模型蒸馏

马浩琨

Sep 20, 2026

近年来，深度学习模型的规模正以惊人的速度增长。从早期的几百万参数，到如今动辄数百亿甚至上万亿参数的巨型模型，性能的提升往往伴随着算力和存储成本的急剧上升。大型模型在云端服务器上运行尚可接受，但一旦要部署到边缘设备、移动终端或者嵌入式系统时，内存占用、推理延迟和功耗便成为难以逾越的障碍。模型蒸馏正是为了弥合这一鸿沟而诞生的技术：通过让一个已经训练好的大型“教师”模型，把自己的知识迁移给一个轻量级的“学生”模型，使后者能够在保持较高精度的同时，显著降低资源消耗。

1　什么是模型蒸馏

模型蒸馏的核心思想是利用教师模型输出的“软标签”来指导学生模型的训练。与传统的硬标签（one-hot 向量）不同，软标签包含了教师模型对不同类别之间的置信度分布，能够传递更加丰富的类别间关系信息。最早可追溯到 2006 年 Buciluǎ 等人提出的模型压缩思想，而真正让该方法广为人知的是 Hinton 及其同事于 2015 年发表的论文《Distilling the Knowledge in a Neural Network》。在这篇工作中，研究者系统性地定义了“温度”参数来软化概率分布，并给出了完整的损失函数设计方案。与剪枝、量化、参数共享等其他压缩手段相比，蒸馏更侧重于知识迁移而非直接减少参数量，因此可以在不改变学生模型结构的前提下实现性能逼近。

2　为什么需要蒸馏

在实际生产环境中，推理延迟直接影响用户体验。例如，一款移动应用若每次调用模型都要等待数百毫秒，用户很快就会流失。同时，内存占用过高会导致低端设备频繁触发交换分区，进一步拖慢系统响应。边缘设备往往还受限于功耗和散热，难以支撑大型矩阵运算。此外，数据隐私法规要求用户数据不出域，这使得在本地部署轻量模型成为刚需。最后，从绿色计算的角度看，减少一次前向传播所需的浮点运算，等同于降低相应的碳排放，符合可持续发展目标。

3　蒸馏的数学与算法原理

温度参数是蒸馏中最关键的超参数之一。标准 Softmax 函数在温度为 1 时退化为普通分类器；当温度 T > 1 时，概率分布会被“软化”，原本接近 0 的概率值会被放大，从而暴露教师模型对错误类别的潜在判断。数学上，软化后的概率可写为：

$$\sigma_i(z, T) = \frac{\exp(z_i / T)}{\sum_j \exp(z_j / T)}$$

其中 z_i 表示类别 i 的 logits。学生模型在训练时同样使用该温度计算概率，随后在推理阶段将温度恢复为 1 以

获得尖锐的预测。

损失函数通常由两部分加权组成：一部分是学生模型与真实标签之间的交叉熵，另一部分是学生与教师软标签之间的 Kullback-Leibler 散度。形式化地：

$$L_{KD} = \alpha \cdot \mathrm{CE}(y, \sigma(z_s, 1)) + (1-\alpha) \cdot \mathrm{KL}(\sigma(z_t, T) \| \sigma(z_s, T))$$

式中 α 控制两个损失的平衡，z_s、z_t 分别代表学生和教师的 logits。仅使用输出层蒸馏有时不足以捕捉教师的中间表征，因此研究者提出了特征蒸馏方法。例如，FitNet 通过最小化学生与教师中间层特征图的均方误差来对齐表征空间；Attention Transfer 则利用注意力图之间的差异作为额外监督信号。关系蒸馏进一步考虑样本之间的成对距离或图结构，使学生不仅模仿单个样本的输出，还模仿样本间的相对关系。

在线蒸馏与离线蒸馏的区别在于教师是否与学生同时更新。在线蒸馏中，多个模型互为师生，梯度在它们之间流动，适合端到端训练；自蒸馏则让模型蒸馏自己，常用于正则化或半监督场景。

4 蒸馏实践全流程

在准备阶段，首先需要一个性能优异的教师模型，可通过公开预训练权重获得，也可针对特定领域继续微调。数据方面，除了使用原始训练集，还可以利用教师模型生成合成数据，以缓解数据匮乏或隐私限制。蒸馏流水线通常包括以下步骤：先以较高温度预热学生，使其快速学习教师的软分布；随后逐步降低温度并引入真实标签，防止学生过拟合软标签；最后在验证集上搜索最优 α 与 T 的组合。

评估指标需兼顾精度与效率。除了常见的 Top-1/Top-5 准确率，还应报告单样本推理延迟、FLOPs、模型体积以及在目标硬件上的能耗。PyTorch 官方提供了一个简洁示例，用于展示如何实现蒸馏损失：

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

def distillation_loss(student_logits, teacher_logits, labels, T=4.0, alpha=0.7):
    # 将教师与学生的 logits 除以温度后做 softmax
    soft_teacher = F.softmax(teacher_logits / T, dim=1)
    soft_student = F.log_softmax(student_logits / T, dim=1)
    # KL 散度部分
    kd_loss = F.kl_div(soft_student, soft_teacher, reduction='batchmean') * (T ** 2)
    # 硬标签部分
    ce_loss = F.cross_entropy(student_logits, labels)
    # 加权求和
    return alpha * kd_loss + (1 - alpha) * ce_loss
```

这段代码首先用温度 T 对教师和学生的 logits 进行缩放，随后分别计算软化的概率分布；KL 散度乘以 T^2 是为了补偿因温度缩放而减小的梯度量级；最终将蒸馏损失与标准交叉熵加权求和，得到总目标函数。

在工程落地时，Hugging Face 的 Text-Generation-Inference 可直接加载蒸馏后的生成模型；ONNX Runtime 与 TensorRT 则提供跨平台的高性能推理后端，可将学生模型进一步量化或编译为硬件专用格式。

5 典型场景与案例

在自然语言处理领域，DistilBERT 通过在预训练阶段引入蒸馏损失，将 BERT 的层数减半、参数量压缩 40%，而下游任务性能仅下降约 3%。TinyBERT 则在教师的每一层都进行特征对齐，实现了更激进的压缩。类似地，MobileBERT 在保持 BERT 级别精度的前提下，将推理速度提升 4 倍。代码生成场景中，CodeT5 的小型版本通过蒸馏可在单块消费级 GPU 上实时补全代码。

计算机视觉方面，ResNet-50 可蒸馏到 MobileNetV3，使 ImageNet 准确率达到 75% 以上，同时延迟降低一个数量级。Vision Transformer 的蒸馏则常采用注意力图对齐策略，使 DeiT-S 在同等 FLOPs 下超越 CNN 基线。

多模态领域，CLIP 的轻量化版本通过蒸馏保留了图文对齐能力，可部署在手机端实现实时图文检索。工业界落地案例包括搜索引擎的排序模型、推荐系统的召回层以及语音识别的声学模型，这些场景对延迟和内存均有严格约束，蒸馏成为标准优化手段。

6 进阶话题

跨模态蒸馏试图让文本、图像、音频模型之间互相迁移知识。例如，将视觉问答模型的表征蒸馏到纯文本模型，使后者无需图像输入也能进行一定程度的视觉推理。强化学习中，策略蒸馏可将复杂教师策略压缩为轻量学生策略，降低机器人控制的计算开销。持续学习场景下，蒸馏被用于防止灾难性遗忘：新任务训练时，旧任务的软标签作为额外监督保留历史知识。安全方面， Membership Inference 攻击可通过教师输出推断训练集成员身份，而蒸馏后的学生模型因信息损失而降低此类风险。可解释性研究则发现，蒸馏后的注意力图往往更加集中，有助于定位模型决策的关键区域。

7 常见陷阱与调优技巧

温度与 α 的选择对最终性能影响显著。实验表明，T=4 通常在视觉任务上表现稳健，而 NLP 任务可能需要 T=2 才能避免软标签过于平滑。教师模型过强时，学生可能因“跟不上”而收敛到次优解，此时可引入中间层对齐或逐步解冻策略缓解。合成数据若与真实数据分布差异较大，会导致学生在真实场景下性能骤降；缓解方法包括混合真实与合成样本，或在蒸馏后期引入对抗训练。损失权重可采用动态调整策略：前期以蒸馏损失为主，后期逐步提高硬标签权重，以保证最终决策的尖锐性。失败案例往往源于忽略数据增强、温度退火或评估硬件不一致等问题。

8 未来方向

零样本蒸馏试图在完全没有真实标签的情况下，仅利用教师输出完成知识迁移，这对隐私敏感场景意义重大。架构搜索与蒸馏联合优化可让搜索算法直接针对蒸馏后的性能进行剪枝与通道重构，从而得到更高效的学生结构。MoE（Mixture-of-Experts）模型虽然参数量巨大，但活跃参数较少，通过蒸馏可进一步降低推理成本。开源社区已出现统一 benchmark，如 KD360，旨在标准化不同方法在相同硬件和数据集上的对比。

9 结论与行动清单

若想快速上手，可按以下步骤操作：选择一个公开预训练教师模型；准备或合成蒸馏数据集；实现前述 distillation_loss 函数；以较高温度预热学生，随后逐步退火并引入真实标签；在目标硬件上测量延迟与精度，迭代调优超参数。推荐阅读包括 Hinton 2015 年的原始论文、Romero 等人提出的 FitNet，以及 Jiao 等人针对 BERT 的 TinyBERT 工作。模型蒸馏让智能不再局限于云端服务器，而是真正走向普惠计算的每一个角落。