大模型原理与应用

第 7 课：微调 | 让基础模型真正可用

理解 SFT、PEFT、LoRA 与任务适配

这一章看什么

- 为什么预训练之后还要继续微调
- SFT 怎样把 base model 变成可用 assistant
- PEFT、LoRA 为什么成为工程主流

本章主线

1. 先回答：为什么只有预训练还不够
2. 再回答：SFT 怎样把模型接到任务上
3. 最后回答：怎样以更低成本完成适配

预训练之后为什么还要微调

这是这一章最关键的出发点。

预训练模型会：

- 补全
- 模仿
- 延续文本分布

但用户真正需要的是：

- 听懂指令
- 回答问题
- 遵守规范
- 拒绝危险请求

所以：

> 语言建模能力 ≠ 指令遵循能力

微调真正改变的是什么

- 我们已经有一个预训练模型
- 现在希望它适应具体任务

微调就是：

- 在下游标注数据上继续优化
- 让模型更贴近目标任务分布

最经典的做法有：

- full fine-tuning
- head tuning
- parameter-efficient fine-tuning

为什么微调在 NLP 中特别有效

因为预训练已经提供了：

- 语言知识
- 参数初始化
- 上下文化表示能力

微调只需要：

- 把这些能力“接到”具体目标上
- 学会任务接口
- 学会输出风格和偏好

因此我们常常只需要远少于预训练的数据。

指令微调（Instruction Tuning）

基本定义：

- 用 `(instruction, response)` 对微调模型
- 让模型学会“看见要求，就按要求回答”

这一步是从 base model 到 chat / assistant model 的第一步。

为什么只有预训练还不够

- 预训练模型能补空、分类、续写
- 但不等于能按用户意图回答
- 更不等于符合价值、安全和有用性要求

典型错位包括：

- 用户要指令执行，模型却继续补全
- 用户要安全回答，模型却输出有害内容
- 用户要实用答案，模型却只学到了语言表面模式

SFT：监督式指令微调

SFT 的核心非常朴素：

- 收集高质量的指令 - 回复样本
- 按 token likelihood 做监督训练

好处：

- 简单稳定
- 效果直接
- 能显著提升 zero-shot / few-shot instruction following

这也是今天几乎所有对齐流程的第一步。

指令数据从哪里来

材料里提到了三大来源：

1. 现有 NLP 数据集改写成 instruction format
2. 人工编写 / 标注
3. 用更强模型生成，再人工筛选或社区共享

也就是：

- synthetic conversion
- human annotation
- AI-generated / community collected data

高质量 SFT 数据长什么样

- FLAN：多任务模板化
- Alpaca：指令 - 输出成对数据
- OpenAssistant：更开放、更复杂的真实问答

它们的差异包括：

- 风格
- 长度
- 是否带引用
- 事实复杂度
- 安全要求

这些差异都会影响模型的行为。

SFT 有哪些局限

- 高质量标注贵
- 很难覆盖开放式任务
- 容易变成逐 token 模仿
- 创造性上限受监督数据限制
- 可能鼓励表面正确但实际幻觉的回答

所以 SFT 很重要，但通常不是终点。

Alignment 在这一章里的位置

到这里为止，这一章已经回答了两个核心问题：

- 为什么预训练之后还要再做一轮适配
- 为什么 SFT 是几乎所有聊天模型的起点

但还有一个下一步问题：

- 当模型已经“会回答”以后，怎样让它“更符合人类偏好、规则和使用预期”？

这个问题会走向后续的 alignment / RL post-training。

这份主讲先不展开 RLHF 细节

原因不是它不重要，而是它值得单独讲一章：

- 开放式任务往往没有唯一标准答案
- 人类更擅长比较“哪个回答更好”
- 一旦引入偏好数据，就会牵涉 reward、stability、failure case

所以这里先保留一个结论：

- SFT 让模型先变得“可用”
- alignment 再让模型变得“更符合人类预期”

微调的另一个现实问题：太贵

- 全参数微调需要更新全部参数
- 每个任务都保存一套完整模型
- 对 70B / 170B 级模型，存储和训练成本极高

所以工程上不能只会“全量微调”。

PEFT：参数高效微调

PEFT 的关键思想是：

- 不去改全部参数
- 只改一小部分参数
- 或加一小组可训练参数

目标：

- 降低显存和存储成本
- 支持一底座多任务
- 尽量减少灾难性遗忘

PEFT 的几类思路

材料里总结了三类：

1. Selective
    - 只训练部分层
2. Additive / Adaptive
    - 新增模块，只训练新增参数
3. Reparameterization
    - 用低秩等方式重参数化更新

这三类几乎覆盖了主流 PEFT 方法谱系。

Prompt Tuning / Prefix Tuning

这类方法的想法是：

- 冻结原模型
- 只学习一组软提示向量

优点：

- 参数极少
- 任务切换方便

局限：

- 性能依赖模型规模、任务类型和设计细节

LoRA 为什么这么流行

LoRA 的思想：

- 假设权重更新的“内在秩”较低
- 把更新写成两个低秩矩阵 BA
- 冻结原始大权重，只训练低秩增量

优点：

- 参数少
- 训练便宜
- 可按任务切换 adapter / LoRA 权重
- 合并后推理延迟几乎不增加

LoRA 在 Transformer 里怎么用

材料提到一个经验事实：

- 常把 LoRA 加在 attention 的 W_q、W_v 上
- rank r 很小也常常有效

这说明：

- 大模型适配不一定需要大幅改动整个网络
- 很多任务知识可以通过很小的“增量参数”注入

微调这一章的完整流水线

把这一章材料串起来，就是一条很清晰的 pipeline：

1. 从已有 base model 出发
2. 准备任务数据或指令数据
3. 先做 SFT，让模型学会任务接口
4. 实际适配时常配合 PEFT 降低成本
5. 如有需要，再进入后续的 alignment / RL post-training

在 NLP 任务中的落地

这套适配方式支撑了大量任务：

- 文本分类
- 情感分析
- 自然语言推理
- 摘要
- 问答
- 机器翻译
- 信息抽取
- 对话系统
- 代码生成

在 NLP 任务中的落地

差别往往不在“是不是 Transformer”，而在：

- 微调数据
- 指令格式
- 对齐方式
- 适配成本

这节课你该带走什么

1. 预训练给模型通用能力，但不能直接替代任务适配
2. SFT 让模型学会按要求回答，是聊天模型的第一道接口层
3. PEFT 让适配更现实、更便宜，LoRA 是其中最重要的方法之一
4. 当模型已经“可用”以后，才会继续进入更深入的 alignment / RL 后训练

为什么这一章定义了大模型适配方式

后面无论讲对齐、多模态、行业应用还是工程部署，都会回到这张图：

- 模型怎么接任务？靠微调
- 行为怎么先变可用？靠 SFT
- 成本怎么降？靠 PEFT / LoRA

所以这一章真正建立的是：

> 现代大模型不只要会训练，更要会适配

第 11 章收束

SFT 给接口，PEFT 给成本解法，LoRA 给工程抓手