第 12 章 精排模型

上一章我们介绍了工业级推荐系统中的粗排模型及其常见网络架构。本章将进一步讨论推荐系统中另一个重要的模型环节——精排模型。顾名思义，精排模型负责对粗排筛选出的候选物品进行更加精细的偏好预估，为后续的多目标融合与排序决策提供关键的预测信号。在本书第6章中，我们已经介绍了精排模块的整体架构、功能定位与核心处理流程。本章则将聚焦精排模型本身，系统介绍其发展历程、核心技术问题、典型网络结构以及工业实践中的优化方法。

对于刚接触推荐系统的读者来说，可能已经听说过 LR、FM、DeepFM、DIN、DIEN 等经典模型，也了解过近年来不断涌现的多任务学习模型与推荐大模型。但如果进一步追问：

笔记 这些模型为什么会在不同阶段出现？它们分别解决了什么问题？为什么一些模型能够在工业场景中长期发挥作用，而另一些模型虽然结构复杂，却未必能够带来更好的线上效果？

要回答这些问题，就不能仅仅停留在模型结构和论文名称的层面，而需要从技术发展的历史脉络与工业推荐系统面临的实际问题两个维度来理解精排模型。

12.1 精排模型的发展历程



图 12.1: 精排模型的发展历程。

如图12.1所示，从技术发展的角度看，精排模型的演进大致可以划分为四个阶段：机器学习时代（2010~2016 年）、深度学习时代（2016~2020 年）、多任务与多目标优化时代（2020~2023 年），以及推荐大模型时代（2023 年至今）。这些时间划分主要用于概括各阶段的技术重点，并不意味着不同技术在某个时间点发生了截然分明的更替。事实上，线性模型、深度网络、序列建模与多任务学习至今仍然广泛共存于工业推荐系统中。

12.1.1 机器学习时代

工业级推荐系统中的精排模型发展的第一阶段是机器学习时代，这一阶段的核心问题是如何通过特征工程刻画用户与物品之间的关系。在深度学习广泛应用之前，精排模型主要依赖逻辑回归（Logistic Regression，LR）、梯度提升树（Gradient Boosting Decision Tree，GBDT）、因子分解机（Factorization Machines，FM）及其扩展模型，例如场感知因子分解机（Field-aware Factorization Machines，FFM）。在这一时期，精排模型的表达能力在很大程度上取决于人工设计的特征，以及如何显式地建模不同特征之间的交互关系。例如，用户历史上是否点击过某类物品、用户所在的地域、物品的类别，以及用户与物品类别之间的交叉特征，都可能成为影响排序结果的重要因素。工程师需要结合业务经验设计特征，并通过特征组合、离散化和参数学习等方式提升模型的预估能力。

这一阶段的主要矛盾在于：如何在有限的计算资源和模型表达能力下，从海量稀疏特征中有效学习用户偏好。随着业务规模扩大，人工特征工程的成本逐渐上升，预先设计的特征交互也难以覆盖复杂的用户行为模式，这推动了深度学习在推荐排序中的应用。

12.1.2 深度学习时代

工业级推荐系统中的精排模型发展的第二阶段是深度学习时代，这一阶段的核心问题是如何自动学习复杂的特征交互与用户兴趣。随着计算资源、训练数据规模和深度学习技术的发展，推荐系统开始广泛采用深度神经网络（Deep Neural Network，DNN）建模用户与物品之间的复杂关系。Wide & Deep、DeepFM 等模型尝试将

显式特征交互与深层网络结合起来，降低模型对人工特征组合的依赖；DIN 则通过注意力机制，根据当前候选物品动态提取用户历史行为中的相关兴趣；DIEN 进一步对用户兴趣的演化过程进行建模。

相比起机器学习时代，这一阶段的重要变化是精排模型设计的重点从人工构造特征逐渐转向网络结构与表示学习。精排模型不再只是对预先定义的特征进行加权组合，而是能够从大规模数据中学习更加复杂的非线性关系。同时，随着 Transformer 等序列建模技术的发展，推荐系统开始探索更灵活的用户行为序列表示方法，为长序列兴趣建模提供了新的技术路线。

不过，网络表达能力的提升并不意味着推荐效果能够无限增长。随着特征规模、行为序列长度和网络复杂度不断增加，训练成本、推理时延以及特征交互效率逐渐成为新的瓶颈。因此，精排模型开始从单纯追求表达能力，转向更加重视模型结构、计算效率与线上收益之间的平衡。

12.1.3 多任务与多目标优化时代

工业级推荐系统中的精排模型发展的第三阶段是多任务与多目标优化时代，这一阶段的核心问题是如何在多个预测目标之间实现有效的知识共享与业务权衡。在进入 2020 年前后，工业推荐系统需要同时预测点击、观看时长、点赞、关注、转化等多种用户反馈。单一预测目标难以完整刻画用户体验与业务价值，而不同任务之间又存在相关性、数据分布差异和优化方向冲突等问题。在这一背景下，多任务学习（Multi-Task Learning，MTL）成为精排模型的重要研究方向。MMoE、PLE、CGC 等模型通过专家网络、门控机制和任务特定结构，在不同任务之间共享部分表示，同时保留各任务的差异性。与此同时，多目标优化（Multi-Objective Optimization，MOO）领域会进一步关注多个目标之间的权衡，避免精排模型过度追求某一项指标而损害其他业务目标。

在这一时期，用户兴趣建模也在持续发展。随着用户行为序列越来越长，如何从海量历史行为中提取与当前候选物品相关的信息，成为工业精排的重要问题。SIM、TWIN 等长序列兴趣建模方法通过检索、筛选等机制降低长序列建模的计算成本，使得精排模型能够利用更长时间范围内的用户行为信息。

因此，这一阶段的核心变化并不是简单地将单任务模型替换成多任务模型，而是精排模型需要同时处理多个预测目标、不同任务之间的知识共享，以及不断增长的特征与行为序列规模。模型效果的提升越来越依赖整体训练目标、网络结构与线上业务需求之间的协调。

12.1.4 推荐大模型时代

工业级推荐系统中的精排模型发展的第四阶段是推荐大模型时代，这一阶段的核心问题是如何利用更大规模的模型与更丰富的行为表示提升推荐能力。从 2023 年以来，ChatGPT 的横空出世以及大语言模型（Large Language Model，LLM）的快速发展推动了工业级推荐系统进一步探索大规模预训练、用户行为序列建模与生成式推荐等技术路线。与传统精排模型相比，推荐大模型更加关注大规模参数、长上下文和统一表示所带来的潜在收益，希望从更丰富的用户行为与物品信息中学习复杂的兴趣模式，并提升模型对稀疏反馈、长尾物品和动态兴趣的处理能力。

在工业实践中，快手的 MARM、字节跳动的 RankMixer 等工作展示了大规模模型在推荐排序场景中的探索；OneRec 等生成式推荐工作则进一步探索了不同于传统逐物品打分的推荐建模范式。这些研究共同推动推荐系统从传统的特征驱动网络向更大规模、更强表示能力的模型发展。

不过，需要注意的是，推荐大模型并不意味着传统精排模型的技术问题已经消失。更大的模型通常也意味着更高的训练成本、推理开销和服务复杂度。如何利用大规模预训练带来的表示能力，如何将用户行为、物品信息与业务特征有效结合，以及如何在工业系统的时延和资源约束下实现稳定的线上收益，仍然是需要重点解决的问题。

总体而言，精排模型的发展并不是模型规模不断增大的单一过程，而是围绕推荐系统中的主要技术矛盾持续演进：从依靠人工特征工程，到自动学习复杂特征交互；从优化单一预测目标，到协调多个任务与业务目标；再到探索利用大规模预训练与统一表示进一步提升推荐能力。不同阶段的技术重点有所变化，但始终围绕着如何更准确地预估用户反馈，并在实际业务约束下提升推荐效果这一核心目标展开。

12.2 精排模型需要解决的核心技术问题

在了解精排模型的历史发展之后，我们还需要进一步回答如下所示的一个问题：

笔记 如果不局限于某一种具体网络结构，在工业级推荐系统中设计和优化精排模型，究竟需要重点考虑哪些核心的技术问题？

精排模型的直接任务是预估用户对候选物品的某种反馈，例如点击概率、有效观看概率或预期观看时长。但模型的预估精度并不是由某一个因素单独决定的。输入模型的特征是否充分、网络结构能否有效利用这些特征，以及训练目标能否引导模型学习到符合业务需求的参数，都会影响最终效果。同时，精排模型还需要满足工业系统对训练成本、推理时延和资源消耗的要求。从技术实现的角度来看，我们可以将精排模型需要解决的问题归纳为三个方面：**特征选择与表示、模型结构设计，以及训练与优化。**

12.2.1 特征选择与表示

精排模型需要解决的第一个核心技术问题是特征选择与表示，即精排模型应该使用哪些信息，以及如何组织这些信息？特征是精排模型进行用户反馈预估的基础。在工业推荐系统中，可供使用的特征非常丰富，既包括用户属性、物品属性、上下文信息等基础特征，也包括用户历史点击、观看、点赞等行为序列，以及通过统计、交叉和预训练得到的衍生特征等。

但可用特征多，并不意味着将所有特征输入模型就一定能够获得更好的效果。一方面，不同特征的信息价值存在差异，部分特征可能包含大量噪声，或者与目标任务的相关性较弱；另一方面，特征之间可能存在冗余、分布差异和统计偏差，特征规模的增加还会带来更高的存储、训练与推理成本。因此，如何选择有效特征、构建合适的特征表示，并根据任务需求组织不同类型的特征，是精排模型设计中的基础问题。

其中，序列特征尤其值得关注。用户的历史行为包含了丰富的兴趣信息，但行为序列通常较长，不同历史行为对当前候选物品的影响也各不相同。模型不仅需要决定使用哪些历史行为，还需要判断这些行为与当前推荐场景之间的关联程度。如何处理这类特征，将直接影响模型对用户兴趣的理解能力。关于特征工程与用户兴趣建模的具体方法，我们将在本书第15章《特征工程》和第20章《用户兴趣建模》中进一步详细介绍。

12.2.2 模型结构设计

精排模型需要解决的第二个核心技术问题是模型结构设计，即如何有效建模特征之间的复杂关系？在确定输入特征之后，还需要设计合适的网络结构，将不同特征转化为准确的预测结果。不同模型结构的主要区别，在于如何表示特征、如何建模特征交互，以及如何利用不同来源的信息。例如，FM 通过低维隐向量建模特征之间的二阶交互，DeepFM 将显式交互与深层网络结合起来，DIN 根据候选物品动态提取相关历史兴趣，而 MMoE 和 PLE 则重点处理多个任务之间的共享与差异。这些模型的出现，分别对应了特征交互、兴趣建模和多任务知识共享等不同问题。

在工业实践中，精排模型结构设计还需要兼顾计算效率。复杂的交叉网络、更长的行为序列和更多的专家网络，都可能提高模型的表达能力，但也会增加训练和推理成本。因此，模型结构不能仅、只以网络深度、参数规模或者理论表达能力作为评价依据，而是应该结合实际的特征规模、候选物品数量、服务时延以及资源预算来进行设计。

随着推荐大模型的发展，这一问题进一步延伸到如何利用预训练表示、长上下文和更大规模的网络结构，以及如何在模型能力与线上服务成本之间取得平衡。无论采用传统 DNN、多任务网络还是推荐大模型，模型结构设计的核心都是让模型以合适的计算代价，有效利用输入信息并学习与目标任务相关的复杂关系。

12.2.3 训练与优化

精排模型需要解决的第三个核心技术问题是训练与优化，即如何让模型学到正确的目标，并将离线训练收益转化为线上业务效果？即使输入特征充分、网络结构合理，也不意味着模型一定能够获得理想的推荐效果。模

型最终学到什么，取决于训练样本、监督信号、损失函数、优化算法以及训练过程中的各种约束。

首先，工业级推荐数据通常存在样本规模大、反馈稀疏、正负样本比例不均衡等特点。如何构造训练样本、选择合适的监督信号，并在有限的计算资源下稳定训练模型，是精排模型需要解决的基础问题。其次，当精排模型同时预测点击、观看时长、点赞等多个目标时，不同任务的损失尺度、数据分布和优化方向可能存在差异。如何设计合理的任务权重与参数共享方式，避免某个任务过度主导训练过程，也是多任务精排中的重要问题。

此外，模型训练所优化的目标并不总是与线上业务指标完全一致。离线训练通常围绕可观测的用户反馈构造损失函数，而线上推荐效果还受到候选集合、排序策略、用户反馈机制以及内容生态等因素的共同影响。因此，如何评估模型的实际收益、识别离线与线上效果之间的差异，并通过持续实验迭代优化模型，是工业级精排模型不可缺少的一部分。还需要注意的是，在推荐大模型时代，精排模型的训练与优化还涉及大规模预训练、参数高效微调、分布式训练和推理优化等问题。

综上所述，精排模型的发展历程反映了工业级推荐系统对特征利用、模型表达能力和训练目标这三个层面不断深入的探索。而特征选择与表示、模型结构设计、训练与优化，则构成了理解和设计精排模型的三个基本视角。这三个方面并不是相互独立的，其中特征决定了模型能够利用哪些信息，网络结构决定了模型如何处理这些信息，训练与优化则决定了模型最终学习到什么，以及这些能力能否在工业场景中得到有效发挥。换言之，精排模型的设计需要在特征、结构和训练目标之间进行协调，以实现对用户兴趣的准确建模和对推荐效果的有效优化。

12.3 传统精排模型

传统精排模型是推荐排序技术发展的重要基础，其核心思路是利用用户、物品及上下文特征预测用户的潜在反馈，并据此对候选物品进行排序。不同模型从不同角度提升了对特征与反馈之间关系的建模能力。逻辑回归（LR）通过线性组合特征进行预测，具有结构简单、训练高效和易于部署等优点；梯度提升决策树（GBDT）通过集成多棵决策树建模非线性关系，能够从特征的分层划分中捕捉一定的特征组合模式；因子分解机（FM）进一步利用低维隐向量显式建模特征之间的二阶交互，从而增强对稀疏特征组合的学习能力；场感知因子分解机（FFM）则在此基础上引入特征域信息，为不同特征域之间的交互学习专门的表示。

这些模型体现了传统精排模型在特征建模方面的不同思路，也各有适用场景与局限。本节将依次介绍 LR、GBDT、FM 和 FFM，分析它们的基本原理、建模特点及工业应用中的主要考虑，为后续理解深度精排模型的架构设计与发展演进奠定基础。

12.3.1 逻辑回归（LR）

逻辑回归（Logistic Regression，LR）是早期工业级推荐系统中广泛应用的线性模型，常用于点击率等用户反馈概率的预估。其核心思想是先对输入特征进行线性组合，再通过 Sigmoid 函数将线性得分映射到 $[0,1]$ 区间，从而得到正反馈发生的预估概率。

假设输入样本的特征向量为 $\mathbf{x}=(x_1,x_2,\ldots,x_d)$，其中 d 为特征维度，w_i 为第 i 个特征对应的权重，w_0 为偏置项。LR 首先计算线性得分：

$$z(\mathbf{x})=w_0+\sum_{i=1}^{d}w_i x_i \tag{12.1}$$

对于二分类任务，模型进一步通过 Sigmoid 函数得到正类概率：

$$\hat{p}(\mathbf{x})=\sigma\big(z(\mathbf{x})\big)=\frac{1}{1+\exp\big(-z(\mathbf{x})\big)} \tag{12.2}$$

在推荐排序场景中，输入特征通常包括用户属性、物品属性、上下文信息以及用户与物品之间的匹配特征。例如，用户的历史行为统计、物品的类别与时长、用户所在的页面位置等，都可以编码为模型输入。对于高维稀疏

特征，LR 模型可以通过稀疏计算和参数更新进行高效训练，因此适合大规模推荐数据上的在线反馈预估。

LR 模型的训练通常采用交叉熵损失。假设训练样本为 $(\mathbf{x}_n, y_n)$，其中 $y_n \in \{0,1\}$，$\hat{p}_n$ 为模型对第 n 个样本预测的正类概率，则损失函数可写为：

$$\mathcal{L}_{\text{LR}} = -\sum_{n=1}^{N} [y_n \log \hat{p}_n + (1-y_n)\log(1-\hat{p}_n)] + \lambda \|\mathbf{w}\|_2^2 \tag{12.3}$$

其中，λ 为正则化系数，$\mathbf{w}$ 表示特征权重向量。需要注意的是，上述公式中第二项的正则化可以抑制部分权重过大，降低模型对训练数据的过度拟合风险。

LR 模型的主要优势在于模型结构简单、参数规模可控、训练和推理效率高，并且各特征对线性得分的贡献相对容易分析；但其表达能力也受到线性结构的限制。在不进行额外特征变换的情况下，LR 模型难以直接刻画不同特征之间的复杂关系。例如，用户属性与物品类别可能单独对点击概率的影响有限，但两者组合后却能体现明显的兴趣偏好。LR 模型本身无法自动学习这种交互关系，需要依赖人工构造交叉特征，才能将其纳入模型。

因此，在早期的工业级推荐系统中，特征工程是提升 LR 模型效果的重要手段。算法工程师需要结合业务理解，对用户行为、物品属性和上下文信息进行组合，挖掘有助于预测点击、转化等用户反馈的特征交叉。类似的思路也体现在早期的一些广告算法竞赛中，例如腾讯广告算法大赛的相关赛题：当传统机器学习模型占据主导时，参赛团队往往会投入大量精力进行人工特征构造，以挖掘数据中有价值的模式，而不仅仅依赖模型结构的复杂化。这种方法能够有效提升模型的预测能力，但也存在较高的人工成本，且难以穷尽不断增长的特征组合。随着推荐场景日趋复杂，如何减少对人工特征交叉的依赖、增强模型对非线性关系和特征交互的自动建模能力，逐渐成为传统精排模型演进的重要方向。后续的树模型和因子分解模型，分别从非线性划分和低维向量交互等角度，为解决这些问题提供了不同的思路。

12.3.2 梯度提升树（GBDT）

梯度提升决策树（Gradient Boosting Decision Tree，GBDT）是一种基于决策树的集成学习方法。与 LR 模型直接对输入特征进行线性组合不同，GBDT 模型通过多棵决策树逐步组合预测结果，能够建模特征与预测目标之间的非线性关系，并从树的分层划分中捕捉一定的特征组合模式。

一棵决策树通过一系列特征和阈值判断，将样本逐步划分到不同的叶节点。例如，模型可以先根据用户历史活跃度进行划分，再根据物品类别或物品时长继续细分，从而学习不同条件组合下的反馈差异。与人工枚举交叉特征相比，这种树结构能够**自动学习特征的划分条件及其组合方式**。不过，这类交互主要由树的分裂路径隐式表达，并不等同于对特征交互参数进行显式建模。

GBDT 模型采用逐步加法模型构建整体预测函数。假设模型包含 T 棵树，第 t 棵树对应的预测函数为 $h_t(\mathbf{x})$，学习率为 η，则整体模型可以表示为：

$$F_T(\mathbf{x}) = F_0(\mathbf{x}) + \eta \sum_{t=1}^{T} h_t(\mathbf{x}) \tag{12.4}$$

其中，$F_0(\mathbf{x})$ 为初始预测函数。在训练时，GBDT 模型在已有模型的基础上逐步添加新树，使得模型的整体预测结果不断向降低目标损失的方向调整。具体而言，每一轮通常根据当前模型的损失梯度构造伪残差，再训练一棵决策树拟合这些伪残差。对于平方误差损失，伪残差对应真实值与当前预测值之差；而对于逻辑损失等其他目标，则需要根据相应的负梯度进行拟合。因此，GBDT 模型并不局限于简单地拟合原始预测误差，而是通过梯度提升机制逐步优化整体目标。

在推荐系统中，GBDT 模型可以用于点击率等反馈的预估，也可以与其他模型组合，以发挥不同模型的优势。一种经典的组合方式是 GBDT+LR 的组合模型，即先利用 GBDT 模型将样本映射到对应的叶节点，再将叶节点转换为离散的指示特征，最后输入 LR 模型进行训练。由于叶节点对应着样本经过一系列特征划分后所处的区域，这种编码方式能够将 GBDT 模型自动学习到的非线性划分与特征组合转化为新的输入特征，再由 LR 模型进一步学习这些特征与目标反馈之间的关系。该方法结合了 GBDT 模型的非线性建模能力与 LR 模型的高效

参数学习能力，减少了对人工构造交叉特征的依赖，在当时的工业推荐与广告系统中得到了广泛应用。

从模型结构的演进来看，GBDT+LR 的组合模型也体现了一种重要的设计思路：先通过一个模块提取或构造更具表达力的特征，再由另一个模块进行组合建模与预测。在深度学习逐渐成为主流之后，这一思路进一步发展为将特征提取、特征交互模块与深度神经网络（DNN）相结合。例如，精排模型可以先通过特征交互模块构造具有一定语义的组合表示，再利用 DNN 模型学习更复杂的非线性关系。两者的具体实现虽然不同，但都试图将特征构造与预测建模相结合，提升模型对复杂推荐场景的表达能力。

不过，GBDT 模型也存在一定的局限。首先，树模型在处理大规模、高维稀疏特征，尤其是大量用户 ID、物品 ID 等高基数离散特征时，通常不如专门面向稀疏特征设计的模型灵活。其次，树的数量和深度会影响训练成本、模型体积及推理开销；在候选数量较大、线上延迟约束严格的排序链路中，需要在模型复杂度与预测收益之间进行权衡。此外，GBDT 模型通过逐轮增加决策树优化预测目标，难以像可微分神经网络那样，直接通过端到端梯度优化复杂的特征表示。

尽管如此，即使在深度学习模型不断发展的今天，GBDT 模型仍然具有值得借鉴的建模思想和实际应用价值。首先，GBDT 模型通过逐轮拟合当前损失函数的负梯度，不断修正已有模型的预测结果，这与深度神经网络中的残差学习思想存在一定的相通之处。例如，ResNet 模型通过残差连接使网络学习相对于输入的残差映射，改善深层网络的优化过程。两者的具体机制并不相同，但都体现了围绕已有表示或预测结果进行增量修正的思路。其次，GBDT 模型在特征划分与组合方面的能力，为后续推荐模型的特征构造和交互建模提供了重要启示，即如何从原始特征中提取有价值的组合关系，始终是推荐系统建模中的关键问题。

除了常规的反馈预估与特征构造，树模型在一些特定的工业场景中仍具有重要价值。例如，在因果效应估计与 uplift 建模中，树模型可以通过划分用户群体，刻画不同用户接受某种干预后产生的反馈差异，从而识别更可能从干预（treatment）中获益的用户。这类方法可用于激励算法、智能发券等场景，帮助系统从预测用户是否转化，进一步转向估计干预所带来的增量收益。在端侧推荐与排序中，经过适当控制树的数量和深度后，树模型也可以凭借相对轻量的计算结构部署在算力与内存受限的移动设备上。因此，虽然 GBDT 模型在大规模高维稀疏特征建模方面存在一定的局限性，但其在特定任务和资源约束下仍有实际应用空间。

总体而言，GBDT 模型擅长通过树结构学习非线性划分和条件组合，既可以独立用于反馈预估，也可以作为特征构造模块与其他模型结合。它的局限推动了后续模型在高维稀疏特征表示和特征交互方面的进一步发展，而其逐步优化与特征组合的思想，也持续影响着推荐模型的设计。接下来介绍的因子分解机（FM），则采用另一种思路：不再依赖树的分层划分，而是通过低维隐向量直接学习特征之间的交互关系。

12.3.3 因子分解机（FM）

因子分解机（Factorization Machine，FM）通过为每个特征学习一个低维隐向量，对特征之间的二阶交互进行显式建模。与 LR 模型依赖线性特征权重、GBDT 模型依赖树结构中的条件划分不同，FM 模型利用隐向量之间的内积计算特征交互强度，使模型能够在不显式枚举所有交叉特征的情况下，学习特征之间的关联关系。

假设输入特征向量为 $\mathbf{x} = (x_1, x_2, \ldots, x_d)$。除了偏置项和一阶特征项外，FM 模型为每个特征 i 学习一个 k 维隐向量 $\mathbf{v}_i \in \mathbb{R}^k$，并通过不同特征隐向量的内积刻画其交互强度。模型的预测得分为：

$$s(\mathbf{x}) = w_0 + \sum_{i=1}^{d} w_i x_i + \sum_{i=1}^{d-1} \sum_{j=i+1}^{d} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j \tag{12.5}$$

其中，w_0 为偏置项，w_i 为特征 i 的一阶权重，$\langle \mathbf{v}_i, \mathbf{v}_j \rangle$ 表示两个特征隐向量的内积。对于二分类反馈预测，可以进一步将得分通过 Sigmoid 函数转换为概率；而对于回归任务，则可以直接利用模型得分进行预测。

从上述公式中可以看到，FM 模型将特征交互系数表示为两个低维隐向量的内积：

$$\langle \mathbf{v}_i, \mathbf{v}_j \rangle = \sum_{f=1}^{k} v_{i,f} v_{j,f} \tag{12.6}$$

这种参数化方式有两个重要特点。第一，不同特征之间的交互不需要分别维护一个独立参数，而是通过各自的

隐向量共同确定交互系数。第二，即使某两个特征在训练数据中很少同时出现，它们仍可以通过各自与其他特征交互所学到的隐向量间接获得一定的泛化能力。这对于推荐系统中普遍存在的高维稀疏特征尤其重要。

例如，在点击率预估中，用户兴趣标签与物品类别之间可能存在显著的关联。如果直接为每一种用户标签和物品类别组合建立独立参数，特征组合数量可能非常庞大，而且部分组合缺少足够的训练样本。FM 则为每个特征维护低维隐向量，通过向量内积计算任意两个特征之间的交互，从而在控制参数规模的同时共享不同特征组合中的统计信息。

尽管 FM 模型的表达式包含所有特征对，其计算并不一定需要显式枚举所有交互项。利用内积展开和平方项的恒等关系，二阶交互部分可以改写为：

$$\sum_{i=1}^{d-1}\sum_{j=i+1}^{d}\langle \mathbf{v}_i,\mathbf{v}_j\rangle x_i x_j = \frac{1}{2}\sum_{f=1}^{k}\left[\left(\sum_{i=1}^{d} v_{i,f}x_i\right)^2 - \sum_{i=1}^{d} v_{i,f}^2 x_i^2\right] \tag{12.7}$$

借助这一等价形式，FM 模型对单个样本的计算复杂度可以由直接枚举特征对时的二次复杂度，降低到约 $O(k\,\mathrm{nnz}(\mathbf{x}))$，其中 $\mathrm{nnz}(\mathbf{x})$ 表示输入特征向量中非零特征的数量。这使 FM 能够较高效地处理大规模稀疏输入。

FM 模型的主要优势是能够自动学习二阶特征交互，对稀疏特征具有较好的适应性，并通过低维参数化控制模型规模。但其标准形式主要显式建模二阶交互，难以直接表达更复杂的高阶关系。同时，同一个特征使用相同的隐向量与其他所有特征交互，没有区分不同特征域带来的语义差异。例如，用户年龄与物品类别之间的交互，与物品类别和上下文位置之间的交互，可能需要不同的表示方式。针对这一问题，场感知因子分解机进一步在特征交互过程中引入了特征域信息。

12.3.4 场感知因子分解机（FFM）

场感知因子分解机（Field-aware Factorization Machine，FFM）是在 FM 基础上的扩展，其核心思想如下：

定义 12.1

一个特征与不同特征域中的特征发生交互时，应该允许使用不同的隐向量表示。

通过显式区分特征所属的域，FFM 模型能够更细致地刻画不同类型特征之间的交互关系。在推荐系统中，特征通常按照来源或语义划分为不同的域（Field），例如用户属性、物品属性、用户历史行为、上下文信息等。FM 为每个特征学习一个统一的隐向量，无论它与哪一个特征交互，都使用同一套向量参数。FFM 则为特征 i 针对不同特征域分别维护隐向量。设特征 i 所属的特征域为 f_i，当它与属于特征域 f_j 的特征 j 交互时，使用 $\mathbf{v}_{i,f_j}$ 表示特征 i 面向特征域 f_j 的隐向量。

因此，FFM 模型的预测得分可以表示为：

$$s(\mathbf{x}) = w_0 + \sum_{i=1}^{d} w_i x_i + \sum_{i=1}^{d-1}\sum_{j=i+1}^{d}\langle \mathbf{v}_{i,f_j},\mathbf{v}_{j,f_i}\rangle x_i x_j \tag{12.8}$$

与 FM 模型相比，FFM 模型的一阶特征项保持不变，主要区别在于二阶交互项的参数化方式。对于特征 i 和特征 j，交互系数由特征 i 面向特征 j 所属域的隐向量，与特征 j 面向特征 i 所属域的隐向量共同决定。这样，同一个特征可以在不同的交互场景下学习不同的表示，不必使用统一的隐向量描述它与所有特征的关系。

例如，对于用户属性、物品属性和上下文信息三个特征域，用户年龄与物品类别之间的交互可以使用专门面向物品域的用户年龄向量；而用户年龄与上下文位置之间的交互，则可以使用面向上下文域的另一组向量。两种交互由不同的参数表示，因此模型能够更灵活地学习特征域之间的差异。对于包含大量离散 ID 与类别特征的推荐任务，这种场感知机制有助于增强模型对特征组合的表达能力。

不过，这种表达能力的提升也带来了额外的计算和存储开销。由于每个特征需要针对不同特征域维护相应的隐向量，所以 FFM 模型的参数规模通常会明显大于 FM 模型。此外，在计算交互项时，FFM 模型也需要逐对考虑非零特征之间的关系，其计算成本会随着单个样本中有效特征数量的增加而快速上升。因此，在大规模推

荐系统中，FFM 模型的应用需要综合考虑特征数量、域的划分方式、隐向量维度以及训练和推理资源。

总体而言，LR、GBDT、FM 和 FFM 模型分别代表了几种不同的传统特征建模思路。其中，LR 模型通过线性权重进行预测，GBDT 模型通过树结构学习非线性划分，FM 模型通过低维隐向量学习二阶特征交互，FFM 模型则进一步使特征交互具备域感知能力。这些模型为推荐排序提供了重要的建模基础，但它们在特征表示方式、交互阶数和参数共享机制方面仍存在各自的限制。随着深度学习的发展，后续的深度模型开始将稀疏特征嵌入、显式交互结构与深层神经网络结合起来，以进一步提升对复杂推荐场景的表达能力。

12.4 深度精排模型的基础架构

随着深度学习在推荐系统中的应用不断深入，精排模型逐渐从以线性组合、树模型和显式特征交互为主的传统架构，发展出融合特征表示学习、非线性建模与多种监督信号的深度网络架构。从模型结构来看，这一演进过程体现了几个重要方向：一类模型将显式特征交互与深层神经网络相结合，通过不同模块的互补提升模型的表达能力；另一类模型围绕点击、观看时长、转化等不同反馈目标组织多个预测任务，以充分利用业务中的多源监督信号；随着 Transformer 等架构的发展，模型也开始探索在统一网络中对更加丰富的特征和用户行为序列进行交互与表示学习。

这些架构并非简单的前后替代关系，而是分别从特征交互、表征学习和监督信号组织等角度拓展了精排模型的能力。其中，显式特征交互与深层表征融合是一类具有代表性的基础架构。Wide & Deep、DeepFM 和 xDeepFM 等模型均体现了将特定的特征交互机制与深度神经网络相结合的设计思路，但它们在交互方式、参数共享和网络组织上存在明显差异。理解这类模型，有助于进一步认识深度精排模型如何从依赖人工构造特征，逐步发展为利用神经网络自动学习特征表示与复杂关系。

精排模型在深度学习时代及后续时代的演进过程中，逐步形成了基于深度神经网络的多种架构。最早期是基于显示特征交互与深层表征融合的模型架构，再到后续结合业务多种不同目标信号所形成的多任务学习的模型架构，再到推荐大模型时代基于 Transformer 的统一精排模型架构。在整个过程中，模型的网络结构从最开始的多模块堆叠到最终使用 Transformer 及其变种的大一统结构进行特征交互与表征学习，体现了从局部特征建模到全局特征建模的演进趋势。同时，随着算力的逐步发展，模型的规模和复杂度也在不断提升，使得精排模型能够处理更加丰富的特征信息和更复杂的用户行为模式，在模型层面也能达到更高的预估精度，进而在推荐排序任务中取得更优的性能。不过，模型规模的扩大并不必然带来线上收益，在实际应用中仍然需要综合考虑预测效果、训练成本、推理延迟及系统部署等多个方面的约束，而不能一味地无脑堆机器进行模型的 scaling，否则可能导致资源浪费和系统性能下降。

12.4.1 显式特征交互与深层表征融合的模型架构

传统精排模型通常需要依赖人工构造特征交叉，才能在预测过程中利用不同特征之间的组合关系。随着深度学习的发展，一类典型的精排模型开始将特征交互模块与深度神经网络结合起来：前者通过预先定义的交叉特征或专门设计的交互运算，强化模型对特定特征关系的建模；后者通过嵌入表示和多层非线性变换，学习更加复杂的特征组合与高阶关系。这种将显式特征交互与深层表征学习相结合的设计，可以概括为显式特征交互与深层表征融合架构。

这类模型设计的关键在于不同模块之间的互补。一方面，显式交互模块能够直接表达特定特征组合，为模型提供结构明确的交互信息；另一方面，深度神经网络能够从训练数据中学习稠密特征表示，并通过多层非线性变换建模更加复杂的关系。两者相结合，既可以利用显式交互对特定模式进行建模，也能够发挥深度网络在特征表示学习方面的优势，实现精排模型对于特征低阶交互与高阶交互的统一建模。

需要注意的是，这类模型并不一定具有严格的“底层交互、上层 MLP”的串行结构。例如，有些模型将特征交互模块与 DNN 设计为并行分支，分别学习特征关系，最后融合两路输出；另一些模型则在网络内部逐层组织特征交互。其中，Wide & Deep 是这一设计思路的经典代表，后续的 PNN、DeepFM、xDeepFM 等模型则进一步探索了特征交互的不同实现方式。

12.4.1.1 Wide & Deep 模型

Wide & Deep 模型由 Google 团队提出，相关论文《Wide & Deep Learning for Recommender Systems》发表于 2016 年 RecSys 会议的 Deep Learning for Recommender Systems（DLRS）研讨会 [4]。该模型针对推荐系统中特征记忆与泛化之间的矛盾展开设计。传统线性模型结合人工构造的交叉特征，能够直接学习特定特征组合与用户反馈之间的关系，但其建模能力受到特征设计的限制，难以覆盖大量潜在的特征交互。深度神经网络则可以通过嵌入表示和多层非线性变换学习更加复杂的特征关系，但对于一些重要且训练样本相对稀疏的特征组合，也未必能够充分学习。为兼顾这两方面的能力，Wide & Deep 将广义线性模型（Wide）与深度神经网络（Deep）结合起来，并通过统一的预测目标进行联合训练，使两个分支能够在同一任务下相互补充。

如图 12.2 所示，Wide & Deep 包含两个并行的建模分支。Wide 分支利用原始特征及显式构造的交叉特征进行线性建模，侧重于记忆具有明确预测价值的特征组合；而 Deep 分支则将高维稀疏特征转换为低维稠密的嵌入表示，再通过多层神经网络学习复杂的非线性关系。两个分支分别计算预测得分，经过融合后得到最终的预测结果。这样的设计既保留了显式特征交叉的直接建模能力，也利用深度网络增强了对未充分观测特征组合的泛化能力。

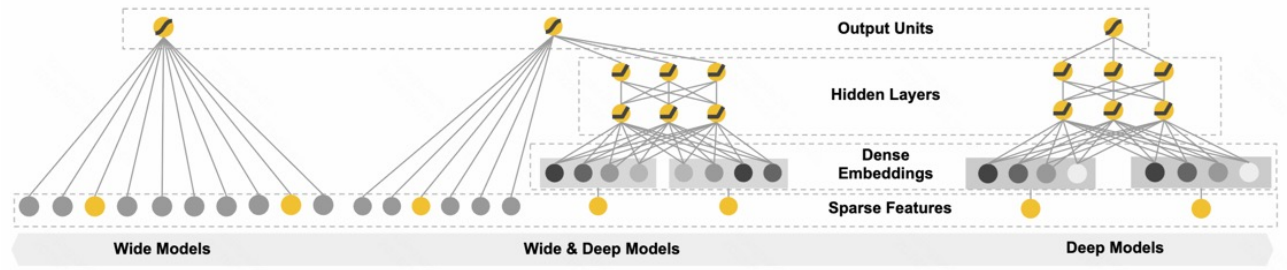


图 12.2: Wide & Deep 模型网络结构。

Wide 分支本质上是一个广义线性模型，其预测得分可以表示为：

$$z_{\text{wide}}(\mathbf{x}) = \mathbf{w}_{\text{wide}}^{\top} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\phi}(\mathbf{x}) \end{bmatrix} + b_{\text{wide}} \tag{12.9}$$

其中，$\mathbf{x}$ 为原始输入特征，$\boldsymbol{\phi}(\mathbf{x})$ 表示由原始特征转换得到的交叉特征，$\mathbf{w}_{\text{wide}}$ 为相应的模型参数，b_{wide} 为偏置项。与直接对原始特征进行线性建模的 LR 模型相比，Wide 分支通过特征转换扩展了模型能够利用的特征空间，使模型能够直接学习特定特征组合对预测结果的影响。

对于由二值特征构成的交叉特征，一种常见的构造方式为：

$$\phi_k(\mathbf{x}) = \prod_{i=1}^{d} x_i^{c_{ki}}, \qquad c_{ki} \in \{0, 1\} \tag{12.10}$$

其中，$c_{ki} = 1$ 表示第 i 个特征参与第 k 个交叉特征的构造，$c_{ki} = 0$ 则表示该特征不参与。当输入特征为二值指示变量时，只有参与交叉的所有特征同时取值为 1，对应的交叉特征才会激活。例如，可以将“用户性别为女性”和“语言为英语”组合成一个新的交叉特征，只有这两个条件同时满足时，该特征才取值为 1。通过这种方式，模型能够直接学习特定用户属性与上下文条件组合所对应的反馈规律，而不必仅依赖各个原始特征的独立贡献。

显式交叉特征能够有效表达具有明确业务意义的特征组合，但其效果在很大程度上依赖特征设计。当特征数量不断增加时，潜在的组合规模也会迅速扩大，人工枚举和维护所有有价值的交叉特征既不现实，也容易造成特征空间膨胀。因此，Wide 分支通常会侧重于建模一部分重要且具有明确预测价值的交叉特征。

与 Wide 分支通过显式交叉特征进行建模不同，Deep 分支采用前馈神经网络，通过特征嵌入和多层非线性变换学习复杂的特征关系。推荐系统中的类别特征通常具有高维、稀疏的特点，例如用户 ID、物品 ID、类别标签和上下文取值等。模型通常先将这些类别特征转换为低维稠密的嵌入向量（Embedding），再将其输入深度网络进行学习。

假设第 j 个类别特征的输入向量为 $\mathbf{x}_j$，其嵌入矩阵为 $\mathbf{E}_j$，则对应的嵌入表示可以写为：

$$\mathbf{e}_j = \mathbf{E}_j^\top \mathbf{x}_j \tag{12.11}$$

对于采用独热编码的输入，$\mathbf{e}_j$ 相当于从嵌入矩阵中取出与当前类别对应的向量。随后，模型将各类别特征的嵌入向量与数值型特征进行拼接，形成深度网络的输入：

$$\mathbf{h}^{(0)} = [\mathbf{e}_1; \mathbf{e}_2; \cdots; \mathbf{e}_m; \mathbf{x}_{\text{dense}}] \tag{12.12}$$

其中，m 为类别特征的数量，$\mathbf{x}_{\text{dense}}$ 为数值型特征向量，分号表示向量拼接。

在此基础上，Deep 分支通过多层全连接网络逐步学习特征之间的非线性关系。第 l 层的计算可以表示为：

$$\mathbf{h}^{(l+1)} = f\left(\mathbf{W}^{(l)}\mathbf{h}^{(l)} + \mathbf{b}^{(l)}\right) \tag{12.13}$$

其中，$\mathbf{W}^{(l)}$ 和 $\mathbf{b}^{(l)}$ 分别表示第 l 层的权重矩阵和偏置向量，$f(\cdot)$ 为非线性激活函数，例如 ReLU 函数。在经过多层变换之后，Deep 分支会得到最终的深层表征 $\mathbf{h}^{(L)}$。与 Wide 分支直接利用预先构造的交叉特征不同，Deep 分支主要通过嵌入学习和多层非线性变换，从训练数据中学习特征之间的复杂关系。这种方式不需要显式枚举所有特征组合，能够在一定程度上缓解人工交叉特征覆盖范围有限的问题。不过，Deep 分支学习到的交互关系通常是隐式的，其效果仍然受到训练数据、网络结构和优化过程的影响。

在完成两个分支的特征建模后，Wide & Deep 模型将它们的输出得分相加，并通过 Sigmoid 函数得到最终的正反馈概率。对于二分类反馈预估任务，模型可以表示为：

$$\hat{p}(Y=1 \mid \mathbf{x}) = \sigma\left(z_{\text{wide}}(\mathbf{x}) + \mathbf{w}_{\text{deep}}^\top \mathbf{h}^{(L)} + b_{\text{deep}}\right) \tag{12.14}$$

其中，$\mathbf{w}_{\text{deep}}$ 为 Deep 分支输出层的权重，b_{deep} 为该分支的偏置项，$\sigma(\cdot)$ 为 Sigmoid 函数。两个分支的得分在进入 Sigmoid 函数之前进行融合，因此各分支输出的数值实际上是各自对于最终预测的贡献。

Wide & Deep 模型采用统一的目标损失函数进行联合训练。以二分类任务的交叉熵损失为例，在训练过程中，模型根据最终预测结果计算损失，并将梯度反向传播到 Wide 分支、Deep 分支及其融合输出涉及的参数，从而同时优化两部分的预测能力。这与传统的独立集成方式有所不同。在独立集成的模型中，不同模型通常先分别训练，再在推理阶段组合各自的预测结果；而在 Wide & Deep 中，两个分支在训练阶段就通过共同的损失函数进行优化，使它们能够围绕同一预测目标形成互补。原论文的实验实现中，Wide 分支采用带有 L_1 正则化的 FTRL（Follow the-regularized-leader）优化方法，Deep 分支采用 AdaGrad 优化方法，以适应两类参数和特征表示的不同特点。这些优化器属于具体的训练配置，并非 Wide & Deep 架构本身的必要组成部分。

总的来说，Wide & Deep 模型的重要贡献在于，将显式特征交叉与深度表征学习组织在同一个模型中，使两种建模方式能够相互补充。Wide 分支擅长记忆已知且具有预测价值的特征组合，Deep 分支则通过学习稠密表示和非线性变换，对更广泛的特征关系进行泛化。不过，Wide 分支仍然需要人工设计和维护交叉特征，而 Deep 分支也没有针对不同阶数的特征交互进行专门建模。当模型需要处理大量高维稀疏特征，并希望减少对人工交叉特征的依赖时，如何进一步改进交互模块便成为后续精排模型的重要研究方向。DeepFM 模型正是在这一思路上进一步发展的，它以因子分解机（FM）模块替代 Wide & Deep 模型中依赖人工交叉特征的 Wide 分支，使模型能够通过特征嵌入自动学习二阶交互，同时保留 DNN 对复杂非线性关系的建模能力。

12.4.1.2 PNN 模型

PNN（Product-based Neural Network）模型源自于上海交通大学研究团队发表在 ICDM 2016 会议的论文《Product-based Neural Networks for User Response Prediction》[16]。该模型针对传统深度 CTR 预估模型主要依赖 MLP 隐式学习特征交互的问题，引入 Product Layer 显式构造不同特征字段之间的乘性交互，再将交互结果输入后续神经网络进行非线性建模。其核心思想如下：

定义 12.2

PNN 模型通过向网络中引入向量乘积操作，让模型能够更直接地学习不同特征之间的关联关系，从而增强对多字段离散特征的表达能力。

在 CTR 预估的场景中，一个训练样本通常包含多个特征字段，例如用户所在城市、请求发生时间、广告位、广告创意 ID 和广告计划 ID 等。这些特征通常以多字段类别特征的形式组织，每个字段内部采用 One-Hot 编码。由于不同字段具有各自的取值空间，直接将所有类别特征展开会产生维度很高且十分稀疏的输入表示。同时，不同字段之间还可能存在复杂的关联关系，例如用户所在城市与广告投放区域之间的关系、用户属性与广告类型之间的关系等。仅仅依靠普通的全连接网络从原始稀疏输入中学习这些关系，可能需要较大的网络容量和较多的训练数据。

PNN 模型通过 Embedding 层将不同字段映射为稠密向量，并在此基础上引入 Product Layer，显式计算字段之间的乘积交互。其整体网络结构如图 12.3 所示，主要包括 Embedding 层、Product Layer 和后续的全连接网络。其中，Product Layer 同时保留原始嵌入表示中的线性信息，并构造不同字段之间的二阶交互信息，最终将两类信号融合后输入 DNN，完成点击概率预测。

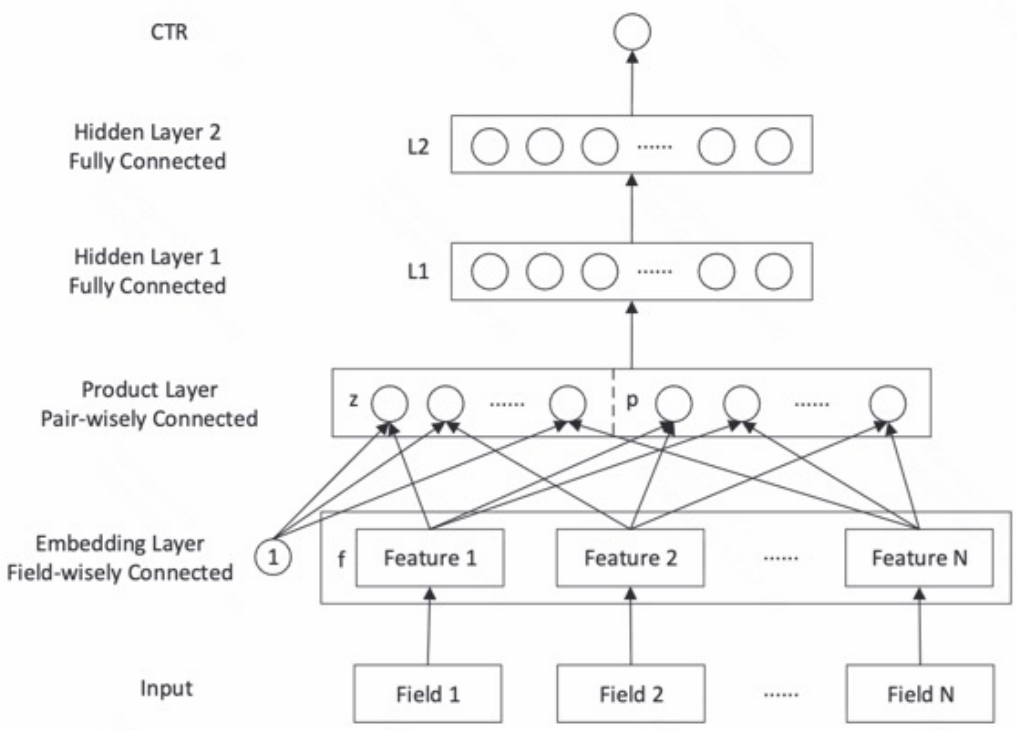


图 12.3: PNN 模型的网络结构。

具体而言，假设输入样本包含 N 个特征字段，第 i 个字段经过 One-Hot 编码后表示为 $\mathbf{x}_i$，对应的嵌入矩阵为 $\mathbf{W}_0^{(i)}$，则该字段的 Embedding 表示为

$$\mathbf{f}_i = \mathbf{W}_0^{(i)}\mathbf{x}_i, \qquad \mathbf{f}_i \in \mathbb{R}^M \tag{12.15}$$

其中，M 为 Embedding 维度。将所有字段的嵌入向量拼接起来，可以得到线性特征表示：

$$\mathbf{z} = \text{Concat}\left(\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_N\right) \in \mathbb{R}^{NM} \tag{12.16}$$

在此基础上，PNN 模型通过 Product Layer 构造特征之间的二阶交互。假设不同特征字段之间的乘积交互结果为 $\mathbf{p}$，则 Product Layer 同时接收原始嵌入表示 $\mathbf{z}$ 和交互表示 $\mathbf{p}$。对于 Product Layer 的第 n 个隐藏节点，它的输出由两部分组成：一部分是对原始嵌入表示进行线性变换得到的信号，另一部分是对乘积交互结果进行加权聚合所

得到的信号。具体可以写为如下形式

$$l_1^{(n)} = \mathrm{ReLU}\left(l_z^{(n)} + l_p^{(n)} + b_1^{(n)}\right) \tag{12.17}$$

其中，

$$l_z^{(n)} = \langle \mathbf{W}_z^{(n)}, \mathbf{z} \rangle, \qquad l_p^{(n)} = \langle \mathbf{W}_p^{(n)}, \mathbf{p} \rangle \tag{12.18}$$

这里的 $\langle \cdot, \cdot \rangle$ 表示对应元素相乘后求和的内积操作，$\mathbf{W}_z^{(n)}$ 和 $\mathbf{W}_p^{(n)}$ 分别表示第 n 个隐藏节点对应的线性变换参数和交互变换参数。将所有隐藏节点的输出组合起来，得到 Product Layer 的输出 $\mathbf{l}_1 \in \mathbb{R}^{D_1}$，其中 D_1 为第一层隐藏维度。

随后，Product Layer 的输出经过多层全连接网络，进一步学习不同特征交互之间的高阶非线性关系：

$$\mathbf{l}_2 = \mathrm{ReLU}\left(\mathbf{W}_2 \mathbf{l}_1 + \mathbf{b}_2\right) \tag{12.19}$$

最终通过 Sigmoid 输出点击概率：

$$\hat{y} = \sigma\left(\mathbf{W}_3 \mathbf{l}_2 + b_3\right) \tag{12.20}$$

其中，$\mathbf{W}_2$、$\mathbf{W}_3$ 以及对应的偏置参数均通过训练学习。PNN 模型采用二元交叉熵损失进行优化：

$$\mathcal{L}_{\mathrm{PNN}} = -\frac{1}{N_s} \sum_{i=1}^{N_s} \left[y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i) \right] \tag{12.21}$$

其中，N_s 为训练样本数量，y_i 为真实点击标签。

PNN 模型的关键在于如何定义 Product Layer 中的特征乘积操作。根据特征乘积方式的不同，PNN 模型可以分为基于内积的 IPNN（Inner Product-based Neural Network）模型和基于外积的 OPNN（Outer Product-based Neural Network）模型。两者的主要区别在于特征交互结果的表达形式不同，其中 IPNN 模型将两个字段的嵌入向量通过内积映射为一个标量，OPNN 模型则通过外积生成一个矩阵，从而保留更加丰富的维度交互信息。在 IPNN 模型中，两个特征字段之间的交互定义为其嵌入向量的内积：

$$p_{i,j} = \langle \mathbf{f}_i, \mathbf{f}_j \rangle = \mathbf{f}_i^{\top} \mathbf{f}_j \tag{12.22}$$

对于包含 N 个字段的输入样本，所有字段两两之间的内积构成一个交互矩阵：

$$\mathbf{p} = \{p_{i,j}\} \in \mathbb{R}^{N \times N} \tag{12.23}$$

由于内积具有交换性，即 $p_{i,j} = p_{j,i}$，因此这个交互矩阵是对称的。Product Layer 再通过可学习的权重矩阵，对这些二阶交互结果进行加权组合。

如果直接为每个隐藏节点学习一个独立的交互权重矩阵，那么随着特征字段数量的增加，交互部分的计算和参数开销也会迅速增长。对于第 n 个隐藏节点，交互信号可以写为

$$l_p^{(n)} = \sum_{i=1}^{N} \sum_{j=1}^{N} \left(\mathbf{W}_p^{(n)}\right)_{i,j} \langle \mathbf{f}_i, \mathbf{f}_j \rangle \tag{12.24}$$

上述计算涉及 N^2 个字段的交互项。考虑到 $\mathbf{W}_p^{(n)}$ 是对称矩阵，PNN 模型进一步借鉴矩阵分解的思想，对交互权重进行低秩分解。最简单的情况是将其表示为两个向量的外积：

$$\mathbf{W}_p^{(n)} = \boldsymbol{\theta}^{(n)} \left(\boldsymbol{\theta}^{(n)}\right)^{\top} \tag{12.25}$$

其中，$\boldsymbol{\theta}^{(n)} \in \mathbb{R}^N$ 为可学习向量。代入原有的交互计算公式，可以得到

$$l_p^{(n)} = \sum_{i=1}^{N} \sum_{j=1}^{N} \theta_i^{(n)} \theta_j^{(n)} \langle \mathbf{f}_i, \mathbf{f}_j \rangle = \left\langle \sum_{i=1}^{N} \theta_i^{(n)} \mathbf{f}_i, \sum_{j=1}^{N} \theta_j^{(n)} \mathbf{f}_j \right\rangle = \left\| \sum_{i=1}^{N} \theta_i^{(n)} \mathbf{f}_i \right\|_2^2 \tag{12.26}$$

这里通过交换求和顺序，将原本需要遍历所有字段对的计算，转化为先对字段嵌入进行加权求和，再计算向量的平方范数。记

$$\boldsymbol{\delta}_i^{(n)} = \theta_i^{(n)} \mathbf{f}_i \tag{12.27}$$

则上述计算可以简化为

$$l_p^{(n)} = \left\| \sum_{i=1}^{N} \boldsymbol{\delta}_i^{(n)} \right\|_2^2 \tag{12.28}$$

因此，原本与字段数量平方相关的交互计算，可以转化为与字段数量线性相关的计算。对于包含 D_1 个隐藏节点、N 个特征字段且 Embedding 维度为 M 的模型，采用上述秩一分解后，交互部分的计算复杂度可以降低到 $O(D_1MN)$。

秩一分解虽然能够有效降低计算开销，但也限制了交互权重矩阵的表达能力。为了在计算复杂度和模型表达能力之间取得平衡，还可以使用更一般的低秩分解形式：

$$\left(\mathbf{W}_p^{(n)}\right)_{i,j} = \langle \boldsymbol{\theta}_i^{(n)}, \boldsymbol{\theta}_j^{(n)} \rangle \tag{12.29}$$

其中，$\boldsymbol{\theta}_i^{(n)} \in \mathbb{R}^K$，$K$ 为分解秩。此时交互信号可以表示为

$$l_p^{(n)} = \sum_{i=1}^{N}\sum_{j=1}^{N} \langle \boldsymbol{\theta}_i^{(n)}, \boldsymbol{\theta}_j^{(n)} \rangle \langle \mathbf{f}_i, \mathbf{f}_j \rangle = \sum_{r=1}^{K} \left\| \sum_{i=1}^{N} \theta_{i,r}^{(n)} \mathbf{f}_i \right\|_2^2 \tag{12.30}$$

当 $K=1$ 时，该形式退化为前面的秩一分解。随着 K 增大，模型能够学习更加丰富的交互权重关系，但相应的参数量和计算量也会增加。

与 IPNN 模型使用内积将两个嵌入向量压缩为一个标量不同，OPNN 模型使用外积来构造字段之间的矩阵交互：

$$p_{i,j} = \mathbf{f}_i \mathbf{f}_j^\top, \qquad p_{i,j} \in \mathbb{R}^{M \times M} \tag{12.31}$$

这种方式不仅能够反映两个特征向量之间的整体相似程度，还能够保留不同 Embedding 维度之间的交互信息。例如，$\mathbf{f}_i$ 的第一个维度与 $\mathbf{f}_j$ 的第二个维度之间的关联，也会在外积矩阵中得到显式表示。因此，相比内积形式，外积能够提供更加丰富的二阶交互特征。

不过，如果为所有字段对分别构造外积矩阵，则交互结果的规模会达到 N^2M^2，计算和存储开销都比较高。为了解决这一问题，OPNN 模型使用了特征叠加（Superposition）的方式，将所有字段的 Embedding 先进行求和，再对聚合后的向量执行外积：

$$\mathbf{f}_\Sigma = \sum_{i=1}^{N} \mathbf{f}_i \tag{12.32}$$

$$\mathbf{p} = \sum_{i=1}^{N}\sum_{j=1}^{N} \mathbf{f}_i \mathbf{f}_j^\top = \left(\sum_{i=1}^{N} \mathbf{f}_i \right) \left(\sum_{j=1}^{N} \mathbf{f}_j \right)^\top = \mathbf{f}_\Sigma \mathbf{f}_\Sigma^\top \tag{12.33}$$

通过这种方式，原本需要为每一对字段分别计算的外积被压缩为一个 $M \times M$ 的矩阵。对于第 n 个隐藏节点，外积交互信号可以写为

$$l_p^{(n)} = \left\langle \mathbf{W}_p^{(n)}, \mathbf{f}_\Sigma \mathbf{f}_\Sigma^\top \right\rangle \tag{12.34}$$

其中，$\mathbf{W}_p^{(n)} \in \mathbb{R}^{M \times M}$ 为对应的交互权重矩阵。由于外积结果是对称矩阵，因此只需要考虑对称部分的权重。采用这种叠加方式后，计算复杂度可以降低到 $O(D_1M(M+N))$。

需要注意的是，OPNN 模型的特征叠加方法虽然显著降低了计算成本，但也会将不同字段对的外积结果聚合起来，不再单独保留每一对特征之间的完整交互表示。这意味着它在控制复杂度的同时，也对交互信息进行了压缩。因此，IPNN 模型与 OPNN 模型并不是简单的计算形式替换，而是在交互信息的表达形式、计算复杂度以及参数规模之间采用了不同的设计取舍。

从整体架构来看，PNN 模型的 Product Layer 是其区别于传统 FNN 模型的关键组件。如果去掉 Product Layer 中的二阶交互部分，仅保留原始 Embedding 的线性输入信号，网络就退化为基于 Embedding 和 MLP 的普通深度模型，即 FNN 类结构。另一方面，如果使用 IPNN 模型的内积交互，并进一步简化后续的隐藏网络，使交互项通过线性方式进行聚合，则可以得到与 FM 具有相似形式的二阶特征交互模型。由此可见，PNN 模型可以看作在传统深度网络中显式引入乘积交互的一种扩展，一方面保留 DNN 对复杂非线性关系的建模能力，另一方面通

过 Product Layer 为网络提供更加直接的特征交互信号。

12.4.1.3 DeepFM 模型

DeepFM 模型由华为诺亚方舟实验室发表在 IJCAI 2017 学术会议上的论文《DeepFM: A Factorization-Machine based Neural Network for CTR Prediction》[9]。该模型针对点击率预估中如下所示的核心问题展开设计：

定义 12.3

推荐系统的输入通常包含大量高维稀疏的用户、物品和上下文特征，模型既需要捕捉单个特征对用户反馈的影响，也需要学习不同特征之间的交互关系。前者决定了用户属性、物品属性等因素各自对预测结果的贡献，后者则有助于识别只有在特定特征组合下才会表现出来的用户偏好。 ♣

此前介绍的 Wide & Deep 模型采用两个相互独立的分支分别建模低阶和高阶特征关系，其中 Wide 部分依赖显式构造的交叉特征，Deep 部分通过嵌入层和多层感知机学习复杂的非线性关系。DeepFM 模型延续了这一设计思路，但将 Wide 分支替换为因子分解机（Factorization Machine，FM），并进一步让 FM 模型与深度网络共享特征嵌入。这样，模型可以通过 FM 模型显式建模一阶和二阶特征关系，同时利用深度网络学习更复杂的高阶特征交互，而不必事先对 FM 模型进行预训练，也不需要像 Wide & Deep 那样为 Wide 分支专门构造交叉特征。

DeepFM 模型的输入是一条包含用户、物品及上下文信息的样本记录。假设训练数据包含 n 条样本，每条样本记为 $(\boldsymbol{x}, y)$，其中 $y \in \{0, 1\}$ 表示用户是否点击目标物品。输入特征可以包含性别、地域等类别特征，也可以包含年龄等连续特征。类别特征通常采用独热编码表示，连续特征则可以直接使用数值，也可以在离散化后转换为相应的特征表示。将各字段转换为统一的稀疏特征向量后，记为

$$\boldsymbol{x} = [x_1, x_2, \ldots, x_d]^\top \tag{12.35}$$

其中 d 为编码后的特征维度。由于类别特征往往具有很高的基数，$\boldsymbol{x}$ 通常具有高维、稀疏的特点，这也是模型需要采用专门的特征交互结构和嵌入表示的原因。

如图 12.4 所示，DeepFM 模型由 FM 组件和 Deep 组件构成，两者接收相同的输入特征，并在最终预测层融合各自的输出。FM 组件负责建模一阶和二阶特征交互，而 Deep 组件则先通过嵌入层将稀疏输入转换为低维稠密表示，再利用多层感知机学习高阶特征交互。两个组件不是分别训练后再进行组合，而是在统一的预测目标下端到端联合训练。

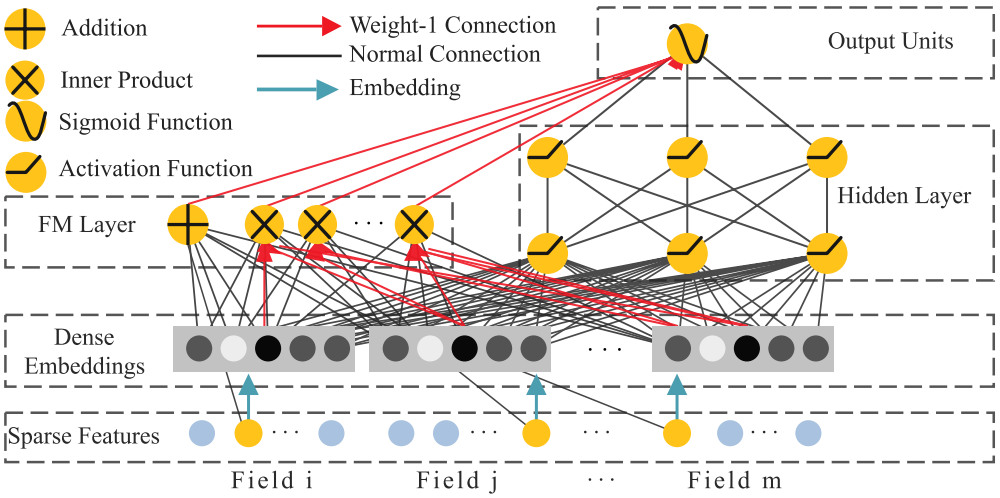


图 12.4: DeepFM 模型网络结构。

对于 FM 组件，模型不仅学习每个特征的一阶权重，还为每个特征学习一个低维隐向量，用于刻画该特征

与其他特征之间的交互关系。其输出可以表示为

$$z_{\mathrm{FM}} = b_{\mathrm{FM}} + \sum_{i=1}^{d} w_i x_i + \sum_{i=1}^{d} \sum_{j=i+1}^{d} \langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle x_i x_j \tag{12.36}$$

其中，b_{FM} 为偏置项，w_i 为第 i 个特征的一阶权重，$\boldsymbol{v}_i \in \mathbb{R}^k$ 为该特征对应的 k 维隐向量，$\langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle$ 表示两个隐向量的内积。公式中的第二项刻画各个特征对预测结果的独立贡献，第三项则刻画任意两个特征之间的二阶交互。

FM 模型的关键在于利用隐向量的内积来表示特征交互。在传统的线性模型中，模型只有在某个特征组合出现在训练样本中时，才能直接学习该组合对应的参数。而当特征组合出现次数较少时，参数估计往往不够充分。FM 模型则通过每个特征自身的隐向量学习其与其他特征的关系。即使某个特征组合很少出现，只要组成该组合的特征分别在其他样本中出现过，FM 模型就可以利用这些特征学到的隐向量来估计它们之间的交互关系。这种参数共享方式能够缓解高维稀疏数据中交互样本不足的问题。

与 FM 组件不同，Deep 组件的目标是学习比二阶交互更加复杂的特征关系。由于原始输入向量维度高且十分稀疏，直接将其输入 DNN 网络，不仅会带来较大的参数规模，也不利于网络的有效学习。因此，DeepFM 模型在输入端设置嵌入层，将不同字段映射为维度相同的低维稠密向量。假设输入包含 m 个字段，第 j 个字段对应的嵌入表示为：

$$\boldsymbol{e}_j = \sum_{i \in \mathcal{F}_j} x_i \boldsymbol{v}_i \qquad j = 1, \ldots, m \tag{12.37}$$

其中，$\mathcal{F}_j$ 表示第 j 个字段所包含的特征索引集合。在一个字段采用独热编码的情况下，上式相当于选取该字段对应特征的隐向量；对于以数值形式输入的特征，嵌入表示则会随输入值进行相应缩放。由于所有字段的嵌入维度均为 k，不同字段的表示可以按照固定顺序拼接，形成 Deep 组件的输入向量：

$$\boldsymbol{a}^{(0)} = [\boldsymbol{e}_1; \boldsymbol{e}_2; \cdots; \boldsymbol{e}_m] \tag{12.38}$$

随后，拼接后的向量被输入 DNN 网络，通过逐层的非线性变换来学习复杂的特征组合关系。其第 ℓ 层的计算过程如下：

$$\boldsymbol{a}^{(\ell+1)} = \phi\left(\boldsymbol{W}^{(\ell)} \boldsymbol{a}^{(\ell)} + \boldsymbol{b}^{(\ell)}\right), \qquad \ell = 0, \ldots, L-1 \tag{12.39}$$

其中，$\boldsymbol{W}^{(\ell)}$ 和 $\boldsymbol{b}^{(\ell)}$ 分别表示第 ℓ 层的权重矩阵和偏置，$\phi(\cdot)$ 为非线性激活函数，L 为隐藏层数。在经过多层变换之后，Deep 组件会得到高阶特征表示，并通过输出层产生相应的预测 logit：

$$z_{\mathrm{DNN}} = \boldsymbol{w}_{\mathrm{DNN}}^{\top} \boldsymbol{a}^{(L)} + b_{\mathrm{DNN}} \tag{12.40}$$

DeepFM 模型将 FM 组件和 Deep 组件的输出在预测层进行融合，得到最终的点击概率：

$$\hat{y} = \sigma\left(z_{\mathrm{FM}} + z_{\mathrm{DNN}}\right) \tag{12.41}$$

其中，$\sigma(\cdot)$ 为 Sigmoid 函数，将融合后的 logit 映射到 $(0, 1)$ 区间。这里，FM 组件直接提供一阶与二阶特征交互的建模结果，Deep 组件则利用多层非线性变换学习高阶特征关系。两者在输出层进行加和，使不同阶次的特征交互能够共同影响最终的预测结果。

DeepFM 模型的另一个重要设计是，FM 组件和 Deep 组件共享同一套特征嵌入参数 $\boldsymbol{v}_i$。在 FM 组件中，这些参数通过特征隐向量之间的内积参与二阶交互计算。而在 Deep 组件中，它们则作为嵌入层参数，将原始稀疏特征转换为稠密表示。因此，同一个特征的嵌入不仅受到二阶交互学习目标的影响，也会受到深度网络学习到的高阶关系的影响。与先训练 FM、再将其隐向量用于初始化深度网络的 FNN 不同，DeepFM 模型将这两个组件直接纳入同一个计算图，无须额外的 FM 模型预训练阶段。这种共享嵌入的方式使得低阶和高阶特征交互能够在同一预测目标下联合学习，也避免了两个分支分别维护特征表示所带来的额外复杂性。

在训练阶段，DeepFM 模型通常采用二元交叉熵损失进行点击率预估。假设训练样本数为 n，其损失函数可以写为：

$$\mathcal{L} = -\frac{1}{n} \sum_{t=1}^{n} \left[y_t \log \hat{y}_t + (1 - y_t) \log(1 - \hat{y}_t) \right] \tag{12.42}$$

模型通过反向传播计算该损失对各组件参数的梯度，并同步更新 FM 组件的一阶权重、特征隐向量以及 Deep 组

件的网络参数。由于两个组件共享隐向量，来自 FM 组件和 Deep 组件的梯度会共同作用于同一组嵌入参数，从而使特征表示同时适应低阶交互和高阶交互的学习需求。对于转化、点赞等其他二分类反馈，也可以采用相应的标签定义和预测目标使用类似的建模方式。

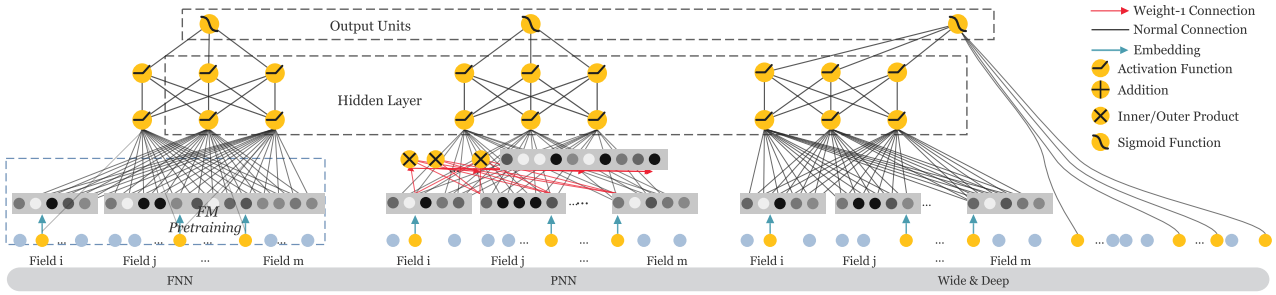


图 12.5: FNN、PNN 和 Wide & Deep 模型的网络结构对比。

为了理解 DeepFM 模型的设计特点，可以将其与同期几种具有代表性的深度 CTR 模型进行对比，如图 12.5 所示。FNN 模型 [24] 通过 FM 组件的预训练来获得特征嵌入，再将其输入深度网络进行高阶特征建模。这种方法利用了 FM 组件学到的特征表示，但需要额外的预训练过程，并且 FNN 模型本身没有像 FM 组件那样显式地建模二阶特征交互。而 PNN 模型 [16] 则在嵌入层之后引入乘积层，通过内积、外积等操作构造特征交互表示，再将其输入后续网络。乘积层能够增强网络对特征交互的建模能力，但不同乘积操作在计算开销和信息保留方面存在差异。Wide & Deep 模型同样通过两个分支分别学习低阶和高阶特征关系，但 Wide 分支依赖人工设计的交叉特征。DeepFM 模型将该分支替换为 FM 组件后，可以通过特征隐向量自动学习二阶交互，并与 Deep 组件共享嵌入参数。因此，相比于上述模型，DeepFM 模型的主要特点如下：

定义 12.4 (DeepFM 模型的主要特点)

DeepFM 模型无需对 FM 组件进行预训练，也无需为 Wide 分支专门构造交叉特征，还能够在统一的端到端训练过程中同时学习一阶、二阶和高阶特征关系。

总体而言，DeepFM 模型将因子分解机的显式低阶交互建模能力与深度神经网络的非线性表达能力结合起来，通过共享嵌入和联合训练，在高维稀疏特征场景下实现了较为统一的特征交互建模。不过，FM 组件显式建模的交互阶数仍然主要是二阶，而 Deep 组件对更高阶交互的学习则主要依赖多层网络的隐式组合。对于希望进一步显式建模高阶特征交互的场景，能否设计一种结构化的交互网络，在控制计算开销的同时直接学习不同阶次的特征组合，仍然是值得研究的问题。基于这一思路，后续的 xDeepFM 模型在 DeepFM 模型的建模框架上引入压缩交互网络（Compressed Interaction Network，CIN），专门用于显式学习高阶特征交互，并与深度网络共同完成预测。

12.4.1.4 xDeepFM 模型

xDeepFM 模型是在 DeepFM 模型基础上进一步改进的深度推荐模型，源自发表于 KDD 2018 会议的论文《xDeepFM: Combining Explicit and Implicit Feature Interactions for Recommender Systems》[11]。DeepFM 模型将因子分解机（FM）与深度神经网络（DNN）结合，能够显式建模二阶特征交互，并通过多层神经网络学习高阶特征关系。不过，FM 组件显式建模的交互阶数主要局限于二阶，而 DNN 对高阶特征关系的学习则依赖多层非线性变换，并没有直接构造不同阶次的特征交互项。因此，如何在保持模型计算和参数规模可控的前提下，进一步增强高阶特征交互的显式建模能力，成为后续模型改进的一个重要方向。

针对这一问题，xDeepFM 模型引入压缩交互网络（Compressed Interaction Network，CIN），在原有深度网络之外增加一个专门用于显式学习高阶特征交互的分支。与 DeepFM 模型主要通过 FM 组件显式建模二阶交互不同，CIN 组件通过逐层组合特征嵌入，使交互阶数随着网络深度增加而递增，从而显式捕捉更高阶的特征组合。同时，xDeepFM 模型还保留 DNN 分支，继续利用其学习隐式的高阶特征关系。由此，xDeepFM 模型整体能够

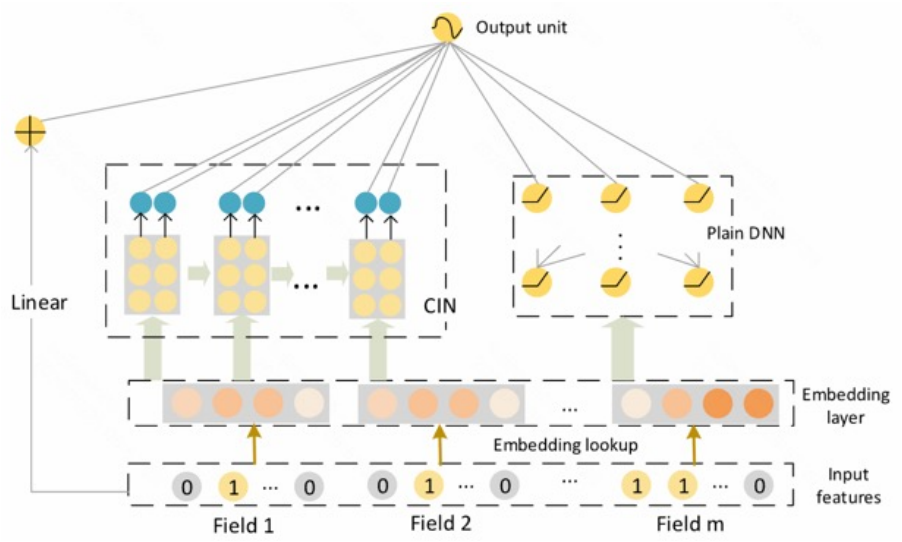


图 12.6: xDeepFM 模型网络结构。

同时利用显式和隐式两种方式建模高阶特征交互。

如图 12.6 所示，xDeepFM 模型主要包含线性组件、CIN 组件和 DNN 组件。线性组件直接利用原始输入特征学习一阶关系；CIN 组件逐层构造显式的高阶特征交互；而 DNN 组件则通过嵌入层和多层感知机学习隐式的非线性关系。三个组件的输出最终在预测层融合，共同完成点击率等用户反馈的预估。

为了说明 CIN 组件如何显式建模高阶特征交互，首先考虑推荐模型中的字段嵌入表示。假设输入样本包含 m 个字段，每个字段经过嵌入层后得到一个 D 维向量 $\boldsymbol{e}_i \in \mathbb{R}^D$。将所有字段的嵌入向量按照字段顺序排列，得到原始特征嵌入矩阵

$$\boldsymbol{X}^{(0)} = \begin{bmatrix} \boldsymbol{e}_1^\top \\ \boldsymbol{e}_2^\top \\ \vdots \\ \boldsymbol{e}_m^\top \end{bmatrix} \in \mathbb{R}^{m \times D} \tag{12.43}$$

其中，$\boldsymbol{X}^{(0)}$ 的每一行对应一个字段的嵌入表示。与直接在原始高维稀疏特征向量上进行交互不同，CIN 组件以字段嵌入向量为基本单位开展计算，从而使不同字段的特征交互能够在统一的低维表示空间中完成。

CIN 组件的核心思想如下：

定义 12.5

在每一层中，将上一层学习到的交互表示与原始字段嵌入再次进行组合。 ♣

从这里可以看出，CIN 组件可以看成是精排模型中一个专门用于显式建模高阶特征交互的模块，属于显式特征交互网络的一种实现方式。其他的显式特征交互网络实现方式我们将会在后续第15章《特征工程》中进行详细讨论。

假设 CIN 组件共包含 T 层，第 k 层的输出表示为

$$\boldsymbol{X}^{(k)} \in \mathbb{R}^{H_k \times D} \tag{12.44}$$

其中，H_k 表示第 k 层包含的特征向量数量，并令 $H_0 = m$。第 k 层中第 h 个输出向量的计算方式为

$$\boldsymbol{X}_{h,:}^{(k)} = \sum_{i=1}^{H_{k-1}} \sum_{j=1}^{m} W_{i,j}^{(k,h)} \left(\boldsymbol{X}_{i,:}^{(k-1)} \circ \boldsymbol{X}_{j,:}^{(0)} \right) \tag{12.45}$$

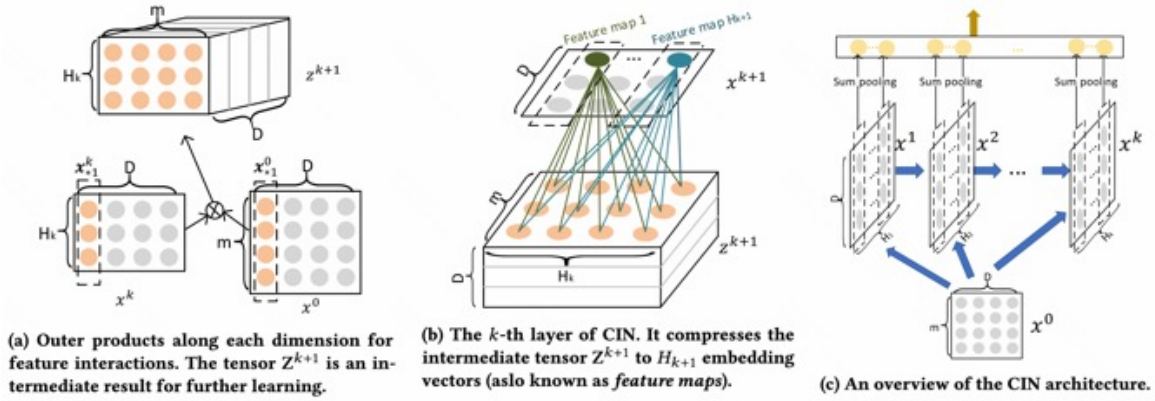
其中，$W^{(k,h)} \in \mathbb{R}^{H_{k-1} \times m}$ 为对应特征向量的可学习参数矩阵，$\circ$ 表示逐元素乘积，也称 Hadamard 积。例如，对于两个三维向量，有

$$[a_1, a_2, a_3] \circ [b_1, b_2, b_3] = [a_1 b_1, a_2 b_2, a_3 b_3] \tag{12.46}$$

从上述计算过程来看，CIN 组件的每个输出向量都由上一层的特征向量与原始字段嵌入两两组合后加权求和得到。第一层以原始嵌入为输入，通过两个字段嵌入之间的逐元素乘积构造二阶交互；第二层将第一层的交互表示再次与原始字段嵌入组合，进一步构造三阶交互；依此类推，随着网络深度增加，模型能够显式建模更高阶的特征关系。

CIN 组件与传统 DNN 组件的另一个区别在于特征交互的计算粒度。DNN 组件通常将多个字段的嵌入向量拼接为一个长向量，再通过全连接层学习不同维度之间的组合关系；CIN 组件则保留每个字段或交互特征向量的结构，并在相同嵌入维度上执行逐元素乘积。因此，CIN 组件在向量层面显式构造特征交互的设计方式使得模型的交互阶数与网络层数建立了更加直接的联系。

图 12.7 展示了 CIN 组件的具体计算结构。从图中可以看出，每一层都接收上一层的交互表示和原始特征嵌入，通过逐元素乘积及可学习的线性组合生成新的特征向量。需要注意 CIN 组件中“压缩”的含义是指上一层的 H_{k-1} 个向量与原始输入中的 m 个向量组合之后，形成了 $H_{k-1}m$ 组潜在交互结果；然后模型再通过可学习的权重，将这些交互结果聚合为 H_k 个输出向量。因此，CIN 组件不需要为每一种潜在的高阶特征组合分别设置独立的输出单元，而是利用有限数量的特征向量逐层提取和组合特征交互信息。



(a) Outer products along each dimension for feature interactions. The tensor Z^{k+1} is an intermediate result for further learning.

(b) The k-th layer of CIN. It compresses the intermediate tensor Z^{k+1} to H_{k+1} embedding vectors (aslo known as *feature maps*).

(c) An overview of the CIN architecture.

图 12.7: xDeepFM 模型中 CIN 模块的网络结构。

在经过 T 层的计算之后，CIN 组件会得到一系列不同交互阶数的特征表示。为了将它们用于最终预测，模型首先对每一层的特征向量沿嵌入维度进行求和池化（Sum Pooling）。对于第 k 层，池化后第 i 个特征向量对应的结果为

$$p_i^{(k)} = \sum_{j=1}^{D} X_{i,j}^{(k)}, \qquad i = 1, \dots, H_k \tag{12.47}$$

由此可以得到第 k 层的池化向量

$$\boldsymbol{p}^{(k)} = [p_1^{(k)}, p_2^{(k)}, \dots, p_{H_k}^{(k)}]^\top \in \mathbb{R}^{H_k} \tag{12.48}$$

然后将所有 CIN 隐藏层的池化结果拼接起来，可以得到最终的交互表示为

$$\boldsymbol{p}^+ = [\boldsymbol{p}^{(1)}; \boldsymbol{p}^{(2)}; \cdots; \boldsymbol{p}^{(T)}] \in \mathbb{R}^{\sum_{k=1}^{T} H_k} \tag{12.49}$$

这种做法保留了不同网络深度对应的交互信息，使得最终的预测不仅能够利用最高阶交互信息，也能够综合利用较低阶的交互表示。

在最终预测阶段，xDeepFM 模型将线性组件、CIN 组件与 DNN 组件的输出进行融合。假设原始输入特征为 $\boldsymbol{x}$，DNN 组件最后一层的表示为 $\boldsymbol{h}^{(L)}$，CIN 组件的池化结果为 $\boldsymbol{p}^+$，则最终预测概率可以写为

$$\hat{y} = \sigma\left(\boldsymbol{w}_{\text{linear}}^\top \boldsymbol{x} + \boldsymbol{w}_{\text{DNN}}^\top \boldsymbol{h}^{(L)} + \boldsymbol{w}_{\text{CIN}}^\top \boldsymbol{p}^+ + b\right) \tag{12.50}$$

其中，$\boldsymbol{w}_{\text{linear}}$、$\boldsymbol{w}_{\text{DNN}}$ 和 $\boldsymbol{w}_{\text{CIN}}$ 分别表示三个组件对应的输出层参数，b 为偏置项，$\sigma(\cdot)$ 为 Sigmoid 函数。三个组件分别从一阶特征、隐式高阶交互和显式高阶交互三个角度提供预测信息，并通过统一的输出层进行融合。

对于点击率预估等二分类任务，xDeepFM 模型通常使用二元交叉熵损失进行训练：

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)] \tag{12.51}$$

其中，N 为训练样本数，y_i 为第 i 条样本的真实标签，$\hat{y}_i$ 为模型的预测概率。在模型训练时，xDeepFM 各个组件的参数通过统一的损失函数进行端到端优化。在需要抑制过拟合时，我们还可以在目标函数中加入参数正则化项。因此，xDeepFM 模型中的线性组件、CIN 组件和 DNN 组件能够围绕同一个预测目标协同学习，而不必分别训练之后再组合到一起。

从模型复杂度的角度来看，CIN 组件通过控制每层输出的特征向量数量 H_k，压缩潜在交互结果的规模。第 k 层的交互参数数量为 $H_k H_{k-1} m$；加上各层连接到最终输出单元的参数，CIN 组件的参数规模可以表示为

$$O\left(\sum_{k=1}^{T} H_k H_{k-1} m + \sum_{k=1}^{T} H_k\right) \tag{12.52}$$

这一参数规模不直接依赖嵌入维度 D。当各层特征向量数量近似为 H 时，CIN 组件的参数规模约为 $O(mTH^2)$。相比之下，普通全连接 DNN 的首层参数量通常会随输入嵌入维度的增加而增大。不过，参数量不依赖 D 并不意味着计算开销也不依赖 D。CIN 组件仍需要在嵌入维度上执行逐元素乘积和加权聚合，其计算成本会受到字段数量、特征向量数量、嵌入维度以及网络深度的共同影响。因此，在工业级推荐系统的落地过程中，如果计算资源不充足且输入特征非常多，CIN 组件巨大的计算开销通常是无法承受的，这也限制了其在大规模场景下的直接应用。因此，在实际应用时，算法工程师仍然需要根据模型效果、训练资源和线上推理延迟来选择合适的网络宽度与深度。

总体而言，xDeepFM 模型延续了 DeepFM 模型将不同类型的特征交互能力整合到同一预测模型中的思路，并通过 CIN 组件进一步增强了显式高阶特征交互的建模能力。DeepFM 模型使用 FM 组件显式建模二阶交互，再由 DNN 组件学习隐式高阶关系；xDeepFM 模型则以线性组件建模一阶特征，并利用 CIN 组件显式建模二阶及更高阶的交互，同时保留 DNN 组件学习隐式高阶关系。两者的区别不仅在于增加了一个网络分支，更在于 xDeepFM 模型将特征交互的阶数与网络结构建立了更直接的联系，从而使得模型能够以结构化的方式学习复杂的特征组合。

然而，xDeepFM 模型的设计重点主要在于如何增强特征交互的表达能力。随着工业推荐系统的规模不断扩大，模型架构还需要兼顾稀疏特征与稠密特征的处理方式、特征交互的计算效率，以及训练和推理过程中的资源开销。这些问题促使研究者进一步探索更加通用且适合大规模工业推荐系统的模型结构。接下来介绍的 DLRM（Deep Learning Recommendation Model）由 Facebook 提出，其采用嵌入层处理稀疏类别特征、底部神经网络处理稠密数值特征，并通过特征交互操作与顶部神经网络完成预测，从另一种架构设计思路出发，兼顾推荐模型的表达能力与工业部署需求。

12.4.1.5 DLRM 模型

DLRM（Deep Learning Recommendation Model）是 Facebook（现 Meta）公司提出的一种通用推荐系统模型架构，源自于 2019 年的论文《Deep Learning Recommendation Model for Personalization and Recommendation Systems》[14]。该模型面向大规模的个性化推荐任务，重点考虑如何统一处理类别特征与连续数值特征，并在有效建模特征交互的同时控制模型的参数规模和计算开销。目前，该模型架构也在工业界大规模推荐系统中得到了一定的应用。

与前面介绍的 DeepFM 和 xDeepFM 模型不同，DLRM 模型并不以显式构造更高阶的特征交互为主要目标，而是采用结构化的特征交互方式，即先将稀疏类别特征映射为低维嵌入向量，再利用多层感知机将稠密数值特征转换为相同维度的表示，随后计算不同特征表示之间的两两内积，最后将交互结果输入顶部神经网络完成预测。这种设计结合了嵌入表示、因子分解机中的二阶交互思想以及深度神经网络的非线性建模能力，在保证特征交互表达能力的同时，避免显式构造规模庞大的高阶交互特征。

DLRM 模型的整体网络结构如图 12.8 所示，可以从特征表示、底部网络、特征交互和顶部网络四个部分理解。首先，模型需要分别处理稀疏类别特征和稠密数值特征。推荐系统中的用户 ID、物品 ID、地域类别等通

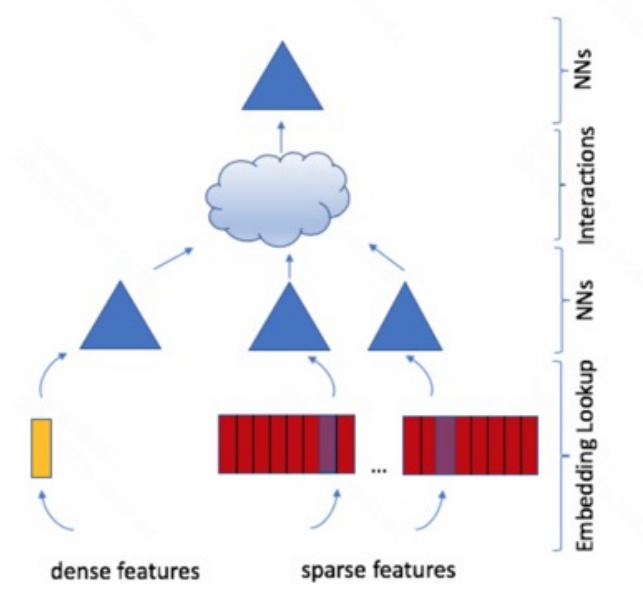


图 12.8: DLRM 模型的网络结构。

常属于类别特征，其取值空间可能非常大，但单条样本实际涉及的类别数量有限。这类特征一般通过嵌入查表（Embedding Lookup）映射为低维稠密向量。假设某个类别特征对应的嵌入表为

$$\boldsymbol{W} \in \mathbb{R}^{M \times d} \tag{12.53}$$

其中 M 为该特征的类别数量，d 为嵌入维度。当输入类别的索引为 i 时，模型从嵌入表中取出对应的向量：

$$\boldsymbol{e}_i = \boldsymbol{W}_{i,:} \tag{12.54}$$

从数学上来看，这相当于使用一个独热向量选择嵌入表中对应的一行。对于包含多个类别值的特征，也可以根据具体实现对多个嵌入向量进行求和或加权聚合。

这种处理方式将原本高维稀疏的类别特征转换为低维稠密表示，使模型能够通过训练学习不同类别在潜在空间中的表示。每个类别对应的嵌入向量可以理解为该类别的可学习特征表示，它既可以独立地参与后续计算，也可以通过与其他特征向量进行交互来表达用户与物品之间的关联。

与类别特征不同，年龄、统计计数及其他连续数值特征通常以稠密数值向量的形式输入模型。DLRM 模型使用底部多层感知机（Bottom MLP，也称 Dense MLP）对这些特征进行变换，将其映射到与类别特征嵌入相同的维度。假设输入的稠密特征为 $\boldsymbol{x}_{\text{dense}}$，则底部网络的输出可以表示为

$$\boldsymbol{e}_{\text{dense}} = f_{\text{bottom}}(\boldsymbol{x}_{\text{dense}}), \qquad \boldsymbol{e}_{\text{dense}} \in \mathbb{R}^d \tag{12.55}$$

其中 $f_{\text{bottom}}(\cdot)$ 表示由多层全连接层和非线性激活函数构成的神经网络。经过这一非线性变换，原始数值特征被编码为能够参与后续特征交互的稠密表示。

在经过上述处理之后，DLRM 模型得到一组维度相同的特征向量，包括各个类别特征对应的嵌入向量，以及由底部 MLP 生成的稠密特征表示。假设共有 m 个类别特征，将这些嵌入向量与稠密特征表示统一记为

$$\boldsymbol{e}_0, \boldsymbol{e}_1, \ldots, \boldsymbol{e}_m \in \mathbb{R}^d \tag{12.56}$$

其中，$\boldsymbol{e}_0$ 表示稠密特征经过底部 MLP 后得到的向量，其余向量表示不同类别特征的嵌入。这样，原本类型和维度各不相同的输入特征便被转换为结构统一的向量表示，为后续特征交互提供了基础。在获得统一的特征表示后，DLRM 模型使用两两内积显式计算特征之间的二阶交互。对于第 i 个和第 j 个特征向量，其交互结果定义为

$$s_{ij} = \langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle = \boldsymbol{e}_i^\top \boldsymbol{e}_j, \qquad 0 \le i < j \le m \tag{12.57}$$

内积将两个 d 维向量压缩为一个标量，用于刻画两种特征表示之间的匹配关系。与逐维拼接两个向量相比，这种方式能够以较低的输出维度概括两个特征之间的交互信息。

DLRM 模型的这一设计与因子分解机（FM）的二阶交互思想相近。FM 模型同样利用特征隐向量之间的内

积来表示二阶特征关系，并通过低维隐向量分解交互参数矩阵，缓解高维稀疏数据中直接学习独立交互参数所带来的问题。不过，DLRM 模型的整体结构并不是简单地在传统 FM 模型的线性预测公式上增加一个神经网络，而是将不同特征的嵌入向量作为交互单元，显式计算它们之间的两两内积，再将这些交互结果交给后续神经网络进一步处理。

对于 $m+1$ 个特征向量，模型需要计算的两两交互数量为

$$N_{\text{interaction}} = \binom{m+1}{2} = \frac{m(m+1)}{2} \tag{12.58}$$

将所有交互结果按照固定顺序排列，可以得到一个一维交互向量：

$$\boldsymbol{s} = [s_{01}, s_{02}, \ldots, s_{m-1,m}] \tag{12.59}$$

随后，DLRM 将该交互向量与原始的稠密特征表示拼接，形成顶部网络的输入：

$$\boldsymbol{z} = [\boldsymbol{e}_{\text{dense}}; \boldsymbol{s}] \tag{12.60}$$

这里保留稠密特征表示十分重要，因为顶部网络不仅能够利用不同特征之间的二阶交互结果，还能够直接访问底部 MLP 提取的稠密特征信息，避免预测过程完全依赖内积交互结果。

需要注意的是，DLRM 模型的特征交互模块并不是传统意义上对所有原始输入特征直接应用一次完整的 FM 预测公式。传统 FM 模型通常将一阶线性项和二阶交互项共同纳入预测函数。而 DLRM 模型则主要通过不同特征向量之间的两两内积构造交互特征，再将其作为顶部神经网络的输入。因此，DLRM 模型的特征交互模块更适合被理解为一种显式的二阶特征交互层。

在完成特征交互后，DLRM 模型的顶部多层感知机（Top MLP，也称 Output MLP）负责进一步组合稠密特征表示与二阶交互结果，学习它们与目标反馈之间的非线性关系。其计算过程可以表示为

$$\boldsymbol{h}^{(0)} = \boldsymbol{z} \tag{12.61}$$

$$\boldsymbol{h}^{(\ell+1)} = \phi\left(\boldsymbol{W}^{(\ell)}\boldsymbol{h}^{(\ell)} + \boldsymbol{b}^{(\ell)}\right), \qquad \ell = 0, \ldots, L-1 \tag{12.62}$$

其中，$\boldsymbol{W}^{(\ell)}$ 和 $\boldsymbol{b}^{(\ell)}$ 分别为对应层的权重矩阵与偏置，$\phi(\cdot)$ 为非线性激活函数。在经过多层变换后，顶部网络将不同二阶交互与稠密特征信息进行组合，最终通过输出层得到预测结果。对于点击率等二分类任务，其预测概率可以表示为

$$\hat{y} = \sigma\left(\boldsymbol{w}_{\text{out}}^{\top}\boldsymbol{h}^{(L)} + b_{\text{out}}\right) \tag{12.63}$$

其中，$\sigma(\cdot)$ 为 Sigmoid 函数，$\boldsymbol{w}_{\text{out}}$ 和 b_{out} 分别为输出层的权重和偏置。与前面几节讲述的 DeepFM 等模型类似，DLRM 模型在训练时通常采用二元交叉熵等适用于二分类任务的损失函数，并通过反向传播联合更新嵌入表、底部 MLP 以及顶部 MLP 的参数。

从特征交互的角度来看，DLRM 模型主要显式建模二阶交互，而非 xDeepFM 模型中 CIN 组件那样逐层构造更高阶的交互表示。虽然顶部 MLP 可以在二阶交互特征的基础上继续学习复杂的非线性组合，但这种能力与 CIN 组件直接构造高阶交互的方式不同。DLRM 模型的设计思路是先以结构化的方式计算特征向量之间的交互，再由神经网络学习这些交互信息与目标反馈之间的关系。

这种结构也体现出 DLRM 模型在模型表达能力与计算效率之间的权衡。假设每个特征向量的维度为 d，直接计算所有两两内积的主要计算量随特征数量和嵌入维度增长，约为 $O(m^2 d)$；而交互结果的数量则为 $O(m^2)$。与将全部嵌入向量拼接后直接输入宽层全连接网络相比，两两内积将每对特征的交互压缩为一个标量，能够控制交互特征的维度，并减少后续网络需要处理的输入规模。不过，当类别特征数量很大时，交互数量仍会随特征数量的平方增长，因此实际应用中仍需要结合特征数量、嵌入维度和延迟要求进行设计。

DLRM 模型的另一个工业特点是，它将稀疏类别特征的嵌入查表与稠密数值特征的神经网络计算明确分开。嵌入表通常包含大量参数，其访问过程具有稀疏性；而底部和顶部 MLP 则主要执行稠密矩阵计算。两类计算在内存访问模式、并行方式和硬件利用率方面存在明显差异。将它们组织为相对独立的模块，有利于根据不同计算特点优化系统实现，也是 DLRM 模型在大规模推荐系统研究中受到关注的重要原因之一。

总体而言，DeepFM 模型、xDeepFM 模型和 DLRM 模型分别体现了不同的特征交互建模思路。DeepFM 模型将 FM 与 DNN 相结合，在统一架构中学习低阶与高阶特征关系；xDeepFM 模型进一步利用 CIN 组件显式构造高阶特征交互；而 DLRM 模型则以统一的特征嵌入为基础，通过两两内积构造显式二阶交互，再交由顶部 MLP 进行非线性组合。与 DeepFM 模型直接融合 FM 和 DNN 的输出不同，DLRM 模型将特征交互结果作为顶部网络的输入。与 xDeepFM 模型侧重显式建模高阶交互不同，DLRM 模型通过控制交互单元的粒度和交互结果的维度，在特征表达能力与计算开销之间取得平衡。这些模型并不存在适用于所有场景的绝对优劣，其实际效果取决于特征体系、训练数据、计算预算以及线上服务约束。对于工业级精排模型而言，模型架构不仅需要具备足够的特征表达能力，还需要兼顾训练效率、推理延迟和资源消耗。

12.4.1.6 DeepMCP 模型

DeepMCP 模型源自于阿里巴巴发表在 ICJAI 2019 会议的论文《Representation Learning-Assisted Click-Through Rate Prediction》[15]。与主要围绕点击率预测目标进行特征交互建模的模型不同，DeepMCP 模型在 CTR 预测任务之外，进一步引入用户与广告之间的匹配关系，以及广告之间的相关性关系，通过多个相关任务的联合训练，学习更具表达能力的特征表示。其核心思想如下：

定义 12.6 (DeepMCP 核心思想)

除了让模型学习哪些特征有助于预测点击行为，DeepMCP 模型还可以利用用户对广告的匹配程度，以及广告在用户行为序列中的关联关系，为特征表示学习提供额外的监督信号。

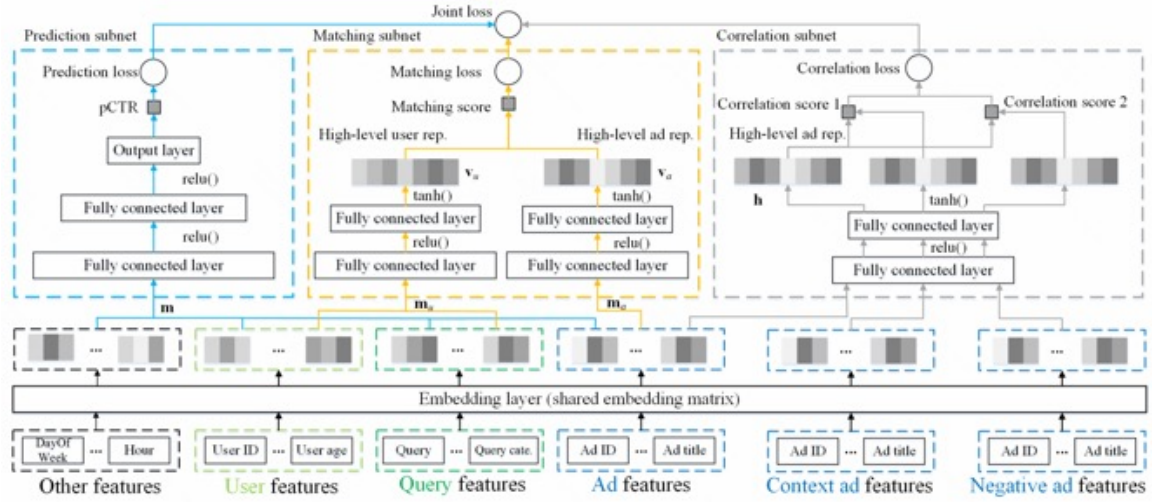


图 12.9: DeepMCP 模型的网络结构。

如图 12.9 所示，DeepMCP 模型由匹配子网络（**M**atching Subnet）、相关性子网络（**C**orrelation Subnet）和预测子网络（**P**rediction Subnet）三部分组成，分别建模用户与广告之间的匹配关系，广告之间的相关性关系，以及特征与点击行为之间的关系。三个子网络共享底层特征嵌入矩阵，并通过联合优化各自的目标函数，使特征表示同时服务于点击率预测和相关关系建模这两类任务。

为了满足不同任务的建模需求，DeepMCP 模型将输入特征划分为四组：用户特征、查询特征、广告特征和其他特征。其中，用户特征包括用户 ID、年龄等信息，查询特征包括查询词、查询类别等信息，广告特征包括广告创意 ID、标题等信息，其他特征则包括小时、星期等上下文信息。三个子网络根据各自的任务选择不同的输入，其中预测子网络使用全部四组特征，匹配子网络使用用户、查询和广告特征，相关性子网络则主要使用广告特征。这种设计使不同任务能够利用与自身目标相关的信息，同时通过共享嵌入参数传递学习信号。

预测子网络承担直接的 CTR 预测任务，它的基本结构与常见的深度 CTR 模型类似。对于输入样本中的每个特征，模型首先通过嵌入层将离散特征映射为低维向量。假设第 i 个特征的嵌入表示为 $\mathbf{e}_i \in \mathbb{R}^K$，其中 K 为嵌入

维度。对于广告标题等包含多个离散元素的多值特征，模型先分别获取各元素的嵌入向量，再通过 Sum Pooling 得到该特征的整体表示。随后，模型将所有特征的嵌入向量拼接为一个长向量，即：

$$\mathbf{m} = \text{Concat}\left(\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n\right) \tag{12.64}$$

其中，n 为输入特征的数量。在此基础上，预测子网络通过多层全连接网络提取高阶非线性特征交互。隐藏层采用 ReLU 激活函数，并在全连接层之后使用 Dropout，以降低过拟合风险。假设最后一层隐藏表示为 $\mathbf{z}$，则点击概率预测结果为

$$\hat{y} = \sigma\left(\mathbf{w}^\top \mathbf{z} + b\right) \tag{12.65}$$

其中，$\mathbf{w}$ 和 b 为输出层的可学习参数，$\sigma(\cdot)$ 为 Sigmoid 函数。对于真实点击标签 $y \in \{0, 1\}$，预测子网络采用二元交叉熵损失：

$$\mathcal{L}_p = -\frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} \left[y \log \hat{y} + (1-y)\log(1-\hat{y})\right] \tag{12.66}$$

其中，$\mathcal{D}$ 表示训练样本集合。该损失直接约束预测概率与真实点击行为之间的差异，是 DeepMCP 模型最终优化的核心目标。需要注意的是，预测子网络并不局限于特定的网络结构，也可以替换为 Wide & Deep、DeepFM 等其他的 CTR 预测模型。

上面这种仅依赖点击标签训练预测子网络的方式，虽然能够使模型学习特征与点击行为之间的关系，但却不一定能够充分刻画用户需求与广告内容之间的匹配程度。因此，DeepMCP 模型引入匹配子网络，进而显式建模用户、查询与广告之间的关联，为用户侧和广告侧的特征表示提供额外的监督信号。

匹配子网络由用户侧和广告侧两个分支组成。用户侧分支接收用户特征与查询特征，将它们的嵌入向量拼接后输入多层全连接网络，得到高层用户表示 $\mathbf{v}_u \in \mathbb{R}^M$。广告侧分支则对广告特征执行类似的处理，得到高层广告表示 $\mathbf{v}_a \in \mathbb{R}^M$。尽管两个分支的原始输入维度可能不同，但经过各自的全连接网络后，它们都会被映射到相同的 M 维表示空间，从而可以直接计算匹配程度。

为了使两个表示能够通过内积反映匹配关系，匹配子网络在最后一层采用 tanh 激活函数。tanh 的输出范围为 $(-1, 1)$，能够保留表示的正负方向。相比之下，ReLU 激活函数会将负值截断为零，可能使表示过于稀疏，从而降低内积作为匹配信号时的区分能力。基于两个高层表示的内积，匹配分数定义为

$$s(u, a) = \sigma\left(\mathbf{v}_u^\top \mathbf{v}_a\right) \tag{12.67}$$

其中，$s(u, a)$ 表示用户及查询与广告之间的匹配程度。DeepMCP 模型采用点式匹配目标，直接复用 CTR 预测任务中的点击标签，而不需要额外构造正负样本对。当用户点击广告时，标签为 1；否则为 0。相应的匹配损失为

$$\mathcal{L}_m = -\frac{1}{|\mathcal{D}|} \sum_{(u,a,y)\in\mathcal{D}} \left[y \log s(u,a) + (1-y)\log(1-s(u,a))\right] \tag{12.68}$$

通过上述这个目标，匹配子网络可以学习用户及查询表示与广告表示之间的关联程度。由于监督信号来自实际点击行为，学到的匹配关系能够与 CTR 预测目标保持一定的一致性，同时为共享的特征嵌入参数提供额外的训练信号。

除了用户与广告之间的匹配关系，广告之间的相关性同样能够为特征表示学习提供有价值的信息。在实际场景中，用户连续点击的广告通常并非彼此独立，而是在主题、内容或用户兴趣等方面存在一定关联。因此，DeepMCP 模型进一步引入相关性子网络，将用户点击过的广告视为具有上下文关系的序列，通过学习广告之间的共现关系，增强广告特征的表示能力。

相关性子网络借鉴了 Word2Vec 中的 Skip-gram 思路。给定用户按时间顺序产生的广告点击序列

$$\mathcal{A} = (a_1, a_2, \ldots, a_L) \tag{12.69}$$

其中 L 为序列长度。模型利用每个广告附近窗口内的其他广告作为上下文，学习当前广告与上下文广告之间的关联。假设上下文窗口大小为 C，则对于序列中的广告 a_i，其上下文由序列中与其距离不超过 C 的其他广告构成。训练目标是提高真实上下文广告与中心广告之间的匹配分数，同时降低随机负样本与中心广告之间的匹配分数。

具体而言，相关性子网络根据广告的特征嵌入及多层全连接网络，得到广告 a_i 的高层表示 $\mathbf{h}_{a_i}$。对于中心

广告 a_i 与真实上下文广告 a_j，模型使用内积计算二者的相关性，并通过 Sigmoid 函数将其映射为正样本得分：

$$p^+(a_j \mid a_i) = \sigma\left(\mathbf{h}_{a_j}^\top \mathbf{h}_{a_i}\right) \tag{12.70}$$

如果直接对所有广告计算归一化概率，训练开销可能随广告数量增长。因此，DeepMCP 模型采用负采样方法，从噪声分布 $P_n(a)$ 中抽取 Q 个负样本。对于中心广告 a_i、真实上下文广告 a_j 及其对应的负样本 $a_1^-, \ldots, a_Q^-$，单个正样本对的训练损失可以写为

$$\ell_c(a_i, a_j) = -\log \sigma\left(\mathbf{h}_{a_j}^\top \mathbf{h}_{a_i}\right) - \sum_{q=1}^{Q} \log \sigma\left(-\mathbf{h}_{a_q^-}^\top \mathbf{h}_{a_i}\right) \tag{12.71}$$

其中，第一项鼓励模型提高真实上下文广告与中心广告之间的相似程度，而第二项则通过负样本抑制无关广告与中心广告之间的相似程度。基于点击序列及上下文窗口构造出正样本集合 $\mathcal{P}$ 之后，相关性子网络的整体损失函数可以表示为

$$\mathcal{L}_c = \frac{1}{|\mathcal{P}|} \sum_{(a_i, a_j) \in \mathcal{P}} \ell_c(a_i, a_j) \tag{12.72}$$

通过这种 Skip-gram 式的训练方式，相关性子网络能够从用户历史点击序列中学习广告之间的关联，并将这些关联信息传递给共享的广告特征嵌入参数。需要注意的是，该子网络并不直接预测目标广告的点击率，也不要求将广告之间的相关性分数作为线上排序分数，而是利用点击序列中的上下文关系改善广告表示，使模型能够从单条 CTR 训练样本之外获得额外的监督信号。

在离线训练阶段，DeepMCP 模型联合优化三个子网络，其整体损失函数为

$$\mathcal{L} = \mathcal{L}_p + \alpha \mathcal{L}_m + \beta \mathcal{L}_c \tag{12.73}$$

其中 $\mathcal{L}_p$、$\mathcal{L}_m$ 和 $\mathcal{L}_c$ 分别表示预测、匹配和相关性损失，α 与 β 为平衡辅助任务与 CTR 预测任务的权重超参数。在模型训练时，三个子网络都会参与前向计算与梯度更新。其中预测任务直接优化点击率估计能力，匹配任务从用户与广告之间的关联出发提供监督，相关性任务则利用点击序列中的上下文关系补充训练信号。由于模型最终服务于 CTR 预测，因此，在实际训练中通常还需要依据独立验证集上的 AUC 等指标来选择模型参数，避免辅助任务过度影响主任务的预测效果。

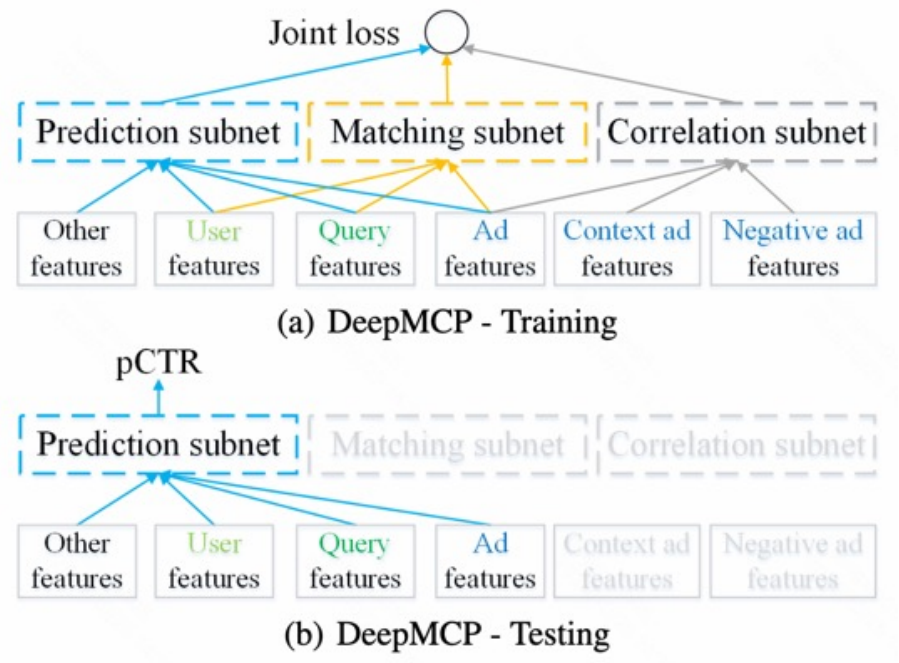


图 12.10: DeepMCP 模型离线训练与线上推理的对比。

此外，DeepMCP 模型还将离线训练结构与线上推理结构相互分离来开，如图12.10所示。在线上推理服务中，DeepMCP 模型只需要执行预测子网络，根据当前请求和候选广告的输入特征计算点击概率，而不需要运行匹配

子网络或相关性子网络。因此，尽管模型在离线训练阶段引入了额外的辅助训练任务与计算开销，但模型的线上推理仍然保持常规深度 CTR 预估模型的完整计算流程，也无须为每个候选广告额外执行匹配打分或广告序列相关性的计算，因此并不会增加模型线上推理时的算力消耗。

最后，我们对 DeepMCP 模型这种重要的精排模型架构进行讨论与总结。

笔记 从模型设计角度来看，DeepFM 模型和 xDeepFM 模型主要通过调整特征交互结构，增强模型对低阶和高阶特征关系的表达能力；DLRM 模型则通过显式构造二阶特征交互，再交由顶部网络进行非线性组合。DeepMCP 模型则采用了完全不同的思路。它保留了以深度 DNN 为核心的 CTR 预测网络，同时引入匹配子网络和相关性子网络，从用户与物品之间的匹配关系、物品之间的关联关系等角度，为主网络的特征表示学习提供额外的监督信号。其中，匹配子网络吸收了召回和粗排阶段双塔模型的设计思想，将用户及查询特征与广告特征分别编码到同一表示空间，从而学习用户到物品（User-to-Item，U2I）的匹配关系。相关性子网络则借助用户历史点击序列中的上下文关系，学习物品到物品（Item-to-Item，I2I）的关联。由此，DeepMCP 模型将传统双塔模型的匹配建模能力与精排 DNN 模型的综合特征建模能力结合起来，并在整体设计思路上与第10章《召回模型》中讲到的 PDN 模型具有一定的相似之处。因此，DeepMCP 模型所体现的关键思路，是不再要求单一的预测网络来独立学习所有有用的信息，而是将不同的关系交给更适合的网络建模，再通过联合训练的方式将这些信息传递给最终承担 CTR 预测任务的主网络。

这种“主网络 + 辅助网络”的架构，进一步体现了模型训练目标与线上推理路径相分离的设计思想。对于工业级精排系统而言，匹配和物品相关性等任务可以在离线训练阶段提供额外的监督信号，却不一定需要在在线服务阶段继续执行。DeepMCP 模型在训练时联合优化三个子网络，而在线推理时只保留了 CTR 预测子网络，因此能够利用辅助任务改善特征表示，同时避免将辅助网络的计算开销直接带入高并发、低延迟的精排链路。这种训练与推理解耦的方式，使得模型设计不必局限于线上服务能够直接承载的计算结构。只要额外的训练任务有助于学习更有效的底层表示，就可以尝试将其引入到训练过程中，而不必要求所有任务都成为线上推断流程的一部分。类似的思路并不局限于原始论文中的广告场景。在短视频、直播等工业级推荐业务中，也广泛存在着其他实践落地成果，通过辅助网络的方式来建模用户兴趣、物品关系以及其他行为信号，从而改善主排序模型效果。在这些场景中，辅助网络往往不直接决定线上排序结果，却能够通过训练阶段的联合优化，为主网络提供有价值的表示学习信号。

从更长的技术演进过程来看，这类**“主网络 + 辅助网络”架构**的重要性不只在于增加了几个网络分支，而在于形成了一种适合工业级推荐系统的模型迭代范式。即线上推理仍然由相对清晰、计算可控的主网络来完成预测，而离线训练则可以围绕主任务持续引入新的辅助目标，探索不同类型的监督信号与关系结构。所以算法工程师不必每次都彻底替换主干模型，也可以通过增加辅助任务、调整联合训练目标和优化参数共享方式，来逐步提升精排模型的效果。这种思路贯穿了深度学习在工业级推荐系统中的长期发展，并在推荐大模型兴起之前，成为精排模型设计与迭代中一种具有代表性的架构范式。当然，辅助网络的数量并非越多越好。只有当辅助任务提供的信息与目标预测任务具有足够的关联性，并且联合优化不会引入过强的噪声信号或者任务冲突时，这些额外的训练信号才能够转化为稳定的线上 A/B 指标收益。因此，这类模型设计的核心并不是简单地叠加网络，而是围绕主任务选择有价值的辅助关系，并通过合理的联合训练与部署解耦，在预测效果、训练成本和线上服务效率之间取得某种平衡。

12.4.1.7 DFN 模型

DFN（Deep Feedback Network）模型源自于腾讯科技发表于 IJCAI 2020 学术会议上的论文《Deep Feedback Network for Recommendation》[22]。传统的 CTR 预估模型通常以用户点击行为为主要监督信号，通过学习特征与点击结果之间的关系来估计用户对候选物品的点击概率。然而，点击行为只能反映用户反馈的一部分信息，用户没有点击某个物品，并不一定意味着不喜欢该物品。与此同时，用户主动点击不喜欢按钮等显式负反馈虽然能够直接反映用户态度，但在实际推荐系统中往往比较稀少。因此，我们会有如下疑问：

笔记 如何充分利用不同类型的反馈信号，既保留高质量行为中的明确偏好，又从大量存在噪声的行为中提取有价值的信息呢？这正是工业推荐系统精排模型需要解决的问题。

针对上述用户反馈信号利用不充分的问题，DFN 模型的核心思路如下：

定义 12.7

DFN 模型联合利用多种正负反馈，通过反馈之间的交互学习更加全面的用户兴趣表示，并将其融入 CTR 预估网络。与仅通过点击标签训练单一预测网络的方式不同，DFN 模型不仅考虑不同历史行为与当前目标物品之间的关系，还利用相对可靠的反馈信号辅助识别其他行为中潜在的正向或负向兴趣。

整体上，DFN 模型由深度反馈交互模块（Deep Feedback Interaction Module）和特征交互模块（Feature Interaction Module）组成。前者从不同类型的历史行为中提取与目标物品相关的偏好表示，后者再将这些表示与用户画像、物品属性及推荐场景等特征进行融合，最终完成点击概率的预测。DFN 模型中的深度反馈交互模块可以认为是一种序列建模的模块，用于捕捉用户历史行为序列中与目标物品相关的潜在兴趣模式，从而为后续的特征交互提供更丰富的表示。其他更多关于用户序列建模的方法我们将会在后续第20章《用户兴趣与序列建模》中进行详细介绍。

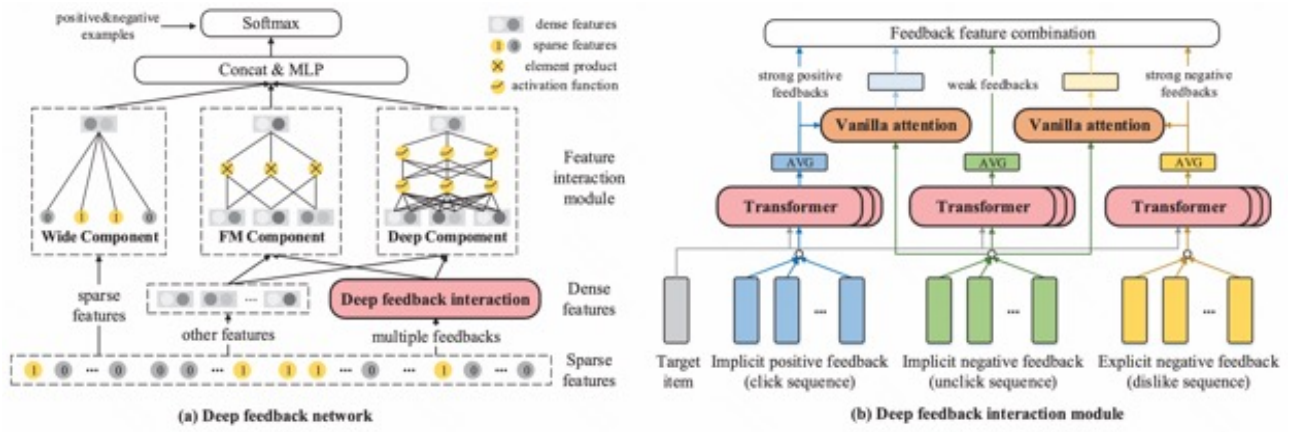


图 12.11: DFN 模型的网络结构。

DFN 模型的网络结构如图 12.11所示。DFN 模型首先将用户的历史行为划分为三类。第一类是隐式正反馈，即用户点击过的物品。点击（click）行为数据规模较大，通常来说也比较容易获取，是传统 CTR 预估模型最常使用的监督信号。第二类是显式负反馈，即用户主动对某个物品执行不喜欢（hate）等操作。这类反馈能够比较直接地表达用户的负向态度，但由于需要用户主动操作，其数量通常远少于点击行为。第三类是隐式负反馈，即已经曝光给用户、但最终没有被点击的物品。这类行为在推荐系统中非常常见，能够提供大量有关用户选择的信息，但它的语义并不是特别明确。用户可能确实对该物品不感兴趣，也可能只是没有注意到该物品，或者是因为其他候选物品更具吸引力而没有点击等。直接将所有未点击行为全部当成确定的负反馈偏好，很容易引入大量错误的监督信号。

因此，DFN 模型并不是简单地将三类反馈拼接之后交给统一的 DNN 组件，而是通过两层反馈交互逐步提取偏好信息。首先，模型分别在点击、显式不喜欢和曝光未点击等行为序列内部，学习历史行为与当前目标物品之间的细粒度关系。随后，模型再利用点击和显式负反馈中相对可靠的信息，对曝光未点击行为进行进一步筛选，分别提取其中可能蕴含的正向和负向偏好。最终得到的反馈表示与其他输入特征共同参与 CTR 预估。

具体而言，对于一个待预测的目标物品 t，假设用户的点击序列、显式不喜欢序列和曝光未点击序列分别为

$$\begin{aligned} \mathcal{C} &= \{c_1, c_2, \ldots, c_{n_c}\} \\ \mathcal{D} &= \{d_1, d_2, \ldots, d_{n_d}\} \\ \mathcal{U} &= \{u_1, u_2, \ldots, u_{n_u}\} \end{aligned} \tag{12.74}$$

其中，n_c、n_d 和 n_u 分别表示三类历史行为的序列长度。为了让模型能够针对当前待预测物品识别出用户的相关兴趣，DFN 模型通常会将目标物品与对应类型的历史行为序列共同输入反馈交互模块。在每类行为序列内部，DFN 模型首先通过自注意力机制学习目标物品与历史行为之间的关系。每个行为的输入表示由物品嵌入和位置

嵌入共同构成，以便同时保留物品内容信息和其在行为序列中的位置信息。以点击序列为例，将目标物品 t 与点击行为序列组合，得到

$$\mathbf{B}_c = [\mathbf{e}_t + \mathbf{p}_0, \mathbf{e}_{c_1} + \mathbf{p}_1, \ldots, \mathbf{e}_{c_{n_c}} + \mathbf{p}_{n_c}] \tag{12.75}$$

其中，$\mathbf{e}_t$ 和 $\mathbf{e}_{c_i}$ 分别表示目标物品与历史点击物品的嵌入向量，$\mathbf{p}_i$ 表示相应的行为位置嵌入。因此，目标物品也成为注意力计算中的一个输入位置，DFN 模型可以根据它与不同历史行为之间的关系，学习当前预估任务所需要的偏好信息。

自注意力机制首先将输入表示映射为查询、键和值矩阵：

$$\mathbf{Q} = \mathbf{B}_c\mathbf{W}_Q, \qquad \mathbf{K} = \mathbf{B}_c\mathbf{W}_K, \qquad \mathbf{V} = \mathbf{B}_c\mathbf{W}_V \tag{12.76}$$

其中 $\mathbf{W}_Q$、$\mathbf{W}_K$ 和 $\mathbf{W}_V$ 为可学习的投影矩阵。随后，DFN 模型根据查询与键之间的相似度计算注意力权重，并对值向量进行加权聚合，即：

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V} \tag{12.77}$$

其中，d_k 为键向量的维度，缩放因子 $\sqrt{d_k}$ 用于控制内积的数值范围，避免注意力分布过于集中。

为了在不同表示子空间中捕获行为之间的关联，DFN 模型进一步使用多头自注意力。假设注意力头数为 H，则第 h 个注意力头的输出为

$$\text{head}_h = \text{Attention}\left(\mathbf{B}_c\mathbf{W}_Q^{(h)}, \mathbf{B}_c\mathbf{W}_K^{(h)}, \mathbf{B}_c\mathbf{W}_V^{(h)}\right) \tag{12.78}$$

其中，$\mathbf{W}_Q^{(h)}$、$\mathbf{W}_K^{(h)}$ 和 $\mathbf{W}_V^{(h)}$ 分别表示第 h 个注意力头对应的投影参数。将各个注意力头的输出拼接后，再经过输出投影矩阵，可以得到行为交互的表示如下：

$$\mathbf{F}_c = \text{Concat}\left(\text{head}_1, \ldots, \text{head}_H\right)\mathbf{W}_O \tag{12.79}$$

其中，$\mathbf{W}_O$ 为输出投影矩阵。由于后续特征交互模块需要使用固定维度的反馈表示，DFN 模型对 $\mathbf{F}_c$ 中的所有位置向量进行平均池化（Average Pooling），进而得到点击反馈的表示：

$$\mathbf{f}_c = \text{AveragePooling}(\mathbf{F}_c) = \frac{1}{n_c + 1}\sum_{i=0}^{n_c}\mathbf{F}_{c,i} \tag{12.80}$$

该表示综合考虑了目标物品和历史点击行为之间的关系，能够反映用户在当前预估任务下与点击行为相关的正向兴趣。

对于显式不喜欢序列和曝光未点击序列，模型采用相同的基本思路，分别构造包含目标物品的行为输入，并通过对应的多头自注意力网络提取反馈表示 $\mathbf{f}_d$ 和 $\mathbf{f}_u$。其中，$\mathbf{f}_d$ 反映与目标物品相关的显式负向偏好，$\mathbf{f}_u$ 则反映曝光未点击行为中的整体关联信息。为了适应不同反馈类型的特征分布，三类反馈分别使用对应的网络参数。至此，DFN 模型已经通过内部反馈交互，在每类行为序列中提取了目标物品相关的反馈表示，但曝光未点击行为中仍然可能混杂着用户真正不喜欢的物品，以及用户实际上感兴趣、却因其他原因没有点击的物品。

这正是 DFN 模型进一步引入外部反馈交互的原因。对于曝光未点击行为，DFN 模型不再将其统一理解为负向信号，而是利用更明确的点击反馈和显式不喜欢反馈，分别提取其中潜在的正向与负向偏好。具体来说，假设 $\mathbf{u}_i$ 为未点击序列中第 i 个行为的表示。模型首先利用显式负反馈表示 $\mathbf{f}_d$，判断每个未点击行为与已知负向兴趣之间的关联程度。为了计算二者的匹配分数，模型将反馈表示与行为表示、二者的差以及逐元素乘积进行拼接，并输入一个两层 MLP：

$$g(\mathbf{a}, \mathbf{b}) = \text{MLP}\left(\text{Concat}\left[\mathbf{a}, \mathbf{b}, \mathbf{a} - \mathbf{b}, \mathbf{a} \odot \mathbf{b}\right]\right) \tag{12.81}$$

其中，$\odot$ 表示逐元素乘法，$g(\mathbf{a}, \mathbf{b})$ 用于衡量两种表示之间的关联程度。基于上述匹配函数，DFN 模型会计算每个未点击行为的负向注意力权重：

$$\alpha_i = \frac{\exp(g(\mathbf{f}_d, \mathbf{u}_i))}{\sum_{j=1}^{n_u}\exp(g(\mathbf{f}_d, \mathbf{u}_j))} \tag{12.82}$$

并通过加权求和得到结合显式负反馈的未点击行为表示：

$$\mathbf{f}_{ud} = \sum_{i=1}^{n_u} \alpha_i \mathbf{u}_i \tag{12.83}$$

在这一过程中，显式不喜欢反馈为注意力计算提供了参照，使模型能够从大量未点击行为中提取与用户负向兴趣更相关的部分。

类似地，DFN 模型再利用点击反馈表示 $\mathbf{f}_c$，提取未点击序列中可能蕴含的正向兴趣。其注意力权重定义为

$$\beta_i = \frac{\exp(g(\mathbf{f}_c, \mathbf{u}_i))}{\sum_{j=1}^{n_u} \exp(g(\mathbf{f}_c, \mathbf{u}_j))} \tag{12.84}$$

相应的正向偏好表示为

$$\mathbf{f}_{uc} = \sum_{i=1}^{n_u} \beta_i \mathbf{u}_i \tag{12.85}$$

通过这一步处理，原本统一的未点击行为被进一步分解为两种不同的兴趣线索：$\mathbf{f}_{uc}$ 强调与已知正向反馈相关的行为，$\mathbf{f}_{ud}$ 则强调与已知负向反馈相关的行为。这里的区分并不意味着模型能够获得每个未点击行为的真实偏好标签，而是通过不同反馈之间的关联，为含噪的隐式反馈提供更细粒度的表示。

最终，DFN 模型将内部反馈交互和外部反馈交互得到的五种表示组合起来，构成精炼后的反馈特征：

$$\mathbf{f}_{\text{Feed}} = \text{Concat}\left(\mathbf{f}_c, \mathbf{f}_d, \mathbf{f}_u, \mathbf{f}_{uc}, \mathbf{f}_{ud}\right) \tag{12.86}$$

其中，$\mathbf{f}_c$ 和 $\mathbf{f}_d$ 分别来自点击行为与显式不喜欢行为，是相对直接的正向和负向反馈表示；而 $\mathbf{f}_u$、$\mathbf{f}_{uc}$ 和 $\mathbf{f}_{ud}$ 则与曝光未点击行为有关，分别保留其整体信息以及经正向、负向反馈引导后提取的偏好信息。这种结构的关键并不是简单增加行为序列的数量，而是让不同质量、不同语义的反馈信号相互补充，从而尽可能减少单独依赖点击或未点击标签所带来的偏差。

在得到精炼的反馈特征后，DFN 模型还需要将其与用户画像、目标物品属性及推荐场景等信息进行融合。因此，DFN 模型构建了由 Wide、FM 和 Deep 三个组件组成的特征交互模块，分别从显式线性关系、二阶特征交互和高阶非线性关系三个角度学习组合特征。该设计与 Wide & Deep、DeepFM 等经典 CTR 预估模型的思路相似，但 DFN 模型将前面提取的多种反馈表示也纳入了特征交互过程，使得用户行为中的正负偏好能够与常规特征共同影响最终预测结果。

假设原始输入特征被划分为 m 个字段，分别记为

$$\mathcal{X} = \{x_1, x_2, \ldots, x_m\} \tag{12.87}$$

这些字段包括用户属性、物品属性、推荐上下文等信息。经过相应的特征编码与嵌入查表之后，每个字段会得到一个稠密向量 $\mathbf{f}_i$。将原始特征表示与五种反馈表示组合，则可以构造出 FM 组件的输入：

$$\mathcal{F}' = \{\mathbf{f}_1, \ldots, \mathbf{f}_m, \mathbf{f}_c, \mathbf{f}_d, \mathbf{f}_u, \mathbf{f}_{uc}, \mathbf{f}_{ud}\} \tag{12.88}$$

因此，FM 组件一共接收 $m+5$ 个特征向量，可以进一步学习常规输入特征与不同反馈表示之间，以及不同反馈表示彼此之间的二阶交互。

Wide 组件主要用于建模各个特征字段与预测目标之间的直接关系。对于第 i 个字段，其线性输出为

$$\mathbf{y}_{\text{Wide},i} = \mathbf{w}_i^\top x_i + b_i \tag{12.89}$$

其中 x_i 表示该字段经过编码后的输入，$\mathbf{w}_i$ 和 b_i 分别为对应的线性权重与偏置。将所有字段的线性输出组合起来，可得到

$$\mathbf{y}_{\text{Wide}} = \text{Concat}\left(\mathbf{y}_{\text{Wide},1}, \ldots, \mathbf{y}_{\text{Wide},m}\right) \tag{12.90}$$

Wide 组件能够保留特征与预测结果之间的直接关联，为模型提供相对简单、可学习的线性表达路径。

在此基础上，FM 组件进一步显式建模不同特征表示之间的二阶交互。DFN 模型使用 Bi-interaction 形式，将所有特征向量两两进行逐元素乘法，再对交互结果求和：

$$\mathbf{y}_{\text{FM}} = \sum_{1 \le i < j \le m+5} \mathbf{f}'_i \odot \mathbf{f}'_j \tag{12.91}$$

其中，$\mathbf{f}'_i, \mathbf{f}'_j \in \mathbb{R}^K$ 表示 $\mathcal{F}'$ 中的特征向量，K 为嵌入维度。与逐一计算全部特征对并将结果分别保留的方式相比，这种结构将各个二阶交互结果压缩到统一的向量中，从而控制交互表示的维度。由于输入集合中包含精炼后的反馈特征，FM 组件不仅可以学习用户属性与物品属性之间的交互，还可以学习历史反馈与当前候选物品之间的二阶关联。

为了进一步捕获高阶非线性关系，DFN 模型还引入了 Deep 组件。DFN 模型将原始特征嵌入和反馈特征拼接为初始输入，即：

$$\mathbf{f}^{(0)} = \text{Concat}\left(\mathbf{f}_1, \ldots, \mathbf{f}_m, \mathbf{f}_{\text{Feed}}\right) \tag{12.92}$$

随后使用两层 MLP 进行非线性特征变换：

$$\mathbf{f}^{(l+1)} = \text{ReLU}\left(\mathbf{W}^{(l)}\mathbf{f}^{(l)} + \mathbf{b}^{(l)}\right), \quad l = 0, 1 \tag{12.93}$$

其中，$\mathbf{W}^{(l)}$ 和 $\mathbf{b}^{(l)}$ 分别表示第 l 层的权重矩阵与偏置，最终得到的高阶特征表示为

$$\mathbf{y}_{\text{Deep}} = \mathbf{f}^{(2)} \tag{12.94}$$

Deep 组件可以对多个字段及反馈表示进行多层非线性组合，弥补线性模型和显式二阶交互在复杂特征关系建模方面的不足。

最后，DFN 模型将三个组件的输出拼接，得到用于预测的聚合特征表示：

$$\mathbf{y} = \text{Concat}\left(\mathbf{y}_{\text{Wide}}, \mathbf{y}_{\text{FM}}, \mathbf{y}_{\text{Deep}}\right) \tag{12.95}$$

与只把原始特征交给 Wide、FM 和 Deep 组件的传统架构相比，DFN 模型通过在特征交互之前引入深度反馈建模，使得不同类型的用户反馈以结构化的表示进入后续预测网络。这一设计将反馈关系建模与常规特征交互结合起来，使最终的 CTR 预估既能利用用户、物品和上下文特征，也能利用历史行为中提炼出的正负偏好信息。

在优化阶段，DFN 模型根据样本的实际反馈类型构建联合损失函数。假设点击样本集合、曝光未点击样本集合和显式不喜欢样本集合分别为 $\mathcal{S}_c$、$\mathcal{S}_u$ 和 $\mathcal{S}_d$，训练样本总数为 N。模型首先根据聚合特征计算点击概率：

$$p(x) = \sigma\left(\mathbf{w}_p^\top \mathbf{y}\right) \tag{12.96}$$

其中，$\mathbf{w}_p$ 为输出层权重，$\sigma(\cdot)$ 为 Sigmoid 函数。对于点击样本，模型希望提高预测概率。对于曝光未点击和显式不喜欢样本，则在最终预测目标中将其作为负样本处理。由此得到的联合损失如下：

$$\mathcal{L} = -\frac{1}{N}\left(\lambda_c \sum_{x \in \mathcal{S}_c} \log p(x) + \lambda_u \sum_{x \in \mathcal{S}_u} \log(1 - p(x)) + \lambda_d \sum_{x \in \mathcal{S}_d} \log(1 - p(x))\right) \tag{12.97}$$

其中，λ_c、λ_u 和 λ_d 分别用于调节三类反馈损失的权重。不同类型反馈的样本规模、可靠性和训练目标并不相同，因此需要合理设置损失权重，以避免某类反馈由于数量过多而主导优化过程。

这里需要区分两个层面的设计。对于最终的 CTR 预测目标，未点击行为仍然作为负样本参与损失计算。而在反馈表示学习过程中，DFN 模型并不将所有未点击行为都视为同等可靠的负向证据，而是通过点击和显式不喜欢反馈的引导，从中分别提取潜在的正向与负向偏好。前者负责定义最终预测任务，后者则通过改进输入表示，帮助模型更充分地利用含噪的隐式反馈。DFN 模型通过联合优化最终损失，使得反馈交互模块与特征交互模块能够共同学习服务于点击率预测的特征表示。

从整体设计来看，DFN 模型将多种反馈关系建模与传统深度 CTR 预估架构结合起来。Wide 组件保留显式的线性特征关系，FM 组件负责二阶特征交互，Deep 组件则学习更高阶的非线性组合。在此基础上，反馈交互模块进一步处理不同类型历史行为之间的关系。其中，内部交互利用目标物品与同类历史行为之间的关联提取反馈表示，外部交互则利用相对可靠的正负反馈，对数量庞大但含噪的未点击行为进行进一步建模。与只以点击行为为主要监督信号的模型相比，这种架构能够从更丰富的行为证据中学习用户偏好，并将其转化为主预测网络可以利用的特征。

除此之外，DFN 模型也体现了工业级精排模型设计中的一个重要方向：在保持 CTR 预测目标不变的基础上，通过专门的反馈建模模块来改善主网络的输入表示，而不只是不断增加 DNN 的层数或调整特征交互结构。不过，与 DeepMCP 模型中部分辅助网络主要在离线训练阶段发挥作用、线上仅保留预测子网络的方式不同，DFN 模型产生的精炼反馈特征会直接参与最终 CTR 预估。因此，在线推断时仍需要处理相应的历史行为序列，并执

行反馈交互与特征融合计算。这意味着其收益不仅取决于反馈表示的质量，也需要结合行为序列的长度、注意力计算开销、线上延迟预算等工程约束进行权衡。对于工业级推荐系统而言，真正有效的架构设计并不是无条件地引入更多行为或更复杂的交互模块，而是根据反馈的可靠性和预测任务的需要，建立合理的行为建模与特征融合机制，并在预估精度、计算成本和线上服务效率之间取得平衡。

12.4.2 多任务学习的模型架构

从前一节介绍的 DLRM、DeepMCP 和 DFN 模型可以看出，工业级精排模型的架构设计可以从不同角度展开。DLRM 模型通过显式建模特征之间的二阶交互，并结合顶部 MLP 学习高阶非线性关系，体现了**低阶交叉与高阶交叉相结合的建模思路**。DeepMCP 模型通过引入匹配子网络和相关性子网络，利用辅助任务改善主预测网络的特征表示，体现了**主网络与辅助网络相结合的架构思路**。DFN 模型则进一步对不同类型的历史反馈进行交互建模，并将提取出的偏好表示与常规特征融合，体现了**序列建模与特征交互相结合的架构思路**。上述这几类设计通常并不是相互排斥的。在实际工业级推荐系统中，一个精排模型既可以使用结构化的特征交互模块，也可以引入辅助网络改善表示学习，还可以结合用户行为序列提取兴趣信息，从而形成更加完整的模型架构。

除了特征交互、辅助任务以及行为序列建模之外，精排模型还需要考虑不同预测目标之间的关系。在短视频推荐中，用户的行为反馈可能包括点击、有效播放、长播、完播、点赞、评论、分享和关注等多种类型，同时还存在曝光后未点击、快速滑过等隐式负反馈。这些反馈在数据规模、获取成本、可靠性和业务价值等方面存在明显差异，也分别反映了用户兴趣的不同侧面。例如，点击主要反映用户是否愿意进一步消费某个物品，有效播放和观看时长反映用户的内容消费情况，而点赞、评论和分享等行为则体现了不同程度的主动交互意愿。仅使用其中一种行为作为训练目标，可能难以全面刻画推荐结果的质量。

针对这一问题，工业推荐系统通常会同时建模多个相关的预测目标，并通过共享部分网络参数，在不同任务之间传递有价值的信息。例如，一个精排模型可以同时预估点击、有效播放、长播和点赞等目标，再根据具体的业务策略使用相应的预测结果。这种通过共享模型结构、联合优化多个相关预测目标来学习特征表示的方式，就是多任务学习（Multi-Task Learning）。需要注意的是，多种行为反馈的存在并不意味着必须采用多任务学习：当这些反馈被构造成独立的预测目标，并在同一模型中联合训练时，多任务学习架构才真正发挥作用。

多任务学习的关键在于如何处理不同任务之间的共享与差异。一方面，相关任务往往具有共同的输入特征和部分一致的兴趣信号，通过共享参数可以提高数据利用效率，并使数据较少的任务从其他任务中受益。另一方面，不同任务的优化目标并不完全一致，甚至可能存在一定冲突。如果所有任务被迫使用完全相同的中间表示，某个任务所需要的特征可能受到其他任务的干扰；反过来，如果每个任务完全独立建模，又会失去任务之间共享信息的机会。因此，多任务学习架构的核心问题，是如何在共享信息与保留任务特异性之间取得平衡。

本节介绍几种经典的多任务学习模型架构，包括 MMoE 和 PLE 模型。它们分别通过任务独立门控、共享与任务专属专家的组合，以及分层渐进式参数分离等方式，探索不同任务之间更灵活的参数共享机制。至于多任务学习的训练目标、梯度冲突、任务权重以及多场景建模等更广泛的问题，将在后续第 21 章《多目标与多场景建模》中进一步讨论。

12.4.2.1 MMoE 模型

MMoE（Multi-gate Mixture-of-Experts）模型源自于谷歌发表在 KDD 2018 会议的论文《Modeling Task Relationships in Multi-task Learning with Multi-gate Mixture-of-Experts》[13]。MMoE 模型针对多任务学习中的参数共享问题，在共享专家网络的基础上，为每个任务设置独立的门控网络，使得不同任务能够根据输入特征，自适应地组合不同专家的输出。与所有任务固定共享同一个底层网络的方式相比，这种设计可以更灵活地表达任务之间的相关关系，并在控制额外参数规模的同时，为不同任务保留一定的建模差异。

如图12.12所示，MMoE 模型建立在混合专家模型（Mixture-of-Experts，MoE）的基础之上。MoE 的基本思想是使用多个专家网络分别处理输入，再通过门控网络生成不同专家的组合权重，根据输入内容动态融合专家输出。假设输入特征为 $\mathbf{x}$，模型包含 n 个专家网络，分别记为 $f_1, \ldots, f_n$，门控网络输出的权重向量为 $\mathbf{g}(\mathbf{x})$，则

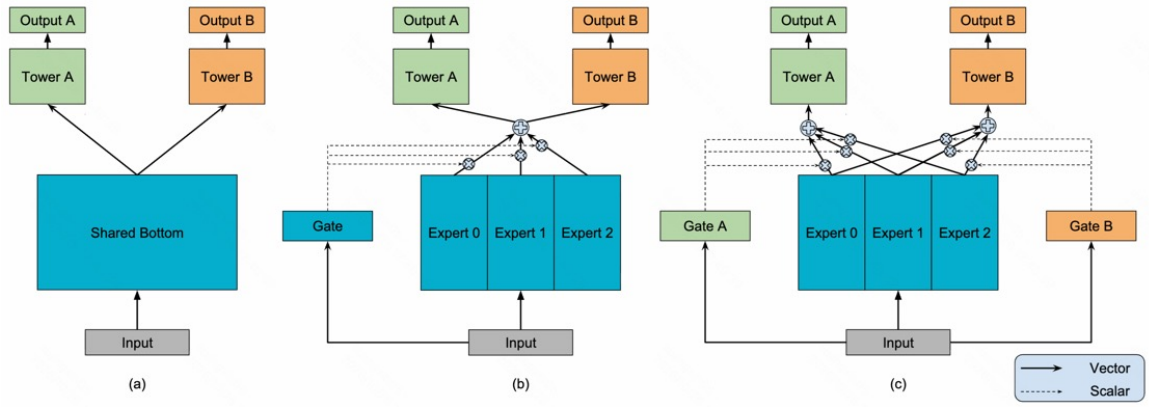


图 12.12: MMoE 模型的网络结构。

MoE 的输出可以表示为

$$\mathbf{f}_{\text{MoE}}(\mathbf{x}) = \sum_{i=1}^{n} g_i(\mathbf{x}) f_i(\mathbf{x}) \tag{12.98}$$

其中，$g_i(\mathbf{x})$ 表示第 i 个专家对应的融合权重，并满足

$$\sum_{i=1}^{n} g_i(\mathbf{x}) = 1, \qquad g_i(\mathbf{x}) \geq 0 \tag{12.99}$$

通常来说，门控网络会通过 Softmax 函数将各个专家的得分转换为归一化的权重。由于专家网络学习不同的特征变换，而门控网络决定这些变换如何组合，模型便能够根据输入样本的不同特点，动态调整各个专家的贡献。

在深度网络中，MoE 模型还可以作为一个可训练的网络层，接收上一层的特征表示，并将融合后的输出传递给后续网络。在这种结构下，专家网络、门控网络和后续预测层可以通过端到端的训练来进行共同优化。不过，传统 MoE 模型本身并不能直接解决多任务学习中的参数共享问题，即：

笔记 如果多个任务共用同一组专家及同一个门控网络，那么它们仍然需要使用相同的专家组合方式，难以充分适应任务之间的差异。

为了解决这一问题，MMoE 模型在共享专家网络的基础上，为每个任务分别设计独立的门控网络。假设模型需要同时学习 T 个任务，包含 n 个共享专家，则第 k 个任务的门控网络根据输入特征生成对应的专家权重为：

$$\mathbf{g}^{(k)}(\mathbf{x}) = \text{softmax}\left(\mathbf{W}_g^{(k)}\mathbf{x}\right), \qquad k = 1, \ldots, T \tag{12.100}$$

其中，$\mathbf{W}_g^{(k)}$ 为第 k 个任务对应的门控网络参数。如果输入特征的维度为 d，则 $\mathbf{W}_g^{(k)} \in \mathbb{R}^{n \times d}$；$\mathbf{g}^{(k)}(\mathbf{x}) \in \mathbb{R}^n$ 表示该任务对各个专家的融合权重。每个专家网络接收相同的输入特征，并学习不同的特征变换：

$$\mathbf{e}_i = f_i(\mathbf{x}), \qquad i = 1, \ldots, n \tag{12.101}$$

MMoE 模型通常采用结构相同但参数独立的多层感知机（MLP）作为专家网络，并通过 ReLU 等激活函数实现非线性映射。对于第 k 个任务，模型根据对应的门控权重对所有专家的输出进行加权求和，就可以得到该任务专属的融合表示：

$$\mathbf{f}^{(k)}(\mathbf{x}) = \sum_{i=1}^{n} g_i^{(k)}(\mathbf{x}) \mathbf{e}_i \tag{12.102}$$

虽然不同的任务使用的是同一组专家，但由于不同任务之间门控网络的参数彼此独立，即使输入特征相同，各任务也可以生成不同的专家权重，从而形成不同的特征组合。

在获得任务专属的融合表示之后，MMoE 模型通常为每个任务设置独立的塔网络（Tower），将融合表示映射为相应的预测结果：

$$\hat{y}_k = h_k\left(\mathbf{f}^{(k)}(\mathbf{x})\right), \qquad k = 1, \ldots, T \tag{12.103}$$

其中，$h_k(\cdot)$ 表示第 k 个任务的塔网络。对于点击率、有效播放率、点赞率等二分类预估任务，塔网络可以通过全连接层和 Sigmoid 函数输出对应的预测概率。而对于其他类型的预测目标，则可以采用与目标相匹配的输出

层和损失函数。

以短视频精排为例，一个模型可以同时预估点击、有效播放和点赞三个目标。三个任务接收相同的用户、物品及上下文特征，并共享同一组专家网络，但点击任务、有效播放任务和点赞任务分别具有独立的门控网络。因此，点击任务可以更多地使用有助于判断用户是否愿意点击的专家表示，有效播放任务可以更多地依赖与内容消费相关的特征组合，而点赞任务则可以选择更适合描述主动交互意愿的专家表示。这里的关键并不是预先人为指定某个专家只负责某一种行为，而是通过数据驱动的联合训练，让各任务学习适合自身目标的专家组合。

从网络结构上看，MMoE 模型可以视为在共享专家层与各任务塔网络之间引入了一个任务相关的动态融合过程。为了进一步理解这一设计的作用，可以将其与 Shared-Bottom 和 One-gate Mixture-of-Experts（OMoE）模型进行比较。在 Shared-Bottom 模型中，所有任务共享同一个底层网络，其输出再分别输入各任务独立的塔网络。该结构能够利用任务之间的共性，但所有任务必须使用相同的底层表示。当不同任务的关注重点存在明显差异时，共享表示可能难以同时满足所有任务的需求。OMoE 模型在共享专家网络的基础上，只设置一个供所有任务共同使用的门控网络。它能够根据输入动态组合专家，相比固定的共享底层网络更加灵活，但不同任务仍然依赖同一组门控权重，因此对于相同输入会采用相同的专家组合。MMoE 则进一步将一个共享门控网络扩展为多个任务专属的门控网络，使每个任务都能够在共享专家的基础上形成自己的特征组合。三者在参数共享机制上的区别如下所示：

$$
\begin{aligned}
\text{Shared-Bottom:} \quad & \mathbf{f}^{(k)}(\mathbf{x}) = \mathbf{f}(\mathbf{x}) \\
\text{OMoE:} \quad & \mathbf{f}^{(k)}(\mathbf{x}) = \sum_{i=1}^{n} g_i(\mathbf{x}) f_i(\mathbf{x}) \\
\text{MMoE:} \quad & \mathbf{f}^{(k)}(\mathbf{x}) = \sum_{i=1}^{n} g_i^{(k)}(\mathbf{x}) f_i(\mathbf{x})
\end{aligned}
\tag{12.104}
$$

上述这三种结构的主要区别并不在于是否使用多个专家，而在于各个任务能否自主决定如何组合共享表示。Shared-Bottom 固定共享底层网络，OMoE 允许根据输入动态选择专家组合，但所有任务共享同一组门控权重；MMoE 则同时根据输入特征和任务身份调整专家组合，为任务之间的参数共享提供了更大的灵活性。

从任务关系的角度来看，MMoE 模型的优势在于能够根据不同任务之间的关联程度，学习不同的参数共享方式。如果两个任务具有相似的优化目标和数据规律，其门控网络可能为相同的专家分配较高权重，使二者共享更多有用的表示。而如果两个任务之间的差异较大，门控网络则可以学习不同的专家权重组合，减少不适合当前任务的特征变换对预测结果的影响。因此，任务之间的关联不需要完全依赖人工设定，而是可以通过门控网络在训练过程中逐步学习。

需要注意的是，MMoE 模型并没有为每个任务分别构建完整的专家网络，而是让所有任务共享专家，仅增加各任务对应的门控网络及任务塔。相较于为每个任务完全独立地训练模型，这种方式能够在保留一定任务差异的同时，控制模型整体的参数规模。尤其是在多任务数量增加时，门控网络通常只引入相对有限的额外参数。因此，MMoE 模型可以在共享学习能力与模型复杂度之间取得较好的平衡。

在训练阶段，MMoE 模型利用各任务对应的监督标签计算损失，并联合优化专家网络、门控网络和任务塔。假设第 k 个任务的标签为 y_k，预测结果为 $\hat{y}_k$，对应的损失函数为 $\mathcal{L}_k$，则模型整体的训练目标可以写为

$$
\mathcal{L}_{\text{MMoE}} = \sum_{k=1}^{T} \lambda_k \mathcal{L}_k(y_k, \hat{y}_k) \tag{12.105}
$$

其中，λ_k 为第 k 个任务的损失权重，用于调节不同任务对联合优化的影响。对于 CTR、有效播放率和点赞率等二分类任务，每个任务通常会采用二元交叉熵损失。不同任务的梯度会共同更新共享专家，而各自任务的门控网络与任务塔也会根据对应任务的训练目标来进行调整。这样模型既能够利用多个任务的监督信号学习通用特征表示，又能够通过不同的门控权重形成任务相关的特征组合。

不过，MMoE 模型也并不意味着任务越多，模型效果就一定会越好。一方面，不同任务之间的关联程度可能并不相同，部分任务甚至会产生相互冲突的优化信号；另一方面，虽然门控网络能够调整专家的使用比例，但所有任务仍然共享同一组专家，任务之间依旧存在潜在的参数竞争。此外，标准 MMoE 模型中的 Softmax 门控

通常会为所有专家分配权重，加权融合并不意味着只计算少数专家，因此它的主要优势是增强参数共享的灵活性。在实际的工业级推荐系统中，算法工程师仍然需要结合不同任务之间的关系、损失权重、专家数量以及线上资源的预算情况来进行合理的设计。

总的来说，MMoE 模型通过任务独立的门控网络改善了共享专家的组合方式，但其所有任务仍然依赖同一组共享专家。如果任务之间的差异较大，只依靠单纯地调整门控权重可能不足以完全消除多任务之间负迁移的问题。后续的 PLE 模型进一步引入共享专家与任务专属专家的区分，并通过分层结构逐步调整不同任务之间的信息共享程度，从而更加显示地在模型结构层面处理任务共性与任务差异之间的关系。

12.4.2.2 CGC 模型与 PLE 模型

PLE（Progressive Layered Extraction）模型源自于腾讯科技发表于 RecSys 2020 会议的论文《Progressive Layered Extraction (PLE): A Novel Multi-Task Learning (MTL) Model for Personalized Recommendations》[19]。PLE 模型针对工业推荐系统多任务学习中的负迁移问题，通过显式区分共享专家与任务专属专家，并在多层网络中逐步分离不同任务的特征表示，改善不同任务之间的信息共享与参数干扰问题。

在工业级推荐系统中，多任务学习的精排模型通常包含离线训练和在线推理两个阶段，其整体流程如图 12.13 所示。在离线阶段，精排模型根据用户历史行为及对应标签，联合训练点击、观看、互动等多个预测任务；在在线阶段，精排模型根据当前请求输出各个任务的预测结果，再由排序策略将不同的预测结果组合为最终的排序分数。例如，在短视频推荐中，可以根据点击率、有效播放率、完播率和互动率等预测结果来构建综合的排序分数，并通过线上实验调整不同目标的权重。因此，多任务学习不仅需要提高各个预测目标的精度，还需要考虑不同任务之间的关联，以及它们对最终排序效果的共同影响。

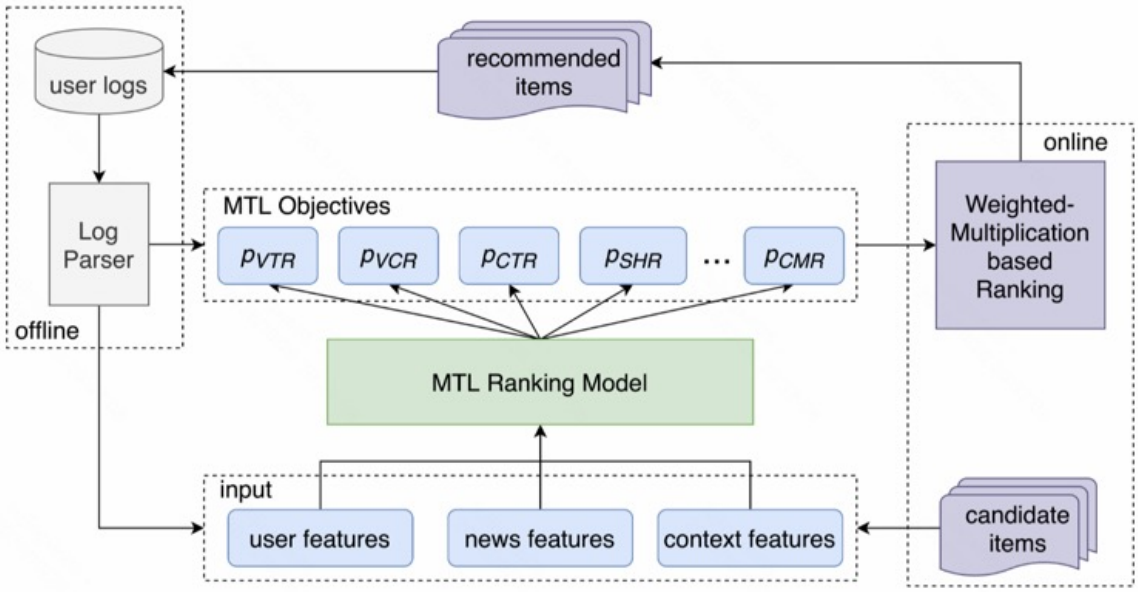


图 12.13: 多任务学习的排序系统离线训练和在线推理框架。

多任务学习的一个核心挑战是负迁移（Negative Transfer），即共享信息并不一定能够使所有任务同时受益。PLE 方法进一步分析了工业推荐中的跷跷板现象（Seesaw Phenomenon）：一个任务的效果得到改善时，另一个任务的效果却可能下降，使得模型难以同时超越各自独立训练的单任务模型。以视频推荐中的有效播放率（VTR）和观看完成比例（VCR）为例，VTR 通常是判断一次播放是否达到有效观看标准的二分类任务，而 VCR 则用于回归预测观看完成比例。两者虽然都与观看行为有关，但其相关关系并不简单。其中有效播放的标签会受到播放行为和观看时长的共同影响，而自动播放、主动点击等不同场景中的行为分布也可能存在差异。因此，不同任务之间的关联不仅取决于任务本身，还可能随输入样本和推荐场景而发生变化。

早期的 MMoE 模型通过任务独立的门控网络，使不同任务能够动态组合共享专家，已经比简单的底层参数

共享网络更加灵活。但在 MMoE 模型中，所有任务仍然使用同一组专家网络。当不同任务的需求存在明显差异时，共享专家仍然有可能引入不适合某个任务的特征信息。PLE 模型针对这一问题提出了改进方案：

定义 12.8

PLE 模型不仅允许不同任务采用不同的特征组合，还显式区分共享知识与任务专属知识，并通过多层网络逐步完成信息提取与任务分离。

与 MMoE 模型相比，PLE 模型首先引入了定制门控控制（Customized Gate Control，CGC）模块，在共享专家之外，为每个任务设置专属专家。然后 PLE 模型进一步将多个 CGC 模块逐层堆叠，形成完整的 PLE 模型架构，如图 12.14 所示。

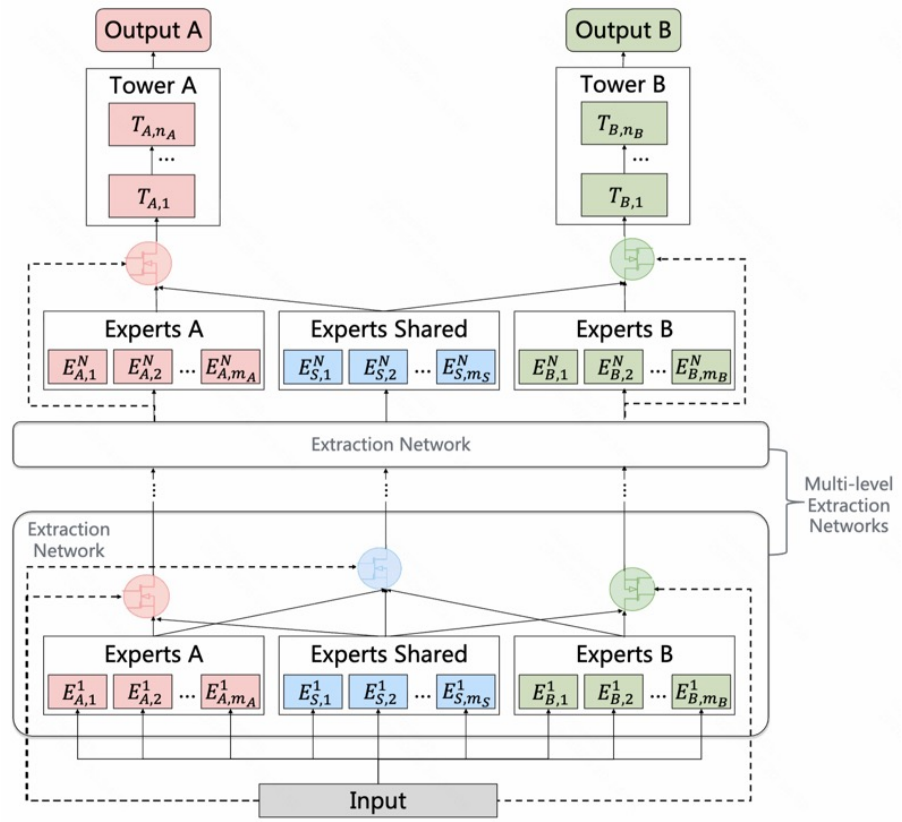


图 12.14: PLE 模型的网络结构。

PLE 模型中 CGC 模块的核心设计是将专家划分为两类，即**共享专家**负责学习多个任务共同需要的特征模式，而**任务专属专家**则侧重提取特定任务所需要的信息。假设任务数量为 T，共享专家数量为 m_s，任务 k 的专属专家数量为 m_k。共享专家和任务专属专家的输出分别记为

$$\mathbf{e}_{s,i} = f_{s,i}(\mathbf{x}), \qquad \mathbf{e}_{k,i} = f_{k,i}(\mathbf{x}) \tag{12.106}$$

其中，$\mathbf{x}$ 为输入特征表示，$f_{s,i}$ 和 $f_{k,i}$ 分别表示共享专家与任务专属专家网络。对于任务 k，CGC 模块仅将共享专家和该任务自身的专属专家作为门控网络的候选输入。门控网络根据输入特征计算各个专家的组合权重：

$$\mathbf{g}_k(\mathbf{x}) = \text{softmax}\left(\mathbf{W}_g^{(k)}\mathbf{x}\right) \tag{12.107}$$

其中，$\mathbf{W}_g^{(k)}$ 为任务 k 的门控参数。将权重与对应专家输出进行加权融合，可得到任务 k 的表示：

$$\mathbf{h}_k = \sum_{i=1}^{m_s} g_{k,s,i}(\mathbf{x})\mathbf{e}_{s,i} + \sum_{i=1}^{m_k} g_{k,k,i}(\mathbf{x})\mathbf{e}_{k,i} \tag{12.108}$$

其中，$g_{k,s,i}$ 和 $g_{k,k,i}$ 分别表示任务 k 分配给共享专家和自身专属专家的权重。融合后的表示再输入对应的任务

塔网络，得到最终的预测结果：

$$\hat{y}_k = t_k(\mathbf{h}_k) \tag{12.109}$$

其中，t_k 表示任务 k 的任务塔网络，用于将融合后的任务表示映射为最终预测结果。

CGC 模块的结构如图 12.15 所示。与 MMoE 模型中所有任务都可以访问全部共享专家的设计不同，CGC 模块通过限制任务塔能够访问的专属专家，避免某个任务直接使用其他任务的专属表示。与此同时，共享专家仍然能够接受多个任务的训练信号，从而保留跨任务知识迁移的能力。需要注意的是，CGC 模块中还可以设置共享门控网络，将共享专家与各任务的专属专家进行融合，以形成供下一层使用的共享表示。这样，模型既能为每个任务提取专属信息，也能持续汇集不同任务之间具有共性的知识。

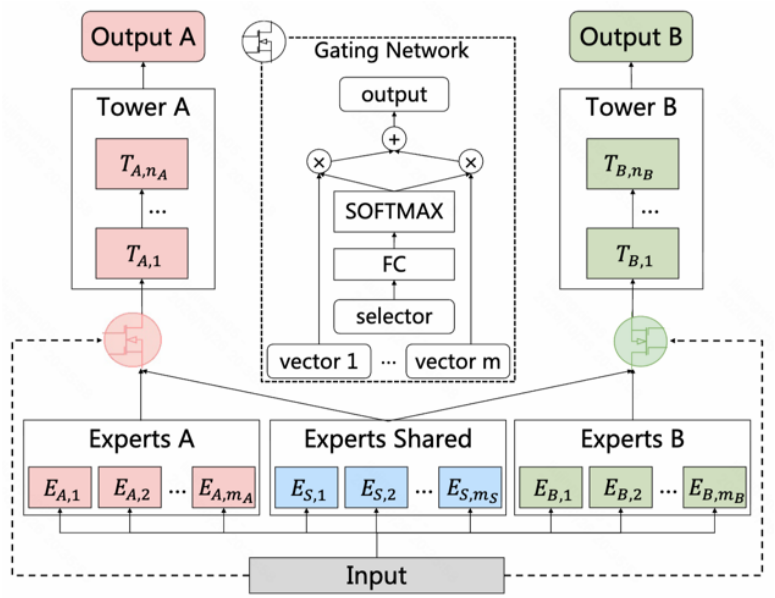


图 12.15: PLE 模型中的 CGC 模块网络结构。

在 CGC 模块的基础上，PLE 模型进一步引入多层专家与门控网络，将不同任务的表示逐层提取和分离。它的核心思想如下：

定义 12.9

PLE 模型网络底层学习到的特征通常同时包含通用信息与任务相关信息，仅在单层网络中将二者完全区分，未必能够充分利用不同任务之间的关联。

因此，PLE 模型在每一层中同时保留共享专家和任务专属专家，并分别通过任务门控网络与共享门控网络完成特征融合。然后，融合之后的任务表示和共享表示再传递给更高层的专家网络，逐步形成更加抽象的特征表示。具体而言，假设第 j 层的任务 k 表示为 $\mathbf{h}_k^{(j)}$，共享表示为 $\mathbf{h}_s^{(j)}$。在该层中，任务 k 的门控网络以较低层对应的任务表示为输入，对共享专家和任务 k 的专属专家进行选择与融合；而共享门控网络则综合当前层的共享专家与各任务专属专家的输出，生成新的共享表示。我们可以将这一过程概括为

$$\begin{aligned}\mathbf{h}_k^{(j)} &= G_k^{(j)}\left(\mathbf{h}_k^{(j-1)}, \mathbf{h}_s^{(j-1)}\right)\\ \mathbf{h}_s^{(j)} &= G_s^{(j)}\left(\mathbf{h}_s^{(j-1)}, \mathbf{h}_1^{(j-1)}, \ldots, \mathbf{h}_T^{(j-1)}\right)\end{aligned} \tag{12.110}$$

其中，$G_k^{(j)}$ 表示任务 k 在第 j 层的专家融合过程，$G_s^{(j)}$ 表示共享分支的专家融合过程。这里的符号是对逐层路由过程的概括，具体实现通过各层的专家网络与 Softmax 门控完成。最终，各个任务的高层表示分别输入对应的任务塔，得到对应目标的预测结果。

通过上述这种渐进式的提取结构，PLE 模型并不是从网络底层开始就将所有任务完全隔离，而是在共享信息与任务专属信息之间建立逐层调整的机制。共享分支可以持续吸收不同任务中具有共性的特征，而任务分支

则逐渐形成更加符合自身预测目标的表示。与单层的 CGC 模型相比，多层 PLE 模型能够进一步处理更高层次的特征关系。而与传统的 MMoE 模型相比，PLE 模型则通过显式区分共享专家和任务专属专家，减少不同任务之间不必要的信息干扰。

图 12.16进一步对比了 PLE 模型与其他多任务学习模型在网络路由方面的区别。左侧展示了单层多任务模型，包括 Hard Parameter Sharing、Asymmetry Routing、Customized Sharing 和 CGC 等结构；右侧则展示了多层共享机制，包括 Cross-Stitch Network、Sluice Network、ML-MMOE 和 PLE 等模型。不同结构的主要区别在于共享表示如何生成、任务专属信息从何处提取，以及任务之间的信息如何在不同网络层之间传递。PLE 模型通过共享专家与任务专属专家的逐层组合，实现渐进式的信息共享与任务分离。

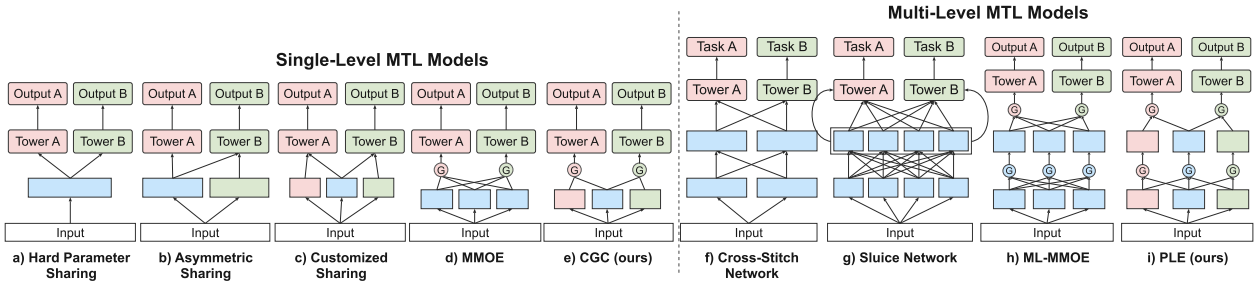


图 12.16: PLE 模型与传统模型在网络路由方面的对比。

在训练阶段，PLE 模型与其他多任务学习模型一样，通过联合优化各任务的损失函数来更新共享参数和任务专属参数。假设任务总数为 T，第 k 个任务的损失为 $\mathcal{L}_k$，其损失权重为 λ_k，则多个任务联合的优化目标可以写为如下形式：

$$\mathcal{L}_{\text{PLE}} = \sum_{k=1}^{T} \lambda_k \mathcal{L}_k \tag{12.111}$$

不过，工业级推荐场景中的不同任务往往具有不同的样本空间。例如，用户只有在点击物品之后才可能产生评论或分享行为，因此评论和分享任务的有效训练样本通常只是点击样本的子集。在实际训练时，我们应该只在样本属于相应任务的有效样本空间时才计算该任务的损失函数，并根据有效样本数量进行归一化，避免将不适用的标签直接作为负样本。与此同时，不同任务的损失权重也可能会影响多任务联合优化的方向，在实际系统中还可以根据训练过程去手动或自动地调整不同任务的损失权重，从而适应不同任务对模型优化的影响。

总体而言，PLE 模型在 MMoE 的任务独立门控思想之上，进一步通过 CGC 模块显式区分共享专家和任务专属专家，并利用多层渐进式路由逐步提取通用知识与任务特异性信息。它所解决的核心问题不是简单增加专家数量或网络深度，而是让不同任务能够在共享有价值信息的同时，减少相互干扰。在点击、观看、互动等目标之间存在复杂关联的工业推荐场景中，这种结构为缓解负迁移和跷跷板现象提供了一种有效的架构设计思路。不过，PLE 模型并不能保证所有任务都同时获得提升，它的实际效果仍取决于任务之间的相关性、专家网络的容量、损失权重以及训练数据的质量。在一些推荐场景中，PLE 模型的效果可能优于 MMoE 模型，而在其他的推荐场景中，PLE 模型可能并不具备明显优势，甚至可能出现部分任务性能下降的情况。因此，在实际落地部署过程中，具体是使用 MMoE 模型还是 PLE 模型最终还是得以线上 A/B 测试的结果为准。

12.4.2.3 多任务学习模型架构的总结与讨论

通过前面几个小节对 MMoE、CGC 和 PLE 模型的介绍，我们可以看到，多任务学习模型架构的演进始终围绕一个核心问题展开，即：

笔记 如何在多个预测任务之间共享有效的信息，同时减少不同任务之间的相互干扰。

Shared-Bottom 模型通过共享底层网络实现参数复用；MMoE 模型进一步引入任务独立的门控网络，使各个任务能够动态组合共享专家；而 PLE 模型则通过显式区分共享专家与任务专属专家，并逐层调整不同任务之间的信息共享方式，进一步提高参数共享的灵活性。这些模型虽然采用了不同的网络路由机制，但目标都是在任务共性与任务差异之间取得更合理的平衡。

在实际工业级推荐系统中，即使采用了 MMoE 或者 PLE 这样的复杂网络结构，也不意味着所有预测任务都能同时获得提升。多任务学习始终绕不开不同任务学习过程中存在的跷跷板效应。因此，多任务学习的精排模型不能仅以整体损失下降作为判断模型是否有效的标准，而需要分别观察各任务的离线指标及线上业务指标。如果一个模型提升了观看时长预测的 AUC，却导致点击或互动任务显著退化，即使总体损失有所下降，也未必能够带来更好的线上排序效果。对于最终由多个预测结果组合而成的排序分数，还需要关注各任务之间的权衡关系，以及不同任务的改善是否真正转化为整体推荐收益。

12.4.2.3.1　门控网络的训练问题与专家退化　除此之外，MMoE 和 PLE 模型通过门控网络为不同任务提供了更加灵活的专家组合方式，但门控机制本身也引入了新的训练问题。理解这些问题时，需要区分专家输出退化与专家路由坍缩这两类相关但并不完全相同的现象。

一方面，门控网络可能将过多的权重分配给少数几个专家，使其他专家长期得不到充分利用。这通常称为**专家路由坍缩（Routing Collapse）**。例如，在训练初期，如果某个专家因为初始化或输入分布的原因取得了略好的预测效果，门控网络可能逐渐提高该专家的权重。随着训练的继续进行，该专家获得更多梯度并进一步增强，而其他专家得到的训练信号相对减弱，最终形成少数专家占据主要权重的现象。对于标准的 MMoE 模型而言，Softmax 通常仍会给其他专家分配非零权重，因此这里的坍缩更多表现为专家贡献严重不均，而不一定意味着其他专家完全不参与计算。

另一方面，即使门控权重没有明显集中在少数专家上，专家自身的输出也可能出现退化。例如，采用 ReLU 激活函数的专家可能在大量输入上产生接近全零的输出，使得其表示能力受到限制。不同专家的输出分布如果存在很大差异，门控网络也可能难以充分利用这些专家。此类问题可以称为**专家表示坍缩或专家退化**。需要注意的是，专家输出大量为零、门控权重集中，以及不同任务争用共享专家，是不同层面的问题，不能仅凭某一个门控统计量就认定模型已经发生了专家坍缩。

针对上述问题，工业系统中通常会从精排模型的网络结构、门控优化和专家正则化等多个方面进行改进。首先，可以对专家输出或专家隐藏层进行 Layer Normalization 等归一化处理，降低不同专家之间的输出尺度差异；也可以采用 Swish 等激活函数，减少 ReLU 在部分输入区域内长期输出零值所造成的训练问题。归一化和激活函数的选择需要结合专家网络结构进行验证，因为过强的归一化同样可能改变专家原本需要学习的表示分布。

其次，可以调整门控网络的初始化、学习率和 Softmax 温度，降低训练初期门控权重过早集中到少数专家的风险。假设第 k 个任务对第 i 个专家的门控得分为 $a_i^{(k)}$，则门控权重可以写为

$$g_i^{(k)} = \frac{\exp(a_i^{(k)}/\tau)}{\sum_{j=1}^{M} \exp(a_j^{(k)}/\tau)} \tag{12.112}$$

其中，M 为专家数量，τ 为 Softmax 温度。当 τ 较小时，权重分布更容易集中于少数专家。适当提高温度可以让早期的专家组合更加平滑，但也可能削弱模型后续形成任务专属路由的能力。因此，温度并不是越大越好，应该结合训练阶段和专家使用情况进行调节。

此外，我们还可以通过辅助的负载均衡损失或门控熵正则化等方法，来改善专家使用过度集中的问题。以负载均衡为例，假设一个批次包含 B 个样本，第 b 个样本对第 i 个专家的门控概率为 $g_i(x_b)$，可以定义专家的平均权重为

$$I_i = \frac{1}{B} \sum_{b=1}^{B} g_i(x_b) \tag{12.113}$$

在希望避免专家过度集中的情况下，可以引入如下正则项：

$$\mathcal{L}_{\text{bal}} = M \sum_{i=1}^{M} I_i^2 - 1 \tag{12.114}$$

当各专家的平均权重完全均匀时，该项为零；反之，当权重集中于少数专家时，该项将增大。另一种方法是直接提高门控分布的熵，避免训练初期形成过于尖锐的权重分布。这些方法可以与主任务损失联合优化，但其作用主要是约束专家使用模式。

在实际的工业级推荐系统中，也不宜简单要求所有专家在任何情况下都具有相同的使用率。不同专家可能

逐渐形成不同的功能分工，某些专家在特定任务或特定样本群体上承担更多工作本身是合理的。如果负载均衡约束过强，反而可能阻碍这种差异化分工。因此，通常需要结合门控权重分布、专家输出范数、激活值分布、梯度大小以及不同任务的预测指标判断是否存在实际问题，再决定是否引入正则化。

除了上述方法外，还可以使用门控 Dropout 或噪声扰动的方法来降低模型对少数专家的过度依赖，或者通过专家输出之间的去相关约束促进专家学习不同的表示。这些方法各有侧重，负载均衡主要改善专家的使用分布，门控正则化主要约束权重分布，输出归一化和激活函数改造则主要改善专家自身的表示质量。在实际应用中，我们应该根据观察到的问题类型来选择相应的优化手段，并通过线上的消融实验来确认优化方法具体的作用。

MMoE 和 PLE 模型的另一个训练特点是，门控网络与专家网络之间存在紧密的梯度耦合。门控权重决定哪些专家的输出对任务预测产生更大影响，而专家的预测表现又会反过来影响门控权重的更新方向。在多个任务共同训练时，这种交互还会受到不同任务梯度的共同影响。因此，相比结构简单的共享底层网络，带有门控的精排模型有时候会表现出更加敏感的训练动态，例如收敛速度下降、专家使用率随训练过程发生明显变化，或者个别任务的门控分布过早集中等现象。不过，这并不意味着引入 Gate 必然降低收敛速度，实际表现仍然取决于专家数量、网络规模、数据分布、优化器以及参数初始化等因素。

除了通过常规的门控正则化改善训练稳定性，我们还可以进一步调整门控网络的设计，使专家选择过程更直接地反映任务自身的需求。传统 MMoE 模型通常使用输入特征生成门控权重：

$$\mathbf{g}^{(k)}(\mathbf{x}) = \mathrm{softmax}\left(\mathbf{W}_g^{(k)}\mathbf{x}\right) \tag{12.115}$$

这种方式虽然允许不同任务采用不同的门控参数，但门控的得分主要由输入特征经过线性或非线性变换得到，并没有显式地将任务网络已经学习到的高层语义表示作为专家选择的参照。因此，对于具有明显差异的预测目标，可以进一步考虑将任务的高层表示引入 MMoE 模型的门控过程，让专家选择不仅取决于输入样本，还与任务自身所需要的语义信息相关。

12.4.2.3.2　MMoE 门控改造方案　在实际系统的多任务模型迭代中，还可以进一步改造 MMoE 模型中的门控机制，使专家选择过程能够利用任务塔已经学习到的高层语义信息。一种可行的思路是，不再直接使用当前请求的原始输入特征计算 Gate 权重，而是将任务塔的高层 Embedding 作为 Gate 的输入。由于该 Embedding 已经经过相应任务的预测目标训练，因此能够包含与任务相关的抽象信息，可以为专家选择提供不同于原始输入特征的参考。

在具体实现中，可以使用 user_id 与 item_id 作为缓存键，保存用户与物品最近一次交互时精排模型中各任务塔产生的高层 Embedding。在当前请求到达时，根据用户和目标物品查询对应的历史表示。假设第 k 个任务最近一次交互时的高层 Embedding 为

$$\mathbf{z}_{u,i,k}^{\mathrm{cache}} \in \mathbb{R}^{d_k} \tag{12.116}$$

其中，u 和 i 分别表示用户与目标物品，d_k 为第 k 个任务塔顶层 Embedding 的维度。该表示反映了模型在此前交互中针对这一用户与物品组合所形成的任务相关表示。

原始 MMoE 模型根据当前输入特征 $\mathbf{x}$ 计算门控权重：

$$\mathbf{g}^{(k)}(\mathbf{x}) = \mathrm{softmax}\left(\mathbf{W}_g^{(k)}\mathbf{x} + \mathbf{b}_g^{(k)}\right) \tag{12.117}$$

在改造后的结构中，可以将 Gate 的输入替换为缓存的历史任务表示：

$$\mathbf{g}^{(k)}(u,i) = \mathrm{softmax}\left(\mathbf{W}_g^{(k)}\,\mathrm{sg}\left(\mathbf{z}_{u,i,k}^{\mathrm{cache}}\right) + \mathbf{b}_g^{(k)}\right) \tag{12.118}$$

其中，$\mathbf{W}_g^{(k)} \in \mathbb{R}^{M\times d_k}$，$\mathbf{b}_g^{(k)} \in \mathbb{R}^{M}$ 为可学习参数，M 为专家数量，$\mathrm{sg}(\cdot)$ 表示停止梯度传播。由于门控网络的输出维度仍然为 M，即使任务塔顶层 Embedding 的维度与原始输入特征不同，也不影响 Gate 为各个专家生成独立的权重。

需要注意的是，该方案并不改变专家网络自身的输入。对于当前请求，专家仍然根据当前特征计算输出：

$$\mathbf{e}_j = f_j(\mathbf{x}), \qquad j = 1,\ldots,M \tag{12.119}$$

然后，模型根据缓存的历史任务表示所生成的门控权重，对当前专家输出进行加权融合：

$$\mathbf{h}_k(\mathbf{x}, u, i) = \sum_{j=1}^{M} g_j^{(k)}(u, i) f_j(\mathbf{x}) \tag{12.120}$$

最终，融合后的表示再输入第 k 个任务对应的塔网络，得到该任务的预测结果：

$$\hat{y}_k = t_k\left(\mathbf{h}_k(\mathbf{x}, u, i)\right) \tag{12.121}$$

因此，这种改造的本质是将原先基于当前输入特征的门控机制，转变为基于历史任务表示的门控机制。当前输入特征仍然决定专家网络如何提取特征，而用户与物品最近一次交互时形成的任务高层表示则用于引导不同专家的组合方式。与通过点积对 Query 和 Key 进行匹配的注意力式门控不同，这里直接通过一个可学习的线性映射将历史任务表示转换为各个专家的门控得分，不需要额外构造逐专家的候选表示。

从建模角度来看，这种设计的优势在于引入了具有历史语义的条件信息。传统 MMoE 模型主要根据当前样本的输入特征计算专家权重，而任务塔的高层 Embedding 已经经过对应任务的训练，包含了模型对用户与物品组合的抽象表示。使用该表示生成 Gate 权重，有可能使专家选择更加符合当前任务所需要的特征模式，从而减少不适合该任务的专家表示对预测结果的干扰。同时，门控网络仍然使用可学习的参数将高层 Embedding 映射为专家权重，因此即使缓存表示与原始输入的维度不同，也可以通过相应的投影矩阵完成适配。与增加新的专家网络相比，这种改造主要改变门控网络的输入来源，不需要改变原有专家网络的主体结构，具有较好的工程兼容性。

在梯度传播方面，缓存机制还使历史任务表示与当前请求的计算图相互解耦。缓存中的 Embedding 来源于用户与物品此前一次交互时的模型计算。因此，在当前前向计算中，历史表示可以作为一个固定输入使用，不会形成当前 Gate 输出与任务塔高层表示之间的循环依赖。对缓存表示执行 Stop Gradient，主要是明确不沿历史表示的生成路径反向传播梯度。它并不会自动切断 Gate 自身的梯度：只要门控映射、Softmax 和后续专家融合保持可微，当前任务的损失仍然能够反向更新 $\mathbf{W}_g^{(k)}$ 和 $\mathbf{b}_g^{(k)}$，使 Gate 学习如何根据历史任务表示选择更合适的专家。

当然，这种设计也引入了需要在工程实现中考虑的问题。首先，它依赖用户与物品组合的历史缓存。对于没有历史交互记录的用户—物品组合，模型无法获得对应的任务高层 Embedding，需要设计缓存缺失时的回退策略，例如使用原始输入 Gate、默认表示或其他可用的历史表示。缓存命中率会直接影响该方案能够覆盖的流量范围，因此需要结合实际推荐场景评估其适用性。

其次，缓存中的表示可能与当前模型存在一定的版本差异。历史 Embedding 通常由先前版本的模型产生，而当前模型的专家网络和任务塔可能已经经过多轮训练更新。如果两者的表示分布发生明显变化，Gate 就可能难以根据历史表示形成稳定的专家选择。因此，可以考虑为缓存记录模型版本和交互时间，并根据实际情况设计缓存刷新、过期和版本切换策略。此外，历史表示主要反映此前交互时的特征与任务状态，不一定能够充分体现当前请求中的实时上下文变化。如果上下文变化对专家选择具有较大影响，还可以进一步探索将历史任务表示与当前输入特征结合的门控结构，但这种扩展需要通过实验确认其必要性。

此外，训练过程也需要尽可能模拟线上缓存的产生和读取方式。在离线训练时，应当确保当前样本使用的是此前交互所产生的历史表示，避免将当前样本自身或未来交互生成的 Embedding 用作该样本的 Gate 输入，从而引入标签泄漏或训练与推理不一致的问题。

在一些工业级场景中，这种以历史任务高层 Embedding 引导 Gate 的 MMoE 改造能够取得显著的线上 A/B 指标提升，说明在特定业务场景下，利用历史任务表示调整专家组合具有实际价值。这一结果也表明，多任务模型的改进不一定需要增加专家数量或引入更复杂的网络路由，还可以从门控网络使用什么信息来决定专家权重这一角度进行优化。不过，线上收益本身并不能单独证明效果完全来自历史任务语义的引入，缓存覆盖率、历史表示的新鲜程度以及门控参数的优化过程都可能对最终效果产生影响。后续可以通过对比原始 MMoE、使用历史任务表示的 Gate，以及不同缓存更新策略等方案，并结合各任务的线上指标、门控权重分布和缓存命中率，进一步分析该改造的作用机制与适用边界。

12.4.2.3.3 MMoE 的后续改进方案 近年来，针对工业多任务学习中专家利用不均衡以及负迁移等问题，也出现了进一步改进的专家网络架构。例如，HoME（Hierarchy of Multi-Gate Experts for Multi-Task Learning at Kuaishou）模型发表于 KDD 2025 学术会议 [21]。该工作结合快手短视频推荐中的实际训练问题，进一步总结了三类现象：部分专家由于输出大量为零而难以有效发挥作用；共享专家可能逐渐被单个任务占据，从而退化为事实上的任务专属专家；部分数据稀疏的任务则可能无法充分训练自身的专属专家，转而过度依赖共享专家。

针对这些问题，HoME 模型通过专家归一化（比如引入 Layer Norm 等）与 Swish 激活改善专家输出分布，通过层级掩码机制调整任务之间的共享路径，并利用 Feature-Gate 与 Self-Gate 等机制改善不同专家获得的训练信号。这些改进表明，工业多任务模型的优化不应只关注门控权重是否均匀，还需要同时考虑专家的有效表示能力、不同任务对共享专家的实际占用，以及任务专属专家能否获得足够的训练信号。HoME 模型也说明，单纯增加专家数量或叠加更多门控层，并不能保证模型的参数共享更加有效，仍然需要针对训练过程中出现的具体问题进行结构设计。

总体而言，多任务精排模型的设计不能简单归结为选择 MMoE、PLE 或者其他某一种固定架构。更重要的是识别实际系统中的主要瓶颈。如果多个任务缺乏有效的信息共享，可以考虑更灵活的门控机制。如果共享参数导致明显的任务冲突，可以通过任务专属专家或渐进式分离减少负迁移。如果专家使用率严重失衡或专家表示出现退化，则需要从输出分布、激活函数、门控优化和路由结构等方面进行针对性改进。对于任务之间存在复杂关联的工业推荐系统，还可以进一步利用任务自身的高层表示来指导专家选择，使门控机制更直接地感知任务差异。

无论采用哪种改造方式，最终都需要回到工业级推荐精排模型的实际目标，即不仅要提高各任务的预估精度，还要保证这些改善能够转化为稳定的线上排序收益。在多任务学习中，网络结构提供的是更灵活的信息共享机制。合理的架构设计需要结合任务关系、样本分布、梯度传播和专家使用情况持续迭代，并通过离线诊断与线上 A/B 实验共同验证其有效性。

12.4.3 基于 Transformer 的统一模型架构

随着 Transformer 架构和大规模预训练模型的发展，工业级推荐系统也开始探索以 Transformer 为基础的统一模型架构。传统精排模型通常采用 Embedding & MLP 的建模范式，将用户、物品和上下文等特征映射为稠密向量，再通过特征拼接、显式交互或多层感知机完成预测。这类方法在工业场景中取得了广泛应用，但当模型同时处理大量异构特征、复杂的用户行为序列和多种预测目标时，特征之间的交互关系往往需要通过不同的网络模块进行分别建模。而 Transformer 提供了另一种思路，即：

定义 12.10

将不同特征或行为统一表示为一组 Token，通过注意力机制动态计算 Token 之间的关联，再利用堆叠的网络层逐步提取更高层次的表示。 ♣

上述这种设计使得 Transformer 模型不仅可以用于用户行为序列建模，也可以用于推荐精排模型中的特征交互。一个 Token 可以对应用户属性、物品属性、上下文特征，也可以对应用户历史行为中的某个物品或某次行为。在完成嵌入表示之后，模型通过 Self-Attention、前馈网络和残差连接等基本组件进行多层变换，最后将目标 Token、池化结果或其他聚合表示输入预测层，完成点击率、有效播放率、观看时长等目标的预估。

Transformer 的另一个重要特点是其架构具有较强的可扩展性。相较于仅依靠堆叠稠密网络增加模型容量的方式，Transformer 可以通过增加层数、隐藏维度、注意力头数和训练数据规模提升表示能力，也可以将部分前馈网络替换为稀疏专家网络，在控制单次前向计算量的同时扩大模型的总参数规模。随着大规模语言模型的发展，**Scaling Law（缩放定律）**也为推荐模型的规模选择和训练策略提供了研究思路。不过，语言模型中的缩放规律并不能直接视为推荐任务中的普遍规律。由于推荐数据具有高基数稀疏特征、快速变化的用户行为、多目标监督和明显的线上服务约束，模型规模、训练数据和计算资源之间的关系仍然需要通过实际实验验证。

本节首先介绍 Transformer 的基本结构及其核心 Attention 机制，再讨论位置编码和高效 Attention 的实现，随后结合 AutoInt、DIN、BST、SASRec 等模型说明 Transformer 相关机制在推荐系统中的应用，最后介绍 Transformer

与 Mixture-of-Experts（MoE）的结合方式，以及 DeepSeekMoE 模型对专家划分与知识共享机制的改进方案。

12.4.3.1 Transformer 模型的基本结构

Transformer 模型最初是由谷歌大脑的 Vaswani 等人在 NeurIPS 2017 会议论文《Attention Is All You Need》中提出的 [20]。其核心变化是以注意力机制作为序列元素之间信息交互的主要方式，不再依赖循环神经网络逐步处理序列，也不需要通过卷积核限制元素之间的交互范围。对于输入序列，模型可以并行计算各个位置之间的注意力关系，从而在同一层中建立不同位置之间的联系。

假设输入序列包含 L 个 Token，每个 Token 的表示维度为 d_{model}，则输入可以写为

$$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_L]^\top \in \mathbb{R}^{L \times d_{\text{model}}} \tag{12.122}$$

在推荐精排模型中，这些 Token 可以来自不同的特征字段，也可以对应用户历史行为中的不同物品。模型首先需要为每个 Token 构造包含内容和必要位置信息的输入表示，再通过注意力层完成信息交互。

Attention 机制的基本计算过程是先将输入投影为 Query、Key 和 Value 三组向量。其中，Query 表示当前 Token 希望获取什么信息，Key 用于描述各个候选信息的匹配特征，而 Value 则承载真正需要聚合的内容。对于输入矩阵 $\mathbf{X}$，我们有

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q, \qquad \mathbf{K} = \mathbf{X}\mathbf{W}_K, \qquad \mathbf{V} = \mathbf{X}\mathbf{W}_V \tag{12.123}$$

其中，$\mathbf{W}_Q$、$\mathbf{W}_K$ 和 $\mathbf{W}_V$ 为可学习参数矩阵。模型通过 Query 与 Key 的内积计算不同 Token 之间的相关性，并使用 Softmax 将相关性转换为归一化权重。为避免点积数值随着向量维度增加而过大，Scaled Dot-Product Attention 引入了一个缩放因子，因此它的完整计算过程如下：

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}} + \mathbf{M}\right)\mathbf{V} \tag{12.124}$$

其中，d_k 为 Key 向量的维度，$\mathbf{M}$ 表示可选的注意力掩码。对于允许参与注意力计算的位置，掩码值为零；而对于需要屏蔽的位置，掩码值会设置为负无穷或相应的数值下界，使这些位置经过 Softmax 后的权重接近于零。如果不需要任何掩码，则可以省略 $\mathbf{M}$ 这个掩码矩阵。

由此可见，Attention 机制并不是简单地将所有 Token 拼接后交给 MLP，而是根据当前输入动态计算信息聚合权重。对于推荐系统中的特征交互，这意味着模型可以根据当前用户、目标物品及上下文，动态调整不同特征的贡献；而对于行为序列建模，则意味着模型能够根据不同历史行为之间的关联，提取更加适合当前任务的信息。

此外，标准的 Transformer 层还包含前馈网络、残差连接和 Layer Normalization 这几个部分。Attention 机制负责在不同 Token 之间交换信息，而前馈网络则对每个位置的表示进行独立的非线性变换。典型的前馈网络可以写为

$$\text{FFN}(\mathbf{x}) = \mathbf{W}_2\,\text{GELU}(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 \tag{12.125}$$

其中，$\mathbf{W}_1$ 通常先将隐藏维度扩大到 d_{ff}，再由 $\mathbf{W}_2$ 将其映射回 d_{model}。通过交替执行 Attention 和 FFN，模型逐层构建包含更丰富上下文信息的 Token 表示。

在实际的代码实现中，Transformer 层通常还会使用残差连接来保留上一层的信息，并利用 Layer Normalization 来稳定不同网络层之间的数值分布。下面给出一个 Pre-LN 结构的 Transformer 层，其先对输入进行归一化，再执行 Attention 或 FFN，最后通过残差连接将结果加回输入：

$$\begin{aligned} \mathbf{X}' &= \mathbf{X} + \text{Attention}\left(\text{LN}(\mathbf{X})\right), \\ \mathbf{X}_{\text{out}} &= \mathbf{X}' + \text{FFN}\left(\text{LN}(\mathbf{X}')\right) \end{aligned} \tag{12.126}$$

需要注意的是，这里省略了投影和多头拆分等内部细节。在实际代码中通常还会在子层输出后使用 Dropout，以降低过拟合的风险。

12.4.3.2 Self-Attention、多头注意力与目标注意力

Self-Attention（自注意力机制）是 Transformer 模型中最基本的交互机制。在 Self-Attention 中，Query、Key 和 Value 都由同一组输入特征投影得到，因此每个 Token 都能够根据自身的内容，动态聚合输入序列中其他 Token 的信息。对于推荐系统而言，如果输入是一组用户、物品和上下文特征，Self-Attention 可以学习这些特征之间的关系；如果输入是按时间排列的用户行为序列，则可以学习不同历史行为之间的依赖。

为了更清楚地理解 Self-Attention 机制的计算过程，下面给出基于 PyTorch 的单头实现。其中，输入张量的形状为 $[B, L, d_{\text{model}}]$，分别表示批次大小、序列长度和隐藏维度。注意这里的布尔掩码约定为 True 表示允许参与注意力计算，False 表示需要屏蔽。

代码 12.1: 基于 PyTorch 手写 Scaled Dot-Product Attention 与 Self-Attention

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


def scaled_dot_product_attention(q, k, v, allowed_mask=None, dropout_p=0.0, training=True):
    # q: [B, H, Lq, d_k]
    # k: [B, H, Lk, d_k]
    # v: [B, H, Lk, d_v]
    scores = torch.matmul(q, k.transpose(-2, -1))
    scores = scores / math.sqrt(q.size(-1))

    if allowed_mask is not None:
        scores = scores.masked_fill(~allowed_mask, torch.finfo(scores.dtype).min)

    weights = F.softmax(scores, dim=-1)
    weights = F.dropout(weights, p=dropout_p, training=training)
    output = torch.matmul(weights, v)
    return output, weights


class SelfAttention(nn.Module):
    def __init__(self, d_model, dropout=0.0):
        super().__init__()
        self.q_proj = nn.Linear(d_model, d_model)
        self.k_proj = nn.Linear(d_model, d_model)
        self.v_proj = nn.Linear(d_model, d_model)
        self.out_proj = nn.Linear(d_model, d_model)
        self.dropout = dropout

    def forward(self, x, allowed_mask=None):
        q = self.q_proj(x).unsqueeze(1)
        k = self.k_proj(x).unsqueeze(1)
        v = self.v_proj(x).unsqueeze(1)

        out, weights = scaled_dot_product_attention(
```

```
            q, k, v, allowed_mask=allowed_mask,
            dropout_p=self.dropout, training=self.training
        )
        out = out.squeeze(1)
        return self.out_proj(out), weights.squeeze(1)
```

上面的实现展示了 Self Attention 的核心计算流程，即先计算 Query 与 Key 的相似度，再进行缩放和 Softmax，最后对 Value 加权求和。在工程实现中，掩码需要与注意力分数的形状正确广播，同时应该避免某一行的所有 Key 都被屏蔽，否则可能产生无效的注意力分布。

为了增强模型对不同关系模式的表达能力，Transformer 模型还将单个 Attention 扩展为多头注意力（Multi-Head Attention）。其基本思路是将隐藏表示投影到多个子空间，分别计算注意力，再将各个注意力头的输出拼接，并通过输出投影融合。假设注意力头数为 H，通常有 $d_k = d_v = d_{\text{model}}/H$，则第 h 个头的输出可表示为

$$\text{head}_h = \text{Attention}\left(\mathbf{Q}\mathbf{W}_Q^{(h)}, \mathbf{K}\mathbf{W}_K^{(h)}, \mathbf{V}\mathbf{W}_V^{(h)}\right) \tag{12.127}$$

最终输出为

$$\text{MHA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_H)\mathbf{W}_O \tag{12.128}$$

因此，不同注意力头能够学习不同的特征交互模式。例如，在推荐精排模型中，某些注意力头可能更加关注用户与物品属性之间的关系，而另一些注意力头则可能关注上下文特征与物品类别之间的关系。需要注意的是，这只是对模型可能学习到的行为模式的直观解释，并不意味着每个注意力头会天然对应一种固定的语义。

下面我们给出多头注意力机制的核心实现。该实现同时支持 Self-Attention 和 Cross-Attention。当 Query、Key 和 Value 来自同一输入时，它是 Self-Attention；当 Query 与 Key、Value 来自不同输入时，则可以用于目标注意力或编码器—解码器交互。

代码 12.2: 基于 PyTorch 手写 Multi-Head Attention

```
class MultiHeadAttention(nn.Module):
    def __init__(self, d_model, num_heads, dropout=0.0):
        super().__init__()
        if d_model % num_heads != 0:
            raise ValueError("d_model must be divisible by num_heads")

        self.d_model = d_model
        self.num_heads = num_heads
        self.d_head = d_model // num_heads
        self.dropout = dropout

        self.q_proj = nn.Linear(d_model, d_model)
        self.k_proj = nn.Linear(d_model, d_model)
        self.v_proj = nn.Linear(d_model, d_model)
        self.out_proj = nn.Linear(d_model, d_model)

    def _split_heads(self, x):
        # [B, L, D] -> [B, H, L, D/H]
        batch, length, _ = x.shape
        x = x.view(batch, length, self.num_heads, self.d_head)
        return x.transpose(1, 2)

    def forward(self, query, key, value, key_padding_mask=None, attn_mask=None, is_causal=False):
```

```
        # key_padding_mask: [B, Lk], True means padding
        # attn_mask: [Lq, Lk] or a broadcastable boolean mask
        q = self._split_heads(self.q_proj(query))
        k = self._split_heads(self.k_proj(key))
        v = self._split_heads(self.v_proj(value))

        batch, _, lq, _ = q.shape
        lk = k.size(-2)
        allowed = torch.ones(lq, lk, dtype=torch.bool, device=q.device)

        if is_causal:
            if lq != lk:
                raise ValueError("This example uses causal masking for self-attention")
            allowed = allowed & torch.tril(allowed)

        if attn_mask is not None:
            allowed = allowed & attn_mask.to(device=q.device, dtype=torch.bool)

        allowed = allowed[None, None, :, :]

        if key_padding_mask is not None:
            valid_keys = ~key_padding_mask
            allowed = allowed & valid_keys[:, None, None, :]

        out, weights = scaled_dot_product_attention(
            q, k, v, allowed_mask=allowed, dropout_p=self.dropout, training=self.training
        )

        out = out.transpose(1, 2).contiguous()
        out = out.view(batch, lq, self.d_model)
        return self.out_proj(out), weights
```

上述代码将注意力头合并到 batch 维度之后的独立维度中，统一执行矩阵乘法，因此不需要逐个注意力头编写循环。需要注意的是，实际项目中应该进一步检查掩码的广播规则，并确保填充位置不会导致某些查询位置对应的所有 Key 均被屏蔽。对于生产环境来说，通常优先使用 PyTorch 或者 TensorFlow 框架提供的优化 Attention 算子，手写版本主要用于理解运算过程、面试练习以及调试特定的网络结构。

除了 Self-Attention 机制之外，多任务推荐和精排模型中还经常使用目标注意力（Target Attention）。它的目的不是让历史行为彼此交互，而是让当前目标物品从用户历史行为中提取与自身相关的信息。其基本结构可以看成一种特殊的 Cross-Attention，即目标物品生成 Query，历史行为生成 Key 和 Value，通过注意力权重对历史行为进行聚合。假设目标物品的表示为 $\mathbf{e}_t$，用户历史行为序列表示为 $\{\mathbf{e}_1, \ldots, \mathbf{e}_L\}$。将目标物品映射为 Query，将

历史行为映射为 Key 和 Value 之后，可以写为

$$
\begin{aligned}
\mathbf{q}_t &= \mathbf{W}_Q \mathbf{e}_t, \\
\mathbf{k}_i &= \mathbf{W}_K \mathbf{e}_i, \\
\mathbf{v}_i &= \mathbf{W}_V \mathbf{e}_i, \\
\alpha_i &= \mathrm{softmax}_i\left(\frac{\mathbf{q}_t^\top \mathbf{k}_i}{\sqrt{d_k}}\right), \\
\mathbf{h}_t &= \sum_{i=1}^{L} \alpha_i \mathbf{v}_i
\end{aligned}
\tag{12.129}
$$

由此得到的 $\mathbf{h}_t$ 是与当前目标物品相关的历史行为表示，可以与用户属性、物品属性及上下文等特征拼接，输入后续的 CTR 预测网络。

需要说明的是，推荐系统中的 Target Attention 并不一定采用 Transformer 中的缩放点积形式。例如，DIN 模型使用局部激活单元，根据目标物品与历史行为的组合特征计算相关性分数，再对历史行为表示进行加权求和[25]。这种方式能够直接将目标物品、历史行为及二者之间的差异和交互作为打分网络的输入，因此不必局限于简单的向量内积。

下面给出一个 DIN 风格的目标注意力实现。该实现使用目标物品与历史行为的拼接、差值和逐元素乘积构造匹配特征，通过 MLP 生成注意力权重。它与缩放点积 Cross-Attention 的区别在于，注意力打分函数本身是一个可学习的非线性网络。

代码 12.3: 基于 PyTorch 手写 Target Attention

```
class TargetAttention(nn.Module):
    def __init__(self, d_model, hidden_dim):
        super().__init__()
        self.score_net = nn.Sequential(
            nn.Linear(4 * d_model, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, 1)
        )

    def forward(self, target, history, valid_mask=None):
        # target: [B, D]
        # history: [B, L, D]
        target = target.unsqueeze(1).expand_as(history)

        features = torch.cat([target, history, target - history, target * history], dim=-1)
        scores = self.score_net(features).squeeze(-1)

        if valid_mask is not None:
            scores = scores.masked_fill(~valid_mask, torch.finfo(scores.dtype).min)

        weights = torch.softmax(scores, dim=-1)

        if valid_mask is not None:
            weights = weights * valid_mask.to(weights.dtype)
            weights = weights / weights.sum(dim=-1, keepdim=True).clamp_min(1e-8)

        pooled = torch.sum(history * weights.unsqueeze(-1), dim=1)
```

```
        return pooled, weights
```

在推荐精排模型中，Target Attention 机制与 Self-Attention 机制可以互补使用。前者根据当前候选物品提取与目标相关的用户兴趣，后者则能够建模历史行为内部的依赖关系。对于历史行为较长、用户兴趣变化较明显的场景，可以先用 Self-Attention 编码行为序列，再通过 Target Attention 从编码后的行为表示中提取目标相关兴趣。这样，精排模型既能学习行为之间的关系，也能针对不同候选物品形成不同的兴趣表示。

12.4.3.3 Masked Self-Attention、Transformer Encoder 与 Decoder

在标准的 Self-Attention 中，每个 Token 可以访问输入序列中的所有位置。这适用于对完整输入进行编码的任务，但如果模型需要根据历史信息预测下一个 Token，就必须避免当前位置访问未来信息。Masked Self-Attention（带掩码的自注意力）正是为此而设计的。

对于长度为 L 的序列，**因果掩码（Causal Mask）**只允许位置 i 关注位置不大于 i 的 Token。假设注意力分数为 $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top/\sqrt{d_k}$，则因果掩码可以写为

$$M_{ij} = \begin{cases} 0, & j \le i, \\ -\infty, & j > i \end{cases} \tag{12.130}$$

将该掩码加到注意力分数上，便有了掩码注意力机制，即：

$$\text{MaskedAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}} + \mathbf{M}\right)\mathbf{V} \tag{12.131}$$

这样序列中位置 i 的输出只能依赖当前位置及之前的输入，而无法使用未来位置的信息。这是自回归语言模型逐 Token 生成的基础，也是 SASRec 等采用因果序列建模方式的推荐模型可以借鉴的机制。

下面我们在前面多头注意力代码的基础上，给出 Masked Self-Attention 的代码实现。其中，因果掩码由下三角矩阵来构造，在传入多头注意力之后，未来位置的注意力分数就会被屏蔽。

代码 12.4: 基于 PyTorch 手写 Masked Self-Attention

```
class MaskedSelfAttention(nn.Module):
    def __init__(self, d_model, num_heads, dropout=0.0):
        super().__init__()
        self.attn = MultiHeadAttention(d_model=d_model, num_heads=num_heads, dropout=dropout)

    def forward(self, x, key_padding_mask=None):
        length = x.size(1)
        causal_mask = torch.tril(torch.ones(length, length, dtype=torch.bool, device=x.device))

        return self.attn(
            query=x, key=x, value=x, key_padding_mask=key_padding_mask, attn_mask=causal_mask
        )
```

需要注意区分因果掩码（Causal Mask）与填充掩码（Padding Mask）。因果掩码用于限制当前位置不能访问未来信息；而填充掩码则用于屏蔽批次中为了对齐长度而补充的无效位置。两种掩码可以同时使用。尤其需要注意，并非所有推荐序列模型都必须使用因果掩码。如果推荐模型的任务是根据完整的历史行为序列预测当前候选物品，那么所有历史行为都已经发生，通常可以让这些行为进行充分交互。只有当训练目标要求严格模拟逐步生成、预测下一个行为，或者明确限制未来信息时，才需要采用因果约束。如果在需要让行为进行充分交互的场景下加入了因果掩码，可能也会让模型的效果反而变差。

除了上面提到的各种掩码之外，Transformer 模型通常会采用 Encoder、Decoder 或者二者组合的架构。Encoder 主要对完整输入序列进行上下文编码，使用不带因果限制的 Self-Attention。而 Decoder 则通常用于自回归生成，

包含 Masked Self-Attention，并可以通过 Cross-Attention 读取 Encoder 的输出。对于纯粹的 CTR 预估或固定候选集合的精排任务，通常不需要完整的 Encoder–Decoder 架构，往往只需要 Encoder 或者经过改造的注意力模块。对于下一物品预测和生成式推荐任务，因果 Decoder 则具有更直接的应用价值。

一个 Encoder Layer 通常包含 Self-Attention 和 FFN 两个子层，两者都通过残差连接与 Layer Normalization 进行组合。为便于理解，下面给出 Pre-LN 结构的核心实现，如代码12.5所示。

代码 12.5: Transformer Encoder Layer 的核心实现

```
class FeedForward(nn.Module):
    def __init__(self, d_model, d_ff, dropout=0.0):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(d_model, d_ff),
            nn.GELU(),
            nn.Dropout(dropout),
            nn.Linear(d_ff, d_model)
        )

    def forward(self, x):
        return self.net(x)


class TransformerEncoderBlock(nn.Module):
    def __init__(self, d_model, num_heads, d_ff, dropout=0.0):
        super().__init__()
        self.norm1 = nn.LayerNorm(d_model)
        self.norm2 = nn.LayerNorm(d_model)
        self.attn = MultiHeadAttention(d_model, num_heads, dropout)
        self.ffn = FeedForward(d_model, d_ff, dropout)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x, key_padding_mask=None):
        h = self.norm1(x)
        attn_out, _ = self.attn(h, h, h, key_padding_mask=key_padding_mask)
        x = x + self.dropout(attn_out)
        x = x + self.dropout(self.ffn(self.norm2(x)))
        return x
```

Decoder Layer 在此基础上增加了 Masked Self-Attention 和 Cross-Attention。前者让解码位置只能读取已经生成的历史信息，后者则让 Decoder 从 Encoder 输出中提取与当前生成位置相关的上下文信息。其核心代码如下。

代码 12.6: Transformer Decoder Layer 的核心实现

```
class TransformerDecoderBlock(nn.Module):
    def __init__(self, d_model, num_heads, d_ff, dropout=0.0):
        super().__init__()
        self.norm1 = nn.LayerNorm(d_model)
        self.norm2 = nn.LayerNorm(d_model)
        self.norm3 = nn.LayerNorm(d_model)
```

```
        self.self_attn = MultiHeadAttention(d_model, num_heads, dropout)
        self.cross_attn = MultiHeadAttention(d_model, num_heads, dropout)
        self.ffn = FeedForward(d_model, d_ff, dropout)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x, memory, target_padding_mask=None, memory_padding_mask=None):
        # Causal self-attention over decoder inputs.
        h = self.norm1(x)
        attn_out, _ = self.self_attn(h, h, h, key_padding_mask=target_padding_mask, is_causal=True)
        x = x + self.dropout(attn_out)

        # Cross-attention to encoder outputs.
        h = self.norm2(x)
        cross_out, _ = self.cross_attn(h, memory, memory, key_padding_mask=memory_padding_mask)
        x = x + self.dropout(cross_out)

        x = x + self.dropout(self.ffn(self.norm3(x)))
        return x
```

这里的 Decoder 代码展示的是核心子层的组织方式，并不是完整的生成模型。实际使用时还需要将 Token 嵌入和位置表示加入输入，堆叠多个 Decoder Layer，并根据任务设置输出投影层、训练目标及推理策略。生成式推荐模型还需要根据物品标识、行为事件或其他推荐对象定义 Token 的表示方式。

12.4.3.4 位置编码与 RoPE

Self-Attention 本身主要根据 Token 内容之间的相似度计算权重，并不会自动感知输入序列的先后顺序。如果将相同的一组 Token 重新排列，且不引入任何与位置相关的信息，标准 Self-Attention 的输出也会随 Token 一同排列，而不会天然理解这种排列变化所代表的时间顺序。因此，对于用户历史行为等具有明确时序语义的输入，需要通过位置编码或其他相对位置信息机制将顺序关系也引入模型。

最经典的位置编码方式是绝对位置编码，包括可学习的位置嵌入和固定的正弦余弦位置编码。以正弦余弦编码为例，位置 p 在第 $2i$ 维和第 $2i+1$ 维的编码分别为

$$\begin{aligned} \mathrm{PE}(p, 2i) &= \sin\left(\frac{p}{10000^{2i/d_{\text{model}}}}\right), \\ \mathrm{PE}(p, 2i+1) &= \cos\left(\frac{p}{10000^{2i/d_{\text{model}}}}\right) \end{aligned} \tag{12.132}$$

绝对位置编码通常直接与 Token 嵌入相加，使网络能够同时利用内容信息和位置信息。另一种思路是在注意力分数中直接添加相对位置偏置，即根据两个 Token 的相对位置调整它们之间的注意力权重：

$$\mathrm{Attention} = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\top}}{\sqrt{d_k}} + \mathbf{B}_{\text{rel}}\right)\mathbf{V} \tag{12.133}$$

其中，$\mathbf{B}_{\text{rel}}$ 表示由相对位置关系产生的偏置矩阵。这类方法更直接地将距离或相对顺序引入注意力计算。

RoPE（Rotary Position Embedding，旋转位置编码）是另一类重要的位置编码方法，由苏剑林在《Roformer: Enhanced Transformer with Rotary Position Embedding》的论文中提出 [18]。它并不是简单地把位置向量加到 Token 嵌入上，而是在计算注意力之前，根据 Token 的位置对 Query 和 Key 的向量分量进行旋转。其一个重要性质是，经过旋转后的 Query 与 Key 的内积能够显式体现两个位置之间的相对距离。

为了说明这一机制，我们考虑每两个相邻维度构成一个二维向量。对位置为 p 的 Query 施加如下的旋转矩阵：

$$\mathbf{R}_{p,\theta} = \begin{bmatrix} \cos(p\theta) & -\sin(p\theta) \\ \sin(p\theta) & \cos(p\theta) \end{bmatrix} \tag{12.134}$$

对于多维向量而言，RoPE 将不同二维分量对应不同的旋转频率，并将各个旋转组合起来。对于位置 m 的 Query 与位置 n 的 Key，注意力打分中的内积可以写为

$$(\mathbf{R}_m\mathbf{q}_m)^\top(\mathbf{R}_n\mathbf{k}_n) = \mathbf{q}_m^\top\mathbf{R}_{n-m}\mathbf{k}_n \tag{12.135}$$

由此可见，虽然旋转操作依赖绝对位置，但两者的内积能够体现相对位置关系。这使得 RoPE 能够在保留向量内容信息的同时，将顺序信息融入到注意力分数中。RoPE 通常只作用于 Query 和 Key，而不直接旋转 Value。下面的代码12.7给出了 ROPE 的 PyTorch 实现。这里代码假设输入形状为 $[B, H, L, d]$，其中每个注意力头的维度 d 必须是偶数。

代码 12.7: RoPE 旋转位置编码的核心实现

```python
def apply_rope(x, base=10000.0):
    # x: [B, H, L, D], D must be even
    batch, heads, length, dim = x.shape
    if dim % 2 != 0:
        raise ValueError("RoPE requires an even head dimension")

    device = x.device
    dtype = x.dtype
    positions = torch.arange(length, device=device, dtype=torch.float32)
    inv_freq = 1.0 / (base ** (torch.arange(0, dim, 2, device=device).float() / dim))
    angles = positions[:, None] * inv_freq[None, :]
    angles = torch.repeat_interleave(angles, 2, dim=-1)

    cos = angles.cos().to(dtype)[None, None, :, :]
    sin = angles.sin().to(dtype)[None, None, :, :]

    pairs = x.reshape(batch, heads, length, dim // 2, 2)
    even = pairs[..., 0]
    odd = pairs[..., 1]
    rotated = torch.stack((-odd, even), dim=-1).flatten(-2)

    return x * cos + rotated * sin
```

在实际的 Transformer 模型中，通常会先通过线性投影得到 Query 和 Key，再分别应用 RoPE，最后计算注意力分数。需要注意的是，以上代码省略了对变长序列位置编号、缓存推理位置偏移及特定模型旋转维度配置的处理，适合用于理解 RoPE 的基本计算机制。在实际工程部署时，通常还需要保证训练和推理采用一致的位置编码策略。

此外，位置编码的选择需要结合输入数据的语义特点。在用户行为序列中，行为的时间顺序通常具有明确意义，因此绝对位置编码、相对位置编码和 RoPE 都可能有助于建模行为之间的依赖关系。对于由离散特征字段组成的精排输入，字段本身的语义通常比字段排列顺序更重要，因此应优先通过字段嵌入或字段类型标识区分不同特征。

笔记 在实际工业级推荐系统中，我们还可以尝试直接将用户历史行为的时间信息引入 Attention 分数，而不一定依赖传统的位置 Embedding。例如，对于当前请求的目标表示 $\mathbf{q}$ 和第 i 个历史行为的表示 $\mathbf{k}_i$，假设当前请求时间为 t_{req}，历史行为发生时间为 t_i，则二者之间的时间间隔为

$$\Delta t_i = t_{\text{req}} - t_i \tag{12.136}$$

在标准 Attention 分数的基础上，可以直接加入时间间隔对应的偏置项：

$$s_i = \frac{\mathbf{q}^\top \mathbf{k}_i}{\sqrt{d_k}} + b(\Delta t_i),$$
$$\alpha_i = \mathrm{softmax}_i(s_i) \tag{12.137}$$

其中，$b(\Delta t_i)$ 用于刻画历史行为与当前请求之间的时间间隔对相关性的影响。例如，可以将时间间隔进行分桶后设置对应的偏置，也可以通过对 $\log(1+\Delta t_i)$ 等时间特征进行变换，计算相应的时间偏置。对于绝对时间戳，还可以将行为发生的时刻转换为小时、星期等具有实际含义的时间特征，并进一步加入注意力分数：

$$s_i = \frac{\mathbf{q}^\top \mathbf{k}_i}{\sqrt{d_k}} + \mathbf{w}_t^\top \phi(t_i) \tag{12.138}$$

其中，$\phi(t_i)$ 表示从时间戳中提取的时间特征，$\mathbf{w}_t$ 为对应的可学习参数。上述公式只是将时间信息融入 Attention 的一种简单方式，具体实现也可以采用更复杂的时间偏置函数。

在实际工程中，直接引入时间戳或时间间隔特征，有时反而比采用基于可学习 Embedding 的位置编码取得更好的线上效果。这可能是因为时间间隔能够更直接地反映用户兴趣的新近程度，而绝对时间特征则能够刻画时段、星期等行为规律；另一方面，可学习的位置 Embedding 也可能因数据分布、训练充分程度或模型结构等因素，未能有效提取所需的时序信息。因此，位置编码并不存在适用于所有推荐场景的固定最优方案，不应仅根据某种方法在理论上的优势就预设其一定有效，而应结合具体业务的数据分布，通过线上 A/B 实验验证不同方案的实际收益。

12.4.3.5 Attention 的改进：稀疏注意力与 FlashAttention

虽然 Attention 具有较强的关系建模能力，但标准 Self-Attention 也存在明显的计算和内存开销。对于长度为 L 的序列，注意力矩阵的形状为 $L \times L$，其计算复杂度通常为 $O(L^2 d)$，其中 d 表示每个注意力头或相应投影的特征维度。仅保存注意力权重矩阵，就需要与 L^2 成正比的内存空间。因此，当输入是包含大量历史行为的长序列时，Attention 可能成为训练和推理中的主要开销之一。

一种思路是减少需要计算的 Token 对。稀疏注意力（Sparse Attention）通过对允许交互的位置施加结构约束，使注意力矩阵不再需要完整计算。例如，局部窗口注意力仅允许每个 Token 关注附近的若干位置；分块稀疏注意力将序列划分为多个块，只计算指定块之间的注意力；还可以结合少量全局 Token 或特定路由规则，保留部分远距离信息交互。如果每个 Token 只关注固定数量的邻近位置，局部注意力的计算量可以从标准注意力的二次增长降低到与序列长度近似线性相关的水平。不过，稀疏模式也可能遗漏对预测有用的远距离依赖，因此需要根据行为序列的特点设计注意力连接规则。

另一类方法不改变标准 Attention 的数学结果，而是改进其计算实现。FlashAttention 正是这种思路的代表 [7]。传统实现通常会在 GPU 显存中显式构造完整的注意力分数矩阵，再执行 Softmax 和 Value 加权求和。FlashAttention 则利用分块计算，将 Query、Key 和 Value 的部分数据加载到更快的片上存储中，分块完成注意力计算，并通过在线归一化避免将完整的注意力矩阵写入显存。此外，FlashAttention-2 进一步优化了线程块和 Warp 的任务划分，改善并行利用率与计算效率 [6]。FlashAttention 的关键优势在于减少 GPU 高带宽显存与片上存储之间的数据传输，从而改善长序列注意力的内存使用与运行效率。需要强调的是：

笔记 FlashAttention 在数学上计算的是精确 Attention，并不是通过近似或者丢弃注意力连接来降低模型表达能力。它主要减少的是中间矩阵的物化以及显存访问开销。

因此，稀疏注意力和 FlashAttention 解决的是不同层面的问题。前者改变允许参与计算的 Token 关系，以减少需要执行的注意力运算；后者尽可能保留完整的注意力关系，通过更高效的底层实现减少数据搬运和中间内存占用。在推荐系统中，如果用户行为序列非常长，可以根据业务特点考虑局部或分块稀疏注意力；如果仍需要完整建模较长序列的关系，则可以优先使用高效的精确 Attention 实现。当然，上述两种方法也可以结合使用。关于 Attention 机制更多的改进方案以及在推荐大模型中的具体应用，我们将会在后续第36章《推荐大模型》中进一步展开介绍与讨论。

最后，需要注意的是，稀疏注意力是否真正带来加速，除了取决于理论计算量，还取决于稀疏模式能否映射

为高效的 GPU 运算。如果稀疏索引和数据搬运开销过大，实际运行时间可能并不会随着理论计算量同步下降。因此，工业级推荐系统需要同时考察模型指标、显存占用、训练吞吐量和端到端延迟等方面，综合评估 Attention 的优化效果。

12.4.3.6 Transformer 在推荐精排特征交互中的应用

Transformer 相关机制在推荐系统中的应用主要包括特征交互建模、目标相关兴趣提取和用户行为序列建模等方向。不同应用虽然都可以使用 Attention，但其输入表示和交互方式有所区别。例如，AutoInt 模型将不同特征字段的嵌入视为 Token，通过 Self-Attention 学习特征之间的交互关系 [17]；DIN 模型根据当前候选物品与用户历史行为之间的关联，通过 Target Attention 提取与目标相关的兴趣表示 [25]；BST 模型则利用 Transformer 对用户历史行为序列进行编码，以捕获不同行为之间的依赖关系 [3]。此外，SASRec 模型采用带因果掩码的 Self-Attention，根据用户历史交互预测后续可能交互的物品 [10]。

这些模型体现了 Attention 在推荐系统中的不同作用，其中 Self-Attention 可以建模特征字段或历史行为之间的关系，而 Target Attention 则能够根据当前候选物品动态提取相关兴趣。对于工业级精排模型而言，可以根据输入数据的组织形式和预测目标选择合适的交互机制，而不必将所有特征统一采用相同的注意力结构。除了 SASRec 模型在前面第10章《召回模型》中进行过介绍，其他基于 Transformer 架构模型的具体架构将在后续章节分别介绍：AutoInt 模型将在第15章《特征工程》中展开，DIN 模型和 BST 模型将在第20章《用户兴趣与序列建模》讨论。

随着推荐模型进一步向大规模统一建模发展，Transformer 也开始用于整合用户行为、物品和上下文等多种信息。HSTU 等模型进一步探索了面向大规模推荐的序列转换架构，为生成式推荐提供了新的建模思路 [23]。相关内容将在第35章《生成式推荐》中进一步介绍。

12.4.3.7 Transformer 与 MoE 的结合

随着 Transformer 模型的规模不断增大，如何在扩大模型参数量的同时控制单次计算开销，成为大规模模型设计的重要问题。标准 Transformer 的主要计算模块包括 Attention 和 FFN，其中 FFN 对每个 Token 独立执行相同的非线性变换。如果每层都采用稠密的 FFN，那么每个 Token 都需要经过同一组完整的前馈网络参数，模型的参数量和计算量会随着隐藏维度及网络层数同步增长。

MoE 为这一问题提供了另一种架构设计思路。它不再要求所有输入都通过同一个稠密 FFN，而是将前馈网络拆分为多个专家（Expert），再通过路由网络（Router）为每个 Token 选择其中一部分专家参与计算。这样可以在保持单个 Token 计算量相对可控的情况下，扩大模型能够容纳的总参数量。

在 Transformer 模型中，MoE 模块通常主要替换原有的 FFN 模块。假设原始 FFN 为

$$\mathrm{FFN}(\mathbf{x}) = \mathbf{W}_2\phi(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 \tag{12.139}$$

将其替换为 MoE 后，可以得到

$$\mathrm{MoE}(\mathbf{x}) = \sum_{i=1}^{N} g_i(\mathbf{x}) E_i(\mathbf{x}) \tag{12.140}$$

其中，E_i 表示第 i 个专家，N 为专家总数，$g_i(\mathbf{x})$ 表示路由网络为该专家分配的权重。如果所有专家都参与运算，这仍然属于稠密的专家融合，并不能够充分发挥出稀疏 MoE 的计算优势。因此，实际的大规模 MoE 通常采用 Top-K 路由，只选择得分最高的 K 个专家参与计算。假设路由网络的输出分数为

$$\begin{aligned} \mathbf{r}(\mathbf{x}) &= \mathbf{W}_r\mathbf{x} \\ \mathbf{p}(\mathbf{x}) &= \mathrm{softmax}(\mathbf{r}(\mathbf{x})) \end{aligned} \tag{12.141}$$

其中，$\mathbf{W}_r$ 为路由网络参数，$\mathbf{p}(\mathbf{x})$ 表示各专家的路由概率。令 $\mathcal{T}_K(\mathbf{x})$ 表示得分最高的 K 个专家的集合，则稀疏 MoE 的输出可以表示为

$$\mathrm{MoE}(\mathbf{x}) = \sum_{i \in \mathcal{T}_K(\mathbf{x})} \widetilde{g}_i(\mathbf{x}) E_i(\mathbf{x}) \tag{12.142}$$

其中，$\tilde{g}_i(\mathbf{x})$ 表示在选中专家集合上归一化后的路由权重。

上述这种结构将原本统一的前馈变换转化为输入条件化的专家选择过程。不同 Token 可以根据自身表示被路由到不同的专家，从而形成不同的非线性变换路径。专家网络可以在训练中逐渐学习不同的模式，而路由网络负责决定每个 Token 应当使用哪些专家。在推荐系统中，MoE 的价值同样可以从条件化计算和模型容量两个角度理解。用户、物品及上下文的组合具有较强的异质性，不同用户群体、内容类型和行为场景可能需要不同的特征变换。通过专家路由，模型可以根据输入的特征表示，动态选择更适合当前样本的网络分支。同时，MoE 也能够扩大模型的参数容量，而不需要让每个样本都经过所有专家。

不过，稀疏 MoE 模块并不一定会降低整个模型的线上推理耗时。虽然单个 Token 每次只激活部分专家，但路由、数据分发、专家间通信以及结果聚合仍然会存在额外的成本。如果专家分布在多个设备上，输入 Token 就需要根据路由结果发送到不同设备，通信开销甚至可能成为模型推理的主要瓶颈。此外，在工业级精排模块中，单次请求通常需要对大量候选物品执行精排模型的计算，batch 大小、专家负载以及机器的 CPU 或 GPU 利用率等都会影响真实的推理成本。因此，在采用 MoE 架构时，需要同时考虑参数容量、激活计算量、路由开销和部署复杂度。

从模型表示的角度看，MoE 也有助于理解 Transformer 中 FFN 的作用。Geva 等人在论文《Transformer Feed-Forward Layers Are Key-Value Memories》[8] 中指出，Transformer 中的 FFN 可以在一定程度上被理解为一种 Key-Value Memory，其中第一层将输入映射到若干中间单元，激活函数决定哪些单元响应当前输入，第二层再将激活结果转换为输出表示。这说明 FFN 不仅仅是一个普通的全连接层，也可以看作一组以输入模式为条件的记忆单元。从图 12.17 可以看出，Transformer 模型中不同层的已激活 Memory 比例整体在 10% 50% 之间，但随着网络层数的加深，尤其是网络层数超过 10 层之后，已激活的 Memory 比例呈现显著下降的趋势，逐步低于 20% 甚至接近 10%，这表明深层 FFN 的激活单元更加稀疏，网络在高层对输入模式的响应更加具有选择性。

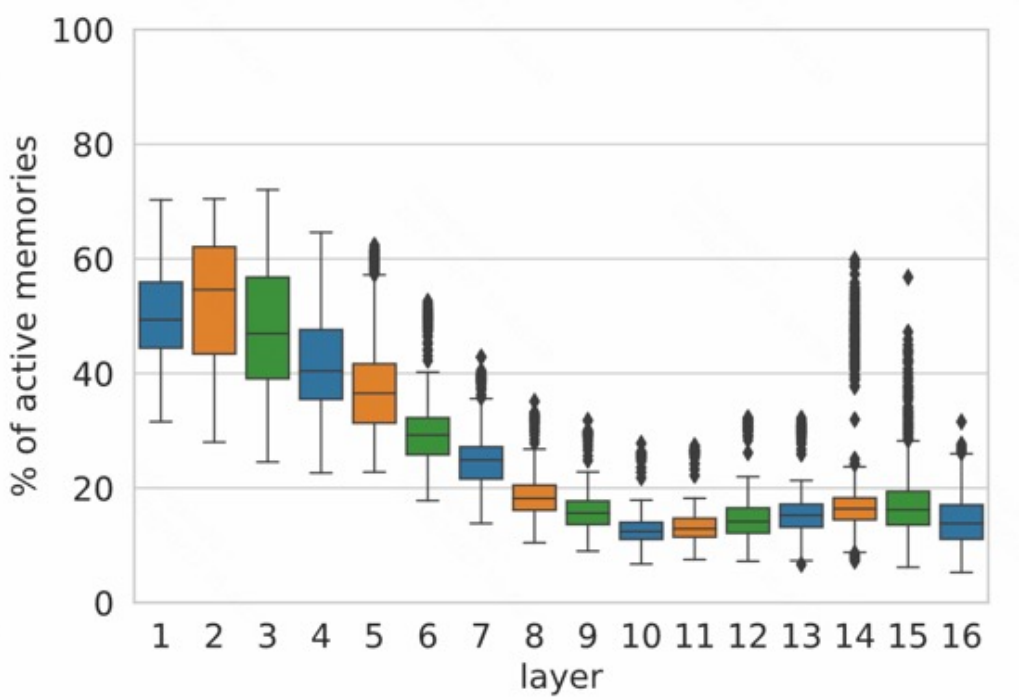


图 12.17: Transformer 中 FFN 的已激活 Memory 比例。

Transformer 中的 FFN 可以当做 Key-Value Memory 这一观点为理解专家专门化提供了有价值的视角。当不同输入激活不同的中间单元时，网络能够学习到不同的输入模式与输出变换之间的关联。而 MoE 则进一步将部分非线性变换划分为独立的专家，并通过显式路由选择参与计算的参数。但需要注意的是，普通稠密的 FFN 即使存在大量未激活的中间神经元，底层矩阵乘法也通常仍然会执行，因此激活稀疏本身并不会自动带来计算加速。只有当实现能够利用稀疏性减少实际执行的运算，或者采用稀疏专家路由减少被调用的网络分支时，才

有机会将这种稀疏性转化为加速计算的收益。

12.4.3.8 DeepSeekMoE：细粒度专家划分与共享专家

DeepSeekMoE 模型源自于深度求索公司发表在 ACL 2024 会议的论文《DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models》[5]。该方法针对传统 MoE 模型中的专家专门化和知识冗余问题，提出细粒度专家划分与共享专家两项核心设计，希望在控制激活计算量的同时，提高专家组合的灵活性，并减少不同专家之间不必要的重复学习。

在传统 MoE 架构中，模型通常包含若干规模相对较大的专家，每个 Token 从中选择 Top-K 个专家。这样的结构虽然支持条件化计算，但仍然面临两个潜在问题。第一个问题是**知识混合性（Knowledge Hybridity）**。当每个专家具有较大的参数规模时，同一个专家可能需要同时学习多种不同的输入模式，难以在训练过程中形成更加集中的功能分工。由于专家的使用方式受路由网络影响，部分专家可能长期承担多种不同类型的计算任务，导致专家内部包含多类混合的知识模式。第二个问题是**知识冗余（Knowledge Redundancy）**。不同 Token 通常都需要部分共同的基础知识，例如常见模式、通用特征变换或跨任务共享的表示。如果所有知识都分散在相互独立的专家中，每个专家可能都需要重复学习这些通用模式，从而产生参数冗余。这不仅会降低参数利用效率，也可能使不同专家难以充分形成独立的功能分工。

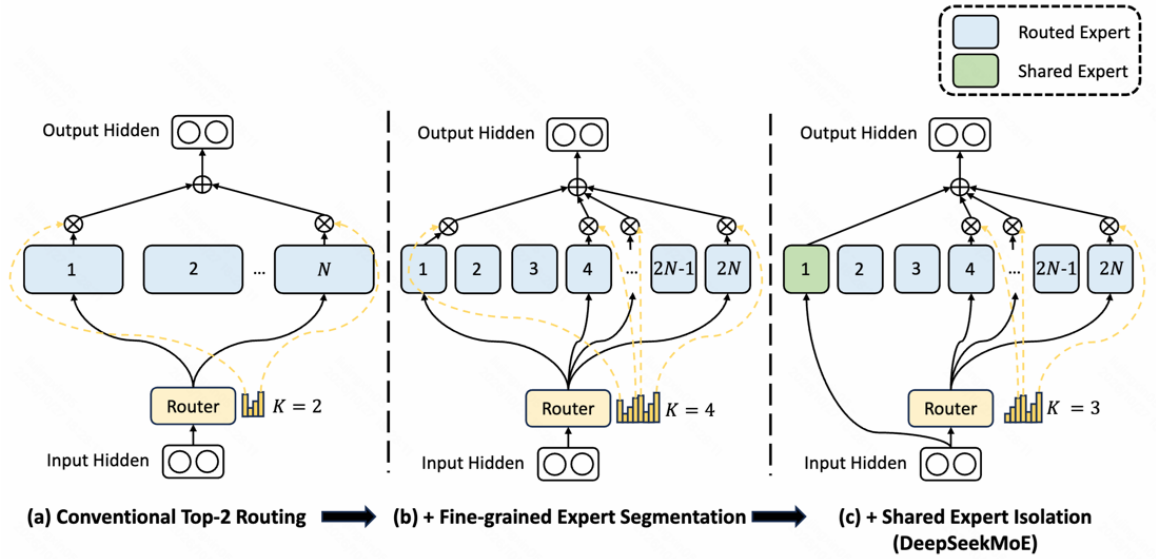


图 12.18: DeepSeekMoE 模型的网络结构。

DeepSeekMoE 模型分别针对上述问题进行了改造，如图12.18所示。首先，它提出细粒度专家划分（Fine-Grained Expert Segmentation），将原有的大型专家进一步拆分为更多、更小的专家，并在单次计算中激活更多的细粒度专家。假设原始结构有 N 个专家，每个 Token 激活 K 个。在细粒度划分之后，将专家数量扩展到约 mN 个，并将单个专家的中间隐藏维度缩小到原来的约 $1/m$，同时将激活数量调整到约 mK 个。这样，在相近的专家激活计算预算下，路由网络可以组合更多的小型专家。

从组合数量上看，原始结构可选择的专家组合数量约为 $\binom{N}{K}$，而细粒度划分之后的组合数量约为 $\binom{mN}{mK}$。后者在通常的参数设置下可以提供更加丰富的组合空间。这里的组合数量只是对路由灵活性的理论刻画，并不意味着每种组合都会在训练中被充分使用，也不意味着专家一定能够自动形成完全独立的专门化能力。它所提供的是更细粒度的参数分工空间，使得模型可以通过组合不同的小型专家适应不同的输入模式。

其次，DeepSeekMoE 模型引入了共享专家（Shared Experts）。共享专家始终参与计算，负责学习多个输入普遍需要的通用知识；另外一部分专家则由路由网络动态选择，负责学习更具差异性的模式。如果共享专家数量

为 K_s，路由专家集合为 $\mathcal{R}(\mathbf{x})$，我们就可以将其输出概括为

$$\mathrm{MoE}(\mathbf{x}) = \sum_{i=1}^{K_s} E_i^{\mathrm{shared}}(\mathbf{x}) + \sum_{j\in\mathcal{R}(\mathbf{x})} g_j(\mathbf{x}) E_j^{\mathrm{routed}}(\mathbf{x}) \tag{12.143}$$

其中，共享专家用于提取常见的通用模式，而路由专家则会根据输入进行动态选择。因此，DeepSeekMoE 模型不必在每个路由专家中重复存储所有通用知识，而是可以将共享信息集中交由共享专家学习，同时为其他专家提供更加明确的专门化空间。

这一结构可以理解为对专家内部知识分工的进一步约束，即：

定义 12.11

共享专家学习更普遍的模式，路由专家则集中学习具有差异性的模式。

在工业级推荐场景中，这一思路同样具有参考价值。例如，用户、物品和上下文之间可能存在所有样本都需要的基础交互，同时不同内容类别、用户行为或者业务目标还需要专属的特征变换。共享专家可以用于承载通用的表示，而路由专家则有机会学习到更加细分的行为与内容模式。不过，推荐数据的特征结构与语言模型中的 Token 分布并不相同，这种设计能否带来收益仍需要根据任务特点进行验证。

DeepSeekMoE 模型的另一个重要工程问题是负载均衡。由于路由网络会根据输入动态选择专家，如果大量 Token 长期集中到少数专家，可能导致这些专家负载过高，而其他专家获得的训练信号不足。这种现象通常称为**路由坍缩或负载不均衡**，会同时影响专家的专门化能力、训练效率以及分布式执行的效率。

为了缓解专家层面的负载不均衡问题，DeepSeekMoE 模型引入了专家级的负载均衡损失。假设一个 batch 包含 T 个 Token，路由专家数量为 N，每个 Token 激活 K 个专家。令 $s_{i,t}$ 表示第 t 个 Token 路由到第 i 个专家的概率，$\mathbb{I}_{i,t}$ 表示该 Token 是否选择了专家 i，则可以定义

$$f_i = \frac{N}{KT}\sum_{t=1}^{T}\mathbb{I}_{i,t}, \qquad P_i = \frac{1}{T}\sum_{t=1}^{T} s_{i,t} \tag{12.144}$$

其中，f_i 描述归一化后的实际路由频率，P_i 描述专家在门控概率分布中的平均权重。一个简化的专家级负载均衡损失可以写为

$$\mathcal{L}_{\mathrm{ExpBal}} = \alpha_1 \sum_{i=1}^{N} f_i P_i \tag{12.145}$$

其中，α_1 为负载均衡损失的权重。该目标通过概率分布与实际路由频率之间的耦合，抑制少数专家长期获得过多路由流量的情况。这里采用的是便于说明机制的写法，在实际实现中还需要结合共享专家是否计入路由集合、Top-K 权重如何归一化等细节调整计算方式。

此外，在大规模分布式训练中，专家还可能分布在不同计算设备上。即使各个专家之间的路由频率并不完全均匀，只要每台设备承担的总计算负载能够保持合理平衡，整个训练系统就有机会达到更高的吞吐量。因此，我们还可以从设备维度来定义负载均衡损失。假设全部路由专家被划分为 D 个设备组，记第 d 个设备上的专家集合为 $\mathcal{E}_d$，则可以定义：

$$f'_d = \frac{1}{|\mathcal{E}_d|}\sum_{i\in\mathcal{E}_d} f_i, \qquad P'_d = \sum_{i\in\mathcal{E}_d} P_i \tag{12.146}$$

并构造如下所示的设备级损失：

$$\mathcal{L}_{\mathrm{DevBal}} = \alpha_2 \sum_{d=1}^{D} f'_d P'_d \tag{12.147}$$

其中，α_2 用于控制设备级均衡的强度。在大规模 MoE 模块的设计中，设备级负载均衡通常需要得到充分的关注，因为设备间的通信和负载不均衡可能会成为模型训练的瓶颈。但如果对单个专家施加太强的负载均衡约束，也可能会限制专家的专门化，导致模型训练效果变差。因此，专家级与设备级的均衡目标需要结合实际系统的通信结构以及专家负载情况来进行权衡。

由此我们可以看出，DeepSeekMoE 模型的设计并不是单纯地增加专家的数量，而是同时优化了专家规模、知

识共享方式以及路由负载的机制。细粒度的专家划分增加了可组合的参数子结构，共享专家减少了通用知识在路由专家中的重复学习，而负载均衡机制则帮助模型在训练和分布式执行中维持了合理的专家利用率。对于推荐精排模型而言，我们更值得借鉴的是 DeepSeekMoE 模型将专家专门化、通用知识共享与执行效率共同考虑的架构思路，从而搭建出训练效率更高、模型容量更大且在分布式环境下负载更均衡的推荐大模型。

12.4.3.9 基于 Transformer 的统一精排架构设计与讨论

从前面小节的内容可以看出，Transformer 为推荐精排模型提供了统一的特征交互机制，但是否能够基于 Transformer 架构构建出高效的工业级推荐精排模型，还取决于如何组合 Token 表示、注意力机制、前馈网络与任务预测层。在一个典型的 Transformer 精排模型中，用户属性、物品属性和上下文可以先编码为字段 Token，历史行为则可以编码为行为 Token。随后，精排模型可以通过 Self-Attention 学习不同特征和行为之间的关联，再结合目标注意力提取当前候选物品相关的用户兴趣，最后对各个 Token 的表示进行汇总，并通过任务塔输出 CTR、CVR 以及其他的排序指标。对于多任务的场景，我们还可以在统一的 Transformer 表示之上设置多个预测头。当模型面临更高的容量需求以及计算效率挑战时，为了在保证计算开销可控的前提下进一步提升模型的表示能力和参数规模，则可以将部分 FFN 模块替换为 MoE 模块，利用稀疏路由的方式来扩展模型的总参数规模。

不过，统一的精排模型架构也并不意味着必须让所有信息经过同一种注意力机制。由于不同输入的语义结构存在明显差异，比如用户行为具有时间顺序，适合采用序列建模机制；而用户、物品和上下文特征可能没有天然的排列顺序，则需要通过字段 Embedding 和特征编码来明确其具体含义。此外，不同的预测任务通常使用不同的监督信号，因此，我们也需要通过多个任务塔等方式来保留不同任务之间的建模差异。因此，工业级精排模型的统一，更应该理解为在统一的表示以及计算框架下来组合不同类型的信息输入。

在模型的计算效率方面，我们还需要区分模型容量与单次请求的实际开销。增加 Transformer 层数、注意力头数和隐藏维度，都可以提高模型的表示能力，但也会增加计算量和显存占用。引入稀疏 Attention 可以减少部分 Token 之间的交互，引入 FlashAttention 可以减少注意力计算中的内存访问开销，而引入 MoE 则可以通过条件化的专家执行扩大参数规模。它们解决的是不同的问题，实际收益也需要分别从预测效果、训练吞吐量、推理延迟以及设备利用率等多个维度来进行评估。

从 Transformer 相关技术的历史研究演进来看，Transformer、RoPE、FlashAttention 以及 DeepSeekMoE 等技术首先在大规模序列建模与大语言模型中得到深入的研究，随后为推荐模型的行为序列编码、特征交互、高效计算以及条件化参数扩展也提供了指导方向。AutoInt、BST 和 SASRec 等基于 Transformer 的推荐模型已经展示了注意力机制在特征交互与行为序列建模这两个方向的重要价值。此外，HSTU、RankMixer 等工作进一步探索了面向大规模推荐的统一序列转换架构。

因此，基于 Transformer 的精排模型可以被视为推荐模型从分散式特征交互与序列建模，逐步走向统一表示学习的一条重要技术路线。它为复杂特征关系、用户行为序列和多任务预测提供了更加灵活的组合方式，也为进一步探索稀疏专家、大规模预训练与统一的推荐模型架构奠定了坚实的基础。不过，这类基于 Transformer 的统一精排模型架构是否真正有效，最终仍然取决于模型能否利用实际数据学到有价值的关系，以及这些收益能否在工业推荐系统中稳定地转化为线上业务效果。

12.5 精排模型的预测目标与损失函数

在完成精排模型的网络架构设计之后，还需要确定模型究竟应该优化什么目标。对于同一批候选物品，不同的训练目标可能使模型学到不同的排序策略。以 CTR 预估为例，模型既需要尽可能准确地估计单个物品被点击的概率，也需要在同一请求的候选集合中，将更有可能被用户点击的物品排在前面。这两个目标相互关联，但并不完全等价。类似地，对于有效播放、观看时长和点赞等目标，除了估计单个物品的反馈概率或反馈值，还需要考虑不同候选物品之间的相对顺序，以及最终排序对用户消费行为的影响。

按照监督信号的组织方式，精排模型的损失函数通常可以分为 Pointwise Loss、Pairwise Loss 和 Listwise Loss 三类。Pointwise Loss 将单个样本作为训练对象，直接约束预测结果与真实标签之间的差异；Pairwise Loss 将同

一请求中的两个候选物品作为训练对象，优化二者的相对排序；Listwise Loss 则将整个候选列表作为训练对象，直接在列表层面建模排序结果与目标排序之间的差异。

这三类损失并不存在绝对的优劣。Pointwise Loss 更适合学习具有明确概率含义的预测目标，Pairwise Loss 更直接地优化候选物品之间的相对顺序，而 Listwise Loss 则能够利用同一请求内完整的候选集合及其标签关系。工业级精排模型通常需要根据训练标签的形式、线上排序策略和业务目标，选择其中一种或多种损失进行组合。

12.5.1 Pointwise Loss

Pointwise Loss 将每个训练样本独立地作为优化对象，直接根据模型预测结果与真实标签之间的差异计算损失。对于 CTR 预估，训练样本通常由一次曝光及其上下文构成，点击标签记为 $y_i \in \{0,1\}$。模型输出一个实数分数 s_i，再通过 Sigmoid 函数将其转换为预测点击概率：

$$\hat{p}_i = \sigma(s_i) = \frac{1}{1+\exp(-s_i)} \tag{12.148}$$

二元交叉熵损失（Binary Cross Entropy，BCE）是 CTR 预估中最常用的 Pointwise Loss，其定义为

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log \hat{p}_i + (1-y_i)\log(1-\hat{p}_i)] \tag{12.149}$$

其中 N 为训练样本数量。对于正样本，模型需要提高预测点击概率；对于负样本，则需要降低预测点击概率。通过对所有样本的交叉熵进行优化，精排模型可以学习输入特征与点击行为之间的关系。

BCE 的一个重要特点是，它不仅关注正负样本之间的相对顺序，还会约束每个样本预测概率的绝对水平。对于单个样本而言，它的损失对预测 Logit s_i 的梯度为

$$\frac{\partial \mathcal{L}_{\text{BCE}}}{\partial s_i} = \frac{1}{N}(\hat{p}_i - y_i) \tag{12.150}$$

所以当真实标签为 1 时，梯度会推动预测概率上升；当真实标签为 0 时，梯度会推动预测概率下降。当然，上述梯度的大小还与预测结果和真实标签之间的差异有关，因此已经正确预测且置信度较高的样本通常只会产生较小的梯度。

在训练数据采样方式与目标分布一致、损失权重设置合理等条件下，BCE 能够有效学习条件点击概率，并为模型输出提供概率校准的基础。这里的概率校准，是指模型预测为某个概率的样本集合，其实际点击比例应当与该概率大致一致。需要注意的是，负样本下采样、正负样本加权和训练数据分布变化等因素仍然可能改变模型输出的概率尺度，因此，即使使用 BCE 的损失函数，也不意味着线上预测值必然完全校准。

通常来说，对于点击、有效播放等二分类目标，BCE 是一种常用的 Pointwise Loss；对于观看时长等连续目标，则可以根据标签分布和业务需求选择不同的回归损失。均方误差（Mean Squared Error，MSE）通过平方误差衡量预测值与真实值之间的差异，其定义为

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \tag{12.151}$$

其中，$\hat{y}_i$ 和 y_i 分别表示第 i 个样本的预测观看时长和真实观看时长。由于误差以平方形式计入损失，MSE 会对较大的预测误差施加更强的惩罚，但也容易受到极端时长样本的影响。Huber Loss 则结合了平方误差和绝对误差的特点，其定义为

$$\mathcal{L}_{\text{Huber}}(e_i) = \begin{cases} \frac{1}{2}e_i^2, & |e_i| \le \delta \\ \delta\left(|e_i| - \frac{1}{2}\delta\right), & |e_i| > \delta \end{cases} \tag{12.152}$$

其中，$e_i = \hat{y}_i - y_i$，δ 为控制两种损失切换的阈值。对于较小误差，Huber Loss 采用平方惩罚；而对于较大误差，Huber Loss 则转为线性惩罚，从而降低极端样本对梯度的影响。除此之外，针对不同的行为反馈，还可以根据标签分布和业务需求设计加权交叉熵、分桶分类损失等形式。它们的共同特点是直接约束单个样本的预测结果，而不是显式地比较同一请求中的不同候选物品。

此外，当训练数据中的正负样本数量存在严重不均衡时，大量容易分类的负样本可能在训练过程中占据较大的损失权重。此时，可以考虑使用 Focal Loss 对 BCE 进行改造 [12]。Focal Loss 引入一个与预测难度有关的

调制因子，降低已经被模型正确分类的容易样本的损失贡献，从而让训练更加关注难以区分的样本。具体来说，我们令

$$p_{t,i} = y_i \hat{p}_i + (1 - y_i)(1 - \hat{p}_i) \tag{12.153}$$

表示模型分配给真实类别的预测概率，并定义类别权重

$$\alpha_{t,i} = \alpha y_i + (1 - \alpha)(1 - y_i) \tag{12.154}$$

则 Focal Loss 可以写为

$$\mathcal{L}_{\text{Focal}} = -\frac{1}{N} \sum_{i=1}^{N} \alpha_{t,i} (1 - p_{t,i})^{\gamma} \log p_{t,i} \tag{12.155}$$

其中，α 用于平衡正负类别的损失贡献，$\gamma \geq 0$ 控制对容易样本的抑制程度。当 $\gamma = 0$ 时，该损失退化为带类别权重的交叉熵；当 γ 增大时，真实类别预测概率较高的样本会被赋予更小的损失权重。

需要注意的是，Focal Loss 并不是所有 CTR 预估任务都需要采用的默认损失。它更适合在样本极度不均衡、容易样本（Easy Sample）主导整体 Loss 梯度方向等情况下进行尝试。由于 Focal Loss 和类别加权会改变不同样本对优化目标的贡献，其输出的概率也可能偏离真实的点击率。因此，在使用 Focal Loss 时，除了关注 AUC、Recall 等指标，还需要评估 LogLoss、预测点击率分布及 PCOC 等校准类指标。如果模型的预测概率还用于后续的分数融合、策略阈值或预估值乘法组合，就更需要关注概率尺度是否稳定。

对于概率校准，工业级推荐系统中常用的指标之一是 PCOC（Predicted Clicks Over Clicks），即预测点击总量与实际点击总量之比：

$$\text{PCOC} = \frac{\sum_{i=1}^{N} \hat{p}_i}{\sum_{i=1}^{N} y_i} \tag{12.156}$$

当预测概率总体校准良好时，PCOC 通常应接近 1。如果 PCOC 明显大于 1，说明模型预测的点击总量高于实际点击总量；反之则意味着模型可能整体低估了点击概率。不过，PCOC 只能反映总体预测量与实际反馈量的比例，无法单独判断样本级校准是否良好，因此通常还需要结合 LogLoss、校准曲线，以及不同用户群体和流量场景下的分桶统计进行分析。

12.5.2 Pairwise Loss

前一节中的 Pointwise Loss 通常直接优化单个物品的预测结果，但精排模块的最终目标通常是对同一请求中的候选物品进行排序。两个预测概率相近的物品，即使都被模型估计得比较准确，也可能出现排序方向错误。相反，某个模型即使不能准确估计绝对概率，也可能将用户更喜欢的物品排在前面。为了直接优化候选物品之间的相对顺序，我们可以在精排模型建模中引入 Pairwise Loss。

假设请求 q 对应的候选物品集合为

$$\mathcal{I}_q = \{i_1, i_2, \ldots, i_{n_q}\} \tag{12.157}$$

其中，n_q 为该请求的候选物品数量。对于标签不同的两个物品 i 和 j，如果 $y_i > y_j$，则希望模型对物品 i 的预测分数高于物品 j。记两个物品的预测 Logit 分别为 s_i 和 s_j，则模型需要学习的相对顺序可以表示为

$$s_i > s_j, \qquad y_i > y_j \tag{12.158}$$

RankNet 是经典的 Pairwise 排序方法之一 [1]。它将两个物品的分数差映射为偏好概率：

$$\hat{p}_{ij} = \sigma(s_i - s_j) \tag{12.159}$$

并根据两个物品的标签构建相应的二元交叉熵。对于只考虑正负样本对的情形，可以将正样本和负样本之间的损失写为

$$\ell_{ij}^{\text{pair}} = -\log \sigma(s_i - s_j) = \log\left(1 + \exp[-(s_i - s_j)]\right), \qquad y_i > y_j \tag{12.160}$$

如果将满足 $y_i > y_j$ 的物品对组成集合 $\mathcal{P}_q$，则请求级的 Pairwise Loss 可以写为

$$\mathcal{L}_{\text{pair},q} = \frac{1}{|\mathcal{P}_q|} \sum_{(i,j)\in\mathcal{P}_q} \ell_{ij}^{\text{pair}} \tag{12.161}$$

在实际训练中，还可以根据标签差异、样本权重或物品在列表中的位置，为不同物品对设置不同权重。

Pairwise Loss 的梯度直接作用于两个物品的相对分数。令 $\Delta_{ij} = s_i - s_j$，则有

$$\begin{aligned} \frac{\partial \ell_{ij}^{\text{pair}}}{\partial s_i} &= -\sigma(-\Delta_{ij}) \\ \frac{\partial \ell_{ij}^{\text{pair}}}{\partial s_j} &= \sigma(-\Delta_{ij}) \end{aligned} \tag{12.162}$$

当正样本的分数低于负样本时，损失较大，模型会推动正样本分数上升、负样本分数下降；当正样本的分数已经明显高于负样本时，梯度则会逐渐减小。与 BCE 的逐样本梯度不同，Pairwise Loss 的梯度直接由物品对之间的分数差决定，因此能够将优化重点放在同一请求中排序方向错误或间隔不足的样本对上。

然而，Pairwise Loss 也存在一个重要的局限。它优化的是相对分数，而不是单个物品的绝对预测概率。对于同一个请求中的所有物品，如果将预测分数统一增加一个常数 c_q，则有

$$(s_i + c_q) - (s_j + c_q) = s_i - s_j \tag{12.163}$$

可以看出，统一给所有预测分数增加 c_q 偏置项这种变换并不会改变请求内的 Pairwise Loss，也不会改变请求内的排序结果。模型可以通过不同的请求级偏移改变预测概率的整体水平，却不影响物品对之间的相对顺序。这说明 Pairwise Loss 的性质如下：

性质 Pairwise Loss *无法单独确定预测分数的绝对基准。*

因此，当模型仍然使用 $\sigma(s_i)$ 将排序分数转换为点击概率时，问题就会体现出来。即使两个模型产生的候选排序基本相同，它们的 Logit 偏移和分布仍可能不同，从而输出明显不同的预测 CTR。因此，纯 Pairwise 模型虽然可以优化请求内的排序，但其输出的 Sigmoid 值不一定具有可靠的概率含义，PCOC 也可能偏离 1。此外，Pairwise Loss 主要由标签不同的样本对产生监督，如果某个请求中的所有物品标签都相同，就无法从该请求构建有效的正负样本对。

从计算开销的角度来看，一个请求中最多包含 $n_q(n_q-1)/2$ 个无序物品对，因此完整的两两比较通常具有 $O(n_q^2)$ 的复杂度。对于候选规模较大的精排请求，直接构造所有物品对可能带来较高的训练开销。在实际应用中，我们可以只构建正负标签不同的物品对，并通过困难负样本选择、候选对采样或按请求归一化等方式控制训练成本，避免少数候选数量特别多的请求主导整体损失。

在工业级推荐系统中，Pairwise Loss 经常用于提升同一请求下候选物品的排序能力。例如，在短视频推荐中，同一个请求可能对应一批曝光视频，用户最终点击、长播或者有效播放的视频通常应当排在其他候选物品前面。相比只约束每个视频的点击概率，引入请求内的 Pairwise Loss 可以直接告诉模型哪些物品应该排在另一些物品之前。因此，在精排模型中，我们可以结合 Pointwise Loss 和 Pairwise Loss，同时优化单个物品的预测概率和请求内的相对排序，从而在保证概率预测合理性的同时提升排序效果。

12.5.3 Listwise Loss

Pairwise Loss 将排序任务拆分为一组物品对，Listwise Loss 则直接把一个请求的整个候选列表作为优化对象。它不仅关注某两个物品之间的相对顺序，还利用列表中多个候选的标签和预测分数共同构建训练目标 [2]。对于同一请求的一屏曝光物品，如果能够同时获取这些物品的点击、有效播放或其他反馈标签，就可以利用 Listwise Loss 直接建模整个曝光列表中的相对偏好。

Listwise Loss 有多种具体形式，其中一种具有代表性的方式是 ListNet。其基本思路是分别将真实相关性标签和模型预测分数转换为列表上的概率分布，再通过交叉熵来度量二者的差异。假设某个请求包含 n_q 个候选物品，其真实相关性标签为 r_i，模型预测分数为 s_i，则可以分别构造目标分布和预测分布：

$$q_i = \frac{\exp(r_i/\tau_y)}{\sum_{j=1}^{n_q}\exp(r_j/\tau_y)}, \qquad p_i = \frac{\exp(s_i/\tau_s)}{\sum_{j=1}^{n_q}\exp(s_j/\tau_s)} \tag{12.164}$$

其中，τ_y 和 τ_s 分别用于控制目标分布与预测分布的平滑程度。因此，单次请求下 ListNet 的列表级损失为

$$\mathcal{L}_{\text{ListNet},q} = -\sum_{i=1}^{n_q} q_i \log p_i \tag{12.165}$$

而 ListNet 整体的训练目标可以对所有有效请求的 $\mathcal{L}_{\text{ListNet},q}$ 取平均得到最终的损失值。与 Pairwise Loss 比较两个物品的分数差不同，ListNet 先把整个列表的相关性标签转换为目标分布，再要求模型输出与目标分布一致的预测分布。因此，某个候选物品的梯度不仅取决于它自身的预测结果，也取决于同一列表中其他候选的分数和标签。对于固定温度系数的 Softmax 交叉熵损失函数，其对预测分数的梯度可以写为

$$\frac{\partial \mathcal{L}_{\text{ListNet},q}}{\partial s_i} = \frac{1}{\tau_s}(p_i - q_i) \tag{12.166}$$

因此，模型能够在整个列表的归一化分布中协调各个候选物品的相对分数。

对于短视频推荐场景来说，如果一个请求对应用户看到的一屏候选物品，就可以把该屏曝光物品及其反馈作为 Listwise Loss 的输入。与逐个样本优化点击概率相比，这种方式能够直接利用同一曝光列表的整体标签结构，帮助模型学习更符合请求内排序需求的候选分布。尤其当业务更关注列表顶部的点击、观看或者互动效果时，可以根据相关性标签的定义，进一步选择适合列表级目标的损失形式。

不过，Listwise Loss 的预测输出通常是同一列表上的归一化分布，并不天然等于单个物品的 CTR。即使两个请求中某个物品对应的 Listwise 分数相同，只要其他候选物品及其分数不同，Softmax 的归一化结果就可能不同。因此，Listwise Loss 同样不能直接替代具有明确概率含义的 Pointwise Loss。如果排序系统不仅需要请求内的相对顺序，还需要将 CTR 作为后续分数融合、策略阈值或业务预估的输入，就仍然需要通过 Pointwise 目标或者其他校准方法来约束绝对概率。

12.5.4 不同损失函数的组合与实践对比

Pointwise、Pairwise 和 Listwise 分别从单个样本、物品对和完整候选列表三个粒度定义优化目标。在工业级推荐精排模型中，这些损失并不是相互排斥的，可以根据具体的业务目标进行组合。例如，在 CTR 预估模型中，可以在 BCE 损失的基础上增加 Pairwise Loss，使得模型在学习单个物品点击概率的同时，也能够直接优化同一请求中不同候选物品之间的相对顺序：

$$\mathcal{L} = \mathcal{L}_{\text{BCE}} + \lambda_{\text{pair}} \mathcal{L}_{\text{pair}} \tag{12.167}$$

其中，λ_{pair} 用于控制排序目标在联合优化中的权重。如果还需要引入列表级监督，则可以进一步扩展为

$$\mathcal{L} = \lambda_{\text{point}} \mathcal{L}_{\text{BCE}} + \lambda_{\text{pair}} \mathcal{L}_{\text{pair}} + \lambda_{\text{list}} \mathcal{L}_{\text{list}} \tag{12.168}$$

需要注意的是，这些损失的权重并不是越大越好。不同损失的数值范围、梯度大小和优化目标都可能存在差异，如果某一项损失的权重过高，其梯度就可能主导模型更新，反而影响其他目标的优化。因此，实际训练中需要结合各项损失的收敛情况、梯度分布以及最终的业务指标，合理调整不同损失的权重。

Pointwise Loss 与 Pairwise Loss 之所以经常需要结合使用，我们可以从二者的优化目标和梯度特征来理解。对于 BCE，其对单个样本预测 Logit s_i 的梯度为

$$\frac{\partial \ell_{\text{BCE},i}}{\partial s_i} = \hat{p}_i - y_i \tag{12.169}$$

其中，$\hat{p}_i = \sigma(s_i)$。BCE 根据单个物品的预测概率与真实标签之间的差异更新模型，既要求正样本的预测概率足够高，也要求负样本的预测概率足够低，因此能够为模型提供明确的概率监督。而 Pairwise Loss 则直接比较两个候选物品的预测分数，根据它们的相对顺序产生梯度。当同一请求中的正样本得分低于负样本时，Pairwise Loss 会推动正样本得分上升、负样本得分下降；当二者的排序关系已经满足要求时，相应的梯度则会逐渐减小。因此，Pairwise Loss 能够为同一请求中排序错误或分数间隔不足的物品对提供直接的优化信号。

从优化目标来看，BCE 和 Pairwise Loss 具有一定的互补性。BCE 关注单个物品的预测概率，能够约束模型输出的绝对水平；Pairwise Loss 关注物品之间的相对顺序，能够针对请求内的排序关系提供额外监督。将二者结合起来，模型既可以学习物品被点击的概率，又可以利用同一请求中候选物品之间的比较信息，改善排序能力。

为了更加直观地比较不同损失函数的特点，我们构建了一个可复现的模拟实验。实验以物品曝光为基本样

本，每个请求包含 16 个候选物品。对于每个物品，首先根据用户上下文、物品特征及二者之间的交互信息生成点击概率，再根据该概率独立采样得到点击标签。实验包含 1800 个训练请求，即 28800 条曝光样本；另外使用 600 个请求，即 9600 条曝光样本作为指标验证集。数据生成的核心代码如代码 12.8 所示。

代码 12.8: 按请求生成逐物品曝光特征与后验点击标签

```
def make_dataset(n_queries, list_size=16, seed=0):
    rng = np.random.default_rng(seed)
    user_context = rng.normal(size=(n_queries, 1, 2))
    item = rng.normal(size=(n_queries, list_size, 4))
    ctx = np.broadcast_to(user_context, (n_queries, list_size, 2))
    inter = np.stack(
        [ctx[..., 0] * item[..., 0], ctx[..., 1] * item[..., 1]], axis=-1
    )
    x = np.concatenate([ctx, item, inter], axis=-1).astype(np.float32)
    w = np.array([0.65, -0.45, 0.90, 0.58, -0.42, 0.25,
                  0.62, -0.38], dtype=np.float32)
    logits = (-2.05 + np.einsum("qld,d->ql", x, w)
              + 0.90 * (item[..., 2] ** 2 - 1.0)
              + 0.65 * np.maximum(item[..., 3], 0.0))
    probs = 1.0 / (1.0 + np.exp(-logits))
    y = rng.binomial(1, probs).astype(np.float32)
    return x, y, probs
```

从上述数据生成过程可以看出，点击概率不仅与单个物品特征有关，还受到用户上下文与物品特征交互的影响，并包含额外的非线性项。这样可以构造一个同时具有线性关系和非线性关系的模拟 CTR 任务，用于观察不同损失函数在相同模型结构下的优化差异。

实验总共会对比五种训练目标，即 Pointwise BCE、纯 Pairwise、BCE + Pairwise、Focal Loss 和 ListNet。其中，Pointwise BCE 和 Focal Loss 将曝光样本全局打散，按照普通的样本级 mini-batch 来进行更新。纯 Pairwise、BCE + Pairwise 和 ListNet 则保留同一请求内的候选关系，因此需要按照请求来组织 mini-batch，以便构造物品对和列表级损失。组合目标采用 $\mathcal{L}_{\text{BCE}} + 0.20\mathcal{L}_{\text{pair}}$。Focal Loss 的参数设置为 $\gamma = 2$、正样本权重 $\alpha = 0.25$。ListNet 使用二值点击标签经过 Softmax 得到目标分布，并跳过不包含正样本的列表。这些参数仅用于本次模拟实验，没有进行专门的超参数搜索。

代码 12.9 给出了三个非标准 BCE 目标的核心实现。其中，Pairwise Loss 只对同一请求内标签不同的正负样本对计算损失，并对有效物品对数量进行归一化；Focal Loss 根据真实类别的预测概率调整单个样本的损失权重；而 ListNet 则在每个有效请求内部计算预测分布与目标分布之间的交叉熵。

代码 12.9: Pairwise、Focal 与 ListNet 的核心损失计算

```
def pairwise_ranknet_loss(scores, labels):
    diff = scores.unsqueeze(2) - scores.unsqueeze(1)
    mask = (labels.unsqueeze(2) > labels.unsqueeze(1)).to(scores.dtype)
    pair_losses = F.softplus(-diff)
    if mask.sum().item() == 0:
        return scores.sum() * 0.0
    return (pair_losses * mask).sum() / mask.sum()

def focal_loss_with_logits(logits, labels, gamma=2.0, alpha=0.25):
    bce = F.binary_cross_entropy_with_logits(logits, labels,
```

```
                                        reduction="none")
    p = torch.sigmoid(logits)
    pt = p * labels + (1.0 - p) * (1.0 - labels)
    alpha_t = alpha * labels + (1.0 - alpha) * (1.0 - labels)
    return (alpha_t * (1.0 - pt).pow(gamma) * bce).mean()

def listnet_loss(scores, labels):
    valid = labels.sum(dim=1) > 0
    if not valid.any():
        return scores.sum() * 0.0
    target = F.softmax(labels[valid], dim=1)
    log_pred = F.log_softmax(scores[valid], dim=1)
    return -(target * log_pred).sum(dim=1).mean()
```

为了排除网络结构差异的影响，五种训练目标均使用相同的线性打分模型：

$$s(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b \tag{12.170}$$

并从相同的全零参数开始训练。代码 12.10 展示了训练过程中 mini-batch 组织方式和损失选择的关键部分。其中，`xt` 和 `yt` 分别表示输入特征和点击标签，形状为 $[Q, 16, 8]$ 和 $[Q, 16]$，Q 为当前批次中的请求数量。对于 Pointwise BCE 和 Focal Loss，模型将请求和物品两个维度展平，直接按照曝光样本进行训练；对于 Pairwise、BCE + Pairwise 和 ListNet，则保留请求维度，以便在每个请求内部构造相应的排序损失。损失选择函数根据当前训练目标，分别计算 BCE、Pairwise、Focal 或 ListNet，组合目标则对 BCE 和 Pairwise Loss 进行加权求和。

代码 12.10: 逐曝光与逐请求的 mini-batch 训练路径（节选）

```
for epoch in range(22):
    model.train()
    if name in ("Pointwise BCE", "Focal loss"):
        flat_x = xt.reshape(-1, n_features)
        flat_y = yt.reshape(-1)
        batches = torch.randperm(len(flat_y), device=DEVICE).split(64 * list_size)
    else:
        batches = torch.randperm(n_queries, device=DEVICE).split(64)
    for idx in batches:
        if name in ("Pointwise BCE", "Focal loss"):
            sb, yb = model(flat_x[idx]), flat_y[idx]
        else:
            xb, yb = xt[idx], yt[idx]
            sb = model(xb.reshape(-1, n_features)).reshape(-1, list_size)
        loss = training_objective(name, sb, yb)
        opt.zero_grad(set_to_none=True)
        loss.backward()
        opt.step()
        if name in ("Pairwise only", "ListNet"):
            with torch.no_grad():
                model.linear.weight[:, :2].zero_()
                model.linear.bias.zero_()
```

上面的训练代码仅展示 mini-batch 组织和参数更新的主要流程，省略了数据转换、模型初始化、优化器构造、

损失选择函数和评估代码。实验主要从排序能力和概率预估能力两个方面进行评估。全局 AUC 用于衡量模型在所有曝光样本上的整体区分能力；NDCG@5 用于评价同一请求中候选列表前五位的排序质量，仅对至少包含一个正样本的请求进行计算；请求内 Pairwise Accuracy 则统计同一请求中正负样本对被正确排序的比例。在概率预估方面，使用 LogLoss、平均 Sigmoid 分数和 PCOC 进行评估。其中，平均 Sigmoid 分数表示所有样本预测概率的均值，PCOC 则反映预测点击总量与实际点击总量之间的比例关系。需要注意的是，纯 Pairwise 的 Sigmoid 分数只是将排序分数经过 Sigmoid 转换后得到的诊断值，并不代表经过概率监督的 CTR 预估；而 ListNet 输出的是请求内归一化的相对分布，不能直接解释为单个物品的点击概率，因此不报告其 LogLoss 和 PCOC。指标验证集的实际点击率为 0.2283。不同损失函数的实验结果如表 12.1 所示。

表 12.1: 不同损失函数在合成 CTR 数据上的实验结果。

训练目标	全局 AUC	NDCG@5	请求内成对准确率	LogLoss	平均 Sigmoid 分数	PCOC
Pointwise BCE	0.7955	0.6308	0.7738	0.4242	0.2375	1.040
Pairwise Only	0.7685	0.6177	0.7734	0.6348	0.4968	2.176
BCE + Pairwise	0.7964	0.6200	0.7733	0.4234	0.2358	1.033
Focal Loss	0.7960	0.6284	0.7730	0.4685	0.2839	1.243
ListNet	0.7644	0.6226	0.7693	–	–	–

从实验结果可以看到，不同损失函数在各项指标上的表现并不完全一致。Pointwise BCE 的全局 AUC 为 0.7955，NDCG@5 为 0.6308，平均预测 CTR 为 0.2375，与实际点击率较为接近。加入 Pairwise Loss 后，全局 AUC 略有提高，由 0.7955 增至 0.7964，但 NDCG@5 从 0.6308 降至 0.6200，请求内 Pairwise Accuracy 也略有下降。因此，在本次模拟实验中，BCE + Pairwise 并没有带来明确的列表顶部排序收益，不能仅根据全局 AUC 的小幅变化就认为其排序效果得到改善。

纯 Pairwise 的结果则体现出另一个现象：它的请求内 Pairwise Accuracy 为 0.7734，与 BCE 的 0.7738 接近，但平均 Sigmoid 分数达到 0.4968，明显高于实际点击率 0.2283，对应的 PCOC 为 2.176。这说明模型虽然学习了请求内的相对排序关系，却没有得到有效的绝对概率约束。Focal Loss 的全局 AUC 和 NDCG@5 与 BCE 接近，但 PCOC 达到 1.243，也表明改变样本的损失权重后，预测概率的整体尺度可能发生变化。

为了进一步观察模型输出的概率尺度，图 12.19 将不同损失函数的平均 Sigmoid 分数与验证集实际点击率进行比较。BCE 和 BCE + Pairwise 的平均预测 CTR 分别为 0.2375 和 0.2358，均接近实际点击率；Focal Loss 的平均预测 CTR 为 0.2839；纯 Pairwise 则达到 0.4968。其中，纯 Pairwise 的 Sigmoid 分数只是对排序分数进行变换后的结果，并不具有可靠的 CTR 概率含义。该结果说明，即使模型能够对候选物品进行相对排序，也不能据此认为其预测概率同样准确。

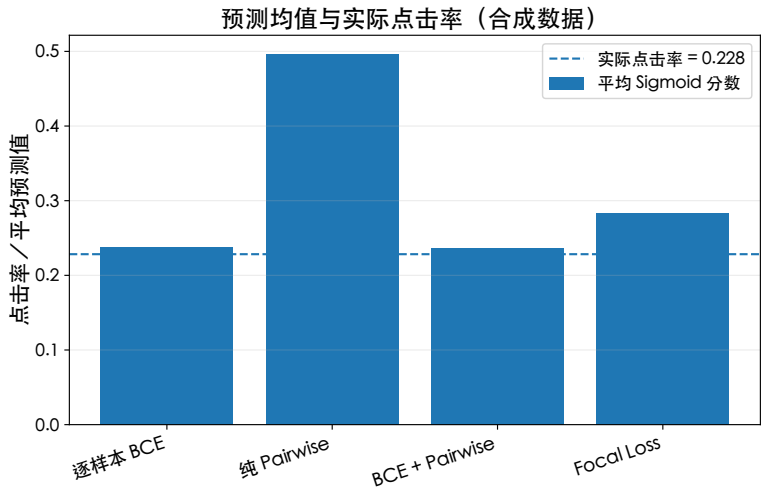


图 12.19: 不同损失函数的平均 Sigmoid 分数与实际点击率比较。

仅观察平均预测值还不足以判断模型的概率校准情况，因此我们进一步使用独立生成的 960 万条曝光样本绘制分桶校准曲线，如图 12.20 所示。具体来说，将预测概率按照固定宽度 0.05 划分为多个区间，分别计算每

个区间内的平均预测概率和实际点击率。图中的对角线表示理想校准情况，即平均预测概率与实际点击率相等。

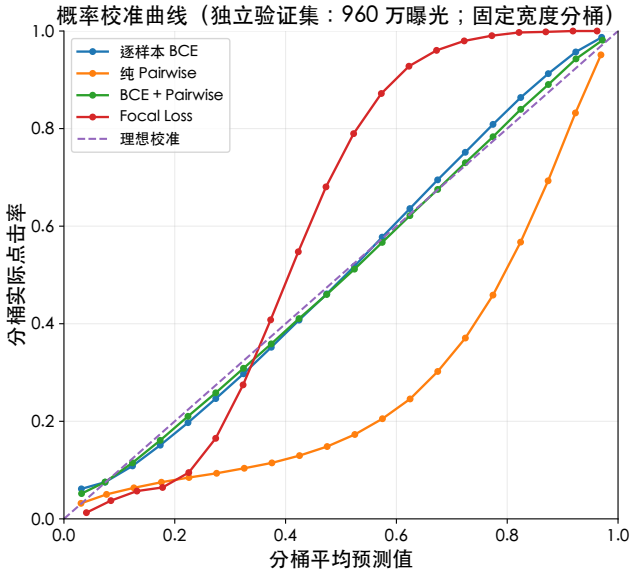


图 12.20: 不同损失函数在独立合成曝光样本上的分桶校准曲线。

从图中可以看到，BCE 和 BCE + Pairwise 的曲线整体较为接近理想校准线，而纯 Pairwise 的 Sigmoid 分数明显偏离对角线。Focal Loss 在部分中高概率区间也出现了一定的校准偏差。这说明，Focal Loss 通过调整不同样本的训练权重，虽然能够改变模型关注的样本分布，但并不保证预测概率仍然与真实点击率保持一致。

需要注意的是，分桶校准曲线还受到各个区间样本数量的影响。尽管这里使用了较大的独立样本集，但高预测概率区间内的样本仍然可能较少。例如，Focal Loss 在 $[0.95, 1]$ 区间内只有 36 条曝光，因此该区间的实际点击率波动较大，不能将其作为高置信度的概率估计。这些曲线主要用于观察整体校准趋势，不能替代独立指标验证集上的模型比较。

除了最终的排序与校准指标，我们还可以观察不同损失在训练过程中的收敛情况。图 12.21 展示了五种训练目标从第 0 轮到第 22 轮的原始损失变化。其中，左图将 BCE、纯 Pairwise 和 BCE + Pairwise 放在同一坐标系中，中图和右图分别展示 Focal Loss 和 ListNet 的训练曲线。由于不同损失的定义和数值尺度不同，各子图采用了不同的纵轴范围，因此不能直接通过损失数值大小比较不同训练目标的优劣。

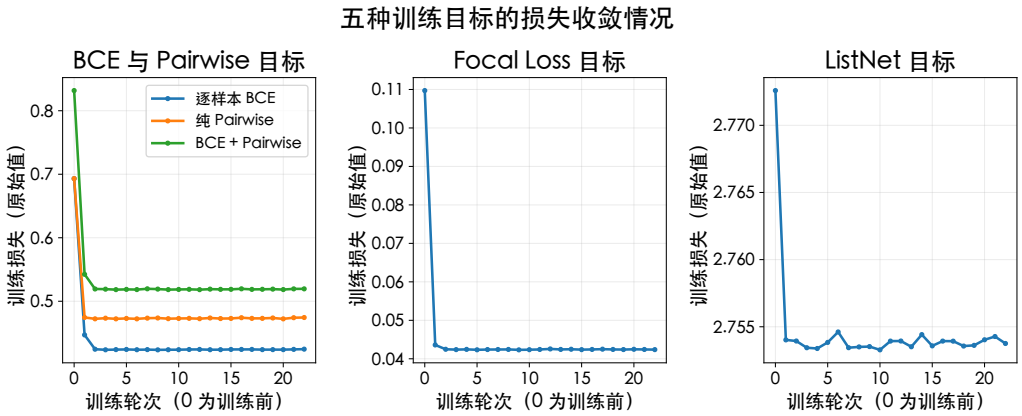


图 12.21: 五种训练目标的原始损失曲线。

从上述训练曲线的变化可以看到，各个训练目标在第一轮训练后均出现明显下降，随后逐渐趋于平稳，说明模型能够在各自的目标下完成基本收敛。需要注意，ListNet 的交叉熵包含目标分布本身的熵，即使预测分布已经接近目标分布，其损失值也不一定趋近于零。因此，ListNet 的原始损失维持在约 2.75，并不意味着模型没有学到有效的排序关系。对于不同训练目标，仍然应当结合对应的排序指标和概率校准指标判断优化效果。

综合来看，Pointwise、Pairwise 和 Listwise Loss 分别从单个物品、物品对和完整列表三个层面来约束模型的预测行为。Pointwise Loss 直接学习单个物品的预测概率或者反馈值，Pairwise Loss 优化同一请求中候选物品之间的相对顺序，而 Listwise Loss 则在整个候选列表上构造目标分布并进行联合优化。在工业级推荐精排模型中，算法工程师可以根据业务目标选择其中一种或多种损失进行组合，但还需要充分考虑不同损失的优化方向、数值尺度以及梯度的影响。最终的模型选择以及 Loss 选择仍然需要结合真实的业务数据，通过离线指标和线上 A/B 实验来进行验证。

12.6 精排模型实践案例

前面的小节介绍了工业级精排模型的典型架构，以及 Pointwise、Pairwise 和 Listwise 等常用损失函数。与粗排模型主要面向大规模候选集筛选不同，精排模型面对的候选规模相对较小，因此可以使用更加丰富的特征、更复杂的交互结构以及更细致的多目标建模方法。图 12.22给出了一个典型的工业级精排模型结构示意图。需要强调的是，该图不是某一个固定模型的实现，而是对工业实践中常见模块及其连接关系的抽象；不同业务可以根据流量规模、延迟预算、特征可用性和目标体系，对其中的模块进行裁剪、替换或组合。

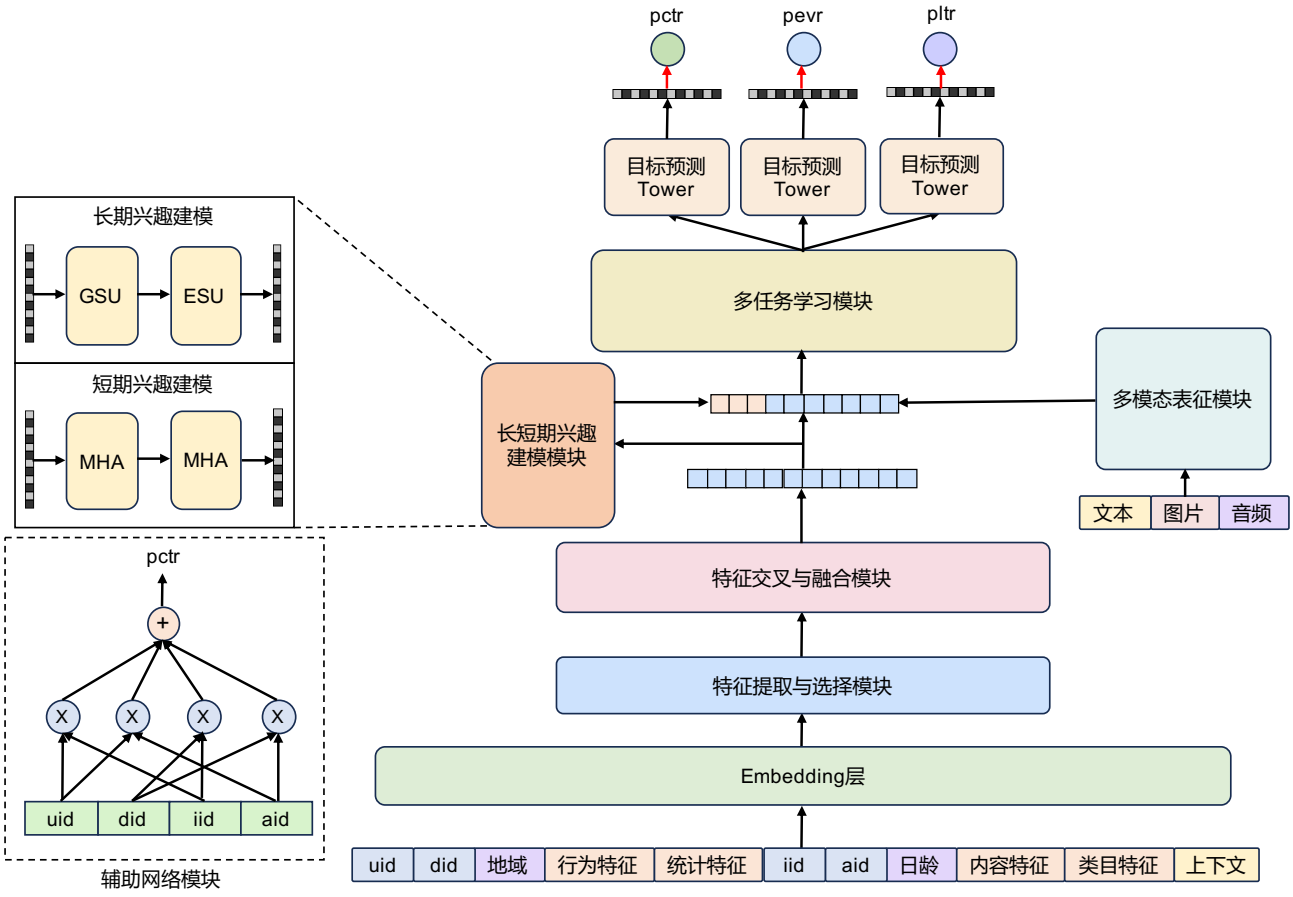


图 12.22: 典型工业级精排模型网络结构示意图。

（1）特征输入与 Embedding 层。图的最底部是精排模型的原始特征输入，通常包括用户、设备、物品、作者、年龄、地域、行为特征、统计特征、内容特征、类目特征以及请求上下文等。用户侧特征用于描述用户身份、画像和历史偏好，物品侧特征用于描述内容属性、作者质量及物品的历史表现，上下文特征则用于刻画当前请求发生时的时间、场景、设备和网络环境。离散 ID、类目和标签等特征通常通过 Embedding 表映射为稠密向量；连续统计特征可以经过归一化、分桶或小型全连接网络后，再与 Embedding 表示融合。图中底部的多个特征槽并不意味着所有特征必须经过完全相同的处理，而是强调不同来源的信息最终需要进入统一的表示空间。

精排模型通常会共享 user id、device id、item id 和 author id 等高价值特征，但共享并不意味着所有模块必须使用完全相同的表示。在实际系统中，可以针对不同目标或不同子网络建立独立 Embedding，也可以在共享 Embedding 之后增加任务特定的投影层，从而同时保留跨任务的统计强度和目标之间的差异。对于更新频繁的统计特征，还需要关注训练样本与线上特征的时间一致性，避免使用未来信息造成标签泄漏。

（2）特征提取与选择模块。Embedding 层之上是特征提取与选择模块，其作用是对不同特征进行进一步变换，并根据当前用户、物品和上下文动态调整特征贡献。该模块可以使用 LHUC、SE-Net、Light-SE 等结构学习特征或隐藏单元的权重，也可以针对稠密特征、统计特征和 ID 特征分别设置轻量子网络。与简单地将所有 Embedding 直接拼接相比，特征选择模块能够降低噪声特征的干扰，突出对当前目标更有价值的信息。例如，同一个物品在不同用户和不同上下文下，作者质量、历史点击统计和内容语义的相对重要性可能并不相同，动态选择机制可以为这些特征分配不同权重。关于 LHUC、SE-Net、Light-SE 等结构，第15章《特征工程》将进行更加深入的讨论；个性化网络参数相关方法将在第24章《个性化网络参数》中进行介绍。

（3）特征交叉与融合模块。特征提取之后的表示会进入特征交叉与融合模块，用于建模用户、物品、上下文和统计特征之间的组合关系。FM 和 DeepFM 适合提取低阶及高阶组合，DCN 和 DCN-V2 通过显式交叉增强特征组合能力，DNN 则可以学习隐式的非线性交互关系，进而捕捉更复杂的特征组合模式。这些特定的特征交叉与融合方法我们将在第15章《特征工程》中详细介绍。此外，在精排阶段，模型不仅需要判断用户是否喜欢某类内容，还需要判断“这个用户在当前场景下是否会对这个物品产生特定行为”。因此，用户 ID 与物品 ID、用户兴趣与内容类目、设备网络与内容时长等交叉关系往往具有重要价值。交叉后的表示通常会与未交叉的基础表示共同保留，避免复杂交互网络丢失原始特征中的稳定信息。

（4）长短期兴趣建模模块。图中部偏左的长短期兴趣建模模块接收主干网络表示，并与用户行为序列编码结果进行融合。短期兴趣主要反映用户最近一段时间的即时偏好，例如最近观看、点击、跳过或互动的内容；而长期兴趣则反映用户在更长时间范围内相对稳定的主题、作者和内容消费倾向。短期序列可以采用 MHA、Transformer 或注意力池化进行编码，通常以当前候选物品或当前请求表示作为 Query，从历史行为中检索与候选最相关的行为。长期序列规模更大，常先通过兴趣检索或分桶召回缩小范围，再利用 GSU、ESU 等兴趣提取单元进行压缩和更新。图 12.22中左侧分别画出了长期兴趣建模和短期兴趣建模的内部流程。前者强调对较长历史进行兴趣归纳，后者强调对近期行为进行细粒度注意力建模。

将长短期兴趣表示与候选物品表示融合后，模型能够同时利用稳定偏好和实时变化的兴趣。需要注意的是，序列建模的收益不仅取决于网络结构，也取决于行为类型、时间窗口、截断策略、候选相关性和样本中的可见性边界。关于用户兴趣与序列建模，后续第20章将进行系统介绍。

（5）多模态表征模块。图右侧的多模态表征模块用于处理文本、图片和音频等内容信息。现代推荐物品往往不只有 ID 和类目，还包含标题、封面、视频帧、语音及其他内容语义。工程上可以直接使用预训练模型生成的多模态 Embedding，也可以增加轻量投影层，对不同模态进行归一化和融合。在计算资源允许时，还可以通过交叉注意力或门控结构建模模态之间的互补关系。多模态表示通常会与 ID 和统计特征共同输入主干网络，从而兼顾物品的语义相似性与具体物品的历史表现。关于多模态表征在推荐模型中的使用以及其他多模态推荐相关的话题我们将会在后续下册的第34章进行介绍。

（6）多任务学习模块与目标预测 Tower。长短期兴趣表示、特征交叉表示以及多模态表示在精排模型网络结构的中部汇合之后，会进入多任务学习模块。该模块负责在共享底层表征的基础上，为不同业务目标学习适合自己的信息。常见目标包括点击率（pCTR）、有效播放率或观看时长相关目标（pEVR）、点赞率、评论率、分享率和转化率等。多任务结构可以采用共享底层加任务特定 Tower 的形式，也可以使用 MMOE、PLE、CGC 等结构，使不同任务通过专家网络和门控网络选择所需的信息。

图 12.22的上方为多个目标预测 Tower，每个 Tower 根据多任务共享表示输出一个具体目标的预估值，例如 pCTR、pEVR 和 pLTR。Tower 可以由若干全连接层、归一化层和输出层组成，也可以针对不同目标使用不同的特征子集和网络深度。多任务训练中还应关注损失尺度、样本选择以及梯度冲突问题，不能仅通过简单地把多个损失相加来保证多任务一定有效。多任务学习相关的内容将会在后续第21章《多目标与多场景建模》中进一步讨论。

（7）辅助网络模块。图 12.22中左下角虚线框表示辅助网络模块。示意图中以 FM 为例，将若干关键 ID 特征及

其交叉表示用于辅助预测 pCTR。在实际的精排模型中，我们也可以使用双塔结构的 Matching Net、浅层 DNN 或其他轻量级网络来作为辅助任务网络。辅助网络的主要作用是为共享 Embedding 和中间表示提供额外监督，使重要特征得到更充分的训练。它通常不直接承担线上最终排序，因而可以采用与主网络不同的结构，但其输入和共享方式必须经过明确设计。

辅助网络可以与主网络共享 user id、device id、item id、author id 等 Embedding，也可以只共享部分底层表示而保留独立的特征投影层。共享表示有助于多个任务互相提供训练信号，独立表示则可以避免辅助目标对主任务产生过强干扰。图中的虚线框强调了辅助网络与主网络之间的辅助关系，而不是在线推荐中必须额外执行的一次模型推理结构。

最后，各目标 Tower 输出的 pxtr 并不等同于最终排序分数。它们通常会进入下游多任务融合模块，根据业务目标、用户状态和场景约束组合成最终排序分数，再交给重排模块完成多样性、序列价值评估以及策略约束处理等。因此，工业级精排模型不仅是网络结构的堆叠，还需要保证特征、标签、任务输出与下游融合逻辑之间的接口一致。

12.7 精排模型实践讨论

工业级精排模型的迭代，通常不是单独追求网络规模或离线 Loss 下降，而是在预估准确性、增量信息、线上延迟、上下游协同和业务收益之间持续寻找平衡。由于精排模型位于粗排之后、重排之前，其输出既受到上游候选集合质量的限制，也会影响下游多任务融合和策略决策。因此，评估一次精排改动时，既要观察单任务离线指标，也要分析该改动是否改变了不同目标之间的相对关系，以及最终是否能够在完整链路中产生收益。

在工业实践的早期阶段，序列建模和特征交叉通常是比较直接的增量来源。一方面，可以引入新的长期序列检索和建模方式，从用户更长时间范围的行为中提取稳定兴趣；另一方面，可以采用 DCN、DCN-V2、Attention 或其他特征交叉结构，增强用户、物品、上下文和统计特征之间的组合建模能力。特征选择网络则可以提高重要特征的权重、降低噪声特征的影响。序列建模尤其需要关注行为时间边界、行为类型、候选相关性和线上可用性，因为一个结构更复杂但无法稳定获取实时序列的方案，未必能够转化为可靠的线上收益。关于这些问题，后续第20章将会进行深入的讨论。

除了序列建模这一重要模块之外，多任务学习模块也是精排模型中承载多目标建模的核心模块。通过共享底层表征，CTR、停留时长、转化率和互动率等任务可以相互提供监督信号，尤其能够缓解部分任务样本稀疏的问题。但是，不同任务的标签定义、正样本比例、噪声水平和优化方向并不相同，简单共享所有网络层可能产生梯度冲突和多任务负迁移。因此，实际设计中需要在共享层、任务特定层和任务间信息传递之间做权衡，并结合损失权重、动态加权、梯度调控以及任务分组等方法进行优化。相关问题将在第21章《多目标与多场景建模》中详细讨论。

除了共享底层表示，多目标 Task Tower 之间的信息共享与传递也非常重要。不同目标之间既存在相关性，也存在不能完全互相替代的差异。例如，点击和有效播放通常具有一定相关性，但点击并不必然意味着深度消费；点赞、评论和分享可能都反映互动意愿，却具有不同的触发条件。多目标感知建模可以通过共享部分 Tower、跨任务注意力、任务关系图或者多目标特征交互等方式建立信息流动机制，同时保留各目标的独立预测能力。

随着精排模型不断迭代，新的网络结构带来的收益通常会逐步降低，此时需要重新审视监督目标是否覆盖业务中的关键行为。加 label 的核心不是机械地增加输出节点，而是利用更多能够反映用户真实反馈的行为信号，扩展模型可学习的用户行为空间。在短视频场景中，除了 CTR，还可以引入有效播放、观看时长、完播、点赞、评论、分享和关注等目标。在直播或电商场景中，还可以加入进入直播间、停留、加购、下单以及支付等目标。合理的辅助标签能够帮助模型理解“点击之后是否真正消费”“消费之后是否产生更强互动”等关系，从而避免模型只围绕浅层点击信号优化。

其中，播放时长类标签在短视频精排中尤其重要。它能够在一定程度上反映用户对内容的实际消费程度，但时长目标也容易受到视频 Duration、曝光位置、网络状况和自动播放策略等因素影响。因此，时长标签需要结合业务定义进行处理，例如区分有效播放、长播放和完播行为，并根据目标采用合适的加权或分布建模方式。精排模型往往不止预估一种播放时长相关目标，后续第23章将会进一步介绍时长预估建模这一话题。

除了时长预估的 pxtr 之外，精排模型通常还会输出多个别的 pxtr，但真正决定候选排序的往往是下游多任务融合模块。这里我们需要特别区分“目标预估”与“最终排序”之间的差异。精排模型并不决定最终的排序，相反精排模型之后的多任务融合机制与排序机制才真正决定精排模块最终的候选排序。多任务融合的方式可以是基于既有业务公式的加权组合，也可以使用强化学习或者 CEM 等方法搜索融合公式及其超参数。此外，近年来还出现了基于深度网络的端到端融合方法。不同融合方式对各 pxtr 的校准、单调性以及数值尺度都有不同要求，因此在修改某个 Task Tower 时，不能只看该任务自己的 AUC 或 Loss，还需要考虑该任务预估的 pxtr 分数对于下游多任务融合的影响、可能的候选排序变化以及最终的业务指标情况。关于这些多任务融合的问题，后续第25章《多任务融合》将会进行详细的介绍。

在工业级推荐系统中，在精排模型输出多个 pxtr 之后，这些 pxtr 还可能被多个上下游模块复用。下游多任务融合模块使用这些预估值完成排序，重排模块可能据此进行多样性和策略控制，而上游粗排模块也可能利用精排结果开展粗精排链路一致性优化。pxtr 的复用越多，单个字段的语义、版本以及校准变化所带来的影响面就会越大。因此，工程上需要明确每个 pxtr 的定义、样本窗口、模型版本、校准方式等，并在改动时进行影响分析。此外，由于前后链路的各种耦合以及优化方向的差异，工业级推荐系统中各个模块之间很容易出现不一致的现象。因此，关于链路一致性优化的相关问题，我们将会在后续第22章《链路一致性建模》中进一步讨论。

此外，精排模型还需要在效果和系统成本之间进行长期权衡。序列网络、多模态编码和复杂交叉结构可能带来更强的表达能力，但也会增加特征准备、模型推理、显存占用以及线上耗时增加等问题。对于实时特征和多模态特征，我们需要关注特征的命中率以及覆盖率情况，以及特征的缺失值处理；对于高成本的序列检索，我们还需要设置候选截断、缓存以及其他性能优化手段。最后，对于模型的发布，线上应该配套监控特征覆盖率、各任务预估均值、融合分数分布、服务耗时 P99 分位数以及异常率等重要指标。只有当模型改动在这些约束下仍然保证稳定性，离线的收益才具有真正的工程意义，同时也才能转化为稳定的线上 A/B 指标收益。

总体来说，工业级精排模型的优化可以概括为三个相互联系的方向：第一，通过序列建模、特征交叉和多模态表示引入更丰富的增量信息；第二，通过加 label、多任务学习和多目标感知建模，使模型更全面地理解用户的行为反馈；第三，通过 pxtr 校准、任务融合以及粗精排协同等方式，使得模型输出能够在完整推荐链路中真正生效。未来的精排模型迭代不能只关注某一个网络模块，而需要同时理解模型结构、训练数据、线上服务和上下游决策之间的关系，才能持续发掘稳定且可复现的增量收益。

12.8 本章小结

本章围绕工业级推荐系统中的精排模型进行了系统介绍。精排模型位于粗排之后，面对规模相对较小但信息更加丰富的候选集合，需要在合理的计算成本下，对用户与候选物品之间的关系进行更加细致的建模，并为后续多任务融合和重排提供可靠的预测信号。本章从精排模型的发展历程、核心技术问题、典型网络结构、预测目标与损失函数以及工业实践等方面进行了讨论。

首先，本章梳理了精排模型从机器学习时代、深度学习时代，到多任务与多目标优化时代以及推荐大模型时代的发展过程。不同阶段的技术演进，分别对应着人工特征工程、复杂特征交互、用户兴趣建模、多目标知识共享以及大规模表示学习等主要问题。精排模型的发展并不是简单追求网络规模变大，而是围绕如何更充分地利用特征和行为信息、如何提升模型表达能力，以及如何满足工业系统的训练和服务约束不断演进。

随后，本章介绍了精排模型需要重点解决的三个核心技术问题，即特征选择与表示、模型结构设计，以及训练与优化。在传统精排模型部分，我们介绍了 LR、GBDT、FM 和 FFM 等经典方法，分析了它们在处理稀疏特征、非线性关系和特征交互方面的特点。在深度精排模型部分，我们进一步讨论了显式特征交互与深层表征融合、辅助网络、用户行为序列建模、多任务学习以及基于 Transformer 的统一模型架构，展示了精排模型如何逐步增强对复杂用户兴趣和用户–物品关系的建模能力。

在预测目标与损失函数部分，本章重点介绍了 Pointwise、Pairwise 和 Listwise 三类优化方式。Pointwise Loss 关注单个样本的预测准确性，Pairwise Loss 关注同一请求内候选物品的相对顺序，Listwise Loss 则从完整候选列表的角度进行联合优化。不同损失函数在排序能力、概率校准、训练稳定性和业务适配性方面各有特点，因此实际选择时不能只比较 Loss 数值，还需要结合 AUC、NDCG、校准指标以及线上 A/B 实验进行综合判断。

在精排模型实践案例中，本章结合典型工业网络结构，说明了特征输入与 Embedding、特征提取与选择、特征交叉与融合、长短期兴趣建模、多模态表征、多任务学习、目标预测 Tower 以及辅助网络等模块的作用。工业级精排模型通常不是某一种单一结构，而是根据业务目标和线上约束，将这些模块进行组合。模型输出的多个 pxtr 还需要经过下游多任务融合，才能形成最终排序分数，因此精排模型的设计必须同时考虑模型内部表示学习和推荐链路中的接口协同。

最后，本章讨论了精排模型在工业实践中的主要优化方向，包括通过序列建模、特征交叉和多模态信息引入增量特征，通过加 label 和多任务学习覆盖更加全面的用户行为反馈，以及通过多目标感知建模、pxtr 校准、任务融合和粗精排链路一致性提升整体推荐收益。同时，模型迭代还需要关注特征覆盖率、数据和版本一致性、服务延迟、资源成本、异常监控以及上下游依赖关系等工程问题。

12.9 参考文献

[1] Chris Burges et al. “Learning to rank using gradient descent”. In: *Proceedings of the 22nd international conference on Machine learning*. 2005, pp. 89–96.

[2] Zhe Cao et al. “Learning to rank: from pairwise approach to listwise approach”. In: *Proceedings of the 24th international conference on Machine learning*. 2007, pp. 129–136.

[3] Qiwei Chen et al. “Behavior sequence transformer for e-commerce recommendation in alibaba”. In: *Proceedings of the 1st international workshop on deep learning practice for high-dimensional sparse data*. 2019, pp. 1–4.

[4] Heng-Tze Cheng et al. “Wide & deep learning for recommender systems”. In: *Proceedings of the 1st workshop on deep learning for recommender systems*. 2016, pp. 7–10.

[5] Damai Dai et al. “Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models”. In: *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*. 2024, pp. 1280–1297.

[6] Tri Dao. “Flashattention-2: Faster attention with better parallelism and work partitioning”. In: *International Conference on Learning Representations*. Vol. 2024. 2024, pp. 35549–35562.

[7] Tri Dao et al. “Flashattention: Fast and memory-efficient exact attention with io-awareness”. In: *Advances in neural information processing systems* 35 (2022), pp. 16344–16359.

[8] Mor Geva et al. “Transformer feed-forward layers are key-value memories”. In: *Proceedings of the 2021 conference on empirical methods in natural language processing*. 2021, pp. 5484–5495.

[9] Huifeng Guo et al. “DeepFM: a factorization-machine based neural network for CTR prediction”. In: *arXiv preprint arXiv:1703.04247* (2017).

[10] Wang-Cheng Kang and Julian McAuley. “Self-attentive sequential recommendation”. In: *2018 IEEE international conference on data mining (ICDM)*. IEEE. 2018, pp. 197–206.

[11] Jianxun Lian et al. “xdeepfm: Combining explicit and implicit feature interactions for recommender systems”. In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2018, pp. 1754–1763.

[12] Tsung-Yi Lin et al. “Focal loss for dense object detection”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2980–2988.

[13] Jiaqi Ma et al. “Modeling task relationships in multi-task learning with multi-gate mixture-of-experts”. In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2018, pp. 1930–1939.

[14] Maxim Naumov et al. “Deep learning recommendation model for personalization and recommendation systems”. In: *arXiv preprint arXiv:1906.00091* (2019).

[15] Wentao Ouyang et al. “Representation learning-assisted click-through rate prediction”. In: *arXiv preprint arXiv:1906.04365* (2019).

[16] Yanru Qu et al. “Product-based neural networks for user response prediction”. In: *2016 IEEE 16th international conference on data mining (ICDM)*. IEEE. 2016, pp. 1149–1154.

[17] Weiping Song et al. “Autoint: Automatic feature interaction learning via self-attentive neural networks”. In: *Proceedings of the 28th ACM international conference on information and knowledge management*. 2019, pp. 1161–1170.

[18] Jianlin Su et al. “Roformer: Enhanced transformer with rotary position embedding”. In: *Neurocomputing* 568 (2024), p. 127063.

[19] Hongyan Tang et al. “Progressive layered extraction (ple): A novel multi-task learning (mtl) model for personalized recommendations”. In: *Proceedings of the 14th ACM conference on recommender systems*. 2020, pp. 269–278.

[20] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[21] Xu Wang et al. “Home: Hierarchy of multi-gate experts for multi-task learning at kuaishou”. In: *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*. 2025, pp. 2638–2647.

[22] Ruobing Xie et al. “Deep feedback network for recommendation”. In: *Proceedings of the twenty-ninth international conference on international joint conferences on artificial intelligence*. 2021, pp. 2519–2525.

[23] Jiaqi Zhai et al. “Actions speak louder than words: Trillion-parameter sequential transducers for generative recommendations”. In: *arXiv preprint arXiv:2402.17152* (2024).

[24] Weinan Zhang, Tianming Du, and Jun Wang. “Deep Learning over Multi-field Categorical Data: –A Case Study on User Response Prediction”. In: *European conference on information retrieval*. Springer. 2016, pp. 45–57.

[25] Guorui Zhou et al. “Deep interest network for click-through rate prediction”. In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2018, pp. 1059–1068.