大模型原理与应用

第 1 课：AI 基础回顾 | 从机器学习到大模型

重新搭起后续课程的技术地基

这一课看什么

- 机器学习问题如何被形式化
- 感知机为什么是神经网络起点
- 训练、优化、泛化为何始终绕不开

本课主线

1. 从任务定义重新理解机器学习
2. 从感知机走到多层神经网络
3. 从损失和梯度走到优化与泛化

机器学习：从数据中学习规律

- 输入：X，也就是 feature vector
- 输出：y，也就是 label 或目标值
- 目标：找到一个函数 $y = f(X)$
- 用训练集学习规律，用测试集检查泛化能力

常见任务：

- 回归：输出实数
- 二分类：输出真 / 假、正 / 负
- 多分类：输出有限类别
- 生成：输出句子、图像、代码等复杂对象

先看两个最基本的例子

监督学习

- 狗的分类：根据高度、重量等特征判断犬种
- 文本分类：垃圾邮件识别、新闻分类

无监督学习

- 聚类：在没有标签时发现数据中的结构

核心：先把问题表述清楚，再谈模型。

机器学习模型的基本元素

- 训练集：模型用来学习
- 测试集：模型用来验证
- 分类器 / 回归器：真正做预测的函数
- 误差 / 损失：衡量预测好不好
- 过拟合：训练集表现很好，但对新数据失效

常见方法：

- 决策树
- KNN
- 支持向量机 SVM
- 感知机
- 神经网络

为什么后来会走向深度学习

来自 MIT 材料的三个关键词：

1. Big Data：数据量足够大
2. Hardware：GPU 等硬件足够快
3. Software：TensorFlow、PyTorch 等工具成熟

深度学习不是突然出现的新思想，而是在数据、算力、软件都成熟后真正爆发。

什么是深度学习

- 机器学习：用数据学习映射关系
- 深度学习：用多层神经网络自动学习表示
- 过去：很多特征需要人工设计
- 现在：模型可以直接从原始数据中学习更抽象的特征

可以把它理解成：

- 浅层模型更多依赖人工特征
- 深层模型更擅长自动表示学习

感知机：神经网络的起点

感知机来自一个非常简单的想法：

- 输入是一个实数向量 x
- 每个输入有一个权重 w
- 先做加权求和
- 再经过激活函数，输出 1 或 −1

直觉上，它像一个最简单的“会做二选一判断”的神经元。

感知机的计算过程

前向传播

1. 输入特征 x
2. 计算加权和 $z = w^T x + b$
3. 经过激活函数得到输出 y

激活函数为什么重要

如果没有激活函数：

- 多层线性变换仍然等价于一层线性变换
- 网络深了也学不出更复杂的边界

激活函数的作用：

- 引入非线性
- 让网络有能力表达复杂模式

激活函数

- 阶跃函数
- Sigmoid
- Tanh
- ReLU

感知机的几何意义

核心视角：

- $w^T x > 0$ 时判为一类
- $w^T x < 0$ 时判为另一类

所以感知机本质上是在空间里画一条线，或者更一般地说，画一个超平面。

它把样本空间分成两边。

错误分类时怎么办

如果一个样本本来是正类，却被判成负类：

- 说明分界面方向不对
- 需要把权重 w 往正确样本的方向转一点

感知机更新的直觉：

- 判对了：参数不动
- 判错了：沿着 y x 的方向修正

这就是最早期的“从错误中学习”。

从单个感知机到神经网络

- 单个感知机：只能做简单线性划分
- 多输出感知机：可以同时预测多个类别
- 单层神经网络：多个神经元并列
- 深层神经网络：把多层表示堆起来

$$Perceptron \rightarrow Single\ Layer\ Network \rightarrow Deep\ Neural\ Network$$

神经网络在做什么

可以把网络理解成一层层特征变换：

- 第一层：从原始输入中抽取初级模式
- 中间层：组合成更抽象的模式
- 输出层：完成分类、回归或生成

例子：

- 图像分类
- 文本分类

模型有了，怎么知道它学得对不对

损失函数回答的是：

> 模型现在到底错了多少?

材料中出现的典型损失：

- 均方误差 MSE
- 经验风险 / empirical loss
- 二元交叉熵 Binary Cross Entropy

没有损失函数，就无法比较不同参数谁更好。

训练的本质，就是不断把损失降下来

训练神经网络可以写成：

- 给定参数 w
- 计算预测
- 计算损失 $J(w)$
- 调整参数，让 $J(w)$ 变小

所以训练问题最终会变成一个优化问题。

梯度下降：最基本的优化方法

- 梯度给出损失上升最快的方向
- 我们沿着负梯度方向走
- 每次迈多大一步，由学习率决定

更新形式：

$$w \leftarrow w - \eta \cdot \nabla J(w)$$

真实训练为什么总是不那么顺

神经网络的损失面通常不是一个漂亮的碗。

常见困难：

- 局部最优 local optima
- 平台区 plateaus
- 鞍点 saddle points
- 不同方向尺度差异很大

所以“朝最陡方向走”并不总是最快、最稳的路线。

反向传播：高效算梯度

如果每个参数都单独试错，成本会非常高。

反向传播做的事是：

- 先完成一次前向传播
- 再从输出层往回传误差信号
- 用链式法则高效计算每一层参数的梯度

这让深层网络训练成为可能。

学习率为什么关键

学习率太小：

- 训练很慢
- 容易陷在平台区

学习率太大：

- 震荡
- 发散
- 错过更优点

课件中明确强调：

- 小学习率不一定更好
- 常常要在训练过程中逐步衰减学习率

只靠最基本的梯度下降还不够

优化课件中继续介绍了更实用的方法：

- Momentum：累计历史方向，减少来回摆动
- AdaGrad：根据历史梯度调整各维步长
- RMSProp：稳定非凸优化中的学习率
- Adam：结合 Momentum 和 RMSProp，常作为默认选择

结论不是“谁永远最好”，而是不同问题、不同阶段有不同合适的优化器。

为什么要用随机梯度下降 SGD

全量梯度下降每次都要扫完整个数据集，代价很大。

SGD / mini-batch 的思路：

- 每次只抽一小批样本
- 用这批样本估计梯度
- 速度更快，能处理大规模数据

现实训练里常用的是：

- shuffled data
- mini-batch
- SGD + momentum，或者 Adam

Mini-batch 的作用

MIT 材料后半段把它讲得很清楚：

- batch 太小：梯度噪声大，但更新快
- batch 太大：梯度更稳，但计算更重
- mini-batch 是速度与稳定性的折中

这也是今天几乎所有深度学习训练的基本套路。

训练好模型，不等于泛化好

一个网络可能：

- 在训练集上越来越准
- 在新数据上越来越差

这就是过拟合。

典型现象：

- 模型太复杂
- 数据太少
- 训练太久

两类经典正则化

Dropout

- 训练时随机丢弃一部分神经元
- 防止网络过度依赖某些局部路径

Early Stopping

- 在验证集表现开始变差之前就停止训练
- 避免继续朝“记住训练集细节”的方向走

这节课你该带走什么

1. 机器学习：从数据里学规律
2. 感知机：最简单的神经元
3. 多层网络：靠堆叠获得表达力
4. 损失函数：定义错多少
5. 反向传播：定义怎么学
6. 优化器：决定学得快不快、稳不稳
7. 正则化：决定会不会过拟合

为什么这节课是后面的地基

后面学习 Transformer 和大模型时，今天这些概念都会反复出现：

- 表示学习
- 前向传播
- 损失函数
- 梯度与反向传播
- mini-batch 训练
- 优化器
- 泛化与正则化

如果这一层地基不稳，后面的“大模型原理”就容易只剩名词。

第 1 课收束

先把基础搭稳，再进入大模型

- 机器学习解决的是“从数据学规律”
- 深度学习解决的是“如何自动学表示”
- 优化与泛化决定模型能不能真正用起来