截面与面板数据分析 H: 人工智能导论课程讲义

第一部分：从普通最小二乘到正则化回归

周云龙

2025 年 10 月 12 日

1 引言：回归问题的本质

在机器学习和统计学中，回归分析是最基础也最重要的工具之一。我们今天要探讨的核心问题是：给定数据 (x_i, y_i)，$i = 1, \ldots, n$，如何找到一个函数 $f(x)$ 来预测 y?

2 普通最小二乘法（OLS）

2.1 问题建模

假设我们有线性模型：

$$y = X\beta + \epsilon \tag{1}$$

其中 $y \in \mathbb{R}^n$ 是响应变量，$X \in \mathbb{R}^{n \times p}$ 是设计矩阵，$\beta \in \mathbb{R}^p$ 是待估计的系数向量，ϵ 是误差项。

2.2 OLS 的目标函数

OLS 的目标是最小化残差平方和：

$$\hat{\beta}_{OLS} = \arg\min_{\beta} \|y - X\beta\|_2^2 = \arg\min_{\beta} \sum_{i=1}^{n} (y_i - x_i^T \beta)^2 \tag{2}$$

2.3 OLS 的解析解

通过对目标函数求导并令其为零，我们得到正规方程：

$$X^T X \beta = X^T y \tag{3}$$

当 $X^T X$ 可逆时，OLS 有唯一解：

$$\hat{\beta}_{OLS} = (X^T X)^{-1} X^T y \tag{4}$$

2.4 OLS 的问题

- **多重共线性**：当特征之间高度相关时，X^TX 接近奇异，导致估计不稳定
- **过拟合**：当特征数量 p 接近或超过样本数量 n 时，模型会过度拟合训练数据
- **无特征选择**：OLS 不会将系数压缩为零，无法自动进行特征选择

3 Ridge 回归（L2 正则化）

3.1 Ridge 的目标函数

为了解决 OLS 的不稳定性，Ridge 回归在目标函数中加入 L2 惩罚项：

$$\hat{\beta}_{Ridge} = \arg\min_{\beta} \left\{ \|y - X\beta\|_2^2 + \lambda \|\beta\|_2^2 \right\} \tag{5}$$

其中 $\lambda > 0$ 是正则化参数，控制惩罚的强度。

3.2 Ridge 的解析解

Ridge 回归有显式解：

$$\hat{\beta}_{Ridge} = (X^TX + \lambda I)^{-1} X^T y \tag{6}$$

注意到 $X^TX + \lambda I$ 总是可逆的（因为加入了 λI），这就解决了 OLS 中的共线性问题。

3.3 通过 SVD 理解 Ridge 回归

3.3.1 SVD 分解

对设计矩阵 X 进行奇异值分解（SVD）：

$$X = UDV^T \tag{7}$$

其中 $U \in \mathbb{R}^{n \times p}$ 是左奇异向量矩阵，$D = \text{diag}(d_1, \ldots, d_p)$ 是奇异值对角矩阵（$d_1 \geq d_2 \geq \cdots \geq d_p \geq 0$），$V \in \mathbb{R}^{p \times p}$ 是右奇异向量矩阵。

3.3.2 OLS 的 SVD 表示

将 SVD 代入 OLS 解：

$$\hat{\beta}_{OLS} = (X^T X)^{-1} X^T y \tag{8}$$
$$= (VD^2V^T)^{-1} VDU^T y \tag{9}$$
$$= VD^{-2}V^T VDU^T y \tag{10}$$
$$= VD^{-1}U^T y \tag{11}$$
$$= \sum_{j=1}^{p} \frac{u_j^T y}{d_j} v_j \tag{12}$$

可以看到，当某个奇异值 d_j 很小时，对应项的系数会变得非常大，导致不稳定。

3.3.3 Ridge 的 SVD 表示

类似地，Ridge 回归的解为：

$$\hat{\beta}_{Ridge} = (X^T X + \lambda I)^{-1} X^T y \tag{13}$$
$$= (VD^2V^T + \lambda I)^{-1} VDU^T y \tag{14}$$
$$= V(D^2 + \lambda I)^{-1} DU^T y \tag{15}$$
$$= \sum_{j=1}^{p} \frac{d_j}{d_j^2 + \lambda} u_j^T y \cdot v_j \tag{16}$$

3.3.4 λ 的几何意义

从 SVD 表示可以清楚地看到 λ 的作用：

- 每个奇异方向 v_j 对应的系数被乘以收缩因子：

$$\frac{d_j^2}{d_j^2 + \lambda} \tag{17}$$

- 当 $d_j^2 \gg \lambda$ 时，收缩因子接近 1，几乎不收缩
- 当 $d_j^2 \ll \lambda$ 时，收缩因子接近 0，大幅收缩
- **关键洞察**：Ridge 对小奇异值对应的方向收缩更多，这正是那些导致不稳定的方向！

因此，λ 越大，所有系数被收缩得越厉害，但 Ridge 永远不会将系数压缩为精确的零。

4 Lasso 回归（L1 正则化）

4.1 Lasso 的目标函数

Lasso 使用 L1 惩罚项：

$$\hat{\beta}_{Lasso} = \arg\min_{\beta} \left\{ \|y - X\beta\|_2^2 + \lambda \|\beta\|_1 \right\} \tag{18}$$

其中 $\|\beta\|_1 = \sum_{j=1}^{p} |\beta_j|$ 是 L1 范数。

4.2 Lasso 没有显式解

与 Ridge 不同，**Lasso 没有解析解**。原因是 L1 范数在零点不可微，导致目标函数不可直接求导求解。通常需要使用数值优化方法，如：

- 坐标下降法（Coordinate Descent）
- 近端梯度下降（Proximal Gradient Descent）
- LARS 算法（Least Angle Regression）

4.3 Lasso 的稀疏性：让系数变为零

Lasso 最重要的性质是**稀疏性**：当 λ 足够大时，Lasso 会将某些系数压缩为精确的零，从而实现自动特征选择。

4.3.1 几何直觉

从约束优化的角度理解：

- Ridge 的约束区域是一个球体：$\|\beta\|_2^2 \leq t$
- Lasso 的约束区域是一个菱形（在高维中是超菱形）：$\|\beta\|_1 \leq t$

等高线（残差平方和）与 Lasso 约束区域相交时，更容易在坐标轴上（即某些 $\beta_j = 0$）相交，因为菱形有尖角。而 Ridge 的球形约束区域没有尖角，所以很少在坐标轴上相交。

4.4 Lasso 不平滑

虽然 Lasso 能产生稀疏解，但它的解路径（solution path）是**不平滑的**：当 λ 变化时，系数可能突然从零跳到非零值，或从非零值跳到零。这与 Ridge 的平滑收缩形成对比。

性质	Ridge (L2)	Lasso (L1)
惩罚项	$\lambda\|\beta\|_2^2$	$\lambda\|\beta\|_1$
解析解	有：$(X^TX+\lambda I)^{-1}X^Ty$	无
稀疏性	无（系数不会为零）	有（系数可以为零）
特征选择	不能	能
解的平滑性	平滑	不平滑
计算复杂度	低（矩阵运算）	较高（需要迭代）
多重共线性	所有相关特征都保留，系数被收缩	倾向于选择一个，丢弃其他

5 Ridge vs. Lasso：总结对比

6 实际应用：让你赚大钱的正则化回归

正则化回归不仅是理论上优雅的工具，在实际应用中更是价值连城。以下是几个能”赚大钱” 的真实应用场景：

6.1 量化交易：股票价格预测

6.1.1 应用场景

对冲基金和量化交易公司使用 Lasso 回归进行因子选择：

- **问题**：有数百个潜在因子（市盈率、动量、波动率、宏观指标等），但只有少数真正有预测能力
- **解决方案**：使用 Lasso 自动筛选出最重要的因子，构建稀疏的多因子模型
- **收益**：减少过拟合，提高样本外预测准确率，每年可带来数百万到数亿美元的超额收益

实例：Renaissance Technologies 等顶级量化基金大量使用正则化方法，其 Medallion 基金年均收益率超过 35%。

6.2 信用评分：预测违约概率

6.2.1 应用场景

银行和金融科技公司使用 Lasso 进行信用风险建模：

- **问题**：客户有上千个特征（收入、负债、消费习惯、社交数据等），需要预测是否会违约

- **解决方案**：Lasso 选出最关键的 10-20 个特征，既提高预测精度，又满足监管对模型可解释性的要求
- **收益**：降低坏账率 1%，对大银行而言意味着每年节省数亿美元损失

实例：FICO 评分系统和各大银行的内部评分卡模型大量应用正则化逻辑回归。

6.3 推荐系统：电商和流媒体

6.3.1 应用场景

Netflix、Amazon 使用 Ridge 回归进行协同过滤：

- **问题**：用户-物品评分矩阵极其稀疏，直接矩阵分解容易过拟合
- **解决方案**：在矩阵分解中加入 Ridge 惩罚，稳定化低秩近似
- **收益**：提高推荐准确率，每提升 1% 的点击率可能带来数千万美元的额外收入

实例：Netflix Prize 竞赛中，获胜团队的关键技术之一就是正则化矩阵分解。

6.4 医疗诊断：基因数据分析

6.4.1 应用场景

制药公司使用 Lasso 分析基因表达数据：

- **问题**：基因芯片有数万个基因，但样本数只有几百，典型的 $p \gg n$ 问题
- **解决方案**：Lasso 识别出与疾病相关的关键基因（通常只有几十个），用于药物靶点发现
- **收益**：加速新药研发，一个成功药物的市场价值可达数十亿美元

实例：Oncotype DX 等癌症基因检测产品使用 Lasso 选出的基因标志物，市场规模超过 10 亿美元。

6.5 在线广告：点击率预测

6.5.1 应用场景

Google、Facebook 使用 Lasso 进行特征工程：

- **问题**：用户和广告有数百万维特征（交叉特征爆炸），需要实时预测点击率
- **解决方案**：Lasso 筛选出最有效的特征组合，减少模型复杂度和计算成本

- **收益**：提高广告投放效率，每年为广告平台带来数百亿美元收入

实例：Google 的 Wide & Deep 模型和 Facebook 的 GBDT+LR 模型都大量使用 L1 正则化。

6.6 房地产估值：自动定价模型

6.6.1 应用场景

Zillow 等房地产平台使用 Ridge 和 Lasso：

- **问题**：房价受多种因素影响（地段、面积、学区、交通等），某些因素高度相关
- **解决方案**：结合 Ridge 处理共线性，Lasso 选择关键特征
- **收益**：准确估值每套房产，支撑数十亿美元的房地产交易

7 总结

从普通最小二乘到正则化回归，我们看到了机器学习如何通过巧妙的数学设计解决实际问题：

- **OLS**：简单但不稳定
- **Ridge**：通过 L2 惩罚稳定化，SVD 分解揭示了 λ 如何智能地收缩不稳定方向
- **Lasso**：通过 L1 惩罚实现稀疏性，自动特征选择，虽无显式解但威力强大

这些看似简单的方法，在金融、医疗、互联网等领域创造了巨大的经济价值，真正实现了”用数学赚钱”的梦想。

人工智能导论课程讲义

第二部分：经典机器学习方法

8 引言：从线性到非线性

在第一部分中，我们学习了线性回归及其正则化方法。但现实世界中的许多问题是非线性的，或者需要做分类而非回归。本节我们将探讨一些经典的、甚至有些" 过时" 但依然重要的机器学习方法。这些方法构成了现代 AI 的基础，理解它们能帮助我们更好地理解今天的深度学习。

9 感知机（Perceptron）

9.1 历史背景

感知机由 Frank Rosenblatt 在 1957 年提出，是神经网络的鼻祖。虽然它非常简单，但它开启了人工智能的第一次浪潮。

9.2 模型定义

感知机是一个二分类线性模型。给定输入 $x \in \mathbb{R}^d$，感知机的预测为：

$$\hat{y} = \text{sign}(w^T x + b) \tag{19}$$

其中 $w \in \mathbb{R}^d$ 是权重向量，$b \in \mathbb{R}$ 是偏置，$\text{sign}(\cdot)$ 是符号函数：

$$\text{sign}(z) = \begin{cases} +1, & z \geq 0 \\ -1, & z < 0 \end{cases} \tag{20}$$

几何上，$w^T x + b = 0$ 定义了一个超平面，将空间分成两部分。

9.3 感知机学习算法

感知机使用在线学习（online learning）方式更新参数。假设训练数据为 (x_i, y_i)，$y_i \in \{-1, +1\}$。

算法流程：

1. 初始化 $w = 0$，$b = 0$

2. 对每个样本 (x_i, y_i)：

- 如果 $y_i(w^T x_i + b) \leq 0$（分类错误），则更新：

$$w \leftarrow w + \eta y_i x_i \tag{21}$$

$$b \leftarrow b + \eta y_i \tag{22}$$

其中 $\eta > 0$ 是学习率

3. 重复步骤 2，直到所有样本都分类正确或达到最大迭代次数

9.4 感知机收敛定理

定理：如果数据是线性可分的，感知机算法保证在有限步内收敛到一个完美分类的解。

但是，如果数据不是线性可分的，感知机永远不会收敛！这是感知机的致命弱点。

9.5 感知机的局限性

XOR 问题：Marvin Minsky 和 Seymour Papert 在 1969 年的著作中指出，单层感知机无法解决 XOR 问题：

考虑 XOR 的真值表：

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

不存在任何直线能将 $(0,1)$ 和 $(1,0)$ 从 $(0,0)$ 和 $(1,1)$ 中分开。这个发现导致了 AI 的第一次寒冬。

9.6 多层感知机（MLP）

解决方案是堆叠多层感知机，形成多层神经网络。两层感知机可以解决 XOR 问题，理论上可以逼近任何连续函数（universal approximation theorem）。但在 1980 年代之前，我们不知道如何有效训练多层网络，直到反向传播算法的出现。

10 支持向量机（SVM）

10.1 从感知机到 SVM

感知机找到的分类超平面不唯一——只要能分开数据，任何超平面都行。SVM 则更进一步：找到**间隔最大**的超平面。

10.2 线性可分 SVM

10.2.1 几何间隔

给定超平面 $w^T x + b = 0$，点 x_i 到超平面的距离为：

$$\text{distance} = \frac{|w^T x_i + b|}{\|w\|} \tag{23}$$

对于正确分类的点（$y_i(w^T x_i + b) > 0$），几何间隔为：

$$\gamma_i = \frac{y_i(w^T x_i + b)}{\|w\|} \tag{24}$$

10.2.2 最大间隔分类器

SVM 的目标是最大化最小间隔：

$$\max_{w,b} \min_i \gamma_i = \max_{w,b} \min_i \frac{y_i(w^T x_i + b)}{\|w\|} \tag{25}$$

等价地，我们可以将问题写成：

$$\min_{w,b} \frac{1}{2}\|w\|^2 \tag{26}$$

$$\text{s.t. } y_i(w^T x_i + b) \geq 1, \quad i = 1, \ldots, n \tag{27}$$

这是一个二次规划问题（QP），可以高效求解。

10.3 软间隔 SVM

现实中数据往往不是完全线性可分的。软间隔 SVM 引入松弛变量 $\xi_i \geq 0$：

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i \tag{28}$$

$$\text{s.t. } y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \tag{29}$$

其中 $C > 0$ 是惩罚参数，控制间隔和误分类的权衡。这与正则化回归中的 λ 类似！

10.4 核技巧（Kernel Trick）

SVM 最强大的特性是可以通过核函数隐式地将数据映射到高维空间，而无需显式计算高维特征。

10.4.1 对偶形式

通过拉格朗日对偶，SVM 的解可以写成：

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i \tag{30}$$

预测时：

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i x_i^T x + b = \sum_{i=1}^{n} \alpha_i y_i \langle x_i, x \rangle + b \tag{31}$$

注意到只需要计算内积 $\langle x_i, x \rangle$！

10.4.2 核函数

定义核函数 $K(x, x') = \langle \phi(x), \phi(x') \rangle$，其中 ϕ 是特征映射。常用核函数：

- **线性核**：$K(x, x') = x^T x'$
- **多项式核**：$K(x, x') = (x^T x' + c)^d$
- **RBF 核（高斯核）**：$K(x, x') = \exp(-\gamma \|x - x'\|^2)$
- **Sigmoid 核**：$K(x, x') = \tanh(\alpha x^T x' + c)$

使用核函数后，预测变为：

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + b \tag{32}$$

10.5 为什么 SVM 曾经很流行？

在 2000-2010 年代，SVM 是最流行的机器学习方法之一，原因包括：

- 理论基础扎实（VC 维、统计学习理论）
- 全局最优解（凸优化问题）
- 核技巧强大（可以处理非线性问题）
- 泛化能力强（最大间隔原则）

但随着深度学习的兴起，SVM 逐渐退居二线，因为深度学习在大规模数据上表现更好，且可以自动学习特征表示。

11 决策树（Decision Trees）

11.1 决策树的直觉

决策树是一种基于规则的模型，通过一系列 if-then-else 规则进行决策。它模仿人类的决策过程，因此非常直观和可解释。

11.2 决策树的构建

11.2.1 基本结构

决策树由以下部分组成：

- **根节点**：包含所有训练数据
- **内部节点**：基于某个特征的某个阈值进行分裂
- **叶节点**：输出预测结果

11.2.2 分裂准则

如何选择最好的特征和阈值进行分裂？我们需要量化”纯度”（purity）。

分类树常用准则：

1. **基尼不纯度（Gini Impurity）**：

$$\text{Gini}(S) = 1 - \sum_{k=1}^{K} p_k^2 \tag{33}$$

其中 p_k 是类别 k 在集合 S 中的比例。

2. **信息熵（Entropy）**：

$$H(S) = -\sum_{k=1}^{K} p_k \log_2 p_k \tag{34}$$

3. **信息增益（Information Gain）**：

$$IG(S, A) = H(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} H(S_v) \tag{35}$$

其中 A 是特征，S_v 是特征 A 取值为 v 的样本子集。

回归树常用准则：

均方误差（MSE）：

$$\text{MSE}(S) = \frac{1}{|S|} \sum_{i \in S} (y_i - \bar{y})^2 \tag{36}$$

其中 $\bar{y}$ 是集合 S 中标签的均值。

11.2.3 CART 算法

分类与回归树（Classification and Regression Trees）是最常用的决策树算法。

算法流程（递归）：

1. 如果满足停止条件（如达到最大深度、节点样本数过少、纯度足够高），创建叶节点并返回
2. 否则，对每个特征 j 和每个可能的分裂点 t：
   - 将数据分为 $S_{\text{left}} = \{x : x_j \leq t\}$ 和 $S_{\text{right}} = \{x : x_j > t\}$
   - 计算分裂后的总不纯度：

$$\text{Cost}(j, t) = \frac{|S_{\text{left}}|}{|S|} G(S_{\text{left}}) + \frac{|S_{\text{right}}|}{|S|} G(S_{\text{right}}) \tag{37}$$

3. 选择使 $\text{Cost}(j, t)$ 最小的特征 j^* 和分裂点 t^*
4. 递归地在左右子节点上构建子树

11.3 决策树的优缺点

优点：

- 易于理解和解释（可视化为树状图）
- 不需要特征缩放
- 可以处理数值和类别特征
- 可以自动处理特征交互
- 计算效率高（预测时只需沿树下降）

缺点：

- 容易过拟合（尤其是深树）
- 不稳定（数据的小变化可能导致完全不同的树）
- 贪心算法（局部最优，不保证全局最优）
- 对于某些问题表现不如线性模型

11.4 剪枝（Pruning）

为了防止过拟合，我们需要剪枝。

11.4.1 预剪枝（Pre-pruning）

在构建树的过程中提前停止：

- 限制最大深度
- 限制叶节点最小样本数
- 限制分裂的最小信息增益

11.4.2 后剪枝（Post-pruning）

先构建完整的树，再自底向上剪枝。使用成本复杂度剪枝（cost-complexity pruning）：

$$R_\alpha(T) = R(T) + \alpha|T| \tag{38}$$

其中 $R(T)$ 是树的训练误差，$|T|$ 是叶节点数，α 是复杂度参数。

12 集成方法（Ensemble Methods）

单个决策树虽然简单，但不够强大。集成方法通过组合多个弱学习器来提升性能。

12.1 Bagging（Bootstrap Aggregating）

12.1.1 基本思想

通过自助采样（bootstrap sampling）生成多个训练集，在每个训练集上训练一个模型，最后对预测结果取平均（回归）或投票（分类）。

算法流程：

1. 对于 $b = 1, \ldots, B$：
   - 从训练集 D 中有放回地采样 n 个样本，得到 D_b
   - 在 D_b 上训练模型 f_b
2. 最终预测：
   - 回归：$\hat{f}(x) = \frac{1}{B}\sum_{b=1}^{B} f_b(x)$
   - 分类：$\hat{y}(x) = \text{majority vote of } f_1(x), \ldots, f_B(x)$

12.1.2 随机森林（Random Forest）

随机森林是 Bagging 的变体，专门用于决策树。除了样本采样，还增加了特征采样：

在每次分裂时，随机选择 m 个特征的子集（通常 $m = \sqrt{d}$ 或 $m = \log_2 d$），只从这些特征中选择最优分裂。

这进一步增加了树之间的多样性，降低相关性，提高泛化能力。

为什么随机森林强大？

- 降低方差（通过平均多个高方差模型）
- 保持低偏差（每棵树都足够复杂）
- 不容易过拟合
- 可以处理高维数据
- 可以评估特征重要性

12.2 Boosting

12.2.1 基本思想

Boosting 是顺序地训练弱学习器，每个新学习器关注前面学习器的错误。

12.2.2 AdaBoost

AdaBoost（Adaptive Boosting）是最经典的 boosting 算法。

算法流程：

1. 初始化样本权重：$w_i^{(1)} = \frac{1}{n}$，$i = 1, \ldots, n$
2. 对于 $t = 1, \ldots, T$：
   - 使用权重 $w^{(t)}$ 训练弱分类器 h_t
   - 计算加权错误率：
   $$\epsilon_t = \sum_{i: h_t(x_i) \neq y_i} w_i^{(t)} \tag{39}$$
   - 计算分类器权重：
   $$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) \tag{40}$$
   - 更新样本权重：
   $$w_i^{(t+1)} = w_i^{(t)} \exp(-\alpha_t y_i h_t(x_i)) \tag{41}$$
   - 归一化权重

3. 最终分类器：

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right) \tag{42}$$

12.2.3 Gradient Boosting

梯度提升将 boosting 看作优化问题。每次添加一个新模型来减少损失函数：

$$F_m(x) = F_{m-1}(x) + \eta h_m(x) \tag{43}$$

其中 h_m 拟合当前模型的负梯度（残差）。

Gradient Boosting 的威力：

- XGBoost、LightGBM、CatBoost 等现代实现在 Kaggle 等竞赛中称霸
- 在结构化数据（表格数据）上常常优于神经网络
- 可以灵活选择损失函数

13 朴素贝叶斯（Naive Bayes）

13.1 贝叶斯定理

贝叶斯定理是概率论的基石：

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} \tag{44}$$

在分类问题中，我们要计算后验概率：

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} \tag{45}$$

选择后验概率最大的类别：

$$\hat{y} = \arg\max_y P(y|x) = \arg\max_y P(x|y)P(y) \tag{46}$$

13.2 ”朴素”假设

问题是 $P(x|y)$ 难以估计（x 是高维的）。朴素贝叶斯做了一个强假设：给定类别 y，特征之间相互独立：

$$P(x|y) = \prod_{j=1}^{d} P(x_j|y) \tag{47}$$

这个假设通常不成立，但在实践中效果却出奇地好！

13.3 朴素贝叶斯分类器

$$\hat{y} = \arg\max_{y} P(y) \prod_{j=1}^{d} P(x_j|y) \tag{48}$$

对于不同类型的特征，有不同的朴素贝叶斯变体：

- **高斯朴素贝叶斯**：连续特征，假设 $P(x_j|y) \sim \mathcal{N}(\mu_{jy}, \sigma_{jy}^2)$
- **多项式朴素贝叶斯**：离散特征（如文本词频），假设多项分布
- **伯努利朴素贝叶斯**：二值特征，假设伯努利分布

13.4 朴素贝叶斯的应用

虽然简单且假设强，朴素贝叶斯在某些领域表现很好：

- **垃圾邮件过滤**：经典应用，训练快、效果好
- **文本分类**：情感分析、主题分类等
- **医疗诊断**：基于症状预测疾病

优点是训练极快、不需要大量数据、可解释性强。

14 k 近邻（k-Nearest Neighbors, kNN）

14.1 非参数方法

与前面的参数方法不同，kNN 是**非参数方法**——它不学习显式的参数，而是直接记住所有训练数据。

14.2 kNN 算法

预测过程：

1. 计算测试点 x 与所有训练点的距离
2. 找到最近的 k 个邻居
3. 对于分类：返回这 k 个邻居中最常见的类别
4. 对于回归：返回这 k 个邻居的标签平均值

14.3 距离度量

常用距离度量：

- **欧氏距离**：$d(x, x') = \sqrt{\sum_{j=1}^{d}(x_j - x'_j)^2}$
- **曼哈顿距离**：$d(x, x') = \sum_{j=1}^{d} |x_j - x'_j|$
- **闵可夫斯基距离**：$d(x, x') = \left(\sum_{j=1}^{d} |x_j - x'_j|^p\right)^{1/p}$

14.4 超参数 k 的选择

- k 太小：对噪声敏感，高方差，容易过拟合
- k 太大：决策边界过于平滑，高偏差，欠拟合
- 通常通过交叉验证选择 k

14.5 kNN 的优缺点

优点：

- 简单直观，易于实现
- 无需训练（lazy learning）
- 对数据分布无假设
- 可以处理多分类问题

缺点：

- 预测慢（需要计算与所有训练点的距离）
- 存储成本高（需要保存所有训练数据）
- 对特征缩放敏感
- 在高维空间表现差（维度灾难）

15 总结：经典方法的遗产

这些”有点过时”的方法为什么仍然重要？

15.1 理论基础

这些方法奠定了机器学习的理论基础：

- 感知机启发了神经网络
- SVM 引入了核方法和间隔理论
- 决策树和集成方法展示了组合弱学习器的威力
- 贝叶斯方法强调了概率建模

15.2 实用价值

在某些场景下，这些经典方法依然是最佳选择：

- **小数据集**：深度学习需要大量数据，经典方法在小数据上更稳健
- **可解释性**：决策树、朴素贝叶斯等方法的决策过程清晰可见
- **结构化数据**：XGBoost 等基于决策树的方法在表格数据上常常优于神经网络
- **快速原型**：朴素贝叶斯、kNN 等方法实现简单，适合快速验证想法

15.3 与深度学习的关系

现代深度学习继承了许多经典方法的思想：

- 神经网络是多层感知机的延伸
- Dropout 类似于 Bagging 的思想
- Batch Normalization 借鉴了特征缩放的重要性
- 注意力机制可以看作一种软性的 kNN

理解这些经典方法，能让我们更深刻地理解现代 AI 技术的本质。正如牛顿所说：”如果我看得更远，那是因为我站在巨人的肩膀上。” 这些经典方法就是我们的巨人。

人工智能导论课程讲义

第三部分：深度学习革命

16 引言：AI 的文艺复兴

1980 年代末，一个算法的出现改变了一切——反向传播（Backpropagation）。它解决了多层神经网络的训练难题，为今天的深度学习革命埋下了种子。虽然这个算法从提出到真正爆发经历了近 30 年，但它最终引领我们进入了新时代的人工智能。

17 反向传播算法（Backpropagation）

17.1 神经网络的前向传播

考虑一个简单的多层感知机（MLP），包含输入层、隐藏层和输出层。

符号定义：

- $x \in \mathbb{R}^{d_0}$：输入向量
- $W^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$：第 l 层的权重矩阵
- $b^{(l)} \in \mathbb{R}^{d_l}$：第 l 层的偏置向量
- $z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}$：第 l 层的线性输出（pre-activation）
- $a^{(l)} = \sigma(z^{(l)})$：第 l 层的激活输出，其中 σ 是激活函数
- $a^{(0)} = x$：输入层

前向传播流程：

$$z^{(1)} = W^{(1)} x + b^{(1)} \tag{49}$$

$$a^{(1)} = \sigma(z^{(1)}) \tag{50}$$

$$z^{(2)} = W^{(2)} a^{(1)} + b^{(2)} \tag{51}$$

$$a^{(2)} = \sigma(z^{(2)}) \tag{52}$$

$$\vdots \tag{53}$$

$$\hat{y} = a^{(L)} \tag{54}$$

17.2 损失函数

对于单个样本 (x, y)，定义损失函数 $\mathcal{L}(\hat{y}, y)$。常见的损失函数：

- **回归**：均方误差 $\mathcal{L} = \frac{1}{2}\|\hat{y} - y\|^2$
- **分类**：交叉熵 $\mathcal{L} = -\sum_k y_k \log \hat{y}_k$

总损失是所有样本损失的平均：

$$J = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(\hat{y}_i, y_i) \tag{55}$$

17.3 梯度下降的挑战

要用梯度下降更新参数：

$$W^{(l)} \leftarrow W^{(l)} - \eta \frac{\partial J}{\partial W^{(l)}} \tag{56}$$

问题是：如何计算 $\frac{\partial J}{\partial W^{(l)}}$？尤其是对于深层网络，手工推导几乎不可能。

17.4 链式法则：反向传播的核心

反向传播的本质是**高效地应用链式法则**。

对于最后一层（第 L 层）：

$$\frac{\partial \mathcal{L}}{\partial W^{(L)}} = \frac{\partial \mathcal{L}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial W^{(L)}} \tag{57}$$

定义 $\delta^{(L)} = \frac{\partial \mathcal{L}}{\partial z^{(L)}}$，称为第 L 层的误差项。

对于输出层：

$$\delta^{(L)} = \frac{\partial \mathcal{L}}{\partial a^{(L)}} \odot \sigma'(z^{(L)}) \tag{58}$$

其中 $\odot$ 表示逐元素乘法。

17.5 反向传播递推公式

关键洞察：第 l 层的误差可以从第 $l+1$ 层的误差推导出来！

$$\delta^{(l)} = (W^{(l+1)})^T \delta^{(l+1)} \odot \sigma'(z^{(l)}) \tag{59}$$

这就是” 反向传播” 名称的由来——误差从输出层反向传播到输入层。

17.6 梯度计算

有了误差项 $\delta^{(l)}$，梯度很容易计算：

$$\frac{\partial \mathcal{L}}{\partial W^{(l)}} = \delta^{(l)} (a^{(l-1)})^T \tag{60}$$

$$\frac{\partial \mathcal{L}}{\partial b^{(l)}} = \delta^{(l)} \tag{61}$$

17.7 反向传播算法总结

算法流程：

1. **前向传播**：计算每层的 $z^{(l)}$ 和 $a^{(l)}$

2. **计算输出层误差**：$\delta^{(L)} = \frac{\partial \mathcal{L}}{\partial a^{(L)}} \odot \sigma'(z^{(L)})$

3. **反向传播误差**：对 $l = L-1, L-2, \ldots, 1$：

$$\delta^{(l)} = (W^{(l+1)})^T \delta^{(l+1)} \odot \sigma'(z^{(l)}) \tag{62}$$

4. **计算梯度**：

$$\frac{\partial \mathcal{L}}{\partial W^{(l)}} = \delta^{(l)} (a^{(l-1)})^T \tag{63}$$

$$\frac{\partial \mathcal{L}}{\partial b^{(l)}} = \delta^{(l)} \tag{64}$$

5. **更新参数**：

$$W^{(l)} \leftarrow W^{(l)} - \eta \frac{\partial J}{\partial W^{(l)}} \tag{65}$$

$$b^{(l)} \leftarrow b^{(l)} - \eta \frac{\partial J}{\partial b^{(l)}} \tag{66}$$

17.8 为什么反向传播如此重要？

- **计算效率**：时间复杂度与前向传播相同，约为 $O(W)$，其中 W 是参数总数
- **自动化**：现代深度学习框架（PyTorch、TensorFlow）实现了自动微分，自动计算梯度
- **通用性**：适用于任何可微的网络结构和损失函数

反向传播是深度学习的基石。没有它，训练深层神经网络几乎不可能。

18 激活函数的演化

在讨论具体网络架构之前，我们需要了解激活函数的重要性。

18.1 为什么需要非线性激活函数？

如果没有非线性激活函数，多层网络等价于单层线性变换，无法拟合复杂函数。

18.2 常见激活函数

18.2.1 Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{67}$$

优点：输出在 $(0, 1)$ 之间，可解释为概率

缺点：

- 梯度消失：当 $|x|$ 很大时，$\sigma'(x) \approx 0$
- 输出不是零中心的

18.2.2 Tanh

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{68}$$

优点：输出在 $(-1, 1)$ 之间，零中心

缺点：仍然存在梯度消失问题

18.2.3 ReLU（Rectified Linear Unit）

$$\mathrm{ReLU}(x) = \max(0, x) \tag{69}$$

优点：

- 计算简单
- 缓解梯度消失（正半轴梯度恒为 1）
- 训练速度快
- 生物学上更合理（神经元的稀疏激活）

缺点：

- ” 神经元死亡”：如果神经元输出始终为负，梯度永远为 0

ReLU 及其变体（Leaky ReLU、PReLU、ELU 等）成为现代深度学习的标准选择。

19 卷积神经网络（CNN）

19.1 全连接网络的局限性

对于图像任务，全连接网络存在严重问题：

- **参数爆炸**：一张 $224 \times 224 \times 3$ 的图像有 150,528 个像素，第一层若有 1000 个神经元，就需要 1.5 亿个参数！
- **忽视空间结构**：将图像展平成向量，丢失了像素之间的空间关系
- **缺乏平移不变性**：同一物体在不同位置需要重新学习

19.2 卷积操作

卷积是 CNN 的核心操作，保留了图像的空间结构。

19.2.1 二维卷积

给定输入 $X \in \mathbb{R}^{H \times W}$ 和卷积核（滤波器）$K \in \mathbb{R}^{k \times k}$，卷积操作为：

$$(X * K)_{ij} = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} X_{i+m,j+n} \cdot K_{mn} \tag{70}$$

19.2.2 卷积的三大特性

1. **局部连接**：每个神经元只连接输入的一小块区域（感受野），大幅减少参数
2. **参数共享**：同一个卷积核在整个图像上滑动，所有位置共享相同参数
3. **平移等变性**：输入平移，输出也相应平移

19.3 CNN 的基本组件

19.3.1 卷积层（Convolutional Layer）

输入：$X \in \mathbb{R}^{C_{in} \times H \times W}$（$C_{in}$ 个通道）
输出：$Y \in \mathbb{R}^{C_{out} \times H' \times W'}$（$C_{out}$ 个通道）
参数：C_{out} 个卷积核，每个大小为 $C_{in} \times k \times k$

19.3.2 池化层（Pooling Layer）

池化用于降采样，减少特征图尺寸。

最大池化（Max Pooling）：

$$Y_{ij} = \max_{m,n \in \text{pool}} X_{i+m,j+n} \tag{71}$$

平均池化（Average Pooling）：

$$Y_{ij} = \frac{1}{k^2} \sum_{m,n \in \text{pool}} X_{i+m,j+n} \tag{72}$$

池化的作用：

- 减少计算量
- 提供平移不变性
- 扩大感受野

19.3.3 全连接层（Fully Connected Layer）

CNN 的最后通常是全连接层，将特征映射到类别。

19.4 典型 CNN 架构

经典 CNN 通常遵循以下模式：

$$\text{INPUT} \to [\text{CONV} \to \text{RELU} \to \text{POOL}]^* \to \text{FC} \to \text{OUTPUT} \tag{73}$$

20 AlexNet：深度学习的觉醒（2012）

20.1 ImageNet 挑战赛

ImageNet 是一个包含 1400 万张图像、2 万个类别的大规模图像数据集。ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 是计算机视觉领域的顶级竞赛。

2012 年之前，最好的方法是基于手工特征（SIFT、HOG）+ SVM，错误率约 25%。

20.2 AlexNet 的架构

AlexNet 由 Alex Krizhevsky、Ilya Sutskever 和 Geoffrey Hinton 提出，在 ILSVRC 2012 上以 15.3% 的 top-5 错误率获得冠军，比第二名低 10 个百分点！

网络结构：

1. **Conv1**：96 个 11×11 卷积核，stride 4，ReLU，Max Pooling
2. **Conv2**：256 个 5×5 卷积核，ReLU，Max Pooling
3. **Conv3**：384 个 3×3 卷积核，ReLU
4. **Conv4**：384 个 3×3 卷积核，ReLU
5. **Conv5**：256 个 3×3 卷积核，ReLU，Max Pooling
6. **FC6**：4096 个神经元，ReLU，Dropout
7. **FC7**：4096 个神经元，ReLU，Dropout
8. **FC8**：1000 个神经元（输出层，1000 类）

总参数量：约 6000 万

20.3 AlexNet 的创新

1. **ReLU 激活函数**：首次在大规模网络中使用 ReLU，训练速度比 tanh 快 6 倍
2. **Dropout**：随机丢弃 50% 的神经元，有效防止过拟合
3. **数据增强**：随机裁剪、水平翻转、颜色抖动等，扩充训练数据
4. **GPU 训练**：使用两块 GTX 580 GPU 并行训练，开启了深度学习的 GPU 时代
5. **局部响应归一化（LRN）**：虽然后来被 Batch Normalization 取代，但当时起到了正则化作用

20.4 AlexNet 的历史意义

AlexNet 证明了深度学习在大规模视觉任务上的优势，引发了深度学习的热潮。从此，CNN 成为计算机视觉的标准工具，深度学习开始席卷各个领域。

21 网络加深：VGGNet 和 GoogLeNet

21.1 VGGNet（2014）

VGGNet 的核心思想：**网络越深，性能越好**。

设计原则：

- 只使用 3×3 卷积核

- 两个 3×3 卷积等价于一个 5×5 的感受野，但参数更少
- 使用 2×2 的 Max Pooling

VGG16 架构：

- 13 个卷积层 + 3 个全连接层 = 16 层
- 通道数从 64 逐渐增加到 512
- 总参数量：约 1.38 亿（主要在全连接层）

VGGNet 展示了"deeper is better"，但也暴露了问题：参数过多、计算量大、训练困难。

21.2 GoogLeNet / Inception（2014）

GoogLeNet 的核心思想：**网络宽度也很重要**。

21.2.1 Inception 模块

与其选择单一的卷积核大小，不如同时使用多个尺度的卷积核：

- 1×1 卷积
- 3×3 卷积
- 5×5 卷积
- 3×3 Max Pooling

将这些操作的输出在通道维度拼接（concatenate），让网络自己学习哪种尺度的特征更重要。

21.2.2 1×1 卷积的妙用

1×1 卷积看似无用，实际上可以：

- **降维**：减少通道数，降低计算量
- **增加非线性**：引入额外的 ReLU

通过 1×1 卷积降维后再做 3×3 或 5×5 卷积，可以大幅减少参数。
GoogLeNet 有 22 层，但只有 500 万参数，比 AlexNet 少 12 倍!

22 ResNet：残差革命（2015）

22.1 深度网络的退化问题

理论上，更深的网络至少不应该比浅网络差（可以让额外的层学习恒等映射）。但实验发现：当网络超过一定深度后，训练误差反而增加！这不是过拟合，而是**退化问题（degradation problem）**。

原因：深层网络难以优化，梯度消失/爆炸问题严重。

22.2 残差学习

何恺明等人提出了残差学习（Residual Learning）：与其学习期望的映射 $H(x)$，不如学习残差 $F(x) = H(x) - x$。

22.2.1 残差块（Residual Block）

$$y = F(x, \{W_i\}) + x \tag{74}$$

其中 $F(x, \{W_i\})$ 是残差函数，通常是两层卷积：

$$F(x) = W_2 \cdot \mathrm{ReLU}(W_1 \cdot x) \tag{75}$$

加上跳跃连接（skip connection）后：

$$y = \mathrm{ReLU}(F(x) + x) \tag{76}$$

22.3 为什么残差连接有效？

22.3.1 梯度流动

反向传播时：

$$\frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial y}\left(\frac{\partial F}{\partial x} + 1\right) \tag{77}$$

注意到有一个直接的"+1" 项，梯度可以直接传播，不会消失！

22.3.2 集成学习视角

ResNet 可以看作是多个不同深度的网络的集成。残差连接提供了多条路径，网络可以选择使用不同深度的特征。

22.3.3 更容易优化

学习恒等映射很难，但学习零映射很容易（$F(x) = 0$）。残差学习让网络更容易学习到接近恒等的映射。

22.4 ResNet 架构

ResNet 的不同深度：

- ResNet-18、ResNet-34：使用基本残差块（两个 3×3 卷积）
- ResNet-50、ResNet-101、ResNet-152：使用瓶颈残差块（1×1 降维 $\rightarrow$ 3×3 卷积 $\rightarrow$ 1×1 升维）

ResNet-152：

- 152 层深
- ILSVRC 2015 冠军，top-5 错误率 3.57%（超越人类水平的 5%）
- 证明了网络可以训练得非常深

22.5 ResNet 的影响

残差连接成为现代深度学习的标配：

- DenseNet：密集连接，每层连接到所有之前的层
- ResNeXt：残差 + Inception 的组合
- EfficientNet：使用神经架构搜索优化的 ResNet 变体

ResNet 证明了：深度是深度学习的关键，而残差连接是训练深层网络的钥匙。

23 Transformer：注意力即一切（2017）

23.1 从 CNN 到 Transformer

CNN 在计算机视觉上大获成功，但它有局限性：

- 感受野有限（需要很深才能看到全局）
- 对长距离依赖建模困难

在自然语言处理（NLP）领域，循环神经网络（RNN）及其变体 LSTM、GRU 长期占据主导地位，但 RNN 也有问题：

- 顺序计算，无法并行
- 长序列梯度消失/爆炸
- 难以捕捉远距离依赖

2017 年，Vaswani 等人提出了 Transformer，标题是”Attention Is All You Need”。Transformer 完全抛弃了卷积和循环，仅使用注意力机制，却在机器翻译任务上超越了所有先前方法。

23.2 自注意力机制（Self-Attention）

23.2.1 注意力的直觉

给定一个序列，自注意力让每个位置关注序列中的所有其他位置，动态聚合信息。

例如，在句子”The animal didn’t cross the street because it was too tired” 中，”it” 应该指向”animal” 而非”street”。注意力机制可以学习到这种关系。

23.2.2 Scaled Dot-Product Attention

给定输入序列 $X \in \mathbb{R}^{n \times d}$（$n$ 个 token，每个 d 维），首先通过三个线性变换得到 Query、Key、Value：

$$Q = XW_Q, \quad Q \in \mathbb{R}^{n \times d_k} \tag{78}$$

$$K = XW_K, \quad K \in \mathbb{R}^{n \times d_k} \tag{79}$$

$$V = XW_V, \quad V \in \mathbb{R}^{n \times d_v} \tag{80}$$

注意力计算：

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{81}$$

直觉理解：

- QK^T：计算每对 token 之间的相似度（注意力分数）
- softmax：归一化为概率分布
- 乘以 V：加权求和，聚合信息
- $\sqrt{d_k}$：缩放因子，防止点积过大导致 softmax 饱和

23.2.3 Multi-Head Attention

单个注意力头可能只关注某一种模式。多头注意力使用多组 (Q, K, V)，并行计算多个注意力：

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{82}$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \tag{83}$$

不同的头可以学习到不同类型的关系（语法、语义等）。

23.3 Transformer 架构

23.3.1 Encoder-Decoder 结构

Transformer 最初为机器翻译设计，包含 Encoder 和 Decoder。

Encoder：

- 输入嵌入（Input Embedding）+ 位置编码（Positional Encoding）
- N 个相同的层，每层包含：
  - Multi-Head Self-Attention
  - Add & Norm（残差连接 + Layer Normalization）
  - Feed-Forward Network（两层 MLP）
  - Add & Norm

Decoder：

- 输出嵌入 + 位置编码
- N 个相同的层，每层包含：
  - Masked Multi-Head Self-Attention（防止看到未来信息）
  - Add & Norm
  - Multi-Head Cross-Attention（关注 Encoder 输出）
  - Add & Norm
  - Feed-Forward Network
  - Add & Norm

23.3.2 位置编码

由于注意力机制本身不包含位置信息（是置换不变的），需要显式加入位置编码。Transformer 使用正弦和余弦函数：

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right) \tag{84}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right) \tag{85}$$

其中 pos 是位置，i 是维度索引。

23.4 Transformer 的优势

1. **并行计算**：所有位置可以同时计算，不像 RNN 需要顺序处理

2. **长距离依赖**：任意两个位置都可以直接交互，路径长度为常数 $O(1)$

3. **可解释性**：注意力权重可视化，能看到模型在关注什么

4. **灵活性**：适用于各种序列任务

23.5 Transformer 的局限性

- **计算复杂度**：自注意力的时间和空间复杂度为 $O(n^2 d)$，对于长序列代价高昂
- **需要大量数据**：缺乏归纳偏置（如 CNN 的局部性），需要更多数据学习
- **位置编码**：固定的位置编码可能不是最优的

23.6 Transformer 的变体和改进

23.6.1 高效注意力

为了处理长序列，出现了多种高效注意力机制：

- **Sparse Attention**（Sparse Transformer）：只关注部分位置
- **Linformer**：将 $O(n^2)$ 降到 $O(n)$
- **Performer**：使用核方法近似注意力
- **Flash Attention**：优化 GPU 内存访问，加速训练

23.6.2 Vision Transformer (ViT, 2020)

将 Transformer 应用到计算机视觉：

- 将图像分割成 patch（如 16×16）
- 每个 patch 线性投影成 token
- 加上位置编码，输入到标准 Transformer Encoder

ViT 在大规模数据集上超越了 CNN，证明了 Transformer 的通用性。

24 从 Transformer 到大语言模型

24.1 预训练-微调范式

Transformer 催生了新的训练范式：

1. **预训练（Pre-training）**：在大规模无标注数据上自监督学习

2. **微调（Fine-tuning）**：在下游任务的少量标注数据上调整

24.2 里程碑模型

24.2.1 BERT（2018）

Bidirectional Encoder Representations from Transformers

核心思想：双向编码器，通过**掩码语言模型（Masked Language Modeling, MLM）**预训练。

MLM 任务：随机遮盖 15% 的 token，让模型预测被遮盖的词。这让模型学习到双向的上下文表示。

BERT 在多个 NLP 任务上刷新记录，开启了预训练大模型时代。

24.2.2 GPT 系列（2018-2023）

Generative Pre-trained Transformer

核心思想：仅使用 Decoder，通过**自回归语言建模（Autoregressive Language Modeling）**预训练。

训练目标：给定前 $t-1$ 个词，预测第 t 个词：

$$\max_{\theta} \sum_{i} \log P(x_i | x_{<i}; \theta) \tag{86}$$

GPT 的演进：

- **GPT-1**（2018）：1.17 亿参数，证明了预训练-微调的有效性
- **GPT-2**（2019）：15 亿参数，展示了 zero-shot 学习能力
- **GPT-3**（2020）：1750 亿参数，few-shot 学习能力惊人，无需微调就能完成多种任务
- **GPT-4**（2023）：参数量未公开（据传超万亿），多模态能力，接近 AGI 的表现

24.2.3 规模定律（Scaling Laws）

OpenAI 等研究发现：模型性能与三个因素呈现幂律关系：

- 模型大小（参数量）
- 数据集大小
- 计算量

这意味着：**更大的模型、更多的数据、更多的计算 = 更好的性能**。这引发了军备竞赛式的模型规模扩张。

24.3 涌现能力（Emergent Abilities）

当模型规模达到某个临界点后，会突然出现小模型不具备的能力：

- **In-context Learning**：给几个示例，模型就能理解新任务
- **Chain-of-Thought Reasoning**：分步推理，解决复杂问题
- **指令遵循**：理解并执行自然语言指令

这些能力的出现是深度学习最令人兴奋的发现之一。

25 总结：深度学习的演化路径

25.1 算法演进

时期	代表	关键突破
1980s	Backprop	训练多层网络成为可能
2012	AlexNet	GPU + ReLU + Dropout
2014-2015	VGG, ResNet	网络深度，残差连接
2017	Transformer	注意力机制，摆脱卷积和循环
2018+	BERT, GPT	预训练大模型，规模定律
2023+	GPT-4, Claude	多模态，接近 AGI

25.2 核心洞察

1. **深度是关键**：深层网络可以学习分层的特征表示，从低级到高级
2. **架构很重要**：好的归纳偏置（卷积的局部性、Transformer 的全局性）能提高效率
3. **残差连接是魔法**：解决了深度网络的训练难题

4. **注意力是通用语言**：从 NLP 到 CV，注意力机制统一了不同领域

5. **规模带来智能**：大模型在足够的数据和计算下展现出惊人的能力

25.3 未来展望

深度学习仍在快速发展：

- **更高效的架构**：减少 Transformer 的计算复杂度
- **多模态融合**：统一视觉、语言、音频等
- **更好的训练方法**：自监督学习、强化学习
- **可解释性**：理解神经网络内部的工作机制
- **小模型的复兴**：知识蒸馏、稀疏化、量化
- **AGI 的探索**：从狭义 AI 走向通用 AI

从感知机到 Transformer，从 AlexNet 到 GPT-4，我们见证了人工智能从” 人工智障” 到” 通用智能” 的演化。深度学习不仅改变了 AI，也正在改变世界。

25.4 实践建议

对于学习者：

1. **掌握基础**：反向传播、卷积、注意力机制是核心

2. **动手实践**：用 PyTorch/TensorFlow 实现经典模型

3. **阅读论文**：追踪最新进展，理解设计思想

4. **参与项目**：在实际问题中应用深度学习

5. **关注工具**：Hugging Face、LangChain 等让 AI 更易用

深度学习的革命才刚刚开始。理解这些经典方法，能让我们更好地把握未来的方向。正如 Yann LeCun 所说：”Deep learning is not just a set of techniques, it’s a way of thinking about intelligence.”

26 附录：关键数学推导

26.1 反向传播的详细推导

考虑简单的两层网络：

$$z^{(1)} = W^{(1)}x + b^{(1)} \tag{87}$$

$$a^{(1)} = \sigma(z^{(1)}) \tag{88}$$

$$z^{(2)} = W^{(2)}a^{(1)} + b^{(2)} \tag{89}$$

$$\hat{y} = a^{(2)} = \sigma(z^{(2)}) \tag{90}$$

损失函数为 $\mathcal{L} = \frac{1}{2}\|\hat{y} - y\|^2$。

第二层梯度：

$$\frac{\partial \mathcal{L}}{\partial z^{(2)}} = \frac{\partial \mathcal{L}}{\partial a^{(2)}} \odot \sigma'(z^{(2)}) \tag{91}$$

$$= (\hat{y} - y) \odot \sigma'(z^{(2)}) = \delta^{(2)} \tag{92}$$

$$\frac{\partial \mathcal{L}}{\partial W^{(2)}} = \delta^{(2)}(a^{(1)})^T \tag{93}$$

$$\frac{\partial \mathcal{L}}{\partial b^{(2)}} = \delta^{(2)} \tag{94}$$

第一层梯度：

$$\frac{\partial \mathcal{L}}{\partial a^{(1)}} = (W^{(2)})^T\delta^{(2)} \tag{95}$$

$$\frac{\partial \mathcal{L}}{\partial z^{(1)}} = \frac{\partial \mathcal{L}}{\partial a^{(1)}} \odot \sigma'(z^{(1)}) \tag{96}$$

$$= (W^{(2)})^T\delta^{(2)} \odot \sigma'(z^{(1)}) = \delta^{(1)} \tag{97}$$

$$\frac{\partial \mathcal{L}}{\partial W^{(1)}} = \delta^{(1)}x^T \tag{98}$$

$$\frac{\partial \mathcal{L}}{\partial b^{(1)}} = \delta^{(1)} \tag{99}$$

这就是反向传播的完整推导。

26.2 注意力机制的计算复杂度

对于长度为 n 的序列，维度为 d：

时间复杂度：

- 计算 QK^T：$O(n^2d)$

- Softmax：$O(n^2)$
- 乘以 V：$O(n^2d)$
- 总计：$O(n^2d)$

空间复杂度：

- 存储注意力矩阵：$O(n^2)$
- 这是长序列的瓶颈！

对比 RNN 的 $O(nd^2)$ 复杂度，Transformer 在 $n > d$ 时更慢，但可以并行化。