第三部分

推荐系统核心算法

第 10 章 召回模型

在前面的章节中我们介绍了推荐系统的级联式架构以及该架构下的召回、粗排、精排、重排、混排还有生成式推荐这些具体模块。从本章开始，我们将视角从模块的维度进一步下探到每一个模块中核心的模型算法维度。本章我们将重点讨论推荐系统中的召回模型。召回模型是推荐系统中至关重要的一环，其主要任务是从海量候选物品中快速筛选出与用户兴趣相关的子集，为后续的排序和精排阶段提供高质量的候选集合。

10.1 协同过滤召回

协同过滤利用用户行为相似性进行推荐，包括基于用户和基于物品方法。适合历史行为丰富的场景，但冷启动能力有限。协同过滤方法是推荐系统中最基石性的方法，其核心思想为：

定义 10.1 (协同过滤)

协同过滤是一种基于用户行为相似性的推荐方法，通过分析用户或物品之间的相似性来进行推荐。

协同过滤方法本质上也是一种根据用户与物品交互行为进行数据驱动的方法。直观的理解协同过滤方法，如果用户 A 和 B 同时与 Item I 交互过，那么可以认为用户 A 和 B 在某种程度上具有相似的兴趣偏好；如果用户 A 和 Item I_1 以及 Item I_2 都有交互记录，那么可以认为用户 A 和 Item I_1 以及 Item I_2 在某种程度上具有相似的兴趣偏好。基于这种假设，协同过滤方法通过计算用户之间的相似性或者物品之间的相似性来进行推荐。协同过滤的思想不仅深刻影响着推荐系统的召回算法，同时也对粗排、精排、重排等模块的算法设计产生了深远的影响。本质上说，粗排、精排、重排等模块的算法设计都是在协同过滤思想的基础上进行扩展和优化的。

协同过滤方法主要分为两类：基于用户的协同过滤（User-based Collaborative Filtering, User CF）和基于物品的协同过滤（Item-based Collaborative Filtering, Item CF）。前者通过计算用户之间的相似性来推荐物品，而后者则通过计算物品之间的相似性来进行推荐。协同过滤方法简单直观，但在面对新用户或新物品时，容易出现冷启动问题。此外，协同过滤方法在处理大规模数据时可能面临计算复杂度高的问题，因此在实际应用中，通常会结合其他召回策略以提高推荐效果。

10.1.1 ItemCF 召回

ItemCF（Item-based Collaborative Filtering，基于物品的协同过滤）是工业界应用最广泛的传统召回算法之一。该模型并不是寻找与当前用户相似的用户，而是寻找与用户历史喜欢物品相似的其他物品。因此，ItemCF 更加关注“物品之间的关联关系”。

为了更直观地理解 ItemCF，我们先来看一个简单的例子，如图10.1所示。假设某位用户曾经观看过《复仇者联盟》《钢铁侠》和《蜘蛛侠》三部电影。由于大量用户都会连续观看这些电影，因此系统可以统计得到《钢铁侠》和《美国队长》具有较高的相似度，《蜘蛛侠》和《毒液》具有较高的相似度。当该用户再次访问推荐系统时，即使他从未观看过《美国队长》或《毒液》，系统仍然可以根据历史观看物品之间的相似关系，将这些电影推荐给用户。

我们可以看出，ItemCF 模型的核心思想如下：

定义 10.2

ItemCF 推荐的依据并不是用户之间是否相似，而是用户历史交互过的物品能够“投票”推荐与自己相似的其他物品。因此，ItemCF 算法的关键就在于如何计算物品之间的相似度。

具体来说，ItemCF 通常基于用户-物品交互矩阵进行建模。假设存在用户-物品交互矩阵 $\mathbf{R}$，其中 R_{ui} 表示用户 u 对物品 i 的交互行为（例如点击、点赞、购买、收藏等）。ItemCF 首先计算任意两个物品之间的相似度，从而构建整个物品相似度矩阵 $\mathbf{S}$。

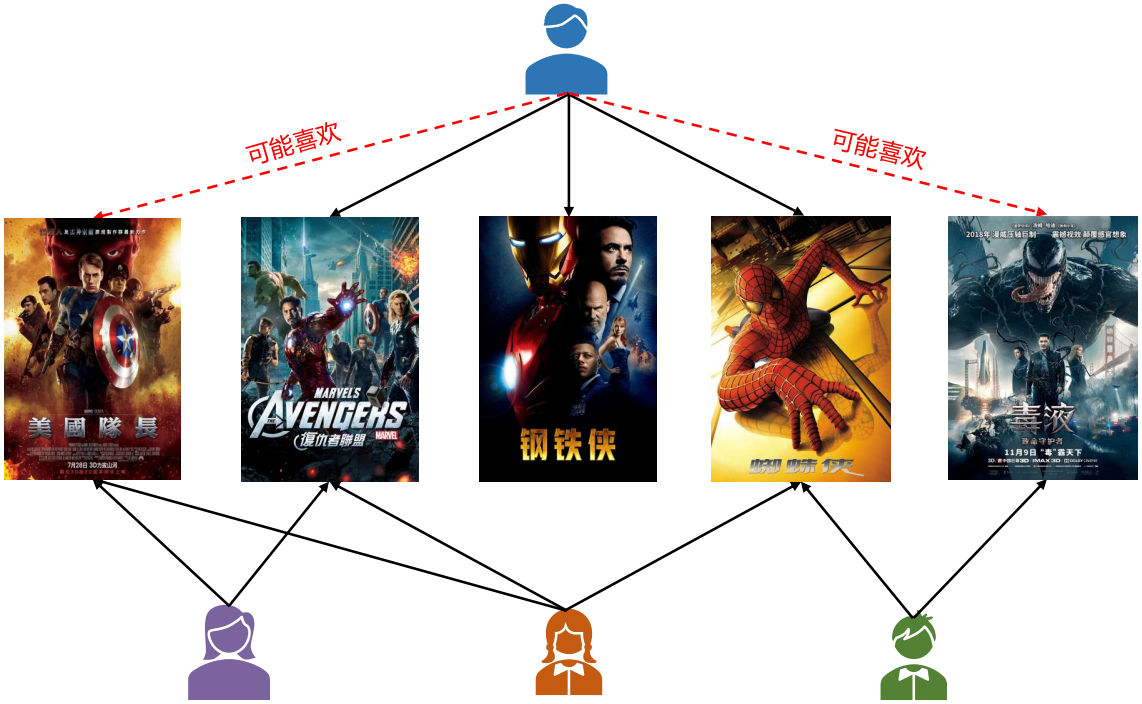

图 10.1: ItemCF 的简单示例。

最常见的相似度计算方式是余弦相似度，其定义如下：

$$S_{ij} = \frac{\mathbf{R}_i \cdot \mathbf{R}_j}{\|\mathbf{R}_i\|\|\mathbf{R}_j\|} = \frac{\sum_u R_{ui} \cdot R_{uj}}{\sqrt{\sum_u R_{ui}^2} \cdot \sqrt{\sum_u R_{uj}^2}} \tag{10.1}$$

其中，S_{ij} 表示物品 i 和物品 j 之间的相似度，$\mathbf{R}_i = (R_{1i}, R_{2i}, \ldots, R_{\|U\|i})$，$R_{ui}$ 表示用户 u 对物品 i 的行为值，可以是点击、购买、评分或隐式反馈等。通过计算所有物品之间的相似度，ItemCF 能够为每个用户推荐与其历史交互物品相似的其他物品，从而实现个性化推荐。

由于在工业级推荐场景中，大部分 ItemCF 都基于隐式反馈进行学习，因此 R_{ui} 通常定义如下：

$$R_{ui} = \begin{cases} 1, & \text{如果用户 } u \text{ 与物品 } i \text{ 有交互行为} \\ 0, & \text{否则} \end{cases} \tag{10.2}$$

基于此定义，R_{ui} 满足：

$$R_{ui}^2 = R_{ui} \tag{10.3}$$

因此有如下公式成立：

$$\sum_u R_{ui}^2 = \sum_u R_{ui} = N(i) \tag{10.4}$$

这里 $N(i)$ 表示与物品 i 有交互行为的用户数量。同时，我们有：

$$\sum_u R_{ui} R_{uj} = |N(i) \cap N(j)| \tag{10.5}$$

成立。将上面的式子代入余弦相似度公式中，就可以得到更常见的 ItemCF 相似度计算公式，即：

$$S_{ij} = \frac{|N(i) \cap N(j)|}{\sqrt{|N(i)| \cdot |N(j)|}} \tag{10.6}$$

上述公式具有十分直观的含义。分子 $|N(i) \cap N(j)|$ 表示同时与物品 i 和 j 发生过交互的用户数量，反映了两个物品共同被用户消费的程度；而分母 $\sqrt{|N(i)||N(j)|}$ 则起到了归一化作用，用于消除热门物品带来的影响。如果两个热门物品都拥有大量用户，仅仅依靠共同用户数量往往会高估它们之间的相似程度，而余弦归一化能够在一定程度上缓解这一问题。

除此之外，工业界还会使用其他相似度计算方法，比如 Jaccard 集合相似度、皮尔逊相关系数等。其中，Jaccard 相似度定义为：

$$S_{ij}^{\text{Jaccard}} = \frac{|N(i) \cap N(j)|}{|N(i) \cup N(j)|} \tag{10.7}$$

这里 $|N(i) \cap N(j)|$ 表示同时与物品 i 和 j 发生过交互的用户数量，$|N(i) \cup N(j)|$ 表示与物品 i 或 j 发生过交互的用户总数。Jaccard 相似度在计算时考虑了两个物品的联合用户集合大小，因此在某些场景下能够更好地反映物品之间的相似性。而皮尔逊相关系数则定义为：

$$S_{ij}^{\text{Pearson}} = \frac{\sum_u (R_{ui} - \bar{R}_i)(R_{uj} - \bar{R}_j)}{\sqrt{\sum_u (R_{ui} - \bar{R}_i)^2} \sqrt{\sum_u (R_{uj} - \bar{R}_j)^2}} \tag{10.8}$$

其中 $\bar{R}_i$ 表示物品 i 的平均评分。不同的相似度计算方法在实际应用中可能会产生不同的推荐效果，因此在工业实践中，通常会根据具体场景和数据特点选择最合适的相似度计算方法。

基于上面这些物品相似度定义，具体在对用户 u 推荐物品 i 的时候，我们可以计算基于 ItemCF 的相似度来计算出用户 u 与物品 i 之间的兴趣打分：

$$p(u, i) = \sum_{j \in N(u)} S_{ij} R_{uj} \tag{10.9}$$

其中 $N(u)$ 表示与用户 u 有过交互行为的物品集合，S_{ij} 表示物品 i 和 j 之间的相似度，R_{uj} 表示用户 u 对于已经有交互的物品 j 的行为取值。假设一个用户最近浏览了三个商品，分别对应物品 A、B 和 C。召回系统分别查询三个物品对应的 Top-K 相似物品列表。如果候选物品 D 同时出现在 A 和 B 的相似列表中，则最终兴趣得分为

$$p(u, D) = S_{AD} + S_{BD} \tag{10.10}$$

若进一步考虑用户行为权重，则也可以写成

$$p(u, D) = S_{AD} R_{uA} + S_{BD} R_{uB} \tag{10.11}$$

其中购买行为、收藏行为或长时观看行为通常具有更高的权重，因此能够产生更大的推荐贡献。最终，系统按照兴趣得分 $p(u, i)$ 排序，选取得分最高的 Top-K 物品作为 ItemCF 召回结果。

在实际工业系统中，ItemCF 召回通常会离线计算所有物品之间的 Top-K 相似物品，并构建 ItemCF 倒排索引（Item Similarity Index）。在线召回时，只需读取用户最近交互过的若干个物品，查询对应的相似物品列表，再根据相似度进行累加打分即可，因此在线计算复杂度非常低。

10.1.2 UserCF 召回

UserCF（User-based Collaborative Filtering，基于用户的协同过滤）与 ItemCF 正好相反，它并不是寻找与用户历史物品相似的其他物品，而是首先寻找与当前用户兴趣最相似的其他用户，再利用这些相似用户喜欢的物品完成推荐。因此，UserCF 更加关注“用户之间的兴趣相似性”。

这里我们还是举一个简单的例子来说明 UserCF 模型的具体思想。如图10.2所示，假设用户 A 最近观看了《流浪地球》、《三体》、《星际穿越》这三部电影，而用户 B 观看了《流浪地球》、《三体》、《火星救援》三部电影。由于这两位用户拥有较高的历史重合度，因此系统认为两位用户具有相似的兴趣偏好。如果用户 B 后来又观看了《沙丘》，而用户 A 尚未观看，那么系统便可以将《沙丘》推荐给用户 A。

因此，UserCF 模型主要包括两个步骤：

1. 计算目标用户与其他所有用户之间的相似度；
2. 找到最相似的 Top-K 用户，并根据这些用户历史交互过的物品完成推荐。

UserCF 模型同样基于用户-物品的交互矩阵 $\mathbf{R}$ 进行建模。设 $\mathbf{R}_A = (R_{A1}, R_{A2}, \cdots, R_{A|I|})$ 表示用户 A 对应的行为向量，$\mathbf{R}_B$ 表示用户 B 对应的行为向量，则用户之间的余弦相似度定义为

$$S_{AB} = \frac{\mathbf{R}_A \cdot \mathbf{R}_B}{\|\mathbf{R}_A\| \|\mathbf{R}_B\|} = \frac{\sum_i R_{Ai} R_{Bi}}{\sqrt{\sum_i R_{Ai}^2} \sqrt{\sum_i R_{Bi}^2}} \tag{10.12}$$

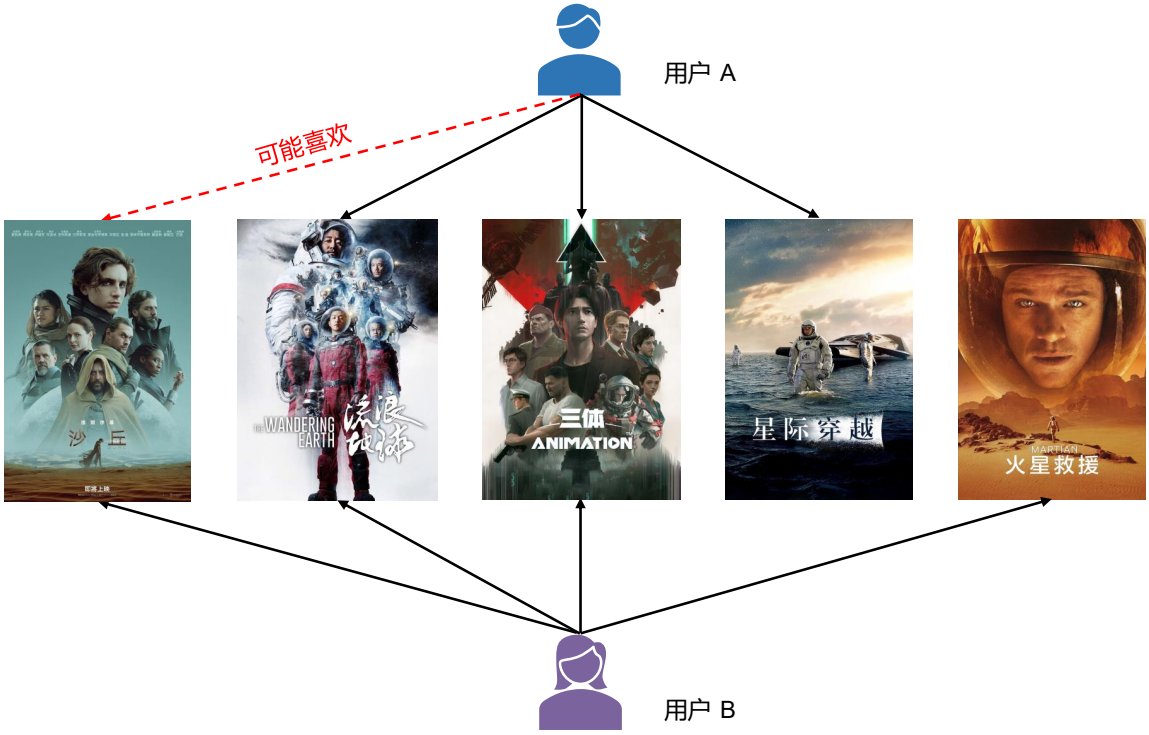

图 10.2: UserCF 的简单示例。

在工业推荐系统中，大多数场景仍然采用隐式反馈。因此我们有

$$R_{ui} = \begin{cases} 1, & \text{用户与物品发生交互} \\ 0, & \text{否则} \end{cases} \tag{10.13}$$

令 $N(A)$ 表示用户 A 历史交互过的物品集合，则有

$$\sum_i R_{Ai} = |N(A)| \tag{10.14}$$

同时，有以下公式成立：

$$\sum_i R_{Ai} R_{Bi} = |N(A) \cap N(B)| \tag{10.15}$$

因此，UserCF 相似度可以进一步写成

$$S_{AB} = \frac{|N(A) \cap N(B)|}{\sqrt{|N(A)|\,|N(B)|}} \tag{10.16}$$

得到用户相似度之后，可以利用相似用户的历史行为预测目标用户对候选物品 i 的兴趣程度：

$$p(A, i) = \sum_{B \in \mathcal{N}(A)} S_{AB} R_{Bi} \tag{10.17}$$

其中 $\mathcal{N}(A)$ 表示与用户 A 最相似的 Top-K 用户集合。

可以看到，该公式与 ItemCF 具有完全对偶（Dual）的结构。ItemCF 利用用户历史交互过的物品进行“投票”，而 UserCF 则利用兴趣相似的用户进行“投票”。最终，根据兴趣得分 $p(A, i)$ 排序即可得到 UserCF 的召回结果。

下面我们主要针对 ItemCF 与 UserCF 在工业界的具体落地与应用进行讨论。

笔记 虽然从数学形式来看，ItemCF 与 UserCF 具有高度相似的结构，但二者在工业界的落地难度却存在较大的差异。其根本原因在于二者计算相似度的对象不同。ItemCF 需要维护的是物品之间的相似关系，而 UserCF 需要维护的是用户之间的相似关系。对于绝大多数互联网推荐系统而言，用户规模通常远大于物品规模。例如，在直播推荐场景中，平台每天可能拥有数千万甚至上亿活跃用户，而同时在线直播间数量通常只有几十万到几百万。

此时，物品数量远小于用户数量，因此离线计算物品之间的相似度矩阵仍然具有一定的可行性，并且可以进一步保留每个物品 Top-K 个最相似的邻居，构建 ItemCF 索引用于在线快速召回。

相比之下，UserCF 需要计算和维护海量用户之间的相似关系，其时间复杂度和存储复杂度都会随着用户规模快速增长。当用户规模达到千万甚至上亿级别时，完整构建用户相似度矩阵几乎难以实现，即使采用离线计算，其计算资源、存储资源以及更新成本也十分巨大。因此，在工业实践中，很少直接维护完整的 UserCF 相似度矩阵，而更多采用近似最近邻（Approximate Nearest Neighbor，ANN）、Embedding 检索、OnlineCF 等方法，对用户相似关系进行近似建模。关于 ANN 算法以及工业界的具体实践我们在后续第17章节中进行详细介绍。

此外，用户兴趣通常具有较强的时效性，会随着时间不断发生变化，因此用户相似关系也需要频繁更新；而物品之间的关联关系通常更加稳定，更新频率相对较低。这也进一步降低了 ItemCF 的维护成本，使其成为工业界应用最广泛的传统协同过滤算法之一。

总体而言，随着 Embedding 召回、双塔模型以及图神经网络召回等深度学习方法的发展，传统的 ItemCF 和 UserCF 已经较少作为独立的工业召回方案直接使用，而是在学术界与小规模数据分析时使用较多。然而，ItemCF 与 UserCF 这两类算法所体现的协同过滤思想仍然贯穿于现代推荐系统的发展过程中。后续介绍的 OnlineCF、Neural Collaborative Filtering（Neural CF）以及 Embedding 召回等方法，本质上都是对协同过滤思想在表示学习、相似度建模和在线更新等方面的进一步扩展与演进。因此，理解 ItemCF 和 UserCF 不仅有助于掌握协同过滤的基本思想，也能够帮助大家更好地理解后续 OnlineCF、Neural CF、双塔模型以及 Embedding 召回等现代召回算法的设计思路。

10.1.3 Online CF 召回

前面介绍的 ItemCF 和 UserCF 本质上都属于离线协同过滤（Offline Collaborative Filtering）。它们通常基于历史行为数据离线计算用户或物品之间的相似度，并定期构建相似度矩阵。然而，在工业级推荐系统中，用户兴趣具有较强的时效性，例如热点新闻、直播内容以及短视频等场景，用户兴趣可能在数分钟内快速变化。如果仍然依赖每天甚至每小时更新一次的离线相似度矩阵，推荐结果往往难以及时反映用户最新的兴趣变化。

为了解决这一问题，工业界提出了 Online Collaborative Filtering（Online CF）方法。

定义 10.3 (Online Collaborative Filtering)

Online CF 是一种基于实时行为流动态维护用户或物品共现关系的协同过滤方法。与传统离线协同过滤不同，Online CF 无需预先计算完整的相似度矩阵，而是在用户行为发生后实时更新物品之间的关联关系，并利用最新统计结果完成在线召回。

♣

工业界通常采用 Online ItemCF（Online I2I）的实现方式，而较少采用 Online UserCF。一方面，物品数量通常远小于用户数量，维护 Item 之间的共现关系具有更低的计算和存储成本；另一方面，物品之间的关联关系相对更加稳定，更适合进行实时维护。因此，在工业推荐系统中，Online CF 通常默认指 Online ItemCF。

10.1.3.1 整体工程架构

Online CF 通常采用“Kafka + Flink + Redis”的流式计算架构，其整体流程如下：

$$\text{Kafka} \longrightarrow \text{Flink} \longrightarrow \text{Redis} \longrightarrow \text{Online Retrieve}$$

其中，Kafka 负责实时采集用户曝光、点击、点赞、关注、分享、长观看等行为日志；Flink 持续消费行为流，对用户最近一段时间内的行为进行聚合统计，并实时更新 Item 之间的共现关系；Redis 则维护 Item Pair 共现分数以及每个 Item 对应的 Top-K 相似 Item 索引，在线召回阶段直接查询 Redis 即可快速完成 Online CF 召回。

相比传统离线 ItemCF 每天更新一次相似度矩阵，Online CF 能够在分钟级甚至秒级完成共现关系更新，因此能够快速响应热点内容以及用户兴趣变化。

10.1.3.2 实时更新流程

结合工业界实际推荐系统，Online CF 召回通常采用固定时间窗口（Time Window）对用户实时行为的数据流进行聚合。例如，将窗口大小设置为 2 分钟，即每 2 分钟统计一次最近产生的用户行为，并更新 Item 之间的共现关系。需要注意的是，这里的用户实时行为的数据流通常是在用户特征和行为反馈标签进行完样本拼接之后的数据流。

基于整个用户行为数据流的整个实时统计过程可以采用 Flink 框架来消费实时数据流的 Kafka Topic 实现，其核心流程如代码10.1所示。

代码 10.1: Online CF 中基于 Flink 实时统计 Item 共现关系

```
public class OnlineCFJob {
    public static void main(String[] args) throws Exception {
        // 创建Flink执行环境
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        // 1. 从Kafka读取用户行为日志
        DataStream<UserBehaviorEvent> behaviorStream = env.fromSource(KafkaSourceBuilder.build(), ↩
            ↪ WatermarkStrategy.noWatermarks(), "BehaviorSource");
        // 2. 生成Item共现Pair
        SingleOutputStreamOperator<Tuple2<ItemPair, Double>> pairStream = behaviorStream.flatMap(↩
            ↪ new GenerateItemPair());
        // 3. 统计两分钟窗口内的Item共现分数
        SingleOutputStreamOperator<Tuple2<ItemPair, Double>> pairScore = pairStream
                .keyBy(record -> record.f0)
                .window(TumblingProcessingTimeWindows.of(Time.minutes(2)))
                .reduce((left, right) -> Tuple2.of(left.f0, left.f1 + right.f1));
        // 4. 更新Redis中的Pair Score
        pairScore.addSink(new PairScoreSink());
        // 5. 更新每个Item对应的Top-K索引
        pairScore.map(new BuildTopKIndex()).addSink(new ItemIndexSink());
        env.execute("OnlineCFPipeline");
    }

    /**
     * 根据用户行为生成Item共现Pair
     */
    static class GenerateItemPair implements FlatMapFunction<UserBehaviorEvent, Tuple2<ItemPair, ↩
        ↪ Double>> {
        @Override
        public void flatMap(UserBehaviorEvent event, Collector<Tuple2<ItemPair, Double>> out) {
            List<Long> historyItems = event.getHistoryItems();
            List<ItemWeight> currentItems = event.getCurrentItems();

            for (Long leftItem : historyItems) {
                for (ItemWeight item : currentItems) {
                    ItemPair pair = new ItemPair(leftItem, item.getItemId());
                    out.collect(Tuple2.of(pair, item.getWeight()));
                }
            }
```

```
        }
    }
}
```

从上述代码可以看出，Online CF 算法在实际工业界推荐系统中落地时采用窗口聚合而不是逐条更新，一方面能够充分利用 Flink 的流式计算能力，另一方面也能够显著减少 Redis 的更新次数，从而降低存储系统的写入压力。

10.1.3.2.1 （1）实时更新共现关系 在 Flink 任务完成窗口聚合后，需要根据用户最近行为实时更新 Item 之间的共现关系。假设用户 u 最近一段时间内的历史行为序列为

$$\mathcal{H}_u = \{i_1, i_2, \cdots, i_m\} \tag{10.18}$$

其中，m 表示用户历史行为序列的物品截断数量，通常 m 可以取 20 或 30，表示取用户最近的 20 或 30 个有交互的物品。假设当前窗口中与用户最新产生交互的物品为 j，则对于截断的最近历史序列中每一个物品 $i \in \mathcal{H}_u$，都会更新对应的共现分数：

$$Score(i, j) \leftarrow Decay(Score(i, j)) + w(j) \tag{10.19}$$

其中，$w(j)$ 表示当前行为对应的权重，$Decay(\cdot)$ 表示时间衰减函数。在工业界推荐系统中，Online CF 的实时 Flink 处理任务通常会根据行为价值赋予不同的权重，例如点击、长观看、点赞、关注、分享等行为的重要程度依次增加，因此物品 j 的行为权重通常可以表示为

$$w(j) = \sum_k \alpha_k x_k(j) \tag{10.20}$$

其中，$x_k(j)$ 表示第 k 种行为是否发生，α_k 表示对应行为的权重系数。通常只有权重超过一定阈值的行为才会参与共现统计，以减少噪声样本带来的影响。

需要注意的是，不同类型的行为可能具有不同的重要性，因此，在实际应用中需要根据业务场景合理设置权重系数。比如在短视频推荐场景中，长时观看行为通常比点击行为更能反映用户的兴趣偏好，因此可以为长时观看行为设置更高的权重。而点赞、关注等稀疏的互动行为则可以设置更高的权重，以突出其在共现统计中的重要性。一种常见的行为权重设置如下表10.1所示：

表 10.1: Online CF 召回行为权重设置示例。

行为	权重
点击	1
长观看	3
点赞	5
关注	8
分享	10

此外，为了使模型能够快速响应热点变化，历史共现分数通常采用时间衰减机制：

$$Decay(x) = xe^{-\lambda \Delta t} \tag{10.21}$$

其中，Δt 表示距离上一次更新时间，λ 表示时间衰减系数。这样近期行为能够具有更大的影响力，而长期未发生交互的共现关系则会逐渐减弱。

在动态更新完物品 i 与 j 的共现分数之后，通常会将共现分数存储到 Redis 中。在 Redis 中一般维护两类数据，其中：

$$\texttt{Key:item\_A\_item\_B} \rightarrow (score, timestamp) \tag{10.22}$$

用于保存物品对之间的共现分数；同时，对于每一个物品，还维护对应 Top-K 相似物品的索引，即：

$$\texttt{Key:item\_A} \rightarrow [(item_1, s_1), \cdots, (item_K, s_K)] \tag{10.23}$$

注意前者物品对的共现分数通常会设置较长的 Redis 过期时间，而每个物品的 Top-K 索引则会采用较短过期时间，从而保证近期的热点变化能够快速同步到在线召回服务中。

10.1.3.2.2 （2）在线召回流程 当用户的一次在线请求到来之后，工业界推荐系统首先会获取到该用户最近的多个行为序列，比如最近点赞、长观看、有效观看以及关注等行为对应的物品集合：

$$\mathcal{B}_u = \{b_1, b_2, \cdots, b_n\} \tag{10.24}$$

随后，线上 Online CF 召回服务会依次查询每个行为对应的 Top-K 相似 Item 索引：

$$Retrieve(b_i) = TopK(b_i) \tag{10.25}$$

最终，召回服务将多个行为对应的召回结果进行合并、去重和排序，并截断得到最终的 Online CF 召回结果。由于整个过程仅涉及到 Redis 中的 Key-Value 查询，因此，Online CF 召回通常能够在毫秒级完成整个召回过程，完全可以满足工业级推荐系统中低延迟、高并发的要求。此外，为了进一步降低 Redis 的存储空间，Online CF 召回的 Flink 任务通常会将每个物品的 Top-K 索引序列化之后采用 Base64 进行压缩编码存储；同时，每个物品只需要维护它对应 Top-K 个相似物品即可，而无需保存完整的 Item 共现矩阵，这样也可以显著降低整个 Flink 任务的存储开销。

10.1.3.3 Online CF 召回算法分析

相比传统离线 ItemCF 召回，Online CF 召回最大的优势在于它能够非常实时地基于用户行为数据流来更新物品之间的共现关联关系。由于 Online CF 召回通常能够快速地响应热点内容以及用户兴趣的变化，因此，在短视频推荐、直播推荐和信息流推荐等时效性要求较高的场景中该方法具有广泛的应用。

另一方面，Online CF 召回并没有改变协同过滤的基本思想，而是将传统的离线统计扩展为实时流式计算，具有工程实现简单、可解释性强以及部署方案成熟等优点。通过 Kafka 消息队列、Flink 框架以及 Redis 缓存等实时计算组件，Online CF 召回能够以较低的计算代价持续维护最新的 Item 共现关系，从而有效弥补离线 ItemCF 召回时效性不足的问题。

总的来说，Online CF 召回仍然属于基于共现统计的召回方法，推荐结果高度依赖于用户历史行为，难以学习更加抽象复杂的用户兴趣表示，同时 Online CF 召回对于冷启物品以及长尾物品的建模能力也相对有限，从而导致整个召回的泛化性不足。因此，在现代工业级推荐系统中，Online CF 召回通常只是作为用户实时兴趣召回的一个组成部分，它与 Embedding 召回、图召回等方法并行执行，从而共同构成了多路召回的框架。其中，Online CF 召回主要负责捕捉用户的短期兴趣和热点变化，而 Embedding 召回则可以建模用户长期稳定的兴趣偏好。这两类方法在实际推荐系统中可以相互补充，共同提升整个召回阶段的覆盖率和准确率。

10.1.4 Swing 召回

前面介绍的 ItemCF、UserCF 以及矩阵分解等方法，本质上都属于协同过滤（Collaborative Filtering）算法。其中，ItemCF 主要根据物品之间的共现关系计算相似度，再利用相似物品完成推荐。由于 ItemCF 具有实现简单、可解释性强以及在线部署方便等优点，因此至今仍然广泛应用于工业界推荐系统。

然而，传统 ItemCF 通常采用共现次数或余弦相似度来衡量两个物品之间的相关性。这种方法虽然简单有效，但也存在一些明显的不足。例如，对于活跃用户而言，由于其浏览了大量物品，因此会人为增加不同物品之间的共现次数，使得大量实际上并不相似的物品也会产生较高的相似度。此外，热门物品由于与大量用户发生交互，也容易形成大量无意义的共现关系，从而影响最终的召回质量。

为了解决上述问题，阿里巴巴在 2020 年发表的论文《Large Scale Product Graph Construction for Recommendation in E-commerce》中提出了 Swing 算法 [58]。Swing 召回可以看作是传统 ItemCF 召回的一种改进方法，其核心思想是在统计物品共现关系时，不仅考虑两个物品的共同用户数量，还进一步考虑共同用户之间的兴趣相似程度，从而更加准确地刻画物品之间的关联关系。

下面我们给出 Swing 召回的定义，即：

定义 10.4 (Swing 召回)

Swing 召回是一种基于用户-物品二部图（User-Item Bipartite Graph）的协同过滤召回算法。相比传统 ItemCF 仅利用物品共现次数计算相似度，Swing 进一步引入共同用户之间的兴趣重叠程度，对活跃用户进行降权，从而获得更加准确的 Item 相似度。

10.1.4.1 Swing 相似度计算

与 ItemCF 召回类似，Swing 召回仍然建立在用户-物品交互图之上。假设

$$U_i = \{u \mid (u, i) \in E\} \tag{10.26}$$

表示与物品 i 发生过交互的用户集合，那么对于两个物品 i 和 j，首先寻找同时与二者发生过交互的共同用户，即 $U_i \cap U_j$。与传统 ItemCF 召回不同的是，Swing 召回并不会直接统计共同用户数量，而是进一步考虑这些共同用户之间的兴趣相似程度。假设用户 u 和 v 历史交互物品集合分别为 I_u 和 I_v，则二者共同兴趣可以表示为

$$|I_u \cap I_v| \tag{10.27}$$

于是，Swing 召回定义物品 i 与 j 之间的相似度为

$$S(i, j) = \sum_{u,v \in U_i \cap U_j} \frac{1}{\alpha + |I_u \cap I_v|} \tag{10.28}$$

其中 α 表示平滑参数。

从该公式可以看出，当两位共同用户拥有大量相同历史行为时，说明二者兴趣十分接近，此时 $|I_u \cap I_v|$ 较大，对应贡献会变小；而当两位用户兴趣差异较大，仅在少数几个物品上发生交集时，则能够为 Item 之间建立更加有效的关联关系。因此，Swing 实际上并不是简单统计用户数量，而是在用户对之间进行二阶建模，从而能够更加准确地区分真正具有代表性的共现关系。

为了进一步降低活跃用户带来的噪声，Swing 通常还会引入用户活跃度权重，对不同用户赋予不同的重要性。设用户 u 历史交互物品数量为 $|I_u|$，则其权重一般定义为

$$w_u = \frac{1}{\sqrt{|I_u|}} \tag{10.29}$$

于是，完整的 Swing 相似度计算公式可以表示为

$$S(i, j) = \sum_{u,v \in U_i \cap U_j} w_u \cdot w_v \cdot \frac{1}{\alpha + |I_u \cap I_v|} \tag{10.30}$$

其中，用户历史行为越多，其对应权重越小。因此，高活跃用户不会过度影响 Item 之间的相似度计算，从而有效缓解热门用户带来的噪声问题。

Swing 算法的示例如图10.3所示。图中，中心物品 h 表示当前需要计算相似物品的目标 Item，$A \sim E$ 表示与物品 h 发生过交互的用户，其余节点 p、q、r、t、x 等表示这些用户还交互过的其他物品。对于每一个候选物品，Swing 都会枚举同时交互过物品 h 和该候选物品的用户对（User Pair），并根据这些用户之间共同兴趣的重叠程度计算最终的相似度。

以物品 p 为例，用户对 (A, B)、(A, C) 以及 (B, C) 都同时交互过物品 h 和物品 p，因此分别构成三个 Swing。其中，对于用户对 (A, B) 而言，二者除了共同交互过 h 和 p 之外，还共同交互了 t 和 r 两个物品，因此共有 3 个共同兴趣物品（即 h、p、t、r 中的三个 Swing），对应贡献为 $\frac{1}{1+3} = \frac{1}{4}$。而用户对 (A, C) 和 (B, C) 之间仅共

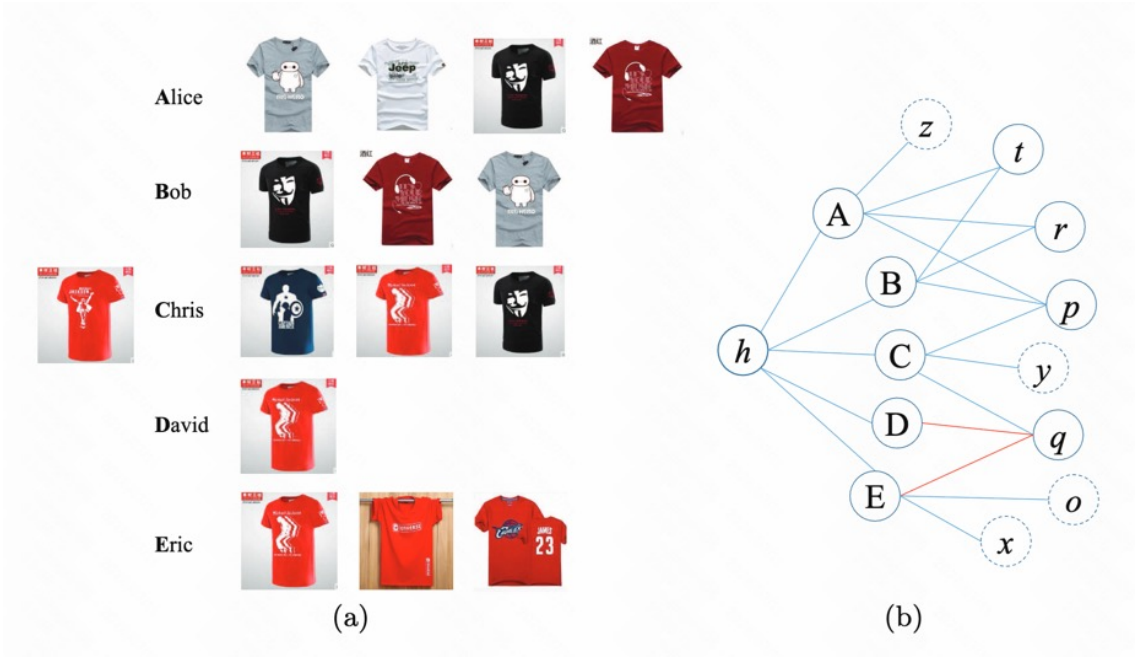

图 10.3: Swing 召回示意图。

同交互了 h 和 p 两个物品，因此分别贡献 $\frac{1}{1+1}=\frac{1}{2}$。最终，物品 p 与物品 h 之间的 Swing 相似度为

$$Swing(h,p)=\frac{1}{4}+\frac{1}{2}+\frac{1}{2}=1.25 \tag{10.31}$$

类似地，可以计算得到

$$Swing(h,q)=\frac{1}{2}+\frac{1}{2}+\frac{1}{2}=1.5 \tag{10.32}$$

而物品 r 和 t 仅对应一个 Swing，因此有

$$Swing(h,r)=Swing(h,t)=\frac{1}{2}. \tag{10.33}$$

最终得到物品之间的相似度排序为 $q>p>r=t>$ others。

从上述例子可以看出 Swing 召回的如下特点：

性质 Swing 并不是简单统计共同用户数量，而是进一步考虑共同用户之间的兴趣重叠程度。当两位用户共同交互的物品越多时，其贡献会按照

$$\frac{1}{\alpha+|I_u\cap I_v|} \tag{10.34}$$

进行衰减，从而降低兴趣过于相似或活跃用户带来的重复贡献；而由不同兴趣用户共同产生的共现关系则能够获得更高的权重。

因此，相比传统 ItemCF 仅依据共现次数计算相似度，Swing 能够更加准确地筛选出真正具有代表性的物品关联关系，提高最终召回结果的准确性和鲁棒性。下面我们给出 Swing 召回相似度计算的示例代码，如下所示：

代码 10.2: Swing 相似度计算示例

```
from itertools import combinations
from collections import defaultdict
def swing_similarity(user_items, alpha=1.0):
    item_users = defaultdict(set)
    for u, items in user_items.items():
        for i in items:
            item_users[i].add(u)
    sim = defaultdict(dict)
```

```
    items = list(item_users.keys())
    for i, j in combinations(items, 2):
        common_users = item_users[i] & item_users[j]
        score = 0.0
        for u, v in combinations(common_users, 2):
            overlap = len(set(user_items[u]) & set(user_items[v]))
            wu = 1.0 / (len(user_items[u]) ** 0.5)
            wv = 1.0 / (len(user_items[v]) ** 0.5)
            score += wu * wv / (alpha + overlap)
        sim[i][j] = score
        sim[j][i] = score
    return sim
```

10.1.4.2 离线 Swing 召回

工业界中的 Swing 召回通常采用离线批量计算的方式构建 Item 相似图。整个过程一般以天级或小时级用户行为日志作为输入，通过 Spark、Hive SQL、Flink 等分布式计算框架离线完成 Swing 相似度计算，并最终生成 Item Top-K 相似物品索引。

整个离线计算流程通常包括以下几个步骤：

1. **用户行为统计**。首先从 Hive 等离线数仓中读取用户行为日志，根据点击、长观看、点赞、购买等行为构建用户–物品交互二部图。实际工程中通常会过滤曝光质量较低或交互权重较小的行为，仅保留高质量行为参与 Swing 计算。
2. **共同用户构建**。对于每个用户，将其历史交互物品进行两两组合，生成 (i, j) 形式的物品 pair 对，同时记录产生该物品 pair 对的用户集合。然后，相同物品 pair 对对应的共同用户会被聚合到同一个 Reducer 中。
3. **Swing 相似度计算**。对于每一个物品 pair 对，进一步枚举共同用户之间的用户 pair 对，根据公式10.30计算每个 Swing 的贡献，并累加得到最终的 Swing 相似度。由于不同物品 pair 对之间相互独立，因此整个计算过程天然适合采用 MapReduce 或 Spark 进行并行计算。
4. **Top-K 索引构建**。完成全部物品 pair 对的相似度计算之后，对于每个物品按照 Swing 得分进行排序，仅保留 Top-K 个最相似物品，并构建物品相似图索引。
5. **索引发布**。最终生成的 Top-K 相似物品索引通常会导入图存储数据库、Redis 这类 KV 缓存或者分布式索引系统中，进而作为在线召回服务的数据查询来源。

整个离线 Swing 召回整体流程如图10.4所示。从计算过程来看，Swing 召回本质上属于典型的离线图构建任务。由于需要枚举大量共同用户以及用户 pair 对，其计算复杂度明显高于传统 ItemCF 召回，因此工业界通常采用 Spark 等分布式计算框架进行批量计算，并按照天级或小时级等周期来增量更新整个物品之间的 Swing 相似度图结构。

10.1.4.3 Online Swing 召回

离线 Swing 通常按照天级或小时级重新构建整个 Swing 相似图，但对于直播、短视频、热点新闻等实时性要求较高的场景，仅依赖离线更新难以及时反映用户兴趣变化。因此，工业界通常会进一步构建 Online Swing 召回，对 Swing 图进行实时增量更新，使物品之间的 Swing 关系能够随着用户行为而不断演化。

Online Swing 召回整体流程与 Online CF 召回类似，同样都采用”Kafka + Flink + Graph Storage” 的流式计算架构，其核心区别在于 Online Swing 召回的维护对象由物品共现矩阵变成了 Swing 相似图。整体流程如下：

$$\text{Kafka} \longrightarrow \text{Flink} \longrightarrow \text{Graph RPC} \longrightarrow \text{Graph Storage}$$

Online Swing 召回的整个更新过程通常包括以下几个步骤。

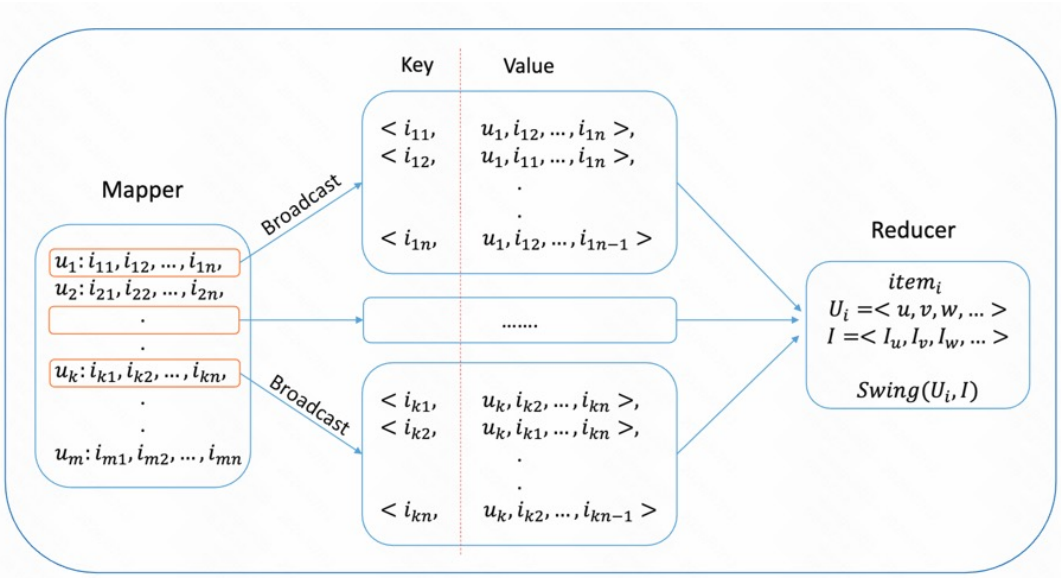

Figure 3: Parallelization implementation of Swing via MR framework

图 10.4: 离线 Swing 召回流程示意图。

1. **实时行为采集**。推荐系统首先持续采集用户曝光、点击、点赞、收藏、购买等行为日志，与推荐系统的用户请求数据进行样本拼接得到拼接后的数据，并实时写入 Kafka。随后，Flink 任务（或定时 Runner 服务）持续消费样本拼接数据流，对用户最近一段时间内的行为进行解析和聚合。
2. **实时构建 Swing 关系**。当用户产生新的交互行为 (u, i) 时，实时计算服务（如 Flink、Runner 或 Kafka Consumer）首先消费用户行为日志，并获取用户历史交互物品集合 $I_u = \{i_1, i_2, \cdots, i_m\}$。随后，将新交互物品 i 与历史物品集合中的每一个物品组成候选物品 pair 对，即：

$$\mathcal{P} = \{(i, i_k) \mid i_k \in I_u\} \tag{10.35}$$

对于每一个物品 pair 对 (i, i_k)，系统需要维护其共同用户集合

$$U_{i,i_k} = U_i \cap U_{i_k} \tag{10.36}$$

其中，U_i 表示与物品 i 发生过交互的用户集合。当新的用户 u 同时交互了物品 i 和 i_k 后，只需将用户 u 增量加入对应的共同用户集合，即

$$U_{i,i_k} \leftarrow U_{i,i_k} \cup \{u\} \tag{10.37}$$

随后，以新增用户 u 为中心，与共同用户集合中的每一个已有用户 $v \in U_{i,i_k}$ 构造新的用户 pair，并计算该用户对 Swing 相似度带来的增量贡献。根据 Swing 算法，相应的增量可以表示为

$$\Delta S(i, i_k) = \sum_{v \in U_{i,i_k} \setminus \{u\}} \frac{w_u w_v}{\alpha + |I_u \cap I_v|} \tag{10.38}$$

其中 $w_u = \frac{1}{\sqrt{|I_u|}}$ 和 $w_v = \frac{1}{\sqrt{|I_v|}}$ 分别表示用户 u 和用户 v 的活跃度权重，$|I_u \cap I_v|$ 表示用户 u 和用户 v 共同交互过的物品数量。

最终，仅需对对应物品 pair 的 Swing 分数进行增量更新，而无需重新遍历全部共同用户：

$$S(i, i_k) \leftarrow S(i, i_k) + \Delta S(i, i_k) \tag{10.39}$$

3. **Graph RPC 更新**。计算得到 Swing 增量之后，Flink 任务通过 Graph RPC 访问图存储服务 (Graph Storage Service)，对对应物品节点之间的边权进行更新。若边已经存在，则直接更新 Swing 分数；若边不存在，则创建新的图边。

图存储服务通常维护如下形式的邻接关系：

$$Item_i \rightarrow \{(Item_j, Swing_{ij}), (Item_k, Swing_{ik}), \cdots\}. \tag{10.40}$$

为了减少图存储系统的写入压力，工业界通常不会在每条用户行为到来时立即更新 Swing 图，而是先在 Flink 框架或流式计算框架中对一定时间窗口内产生的 Swing 增量进行聚合，然后再批量写入图存储服务。这样既能够减少网络 RPC 调用次数，又能够提高图数据库的写入吞吐能力。

4. **维护 Top-K 邻居**。当某条边权发生变化后，图存储服务会同步维护对应节点的 Top-K 邻居列表，仅保留 Swing 分数最高的若干个相似物品。在线召回阶段即可直接读取该邻接表，而无需再次遍历整个图。

下面我们给出 Online Swing 召回的示例代码，如下所示：

代码 10.3: Online Swing 实时增量更新流程

```
public class OnlineSwingJob {
    private static final double ALPHA = 1.0;
    private final KafkaConsumer<UserBehaviorEvent> consumer;
    private final UserGraphRpc userGraphRpc;
    private final PairGraphRpc pairGraphRpc;
    private final SwingGraphRpc swingGraphRpc;

    public OnlineSwingJob() {
        consumer = KafkaFactory.createConsumer("user_behavior");
        userGraphRpc = new UserGraphRpc();
        pairGraphRpc = new PairGraphRpc();
        swingGraphRpc = new SwingGraphRpc();
    }

    public void run() {
        while (consumer.hasNext()) {
            // 消费用户实时行为
            UserBehaviorEvent event = consumer.next();
            processBehavior(event);
        }
    }

    /**
     * 增量更新Swing图
     */
    private void processBehavior(UserBehaviorEvent event) {
        long userId = event.getUserId();
        long currentItem = event.getItemId();
        // Step1 查询用户历史行为
        List<Long> historyItems = userGraphRpc.getHistoryItems(userId);
        // Step2 遍历历史Item形成Item Pair
        for (Long historyItem : historyItems) {
            if (historyItem == currentItem) {
                continue;
            }
            updateSwing(userId, currentItem, historyItem);
        }
```

```
        // Step3 更新User -> Item图
        userGraphRpc.addEdge(userId, currentItem);
    }

    /**
     * 更新一个Item Pair对应Swing分数
     */
    private void updateSwing(long userId, long itemA, long itemB) {
        // 查询Item Pair对应共同用户
        Set<Long> commonUsers = pairGraphRpc.getCommonUsers(itemA, itemB);
        double deltaScore = 0.0;
        // 新用户与已有共同用户形成新的User Pair
        for (Long otherUser : commonUsers) {
            int overlap = userGraphRpc.getCommonItemCount(userId, otherUser);
            int degreeU = userGraphRpc.getItemCount(userId);
            int degreeV = userGraphRpc.getItemCount(otherUser);
            double weightU = 1.0 / Math.sqrt(degreeU);
            double weightV = 1.0 / Math.sqrt(degreeV);
            deltaScore += weightU * weightV / (ALPHA + overlap);
        }
        // 更新Swing分数
        swingGraphRpc.incrementScore(itemA, itemB, deltaScore);
        swingGraphRpc.incrementScore(itemB, itemA, deltaScore);
        // 更新Item Pair共同用户
        pairGraphRpc.addCommonUser(itemA, itemB, userId);
        // 动态维护Top-K邻居
        swingGraphRpc.refreshTopK(itemA);
        swingGraphRpc.refreshTopK(itemB);
    }
}
```

相比起离线 Swing 召回，Online Swing 召回能够持续维护动态图结构，使最新产生的用户行为能够快速传播到 Swing 相似图中，因此对于热点内容、爆款商品以及实时兴趣变化具有更好的响应能力。不过，由于 Swing 召回涉及共同用户关系以及兴趣重叠程度的计算，其在线更新复杂度明显高于 Online CF 召回，因此工业界通常采用窗口聚合、批量更新以及异步 Graph RPC 等工程优化方式，以保证整个图更新过程具有较高的吞吐率和稳定性。

10.1.4.4 Swing 召回模型分析

从模型结构来看，Swing 召回仍然属于基于统计的协同过滤方法，其核心区别在于**重新设计了物品之间的相似度计算方式**。相比传统 ItemCF 召回仅利用共同用户数量计算相似度，Swing 召回进一步考虑了共同用户之间的兴趣重叠程度，并对活跃用户进行了降权处理操作，因此 Swing 召回能够有效降低热门用户和热门物品带来的噪声与偏置，使最终得到的物品相似图具有更高的质量。

另一方面，Swing 召回并不需要模型训练，也无需学习 Embedding 向量，因此具有实现简单、可解释性强以及部署成本低等优点，非常适合作为工业推荐系统中的统计召回通路。不过，Swing 召回仍然依赖历史交互数据，无法利用用户画像、物品内容等丰富特征，也难以学习更加抽象的语义关系以及泛化能力。因此，在工业界推荐系统中，我们通常会将 Swing 召回作为多路召回中的一路统计召回，与 Online CF 召回、Embedding 召回、双塔召回以及图神经网络召回等共同使用，以兼顾整个推荐结果的准确性、覆盖率与实时性。

10.1.5 矩阵分解召回

除了前面介绍的 ItemCF 和 UserCF 之外，矩阵分解（Matrix Factorization，MF）也是协同过滤中最具代表性的算法之一。相比于直接计算用户之间或物品之间的相似度，矩阵分解更加关注学习用户和物品的低维潜在表示（Latent Representation）。从某种意义上说，矩阵分解可以看作现代 Embedding 召回方法的起点，也是推荐系统从传统协同过滤逐步迈向表示学习的重要一步。为了方便大家理解后续的 Embedding 召回等模型，我们仍然从协同过滤的角度来介绍矩阵分解算法。

事实上，推荐系统中的矩阵分解算法最早来源于线性代数中的奇异值分解（Singular Value Decomposition，SVD）。因此，在介绍推荐系统中的矩阵分解算法之前，我们首先介绍标准的 SVD 方法。

10.1.5.1 奇异值分解（SVD）

SVD 是线性代数中最经典的矩阵分解方法，其核心思想是将一个高维矩阵分解为多个低维矩阵的乘积，从而获得矩阵的低秩近似表示。这里我们对 SVD 这种经典的矩阵分解方法进行简要的介绍说明。

假设存在用户-物品交互矩阵 $\mathbf{R} \in \mathbb{R}^{|U|\times|I|}$，则 SVD 可以将其分解为

$$\mathbf{R} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{10.41}$$

其中，$\mathbf{U} \in \mathbb{R}^{|U|\times|U|}$ 表示左奇异向量矩阵；$\mathbf{V} \in \mathbb{R}^{|I|\times|I|}$ 表示右奇异向量矩阵；$\mathbf{\Sigma}$ 表示奇异值对角矩阵，其对角线元素满足 $\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$。由于较大的奇异值通常包含了矩阵中的主要信息，因此实际应用中通常只保留最大的前 d 个奇异值，从而得到矩阵的低秩近似表示：

$$\mathbf{R} \approx \mathbf{U}_d\mathbf{\Sigma}_d\mathbf{V}_d^T \tag{10.42}$$

其中 d 远小于用户数量和物品数量，即

$$d \ll |U|, \qquad d \ll |I| \tag{10.43}$$

从表示学习的角度来看，SVD 实际上已经完成了对用户和物品的低维表示学习。其中，矩阵 $\mathbf{U}_d\mathbf{\Sigma}_d^{1/2}$ 可以看作用户的低维表示，而矩阵 $\mathbf{V}_d\mathbf{\Sigma}_d^{1/2}$ 则可以看作物品的低维表示。因此，SVD 也被看做是最早利用低维向量来表示用户和物品的一种方法。

从损失函数的角度去理解，SVD 本质上求解的是矩阵的最佳低秩逼近问题，即

$$\min_{\text{rank}(\mathbf{X})\leq d} \|\mathbf{R} - \mathbf{X}\|_F^2 \tag{10.44}$$

其中 $\|\cdot\|_F$ 表示 Frobenius 范数。根据 Eckart-Young 定理，截断后的 SVD 正是上述优化问题的解析最优解，因此标准 SVD 不需要像神经网络一样采用梯度下降进行训练，而是可以直接通过矩阵分解得到最优结果。

10.1.5.2 矩阵分解（Matrix Factorization）

标准 SVD 算法虽然能够学习到用户和物品的低维表示，但其直接应用于推荐系统仍然存在两个重要问题。首先，推荐系统中的用户-物品交互矩阵通常具有极高的稀疏性。对于绝大多数互联网推荐平台而言，用户通常只会与极少数物品发生交互，因此交互矩阵中的绝大多数元素实际上都是未知值（Unknown），而不是数值 0。例如，一个拥有 1 亿用户和 500 万物品的推荐系统，其理论交互矩阵规模可达到 $10^8 \times 5 \times 10^6$，但其中真正发生交互的元素仅占极小的一部分。标准 SVD 要求输入矩阵中的所有元素均为已知值，因此无法直接处理如此大规模的稀疏矩阵。其次，即使采用某种方式对未知元素进行补全，标准 SVD 仍然需要对整个矩阵进行奇异值分解，其计算复杂度和存储复杂度都十分高昂。当用户规模和物品规模达到千万甚至上亿级别时，标准 SVD 几乎无法满足工业级推荐系统对于计算效率和模型更新效率的要求。

为了解决上述问题，研究者提出了矩阵分解（Matrix Factorization，MF）算法。与标准 SVD 算法不同的地方在于，矩阵分解不再要求对整个用户-物品交互矩阵进行完整的分解，而只需要使用推荐系统中已有的用户与

物品交互的行为数据来学习用户和物品的低维表示就可以。因此，矩阵分解可以看作是针对推荐系统的稀疏数据这一特点而设计的矩阵近似方法，该方法也是现代基于 Embedding 召回模型的重要基础。

矩阵分解算法的核心思想如下：

定义 10.5 (矩阵分解)

矩阵分解算法通过学习用户 Embedding 和物品 Embedding，将用户和物品共同映射到同一个低维潜在空间中。在该潜在空间内，用户与物品之间的匹配程度可以通过向量之间的相似度进行刻画，从而完成推荐。

♣

下面，我们给出矩阵分解算法具体的数学形式及符号约定。假设用户集合为 U，物品集合为 I，用户-物品交互矩阵记为 $\mathbf{R} \in \mathbb{R}^{|U|\times|I|}$，其中 R_{ui} 表示用户 u 与物品 i 之间的交互行为，可以是评分、点击、购买、收藏、观看等各种反馈信号。矩阵分解算法的目标是学习两个低维矩阵 $\mathbf{P} \in \mathbb{R}^{|U|\times d}$ 和 $\mathbf{Q} \in \mathbb{R}^{|I|\times d}$，其中 d 表示 Embedding 维度，并满足 $d \ll |U|, \quad d \ll |I|$。于是，用户-物品交互矩阵可以近似表示为

$$\mathbf{R} \approx \mathbf{P}\mathbf{Q}^T \tag{10.45}$$

其中，

$$\mathbf{p}_u = (p_{u1}, p_{u2}, \cdots, p_{ud}) \tag{10.46}$$

表示用户 u 对应的 Embedding 向量，

$$\mathbf{q}_i = (q_{i1}, q_{i2}, \cdots, q_{id}) \tag{10.47}$$

表示物品 i 对应的 Embedding 向量。而对于任意用户和物品，矩阵分解算法预测出的用户与物品之间的交互值可以表示为

$$\hat{R}_{ui} = \mathbf{p}_u^T \mathbf{q}_i \tag{10.48}$$

可以看到，相较于标准 SVD，矩阵分解不再显式求解奇异值矩阵，而是直接学习用户 Embedding 和物品 Embedding，并利用两者的内积作为用户与物品之间的兴趣匹配程度。事实上，如果将标准 SVD 中的奇异值矩阵均匀分配到左右两侧，即

$$\mathbf{P} = \mathbf{U}_d \mathbf{\Sigma}_d^{1/2}, \qquad \mathbf{Q} = \mathbf{V}_d \mathbf{\Sigma}_d^{1/2} \tag{10.49}$$

则会有

$$\mathbf{R} \approx \mathbf{P}\mathbf{Q}^T \tag{10.50}$$

因此，从数学形式上来看，矩阵分解可以看作是标准 SVD 在推荐系统中的一种推广形式，只不过它不再依赖完整矩阵，而是直接学习用户和物品的低维表示。

在具体工业界推荐系统中落地矩阵分解算法时，通常只基于用户与物品之间实际的交互数据来进行模型的训练。为了学习用户 Embedding 和物品 Embedding，矩阵分解算法通常会最小化预测值与真实交互之间的误差。矩阵分解算法的一种经典优化目标如下所示：

$$\min_{\mathbf{P},\mathbf{Q}} \sum_{(u,i)\in\Omega} \left(R_{ui} - \mathbf{p}_u^T \mathbf{q}_i\right)^2 + \lambda \left(\|\mathbf{P}\|_F^2 + \|\mathbf{Q}\|_F^2\right) \tag{10.51}$$

其中，Ω 表示所有存在交互行为的用户-物品样本集合，λ 表示正则化系数，$\|\cdot\|_F$ 表示 Frobenius 范数，用于防止模型过拟合。

在模型训练完成之后，对于任意用户 u，只需计算用户 Embedding 与所有候选物品 Embedding 之间的内积得分

$$score(u,i) = \mathbf{p}_u^T \mathbf{q}_i \tag{10.52}$$

随后按照得分排序即可得到 Top-K 召回结果。

矩阵分解算法在推荐系统中的广泛应用，最早可以追溯到 Simon Funk 在 Netflix Prize 竞赛期间提出的 Funk SVD 算法。虽然其名称中包含“SVD”，但 Funk SVD 实际上并没有采用传统线性代数中的奇异值分解（Singular

Value Decomposition）来求解用户-物品交互矩阵，而是针对推荐系统交互矩阵高度稀疏的特点，直接学习用户 Embedding 和物品 Embedding，并采用随机梯度下降（Stochastic Gradient Descent，SGD）对矩阵分解目标函数进行优化。因此，从严格意义上来说，Funk SVD 并不是传统意义上的 SVD 算法，而是一种基于矩阵分解思想的推荐模型，也是矩阵分解算法在推荐系统中成功落地应用的重要里程碑。

除了 Funk SVD 之外，研究者也提出了一些基于 SVD 的改进算法，比如 BiasSVD [28]、SVD++ [27]、Probabilistic Matrix Factorization（PMF）[37] 等。这些方法均继承了矩阵分解的基本框架，只是在偏置建模、隐式反馈建模以及概率建模等方面进行了进一步扩展，并在很长一段时间内成为工业界推荐系统的重要基础模型。随着深度学习的发展，Neural CF、DSSM、YouTube DNN 以及双塔模型等基于深度学习的 Embedding 召回模型逐渐成为主流方案。然而，这些模型并非脱离矩阵分解算法另起炉灶，而是在其基础之上进一步增强了特征表达的能力以及非线性建模的能力。因此，这些算法也可以看作是矩阵分解在深度学习时代的自然演进。

相比于 ItemCF 和 UserCF 召回，矩阵分解召回最大的优势在于能够学习更加抽象的潜在兴趣表示，而不仅仅依赖于用户和物品之间的显式共现关系。例如，两部电影虽然很少被同一批用户共同观看，但如果它们分别受到兴趣相近用户群体的喜爱，那么矩阵分解仍然能够学习到它们具有相近的 Embedding 表示，从而将其映射到潜在空间中相邻的位置，实现更好的推荐效果。这表明，矩阵分解能够挖掘用户与物品之间更加深层次的潜在关联，因此相比传统协同过滤算法具有更强的泛化能力。

更重要的是，矩阵分解首次将用户和物品统一映射到了同一个低维向量空间中，奠定了现代 Embedding 召回模型的基本框架。后续介绍的 Neural CF、DSSM、YouTube DNN 以及双塔模型等方法，都延续了“用户 Embedding + 物品 Embedding + 相似度匹配”的核心思想，只是在 Embedding 生成方式、特征建模能力以及匹配函数等方面进行了进一步扩展。例如，传统矩阵分解通常采用向量内积作为匹配函数，而现代双塔模型则可以利用深度神经网络学习更加丰富的用户表示和物品表示，并结合 MLP、注意力机制等非线性结构构建更加复杂的匹配函数，从而进一步提升召回效果。因此，从技术发展的脉络来看，现代 Embedding 召回模型本质上是在矩阵分解框架上的持续演进与发展。

10.1.6 Neural CF 召回

随着矩阵分解（Matrix Factorization，MF）算法的发展，推荐系统已经能够通过学习用户 Embedding 和物品 Embedding 来刻画用户与物品之间的潜在兴趣关系。然而，传统矩阵分解通常采用向量内积作为用户与物品之间的匹配函数，即 $\hat{y}_{ui} = \mathbf{p}_u^T \mathbf{q}_i$。这种建模方式虽然计算简单、易于优化，并且能够很好地支持大规模召回，但本质上仍然属于一种线性模型。当用户兴趣模式较为复杂时，仅依靠向量内积往往难以刻画用户与物品之间复杂的非线性交互关系，其模型表达能力存在一定的局限性。

读到这里，大家就会有一个疑问：

笔记 矩阵分解模型是一种线性模型，表达能力有限。那么是否有方法可以进一步提升其表达能力呢？

答案是肯定的。事实上，近年来学术界和工业界都在积极探索如何增强协同过滤模型的表达能力，以更好地刻画用户与物品之间复杂的兴趣关系。为了进一步提升协同过滤模型的表示能力，2017 年 WWW 会议论文《Neural Collaborative Filtering》[21] 提出了 Neural Collaborative Filtering（Neural CF）模型。该工作首次利用深度神经网络替代传统矩阵分解中的内积匹配函数，在保留 Embedding 表示学习框架的基础上，引入神经网络学习用户与物品之间更加复杂的非线性交互关系，因此也被认为是深度学习协同过滤（Deep Collaborative Filtering）的代表性工作之一。

Neural CF 召回的核心思想如下：

定义 10.6 (Neural Collaborative Filtering)

Neural Collaborative Filtering（Neural CF）召回利用深度神经网络学习用户 Embedding 与物品 Embedding 之间的非线性匹配函数，从而替代传统矩阵分解中的向量内积，进一步提升协同过滤模型对复杂用户兴趣模式的建模能力。

从整体框架来看，Neural CF 仍然遵循现代 Embedding 召回模型的基本范式，即首先学习用户 Embedding 和

物品 Embedding，然后根据两者之间的匹配程度预测用户对物品的兴趣。如图10.5所示，整个模型主要由用户 Embedding 层（User Embedding）、物品 Embedding 层（Item Embedding）以及多层感知机（Multi-Layer Perceptron，MLP）三部分组成。其中，用户 Embedding 层负责学习用户的低维表示，物品 Embedding 层负责学习物品的低维表示。随后，将两者输入神经网络进行非线性特征交互，并最终输出用户对物品的兴趣预测值。

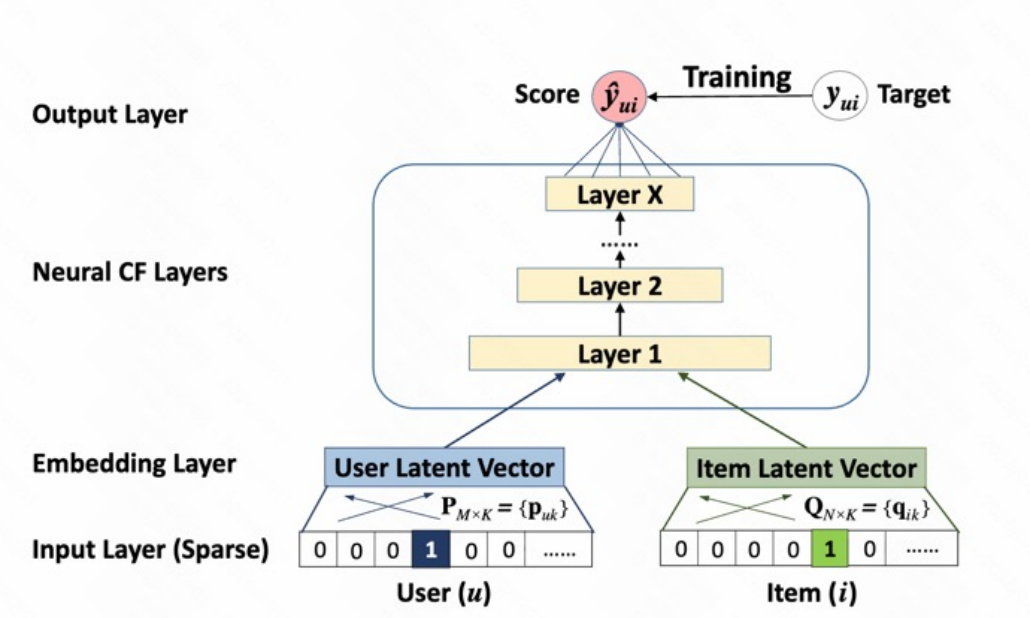

图 10.5: Neural CF 模型结构示意图。

与上一节介绍的矩阵分解相比，两种方法都采用了用户 Embedding 和物品 Embedding 作为用户与物品的潜在表示，因此整体框架具有高度的一致性。两者最大的区别在于，矩阵分解采用固定的向量内积完成匹配，而 Neural CF 则利用深度神经网络自动学习更加复杂的匹配函数，从而增强模型的表达能力。

假设用户 Embedding 矩阵和物品 Embedding 矩阵分别表示为 $\mathbf{P} \in \mathbb{R}^{|U|\times d}$ 和 $\mathbf{Q} \in \mathbb{R}^{|I|\times d}$，其中 d 表示 Embedding 维度。对于任意用户 u 和物品 i，其对应的 Embedding 表示分别为

$$\mathbf{p}_u = \mathbf{P}^T \mathbf{v}_u^U, \qquad \mathbf{q}_i = \mathbf{Q}^T \mathbf{v}_i^I \tag{10.53}$$

其中，$\mathbf{v}_u^U$ 表示用户 u 对应的 One-Hot 编码向量，$\mathbf{v}_i^I$ 表示物品 i 对应的 One-Hot 编码向量。通过 Embedding Lookup 操作，可以将原始高维稀疏的 One-Hot 表示映射为低维稠密向量，从而学习用户和物品的潜在语义表示。

在矩阵分解中，用户兴趣预测通常采用向量内积完成，即 $\hat{y}_{ui} = \mathbf{p}_u^T \mathbf{q}_i$，而在 Neural CF 中，这一固定的线性匹配函数被推广为一个可学习的非线性函数 $\hat{y}_{ui} = f(\mathbf{p}_u, \mathbf{q}_i; \Theta)$，其中，$f(\cdot)$ 表示由神经网络学习得到的非线性匹配函数，Θ 表示神经网络中的所有参数。具体来说，Neural CF 首先将用户 Embedding 和物品 Embedding 进行拼接（Concatenation）得到 $\mathbf{x}_0 = [\mathbf{p}_u; \mathbf{q}_i]$，随后依次输入多层感知机进行非线性变换。假设整个 MLP 共有 L 层，则第 l 层隐藏层表示可以写为

$$\mathbf{x}_l = \phi_l(\mathbf{W}_l \mathbf{x}_{l-1} + \mathbf{b}_l) \qquad l = 1, \cdots, L, \tag{10.54}$$

其中，$\mathbf{W}_l$ 和 $\mathbf{b}_l$ 分别表示第 l 层网络的权重矩阵和偏置向量，$\phi_l(\cdot)$ 表示激活函数，例如 ReLU、Sigmoid 或 Tanh 等。最终，输出层根据最后一层隐藏向量预测用户对物品的兴趣概率，即

$$\hat{y}_{ui} = \sigma(\mathbf{h}^T \mathbf{x}_L) \tag{10.55}$$

其中，$\mathbf{h}$ 表示输出层参数，$\sigma(\cdot)$ 表示 Sigmoid 激活函数。当输出为点击概率、购买概率等二分类任务时，Sigmoid 能够将模型输出映射到 $(0, 1)$ 区间，从而表示用户与物品发生交互的概率。

可以看到，相比于矩阵分解仅采用一次向量内积计算用户与物品之间的相关性，Neural CF 利用多层神经网络对用户 Embedding 和物品 Embedding 进行逐层非线性变换，不仅能够学习更加丰富的高阶特征表示，还能够建模用户与物品之间复杂的非线性交互关系，从而进一步提升推荐模型的表达能力。

在 Neural CF 论文中，作者并没有直接提出最终的 NeuMF 模型，而是按照模型复杂度逐步构建了三种不同的协同过滤模型，分别为 Generalized Matrix Factorization（GMF）、MLP 以及 NeuMF。其中，GMF 可以看作传统矩阵分解的非线性推广，MLP 利用深度神经网络学习更加复杂的交互关系，而 NeuMF 则融合了 GMF 和 MLP 两种结构，成为论文最终采用的完整模型。下面分别介绍这三种模型。

10.1.6.1 GMF 与 MLP 模型

传统矩阵分解采用向量内积作为用户与物品之间的匹配函数，即 $\hat{y}_{ui} = \mathbf{p}_u^T \mathbf{q}_i$。虽然向量内积计算简单，但它隐含地假设 Embedding 各维度之间具有相同的重要性，模型表达能力受到一定限制。为此，论文首先提出了 Generalized Matrix Factorization（GMF）模型。

GMF 保留了矩阵分解中学习用户 Embedding 和物品 Embedding 的思想，但不再直接计算两者的内积，而是首先计算 Embedding 之间的逐元素乘积（Hadamard Product），即 $\mathbf{z}_{ui} = \mathbf{p}_u \odot \mathbf{q}_i$，其中，$\odot$ 表示逐元素乘法。随后，将得到的交互向量输入输出层进行预测，即

$$\hat{y}_{ui}^{GMF} = \sigma\left(\mathbf{h}_{GMF}^T \left(\mathbf{p}_u \odot \mathbf{q}_i\right)\right) \tag{10.56}$$

其中，$\mathbf{h}_{GMF}$ 表示输出层参数。

相比传统矩阵分解，GMF 最大的区别在于不再简单地将所有 Embedding 维度直接求和，而是允许模型自动学习不同 Embedding 维度对于最终预测的重要程度，因此具有更强的表达能力。从这一角度来看，GMF 可以视为传统矩阵分解的一种推广形式。虽然 GMF 相比矩阵分解具有更高的灵活性，但本质上仍然采用 Embedding 逐元素乘积作为用户与物品之间的交互方式，其表达能力依然有限。

因此，论文进一步提出了基于多层感知机（Multi-Layer Perceptron，MLP）的协同过滤模型。与 GMF 不同，MLP 首先将用户 Embedding 和物品 Embedding 进行拼接得到 $\mathbf{z}_0 = [\mathbf{p}_u; \mathbf{q}_i]$，随后将 $\mathbf{z}_0$ 输入到多层神经网络中进行非线性变换，即：

$$\mathbf{z}_l = \phi_l(\mathbf{W}_l \mathbf{z}_{l-1} + \mathbf{b}_l), \qquad l = 1, \cdots, L \tag{10.57}$$

其中，$\mathbf{W}_l$ 和 $\mathbf{b}_l$ 分别表示第 l 层网络参数，$\phi_l(\cdot)$ 表示激活函数。最终，模型输出可以表示为

$$\hat{y}_{ui}^{MLP} = \sigma(\mathbf{h}_{MLP}^T \mathbf{z}_L) \tag{10.58}$$

其中，$\mathbf{h}_{MLP}$ 表示输出层参数。

相比起 GMF 模型，MLP 模型最大的优势在于利用多层神经网络不断学习用户 Embedding 与物品 Embedding 之间更加复杂的高阶交互关系，因此 MLP 模型能够突破传统矩阵分解线性匹配函数的限制。不过，完全依赖 MLP 也存在一定问题。传统矩阵分解中的向量内积能够较好地建模用户与物品之间的线性关系，而 MLP 虽然具有较强的非线性表达能力，却可能削弱这种有效的线性匹配模式。因此，只采用 GMF 或者只采用 MLP 都会存在一定局限性。

10.1.6.2 NeuMF 模型

为了能够同时兼顾矩阵分解良好的线性建模能力和深度神经网络优秀的非线性建模能力，论文最终提出了 Neural Matrix Factorization（NeuMF）模型。NeuMF 整体结构如图10.6所示。

与 GMF 和 MLP 最大的区别在于，NeuMF 包含两个相互独立的网络分支。其中，左侧 GMF 分支负责学习用户与物品之间较为简单的线性交互关系；右侧 MLP 分支则利用深度神经网络建模更加复杂的非线性交互关系。值得注意的是，这两个分支分别学习各自独立的用户 Embedding 和物品 Embedding，而不是共享同一组 Embedding 参数，从而使两个分支能够分别学习更加适合自身网络结构的特征表示。GMF 分支输出表示为

$$\mathbf{z}_{GMF} = \mathbf{p}_u^G \odot \mathbf{q}_i^G, \tag{10.59}$$

其中，$\mathbf{p}_u^G$ 和 $\mathbf{q}_i^G$ 分别表示 GMF 分支对应的用户 Embedding 和物品 Embedding。MLP 分支则可以表示为

$$\mathbf{z}_{MLP} = f_{MLP}(\mathbf{p}_u^M, \mathbf{q}_i^M) \tag{10.60}$$

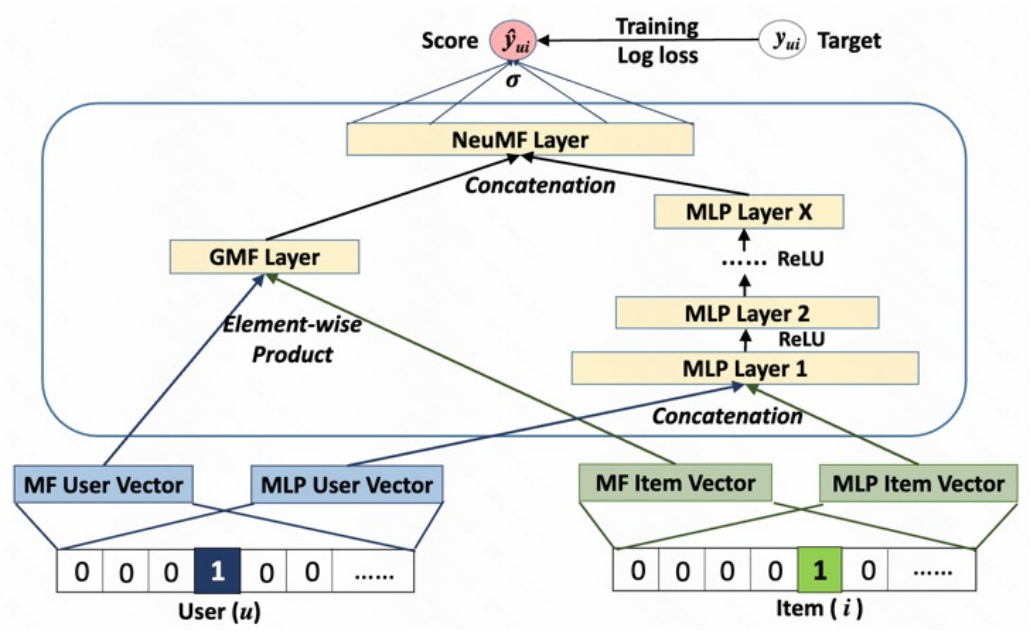

图 10.6: NeuMF 模型结构示意图。

其中，$\mathbf{p}_u^M$ 和 $\mathbf{q}_i^M$ 表示 MLP 分支对应的 Embedding，$f_{MLP}(\cdot)$ 表示整个多层感知机网络。

最终，NeuMF 将两个分支最后得到的隐藏表示进行拼接 $\mathbf{z} = [\mathbf{z}_{GMF}; \mathbf{z}_{MLP}]$，并利用输出层完成最终预测

$$\hat{y}_{ui} = \sigma(\mathbf{h}^T \mathbf{z}) \tag{10.61}$$

这里 $\mathbf{h}^T$ 表示输出层中的网络参数，具体定义如下：

$$\mathbf{h}^T = [\alpha \mathbf{h}_{GMF}^T; (1-\alpha)\mathbf{h}_{MLP}^T] \tag{10.62}$$

其中，α 表示 GMF 分支和 MLP 分支的融合权重，$\mathbf{h}_{GMF}^T$ 和 $\mathbf{h}_{MLP}^T$ 分别表示 GMF 分支和 MLP 分支的输出向量。通过这种融合方式，NeuMF 既能够保留矩阵分解在线性协同过滤方面的优势，又能够充分利用深度神经网络强大的非线性建模能力，因此在论文实验中取得了优于 GMF 和 MLP 单独使用时的推荐效果。

10.1.6.3 Neural CF 模型训练

Neural CF 模型的训练方式与矩阵分解类似，都是通过最小化预测值与真实标签之间的误差来学习用户 Embedding、物品 Embedding 以及神经网络参数。不同的是，矩阵分解通常直接优化 Embedding 之间的内积，而 Neural CF 还需要同时学习神经网络中的非线性匹配函数，因此模型参数更加丰富，表达能力也更强。

Neural CF 论文主要针对隐式反馈（Implicit Feedback）推荐场景进行研究，因此采用二分类交叉熵（Binary Cross Entropy，BCE）作为模型的优化目标。对于任意用户-物品样本 (u, i)，设模型预测用户与物品发生交互的概率为 $\hat{y}_{ui}$，真实标签为 $y_{ui} \in \{0, 1\}$，则损失函数可以表示为

$$\mathcal{L} = -\sum_{(u,i)\in\Omega} [y_{ui} \log(\hat{y}_{ui}) + (1 - y_{ui}) \log(1 - \hat{y}_{ui})] \tag{10.63}$$

其中，Ω 表示训练样本集合。当推荐任务采用显式评分（Explicit Feedback）时，也可以采用均方误差（Mean Squared Error，MSE）作为优化目标，即

$$\mathcal{L} = \sum_{(u,i)\in\Omega} (y_{ui} - \hat{y}_{ui})^2 \tag{10.64}$$

不过，在工业界推荐系统中，大多数召回模型通常采用点击、观看、购买等隐式反馈进行训练，因此交叉熵损失函数更加常见。

除了模型结构之外，Neural CF 论文还提出了一种有效的训练策略，即预训练（Pre-training）。由于 NeuMF

同时包含 GMF 分支和 MLP 分支，如果直接随机初始化整个网络进行联合训练，模型容易陷入局部最优，训练过程也较为困难。因此，论文首先分别训练 GMF 模型和 MLP 模型，当两个分支分别收敛之后，再利用两者训练得到的参数初始化 NeuMF 模型，并进一步进行联合微调（Fine-tuning）。实验结果表明，这种预训练策略能够有效提高模型收敛速度，并进一步提升最终推荐性能。

Neural CF 最大的贡献在于突破了传统矩阵分解固定内积匹配函数的限制。它首次将深度神经网络引入协同过滤框架，以可学习的非线性匹配函数替代传统矩阵分解中的向量内积，使模型能够学习用户与物品之间更加复杂的高阶交互关系，从而进一步提升了协同过滤模型的表达能力。尽管由于在线计算效率等因素，Neural CF 并未成为工业界召回系统的主流方案，但其提出的“Embedding 表示学习 + 神经网络匹配”思想对后续深度学习推荐模型的发展产生了深远影响。从技术演进的角度来看，Neural CF 既是传统矩阵分解向深度学习协同过滤过渡的重要桥梁，也是现代 Embedding 召回模型发展的重要里程碑，为后续更加复杂的深度学习召回算法奠定了坚实的基础。

10.1.7 Heterogeneous CF 召回

随着 Embedding 召回模型的发展，越来越多的推荐系统开始利用点击、点赞、收藏、购买等多种用户行为进行联合建模。然而，大多数传统协同过滤模型通常仅针对单一行为进行训练，并通过负采样（Negative Sampling）构造正负样本完成模型优化。这种训练方式虽然简单有效，但也存在训练效率较低、负样本质量依赖较强以及难以充分利用多行为监督信息等问题。

针对上述问题，2020 年 AAAI 会议论文《Efficient Heterogeneous Collaborative Filtering without Negative Sampling for Recommendation》[6] 提出了 Heterogeneous Collaborative Filtering（Heterogeneous CF，简称 EHCF）模型。EHCF 的核心思想是将点击、点赞、收藏、购买等不同类型的用户行为统一建模，并利用行为之间的关联关系进行知识迁移（Knowledge Transfer），从而在无需负采样的情况下学习更加准确的用户和物品 Embedding。

与传统协同过滤模型不同，EHCF 中的”Heterogeneous” 并不是指异构图神经网络中的异构图结构，而是指不同类型的用户反馈行为（Heterogeneous Feedback）。例如，一个用户可能依次发生” 曝光 → 点击 → 点赞 → 收藏 → 购买” 等多个行为，这些行为之间通常具有较强的相关性。如果仅利用最终购买行为训练模型，大量其他行为信息将无法得到充分利用；而 EHCF 则通过联合建模不同类型的行为标签，并学习不同标签之间的迁移关系，从而进一步提升模型的表示能力。

另一方面，在工业级推荐系统中，大多数 Embedding 召回模型通常需要构造负样本进行训练。一方面，负采样能够帮助模型学习用户不感兴趣的物品，提高模型的判别能力；另一方面，通过引入 Hard Negative 样本，还可以缓解曝光偏差（Exposure Bias）带来的影响。然而，负采样也存在一定局限性。例如，不同负样本采样策略会直接影响模型训练效果，同时负采样还会带来额外的数据构建和训练开销。因此，如何在不依赖负采样的情况下完成 Embedding 学习，一直是推荐系统中的重要研究方向。EHCF 正是在这一背景下提出的一种无需负采样的协同过滤模型。

与 Neural CF 类似，EHCF 仍然采用 Embedding 表示学习的框架，将用户和物品分别映射到统一的低维向量空间中。但与 Neural CF 利用 MLP 学习复杂非线性交互不同，EHCF 仍然采用简单的线性匹配函数，而将模型能力主要集中于多行为联合建模和跨行为知识迁移，从而兼顾模型效果与训练效率。

EHCF 整体结构如图10.7所示，主要包括多行为协同过滤框架（Multi-behavior CF）和 Transfer-based Prediction Layer 两个部分。在图10.7左侧的多行为协同过滤框架中，用户 Embedding 和物品 Embedding 首先通过 Hadamard 积进行特征交互，得到用户-物品联合表示（Joint Representation）。随后，针对不同类型的行为标签（如点击、点赞、收藏、购买等），模型利用 Transfer-based Prediction Layer 分别计算对应行为下的兴趣预测得分。图10.7右侧展示了 Transfer-based Prediction Layer 的具体结构。该模块通过行为标签之间的迁移矩阵（Transfer Matrix）在不同任务对应的隐空间中进行线性变换，实现各行为预测层参数之间的知识迁移（Knowledge Transfer）。因此，高频行为（如点击）所学习到的知识可以迁移到低频行为（如购买），从而实现跨行为的信息共享，提升稀疏行为标签的建模能力，并进一步增强模型整体的推荐效果。

假设 $\mathbf{p}_u \in \mathbb{R}^d$ 和 $\mathbf{q}_v \in \mathbb{R}^d$ 分别表示用户 u 和物品 v 对应的 Embedding，则首先利用 Hadamard 积完成用户和

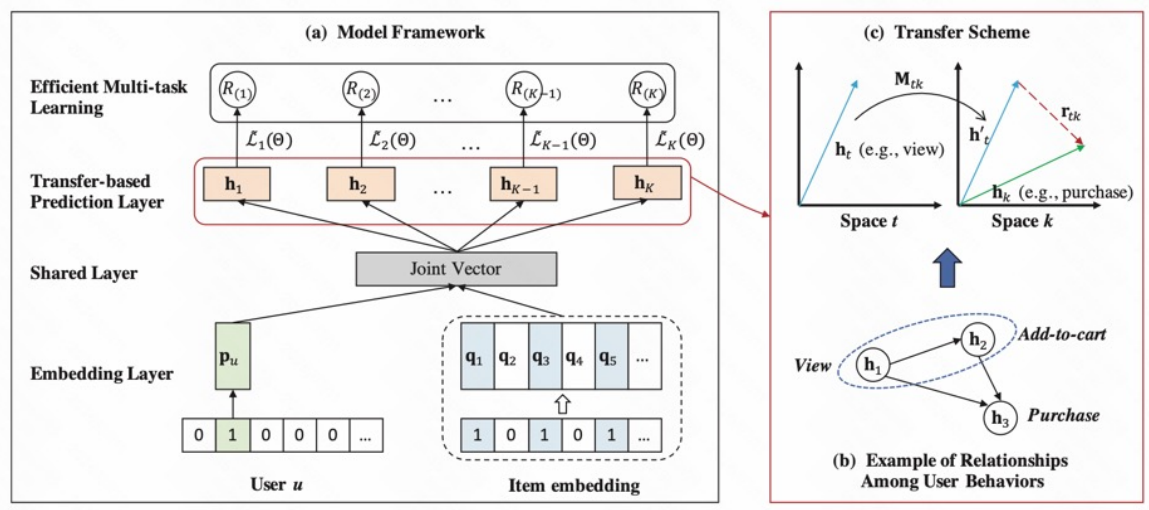

图 10.7: Heterogeneous CF 模型结构示意图。

物品之间的特征交互

$$\mathbf{z}_{uv} = \mathbf{p}_u \odot \mathbf{q}_v \tag{10.65}$$

随后，对于第 k 种行为标签，采用一个线性预测层计算对应的兴趣得分

$$\hat{R}_{uv}^{(k)} = \mathbf{h}_k^T \mathbf{z}_{uv} = \mathbf{h}_k^T (\mathbf{p}_u \odot \mathbf{q}_v) \tag{10.66}$$

其中，$\mathbf{h}_k$ 表示第 k 个行为标签对应的预测参数。

EHCF 最大的特点在于提出了 Transfer-based Prediction Layer，用于实现不同反馈行为之间的知识迁移。由于点击、点赞、收藏、购买等行为之间通常存在较强的关联性，因此模型并不是分别独立学习每一种行为，而是利用行为之间的迁移关系共享模型参数，从而提升模型对于稀疏行为标签的学习能力。具体来说，对于标签 t 到标签 k 之间的知识迁移，其转移函数定义为

$$f_{\mathbf{h}_t \to \mathbf{h}_k} = \mathbf{h}_t \mathbf{M}_{tk} + \mathbf{r}_{tk} \tag{10.67}$$

其中，$\mathbf{M}_{tk}$ 表示行为 t 到行为 k 之间的迁移矩阵，$\mathbf{r}_{tk}$ 表示迁移偏置。因此，第 k 个标签对应的预测参数可以进一步表示为

$$\mathbf{h}_k = \sum_{t \in \mathcal{N}(k)} (\mathbf{h}_t \mathbf{M}_{tk} + \mathbf{r}_{tk}) \tag{10.68}$$

其中，$\mathcal{N}(k)$ 表示能够向标签 k 传递知识的相关行为集合。通过这种方式，不同行为之间可以共享知识，从而缓解部分行为样本不足带来的训练困难。

EHCF 模型最大的特点之一在于其训练过程无需依赖负采样（Negative Sampling）。与 Neural CF 等模型通常采用二分类交叉熵损失不同，EHCF 借鉴了隐式反馈矩阵分解（Weighted Matrix Factorization，WMF）[23] 的思想，对整个用户-物品交互矩阵进行加权回归优化。

对于第 k 种行为（例如点击、点赞或购买），设对应的行为矩阵为 $\mathbf{R}^{(k)}$，模型预测值为 $\hat{\mathbf{R}}^{(k)}$。对于一个 Mini-batch 中的用户集合 B 以及全部物品集合 V，EHCF 首先定义如下加权平方损失函数：

$$L_k(\Theta) = \sum_{u \in B} \sum_{v \in V} c_{uv}^{(k)} \left(R_{uv}^{(k)} - \hat{R}_{uv}^{(k)} \right)^2, \tag{10.69}$$

其中，Θ 表示模型参数，$c_{uv}^{(k)}$ 表示对应样本的置信权重（Confidence Weight）。

通常对于观测到的正样本赋予较大的权重，而未观测到的样本则赋予较小的权重。如果直接按照上述公式进行优化，每一次参数更新都需要遍历整个物品集合，其时间复杂度为 $O(|B||V|d)$，其中 $|B|$ 表示 Mini-batch 中

的用户数，$|V|$ 表示物品数量，d 表示 Embedding 维度。在工业级推荐系统中，由于物品规模通常达到百万甚至千万级，这种计算方式显然难以直接应用。

为了避免遍历所有未曝光物品，EHCF 充分利用了隐式反馈数据的特点。由于绝大多数未观测交互均满足 $R_{uv}^{(k)} = 0$，因此可以将整个损失函数重新拆解为正样本损失和全样本损失两部分，即

$$\tilde{L}_k(\Theta) = L_k^P(\Theta) + L_k^A(\Theta) \tag{10.70}$$

其中，

$$L_k^P(\Theta) = \sum_{(u,v)\in\Omega_k} \left(c_{uv}^+ - c_{uv}^-\right) \hat{R}_{uv}^{(k)2} - 2c_{uv}^+ R_{uv}^{(k)} \hat{R}_{uv}^{(k)} \tag{10.71}$$

表示正样本对应的损失，而

$$L_k^A(\Theta) = \sum_{u\in B} \sum_{v\in V} c_{uv}^- \hat{R}_{uv}^{(k)2} \tag{10.72}$$

表示所有样本的统一损失项。

进一步地，由于 EHCF 采用 Hadamard 积作为预测函数，即：

$$\hat{R}_{uv}^{(k)} = \mathbf{h}_k^T(\mathbf{p}_u \odot \mathbf{q}_v) \tag{10.73}$$

因此平方项可以展开为

$$\hat{R}_{uv}^{(k)2} = \sum_{i=1}^{d}\sum_{j=1}^{d} h_{k,i}h_{k,j}(p_{u,i}p_{u,j})(q_{v,i}q_{v,j}) \tag{10.74}$$

可以看到，用户 Embedding 与物品 Embedding 已经能够分别进行重组，因此原本关于用户和物品的双重求和可以重新组织为两个彼此独立的求和过程，即分别统计用户侧和物品侧的二阶统计量，再进行矩阵乘积计算。这样便避免了对所有用户-物品组合逐一计算预测值，从而大幅降低了模型训练复杂度，实现了 Sampling-free Optimization。

由于 EHCF 同时建模了多种用户行为，因此每一种行为都会对应一个独立的损失函数 $L_k(\Theta)$。最终，整个模型采用多任务学习（Multi-task Learning）的方式进行联合优化，其总体目标函数可以表示为

$$\mathcal{L} = \sum_{k=1}^{K} \lambda_k L_k(\Theta) \tag{10.75}$$

其中 K 表示行为类型的数量，例如点击、点赞、收藏、购买等；λ_k 表示第 k 个行为的权重。通过联合优化多个行为目标，并结合 Transfer-based Prediction Layer 实现跨行为知识迁移，EHCF 能够充分利用不同反馈行为之间的关联信息，在无需负采样的情况下学习更加准确的用户 Embedding 和物品 Embedding。

总体来看，EHCF 仍然属于 Embedding 召回模型，其最大的创新点并不在于设计更加复杂的神经网络结构，而是通过联合建模多种用户行为，并利用 Transfer-based Prediction Layer 实现跨行为知识迁移，从而有效缓解了传统协同过滤对于负采样的依赖，并进一步提升了 Embedding 表示学习的效果。这一思想对于后续多任务学习（Multi-task Learning）和多行为推荐模型的发展也具有重要的启发意义。

10.2 图召回

在本章中，前面介绍的协同过滤召回主要利用用户与物品之间的历史交互关系进行推荐，例如 ItemCF 召回、UserCF 召回以及 Swing 召回等方法，本质上都是基于用户行为统计物品之间的关联关系。然而，这类方法通常主要利用用户与物品之间的局部交互信息，对于图结构中的高阶邻居关系以及更加复杂的关联模式建模能力有限。例如，两个物品之间虽然没有直接共现，但可能通过多个共同用户、多跳路径或社交关系形成潜在关联，这些关系仅依靠传统协同过滤方法往往难以充分挖掘。

近年来，**图表示学习（Graph Representation Learning）**逐渐成为推荐系统的重要研究方向。图表示学习的核心思想是将用户、物品以及它们之间的各种关系统一建模为图结构，并通过图学习算法学习每个节点的低维表示，从而充分利用图中的高阶结构信息，提升推荐系统对于复杂兴趣关系的建模能力。相比起传统的协同过

滤方法，图召回不仅能够利用用户与物品之间的直接交互，还能够充分挖掘图中的多跳邻居关系、社区结构以及潜在的语义信息，因此近年来被广泛应用于电商、短视频、直播以及社交推荐等场景。

按照所采用的图学习方法不同，工业界常见的图召回方法大致可以分为两类：一类是**图嵌入召回（Graph Embedding Retrieval）**，另一类是**图神经网络召回（Graph Neural Network Retrieval）**。其中，图嵌入召回首先通过随机游走或图表示学习算法生成节点 Embedding，再利用 Embedding 进行近邻检索；图神经网络召回则利用消息传递（Message Passing）机制直接学习节点表示，从而进一步增强对图结构信息的建模能力。本章将分别介绍这两类图召回方法的基本原理、模型结构以及工业界中的典型应用案例。

10.2.1 图嵌入召回

图嵌入召回（Graph Embedding Retrieval）是图召回中一类在工业界广泛应用的方法，它的核心思想如下：

定义 10.7

图嵌入召回将图中的节点映射到低维连续向量空间，使图中邻近或结构相似的节点在 Embedding 空间中也能具有较高的相似性。在完成 Embedding 训练后，线上就可以利用向量检索技术快速完成 Top-K 近邻召回。

在推荐系统中，图嵌入方法通常首先根据用户历史交互行为构建用户-物品二部图（User-Item Bipartite Graph）、物品图（Item Graph）或作者图（Author Graph），随后利用图表示学习算法学习节点 Embedding，并最终将 Embedding 写入 ANN 向量检索系统用于在线召回。常见的图嵌入方法包括 DeepWalk[41]、Node2Vec[16] 以及 EGES[52] 等。

10.2.1.1 DeepWalk 召回

DeepWalk 算法最早由 Perozzi 等人在 2014 年提出，发表于 KDD 会议的论文《DeepWalk: Online Learning of Social Representations》[41]。它首次将自然语言处理中 Word2Vec [36] 的思想引入图表示学习，通过随机游走（Random Walk）将图结构转化为节点序列，再利用 Skip-Gram 模型 [36] 学习节点 Embedding，因此也被认为是 Graph Embedding 领域最具代表性的工作之一。

在推荐系统中，DeepWalk 算法通常不会直接作用于用户-物品二部图，而是首先根据用户行为构建物品图（Item Graph）或作者图（Author Graph），随后在图上进行随机游走，最终学习每个节点对应的 Embedding 表示，并用于 Embedding 召回。

10.2.1.1.1 图结构构建 DeepWalk 算法的第一步是构建图结构。对于推荐系统而言，图中的节点通常对应物品或作者等推荐对象，而图中的边则来源于用户行为产生的共现关系。这里举个例子，如果大量用户连续浏览了物品 i 和 j，那么就可以在两个节点 i 和 j 之间建立一条边，其边权可以表示为

$$w_{ij} = CoOccur(i, j) \tag{10.76}$$

其中 $CoOccur(i, j)$ 表示两个物品的共现次数，也可以采用 ItemCF、Swing 等方法计算得到的相似度作为边权。

因此，一个推荐图可以表示为 $G = (V, E)$，其中 V 表示所有物品节点，E 表示物品之间的连接关系。相比用户-物品二部图，工业界更倾向于构建物品图或者作者图，因为这两类图的规模更加稳定，同时便于离线更新和在线部署。

10.2.1.1.2 Random Walk 采样 在构建完成图结构之后，DeepWalk 算法并不会直接训练 Embedding，而是首先在图上执行大量随机游走（Random Walk）。对于每个起始节点 v_0，随机游走过程可以表示为

$$v_0 \to v_1 \to v_2 \to \cdots \to v_L \tag{10.77}$$

其中 L 表示游走长度。每一步都会随机选择当前节点的一个邻居继续游走，其转移概率定义为

$$P(v_{t+1} = x | v_t) = \frac{w_{v_t x}}{\sum_{u \in \mathcal{N}(v_t)} w_{v_t u}} \tag{10.78}$$

其中 $\mathcal{N}(v_t)$ 表示当前节点的邻居集合。

经过多次随机游走之后，每个节点都会生成大量类似于自然语言句子的节点序列，例如

$$A \to B \to D \to F \to E \to \cdots \tag{10.79}$$

这些序列能够同时保留图中的局部结构信息和高阶邻居关系。

10.2.1.1.3 Skip-Gram 训练 在得到随机游走序列之后，DeepWalk 召回将其视为自然语言中的句子，再利用 Word2Vec 算法中的 Skip-Gram 模型学习节点 Embedding 向量。假设节点 v 的 Embedding 表示为 $\mathbf{e}_v$，上下文窗口大小为 c，则 Skip-Gram 的优化目标为

$$\min -\sum_{v \in V} \sum_{u \in \mathcal{N}_c(v)} \log P(u|v) \tag{10.80}$$

其中 $P(u|v)$ 的定义如下：

$$P(u|v) = \frac{\exp(\mathbf{e}_u^\top \mathbf{e}_v)}{\sum_{k \in V} \exp(\mathbf{e}_k^\top \mathbf{e}_v)} \tag{10.81}$$

由于完整 Softmax 需要遍历全部节点，因此实际训练通常采用 Negative Sampling 或 Hierarchical Softmax 进行加速。其中，Hierarchical Softmax 利用霍夫曼树（Huffman Tree）将计算复杂度由 $O(|V|)$ 降低至 $O(\log |V|)$，从而能够有效支持百万甚至千万级节点的图训练。

下面我们给出 DeepWalk 召回的核心实现代码，如下所示。

代码 10.4: DeepWalk 召回训练流程示例

```
import random
from gensim.models import Word2Vec

def random_walk(graph, start, walk_length):
    walk = [start]
    while len(walk) < walk_length:
        cur = walk[-1]
        neighbors = graph[cur]
        if len(neighbors) == 0:
            break
        walk.append(random.choice(neighbors))
    return walk

def train_deepwalk(graph, walk_length=40, num_walks=20, dim=128):
    walks = []
    for node in graph:
        for _ in range(num_walks):
            walks.append(random_walk(graph, node, walk_length))

    model = Word2Vec(walks, vector_size=dim, window=5, sg=1, negative=5, min_count=0)
    return model
```

10.2.1.1.4 DeepWalk 召回流程 在工业推荐系统中，DeepWalk 通常作为一种离线图 Embedding 召回模型，其整体流程可以分为离线训练和在线召回两个阶段。

在离线阶段，我们首先可以根据用户点击、浏览、点赞、购买等历史行为构建物品图（Item Graph）或作者图（Author Graph）。常见的做法是将用户连续交互、同 Session 共现或同一次曝光中的物品建立连接，并利用共

现次数、行为权重等作为边权。随后，在图上执行大量随机游走（Random Walk），生成节点访问序列，并利用 Skip-Gram 模型学习每个节点对应的 Embedding。训练完成后，将所有节点 Embedding 建立 ANN 索引（如 Faiss、ScaNN 或 HNSW 等），用于在线近邻检索。

在在线请求到来之后，推荐系统首先根据用户近期行为选择若干个 Trigger 物品作为查询种子（Seed Item）。工业界通常不会使用用户全部历史行为，而是选取最近一段时间内具有代表性的交互行为，例如最近点击、最近长观看、最近点赞、最近购买等，并分别限制不同 Trigger 类型的数量，例如：

$$\mathcal{T}_u = \{t_1, t_2, \cdots, t_n\} \tag{10.82}$$

其中，n 通常为 5~20。例如，可以选择最近 5 个点赞的物品、最近 5 个长播观看的物品以及最近 10 个点击的物品作为整个 Trigger 集合。

对于 Trigger 集合中的每一个 Trigger 物品，都利用 ANN 索引查询 Embedding 空间中的 Top-K 近邻，即

$$\mathcal{C}_i = ANN(t_i, K) \tag{10.83}$$

其中 $\mathcal{C}_i$ 表示 Trigger 物品对应的候选集合。随后，将多个 Trigger 产生的候选集合进行融合、去重，并按照 Embedding 相似度或行为权重进行加权累积，例如：

$$Score(v) = \sum_{t_i \in \mathcal{T}_u} w_i \cdot Sim(t_i, v) \tag{10.84}$$

其中，v 表示召回出的候选物品，$Sim(\cdot)$ 表示 Embedding 相似度，w_i 表示不同 Trigger 对应的行为权重，例如点赞通常高于点击，购买通常高于浏览。最后，对所有候选物品按照融合得分重新排序，并截断得到 Top-N 个召回结果。

由于 DeepWalk 召回学习得到的 Embedding 向量已经融合了图中的高阶邻域信息，因此，即使两个物品之间不存在直接共现关系，只要它们具有相似的图结构上下文，也能够获得较高的 Embedding 向量相似度。这使得 DeepWalk 召回相比传统 ItemCF 召回具有更强的高阶关联建模能力以及泛化能力，同时结合 ANN 向量检索系统，也能够使得 DeepWalk 召回满足工业级推荐系统中对于低延迟、高吞吐在线召回的要求。

10.2.1.1.5 DeepWalk 召回模型分析 DeepWalk 召回算法最大的贡献在于将图结构表示学习转化为序列建模问题，使 Word2Vec 方法能够直接应用于图结构的数据。相比起 ItemCF 召回等仅利用一阶共现关系的方法，DeepWalk 召回能够通过随机游走捕获节点之间的高阶结构信息，使得 DeepWalk 召回模型学习得到的 Embedding 具有更好的泛化能力。

不过，DeepWalk 召回的随机游走完全依赖图结构，无法利用节点属性信息，也无法区分不同类型节点之间的异构关系。此外，其随机游走策略属于均匀采样，对局部邻域和全局结构的平衡能力有限。后续提出的 Node2Vec、EGES 以及各种图神经网络模型，均是在 DeepWalk 基础上的进一步改进。

10.2.1.2 Node2Vec 召回

Node2Vec 算法是 Grover 等人在 2016 年提出的一种经典图嵌入算法，发表于 KDD 会议论文《Node2Vec: Scalable Feature Learning for Networks》[16]。它是在 DeepWalk 算法基础上的进一步改进，通过设计带偏置（Biased Random Walk）的随机游走策略，使随机游走能够在广度优先搜索（Breadth-First Search，BFS）和深度优先搜索（Depth-First Search，DFS）之间取得平衡，从而学习更加丰富的图结构表示。

10.2.1.2.1 Node2Vec 算法原理 相比于 DeepWalk 算法采用完全随机的游走方式，Node2Vec 算法认为不同的推荐任务更关注图结构中的不同信息。例如，在商品推荐场景中，希望具有相同类别、相同品牌的商品能够聚集在一起，此时更倾向于学习局部邻域结构；而在兴趣发现、长尾推荐等场景中，则更希望能够探索距离较远但潜在相关的节点。因此，Node2Vec 通过引入游走偏置，使 Embedding 既能够保持局部相似性，又能够学习更高阶的图结构关系。

Node2Vec 算法中 DFS 和 BFS 搜索策略的示意图如图10.8所示，其中 DFS 搜索策略更倾向于访问距离较远

的节点，而 BFS 搜索策略则更倾向于访问当前节点附近的邻居。通过调节 DFS 和 BFS 之间的平衡，Node2Vec 能够在局部邻域建模与全局结构建模之间取得较好的平衡。

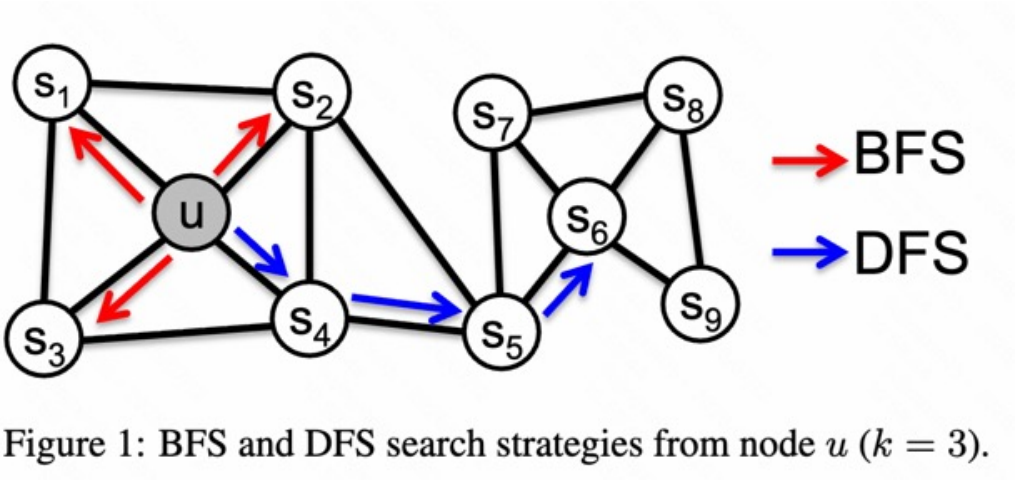

Figure 1: BFS and DFS search strategies from node u ($k = 3$).

图 10.8: Node2Vec 中的 DFS 与 BFS 搜索策略。

与 DeepWalk 算法不同的是，Node2Vec 算法的随机游走不仅依赖当前节点，还会考虑上一步访问的节点。具体来说，假设当前游走路径为

$$t \to v \to x \tag{10.85}$$

其中 t 表示上一时刻节点，v 表示当前节点，x 表示下一步候选节点，则随机游走的转移概率定义为

$$P(x|v,t) = \frac{\pi_{vx}}{\sum_{z \in \mathcal{N}(v)} \pi_{vz}} \tag{10.86}$$

其中

$$\pi_{vx} = \alpha_{pq}(t,x) \cdot w_{vx} \tag{10.87}$$

w_{vx} 表示边权重，$\alpha_{pq}(t,x)$ 表示偏置系数，其定义如下：

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p}, & d(t,x) = 0, \\ 1, & d(t,x) = 1, \\ \frac{1}{q}, & d(t,x) = 2, \end{cases} \tag{10.88}$$

其中 $d(t,x)$ 表示节点 t 与节点 x 之间的最短距离。

这种带偏置的随机游走策略如图10.9所示。需要注意的是，Node2Vec 算法中的偏置随机游走策略通过两个超参数 p 和 q 分别控制随机游走的方向。当 p 较大时，随机游走返回上一节点的概率较低，更倾向于继续向前探索；当 p 较小时，则更容易回到上一节点，从而增强局部结构学习。而当 $q < 1$ 时，算法更加倾向于访问距离较远的节点，随机游走更接近 DFS，有利于学习社区之间的高阶关系；当 $q > 1$ 时，则更倾向于访问当前节点附近的邻居，随机游走更接近 BFS，更容易学习局部相似节点。因此，通过调节 p 和 q 这两个超参数，Node2Vec 算法能够在局部邻域建模与全局结构建模之间取得较好的平衡。

下面给出 Node2Vec 算法带偏置随机游走的核心实现代码示例，具体如下所示。

代码 10.5: Node2Vec 算法带偏置随机游走代码示例

```
import random
def node2vec_walk(graph, start_node, walk_length=40, p=1.0, q=0.5):
    walk = [start_node]
    while len(walk) < walk_length:
        cur = walk[-1]
        neighbors = graph[cur]
```

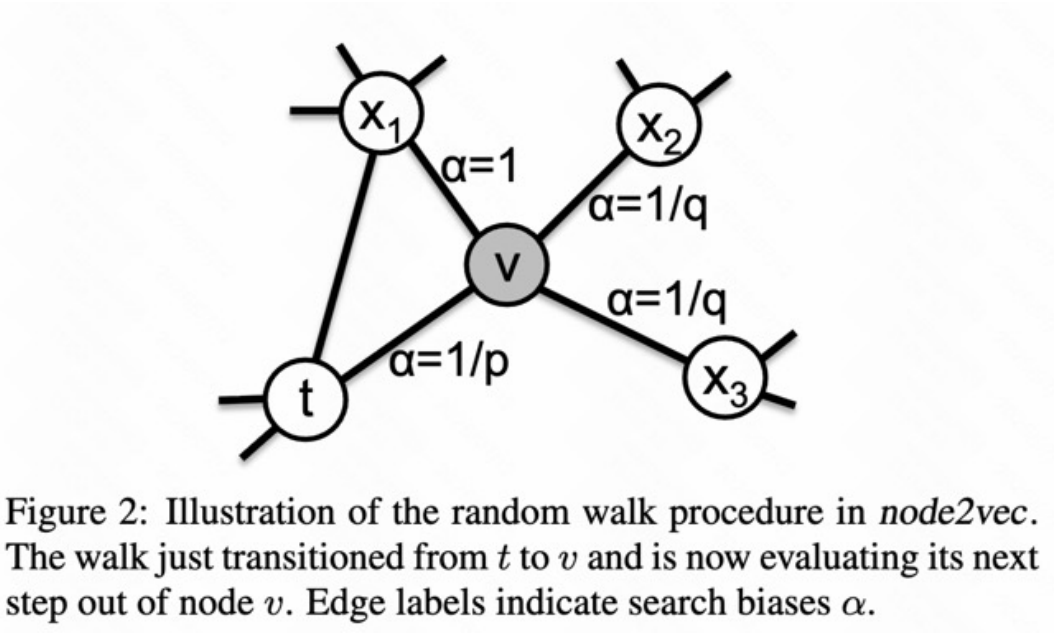

Figure 2: Illustration of the random walk procedure in *node2vec*. The walk just transitioned from t to v and is now evaluating its next step out of node v. Edge labels indicate search biases α.

图 10.9: Node2Vec 偏置随机游走示意图。

```
        if len(neighbors) == 0:
            break
        if len(walk) == 1:
            walk.append(random.choice(neighbors))
            continue
        prev = walk[-2]
        weights = []
        for nxt in neighbors:
            if nxt == prev:
                bias = 1.0 / p
            elif graph.has_edge(prev, nxt):
                bias = 1.0
            else:
                bias = 1.0 / q
            weights.append(bias)
        walk.append(random.choices(neighbors, weights=weights, k=1)[0])
    return walk
```

10.2.1.2.2 Node2Vec 召回工程实现 Node2Vec 召回在工业推荐系统中的整体架构与 DeepWalk 召回基本一致，同样采用“离线训练 + 在线检索”的两阶段架构。其区别主要体现在离线图 Embedding 训练过程中采用了带偏置（Biased Random Walk）的随机游走策略，而在线召回阶段仍然采用 ANN 向量检索完成候选召回。

Node2Vec 召回首先需要根据用户行为日志来构建图结构。在推荐系统中，最常见的构建方式是构建 Item 图（Item Graph）或 Author 图（Author Graph）。在具体实践中，我们通常将用户连续点击、连续观看、购买、点赞等行为视为物品之间存在着某种关联关系，当同一个用户在一定的时间窗口内连续与两个物品发生交互行为时，则会在这两个物品之间连接一条边，并利用物品共现次数、用户观看时长、用户行为权重等统计信息来作为边权。除了 Item 图之外，在实际业务中还可以根据不同场景构建 Author 图、Topic 图、Category 图以及异构用户-物品图等，从而学习不同实体之间的 Embedding 表示。

在完成图构建之后，Node2Vec 召回首先利用带偏置的随机游走算法生成大量节点访问序列。相比 DeepWalk

召回采用均匀随机游走的策略，Node2Vec 召回能够根据超参数 p 和 q 来动态调整整个随机游走的方向，在局部邻域建模与全局结构建模之间取得一定的平衡。

通过带偏置的随机游走策略生成完大量的节点序列之后，Node2Vec 召回利用 Skip-Gram 模型学习每个节点对应的 Embedding 表示。Node2Vec 召回的整个离线训练流程通常包括如下几个步骤：

1. 根据用户行为日志构建 Item 图或 Author 图；
2. 在图结构上执行 Node2Vec 算法的偏置随机游走策略，生成大量节点序列；
3. 将节点序列作为 Skip-Gram 模型的训练语料，学习节点的 Embedding 表示；
4. 根据训练得到的 Embedding 向量建立 ANN 索引，并部署至在线向量检索服务。

在训练完成后，每个节点都会对应一个 d 维 Embedding 向量，可直接作为在线召回阶段的向量索引。

在线上阶段，一次用户请求到来时，推荐系统首先会根据用户近期行为历史选择若干个触发物品作为召回种子，而不会直接使用用户全部历史行为。工业界通常会针对不同类型的行为分别选择一定数量的触发物品 (Trigger Item)，例如最近点击、最近长观看、最近点赞、最近购买等行为，每类行为分别保留最近若干个物品，从而构成触发物品集合

$$\mathcal{T}_u = \{t_1, t_2, \cdots, t_n\} \tag{10.89}$$

其中 n 通常取 5∼20。例如，可以选择最近 10 个点击物品、最近 5 个长观看物品以及最近 5 个点赞物品作为召回触发源。随后，对于每一个触发物品，利用 ANN 索引查询 Embedding 空间中的 Top-K 近邻：

$$\mathcal{C}_i = ANN(t_i, K) \tag{10.90}$$

每个触发物品通常召回几十到数百个候选物品，多个触发物品得到的候选集合进一步进行合并、去重，并按照 Embedding 相似度和行为权重进行融合。例如，可以采用如下加权方式：

$$Score(v) = \sum_{t_i \in \mathcal{T}_u} w_i \cdot Sim(t_i, v) \tag{10.91}$$

其中，$Sim(\cdot)$ 表示 Embedding 相似度，w_i 表示不同 Trigger 对应的行为权重。例如，购买行为通常高于点赞，点赞行为高于点击，长观看行为则高于普通曝光。最后，推荐系统将所有的候选物品按照融合后的得分进行重新排序，并截断得到 Top-N 个召回结果。

总结起来，Node2Vec 召回最大的改进在于随机游走策略。相比起 DeepWalk 召回固定采用均匀随机游走，Node2Vec 召回能够根据参数 p 和 q 灵活调整图结构的探索方式，因此学习得到的 Embedding 通常能够更好地兼顾局部邻域关系和高阶结构关系。在推荐系统中，当希望学习类别相近、属性相似的物品关系时，可以适当提高 q，使随机游走更加偏向 BFS；而对于兴趣扩散、跨类别推荐或长尾物品发现等任务，则可以适当减小 q，使随机游走更加偏向 DFS，从而学习更加丰富的高阶关联关系。因此，相比 DeepWalk 召回，Node2Vec 召回通常能够获得更加准确的物品 Embedding 向量，在跨兴趣迁移、长尾推荐以及兴趣扩展等任务中具有更好的表现。

不过，Node2Vec 召回本质上仍属于离线图嵌入模型，需要周期性地重新构建图结构并完成 Embedding 训练，因此难以像 Online CF 或 Online Swing 那样实时响应用户兴趣变化。工业界通常将 Node2Vec 召回作为离线 Embedding 召回的重要组成部分，与双塔召回、图神经网络召回以及统计召回等多种召回方式共同组成多路召回架构。

10.2.1.3 EGES 召回

前面介绍的 DeepWalk 和 Node2Vec 召回都是典型的 Graph Embedding 方法，它们通过随机游走（Random Walk）和 Skip-Gram 模型来学习图中每个节点的 Embedding 表示。然而，这类方法仅利用了图结构本身，而没有考虑节点自身所携带的**大量属性信息（Side Information）**。在工业级推荐系统中，一个物品通常不仅对应一个物品 ID，通常还包含品牌（Brand）、类目（Category）、店铺（Shop）、价格区间（Price Level）以及作者（Author）等丰富的属性字段。这些属性字段往往蕴含着重要的语义信息，如果只依赖用户行为构建的图结构很难充分利用这些信息。例如，一个新上线的商品虽然几乎没有任何用户交互行为，但它已经具有品牌、类目、店铺等属性信息。如果只采用 DeepWalk 或 Node2Vec 算法来学习物品的 Embedding 表示，那么由于缺乏足够的随机游走路

径，这种商品就难以学习到高质量的向量表示。这也是传统 Graph Embedding 方法在推荐系统中面临的重要冷启动问题。

为了充分利用物品的属性信息，阿里巴巴在 2018 年提出了 EGES（Edge Enhanced Graph Embedding for Recommendation）模型，并发表了 WWW 会议论文《EGES: Graph Embedding with Side Information for Recommendation》[52]。EGES 召回在 Graph Embedding 的基础上进一步引入多种属性信息，并根据不同属性的重要程度自适应地学习各类属性的贡献权重，从而获得更加准确的节点 Embedding 表示。

需要说明的是，EGES 召回实际上是在 GES（Graph Embedding with Side Information）模型基础上的进一步改进。GES 算法首先提出利用多个属性信息共同表示一个物品，而 EGES 算法进一步认为，不同类型的属性信息对于最终 Embedding 的重要程度并不相同，因此，EGES 论文中提出了可学习的加权融合（Weighted Average）机制，来自适应学习不同属性的重要性。

10.2.1.3.1 Side Information 建模 假设物品 v 包含 n 种 Side Information（简称 Side Info），例如 Item ID、Category、Brand、Shop、Author、Price Level 以及 Topic 等。对于每一种 Side Info，模型分别维护一个独立的 Embedding 矩阵。设 $\mathbf{W}_s(v) \in \mathbb{R}^d$ 表示物品 v 在第 s 类 Side Info 下对应的 Embedding，其中 $s=0$ 表示物品自身（Item ID）的 Embedding，而 $s=1,\cdots,n$ 分别表示不同类型属性对应的 Embedding。

例如，对于某个商品，其 Embedding 可以表示为

$$\{\mathbf{W}_0(v), \mathbf{W}_{brand}(v), \mathbf{W}_{shop}(v), \mathbf{W}_{category}(v), \mathbf{W}_{author}(v)\} \tag{10.92}$$

其中 $\mathbf{W}_0(v)$ 表示商品 ID 对应的 Embedding，$\mathbf{W}_{brand}(v)$ 表示商品品牌对应的 Embedding，$\mathbf{W}_{shop}(v)$ 表示商品店铺对应的 Embedding，$\mathbf{W}_{category}(v)$ 表示商品类目对应的 Embedding，$\mathbf{W}_{author}(v)$ 表示商品作者对应的 Embedding。

相比 DeepWalk 召回只能学习一个物品 Embedding，EGES 召回同时学习多个属性的 Embedding，从而能够利用更加丰富的语义信息。当新物品缺乏历史用户交互行为时，仍然可以利用商标、类目等属性 Embedding 来获得较好的初始化表示，因此能够有效缓解推荐系统中的冷启动问题。EGES 召回的整体结构如下图10.10所示。

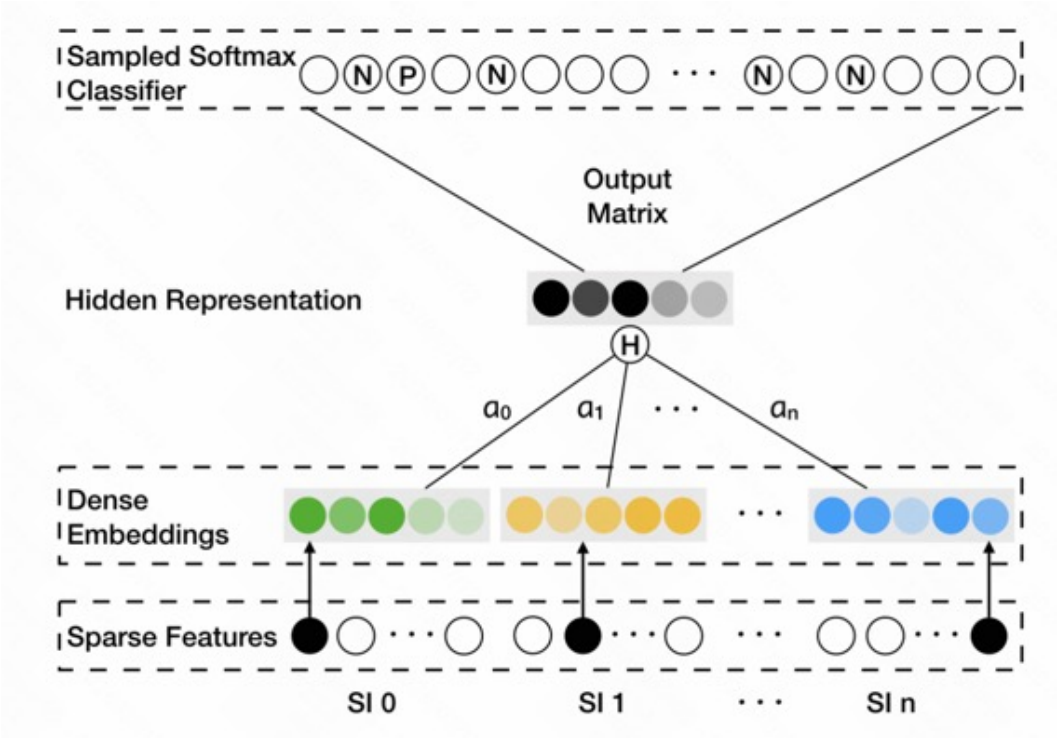

图 10.10: EGES 模型整体结构。

10.2.1.3.2 Weighted Average Layer GES 算法虽然利用了多种 Side Info，但该方法最大的不足之处在于它默认所有属性的重要程度完全一致，即直接采用平均池化（Average Pooling）的方式来得到最终的物品 Embedding：

$$\mathbf{H}_v = \frac{1}{n+1}\sum_{s=0}^{n}\mathbf{W}_s(v) \tag{10.93}$$

然而，这一假设在实际推荐系统中往往并不成立。不同属性对于用户行为的影响通常存在明显差异。例如，在电子产品推荐中，品牌（Brand）通常具有较大的影响力，购买 iPhone 的用户往往更容易继续浏览 MacBook、iPad 等 Apple 产品；而在电商服饰场景中，用户更多受到店铺（Shop）或价格因素的影响，同一店铺中的不同品牌商品也可能被一起购买。因此，不同 Side Information 对于最终 Embedding 的贡献应当具有不同的重要程度，而不是简单平均。

针对这一问题，EGES 算法提出了 Weighted Average Layer 的网络层，来为每一种 Side Info 学习一个可训练的权重参数。Weighted Average Layer 的参数矩阵为 $\mathbf{A} \in \mathbb{R}^{|V|\times(n+1)}$，$a_v^s = \mathbf{A}_{vs}$ 表示参数矩阵 $\mathbf{A}$ 中物品 v 的第 s 种 Side Info 对应的权重，则 EGES 算法中最终的物品 Embedding 可以定义为：

$$\mathbf{H}_v = \frac{\sum_{s=0}^{n} e^{a_v^s}\mathbf{W}_s(v)}{\sum_{s=0}^{n} e^{a_v^s}} \tag{10.94}$$

其中，指数函数 $e^{a_v^s}$ 保证所有权重均为正数，而分母则完成归一化，使不同 Side Information 的贡献之和为 1。因此，公式 (10.94) 本质上可以看作一种 Softmax 加权平均，其思想与 Attention 机制具有一定的相似性。

相比起固定平均池化的操作，Weighted Average Layer 能够根据训练数据自动学习不同属性的重要程度。例如，当品牌属性对于当前商品更加重要时，其对应权重将自动增大；而对于其他影响较小的属性，其权重则会逐渐减小。因此，最终得到的物品 Embedding 能够更加准确地融合不同来源的语义信息。

完成 Side Info 融合之后，EGES 召回得到节点最终的 Embedding 表示 $\mathbf{H}_v$。随后，与 DeepWalk 和 Node2Vec 召回类似，EGES 召回仍然采用 Skip-Gram 模型进行训练，只不过 Skip-Gram 模型的输入由原来的物品 Embedding 替换为了融合后的 Embedding 表示 $\mathbf{H}_v$。

10.2.1.3.3 Graph Embedding 构建过程 EGES 召回的 Graph Embedding 构建过程如图10.11所示，整个流程主要包括用户行为序列构建、Item 图生成、随机游走以及 Embedding 训练四个步骤。与传统 ItemCF 召回直接统计物品的共现关系不同，EGES 召回充分利用了用户行为的时序信息，将用户连续访问的行为转化为图结构，并进一步学习物品在低维空间中的 Embedding 表示。

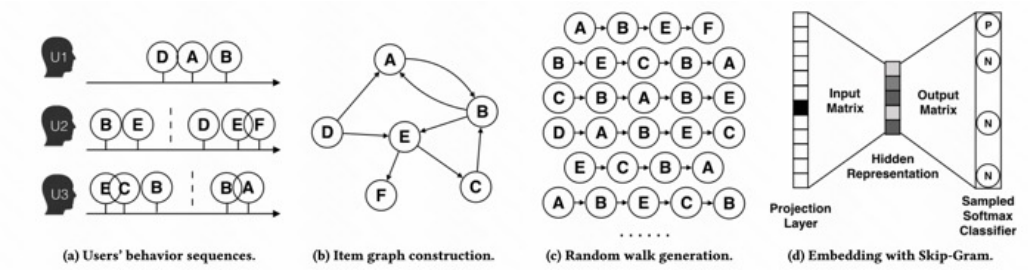

图 10.11: EGES 召回的 Graph Embedding 构建过程。

图10.11(a) 展示了用户行为序列的构建过程。在工业级推荐系统中，通常会根据用户的点击、浏览、购买等行为日志来构建用户行为序列。考虑到用户兴趣具有较强的时效性，如果直接利用整个历史行为序列，不仅计算开销很大，而且容易受到用户兴趣漂移（Interest Drift）的影响。因此，实际工程中通常采用 Session 机制，将用户在固定时间窗口内（例如 1 小时）或者一次打开 APP 的连续行为划分为一个 Session，然后只利用 Session 内的行为来构建图结构。

假设用户在某一个 Session 内的行为序列可以表示为 $S_u = (i_1, i_2, \cdots, i_n)$，其中 i_t 表示用户依次访问的物品。随后，根据所有用户的 Session 构建 Item 图，如图10.11(b) 所示。当两个物品在同一 Session 中连续出现时，即：$i_t \to i_{t+1}$，则在图中建立一条由 i_t 指向 i_{t+1} 的有向边。然后进一步统计所有 Session 中的连续转移次数，将

边权定义为

$$w_{ij} = Count(i \to j) \tag{10.95}$$

其中 $Count(i \to j)$ 表示物品 i 转移至物品 j 的累计次数。因此，最终可以得到一个带权有向图 $G = (V, E, W)$，其中 V 表示节点集合，E 表示边集合，W 表示边权集合。在这个带权有向图中，节点表示物品，边表示用户行为转移关系，边权反映不同物品之间的转移强度。

为了降低噪声对图结构的影响，在构图之前通常还需要进行用户行为数据的清洗。例如，停留时间过短的误点击行为通常会被过滤掉；短时间内产生大量 hack 点击或购买行为的异常用户（Spam User）也会被剔除；此外，对于已经失效、下架或内容发生较大变化的物品，同样也需要从图中进行移除，从而保证构建好的图结构能够真实地反映出物品之间的关联关系。

在完成图构建之后，EGES 召回在带权有向图上执行 Random Walk 策略生成节点访问序列，如图10.11(c) 所示。Random Walk 策略从每个节点出发，根据边权随机访问邻居节点，生成大量节点序列。这些节点序列与自然语言中的句子具有相似的形式，因此可以作为后续 Embedding 训练的语料。

最后，如图10.11(d) 所示，将 Random Walk 策略生成的节点序列输入 Skip-Gram 模型来进行训练，从而学习到每个物品对应的 Embedding 表示。在 DeepWalk 和 Node2Vec 召回中，Skip-Gram 模型通常直接学习节点 Embedding；而 EGES 召回则进一步引入属性信息，对多个 Embedding 向量进行融合，并且联合优化物品 Embedding 与属性信息的 Embedding，从而获得更加准确的物品表示。

10.2.1.3.4 EGES 召回模型训练流程 总结起来，EGES 召回模型的整体训练流程包括划分为图构建、随机游走、Side Information 融合以及 Weighted Skip-Gram 训练四个阶段，其整体训练算法如代码10.6所示。

代码 10.6: EGES 训练流程示例代码

```
def train_eges(graph, side_info, num_walks, walk_length, window_size, num_negative, embed_dim):
    # 1. 初始化Embedding
    item_embedding = initialize_embedding(graph.num_nodes(), embed_dim)
    side_embeddings = initialize_side_embedding(side_info, embed_dim)

    # Side Information权重
    weight_matrix = initialize_weight(graph.num_nodes(), side_info.num_types())

    # 2. 多轮Random Walk
    for _ in range(num_walks):
        for node in graph.nodes():
            # Random Walk生成节点序列
            walk_seq = random_walk(graph, start=node, walk_length=walk_length)
            # Weighted Skip-Gram训练
            weighted_skipgram(walk_seq, item_embedding, side_embeddings, weight_matrix, window_size,↩
                ↪ num_negative)
    return item_embedding, side_embeddings, weight_matrix
```

与 DeepWalk 和 Node2Vec 这两种召回不同的是，EGES 召回在标准 Skip-Gram 模型的基础上进一步提出了 Weighted Skip-Gram 的方法。它的核心思想如下：

定义 10.8

在 Weighted Skip-Gram 模型的训练过程中，中心节点首先根据多个 Side Info 动态融合得到最终的物品 Embedding，然后再进行 Skip-Gram 训练。

基于 Random Walk 生成的节点访问序列 $Seq = (v_1, v_2, \cdots, v_L)$，对于序列中的每一个中心节点 v_i，首先根

据窗口大小 k 确定其上下文节点集合

$$Context(v_i) = \{v_j \mid |i-j| \leq k,\ j \neq i\} \tag{10.96}$$

随后，将中心节点与窗口内的上下文节点组成正样本 $(v_i, u, 1)$。与此同时，为了提高训练效率，EGES 召回采用 Negative Sampling 策略来构造负样本。具体而言，对于每一个正样本，从整个全局物品集合中按照节点出现频率的幂律分布（通常采用节点频率的 3/4 次幂）随机采样若干个未出现在当前上下文中的物品，构造负样本 $(v_i, u, 0)$，其中负采样数量通常在 20 ~ 50 之间。因此，Skip-Gram 训练过程实际上是通过区分正样本与负样本来学习节点及其 Side Info 的 Embedding 表示，而无需计算完整的 Softmax 函数，从而显著降低在大规模图上训练的复杂度。

具体来说，对于每一个训练样本，首先根据公式 (10.94) 计算融合后的中心节点 Embedding

$$\mathbf{H}_v = \sum_{s=0}^{n} \beta_s \mathbf{W}_v^{(s)} \tag{10.97}$$

其中，$\mathbf{W}_v^{(s)}$ 表示节点 v 在第 s 类 Side Info 下的 Embedding，β_s 表示第 s 类 Side Info 的可学习权重。然后，EGES 召回利用 Skip-Gram 模型的优化目标计算损失函数，即：

$$L(v, u, y) = -\left[y \log \sigma(\mathbf{H}_v^\top \mathbf{Z}_u) + (1-y)\log(1-\sigma(\mathbf{H}_v^\top \mathbf{Z}_u))\right] \tag{10.98}$$

其中，$y=1$ 表示正样本，$y=0$ 表示负样本，$\mathbf{Z}_u$ 表示上下文节点 u 的 Embedding 向量，$\sigma(\cdot)$ 表示 Sigmoid 函数。通过最小化损失函数，EGES 召回能够同时优化中心节点 Embedding 向量、上下文节点 Embedding 向量以及各类 Side Info 的 Embedding 向量，从而获得更加准确的物品表示。

与普通 Skip-Gram 模型最大的区别在于，EGES 模型不仅需要更新上下文节点的 Embedding 向量，还需要同时更新各类 Side Info 对应的 Embedding 向量以及融合权重，因此梯度会分别传播到三个部分：上下文节点 Embedding、Side Info Embedding 以及 Side Info 对应的权重参数。

整个 Weighted Skip-Gram 模型的训练过程如代码10.7所示。

代码 10.7: Weighted Skip-Gram 训练流程伪代码

```
import numpy as np
def weighted_skipgram(sequence, item_embedding, side_embedding, weight_matrix, window_size,
                      num_negative, learning_rate):
    seq_len = len(sequence)
    for center in range(seq_len):
        v = sequence[center]
        left = max(0, center - window_size)
        right = min(seq_len, center + window_size + 1)

        # 遍历上下文窗口
        for pos in range(left, right):
            if pos == center:
                continue
            u = sequence[pos]
            # ---------- 正样本 ----------
            update(center=v, context=u, label=1, item_embedding=item_embedding,
                side_embedding=side_embedding, weight_matrix=weight_matrix, lr=learning_rate
            )

            # ---------- 负采样 ----------
            for _ in range(num_negative):
                # 按Unigram^{3/4}分布采样负样本
```

```
            neg = negative_sampling()
            update(center=v, context=neg, label=0, item_embedding=item_embedding,
                side_embedding=side_embedding, weight_matrix=weight_matrix, lr=learning_rate
            )


def update(center, context, label, item_embedding, side_embedding, weight_matrix, lr):
    # ---------- Step1: 计算加权Embedding ----------
    H = weighted_average_embedding(center, item_embedding, side_embedding, weight_matrix)
    # Context节点Embedding
    Z = item_embedding[context]

    # ---------- Step2: Skip-Gram预测 ----------
    logit = np.dot(H, Z)
    pred = sigmoid(logit)
    # Binary Cross Entropy Loss
    loss = -(label*np.log(pred + 1e-8) +(1 - label)*np.log(1 - pred + 1e-8))

    # ---------- Step3: 梯度更新 ----------
    grad = pred - label
    # 更新Context Embedding
    item_embedding[context] -= lr * grad * H
    # 更新各Side Info权重
    update_side_weight(center, grad, Z, side_embedding, weight_matrix, lr)
    # 更新各Side Info Embedding
    update_side_embedding(center, grad, Z, side_embedding, weight_matrix, lr)
    return loss
```

需要注意的是，上述代码中的负采样仅用于说明 Weighted Skip-Gram 的训练流程。实际训练前，通常需要首先根据随机游走生成的节点序列统计每个节点的出现频率，并按照 Unigram 分布的 3/4 次幂构建负采样概率，即

$$P(v) = \frac{f(v)^{3/4}}{\sum_{u \in V} f(u)^{3/4}}, \tag{10.99}$$

其中 $f(v)$ 表示节点 v 在随机游走序列中的出现频率。相比直接按照节点频率进行采样，采用 3/4 次幂能够适当降低高频节点的采样概率，同时提高长尾节点被采样的机会，从而获得更好的 Embedding 训练效果。

由于工业级推荐系统中的节点规模通常达到数千万甚至上亿，若每次负采样都遍历完整的概率分布进行随机采样，其时间复杂度较高。因此，实际工程中通常会预先根据上述概率分布构建 Alias Table（别名采样表）。Alias Table 通过预处理将任意离散概率分布转换为两张查找表（Probability Table 和 Alias Table），使得每次采样仅需随机生成一个整数索引和一个 $[0, 1]$ 之间的随机数，再经过一次概率比较即可完成采样，从而将单次采样的时间复杂度由 $O(N)$ 降低到 $O(1)$。因此，Alias Table 也是目前 Word2Vec、DeepWalk、Node2Vec 以及 EGES 等图嵌入模型最常用的负采样实现方式。

此外，在在线推荐或流式训练场景中，也常采用 Reservoir Sampling（蓄水池采样）维护固定大小的动态样本池，从持续到来的行为流中随机抽取训练样本，以支持在线模型更新。由于本书主要关注 EGES 召回模型本身的训练流程，因此这里只给出一个简化的负采样实现逻辑，具体算法如代码10.8所示。

代码 10.8: Unigram$^{3/4}$ 负采样代码

```
import numpy as np
```

```
def build_sampling_table(item_frequency, power=0.75):
    """
    根据节点出现频率构建负采样概率
    """
    item_ids = list(item_frequency.keys())
    freq = np.array([item_frequency[i] for i in item_ids], dtype=np.float64)

    sampling_prob = np.power(freq, power)
    sampling_prob /= sampling_prob.sum()
    return item_ids, sampling_prob


def negative_sampling(item_ids, sampling_prob):
    """
    根据Unigram^{3/4}分布采样一个负样本
    """
    return np.random.choice(item_ids, p=sampling_prob)
```

10.2.1.3.5 EGES 召回工程实现 EGES 召回的整体流程与 DeepWalk、Node2Vec 召回基本一致，不同之处主要体现在 Embedding 训练阶段。首先，根据用户点击、观看、购买等行为日志构建物品图（Item Graph），并收集每个 Item 对应的 Side Info，例如 Category、Brand、Shop、Seller、Author 等属性。随后，在图结构上执行 Random Walk 策略生成节点访问序列，并采用 Weighted Skip-Gram 模型联合学习 Item Embedding 与各类 Side Info Embedding。

在训练完成之后，每个物品都会得到最终融合之后的 Embedding 表示。工业界通常将所有的物品 Embedding 写入 ANN 向量检索系统，进而用于在线 Embedding 向量召回。

在线阶段，EGES 召回的 Trigger 选择方式与 DeepWalk 召回基本一致。推荐系统首先根据用户最近点击、长观看、点赞、购买等高价值行为构建 Trigger 集合 $\mathcal{T}_u = \{t_1, t_2, \cdots, t_n\}$，通常仅保留最近 5 ∼ 20 个行为作为 Trigger。对于每个 Trigger 物品，推荐系统首先查询该物品对应的 Embedding 向量，然后利用 ANN 索引检索 Top-K 近邻，即

$$Retrieve(t_i) = ANN(E_{t_i}) \tag{10.100}$$

随后将多个 Trigger 返回的候选集合进行融合与去重，并按照 Embedding 相似度、行为权重以及时间衰减等因素重新排序，最终得到 EGES 召回候选集合。

总结起来，EGES 召回可以看作 DeepWalk 和 Node2Vec 召回在工业级推荐场景下重要的改进版本。相比起只利用图结构学习节点的 Embedding，EGES 召回进一步将 Category、Brand、Shop、Seller 等 Side Info 引入 Embedding 学习的过程，并通过可学习的加权融合机制自动学习不同属性的重要性，从而避免人为设定各类属性权重。

另一方面，EGES 召回的整体训练流程仍然保持了 Random Walk + Skip-Gram 的框架，能够方便地复用 DeepWalk 和 Node2Vec 召回已有的大规模训练框架，因此在工业界具有较好的可扩展性。目前，EGES 召回已广泛应用于电商推荐、广告推荐以及内容推荐等场景，并成为工业界 Graph Embedding 召回的一种重要代表方法，具有很强的工程落地价值。

不过，EGES 召回仍属于基于随机游走的静态图嵌入模型，需要定期重新执行 Random Walk 并完成 Embedding 训练，因此比较难适应动态图结构上的实时更新。随着图神经网络的发展，越来越多的推荐系统开始采用 GNN 直接进行端到端的图表示学习，但 EGES 召回由于训练效率高、部署简单、工程成熟，在工业召回系统中仍然具有广泛应用价值。此外，由于 EGES 召回能够充分利用物品的属性信息，因此在冷启动物品的召回任务中仍然具有较好的表现。

10.2.2 图网络召回

前面介绍的 DeepWalk、Node2Vec、EGES 等方法本质上都属于基于 Graph Embedding 的召回模型，它们通常首先通过随机游走生成节点序列，再利用 Skip-Gram 模型学习节点的 Embedding 表示，因此属于一种**“图结构上的无监督表示学习”方法**。这类方法实现简单、训练效率较高，在工业界得到了广泛应用。然而，它们的图结构信息主要来源于随机游走产生的共现关系，并不能直接利用邻居节点之间的特征传播过程，因此对于复杂图结构的建模能力仍然存在一定局限。

近年来，图神经网络（Graph Neural Network，GNN）的提出进一步推动了图召回的发展。与 Graph Embedding 模型不同，GNN 模型通常直接以图结构作为模型的输入，然后通过多层**消息传递网络层（Message Passing Layer）**不断聚合邻居节点的信息，从而学习节点表示。随着网络层数的增加，节点能够逐渐感知更远距离的高阶邻居关系，因此相比传统 Graph Embedding 模型，GNN 模型通常具有更强的表示学习能力。

目前推荐系统中常见的图神经网络包括 GCN（Graph Convolutional Network）、GAT（Graph Attention Network）、GraphSAGE、LightGCN 等。除此之外，工业界还提出了 PinSage、HybridGNN 等专门面向推荐场景的大规模图学习模型。下面我们首先介绍最经典的 GCN 模型。

10.2.2.1 GCN 召回

GCN（Graph Convolutional Network）模型最早由 Kipf 等人在 2016 年提出，对应论文《Semi-Supervised Classification with Graph Convolutional Networks》[26]。GCN 首次将卷积神经网络推广到了图结构数据上，使节点能够通过不断聚合邻居信息学习更加丰富的表示，因此成为图神经网络领域一个具有基石性意义的工作。

10.2.2.1.1 GCN 算法原理 GCN 中的图卷积最早来源于谱图卷积（Spectral Graph Convolution）理论。与卷积神经网络在欧式空间上的卷积不同，图结构数据不存在规则的空间邻域，因此无法直接定义传统卷积操作。谱图卷积的基本思想是借助图拉普拉斯矩阵（Graph Laplacian）的特征分解，将节点特征映射到图傅里叶域（Graph Fourier Domain）进行滤波，再映射回原始图空间。

假设图的归一化拉普拉斯矩阵表示为

$$\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T \tag{10.101}$$

其中 $\mathbf{U}$ 表示拉普拉斯矩阵对应的特征向量矩阵，$\mathbf{\Lambda}$ 表示特征值构成的对角矩阵，则图上的谱卷积可以表示为

$$g_\theta * x = \mathbf{U} g_\theta(\mathbf{\Lambda})\mathbf{U}^T x \tag{10.102}$$

然而，上式需要对整个图进行特征值分解，其计算复杂度约为 $O(N^2)$，对于工业级推荐系统中的超大规模图而言几乎不可行。

为了降低计算复杂度，Hammond 等人提出利用 Chebyshev 多项式对谱卷积进行近似，将滤波器展开为 K 阶 Chebyshev 多项式：

$$g_\theta(\mathbf{\Lambda}) \approx \sum_{k=0}^{K} \theta_k T_k(\tilde{\mathbf{\Lambda}}), \tag{10.103}$$

其中 $T_k(\cdot)$ 表示 Chebyshev 多项式。进一步地，GCN 论文仅保留一阶邻域（$K=1$），并令最大的特征值 $\lambda_{max} \approx 2$，最终将图卷积进一步简化为

$$g_\theta * x \approx \theta\left(\mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\right) x \tag{10.104}$$

为了避免网络不断堆叠过程中数值不稳定的问题，GCN 进一步提出了 Renormalization Trick（重归一化技巧），即

$$\mathbf{A} + \mathbf{I} \to \tilde{\mathbf{A}}, \qquad \tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij} \tag{10.105}$$

最终便得到 GCN 模型中广泛采用的传播公式。

GCN 召回的核心在于图卷积（Graph Convolution）操作。对于第 l 层网络，每个节点都会聚合自身以及邻居

节点的信息，并经过线性变换得到下一层表示，其传播公式可以写成

$$\mathbf{H}^{(l+1)} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right) \tag{10.106}$$

其中，

$$\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I} \tag{10.107}$$

表示加入自连接（Self-loop）的邻接矩阵，$\tilde{\mathbf{D}}$ 表示对应的度矩阵，$\mathbf{H}^{(l)}$ 表示第 l 层节点 Embedding，$\mathbf{W}^{(l)}$ 表示可学习参数，$\sigma(\cdot)$ 通常采用 ReLU 等非线性激活函数。需要注意的是，$\mathbf{H}^{(0)} = \mathbf{X} \in \mathbb{R}^{|V|\times F}$ 表示节点的初始特征向量，其中 N 为节点数量，F 为特征维度。此外，$\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ 是对邻接矩阵进行归一化处理，能够避免节点度数差异过大导致的梯度消失或梯度爆炸问题。

从上述公式可以看出，每经过一层图卷积层，每个节点都会融合一阶邻居的信息；经过两层图卷积层，则能够聚合二阶邻居的信息；经过 L 层图卷积层之后，节点理论上能够感知 L 跳邻域的信息。因此，相比传统 ItemCF 只能建模一阶共现关系，GCN 能够直接学习更加丰富的高阶图结构关系。

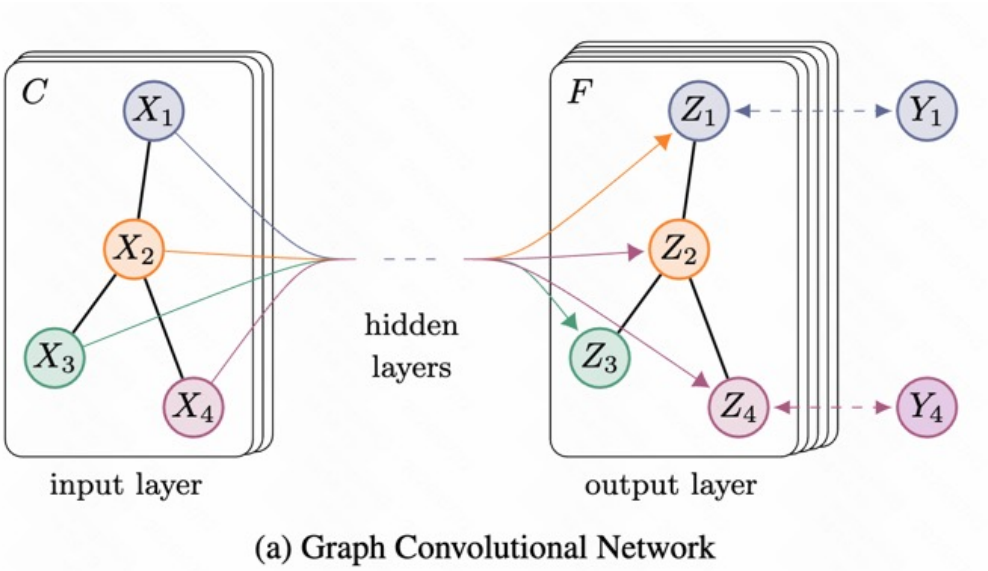

图 10.12: GCN 召回网络结构示意图。

GCN 召回模型的网络结构如图10.12所示，通常会堆叠多层的图卷积层来提升模型对于图节点之间高阶关系的建模能力。相比起 Graph Embedding 的方法，GCN 模型最大的优势在于能够直接利用图结构进行特征传播，从而学习更加丰富的高阶节点表示，对于复杂关系建模通常具有更好的效果。然而，GCN 也存在一些局限性。首先，每层 GCN 都需要利用完整的图邻接矩阵来进行邻居聚合，当图规模达到数亿节点时，训练和推理开销都会显著增加；其次，随着网络层数不断加深，图节点的 Embedding 容易逐渐趋于一致，即出现 Over-smoothing（过平滑）的问题，使不同节点之间的可区分性迅速下降。从图信号传播的角度来看，每一层 GCN 都会执行一次邻居特征平均操作，当网络不断加深时，各节点 Embedding 会逐渐趋近于图拉普拉斯矩阵对应的低频子空间，使得不同节点之间的表示越来越相似，最终导致模型区分能力明显下降。因此，在实际工业应用中，GCN 模型通常只堆叠 2～3 层，很少采用更深的网络结构。

GCN 模型的原始论文主要针对节点分类任务，因此采用半监督学习（Semi-Supervised Learning）的训练方式。假设图中仅有部分节点具有真实标签，则仅在有标签节点集合 $\mathcal{V}_L$ 上计算交叉熵损失：

$$\mathcal{L} = -\sum_{v_i \in \mathcal{V}_L}\sum_{c=1}^{C} y_{ic}\log \hat{y}_{ic} \tag{10.108}$$

其中，C 表示类别数量，y_{ic} 表示节点 v_i 对应类别 c 的真实标签，$\hat{y}_{ic}$ 表示 GCN 预测得到的类别概率。由于只有少量节点参与监督，因此 GCN 能够充分利用图结构传播标签信息，实现半监督学习。

为了进一步提升 GCN 模型在大规模图上的训练效率，后续研究又提出了 FastGCN [8]、GraphSAINT [61]、

Cluster-GCN [9] 等采样方法；同时，为了更适合推荐系统场景，又发展出了 GraphSAGE、GAT、LightGCN 以及 PinSage 等更加高效的图神经网络模型，这些方法目前也是工业界图召回的重要基础。

10.2.2.1.2 GCN 召回工程实现 在推荐系统中，GCN 召回通常首先根据用户行为构建图结构，然后通过多层图卷积不断聚合邻居节点信息，使节点能够融合多跳邻域的信息，最终学习得到每个节点对应的 Embedding 表示。根据图结构构建方式的不同，GCN 召回通常可以分为**用户-物品（User-Item）二部图召回**和**物品图（Item Graph）召回**两种典型形式。

对于 User-Item 二部图，图中的节点由用户节点和物品节点共同组成，边表示用户与物品之间的点击、浏览、点赞、购买等交互行为。经过多层图卷积之后，可以同时得到用户 Embedding 和物品 Embedding，因此该类模型通常用于 User-to-Item（U2I）召回。

对于 Item 图召回，则首先根据用户行为日志构建 Item 之间的关联关系。例如，当用户在同一个 Session 内连续点击两个物品时，则在两个 Item 之间建立一条边，并根据共现次数、点击次数、停留时长或者转化次数等统计信息作为边权。经过 GCN 传播之后，仅得到 Item 对应的 Embedding 表示，因此该类模型通常用于 Item-to-Item（I2I）召回，也是工业界更加常见的图召回方案。

通常来讲，GCN 召回包括如下几个步骤：

1. 根据用户点击、观看、购买等行为构建用户-物品二部图或 Item 图；
2. 为每个节点初始化 Embedding，可以采用随机初始化，也可以利用物品属性、文本、图像等预训练特征作为初始表示；
3. 利用多层 GCN 不断聚合邻居节点信息，学习节点新的表示；
4. 得到最终节点 Embedding 向量后建立 ANN 索引，并部署到在线向量检索系统中；
5. 在线阶段根据不同召回方式完成近邻检索：对于 U2I 召回，首先根据用户 Embedding 检索 Top-K 个最相似的 Item；对于 I2I 召回，则根据用户最近点击序列中若干 Trigger Item（例如最近 5～20 个点击物品），分别查询每个 Trigger Item 对应的 Top-K 近邻，然后对多个 Trigger 产生的候选集合进行融合、去重与排序，生成最终召回结果。

在 GCN 召回的模型训练阶段，不同图结构对应的监督目标也有所不同。对于 User-Item 二部图，通常利用用户行为构造监督信号。设用户 u 点击了正样本物品 i，同时随机采样负样本 j，则通常采用 BPR Loss 进行优化：

$$\mathcal{L}_{BPR} = -\sum_{(u,i,j)} \log \sigma(\hat{y}_{ui} - \hat{y}_{uj}) \tag{10.109}$$

其中，

$$\hat{y}_{ui} = \mathbf{e}_u^T \mathbf{e}_i \tag{10.110}$$

表示用户 Embedding 向量与物品 Embedding 向量之间的匹配得分。

对于 Item 图召回，由于图中不存在用户节点，因此训练样本通常来自物品之间的共现关系。例如，对于同一用户 Session 内连续访问的物品 (i, j) 作为正样本，而随机采样得到物品 k 作为负样本，则同样可以采用 Pairwise Ranking Loss 进行优化：

$$\mathcal{L}_{Item} = -\sum_{(i,j,k)} \log \sigma(\mathbf{e}_i^T \mathbf{e}_j - \mathbf{e}_i^T \mathbf{e}_k) \tag{10.111}$$

此外，对于 CTR 预测等任务，也可以采用 Binary Cross Entropy 作为训练目标：

$$\mathcal{L}_{BCE} = -\sum_{(u,i)} [y_{ui} \log \hat{y}_{ui} + (1 - y_{ui}) \log(1 - \hat{y}_{ui})] \tag{10.112}$$

其中，y_{ui} 表示用户 u 对物品 i 的真实点击标签，$\hat{y}_{ui}$ 表示 GCN 召回模型预测得到的点击概率。

近年来，为了进一步提升图表示的判别能力，许多 GCN 召回模型还会结合对比学习损失（Contrastive Loss）进行自监督训练，通过不同视角下图结构的一致性约束，使相同节点在不同图增强下学习到更加稳定的 Embedding

表示。在实际工业级推荐系统中，GCN 召回通常也会采用多任务联合优化的方式，将 BPR Loss、Binary Cross Entropy Loss 以及 Contrastive Loss 等目标进行加权组合，从而同时兼顾用户兴趣建模、排序能力以及 Embedding 空间的判别性，进一步提升 GCN 模型的召回效果。

10.2.2.2 GAT 召回

GAT（Graph Attention Network）召回最早由 Veličković 等人在 2018 年提出，对应 ICLR 会议论文《Graph Attention Networks》[50]。相比起 GCN 模型采用固定归一化邻接矩阵进行邻居聚合的方式，GAT 模型最大的创新在于引入了**图注意力机制（Graph Attention）**，使得模型能够根据不同邻居节点的重要程度，自适应地去学习不同的聚合权重，而不再对所有邻居进行平均聚合。因此，对于邻居数量差异较大、节点关系复杂或者图结构噪声较多的推荐场景，GAT 模型通常能够学习到更加具有判别性的节点表示。

从整体召回工程实现的流程来看，GAT 召回与 GCN 召回基本一致，同样包括图构建、节点 Embedding 初始化、多层图网络训练、ANN 索引构建以及在线向量检索等多个阶段。因此，本章节主要介绍 GAT 模型自身的网络结构与注意力机制，具体的召回流程、在线部署方式以及训练样本构建等内容可参考上一节 GCN 召回，在此不再重复介绍。

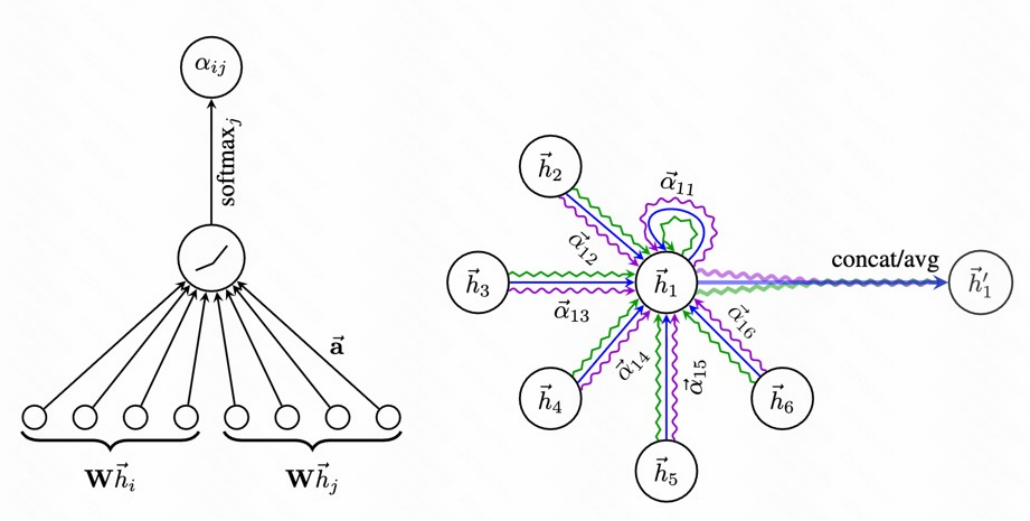

图 10.13: GAT 模型网络结构示意图。

GAT 模型的网络结构如图10.13所示。相比起 GCN 模型采用固定的邻接矩阵归一化权重进行消息传播，GAT 模型通过学习得到节点之间的注意力分数（Attention Score），使得每个节点能够根据邻居的重要程度进行加权聚合。对于节点 i 及其邻居节点 j，首先分别进行线性映射：

$$\mathbf{z}_i = \mathbf{W}\mathbf{h}_i, \qquad \mathbf{z}_j = \mathbf{W}\mathbf{h}_j \tag{10.113}$$

其中，$\mathbf{W}$ 表示共享的线性变换矩阵。随后，将中心节点与邻居节点的表示进行拼接，并利用一个共享的注意力向量计算未归一化的注意力分数：

$$e_{ij} = \text{LeakyReLU}\left(\mathbf{a}^T[\mathbf{z}_i \| \mathbf{z}_j]\right) \tag{10.114}$$

其中，$\mathbf{a}$ 表示可学习的注意力参数，$\|$ 表示向量拼接（Concatenation）。为了保证所有邻居权重之和为 1，需要对注意力分数进行 Softmax 归一化：

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum\limits_{k \in \mathcal{N}(i)} \exp(e_{ik})} \tag{10.115}$$

其中，α_{ij} 表示节点 i 分配给邻居 j 的注意力权重。最后，根据注意力权重完成邻居聚合：

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}^{(l)} \mathbf{W}^{(l)} \mathbf{h}_j^{(l)}\right) \tag{10.116}$$

其中，$\sigma(\cdot)$ 通常表示 ELU 或 ReLU 等非线性激活函数。

可以发现，相比起 GCN 模型固定使用 $\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ 作为传播权重，GAT 将邻居的重要程度交由模型自动学习，因此不同邻居对于中心节点的贡献可以完全不同。例如，在 User-Item 图中，一个用户最近点击的核心兴趣物品通常比很久以前浏览过的长尾物品更加重要；或者在 Item 图中，与当前物品高度相关的邻居也应当拥有更大的传播权重。上述这些关系都能够通过 Attention 机制自动学习得到。

为了进一步增强模型表达能力，GAT 通常采用多头注意力（Multi-Head Attention）机制。假设共有 K 个 Attention Head，则每个 Head 分别独立计算注意力权重：

$$\mathbf{h}_i^{(l+1)} = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^{(k)}\mathbf{W}^{(k)}\mathbf{h}_j\right) \tag{10.117}$$

其中，$\Vert$ 表示多个 Attention Head 输出的拼接（Concatenation）。在最后一层，也可以采用平均方式进行融合：

$$\mathbf{h}_i = \frac{1}{K}\sum_{k=1}^{K}\mathbf{h}_i^{(k)} \tag{10.118}$$

多头注意力能够从不同的子空间学习节点之间的关系，提高模型稳定性，同时降低单个 Attention Head 学习失败所带来的影响，因此也是 GAT 模型区别于普通 Attention 的重要特点。

在推荐系统中，GAT 模型既可以应用于 User-Item 二部图完成 User-to-Item（U2I）召回，也可以应用于 Item 图实现 Item-to-Item（I2I）召回。对于 U2I 场景，通常可以利用用户点击、观看、购买等行为构建二部图，通过图注意力聚合用户与物品之间的信息，最终学习到用户 Embedding 与物品 Embedding；而对于 I2I 场景，则可以根据物品之间的共现关系、Session 行为或者物品预训练 Embedding 的相似度构建物品图，然后学习带有图结构高阶信息的物品 Embedding，并部署到 ANN 检索系统完成相似物品召回。GAT 召回在模型训练阶段一般仍采用 BPR Loss、Binary Cross Entropy Loss 或者 Contrastive Loss 等目标函数，与 GCN 召回基本一致，因此这里不再展开介绍。

相比起 GCN 模型，GAT 模型最大的优势在于能够根据邻居的重要程度进行自适应加权聚合，不再依赖固定的归一化邻接矩阵，因此对于节点度数分布极不均衡或者图中存在大量噪声边的场景通常具有更好的表现。同时，由于 Attention 能够自动学习不同邻居的重要程度，因此 GAT 模型具有更强的表达能力和可解释性。

不过，GAT 模型也存在一定的局限性。首先，Attention 机制需要对每一条边都计算注意力系数，因此 GAT 模型的计算复杂度明显高于 GCN 模型。当图规模达到千万甚至上亿节点时，Attention 机制的计算容易成为训练的瓶颈。其次，多头 Attention 机制进一步增加了模型的参数量和显存的占用，使得 GAT 召回模型在超大规模工业推荐系统中的部署成本比较高。此外，与 GCN 模型类似，当网络层数不断加深时，GAT 模型同样会受到 Over-smoothing（过平滑）以及 Over-squashing（过压缩）等问题的影响，因此工业界通常也仅采用 2～3 层图注意力网络，很少堆叠更深的网络结构。

近年来，为了兼顾表达能力与计算效率，又相继提出了 GATv2 [4]、TransformerConv [46]、HGT（Heterogeneous Graph Transformer）[24] 等改进模型，在异构图推荐、多关系图建模以及超大规模图学习任务中取得了更好的效果。不过，在工业级推荐系统中，由于 GCN 模型及其简化版本（如 LightGCN）的训练效率更高，因此 GAT 模型更多应用于对模型表达能力要求较高的场景，而不是大规模在线召回系统的主流方案。

10.2.2.3 GraphSAGE 召回

GraphSAGE（Graph Sample and Aggregation）召回最早由 Hamilton 等人在 2017 年提出，对应 NeurIPS 论文《Inductive Representation Learning on Large Graphs》[18]。GraphSAGE 模型的核心思想如下：

定义 10.9

GraphSAGE 模型在 GCN 的基础上引入邻居采样（Neighbor Sampling）和归纳学习（Inductive Learning）机制，通过学习一个通用的邻居聚合函数（Aggregator），而不是直接学习每个节点对应的 Embedding 参数，从而能够在动态图以及新增节点场景下快速生成节点表示，因此成为工业界图神经网络的重要基础模型。♣

前面几节介绍的 GCN 和 GAT 模型本质上都属于 Transductive Learning（传导学习）方法，即模型训练完成之后，仅能够为训练图中的节点生成 Embedding。如果图中新增节点或者新增边，则通常需要重新训练整个图模型。而 GraphSAGE 模型则不同，它学习的是一种节点表示生成函数（Embedding Generator），因此对于训练过程中未出现的新节点，只需要获取其邻居信息，即可直接计算对应的 Embedding 表示，而无需重新训练整个模型，因此具有更好的泛化能力和工程可扩展性。

与此同时，GCN 模型每一层都需要对整张图进行消息传播，当节点数量达到千万甚至上亿规模时，训练复杂度和显存开销都会迅速增长。GraphSAGE 模型进一步提出 Neighbor Sampling 策略，每层仅随机采样固定数量的邻居参与聚合过程，从而将模型每层的计算复杂度由依赖整个邻居集合降低到只依赖固定的采样数量，使图神经网络首次能够应用于工业级大规模图学习的场景中。

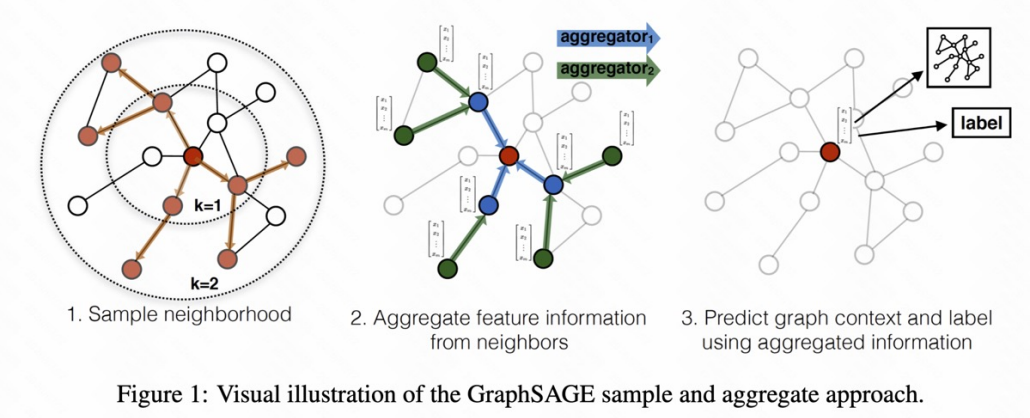

Figure 1: Visual illustration of the GraphSAGE sample and aggregate approach.

图 10.14: GraphSAGE 召回模型网络结构示意图。

GraphSAGE 召回模型的网络结构如图10.14所示。整个 GraphSAGE 模型主要包括邻居采样（Neighbor Sampling）、邻居聚合（Neighbor Aggregation）以及节点表示更新（Embedding Update）三个阶段。首先，对于中心节点 v，GraphSAGE 并不会聚合全部邻居，而是随机采样固定数量的邻居：

$$\mathcal{S}(v) \subseteq \mathcal{N}(v), \qquad |\mathcal{S}(v)| = S, \tag{10.119}$$

其中 $\mathcal{N}(v)$ 表示节点 v 的全部邻居，S 表示采样数量。通常每层都会设置不同的采样数，例如第一层采样 25 个邻居，第二层采样 10 个邻居，从而有效控制计算复杂度。随后，对采样得到的邻居进行聚合：

$$\mathbf{h}_{\mathcal{N}(v)}^{(l)} = \text{AGGREGATE}^{(l)}\left(\{\mathbf{h}_u^{(l-1)}, u \in \mathcal{S}(v)\}\right) \tag{10.120}$$

其中，AGGREGATE$(\cdot)$ 表示聚合函数。最后，将邻居聚合结果与中心节点自身表示进行拼接，并更新新的 Embedding：

$$\mathbf{h}_v^{(l)} = \sigma\left(\mathbf{W}^{(l)} \cdot \text{CONCAT}(\mathbf{h}_v^{(l-1)}, \mathbf{h}_{\mathcal{N}(v)}^{(l)})\right) \tag{10.121}$$

随后进行 L2 归一化：

$$\mathbf{h}_v^{(l)} = \frac{\mathbf{h}_v^{(l)}}{\|\mathbf{h}_v^{(l)}\|_2} \tag{10.122}$$

经过 K 层传播之后，最终节点表示为

$$\mathbf{z}_v = \mathbf{h}_v^{(K)} \tag{10.123}$$

从上述介绍可以发现，GraphSAGE 与 GCN 模型都会保留中心节点自身的信息，区别在于两者的信息融合方式不同。GCN 模型首先利用归一化邻接矩阵对中心节点及其邻居节点进行**统一的加权聚合**，而 GraphSAGE 模型则首先利用 Aggregator 分别聚合邻居节点的信息，再与中心节点自身的表示进行 Concat，最后通过线性变换得到新的节点表示。因此，GraphSAGE 模型能够**分别建模节点自身特征与邻域特征之间的关系**，具有更强的表达能力及灵活性。此外，GraphSAGE 模型引入了 Neighbor Sampling 机制，只对固定数量的邻居进行聚合，使其能够扩展到大规模动态图场景，这也是其相比 GCN 最重要的工程优势。

GraphSAGE 模型的前向传播流程如代码10.9所示。

代码 10.9: GraphSAGE 前向传播示例代码。

```
def graphsage_forward(graph, features, num_layers, sample_size, aggregators, weights):
    h = features
    for layer in range(num_layers):
        next_h = {}
        for node in graph.nodes():
            # Neighbor Sampling
            neighbors = sample_neighbors(graph, node, sample_size[layer])
            # Neighbor Aggregation
            neigh_emb = aggregators[layer]([h[n] for n in neighbors])
            # Concat + Linear
            x = concat(h[node], neigh_emb)
            x = relu(weights[layer] @ x)
            # L2 Normalize
            next_h[node] = l2_normalize(x)
        h = next_h
    return h
```

GraphSAGE 最大的特点之一是聚合函数（Aggregator）可以灵活设计，不同的聚合方式对应不同的信息融合能力。论文中主要提出了如下几种 Aggregator。

- **Mean Aggregator**

$$\text{AGGREGATE} = \frac{1}{|\mathcal{S}(v)|} \sum_{u \in \mathcal{S}(v)} \mathbf{h}_u \tag{10.124}$$

这是最简单也是工业界应用最广泛的方法。

- **Max Pooling Aggregator** 首先对邻居 Embedding 进行 MLP 映射：

$$\mathbf{m}_u = \sigma(\mathbf{W}\mathbf{h}_u + \mathbf{b}) \tag{10.125}$$

随后进行逐维 Max Pooling：

$$\mathbf{h}_{\mathcal{N}(v)} = \max_{u \in \mathcal{S}(v)} \mathbf{m}_u \tag{10.126}$$

相比 Mean 能够学习更加复杂的邻居表示。

- **LSTM Aggregator** 将邻居 Embedding 输入 LSTM：

$$\mathbf{h}_{\mathcal{N}(v)} = LSTM(\mathbf{h}_{u_1}, \cdots, \mathbf{h}_{u_m}) \tag{10.127}$$

由于邻居本身没有固定顺序，因此通常会随机打乱邻居顺序，多次训练减弱顺序带来的影响。

在原始论文中，GraphSAGE 模型采用无监督方式进行训练。首先利用 Random Walk 在图结构上生成节点访问序列，然后借鉴 Skip-Gram 的训练思想，将窗口内共同出现的节点作为正样本，而从全图随机采样得到负样本。假设 (u, v) 表示一个正样本节点对，则 GraphSAGE 模型的无监督优化目标可以表示为

$$\mathcal{L} = -\sum_{(u,v)} \left[\log \sigma\left(\mathbf{z}_u^T \mathbf{z}_v\right) + Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \left[\log \sigma\left(-\mathbf{z}_u^T \mathbf{z}_{v_n}\right)\right]\right] \tag{10.128}$$

其中，(u, v) 表示由 Random Walk 生成的正样本节点对，v_n 表示从负采样分布 $P_n(v)$ 中采样得到的负样本节点，Q 表示每个正样本对应的负采样数量，$\mathbf{z}_u$ 和 $\mathbf{z}_v$ 分别表示节点 u 和节点 v 最终学习得到的 Embedding 表示。

需要注意的是，上式中的负样本项采用的是数学期望形式。在实际训练过程中，该期望通常利用蒙特卡洛采样（Monte Carlo Sampling）进行近似，即从分布 $P_n(v)$ 中独立采样 Q 个负样本：

$$v_1^-, v_2^-, \cdots, v_Q^- \sim P_n(v) \tag{10.129}$$

则有

$$Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \left[\log \sigma \left(-\mathbf{z}_u^T \mathbf{z}_{v_n} \right) \right] \approx Q \sum_{q=1}^{Q} \frac{1}{Q} \log \sigma \left(-\mathbf{z}_u^T \mathbf{z}_{v_q^-} \right) = \sum_{q=1}^{Q} \log \sigma \left(-\mathbf{z}_u^T \mathbf{z}_{v_q^-} \right) \tag{10.130}$$

因此，在实际工程实现中，GraphSAGE 模型更常采用如下的损失函数来进行优化：

$$\mathcal{L} = -\sum_{(u,v)} \left[\log \sigma \left(\mathbf{z}_u^T \mathbf{z}_v \right) + \sum_{q=1}^{Q} \log \sigma \left(-\mathbf{z}_u^T \mathbf{z}_{v_q^-} \right) \right] \tag{10.131}$$

相比于数学期望形式，上述损失函数更加符合工业界的实际训练流程，也是 DeepWalk、Node2Vec 等图表示学习模型中最常见的 Skip-Gram 负采样优化目标。

下面给出 GraphSAGE 模型的无监督训练过程，如代码10.10所示。

代码 10.10: GraphSAGE 无监督训练示例代码。

```
for center, positive in positive_pairs:
    z_center = encoder(center)
    z_pos = encoder(positive)
    pos_loss = -log_sigmoid(dot(z_center, z_pos))

    neg_loss = 0.0
    for _ in range(num_negative):
        neg = negative_sampling()
        z_neg = encoder(neg)
        neg_loss += -log_sigmoid(-dot(z_center, z_neg))
    loss = pos_loss + neg_loss
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

GraphSAGE 召回的整体工程流程与 GCN 召回基本一致，通常首先根据用户点击、观看、购买等行为构建 User-Item 图或 Item 图，然后通过多层 Neighbor Sampling 与 Neighbor Aggregation 学习节点 Embedding，并建立 ANN 向量索引部署到在线召回系统中。

由于 GraphSAGE 模型采用归纳学习方式，因此在线阶段 GraphSAGE 模型不仅能够查询已有节点对应的 Embedding，对于新出现的物品，只要能够获取其邻居信息，也能够快速计算对应的 Embedding，而无需重新训练整个模型。因此，GraphSAGE 模型相比 GCN 模型更加适用于动态图、增量图以及超大规模推荐图场景。目前，GraphSAGE 模型也是 PinSage、HybridGNN 等工业级图召回模型的重要基础组件。

10.2.2.4 LightGCN 召回

LightGCN 召回是何向南等人在 2020 年提出的一种轻量级图卷积网络模型，发表于 SIGIR 会议的论文《LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation》[20]。LightGCN 模型从推荐系统协同过滤的特点出发，对传统 GCN 进行了针对性的简化。作者认为，在推荐系统中，节点通常并不存在丰富的语义特征，模型学习的对象主要是用户和物品的 ID Embedding，因此 GCN 模型中的**特征变换（Feature Transformation）**和**非线性激活**不仅难以带来收益，反而会增加模型复杂度并影响 Embedding 传播效果。因此，LightGCN 仅保留图卷积中的邻居聚合（Neighborhood Aggregation）操作，从而在降低模型复杂度的同时获得了更好的推荐性能。

LightGCN 模型的核心思想如下：

定义 10.10

LightGCN 通过去除 GCN 中的特征变换、非线性激活以及显式自连接，仅保留图结构上的邻居传播操作，使 Embedding 能够在用户-物品图上进行更加充分的信息传播，从而学习高质量的用户与物品表示。 ♣

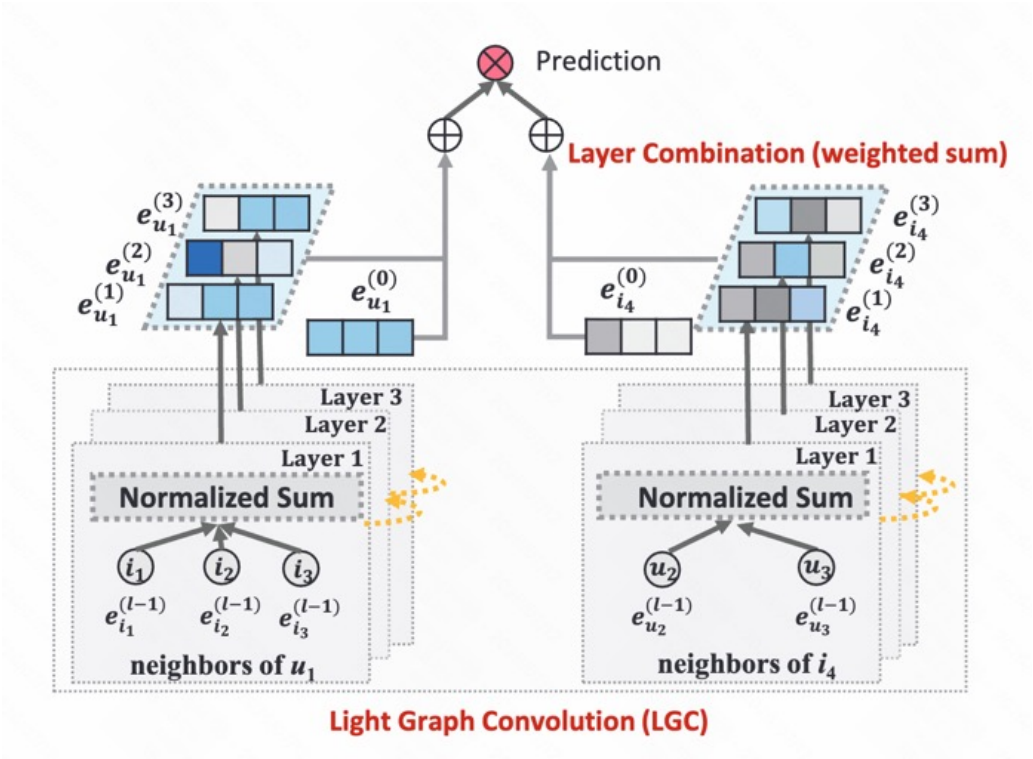

图 10.15: LightGCN 召回模型网络结构示意图。

LightGCN 模型的网络结构如图10.15所示。相比传统 GCN 模型，LightGCN 模型的网络更加简洁，整个模型几乎没有额外的神经网络参数，唯一需要学习的参数仅为用户和物品在第 0 层初始化得到的 Embedding，因此训练效率和在线部署效率都明显高于传统的 GCN 模型。LightGCN 模型的核心在于 Light Graph Convolution（LGC）传播操作。对于用户节点和物品节点，其 Embedding 更新分别为

$$\begin{aligned}\mathbf{h}_i^{(l+1)} &= \sum_{j\in\mathcal{N}(i)} \frac{1}{\sqrt{|\mathcal{N}(i)|}\sqrt{|\mathcal{N}(j)|}}\mathbf{h}_j^{(l)}, \\ \mathbf{h}_j^{(l+1)} &= \sum_{i\in\mathcal{N}(j)} \frac{1}{\sqrt{|\mathcal{N}(j)|}\sqrt{|\mathcal{N}(i)|}}\mathbf{h}_i^{(l)}\end{aligned} \tag{10.132}$$

其中，$\mathcal{N}(i)$ 表示节点 i 的一阶邻居集合，归一化系数采用与 GCN 模型相同的对称归一化形式，用于避免随着图卷积层数增加而导致 Embedding 尺度不断扩大。

相比起 GCN 模型，LightGCN 模型最大的区别在于传播过程中去除了线性变换矩阵 $\mathbf{W}$ 以及非线性激活函数，仅保留最基本的邻居信息传播。LightGCN 原始论文指出，在协同过滤的场景下，Embedding 传播本身已经能够充分挖掘出高阶的协同关系，而额外的参数变换反而容易带来过拟合问题。因此，更简单的传播方式反而能够取得更好的效果。

此外，LightGCN 模型没有像 GCN 模型一样显式加入自循环（Self-loop）的结构，而是采用多层 Embedding 融合（Layer Combination）的方式来保留节点自身的信息。假设 LightGCN 模型具有 K 层，则最终用户和物品 Embedding 表示为

$$\mathbf{e}_u = \sum_{k=0}^{K} \alpha_k \mathbf{e}_u^{(k)}, \qquad \mathbf{e}_i = \sum_{k=0}^{K} \alpha_k \mathbf{e}_i^{(k)} \tag{10.133}$$

其中，α_k 表示第 k 层 Embedding 对应的融合权重。实际工程中通常直接采用均匀加权，即

$$\alpha_k = \frac{1}{K+1} \tag{10.134}$$

论文指出，这种多层融合具有三个优点：一方面能够保留不同传播深度下的图结构信息；另一方面可以缓解深层 GCN 模型中的 Over-smoothing 问题；同时，多层 Embedding 加权求和实际上能够起到与 Self-loop 类似的作用，因此无需额外引入自连接。

LightGCN 模型的训练目标通常采用 BPR Loss、Binary Cross Entropy Loss 或者 Contrastive Loss 等目标函数，与 GCN 召回基本一致，因此这里不再展开介绍。LightGCN 原始论文中采用 BPR Loss 进行训练，其目标函数为

$$\mathcal{L} = -\sum_{(u,i,j)\in\mathcal{D}} \log\sigma\left(\hat{y}_{ui} - \hat{y}_{uj}\right) \tag{10.135}$$

其中，(u,i,j) 表示用户 u 对应的正样本物品 i 和负样本物品 j，预测得分 $\hat{y}_{ui}$ 定义为

$$\hat{y}_{ui} = \mathbf{e}_u^T \mathbf{e}_i \tag{10.136}$$

即最终用户 Embedding 与物品 Embedding 的内积。

在 LightGCN 召回的工程实现中，通常首先根据用户点击、观看、购买等行为构建 User-Item 二部图，然后通过多层 Light Graph Convolution 离线学习用户和物品的 Embedding 向量，并建立 ANN 向量索引部署到在线召回系统中。在线阶段，根据用户最近行为生成用户兴趣 Embedding 向量，再通过 Faiss、ScaNN、HNSW 等 ANN 检索系统快速召回 Top-K 候选物品，从而实现整个的 U2I 召回过程。由于 LightGCN 召回的传播过程简单、参数量少，同时能够有效建模高阶的协同关系，因此，LightGCN 目前已经成为工业界图召回模型中应用最广泛的方法之一。

10.2.2.5 PinSage 召回

PinSage 召回是 Pinterest 公司的研究团队于 2018 年提出的一种面向超大规模推荐系统的图神经网络模型，发表于 KDD 会议的论文《Graph Convolutional Neural Networks for Web-Scale Recommender Systems》[60]。PinSage 建立在 GraphSAGE 模型的基础之上，针对工业级推荐系统中图规模巨大、邻居数量庞大以及训练效率低等问题进行了系统性的工程优化，使图神经网络首次成功应用于拥有数十亿节点和数百亿边的大规模推荐系统。

GraphSAGE 虽然提出了归纳式（Inductive）的图表示学习框架，能够通过邻居采样和聚合函数学习节点 Embedding，但其默认采用 Uniform Sampling 随机采样邻居，并未充分考虑不同邻居的重要程度。同时，对于工业推荐系统中的超大规模图，随机采样容易遗漏真正重要的兴趣关联节点，导致模型效果受到限制。此外，GraphSAGE 对于训练过程中的邻居采样、Mini-batch 构建以及 GPU 训练效率等工程问题也缺乏针对性的优化。

针对上述问题，PinSage 提出了一整套适用于工业推荐系统的图表示学习方案，其核心思想可以概括为：首先利用 Random Walk with Restart（RWR）在图结构上搜索与目标节点最相关的邻居；随后根据随机游走过程中节点的访问次数（Visit Count）估计邻居的重要程度（Importance Weight）；最后利用 Importance Pooling 对不同邻居进行加权聚合，从而学习得到更加准确的节点 Embedding 表示。相比 GraphSAGE 简单地随机采样邻居，PinSage 能够更加关注真正重要的兴趣传播路径，因此具有更好的推荐效果。

PinSage 模型的核心思想如下：

定义 10.11

PinSage 通过 Random Walk with Restart 策略来寻找目标节点的重要邻居，并根据随机游走访问次数构建 Importance Weight，在此基础上结合 GraphSAGE 式的邻居聚合方式来学习节点 Embedding，从而实现面向超大规模推荐系统的高效图表示学习。

PinSage 模型整体网络结构如图10.16所示。从图10.16可以看出，PinSage 整体仍然遵循 GraphSAGE 模型中“邻居采样——邻居聚合——Embedding 更新”的基本框架，但在各个阶段均进行了针对推荐系统的大规模优化。整个模型主要包括以下几个步骤：

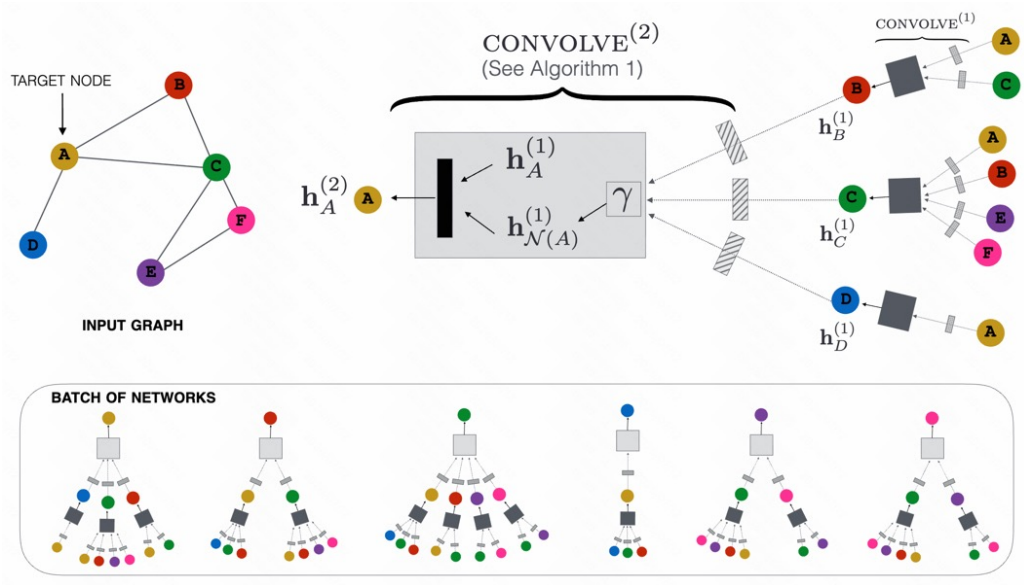

图 10.16: PinSage 召回模型网络结构示意图。

1. 根据用户点击、收藏、购买等行为构建 User-Item 二部图（User-Item Bipartite Graph）；
2. 对目标节点执行 Random Walk with Restart 策略，统计随机游走过程中各个邻居节点的访问次数，并筛选 Top-T 个最重要的邻居；
3. 根据访问次数构建 Importance Weight，对不同的邻居赋予不同的重要性权重；
4. 利用 Importance Pooling 聚合邻居节点信息，并结合 GraphSAGE 式的 Graph Convolution 操作更新节点的 Embedding；
5. 经过多层图卷积传播后，得到最终的用户 Embedding 和物品 Embedding，用于后续 ANN 向量召回。

表 10.2: GraphSAGE 与 PinSage 模型的主要区别

比较维度	GraphSAGE	PinSage
邻居采样	Uniform Sampling 随机采样	Random Walk with Restart 重要邻居采样
邻居权重	所有邻居权重相同	根据 Visit Count 计算 Importance Weight
聚合方式	Mean、Pooling、LSTM Aggregator	Importance Pooling 加权聚合
负样本	随机负采样	Hard Negative Sampling
训练方式	普通 Mini-batch	Mini-batch 子图构建与流水线训练
适用规模	中小规模图	十亿级节点工业推荐图

相比起 GraphSAGE 模型，PinSage 模型最大的贡献并不在于提出新的图卷积结构，而是在 GraphSAGE 模型的基础上完成了一系列工程优化，使其能够真正部署到工业级推荐系统中。二者之间的主要区别如表10.2所示。可以发现，PinSage 并没有重新设计一种新的图神经网络，而是在 GraphSAGE 框架基础上进行了全面的工程优化。其中，Random Walk 采样、Importance Pooling 以及 Mini-batch 子图构建是 PinSage 最核心的三个创新点。这些优化不仅提升了模型对于重要邻居的建模能力，而且显著降低了超大规模图上的训练成本，使 PinSage 成为工业界图召回模型的重要代表方法。下面我们将分别介绍这些关键模块的具体实现原理。

10.2.2.5.1 Importance-based Neighbor Sampling GraphSAGE 模型通常采用 Uniform Sampling 的方法，从节点的一阶邻居中随机采样固定数量的邻居进行聚合。这种方法虽然能够有效控制计算复杂度，但在工业级推荐系统中存在明显不足。一方面，推荐系统中的图结构通常具有高度稀疏且长尾分布的特点，比如一个热门物品往往拥有数百万甚至上亿个邻居节点，通过随机采样的方式很容易遗漏真正重要的兴趣关联；另一方面，不同的邻居对于目标节点的重要程度并不相同，简单的均匀采样无法充分利用图中的结构信息。

针对这一问题，PinSage 模型提出了基于 Random Walk with Restart（RWR）的重要邻居采样方法。对于目标

节点 v，PinSage 首先从节点 v 出发执行多次随机游走。每一步随机游走都有一定概率继续沿着图边访问下一跳邻居，也有一定概率重新返回起始节点 v 重新开始随机游走。设重启概率为 r，则一次随机游走过程可以表示为

$$P(v_{t+1}=u)=\begin{cases} r, & u=v_0, \\ (1-r)P(u|v_t), & u\in\mathcal{N}(v_t) \end{cases} \tag{10.137}$$

其中，v_0 表示随机游走的起始节点，$\mathcal{N}(v_t)$ 表示当前节点 v_t 的一阶邻居集合。

相比起普通的 Random Walk，Random Walk with Restart 能够始终围绕目标节点附近进行局部探索，因此既能够捕获高阶邻居信息，又不会由于随机游走的距离过远而偏离目标节点附近重要的兴趣区域。在完成多次随机游走之后，PinSage 模型进一步统计每个节点在所有随机游走路径中的访问次数（Visit Count）。如果访问次数越高，则说明该节点越容易从目标节点到达，其结构相关性也越强。因此，访问次数可以作为邻居重要性的度量指标。

假设节点 u 在所有随机游走过程中共访问 c_u 次，则其 Importance Weight 定义为

$$\alpha_u=\frac{c_u}{\sum_{v\in\mathcal{N}(v)}c_v} \tag{10.138}$$

随后按照访问次数从高到低排序，仅保留 Top-T 个访问次数最高的邻居作为后续 Graph Convolution 的输入，而访问次数则作为 Importance Weight 参与邻居聚合过程。相比起 GraphSAGE 模型随机选择固定数量邻居，PinSage 模型更加关注真正重要的兴趣传播路径，因此采样得到的邻居集合具有更高的相关性，也能够显著提升 Embedding 的质量。

10.2.2.5.2 Importance Pooling 在完成重要邻居采样之后，PinSage 模型进一步提出了 Importance Pooling 机制来聚合邻居信息。传统 GraphSAGE 模型通常使用 Mean Pooling、Max Pooling 或者 LSTM Aggregator 等方式，将所有邻居视为同等重要进行聚合。而 PinSage 模型则认为：不同邻居对于中心节点的贡献存在明显差异，因此需要根据 Importance Weight 进行加权聚合。

假设节点 v 对应的重要邻居集合为 $\mathcal{N}(v)$，邻居节点的 Embedding 表示为 $\mathbf{h}_u$，Importance Weight 表示为 α_u。这里首先利用一个共享的 MLP 网络来对邻居的 Embedding 进行特征变换，即：

$$\mathbf{m}_u=\mathrm{ReLU}(\mathbf{Q}\mathbf{h}_u+\mathbf{q}), \tag{10.139}$$

其中，$\mathbf{Q}$ 表示可学习参数矩阵，$\mathbf{q}$ 表示偏置项。随后我们根据 Importance Weight 进行加权 Pooling：

$$\mathbf{n}_v=\gamma\left(\{\mathbf{m}_u,u\in\mathcal{N}(v)\},\alpha\right) \tag{10.140}$$

其中，$\gamma(\cdot)$ 表示 Importance Pooling 操作，其本质可以理解为加权平均，即：

$$\mathbf{n}_v=\sum_{u\in\mathcal{N}(v)}\alpha_u\mathbf{m}_u \tag{10.141}$$

可以看到，当所有 Importance Weight 相等时，上述公式便退化为 GraphSAGE 模型中的 Mean Aggregation；而 PinSage 模型通过 Importance Weight 赋予重要邻居更大的权重，因此能够更加准确地建模节点之间真实的关联关系。

随后，PinSage 模型将中心节点自身 Embedding 与聚合后的邻居 Embedding 进行拼接：

$$\mathbf{z}_v^{new}=\mathrm{ReLU}\left(\mathbf{W}\left[\mathbf{z}_v\|\mathbf{n}_v\right]+\mathbf{w}\right) \tag{10.142}$$

其中，$\mathbf{W}$ 表示线性变换矩阵，$\mathbf{w}$ 表示偏置项。最后，为了防止 Embedding 的范数不断增大，同时保证 Embedding 能够直接用于向量检索，PinSage 模型会对输出 Embedding 进行 ℓ_2 归一化：

$$\mathbf{z}_v^{new}=\frac{\mathbf{z}_v^{new}}{\|\mathbf{z}_v^{new}\|_2} \tag{10.143}$$

上述过程对应 PinSage 论文中的 Convolution 模块，其整体流程如算法10.11所示。

10.2.2.5.3 Convolution 模块实现 根据上述 Importance Sampling 和 Importance Pooling 机制，PinSage 每一层图卷积对应论文中的 Convolution 模块，其整体流程如代码10.11所示。

代码 10.11: PinSage Convolution 模块示例代码。

```
class PinSageConv(nn.Module):
    def __init__(self, dim):
        super().__init__()
        # Neighbor projection
        self.neighbor_fc = nn.Linear(dim, dim)
        # Node update
        self.update_fc = nn.Linear(dim * 2, dim)

    def forward(self, center_emb, neighbor_embs, importance_weight):
        # Shared MLP
        h = F.relu(self.neighbor_fc(neighbor_embs))
        # Importance Pooling
        weight = importance_weight
        weight = weight / weight.sum()
        pooled = (weight.unsqueeze(-1) * h).sum(dim=0)

        # Concat
        out = torch.cat([center_emb, pooled], dim=-1)
        # Update
        out = F.relu(self.update_fc(out))
        # L2 Normalization
        out = F.normalize(out, p=2, dim=-1)
        return out
```

从上述代码可以看出，PinSage 每一层图卷积主要包括四个步骤：首先利用共享 MLP 提取邻居 Embedding；随后根据 Random Walk 得到的重要性权重进行 Importance Pooling；然后将中心节点 Embedding 与邻居聚合结果进行拼接，并通过线性映射生成新的节点表示；最后进行 ℓ_2 归一化，使 Embedding 能够直接用于后续向量检索。

10.2.2.5.4 Mini-batch 子图训练 对于工业级推荐系统而言，整个推荐图通常包含数十亿节点以及数百亿条边。如果每次训练都对完整图进行传播，不仅 GPU 显存无法容纳，而且计算复杂度也无法接受。因此，PinSage 采用 Mini-batch Subgraph 的方法进行训练。对于当前 Batch 中的节点，仅构建其 K 跳邻域所形成的局部子图，并仅在该子图上执行图卷积，从而大幅降低训练成本。PinSage 模型的整个 Embedding 生成过程如代码10.12所示。

代码 10.12: PinSage Mini-batch 训练流程示例代码。

```
def generate_embeddings(batch_nodes):
    # Multi-hop sampling
    sampled_graph = sample_neighbors(batch_nodes)
    hidden = initialize_features(sampled_graph)

    # Multi-layer propagation
    for layer in range(num_layers):
        next_hidden = {}
        for node in sampled_graph[layer]:
            neighbors = get_neighbors(node)
```

```
            weights = get_importance_weight(node)
            next_hidden[node] = pinsage_conv(hidden[node], hidden[neighbors], weights)
        hidden = next_hidden

    # Projection Layer
    embedding = projection(hidden[batch_nodes])
    return embedding
```

可以看出，PinSage 模型在训练过程中仅需对当前 Mini-batch 所涉及的局部子图进行消息传播，而无需访问整个推荐图，因此 GPU 计算复杂度与 Batch 大小近似呈线性增长，能够很好地支持超大规模图结构上的高效训练，具有良好的可扩展性。

10.2.2.5.5 PinSage 模型训练 PinSage 模型的原始论文中采用 Margin Ranking Loss 进行训练，而不是 GraphSAGE 模型中基于 Skip-Gram 的无监督目标。在训练过程中，对于每个查询物品 q，分别构造一个正样本物品 i 以及一个负样本物品 j，则 PinSage 模型的优化目标为

$$\mathcal{L} = \sum_{(q,i,j)} \max\left(0, \mathbf{z}_q^T \mathbf{z}_j - \mathbf{z}_q^T \mathbf{z}_i + \Delta\right) \tag{10.144}$$

其中，Δ 表示 Margin 超参数。

与随机负采样不同，PinSage 更加关注 Hard Negative 样本的构造。论文指出，只采用随机负样本很容易导致整个训练过于简单，因此 PinSage 模型会优先选择与查询物品具有一定相似性、但用户最终没有点击或保存的物品作为 Hard Negative 样本，从而提高模型对于相似物品之间细粒度差异的建模及区分能力。在 PinSage 模型训练完成之后，只保留最终学习得到的物品 Embedding，并建立 ANN 索引部署到在线召回系统中。

10.2.2.5.6 PinSage 工程优化 除了模型本身之外，PinSage 召回最大的贡献还在于提出了一套完整的工业级图训练框架，使得 GNN 首次成功部署到 Pinterest 超大规模的推荐系统中。首先，在邻居采样阶段，PinSage 模型采用 CPU 执行 Random Walk Sampling，而 GPU 只负责图卷积计算，从而避免 GPU 的资源浪费。其次，PinSage 模型采用了 Producer-Consumer 的流水线架构，将 Random Walk 采样、Mini-batch 构建以及 GPU 训练这三个阶段进行解耦。CPU 会持续构建训练子图并生成 Mini-batch，而 GPU 则会不断消费这些 Batch 进行前向传播与反向更新，使得数据采样与模型训练能够并行执行，从而显著提升整体训练吞吐量。

此外，PinSage 模型还采用动态图缓存（Graph Cache）、异步数据加载（Asynchronous Data Loading）以及 Mini-batch 子图复用等多种工程优化策略，进一步降低了大规模图训练过程中的 I/O 开销和通信成本，使整个系统能够稳定运行于包含数十亿节点的大规模推荐图之上。

总体来看，PinSage 召回并没有重新设计一种全新的图神经网络，而是在 GraphSAGE 模型框架的基础上进行了大量的工程优化，包括 Importance Sampling、Importance Pooling、Mini-batch 子图训练以及流水线并行等关键技术，使得图神经网络真正具备了工业级推荐系统中的可扩展性。因此，PinSage 模型也被认为是工业界图召回模型发展的一个重要里程碑，并为后续的 LightGCN、HybridGNN 等方法奠定了重要基础。

10.2.2.6 HybridGNN 召回

HybridGNN 召回是快手科技于 2022 年提出的一种面向多重关系异构图（Multiplex Heterogeneous Graph）的图神经网络模型，详细内容见论文《HybridGNN: Learning Hybrid Representation for Recommendation in Multiplex Heterogeneous Networks》[17]。相比起 GCN、GraphSAGE 以及 PinSage 等传统图神经网络主要针对单一关系图进行建模，HybridGNN 模型进一步考虑了推荐系统中用户反馈行为具有多关系、多语义的特点，例如点击（Click）、点赞（Like）、评论（Comment）、分享（Share）、关注（Follow）等不同的交互行为分别对应不同类型的关系边，因此能够更加充分地挖掘不同关系之间的协同信息。

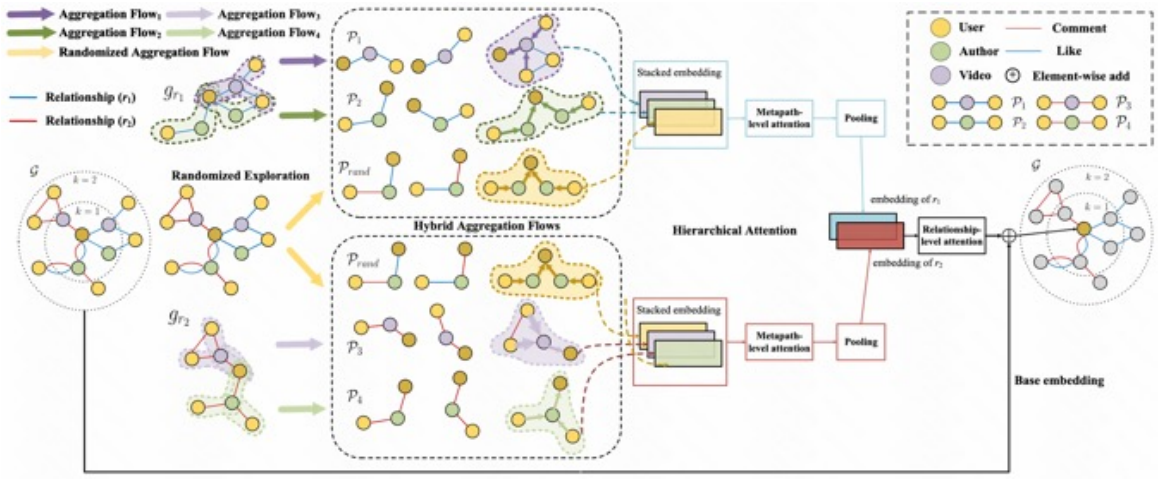

图 10.17: HybridGNN 召回模型网络结构示意图。

HybridGNN 召回模型的整体框架如图10.17所示，模型主要由三个核心模块组成：**跨关系随机探索（Randomized Inter-relationship Exploration）、混合聚合（Hybrid Aggregation Flows）以及层次化注意力机制（Hierarchical Attention）**。整个模型首先分别从预定义 Meta Path 和随机跨关系游走中提取节点表示，然后利用两级 Attention 机制完成不同 Meta Path 以及不同关系之间的信息融合，最终得到适用于推荐召回的节点 Embedding。

10.2.2.6.1 跨关系随机探索 传统异构图模型通常依赖人工设计的 Meta Path 进行消息传播，例如

$$\text{User} \to \text{Video} \to \text{User} \to \text{Video} \to \cdots$$

虽然能够充分利用预定义路径中的语义信息，但 Meta Path 通常依赖人工经验设计，只能建模有限的关系模式，难以发现不同关系之间潜在的交互规律。

为此，HybridGNN 模型提出了跨关系随机探索（Randomized Inter-relationship Exploration）模块，通过一种两阶段采样（Two-phase Sampling）策略，在多关系异构图中自动探索隐藏的跨关系连接模式。具体而言，对于当前节点 v_t，HybridGNN 模型首先随机采样下一跳对应的关系类型 r_{t+1}，然后再从该关系对应的邻居集合中随机采样下一跳节点 v_{t+1}。因此，一次随机游走可以表示为

$$v_0 \xrightarrow{r_1} v_1 \xrightarrow{r_2} \cdots \xrightarrow{r_t} v_t \xrightarrow{r_{t+1}} v_{t+1} \to \cdots$$

整个转移过程可以表示为关系采样与节点采样组成的联合概率：

$$p(v_{t+1}, r_{t+1}|v_t) = p(r_{t+1}|v_t)\, p(v_{t+1}|r_{t+1}, v_t) \tag{10.145}$$

其中，关系类型首先按照均匀分布从当前节点所有存在邻居的关系中进行采样：

$$p(r_{t+1}|v_t) = \begin{cases} \dfrac{1}{|\{r \,|\, N_r(v_t) \neq \varnothing, \forall r \in \mathcal{R}\}|} & N_{r_{t+1}}(v_t) \neq \varnothing \\ 0 & \text{otherwise} \end{cases} \tag{10.146}$$

其中 $N_r(v_t)$ 表示节点 v_t 在关系 r 下的邻居集合。随后，在选定关系 r_{t+1} 之后，再从对应关系的邻居集合中均匀采样下一跳节点：

$$p(v_{t+1}|r_{t+1}, v_t) = \frac{1}{|N_{r_{t+1}}(v_t)|} \tag{10.147}$$

这里 $N_{r_{t+1}}(v_t)$ 表示节点 v_t 在关系 r_{t+1} 下的所有邻居节点。

相比传统仅沿固定 Meta Path 传播信息的方法，这种“先采样关系、再采样节点”的两阶段式随机探索机制能够突破人工设计 Meta Path 的限制，在不同关系之间自动发现潜在的高阶关联模式，从而为后续 Hybrid Aggregation 模块提供更加丰富的跨关系语义信息。

10.2.2.6.2 混合聚合 在完成跨关系的随机探索之后，HybridGNN 召回模型进一步提出了 Hybrid Aggregation（混合聚合）模块，来对不同来源的邻域信息进行统一的建模。与传统 GraphSAGE 模型只沿着单一路径去聚合邻居的方式不同，HybridGNN 模型同时利用了预定义的 Meta Path 与随机探索得到的跨关系邻域，共同构建起多条聚合流（Aggregation Flow），从而兼顾异构图中的显式的语义关系与隐藏的跨关系交互模式。

假设关系 r_l 对应的 Meta Path 集合为

$$PS_{r_l} = \{P_1, P_2, \cdots, P_n\} \tag{10.148}$$

对于节点 v_i，满足条件的 Meta Path 集合记为 $\rho(v_i) \cap PS_{r_l}$。对于每一条 Meta Path $P_z \in \rho(v_i) \cap PS_{r_l}$，模型首先根据 Meta Path 采样得到第 k 阶邻居 $N_{P_z}^k(v_i)$，随后采用 GraphSAGE 式聚合方式递归更新节点表示：

$$\mathbf{h}_{v_i|P_z}^{(k)} = AGG_{P_z}\left(\mathbf{h}_{v_i|P_z}^{(k-1)}, \left\{\mathbf{h}_{v_j|P_z}^{(k-1)} \,\middle|\, v_j \in N_{P_z}^{K-k+1}(v_i)\right\}\right) \tag{10.149}$$

其中 $AGG(\cdot)$ 表示邻域聚合函数，可以采用 GraphSAGE 模型提出的 Mean Pooling、LSTM Pooling 或 Pooling Aggregator 等方式。在 HybridGNN 原始论文中主要采用 Mean Aggregator 完成邻域聚合。

除了 Meta Path 分支之外，HybridGNN 模型还利用随机探索得到的跨关系邻域建立另一条 Aggregation Flow。其传播过程与 Meta Path 分支类似，只是邻域节点来自跨关系随机探索过程，而不是预定义的 Meta Path，即：

$$\mathbf{h}_{v_i|rand}^{(k)} = AGG_{rand}\left(\mathbf{h}_{v_i|rand}^{(k-1)}, \left\{\mathbf{h}_{v_j|rand}^{(k-1)} \,\middle|\, v_j \in N_{rand}^{K-k+1}(v_i)\right\}\right) \tag{10.150}$$

其中随机探索得到的所有路径共享同一套聚合参数，因此能够学习不同关系之间共同的潜在交互模式，无需人为设计对应的 Meta Path。

经过上述多个 Aggregation Flow 之后，HybridGNN 模型将所有 Meta Path 对应的节点表示与随机探索得到的节点表示进行拼接，形成统一的混合表示，即：

$$\mathbf{H}_{\rho(v_i)} = \text{Concat}\left(\mathbf{h}_{v_i|rand}^{(K)}, \mathbf{h}_{v_i|P_1}^{(K)}, \cdots, \mathbf{h}_{v_i|P_m}^{(K)}\right) \tag{10.151}$$

这里 $m = |\rho(v_i)|$ 表示节点 v_i 对应 Meta Path 的数量。

从这里我们可以看出，混合聚合过程并非只是依赖某一条 Meta Path 来进行信息传播，而是同时融合了 Meta Path 所提供的显式语义信息以及随机探索发现的隐式跨关系信息，从而构建出多条聚合流来共同学习异构图中节点的表示。这也是 HybridGNN 模型区别于 GraphSAGE、PinSage 等传统图召回模型的重要特点之一，为后续层次化 Attention 进一步融合不同 Meta Path 和不同关系的信息提供了基础。

10.2.2.6.3 层次化注意力机制 经过混合聚合模块之后，每个节点都已经获得了多个聚合流对应的 Embedding 表示，包括不同 Meta Path 对应的 Embedding 表示以及随机探索得到的 Embedding 表示。然而，这些聚合结果对于当前推荐任务的重要程度并不相同。例如，某些 Meta Path 能够更好地描述用户兴趣，而随机探索得到的路径则可能包含更加丰富的潜在关联关系。因此，HybridGNN 模型进一步提出了 Hierarchical Attention（层次化注意力机制），分别在 Meta Path 层和关系层进行两级注意力融合。

首先，在 Meta Path 级别，模型将上一阶段得到的混合表示 $\mathbf{H}_{\rho(v_i)}$ 作为 Self-Attention 的输入，即

$$\mathbf{Q} = \mathbf{H}_{\rho(v_i)}, \qquad \mathbf{K} = \mathbf{H}_{\rho(v_i)}, \qquad \mathbf{V} = \mathbf{H}_{\rho(v_i)} \tag{10.152}$$

并利用 Scaled Dot-Product Attention 计算不同 Aggregation Flow 之间的重要性：

$$\hat{\mathbf{H}}_{\rho(v_i)} = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{W}_Q(\mathbf{K}\mathbf{W}_K)^T}{\sqrt{d_k}}\right)\mathbf{V}\mathbf{W}_V \tag{10.153}$$

其中 $\mathbf{W}_Q$、$\mathbf{W}_K$ 和 $\mathbf{W}_V$ 均为可学习参数，d_k 表示 Attention 隐藏维度。经过 Self-Attention 之后，不同 Meta Path 以及随机探索得到的 Embedding 都会重新分配权重，得到重新加权后的表示 $\hat{\mathbf{h}}_{v_i|P_1}, \hat{\mathbf{h}}_{v_i|P_2}, \cdots, \hat{\mathbf{h}}_{v_i|rand}$。随后，模型采用 Mean Pooling 获得节点在当前关系 r_l 下的局部表示：

$$\hat{\mathbf{h}}_{v_i,r_l} = \frac{1}{|\rho(v_i)|+1}\left(\hat{\mathbf{h}}_{v_i|rand} + \sum_{P_j \in \rho(v_i) \cap PS_{r_l}} \hat{\mathbf{h}}_{v_i|P_j}\right) \tag{10.154}$$

在完成 Meta Path 级的融合之后，每一种关系（如点击、点赞、评论、分享等）都会对应一个局部 Embedding。为了进一步建模不同关系之间的重要程度，HybridGNN 模型继续设计了 Relationship-level Attention 机制。该机

制首先将所有关系对应的局部表示进行拼接，即：

$$\mathbf{U}_{v_i,R} = \text{Concat}\left(\hat{\mathbf{h}}_{v_i,r_1}, \hat{\mathbf{h}}_{v_i,r_2}, \cdots, \hat{\mathbf{h}}_{v_i,r_L}\right) \tag{10.155}$$

其中 L 表示关系类型的数量。随后，同样采用 Self-Attention 学习不同关系之间的重要程度：

$$\hat{\mathbf{U}}_{v_i,R} = \text{Softmax}\left(\frac{\mathbf{U}_{v_i,R}\mathbf{W}_Q(\mathbf{U}_{v_i,R}\mathbf{W}_K)^T}{\sqrt{d_k}}\right)\mathbf{U}_{v_i,R}\mathbf{W}_V. \tag{10.156}$$

最终，模型得到节点在各个关系下的关系特定表示

$$\hat{\mathbf{U}}_{v_i,R} = (\mathbf{e}_{v_i,r_1}, \mathbf{e}_{v_i,r_2}, \cdots, \mathbf{e}_{v_i,r_L}) \tag{10.157}$$

并将对应关系的局部 Embedding 与节点基础 Embedding 进行融合，得到最终节点表示：

$$\mathbf{e}^*_{v_i,r_l} = \mathbf{e}_{v_i} + \mathbf{W}_{r_l}\mathbf{e}_{v_i,r_l} \tag{10.158}$$

其中，$\mathbf{e}_{v_i}$ 表示节点的基础 Embedding，$\mathbf{e}_{v_i,r_l}$ 表示节点在关系 r_l 下学习得到的局部表示，$\mathbf{W}_{r_l}$ 为对应关系的可学习映射矩阵。

10.2.2.6.4 HybridGNN 模型训练 HybridGNN 模型仍然采用与 Skip-Gram 模型类似的自监督训练方式。HybridGNN 模型首先沿着 Meta Path 进行随机游走生成节点序列，再利用滑动窗口构造中心节点及上下文节点作为正样本，并采用异构负采样（Heterogeneous Negative Sampling）构造负样本进行训练。

对于中心节点 v_i 及其上下文节点 v_j，模型采用负采样后的 Skip-Gram 目标函数进行优化：

$$\mathcal{L} = -\log\sigma(\mathbf{c}_j^T\mathbf{e}^*_{v_i}) - \sum_{k=1}^{Q}\log\sigma\left(-\mathbf{c}_k^T\mathbf{e}^*_{v_i}\right) \tag{10.159}$$

其中，$\mathbf{e}^*_{v_i}$ 表示 HybridGNN 模型学习得到的节点 Embedding，$\mathbf{c}_j$ 表示上下文节点 Embedding，$\mathbf{c}_k$ 表示负采样节点 Embedding，Q 表示负采样数量。

10.2.2.6.5 HybridGNN 召回工程落地 在工业级推荐系统中，HybridGNN 模型主要应用于 U2I 召回场景。首先，根据用户的点击、点赞、评论、分享、关注等多种交互行为来构建多关系异构图（Multiplex Heterogeneous Graph），不同类型的行为对应图中的不同关系边，例如点击关系、点赞关系、评论关系以及分享关系等。随后，利用 HybridGNN 模型离线学习用户节点与物品节点的 Embedding 表示，并通过层次化 Attention 机制融合不同关系下模型学习得到的局部 Embedding 表示，最终得到统一的用户 Embedding 和物品 Embedding。

在 HybridGNN 模型训练完成之后，可以同时得到用户节点和物品节点对应的 Embedding 表示。其中，物品 Embedding 通常用于构建 ANN 向量索引，并部署到在线向量检索系统中；在线上阶段，当用户发起推荐请求时，只需要根据用户 ID 来查找对应的用户 Embedding，再利用 ANN 检索系统快速检索出 Top-K 的近邻物品，即可生成 HybridGNN 召回的候选集合。由于 HybridGNN 模型在 Embedding 学习过程中同时建模了点击、点赞、评论、分享等多种交互关系，因此模型学习得到的节点表示能够融合更加丰富的行为语义。相比起 GraphSAGE、PinSage 等主要依赖单一交互关系进行表示学习的方法，HybridGNN 模型能够更加全面地刻画用户兴趣以及物品之间的潜在关联，从而进一步提升召回结果的准确率、多样性以及长尾内容的覆盖能力。

需要注意的是，HybridGNN 召回通常采用离线训练、在线检索的部署方式，即图神经网络只在离线阶段完成 Embedding 的学习，而在线阶段则无需实时执行图卷积传播操作，只需通过 ANN 近邻检索即可，因此能够满足工业级推荐系统对于低延迟、高吞吐在线服务的要求。在实际工程中，HybridGNN 召回通常与协同过滤召回、Embedding 召回、序列召回等多种召回通路共同组成多路召回体系。各路召回相互补充，从不同角度挖掘用户兴趣，从而进一步提升整体召回的覆盖率、召回质量以及最终的推荐效果。

10.3 向量召回

向量召回（Embedding Retrieval）是目前工业级推荐系统中应用最广泛的一类召回方法，其核心思想如下：

定义 10.12

向量召回是指利用深度学习模型学习用户和物品的低维稠密表示（Embedding），并将二者映射到统一的向量空间中，再通过计算用户 Embedding 与物品 Embedding 之间的相似度完成候选物品召回的一类方法。♣

从广义上来说，凡是能够学习用户 Embedding 或者物品 Embedding，并基于向量相似度计算完成召回的方法，都可以归属于向量召回。例如，在协同过滤召回章节介绍的 Neural CF、Heterogeneous CF 等方法，本质上都是通过学习 Embedding 表示完成用户与物品之间的匹配；而在图召回章节介绍的 GCN、GraphSAGE、LightGCN、PinSage 以及 HybridGNN 等模型，同样也是利用图神经网络学习节点 Embedding，再结合 ANN 索引完成在线召回。因此，这些方法都属于向量召回的范畴。

不过，上述模型分别属于协同过滤召回和图召回这两类具有代表性的技术路线，因此我们已经在对应的章节中进行了详细的介绍。本章节我们将重点介绍除了协同过滤召回与图召回之外，工业界广泛应用的其他向量召回方法，主要包括 YouTubeDNN 召回以及双塔召回（Two-Tower Retrieval）这两类经典模型。这两类模型也是当前工业界 Embedding 召回的基础框架，后续大量的工作都是在此基础上不断演化而来的。

此外，近年来广泛应用于工业级推荐系统中的对比学习召回（Contrastive Retrieval）本质上也是建立在双塔结构之上的向量召回模型，只是在训练目标和样本构造方面引入了对比学习思想。因此，本书将在下一章序列召回中结合对比学习方法，对其进行更加详细的介绍。

10.3.1 YoutubeDNN 召回

YoutubeDNN 召回是 Google 在 2016 年提出的一种经典 Embedding 召回模型，最初应用于 YouTube 视频推荐场景，源自论文《Deep Neural Networks for YouTube Recommendations》[11]。该模型利用深度神经网络学习用户和物品的低维 Embedding 向量，并将二者映射到统一的向量空间中，再结合 ANN 近似最近邻检索实现大规模候选物品召回。相比传统协同过滤方法，YoutubeDNN 不仅能够更好地建模用户兴趣的非线性特征，还能够方便地融合用户画像、搜索历史、上下文等多种异构特征，因此成为工业界 Embedding 召回模型的重要基础。

10.3.1.1 YoutubeDNN 召回模型原理

YoutubeDNN 召回模型的整体结构如图10.18所示。模型采用典型的深度神经网络结构，输入包括多种用户侧的离散的特征。除了用户历史行为特征之外，YoutubeDNN 模型还充分利用了多种异构的特征。比如，用户搜索历史经过分词后映射为 Embedding 表示；用户所在地区、设备类型等类别特征同样采用 Embedding 表示；年龄、性别、登录状态等连续特征则经过归一化后直接输入网络。这种统一的 Embedding 建模方式，使得模型能够方便地融合不同来源的信息，进一步提升了用户兴趣的表示能力。

此外，YoutubeDNN 模型的论文中还提出了一个十分具有工程价值的 Example Age 特征。由于在线视频平台每天都会产生大量新内容，而训练数据通常来自历史日志，因此模型容易产生偏向历史热门视频的现象，而对新视频不太友好。为了缓解这一问题，YoutubeDNN 模型在训练阶段引入样本年龄（Example Age）作为输入特征，用于表示样本距离当前时间的间隔；在线推理阶段，则将该特征设置为接近当前时刻的数值，使模型能够更加准确地学习内容的新鲜度，从而提升新视频的曝光能力并缓解训练数据带来的时间偏置。

与 Word2Vec 模型中的 Continuous Bag-of-Words（CBOW）思想类似，YoutubeDNN 模型将用户历史观看的视频序列表示为多个视频 Embedding 的集合，并采用平均池化（Average Pooling）的方式生成固定长度的兴趣表示。此外，用户历史搜索的 query 信息同样会被分词后映射为 Embedding，再进行平均池化，并与其它用户侧特征共同作为模型输入。在 YouTubeDNN 模型中，所有离散特征首先通过 Embedding 层映射为低维稠密向量，再进行拼接，并输入多层全连接网络进行特征交互，最终得到用户 Embedding。与此同时，每个视频对应一个可学习的物品 Embedding，两者共同构成统一的 Embedding 空间。

最终，YoutubeDNN 召回模型将候选召回建模为一个超大规模多分类问题。对于用户 u，上下文 c，模型预

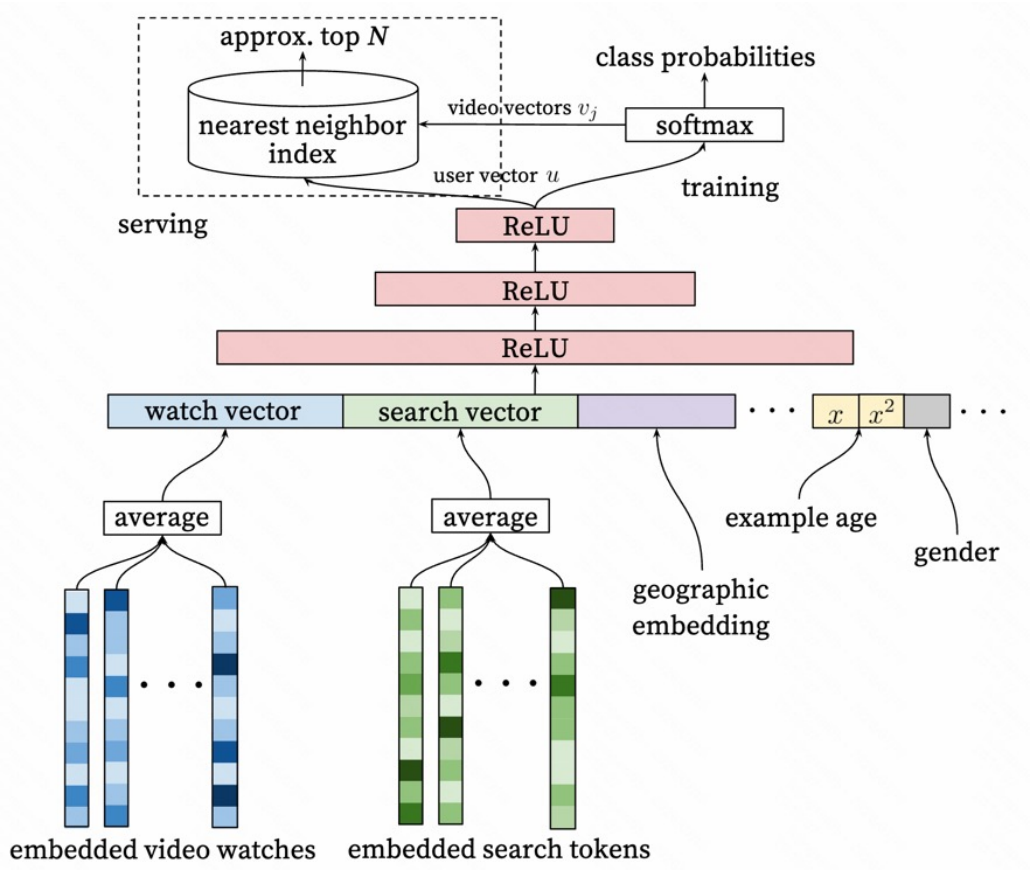

图 10.18: YoutubeDNN 召回模型结构。

测其观看视频 i 的概率可表示为

$$P(i|u,c) = \frac{\exp(\mathbf{e}_i^T \mathbf{e}_u)}{\sum_{j \in V} \exp(\mathbf{e}_j^T \mathbf{e}_u)} \tag{10.160}$$

其中，$\mathbf{e}_u$ 表示用户 Embedding，$\mathbf{e}_i$ 表示视频 Embedding，V 表示整个视频库。由于工业级推荐系统中往往包含数百万甚至上亿个候选物品，直接计算完整 Softmax 的代价是非常高的。因此，YoutubeDNN 模型采用候选采样（Candidate Sampling）的方式来近似训练 Softmax，只对正样本及采样得到的部分负样本计算交叉熵损失函数，从而显著降低训练复杂度。根据 YouTubeDNN 的论文我们可以发现，相比起完整计算 Softmax 函数，使用候选采样能够带来 100 倍以上的训练加速，同时其效果也优于分层 Softmax 等近似的方法。

10.3.1.2 YoutubeDNN 召回模型部署与应用

YoutubeDNN 通常采用离线训练、在线检索的部署方式。在离线阶段，根据用户历史观看、搜索等历史行为信息以及用户侧的静态特征来训练模型，同时生成用户 Embedding 和物品 Embedding。其中，物品 Embedding 通常离线计算并建立 ANN 向量索引，部署到在线向量检索系统中；用户 Embedding 则可以根据业务场景选择离线预计算或在线实时推理。当用户发起推荐请求时，系统获取当前用户 Embedding，并通过 ANN 检索系统快速返回 Top-K 最相似的物品，进而得到 YouTubeDNN 召回的候选集。由于在线阶段无需遍历整个物品库，仅需要完成一次用户 Embedding 计算（或直接读取）以及一次 ANN 近邻检索，因此整个召回过程通常能够控制在毫秒级，能够很好地满足工业级推荐系统对于低延迟、高吞吐在线服务的要求。

虽然 YoutubeDNN 召回模型奠定了工业级 Embedding 召回模型的基础，但其也存在一定局限性。首先，用户历史行为通常采用平均 Pooling 进行聚合，难以刻画用户兴趣的时序变化以及行为之间的依赖关系；其次，模型学习得到的是固定长度的用户 Embedding，对于具有多种兴趣的用户容易产生兴趣混叠（Interest Collapse）问题；此外，模型主要依赖物品 ID 学习物品 Embedding，对于物品侧丰富的类别、文本、图像等内容特征利用不

足，因此在冷启动场景下的泛化能力也相对有限。

与后续广泛采用的双塔召回模型相比，YoutubeDNN 模型仅在用户侧采用深度神经网络进行兴趣建模，而物品侧通常仅维护可学习的 Embedding 参数，缺乏独立的物品侧编码网络，因此难以充分建模物品的多维特征以及复杂的语义信息。随着推荐系统的发展，工业界逐渐演化出更加通用的双塔召回框架，在用户塔和物品塔两侧分别构建独立的编码网络，从而能够充分融合用户与物品两侧的丰富特征，并学习更加高质量的 Embedding 表示。在此基础上，近年来又进一步引入 Transformer、多兴趣网络、图神经网络以及多模态表示学习等技术，不断提升 Embedding 召回模型的表达能力和召回效果。

10.3.2 双塔召回

双塔召回（Two-Tower Retrieval）是目前工业界应用最广泛的 Embedding 召回模型之一，也是深度学习推荐系统中最经典的向量召回框架。双塔模型通过分别学习用户表示（User Embedding）和物品表示（Item Embedding），将推荐问题转化为向量空间中的近邻搜索（Nearest Neighbor Search）问题，再结合 ANN（Approximate Nearest Neighbor）检索技术实现毫秒级的大规模召回，因此被广泛应用于短视频推荐、电商推荐、直播推荐以及广告推荐等场景。

10.3.2.1 双塔召回模型原理

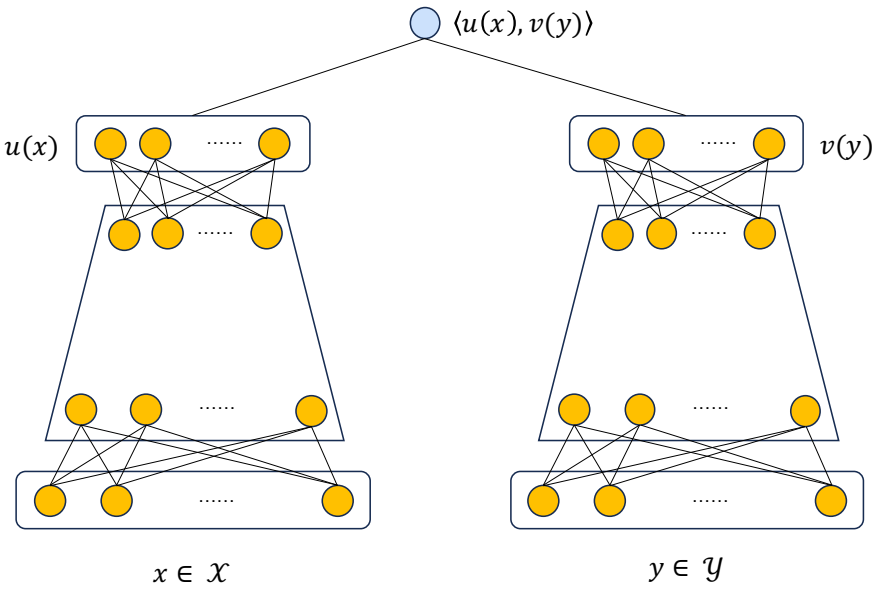

图 10.19: 双塔召回模型示意图。

双塔召回模型的整体结构如图10.19所示。通常情况下，用户塔负责学习用户兴趣表示，而物品塔负责学习物品语义表示。用户塔通常输入用户 ID、性别、年龄、地域、历史点击序列、历史观看序列、兴趣标签等用户侧特征；物品塔则输入物品 ID、类别、作者、文本、图像、视频以及统计特征等物品侧信息。两侧特征首先经过 Embedding 层映射为低维稠密向量，再通过 MLP、Transformer、DIN、BST 等网络进行特征交互，最终分别输出用户 Embedding 和物品 Embedding。具体来说，假设 $\mathbf{x}_u$ 表示用户侧特征，$\mathbf{x}_i$ 表示物品侧特征，则双塔模型的前向计算过程可以表示为：

$$\mathbf{z}_u = f_u(\mathbf{x}_u), \qquad \mathbf{z}_i = f_i(\mathbf{x}_i) \tag{10.161}$$

其中，$f_u(\cdot)$ 和 $f_i(\cdot)$ 分别表示用户塔和物品塔。在得到两侧 Embedding 之后，通常采用向量点积作为匹配分数：

$$\hat{y}_{ui} = \mathbf{z}_u^T \mathbf{z}_i \tag{10.162}$$

当然，这里的相似度计算也可以采用余弦相似度、欧氏距离等方式计算用户与物品之间的匹配程度，但工业界最常见的方法仍然是点积形式，因为其能够直接结合 ANN 检索系统进行高效的向量召回。

双塔召回模型最大的特点在于用户侧网络（User Tower）与物品侧网络（Item Tower）相互独立，二者分别根据各自特征学习 Embedding 表示，最终通过向量相似度衡量用户与物品之间的匹配程度。由于物品 Embedding 可以提前离线计算并建立 ANN 索引，因此在线阶段仅需要计算用户 Embedding，再进行一次向量检索即可完成整个召回过程，从而兼顾了推荐效果与在线服务效率。

10.3.2.1.1 模型训练 双塔模型本质上属于表示学习（Representation Learning）模型，其目标是学习用户与物品的低维 Embedding，使用户 Embedding 能够更加接近其感兴趣的物品，同时远离不感兴趣的物品。根据监督信号和优化目标的不同，目前工业界主要采用 Pointwise、Pairwise 以及 Contrastive Learning 三类训练方式。

最简单也是应用最广泛的方法是 Pointwise 训练，即直接预测用户对物品的后验反馈行为，例如点击率（CTR）、有效播放率（EVR）、点赞率（LTR）、关注率（WTR）等。设样本标签为 $y \in \{0, 1\}$，模型输出预测概率 $\hat{y}$，则二分类交叉熵损失函数可以表示为

$$\mathcal{L}_{BCE} = -\sum_{(u,i)} \left[y_{ui} \log \hat{y}_{ui} + (1 - y_{ui}) \log(1 - \hat{y}_{ui}) \right] \tag{10.163}$$

其中，$y_{ui} = 1$ 表示用户 u 对物品 i 产生了正向反馈行为，$y_{ui} = 0$ 表示用户 u 对物品 i 没有产生正向反馈行为。在具体实践中，可以直接将曝光未点击样本作为 Pointwise Loss 的负样本参与训练。Pointwise Loss 能够充分利用曝光日志进行监督学习，实现简单且训练稳定，因此目前仍然是工业界双塔召回模型最常见的训练方式之一。

另一类方法采用 Pairwise Learning 的范式，通过同时构造正负样本，使正样本物品得分高于负样本物品。最经典的方法是 BPR Loss：

$$\mathcal{L}_{BPR} = -\sum_{(u,i,j)} \log \sigma \left(\mathbf{z}_u^T \mathbf{z}_i - \mathbf{z}_u^T \mathbf{z}_j \right) \tag{10.164}$$

其中 i 表示正样本物品，j 表示负样本物品。相比 Pointwise Loss，Pairwise Loss 直接优化物品之间的排序关系，因此更加符合召回任务的优化目标。

近年来，随着对比学习的发展，越来越多的双塔模型开始采用 Softmax Loss（InfoNCE）进行训练。设一个 Mini-batch 中共有 B 组用户与正样本，则第 i 个用户对应的损失函数为

$$\mathcal{L}_i = -\log \frac{\exp(\mathbf{z}_{u_i}^T \mathbf{z}_{i_i} / \tau)}{\sum_{j=1}^{B} \exp(\mathbf{z}_{u_i}^T \mathbf{z}_{i_j} / \tau)} \tag{10.165}$$

其中 τ 表示温度系数（Temperature）。由于 Batch 内其它正样本天然可以作为负样本，因此无需额外进行随机负采样，这种训练方式通常称为 **Batch 内负采样（In-Batch Negative Sampling）**，目前已经成为工业界 Embedding 召回模型的重要训练范式。为了进一步提高模型对相似物品的判别能力，实际工程中通常还会结合 Hard Negative Sampling，从曝光未点击样本、同类别物品或上一版本召回结果中构造更加困难的负样本，使 Embedding 空间具有更好的区分能力。

除此之外，双塔模型还可以结合多任务学习（Multi-task Learning）进行联合优化。例如，在短视频推荐场景中，可以同时预测点击、有效播放、点赞、关注等多个目标，其总体损失函数通常表示为

$$\mathcal{L}_{total} = \lambda_1 \mathcal{L}_{ctr} + \lambda_2 \mathcal{L}_{evr} + \lambda_3 \mathcal{L}_{wtr} \tag{10.166}$$

其中 λ_i 表示不同任务对应的权重系数。通过共享底层 Embedding 表示，并利用多个行为信号共同监督训练，可以学习更加全面、更加鲁棒的用户兴趣表示，从而进一步提升召回效果。

需要强调的是，双塔模型的训练目标并不只是使用 Pointwise、Pairwise 或 Contrastive Loss 进行单一目标优化，而是可以根据业务需求灵活进行优化目标的组合。例如，在短视频推荐场景中，我们除了考虑优化用户的多个反馈目标的 Pairwise Loss 之外，还可以结合 Batch 内负采样的 Contrastive Loss，进一步提升 Embedding 空间的判别能力，从而获得更加优质的召回结果。

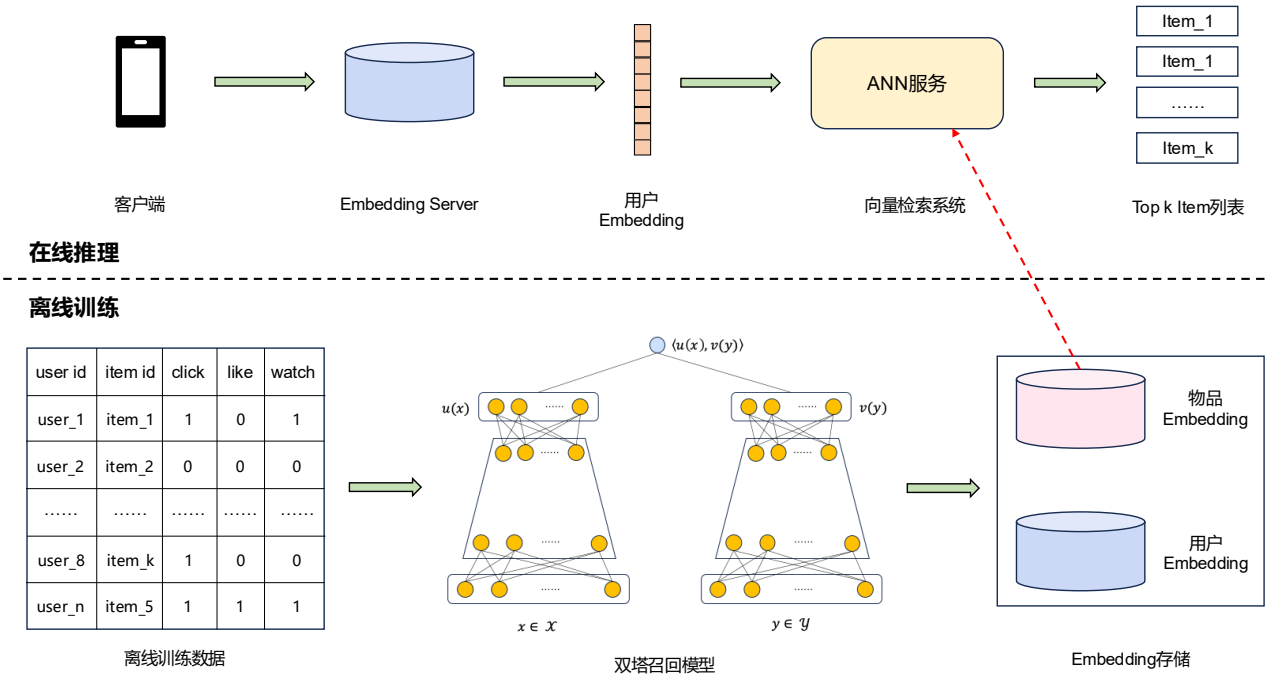

图 10.20: 双塔召回工程落地示意图。

10.3.2.1.2 双塔召回工程落地 双塔召回通常采用离线训练、在线检索的部署方式，如图10.20所示。在离线阶段，根据用户历史行为构造训练样本，通过双塔模型分别学习用户 Embedding 与物品 Embedding，并将训练得到的 Embedding 存储到 Embedding Server 中。其中，物品 Embedding 通常采用离线批量计算的方式生成，并进一步构建 ANN 向量索引，部署到在线向量检索系统中。

在线阶段，当用户发起推荐请求时，根据业务场景和时延预算的不同，用户 Embedding 通常有两种获取方式：当需要实时感知用户最新兴趣变化时，可根据用户当前行为及实时特征在线前向计算用户 Embedding；而对于时延要求较高的场景，则通常直接从 Embedding Server 中读取离线预计算的用户 Embedding。在获得用户 Embedding 之后，通过 ANN 检索系统快速检索 Top-K 近邻物品，即可生成召回候选集合。即使采用在线实时计算用户 Embedding 的方式，由于在线阶段仅需完成一次模型前向推理和一次 ANN 近邻检索，而无需遍历整个物品库，因此整个召回过程通常可以控制在数毫秒以内，能够满足工业级推荐系统低延迟、高并发的在线服务要求。

由于双塔召回兼具深度学习模型强大的表示学习能力和 ANN 近邻检索的高效性，能够在保证召回质量的同时实现大规模、高效率的在线检索，因此已经成为当前工业级推荐系统中应用最广泛的召回方案之一。在实际工程中，双塔召回通常与协同过滤召回、图召回、序列召回等多种召回通路共同组成多路召回体系，从不同角度挖掘用户兴趣，进一步提升系统整体的召回覆盖率和推荐效果。

10.3.2.1.3 双塔召回模型总结 双塔召回模型之所以成为工业界最主流的向量召回模型，主要具有以下的几个优势：

1. **检索效率高**：用户 Embedding 与物品 Embedding 互相解耦，能够结合 ANN 服务完成毫秒级向量检索。
2. **表达能力强**：用户塔与物品塔不仅可以接入 ID 特征、统计特征，还可以灵活接入文本、图像以及视频等多模态特征。
3. **工程部署简单**：物品 Embedding 可以离线预计算，线上只需计算用户 Embedding 即可完成召回过程。
4. **易于扩展**：可以方便结合 DIN、Transformer 以及多模态模型等网络结构，不断提升 Embedding 的质量。
5. **适用于大规模推荐**：双塔模型训练与推理均支持分布式部署，目前已成为工业界大规模 Embedding 召回的标准方案。

10.3.2.2 双塔召回负采样偏差处理

在双塔召回中，通常会采用负采样的方式构造训练样本。由于工业级推荐系统中通常存在大量的物品，因此在训练阶段往往无法遍历所有物品进行完整的 Softmax 计算，而是通过 Batch 内随机负采样的方式来构造负样本。然而，由于 Batch 内负采样的数据分布通常会受到物品流行度的影响从而引入较大的偏差，导致模型学习到的 Embedding 空间分布与真实分布存在差异，从而影响召回效果。

因此，谷歌在 RecSys2019 会议上发表的论文《Sampling-Bias-Corrected Neural Modeling for Large Corpus Item Recommendations》[59] 中重点提到了这一问题，并给出了相应的解决方案。该工作首先采用经典的双塔（Two-Tower）结构分别对用户和物品进行 Embedding 建模。假设用户特征为 $\mathbf{x}$，物品特征为 $\mathbf{y}$，分别通过用户塔和物品塔映射到同一向量空间：

$$u(\mathbf{x};\theta), \quad v(\mathbf{y};\theta) \tag{10.167}$$

其中 θ 表示模型参数。最终采用两者的内积作为用户与物品之间的相关性得分：

$$s(\mathbf{x},\mathbf{y}) = \langle u(\mathbf{x};\theta), v(\mathbf{y};\theta)\rangle \tag{10.168}$$

理论上，对于给定用户 $\mathbf{x}$，模型应在全部候选物品集合上进行 Softmax 计算，其概率形式为

$$P(\mathbf{y}|\mathbf{x}) = \frac{\exp(s(\mathbf{x},\mathbf{y}))}{\sum_{j=1}^{M}\exp(s(\mathbf{x},\mathbf{y}_j))} \tag{10.169}$$

并采用加权交叉熵作为训练目标：

$$L = -\frac{1}{T}\sum_{i=1}^{T} r_i \log P(\mathbf{y}_i|\mathbf{x}_i) \tag{10.170}$$

其中 r_i 表示样本对应的监督信号，T 表示数据集中的样本总数。在实际推荐系统中，r_i 不仅可以表示点击等二值标签，还可以表示观看时长、停留时间等连续标签，从而统一支持 CTR、Watch Time 等多种优化目标。

然而，工业级推荐系统中的候选物品规模通常达到百万甚至上亿级别，完整 Softmax 的计算代价极高。在实际训练时，我们通常采用 Batch 内负采样，即将同一个 Batch 中的其它正样本作为当前样本的负样本。因此基于 Batch 的 Softmax 形式为：

$$P_B(\mathbf{y}_i|\mathbf{x}_i) = \frac{\exp(s(\mathbf{x}_i,\mathbf{y}_i))}{\sum_{j=1}^{B}\exp(s(\mathbf{x}_i,\mathbf{y}_j))} \tag{10.171}$$

这种方式虽然能够显著地降低训练的开销，但会存在明显的**采样偏差（Sampling Bias）**问题。由于训练数据通常服从长尾分布，热门物品会以更高概率出现在 Batch 中，因此也更容易作为其它样本的负样本参与训练。这意味着普通的 Batch 内随机负采样会存在如下的问题，即：

笔记 热门物品被过度惩罚，而长尾物品又很少作为负样本出现，最终导致 Batch Softmax 与真实全局 Softmax 之间产生较大的偏差。

为了解决这个问题，论文借鉴了 Sampled Softmax 中的 LogQ 修正（LogQ Correction）的思想，来对每个候选物品的 Logit 进行采样概率的修正：

$$s_c(\mathbf{x}_i,\mathbf{y}_j) = s(\mathbf{x}_i,\mathbf{y}_j) - \log p_j \tag{10.172}$$

其中 p_j 表示物品 y_j 在训练过程中被采样进入当前 Batch 的概率。当某个物品由于流行度较高而具有较大的采样概率时，其对应的 $\log p_j$ 也更大，从而会适当降低其 logit 值，抵消由于高频采样所带来的偏差。因此，修正之后的 Batch Softmax 公式表示为

$$P_B^c(\mathbf{y}_i|\mathbf{x}_i) = \frac{\exp(s_c(\mathbf{x}_i,\mathbf{y}_i))}{\exp(s_c(\mathbf{x}_i,\mathbf{y}_i)) + \sum_{j\neq i}\exp(s_c(\mathbf{x}_i,\mathbf{y}_j))} \tag{10.173}$$

最终训练目标变为

$$L_B = -\frac{1}{B}\sum_{i=1}^{B} r_i \log P_B^c(\mathbf{y}_i|\mathbf{x}_i) \tag{10.174}$$

其中，B 表示 Batch 大小，r_i 表示样本对应的监督信号。通过这种方式，模型能够在不增加额外计算开销的情况下，显著缓解 Batch 内负采样所带来的采样偏差问题，从而提升双塔召回模型的召回效果。

除了**采样偏差修正（Sampling Bias Correction）**之外，论文还提出了两项能够进一步提升双塔召回效果的训练技巧。首先，对用户 Embedding 和物品 Embedding 分别进行 L_2 归一化，使其分布在单位超球面上，即

$$u \leftarrow \frac{u}{\|u\|_2}, \qquad v \leftarrow \frac{v}{\|v\|_2} \tag{10.175}$$

此时模型学习的实际上是 Embedding 之间的余弦相似度，有助于提升训练稳定性和向量检索性能。其次，在计算相似度时引入温度系数（Temperature）τ：

$$s(\mathbf{x}, \mathbf{y}) = \frac{\langle u(\mathbf{x}), v(\mathbf{y}) \rangle}{\tau} \tag{10.176}$$

其中 τ 通常作为超参数进行调节。当 τ 较小时，Softmax 分布更加尖锐，模型能够更加关注难负样本；当 τ 较大时，预测分布更加平滑，有助于训练稳定。实际工程中通常通过离线 Recall、Precision 等指标对 τ 进行调优。

最后，我们需要关注上述采样偏差修正 Softmax 中的一个关键问题，即如何获得每个物品对应的采样概率 p_j。

笔记 *采样偏差修正之后的 Batch Softmax 中，每个物品的采样概率 p_j 在实时训练过程中应该如何计算？*

这一问题直接决定了 LogQ Correction 是否能够准确修正采样偏差。在离线训练中，可以通过统计整个训练集得到每个物品的采样频率。然而，对于工业级推荐系统而言，训练数据通常以流式（Streaming）的方式持续产生，物品的流行度会随着时间不断变化，因此采样概率 p_j 也是一个动态变化的量。如果仍然采用固定统计值，不仅无法及时反映热门物品与长尾物品之间的分布变化，还会削弱采样偏差修正的效果。

针对这一问题，论文提出了 **Streaming Frequency Estimation（流式频率估计）**方法，对物品的采样概率进行在线估计。流式频率估计的基本思想如下：

定义 10.13 (流式频率估计)

流式频率估计并不是直接统计物品在窗口内出现的次数，而是利用训练过程中的全局训练步数（Global Step），记录每个物品两次被采样之间的时间间隔（Step Gap）。

假设物品 y 两次连续被采样之间的平均间隔为 δ，则其采样概率可以近似表示为

$$p_j \approx \frac{1}{\delta} \tag{10.177}$$

为了实现在线估计，系统为每个物品维护最近一次被采样的训练步数，并采用指数滑动平均（Exponential Moving Average，EMA）的方式不断更新平均采样间隔。当某个物品再次出现在当前 Batch 时，只需根据当前 Global Step 与上一次出现的 Step 计算新的间隔，并利用 EMA 完成更新。由于这种更新方式仅依赖最近一次采样信息，因此无需保存完整的历史数据，也无需维护固定长度的滑动窗口，计算和存储开销都非常低。

进一步地，为了适应工业级分布式训练环境，论文利用全局同步的 Global Step，使多个训练 Worker 能够共享同一套频率估计结果，从而保证采样概率估计的一致性。同时，为了避免维护所有物品 ID 带来的巨大存储开销，论文采用哈希表（Hash Table）的方式来保存采样的信息，并进一步借鉴 Count-Min Sketch 的思想，引入多重哈希（Multiple Hashing）来减轻哈希冲突带来的频率估计误差，从而提高采样概率估计的准确性。

通过上述流式频率估计的方法，双塔召回模型能够持续跟踪物品流行度的变化，并实时更新采样概率 p_j，最终与 LogQ Correction 结合，实现对 Batch 内负采样偏差的动态修正。这种方法不仅具有较低的计算复杂度，而且能够很好地适应工业级推荐系统中持续变化的数据分布，因此已经成为双塔召回模型中处理负采样偏差（Sampling Bias）的一种经典方法。

读者在推荐系统的学习以及具体的工业界实践中一定需要**关注负采样偏差这一问题**，因为采样偏差的存在会显著影响 Embedding 空间的学习质量，从而直接影响召回效果。如果不对采样偏差进行处理和修正，就会因为线上召回结果与训练数据分布不一致而导致召回效果下降，甚至出现冷启动物品无法被召回的情况。

10.3.2.3 双塔召回模型的发展趋势

双塔模型凭借结构简单、训练高效、易于部署等优势，已经成为工业级推荐系统中最主流的 Embedding 召回框架之一。然而，随着推荐场景不断复杂化以及用户行为数据持续增长，传统双塔模型在用户兴趣建模、表示能力以及训练方式等方面也逐渐暴露出一定局限。因此，近年来围绕双塔模型的研究主要集中在以下几个方向。

10.3.2.3.1 更加精准的用户兴趣建模 传统双塔模型通常利用用户 ID 及历史行为，通过 Pooling 或 MLP 生成固定长度的用户 Embedding。这种方式计算效率较高，但难以充分刻画用户兴趣随时间变化的动态特性。近年来，越来越多的工作开始引入 Transformer、DIN、BERT [12] 等模型，并结合长行为序列建模，对用户历史行为进行更加细粒度的兴趣建模，使生成的用户 Embedding 能够更加准确地反映用户当前的兴趣状态，从而进一步提升召回效果。

10.3.2.3.2 更加丰富的负样本构造 负样本构造直接影响 Embedding 空间的学习质量。传统双塔模型通常采用随机负采样，但随机负样本往往过于简单，模型容易快速收敛，难以学习相似物品之间细粒度的区别。因此，目前更加常见的做法是采用 Hard 负采样以及 Batch 内负采样等策略。其中，Hard 负采样通过选择与正样本较为相似的物品作为负样本，提高模型的判别能力；Batch 内负采样则直接利用同一个 Batch 中的其它样本作为负样本，在无需额外采样的情况下即可获得大量负样本。此外，还可以结合全局负样本池、图结构负采样等方式进一步优化 Embedding 空间的分布质量。

10.3.2.3.3 多模态 Embedding 学习 随着短视频、电商以及内容推荐的发展，越来越多的物品同时包含文本、图像、视频以及语音等多种模态信息。传统双塔模型主要依赖 ID 特征和统计特征，难以充分利用丰富的内容信息。因此，近年来大量论文工作将文本编码器、图像编码器以及视频编码器等引入物品塔，将多模态特征映射到统一的 Embedding 空间，从而学习更加具有语义表达能力的物品表示。这不仅能够缓解冷启动问题，还能够提升相似内容之间的召回能力。关于多模态推荐的更多内容，本书将在第**??**章进行详细介绍。

10.3.2.3.4 多目标联合优化 在工业级推荐系统中，通常需要同时兼顾点击率、观看时长、点赞、评论、分享、关注以及转化率等多个业务目标。因此，双塔模型逐渐由单个目标训练发展为多目标联合优化。常见的方法包括共享底层 Embedding、多任务学习以及多目标蒸馏等，通过融合多个行为信号共同监督模型训练，使学习得到的 Embedding 能够更加全面地刻画用户兴趣，从而兼顾不同业务目标，提高整体召回质量。

10.3.2.3.5 与图神经网络和大模型结合 近年来，图神经网络和大模型的发展进一步推动了双塔召回模型的演进。一方面，GraphSAGE、PinSage、LightGCN、HybridGNN 等图神经网络能够充分利用图结构信息学习更加高质量的用户和物品 Embedding，并进一步提升召回效果；另一方面，大语言模型以及多模态大模型能够生成具有丰富语义信息的 Embedding 表示，再结合双塔结构完成高效的 ANN 向量检索，实现语义理解能力与工业部署效率之间的有效结合。

总体来看，近年来双塔模型的发展主要围绕如何学习更加高质量的 Embedding 展开，包括更加精准的用户兴趣建模、更加合理的负样本构造、多模态表示学习、多目标联合优化以及图表示学习等多个方向。尽管模型结构不断演进，但其“学习统一 Embedding 空间，并结合 ANN 完成高效检索”这一个核心思想始终没有发生变化。因此，双塔模型目前仍然是工业级推荐系统中最重要、应用最广泛的 Embedding 召回框架之一，也是后续 YouTubeDNN、序列召回以及生成式召回等方法的重要基础。

10.3.3 基于预训练语义模型的向量召回

随着预训练模型以及大语言模型（Large Language Models, LLMs）和多模态预训练模型的发展，越来越多的研究开始探索如何将这些强大的语义理解能力应用于推荐系统的召回阶段，从而提升召回质量和用户体验。在工业界，基于预训练语义模型的向量召回方法逐渐兴起，并在短视频推荐、电商推荐以及内容推荐等场景中得

到了广泛应用。相比传统的 ID 特征和统计特征，预训练语义模型能够更好地理解文本、图像、视频等多模态内容的语义信息，从而生成更加丰富且具有语义表达能力的 Embedding 表示。

在大语言模型尚未兴起之前，即 2023 年 ChatGPT 发布之前，工业界已经开始尝试将 BERT [12]、ALBERT [29] 等预训练语言模型应用于推荐系统的召回阶段。例如，在电商推荐中，可以将商品标题、描述以及用户评论等文本信息输入到预训练语言模型中，生成商品的语义 Embedding；同时，将用户的历史行为序列、搜索查询以及兴趣标签等信息输入到同样的预训练模型中，生成用户的语义 Embedding。通过计算用户与商品之间的语义相似度，可以实现更加精准的向量召回。

因此，本节主要介绍前大模型时代较为经典的预训练语义模型及其在向量召回中的应用。在大语言模型兴起之后，如何利用 LLM 和多模态预训练模型进一步提升召回质量和用户体验，本书将在下册的第33和第**??**章分别对多模态推荐、LLM 与 Agent 推荐进行详细介绍。

10.3.3.1 FastText 模型

FastText 模型最早源自 Facebook 于 2016 年提出的论文《Enriching Word Vectors with Subword Information》[3]，是在 Word2Vec 算法中 Skip-Gram 模型基础上的一种改进方法。FastText 模型最大的特点是在学习词向量时不仅考虑完整词（Word），还进一步利用词内部的子词（Subword）信息，从而能够更好地建模词语的形态结构（Morphology），尤其适用于低频词和未登录词（Out-of-Vocabulary，OOV）的表示学习。

FastText 模型仍然采用 Skip-Gram 的训练框架，通过预测上下文单词来学习词向量，并利用负采样代替完整 Softmax 进行模型优化。与 Word2Vec 模型不同的是，FastText 模型并非直接为每个词学习一个独立的 Embedding，而是首先将一个词划分为多个字符级 n-gram。例如，对于单词“where”，在两端增加边界符号后，可生成“<wh”、“whe”、“her”、“ere”、“re>”等多个字符 n-gram，同时还将完整单词“<where>”作为一个特殊的子词。通常可以采用长度为 3～6 的字符 n-gram 作为基本组成单元。

FastText 模型为每一个字符 n-gram 学习一个 Embedding，最终一个词的表示由其所有子词 Embedding 求和得到，即

$$\mathbf{e}_w = \sum_{g \in G(w)} \mathbf{z}_g \tag{10.178}$$

其中，$G(w)$ 表示单词 w 包含的所有字符 n-gram 集合，$\mathbf{z}_g$ 表示对应子词的 Embedding。随后，将得到的词向量与上下文词向量进行内积计算，并结合 Skip-Gram 目标函数进行训练。与传统 Word2Vec 模型相比，FastText 模型在 Skip-Gram 的打分函数定义上进行了改进，具体表示为：

$$s(w, c) = \sum_{g \in G(w)} \mathbf{z}_g^T \mathbf{v}_c \tag{10.179}$$

其中，w 表示中心词，c 表示上下文词，$\mathbf{v}_c$ 表示上下文词的 Embedding。通过这种方式，FastText 模型能够在训练过程中充分利用词内部的子词信息，从而学习到更加丰富的语义表示。

由于不同单词之间通常会共享大量相同的字符 n-gram，因此 FastText 能够实现不同词之间的参数共享。例如“play”、“playing”和“played”等单词具有大量公共子词，它们学习得到的 Embedding 也会具有更高的语义相似性。同时，即使训练过程中没有出现过某个新词，只要能够提取出其字符 n-gram，FastText 仍然可以组合得到对应的 Embedding，因此具有较好的泛化能力。为了降低海量 n-gram 带来的存储开销，论文进一步采用哈希技术对字符 n-gram 进行编码，在保证较低内存占用的同时实现高效训练。

在推荐系统中，FastText 主要用于对商品标题、商品描述、用户评论、搜索 Query 等文本内容进行语义建模，并生成对应的文本 Embedding。当文本信息能够较好地反映物品属性或用户兴趣时，可直接利用 FastText 生成的 Embedding 构建向量召回索引，从而实现基于语义相似度的召回。此外，由于 FastText 能够为低频词和新词生成较高质量的表示，因此在一定程度上也能够缓解推荐系统中的冷启动问题。

下面给出一个基于 FastText 进行向量召回的示例，展示如何利用 FastText 生成商品 Embedding，并通过 ANN 检索实现相似商品的召回。假设我们有一个包含 10 个商品的电商数据集，每个商品都有对应的标题和描述信息。首先，我们使用 FastText 对商品标题进行训练，生成每个商品的 Embedding 向量。随后，将这些 Embedding 向

量存储到 FAISS 索引中，以便进行高效的向量检索。

首先需要对相关的 Python 库进行安装：

```
pip3 install fasttext numpy faiss-cpu pandas
```

然后首先读取商品数据集，这里我们使用人工构造的商品数据集作为示例，如代码10.13所示：

代码 10.13: 读取商品数据集示例

```
import fasttext
import numpy as np
import pandas as pd
import faiss
import random

# ====================== 1. 模拟电商商品数据集 ======================
# 字段：item_id, title, category, desc
data_raw = [
    {"item_id": 1, "title": "无线蓝牙耳机降噪长续航", "category": "3C数码/耳机", "desc": "主动降噪，↩
        ↪ 24小时续航，蓝牙5.3"},
    {"item_id": 2, "title": "入耳式运动蓝牙耳机防水", "category": "3C数码/耳机", "desc": "IPX7防水，↩
        ↪ 跑步专用低延迟"},
    {"item_id": 3, "title": "机械键盘青轴办公打字", "category": "3C数码/键盘", "desc": "全键热插拔，↩
        ↪ RGB背光，有线USB"},
    {"item_id": 4, "title": "静音薄膜键盘无线便携", "category": "3C数码/键盘", "desc": "超薄静音，蓝↩
        ↪ 牙双模，笔记本专用"},
    {"item_id": 5, "title": "4K高清显示器27英寸", "category": "3C数码/显示器", "desc": "IPS面板144Hz↩
        ↪ 高刷，Type-C反向充电"},
    {"item_id": 6, "title": "电竞显示器24寸165Hz", "category": "3C数码/显示器", "desc": "Fast IPS，↩
        ↪ 低蓝光，游戏专用"},
    {"item_id": 7, "title": "纯棉宽松短袖T恤男士", "category": "服饰/男装", "desc": "夏季透气纯棉，↩
        ↪ 圆领百搭基础款"},
    {"item_id": 8, "title": "冰丝速干运动短袖男款", "category": "服饰/男装", "desc": "健身跑步速干，↩
        ↪ 轻薄不闷汗"},
    {"item_id": 9, "title": "防晒连衣裙碎花长款女", "category": "服饰/女装", "desc": "夏季冰丝透气，↩
        ↪ 宽松遮肉显瘦"},
    {"item_id": 10, "title": "高腰牛仔短裤女夏季薄款", "category": "服饰/女装", "desc": "弹力修身，↩
        ↪ 百搭显瘦热裤"}
]
df = pd.DataFrame(data_raw)

# 拼接商品全部文本特征，作为FastText训练输入
df["text_all"] = df["category"] + " " + df["title"] + " " + df["desc"]

# 写入FastText训练文件（无监督词向量训练格式，每行一段文本）
train_txt_path = "fasttext_retrieve_train.txt"
with open(train_txt_path, "w", encoding="utf-8") as f:
    for text in df["text_all"].tolist():
        f.write(text + "\n")
```

接下来进行 FastText 模型训练，生成商品的文本 Embedding，如代码10.14所示：

代码 10.14: FastText 模型训练示例

```
# ====================== 2. FastText 无监督训练商品文本向量 ======================
# unsupervised: skipgram / cbow，推荐skipgram捕捉细粒度商品语义
model = fasttext.train_unsupervised(
    input=train_txt_path,
    model="skipgram", # 训练模式
    dim=64,           # 输出向量维度 64维，推荐系统常用16/32/64/128
    ws=3,             # 上下文窗口大小
    minCount=1,       # 最低词频，商品稀有类目不丢弃
    epoch=20,         # 迭代轮次
    lr=0.05,          # 学习率
    thread=4
)

# 保存/加载模型（线上推理直接加载，不用重复训练）
model.save_model("fasttext_item_retrieve.bin")
# model = fasttext.load_model("fasttext_item_retrieve.bin")

# ====================== 3. 生成全部商品Item向量 ======================
def get_item_vector(text: str, ft_model) -> np.ndarray:
    """输入商品全文本，输出归一化句向量"""
    vec = ft_model.get_sentence_vector(text)
    # L2归一化，内积等价余弦相似度，检索更快
    norm = np.linalg.norm(vec)
    if norm > 1e-8:
        vec = vec / norm
    return vec

# 批量生成所有商品向量
item_vec_list = []
item_id_map = [] # 保存索引对应item_id，召回后映射回商品ID
for _, row in df.iterrows():
    vec = get_item_vector(row["text_all"], model)
    item_vec_list.append(vec)
    item_id_map.append(row["item_id"])

item_vec_np = np.array(item_vec_list).astype(np.float32)
```

然后我们定义一个 FAISS 索引以及向量召回函数，用于快速检索 Top-K 相似商品，如代码10.15所示：

代码 10.15: FAISS 索引构建与向量召回函数示例

```
# ====================== 4. 构建FAISS向量索引（线上召回引擎） ======================
dim = item_vec_np.shape[1]
# 内积索引（归一化向量 = 余弦相似度）
index = faiss.IndexFlatIP(dim)
index.add(item_vec_np)
```

```
print(f"FAISS索引构建完成，商品总数：{index.ntotal}")

# ====================== 5. 召回函数：输入商品ID，召回TopN相似商品 ======================
def retrieve_similar_items(target_item_id: int, top_k: int = 3):
    # 1. 找到目标商品文本
    target_row = df[df["item_id"] == target_item_id].iloc[0]
    target_text = target_row["text_all"]
    # 2. 生成目标向量
    target_vec = get_item_vector(target_text, model).reshape(1, -1).astype(np.float32)
    # 3. FAISS检索
    sim_scores, idx_array = index.search(target_vec, top_k + 1) # +1 避免召回自身
    # 4. 过滤掉自身，返回结果
    retrieve_result = []
    for score, idx in zip(sim_scores[0], idx_array[0]):
        curr_item_id = item_id_map[idx]
        if curr_item_id == target_item_id:
            continue
        retrieve_result.append({
            "retrieve_item_id": curr_item_id,
            "similarity": float(score),
            "item_text": df[df["item_id"] == curr_item_id]["text_all"].iloc[0]
        })
        if len(retrieve_result) >= top_k:
            break
    return retrieve_result
```

最后，使用上述召回函数进行测试，输入不同商品 ID，查看召回的 Top-K 相似商品，如代码10.16所示：

代码 10.16: FAISS 向量召回示例

```
# ====================== 6. 测试召回效果 ======================
if __name__ == "__main__":
    # 测试1：耳机商品召回相似耳机
    print("===== 输入商品ID=1（无线蓝牙耳机），召回Top3相似商品 =====")
    res1 = retrieve_similar_items(target_item_id=1, top_k=3)
    for item in res1:
        print(f"商品ID:{item['retrieve_item_id']} 相似度:{item['similarity']:.4f} 文本:{item['↩
            ↪ item_text'][:30]}...")

    print("\n===== 输入商品ID=7（男士纯棉T恤），召回Top3相似商品 =====")
    res2 = retrieve_similar_items(target_item_id=7, top_k=3)
    for item in res2:
        print(f"商品ID:{item['retrieve_item_id']} 相似度:{item['similarity']:.4f} 文本:{item['↩
            ↪ item_text'][:30]}...")
```

运行上述代码后，输出结果如下所示：

```
FAISS索引构建完成，商品总数：10
===== 输入商品ID=1（无线蓝牙耳机），召回Top3相似商品 =====
商品ID:2 相似度:0.4485 文本:3C数码/耳机 入耳式运动蓝牙耳机防水 IPX7防水，跑步专...
```

```
商品ID:3 相似度:0.3093 文本:3C数码/键盘 机械键盘青轴办公打字 全键热插拔，RGB背光...
商品ID:10 相似度:0.1981 文本:服饰/女装 高腰牛仔短裤女夏季薄款 弹力修身，百搭显瘦热裤...

===== 输入商品ID=7（男士纯棉T恤），召回Top3相似商品 =====
商品ID:8 相似度:0.2046 文本:服饰/男装 冰丝速干运动短袖男款 健身跑步速干，轻薄不闷汗...
商品ID:6 相似度:0.1067 文本:3C数码/显示器 电竞显示器24寸165Hz Fast IP...
商品ID:9 相似度:0.0910 文本:服饰/女装 防晒连衣裙碎花长款女 夏季冰丝透气，宽松遮肉显瘦...
```

在工业级推荐系统中，FastText 模型主要应用于基于语义信息的向量召回，其应用方式主要包括 I2I（Item-to-Item）召回和 U2I（User-to-Item）召回两类。对于 I2I 召回，首先可以利用 FastText 对商品标题、描述等文本信息进行编码，生成商品的语义 Embedding；随后基于 Embedding 之间的相似度构建向量索引，从而实现相似商品召回。相比起传统基于协同过滤的方法，这种方式能够充分利用商品的文本语义信息，对于新品以及交互数据较少的长尾商品也能够召回语义相近的商品，从而缓解冷启动问题。

对于 U2I 召回，则可以利用 FastText 模型对用户历史行为中的文本信息进行语义建模，聚合用户历史浏览、点击或购买商品对应的文本 Embedding，生成用户的语义 Embedding，并与商品 Embedding 进行向量相似度计算，从而实现个性化召回。此外，在工业界更加常见的做法是将 FastText 生成的商品文本 Embedding 作为物品侧的重要语义特征，与 ID、类别、品牌等其它特征共同输入双塔召回模型的物品塔进行联合建模，从而增强物品的语义表示能力，进一步提升召回效果。

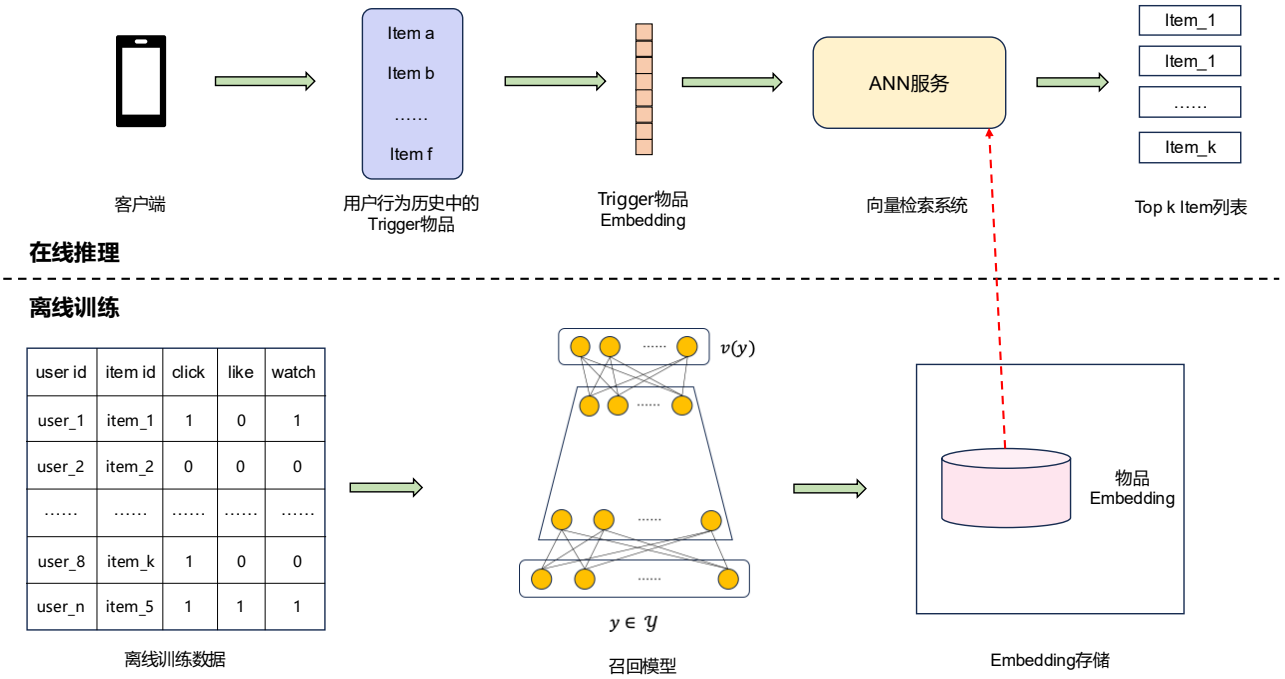

图 10.21: 基于 FastText 的 I2I 语义召回工程架构示意图

关于 U2I 向量召回的工程实现，本章前面的双塔召回部分已经进行了详细介绍，因此这里不再赘述。下面主要介绍基于 FastText 的 I2I 语义召回工程架构。整体流程与 U2I 向量召回较为相似，但两者最大的区别在于查询对象不同：U2I 召回以用户 Embedding 作为查询，在向量索引中检索相关商品；而 I2I 召回则以目标商品作为查询，通过该商品的语义 Embedding 在向量索引中检索语义相似的商品，如图 10.21所示。

在离线训练阶段，首先利用商品标题、商品描述等文本数据训练 FastText 模型，学习每个商品对应的语义 Embedding。随后，将所有商品 Embedding 离线构建为 ANN 向量索引，并存储到 Embedding Server 中，以支持在线高效检索。在线召回阶段，推荐系统通常会从用户最近点击、浏览、收藏或购买过的商品中选取一个或多个 Trigger 商品，并根据这些 Trigger 商品对应的 Embedding 作为查询向量，在 Embedding Server 中进行 ANN 检索，快速召回一批语义相似的候选商品。若存在多个 Trigger 商品，还可以对各路召回结果进行合并、去重及排

序，最终生成 I2I 召回结果。

相比传统基于协同过滤的 I2I 召回，FastText 模型能够充分利用商品文本中的语义信息，即使两个商品之间缺乏共同的交互行为，只要它们在标题或描述上具有较高的语义相似性，也能够被成功召回。因此，该方法不仅能够提升推荐结果的相关性，还能够有效缓解新品和长尾商品的冷启动问题，在电商、文本内容推荐等场景中得到了广泛应用。

10.3.3.2 BERT 模型

BERT（Bidirectional Encoder Representations from Transformers）模型是 Google 于 2018 年提出的一种预训练语言模型，最早发表于论文《BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding》[12]。BERT 基于 Transformer Encoder 架构，通过在大规模无监督语料上进行预训练，学习具有丰富上下文语义信息的文本表示。相比于此前广泛使用的 Word2Vec、GloVe [40] 等静态词向量模型，BERT 模型最大的创新在于：

定义 10.14

BERT 模型采用了双向上下文编码（**Bidirectional Context Encoding**）机制，即在表示某个词时，同时利用其左侧和右侧的上下文信息进行建模，从而能够学习更加准确、更加丰富的语义表示。

BERT 模型提出之后迅速推动了自然语言处理领域的发展，并成为随后 RoBERTa [34]、ALBERT [29]、ERNIE [63]、DeBERTa [19]、Sentence-BERT [44]、SimCSE [15]、E5 [54]、BGE [56] 等大量语义 Embedding 模型的重要基础。可以这么说，BERT 模型正式开启了预训练语言模型（Pre-trained Language Model，PLM）时代。

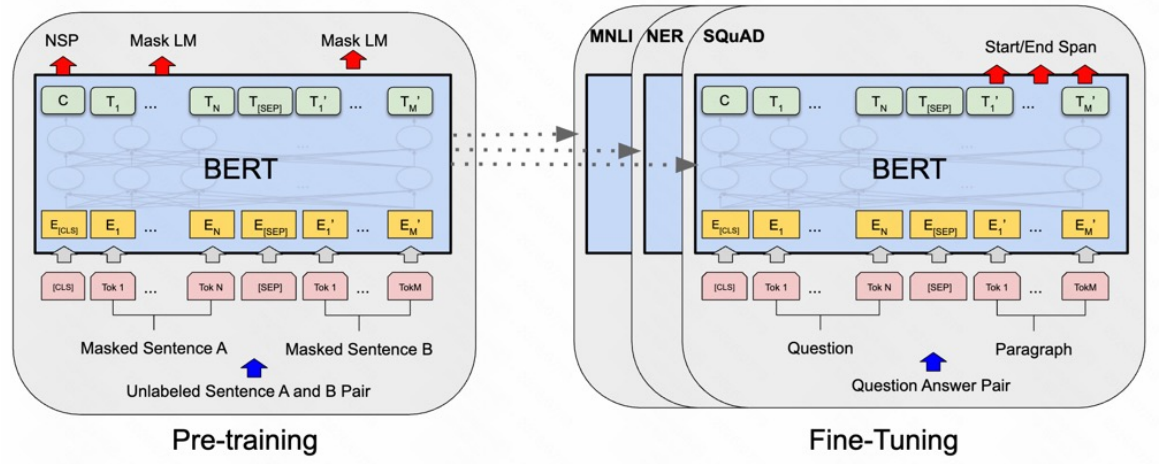

图 10.22: BERT 模型网络结构示意图。

BERT 模型整体采用“预训练（Pre-training）+ 微调（Fine-tuning）”两阶段的训练范式，其整体流程如图 10.22所示。在预训练阶段，BERT 模型首先利用海量无标签文本学习通用语言表示；随后，在具体下游任务上利用少量标注数据进行微调，使模型能够快速适应文本分类、问答、命名实体识别、文本匹配等不同任务。由于绝大部分参数均来源于预训练，因此，不同任务通常只需进行少量微调即可获得较好的性能。

BERT 模型采用标准的 Transformer Encoder 作为基础网络结构。整个模型由多层 Transformer Encoder Block 堆叠组成，每个 Block 内部均包含多头自注意力机制（Multi-Head Self-Attention）以及前馈神经网络（Feed-Forward Network，FFN）。论文中给出了两个主要版本：

- **BERT**$_{BASE}$：包含 12 层 Transformer Encoder、768 维隐藏层、12 个 Attention Head，总参数量约 110M。
- **BERT**$_{LARGE}$：包含 24 层 Transformer Encoder、1024 维隐藏层、16 个 Attention Head，总参数量约 340M。

与 GPT 等单向语言模型不同，BERT 模型采用双向 Self-Attention 机制，每个 Token 在编码过程中能够同时关注左右两侧所有 Token 的信息，因此能够学习更加完整的上下文语义表示。

BERT 模型为了统一各种自然语言处理任务，设计了一套统一的输入表示方式。对于输入序列，每个 Token 最终输入 Transformer 之前的 Embedding 由以下三部分相加得到：

1. **Token Embedding**：表示当前 Token 对应的词向量，采用 WordPiece 分词方式构建约 30K 词表。
2. **Segment Embedding**：用于区分不同句子。当输入包含两个句子时，第一个句子属于 Sentence A，第二个句子属于 Sentence B，通过不同的 Segment Embedding 进行区分。
3. **Position Embedding**：用于表示 Token 在序列中的位置信息，使 Transformer 能够感知序列顺序。

除此之外，BERT 模型还引入了两个特殊 Token：

- **[CLS]**：位于整个输入序列最前端，其最终输出 Embedding 通常作为整个句子（Sentence）的语义表示，并用于文本分类等任务。
- **[SEP]**：用于分隔不同句子，同时标识输入序列结束位置，在句子匹配、问答等双文本任务中被广泛使用。

BERT 能够学习高质量语义表示的重要原因在于其设计了两个互补的预训练任务，即 **Masked Language Model（MLM）**和 **Next Sentence Prediction（NSP）**。在 Masked Language Model（MLM）预训练任务中，传统语言模型通常只能采用从左到右（Left-to-Right）或者从右到左（Right-to-Left）的方式预测下一个词，因此无法真正实现双向建模。为了突破这一限制，BERT 提出了 Masked Language Model 任务。具体而言，模型随机选择输入序列中约 15% 的 Token 作为预测目标，其中 80% 替换为 [MASK]，10% 替换为随机 Token，另外 10% 保持原词不变，然后利用上下文信息预测原始 Token。由于被预测 Token 在输入中不可见，因此模型必须同时利用左右两侧上下文进行推断，从而学习双向语义表示。

在 Next Sentence Prediction（NSP）的预训练任务中，除了词级语义理解之外，许多自然语言处理任务还需要理解句子之间的关系，例如问答系统、自然语言推理以及文本匹配等。因此，BERT 进一步设计了 Next Sentence Prediction 任务。训练过程中，对于输入的句子对 (A, B)，其中 50% 来自语料中的真实相邻句子（IsNext），另外 50% 随机采样得到（NotNext）。模型根据 [CLS] 对应的输出向量判断两个句子是否连续，从而学习句子级语义关系。通过 MLM 和 NSP 两个任务联合训练，BERT 不仅能够学习词级上下文语义，还能够学习句子之间的关系，为各种下游任务提供统一的语义表示。

在完成预训练之后，BERT 模型可以针对不同下游任务进行端到端微调（Fine-tuning）。与传统方法需要分别设计不同模型不同，BERT 模型几乎保持相同的网络结构，仅需在顶部增加一个简单的任务输出层即可完成各种任务。例如，对于文本分类任务，通常直接使用 [CLS] 对应的输出向量进行分类；对于序列标注任务，则使用每个 Token 对应的输出 Embedding 进行预测；对于问答任务，则预测答案的起始位置和结束位置。由于预训练已经学习了丰富的语言知识，因此微调阶段通常只需较少的数据即可取得较好的效果。

BERT 模型的提出极大推动了语义表示学习的发展，在文本分类、文本匹配、自然语言推理、机器阅读理解、命名实体识别等众多自然语言处理任务上均取得了显著提升。在推荐系统中，BERT 模型主要用于商品标题、商品描述、搜索 Query、用户评论以及内容文本等信息的语义建模，将文本映射到连续向量空间中，为向量召回提供更加准确的语义表示。例如，可以利用 BERT 模型分别编码用户搜索 Query 与商品标题，通过计算二者 Embedding 之间的相似度实现语义检索；也可以利用 BERT 模型生成内容 Embedding，再结合 ANN 近似最近邻检索实现语义召回。

下面给出一个基于 BERT 进行向量召回的示例，展示如何利用 BERT 模型生成商品 Embedding，并通过 ANN 检索实现相似商品的召回。假设我们有一个包含 6 个商品的电商数据集，每个商品都有对应的商品 ID、商品标题、描述信息以及用户评论这几个字段。首先，我们使用 BERT 模型对商品标题、描述以及评论进行编码，来生成每个商品的语义 Embedding。随后，将这些 Embedding 向量存储到 FAISS 索引中，以便进行高效的向量检索。

首先需要进行相关 Python 库的安装，如下所示：

```
pip3 install torch transformers faiss-cpu numpy tf-keras tensorflow
```

然后，我们可以利用 transformers 库加载预训练好的 BERT 模型，即：

```
import os
# 禁用TensorFlow，强制使用PyTorch，防止segmentation fault
```

```
os.environ["USE_TF"] = "0"

import torch
import numpy as np
import faiss
from transformers import BertTokenizer, BertModel

# 优先读取本地，没有再自动镜像下载
MODEL_NAME = "bert-base-chinese"
tokenizer = BertTokenizer.from_pretrained(MODEL_NAME)
bert = BertModel.from_pretrained(MODEL_NAME)
bert.eval()

device = torch.device("cpu")
bert = bert.to(device)
```

接着，我们可以根据预训练的 BERT 模型进行前向传播来得到每个商品的 Embedding：

```
# ==================== 2. 原生BERT编码函数（取CLS向量作为文本语义表征） ====================
def get_bert_cls_embedding(texts, max_len=64):
    """
    原生BERT生成文本语义向量
    :param texts: 文本列表，支持批量商品/query/评论
    :param max_len: 输入文本截断长度
    :return: 归一化向量矩阵 (N, hidden_dim)，内积等价余弦相似度
    """
    # 分词编码
    inputs = tokenizer(texts, max_length=max_len, padding="max_length", truncation=True, ↩
        ↪ return_tensors="pt").to(device)

    # 前向传播不计算梯度，加速推理
    with torch.no_grad():
        output = bert(**inputs)

    # 取 位置CLS token作为整句语义向量
    cls_vec = output.last_hidden_state[:, 0, :]
    vec_np = cls_vec.cpu().numpy()

    # L2归一化，FAISS内积索引直接等价余弦相似度
    norm = np.linalg.norm(vec_np, axis=1, keepdims=True)
    vec_norm = vec_np / (norm + 1e-8)
    return vec_norm.astype(np.float32)
```

我们定义如下的电商商品数据集作为后续的测试数据：

```
# ==================== 3. 推荐系统模拟业务数据 ====================
# 商品库：商品ID + 标题 + 描述 + 用户评论（全部文本用于语义建模）
item_data = [
    {
```

```
        "item_id": 1,
        "title": "主动降噪无线蓝牙耳机超长续航",
        "desc": "蓝牙5.3低延迟，通勤听歌专用，支持快充",
        "comment": "降噪效果很好，长时间佩戴耳朵不痛"
    },
    {
        "item_id": 2,
        "title": "运动入耳式蓝牙耳机防水防汗",
        "desc": "IPX7防水，跑步健身低延迟，安卓苹果通用",
        "comment": "运动出汗不会脱落，音质清晰无杂音"
    },
    {
        "item_id": 3,
        "title": "27英寸4K 144Hz电竞IPS显示器",
        "desc": "Type-C反向充电，低蓝光护眼，游戏设计两用",
        "comment": "色彩精准，高刷玩游戏画面丝滑不卡顿"
    },
    {
        "item_id": 4,
        "title": "24英寸165Hz FastIPS游戏显示器",
        "desc": "电竞专用面板，低延迟，支持HDR显示",
        "comment": "游戏画面拖影少，性价比很高"
    },
    {
        "item_id": 5,
        "title": "男士纯棉宽松短袖夏季基础T恤",
        "desc": "透气吸汗，圆领百搭，日常通勤穿搭",
        "comment": "面料柔软，夏天穿不闷汗"
    },
    {
        "item_id": 6,
        "title": "男士冰丝速干健身运动短袖",
        "desc": "轻薄弹力面料，跑步健身专用，速干不粘身",
        "comment": "健身出汗很快干透，上身清爽"
    }
]

# 拼接单商品全部文本信息：标题+描述+评论，统一送入BERT建模
item_texts = []
item_id_list = []
for item in item_data:
    full_text = f"{item['title']} {item['desc']} {item['comment']}"
    item_texts.append(full_text)
    item_id_list.append(item["item_id"])
```

然后，我们为上述电商商品数据集中的每个物品都获取其对应的 BERT 语义 Embedding，并构建 FAISS 向量索引：

```
# ==================== 4. 全量商品生成BERT语义向量，构建ANN FAISS索引 ====================
```

```
# 批量编码所有商品文本
item_embeds = get_bert_cls_embedding(item_texts)
dim = item_embeds.shape[1]

# 构建近似最近邻检索索引（归一化向量用内积，等价余弦相似度）
faiss_index = faiss.IndexFlatIP(dim)
faiss_index.add(item_embeds)
print(f"BERT向量库构建完成，商品总数：{faiss_index.ntotal}，向量维度：{dim}")
```

然后，我们定义核心的 BERT 语义召回逻辑：

```
# ==================== 5. 语义召回核心函数 ====================
def semantic_retrieval(query_text: str, top_k=3):
    """
    两种使用场景：
    1. 用户搜索Query输入，召回语义相似商品；
    2. 输入某商品文本，召回同品类相似内容商品
    """
    # 1. 对Query做BERT编码
    query_emb = get_bert_cls_embedding([query_text])
    # 2. ANN近似最近邻检索
    sim_scores, index_ids = faiss_index.search(query_emb, top_k + 1)
    # 3. 整理召回结果
    retrieval_res = []
    for score, idx in zip(sim_scores[0], index_ids[0]):
        curr_item_id = item_id_list[idx]
        curr_full_text = item_texts[idx]
        # 避免召回自身（输入商品文本场景）
        if curr_full_text == query_text:
            continue
        retrieval_res.append({"item_id": curr_item_id, "similarity": round(float(score), 4), "text"↩
            ↪ : curr_full_text[:40] + "..."})
        if len(retrieval_res) >= top_k:
            break
    return retrieval_res
```

最后，我们就可以进行测试，输入不同的商品文本或用户搜索 Query，查看召回的 Top-K 相似商品：

```
# ==================== 6. 业务场景测试 ====================
if __name__ == "__main__":
    print("===== 场景1：用户搜索Query语义检索（原生BERT编码Query） =====")
    query1 = "降噪蓝牙耳机通勤听歌"
    res1 = semantic_retrieval(query1, top_k=2)
    print(f"用户搜索Query：{query1}")
    for info in res1:
        print(f"商品ID:{info['item_id']} 相似度:{info['similarity']} 文本:{info['text']}")

    print("\n===== 场景2：商品内容文本相似召回（内容Embedding ANN检索） =====")
    target_item_text = item_texts[2] # 27寸4K电竞显示器
```

```
res2 = semantic_retrieval(target_item_text, top_k=2)
print(f"基准商品文本: {target_item_text[:40]}...")
for info in res2:
    print(f"商品ID:{info['item_id']} 相似度:{info['similarity']} 文本:{info['text']}")

print("\n===== 场景3: 用户搜索运动短袖语义召回 =====")
query3 = "男士健身速干运动半袖"
res3 = semantic_retrieval(query3, top_k=2)
print(f"用户搜索Query: {query3}")
for info in res3:
    print(f"商品ID:{info['item_id']} 相似度:{info['similarity']} 文本:{info['text']}")
```

运行上述代码可以得到如下的输出结果：

```
BERT向量库构建完成，商品总数: 6，向量维度: 768
===== 场景1: 用户搜索Query语义检索（原生BERT编码Query） =====
用户搜索Query: 降噪蓝牙耳机通勤听歌
商品ID:1 相似度:0.7842 文本:主动降噪无线蓝牙耳机超长续航 蓝牙5.3低延迟，通勤听歌专用，支持快充降噪效果...
商品ID:2 相似度:0.7682 文本:运动入耳式蓝牙耳机防水防汗 IPX7防水，跑步健身低延迟，安卓苹果通用运动出汗...

===== 场景2: 商品内容文本相似召回（内容Embedding ANN检索） =====
基准商品文本: 27英寸4K 144Hz电竞IPS显示器 Type-C反向充电，低蓝光护眼，游戏...
商品ID:4 相似度:0.9322 文本:24英寸165Hz FastIPS游戏显示器 电竞专用面板，低延迟，支持HDR显...
商品ID:2 相似度:0.8547 文本:运动入耳式蓝牙耳机防水防汗 IPX7防水，跑步健身低延迟，安卓苹果通用运动出汗...

===== 场景3: 用户搜索运动短袖语义召回 =====
用户搜索Query: 男士健身速干运动半袖
商品ID:6 相似度:0.7835 文本:男士冰丝速干健身运动短袖 轻薄弹力面料，跑步健身专用，速干不粘身 健身出汗很快干...
商品ID:5 相似度:0.7096 文本:男士纯棉宽松短袖夏季基础T恤 透气吸汗，圆领百搭，日常通勤穿搭 面料柔软，夏天穿...
```

需要指出的是，虽然 BERT 模型能够生成高质量的文本表示，但其最初并不是专门为向量检索设计的。由于 BERT 模型通常需要将待匹配的两段文本同时输入模型进行联合编码，其计算复杂度较高，难以直接应用于大规模召回场景。因此，后续研究提出了 Sentence-BERT（SBERT）、SimCSE、E5、BGE 等专门面向文本 Embedding 优化的模型，将文本分别编码为独立向量，再利用余弦相似度进行高效检索，从而成为目前工业界语义召回和文本 Embedding 的主流方案。

10.3.3.3 Sentence-BERT 模型

Sentence-BERT（SBERT）模型是 Nils Reimers 等人于 2019 年提出的一种面向句子语义表示学习的预训练模型，最早发表于论文《Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks》[44]。SBERT 在 BERT 的基础上进行了针对语义匹配任务的结构改进，使模型能够直接生成具有丰富语义信息的句子级 Embedding，并广泛应用于语义检索、文本匹配、问答系统、文本聚类以及推荐系统向量召回等任务。

SBERT 模型提出的主要动机来源于传统 BERT 模型在语义检索中的计算效率问题。对于文本相似度计算任

务，原始 BERT 模型通常采用 Cross-Encoder 结构，将两个句子拼接后共同输入模型进行联合编码，例如：

$$[\text{CLS}]\ Sentence_A\ [SEP]\ Sentence_B\ [SEP] \tag{10.180}$$

随后利用 CLS 对应的输出向量完成分类或者相似度预测。这种方式能够充分利用两个句子之间的交互信息，因此具有较高的匹配精度。然而，对于包含百万甚至千万级文档的大规模检索场景，每个 Query 都需要分别与所有候选文本进行联合编码，其计算复杂度约为 $O(N)$ 次 Transformer 前向计算，无法满足工业级在线召回系统对于低延迟和高吞吐的要求。

为了解决这一问题，SBERT 模型借鉴了 Siamese Network（孪生网络）的思想，将原始 BERT 改造成双塔（Bi-Encoder）结构。两个输入句子分别经过**共享参数的 BERT 编码器**进行独立编码，然后通过 Pooling 层得到固定长度的句向量（Sentence Embedding），最后利用余弦相似度衡量两个句子的语义相似性。其整体结构如图 10.23所示。相比传统 BERT 模型最大的不同在于，SBERT 模型将文本编码与文本匹配两个过程进行了彻底解耦。所有

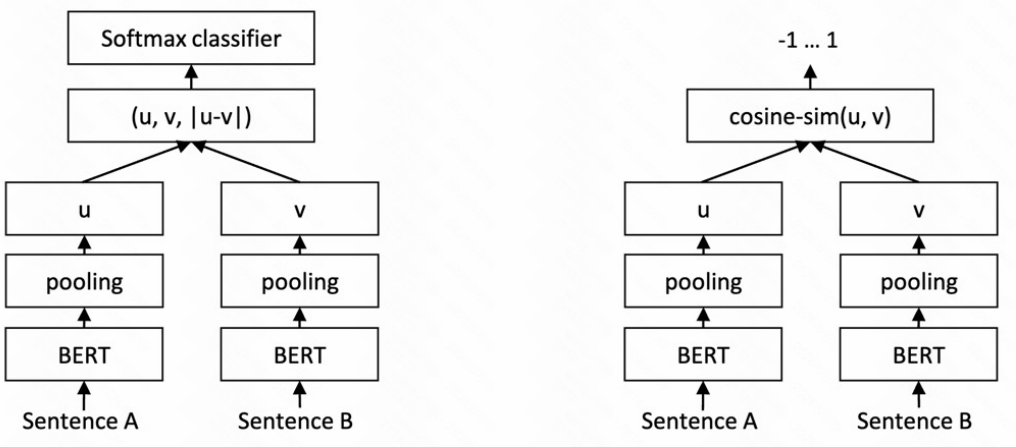

图 10.23: Sentence-BERT（SBERT）模型结构示意图。

候选文本可以提前离线编码得到 Embedding，并建立 ANN 向量索引；在线阶段仅需编码用户 Query，再通过向量检索快速找到最相似的候选文本即可。因此，大规模文本检索的计算复杂度由原来的两两匹配降低为一次编码加一次 ANN 检索，大幅提升了在线召回效率，这也是目前工业界 Embedding 召回普遍采用 Bi-Encoder 结构的重要原因。

在得到 BERT 编码器最后一层输出之后，由于每个 Token 都会对应一个隐藏向量，而句子长度通常不固定，因此 SBERT 模型进一步引入 Pooling 层，将 Token 级表示聚合为固定维度的句向量。论文中主要比较了三种 Pooling 方式：

1. **CLS Pooling**：直接使用 CLS Token 对应的输出向量作为整个句子的表示。
2. **Mean Pooling**：对所有 Token 对应的输出向量进行平均池化（Mean Pooling），作为句子 Embedding。
3. **Max Pooling**：对所有 Token 对应维度分别取最大值（Max Pooling），得到最终句向量。

实验结果表明，Mean Pooling 通常能够获得最稳定、最优的语义表示，因此也是 SBERT 模型默认采用的 Pooling 策略。目前包括 E5、BGE、GTE 等大量 Embedding 模型也普遍采用 Mean Pooling 生成最终文本 Embedding。

为了使生成的句向量能够更好地反映语义相似性，SBERT 采用共享参数的 Siamese Network 进行微调训练。根据不同的数据集形式，论文分别设计了三种训练目标。第一种训练目标是**分类目标（Classification Objective）**。对于自然语言推理（Natural Language Inference，NLI）等分类任务，首先分别编码两个句子得到 Embedding u 和 v，随后将二者以及它们的逐元素差值拼接：

$$[u,\ v,\ |u-v|] \tag{10.181}$$

再经过全连接层和 Softmax 分类器预测句子之间的关系（Entailment、Neutral 或 Contradiction），并采用交叉熵损失（Cross Entropy Loss）进行训练。这也是论文中训练 SBERT 最主要采用的方式。第二种训练目标是**回归目标（Regression Objective）**。对于语义相似度（Semantic Textual Similarity，STS）任务，模型直接计算两个句向量之

间的余弦相似度：

$$\text{sim}(u, v) = \frac{u^\top v}{\|u\|\|v\|} \tag{10.182}$$

并利用均方误差（Mean Squared Error，MSE）损失，使预测相似度逼近人工标注的相似度分数。第三种训练目标是 **Triplet Loss**。对于检索任务，SBERT 进一步采用 Triplet Loss 进行优化。训练样本由三部分组成：Anchor（查询句子）、Positive（语义相关文本）和 Negative（语义无关文本）。SBERT 模型希望 Anchor 与 Positive 之间的距离尽可能接近，而 Anchor 与 Negative 之间的距离尽可能远，其优化目标可表示为：

$$L = \max\left(d(a, p) - d(a, n) + \epsilon, 0\right) \tag{10.183}$$

其中 $d(\cdot)$ 表示欧氏距离或余弦距离，ϵ 表示 Margin。Triplet Loss 能够直接优化 Embedding 空间中的几何结构，因此特别适用于语义检索和向量召回等任务。

SBERT 模型在训练过程中主要采用 SNLI（Stanford Natural Language Inference）和 MultiNLI 两个自然语言推理数据集，共计约 100 万条句子对作为监督数据。模型采用 Adam 优化器进行训练，Batch Size 为 16，学习率为 2×10^{-5}，默认采用 Mean Pooling 作为句向量生成方式。

SBERT 模型最大的贡献在于首次将 BERT 模型成功改造成适用于向量检索的 Bi-Encoder 架构，使得文本 Embedding 可以独立计算并离线存储，再结合 ANN 近似最近邻检索实现毫秒级语义召回。其提出之后，几乎成为后续所有文本 Embedding 模型的基础框架。包括 SimCSE、E5、BGE、GTE、NV-Embed、Nomic-Embed 等近年来广泛应用于工业界的 Embedding 模型，均可以看作是在 SBERT 整体框架基础上的进一步改进，例如采用更大的预训练语料、更优的对比学习目标、更困难的负样本构造以及更强大的预训练语言模型作为编码器。

在推荐系统中，SBERT 模型主要用于生成商品标题、商品描述、搜索 Query、用户评论以及内容文本等信息的语义 Embedding，并结合 ANN 向量检索实现 Embedding 召回。例如，可以离线计算所有商品标题的 Sentence Embedding 建立向量索引；在线阶段仅需对用户搜索 Query 或兴趣文本进行编码，即可快速检索语义最相关的商品或内容。相比传统基于关键词匹配的方法，SBERT 模型能够更好地理解文本语义关系，因此在搜索推荐、内容推荐以及多模态推荐等场景中得到了广泛应用。

由于 SBERT 模型在具体工业界实践中用法与 BERT 模型类似，因此这里不再赘述。这里主要给出 SBERT 模型的加载代码，方便读者后续使用 SBERT 模型进行向量召回。首先我们需要安装相应的 Python 库：

```
pip3 install torch sentence-transformers numpy
```

然后我们给出 SBERT 模型的加载流程，如代码所示：

代码 10.17: SBERT 模型加载代码。

```
import os
os.environ["USE_TF"] = "0"

from sentence_transformers import SentenceTransformer
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

# 多语言SBERT模型名称
model_name = "sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2"
# 加载SBERT模型
sb_model = SentenceTransformer(model_name)

def get_multilingual_embedding(text_list):
    """生成归一化多语言句向量，适配余弦相似度检索"""
    embeddings = sb_model.encode(
```

```
        text_list,
        convert_to_numpy=True,
        show_progress_bar=True
    )
    # L2归一化：内积 = 余弦相似度
    norm = np.linalg.norm(embeddings, axis=1, keepdims=True)
    emb_norm = embeddings / (norm + 1e-8)
    return emb_norm.astype(np.float32)

# 多语言测试
if __name__ == "__main__":
    texts = [
        "无线降噪蓝牙耳机",
        "Wireless noise cancelling earphones",
        "長時間再生ワイヤレスイヤホン"
    ]
    labels = ["中文耳机", "英文耳机", "日文耳机"]
    vecs = get_multilingual_embedding(texts)
    print(f"向量维度：{vecs.shape[1]}")
    print("===== 中英日三语相同语义向量两两余弦相似度 =====")

    # 计算全部两两相似度矩阵
    sim_matrix = cosine_similarity(vecs)

    # 打印两两对比结果
    for i in range(len(texts)):
        for j in range(len(texts)):
            if i == j:
                continue
            print(f"{labels[i]}  {labels[j]} 相似度：{sim_matrix[i][j]:.4f}")

    # 单独演示：用中文query检索最相似的跨语言文本（模拟召回）
    print("\n===== 模拟跨语言语义召回：以中文「无线降噪蓝牙耳机」为查询 =====")
    query_vec = vecs[0].reshape(1, -1)
    scores = cosine_similarity(query_vec, vecs)[0]

    # 排序，过滤自身
    score_with_idx = sorted([(scores[idx], idx) for idx in range(len(texts)) if idx != 0], reverse↩
        ↪ =True)
    for score, idx in score_with_idx:
        print(f"匹配文本：{texts[idx]} 相似度：{score:.4f}")
```

运行上述代码，可以得到如下输出结果：

```
向量维度：384
===== 中英日三语相同语义向量两两余弦相似度 =====
中文耳机  英文耳机 相似度：0.5780
中文耳机  日文耳机 相似度：0.5762
英文耳机  中文耳机 相似度：0.5780
```

```
英文耳机   日文耳机 相似度: 0.5032
日文耳机   中文耳机 相似度: 0.5762
日文耳机   英文耳机 相似度: 0.5032

===== 模拟跨语言语义召回: 以中文「无线降噪蓝牙耳机」为查询 =====
匹配文本: Wireless noise cancelling earphones 相似度: 0.5780
匹配文本: 長時間再生ワイヤレスイヤホン 相似度: 0.5762
```

10.3.3.4 SimCSE 模型

SimCSE（Simple Contrastive Learning of Sentence Embeddings）模型是清华大学研究团队于 2021 年提出的一种基于对比学习（Contrastive Learning）的句子表示学习模型，最早发表于论文《SimCSE: Simple Contrastive Learning of Sentence Embeddings》[15]。SimCSE 模型在 BERT 和 Sentence-BERT（SBERT）模型的基础上进一步改进了句向量的训练方式，通过一种极其简单但十分有效的对比学习策略，使模型能够学习更加符合语义结构的 Embedding 表示，在语义文本相似度（Semantic Text Similarity, STS）、文本检索以及语义召回等任务上均取得了优异的性能。

SimCSE 模型的提出主要是为了进一步解决 BERT 和 SBERT 模型在句子向量表示方面仍然存在的问题。虽然 SBERT 模型已经采用 Siamese 双塔结构生成 Sentence Embedding，并能够利用余弦相似度进行快速语义检索，但其训练过程主要依赖自然语言推理（Natural Language Inference，NLI）等监督数据，而 Embedding 空间并没有直接按照向量之间的几何关系进行优化，因此生成的句向量仍然存在一定程度的**表示退化（Representation Collapse）**以及**各向异性（Anisotropy）**问题，即不同句子的 Embedding 容易聚集在向量空间中的少数方向，导致句向量之间缺乏足够的区分能力。

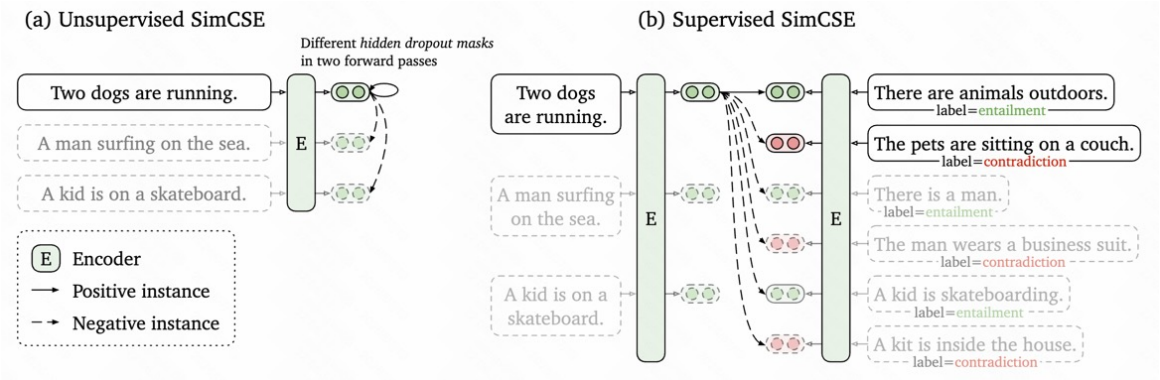

图 10.24: SimCSE 模型训练框架示意图。

针对这一问题，SimCSE 模型首次将对比学习（Contrastive Learning）引入句子 Embedding 学习，通过直接优化 Embedding 空间中的距离关系，使语义相近的句子距离更近，而语义无关的句子距离更远，从而显著提升句向量的判别能力。SimCSE 模型包括两种训练方式：**无监督 SimCSE（Unsupervised SimCSE）**和**有监督 SimCSE（Supervised SimCSE）**，整个训练框架如图 10.24所示。两种方法均采用 Bi-Encoder 结构独立编码文本，并利用 InfoNCE 对比学习损失优化 Embedding 空间，不同之处主要体现在正负样本的构造方式。

10.3.3.4.1 无监督 SimCSE 模型 无监督 SimCSE 模型最大的创新在于提出了一种极其简单的正样本构造方式。对于输入句子 $\mathbf{x}$，无监督 SimCSE 模型并不需要进行任何数据增强，也无需人为构造同义句，而是直接将同一句子输入编码器两次：

$$\mathbf{x}^+ = \mathbf{x} \tag{10.184}$$

唯一的区别在于，两次前向传播分别采用不同的 Dropout Mask。设编码器表示为 $f_\theta(\cdot)$，随机 Dropout 掩码分别为 z 和 z'，则可以得到两个不同的句向量：

$$\mathbf{h}_i = f_\theta(\mathbf{x}_i, z), \qquad \mathbf{h}_i^+ = f_\theta(\mathbf{x}_i, z') \tag{10.185}$$

虽然输入文本完全相同，但由于 Dropout 随机失活神经元，两次编码得到的 Embedding 存在轻微扰动，从而形成了一组天然的正样本对（Positive Pair）。对于一个 Batch 内的其它句子，其 Embedding 均作为负样本（Negative Samples），因此无需额外采样即可利用 Batch 内其它样本构造大量负样本。

最终，无监督 SimCSE 模型采用 InfoNCE 损失进行优化：

$$L_i = -\log \frac{\exp(\text{sim}(\mathbf{h}_i, \mathbf{h}_i^+)/\tau)}{\sum_{j=1}^{N} \exp(\text{sim}(\mathbf{h}_i, \mathbf{h}_j^+)/\tau)} \tag{10.186}$$

其中，$\text{sim}(\cdot)$ 表示余弦相似度，τ 表示温度参数（Temperature），N 表示每个 Batch 的大小。整个训练过程中唯一的数据增强便是 Transformer 内部原本就存在的 Dropout，因此论文将其称为一种**“最小数据增强（Minimal Data Augmentation）”**。实验结果表明，相比起随机删除单词、随机替换单词等传统文本增强方法，只利用 Dropout 噪声即可获得更好的句向量表示。SimCSE 模型的论文进一步分析发现，如果关闭 Dropout（Dropout=0），或者两次编码采用完全相同的 Dropout Mask，则两个 Embedding 完全一致，模型很容易发生表示坍塌（Collapse），训练效果明显下降。这说明随机 Dropout 不仅能够提供正样本扰动，同时还能有效保持 Embedding 空间中的对齐性（Alignment）与均匀性（Uniformity），从而成为无监督 SimCSE 模型成功的关键。

10.3.3.4.2 有监督 SimCSE 模型 相比起无监督方法，有监督 SimCSE 模型进一步利用自然语言推理（Natural Language Inference，NLI）数据集提供更加准确的监督信号。论文中采用 SNLI（Stanford Natural Language Inference）和 MultiNLI 两个数据集，其中每个样本均包含：Premise（前提句）、Entailment（蕴含句）、Contradiction（矛盾句）、Neutral（中立句）。对于每个 Premise，Entailment 句子天然作为正样本，而 Contradiction 句子则作为 Hard Negative（困难负样本），从而形成三元组：

$$(\mathbf{x},\ \mathbf{x}^+,\ \mathbf{x}^-) \tag{10.187}$$

其中：$\mathbf{x}$ 表示 Anchor（查询句），$\mathbf{x}^+$ 表示 Entailment（正样本），$\mathbf{x}^-$ 表示 Contradiction（困难负样本）。

在训练过程中，有监督 SimCSE 模型仍采用 InfoNCE 损失进行优化，只是在分母中额外加入 Hard Negative。模型的损失函数如下所示：

$$L_i = -\log \frac{\exp(\text{sim}(\mathbf{h}_i, \mathbf{h}_i^+)/\tau)}{\sum_{j=1}^{N} \left(\exp(\text{sim}(\mathbf{h}_i, \mathbf{h}_j^+)/\tau) + \exp(\text{sim}(\mathbf{h}_i, \mathbf{h}_j^-)/\tau)\right)} \tag{10.188}$$

其中 $\mathbf{h}_i$ 表示当前样本的 Anchor 句向量（Premise），$\mathbf{h}_i^+$ 表示与 $\mathbf{h}_i$ 语义蕴含（Entailment）的正样本句向量，$\mathbf{h}_i^-$ 表示与 $\mathbf{h}_i$ 语义矛盾（Contradiction）的 Hard Negative 句向量，N 表示 Mini-Batch 中的样本数量，τ 表示温度参数（Temperature）。

需要特别注意的是，分母中同时包含 $\mathbf{h}_j^+$ 和 $\mathbf{h}_j^-$，其含义并不相同。对于当前 Anchor $\mathbf{h}_i$ 而言，只有 $\mathbf{h}_i^+$ 是其真正的正样本；而 Batch 中其它样本的正样本 $\mathbf{h}_j^+ (j \neq i)$ 虽然对各自的 Anchor 是正样本，但相对于当前 Anchor $\mathbf{h}_i$ 来说，它们通常是语义无关的，因此被当作普通负样本（In-Batch Negatives）。例如，假设一个 Batch 中包含三组样本：

$$\left(\mathbf{h}_1, \mathbf{h}_1^+, \mathbf{h}_1^-\right), \quad \left(\mathbf{h}_2, \mathbf{h}_2^+, \mathbf{h}_2^-\right), \quad \left(\mathbf{h}_3, \mathbf{h}_3^+, \mathbf{h}_3^-\right)$$

当计算 L_1 时，正样本为 $\mathbf{h}_1^+$，普通负样本为 $\mathbf{h}_2^+$ 和 $\mathbf{h}_3^+$，而 Hard Negative 为 $\mathbf{h}_1^-$、$\mathbf{h}_2^-$ 和 $\mathbf{h}_3^-$。因此，分母实际上包含了两类竞争项：

1. **Batch 内其它样本的正样本 $\mathbf{h}_j^+$**：这些句子虽然是其它 Anchor 的匹配句，但对于当前 Anchor 通常不相关，因此作为普通负样本参与对比。
2. **矛盾句 Hard Negative $\mathbf{h}_j^-$**：这些句子与各自 Anchor 具有较强语义相关性但语义方向相反，属于更加困难的负样本。

从优化目标的角度看，InfoNCE 希望最大化当前 Anchor 与其真实正样本 $\mathbf{h}_i^+$ 之间的相似度，同时最小化其与所

有其它候选向量之间的相似度。因此，该损失可以理解为：

$$\text{正确匹配 } \mathbf{h}_i^+ \quad \text{vs.} \quad \text{所有其它候选向量} \tag{10.189}$$

其中“所有其它候选向量”既包括 Batch 内其它样本的正样本，也包括专门构造的 Hard Negative。

从上述示例以及分析可以看出，在有监督的 SimCSE 模型中引入 $\mathbf{h}_j^-$ 的核心价值在于：普通 Batch 内负样本往往与 Anchor 差异较大，模型容易区分；而语义矛盾的句子在词汇层面通常与 Anchor 高度相似，例如共享大量关键词，但语义上却完全相反，因此能够为模型提供更强的判别信号，迫使 Embedding 空间学习更加细粒度的语义边界。这也是有监督 SimCSE 模型相比无监督 SimCSE 模型性能进一步提升的重要原因。

10.3.3.4.3 SimCSE 模型分析与实践 总体来看，无监督 SimCSE 模型利用 Dropout 噪声自动构造正样本，仅依赖大规模无标注语料即可训练高质量句向量；而有监督 SimCSE 模型则进一步利用 NLI 数据集中的 Entailment 和 Contradiction 关系，引入更加准确的监督信号，使 Embedding 空间具有更好的语义判别能力。

SimCSE 模型最大的贡献在于证明了高质量文本 Embedding 并不一定依赖复杂的数据增强或网络结构，仅利用 Transformer 自身的 Dropout 机制配合 InfoNCE 对比学习损失函数，便能够学习具有良好对齐性与均匀性的 Embedding 空间。在 SimCSE 模型提出之后，对比学习逐渐成为文本 Embedding 模型的主流训练范式，后续大量模型，如 E5、BGE、GTE、NV-Embed 以及 Qwen3-Embedding 等，均采用了与 SimCSE 模型类似的对比学习框架，并进一步结合更大规模的预训练语料、弱监督数据以及困难负样本挖掘等技术来不断提升文本表示能力。

在推荐系统中，SimCSE 模型同样可以用于学习商品标题、搜索 Query、用户评论以及内容文本等信息的语义 Embedding，并结合 ANN 向量检索实现高效语义召回。由于其无需大量标注数据即可训练，因此特别适合长尾商品、冷启动内容以及开放领域文本检索等场景。目前，SimCSE 也被广泛作为各类文本 Embedding 模型的预训练或初始化方法，为现代语义召回模型的发展奠定了重要基础。

10.3.3.5 E5 模型

E5（Embedding from Weakly-supervised Text Embeddings）模型是微软亚洲研究院（Microsoft Research Asia，MSRA）提出的一种通用文本表示模型，最早发表于 2022 年的论文《Text Embeddings by Weakly-Supervised Contrastive Pre-training》[54]。E5 模型旨在学习具有良好泛化能力的统一文本 Embedding，使其能够直接应用于语义检索、文本匹配、文本分类、文本聚类以及问答系统等多种自然语言处理任务。

与传统的 BERT 主要面向分类任务不同，E5 模型从设计之初便以文本向量表示（Text Embedding）为目标，其核心思想是通过大规模弱监督文本对进行对比学习（Contrastive Learning）预训练，再利用少量高质量标注数据进行监督微调，从而学习具有较强语义表达能力的文本 Embedding。整个训练过程可以划分为两个阶段，如图 10.25所示。

第一阶段为**弱监督对比学习预训练（Weakly-supervised Contrastive Pre-training）**。E5 模型首先收集海量无标注文本对（CCPairs）作为该模型的训练数据，其中每个样本由一个文本对 (q_i, p_i) 组成，分别表示查询（Query）和对应的相关文本（Passage）。在训练过程中，E5 模型仍然采用双塔（Bi-Encoder）模型的结构，但共享同一个 Transformer 编码器分别编码 Query 和 Passage，并通过平均池化（Mean Pooling）来获得固定长度的文本 Embedding：

$$\mathbf{e}_q = \text{Encoder}(q), \qquad \mathbf{e}_p = \text{Encoder}(p) \tag{10.190}$$

随后利用余弦相似度计算两者之间的语义相关性，并通过温度参数 τ 进行缩放：

$$s(q, p) = \frac{\cos(\mathbf{e}_q, \mathbf{e}_p)}{\tau} \tag{10.191}$$

为了学习到具有判别能力的 Embedding，E5 模型采用了 InfoNCE 对比学习的损失函数，将真实文本对作为正样本，而 Batch 内其它样本对应的 Passage 作为负样本，即 Batch 内负采样策略。E5 模型的对比学习损失函数如下：

$$L_{\text{cont}} = -\log \frac{\exp(s(q_i, p_i))}{\exp(s(q_i, p_i)) + \sum_j \exp(s(q_i, p_j^-))}. \tag{10.192}$$

这种训练方式能够不断拉近正样本 Embedding 之间的距离，同时拉远负样本 Embedding 之间的距离，从而构建具有良好语义结构的向量空间。相比于传统全局随机负采样策略，Batch 内负采样策略不仅实现更加简单，而且

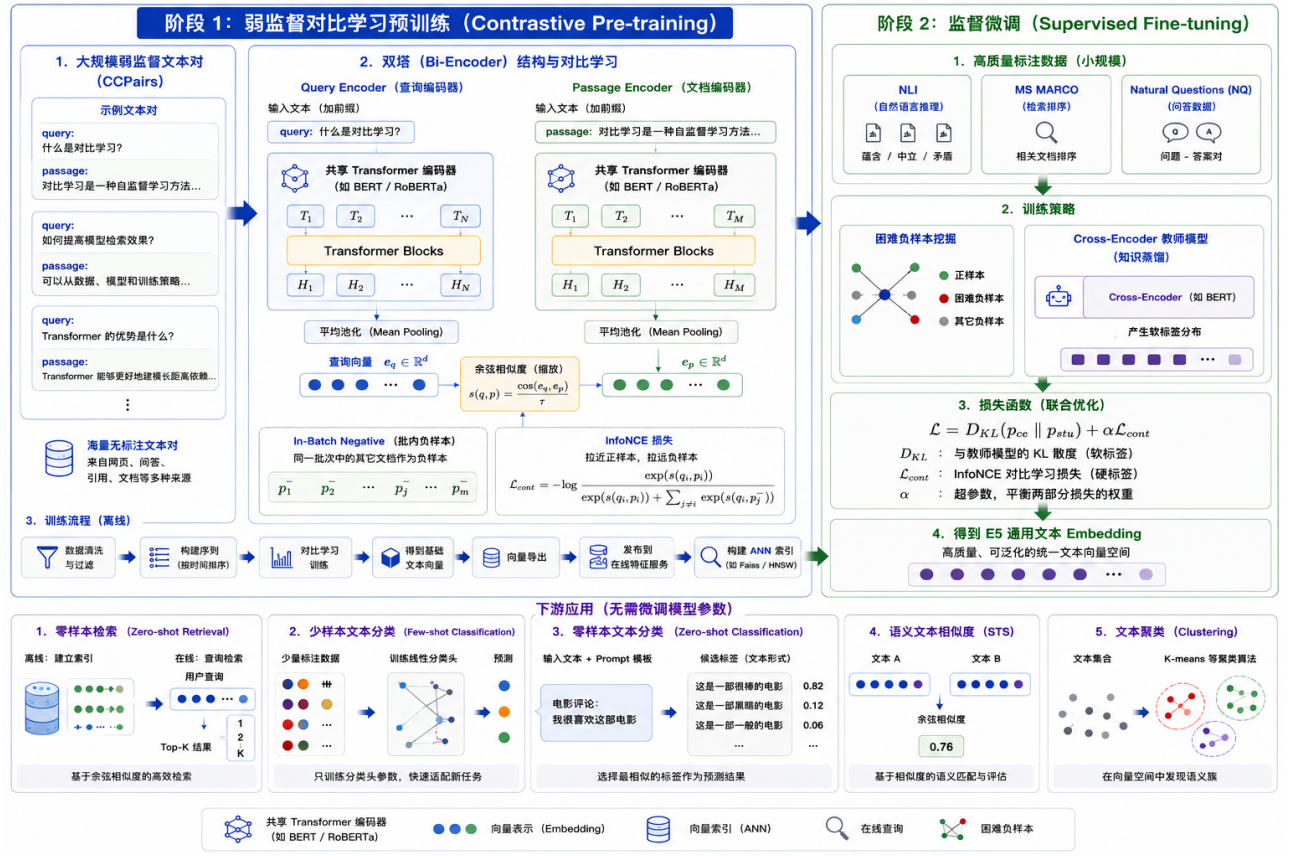

图 10.25: E5 模型训练框架示意图。

能够充分利用大 Batch 训练带来的大量困难负样本，因此目前已经成为 Embedding 模型训练中的主流方案。

为了进一步区分查询侧与文档侧的语义角色，E5 在输入文本前分别添加“`query:`”和“`passage:`”两个前缀，例如：

```
query: What is contrastive learning?
passage: Contrastive learning is a self-supervised...
```

这种非对称输入方式能够帮助模型学习不同角色之间的语义关系，对于信息检索等 Query-Document 匹配任务具有更加明显的效果，因此后来也被许多 Embedding 模型广泛采用。

在完成大规模对比学习预训练之后，E5 模型进入到第二阶段，即**监督微调（Supervised Fine-tuning）**。虽然弱监督训练已经能够学习较好的通用 Embedding，但为了进一步提升模型性能，E5 模型继续利用少量高质量标注数据进行微调，包括自然语言推理（Natural Language Inference，NLI）、MS MARCO Passage Ranking 以及 Natural Questions（NQ）等经典数据集。其中，对于 MS MARCO 和 NQ 等检索任务，E5 模型进一步引入 Cross-Encoder 教师模型进行知识蒸馏（Knowledge Distillation），并结合困难负样本（Hard Negative）共同优化模型。

最终，E5 模型的训练目标由 InfoNCE 对比学习损失与教师模型蒸馏损失共同组成，即：

$$L = D_{KL} + \alpha L_{\text{cont}} \tag{10.193}$$

其中，D_{KL} 表示学生模型与 Cross-Encoder 教师模型之间的 KL 散度损失，L_{cont} 表示 InfoNCE 对比学习损失，α 用于平衡两部分损失。通过引入教师模型提供的软标签信息，可以进一步提升 Embedding 模型对于文本相关性的刻画能力。

经过上述两个阶段训练后，E5 能够学习到具有较强泛化能力的统一文本 Embedding。在线应用时，目标文本首先离线计算 Embedding 并建立 ANN 向量索引；在线查询阶段，仅需计算用户 Query 的 Embedding，即可利用余弦相似度完成高效的 Top-K 语义检索。因此，E5 不仅能够应用于语义召回，还能够直接迁移到语义相似度

计算、文本聚类、零样本文本分类（Zero-shot Classification）、少样本文本分类（Few-shot Classification）以及问答系统等多种自然语言处理任务。

总体而言，E5 模型开创了**“大规模弱监督对比学习预训练 + 少量监督微调”**的现代文本 Embedding 训练范式，在 MTEB（Massive Text Embeddings Benchmark）等公开基准测试中取得了优异性能。此后，大量优秀的文本 Embedding 模型，例如 BGE [56]、GTE [33]、NV-Embed [30] 以及后续基于大语言模型构建的 Embedding 模型，都在不同程度上借鉴了 E5 模型的训练思想，因此 E5 模型也成为近年来语义 Embedding 领域最具代表性的基础模型之一。基于 E5 模型的向量召回方法原理与前文介绍的 FastText、BERT、SBERT 以及 SimCSE 等模型类似，主要通过计算 Trigger 文本与候选文本之间的 Embedding 相似度完成语义召回，具体实现方式可参考前文的 FastText 示例。

10.3.3.6 Sentence Transformers 库

Sentence Transformers 是由 Sentence-BERT（SBERT）作者维护的开源框架和模型库，目前已经成为文本语义表示（Text Embedding）领域最流行的开源工具之一。该框架不仅提供了丰富的预训练模型，还提供了统一的模型加载、训练、微调以及推理接口，使研究人员和工程师能够方便地构建语义检索、文本匹配以及信息检索等应用。

Sentence Transformers 最初主要用于 Sentence-BERT 系列模型的训练与推理，随着近年来文本表示技术的发展，目前已经逐渐演化为一个统一的文本表示框架。除 SBERT 系列模型外，该框架还支持 MiniLM、MPNet 等经典文本向量模型，并能够兼容加载 E5、BGE、GTE 等主流检索模型以及近年来大量基于大语言模型（LLM）的 Embedding 模型。Sentence Transformers 除了支持生成文本 Embedding 外，还支持 Cross-Encoder 模型进行精细化重排序（Reranking），以及 Sparse Encoder 模型生成稀疏向量表示（Sparse Embedding），因此广泛应用于语义检索（Semantic Search）、文本相似度计算（Semantic Textual Similarity）、同义句挖掘（Paraphrase Mining）、推荐系统以及问答系统等任务。

目前，Hugging Face 已提供超过 1.5 万个可直接使用的 Sentence Transformers 预训练模型，其中包含大量在 Massive Text Embeddings Benchmark（MTEB）排行榜上表现优异的文本表示模型。此外，该框架还提供了完整的训练与微调工具，支持用户针对具体业务场景训练或微调 Embedding 模型、Reranker 模型以及 Sparse Encoder 模型。Sentence Transformers 的 Github 地址为：`https://github.com/huggingface/sentence-transformers`，对应的 Hugging Face 模型库地址为：`https://huggingface.co/sentence-transformers`。用户既可以直接下载预训练模型进行推理，也可以利用自身数据进行微调训练，以适应不同业务场景和数据分布。

Sentence Transformers 的整体功能框架如图 10.26所示，主要涵盖以下几个方面：

1. **文本向量生成（Text Embedding Generation）**：支持生成高质量的句向量（Sentence Embedding），广泛应用于语义检索、文本相似度计算、推荐召回等任务。支持的模型主要包括以下几类：（1）预训练语言模型（PLM Encoder），如 BERT、RoBERTa [34]、MiniLM [55]、MPNet [47]、DeBERTa [19] 等；（2）Sentence Embedding 模型，如 SBERT、SimCSE 等；（3）检索模型（Retrieval Embedding），如 E5、BGE [56]、GTE [33]、NV-Embed [30]、Nomic-Embed[39] 等；（4）基于大语言模型的 Embedding 模型，如 e5-Mistral [53]、gte-Qwen2、Qwen3-Embedding [62]、Llama-Embedding [1]、Gemma-Embedding [51] 等；（5）多语言 Embedding 模型，如 mBERT、XLM-R [10]、LaBSE [13] 等。
2. **精细化重排序（Reranking）**：支持利用 Cross-Encoder 模型对候选文本进行重新排序，以进一步提升检索结果的相关性和准确性。典型模型包括 bge-reranker、bge-reranker-v2、e5-reranker、MiniLM-reranker、MonoT5 [38, 42] 等。
3. **稀疏向量表示（Sparse Embedding）**：支持利用 Sparse Encoder 生成稀疏向量表示，在大规模信息检索中能够与稠密向量形成互补。典型模型包括 SPLADE [14]、SparseEncoder、SparseEncoder-v2 等。
4. **训练与微调（Training and Fine-tuning）**：提供统一的训练框架，支持 Embedding、Reranker 以及 Sparse Encoder 模型的训练、微调与模型评估。
5. **模型生态（Model Zoo）**：依托 Hugging Face 丰富的模型生态，用户可以方便地下载、部署以及微调各类文

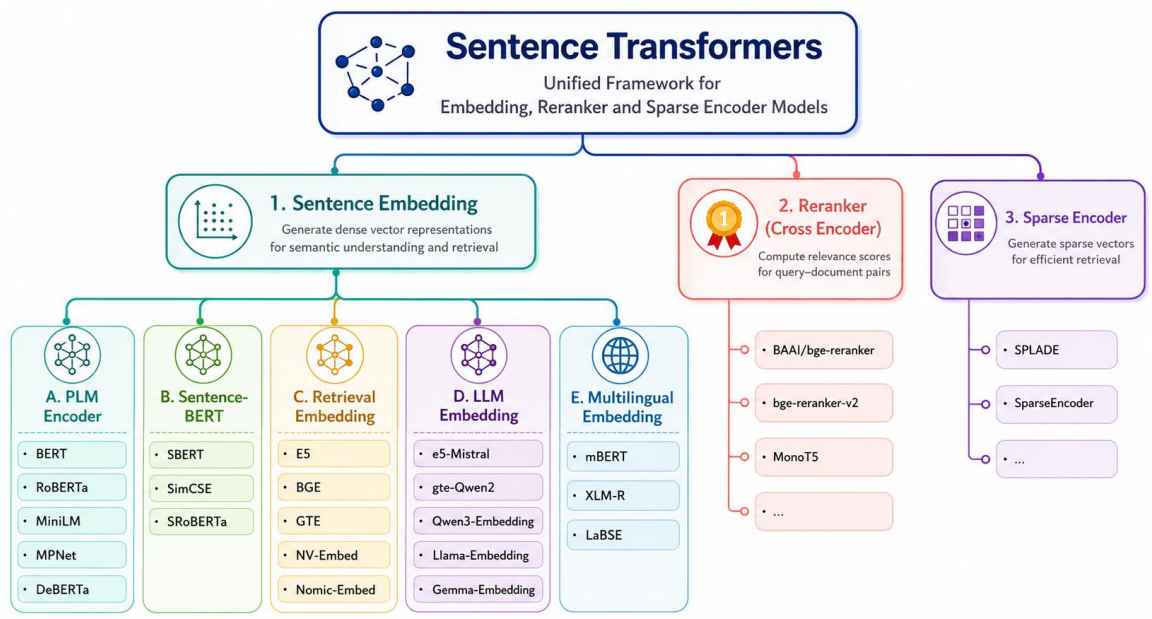

图 10.26: Sentence Transformers 库涵盖内容示意图

本表示模型，并持续获得社区最新发布的模型与算法。

Sentence Transformers 库是当前语义 Embedding 与文本检索领域最重要的开源框架之一，它为研究人员和工程师提供了统一而灵活的开发接口，使得文本向量生成、相似度计算、重排序以及模型训练等工作能够更加高效地完成。借助该框架，用户可以快速构建高质量的文本表示模型，并广泛应用于推荐系统、搜索引擎、问答系统以及 RAG 等场景，实现更加精准的语义匹配与信息检索。

近年来，文本表示模型经历了从传统预训练语言模型，到 Sentence Embedding 模型，再到检索模型（Retrieval Embedding）以及基于大语言模型的 Embedding 模型的快速发展。Sentence Transformers 对这些主流模型进行了统一封装，使研究人员能够在同一框架下方便地使用、训练和微调不同类型的文本表示模型，为工业界语义检索系统的研发提供了完善的基础设施。

本节对 Sentence Transformers 库的主要功能与应用场景进行了概述，其目的并非介绍某一个具体模型，而是希望帮助读者从整体上理解当前文本表示技术的发展脉络，以及现代语义 Embedding 框架的整体生态。与此同时，也希望读者能够进一步理解这些语义表示技术是如何与工业级推荐系统中的 Embedding 召回、向量检索（ANN）等技术相结合，共同构建现代语义召回系统。在实际应用中，读者可以根据具体业务需求选择合适的文本表示模型，并结合向量检索框架构建高效、准确的语义召回系统。

10.4 序列召回

在前面的章节中，我们介绍了推荐系统中最常见的三类召回模型，即基于协同过滤（Collaborative Filtering）的召回模型、基于图（Graph）的召回模型以及基于 Embedding 的向量召回模型。这些方法大多关注用户与物品之间的静态关联关系，例如用户是否点击过某个物品、两个物品是否具有较高的相似度，或者用户与物品是否位于相近的向量空间中。然而，这类方法存在以下问题：

定义 10.15 (静态兴趣假设问题)

基于协同过滤的召回模型、基于图的召回模型以及基于 Embedding 的向量召回模型通常假设用户兴趣是相对稳定的，并没有充分考虑用户兴趣随时间不断演化这一事实。

事实上，在工业级推荐系统中，用户兴趣往往具有**明显的动态变化特性**。例如，用户今天浏览手机，明天可能开始关注手机壳和耳机；用户刚购买了一本机器学习教材，后续更可能浏览深度学习、强化学习等相关书籍，

而不是再次购买相同的教材。在我们日常使用电商软件的过程中，我们也会表现出类似的兴趣动态。比如，我们今天下单了一台笔记本电脑，如果过两天电商 APP 仍然给我们推荐电脑，那我们也不会购买了。因此，在工业级推荐系统中如果只利用用户的静态兴趣来进行召回，往往难以准确捕捉到用户当前的真实需求。

为了刻画用户兴趣的动态演化过程，工业界逐渐提出了序列召回（Sequential Retrieval）的思想。序列召回不仅关注用户点击了哪些物品，更关注这些物品出现的先后顺序以及兴趣演变过程，通过分析用户历史行为序列，学习用户的短期兴趣与长期兴趣，从而实现更加精准的个性化推荐。目前，序列召回已经广泛应用于短视频推荐、新闻推荐、电商推荐、音乐推荐等场景，也是现代推荐系统区别于传统协同过滤方法的重要发展方向。

10.4.1 List2Vec 召回

List2Vec 并不是某一个具体模型，而是一类基于用户行为列表（List）的表示学习方法，它的核心思想是：

定义 10.16 (List2Vec 召回)

List2Vec 召回的核心思想是将用户历史行为列表看作一种特殊的“文本序列”，利用自然语言处理中词向量学习（Word Embedding）的思想，学习每个物品的低维向量表示（Item Embedding），再基于这些向量完成基于向量相似度的相似物品召回。

与传统基于文本语义 Embedding 的召回模型不同，List2Vec 召回模型通常更加关注用户行为本身所蕴含的协同过滤信号。它并不依赖商品标题、商品描述等内容特征，而是直接根据大量用户的历史交互行为学习物品表示。因此，即使两个商品没有任何文本上的相似性，只要它们经常被相同用户共同浏览、点击或购买，也能够学习得到相近的向量表示。这种思想实际上属于协同过滤与表示学习相结合的一种典型方法。

List2Vec 是一种较为宽泛的方法范式，其中最经典、应用最广泛的方法便是 Item2Vec。

10.4.1.1 Item2Vec 模型

在 List2Vec 中，一个最核心的问题就是如何学习每个物品的向量表示。

笔记 List2Vec 召回中，每个物品的 Embedding 应该如何学习？

在前面基于预训练语义模型的向量召回章节中，我们介绍了 FastText、BERT、Sentence-BERT、SimCSE、E5、BGE 等文本表示模型。这些模型能够根据商品标题、商品描述等文本信息学习物品的语义表示。然而，仅利用文本语义往往无法充分反映用户真实的兴趣偏好。例如，两部电影在文本描述上可能完全不同，但却经常被相同用户连续观看；两件商品虽然没有任何共同关键词，却经常被用户一起购买。这种由用户行为产生的协同过滤信号，是纯文本模型难以学习到的。

因此，Item2Vec 模型提出了直接利用用户行为数据学习 Item Embedding 的思想。Item2Vec 模型来源于 2016 年发表的论文《Item2Vec: Neural Item Embedding for Collaborative Filtering》[2]。该方法借鉴了自然语言处理中 Word2Vec 的 Skip-Gram with Negative Sampling（SGNS）模型，将用户行为数据转换为类似文本语料的形式，从而学习物品之间的低维向量表示。

与 Word2Vec 模型不同的是，Item2Vec 模型并没有利用行为发生的时间顺序。Item2Vec 模型的原论文将每位用户的一次行为列表看作一个无序集合（Set 或 Basket），只要两个物品出现在同一个集合中，就认为它们具有一定的关联关系，而不关心它们出现的先后顺序。因此，Item2Vec 召回模型本质上学习的是物品之间的共现关系（Co-occurrence），而不是严格意义上的时序依赖关系。例如，一位用户的一次浏览行为可以表示为：

$$\{i_1, i_2, i_3, i_4\},$$

Item2Vec 模型将认为集合中的任意两个物品均构成一个正样本，例如：

$$(i_1, i_2),\ (i_1, i_3),\ (i_1, i_4),\ (i_2, i_3),\ \cdots$$

这些物品对共同参与模型训练，使模型学习到它们之间的潜在关联关系。相比于 Word2Vec 模型中固定大小的滑

动窗口，Item2Vec 模型将整个行为集合看作一个动态窗口。假设用户行为集合为：

$$S = \{i_1, i_2, \cdots, i_K\} \tag{10.194}$$

则集合中任意满足 $i_a \neq i_b$ 的物品对均可作为训练样本。论文中对应的优化目标可表示为：

$$\mathcal{L} = -\frac{1}{K}\sum_{i=1}^{K}\sum_{j\neq i}^{K}\log p(i_j|i_i) \tag{10.195}$$

其中，K 表示用户行为集合中的物品数量，$p(i_j|i_i)$ 采用 Skip-Gram with Negative Sampling（SGNS）进行优化，训练过程与 Word2Vec 模型基本一致，仅将“单词（Word）”替换为“物品（Item）”即可。

在 Item2Vec 模型训练完成后，每个物品都会对应一个低维稠密向量（Item Embedding）。由于经常共同出现在同一用户行为中的物品具有相似的兴趣属性，因此这些物品在向量空间中也会更加接近。例如，经常浏览《复仇者联盟》的用户通常也会浏览《钢铁侠》、《美国队长》、《蜘蛛侠》等电影，因此这些电影最终会聚集到向量空间中的相邻区域。

需要指出的是，虽然 Item2Vec 模型经常被归类为序列召回方法，但它实际上并未利用用户行为的时间顺序。例如，对于行为序列 $A \to B \to C$ 和 $C \to B \to A$，Item2Vec 模型得到的训练样本完全相同，因此无法刻画用户兴趣随时间变化的动态过程。这一局限性也推动了后续工业界利用 RNN 等时序建模方法的序列推荐模型的发展，实现更加精准的序列召回。

10.4.1.2 List2Vec 召回算法原理

Item2Vec 模型学习得到每个物品的 Embedding 之后，就可以进一步构建基于行为序列的召回系统。工业界通常将这一类利用用户历史行为列表（List）作为触发源（Trigger），结合物品 Embedding 完成相似物品检索的方法统称为 List2Vec 召回。

List2Vec 召回的整体思想十分简单：首先利用大量用户行为数据训练得到高质量的物品 Embedding，然后在线根据用户最近发生的一系列行为作为查询（Query），分别检索与这些行为最相似的物品，最后将多个触发源的召回结果进行融合，形成最终的候选集合。与传统的 ItemCF 召回不同，ItemCF 召回通常依赖于显式构建物品-物品相似度矩阵，而 List2Vec 召回则将物品表示为低维稠密向量，通过向量空间中的距离刻画物品之间的相似关系，因此能够更好地学习高阶语义关系，并且更加容易扩展到海量物品场景。

List2Vec 召回流程整体上可以划分为离线训练与在线召回两个阶段。离线阶段首先收集用户历史行为日志，并根据业务需求筛选高质量行为。例如，仅保留点击后具有较长观看时长、至少完成一次完整播放的视频，过滤快速划走、误点击等低质量行为，从而降低噪声样本对 Embedding 学习的影响。随后，将每位用户的历史行为按照时间顺序组织为一个行为列表：

$$L_u = (i_1, i_2, \cdots, i_n)$$

其中，i_k 表示用户交互过的第 k 个物品。尽管 Item2Vec 模型本身不利用行为顺序进行训练，但工业界通常仍然保留时间顺序，便于后续训练其他序列模型以及进行统一的数据管理。

接着，以所有用户行为列表作为训练语料，利用 FastText 或者 Item2Vec 模型训练物品 Embedding。由于每天都会产生新的用户行为，因此工业系统通常采用天级或小时级增量训练方式，不断更新物品 Embedding，以保证模型能够及时反映最新的内容分布和用户兴趣变化。

在训练完成后，每个物品均对应一个固定长度的向量：

$$i \to \mathbf{e}_i \in \mathbb{R}^d \tag{10.196}$$

其中，d 表示 Embedding 的维度，通常取值为 64、128 或 256。随后将所有 Embedding 写入 Hive 或特征存储，并同步至向量检索系统，用于在线 ANN 检索。

List2Vec 召回的整体流程可总结如代码 10.18所示。

代码 10.18: List2Vec 召回算法流程（伪代码）

```
############################################################
# 离线训练阶段
############################################################

# Step1. 从Hive表加载最新的用户行为日志
behavior_logs = load_behavior_logs(date = yesterday, user_filter = "old_user", exposure > 200, ↩
    ↪ author_fans > 200)

# Step2. 过滤带有噪声行为的数据
# 过滤短播播放行为，过滤负反馈行为，保留有效交互行为
behavior_logs = filter_behavior(remove_short_view = True, remove_hate = True, keep = {like, ↩
    ↪ comment, follow, forward, long_view})

# Step3. 构造用户行为列表
# 示例:
# User A -> [Item1 Item5 Item9 Item7 ...]
training_sequences = build_sequence(behavior_logs, sort_by = timestamp)

# Step4. 加载物品侧属性特征（可选）
photo_features = load_photo_features()

# Step5. 热启动加载上一版本的物品Embedding作为预训练向量
# 复用前一天的Embedding作为初始化
pretrained_embedding = load_previous_embedding()

# Step6. 训练 Item2Vec / FastText 模型
item_embedding = FastText.train(corpus = training_sequences, feature = photo_features, pretrained ↩
    ↪ = pretrained_embedding, dim = 128, window = 7, negative = 5, epoch = 1)

# Step7. 保存Embedding
save_to_hive(item_embedding)

# Step8. 将Embedding发布到在线存储
publish_to_online(item_embedding)

# Step9. 构建（或增量更新）ANN索引
ANN.build(item_embedding)

############################################################
# 在线检索阶段
############################################################
# Step10. 收集多种触发行为
trigger_lists = [LikeList(limit=5), FollowList(limit=5), ForwardList(limit=5), LongViewList(limit↩
    ↪ =10), EffectiveViewList(limit=20)]

# Step11. ANN检索
candidate_set = []
for trigger in trigger_lists:
    for item in trigger:
```

```
        embedding = item_embedding[item]
        # 召回Top-30相似物品
        neighbors = ANN.search(embedding, topk = 30)
        candidate_set.extend(neighbors)

# Step12. 合并多Trigger的召回结果
candidate_set = deduplicate(candidate_set)

# Step13. 截断候选集规模
candidate_set = topN(candidate_set, N = 800)
return candidate_set
```

10.4.1.3 List2Vec 召回算法工程实现

工业界的 List2Vec 召回通常采用离线训练、在线检索的两阶段式的架构。离线部分主要负责训练物品 Embedding。系统每天定时从 Hive 表中抽取最近一天或最近数天的用户行为日志，经过行为过滤、样本清洗以及序列构建之后，利用 FastText 模型或 Item2Vec 等模型对所有物品进行增量训练。

为了充分利用历史模型，工业系统通常采用热启动（Warm Start）训练的方式，将上一版本的物品 Embedding 作为预训练向量（Pretrained Vectors），只对新增物品以及已有物品进行持续优化，从而提高训练效率并保证 Embedding 空间的连续性。在训练完成后，训练任务会将最新的物品 Embedding 同步至在线特征系统（如 Redis 缓存、Kafka 消息队列等），随后构建 ANN 索引或增量更新向量索引，使线上检索系统能够及时使用最新模型。

在线部分则主要负责 ANN 检索。系统首先根据用户最近发生的重要行为构建多个召回 Trigger，不同 Trigger 对应不同的行为类型。例如：

- 最近点赞列表（Latest Like List）；
- 最近关注列表（Latest Follow List）；
- 最近转发列表（Latest Forward List）；
- 最近长观看列表（Latest Long View List）；
- 最近有效观看列表（Latest Effective View List）。

为了控制在线计算开销，不同 Trigger 通常会限制最大长度，例如最近点赞视频保留最近 5 个，长观看视频保留最近 10 个，有效观看视频保留最近 20 个。对于每个 Trigger Item，ANN 系统分别检索固定数量的相似物品（例如 Top-30），最终将所有 Trigger 的召回结果融合、去重，并截断为固定规模（例如 500～1000 个候选），即得到 List2Vec 召回的召回候选集。

整个工程流程可以概括为：

行为日志 → FastText/Item2Vec 模型训练 → 物品 Embedding → ANN 索引 → 多 Trigger 检索 → 候选融合

需要指出的是，上述工程实现仅是工业界 List2Vec 召回的一种典型方案。不同公司在训练周期、行为过滤策略、Trigger 类型的选择以及 ANN 检索框架等方面可能存在一定差异，但整体架构基本一致，即通过离线学习高质量的物品 Embedding，再结合在线 ANN 检索完成高效的 I2I 序列召回。

10.4.2 RNN 序列召回

随着 Embedding 召回技术的发展，研究者逐渐意识到，仅利用平均 Pooling 对用户历史行为进行聚合，难以准确刻画用户兴趣随时间变化的动态过程。例如，对于用户先后浏览“篮球鞋”、“篮球比赛”、“NBA 精彩集锦”等内容，其最近行为往往比较早发生的行为更能反映当前兴趣，而平均 Pooling 无法区分行为发生的先后顺序，也无法建模行为之间的时序依赖关系。

为了解决这一问题，研究者开始将循环神经网络（Recurrent Neural Network，RNN）引入 Embedding 召回

中，利用 RNN 对用户历史行为序列进行建模，从而学习更加准确的用户兴趣表示。其中，以 GRU4Rec 为代表的 RNN 序列推荐模型首次证明了循环神经网络能够有效建模用户兴趣演化过程，并推动了序列推荐（Sequential Recommendation）的快速发展。

与 YouTubeDNN 相比，RNN 序列召回并没有改变 Embedding 召回的整体框架，而是将用户侧 Encoder 由平均 Pooling 替换为 GRU 或 LSTM 等循环神经网络。如图 10.27所示，用户历史行为序列首先经过 Embedding 层映射为低维向量，然后依次输入 RNN 网络，模型按照时间顺序逐步更新隐藏状态（Hidden State），最终利用最后一个隐藏状态作为当前用户兴趣表示（User Embedding），并与物品 Embedding 进行相似度计算，实现候选物品召回。

相比于平均 Pooling，RNN 能够显式建模用户行为之间的时间依赖关系，使模型更加关注最近发生的重要行为，从而更准确地反映用户兴趣的动态变化。因此，RNN 序列召回在新闻推荐、短视频推荐以及电商推荐等具有明显时序特征的场景中取得了较好的效果。

然而，由于 RNN 采用递归计算方式，模型必须按照时间顺序逐步处理整个行为序列，因此无法像 Transformer 一样实现并行计算。当用户行为序列较长时，RNN 不仅计算效率较低，而且容易出现长期依赖建模能力不足的问题。因此，近年来工业界逐渐采用 Transformer 替代 RNN 作为用户行为序列编码器，RNN 序列召回也逐渐演变为 Transformer 序列召回。

10.4.2.1 GRU4Rec 模型

GRU4Rec（Gated Recurrent Unit for Recommendation）是 Hidasi 等人在论文《Session-based Recommendations with Recurrent Neural Networks》[22] 中提出的一种经典序列推荐模型，也是最早将循环神经网络应用于推荐系统的重要工作之一。该模型采用门控循环单元（Gated Recurrent Unit，GRU）对用户行为序列进行建模，通过学习用户兴趣随时间的动态演化过程，提高下一兴趣预测的准确性。

需要指出的是，GRU4Rec 最初提出时主要面向 Session-based Recommendation 任务，即根据用户当前 Session 中的历史点击行为预测下一次点击物品，而并非专门针对 Embedding 召回设计。然而，其利用 GRU 学习用户兴趣表示的思想，很快被工业界应用到 Embedding 召回模型中，成为早期序列召回的重要代表方法。

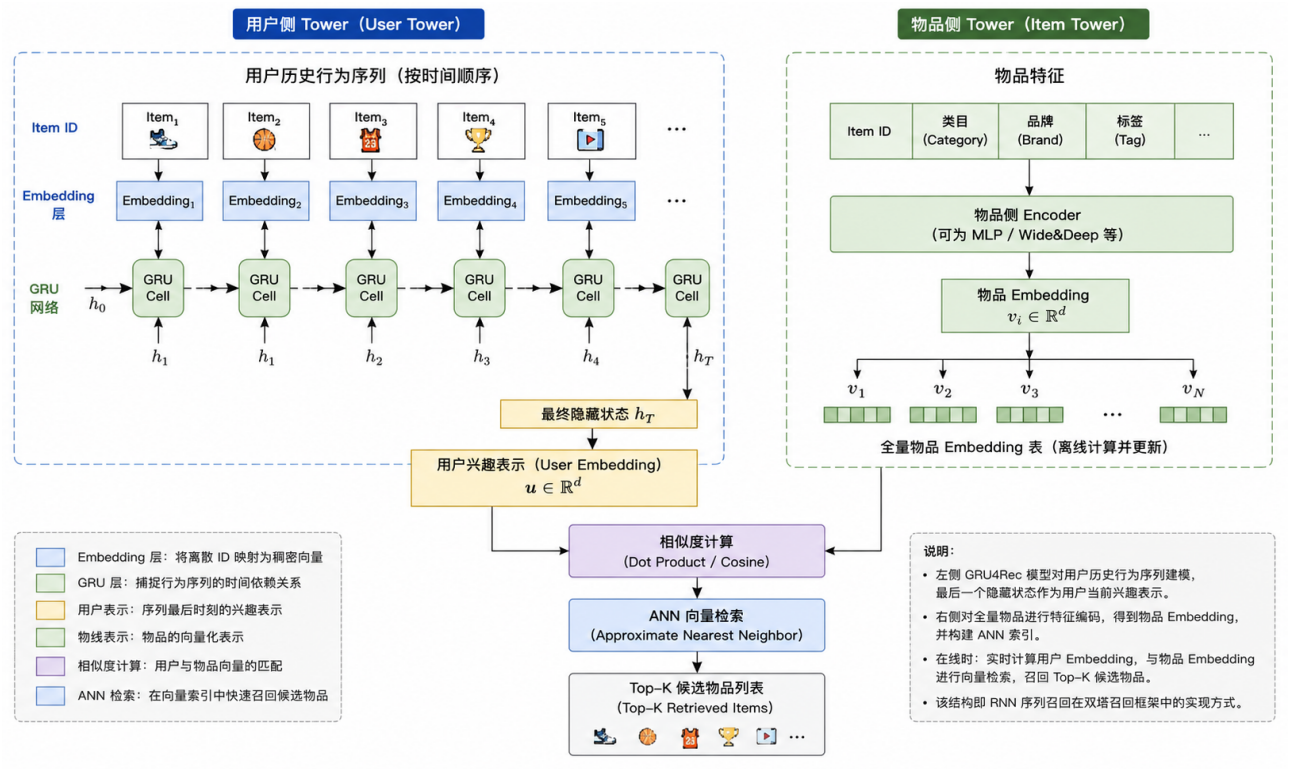

图 10.27: GRU4Rec 在召回模型中的应用示意图。

GRU4Rec 模型整体结构如图 10.27所示。对于用户行为序列：

$$S = [i_1, i_2, \cdots, i_T] \tag{10.197}$$

首先，每个物品经过 Embedding 层映射为连续向量：

$$\mathbf{e}_t = \text{Embedding}(i_t) \tag{10.198}$$

随后，将物品的 Embedding 序列依次输入 GRU 网络：

$$\mathbf{h}_t = \text{GRU}(\mathbf{e}_t, \mathbf{h}_{t-1}) \tag{10.199}$$

其中，$\mathbf{h}_t$ 表示第 t 个时间步对应的隐藏状态。随着用户行为不断输入，GRU 能够不断更新用户兴趣表示，使隐藏状态能够综合反映用户历史行为信息。

经过整个行为序列编码后，通常采用最后一个隐藏状态作为当前用户兴趣表示：

$$\mathbf{u} = \mathbf{h}_T \tag{10.200}$$

在 Embedding 召回场景中，用户 Embedding 进一步与候选物品 Embedding 计算向量相似度：

$$\text{score}(\mathbf{u}, \mathbf{v}) = \mathbf{u}^T \mathbf{v} \tag{10.201}$$

其中，$\mathbf{v}$ 表示物品 Embedding。训练阶段通常采用 Sampled Softmax、BPR Loss 等目标函数优化模型；在线阶段则利用用户 Embedding 作为查询向量，在 ANN 索引中快速检索 Top-K 候选物品。相比于 YouTubeDNN 采用平均 Pooling 聚合用户行为，GRU4Rec 能够充分利用行为发生的时间顺序，对用户兴趣演化进行动态建模，因此在序列推荐任务中取得了显著优于传统 Pooling 方法的效果。

不过，GRU4Rec 仍然存在一定局限性。一方面，GRU 采用递归计算方式，训练和推理过程无法充分利用 GPU 并行计算能力；另一方面，当用户行为序列较长时，模型容易出现长期依赖建模不足的问题。因此，随着 Transformer 模型的发展，越来越多的工业系统开始采用自注意力机制替代 GRU，序列召回模型也逐渐由 RNN 演进到 Transformer 架构。

10.4.2.2 RNN 序列召回工程实现

RNN 序列召回的整体工程实现与前文介绍的双塔召回基本一致，其主要区别在于用户侧 Encoder 采用 GRU 等循环神经网络对用户行为序列进行建模，而物品侧通常仍采用轻量级 Encoder 学习物品 Embedding。在离线训练阶段，系统首先从 Hive 等离线数仓或 Kafka 等实时消息队列中抽取用户行为日志，经过行为过滤、序列构建以及样本生成等预处理后，利用 GRU4Rec 模型训练用户兴趣表示。模型训练完成后，最新的模型 Checkpoint 会发布至在线推理服务，用于后续用户 Embedding 的实时计算。与此同时，物品侧 Encoder 通常采用离线批处理方式，对全量物品执行前向推理，生成对应的物品 Embedding，并将其导入 Embedding Server 构建 ANN 向量索引。若物品侧采用神经网络 Encoder 而非简单的 ID Embedding，则通常由离线 Runner 服务周期性地对全量物品执行前向计算，生成最新的顶层 Embedding，再同步更新至 Embedding Server，从而保证在线召回能够使用最新的物品表示进行向量检索。

在线阶段，当用户发起推荐请求时，推荐系统首先获取用户最近一段时间的行为序列，将其输入 GRU 网络，实时计算当前用户的 Embedding；随后，以用户 Embedding 作为查询向量，在 ANN 索引中快速检索出 Top-K 候选物品。整个流程可以表示为：

$$\text{User Behavior Sequence} \rightarrow \text{GRU Encoder} \rightarrow \text{User Embedding} \rightarrow \text{ANN Retrieval} \rightarrow \text{Top-}K\ Items$$

总体来看，RNN 序列召回可以理解为 YouTubeDNN 的一种自然演进，其主要贡献在于利用循环神经网络替代平均 Pooling，更好地建模用户兴趣随时间的动态变化。然而，由于 RNN 难以并行计算且长序列建模能力有限，目前工业界已逐渐采用 Transformer 作为新的用户行为编码器，形成了当前主流的 Transformer 序列召回框架。RNN 序列召回也因此成为连接传统 Embedding 召回与 Transformer 序列召回的重要过渡阶段，在序列推荐的发展过程中具有重要的历史意义。

10.4.3 Transformer 序列召回

随着 Transformer 模型在自然语言处理领域取得巨大成功，越来越多的研究工作开始尝试将其应用于推荐系统中的用户兴趣建模任务。近年来，以 SASRec、BERT4Rec 为代表的一系列 Transformer 序列推荐模型相继提出，使得 Transformer 逐渐成为序列推荐（Sequential Recommendation）领域最主流的技术路线之一，并进一步推动了 Embedding 召回模型的发展。

回顾 Embedding 召回的发展历程可以发现，不同模型之间最大的区别并不在于召回框架，而在于用户兴趣表示（User Embedding）的建模方式。正如前文介绍的 YouTubeDNN 模型，其用户侧采用平均 Pooling（Average Pooling）对历史行为 Embedding 进行聚合，然后通过多层感知机（MLP）学习用户 Embedding。这种方法结构简单、计算效率高，但由于平均 Pooling 忽略了行为之间的顺序关系，因此难以准确刻画用户兴趣随时间变化的动态演化过程。为了更好地建模用户行为序列，GRU4Rec 等方法进一步将循环神经网络（Recurrent Neural Network, RNN）引入用户侧 Encoder，利用 GRU 或 LSTM 对用户历史行为进行逐步编码，从而能够捕捉行为之间的时序依赖关系。然而，RNN 本身采用递归计算方式，模型必须按照时间顺序逐步处理整个行为序列，因此训练和推理过程难以并行。此外，当用户行为序列较长时，RNN 仍然容易出现长期依赖建模能力不足的问题，较早发生的行为信息可能随着时间推移逐渐衰减，从而影响最终用户兴趣表示的准确性。

Transformer 模型的提出为上述问题提供了一种新的解决思路。与 RNN 逐步更新隐藏状态不同，Transformer 利用自注意力机制（Self-Attention）直接建立序列中任意两个行为之间的关联关系，从而能够同时关注用户历史中的多个重要行为，并根据不同历史行为对当前兴趣预测的重要程度动态分配注意力权重。因此，Transformer 不仅能够更好地捕捉长距离依赖关系，而且能够充分利用 GPU 实现整个行为序列的并行计算，大幅提高模型训练效率。

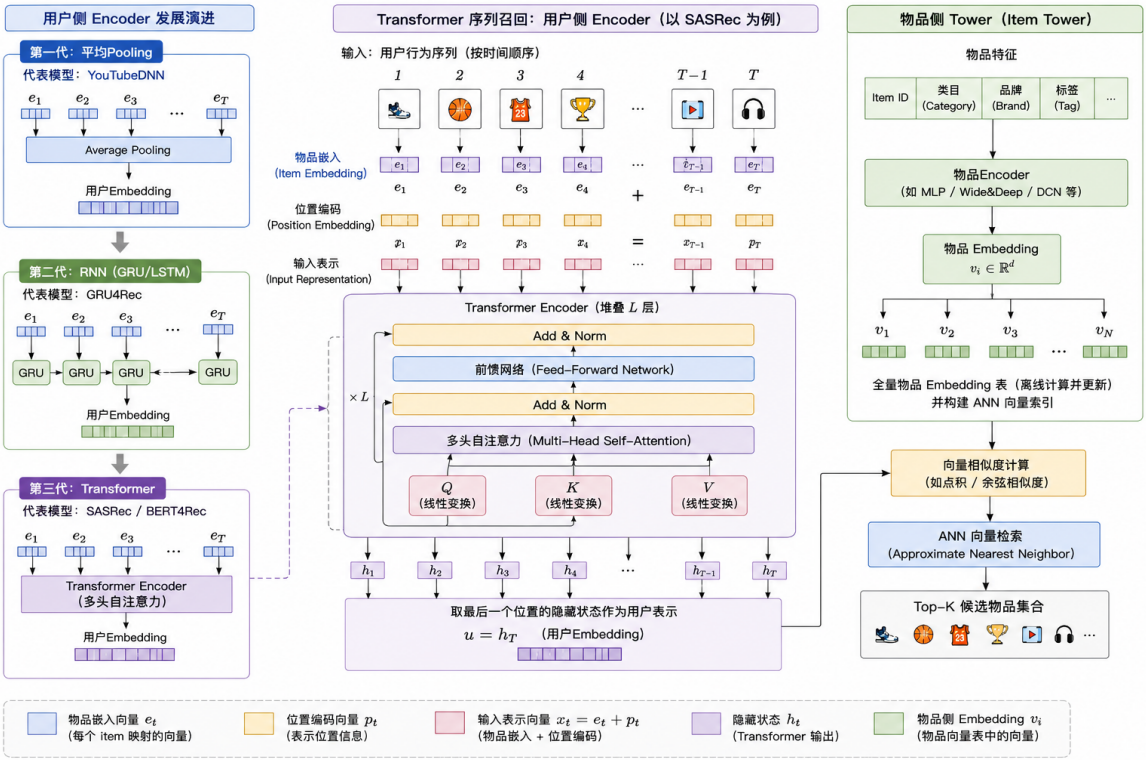

图 10.28: Transformer 序列召回与用户侧 Encoder 发展演进。

如图 10.28所示，Transformer 序列召回并不是一种全新的召回架构，而是在传统 Embedding 召回框架基础上的进一步升级。无论是 YouTubeDNN、GRU4Rec，还是后续的 SASRec、BERT4Rec，其整体框架通常仍然采用双塔（Two-Tower）或者双编码器（Dual Encoder）的设计思想，即分别学习用户 Embedding 和物品 Embedding，并通过向量相似度完成候选物品召回。Transformer 主要应用于用户侧 Encoder 网络，用于替代传统模型中的平

均 Pooling 或 RNN 网络，对用户历史行为序列进行更加精细的建模；而物品侧 Encoder 通常仍然采用 MLP 等轻量级网络对物品 ID、类别、文本、图片等特征进行编码，生成对应的物品 Embedding。最终，推荐系统通过计算用户 Embedding 与物品 Embedding 之间的相似度，并结合 ANN 索引完成大规模候选物品检索。从这一角度来看，Transformer 序列召回实际上可以理解为 YouTubeDNN 模型的一种自然演进，其核心变化并不是整个召回框架，而是用户兴趣建模方式的升级。对于用户侧 Encoder，其发展过程经历了平均 Pooling、RNN 以及 Transformer 三个阶段。随着兴趣建模能力不断增强，模型能够学习更加丰富的用户行为模式，从而进一步提升召回结果的准确率。

近年来，工业界大规模推荐系统普遍采用 Transformer 作为用户行为序列编码器。一方面，Transformer 能够有效融合用户长期兴趣与短期兴趣，提高用户 Embedding 的表达能力；另一方面，其良好的并行计算能力也更加符合工业推荐系统对于大规模训练和在线推理效率的要求。因此，目前主流的序列召回模型几乎都建立在 Transformer 架构之上，其中最具代表性的工作包括基于单向自注意力机制的 SASRec（Self-Attentive Sequential Recommendation）以及基于双向 Transformer 的 BERT4Rec（Bidirectional Encoder Representations for Sequential Recommendation）。下面分别对这两种经典模型进行介绍，并进一步说明 Transformer 模型如何与双塔召回架构相结合，实现工业级序列召回。

10.4.3.1 SASRec 模型

SASRec（Self-Attentive Sequential Recommendation）是 2018 年发表的论文《Self-Attentive Sequential Recommendation》[25] 中提出的一种基于 Transformer 自注意力机制的序列推荐模型，也是 Transformer 首次在序列推荐领域取得成功的代表性工作。SASRec 将自然语言处理中 Transformer 的思想引入用户行为序列建模，通过自注意力机制学习用户历史行为之间的依赖关系，从而获得更加准确的用户兴趣表示。与 GRU4Rec 等基于循环神经网络的模型相比，SASRec 模型与它们最大的区别在于其不再采用递归方式逐步处理用户行为序列，而是利用 Self-Attention 直接计算序列中任意两个行为之间的相关性。因此，模型既能够捕捉长距离兴趣依赖，又能够充分利用 GPU 实现并行计算，大幅提升模型训练效率。

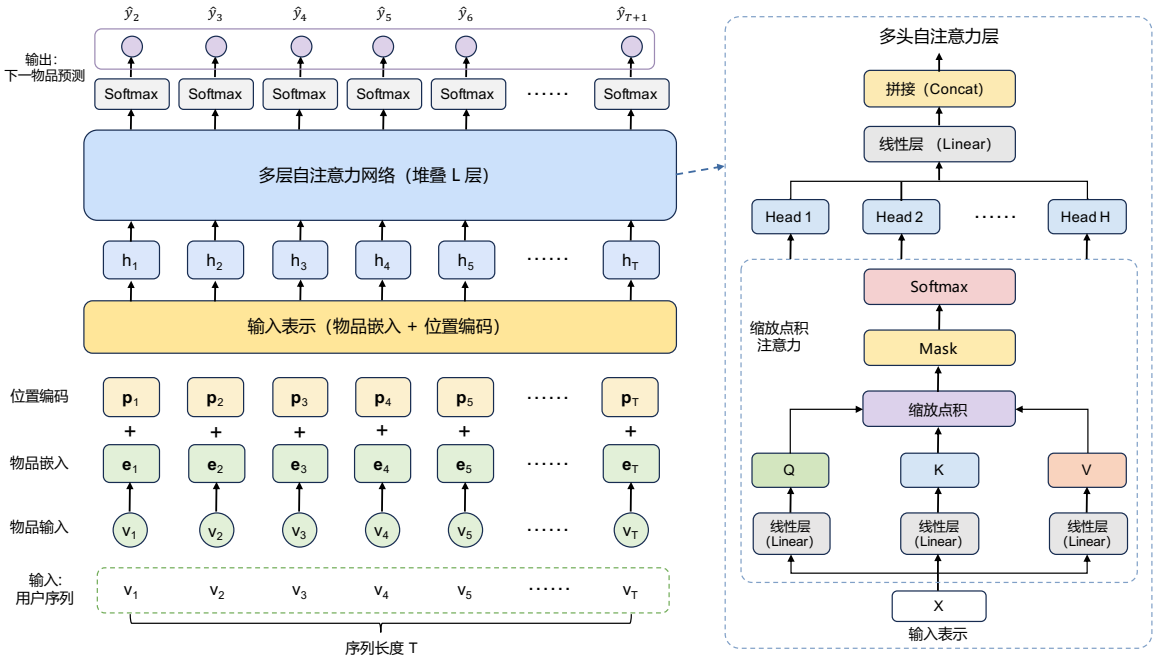

图 10.29: SASRec 模型示意图。

SASRec 模型的整体网络结构如图 10.29所示。首先，SASRec 模型将用户历史行为序列表示为：

$$S = [i_1, i_2, \cdots, i_T] \tag{10.202}$$

其中，i_t 表示用户在第 t 个时间步发生交互的物品。随后，每个 Item 首先经过 Embedding 层映射为低维向量：

$$\mathbf{e}_t = \text{Embedding}(i_t) \tag{10.203}$$

由于 Transformer 本身无法感知序列顺序，因此需要进一步加入位置编码（Positional Embedding）：

$$\mathbf{x}_t = \mathbf{e}_t + \mathbf{p}_t \tag{10.204}$$

其中 $\mathbf{p}_t$ 表示第 t 个位置对应的位置向量。再然后，整个行为序列输入多层 Transformer Encoder 进行编码。对于序列中的每一个位置，Self-Attention 都会计算其与其它历史行为之间的注意力权重：

$$Attention(Q, K, V) = Softmax\left(\frac{QK^T}{\sqrt{d}}\right)V \tag{10.205}$$

其中，Q、K 和 V 分别表示 Query、Key 和 Value 矩阵。通过这种方式，模型能够自动学习哪些历史行为对当前兴趣预测更加重要，并动态调整不同历史行为的贡献程度。

由于推荐任务本质上属于自回归预测（Next Item Prediction），因此 SASRec 采用了因果掩码（Causal Mask）机制，使当前位置只能关注之前已经发生的行为，而不能访问未来信息，从而保证训练过程与在线推理保持一致。经过多层 Transformer Encoder 之后，可以得到每一个位置对应的隐藏表示：

$$H = [\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_T] \tag{10.206}$$

其中最后一个位置对应的隐藏状态 $\mathbf{h}_T$ 通常作为整个用户当前兴趣表示（User Embedding）：

$$\mathbf{u} = \mathbf{h}_T \tag{10.207}$$

随后，利用用户 Embedding 与候选物品 Embedding 计算相似度：

$$score(\mathbf{u}, i) = \mathbf{u}^T \mathbf{e}_i \tag{10.208}$$

其中 $\mathbf{e}_i$ 表示候选物品 Embedding。在训练阶段，通常采用 Sampled Softmax、BPR Loss 或 InfoNCE 等目标函数学习用户与物品 Embedding；在线阶段，则利用用户 Embedding 作为查询向量，在 ANN 索引中快速检索 Top-K 相似物品，实现高效的 Embedding 召回。

相比于 YouTubeDNN 和 GRU4Rec，SASRec 具有以下几个优点：

- **能够建模长距离兴趣依赖。** Self-Attention 可以直接建立任意两个历史行为之间的联系，对于长行为序列具有更好的建模能力。
- **训练效率更高。** Transformer 能够并行处理整个行为序列，而 RNN 只能逐步计算，因此 GPU 利用率更高，更适合大规模工业训练。
- **兴趣关注更加灵活。** 不同于 RNN 只能依赖最终隐藏状态传递信息，Self-Attention 能够根据当前预测目标动态关注不同历史行为，更容易捕捉用户兴趣变化。
- **易于扩展。** Transformer 能够方便地融合时间特征、位置特征、多模态特征以及上下文信息，因此具有较好的模型扩展能力。

需要指出的是，SASRec 模型最初提出的任务属于序列推荐（Sequential Recommendation），即根据用户历史行为预测下一次交互物品，其输出通常为整个 Item 集合上的分类概率。而在工业级推荐系统中，SASRec 模型更多作为 Embedding 召回模型中的 User Encoder 使用，通过 Transformer 学习用户 Embedding，再结合双塔模型与 ANN 向量检索完成大规模候选召回。因此，目前工业界实际部署的 SASRec 模型通常会对原始模型进行适当修改，使其能够更好地适配 Embedding 召回框架。这是工业界在应用序列推荐模型时与学术界研究序列推荐的一个主要区别。下一节将会进一步介绍另外一种经典 Transformer 序列推荐模型，即 BERT4Rec，并分析其与 SASRec 在训练方式和兴趣建模上的区别。

10.4.3.2 BERT4Rec 模型

BERT4Rec（Bidirectional Encoder Representations from Transformer for Recommendation）是 2019 年发表的论文《BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer》[48] 提出的一种基于双向 Transformer 的序列推荐模型。与 SASRec 采用单向自回归建模不同，BERT4Rec 借鉴了自然语言

处理领域 BERT 模型的思想，将 Masked Language Model（MLM）训练目标引入推荐系统，通过随机遮蔽（Mask）用户历史行为中的部分物品，并利用上下文信息恢复被遮蔽的物品，从而学习更加丰富的用户兴趣表示。

BERT4Rec 模型的网络结构以及与 SASRec、RNN 模型的对比如图 10.30所示。

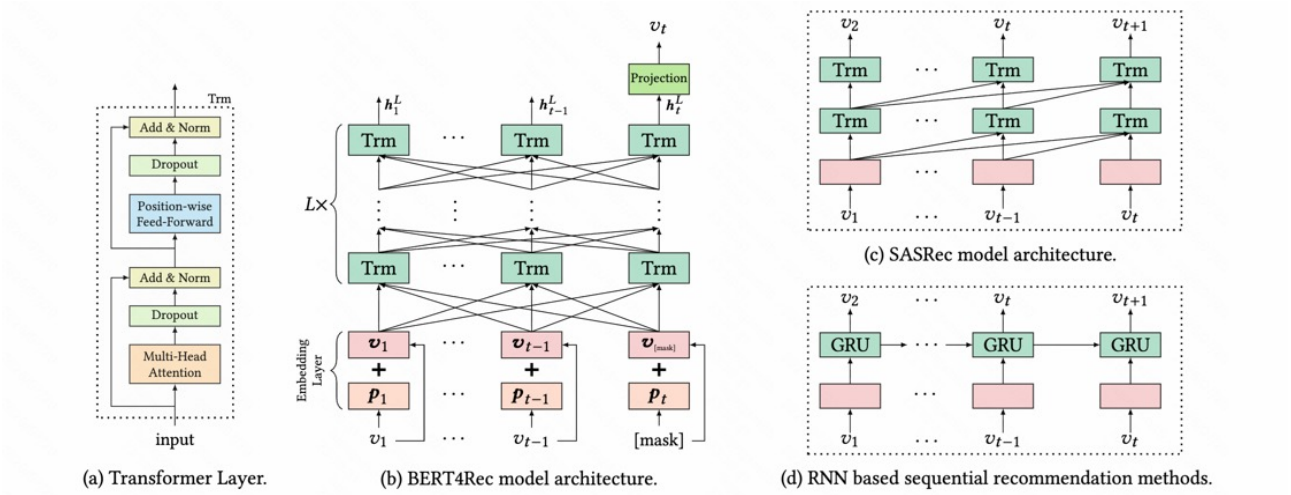

Figure 1: Differences in sequential recommendation model architectures. BERT4Rec learns a bidirectional model via Cloze task, while SASRec and RNN based methods are all left-to-right unidirectional model which predict next item sequentially.

图 10.30: BERT4Rec 模型结构示意图。

从图中可以看出，BERT4Rec 模型整体仍采用 Transformer Encoder 作为用户行为序列编码器，其输入由物品 Embedding 和位置 Embedding 组成。与 SASRec 模型采用单向序列建模不同，BERT4Rec 模型在训练过程中会随机 Mask 序列中的部分 Item，并利用其前后文信息恢复被 Mask 的位置，从而实现双向的序列建模。相比之下，RNN 模型仅使用物品 Embedding，而未显式引入位置 Embedding，因此在刻画行为顺序信息方面存在一定局限性。此外，RNN 模型通过 GRU 或 LSTM 对历史行为进行逐步编码，而 SASRec 和 BERT4Rec 模型则利用自注意力机制直接建模序列中任意两个行为之间的关联关系，因此能够更有效地捕捉用户兴趣的长距离依赖。

需要强调的是，SASRec 和 BERT4Rec 模型最大的区别并不在于网络结构，而在于模型的训练方式。

结论 SASRec 模型采用因果掩码（**Causal Mask**），当前位置只能关注历史行为，因此属于单向序列建模（Unidirectional Modeling）；而 BERT4Rec 模型采用随机 **Mask** 机制，模型能够同时利用当前位置前后的行为信息，因此属于双向序列建模（Bidirectional Modeling）。

假设用户历史行为序列表示为：

$$S = [i_1, i_2, \cdots, i_T] \tag{10.209}$$

在训练过程中，系统会随机选择序列中的若干个 Item 进行 Mask，例如：

$$[i_1, i_2, i_3, i_4, i_5] \Longrightarrow [i_1, [MASK], i_3, [MASK], i_5] \tag{10.210}$$

随后，将加入 Mask 后的行为序列输入 Transformer Encoder，模型根据其余行为恢复被 Mask 的位置。例如预测目标可以表示为：

$$P(i_t | S_{\backslash t}) \tag{10.211}$$

其中，$S_{\backslash t}$ 表示去除第 t 个行为后的上下文信息。由于 Transformer 能够同时关注目标位置前后的历史行为，因此相比于单向建模，BERT4Rec 能够学习更加完整的用户兴趣表示。

BERT4Rec 整体仍然采用 Transformer Encoder 作为用户行为序列编码器，其输入包括物品 Embedding 和位置 Embedding：

$$\mathbf{x}_t = \mathbf{e}_t + \mathbf{p}_t, \tag{10.212}$$

其中，$\mathbf{e}_t$ 表示物品 Embedding，$\mathbf{p}_t$ 表示位置 Embedding。经过多层 Transformer Encoder 编码后，可以得到整个行为序列对应的隐藏表示：

$$H = [\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_T] \tag{10.213}$$

对于被 Mask 的位置，模型利用对应位置的隐藏向量预测真实 Item，并采用交叉熵损失进行优化：

$$L = -\sum_{t \in \mathcal{M}} \log P(i_t | S_{\backslash t}) \tag{10.214}$$

其中 $\mathcal{M}$ 表示被 Mask 的位置集合。

由于 BERT4Rec 能够同时利用用户行为序列的前后文信息，因此通常能够学习到更加丰富的行为依赖关系，尤其是在行为序列较长时，其推荐效果通常优于单向序列建模方法。此外，由于所有位置均可参与监督训练，一个序列能够产生更多训练样本，因此模型具有更高的数据利用效率。

不过，BERT4Rec 模型也存在一定局限性。首先，其训练目标来源于 Masked Prediction，而在线推荐任务通常需要预测用户未来行为，两者之间存在一定差异。其次，由于 BERT4Rec 模型需要同时关注整个行为序列，其计算复杂度仍然为 $O(n^2)$，当用户行为序列较长时，会带来较大的计算和存储开销。因此，在工业界大规模推荐系统中完全使用原始 BERT4Rec 模型进行召回的案例仍然较少。

笔记 最后，需要强调的是，与 SASRec 模型一样，BERT4Rec 模型最初提出时属于序列推荐模型，而不是专门面向召回阶段设计的 Embedding 召回模型。因此在工业界应用这些序列推荐相关的模型时通常都是将其作为用户行为序列编码器使用，结合双塔模型与 ANN 索引完成大规模候选召回。

10.4.3.3 Transformer 序列召回工程实现

前文介绍的 SASRec 和 BERT4Rec 本质上都属于序列推荐（Sequential Recommendation）模型，其原始任务通常是根据用户历史行为预测下一次交互物品。然而，在工业级推荐系统中，这些模型更多地作为 Embedding 召回模型中的用户兴趣编码器（User Encoder）使用，而不是直接作为分类模型预测所有 Item 的概率。因此，在实际部署过程中，通常需要对原始模型进行一定改造，使其能够融入双塔召回（Two-Tower Retrieval）架构，并结合向量检索（ANN）完成大规模候选召回。

Transformer 序列召回的核心思想与 YouTubeDNN 保持一致，即分别学习用户 Embedding 和物品 Embedding，再利用向量相似度完成候选检索。不同之处在于，Transformer 替代了传统双塔模型中的用户侧 Encoder，使模型能够更加准确地建模用户历史行为序列。Transformer 序列召回的整个召回流程仍然是包括用户行为编码、物品编码、双塔训练以及在线向量召回四个阶段。

10.4.3.3.1 用户行为编码 用户侧 Encoder 负责学习用户兴趣表示。首先，将用户最近一段时间的行为序列按照时间顺序组织为：

$$S = [i_1, i_2, \cdots, i_T] \tag{10.215}$$

其中，i_t 表示第 t 次交互的物品。随后，每个物品首先经过 Embedding 层映射到连续向量空间，并加入位置 Embedding：

$$\mathbf{X} = \mathbf{E} + \mathbf{P}, \tag{10.216}$$

其中，$\mathbf{E}$ 表示物品 Embedding 矩阵，$\mathbf{P}$ 表示位置 Embedding。之后，将整个行为序列输入多层 Transformer Encoder 进行编码：

$$\mathbf{H} = \text{Transformer}(\mathbf{X}) \tag{10.217}$$

得到每一个行为对应的隐藏表示：

$$\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_T] \tag{10.218}$$

工业界通常采用最后一个位置对应的隐藏向量作为当前用户兴趣表示 $\mathbf{u} = \mathbf{h}_T$。对于 BERT4Rec 等双向模型，也可以采用平均 Pooling、Attention Pooling 或者 CLS Token 作为最终的用户侧 Embedding，不同方案主要影响用户兴趣聚合方式，而整体召回流程保持一致。

10.4.3.3.2 物品编码 与用户侧相比，物品侧通常不需要采用复杂的 Transformer 结构，而是使用轻量级编码器学习物品 Embedding。对于每个物品，其输入特征通常包括：物品 ID、类别、品牌、文本特征、图片特征、多模态 Embedding 等。这些特征经过 Embedding 层以及 MLP 编码后得到物品 Embedding：

$$\mathbf{v} = f_{item}(Item) \tag{10.219}$$

其中 f_{item} 表示物品侧编码器。由于 Item 数量通常达到千万甚至亿级，因此物品 Embedding 一般离线计算，并建立 ANN 索引，避免在线重复计算。

10.4.3.3.3 双塔模型训练 在训练阶段，用户塔输出用户 Embedding，物品塔输出物品 Embedding，二者通过向量相似度计算匹配程度：

$$score(u, v) = \mathbf{u}^T \mathbf{v} \tag{10.220}$$

随后结合正负样本进行训练。工业界通常采用 Sampled Softmax、InfoNCE 以及 BPR Loss 等目标函数，其中以 Sampled Softmax 最为常见。训练完成后，模型能够将兴趣相似的用户与物品映射到同一 Embedding 空间。

需要注意的是，Transformer 主要负责学习更加准确的用户 Embedding，而整个双塔训练框架并没有发生变化。因此，从工程角度来看，Transformer 序列召回可以理解为 YouTubeDNN 的一种升级版本，其核心区别仅在于用户兴趣编码方式。

10.4.3.3.4 在线 ANN 召回 模型训练完成后，所有物品 Embedding 会离线构建向量索引，例如 Faiss、ScaNN 或 HNSW 等 ANN 检索库。在线请求到来时，推荐系统首先获取用户最近的行为序列 $S = [i_1, i_2, \cdots, i_T]$，然后利用 Transformer Encoder 实时计算当前用户 Embedding：

$$\mathbf{u} = \text{Transformer}(S) \tag{10.221}$$

随后，将用户 Embedding 作为查询向量，在 ANN 索引中快速检索 Top-K 最相似的物品：

$$TopK = \text{ANN}(\mathbf{u}) \tag{10.222}$$

最终得到数百至数千个候选物品组成召回候选集。整个在线流程无需遍历全部物品，因此能够在毫秒级完成大规模候选召回。

10.4.3.3.5 工业实现优化 由于 Transformer 计算复杂度为 $O(n^2)$，直接部署标准 Transformer 仍然具有较大的推理开销。因此，工业界通常会结合一系列优化策略降低在线计算成本，包括：

- 限制用户行为序列长度，例如保留最近 50～200 条行为；
- 采用轻量级 Transformer Encoder，减少网络层数和 Attention Head 数量。例如将层数减少到 2～4 层，Attention Head 数量减少到 2～4 个；
- 离线缓存用户 Embedding，仅对新增行为进行增量更新；
- 利用模型蒸馏、量化等技术降低模型参数规模；
- 结合 GPU Batch 推理和 ANN 向量检索，提高整体在线吞吐能力。

总体而言，Transformer 序列召回并没有改变 Embedding 召回的整体框架，而是在双塔模型中利用 Transformer 替代传统 Pooling 或 RNN 作为用户兴趣编码器，从而获得更加准确的用户 Embedding 表示。近年来，随着 Transformer 模型不断发展，工业界越来越多的 Embedding 召回模型都采用 Transformer 作为默认的用户编码器，使其逐渐成为当前序列召回的主流技术路线。

10.4.4 SDM 召回

SDM（Sequential Deep Matching，序列深度匹配）召回最早由阿里巴巴于 2019 年提出，源自论文《SDM: Sequential Deep Matching Model for Online Large-scale Recommender System》[35]。SDM 是在 YoutubeDNN 和双塔召回基础上的一种序列建模召回模型，其核心思想如下：

定义 10.17

SDM 召回通过引入短期兴趣建模模块和长期兴趣建模模块，分别学习用户短期兴趣（Short-term Interest）和长期稳定兴趣（Long-term Interest），并利用门控融合机制自适应地结合两种兴趣表示，从而获得更加准确的用户 Embedding，最终结合 ANN 向量检索完成大规模候选召回。

10.4.4.1 SDM 召回算法原理

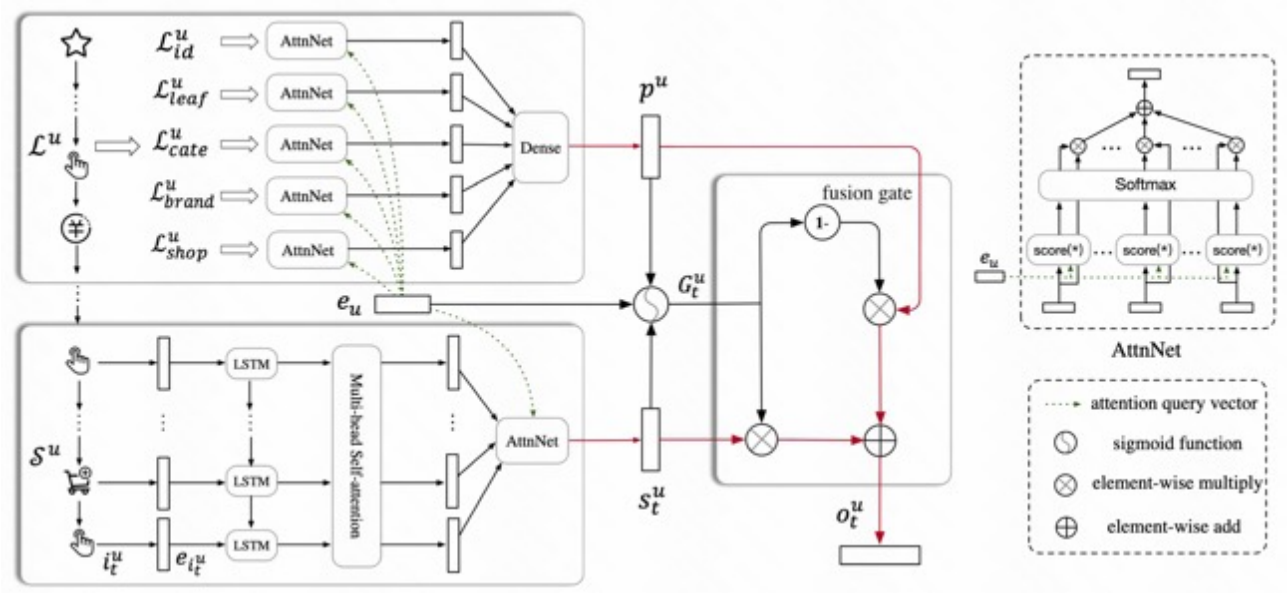

图 10.31: SDM 召回模型结构示意图。

SDM 整体网络结构如图 10.31所示。模型主要由输入 Embedding 层、短期兴趣建模模块、长期兴趣建模模块、兴趣融合模块以及 Embedding Matching 模块组成。其中，用户最近一次 Session 中的行为序列作为短期兴趣序列，首先经过 Embedding 层映射为连续向量表示，再利用 LSTM 对用户行为序列进行编码，以捕获行为之间的时序依赖关系：

$$\mathbf{h}_t = \text{LSTM}(\mathbf{e}_t, \mathbf{h}_{t-1}) \tag{10.223}$$

其中 $\mathbf{e}_t$ 表示第 t 个行为对应的 Embedding，$\mathbf{h}_t$ 表示 LSTM 输出的隐藏状态。由于用户在一次 Session 中可能存在无关点击或兴趣漂移，仅利用 LSTM 最后一个隐藏状态难以准确刻画用户当前兴趣。因此，SDM 进一步引入多头自注意力（Multi-head Self-Attention）机制，对整个行为序列进行重新建模：

$$\hat{\mathbf{H}} = \text{MultiHead}(\mathbf{H}) \tag{10.224}$$

其中 $\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_T]$ 表示 LSTM 输出序列。随后，以用户侧特征拼接之后的 Embedding 作为查询向量，对行为序列进行 Attention Pooling，从而得到当前时刻的短期兴趣表示：

$$\mathbf{s} = \sum_{t=1}^{T} \alpha_t \hat{\mathbf{h}}_t, \qquad \alpha_t = \frac{\exp(\hat{\mathbf{h}}_t^T \mathbf{e}_u)}{\sum_j \exp(\hat{\mathbf{h}}_j^T \mathbf{e}_u)} \tag{10.225}$$

其中 $\mathbf{e}_u$ 表示用户画像 Embedding。

除了短期兴趣之外，SDM 召回还引入了长期兴趣建模模块。模型认为用户长期兴趣通常体现在多个不同的特征维度，例如商品 ID、叶子类目（Leaf Category）、一级类目（Category）、品牌（Brand）以及店铺（Shop）等。因此，首先分别构建各个特征维度对应的长期行为集合，例如过去一段时间内（比如 7 天内）浏览或购买过的商品集合、品牌集合、店铺集合等。对于每一种特征维度 f，模型首先将集合中的所有特征映射为 Embedding，然后以用户特征拼接之后的 Embedding 作为查询，对该集合进行 Attention Pooling，从而得到当前特征维度对应的

长期兴趣表示：

$$\alpha_k = \frac{\exp(\mathbf{g}_k^T \mathbf{e}_u)}{\sum_j \exp(\mathbf{g}_j^T \mathbf{e}_u)}, \qquad \mathbf{z}_f = \sum_k \alpha_k \mathbf{g}_k \tag{10.226}$$

其中 $\mathbf{e}_u$ 表示用户画像 Embedding，$\mathbf{g}_k$ 表示第 k 个长期行为对应的 Embedding，$\mathbf{z}_f$ 表示特征维度 f 对应的长期兴趣表示。随后，将不同特征维度得到的长期兴趣表示进行拼接，并通过全连接网络融合，得到最终的长期兴趣 Embedding：

$$\mathbf{z} = \text{Concat}(\mathbf{z}_{id}, \mathbf{z}_{leaf}, \mathbf{z}_{cate}, \mathbf{z}_{brand}, \mathbf{z}_{shop}) \tag{10.227}$$

$$\mathbf{p} = \tanh(\mathbf{W}_p \mathbf{z} + \mathbf{b}) \tag{10.228}$$

其中 $\mathbf{p}$ 表示最终学习得到的长期兴趣表示。

为了充分结合短期兴趣和长期兴趣，SDM 进一步设计了门控融合（Gated Fusion）结构，根据用户当前状态自适应决定两种兴趣表示的贡献比例：

$$\mathbf{G} = \sigma\left(\mathbf{W}_1 \mathbf{e}_u + \mathbf{W}_2 \mathbf{s} + \mathbf{W}_3 \mathbf{p} + b\right) \tag{10.229}$$

其中 σ 表示 Sigmoid 激活函数，$\mathbf{W}_1$、$\mathbf{W}_2$、$\mathbf{W}_3$ 和 b 表示可学习参数，$\mathbf{G}$ 表示门控向量，其每个元素的取值范围为 $[0, 1]$，用于控制短期兴趣和长期兴趣在最终用户 Embedding 中的权重分配。最终可以得到用户 Embedding 如下：

$$\mathbf{o} = (1 - \mathbf{G}) \odot \mathbf{p} + \mathbf{G} \odot \mathbf{s} \tag{10.230}$$

其中，$\odot$ 表示逐元素乘法。当用户当前行为较为活跃时，模型会更加关注短期兴趣；而对于长期稳定兴趣，则会赋予长期兴趣更大的权重，从而实现兴趣的动态融合。

10.4.4.2 SDM 召回模型训练与部署

SDM 召回本质上仍属于 Embedding 召回模型，其训练目标与 YoutubeDNN 和双塔召回类似。模型采用采样 Softmax（Sampled Softmax）进行训练，将用户下一次点击商品作为正样本，同时随机采样若干负样本，利用交叉熵损失优化用户 Embedding 与物品 Embedding 之间的匹配关系：

$$\mathcal{L} = -\sum_i y_i \log(\hat{y}_i) \tag{10.231}$$

其中 $\hat{y}_i$ 表示 Softmax 预测概率。由于全量物品规模通常达到百万甚至千万级，因此训练阶段通常仅采样少量负样本参与 Softmax 计算，以降低训练成本。

SDM 召回采用离线训练、在线实时推理的部署方式，与 YoutubeDNN 和双塔召回整体流程基本一致。离线阶段完成模型训练，并构建物品 Embedding 对应的 ANN 向量索引；在线阶段，根据用户最新行为序列实时计算用户 Embedding，再利用 Faiss、HNSW 或 ScaNN 等 ANN 检索系统快速召回 Top-K 候选物品，即为 SDM 召回的最终候选集。

相比起 YoutubeDNN 主要采用平均 Pooling 建模用户兴趣，SDM 召回通过 LSTM、自注意力以及门控融合机制分别建模短期兴趣和长期兴趣，使用户 Embedding 能够更加准确地反映用户当前兴趣状态，因此在用户兴趣变化较快的推荐场景中通常能够取得更好的召回效果。同时，SDM 召回整体仍然保持 Embedding 召回与 ANN 检索的工程架构，因此具有较好的在线部署效率，目前已成为工业界序列召回模型的重要代表方法之一。

10.4.5 对比学习序列召回

近年来，随着对比学习（Contrastive Learning）在计算机视觉和自然语言处理等领域取得成功，其也逐渐被应用到推荐系统中，并成为序列召回模型的重要发展方向之一。与传统序列召回模型主要依赖监督信号学习用户兴趣表示不同，对比学习通过构造同一样本的不同视图（View），使模型在无需额外标注信息的情况下学习更加鲁棒、更具判别能力的 Embedding 表示，从而进一步提升召回性能。

对于序列推荐任务而言，用户历史行为序列本身包含丰富的监督信息。对比学习序列召回通常首先对同一用户的行为序列进行不同方式的数据增强，生成两个不同的序列视图，并将其作为正样本对；而来自其它用户的行为序列则作为负样本。训练过程中，通过最大化同一用户不同视图之间的相似度，同时最小化不同用户行为序列之间的相似度，使模型学习到更加稳定且具有区分能力的用户兴趣表示。

与传统 Sampled Softmax、Pointwise Loss 以及 Pairwise Loss 等监督学习目标相比，对比学习并不是用于替代下一物品预测任务，而是作为一种辅助训练目标，引导 Embedding 空间学习更加合理的结构。因此，目前大多数对比学习序列召回模型都会采用多任务联合训练方式，同时优化下一物品预测损失和对比学习损失，其整体目标可以表示为：

$$\mathcal{L} = \mathcal{L}_{rec} + \lambda \mathcal{L}_{cl} \tag{10.232}$$

其中，$\mathcal{L}_{rec}$ 表示传统的序列推荐损失（如 Sampled Softmax），$\mathcal{L}_{cl}$ 表示对比学习损失，λ 用于平衡两部分目标。

本节主要介绍近年来具有代表性的几种对比学习序列召回模型，包括 CL4SRec 与 ICSRec。这些方法均以序列表示学习为核心，通过设计不同的数据增强方式和对比学习策略，提高用户兴趣表示质量，从而进一步提升 Embedding 召回效果。

10.4.5.1 CL4SRec 召回

CL4SRec（Contrastive Learning for Sequential Recommendation）模型发表于 2021 年，源自于阿里巴巴团队发表的论文《Contrastive Learning for Sequential Recommendation》[57]。CL4SRec 模型是最早将对比学习系统性引入序列推荐领域的代表性工作之一。该方法以 SASRec 模型作为用户兴趣编码器，在传统下一物品预测任务的基础上增加自监督对比学习任务，使模型能够学习更加鲁棒的用户兴趣表示。

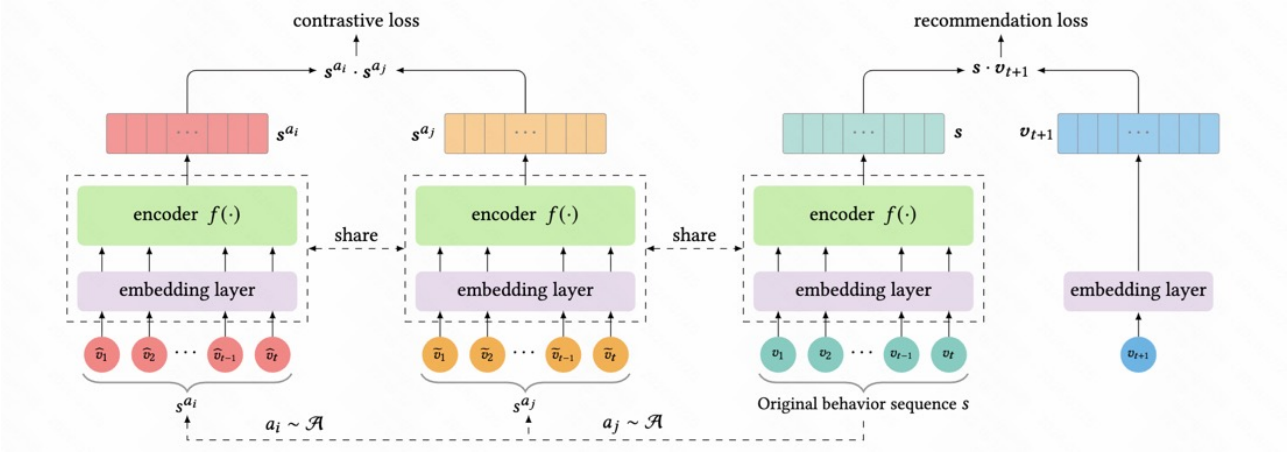

图 10.32: CL4SRec 召回模型结构示意图。

CL4SRec 召回模型的整体框架如图10.32所示。模型整体框架主要包括数据增强模块、序列编码器以及对比学习目标三个部分。首先，对每个用户历史行为序列随机进行两次数据增强，得到两个不同的序列视图，并将二者作为正样本对；同一个 Mini-Batch 中其它用户对应的增强序列则作为负样本。随后，将增强后的序列输入 Transformer 编码器得到对应的用户 Embedding，并利用 InfoNCE 损失函数拉近正样本之间的距离、拉远负样本之间的距离。

CL4SRec 召回提出了三种适用于序列推荐的数据增强方式，如图10.33所示。其中，**Item Crop（序列裁剪）**策略随机截取用户历史行为中的一段连续子序列，使模型能够学习局部兴趣表示，提高模型对于不同长度行为序列的泛化能力。**Item Mask（序列掩码）**策略随机将部分历史物品替换为 Mask 标记，模拟行为缺失情况，增强模型对噪声数据和缺失数据的鲁棒性。**Item Reorder（序列重排）**策略随机打乱局部连续子序列中物品的顺序，降低模型对严格顺序关系的依赖，提高兴趣表示的稳定性。

假设同一用户经过两种随机增强后的序列表示分别为 $\mathbf{h}_i$ 和 $\mathbf{h}_j$，则 CL4SRec 模型采用 InfoNCE 损失进行优

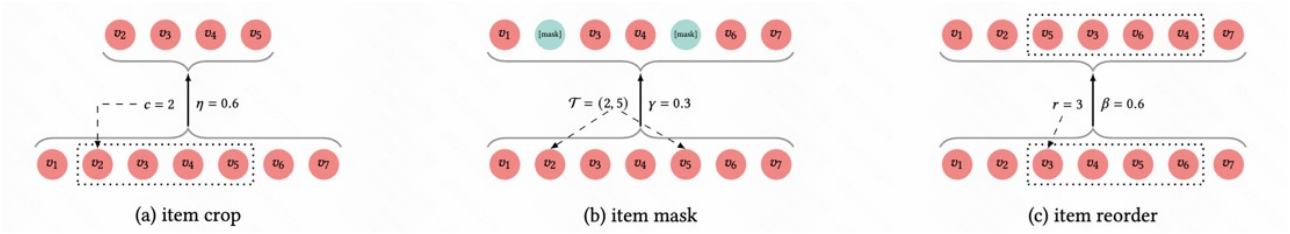

图 10.33: CL4SRec 数据增强方式示意图。

化：

$$\mathcal{L}_{cl} = -\log \frac{\exp(\text{sim}(\mathbf{h}_i, \mathbf{h}_j)/\tau)}{\sum_k \exp(\text{sim}(\mathbf{h}_i, \mathbf{h}_k)/\tau)} \tag{10.233}$$

其中，$\text{sim}(\cdot,\cdot)$ 通常采用向量点积或余弦相似度，τ 表示温度系数。为了保证推荐任务本身的性能，CL4SRec 模型采用多任务联合训练，同时优化下一物品预测任务与对比学习任务：

$$\mathcal{L}_{total} = \mathcal{L}_{rec} + \lambda \mathcal{L}_{cl} \tag{10.234}$$

其中，$\mathcal{L}_{rec}$ 通常采用 Sampled Softmax 损失预测用户下一次交互物品，λ 用于控制对比学习目标的重要程度。

CL4SRec 召回模型最大的贡献在于证明了序列推荐模型可以充分利用用户行为序列自身构造自监督学习信号，而无需引入额外标签信息。通过简单的数据增强和对比学习训练，模型能够学习更加稳定、更具判别能力的用户兴趣表示，在多个公开数据集上均取得了优于 SASRec 等传统序列召回模型的效果。

不过，CL4SRec 召回采用的数据增强方式主要依赖随机裁剪、随机掩码以及随机重排等通用策略，并未充分考虑推荐场景下用户兴趣变化规律，因此生成的增强序列有时可能改变原始兴趣语义。为了解决这一问题，后续研究开始探索更加符合推荐任务特点的数据增强方法，例如 ICSRec 召回进一步结合粗粒度兴趣与细粒度兴趣的对比学习来得到更加稳定的用户兴趣表示，从而进一步提升了对比学习序列召回模型的性能。

10.4.5.2 ICSRec 召回

ICSRec（Interest Contrastive Sequential Recommendation）模型是 WSDM 2024 提出的一种基于兴趣对比学习（Interest Contrastive Learning）的序列推荐模型，源自于论文《ICSRec: Interest Contrastive Learning for Sequential Recommendation》[43]。与 CL4SRec 召回模型主要依赖随机数据增强构造正样本不同，ICSRec 召回模型认为用户兴趣本身具有明确的语义结构，因此提出直接从用户行为序列中挖掘兴趣监督信号，通过兴趣级别的对比学习提升用户表示质量，从而进一步提高序列召回效果。

ICSRec 召回模型的整体框架如图 10.34所示，主要包括兴趣监督信号构建（Supervisory Signal Construction）、兴趣表示学习（Intent Representation Learning）以及下一物品预测（Next Item Prediction）三个部分。ICSRec 召回首先将用户行为序列划分为多个子序列，并根据下一点击目标（Target Item）构造兴趣监督信号；随后利用 Transformer 编码用户序列表示，并设计粗粒度兴趣对比学习和细粒度兴趣对比学习两个辅助任务，共同优化用户兴趣表示；最终结合序列预测任务进行联合训练。

首先，ICSRec 召回模型将每个用户行为序列划分为多个训练子序列，每个子序列均以其下一点击物品作为监督目标。具有相同目标物品的子序列被划分到同一个集合中，从而构建粗粒度兴趣监督信号。随后，ICSRec 召回仍然采用 SASRec 模型作为序列编码器的基座，然后对输入的用户行为序列进行编码，进而得到对应的兴趣表示：

$$\mathbf{H} = f_\theta(\mathbf{E}) \tag{10.235}$$

其中 $f_\theta(\cdot)$ 表示 Transformer 编码器，$\mathbf{E}$ 表示输入序列 Embedding，最后一个位置对应的隐藏状态作为当前序列的兴趣表示。

在此基础上，ICSRec 召回首先提出了粗粒度兴趣对比学习（Coarse-grained Intent Contrastive Learning，CICL）。模型首先将每条用户行为序列切分为多个“历史行为序列 + 下一目标物品”训练样本，即将用户历史交互序列作

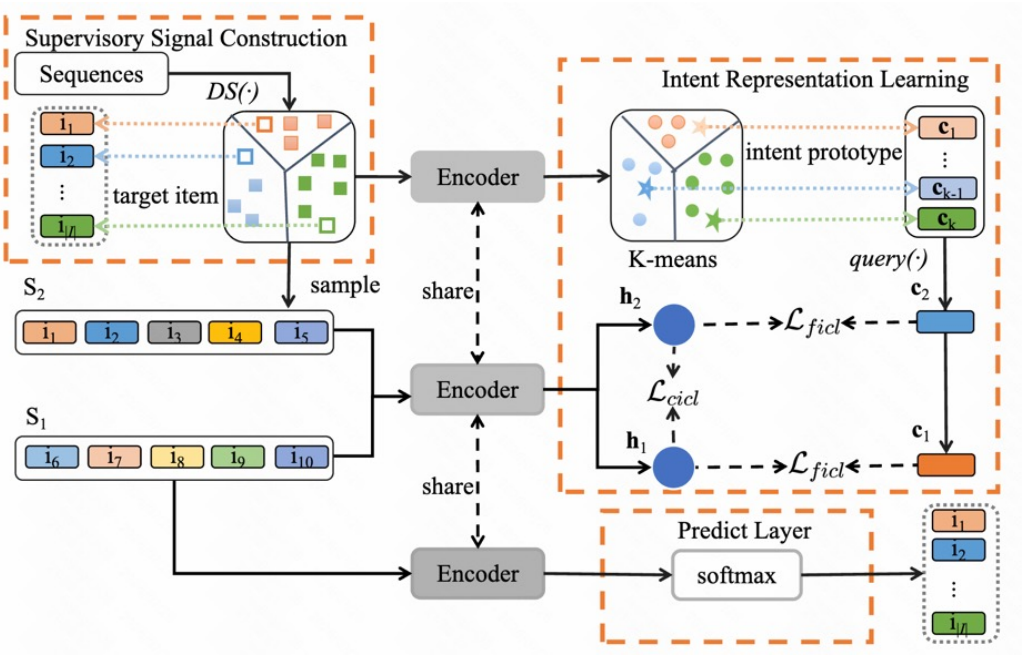

图 10.34: ICSRec 召回模型结构示意图。

为输入子序列，而该子序列对应的下一次交互物品作为目标物品（Target Item）。例如，对于序列 $\{A, B, C, D\}$，可以构造出 $\{A\} \rightarrow B$、$\{A, B\} \rightarrow C$ 以及 $\{A, B, C\} \rightarrow D$ 等多个训练样本。随后，ICSRec 召回将所有具有相同目标物品的子序列划分到同一个集合中。其基本假设是：虽然不同用户到达同一目标物品的行为路径可能不同，但这些子序列往往反映了相似的潜在购买意图或兴趣需求。因此，对于同一目标物品对应的两个子序列，模型将其兴趣表示作为正样本对，通过对比学习拉近二者在 Embedding 空间中的距离，而目标物品不同的子序列则作为负样本，使模型学习更加稳定且具有判别性的兴趣表示。因此，ICSRec 召回的粗粒度兴趣对比学习损失可以表示为：

$$\mathcal{L}_{\text{CICL}} = \mathcal{L}_{\text{Con}}(\mathbf{h}_1, \mathbf{h}_2) \tag{10.236}$$

其中 $\mathbf{h}_1$ 和 $\mathbf{h}_2$ 分别表示两个具有相同目标物品的子序列对应的兴趣表示，$\mathcal{L}_{\text{Con}}(\cdot)$ 采用基于 InfoNCE 的对比学习目标进行优化。这里需要注意，与传统 InfoNCE 不同，为了避免同一 Batch 内存在多个具有相同目标物品的子序列而被错误地视为负样本，ICSRec 模型进一步提出了 False Negative Mask（FNM）机制，对这些伪负样本进行屏蔽，从而提高对比学习的训练效果。

进一步地，ICSRec 召回提出了细粒度兴趣对比学习（Fine-grained Intent Contrastive Learning，FICL）。论文认为，即使最终交互的是同一个目标物品，不同用户的兴趣动机仍可能存在差异。例如，同一件商品可能既会被价格敏感型用户购买，也可能被品牌偏好型用户购买，仅依据目标物品构造的粗粒度监督仍然会将这两类兴趣混合在一起。为此，ICSRec 模型首先利用编码器得到所有子序列的兴趣表示，并采用 K-Means 聚类算法对这些兴趣表示进行聚类，将各聚类中心作为兴趣原型（Intent Prototype）。随后，对于每个子序列，ICSRec 模型根据最近邻搜索找到与其最相似的兴趣原型，并通过对比学习进一步拉近兴趣表示与对应兴趣原型之间的距离，从而学习更加细粒度的兴趣表示。其细粒度兴趣对比学习损失定义为：

$$\mathcal{L}_{\text{FICL}} = \mathcal{L}_{\text{Con}}(\mathbf{h}_1, \mathbf{c}_1) + \mathcal{L}_{\text{Con}}(\mathbf{h}_2, \mathbf{c}_2) \tag{10.237}$$

其中，$\mathbf{h}_1$ 和 $\mathbf{h}_2$ 表示两个子序列对应的兴趣表示，$\mathbf{c}_1$ 和 $\mathbf{c}_2$ 分别表示与其最近邻匹配得到的兴趣原型，$\mathcal{L}_{\text{Con}}(\cdot)$ 与粗粒度兴趣对比学习采用相同的对比学习目标进行优化。与 CICL 类似，为避免多个样本对应相同兴趣原型而产生伪负样本，FICL 同样采用 False Negative Mask 机制对这些负样本进行屏蔽，从而提高兴趣表示学习的稳定性。

最后，ICSRec 召回采用多任务联合优化方式，同时优化下一物品预测任务、粗粒度兴趣对比学习任务以及细粒度兴趣对比学习任务，其整体损失函数表示为：

$$\mathcal{L} = \mathcal{L}_{rec} + \lambda \mathcal{L}_{cicl} + \beta \mathcal{L}_{ficl} \tag{10.238}$$

其中，$\mathcal{L}_{rec}$ 表示序列预测损失，$\mathcal{L}_{cicl}$ 表示粗粒度兴趣对比学习损失，$\mathcal{L}_{ficl}$ 表示细粒度兴趣对比学习损失，λ 和 β 分别表示对应辅助任务的权重系数。

总体来看，相较于 CL4SRec 召回模型主要依赖随机数据增强构造正样本，ICSRec 召回进一步利用用户行为序列中蕴含的兴趣监督信号进行对比学习，并引入兴趣原型以及伪负样本屏蔽机制，使学习得到的用户兴趣表示更加稳定、更加具有判别能力，从而进一步提升了序列召回模型的性能。ICSRec 召回模型也表明，对比学习序列召回的发展方向正逐渐由基于随机数据增强的自监督学习，演进到结合兴趣建模和语义监督的信息增强学习，为后续兴趣表示学习的发展提供了新的思路。

虽然 ICSRec 召回模型在公开数据集上取得了较好的实验效果，但其在工业级推荐系统中的实际落地仍然面临一定挑战。首先，ICSRec 模型需要对用户行为序列进行划分，并构建兴趣原型进行聚类，这意味着训练过程中通常需要引入额外的离线聚类流程。由于聚类操作本身不可微，难以与模型训练进行端到端联合优化，因此工程实现和训练流程都会变得更加复杂。其次，兴趣原型需要持续维护和更新，在用户兴趣快速变化的大规模推荐场景下，将带来额外的存储和计算方面的开销。最后，ICSRec 模型采用了推荐目标、粗粒度兴趣对比学习以及细粒度兴趣对比学习等多任务联合优化方式，在提升模型表达能力的同时，也进一步增加了模型训练复杂度和超参数调节成本。因此，在实际工业部署过程中，通常需要结合具体业务场景和系统资源，对兴趣原型构建、聚类更新以及训练流程进行针对性的工程优化，以兼顾模型效果、训练效率以及在线部署成本。

10.4.5.3 对比学习序列召回落地讨论

尽管对比学习序列召回近年来取得了较好的研究进展，但在工业界实际落地过程中仍然面临诸多工程挑战。下面结合工业界实践，对对比学习序列召回在落地中的几个典型问题进行详细的讨论。首先，数据增强策略的选择是影响对比学习效果的关键因素。对比学习序列召回依赖于对用户行为序列构建不同的数据视图，因此不同的数据增强策略会直接影响模型学习到的用户表示质量。常见的数据增强方式包括随机裁剪（Cropping）、随机掩码（Masking）、序列重排（Reordering）以及语义一致性增强等。然而，在工业界实践中，这一问题其实并没有标准答案，不同业务场景对应的最优增强策略往往存在较大差异。很多时候只能设计多种增强方案，通过线上 A/B 实验不断验证其收益，并根据实验结果持续迭代优化。从某种意义上来说，这类问题最终仍然需要回归实验科学和经验科学，而不是依赖某一种理论上的最优方案。

其次，正负样本的构建方式直接决定了对比学习能够学习到怎样的 Embedding 空间。通常将同一用户行为序列经过不同数据增强得到的两个视图作为正样本对，而其他用户的行为序列作为负样本。负样本的数量以及采样策略都会直接影响模型的判别能力。如果负样本数量过少，或者负样本过于容易学习，模型很容易陷入局部最优，使学习得到的 Embedding 空间缺乏足够的区分能力。因此，在实际工程中通常需要设计更加合理的负采样策略，例如动态负采样（Dynamic Negative Sampling）、难负样本挖掘（Hard Negative Mining）等方法，不断提高负样本的训练难度，从而获得更好的线上效果。

此外，对比学习通常需要与推荐任务联合优化，因此多任务训练策略也是工业部署中的重要问题。在学术界的序列推荐研究中，通常采用下一物品预测（Next-item Prediction）作为推荐目标，并与对比学习损失共同训练。而在工业界推荐系统中，对比学习序列召回一般不会直接采用 Next-item Prediction 作为主要监督目标，而是仍然采用用户后验反馈行为对应的 Label 组合作为监督信号，例如点击、播放时长、点赞、关注等业务标签，引导用户 Embedding 的学习。这种训练方式能够更加贴近真实业务目标，因此在线上的 A/B 实验中通常也能够取得更加稳定的收益。

另一方面，对比学习在训练阶段通常需要计算大量样本之间的相似度，同时还需要进行负样本采样，因此训练开销明显高于传统召回模型。随着训练的 batch size 不断增大，相似度计算和负采样所带来的计算成本也会快速增加。因此，在工业部署过程中，需要综合考虑训练效率与模型效果之间的平衡，例如采用更加高效的负采样策略、优化相似度计算流程等方式，在有限的计算资源下尽可能获得更好的模型收益。

最后，冷启动问题仍然是对比学习序列召回需要面对的重要挑战。对于新用户以及低活跃用户，由于历史行为序列较短，用户兴趣表达不够充分，因此较难构建高质量的正负样本对。通常需要结合用户画像、内容特征、迁移学习或者跨域知识等方式缓解冷启动问题。从实际工业界的经验来看，对比学习序列召回对于老用户

以及高活跃用户的效果提升通常更加明显，线上 A/B 实验往往能够取得较大的收益；而对于新用户和低活跃用户，由于可利用的行为信息有限，其效果提升通常相对有限。

10.5 多兴趣召回

在前面的序列召回章节中，我们介绍了 YoutubeDNN、双塔召回以及结合 Transformer、对比学习等技术的 Embedding 召回模型。这些方法通过对用户历史行为序列进行建模，能够学习更加准确的用户兴趣表示，从而有效提升召回效果。然而，这类模型通常采用单一 Embedding 来表示用户兴趣，即每个用户最终只对应一个固定长度的兴趣向量。当用户兴趣较为单一时，这种表示方式能够取得较好的效果；但对于兴趣广泛、行为多样的用户而言，单一兴趣向量往往难以同时刻画用户在不同领域、不同场景下的多样化兴趣，从而限制了召回模型的表达能力。

事实上，用户兴趣往往具有多样性、动态性和迁移性的特点。例如，在短视频推荐场景中，同一用户可能既关注科技资讯，也喜欢美食探店和体育赛事；在不同时间段，其兴趣重心还会不断发生变化。因此，传统的 Embedding 召回模型通常会存在以下的问题：

笔记 由于多个兴趣通常被压缩到同一个 Embedding 中，不同兴趣之间容易相互干扰，产生兴趣混叠（**Interest Entanglement**）问题，使得最终学习到的用户表示更偏向于用户的主导兴趣，而难以充分覆盖长尾兴趣，进而影响召回结果的准确性和多样性。

为了解决上述的问题，多兴趣召回（Multi-Interest Recommendation）逐渐成为近年来 Embedding 召回领域的重要研究方向。多兴趣召回的核心思想如下：

定义 10.18 (多兴趣召回)

多兴趣召回不再采用单一用户 Embedding，而是通过多兴趣表示学习，为每位用户学习多个兴趣向量，每个兴趣向量分别刻画用户某一类兴趣偏好。在线召回阶段，不同兴趣向量可以分别进行 ANN 近邻检索，再将多个兴趣对应的召回结果进行融合，从而同时覆盖用户的多种兴趣，提高召回结果的准确率、覆盖率以及多样性。

本节主要为读者介绍多兴趣召回中的两个经典模型：MIND（Multi-Interest Network with Dynamic Routing）和 ComiRec（Controllable Multi-Interest Recommendation）。其中，MIND 召回通过动态路由机制自动从用户历史行为中提取多个兴趣表示，实现用户兴趣的自动聚类；ComiRec 召回则进一步提出了可控的多兴趣学习框架，在兴趣表示学习的基础上兼顾召回准确率与结果多样性。这两个模型奠定了多兴趣 Embedding 召回的发展基础，也成为工业级推荐系统中多兴趣召回的代表性方法。

10.5.1 MIND 召回

MIND（Multi-Interest Network with Dynamic Routing）召回模型是阿里巴巴于 2019 年提出的一种经典多兴趣召回模型，源自论文《Multi-Interest Network with Dynamic Routing for Recommendation at Tmall》[31]。MIND 首次将胶囊网络（Capsule Network）[45] 中的动态路由（Dynamic Routing）思想引入推荐系统召回阶段，通过学习多个兴趣向量而非单一用户 Embedding，对用户多样化兴趣进行建模，从而有效提升 Embedding 召回的准确率和覆盖率。

MIND 召回模型整体结构如图 10.35所示，主要由 Embedding 层、多兴趣提取层（Multi-Interest Extractor）、Label-aware Attention 层以及召回层组成。与传统 YoutubeDNN 模型仅学习一个用户 Embedding 不同，MIND 召回模型认为一个用户通常同时具有多个兴趣。例如，在电商推荐场景中，同一个用户既可能关注数码产品，也可能关注运动装备、图书阅读等多个不同类别。如果仅采用单一 Embedding 表示用户兴趣，不同兴趣容易相互混合，导致召回精度下降。因此，MIND 召回模型将用户表示扩展为多个兴趣向量：

$$\mathbf{V}_u = \{\mathbf{v}_u^1, \mathbf{v}_u^2, \cdots, \mathbf{v}_u^K\} \tag{10.239}$$

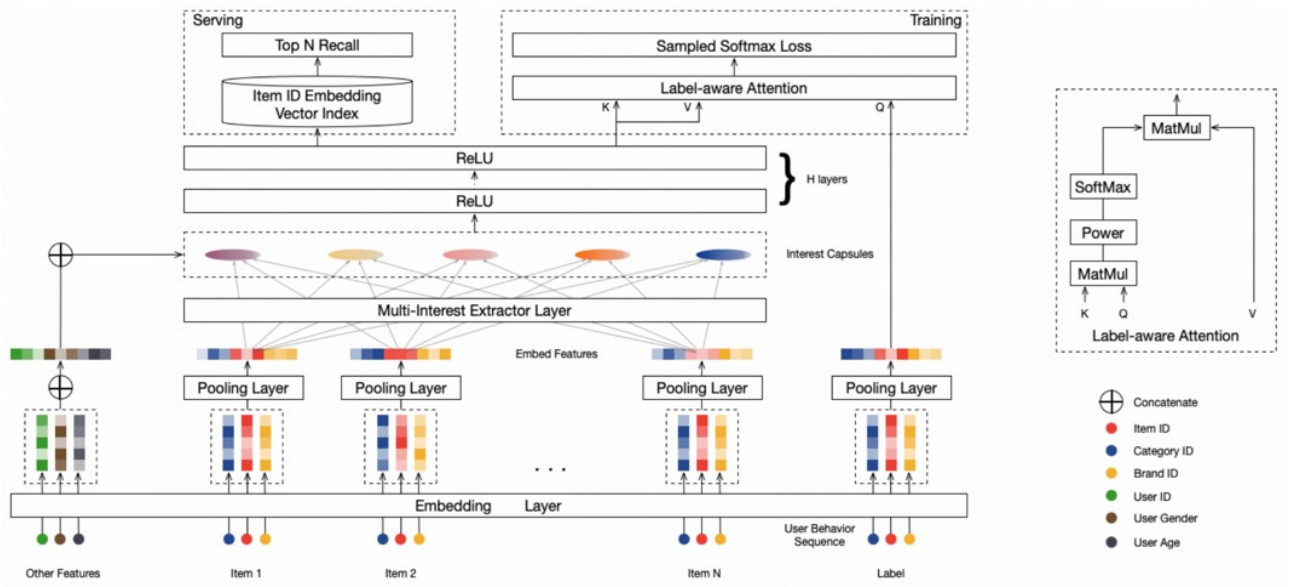

图 10.35: MIND 召回模型结构示意图。

其中，K 表示兴趣向量个数，每个兴趣向量对应用户的一类潜在兴趣。

10.5.1.1 动态路由机制

首先，MIND 召回模型将用户画像、历史行为序列以及目标物品分别映射到统一的 Embedding 空间。用户历史行为序列经过 Embedding 层后得到行为 Embedding 集合：

$$\mathbf{E}_u = \{\mathbf{e}_1, \mathbf{e}_2, \cdots, \mathbf{e}_n\} \tag{10.240}$$

其中，$\mathbf{e}_i \in \mathbb{R}^d$ 表示用户历史行为中第 i 个物品对应的 Embedding 向量，d 表示 Embedding 维度。整个行为序列可以看作由若干行为胶囊（Behavior Capsule）组成，而 MIND 召回模型希望进一步从这些行为胶囊中自动聚合得到 K 个兴趣胶囊（Interest Capsule）。

不同于传统 Attention 机制直接对所有行为进行一次加权求和，MIND 召回采用 Capsule Network 中的动态路由（Dynamic Routing）机制进行多轮聚类。Routing 过程可以理解为一种端到端可学习的软聚类算法，其目标是不断调整每个行为属于不同兴趣的概率，并逐步形成多个兴趣中心。对于每个用户，模型维护 K 个兴趣胶囊，其表示记为

$$\mathbf{U}_u = \{\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_K\} \tag{10.241}$$

其中 $\mathbf{u}_j \in \mathbb{R}^d$ 表示第 j 个兴趣胶囊的表示向量。需要说明的是，在 Routing 开始之前，这些兴趣胶囊并不是已经学习好的参数，而是通过随机初始化的 Routing Logits 经过第一次聚合得到；随后在每一轮 Routing 过程中不断更新，因此 $\mathbf{u}_j$ 实际上表示上一轮 Routing 得到的兴趣中心。

接着，模型计算每个行为胶囊与各个兴趣胶囊之间的路由得分（Routing Logits）：

$$b_{ij} = \mathbf{u}_j^T \mathbf{S} \mathbf{e}_i \tag{10.242}$$

其中，$\mathbf{e}_i \in \mathbb{R}^d$ 表示第 i 个行为 Embedding；$\mathbf{u}_j \in \mathbb{R}^d$ 表示当前第 j 个兴趣胶囊；$\mathbf{S} \in \mathbb{R}^{d \times d}$ 表示所有兴趣共享的双线性映射矩阵（Shared Bilinear Mapping Matrix）；b_{ij} 表示第 i 个行为属于第 j 个兴趣的匹配程度（Routing Logit）。可以看出，b_{ij} 本质上衡量了行为 Embedding 与当前兴趣中心之间的相似程度。当二者越接近时，对应的路由得分越大，说明该行为更倾向于分配给该兴趣。

随后，对所有兴趣胶囊上的 Routing Logits 进行 Softmax 归一化，得到行为分配到各兴趣上的概率：

$$w_{ij} = \frac{\exp(b_{ij})}{\sum_{k=1}^{K} \exp(b_{ik})} \tag{10.243}$$

其中 w_{ij} 表示第 i 个行为 Embedding 分配给第 j 个兴趣胶囊的权重，满足 $\sum_{j=1}^{K} w_{ij} = 1$。因此，每个行为不会只属于某一个兴趣，而是以不同概率分配到多个兴趣，这也是动态路由能够建模兴趣重叠的重要原因。接下来，根据得到的权重，对所有行为 Embedding 进行加权聚合，计算第 j 个兴趣胶囊对应的候选表示：

$$\mathbf{z}_j = \sum_{i=1}^{n} w_{ij} \mathbf{S} \mathbf{e}_i \tag{10.244}$$

其中 $\mathbf{z}_j \in \mathbb{R}^d$ 表示第 j 个兴趣胶囊在当前 Routing 轮次中的聚合结果。为了保持 Capsule Embedding 的稳定性，MIND 召回进一步采用 Capsule Network 提出的 Squash 函数对聚合结果进行归一化：

$$\mathbf{u}_j = \text{Squash}(\mathbf{z}_j) = \frac{\|\mathbf{z}_j\|^2}{1 + \|\mathbf{z}_j\|^2} \cdot \frac{\mathbf{z}_j}{\|\mathbf{z}_j\|} \tag{10.245}$$

Squash 函数能够压缩向量长度，使其始终小于 1，同时保留向量方向。其中，向量方向表示兴趣的语义信息，而向量长度则表示对应兴趣存在的置信程度。

完成上述步骤后，MIND 召回会重新利用新的兴趣胶囊 $\mathbf{u}_j$ 再次计算 Routing Logits，并重复上述更新过程。MIND 召回论文中提到 Routing 通常迭代 3 轮左右即可收敛。随着迭代不断进行，兴趣中心逐渐稳定，相似行为不断聚集到同一个兴趣胶囊，从而形成多个不同兴趣簇。因此，动态路由本质上是一种可学习的软聚类过程，其聚类中心并不是固定参数，而是在前向传播过程中不断迭代优化得到。Routing 结束之后，得到最终的兴趣胶囊集合：

$$\mathbf{U}_u = \{\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_K\} \tag{10.246}$$

根据图 10.35，在动态路由机制结束之后，每个兴趣胶囊随后还会经过一层共享的全连接映射（Shared DNN），得到最终用于召回的多兴趣表示：

$$\mathbf{v}_u^j = f_{\text{DNN}}(\mathbf{u}_j) \tag{10.247}$$

最终形成用户多兴趣 Embedding 集合

$$\mathbf{V}_u = \{\mathbf{v}_u^1, \mathbf{v}_u^2, \cdots, \mathbf{v}_u^K\} \tag{10.248}$$

虽然在 MIND 召回的论文正文中并未对该共享 DNN 层进行展开介绍，但从网络结构图 10.35中可以看出，该映射层主要用于进一步增强兴趣表示能力，并统一用户兴趣向量与物品 Embedding 所在的表示空间。

下面我们给出 MIND 召回 Dynamic Routing 的具体实现，供读者参考，如代码10.19所示：

代码 10.19: MIND 召回 Dynamic Routing 实现示例

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

def squash(x, eps=1e-9):
    """
    x: [K, d]
    """
    norm = torch.norm(x, dim=-1, keepdim=True)
    scale = norm**2 / (1.0 + norm**2)
    return scale * x / (norm + eps)


class B2IRouting(torch.nn.Module):

    def __init__(self, hidden_dim, interest_num, routing_iter=3):
        super().__init__()

        self.d = hidden_dim
```

```
        self.K = interest_num
        self.routing_iter = routing_iter

        # Shared bilinear matrix S
        self.S = torch.nn.Parameter(torch.randn(hidden_dim, hidden_dim))

    def forward(self, item_emb):
        """
        item_emb: [L, d]

        Returns
        -------
        interest: [K, d]
        """
        L = item_emb.size(0)
        # Step1. 初始化 Routing Logits b_ij
        b = torch.randn(L, self.K, device=item_emb.device) * 0.1

        # Routing
        for t in range(self.routing_iter):
            # Step2. Softmax得到行为属于每个兴趣的概率 w_ij
            w = F.softmax(b, dim=1)

            # Step3. Behavior -> Interest 聚合 Se_i
            proj_item = item_emb @ self.S.T
            # z_j
            z = torch.matmul(w.transpose(0, 1), proj_item)

            # Step4. Squash得到兴趣胶囊
            u = squash(z)

            # Step5. 更新Routing Logits, b_ij = u_j^T S e_i
            b = torch.matmul(proj_item, u.transpose(0, 1))
        return u
```

从上述代码中可以看出，动态路由机制的整体思想与 K-Means 聚类具有较高的相似性。动态路由同样采用了”聚类分配—聚类中心更新”的迭代优化策略：首先根据当前兴趣胶囊计算每个行为属于各兴趣簇的分配关系，再依据分配权重重新聚合得到新的兴趣表示，并不断重复这一过程直至收敛。不同之处在于，传统 K-Means 采用离散的簇分配和交替优化，而 MIND 召回采用可微分的 Softmax 函数计算行为分配权重，从而得到连续的软聚类结果。同时，动态路由中的共享双线性映射矩阵 **S** 作为模型参数，通过推荐任务的损失函数进行梯度反向传播学习，而无需像传统聚类算法那样单独优化聚类目标。此外，论文在消融实验中还分析了 Routing Logits 随机初始化方差对模型性能的影响，实验结果表明，不同初始化方差对最终召回效果影响较小，因此实际工程中通常将其初始化方差设置为 1.0 即可。

需要注意的是，虽然动态路由具有类似聚类算法的多轮迭代更新形式，但整个 Routing 过程实际上完全由双线性映射、Softmax、加权求和以及 Squash 非线性函数等连续可微运算组成，因此可以作为神经网络中的一个可微计算模块嵌入端到端训练框架。在模型训练过程中，推荐损失首先经过后文要讲述的 Label-aware Attention 先作用于对应的兴趣向量，再通过共享 DNN 反向传播至兴趣胶囊，进一步更新动态路由中的共享映射矩阵 **S** 以及底层 Embedding 参数。需要强调的是，Routing 过程中更新的路由得分 b_{ij} 并不是模型参数，而是每次前向传播

过程中根据当前兴趣表示动态计算得到的中间变量，其作用类似于 Soft K-Means 中的聚类分配概率，因此不会参与梯度更新。最终，整个多兴趣提取模块能够随着推荐目标进行联合优化，而无需像传统聚类算法那样执行聚类与模型训练交替进行的优化过程。

因此，可以将 MIND 中的动态路由理解为一种可微分的 Soft K-Means 聚类模块：它保留了聚类算法发现多个兴趣簇的思想，同时又能够与深度神经网络进行端到端联合训练，这也是 MIND 能够首次将多兴趣建模成功应用于工业级 Embedding 召回的重要原因之一。对比起序列召回章节中提到的 ICSRec 召回模型，MIND 召回模型的动态路由机制并不需要像 ICSRec 召回那样引入额外的离线聚类流程，也不需要维护兴趣原型，因此相比起 ICSRec 召回，MIND 召回在工业界实际应用过程中具有更高的工程落地可行性。

10.5.1.2 Label-aware Attention 机制

为了使不同目标物品能够选择最相关的兴趣向量参与训练，MIND 召回进一步提出了 Label-aware Attention 机制。具体而言，以目标物品 Embedding$\mathbf{e}_i$ 作为查询，多兴趣表示 $\mathbf{V}_u$ 同时分别作为键和值，来计算 Attention 分数并进行聚合得到针对当前目标物品对应的用户表示 $\mathbf{v}_u$：

$$\begin{aligned}\mathbf{v}_u &= \text{Attention}(\mathbf{e}_i, \mathbf{V}_u, \mathbf{V}_u) \\ &= \mathbf{V}_u \text{Softmax}(\text{pow}(\mathbf{e}_i^T \mathbf{V}_u, p))\end{aligned} \tag{10.249}$$

其中，p 为幂函数的指数，用于控制 Attention 分数的放缩程度。通过上述对 Attention 分数施加幂函数放缩的方式，当超参数 p 的取值趋于 0 时，整个 Attention 机制退化为传统的 Soft Attention，即所有兴趣向量均参与训练；而当 p 的取值趋于无穷大时，整个 Attention 机制逐渐逼近 Hard Attention，即最终主要选择最相关的一个兴趣向量参与训练，从而促使不同兴趣向量分别学习不同类型的兴趣模式。MIND 召回论文中发现采用类似 Hard Attention 的方式能够取得更快的收敛速度和更好的召回效果，因此在实际训练中通常将 p 设置为较大的值，例如 $p = 10$。

10.5.1.3 MIND 召回工程落地

在训练阶段，MIND 召回仍然采用 Embedding 召回中常见的 Sampled Softmax 损失函数来进行优化，其点击概率表示为：

$$P(i|u) = \frac{\exp(\mathbf{v}_u^T \mathbf{e}_i)}{\sum_j \exp(\mathbf{v}_u^T \mathbf{e}_j)} \tag{10.250}$$

其中 $\mathbf{v}_u$ 表示经过 Label-aware Attention 选择后的用户兴趣表示。由于工业推荐系统通常包含数亿甚至数十亿规模的物品，因此实际训练过程中采用 Sampled Softmax 近似计算 Softmax 分母，从而显著降低训练开销。

在线部署阶段，Label-aware Attention 模块会被移除，仅保留动态路由网络生成多个兴趣向量。随后，每个兴趣向量分别在 ANN 向量索引中独立进行近邻检索，最终将多个兴趣向量召回得到的候选物品进行合并去重，形成最终召回结果。因此，相比起 YoutubeDNN 召回仅利用一个用户 Embedding 进行一次 ANN 检索，MIND 召回实际上利用多个兴趣向量分别覆盖用户不同兴趣方向，从而能够显著提升召回结果的多样性和覆盖率。

总体来看，MIND 召回模型首次提出了多兴趣 Embedding 召回框架，并利用动态路由机制实现了用户兴趣的自动可微分聚类，使用户兴趣 Embedding 由单个向量扩展为多个向量表示。该方法不仅显著提升了兴趣多样化用户的召回效果，而且能够无缝结合工业界成熟的 ANN 向量检索系统，因此 MIND 召回迅速成为工业推荐系统中最具代表性的多兴趣召回模型之一，并为后续 ComiRec [5]、SINE [49]、PIMI [7] 等多兴趣召回模型奠定了基础。

10.5.2 ComiRec 召回

ComiRec（Controllable Multi-Interest Framework for Recommendation）是阿里巴巴在 2020 提出的一种多兴趣召回模型，源自于 KDD 会议论文《Controllable Multi-Interest Framework for Recommendation》[5]。ComiRec 召

回模型在 MIND 召回模型的基础上进一步提出了可控的多兴趣学习框架（Controllable Multi-Interest Framework），能够在保证召回准确率的同时兼顾召回结果的多样性，因此在工业推荐系统中具有较高的应用价值。

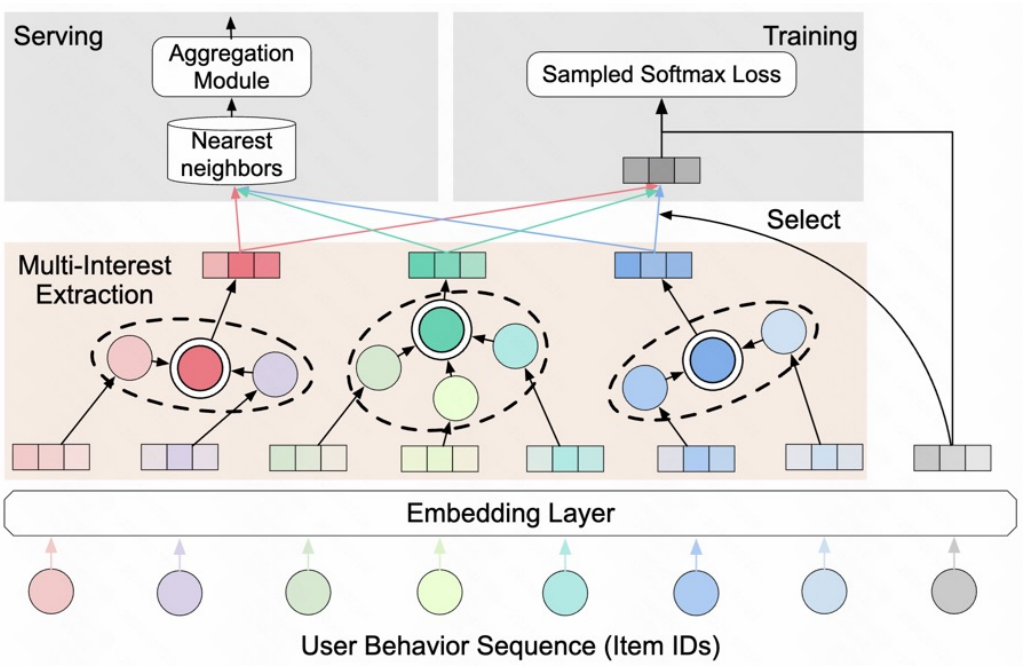

图 10.36: ComiRec 召回模型结构示意图。

ComiRec 召回模型整体结构如图 10.36所示，主要包括 Embedding 层、多兴趣提取层（Multi-Interest Extractor）、兴趣选择层（Interest Selector）以及召回聚合层（Aggregation Module）四个部分。与 MIND 召回相同，ComiRec 同样认为单一用户 Embedding 难以表达用户复杂、多样化的兴趣，因此采用多个兴趣向量共同表示用户：

$$\mathbf{V}_u = \left[\mathbf{v}_u^1, \mathbf{v}_u^2, \cdots, \mathbf{v}_u^K\right] \tag{10.251}$$

其中，K 表示兴趣向量个数，每个兴趣向量对应用户某一类潜在兴趣。

ComiRec 召回模型最大的特点在于其多兴趣提取模块具有较好的可扩展性。论文中提出了两种不同的兴趣提取方式，分别对应两个模型版本：ComiRec-DR（Dynamic Routing）和 ComiRec-SA（Self-Attention）。其中，ComiRec-DR 召回模型采用 Capsule Network 中的动态路由机制学习多个兴趣表示，其整体流程与 MIND 召回中的动态路由基本一致，因此这里不再赘述。而 ComiRec-SA 召回模型则利用 Self-Attention 机制来直接学习多个兴趣表示，而不再依赖动态路由的聚类过程。

对于 ComiRec-SA 召回模型，假设用户行为序列 Embedding 表示为：

$$\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_n] \in \mathbb{R}^{d \times n} \tag{10.252}$$

其中，n 表示用户历史行为长度，d 表示 Embedding 维度。首先利用两层注意力网络计算每个兴趣对应的 Attention 权重：

$$\mathbf{A} = \text{Softmax}\left(\mathbf{W}_2^T \tanh(\mathbf{W}_1\mathbf{H})\right)^T \tag{10.253}$$

其中，$\mathbf{W}_1 \in \mathbb{R}^{d_a \times d}$，$\mathbf{W}_2 \in \mathbb{R}^{d_a \times K}$ 均为可学习参数，$\mathbf{A} \in \mathbb{R}^{n \times K}$ 表示不同兴趣对应的注意力权重矩阵，每一列对应一种兴趣在所有历史行为上的注意力分布。随后，根据 Attention 权重对用户历史行为进行加权求和，即可得到多个兴趣表示：

$$\mathbf{V}_u = \mathbf{H}\mathbf{A} \tag{10.254}$$

可以看出，与 MIND 召回利用动态路由不断进行行为聚类不同，ComiRec-SA 召回直接采用多个 Attention Head 学习不同的用户兴趣，每个 Attention Head 会自动关注用户行为序列中的不同部分，因此训练更加简单，同时能够充分利用 Transformer 在序列建模方面的优势。

在得到多个用户兴趣表征之后，ComiRec 召回会采用兴趣选择器（Interest Selector）选择当前目标物品对应

的兴趣向量参与训练。具体而言，对于目标物品 Embedding $\mathbf{e}_i$，首先计算其与所有兴趣向量之间的相似度：

$$k^* = \arg\max_k \left((\mathbf{v}_u^k)^T \mathbf{e}_i\right) \tag{10.255}$$

随后选择最相关的兴趣向量作为当前用户的 Embedding：

$$\mathbf{v}_u = \mathbf{V}_u[:, k^*] \tag{10.256}$$

即每一个训练样本仅更新与目标物品最相关的兴趣表示，使不同兴趣向量能够分别学习不同兴趣方向，而不会互相干扰。随后，ComiRec 召回模型仍然采用 Sampled Softmax 损失函数来优化下一物品预测任务：

$$P(i|u) = \frac{\exp(\mathbf{v}_u^T \mathbf{e}_i)}{\sum_j \exp(\mathbf{v}_u^T \mathbf{e}_j)} \tag{10.257}$$

在工业场景下通常采用 Sampled Softmax 近似计算 Softmax 分母，从而降低模型训练的复杂度。

除了多兴趣建模之外，ComiRec 召回最重要的创新在于提出了可控的召回聚合（Controllable Aggregation）机制。传统多兴趣召回通常将每个兴趣向量分别进行 ANN 检索，每个兴趣独立召回 Top-N 候选物品，然后按照多个兴趣中最大的匹配得分进行融合：

$$f(u, i) = \max_{1 \le k \le K} \left(\mathbf{e}_i^T \mathbf{v}_u^k\right) \tag{10.258}$$

其中，$\mathbf{e}_i$ 表示物品 Embedding，$\mathbf{v}_u^k$ 表示用户的第 k 个兴趣向量。该策略能够充分利用多个兴趣进行召回，因此具有较高的召回准确率。然而，不同兴趣召回得到的候选物品往往存在较高的重叠度，最终推荐结果容易集中于少数兴趣方向，从而导致推荐列表缺乏多样性和覆盖率。

针对这一问题，ComiRec 进一步提出了可控聚合策略，在传统召回得分的基础上增加了多样性约束，构建如下优化目标：

$$Q(u, S) = \sum_{i \in S} f(u, i) + \lambda \sum_{i,j \in S} g(i, j) \tag{10.259}$$

其中，第一项表示推荐结果与用户兴趣之间的匹配程度，即推荐准确率；第二项表示推荐集合内部的多样性，$g(i, j)$ 通常用于衡量两个物品之间的差异性，例如是否属于不同类别。超参数 λ 用于控制推荐准确率与多样性之间的权衡。

由于上述目标函数属于组合优化问题，直接搜索最优 Top-N 推荐集合的计算复杂度较高，因此论文采用贪心推理（Greedy Inference）算法进行近似求解。具体而言，首先将多个兴趣向量分别进行 ANN 检索，并合并得到候选集合 M。随后从空集合开始，按照贪心策略逐步构建最终推荐集合 S：每一步遍历所有尚未被选中的候选物品，计算其加入当前推荐集合后的目标函数增益

$$f(u, i) + \lambda \sum_{k \in S} g(i, k) \tag{10.260}$$

选择能够使目标函数取得最大值的候选物品加入集合 S，并重复上述过程，直到获得最终的 Top-N 推荐结果。相比直接按照预测得分排序，该方法不仅考虑了候选物品与用户兴趣之间的相关性，还兼顾了候选物品之间的差异性，因此能够有效提升推荐列表的多样性。

当 $\lambda = 0$ 时，目标函数仅保留兴趣匹配得分，ComiRec 召回退化为传统 Top-N 召回，仅关注召回准确率；随着 λ 不断增大，贪心推理的算法会更加倾向于选择与当前推荐列表差异更大的候选物品，从而提高推荐结果的类别覆盖率和多样性。由于每一步仅选择当前能够带来最大目标函数增益的候选物品，整个贪心算法具有较低的在线计算复杂度，因此能够在保证推荐效率的同时，实现推荐准确率与多样性的可控平衡，这也是 ComiRec 召回相比 MIND 召回的重要创新之一。

总体来看，相较于 MIND 召回模型，ComiRec 召回模型最大的贡献并不在于提出新的多兴趣建模方法，而是在统一的多兴趣框架下同时支持 Dynamic Routing 和 Self-Attention 两种兴趣提取方式，并进一步引入可控的召回聚合策略，使模型能够根据业务需求灵活权衡推荐准确率与多样性。尤其是在首页推荐、短视频推荐等强调内容丰富性的业务场景中，ComiRec 召回能够有效缓解不同兴趣召回结果高度重叠的问题，因此具有较好的工程应用价值。

不过，ComiRec 召回同样存在一定局限性。一方面，ComiRec-DR 召回模型仍然依赖动态路由机制，计算复

杂度相对较高；另一方面，ComiRec-SA 召回模型虽然训练效率更高，但 Attention Head 之间可能学习到相似的兴趣模式，从而降低兴趣表示的多样性。此外，聚合模块中的多样性超参数 λ 需要根据具体业务场景进行调节，不同业务往往需要不同的准确率与多样性平衡策略。因此，在工业推荐系统中，ComiRec 召回通常需要结合具体业务特点设计更加复杂的兴趣融合与结果重排策略，以进一步提升整体推荐效果。

10.6 其他召回模型

本章前面的几个小节已经围绕工业级推荐系统中最主流的召回技术路线，对协同过滤召回、图召回、向量召回、序列召回以及多兴趣召回等方法进行了系统性的介绍，基本覆盖了当前工业界召回系统的核心算法。然而，在实际业务中，为了进一步提升召回效果、扩大召回覆盖范围以及满足超大规模在线检索需求，工业界还提出了许多具有针对性的召回模型。这些方法往往并不属于某一类独立的召回范式，而是从建模方式或系统架构等角度对传统召回模型进行了进一步扩展和优化。其中，一类工作通过显式建模用户与目标物品之间的多跳关联路径，实现更加精细的兴趣传播与匹配，代表性方法包括 PDN（Path-based Deep Network）；另一类工作则从大规模索引结构出发，将深度学习模型与树结构检索相结合，以突破 Embedding 召回在线检索效率的瓶颈，代表性方法包括 TDM（Tree-based Deep Model）。

本节将重点介绍 PDN 召回和 TDM 召回两种具有代表性的工业召回模型。除此之外，近年来提出的链路一致性召回等方法将在第20章进行介绍；生成式召回、召回排序一体化等新一代召回框架将在第34章进行介绍；而基于大语言模型（LLM）和 Agent 的智能召回方法则将在第36章进行介绍。

10.6.1 PDN 召回

PDN（Path-based Deep Network）召回模型是阿里巴巴于 2021 年提出的一种基于多跳路径建模（Multi-hop Path Modeling）的召回模型，源自论文《Path-based Deep Network for Candidate Item Matching in Recommenders》[32]。PDN 召回模型的核心思想如下：

定义 10.19

PDN 召回是将传统召回过程中的用户到物品（User-to-Item，U2I）匹配扩展为用户 → 历史行为物品 → 目标物品（User→Item→Item，U2I2I）的两跳路径建模，通过显式建模每一条路径上的兴趣传播过程，从而充分融合用户兴趣、历史行为以及物品关联关系，提高召回模型的准确率和覆盖率。

10.6.1.1 PDN 召回算法原理

传统 Embedding 召回（Embedding-based Retrieval，EBR）通常将用户历史行为压缩为一个固定长度的用户 Embedding，再与目标物品 Embedding 计算相似度完成召回。例如 YoutubeDNN、双塔召回等方法本质上均属于这一范式。然而，将用户全部历史行为压缩为单一 Embedding 容易丢失不同兴趣之间的细粒度关系；另一方面，传统 ItemCF 召回虽然能够利用物品之间的共现关系，但缺乏用户画像和上下文信息。因此，PDN 召回提出采用路径（Path）作为基本建模单元，将用户与目标物品之间的关联表示为多条两跳路径：

$$u \rightarrow j \rightarrow i \tag{10.261}$$

其中，u 表示用户，j 表示用户历史交互过的 Trigger 物品，i 表示待召回的目标物品。整个匹配过程可以表示为：

$$\hat{y}_{ui} = f\left(\mathbf{z}_u, \mathbf{x}_i, \{\mathbf{x}_j\}, \{\mathbf{a}_{uj}\}, \{\mathbf{c}_{ji}\}\right) \tag{10.262}$$

其中，$\mathbf{z}_u$ 表示用户画像特征，$\mathbf{x}_i$ 表示目标物品特征，$\mathbf{x}_j$ 表示历史行为物品特征，$\mathbf{a}_{uj}$ 表示用户对历史物品 j 的行为特征（如点击次数、停留时长等），$\mathbf{c}_{ji}$ 表示历史物品 j 与目标物品 i 之间的相关性特征，例如 ItemCF 召回相似度、共现统计等。

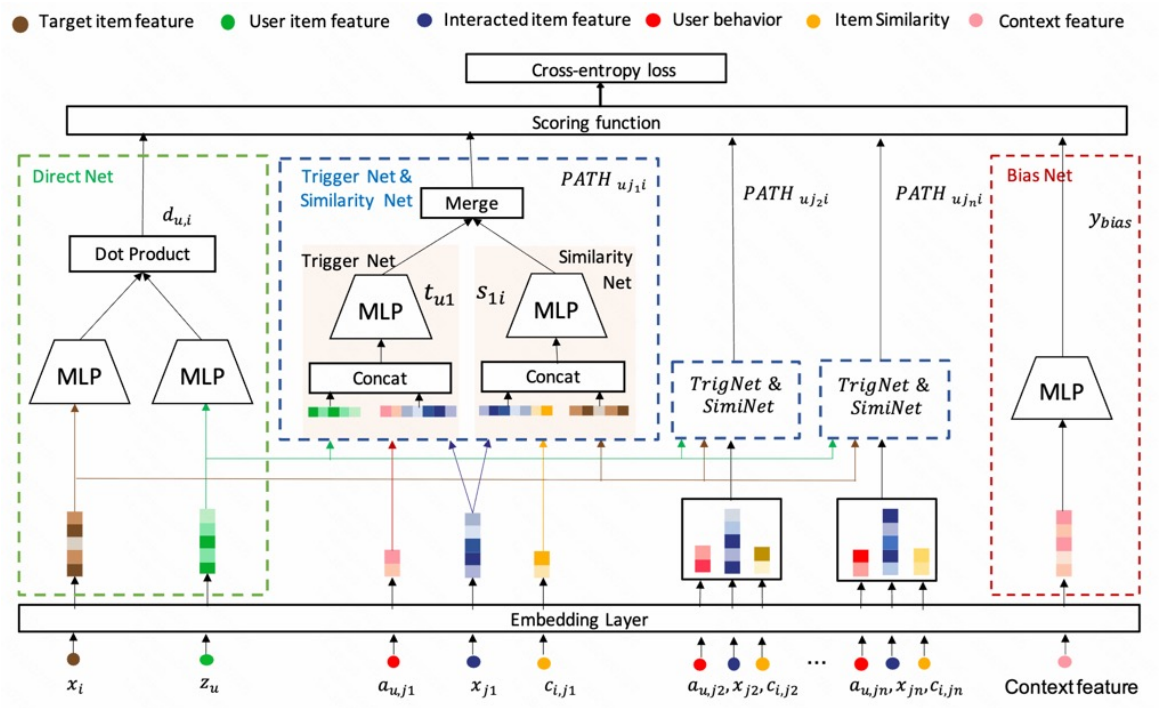

图 10.37: PDN 召回模型结构示意图。

PDN 召回模型整体结构如图 10.37所示，主要包括 Embedding 层、Trigger Net、Similarity Net（SimNet）、Direct Net 以及 Bias Net 五个部分。其中，Embedding 层首先将用户特征、历史行为、目标物品以及物品共现信息统一映射到 Embedding 空间。随后，对于每一条两跳路径，PDN 分别建模第一跳和第二跳的相关性。

第一跳由 Trigger Net 负责建模用户对于历史行为物品的兴趣程度。具体而言，Trigger Net 以用户画像、用户行为信息以及历史物品 Embedding 作为输入，通过 MLP 输出用户对于每个 Trigger 物品的重要性：

$$t_{uj} = \text{TriggerNet}(\mathbf{z}_u, \mathbf{a}_{uj}, \mathbf{x}_j) \tag{10.263}$$

其中，t_{uj} 表示用户对历史物品 j 的兴趣权重。由于每个历史物品都会对应一个独立的兴趣得分，因此整个用户兴趣可以表示为：

$$\mathbf{T}_u = [t_{u1}, t_{u2}, \cdots, t_{un}] \tag{10.264}$$

相比 YoutubeDNN 等方法利用固定长度 Embedding 表示用户兴趣，TriggerNet 显式学习了每一个历史行为的重要程度，因此能够更加细粒度地描述用户的多样化兴趣。

第二跳由 Similarity Net（SimNet）负责建模历史物品与目标物品之间的关联关系。SimNet 输入历史物品 Embedding、目标物品 Embedding 以及两者之间的共现特征，通过另一组 MLP 网络来计算二者之间的相关性：

$$s_{ji} = \text{SimNet}(\mathbf{x}_j, \mathbf{c}_{ji}, \mathbf{x}_i) \tag{10.265}$$

其中，s_{ji} 表示历史物品 j 与目标物品 i 之间的相似程度。与传统 ItemCF 召回直接利用统计相似度不同，SimNet 能够同时融合物品属性、共现关系以及辅助特征，从而学习更加准确的 Item-to-Item 相关性。此外，由于 SimNet 仅依赖物品侧特征，因此训练完成后还可以单独部署，用于构建 I2I 召回索引。

在得到第一跳兴趣权重和第二跳相似度之后，PDN 召回将两部分信息融合得到每条路径的贡献值：

$$\text{PATH}_{uji} = \ln\left(1 + \exp(t_{uj})\exp(s_{ji})\right) \tag{10.266}$$

其中，Softplus 函数形式能够保证每条路径的贡献始终为正值，避免不同路径之间由于正负抵消导致训练不稳定。PDN 召回的论文指出，相比允许路径权重取负值，这种正值约束更加符合真实推荐场景中路径贡献累加的物理意义，也能够有效缓解模型过拟合的问题。

除了两跳路径之外，PDN 召回还设计了 Direct Net 用于建模用户与目标物品之间的一跳兴趣关系。Direct Net 采用经典的双塔结构分别学习用户表示和目标物品表示，其中用户侧输入用户画像等静态特征，物品侧输入目标物品特征，经过 Embedding 层和 MLP 编码后分别得到用户 Embedding 和目标物品 Embedding：

$$\mathbf{p}_u = \mathrm{MLP}(\mathbf{E}(\mathbf{z}_u)), \qquad \mathbf{q}_i = \mathrm{MLP}(\mathbf{E}(\mathbf{x}_i)) \tag{10.267}$$

两者通过内积计算用户与目标物品之间的一跳相关性：

$$d_{ui} = \mathbf{p}_u^T \mathbf{q}_i \tag{10.268}$$

与 TrigNet 建模用户对具体 Trigger Item 的兴趣不同，Direct Net 主要负责学习用户整体偏泛化的兴趣，例如年龄、性别、消费能力等稳定画像信息所带来的偏好，因此能够与两跳路径建模形成有效互补。

除此之外，PDN 召回还额外引入了 Bias Net 用于学习训练数据中的各种曝光偏置（Selection Bias）。例如，在推荐系统中，排序位置越靠前的物品往往更容易获得点击；此外，不同时间段、不同场景下用户点击行为也存在明显偏差。如果直接利用点击数据进行训练，模型容易将这些曝光偏置误认为是真实兴趣，从而影响召回模型学习。因此，PDN 召回利用一个浅层网络对位置特征、时间特征等偏置信息进行建模，输出 Bias Logit：

$$y_{\text{bias}} = \mathrm{BiasNet}(\mathbf{x}_{\text{bias}}) \tag{10.269}$$

其中，$\mathbf{x}_{\text{bias}}$ 表示位置、时间等与曝光机制相关的偏置特征。需要注意的是，Bias Net 仅参与模型训练，用于帮助主模型消除点击日志中的曝光偏置；在线 Serving 阶段会直接移除 Bias Net，仅保留用户真实兴趣相关的打分结果，从而获得更加无偏（Unbiased）的召回结果。

最终，PDN 召回模型将 Direct Net、一跳与两跳路径以及 Bias Net 共同融合得到最终的匹配得分：

$$\hat{y}_{ui} = \mathrm{softplus}(d_{ui}) + \sum_{j \in N(u)} \mathrm{PATH}_{uji} + \mathrm{softplus}(y_{\text{bias}}) \tag{10.270}$$

其中，$N(u)$ 表示用户历史交互物品集合。论文采用 Softplus 函数约束 Direct Net 和 Bias Net 输出为正值，使各条路径均表示对点击概率的正向贡献，从而避免不同路径之间出现相互抵消的问题，也使路径权重具有更加明确的物理意义。随后，将最终匹配得分转换为点击概率：

$$p_{ui} = 1 - \exp(-\hat{y}_{ui}) \tag{10.271}$$

其中，$\hat{y}_{ui} \in (0, +\infty)$，因此经过上述变换后点击概率自然落入 $(0, 1)$ 区间。模型最终采用 Cross-Entropy Loss 进行训练：

$$\mathcal{L} = -\left(y_{ui} \log p_{ui} + (1 - y_{ui}) \log(1 - p_{ui})\right) \tag{10.272}$$

其中，$y_{ui} \in \{0, 1\}$ 表示用户是否点击目标物品。通过联合优化 TrigNet、SimNet、Direct Net 以及 Bias Net 四个模块，PDN 召回能够同时学习用户一跳的泛化兴趣、多跳关联关系以及曝光偏置，从而获得更加准确且具有较强泛化能力的召回模型。

10.6.1.2 PDN 召回工程落地

PDN 召回的工程架构如图10.38所示。虽然 PDN 召回能够利用用户与目标物品之间的两跳路径进行更加精细的兴趣建模，但如果在线阶段直接计算用户与全量物品之间所有可能的两跳路径，其计算复杂度将达到 $O(|N(u)| \times |\mathcal{I}|)$，这很难满足工业级推荐系统对于召回模型在线推理毫秒级响应的要求。因此，PDN 召回针对两跳路径设计了一套离线索引构建与在线路径检索相结合的 Serving 架构，将大量计算提前迁移至离线阶段，实现模型效果与在线性能之间的平衡。

PDN 召回系统的整个在线召回流程主要包括索引生成、Trigger 筛选以及候选检索三个阶段。首先，在索引生成阶段，PDN 召回主要利用训练完成的 SimNet 计算物品之间的两两关联得分 s_{ji}。由于直接计算全量物品之间的相似度矩阵需要 $O(|\mathcal{I}|^2)$ 的计算复杂度，因此论文首先通过共现关系、同品牌、同类目等策略生成候选物品对，将候选规模由 $|\mathcal{I}|^2$ 降低至 $|\mathcal{I}| \times \hat{K}$（其中 $\hat{K} \gg K$）。随后，利用 SimNet 对候选物品对进行打分，并为每个

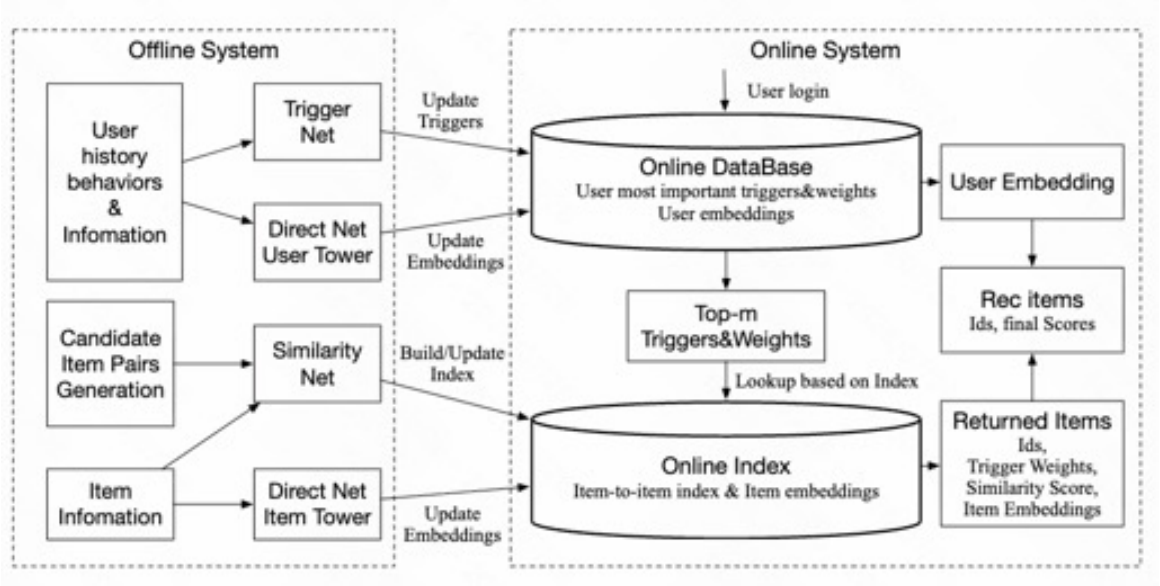

图 10.38: PDN 召回在线 Serving 架构示意图。

物品保留得分最高的 Top-K 个关联物品，最终构建得到 Item-to-Item 倒排索引：

$$\mathcal{I}_j = \{(i, s_{ji}) \mid i \in \text{Top-}K(j)\} \tag{10.273}$$

其中，$\mathcal{I}_j$ 表示以物品 j 为 Trigger Item 对应的 Top-K 关联物品集合，同时保存对应的 SimNet 相关性得分 s_{ji}。由于整个索引完全离线生成，因此不会增加在线计算开销。

随后，在在线阶段，当用户发起推荐请求时，推荐系统首先获取用户最近的历史交互序列，并利用 Trigger Net 计算用户对每个历史行为的兴趣权重：

$$t_{uj} = \text{TriggerNet}(\mathbf{z}_u, \mathbf{a}_{uj}, \mathbf{x}_j) \tag{10.274}$$

其中，t_{uj} 表示用户对历史物品 j 的兴趣程度。随后，仅保留得分最高的 Top-m 个 Trigger Item 作为路径扩展的起点：

$$\mathcal{T}_u = \text{Top-}m(\{t_{uj}\}) \tag{10.275}$$

从而避免对所有历史行为同时进行路径扩展，大幅降低在线检索复杂度。

对于每一个 Trigger 物品，推荐系统进一步查询对应的 Item-to-Item 倒排索引，获得其 Top-K 关联物品，因此最终可得到约 $m \times K$ 个候选物品。相比直接在数十亿规模物品库中进行检索，整个候选集合规模已经大幅缩小。随后，推荐系统综合 Direct Net、一跳路径以及两跳路径的信息，对候选物品重新进行打分：

$$\hat{s}_{ui} = \text{softplus}(d_{ui}) + \sum_{j \in \mathcal{T}_u} \text{softplus}(t_{uj} + s_{ji}) \tag{10.276}$$

其中，$\mathcal{T}_u$ 表示 $m \times K$ 个候选物品组成的集合，d_{ui} 表示 Direct Net 输出的一跳兴趣得分，t_{uj} 表示用户对 Trigger Item 的兴趣权重，s_{ji} 表示 SimNet 学习得到的 Trigger 物品与候选物品之间的关联强度。推荐系统依据上述打分结果重新排序，返回 Top-N 个候选物品作为 PDN 召回的最终候选集。这里需要注意的是，整体在线召回阶段 Bias Net 不会参与推理，该网络仅在训练阶段用于建模曝光偏置，因此最终得到的是更加无偏的用户兴趣得分。

从整体架构来看，PDN 召回最大的工程创新在于将复杂的两跳路径建模解耦为离线和在线两个阶段：即 SimNet 负责离线学习物品之间的关联关系，并提前构建 Item-to-Item 倒排索引；Trigger Net 负责在线学习用户当前最重要的兴趣触发点；Direct Net 负责建模用户长期兴趣偏好。三者共同组成完整的 U2I2I 路径建模框架，使得 PDN 召回模型既能够充分利用用户行为、物品共现以及侧信息，又能够满足工业推荐系统对于低时延、高吞吐在线推理的要求。此外，由于 SimNet 本质上学习的是物品之间的语义关联，因此其生成的倒排索引不仅能够服务于 PDN 召回，也可以直接应用于传统 Item-to-Item 召回场景，具有较高的工程复用价值。这种将复杂模型拆解为“离线建索引 + 在线路径检索”的设计思路，也为后续工业界多跳召回模型的工程落地提供了重要参考。

10.6.2 TDM 召回

TDM（Tree-based Deep Model）召回是一种基于树结构索引的深度学习召回方法，最早由阿里巴巴于 2018 年提出，源自论文《Learning Tree-based Deep Model for Recommender Systems》[64]。TDM 召回的核心思想如下：

定义 10.20

TDM 召回是在物品集合之上构建一棵层次化的推荐树（Recommendation Tree），将传统 Embedding 召回中的全局最近邻搜索问题转化为树上的逐层层次化检索（Hierarchical Retrieval）。相比传统 Embedding 召回依赖 ANN 索引进行近似最近邻搜索，TDM 召回能够充分利用树结构对候选空间进行逐层筛选，在保持较高召回效果的同时，将在线检索复杂度降低至对数级，因此成为工业推荐系统中一种经典的大规模召回方法。

传统 Embedding 召回模型（如 YoutubeDNN、DSSM 等）通常首先学习用户 Embedding 与物品 Embedding，然后利用二者之间的内积作为用户对物品的兴趣得分：

$$s(u,i) = \mathbf{u}^T \mathbf{e}_i \tag{10.277}$$

其中，$\mathbf{u} \in \mathbb{R}^d$ 表示用户 Embedding，$\mathbf{e}_i \in \mathbb{R}^d$ 表示物品 Embedding，d 表示 Embedding 维度。在线召回时，需要根据用户 Embedding 在整个物品 Embedding 空间中搜索最相似的 Top-K 物品。虽然工业界通常采用 Faiss、ScaNN、HNSW 等 ANN 技术加速检索，但随着物品规模不断增长至数亿甚至数十亿级别，ANN 索引的构建、维护以及更新成本也不断增加。此外，每当 Embedding 模型发生更新时，往往需要重新构建整个向量索引，因此整体工程维护成本较高。

为了解决上述问题，TDM 召回提出了另一种完全不同的召回思路。它不再直接在整个物品空间中搜索目标物品，而是首先利用一棵层次化的推荐树对整个物品集合进行组织。树中的叶子节点（Leaf Node）与真实物品一一对应，而非叶子节点（Non-leaf Node）则表示多个物品组成的粗粒度兴趣概念（Coarse-grained Concept）。因此，推荐树实际上建立了一种由粗到细的兴趣层次结构，使模型能够逐步缩小候选空间，而不是一次性面对整个物品集合进行搜索。

假设推荐树包含节点集合 $\mathcal{N}$，则整棵推荐树可以表示为：

$$\mathcal{N} = \{n_1, n_2, \cdots, n_{|\mathcal{N}|}\} \tag{10.278}$$

其中，n_1 表示根节点（Root Node），每个叶子节点唯一对应一个真实物品，而内部节点则表示若干叶子节点所组成的兴趣集合。需要注意的是，TDM 召回论文中的推荐树并不要求必须是完全二叉树，也不限制每个节点的子节点数量，因此能够根据实际业务灵活构建不同规模和不同形态的树结构。例如，在电商推荐场景中，根节点可以表示整个商品空间，第一层节点对应服饰、数码、食品、家居等一级兴趣类别；继续向下划分后，可以进一步形成男装、女装、运动鞋、手机、电脑等更加细粒度的兴趣节点；最终，树的叶子节点则对应具体的商品。随着树层数的不断增加，每一层兴趣节点所表示的语义粒度逐渐变细，因此推荐过程也由粗粒度兴趣逐渐定位到具体的商品。

TDM 召回模型的整体结构如图 10.39所示，主要由 Embedding 层、用户兴趣建模网络（Deep Matching Network）、层次化树结构（Recommendation Tree）以及层次化检索模块（Hierarchical Retrieval）组成。首先，模型根据用户画像、历史行为序列以及上下文信息学习用户兴趣表示；随后，在推荐树的每一层预测用户最感兴趣的若干兴趣节点，仅保留得分最高的 Top-K 节点继续向下一层扩展；最终逐层搜索至叶子节点，得到最终的召回结果。由于每一层仅需要访问极少量候选节点，因此整个检索复杂度由传统 Embedding 召回近似遍历整个物品空间，降低为树结构上的逐层搜索，整体复杂度约为 $O(K \log |\mathcal{I}|)$，其中 $|\mathcal{I}|$ 表示物品数量。

需要指出的是，TDM 召回虽然采用树结构进行层次化检索，但其思想与传统自然语言处理中的 Hierarchical Softmax 并不相同。Hierarchical Softmax 主要用于降低 Softmax 计算复杂度，而 TDM 召回则重新定义了树节点的兴趣表示和训练目标，使推荐任务能够真正支持高效的 Top-K 层次化检索。论文指出，如果直接采用 Hierarchical Softmax 进行推荐召回，不仅无法保证检索得到全局最优候选，而且容易受到上层路径决策错误的影响。因此，TDM 召回进一步提出了基于 Max-Heap 思想的树模型（Tree-based Model Formulation），并重新设计了节点训练方

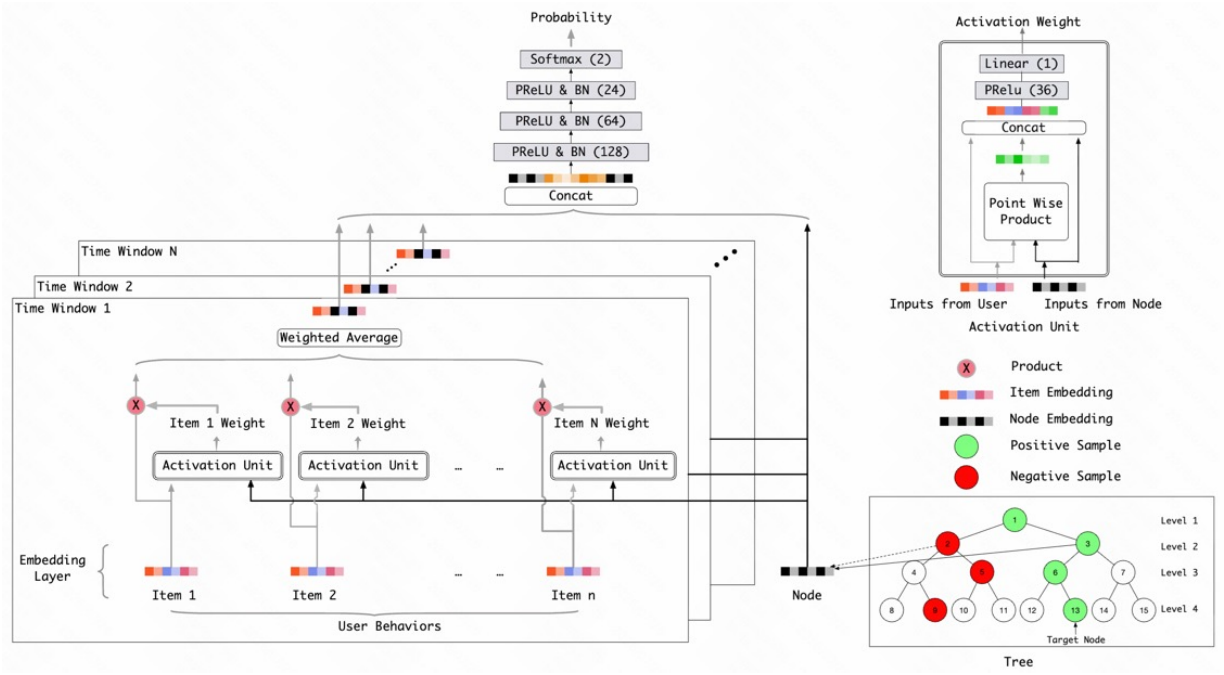

图 10.39: TDM 召回模型结构示意图。

式和层次化检索算法，从而成为一种真正适用于工业推荐系统的大规模树结构召回模型。下面将分别介绍 TDM 召回的树模型构建方法、训练方式以及在线层次化检索过程。

10.6.2.1 Hierarchical Softmax 存在的问题

推荐树并不是 TDM 召回首次提出的思想。在自然语言处理领域，Hierarchical Softmax（HS）早已利用树结构降低 Softmax 的计算复杂度。其基本思想是将每个类别（或单词）映射到树的叶子节点，一个叶子节点对应从根节点到叶子节点的一条唯一路径。模型不再直接计算所有类别上的 Softmax 概率，而是将最终概率分解为路径上每一级二分类概率的乘积，从而避免传统 Softmax 需要遍历整个类别空间带来的巨大计算开销。

假设叶子节点 n 对应某个目标物品，其从根节点到叶子节点的路径长度为 L，则 Hierarchical Softmax 将用户对该物品的兴趣概率表示为

$$P(n|u) = \prod_{j=1}^{L} P\left(b_j(n) \mid l_j(n), u\right), \tag{10.279}$$

其中，$b_j(n)$ 表示路径在第 j 层的分支选择（例如左分支或右分支），$l_j(n)$ 表示路径在第 j 层对应的父节点。也就是说，最终到达某个叶子节点的概率等于整条路径上所有二分类概率的乘积。这种方式能够有效降低训练阶段 Softmax 的计算复杂度，因此被广泛应用于 Word2Vec 等自然语言处理模型。然而，论文指出，Hierarchical Softmax 并不适用于推荐系统的大规模召回任务，主要存在以下两个问题。

首先，Hierarchical Softmax 并不能直接解决 Top-K 检索问题。虽然训练阶段避免了遍历整个物品集合，但在线检索时，如果希望找到概率最大的 Top-K 物品，理论上仍然需要计算所有叶子节点对应的路径概率，因此无法真正降低召回阶段的计算复杂度。一个直观的想法是在树上采用贪心搜索，每一层始终选择概率最大的子节点继续向下搜索。然而，这种方法并不能保证最终到达全局最优的叶子节点。一旦模型在树的高层做出错误决策，后续所有正确候选都会被提前剪枝，最终得到的召回结果可能远离真实最优解。

其次，Hierarchical Softmax 采用的是局部二分类训练方式。对于树上的每一个内部节点，模型只需要判断当前用户更倾向于左子树还是右子树，即每一层实际上训练的是一个局部二分类器。这意味着模型重点学习的是如何区分两个兄弟节点，而不是在整层所有节点之间学习全局排序能力。然而，在推荐系统中，同一父节点下的两个子节点通常表示语义相近的兴趣类别，例如” 篮球鞋” 与” 运动鞋”、” 苹果手机” 与” 安卓手机” 等，用户

往往可能同时对两类物品感兴趣。将它们视为互斥类别进行二分类训练，容易损失模型的表达能力，也不利于最终召回质量。

事实上，YoutubeDNN 等工业 Embedding 召回工作也曾尝试采用 Hierarchical Softmax 学习用户和物品表示，但实验结果表明，其效果普遍不如 Sampled Softmax 训练方式。因此，TDM 并没有直接采用 Hierarchical Softmax，而是重新定义了推荐树上的兴趣概率，并提出了一种类似最大堆（Max-Heap）的树模型，使每一层都能够学习全局排序能力，从而支持高效且准确的层次化 Top-K 检索。下面将介绍 TDM 召回提出的树模型。

10.6.2.2 TDM 召回的树模型

针对 Hierarchical Softmax 存在的上述问题，TDM 召回并没有继续采用路径概率连乘的形式，而是重新定义了推荐树上的兴趣概率。论文认为，推荐系统的目标并不是精确估计每个叶子节点的概率，而是快速找到用户最感兴趣的 Top-K 物品。因此，TDM 召回提出了一种类似最大堆（Max-Heap）的树模型（Tree-based Model），使父节点能够表示其所有子节点中的最大兴趣程度，从而支持层次化的 Top-K 检索。

假设推荐树第 j 层节点 n 的兴趣概率表示为 $P^{(j)}(n|u)$，则 TDM 定义父节点与子节点之间满足如下关系：

$$P^{(j)}(n|u) = \frac{\max\limits_{n_c \in \text{Child}(n)} P^{(j+1)}(n_c|u)}{\alpha^{(j)}} \tag{10.280}$$

其中，$\text{Child}(n)$ 表示节点 n 的所有子节点，$\alpha^{(j)}$ 表示第 j 层的归一化系数，用于保证同一层所有节点的兴趣概率之和为 1。该定义意味着，父节点的兴趣程度由其子树中最感兴趣的节点决定，而不是所有子节点兴趣的平均值。因此，每一个内部节点都可以理解为其整个子树兴趣的代表。当用户对某个叶子节点具有较高兴趣时，该叶子节点对应路径上的所有祖先节点都会具有较高的兴趣得分，从而保证真正感兴趣的商品能够沿着树结构逐层保留下来。与 Hierarchical Softmax 相比，这种定义最大的不同在于，模型学习的是每一层节点之间的全局排序关系，而不是兄弟节点之间的局部二分类关系。因此，在线检索时，只需要保留当前层得分最高的 Top-K 节点，并继续扩展这些节点的子节点即可，不再依赖单一路径的贪心搜索，从而有效缓解了 Hierarchical Softmax 存在的误差累积问题。

然而，上述兴趣概率仅作为理论定义，在实际训练过程中难以直接获得。因此，TDM 召回进一步将兴趣预测问题转化为每一层节点上的二分类任务。对于用户的一条正样本行为，即用户点击了某个叶子节点对应的物品，则该叶子节点以及从根节点到该叶子节点路径上的所有祖先节点均视为正样本，而同一层随机采样得到的其他节点则作为负样本，如图10.39所示。假设用户 u 在所有层上的正样本集合和负样本集合分别表示为 Y_u^+ 和 Y_u^-，则模型采用统一的深度神经网络预测用户是否对节点 n 感兴趣，并使用 Binary Cross-Entropy Loss 进行训练：

$$\mathcal{L} = -\sum_{u} \sum_{n \in Y_u^+ \cup Y_u^-} [y_u(n) \log \hat{y}_u(n) + (1 - y_u(n)) \log(1 - \hat{y}_u(n))] \tag{10.281}$$

其中，$y_u(n) \in \{0, 1\}$ 表示节点 n 是否为用户 u 的正样本，$\hat{y}_u(n)$ 表示模型预测用户对节点 n 感兴趣的概率。

需要注意的是，TDM 召回与 Hierarchical Softmax 最大的区别就在于负样本的构造方式。Hierarchical Softmax 仅利用路径上的左右子节点进行局部二分类训练，而 TDM 召回则在同一层所有节点中随机采样负样本，使模型能够学习当前层节点之间的全局排序关系，而不是局部路径决策。这种训练方式使每一层都能够独立判断用户最感兴趣的候选节点，即使上层保留了一些质量较低的节点，下层模型仍然能够从这些候选中继续筛选出更优的节点，从而显著提高了层次化检索的准确率，也为后续在线 Top-K 层次化搜索提供了理论基础。

10.6.2.3 TDM 召回的深度网络

在完成树结构建模之后，TDM 召回还需要学习用户对树中各个节点的兴趣程度。为此，论文设计了统一的深度神经网络作为节点兴趣预测模型，其整体结构如图10.39所示。模型输入包括用户历史行为序列、用户画像特征以及待预测树节点的信息，输出用户对当前树节点的兴趣概率。

首先，模型为树中的每一个节点学习一个 Embedding 表示。需要注意的是，这里的 Embedding 并不仅对应叶子节点所表示的物品，而是树中的所有节点均拥有各自独立的 Embedding 参数，因此内部节点也能够学习对应粗粒度兴趣概念的语义表示。对于用户历史行为，TDM 并没有直接对整个行为序列进行平均池化，而是参考 CTR 模型中的 Attention 机制，根据当前候选节点动态计算用户历史行为的重要程度。具体而言，模型首先将用户历史交互的物品 Embedding 进行编码，再利用 Attention 机制计算不同历史行为与当前树节点之间的相关性，从而获得更加具有针对性的用户兴趣表示。相比传统平均池化，这种方式能够更加准确地刻画用户当前兴趣，提高不同树节点之间的判别能力。

此外，为了进一步建模用户兴趣随时间的变化，论文还提出了 Block-wise Input 的输入方式。具体而言，模型按照时间窗口将用户历史行为划分为多个时间段，例如最近一天、最近一周、最近一个月等，每个时间窗口内分别进行 Embedding 聚合，随后再输入后续神经网络进行融合。这样既能够保留用户长期兴趣，又能够突出近期行为的重要性，使模型同时具备长期兴趣建模与短期兴趣建模能力。

最终，用户兴趣表示与候选树节点 Embedding 共同输入多层感知机（MLP），输出用户对当前树节点的兴趣概率。由于模型直接学习的是用户与树节点之间的非线性匹配关系，因此其表达能力明显强于传统 Embedding 内积形式，也能够方便地融合用户画像、上下文环境以及统计特征等丰富的侧信息。

值得注意的是，TDM 召回的深度网络本身并不是论文的主要创新，其真正的重要性在于将复杂的深度神经网络与层次化树检索相结合。传统 Embedding 召回为了保证 ANN 检索效率，通常只能采用双塔模型学习用户 Embedding 和物品 Embedding，并通过向量内积计算相似度。而 TDM 召回由于每次在线推理仅需要计算少量树节点，因此可以采用更加复杂的 Attention 网络和 MLP 结构，在保证在线效率的同时显著提升模型表达能力，这也是 TDM 召回能够取得较好召回效果的重要原因之一。

10.6.2.4 TDM 召回的树构建与学习

推荐树不仅决定了在线检索路径，也直接影响模型的训练效果，因此树结构本身也是 TDM 召回模型的重要组成部分。论文认为，一个合理的推荐树应当满足两个要求：一方面，相似物品应尽可能聚集在同一子树中，从而保证高层节点能够学习到具有明确语义的粗粒度兴趣；另一方面，整棵树应尽量保持平衡，以避免不同分支深度差异过大，从而影响模型训练效率与在线检索性能。

论文首先提出了一种基于类目的树初始化方法（Category-based Initialization）。由于大多数工业级推荐系统中的商品都具有类别（Category）等属性，因此首先按照商品所属类别进行排序，使同一类别的商品尽可能相邻排列。随后采用递归二分（Recursive Partition）的方式不断将商品集合划分为两个规模相近的子集合，并依次构建对应的左右子树，最终得到一棵近似完全二叉树。相比完全随机构建树结构，这种初始化方式能够使语义相近的商品聚集在同一子树中，从而提高模型初始训练效果。

然而，仅依赖人工类别构建的树结构并不能充分反映商品之间真实的语义关系。随着模型训练不断进行，商品 Embedding 逐渐能够学习到更加准确的语义表示，因此论文进一步提出了树结构学习（Tree Learning）机制，使推荐树能够随着 Embedding 不断更新。具体而言，当模型训练收敛后，首先利用当前模型学习得到的叶子节点 Embedding 作为商品表示，然后采用 K-Means 聚类算法重新组织整个商品集合。聚类过程中，每一步均将当前商品集合划分为两个规模尽可能相等的子集合，并递归地继续划分，直到每个叶子节点仅对应一个商品，从而重新生成一棵新的平衡二叉树。由于聚类依据的是模型学习得到的 Embedding，因此新的树结构能够更加真实地反映商品之间的语义相似关系，使相似商品逐渐聚集到相邻的子树中。

完成树结构更新之后，模型再次基于新的推荐树重新进行训练。论文采用模型参数与树结构交替优化（Alternating Optimization）的方式，不断重复“模型训练 → 树结构重建 → 模型重新训练”这一过程，直到模型性能收敛。整个过程可表示为：

1. 根据商品类别初始化推荐树；
2. 基于当前推荐树训练 TDM 模型；
3. 利用学习得到的商品 Embedding 重新聚类生成新的推荐树；
4. 基于新的推荐树重新训练模型；

5. 重复步骤 3～4 直至模型收敛。

这种树模型联合优化策略也是 TDM 召回区别于传统树索引方法的重要特点。传统树索引通常采用固定的层次结构，树构建完成后不会再发生变化；而 TDM 召回将树结构视为模型参数的一部分，使树能够随着 Embedding 不断演化，从而逐渐形成更加符合商品语义分布的层次结构。经过多轮迭代之后，相似商品会自然聚集到同一子树，不仅能够提高各层节点判别模型的训练效果，也能够提升最终在线层次化检索的召回准确率。

10.6.2.5 TDM 召回工程落地

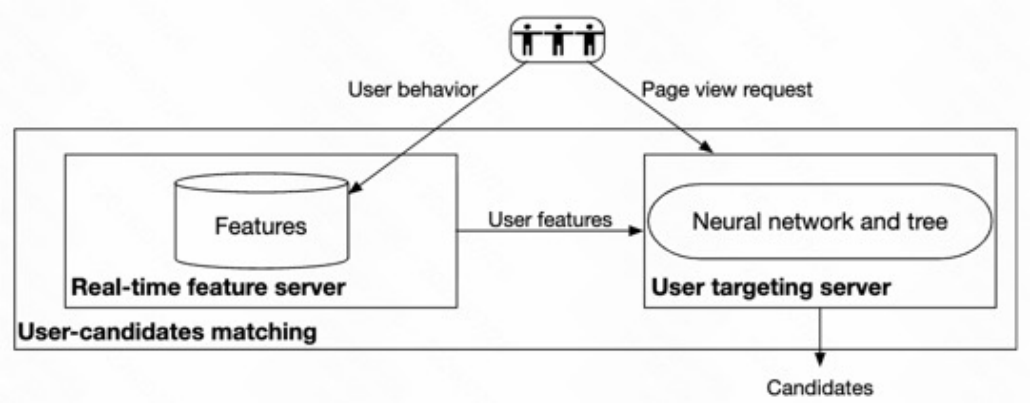

图 10.40: TDM 召回在线 Serving 架构示意图。

TDM 召回的在线 Serving 架构如图10.40所示。相比传统 Embedding 召回需要直接在数千万甚至数十亿规模的物品库中进行最近邻检索，TDM 召回利用层次化推荐树将大规模召回问题分解为多个局部二分类问题，从而显著降低了在线计算复杂度。TDM 召回的在线推理阶段无需遍历整个物品集合，而是按照树结构逐层完成节点筛选，仅保留每一层得分最高的少量候选节点，并继续向下一层展开搜索。具体而言，当用户发起推荐请求后，推荐系统首先根据用户画像、历史行为序列以及上下文信息构建用户兴趣表示，并将根节点作为初始候选节点输入深度模型进行打分。随后，深度模型预测用户对根节点所有子节点的兴趣概率，并保留当前层得分最高的 Top-K 个节点。进入下一层后，仅对这 K 个节点对应的所有子节点继续进行兴趣预测，再次选取得分最高的 Top-K 个节点，如此不断重复，直到遍历至叶子节点所在层。最终保留下来的 Top-K 叶子节点对应的商品即作为最终召回结果。

整个在线检索过程实际上是一种自顶向下（Top-Down）的层次化束搜索（Beam Search）过程，其基本流程如下：

1. 将根节点作为初始候选节点；
2. 利用深度模型预测当前候选节点所有子节点的兴趣概率；
3. 保留当前层得分最高的 Top-K 个节点；
4. 对保留下来的节点继续展开下一层搜索；
5. 重复上述过程，直至到达叶子节点；
6. 输出最终 Top-K 个叶子节点对应的商品作为召回结果。

假设推荐树为完全二叉树，物品规模为 $|I|$，树高约为 $\log_2|I|$。由于每一层仅需扩展当前 Top-K 个节点，因此整个检索过程中需要访问的节点数量约为 $O(K\log|I|)$ 远低于传统全量遍历或大规模 Softmax 计算所需的 $O(|I|)$ 复杂度。当物品规模达到千万甚至数十亿级别时，这种层次化检索能够显著降低在线计算开销，使复杂深度模型也能够满足工业推荐系统毫秒级在线响应要求。

除了降低检索复杂度之外，TDM 召回最大的工程价值在于解耦了模型复杂度与检索效率之间的矛盾。传统 Embedding 召回通常要求用户表示与物品表示满足向量内积形式，以便利用 Faiss、HNSW 等 ANN 索引完成快速检索，因此模型大多采用双塔结构，表达能力受到一定限制。而 TDM 召回并不依赖向量内积检索，而是在每个树节点上完成一次独立的二分类判断，因此可以采用 Attention、Transformer、DIN 甚至更复杂的神经网络结构

学习用户兴趣，而不会影响整体在线检索效率。这使得 TDM 召回能够充分利用丰富的行为序列、用户画像以及上下文特征，显著提升模型表达能力。

总体来看，TDM 召回首次将大规模推荐召回转化为树上的层次化搜索问题，并提出了树结构与深度模型联合优化的整体框架。一方面，推荐树不断根据商品 Embedding 迭代优化，使相似商品逐渐聚集到同一子树；另一方面，在线 Serving 阶段利用逐层筛选策略，仅需访问极少量树节点即可完成大规模候选检索，从而兼顾了召回精度、模型表达能力以及在线检索效率。TDM 召回不仅突破了传统 Embedding 召回必须采用双塔内积检索的限制，也为后续树索引召回、层次化检索以及生成式推荐等方向的发展提供了重要启发。

10.7 召回模型实践讨论

前面的几个章节系统性介绍了工业级推荐系统中常见的协同过滤召回、图召回、Embedding 向量召回、序列召回、多兴趣召回以及其他典型召回模型。从这些方法的发展过程可以看出，不同召回模型各有优势，也各自适用于不同的业务场景。在实际工业级推荐系统中，召回模型的设计并不存在绝对最优的方法，而是需要综合考虑模型效果、系统性能以及工程成本等多个因素。因此，在模型选型与架构设计过程中，通常需要重点关注以下几个方面。

首先，需要综合权衡**召回精度与召回覆盖率**。不同召回模型对于用户兴趣的建模能力存在明显差异。例如，协同过滤召回能够充分利用历史交互信息，在用户行为较丰富的场景下通常具有较高的召回精度，但对于冷启动用户和长尾物品往往存在较大的局限性；图召回能够利用图结构建模多跳关系，发现更加隐式的兴趣传播路径，从而提升长尾物品的覆盖能力；Embedding 召回则能够学习用户和物品的深层语义表示，在保证召回效率的同时获得较好的泛化能力。因此，在实际业务中，通常会部署多路召回，通过不同召回模型优势互补，共同提升整体召回质量。

其次，召回模型还需要**兼顾计算复杂度与在线响应时间**。召回模块位于推荐系统级联架构的最上游，需要面对海量物品库完成毫秒级甚至亚毫秒级的候选检索，因此在线推理效率往往比模型复杂度更加重要。例如，Embedding 召回通常依赖 ANN 近似最近邻索引完成快速检索，而序列召回、多兴趣召回等方法虽然能够学习更加准确的用户表示，但最终仍然需要映射到 Embedding 空间进行向量检索。对于图召回、多跳召回等模型，则通常需要提前构建离线索引或缓存结构，将复杂计算提前完成，从而保证在线服务的实时性。因此，一个优秀的召回模型不仅需要具有较高的召回效果，还需要具备良好的线上推理能力。

除此之外，**召回模型的可扩展性与工程维护成本**也是工业界非常关注的问题。随着用户规模和物品规模不断增长，Embedding 索引、图结构以及召回缓存都需要持续更新和维护。例如，Embedding 召回需要定期重新训练模型并更新 ANN 索引；图召回需要持续维护动态图结构以及边关系；而多兴趣召回、序列召回等模型还需要维护用户兴趣状态和行为序列缓存。这些都会直接影响模型上线后的维护成本。因此，在工业实践中，模型效果并不是唯一的评价标准，工程复杂度、系统稳定性以及维护成本同样需要综合考虑。

另一方面，**不同业务场景对于召回模型的需求**也存在较大的差异。例如，在电商推荐中，更关注用户的购买意图和转化效果，因此通常更加重视商品语义、购买行为以及用户长期兴趣建模；而在短视频、直播等内容推荐场景中，则更加关注内容的新颖性、多样性以及实时兴趣变化，因此更倾向于采用序列召回、多兴趣召回以及图召回等能够建模兴趣动态变化的方法。此外，不同业务对于召回数量、响应时间以及覆盖率的要求也有所不同，因此召回模型往往需要围绕具体业务目标进行定制化设计。

最后，**数据特征与用户行为模式**同样决定了召回模型的适用范围。对于用户行为较丰富的数据集，协同过滤、Embedding 召回通常能够取得较好的效果；而对于关系结构较复杂的数据，图召回能够充分挖掘高阶关联关系；对于兴趣变化速度较快的业务，则序列召回、多兴趣召回能够更加准确地刻画用户当前兴趣。因此，在工业实践中，算法工程师通常需要充分分析数据分布、用户行为特点以及业务特性，再选择最适合当前业务的召回方案，而不是简单地追求更加复杂的模型结构。

总体来看，召回模型的发展经历了从协同过滤到 Embedding 学习，再到序列建模、多兴趣建模以及图结构建模的发展过程。早期的协同过滤主要学习用户与物品之间的一跳共现关系，随后逐渐扩展到多跳关联关系，并进一步发展出了图召回方法，用于建模更加复杂的高阶兴趣传播路径。与此同时，随着 Embedding 学习技术的

发展，无论是在自然语言处理还是推荐系统领域，都逐渐形成了“万物皆可 Embedding”的思想。Embedding 召回也因此成为工业级推荐系统中最主流的召回范式，并进一步衍生出了双塔召回、多兴趣召回以及序列召回等大量工作。后续研究的重点，也逐渐从设计不同的召回结构，转向如何提升用户侧 Encoder 的建模能力，通过引入 GRU、Transformer、BERT、对比学习等技术学习更加准确的用户兴趣表示，从而提升 Embedding 质量以及最终召回效果。

值得强调的是，在工业级推荐系统中，召回模型真正落地往往并不像论文中描述得那样简单。相比模型结构本身，工程实践中更大的挑战来自于召回模型所处的位置。召回模块位于整个级联推荐系统的最上游，其输出结果还需要经过粗排、精排、重排以及大量业务策略的层层过滤，因此召回模型在实际迭代过程中遇到的最大问题往往不是模型训练失败，而是模型上线之后“没有效果”或者“透出占比太低”。很多情况下，即使召回模型本身已经取得了不错的离线指标，由于后续排序模型、过滤策略或者业务规则的影响，其优势也可能无法最终传递到线上结果。

因此，工业界算法工程师除了需要具备扎实的召回模型设计能力之外，更重要的是具备完整的推荐链路分析能力。一方面，需要深入理解模型训练、特征设计以及超参数优化等算法细节；另一方面，还需要能够结合整个推荐链路进行问题定位，分析究竟是哪一个下游模型、哪一种过滤策略或者哪一个业务规则限制了当前召回模型的效果。只有能够快速定位召回模型在线上链路中的瓶颈，并针对性地进行优化和改进，才能真正提升召回模型的线上收益，并最终在 A/B 实验中获得稳定的业务增益。这也是工业级推荐系统研发与学术研究之间最大的区别之一，更是算法工程师需要不断积累的重要实践经验。

10.8 本章小结

本章节系统介绍了工业级推荐系统中的召回模型，并从协同过滤召回、图召回、Embedding 向量召回、序列召回、多兴趣召回以及其他典型召回模型等多个角度，对当前工业界主流的召回技术进行了全面梳理。在协同过滤召回部分，我们首先介绍了最基础的 UserCF 和 ItemCF 算法，并进一步讨论了工业界广泛应用的 Online CF 召回。相比传统离线全量 ItemCF，Online CF 能够更好地兼顾工业级推荐系统对于存储成本、计算效率以及在线时延等方面的要求。此外，本章还介绍了 Swing 召回、矩阵分解召回、NeuralCF 召回以及 Heterogeneous CF 召回等经典方法，并重点分析了 Swing 召回在工业推荐系统中的工程实现与实践经验。

在图召回部分，本章系统介绍了 DeepWalk、Node2Vec、EGES 等经典图表示学习方法，并进一步介绍了 GCN、GAT、GraphSAGE 以及 LightGCN 等代表性的图神经网络模型。同时，也结合工业实践，对 Pinterest 提出的 PinSage 以及快手提出的 HybridGNN 等已经成功落地的图召回方案进行了介绍。

在 Embedding 向量召回部分，本章详细介绍了 Embedding 召回的基本原理、模型训练方式、ANN 近似最近邻索引以及工业级工程落地方案。在模型方面，Embedding 向量召回部分重点介绍了 YoutubeDNN 与双塔模型这两种最经典的 Embedding 召回范式；与此同时，还介绍了近年来广泛应用于语义召回的预训练模型，包括 FastText、BERT、Sentence-BERT、SimCSE 以及 E5 等方法，并对 Sentence Transformers 这一经典 Embedding 训练框架进行了介绍。同时也针对这些语义召回预训练模型如何在推荐场景中作召回进行了具体的案例分析与代码讲解。

在序列召回部分，本章介绍了 List2Vec 这一基于 Item2Vec 扩展得到的序列召回方法，并进一步介绍了基于 RNN 和 Transformer 两条技术路线发展的序列召回模型。其中包括 GRU4Rec、SASRec、BERT4Rec 等经典序列推荐模型，同时也介绍了 SDM 召回以及对比学习序列召回中的 CL4SRec、ICSRec 等代表性工作，展示了近年来序列建模技术以及对比学习技术在召回模型迭代中的发展趋势。

在多兴趣召回部分，本章重点介绍了 MIND 和 ComiRec 两种经典的多兴趣建模方法，并分析了动态兴趣建模、多兴趣表示学习以及兴趣聚合策略在工业推荐系统中的实际应用价值。最后，在其他召回模型部分，本章介绍了 PDN 这一经典的 U2I2I 多跳路径召回模型，以及工业界广泛应用的 TDM 树索引召回方法，并重点讨论了 TDM 召回的树结构构建、层次化检索策略、深度网络设计以及在线 Serving 架构，进一步展示了工业级召回模型在大规模检索场景下的工程实现思路。

总体来看，召回模块位于整个推荐系统级联架构的最上游，其主要目标是在海量物品库中快速筛选出少量

高质量候选，为后续粗排、精排以及重排模型提供输入。因此，工业级推荐系统通常会采用多路召回协同工作的方式，充分发挥不同召回模型在兴趣建模、语义表达、关系传播以及长尾覆盖等方面的优势，从而兼顾召回精度、覆盖率与多样性。本章对当前工业界主流召回模型进行了较为系统的介绍，为后续排序模型的学习奠定了基础。

下一章节将进入推荐系统排序阶段的介绍，重点讨论粗排模型的设计与实现。与召回阶段通常采用多路模型共同工作的架构不同，排序阶段无论是粗排还是精排，通常都围绕一个主模型进行持续迭代优化。因此，在后续章节中，我们将重点介绍工业级推荐系统中粗排与精排模型的发展脉络、经典网络结构以及工程实践方法。

10.9 参考文献

[1] Yauhen Babakhin et al. “Llama-Embed-Nemotron-8B: A Universal Text Embedding Model for Multilingual and Cross-Lingual Tasks”. In: *arXiv preprint arXiv:2511.07025* (2025).

[2] Oren Barkan and Noam Koenigstein. “Item2vec: neural item embedding for collaborative filtering”. In: *2016 IEEE 26th international workshop on machine learning for signal processing (MLSP)*. IEEE. 2016, pp. 1–6.

[3] Piotr Bojanowski et al. “Enriching word vectors with subword information”. In: *Transactions of the association for computational linguistics* 5 (2017), pp. 135–146.

[4] Shaked Brody, Uri Alon, and Eran Yahav. “How attentive are graph attention networks?” In: *arXiv preprint arXiv:2105.14491* (2021).

[5] Yukuo Cen et al. “Controllable multi-interest framework for recommendation”. In: *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 2020, pp. 2942–2951.

[6] Chong Chen et al. “Efficient heterogeneous collaborative filtering without negative sampling for recommendation”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 34. 01. 2020, pp. 19–26.

[7] Gaode Chen et al. “Exploring periodicity and interactivity in multi-interest framework for sequential recommendation”. In: *arXiv preprint arXiv:2106.04415* (2021).

[8] Jie Chen, Tengfei Ma, and Cao Xiao. “Fastgcn: fast learning with graph convolutional networks via importance sampling”. In: *arXiv preprint arXiv:1801.10247* (2018).

[9] Wei-Lin Chiang et al. “Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks”. In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 257–266.

[10] Alexis Conneau et al. “Unsupervised cross-lingual representation learning at scale”. In: *Proceedings of the 58th annual meeting of the association for computational linguistics*. 2020, pp. 8440–8451.

[11] Paul Covington, Jay Adams, and Emre Sargin. “Deep neural networks for youtube recommendations”. In: *Proceedings of the 10th ACM conference on recommender systems*. 2016, pp. 191–198.

[12] Jacob Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 2019, pp. 4171–4186.

[13] Fangxiaoyu Feng et al. “Language-agnostic BERT sentence embedding”. In: *Proceedings of the 60th annual meeting of the association for computational linguistics (volume 1: Long papers)*. 2022, pp. 878–891.

[14] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. “Splade: Sparse lexical and expansion model for first stage ranking”. In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2021, pp. 2288–2292.

[15] Tianyu Gao, Xingcheng Yao, and Danqi Chen. “Simcse: Simple contrastive learning of sentence embeddings”. In: *Proceedings of the 2021 conference on empirical methods in natural language processing*. 2021, pp. 6894–6910.

[16] Aditya Grover and Jure Leskovec. “node2vec: Scalable feature learning for networks”. In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 2016, pp. 855–864.

[17] Tiankai Gu et al. “Hybridgnn: Learning hybrid representation in multiplex heterogeneous networks”. In: *arXiv preprint arXiv:2208.02068* (2022).

[18] Will Hamilton, Zhitao Ying, and Jure Leskovec. “Inductive representation learning on large graphs”. In: *Advances in neural information processing systems* 30 (2017).

[19] Pengcheng He et al. “Deberta: Decoding-enhanced bert with disentangled attention”. In: *arXiv preprint arXiv:2006.03654* (2020).

[20] Xiangnan He et al. “Lightgcn: Simplifying and powering graph convolution network for recommendation”. In: *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 2020, pp. 639–648.

[21] Xiangnan He et al. “Neural collaborative filtering”. In: *Proceedings of the 26th international conference on world wide web*. 2017, pp. 173–182.

[22] Balázs Hidasi et al. “Session-based recommendations with recurrent neural networks”. In: *arXiv preprint arXiv:1511.06939* (2015).

[23] Yifan Hu, Yehuda Koren, and Chris Volinsky. “Collaborative filtering for implicit feedback datasets”. In: *2008 Eighth IEEE international conference on data mining*. Ieee. 2008, pp. 263–272.

[24] Ziniu Hu et al. “Heterogeneous graph transformer”. In: *Proceedings of the web conference 2020*. 2020, pp. 2704–2710.

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

[26] Thomas N Kipf and Max Welling. “Semi-supervised classification with graph convolutional networks”. In: *arXiv preprint arXiv:1609.02907* (2016).

[27] Yehuda Koren. “Factorization meets the neighborhood: a multifaceted collaborative filtering model”. In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2008, pp. 426–434.

[28] Yehuda Koren, Robert Bell, and Chris Volinsky. “Matrix factorization techniques for recommender systems”. In: *Computer* 42.8 (2009), pp. 30–37.

[29] Zhenzhong Lan et al. “Albert: A lite bert for self-supervised learning of language representations”. In: *arXiv preprint arXiv:1909.11942* (2019).

[30] Chankyu Lee et al. “Nv-embed: Improved techniques for training llms as generalist embedding models”. In: *International Conference on Learning Representations*. Vol. 2025. 2025, pp. 79310–79333.

[31] Chao Li et al. “Multi-interest network with dynamic routing for recommendation at Tmall”. In: *Proceedings of the 28th ACM international conference on information and knowledge management*. 2019, pp. 2615–2623.

[32] Houyi Li et al. “Path-based deep network for candidate item matching in recommenders”. In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2021, pp. 1493–1502.

[33] Zehan Li et al. “Towards general text embeddings with multi-stage contrastive learning”. In: *arXiv preprint arXiv:2308.03281* (2023).

[34] Yinhan Liu et al. “Roberta: A robustly optimized bert pretraining approach”. In: *arXiv preprint arXiv:1907.11692* (2019).

[35] Fuyu Lv et al. “SDM: Sequential deep matching model for online large-scale recommender system”. In: *Proceedings of the 28th ACM international conference on information and knowledge management*. 2019, pp. 2635–2643.

[36] Tomas Mikolov et al. “Efficient estimation of word representations in vector space”. In: *arXiv preprint arXiv:1301.3781* (2013).

[37] Andriy Mnih and Russ R Salakhutdinov. “Probabilistic matrix factorization”. In: *Advances in neural information processing systems* 20 (2007).

[38] Rodrigo Nogueira et al. “Document ranking with a pretrained sequence-to-sequence model”. In: *Findings of the association for computational linguistics: EMNLP 2020*. 2020, pp. 708–718.

[39] Zach Nussbaum et al. “Nomic embed: Training a reproducible long context text embedder”. In: *arXiv preprint arXiv:2402.01613* (2024).

[40] Jeffrey Pennington, Richard Socher, and Christopher D Manning. “Glove: Global vectors for word representation”. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[41] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. “Deepwalk: Online learning of social representations”. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014, pp. 701–710.

[42] Ronak Pradeep, Rodrigo Nogueira, and Jimmy Lin. “The expando-mono-duo design pattern for text ranking with pretrained sequence-to-sequence models”. In: *arXiv preprint arXiv:2101.05667* (2021).

[43] Xiuyuan Qin et al. “Intent contrastive learning with cross subsequences for sequential recommendation”. In: *Proceedings of the 17th ACM international conference on web search and data mining*. 2024, pp. 548–556.

[44] Nils Reimers and Iryna Gurevych. “Sentence-bert: Sentence embeddings using siamese bert-networks”. In: *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*. 2019, pp. 3982–3992.

[45] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. “Dynamic routing between capsules”. In: *Advances in neural information processing systems* 30 (2017).

[46] Yunsheng Shi et al. “Masked label prediction: Unified message passing model for semi-supervised classification”. In: *arXiv preprint arXiv:2009.03509* (2020).

[47] Kaitao Song et al. “Mpnet: Masked and permuted pre-training for language understanding”. In: *Advances in neural information processing systems* 33 (2020), pp. 16857–16867.

[48] Fei Sun et al. “BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer”. In: *Proceedings of the 28th ACM international conference on information and knowledge management*. 2019, pp. 1441–1450.

[49] Qiaoyu Tan et al. “Sparse-interest network for sequential recommendation”. In: *Proceedings of the 14th ACM international conference on web search and data mining*. 2021, pp. 598–606.

[50] Petar Veličković et al. “Graph attention networks”. In: *arXiv preprint arXiv:1710.10903* (2017).

[51] Henrique Schechter Vera et al. “Embeddinggemma: Powerful and lightweight text representations”. In: *arXiv preprint arXiv:2509.20354* (2025).

[52] Jizhe Wang et al. “Billion-scale commodity embedding for e-commerce recommendation in alibaba”. In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2018, pp. 839–848.

[53] Liang Wang et al. “Improving text embeddings with large language models”. In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2024, pp. 11897–11916.

[54] Liang Wang et al. “Text embeddings by weakly-supervised contrastive pre-training”. In: *arXiv preprint arXiv:2212.03533* (2022).

[55] Wenhui Wang et al. “Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers”. In: *Advances in neural information processing systems* 33 (2020), pp. 5776–5788.

[56] Shitao Xiao et al. “C-pack: Packed resources for general chinese embeddings”. In: *Proceedings of the 47th international ACM SIGIR conference on research and development in information retrieval*. 2024, pp. 641–649.

[57] Xu Xie et al. “Contrastive learning for sequential recommendation”. In: *2022 IEEE 38th international conference on data engineering (ICDE)*. IEEE. 2022, pp. 1259–1273.

[58] Xiaoyong Yang et al. “Large scale product graph construction for recommendation in e-commerce”. In: *arXiv preprint arXiv:2010.05525* (2020).

[59] Xinyang Yi et al. “Sampling-bias-corrected neural modeling for large corpus item recommendations”. In: *Proceedings of the 13th ACM conference on recommender systems*. 2019, pp. 269–277.

[60] Rex Ying et al. “Graph convolutional neural networks for web-scale recommender systems”. In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2018, pp. 974–983.

[61] Hanqing Zeng et al. “Graphsaint: Graph sampling based inductive learning method”. In: *arXiv preprint arXiv:1907.04931* (2019).

[62] Yanzhao Zhang et al. “Qwen3 embedding: Advancing text embedding and reranking through foundation models”. In: *arXiv preprint arXiv:2506.05176* (2025).

[63] Zhengyan Zhang et al. “ERNIE: Enhanced language representation with informative entities”. In: *Proceedings of the 57th annual meeting of the association for computational linguistics*. 2019, pp. 1441–1451.

[64] Han Zhu et al. “Learning tree-based deep model for recommender systems”. In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2018, pp. 1079–1088.