

深度学习 — 第二讲

江伟森

FNN	如何定义从输入到预测的计算
Backpropagation	如何高效计算损失对参数的梯度
Automatic Differentiation	框架如何自动执行梯度计算
Convergence	收敛性
Generalization	泛化性

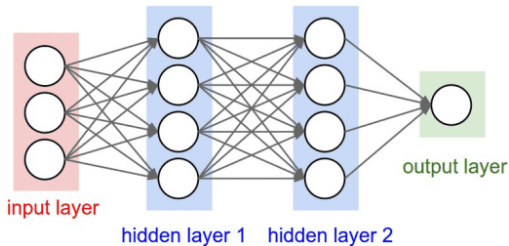
什么是前馈神经网络？

定义

前馈神经网络 (Feedforward Neural Network, FNN): 信息从输入层出发, 逐层流向输出层, 网络中不存在循环连接。

基本特征:

- 每一层接收上一层的输出, 计算后传给下一层
- 多层感知机 (MLP) 是最典型的 FNN
- 整个网络定义了一个从输入到输出的确定性映射



从线性模型到神经元

线性模型：

$$z = \mathbf{w}^\top \mathbf{x} + b$$

加入激活函数 σ ，得到一个神经元：

$$a = \sigma(z) = \sigma(\mathbf{w}^\top \mathbf{x} + b)$$

符号	含义
$\mathbf{x}$	输入向量
$\mathbf{w}, b$	可学习参数（权重与偏置）
$z = \mathbf{w}^\top \mathbf{x} + b$	预激活值 (pre-activation)
$a = \sigma(z)$	神经元输出 (activation)

一个神经元 = 线性变换 + 非线性激活。

为什么需要非线性?

考虑连续两个线性层（不加激活函数）：

$$\mathbf{h} = W_1 \mathbf{x} + \mathbf{b}_1$$

$$\mathbf{y} = W_2 \mathbf{h} + \mathbf{b}_2$$

将 $\mathbf{h}$ 代入：

$$\mathbf{y} = W_2(W_1 \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 = \underbrace{(W_2 W_1)}_{\tilde{W}} \mathbf{x} + \underbrace{(W_2 \mathbf{b}_1 + \mathbf{b}_2)}_{\tilde{\mathbf{b}}}$$

结论

没有激活函数时，无论堆叠多少线性层，整体仍然是一个线性模型。

常见激活函数

名称	定义	特点	典型场景
Sigmoid	$\frac{1}{1+e^{-z}}$	输出 (0, 1); 易梯度饱和	二分类输出层
Tanh	$\frac{e^z - e^{-z}}{e^z + e^{-z}}$	输出 (-1, 1); 零中心	RNN
ReLU	$\max(0, z)$	简单高效; 死亡神经元	最常用
GELU	$z\Phi(z)$	平滑门控	Transformer
SiLU	$z\sigma(z)$	平滑、自门控	现代深度网络

ReLU 是目前使用最广泛的激活函数:

$$\text{ReLU}(z) = \max(0, z), \quad \text{ReLU}'(z) = \begin{cases} 1 & z > 0 \\ 0 & z < 0 \end{cases}$$

计算简单, 梯度不饱和 (正区间恒为 1), 但 $z < 0$ 时梯度为零 (“死亡神经元”)。

单隐藏层 FNN

最简单的 FNN：输入层 $\rightarrow$ 隐藏层 $\rightarrow$ 输出层。

数学表达：

$$\mathbf{h} = \sigma(W_1 \mathbf{x} + \mathbf{b}_1)$$

$$\hat{\mathbf{y}} = g(W_2 \mathbf{h} + \mathbf{b}_2)$$

隐藏层：线性变换 + 激活

输出层：线性变换 + 输出激活

- σ ：隐藏层激活函数（如 ReLU）
- g ：输出层激活函数（取决于任务）
- 隐藏层不直接给出预测，而是学习中间表示（representation）

关键思想

隐藏层将输入 $\mathbf{x}$ 变换到一个新的特征空间，使得输出层更容易完成任务。

深层前馈网络

L 层 FNN 的一般形式:

$$\begin{aligned} \mathbf{a}^{(0)} &= \mathbf{x} \\ \mathbf{z}^{(l)} &= \mathbf{W}^{(l)} \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}, \quad \mathbf{a}^{(l)} = \sigma^{(l)}(\mathbf{z}^{(l)}), \quad l = 1, 2, \dots, L \end{aligned}$$

可训练参数: $\theta = \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \dots, \mathbf{W}^{(L)}, \mathbf{b}^{(L)}\}$

逐层理解:

- 每层执行一次特征变换: 仿射变换 + 非线性
- 后一层在前一层表示的基础上继续学习
- 整个网络是多个函数的复合

FNN 就是函数复合

定义每层变换 $f_l(\cdot) = \sigma^{(l)}(W^{(l)}(\cdot) + \mathbf{b}^{(l)})$, 则整个网络:

$$f_{\theta}(\mathbf{x}) = f_L \circ f_{L-1} \circ \cdots \circ f_1(\mathbf{x})$$

这个视角有两个重要意义:

- ① 前向传播就是按 $l = 1, 2, \dots, L$ 的顺序逐层计算
- ② 反向传播就是对复合函数应用链式法则

伏笔

$$\frac{\partial L}{\partial W^{(l)}} = \frac{\partial L}{\partial \mathbf{a}^{(L)}} \cdot \frac{\partial \mathbf{a}^{(L)}}{\partial \mathbf{a}^{(L-1)}} \cdots \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} \cdot \frac{\partial \mathbf{z}^{(l)}}{\partial W^{(l)}}$$

前向传播是函数复合, 反向传播是对复合函数反向应用链式法则。

张量形状

设 Batch size B , 输入维度 d , 隐藏维度 h , 输出类别数 c 。

单隐藏层网络 $\hat{Y} = g(\sigma(XW_1 + \mathbf{b}_1)W_2 + \mathbf{b}_2)$ 中各张量形状:

张量	形状	说明
X	$B \times d$	一批输入样本
W_1	$d \times h$	第一层权重
$\mathbf{b}_1$	h	第一层偏置 (广播到 B 行)
$H = \sigma(XW_1 + \mathbf{b}_1)$	$B \times h$	隐藏层输出
W_2	$h \times c$	第二层权重
$\mathbf{b}_2$	c	第二层偏置
$\hat{Y}$	$B \times c$	一批预测结果

参数数量

对于全连接层 $\mathbf{z} = \mathbf{x}W + \mathbf{b}$ ，输入维度 d_{in} ，输出维度 d_{out} ：

$$\text{参数量} = \underbrace{d_{\text{in}} \times d_{\text{out}}}_W + \underbrace{d_{\text{out}}}_b$$

课堂例题

MNIST 手写数字分类器：输入 784，隐藏层 256，输出 10。

第 1 层： $784 \times 256 + 256 = 200,960$

第 2 层： $256 \times 10 + 10 = 2,570$

总计： **203,530**

现代网络（如 ResNet-50: 25M, GPT-3: 175B）的参数数量远超于此，这正是“过参数化”的来源（下半节课讨论）。

批量化与向量化计算

单个样本：

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

一批 B 个样本：

$$\mathbf{Z} = \mathbf{X}\mathbf{W} + \mathbf{b}$$

其中 $\mathbf{X} \in \mathbb{R}^{B \times d}$ ，偏置 $\mathbf{b}$ 沿 batch 维度广播 (broadcasting)。

深度学习框架 (PyTorch, JAX 等) 的核心就是高效的张量运算引擎。

为什么要批量化？

- ① 向量化：矩阵乘法比循环快几个数量级
- ② GPU 并行：GPU 擅长大规模并行矩阵运算
- ③ 梯度估计：mini-batch 提供梯度的随机近似
- ④ 内存效率：batch 大小可控制显存占用

完整前向传播例题

网络：2 输入 $\rightarrow$ 2 隐藏 (ReLU) $\rightarrow$ 1 输出。给定参数：

$$W_1 = \begin{pmatrix} 0.5 & -0.3 \\ 0.2 & 0.8 \end{pmatrix}, \mathbf{b}_1 = \begin{pmatrix} 0 \\ 0.1 \end{pmatrix}, W_2 = \begin{pmatrix} 0.6 \\ -0.4 \end{pmatrix}, b_2 = 0.1, \mathbf{x} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

逐步计算：

$$\mathbf{z}_1 = W_1 \mathbf{x} + \mathbf{b}_1 = \begin{pmatrix} 0.5 - 0.6 \\ 0.2 + 1.6 \end{pmatrix} + \begin{pmatrix} 0 \\ 0.1 \end{pmatrix} = \begin{pmatrix} -0.1 \\ 1.9 \end{pmatrix}$$

$$\mathbf{h} = \text{ReLU}(\mathbf{z}_1) = \begin{pmatrix} 0 \\ 1.9 \end{pmatrix}$$

$$z_2 = W_2^\top \mathbf{h} + b_2 = 0.6 \times 0 + (-0.4) \times 1.9 + 0.1 = -0.66$$

$$\hat{y} = -0.66 \quad (\text{回归任务, 无输出激活})$$

注意：ReLU 将 $z_1^{(1)} = -0.1$ 截断为 0，该神经元对本次预测无贡献。

输出层取决于任务

隐藏层负责表示学习，输出层需要匹配具体任务：

任务	输出激活	输出形状	常用损失
回归	无 (Linear)	$B \times 1$	MSE: $\frac{1}{N} \sum (y_i - \hat{y}_i)^2$
二分类	Sigmoid	$B \times 1$	Binary CE: $-y \log \hat{y} - (1-y) \log(1-\hat{y})$
多分类	Softmax	$B \times c$	Cross-Entropy: $-\sum_k y_k \log \hat{y}_k$
多标签	多个 Sigmoid	$B \times c$	逐标签 Binary CE

Softmax: 将 logits $\mathbf{z} \in \mathbb{R}^c$ 映射为概率分布：

$$\hat{y}_k = \frac{e^{z_k}}{\sum_{j=1}^c e^{z_j}}, \quad \sum_k \hat{y}_k = 1$$

FNN 的表达能力与现代应用

逼近定理 (Universal Approximation)

单隐藏层 FNN（宽度足够大）可以以任意精度逼近任意连续函数。

但“宽度足够大”可能是指数级的——深度可以更高效地表示具有组合结构的函数。

FNN 在现代架构中仍然无处不在：

- 图像分类器的全连接分类头
- Transformer 中的 **MLP** 层：

$$\text{FFN}(\mathbf{x}) = W_2 \sigma(W_1 \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2$$

- MLP-Mixer：纯 MLP 架构做图像分类
- 推荐系统中的 MLP 特征交互层

从预测到损失

模型训练的目标是最小化经验风险 (一般也记为 $L(\theta)$):

$$\min_{\theta} \hat{R}(\theta) = \frac{1}{N} \sum_{i=1}^N \ell(f_{\theta}(\mathbf{x}_i), y_i)$$

三个角色的分工:

- ❶ **FNN** f_{θ} : 计算预测 $\hat{y} = f_{\theta}(\mathbf{x})$
- ❷ 损失函数 ℓ : 评价预测与真实标签的差距
- ❸ 优化器: 寻找使损失较小的参数 θ

核心问题

优化器需要梯度 $\nabla_{\theta} \hat{R}$ 。对于包含百万参数的深层网络, 如何高效计算这个梯度?

训练循环中的四个角色

每一步参数更新包含：

步骤	名称	作用
1	Forward Propagation	计算预测 $\hat{y}$ 与损失 L
2	Backpropagation	计算参数梯度 $\nabla_{\theta} L$
3	Optimizer	利用梯度更新参数

参数更新（最简单的梯度下降）：

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta} L$$

其中 $\eta > 0$ 是学习率。

计算图

任何前向计算都可以表示为计算图——一个有向无环图 (DAG)。

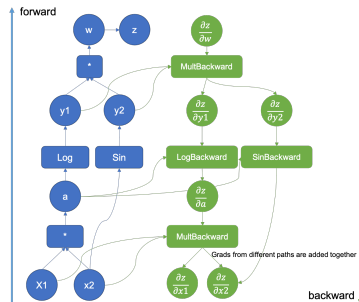
例：单神经元 + 平方损失：

$$z = wx + b$$

$$a = \sigma(z)$$

$$L = \frac{1}{2}(a - y)^2$$

计算图中每个节点代表一个基本运算（乘法、加法、 σ 、……），边代表数据的依赖关系。



关键性质

- 每个节点知道自己的局部导数（如 $\partial(wx)/\partial w = x$ ）
- 将所有节点的局部导数按链式法则组合 $\Rightarrow$ 得到任意参数的梯度

链式法则与局部梯度

如果 $y = f(u)$, $u = g(x)$, 则:

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$$

多变量推广 (向量链式法则):

$$\frac{\partial L}{\partial \mathbf{x}} = \frac{\partial L}{\partial \mathbf{y}} \cdot \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \quad (\text{Jacobian 矩阵相乘})$$

反向传播的基本规则

节点的梯度 = 上游梯度 $\times$ 局部导数

若节点 v 的输出被 $v_1, v_2, \dots, v_k$ 使用 (分支), 则:

$$\frac{\partial L}{\partial v} = \sum_{j=1}^k \frac{\partial L}{\partial v_j} \cdot \frac{\partial v_j}{\partial v} \quad (\text{梯度相加})$$

完整数值反向传播例题

沿用第 12 页的网络 ($\mathbf{x} = (1, 2)^\top$, $y = 1$, MSE 损失)。

前向 (已算): $\mathbf{z}_1 = (-0.1, 1.9)^\top$, $\mathbf{h} = (0, 1.9)^\top$, $\hat{y} = -0.66$ 。

$$L = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}(-0.66 - 1)^2 = 1.3778$$

反向: 从 L 开始逐步回传。

$$\frac{\partial L}{\partial \hat{y}} = \hat{y} - y = -1.66$$

$$\frac{\partial L}{\partial \mathbf{W}_2} = \frac{\partial L}{\partial \hat{y}} \cdot \mathbf{h} = -1.66 \times \begin{pmatrix} 0 \\ 1.9 \end{pmatrix} = \begin{pmatrix} 0 \\ -3.154 \end{pmatrix}$$

$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial \hat{y}} = -1.66$$

$$\frac{\partial L}{\partial \mathbf{h}} = \frac{\partial L}{\partial \hat{y}} \cdot \mathbf{W}_2^\top = -1.66 \times (0.6, -0.4) = (-0.996, 0.664)$$

ReLU 在 $h_1 = 0$ 处梯度为 0, 所以 $\partial L / \partial \mathbf{W}_1$ 的第一行全为零。

为什么 BP 高效?

反向传播的五个关键:

- ① 保存中间结果: 前向传播时存储各层的 $\mathbf{a}^{(l)}$, 反向时直接使用
- ② 局部导数: 每个节点只需计算自己的局部导数 (如 $\sigma'(z)$)
- ③ 反向遍历: 从输出到输入, 按拓扑逆序遍历计算图
- ④ 梯度复用: 每个节点的上游梯度只计算一次, 传给所有父节点
- ⑤ 分支求和: 若一个节点有多个下游, 梯度相加

复杂度: 一次反向传播的计算量 $\approx$ 一次前向传播的 2-3 倍。

BP 是什么?

BP 不是一种参数更新方法, 而是一种高效计算梯度的算法。
参数更新由 Optimizer (如 SGD, Adam) 负责。

自动微分如何记录计算？

以如下计算为例：

$$z = w \cdot x + b$$

$$a = \sigma(z)$$

$$L = \frac{1}{2}(a - y)^2$$

自动微分系统（如 PyTorch Autograd）在前向计算时动态记录：

步骤	运算	输入	局部导数
1	$v_1 = w \cdot x$	w, x	$\partial v_1 / \partial w = x$
2	$z = v_1 + b$	v_1, b	$\partial z / \partial v_1 = 1$
3	$a = \sigma(z)$	z	$\partial a / \partial z = \sigma'(z)$
4	$v_2 = a - y$	a, y	$\partial v_2 / \partial a = 1$
5	$L = \frac{1}{2} v_2^2$	v_2	$\partial L / \partial v_2 = v_2$

这张“记录表”就是计算图（或称 tape / Wengert list）。

反向模式自动微分

反向模式从输出开始，逆向传播梯度。

定义伴随值 (adjoint):

$$\bar{v}_i = \frac{\partial L}{\partial v_i}$$

从 $\bar{L} = 1$ 开始，反向遍历计算图：

$$\bar{v}_j += \bar{v}_i \cdot \frac{\partial v_i}{\partial v_j} \quad (\text{对 } v_j \text{ 的所有下游节点 } v_i \text{ 累加})$$

本质：计算 **vector-Jacobian product (VJP)**: $\mathbf{v}^\top J$

特点

- 一次反向传播得到损失对所有参数的梯度
- 需要先完成一次前向计算（存储中间值）
- 适合 $f: \mathbb{R}^P \rightarrow \mathbb{R}$ 且 P 很大——正好是深度学习的场景

PyTorch Autograd 示例

```
import torch

x = torch.tensor(2.0)
w = torch.tensor(0.5,
                 requires_grad=True)
b = torch.tensor(0.0,
                 requires_grad=True)
y = torch.tensor(1.0)

z = w * x + b
y_hat = torch.sigmoid(z)
loss = 0.5 * (y_hat - y)**2

loss.backward()
print(w.grad) # dL/dw
print(b.grad) # dL/db
```

逐行解释：

- `requires_grad=True`:
跟踪该张量的运算
- 前向计算时动态构建计算图
(define-by-run)
- `loss.backward()`:
反向模式自动微分
- `.grad`:
存储梯度值

Autograd 常见注意事项

1. 梯度会累积——每次反传前必须清零

```
optimizer.zero_grad() # 清除旧梯度  
loss.backward()       # 计算新梯度  
optimizer.step()      # 更新参数
```

2. 推理时关闭梯度记录（节省内存和计算）

```
with torch.no_grad():  
    prediction = model(x)
```

3. 从计算图中分离张量

```
x_detached = x.detach() # 不再跟踪梯度
```

其他：只有标量输出才能直接调用无参数的 `backward()`；原地操作 (in-place) 可能破坏反向传播需要的中间结果；数值梯度检查可用于验证自动微分的正确性。

前半部分总结

FNN——“计算什么”

定义从输入到输出的复合函数：

$$f_{\theta}(\mathbf{x}) = f_L \circ \cdots \circ f_1(\mathbf{x})$$

- 每层：仿射变换 + 非线性
- 输出层匹配具体任务
- 参数量 = $\sum (d_{\text{in}} d_{\text{out}} + d_{\text{out}})$

Autodiff——“如何自动化”

- 自动记录计算，执行反向模式
- 精确、高效：一次反传得所有梯度
- PyTorch: `loss.backward()`

BP——“如何求梯度”

在计算图上反向应用链式法则：

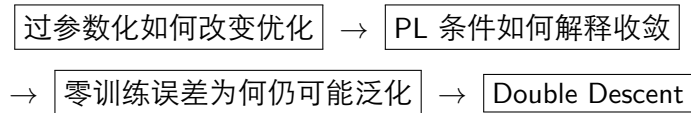
$$\nabla_{\theta} L = \text{反向遍历计算图}$$

从反向传播到更深层的问题

三个核心问题：

- ① 非凸损失为什么能够优化？
- ② 为什么模型经常达到接近零训练误差？
- ③ 零训练误差为什么不一定导致严重过拟合？

逻辑线索：



“收敛”到底是什么意思？

“收敛”有多种含义，需要区分：

- ① 损失函数收敛： $\mathcal{L}(\mathbf{w}_t) \rightarrow c$
- ② 梯度趋近于零： $\|\nabla \mathcal{L}(\mathbf{w}_t)\| \rightarrow 0$
- ③ 参数序列收敛： $\mathbf{w}_t \rightarrow \mathbf{w}^*$
- ④ 收敛到全局最优： $\mathcal{L}(\mathbf{w}_t) \rightarrow \mathcal{L}^*$
- ⑤ 测试误差收敛： $R_{\text{test}}(\mathbf{w}_t) \rightarrow ?$

关键警告

训练损失收敛 $\nRightarrow$ 泛化误差小

本节课目标：

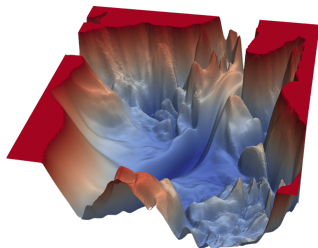
- 在什么条件下训练损失能收敛到零？
- 零训练误差何时仍有好的泛化？

深度网络的非凸损失景观

深度网络的损失函数 $\mathcal{L}(\mathbf{w})$ 通常是关于参数 $\mathbf{w}$ 的**非凸函数**。

非凸景观的典型特征：

- 局部极小值： $\nabla \mathcal{L} = 0$ ， $\nabla^2 \mathcal{L} \succeq 0$ ， 但 $\mathcal{L} > \mathcal{L}^*$
- 鞍点： $\nabla \mathcal{L} = 0$ ， 但 $\nabla^2 \mathcal{L}$ 有正有负特征值
- 平坦方向： $\nabla^2 \mathcal{L}$ 有接近零的特征值， 梯度下降几乎停滞



什么是过参数化?

设模型有 p 个可训练参数，训练集有 n 个样本。

区域	条件	特征
欠参数化	$p \ll n$	可能不存在零训练误差解
	$p \approx n$	恰好能拟合训练数据
过参数化	$p \gg n$	存在大量零训练误差解

深度网络通常处于过参数化区域

参数数量 $p \gg$ 训练样本数量 n

例：ResNet-50 ($p \approx 25\text{M}$, ImageNet $n \approx 1.2\text{M}$)；GPT-3 ($p = 175\text{B}$)。

过参数化改变了什么？

过参数化不仅增加表达能力，还改变优化问题的几何结构。

欠参数化 ($p < n$)

- 零训练误差解可能不存在
- 若存在，通常是少数孤立点
- 优化可能陷入局部极小

过参数化 ($p \gg n$)

- 零训练误差解大量存在
- 所有解可能形成一个高维流形
- 优化更容易到达全局最优

直觉：更大的网络有时反而更容易训练——解集更宽广，优化路径更通畅。

从孤立解到解流形

考虑最小化训练损失： $\min_{\mathbf{w}} \mathcal{L}(\mathbf{w})$ ，设全局最小值 $\mathcal{L}^* = 0$ 。

定义解集 $\mathcal{S} = \{\mathbf{w} : \mathcal{L}(\mathbf{w}) = 0\}$ 。

- 参数少 ($p < n$)： $\mathcal{S}$ 通常为空集或包含少数孤立点。
- 参数多 ($p \gg n$)： $\mathcal{S}$ 通常是一个 $(p - n)$ 维流形——大量参数组合具有相同的零训练损失。

非线性系统的解流形

对于非线性映射 $F: \mathbb{R}^p \rightarrow \mathbb{R}^n$ ($p > n$)，方程 $F(\mathbf{w}) = \mathbf{y}$ 的解集在正则条件下是一个 $(p - n)$ 维子流形。

注意：不同于线性系统，非线性解流形一般不是凸集。这正是深度网络损失“本质非凸”的来源之一。

过参数化线性回归

考虑线性系统 $\mathbf{X}\mathbf{w} = \mathbf{y}$, 其中 $\mathbf{X} \in \mathbb{R}^{n \times p}$.

- $p < n$: 方程数多于未知数, 通常无精确解, 只能最小化残差。
- $p = n$: 若 $\mathbf{X}$ 满秩, 通常有唯一解。
- $p > n$: 方程少于未知数, 通常有无穷多解。

核心问题

当 $p > n$ 时, 存在无穷多个满足 $\mathbf{X}\mathbf{w} = \mathbf{y}$ 的解。

梯度下降会从无穷多个解中选择哪一个?

这个问题的答案引出隐式正则化的概念。

优化算法的隐式选择

定理（线性回归中的隐式偏置）

对于线性系统 $\mathbf{X}\mathbf{w} = \mathbf{y}$ ($p > n$)，从 $\mathbf{w}_0 = \mathbf{0}$ 出发的梯度下降收敛到最小范数解：

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} \|\mathbf{w}\|_2 \quad \text{s.t. } \mathbf{X}\mathbf{w} = \mathbf{y}$$

闭式解： $\mathbf{w}^* = \mathbf{X}^\top (\mathbf{X}\mathbf{X}^\top)^{-1} \mathbf{y}$ （Moore-Penrose 伪逆）

隐式正则化 (Implicit Regularization)

即使目标函数没有显式正则项，优化算法本身也可能偏向某些解。

- GD 从零初始化出发 $\Rightarrow$ 偏向小范数解
- 不同优化器（GD vs. Adam）可能选择不同的解
- 这种隐式偏好对泛化至关重要

非凸函数还能有收敛保证吗？

传统优化理论的收敛分析通常要求：

- 凸性：每个局部极小都是全局极小
- 强凸性：可以推出线性收敛速率

但深度网络的损失函数通常不是凸的！

我们需要一个更弱的条件——

- 不要求凸性
- 仍然能保证收敛到全局最优
- 在过参数化网络中可能被满足

这就是 **Polyak-Łojasiewicz (PL) 条件** (Polyak, 1963)。

逻辑链条：

过参数化 $\implies$ PL 条件（局部） $\implies$ GD 指数收敛

定义 (μ -PL)

函数 $\mathcal{L}(\mathbf{w})$ 在集合 B 上满足 μ -PL 条件, 若存在 $\mu > 0$ 使得 $\forall \mathbf{w} \in B$:

$$\frac{1}{2} \|\nabla \mathcal{L}(\mathbf{w})\|^2 \geq \mu (\mathcal{L}(\mathbf{w}) - \mathcal{L}^*)$$

其中 $\mathcal{L}^* = \min_{\mathbf{w}} \mathcal{L}(\mathbf{w})$ 。

直觉: 当损失离最优值还很远时, 梯度不能过小。

PL 条件排除了以下 “坏” 情况:

$$\mathcal{L}(\mathbf{w}) - \mathcal{L}^* \gg 0 \text{ 但 } \|\nabla \mathcal{L}(\mathbf{w})\| \approx 0 \text{ (梯度小但损失高)}$$

即: 不存在 “远离最优但梯度消失” 的区域。

PL 与凸性的关系

关键事实

- 强凸 $\implies$ PL 条件成立
- PL 条件 $\not\Rightarrow$ 凸性
- 满足 PL 的函数可以是非凸的

为什么 PL 比凸性更适合深度学习？

- ① 深度网络损失一般不满足凸性，但可能在局部区域满足 PL
- ② PL 已经足以推出指数级收敛
- ③ 过参数化网络的 tangent kernel 的最小特征值 > 0 可以推出 PL

与 tangent kernel 的联系 (Liu, Zhu & Belkin, 2020)

定义 tangent kernel $K(\mathbf{w}) = \nabla F(\mathbf{w}) \nabla F(\mathbf{w})^\top \in \mathbb{R}^{n \times n}$ 。

若 $\lambda_{\min}(K(\mathbf{w})) \geq \mu > 0$ ($\forall \mathbf{w} \in B$)，则平方损失满足 μ -PL。

Smoothness 与下降引理

L -光滑 (Smoothness)

$\nabla \mathcal{L}$ 是 L -Lipschitz 连续的:

$$\|\nabla \mathcal{L}(\mathbf{w}) - \nabla \mathcal{L}(\mathbf{w}')\| \leq L \|\mathbf{w} - \mathbf{w}'\|, \quad \forall \mathbf{w}, \mathbf{w}'$$

下降引理 (Descent Lemma)

若 $\mathcal{L}$ 是 L -光滑的, GD 取步长 $\eta \leq \frac{1}{L}$, 则每步保证下降:

$$\mathcal{L}(\mathbf{w}_{t+1}) \leq \mathcal{L}(\mathbf{w}_t) - \frac{\eta}{2} \|\nabla \mathcal{L}(\mathbf{w}_t)\|^2$$

证明思路: Taylor 展开 + Lipschitz 条件给出二次上界, GD 步 $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla \mathcal{L}(\mathbf{w}_t)$ 恰好最小化此上界。

注意: 学习率太大 ($\eta > 2/L$) 会导致损失上升甚至发散。

PL + Smoothness $\Rightarrow$ 线性收敛

将下降引理与 PL 条件结合：

$$\mathcal{L}(\mathbf{w}_{t+1}) - \mathcal{L}^* \leq \mathcal{L}(\mathbf{w}_t) - \mathcal{L}^* - \frac{\eta}{2} \|\nabla \mathcal{L}(\mathbf{w}_t)\|^2 \quad (\text{下降引理})$$

$$\leq \mathcal{L}(\mathbf{w}_t) - \mathcal{L}^* - \eta\mu(\mathcal{L}(\mathbf{w}_t) - \mathcal{L}^*) \quad (\text{PL 条件})$$

$$= (1 - \eta\mu)(\mathcal{L}(\mathbf{w}_t) - \mathcal{L}^*)$$

递推得到**指数（线性）收敛**：

$$\mathcal{L}(\mathbf{w}_t) - \mathcal{L}^* \leq (1 - \eta\mu)^t (\mathcal{L}(\mathbf{w}_0) - \mathcal{L}^*)$$

取 $\eta = 1/L$ ，收敛速率由 $\kappa = L/\mu$ （条件数）决定。

重要提醒

不能笼统地说所有深度网络都满足全局 PL 条件。相关结果通常需要：足够宽的网络、特定初始化、数据假设与训练区域限制。

经验风险与总体风险

经验风险 (Empirical Risk)——训练集上可直接计算

$$\hat{R}(f) = \frac{1}{n} \sum_{i=1}^n \ell(f(x_i), y_i)$$

总体风险 (Population Risk)——真实分布上的期望，无法直接计算

$$R(f) = \mathbb{E}_{(x,y) \sim \mathcal{D}} [\ell(f(x), y)]$$

泛化间隙：

$$\text{Generalization Gap} = R(f) - \hat{R}(f)$$

泛化理论的核心问题：什么条件下泛化间隙较小？

训练误差、验证误差与测试误差

数据集	用途	使用频率
训练集 (Training)	学习模型参数 w	反复使用
验证集 (Validation)	选择超参数、模型架构	可多次评估
测试集 (Test)	最终评估泛化性能	只使用一次

测试集泄漏

若多次根据测试结果调整模型 $\Rightarrow$ 信息从测试集流入模型选择 $\Rightarrow$ 测试误差不再是泛化性能的可靠估计。

经典 Bias-Variance Trade-off

对于平方损失和随机训练集 S :

$$\mathbb{E}_S[(y - \hat{f}_S(x))^2] = \underbrace{(\mathbb{E}_S[\hat{f}_S(x)] - f^*(x))^2}_{\text{Bias}^2} + \underbrace{\mathbb{E}_S[(\hat{f}_S(x) - \mathbb{E}_S[\hat{f}_S(x)])^2]}_{\text{Variance}} + \underbrace{\sigma^2}_{\text{Noise}}$$

经典观点:

- 模型过小 $\Rightarrow$ 高偏差, 欠拟合
- 模型过大 $\Rightarrow$ 高方差, 过拟合
- 中间存在**最佳复杂度**, 测试误差最低 (U 型曲线)

这个框架在传统统计学中非常成功, 但它能解释现代深度学习吗?

经典泛化理论

经典理论通过假设空间 $\mathcal{H}$ 的容量约束泛化间隙。

泛化上界（示意性）

以高概率：

$$R(f) \leq \hat{R}(f) + \sqrt{\frac{\text{Complexity}(\mathcal{H})}{n}}$$

常见容量度量：

- 参数数量 p
- **VC 维度**： $\mathcal{H}$ 能“打碎” (shatter) 的最大样本数
- **Rademacher 复杂度**： $\mathcal{R}_n(\mathcal{H}) = \mathbb{E}_{\sigma} \left[\sup_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^n \sigma_i h(x_i) \right]$, $\sigma_i \in \{+1, -1\}$ 均匀随机

核心逻辑： $\mathcal{H}$ 越复杂 $\Rightarrow$ 上界越大 $\Rightarrow$ 泛化可能越差。

经典理论遇到深度学习

现代深度网络：

- 参数量 p 可达数十亿甚至更多
- $p \gg n$ (参数远多于训练样本)
- 训练误差 $\hat{R}(f) \approx 0$
- 测试误差仍然很低

按经典上界：

$$R(f) \leq \underbrace{\hat{R}(f)}_{\approx 0} + \underbrace{\sqrt{\frac{p}{n}}}_{\text{非常大!}} \longrightarrow \text{上界极其松散，几乎没有信息量}$$

根本矛盾

简单使用参数数量衡量模型复杂度过于粗糙。

需要更精细的复杂度度量，或需要考虑优化算法的作用。

随机标签实验 (Zhang et al., 2017)

实验设计：用同一个大型网络分别拟合真实标签和随机标签。

真实标签

- 训练误差 $\rightarrow 0$ ✓
- 测试误差低 ✓

随机标签

- 训练误差 $\rightarrow 0$ ✓
- 测试误差 $\approx$ 随机猜测 $\times$

含义：

- ① 网络具有**极强的记忆能力**——可以拟合任何标签
- ② 仅靠网络结构/容量不足以解释泛化（两种情况结构相同）
- ③ 真实数据上的泛化不是“容量决定”的

关键问题：为什么在真实数据上，训练过程通常找到具有较好泛化能力的解？

显式正则化

常见显式正则化方法

- **Weight Decay**: $\mathcal{L}_{\text{reg}}(\mathbf{w}) = \mathcal{L}(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2$
- **Dropout**: 训练时随机将一部分激活置零
- **数据增强**: 随机翻转、裁剪、颜色扰动等
- **Early Stopping**: 在验证误差开始上升时停止训练
- **Label Smoothing**: $y_{\text{soft}} = (1 - \varepsilon) y + \varepsilon / K$

但 Zhang et al. 发现：即使去掉所有显式正则化，深度网络仍能泛化。
正则化有帮助但不是必需的。这意味着还有其他机制在起作用——隐式正则化。

隐式正则化与隐式偏置

即使没有显式正则项，以下因素也会影响最终解的选择：

- SGD 与 mini-batch 噪声
- 参数初始化方式
- 学习率大小与调度
- 网络架构 (CNN / Transformer / ...)
- 参数化方式
- 训练时间 (epoch 数)

典型例子 (Soudry et al., 2018)

线性可分数据上的逻辑回归：

- GD (指数尾损失)
⇒ 收敛到最大间隔方向
- 即 GD 隐式地做了 SVM

$$\frac{\mathbf{w}(t)}{\|\mathbf{w}(t)\|} \rightarrow \frac{\mathbf{w}_{\text{SVM}}}{\|\mathbf{w}_{\text{SVM}}\|}$$

最终模型 = 模型结构 + 数据 + 优化算法

传统观点

训练误差 = 0 $\implies$ 严重过拟合

现代观点：Benign Overfitting

某些过参数化模型即使完全拟合训练数据（包括噪声），仍然具有较小的测试误差。

关键洞察：不是所有零训练误差的解都一样好——哪个零训练误差解才是决定泛化的关键。

Benign overfitting 需要条件：

- 数据分布的结构
- 特征协方差谱
(少数大特征值 + 大量小特征值)
- 噪声水平
- 优化算法选择的插值解
- 解的范数或平滑性

经典 U 型风险曲线

经典统计学的预测（横轴为模型复杂度，纵轴为测试误差）：

- ① 模型过小：高偏差，欠拟合，训练和测试误差都高
- ② 模型适中：偏差和方差达到平衡，测试误差最低
- ③ 模型过大：高方差，过拟合，测试误差上升

$$\text{测试误差} \approx \underbrace{\text{Bias}^2}_{\searrow \text{随复杂度}} + \underbrace{\text{Variance}}_{\nearrow \text{随复杂度}} + \text{Noise} \longrightarrow \text{U 型曲线}$$

这个图景在低维参数统计模型中非常成功。

但当模型复杂度继续增加，超过插值阈值之后，会发生什么？

插值阈值 (Interpolation Threshold)

定义

插值阈值：模型刚好能够将训练误差降到零的位置。
对于线性回归，即 $p = n$ （参数数 = 样本数）。

在插值阈值附近 ($p \approx n$):

- 模型对数据扰动**非常敏感**——标签中的微小噪声被剧烈放大
- 解**不稳定**——微小的数据变化导致参数剧烈变化
- 测试误差可能出现**峰值**

直觉：“刚好够用”的参数必须“使尽全力”去拟合每一个训练点，对噪声的余量为零，因此泛化最差。

Double Descent Curve (Belkin et al., 2019)

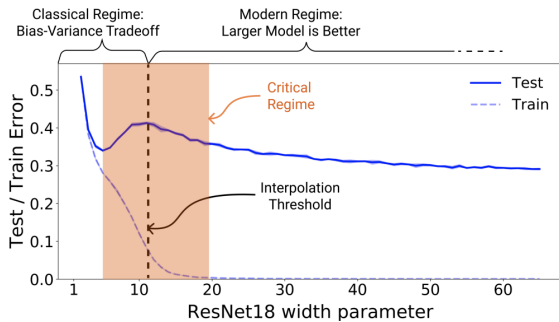
完整的风险曲线包含两个下降阶段：

阶段	区域	测试误差	训练误差
①	经典区域 ($p \ll n$)	先降后升 (U 型前半段)	逐渐下降
②	插值阈值 ($p \approx n$)	达到峰值	接近零
③	过参数化 ($p \gg n$)	再次下降	保持为零

关键观察：

- 经典 U 型曲线只是完整图景的前半段
- 过参数化区域中，测试误差可能降低到低于经典最优
- 训练误差在插值阈值后保持为零，但测试误差继续改善

Model-wise Double Descent



- **Classical Regime** ($p < n$): 模型增大 $\Rightarrow$ bias $\downarrow$, variance $\uparrow$, 经典 U 形曲线
- **Critical Regime** ($p \approx n$): 插值阈值附近, 测试误差达到**峰值**
- **Modern Regime** ($p \gg n$): 继续增大模型, 测试误差再次下降

训练误差在插值阈值后降为零, 但测试误差不升反降——这与经典统计学直觉相悖。

为什么插值阈值附近风险最高？(1/2)

以线性回归为例。数据 $(\mathbf{X}, \mathbf{y})$, $\mathbf{X} \in \mathbb{R}^{n \times p}$, 真实模型 $\mathbf{y} = \mathbf{X}\boldsymbol{\beta}^* + \boldsymbol{\varepsilon}$ 。

当 $p \leq n$ 时, 最小二乘解:

$$\hat{\mathbf{w}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} = \boldsymbol{\beta}^* + \underbrace{(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \boldsymbol{\varepsilon}}_{\text{噪声放大项}}$$

关键: $\hat{\mathbf{w}}$ 的估计方差取决于 $(\mathbf{X}^\top \mathbf{X})^{-1}$ 的特征值大小。

当 $p \rightarrow n^-$ 时, Gram 矩阵 $\mathbf{X}^\top \mathbf{X} \in \mathbb{R}^{p \times p}$ 逐渐退化:

- ① n 个样本最多支撑 n 个独立方向, p 逼近 n 时信息不足
- ② 最小特征值 $\lambda_{\min}(\mathbf{X}^\top \mathbf{X}) \rightarrow 0$
- ③ 逆矩阵的最大特征值 $= 1/\lambda_{\min} \rightarrow \infty$
- ④ 条件数 $\kappa(\mathbf{X}^\top \mathbf{X}) = \lambda_{\max}/\lambda_{\min} \rightarrow \infty$ (极度病态)

为什么插值阈值附近风险最高？(2/2)

参数估计的协方差为 $\text{Cov}(\hat{\mathbf{w}}) = \sigma^2(\mathbf{X}^\top \mathbf{X})^{-1}$ 。

当 $(\mathbf{X}^\top \mathbf{X})^{-1}$ 的特征值爆炸时，噪声 ε 中的微小扰动被放大成巨大的参数偏移，参数估计的方差急剧增大，测试误差飙升。

Belkin, Hsu & Xu (2019) 精确风险公式

设 $\mathbf{x}_i \sim \mathcal{N}(\mathbf{0}, I_p)$, $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$ 。当 $p \leq n - 2$ 时：

$$\text{Risk}(p) = \underbrace{\|\beta_{T^c}\|^2}_{\text{bias}^2} \cdot \left(1 + \frac{p}{n - p - 1}\right)$$

- $\|\beta_{T^c}\|^2$ ：未被模型捕获的真实信号能量（随 p 增大而减小）
- $\frac{p}{n - p - 1}$ ：方差放大倍数（ $p \rightarrow n$ 时趋于 ∞ ）
- 放大速度 $\gg$ bias 下降速度 $\implies$ 总风险在 $p \approx n$ 处发散

为什么继续增加参数后风险可能下降？

当 $p > n$ 时，存在无穷多插值解。最小范数插值解：

$$\hat{\mathbf{w}} = \mathbf{X}^\top (\mathbf{X}\mathbf{X}^\top)^{-1} \mathbf{y}$$

直觉：

- p 增大 $\Rightarrow$ 满足 $\mathbf{X}\mathbf{w} = \mathbf{y}$ 的解越来越多
- 优化算法可以选择范数更小的解： $\|\hat{\mathbf{w}}\| \searrow$ 当 $p \nearrow$
- 信号分散到更多参数方向，预测函数更平滑

Belkin, Hsu & Xu (2019): 当 $p \geq n + 2$, $\text{Risk} = \|\beta_T\|^2(1 - n/p) + \|\beta_{T^c}\|^2(1 + n/(p - n - 1))$ 。
当 $p \rightarrow \infty$, $\text{Risk} \rightarrow 0$ (若 $\beta_{T^c} = 0$)。

Rademacher 复杂度 + 有界范数 $\Rightarrow$ 好的泛化上界。

最小范数 $\|\hat{\mathbf{w}}\|$ 随 p 增大而减小，正好对应泛化上界收紧。

但第二次下降并非无条件发生！

不同形式的 Double Descent

Double descent 不仅在模型规模维度上出现：

- ① **Model-wise double descent**：固定数据，随模型规模变化
 - 增加网络宽度或深度 $\Rightarrow$ 先过拟合，再好转
- ② **Sample-wise double descent**：固定模型，随训练样本数量变化
 - 增加数据量在某个阶段可能暂时增大测试误差
- ③ **Epoch-wise double descent**：固定模型和数据，随训练时间变化
 - 测试误差先降、再升、再降
 - 与 **early stopping** 直接相关：停得太早或太晚可能都不是最差的

三种形式本质上都反映了有效模型复杂度穿越插值阈值时的现象。

Double Descent 的边界与误区

需要避免的错误结论

- ✗ “模型越大一定越好”
- ✗ “过拟合已经不存在”
- ✗ “正则化不再有用”
- ✗ “所有任务都有明显的 double descent”

是否出现及曲线形状取决于

- 标签噪声水平
- 数据分布
- 模型架构
- 优化器与学习率
- 显式正则化（如 weight decay）
- 横轴采用的复杂度定义

总结：Double Descent 揭示了一种现象，而非万能的实践指导。其价值在于：

- 解释为什么“更大的模型有时表现更好”
- 说明经典 bias-variance 框架需要扩展

第二讲知识地图

前半部分：

FNN → 前向传播 → 损失函数 → BP / Autodiff → 梯度下降

后半部分——两个核心问题：

问题一：能否优化成功？

- ① 过参数化改变优化几何
解集从孤立点 → 高维流形
- ② PL 条件作为非凸收敛保证
 $\frac{1}{2}\|\nabla\mathcal{L}\|^2 \geq \mu(\mathcal{L} - \mathcal{L}^*)$
- ③ PL + Smoothness $\Rightarrow$ 指数收敛
 $\mathcal{L}(\mathbf{w}_t) - \mathcal{L}^* \leq (1 - \eta\mu)^t(\mathcal{L}(\mathbf{w}_0) - \mathcal{L}^*)$

问题二：能否在新数据上表现好？

- ① 经典泛化理论的局限
参数量 $\neq$ 有效复杂度
- ② 隐式偏置选择“好”的解
优化算法 $\Rightarrow$ 最小范数
- ③ **Double Descent** 的完整图景
经典 U 曲线 $\rightarrow$ 插值峰 $\rightarrow$ 第二次下降

PL 条件回答“为什么可能收敛”； 泛化 + Double Descent 回答“收敛到零训练误差之后发生了什么”。

思考题

- ① 反向传播要求存储前向传播的所有中间激活值。如果网络有 1000 层，这会带来什么问题？你能想到什么解决思路？
- ② 网络结构本身能解释泛化吗？用同一个网络分别拟合真实标签和随机标签，训练误差都降到零，但泛化表现截然不同。那么，真正决定泛化好坏的是什么？
- ③ 为什么“参数刚好够用”比“参数远多于数据”更危险？在 Double Descent 中， $p \approx n$ 时测试误差最差， $p \gg n$ 时反而下降。直觉上如何理解这一点？