元以抑制过拟合的技巧；推理时关闭。

感受野（receptive field）

某层神经元能“看见”的输入区域大小；随层数加深逐层扩大。

1×1 卷积

不扫空间、只在通道间做加权组合的卷积；用作瓶颈降维省算力，是轻量化标配。

Inception 模块

GoogLeNet 的积木：多尺度卷积与池化并行再拼接，让网络自选该信哪个尺度。

退化（degradation）

普通深网络训练误差反而高于浅网络的现象；源于深网络难优化，而非过拟合或梯度消失。

残差连接（residual connection）

输出 = 主路计算结果 + 输入的捷径相加结构；让网络学“修改量”，梯度可沿捷径直通。

恒等捷径（identity shortcut）

不做任何变换、把输入原样抄到输出端相加的那条边；ResNet 的灵魂。

ResNet

何恺明等 2015 年提出的残差网络，最深 152 层，以 3.57% top-5 错误率首次超越人类水平。

第 8 讲　轻量化网络、全卷积网络与卷积操作模式设计

先把两个数字摆在一起感受一下。VGG-16，第 7 讲的功勋网络：1.38 亿参数，按最常用的 FP32 格式存成文件超过 500 MB——而一部普通手机的运行内存是 8 GB，还要留给系统、微信和相机进程——**一个分类模型就要吃掉小半 GB 的内存预算**，更别说再塞一个检测器、一个分割模型。第二个数字：产线上的工业相机一般按每秒 30 帧出图，那就是**每 33 毫秒必须出一次结果**——这段时间里要完成图像搬运、网络前向、结果后处理，留给卷积的预算只有十几毫秒；而一块 RTX 4090 的功耗超过 400 瓦，你手机充电器总共才 20 瓦。**大，是服务器上的特权；小，才是落地的入场券**。本讲就讲“小而美”的手艺：怎么把网络天生设计得小（深度可分离卷积）、怎么把大网络压小（压缩四板斧）、怎么让网络输出到每个像素（全卷积与语义分割），以及怎么把它真正装进设备（ONNX 与推理引擎）——最后一节，就是你们项目交付格式的说明书。

本讲地图

讲什么：模型压缩四板斧（剪枝/量化/蒸馏/轻量设计）、深度可分离卷积与 MobileNet、其他轻量家族速览、全卷积网络与语义分割、卷积操作模式工具箱（空洞/分组/可变形/转置）、端侧部署三件套（ONNX/推理引擎/INT8）。**为什么值得学**：六目九头是“几十 KB 的小网络”、组 8 的量化对比、数据基座的 llama.cpp 量化部署、实验二的本地部署——全在本讲。**学完你能**：手算深度可分离卷积省了多少；读懂一份“模型体检报告”；说清 FP32/FP16/INT8 的取舍与 ONNX 在交付链中的位置。**课后阅读**：许庆阳《深度学习模型与算法基础》第 101–115 页。

8.1　模型压缩四板斧

一个训好的网络，体积来自两个数：参数量和每个参数的存储精度。压缩的一切手段都是在这两个数上做文章，外加一条“从源头就小”的路线。江湖俗称四板斧：

表 8.1　模型压缩四板斧总览：目标一致，路径各异，实战中常常组合使用

板斧	一句话原理	典型收益	代价/风险
剪枝	把不重要的权重或通道置零删掉	参数减半到十倍级	需重训恢复精度
量化	FP32 → INT8，精度换体积	体积 1/4，还能加速	少数样本上精度下滑
知识蒸馏	大老师教小学生	小模型逼近大模型	训练流程变长
轻量设计	结构上天生省	从根源省算力	单体精度略降

剪枝（pruning）像修剪果树：训练完发现大量权重接近零、砍掉也不怎么影响输出，就把它们（或整个通道）删掉。**量化**（quantization）把每个参数从 32 位浮点压到 8 位整数，体积立省 75%，还能借处理器的整数指令提速——8.6 节重点讲。**知识蒸馏**（knowledge distillation）让一个小“学生”模型去模仿大“教师”模型的输出分布，学到的常比直接用硬标签更多——前沿速递盒子里有它的 2026 年新剧情。**轻量结构设计**则是从图纸阶段就按“手机预算”画楼，主角是下一节。下图的散点把常见模型按“大小 vs 精度”铺开，左下角那片“小而准”的区域，就是本讲全部手艺要守住的

阵地。

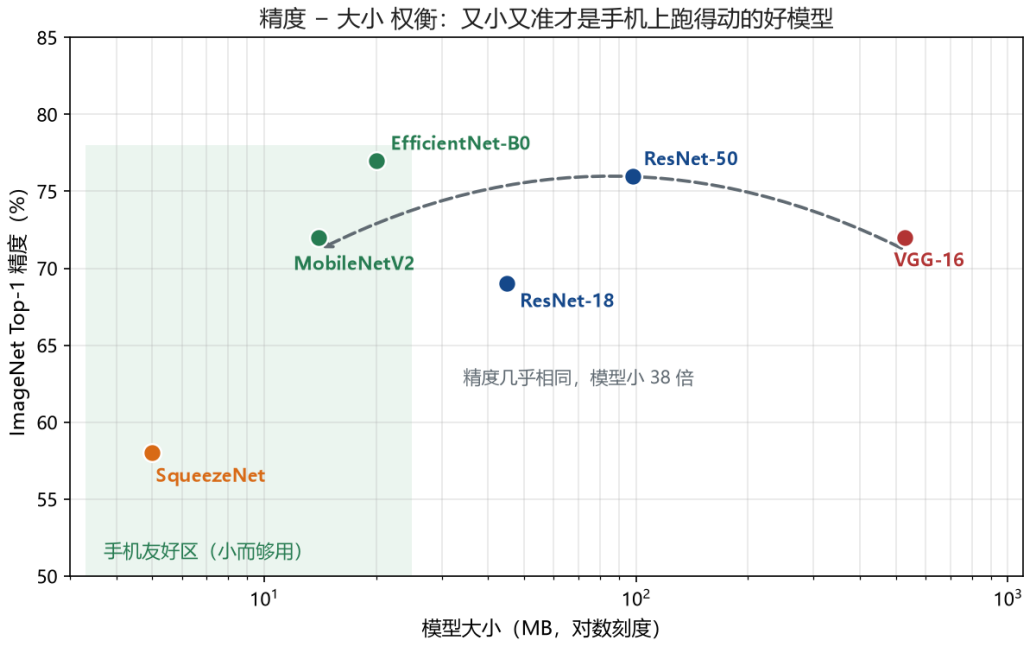


图 8.1　图题：模型大小与精度散点图：VGG、ResNet、MobileNet 等各居其位，左下方圈出“手机可跑区”——轻量化的目标是把精度曲线往左下推

8.2　深度可分离卷积：轻量化的核心积木

四板斧里，对你们影响最深远的是第四板斧，而它的核心积木只有一个：**深度可分离卷积**（depthwise separable convolution）。先回忆标准卷积在干什么（第 5 讲）：一个 $3 \times 3 \times C_{in}$ 的核，同时干两件事——在**空间**上扫描（左右上下找纹样），在**通道**间混合（把 64 个通道揉成 1 个）。两件事捆绑销售，每个输出像素都要付 $3 \times 3 \times C_{in}$ 的价钱，乘上 C_{out} 个输出通道，账单巨大。

拆开卖。深度可分离卷积把这一步拆成两小步：**逐通道卷积**（depthwise convolution）——每个通道自己卷自己的 3×3，只在空间上扫描、通道间互不往来；**逐点卷积**（pointwise convolution）——1×1 卷积（第 7 讲见过的老朋友）负责把逐通道卷出的特征在通道间混合。下图把两笔账并排摆开：

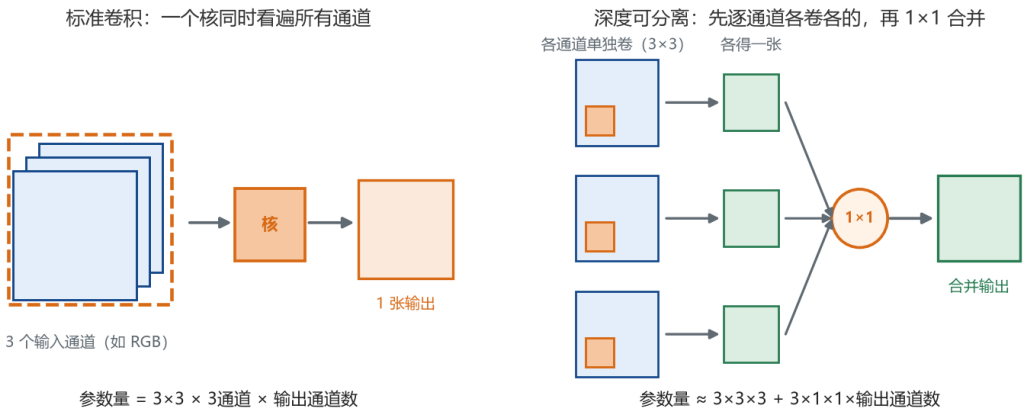


图 8.2　图题：标准卷积 vs 深度可分离卷积：同样“空间扫描 + 通道混合”，拆成先逐通道再逐点两步后，计算量降到约八分之一

算一笔账感受差距（这正是本讲课后题的手法）。设输入 64 通道、输出 64 通道、特征图 56×56、核 3×3：

$$\text{标准：}\quad 56\times56\times(3\times3\times64)\times64\approx1.16\times10^{8} \tag{8.1}$$

$$\text{拆分：}\quad \underbrace{56\times56\times(3\times3\times64)}_{\text{逐通道}}+\underbrace{56\times56\times(1\times1\times64)\times64}_{\text{逐点}}\approx1.5\times10^{7} \tag{8.2}$$

人话翻译：同样的活儿，拆开干只花约**八分之一**的乘加次数。通式更漂亮：当输入输出通道数都是 N 时，拆分版的计算量约为标准版的 $\frac{1}{N}+\frac{1}{9}$——通道越多越赚，极限逼近九分之一。

2017 年，Google 把这块积木砌成整栋楼，就是 **MobileNet V1**：420 万参数、17 MB 的文件，ImageNet 精度 71.7%——第一次让“手机上跑像样的 CNN”不再是口号。2018 年的 V2 补上关键一刀：**倒残差**（inverted residual）+ 线性瓶颈——把第 7 讲 ResNet 的“宽–窄”残差块反过来，做成“窄–宽–窄”的沙漏：先在窄通道间加捷径，中间用逐通道卷积在膨胀后的宽通道里提特征，最后压缩。宽度上的钱省了，残差的高速公路还在。

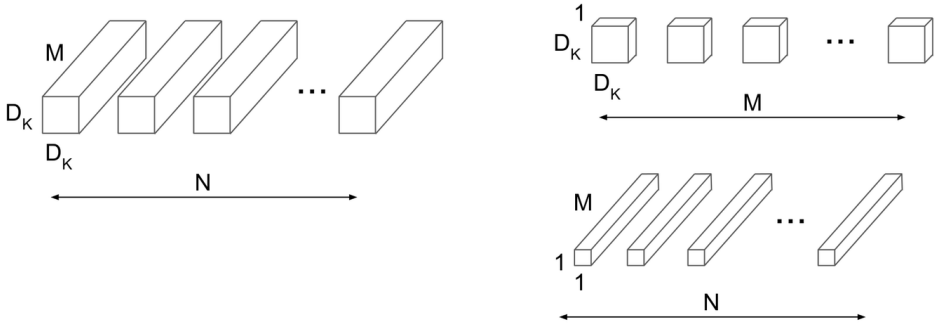


图 8.3　MobileNet 论文图：标准卷积（上）与深度可分离卷积（下）的模块对比——先逐通道扫空间、再逐点混通道，计算量大幅下降

图片来源：Howard et al., MobileNets: Efficient CNNs for Mobile Vision Applications, 2017

动手时间：替你的项目算一笔“卷积账”

六目项目九个参数头都是小网络，选型时这笔账要会算。设输入 32 通道、输出 128 通道、特征图 28×28、核 3×3：**(1)** 标准卷积的乘加次数是多少？**(2)** 深度可分离卷积（逐通道 + 逐点）呢？**(3)** 比值是多少？更接近 1/8 还是 1/9？**(4) 想一想**：若输出通道翻倍到 256，比值怎么变？为什么通道越多越划算？——答案：(1) $28\times28\times9\times32\times128\approx2.9\times10^7$；(2) $28\times28\times9\times32+28\times28\times32\times128\approx3.4\times10^6$；(3) 约 0.12，介于 1/8 与 1/9 之间、更近 1/8；(4) 比值降到约 0.113——N 越大，$1/N$ 越小，比值越贴着下限 1/9。

8.3　轻量家族一分钟点名

MobileNet之外，轻量家族还有几位常客，一表速览——不求都记住，混个脸熟，将来选型时知道去翻谁：

表 8.2　轻量化网络家族速览：各家的“省”法不一样

家族	省在哪	一句话点评
SqueezeNet	先 1×1 压通道再 3×3	2016 年老前辈，AlexNet 精度、五十分之一体积
ShuffleNet	分组卷积 + 通道混洗	组间老死不相往来？洗个牌互通信息
GhostNet	一半特征用廉价变换生成	卷出来的一半是“影子”，影子不必真卷
NAS 自动设计	让算法搜结构	不人肉画楼，把设计目标写成搜索题

表的第一行 SqueezeNet（2016）值得单独两句：它证明了一件震撼业界的事——**AlexNet 的精度，可以用约五十分之一的参数做到**，权重文件仅 4.8 MB。它的积木叫 **fire 模块**：先用一排 1×1 卷积把通道**压**薄（squeeze，挤一下），再用 1×1 与 3×3 两路**并联展开**通道（expand，撑开）——一挤一撑之间，卷积核统统做小，VGG 式的肥大全连接层则根本不设。它没有发明任何新算子，全靠把第 7 讲的 1×1 省钱术用到极致，是“轻量结构设计”思想的第一次完整示范。

其中两个概念值得多说一句。**分组卷积**（group convolution）：把输入通道分成 g 组，每组各自卷各自、互不往来，计算量直接除以 g——它最早出现在 AlexNet，不过是当年两块显卡各存一半通道的无奈之举，如今成了轻量化的正经武器；代价是组与组之间信息不通，于是 ShuffleNet 在分组卷积之间插入“通道混洗”（channel shuffle），把各组的通道打乱重排，让下一轮分组时信息交叉——好比分组讨论到一半强制换一次座位，保证信息在组间流通；靠着“分组 + 洗牌”这两手，ShuffleNet 在精度相当时把计算量压到 MobileNet V1 的几分之一。**NAS**（神经架构搜索，neural architecture search）：干脆不人肉设计，把“精度高、延迟低”写成搜索目标，让算法在海量候选结构里自动淘金——2019 年上线的手机端模型不少出自这类“硬件感知”的自动搜索。你会发现轻量化的思路高度一致：**减少冗余计算，让每一分算力都花在刀刃上**。

8.4　全卷积网络 FCN：从“是什么”到“在哪儿”

到目前为止，网络都在回答“这张图**是什么**”。但自动驾驶要知道“可行驶路面**在哪儿**”，医学影像要知道“肿瘤边界**在哪儿**”，锚链检查要知道“缺陷**在**哪个链环**上**”——这类任务叫**语义分割**（semantic segmentation）：**给图里每一个像素发一个类别标签**，相当于把“一个标签”变成“几十万个标签”。

难度立刻暴露：第 6 讲户型图的末端是几层全连接——它们把特征图拉平、揉成 10 个数，**空间位置信息全丢了**，输出尺寸也焊死成类别数，根本到不了“每像素一个标签”。2015 年 Long、Shelhamer 与 Darrell 的 FCN 论文给出漂亮的手术方案，两步：

第一步，全连接换卷积。把分类头里的全连接层换成 1×1 卷积（又是它！），网络末端输出的就不再是 10 个数，而是一张与输入位置一一对齐的**热图**（heatmap）——每个像素位置上放着各类别的得分。这种“从头到尾只有卷积层、输入输出都是空间图”的网络，就叫**全卷积网络**（Fully Convolutional Network，FCN）。它还有个可爱性质：不挑输入尺寸，图多大热图就多大。**第二步，上采样回原尺寸**。一路卷积池化下来，热图只有原图的几十分之一大，需要**上采样**（upsampling）放大回原尺寸、逐像素对齐——放大手段从简单到高级有最近邻插值、双线性插值和可学习的**转置卷积**（transposed convolution，俗称反卷积，8.5 节再表）。两步合起来，一个端到端的“图进、像素级标签出”的分割网络就诞生了。

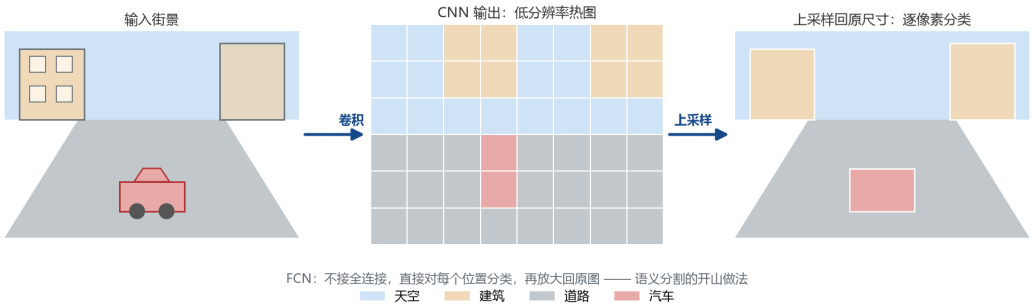


图 8.4　图题：FCN 分割示意：输入图像经 CNN 得到低分辨率热图，再上采样回原尺寸，输出与原图对齐的彩色分割图（每个像素一个类别色）

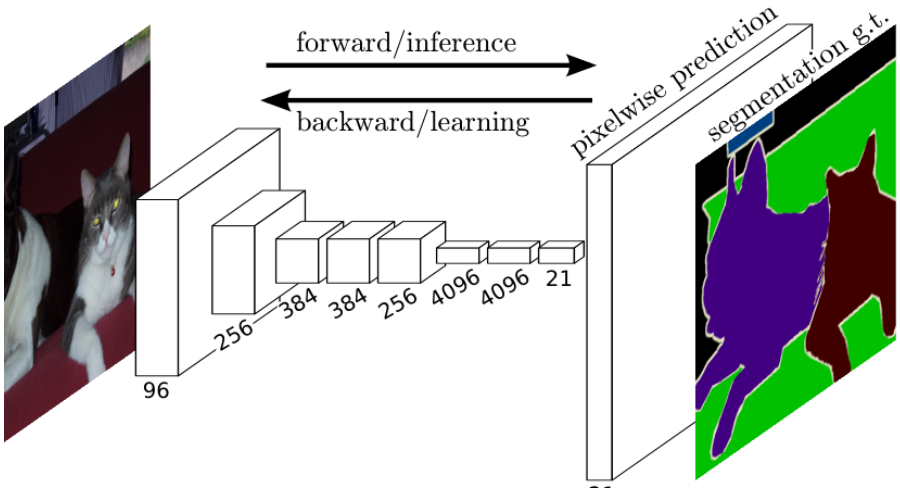


图 8.5　FCN 论文分割结果图：输入照片、真值标注与 FCN 预测的分割结果对照——粗糙但成体系，语义分割从此端到端

图片来源：Long, Shelhamer & Darrell, Fully Convolutional Networks, CVPR 2015

原论文里还有点睛的第三笔：**跳级融合**（skip fusion）。直接把最深层的热图放大 32 倍（论文称 FCN-32s），边界粗得像马赛克——深层热图“懂语义”却“丢细节”。于是 FCN 把**浅层**特征图（细节丰富）也接了上来：深层热图先上采样，与浅层特征图**逐点相加**，再继续放大——FCN-16s 融一次、FCN-8s 融两次，分割边界随之越来越细。**浅层管细节、深层管语义，一次相加把两种优势合并**；这记“跳跃连接”的下文马上就到——下一段的 U-Net 会把它用成传奇。

FCN 之后，分割精度靠两个方向持续抬高。一是**看得更大更全**：DeepLab 系列（v3+，2018）用空洞卷积扩大感受野、用多尺度池化并行捕捉大小物体，长期是分割标杆（效果见下图，典型的自动驾驶街景风格）；医学影像领域的 U-Net 则靠“编码–解码 + 跳跃连接”结构成为另一个传奇——你又在跳跃连接，ResNet 的幽灵无处不在。二是**见得更多**：这正是 SAM 登场的剧情，见下下段。分割应用一眼望去：自动驾驶的可行驶区域与车道线、医学影像的器官与病灶勾画、卫星图的建筑物提取、产线质检的缺陷区域框定——**凡是“位置本身就是答案”的任务，都是 FCN 的子孙**。

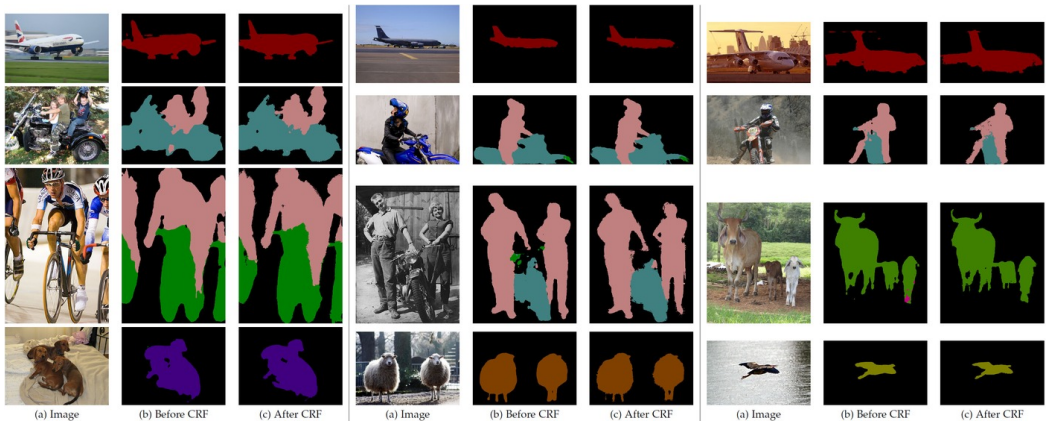


图 8.6　DeepLab 语义分割效果图：街景中行人、车辆、道路、天空被逐像素上色——每个像素都有自己的类别标签

图片来源：DeepLab v3+, Chen et al., 2018

2023 年，Meta 发布 **SAM**（Segment Anything，“分割一切”），把分割推入基础模型时代：给它一张新图，你点一个点、框一个框，它就瞬间把对应物体精确抠出——**零样本**，不需要为你的领域重新训练。支撑这个魔法的是一个史无前例的数据集：1100 万张图、11 亿个掩码。它对你还有一个具体的意味：**标注行业的工作方式变了**——过去人工抠一个物体要几分钟，现在 SAM 打初稿、人只做修边检查，效率翻数倍；你水下项目（第 二 章）调研中的“自动标注”路线，走的就是“大模型打初稿 + 人审核（或教师模型打伪标签）”这条线索。

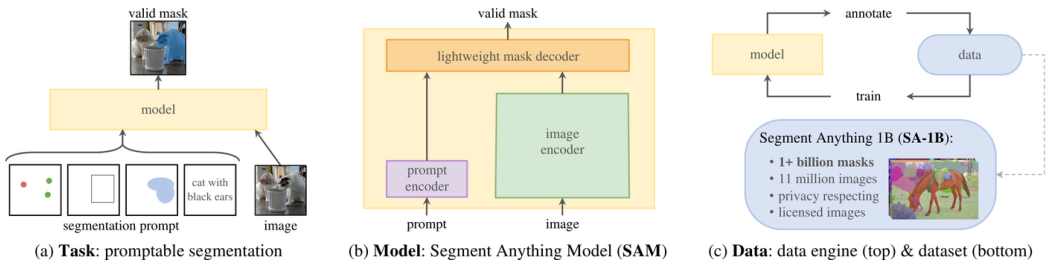


图 8.7　SAM（Segment Anything）示例：用户在图上点几个点（或框一个框），模型零样本分割出指定物体，边缘精确到发丝

图片来源：Meta AI, Segment Anything 项目页, 2023

扫盲：分类、检测、分割——三个任务的差别

同一张街景图，三个任务三种输出：**分类**输出一个词（“街景”）；**检测**输出若干个框，每个框一个类别（“此处有一车、此处有一人”）；**分割**输出一张与原图同尺寸的“涂色图”，每个像素都有自己的类别。信息量依次暴涨，标注成本也一样——分类一张图一个标签，分割一张图几十万张“小标签”。所以分割数据集总是又小又贵，这也是 SAM 这类零样本分割工具对行业冲击巨大的原因。

8.5　卷积操作模式工具箱

走到这里，值得把“卷积”这个词底下的各种玩法收进一个工具箱。它们都是 8.2 节“拆解与重组”思想的不同方向：

表 8.3　卷积操作模式速查表：改哪个自由度，换什么能力

模式	改的是	得到什么	代表应用
标准卷积	——	基线	LeNet/VGG全家
深度可分离	通道混合与空间扫描解耦	省约 1/8 计算量	MobileNet（8.2 节）
分组卷积	通道分组各卷各的	计算量 $/g$	AlexNet/ShuffleNet
空洞卷积	核内插“孔”不加大核	感受野翻倍、参数不变	DeepLab 分割
可变形卷积	采样位置可学习	贴合物体形变	实例分割/姿态
转置卷积	反向的卷积	可学习的上采样	FCN/生成模型

空洞卷积（dilated/atrous convolution）最值得展开：把 3×3 核的九个采样点彼此拉开距离（中间插“孔”，间距称膨胀率 rate），核还是九个参数，视野却从 3×3 扩到 5×5（rate=2）——**不花一个新参数，白得大感受野**。它解决了分割任务的一个死结：分割网络常去掉了池化层（保住分辨率），于是感受野缩水，空洞卷积正好不靠池化就能把视野补回来。**可变形卷积**（deformable convolution）则更进一步：连“采样点摆在哪”都交给网络学，卷积窗口随物体的形状扭曲——猫的尾巴细长，采样点就顺势拉长。机制上，它额外挂一支小型卷积网络，为每个采样点预测一个位移量，整套位移合称**偏移场**（offset field）——连窗口的形状本身都成了可学习的参数。**转置卷积**（transposed convolution）是 FCN 用过的上采样招式：普通卷积把大图卷小，它把小图“逆卷”回大图，且放大方式可学习。看到规律了吗？**卷积不再是一个固定的算子，而是一个可设计的部件家族**——这正是本章标题里“卷积操作模式设计”的含义，也是工业界大量专利与论文的聚集地。

工具箱还该补上 2020 年代最有戏剧性的一章：**大核卷积回潮**。第 7 讲的 VGG 算术——“小核堆叠完胜大核”——统治了此后七年，几乎成了教条；2022 年，回旋镖结结实实飞了回来。清华团队的 **RepLKNet**（CVPR 2022）把卷积核一口气开到 31×31，配合**结构重参数化**（训练时用多条小分支搭出等效结构以稳住训练，部署时全部折叠成一个真正的大核，推理时不多花一分钱），在多项下游任务上明显胜过同体量的小核网络；同期的 **SLaK** 更把稀疏布置的大核推到 51×51。大核带来的不只是感受野的数字，而是**“一眼看到整个物体”的归纳偏置**，代价是必须靠重参数化与精心安排才跑得快。给初学者的启示并非“大核更好”，而是：**算术上的最优解未必是工程上的最优解，任何教条都值得用实验重新问一遍**。

另一个必须点名的是 **SENet**（Squeeze-and-Excitation，2017 年 ImageNet 分类冠军），它给工具箱添上了**通道注意力**（channel attention），三步走：先对每张特征图做全局平均池化，把“这个通道整体有多活跃”压成一个数；再过两层全连接（先收缩、再放宽，中间夹激活），输出每个通道的重要性权重；最后把权重逐通道乘回特征图——**让网络自己决定哪个通道重要**：重要的放大，没用的压暗。整套 SE 模块几乎不加参数、即插即用，今天大量骨干网络里仍躺着它的变体。用本讲的语言总结：深度可分离卷积拆出了“空间”与“通道”两个自由度，SENet 给“通道”这个自由度装上了看内容定权重的智能开关。

8.6　端侧部署三件套：ONNX、推理引擎与 INT8

网络训好只是半场，**部署**（deployment）才是把模型真正变成产品的那半场——也是你们实验二（11 月 2 日）“本地部署”的考试范围。三件套按数据流顺序登场。

第一件：ONNX（Open Neural Network Exchange，开放神经网络交换格式）。PyTorch、TensorFlow 各说各的方言，直接把模型交给部署端就像把 `.doc` 发给只有 WPS 的同事。ONNX 是行业的“通用普通话”：把网络翻译成一份标准的**计算图**文件（图上每个节点是一个算子、每条边是数据），任何支持它的运行时都能执行。`torch.onnx.export` 一行即可导出。**你们六目项目的九个参数头与端到端 `ropstitch.onnx`，交付格式就是它**——头文件配同名 `metrics.json`，拷入 `build/Release/models/` 即被系统识别。**第二件：推理引擎**。拿到 ONNX 后，谁在设备上执行这张图？**推理引擎**（inference engine）：NVIDIA 显卡上的 TensorRT 会做算子融合、自动半精度等图级优化；手机 CPU 上的 NCNN（腾讯开源）零依赖、对 ARM 指令深度优化；同门还有 TFLite、MNN 等。选引擎先选硬件，这在实验二里你们会亲手体会。**第三件：INT8 量化**，压轴戏，先用知识盒补一个前置概念。

知识补齐：FP32 / FP16 / INT8——三种记账精度

计算机存数靠位数，位数就是“账本精细度”。**FP32**（32 位浮点）：1 位符号 + 8 位指数 + 23 位尾数，约 7 位有效数字——像精确到分的账本，训练默认用它。**FP16**（16 位浮点）：1+5+10 位，约 3 位有效数字——账本只记到毛票，但体积减半、很多芯片算它反而更快。**INT8**（8 位整数）：只能记 -128 到 127 的整数——像只划“正”字的计分板。那小数怎么办？**量化的本质**：先量出这批数的范围 $[\min, \max]$，把范围等分成 255 格，每个数只记“落在第几格”，另存一个缩放系数（scale）负责换算回来。精度损失来自“格子变粗”，收益是体积 1/4、整数指令提速、内存带宽省 4 倍。

量化不是免费午餐，权衡如下表。分类任务的实操经验：训练完成后用几百张“校准图”统计范围、直接转 INT8（称**训练后量化**，PTQ），精度往往只掉零点几个百分点；但任务越精密（回归、立体匹配这类输出连续值的任务），INT8 越容易掉链子，必要时得用**量化感知训练**（QAT）——训练阶段就“假装”自己是 INT8，让网络提前适应粗格子。

表 8.4　三种精度档次的典型权衡（分类任务的量级参考，具体任务必须实测）

精度	体积	速度	精度损失	适用
FP32	1×	基线	0	训练与精度基准
FP16	0.5×	常快于 FP32	几乎无	GPU部署首选
INT8	0.25×	CPU/GPU 均显著提速	通常 <1%，个别任务更大	产线/端侧大批量

所以工程上的铁律是：**量化必实测**——在部署机上用真实任务指标验收，而不是相信“通常掉不到 1%”。你们项目的验收线就是这条铁律的定量版：组 8 的量化版 onnx，重投影误差劣化必须小于 10% 才允许交付。

前沿速递：端侧 AI 大爆发——大模型正在塞进你的手机

本讲的手艺正在经历史上最大的一次应用浪潮。**端侧大模型**：2024–2026年，手机厂商与芯片厂把量化后的大语言模型直接装进旗舰机（Google 的 Gemini Nano、苹果的 Apple Intelligence、各家端侧版 Qwen 等），本地就能做摘要、改写、识图——不联网、不泄密、零延迟。支撑它的正是“蒸馏 + 量化”双板斧：几十亿参数的教师蒸馏成 7B 上下的小模型，再经 4 比特量化（GGUF

格式）压到 4 GB 上下，8 GB 内存的手机就装得下；开源社区 `llama.cpp` 一套代码，把量化模型跑在手机、笔记本、开发板上——你们数据基座项目在本地用 llama.cpp 量化部署 Gemma 12B，走的就是这条路。同时，NCNN/MNN/TFLite 构成的端侧推理生态日臻成熟，“手机跑 CV 模型”已是无传感器厂商的标配。更值得年轻人注意的趋势是**知识蒸馏 2.0**：教师不再只是更大的分类网络，而是大语言模型本身——用它海量的“软输出”去教一个专用小模型。你们水下项目“FoundationStereo 当教师、UWStereo 当学生”的伪标签闭环（第 二 章），方法论上与它同宗同源。**本讲学的压缩与部署手艺，是 AI 从机房走进世界的必经窄门。**

项目连线：第 8 讲 × 三个项目——这一讲就是你们的“部署说明书”

六目拼接（第 一 章）：九个参数头每个都是 ≤ 几十 KB 的轻量小网络，CPU 上实时跑——“轻量”在这里不是选修课，是入场券；8.6 节的 ONNX + `metrics.json` 正是它们的交付格式。组 8（端到端优化组）的本职工作就是本讲 8.6 节的完整实操：把 `ropstitch.onnx` 在部署机上做 **FP32/FP16/INT8 三档**的耗时与重投影误差对比，**INT8 劣化 <10% 才交付量化版**，否则回退 FP32——把本讲的权衡表变成一张真实的验收表。**数据基座**（第 三 章）：Gemma 12B 在本地用 llama.cpp 做量化部署（GPU、上下文 8192、并发 1），量化档位的选择就是精度–显存的实时权衡——大模型的轻量化，与本讲同一套思想。**水下双目**（第 二 章）：部署组在测显存与吞吐、做 FP16 与 ONNX 导出初探时，用的全是本讲工具箱。最后划重点：**实验二（11 月 2 日）“匹配合适程序并执行本地部署”就是本讲的实操考场**——ONNX、推理引擎、量化、llama.cpp 都会在那场实验里与你见面，请带着本讲去上课。

本讲课后题：给你的模型做一次“体检”，把“轻量”从口号变成四个具体的数字。五要素如下。

第 8 讲课后编程题（用 AI 实现）

题目：指挥 AI 辅助完成一份“模型体检报告”：取 torchvision 自带的两个预训练模型 `resnet18` 与 `mobilenet_v2`，在一个猫狗二分类数据集上（自备几百张猫狗图，或用 CIFAR-10 的猫/狗子集，替换分类头后各微调 2 个 epoch）分别测量四项指标：(1) 参数量（百万）；(2) 权重文件大小（MB）；(3) CPU 上单张图片的平均推理耗时（测 20 张取平均）；(4) 测试集准确率（%）。把结果整理成一张对比表，并写 50 字以上的选型建议：手机端产线质检该选谁？服务器端呢？

交付物：(1) 代码文件 `model_checkup.py`；(2) 四项指标对比表截图；(3) 50 字以上的选型建议与“我改了什么”说明。

提示词建议（可直接抄）：“我是零基础 Python 学生。请写一个脚本比较 torchvision 预训练的 resnet18 与 mobilenet_v2：(1) 用 sum(p.numel() for p in model.parameters()) 统计参数量；(2) 用 torch.save 保存权重后用 os.path.getsize 报告文件 MB 数；(3) 把模型放到 CPU，用 time.time() 计时对同一张图片连续推理 20 次取平均耗时；(4) 在我提供的猫狗图文件夹上微调 2 个 epoch（batch size 32、学习率 0.001、Adam）后在测试集上算准确率。最后用 print 输出一张对齐的四行对比表；每行代码加中文注释；不要用 GPU。”

验收线：四项指标齐全且来自你自己电脑的真实运行；能不看资料解释“MobileNetV2 为什么更小更快、代价是什么”（提示：8.2 节那笔账）；至少有一处自己的修改（换第三个模型、换输入尺寸、换数据集都算）。**代码是 AI 写的没关系，讲不清楚就是没做。**

加分项：把其中一个模型导出为 ONNX（`torch.onnx.export`），用 `onnxruntime` 在 CPU 上重测推理耗时，与 PyTorch 原生对比——谁更快？差多少？用一两句话解释推理引擎做了什么优化。

进一步可尝试 FP16 权重再测体积与耗时。

8.7　从这一讲到 2026

2026 年的“轻量化”三个字已经换了含义：不再是“把小模型剪到能跑”，而是**“把大模型压进手机”**。五条硬事实，按“架构—引擎—量化”三层摆开。

架构层。MobileNetV4 最大的 Hybrid-Large 款在 Pixel 8 的 EdgeTPU 上 **3.8 毫秒**完成一次推理、ImageNet-1K 精度 87%，其面向移动加速器设计的 Mobile MQA 注意力带来 39% 提速，在手机 CPU/GPU/DSP、苹果神经引擎上近乎处处帕累托最优——“为每一种端侧芯片都高效”成了小模型的新标准。分割这边，MIT Han Lab 的 EfficientViT-SAM 用多尺度线性注意力换掉 SAM 的重型编码器，TensorRT 实测较 SAM-ViT-H **提速 48.9 倍**而精度不降，医疗版还拿下 CVPR 2024 笔记本医疗 SAM 挑战赛冠军——分割基础模型的端侧化宣告完成。

引擎层。阿里开源的推理引擎 MNN，核心库只有约 **800 KB**，却部署在淘宝、天猫、钉钉等 30 多个应用里，通过 MNN-LLM 与 MNN-Diffusion 让手机**本地**跑大语言模型和扩散模型，3.6.1 版更新增了高通 Hexagon NPU 后端——8.6 节那张“推理引擎”名单，如今的主战场已是大模型。

量化层。苹果的 Apple Intelligence 用约 **30 亿**参数的端侧基座在设备上运行，更大的模型留给私有云——手机 NPU 跑基座模型已是消费级现实。获 MLSys 2024 最佳论文奖的 **AWQ**（激活感知权重量化）证明：只需重点保护约 1% 的“显著权重”，就能放心做 W4A16 的 4 比特量化，其 TinyChat 引擎在桌面与移动 GPU 上较 FP16 提速 3 倍以上——**4 比特已成为端侧大模型的默认档位**，你数据基座项目里 llama.cpp 的量化档位选择（第 三 章），正是同一条技术路线的现场版。

一句话总结这三年：**轻量化的终局不是把小模型做小，而是把大模型做进口袋**——量化、蒸馏与端侧引擎，让手机 NPU 成了新的“显卡”。你在本讲学的每一招，都直接通向那道窄门。

模型压缩
让网络更小更快的各类技术总称；四板斧为剪枝、量化、知识蒸馏、轻量结构设计。

剪枝（pruning）
删除接近零或不重要的权重/通道以缩小模型；通常需微调恢复精度。

量化（quantization）
降低参数存储精度（如 FP32 转 INT8）换取四分之一体积与加速；需实测精度损失。

知识蒸馏（knowledge distillation）
让小模型模仿大模型的输出分布来学习，“大老师教小学生”。

深度可分离卷积（depthwise separable convolution）
逐通道卷积（管空间）+ 逐点 1×1 卷积（管通道）；计算量约为标准卷积的 $1/N+1/9$。

倒残差（inverted residual）
MobileNetV2 的“窄–宽–窄”残差块；捷径连在窄通道之间，中间膨胀后提特征。

分组卷积（group convolution）
输入通道分组各卷各的，计算量除以组数；ShuffleNet 用通道混洗打通组间信息。

语义分割（semantic segmentation）
给图中每个像素一个类别标签；输出是与原图对齐的“涂色图”。

全卷积网络（FCN）

用 1×1 卷积替换全连接、输出热图再加采样的网络；语义分割的端到端开端。

上采样（upsampling）

把小分辨率特征图放大回原尺寸的操作；插值或可学习的转置卷积。

空洞卷积（dilated convolution）

核内采样点拉开间距以扩大感受野，参数量不变；DeepLab 的主力。

转置卷积（transposed convolution）

实现“可学习放大”的算子，俗称反卷积；FCN 上采样的手段。

ONNX

开放神经网络交换格式：把模型导出为通用计算图文件；本项目所有头网络的交付格式。

推理引擎（inference engine）

在具体硬件上执行模型计算图的运行时（TensorRT、NCNN、TFLite 等）。

训练后量化（PTQ）/量化感知训练（QAT）

训完用校准集直接量化；或训练时就模拟量化让网络适应。前者省事，后者更保精度。

第 9 讲　循环神经网络 RNN

先做个一秒钟的小实验：打开手机输入法，打下“今天天气”，看它猜的下一个词——多半是“真不错”“怎么样”。这个你每天都在用的功能，背后站着本讲的主角：一个**边读边记**的网络。而在它之前，我们认识的全部网络——感知器（第 1 讲）、多层感知器（第 2 讲）、卷积神经网络（第 5 讲）——都有一条共同的隐含设定：**输入一次给全，答案当场给出**。它们像“一眼定终身”的裁判：照片整张递进来，结论直接吐出去，案子即告终结。没有“上一张照片”，也没有“下一个词”。

可惜真实世界的数据一多半不是这样递过来的。你读一句话是**一个字一个字**读的，读到“它”字要靠前文才知道它指谁；听语音是**一段一段**听到的；相机工作时的匹配数、增益是**一帧一帧**积累出来的。这类“有先后次序”的数据叫**序列**（sequence），对付它，网络必须带上记忆。1980 年代就有了答案的**循环神经网络**（Recurrent Neural Network，RNN），是给网络装上记忆的第一代成熟方案；更重要的是，它是你理解 2026 年一切序列模型（包括大模型）绕不开的地基——第 11 讲你会看到，大模型的看家本领“文字接龙”，原型正是本讲的 9.3 节。

本讲地图

讲什么：序列数据与时间步；RNN的核心思想（同一个网络反复用，隐藏状态当便签）；按时间展开；用 RNN 做语言模型、玩文字接龙；沿时间反向传播（BPTT）；梯度消失与爆炸；RNN 的应用与局限。**为什么值得学**：第 11 讲的大模型与三次动手实验（10 月 12 日 agent 配置、11 月 2 日本地部署、11 月 23 日离线训练）的主角都是“接龙高手”，接龙的源头在本讲；而相机状态、ROV 遥测这类小数据工业时序，2026 年仍是 RNN 家族的主场。**学完你能**：手算一步 RNN 状态传递；说清“权值共享在时间维”是什么意思；解释梯度为什么会连乘到消失；判断一个任务该不该上 RNN。

9.1　序列数据无处不在：世界是按帧来的

序列数据（sequential data）就是一串**次序本身携带信息**的数据：词的先后排成句意，帧的先后排成动作，采样的先后排成趋势。判断一类数据是不是序列，有个毒舌但有效的检验——**打乱次序，看信息还剩多少**。把一句话的字序打乱，剩下一堆乱码；把视频的帧序打乱，剩下一堆幻灯片；把心电图按时间乱排，医生当场报警。对比之下，把一张猫照片的像素行随机重排成另一张“图”，对“判断是不是猫”这个任务来说固然也是灾难——但猫图之间彼此独立，而序列里的元素彼此依赖：**每一个元素的含义，取决于它前面的元素**。

下表把四类最常见的序列摆在一起。注意“一个时间步是什么”这一列：同是序列，切法天差地别。

表 9.1　四类常见序列：一个时间步各是什么

序列类型	一个时间步	典型长度	打乱次序后剩什么
文本（一句话/一段话）	一个字或一个词	几十到几千步	乱码
语音	一小段声波（约 20 毫秒）	上万步	噪音
视频	一帧图像	几十到几千帧	幻灯片
传感器时序（温度/相机状态/遥测）	一次采样读数	几十到上百万点	趋势全丢

生活化类比：时间步——给序列装上一台滴答钟

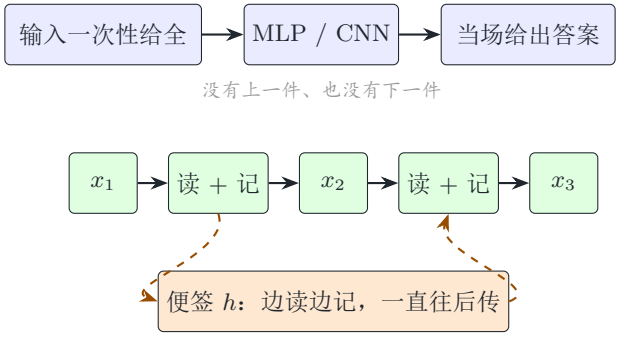
处理序列的第一步，是把连续的流切成一帧帧，切出来的每一份叫一个**时间步**（time step），用 t 编号：$t=1$ 是第一步，$t=2$ 是第二步……就像给流水装上一台滴答作响的钟，钟每响一下，网络吃进一份输入。顺带提前认识一个你在项目里天天会见的形状写法 $[B,T,D]$：B（batch）是一批样本数，T 是序列长度（时间步数），D 是每个时间步的特征维数——比如“一个批次的相机状态序列”就是 $[8,32,24]$：8 组样本、每组 32 步、每步 24 个数。以后见到张量形状，先数清这三个字母。

为什么无记忆的网络干不了这活儿？两个理由。其一，**单步看没有答案**：只看“它”一个字，神仙也不知道它指可丽饼还是指天气；答案藏在上下文里，而上下文就是“前面读过的所有步”。其二，**序列长度不定**：MLP和 CNN 的输入尺寸是固定的，而句子有长有短、音频有长有短，总不能每换一个长度就重造一个网络。于是需求非常明确：造一个**能一步一步吃输入、还记得自己吃过什么**的网络。

在 RNN 登场之前，人们怎么让机器“读懂”一句话？补一段前史，帮你看清 RNN 到底新在哪。第一条路线叫**词袋**（bag of words）：把句子拆成一袋词，只统计“哪些词出现过、各出现几次”，词的先后顺序整个扔掉——“我打了他”和“他打了我”在词袋里一模一样，谁动了谁无从谈起。第二条路线叫 **N-Gram**：学乖了一点，把相邻 N 个词的搭配存进一张大表——2-Gram 存“我昨天”“昨天上学”这样的词对，3-Gram 存“我昨天上学”这样的三连词组，预测下一个词就靠查表。可哪怕“我昨天上学迟到了”这样一句家常话，想接得准也已经要看三四个词的搭配；而理论上搭配的数目是**词表大小的 N 次方**——N 每加大 1，表中条目就要乘上一整个词表：2-Gram 尚可硬存，3-Gram 已是海量，N 再往上去，存储与统计双双**指数爆炸**。何况 N 再大也有限，N 个词之外的上下文它一概不知。两条路线的共同病根是**死记硬背**——背的是组合，不是规律。所以最终胜出的，是下一节这位带着记忆、边读边消化的选手。

9.2　RNN 的核心思想：一个网络用到底，一张便签传下去

RNN的设计朴素得惊人，两句话说完。**第一句：同一个网络反复用**——每个时间步都用同一套权重读一份新输入。**第二句：留一张便签当记忆**——网络维护一个向量，叫**隐藏状态**（hidden state），读每一步时先瞄一眼便签上记着什么，再把新读到的内容写进便签，然后传给下一步。下图把“无记忆”与“有记忆”两种工作方式摆在一起对比：上面一行看完就散，下面一行靠着那张橙色便签把信息一站站传下去。



写成公式只有一行。第 t 步，网络拿新输入 x_t 和旧便签 h_{t-1}，算出新便签 h_t：

$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b) \tag{9.1}$$

人话：新来的输入 x_t 乘矩阵 W_x（“这次来的事怎么记”），旧便签 h_{t-1} 乘矩阵 W_h（“以前记的事怎么留”），两股信息相加、再加偏置 b，最后过 tanh（第 2 讲激活函数之一）把每个数压进 -1 到 1——防止便签越写越爆。$t=1$ 时 h_0 取全零（一张空白便签）。若每步还要输出（比如猜下一个字），再接一层 $y_t = W_y h_t + b_y$ 即可。W_x、W_h、W_y 这几套矩阵，就是 RNN 要学的全部参数——注意它们**与序列长度无关**：句子再长，参数一个不多，这正是“同一个网络反复用”的底气。

把上面的循环在纸上按时间“拉开”，就是下图：同一个模块（灰色块，权重完全相同）被复制三份排开，隐藏状态像接力棒横向传递。这张“展开图”是理解 RNN 的一切钥匙——本讲的训练与病灶分析全在它上面进行。

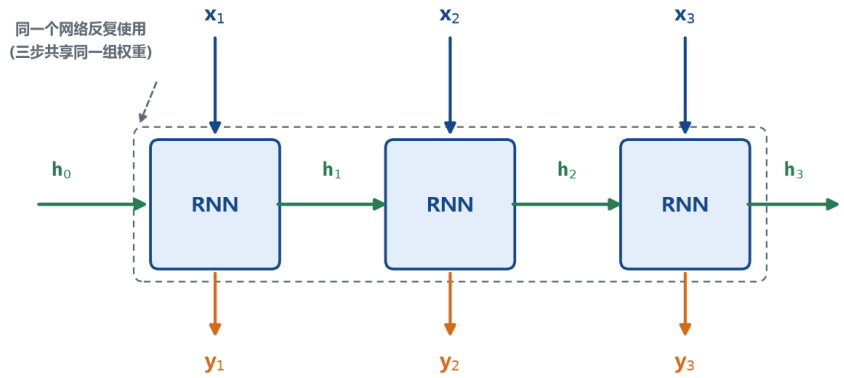


图 9.1　RNN 按时间展开：同一个模块复制三步，输入 x_1, x_2, x_3 逐步入场，隐藏状态 h 沿横向箭头一站站传递，每步可给出输出 y

同一个循环，Christopher Olah 2015 年那篇著名博客画得更写意——下面这张图后来成了 RNN 的“标准照”：左边的折叠形态里，网络块自带一条打弯的自环箭头；沿着时间拉开，就是右边的展开形态。看懂那条打弯的箭头，就看懂了“循环”二字的全部：

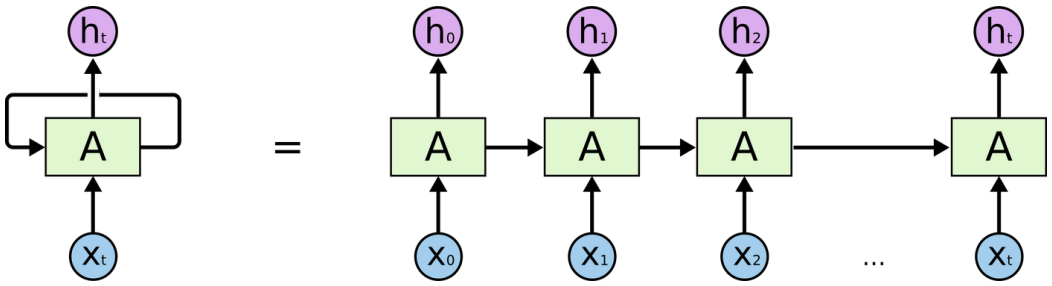


图 9.2　RNN 按时间展开的经典图：左侧折叠形态（模块带自环箭头），右侧为沿时间展开后的形态，隐藏状态逐站横向传递

图片来源：Christopher Olah, ”Understanding LSTM Networks” 博客文章, 2015

再品一层：**权值共享**。第 5 讲 CNN 的卷积核是“在空间维共享”——同一块模板扫遍全图每个位置；RNN 的 W_x, W_h 是“在时间维共享”——同一套“怎么读、怎么记”的规则用在第 1 步、第 7 步、第 100 步。两者的动机一模一样：**规律不挑位置**。猫耳长在图里哪个角落都还是猫耳，“主语到谓语”的语法距离十个字还是一百个字都成立。共享让参数量与数据尺寸脱钩，这是深度学习两大支柱（CNN 与 RNN）共用的记账智慧。

动手时间：亲手当一次 RNN——三步手算看“便签饱和”

取最简单的情形：x_t 和 h_t 都是单个数，$W_x=1$，$W_h=0.5$，$b=0$。连喂三个 1：$x_1=x_2=x_3=1$。拿草稿纸（或手机计算器）逐步算 $h_t=\tanh(1+0.5\,h_{t-1})$，tanh 值可查表或按 $\tanh(u)=\dfrac{e^{2u}-1}{e^{2u}+1}$ 现算：

- $h_1=\tanh(1+0)=0.76$；
- $h_2=\tanh(1+0.5\times 0.76)=\tanh(1.38)=0.88$；
- $h_3=\tanh(1+0.5\times 0.88)=\tanh(1.44)=0.89$。

盯着这串数看两件事：其一，三次输入完全相同，便签却一次比一次大——**新输入在不断覆写旧记忆**，h_1 里的“0.76”到 h_3 时只剩一点影子；其二，便签在向 1 饱和，tanh 的压缩让 h 装不下更多东西。再顺手用计算器算 $0.8^{20}\approx 0.012$、$1.2^{30}\approx 237$ 记在草稿边——9.5 节讲梯度消失与爆炸时，这两个数就是全部剧情。

收好草稿纸，最后认识 RNN 这块积木的两种经典拼法，各记一句话，第 10 讲在 PyTorch 里改一行参数就能打开。其一，**堆叠 RNN**（stacked RNN）：把两层乃至更多层 RNN 像积木一样竖着摞——下层逐字读出“字面”记忆，上层在下层的记忆基础上再加工，学更高层、更抽象的模式，与多层感知器（第 2 讲）层层堆出“从边缘到零件再到整脸”是同一个道理。其二，**双向 RNN**（bidirectional RNN，简称 Bi-RNN）：把序列正着读一遍得到一份记忆，再倒着读一遍得到另一份记忆，两份拼接起来用。它为什么有用？想想“完形填空”：只看左半句你只能蒙，前后文都看全才有把握——词性标注、命名实体识别这类“整句读完才好下结论”的任务里，双向几乎成了标配。

9.3　用 RNN 做语言模型：文字接龙

RNN最经典的应用是**语言模型**（language model）：给“接下来的字”算概率。玩法见下图：把一句话逐字喂进 RNN，每读一个字，网络的输出层（softmax，第 6 讲）就给出一张“下一字概率榜”——比如读完“今天天气”，榜上“真”得 0.31、“很”得 0.22、“不”得 0.09……训练目标简单到令人发指：**语料里下一个字是谁，就把榜上谁的分推高**。整句读完，每一步的预测误差加起来算损失，误差经反向传播把 W_x, W_h, W_y 修一点。

知识补齐：最小语言模型实例——一句话如何变成一道数学题

“给接下来的字算概率”说起来抽象，现在拿一句最短的语料（整个训练集就这一句）把它变成一道看得见的数学题：**“我每天要吃千层面”**，8 个字。**第一步，编词典**：给语料里每个字发编号——我 =0、每 =1、天 =2、要 =3、吃 =4、千 =5、层 =6、面 =7。**第二步，向量化**：把每个字变成 **one-hot 向量**——长度等于词典大小、只有自己那一格是 1 的向量（“吃”就是第 4 格为 1、其余全 0 的 8 维向量）。**第三步**：RNN 每读入一个字的 one-hot，softmax 就输出一张长度为 8 的概率榜，训练目标不变：榜上“下一个字”那一格的分被推高。训练时网络“吃的题”与“对的答案”一字排开，就是下表：

时间步	输入（字 → one-hot）	标签（下一个字）	这一步教会网络什么
$t=1$	我（第 0 格为 1）	每	“我”后面接“每”
$t=2$	每（第 1 格为 1）	天	“每”后面接“天”
$t=3$	天（第 2 格为 1）	要	“天”后面接“要”
…	…	…	…
$t=7$	层（第 6 格为 1）	面	“层”后面接“面”

one-hot 的毛病也当场现形：真实语料的词典动辄几万字，每个字就是一根几万维、只亮一格的“又高又瘦”的稀疏长条——浪费，而且任意两字“人人等距”，谁跟谁更像完全看不出。所以工程上 one-hot 只当临时工，转身就换成可学习的稠密词向量（embedding，第 11 讲的 Transformer 正式用它）。但骨架你已经全部看清：**词典编号 → 向量化 → 逐字预测下一字 → softmax 概率榜**——第 11 讲那些万亿参数的大模型玩“文字接龙”，第一天也是用这张 8 行小表练出来的童子功。

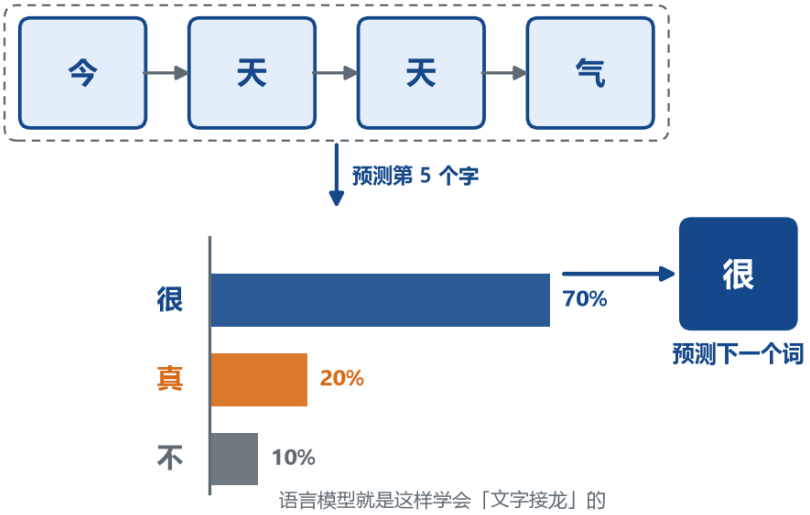


图 9.3　用 RNN 玩文字接龙：“今天/天气/很/好”逐字读入，每步输出一张“下一字”概率榜（右侧条形），榜上取一个字拼回去，接着猜再下一个

生成文字就是把接龙玩起来：榜上抽一个字，拼回输入，再猜下一个字。几行伪代码就能说清这个循环：

```
h = zeros                # 便签清零（h0 = 0）
for ch in "今天天气":     # 逐字读入序列
    h = rnn_step(ch, h)  # 每读一字：h = tanh(Wx*x + Wh*h + b)
scores = softmax(Wy @ h) # 便签给出"下一字"概率榜
next_ch = sample(scores) # 榜上抽一个字，比如 "真"
print(next_ch)           # 拼回去，接着接龙
```

第 3 行就是本讲主角公式，第 4 行把便签翻译成概率榜。请把**文字接龙**这个词用荧光笔标出来——它将一路贯穿到第 11 讲：GPT 的看家本领，本质仍是“读完上文，预测下一个词”，四十年未变的任务设定；变的只是接龙的网络（RNN 换成了 Transformer）与语料的规模。生成式预训练变换

器（Generative Pre-trained Transformer）这个名字里的 G（生成式），说的就是“从概率榜里采样接龙”这件事。你在这几行代码里看到的循环结构，就是 2026 年最强大模型的直系祖先。

顺带回答一个自然会冒出的疑问：采样时要不要总取榜一？总取榜一，生成的文字四平八稳且容易复读；偶尔按概率抽冷门字，文字才活。这个“抽签温度”的旋钮，留到第 11 讲再拧。

9.4　BPTT：把展开图当普通网络训练

训练 RNN 的思路就一句话：**把按时间展开的那张图当成一个普通深层网络，直接用第 2 讲的反向传播**。展开图里输入是 $x_1, \dots, x_T$，“层”就是一个个时间步，“深度”等于序列长度 T；唯一的区别是各层参数相同（共享的 W_x, W_h, W_y）。损失取每个时间步预测误差之和：$L = L_1 + L_2 + \cdots + L_T$（每步的 L_t 是交叉熵，第 6 讲）。这套流程有个专门的名字：**沿时间反向传播**（Backpropagation Through Time，BPTT）——听着唬人，展开图一画，其实就是老朋友反向传播换了个时间维的舞台。

对共享参数求梯度有个小讲究：W_h 出现在展开图的**每一层**，所以它的梯度是各时间步贡献的**总和**（$\partial L/\partial W_h = \sum_{t=1}^{T} \partial L_t/\partial W_h$），实现时把各步算出的梯度累加即可。真正的大坑在误差信号本身：要让第 T 步的预测错误“教育”到第 1 步的读法，误差得沿横向箭头倒着走 $T-1$ 步——而链式法则规定，每倒一步都要乘一次因子。

知识补齐：梯度为什么会连乘——链式法则又立功了

回忆开课导引（第 0 讲）里那张链式法则小图：$x \to u \to y$ 复合函数的导数，等于沿箭头把各段导数**相乘**。反向传播（第 2 讲）把这条规则用到整个网络上：误差从输出层倒回第 1 层，沿路逐段相乘。普通 20 层 MLP 倒回 20 段；而 RNN 展开后深度就是序列长度，一段 100 字的语料就是 100 层。更要命的是，RNN 每段乘的因子里都含同一个 W_h——倒走 k 步，就要乘 W_h 的 k 次幂的量级。**连乘**与**同一个数的幂**，这两个词凑在一起，指数级的放大或缩小就在路上了。9.5 节把这笔账算给你看。

9.5　梯度消失与爆炸：记忆的大敌

9.2节动手算的那两个数现在派上用场。误差信号从第 t 步倒回 k 步之前，幅值大约要乘 γ^k，其中 γ 是 W_h（连同导数项）的“平均放大倍数”。下图画出了两种命运：$\gamma > 1$ 时梯度随距离**指数爆炸**（$1.2^{30} \approx 237$ 倍，数值冲出浮点范围，训练直接变成 NaN）；$\gamma < 1$ 时梯度**指数消失**（$0.8^{20} \approx 0.012$，走 20 步只剩 1%，走 50 步连浮点数都懒得表示它）。

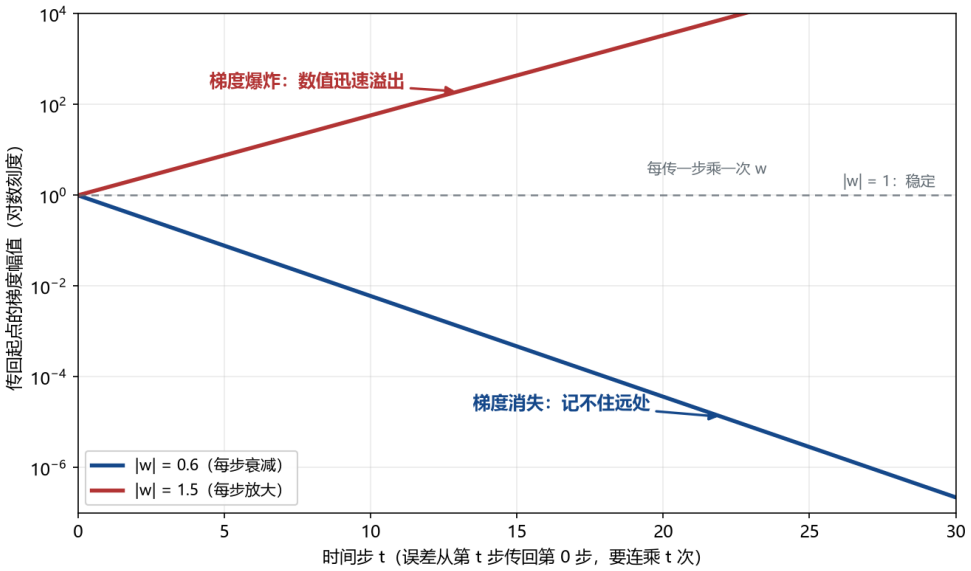
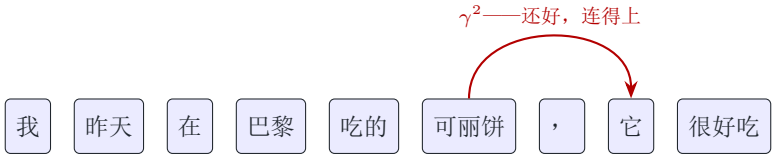


图 9.4　长序列上的两种命运：梯度幅值随回传步数指数变化——$\gamma > 1$ 连乘爆炸（扶摇直上），$\gamma < 1$ 连乘消失（归零）；横轴是误差要倒走的步数

两个病，一个轻一个重。**爆炸**好治，一句话的药方：**梯度裁剪**（gradient clipping）——每次更新前看一眼梯度向量的长度，超过阈值就按比例缩短，方向不变。相当于给误差信号装了个限流阀，工程上一行代码的事。**消失**难治，而且它砸的正是 RNN 的立身之本——记忆。“消失”翻译成人话：**网络记不住远处的词**。看这句话：



“它”指谁？指可丽饼。人读一遍就懂，因为你的短时记忆把“可丽饼”稳稳托到了“它”出现的那一刻。RNN 想学会这条规则，训练时误差就得从“它很好吃”倒着传回“可丽饼”，把“该把可丽饼写进便签”的读法奖励一遍。本例只隔 2 步，γ^2 还撑得住；可真实文本里指代隔十几步、时序数据里的因果关系隔几十步是常事——按 0.8 连乘，20 步后误差信号只剩约 1%，50 步后约 10^{-5}：**到不了，等于没学**。这解释了 9.3 节那个现象的深层版：字符 RNN 接龙超过几十字就忘了开头设定，不是它懒，是梯度根本走不到那么远。

病根还有一个帮凶：唯一的便签 h_t 每一步都被整页重写（9.2 节手算你亲眼见了），旧信息每过一站就稀释一层。想治它，就得给记忆**修一条不被反复覆写的专用通道**——这正是下一讲 LSTM 登场的理由。1997 年的两位研究者为此设计的“传送带加三扇门”，将统治序列建模二十年。

9.6　RNN 的应用与局限，以及 2026 年为什么还学它

把 RNN 家族能干的活儿摆一张表：

表 9.2　RNN 的典型应用：输入输出都是“按时间步对齐”的活儿

应用	输入 → 输出	例子
语言模型	上文 → 下一字概率榜	输入法联想、接龙生成
时序预测	过去 T 步 → 下一步	负荷预测、温度趋势、相机参数预测
序列标注	每步输入 → 每步标签	分词、词性标注、设备日志标异常
语音与翻译	声波/文本 → 文本	2014–2016 年神经机器翻译的主力（下一讲）

局限也在表外明摆着两条。**第一，慢**：第 t 步必须等第 $t-1$ 步的便签传过来才能开算，串行天注定——第 5 讲卷积是“每个位置同时算”，GPU 几千个核心一起开火；RNN 的 GPU 只能眼睁睁看时间步一个个排队，训练快不起来。**第二，难长记忆**：上一节的梯度消失，50 步外的依赖基本学不到。这两条罪状，正是 2017 年后 Transformer（第 11 讲）取而代之的资本：全并行训练 + 注意力直达任意距离。

那为什么 2026 年的你还必须学它？三个理由，一个比一个实际。其一，**它是地基**：第 11 讲的注意力、本章末尾要讲的 Mamba，全都在跟“循环 + 状态”这组概念对话，跳过本讲去读大模型材料，处处是看不懂的黑话。其二，**小数据工业时序是它剩下的主场**：大模型要海量语料，而工厂里的传感器序列、设备状态、相机参数往往一天就攒几千步样本——这种小数据、低算力、要低延迟的场景，LSTM/GRU 至今是产线主流（下一讲给你选型表）。其三，**状态恒定**：RNN 无论读多长，随身只带一张便签，内存不随序列变长而涨——凡是“边收数据边出结果”的流式系统（逐帧相机、多轮对话 agent），都贪图这个便宜。

前沿速递：RNN 的 2024–2026 文艺复兴——循环思想没死，换了个马甲

Transformer 有个与生俱来的代价：注意力要两两比对，序列长 T，计算量就是 T^2（平方复杂度）——读 10 万字的文档，账单是 1 万字的 100 倍。于是 2023–2026 年，**状态空间模型**（State Space Model，SSM）高调回归：S4（2022）先用“循环 + 压缩状态”在超长序列上崭露头角，**Mamba**（2023 年底，Gu & Dao）靠“选择性状态空间”一战成名，2024 年的 Mamba-2 把理论地基夯实。看它的推理方式：每来一个新词，用一个固定大小的状态做一次递推更新——**这不就是带门控的 RNN 吗？**。Mamba 的计算量随序列线性增长（$O(T)$），状态恒定、推理如飞，2024–2026 年在基因组（百万级序列）、长文档、音频上全面开花；同门的 RWKV 则干脆长着 Transformer 的脸、跑着 RNN 的心。业内一句行话送给你：“Attention 之外，循环未死——它只是学会了先压缩、再记忆。”你在本讲学到的 h_t 递推，正是这场文艺复兴的语法起点。

项目连线：你的三个项目里，处处是时间步

六目拼接：九个可学习“参数头”网络（第 一 章）里有一个叫 `temporal_state`：输入形状 $[B, T, 24]$——T 个时间步、每步 24 维相机状态（匹配数、增益、运动估计等），输出预测**下一步**该走的参数。把 32 步状态序列切成帧、逐步入网、预测下一步——这正是本讲 9.1 节的 $[B, T, D]$ 与 9.2 节的循环结构在工业现场的微缩版；T 只有几十步，一个小 RNN 单元绰绰有余。**水下双目**：ROV 下潜拍回的是**时序帧流**（第 二 章），伪标签过滤的重重关卡里有一关叫“时序一致”：相邻帧的视差与深度不该跳变，跳了就说明上一帧的标签不可信——判断“一致与否”，前提是记得上一帧，这就是循环网络“带便签上班”的思想。**数据基座**：对话式问数（第 三 章）要记住上下文——多轮对话不串题，靠的是一套“哪些记、哪些忘、哪些此刻说”的上下文管理机制，A 组

“结果固定不串题”的验收线，考的正是下一讲门控思想的工程版。三次实验里跑起来的 agent 与训练系统，接上时序数据时的选型判断，就从本讲与下一讲来。

9.7　从这一讲到 2026

9.6 节的前沿盒子讲了循环思想的文艺复兴，这里把 2026 年的战报再往前递一页，三件事值得你记住。**第一，长序列上首次反超**：FlashAttention-2 是注意力阵营引以为傲的推理标配引擎，而 Mamba-2 在超过 8K token 的长序列上，推理速度与显存效率双双反超它（2025 年的对比实验给出了硬数据）——“线性复杂度干翻平方复杂度”第一次从口号变成实测。**第二，老牌选手重装上阵**：1997 年提出 LSTM 的霍赫赖特团队 2024 年推出 xLSTM，2025 年进一步放出 70 亿参数的 xLSTM-7B——生成质量与同规模的 Transformer 语言模型打得有来有回，短板只剩训练与推理速度还落后 Mamba 系一截。**第三，最扎实的战场在基因组学**：一条 DNA 序列动辄百万级碱基对起步，注意力那本平方账在这里直接爆表；Caduceus、Hawk 等状态空间模型凭线性复杂度把超长 DNA 当序列整段“读”进去，长程基准 LRB 专门为这类生物长序列任务评测它们的线性扩展能力——“边读边记”的循环思想，正在帮生物学家读生命的天书。

所以，别把本讲当成“被淘汰者”的传记。你在 9.2 节亲手拨弄过的那张便签 h_t，2026 年正以 Mamba、xLSTM 之名，在 DNA 测序数据与超长文档里每天运行——**循环思想没有死，它在等算力与硬件追上它的数学**。等你在第 11 讲看完注意力的繁华，不妨回来再翻一遍本讲的展开图：两条路线争的其实是同一道题——**用有限的状态，装下无限的历史**。

本讲课后题，是亲手把“便签”跑起来——字符接龙或正弦预测二选一，顺便亲眼看一次梯度消失。五要素如下。

第 9 讲课后编程题（用 AI 实现）

题目：指挥 AI 辅助，实现一个最小 RNN 时序实验，二选一。**(A) 字符级文字接龙**：找一段 5000 字以上你喜欢的中文文本（小说、新闻、歌词皆可）存为 `corpus.txt`，训练一个字符级 RNN（隐层 64–128 足够，CPU 也跑得动），训练完从种子字“今”开始采样生成 100 个字。**(B) 正弦波预测**：生成 600 点序列 $y=\sin(2\pi t/50)$ 加少量噪声，用滑动窗口“前 20 步预测第 21 步”，训练后从序列中段起递推预测 50 步，画出“预测 vs 真值”曲线。实现可用 PyTorch 的 `nn.RNN`，但代码里必须注释标清：h 在哪一步被更新、“按时间展开”发生在哪里。

交付物：(1) 代码文件 `rnn_toy.py`；(2) 生成样本截图或“预测 vs 真值”曲线 png 一张；(3) 50 字以上说明：模型学到了什么（接龙的语感？正弦的周期?）；(4) 一段 50 字以上的“我改了 AI 代码里的什么、为什么”。

提示词建议（可直接抄）：“我是零基础学生，正在学循环神经网络。请用 Python + PyTorch 帮我实现字符级 RNN 文字接龙：(1) 读入本地 corpus.txt（UTF-8），按字符建立词表；(2) 用 nn.RNN（hidden_size=128）搭一个‘输入当前字、预测下一字’的模型，交叉熵损失，Adam 学习率 0.003；(3) 训练 20 轮左右，打印每轮 loss；(4) 从种子字‘今’开始按概率采样生成 100 个字并打印；(5) 每行中文注释，并特别注明：隐藏状态 h 在哪一步被更新、‘按时间展开’发生在代码的哪里；(6) 最后用三句话回答：为什么生成到几十字以后，模型会开始忘掉开头?”（选 B 的同学把语料部分换成生成正弦序列与滑窗即可。）

验收线：代码能跑、loss 曲线整体下降；能对同学讲清 $h_t=\tanh(W_x x_t+W_h h_{t-1}+b)$ 里每个符号是什么、“展开”是什么意思；能当场回答两问——“RNN 为什么慢”（时间步串行，GPU 吃

不满）与“为什么记不住 50 步前的字”（梯度连乘消失，9.5 节）；至少一处自己的修改（换语料、改窗口长度 T、改隐层大小、把 tanh 换成 ReLU 看看会不会更糟，都算）。**代码是 AI 写的没关系，讲不清楚就是没做。**

加分项（任选其一）：(1) 画出梯度范数随回传步数的衰减曲线（把 9.5 节的“两种命运”亲手复现一遍）；(2) 把窗口长度从 20 加到 100，对比预测曲线，亲眼看一次“RNN 记不住长序列”；(3) 把语料中每个字的隐藏状态取出来做二维散点图（第 4 讲隐空间的手法），看看标点、数字、汉字是否各占一簇。

课后阅读：教材《深度学习模型与算法基础》第 167–186 页，与本讲内容对应；读到 RNN 展开图时，用本讲的“便签”语言把它自己讲一遍；读到梯度消失一节，对照 9.5 节的两条曲线。

序列数据（sequence）

次序本身携带信息的一串数据：文本、语音、视频、传感器读数；打乱次序即毁掉信息。

时间步（time step）

把序列切成的“一帧”，用 t 编号；张量形状 $[B, T, D]$ 里的 T 即时间步数。

循环神经网络（RNN）

同一个网络逐时间步反复使用、用隐藏状态传递记忆的序列模型；参数与序列长度无关。

隐藏状态（hidden state）

RNN 的“便签”：$h_t = \tanh(W_x x_t + W_h h_{t-1} + b)$，每步被整页重写。

按时间展开（unfold）

把循环沿时间拉开成深层网络图：深度即序列长度，是分析与训练 RNN 的通用视角。

权值共享（时间维）

同一套 W_x, W_h 用于每个时间步；与 CNN“模板扫遍全图”的空间维共享同理。

BPTT（沿时间反向传播）

把展开图当普通网络做反向传播；共享参数的梯度为各时间步贡献之和。

语言模型 / 文字接龙

逐字读入上文、预测下一字概率分布的任务；GPT 的看家本领与其一脉相承。

梯度消失/爆炸

误差沿时间回传要连乘 W_h 类因子：大于 1 连乘爆炸（NaN），小于 1 连乘消失（记不住远处的词）。

梯度裁剪（gradient clipping）

治爆炸的限流阀：梯度向量超过阈值就按比例缩短、方向不变。

第 10 讲　长短时记忆网络 LSTM 与 GRU

1991 年，慕尼黑工业大学有个叫霍赫赖特（Hochreiter）的学生，在毕业论文里把上一讲那个“RNN 为什么记不住远处的词”的病根算了个体无完肤——梯度沿时间连乘，小于 1 就指数消失。1997 年，他和导师施密德胡伯（Schmidhuber）发表了对症的新结构：**长短时记忆网络**（Long Short-Term Memory，LSTM）。名字起得很妙——“长的短时记忆”：还是短期记忆那一套用法，但这次能维持**很长**时间。此后二十年，这套设计统治了序列建模：2014 年神经机器翻译起飞靠它，2016 年 Google 上线神经网络翻译、错误率直降约 60%，底层是一摞 LSTM；你手机上的语音助手，2014–2017 年间几乎清一色 LSTM。2017 年 Transformer（第 11 讲）夺走了王座，但本讲你会看到，LSTM 发明的思想从未离场——它们只是换了住处。

上一讲的病历写得清楚：唯一一张便签 h_t 每步整页重写，tanh 还把数值摁在 -1 到 1，老信息每过一站稀释一层；梯度想跨越几十步去奖励“该记住的读法”，路上连乘到归零。LSTM 的药方两条，本讲逐条讲：**把记忆拆成两份**（一条不被反复覆写的传送带 + 一张便签），**给记忆装上闸门**（读什么、记什么、说什么，都由可学习的开关控制）。

本讲地图

讲什么：细胞状态传送带；三扇门——遗忘门、输入门、输出门；LSTM六行公式逐行人话翻译；精简版 GRU（更新门、重置门）与选型表；RNN vs LSTM 实战对比；2026 年什么场合还用 RNN 家族。**为什么值得学**：三个项目直接踩在门控思想上——六目的 `temporal_state` 头被要求“考虑门控结构”，水下伪标签的“时序一致”过滤是门控的朴素版，数据基座的多轮对话记忆管理是门控的工程版；小数据工业时序在 2026 年仍是 LSTM/GRU 的主场。**学完你能**：把 LSTM 每行公式翻译成人话；说清它凭什么记得住远处的词；在 LSTM/GRU/Transformer 之间做选型并给出理由。

10.1　细胞状态：一条直通远方的传送带

先复盘病根。普通 RNN 只有一份记忆 h，被迫身兼两职：既是**工作记忆**（算输出时要用），又是**长期档案**（要替远处留存线索）。两个职务打架：每来一个新输入，$h_t = \tanh(W_x x_t + W_h h_{t-1} + b)$ 整个被重算一遍——相当于档案每翻一页就得全文重抄一遍，抄十几页后，第一页的内容早被抄走样了。梯度消失的微观机制也在这：信息在 h 里每站都要过一次 tanh 与矩阵乘，误差回传时同样的路再走一遍，连乘不歇。

LSTM 的第一刀切得极准：**把两份职责拆开**。新增一条独立的记忆主干，叫**细胞状态**（cell state），记作 c_t——你可以把它想象成工厂里那条**传送带**：默认动作是**原样前移**（c_t 基本等于 c_{t-1}），只在有闸门指令时，往带上放一点新货、取走一点旧货。于是“昨天在巴黎吃的可丽饼”一旦上了传送带，就一路平移直通“它”字出现的那一刻，中途不再被抄写稀释。接下来的公式与图里你会反复看到这个分工：c 走传送带，h 走便签；h 随时可丢可换，c 只在门控批准时增删。

这一招还有个数学红利，值得你用红笔圈住：传送带的更新是**加法**——$c_t = c_{t-1} +$（小改动），而不是重写。误差沿加法通道回传，几乎原封不动地直达远方，梯度消失被釜底抽薪。眼熟吗？第 7 讲 ResNet 的残差捷径正是同一个智慧：**给信息修一条加法高速公路**。1997 年的 LSTM 与 2015 年的 ResNet 隔着十八年殊途同归——“深层网络里，让该原样通过的东西走加法通道”这一句话，值两

座图灵奖级别的洞察。

生活化类比：便签、传送带与三道闸

想象一条工厂流水线：底部一条**传送带**（细胞状态 c）载着货箱长途运输；传送带上方坐着一名质检员，手里一张**便签**（隐藏状态 h），随手记录当前工位的情况。流水线沿线有三道闸：第一道管旧货**取走多少**（该忘的忘），第二道管新货**上带多少**（该记的记），第三道管质检员**从带上抄多少**去写便签、对外报告（该说的说）。三道闸的开度全由质检员根据“新到货 + 便签”现场决定——记住这幅图，10.3 节的六行公式你一条都不会背错。

单元全貌见下图：那条横穿单元、几乎不打弯的长箭头就是细胞状态传送带，沿途三道闸门依次放行、写入、抄送；所有支路都从传送带上分叉或汇入，这是 LSTM 结构图最醒目的特征。

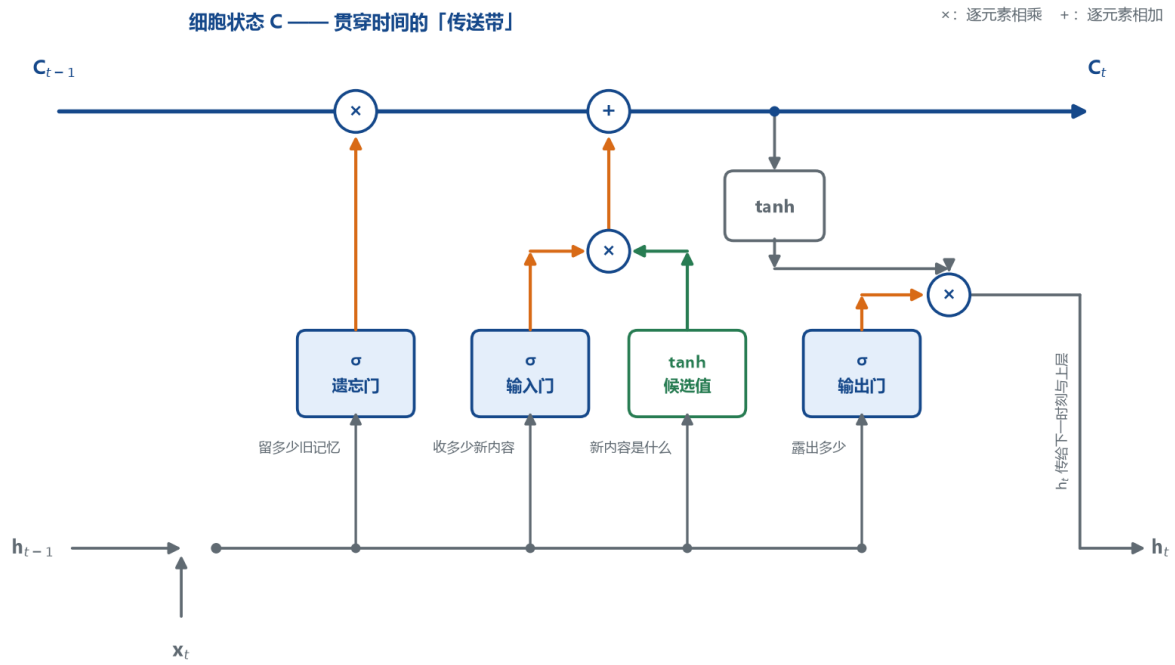


图 10.1　LSTM 单元内部全貌：底部粗箭头是细胞状态传送带（c 原样前移），沿途三扇门——遗忘门（乘 f 取走旧货）、输入门（乘 i 放入新货 $\tilde{c}$）、输出门（乘 o 抄送出 h）；h 与 x 一起决定每扇门的开度

这套“传送带 + 三扇门”的画法早已是序列建模的通用语言。下面这张经典图出自 Christopher Olah 2015 年的博客文章——它把上面那张工程版翻译成了更直观的彩色版，一代工程师是读着它入门的：

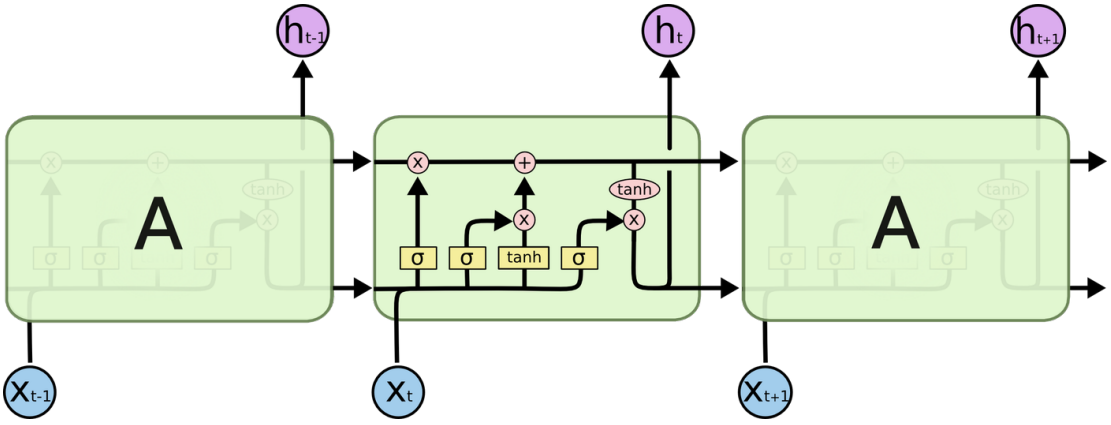


图 10.2　LSTM 门结构的经典画法：上方水平主线即细胞状态传送带，三列黄色模块依次是遗忘门、输入门、输出门，各门输出与传送带逐元素相乘

图片来源：Christopher Olah, "Understanding LSTM Networks" 博客文章, 2015

10.2　三扇门：该忘的忘，该记的记，该说的说

先说什么是**门**（gate）。每扇门就是一个 sigmoid 小网络（一层全连接）：拿两样东西当输入——**新来的输入** x_t 和**旧便签** h_{t-1}——输出一个 0 到 1 之间的数。这个数的含义是**闸门开度**：0 全关，1 全开，0.5 放半。因为 sigmoid 连续可导，"开多大"本身是可以被梯度学出来的：网络训练时自己决定每一步该开几分。三扇门各司其职：

- **遗忘门**（forget gate）——**该忘的忘**。像水库的**水闸**：传送带上哪些旧货价值耗尽了（话题换了、任务完成了），开闸放走；开度接近 1 就是一点不忘，接近 0 就是全忘。
- **输入门**（input gate）——**该记的记**。像**安检门**：新信息先在门口打分，值钱的才放上传送带、上多少份量；无用的杂讯在门口就被拦下。
- **输出门**（output gate）——**该说的说**。像**录音笔的红灯**：档案里什么都记着，但此刻要不要放出去、放多少，红灯说了算——细胞状态满载，便签上只写当前需要的那部分。

知识补齐：sigmoid 是"软开关"——为什么不用阶跃

门的本质是开关，可开关的数学版——阶跃函数（0 或 1 二选一）——在第 2 讲就见过它的死穴：不连续、不可导，梯度传不过去，学不了。sigmoid（第 2 讲激活函数）是阶跃的"软化版"：平滑地从 0 爬到 1，处处可导。于是"开或关"变成"开几分"，硬决策变成软比例，梯度能顺着门开度一路回传，把"什么时候该开大、什么时候该关小"学成参数。深度学习里反复出现这一手：想要离散决策，就用 sigmoid/softmax 的连续版来近似，训练完再（需要时）取整。

还有个细节别漏：门是**逐维**的。h 和 c 都是向量，sigmoid 输出的开度也是同长度的向量，逐个格子对应——同一时刻完全可以"可丽饼那一格全开、天气那一格全关"。这正是下一节公式里那个圈点符号 $\odot$ 的用武之地。

动手时间：你就是 LSTM——手动拨三扇门

手机语音助手连听五句话。请给每一步的三扇门填开度（0–1），并写一句理由，感受"门控 = 记忆管理"：

步	你听到的话	遗忘门（旧货取走多少）	输入门（新货上带多少）	输出门（此刻说多少）
1	“记住：明天十点开会”	（空带上没旧货）		
2	“顺便记一下：买咖啡”			
3	“开会的事取消啦”			
4	“咖啡钱记我账上”			
5	“今天就这些，谢谢”			

参考要点：步 1 输入门全开（时间信息上带）；步 2 遗忘门全开（买咖啡是另开一格新货，开会那格别动——回忆“门是逐维的”）；步 3 遗忘门对“开会”那一格开大（这条任务作废），对“买咖啡”保持全关；步 4 输入门开、输出门开始放行；步 5 任务清场，遗忘门整体开大。做完这道题，你已经用自己的手执行了一遍 LSTM——剩下的只是把这双手翻译成矩阵。

10.3　LSTM 逐步计算：六行公式，六句人话

上闸门了。约定符号：σ 是 sigmoid（读“西格玛”），$\odot$（读“圈点”）是**逐元素相乘**——两个同长度向量对应格子相乘，正是“开度乘货量”。

知识补齐：Hadamard 积 $\odot$——“开度乘货量”的数学写法

公式里的圈点有个学名：*Hadamard* 积（哈达玛积，Hadamard product），规矩只有一条——**同形状，对应格子相乘**。三条具体规则各一句话：**向量 $\odot$ 向量**：长度相同，对应格相乘，结果还是同长度向量（本节最常用：门开度 $\odot$ 货量）；**矩阵 $\odot$ 矩阵**：行列数完全相同，对应格相乘；**向量 $\odot$ 矩阵**：按 NumPy 的“广播”规则把向量沿行（或列）铺成同形状的矩阵，再逐格相乘。注意它与矩阵乘法 AB（行乘列、乘完还要逐对求和）是两码事：$\odot$ 不求和、不改形状，纯粹“点对点”相乘——所以每扇门“该开哪几格、各开几分”，一个 $\odot$ 就写得明明白白。

每扇门都由同一个配方生成：$\sigma(Wx_t+Uh_{t-1}+b)$，不同的门用不同的权重。下面六行就是 LSTM 的全部，每行配一句人话：

第一组，三扇门的开度：

$$f_t=\sigma(W_f x_t+U_f h_{t-1}+b_f) \qquad \text{人话：遗忘门开几成？} \tag{10.1}$$

$$i_t=\sigma(W_i x_t+U_i h_{t-1}+b_i) \qquad \text{人话：输入门开几成？} \tag{10.2}$$

$$o_t=\sigma(W_o x_t+U_o h_{t-1}+b_o) \qquad \text{人话：输出门开几成？} \tag{10.3}$$

第二组，备货与更新传送带：

$$\tilde{c}_t=\tanh(W_c x_t+U_c h_{t-1}+b_c) \qquad \text{人话：候选记忆——这步“值得记的内容”先打个草稿} \tag{10.4}$$

$$c_t=f_t\odot c_{t-1}+i_t\odot \tilde{c}_t \qquad \text{人话：传送带 = 旧货按遗忘门比例保留 + 新货按输入门比例上带} \tag{10.5}$$

第三组，对外输出：

$$h_t=o_t\odot\tanh(c_t) \qquad \text{人话：便签 = 把传送带过一遍 tanh，按输出门开度抄出来} \tag{10.6}$$

六行里最核心的是第 5 行：$c_t=f_t\odot c_{t-1}+i_t\odot\tilde{c}_t$。**这就是“记语文”：旧记忆的每一格乘一个 0–1 的保留比例，新草稿的每一格乘一个 0–1 的写入比例，两者相加。**想让某个格子永志不忘？把

那格的 f_t 学成 1、i_t 学成 0——于是 c_t 的那格 $= c_{t-1}$ 的那格，原样平移，一路走到天涯海角。想让某格清零？f_t 学成 0 即可。梯度回传时沿这条加法通道几乎无衰减地倒走几十上百步——上一讲的 γ^k 连乘惨案，在这里被 $f_t \approx 1$ 的通道强行叫停。

和普通 RNN 对个账：普通 RNN 一层 tanh、一份记忆；LSTM 三层 σ 加一层 tanh、两份记忆（c 与 h），权重矩阵从 2 套涨到 8 套——**参数约为普通 RNN 的四倍**，每步计算量同比例变贵。贵得值不值？10.5 节见分晓。训练流程与上一讲完全一样：展开、逐时间步算损失、BPTT 回传——变的只是每个时间步里发生的事。

10.4　GRU：两扇门的精简版

三扇门是标配吗？2014年，曹（Cho）等人在做机器翻译时提出**门控循环单元**（Gated Recurrent Unit，GRU），砍掉一扇门、取消独立传送带，性能却与 LSTM 大体相当。它的手术刀切在两处（为省版面，下列公式从略偏置项）。

第一处：把遗忘门和输入门**合并成一扇更新门**（update gate）z_t，并让两者**此消彼长**——旧记忆保留 $1 - z_t$ 份，新内容写入 z_t 份，一个旋钮管两头：

$$z_t = \sigma(W_z x_t + U_z h_{t-1}) \qquad \text{人话：更新门——“留旧 } vs \text{ 迎新”的比例拨到几成？} \tag{10.7}$$

第二处：加一扇**重置门**（reset gate）r_t，只管一件事——生成新候选草稿时，回头看多少旧便签：

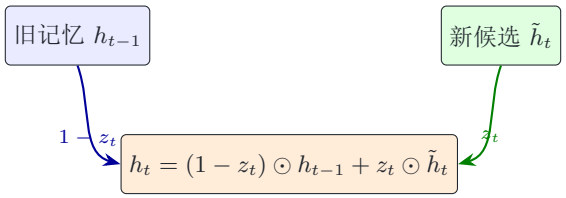
$$r_t = \sigma(W_r x_t + U_r h_{t-1}) \qquad \text{人话：重置门——打新草稿时，旧便签参考几成？} \tag{10.8}$$

$$\tilde{h}_t = \tanh(W_h x_t + U_h (r_t \odot h_{t-1})) \qquad \text{人话：候选记忆——在“被重置门过滤过的旧记忆”基础上写新货} \tag{10.9}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \qquad \text{人话：新记忆 } = \text{旧记忆按比例混入新货} \tag{10.10}$$

看最后一行（几何直观见下图）：h_t 是“旧记忆”与“新候选”的**加权平均**，落在两点连线上的某处——$z_t = 0$ 则整根记忆原样平移（GRU 版的“传送带”，只是焊在了 h 上），$z_t = 1$ 则彻底改写。没有独立的 c_t，记忆的“长途档”与“工作档”合并在一根 h 里，全靠更新门拨比例。

更新门 z_t：一个旋钮管两头，此消彼长



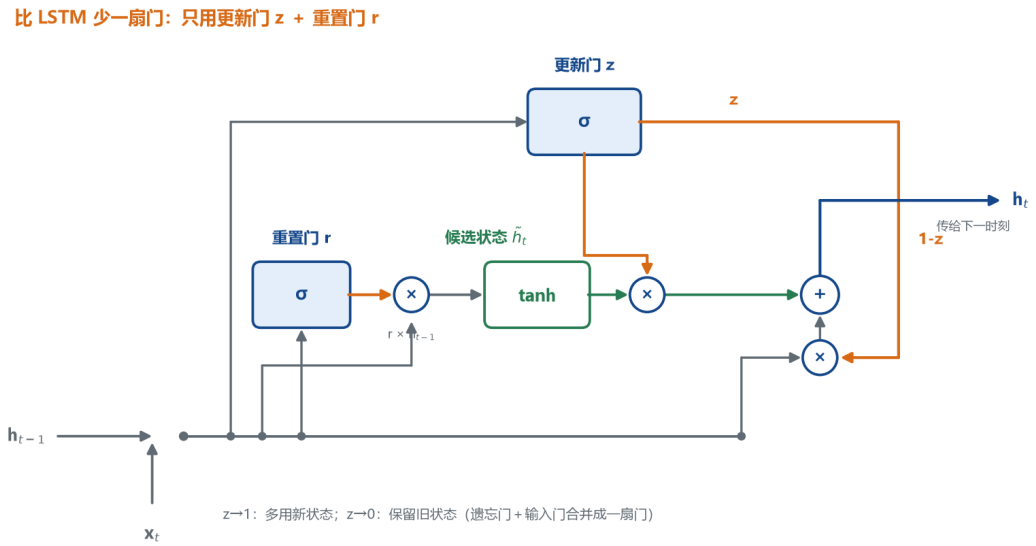


图 10.3　GRU 单元：只剩更新门与重置门两扇门，记忆的更新是“旧记忆与新候选的加权平均”，结构比 LSTM 简洁一截

账面变化一目了然：权重矩阵从 LSTM 的 8 套降到 6 套，**参数少约 25%**，训练更快、更不容易在小数据上过拟合。效果呢？大量实证研究的结论朴实无华：**多数任务上两者打平**，谁也不总赢。给你一张选型表，三条经验加一条最诚实的：

表 10.1　LSTM vs GRU 选型速查：够用就好

你的处境	倾向	一句话理由
数据少、想快点出结果	GRU	参数少 25%，小数据上更快收敛、不易过拟合
序列长、依赖远	LSTM	独立细胞状态的传送带更稳，长程记忆略占优
要上嵌入式、低算力	GRU	参数少推理快，状态内存恒定
拿不准	都跑一遍	成本低到不值得犹豫：两个都训，谁的验证集好选谁

表里的建议是“按处境选人”，再把两兄弟的**结构账**并排算一遍——同一道“记忆管理”的题，LSTM 与 GRU 给出了两种答案：

表 10.2　LSTM vs GRU：同一道题的两种答案

对比项	LSTM	GRU
记忆的产生方式	独立细胞状态 c_t 当传送带，长途记忆走“专线”	无独立细胞状态，长途档与工作档合并在一根 h 里
新状态的产生	遗忘门 + 输入门两扇门分工：一扇管留旧、一扇管迎新	更新门一扇门管两头：留旧 $1-z_t$、迎新 z_t，此消彼长
输出调节	有输出门：先想好“此刻说多少”，再决定便签 h_t 抄多少	无输出门：新便签 h_t 直接全量对外
参数量与速度	三扇门加一份候选，共 4 套权重（约 8 套矩阵）	两扇门加一份候选，共 3 套权重（约 6 套矩阵），少约 25%，更快
适用场景	序列长、依赖远时更稳	数据少、要快、要上嵌入式时更划算

一张表收拢成一句话：**LSTM 用“两扇门 + 独立传送带”换长途稳定性，GRU 用“一扇门 +**