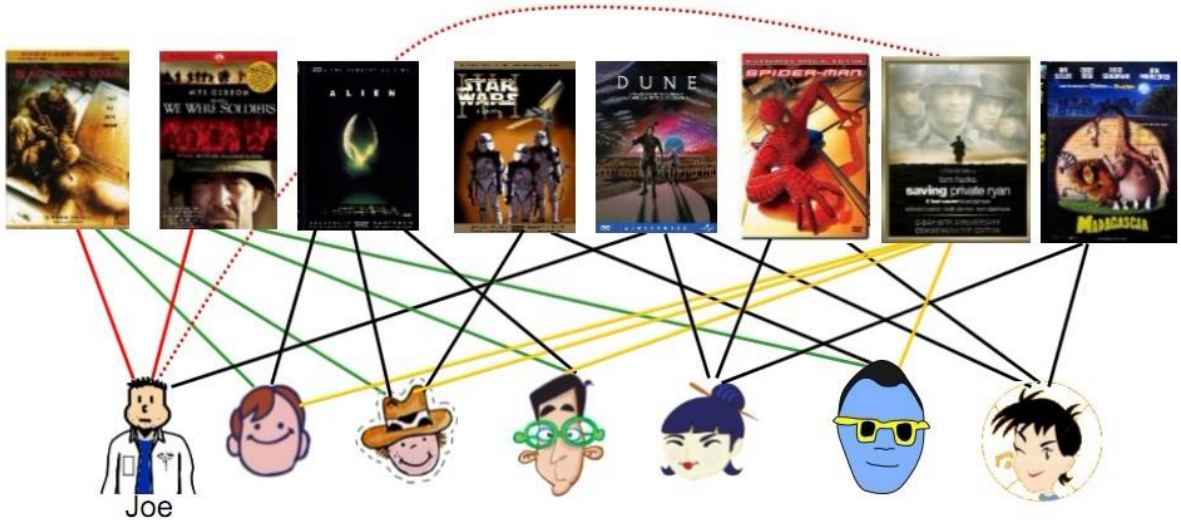
推荐系统-常考面试题

1、请详细说说协同过滤的原理

解析:
1 推荐引擎的分类
推荐引擎根据不同依据如下分类:
根据其是不是为不同的用户推荐不同的数据，分为基于大众行为（网站管理员自行推荐，或者基于系统所有用户的反馈统计计算出的当下比较流行的物品）、及个性化推荐引擎（帮你找志同道合，趣味相投的朋友，然后在此基础上实行推荐）;
根据其数据源，分为基于人口统计学的（用户年龄或性别相同判定为相似用户）、基于内容的（物品具有相同关键词和 Tag，没有考虑人为因素），以及基于协同过滤的推荐（发现物品，内容或用户的相关性推荐，分为三个子类，下文阐述）;
根据其建立方式，分为基于物品和用户本身的（用户-物品二维矩阵描述用户喜好，聚类算法）、基于关联规则的（The Apriori algorithm 算法是一种最有影响的挖掘布尔关联规则频繁项集的算法）、以及基于模型的推荐（机器学习，所谓机器学习，即让计算机像人脑一样持续学习，是人工智能领域内的一个子领域）。
关于上述第二个分类(2、根据其数据源)中的基于协同过滤的推荐：随着 Web2.0 的发展，Web 站点更加提倡用户参与和用户贡献，因此基于协同过滤的推荐机制因运而生。它的原理很简单，就是根据用户对物品或者信息的偏好，发现物品或者内容本身的相关性，或者是发现用户的相关性，然后再基于这些关联性进行推荐。
而基于协同过滤的推荐，又分三个子类:
基于用户的推荐(通过共同口味与偏好找相似邻居用户，K-邻居算法，你朋友喜欢，你也可能喜欢),
基于项目的推荐(发现物品之间的相似度，推荐类似的物品，你喜欢物品 A，C 与 A 相似，可能也喜欢 C),
基于模型的推荐(基于样本的用户喜好信息构造一个推荐模型，然后根据实时的用户喜好信息预测推荐)。
我们看到，此协同过滤算法最大限度的利用用户之间，或物品之间的相似相关性，而后基于这些信息的基础上实行推荐。下文还会具体介绍此协同过滤。
不过一般实践中，我们通常还是把推荐引擎分两类:
第一类称为协同过滤，即基于相似用户的协同过滤推荐（用户与系统或互联网交互留下的一切信息、蛛丝马迹，或用户与用户之间千丝万缕的联系），以及基于相似项目的协同过滤推荐（尽最大可能发现物品间的相似度）;
第二类便是基于内容分析的推荐（调查问卷，电子邮件，或者推荐引擎对本 blog 内容的分析）。
2 协同过滤推荐
协同过滤是利用集体智慧的一个典型方法。要理解什么是协同过滤 (Collaborative Filtering，简称 CF)，首先想一个简单的问题，如果你现在想看个电影，但你不知道具体看哪部，你会怎么做？大部分的人会问问周围的朋友或者称之为广义上的邻居(neighborhood)，看看最近有什么好看的电影推荐，而我们一般更倾向于从口味比较类似的朋友那里得到推荐。这就是协同过滤的核心思想。如下图，你能从图中看到多少信息?



2.1 协同过滤推荐步骤

做协同过滤推荐，一般要做好以下几个步骤：

1）若要做协同过滤，那么收集用户偏好则成了关键。可以通过用户的行为诸如评分（如不同的用户对不同的作品有不同的评分，而评分接近则意味着喜好口味相近，便可判定为相似用户），投票，转发，保存，书签，标记，评论，点击流，页面停留时间，是否购买等获得。如下面第 2 点所述：所有这些信息都可以数字化，如一个二维矩阵表示出来。

2）收集了用户行为数据之后，我们接下来便要对数据进行减噪与归一化操作(得到一个用户偏好的二维矩阵，一维是用户列表，另一维是物品列表，值是用户对物品的偏好，一般是 [0,1] 或者 [-1, 1] 的浮点数值)。

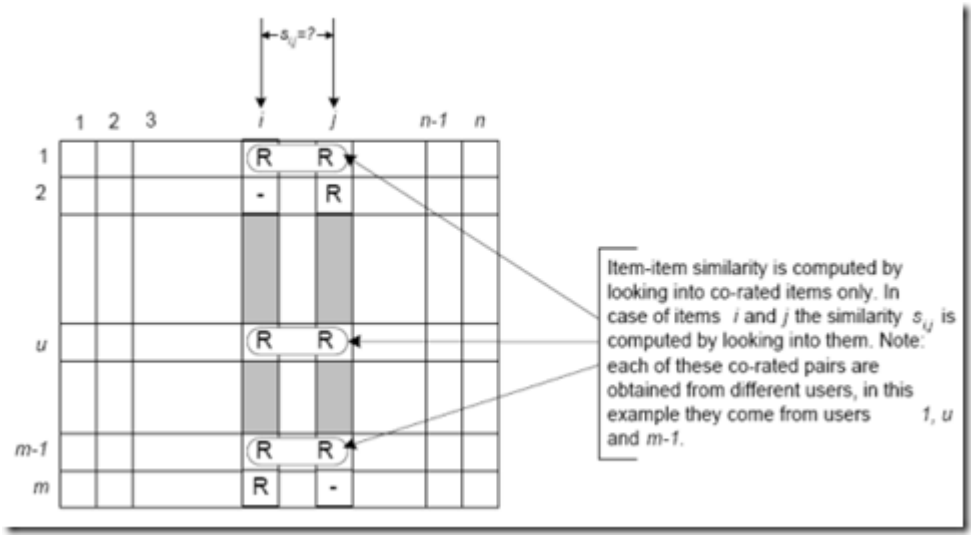
下面再简单介绍下减噪和归一化操作。

所谓减噪：用户行为数据是用户在使用应用过程中产生的，它可能存在大量的噪音和用户的误操作，我们可以通过经典的数据挖掘算法过滤掉行为数据中的噪音，这样可以是我们的分析更加精确（类似于网页的去噪处理）。

所谓归一化：将各个行为的数据统一在一个相同的取值范围中，从而使得加权求和得到的总体喜好更加精确。最简单的归一化处理，便是将各类数据除以此类中的最大值，以保证归一化后的数据取值在 [0,1] 范围中。至于所谓的加权，很好理解，因为每个人占的权值不同，类似于一场唱歌比赛中对某几个选手进行投票决定其是否晋级，观众的投票抵 1 分，专家评委的投票抵 5 分，最后得分最多的选手直接晋级。

3）找到相似的用户和物品，通过什么途径找到呢？便是计算相似用户或相似物品的相似度。

4）相似度的计算有多种方法，不过都是基于向量 Vector 的，其实也就是计算两个向量的距离，距离越近相似度越大。在推荐中，用户-物品偏好的二维矩阵下，我们将某个或某几个用户对莫两个物品的偏好作为一个向量来计算两个物品之间的相似度，或者将两个用户对某个或某几个物品的偏好作为一个向量来计算两个用户之间的相似度。



所以说，很简单，找物品间的相似度，用户不变，找多个用户对物品的评分；找用户间的相似度，物品不变，找用户对某些个物品的评分。

5）而计算出来的这两个相似度则将作为基于用户、项目的两项协同过滤的推荐。

常见的计算相似度的方法有：欧几里德距离，皮尔逊相关系数（如两个用户对多个电影的评分，采取皮尔逊相关系数等相关计算方法，可以抉择出他们的口味和偏好是否一致），Cosine 相似度，Tanimoto 系数。

下面，简单介绍其中的欧几里得距离与皮尔逊相关系数：

欧几里德距离（Euclidean Distance）是最初用于计算欧几里德空间中两个点的距离，假设 x，y 是 n 维空间的两个点，它们之间的欧几里德距离是：

$$d(x,y)=\sqrt{(\sum(x_i-y_i)^2)}$$

可以看出，当 n=2 时，欧几里德距离就是平面上两个点的距离。当用欧几里德距离表示相似度，一般采用以下公式进行转换：距离越小，相似度越大（同时，避免除数为 0）：

$$sim(x,y)=\frac{1}{1+d(x,y)}$$

余弦相似度 Cosine-based Similarity

两个项目 i ，j 视作为两个 m 维用户空间向量，相似度计算通过计算两个向量的余弦夹角，那么，对于 m*n 的评分矩阵，i ，j 的相似度 sim(i , j ）计算公式：

$$sim(i,j)=\cos(\vec{i},\vec{j})=\frac{\vec{i}\cdot\vec{j}}{\|\vec{i}\|_2*\|\vec{j}\|_2}$$

（其中 " · "记做两个向量的内积）

皮尔逊相关系数一般用于计算两个定距变量间联系的紧密程度，为了使计算结果精确，需要找出共同评分的用户。记用户集 U 为既评论了 i 又评论了 j 的用户集，那么对应的皮尔森相关系数计算公式为：

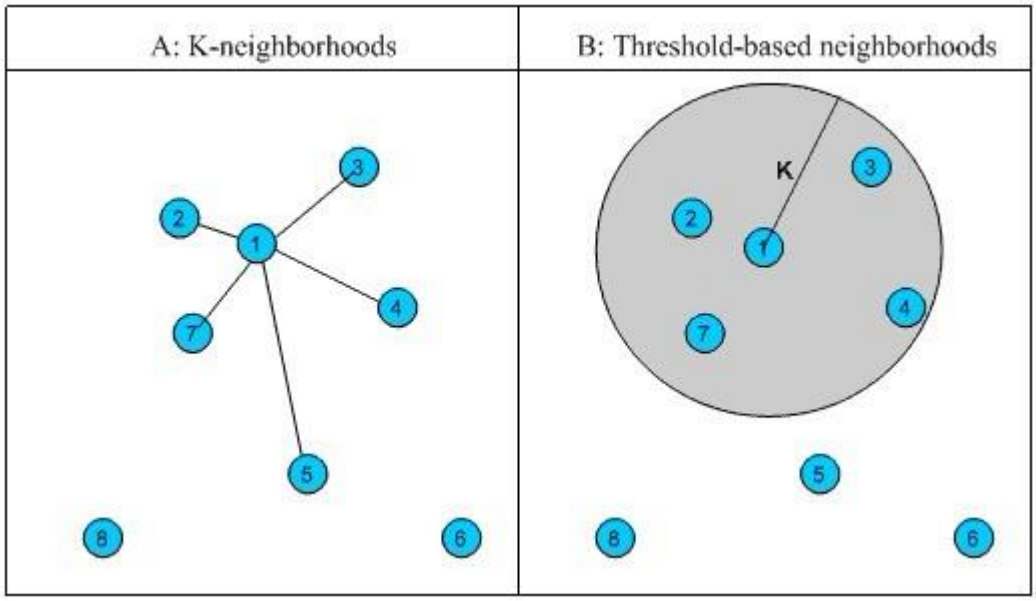
$$sim(i,j)=\frac{\sum_{u\in U}(R_{u,i}-\bar{R}_i)(R_{u,j}-\bar{R}_j)}{\sqrt{\sum_{u\in U}(R_{u,i}-\bar{R}_i)^2}\sqrt{\sum_{u\in U}(R_{u,j}-\bar{R}_j)^2}}$$

其中 Ru,i 为用户 u 对项目 i 的评分，对应带横杠的为这个用户集 U 对项目 i 的评分评分。

6）相似邻居计算。邻居分为两类：

1、固定数量的邻居 K-neighborhoods （或 Fix-size neighborhoods），不论邻居的“远近”，只取最近的 K 个，作为其邻居，如下图 A 部分所示；

2、基于相似度门槛的邻居，落在以当前点为中心，距离为 K 的区域中的所有点都作为当前点的邻居，如下图 B 部分所示。



再介绍一下 K 最近邻(k-Nearest Neighbor，KNN)分类算法：这是一个理论上比较成熟的方法，也是最简单的机器学习算法之一。该方法的思路是：如果一个样本在特征空间中的 k 个最相似(即特征空间中最邻近)的样本中的大多数属于某一个类别，则该样本也属于这个类别。

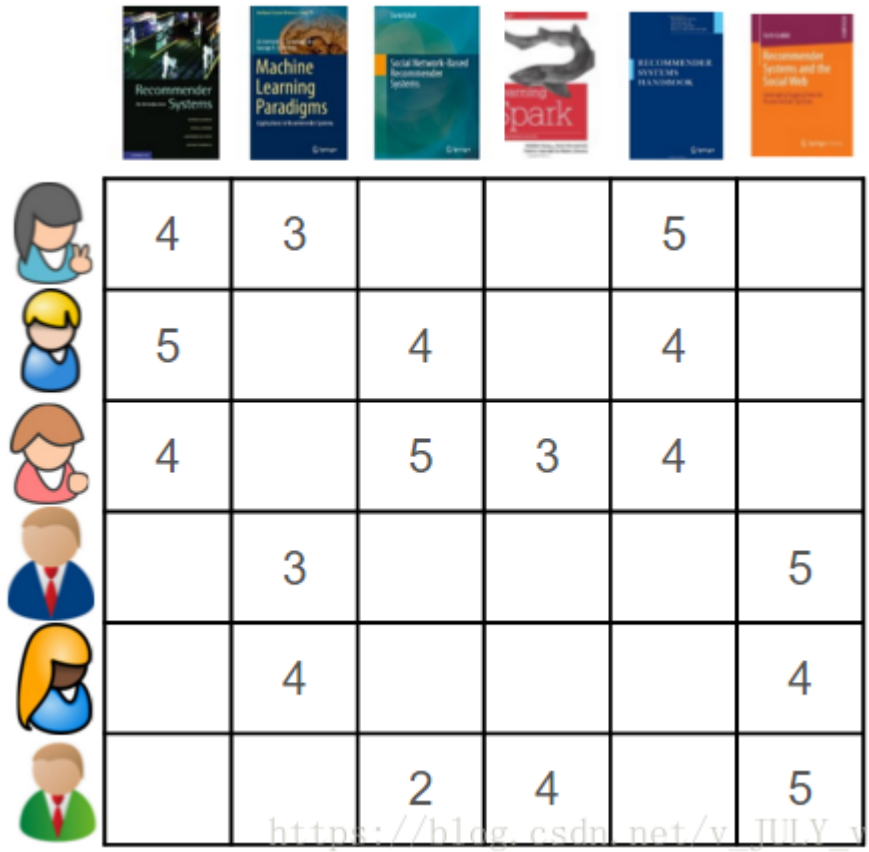
7）经过 4)计算出来的基于用户的 CF(基于用户推荐之用：通过共同口味与偏好找相似邻居用户，K-邻居算法，你朋友喜欢，你也可能喜欢)，基于物品的 CF(基于项目推荐之用：发现物品之间的相似度，推荐类似的物品，你喜欢物品 A，C 与 A 相似，那么你可能也喜欢 C)。

2.2 基于基于用户相似度与项目相似度

上述 3.1 节中三个相似度公式是基于项目相似度场景下的，而实际上，基于用户相似度与基于项目相似度计算的一个基本的区别是，基于用户相似度是基于评分矩阵中的行向量相似度求解，基于项目相似度计算式基于评分矩阵中列向量相似度求解，然后三个公式分别都可以适用，如下图：

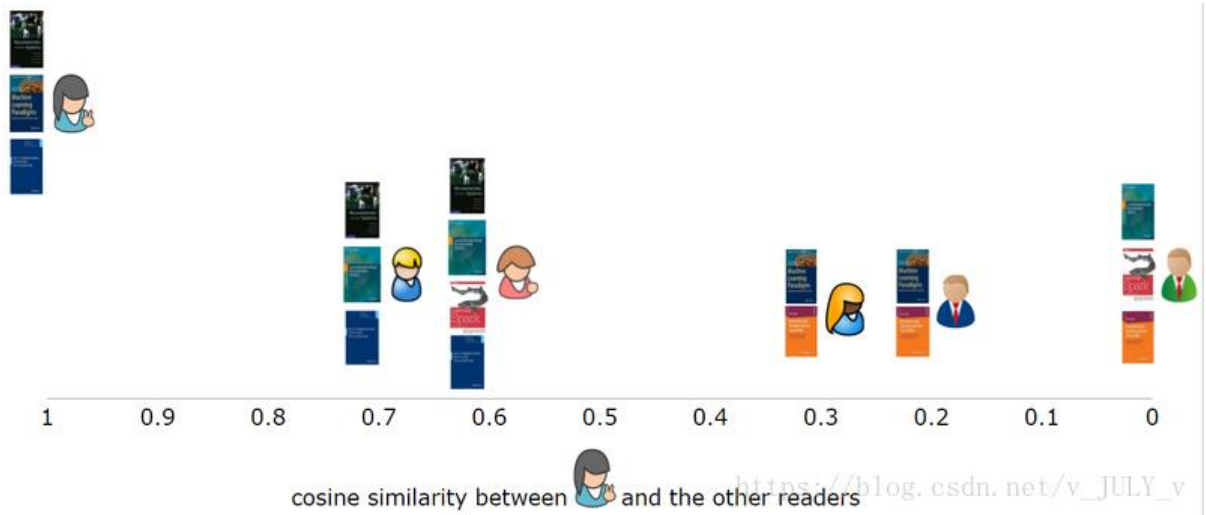
	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
User 1	4	0	1	3	2	3
User 2	3	0	2	3	3	4
User 3	4	5	6	0	2	0
User 4	0	3	1	0	0	2
User 5	5	0	2	0	0	2
User 6	4	2	2	1	2	4

（其中，为 0 的表示未评分）
基于项目相似度计算式计算如 Item3，Item4 两列向量相似度；
基于用户相似度计算式计算如 User3，User4 量行向量相似度。
千言万语不如举个例子。我们来看一个具体的基于用户相似度计算的例子。
假设我们有一组用户，他们表现出了对一组图书的喜好。用户对一本图书的喜好程度越高，就会给其更高的评分。我们来通过一个矩阵来展示它，行代表用户，列代表图书。
如下图所示，所有的评分范围从 1 到 5，5 代表喜欢程度最高。第一个用户（行 1）对第一本图书（列 1）的评分是 4，空的单元格表示用户未给图书评分。

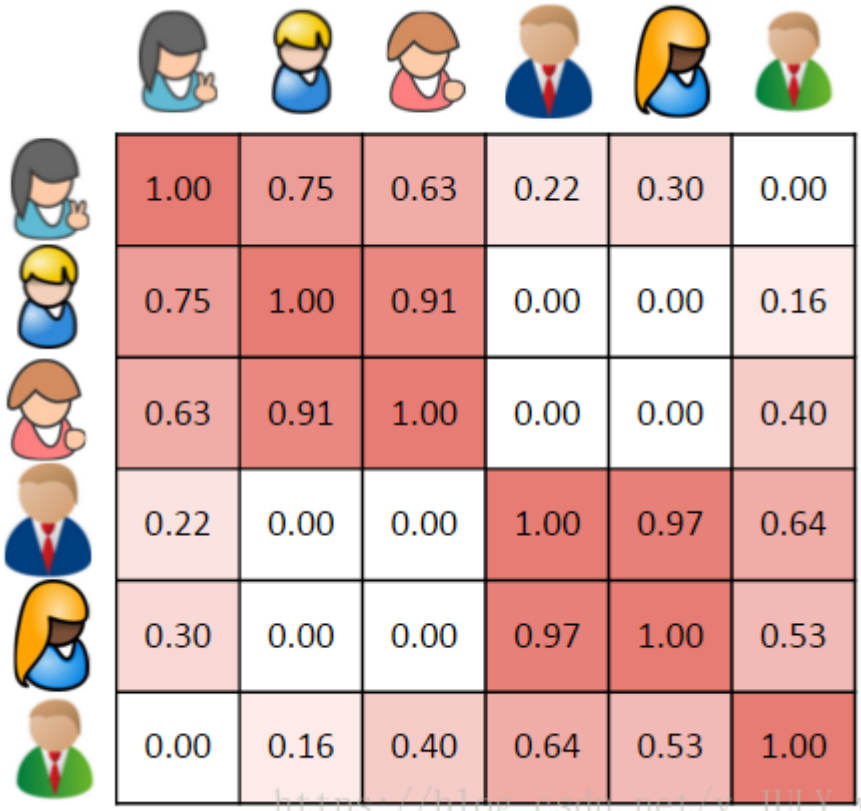


使用基于用户的协同过滤方法，我们首先要做的是基于用户给图书做出的评价，计算用户之间的相似度。

让我们从一个单一用户的角度考虑这个问题，看图 1 中的第一行，要做到这一点，常见的做法是将使用包含了用户喜好项的向量（或数组）代表每一个用户。相较于使用多样化的相似度量这种做法，更直接。
在这个例子中，我们将使用余弦相似性去计算用户间的相似度。
当我们把第一个用户和其他五个用户进行比较时，就能直观的看到他和其他用户的相似程度。对于大多数相似度量，向量之间相似度越高，代表彼此更相似。本例中，第一个用户第二、第三个用户非常相似，有两本共同书籍，与第四、第五个用户的相似度低一些，只有一本共同书籍，而与最后一名用户完全不相似，因为没有一本共同书籍。

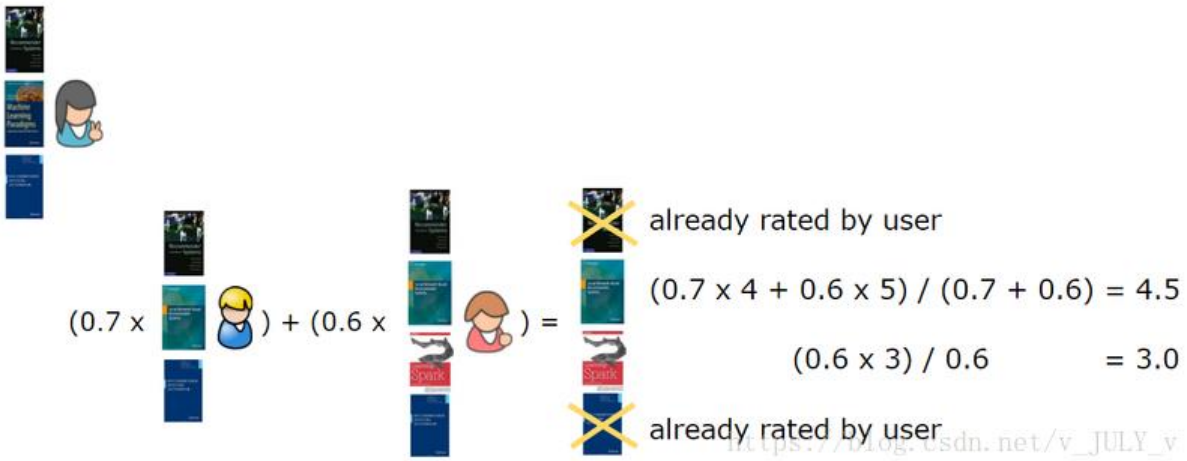


更一般的，我们可以计算出每个用户的相似性，并且在相似矩阵中表示它们。这是一个对称矩阵，单元格的背景颜色表明用户相似度的高低，更深的红色表示它们之间更相似。

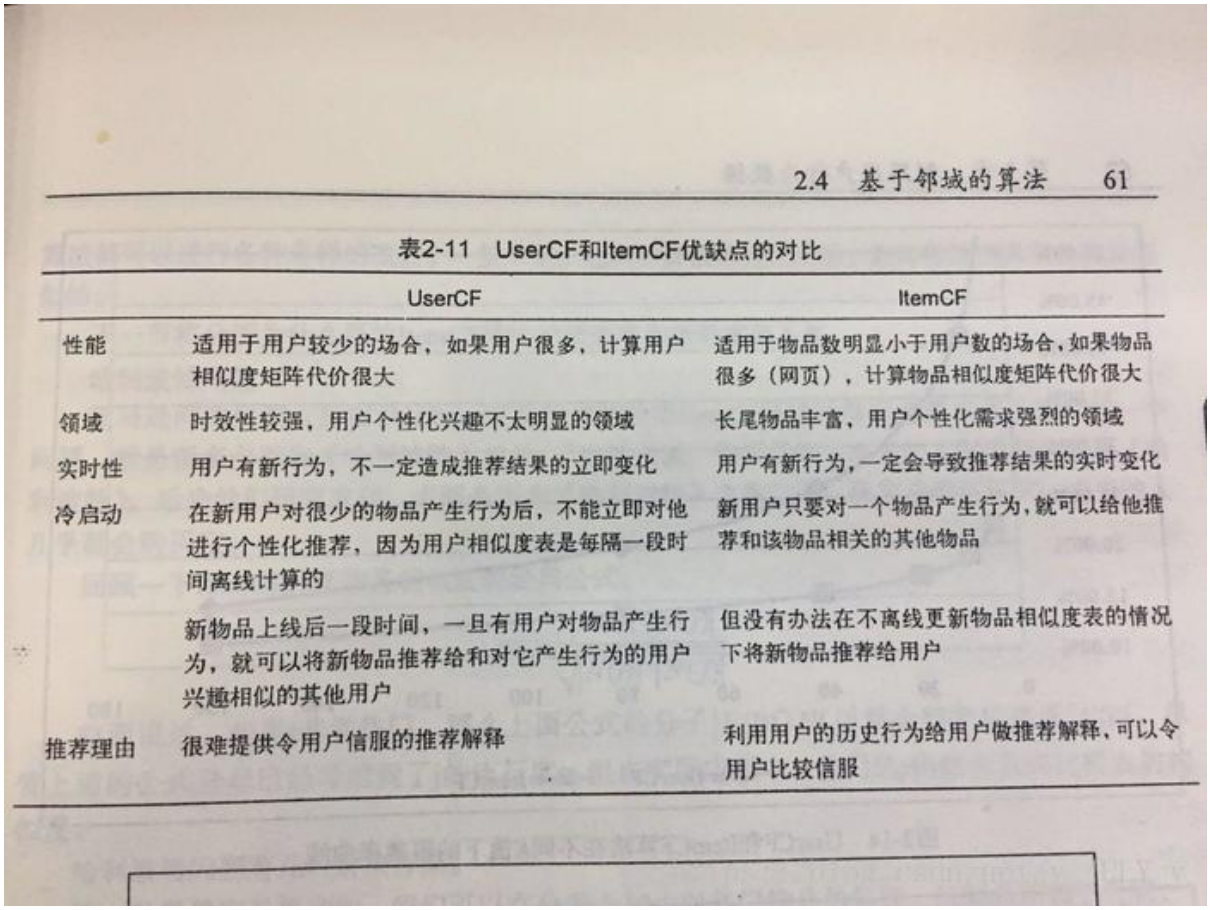


所以，我们找到了与

第一个用户最相似的第二个用户，删除用户已经评价过的书籍，给最相似用户正在阅读的书籍加权，然后计算出总和。
在这种情况下，我们计算出 n=2，表示为了产生推荐，需要找出与目标用户最相似的两个用户，这两个用户分别是第二个和第三个用户，然后第一个用户已经评价了第一和第五本书，故产生的推荐书是第三本（4.5 分），和第四本（3 分）。



此外，什么时候用 item-base，什么时候用 user-base 呢：
http://weibo.com/1580904460/zhZ9AilkZ?mod=weibotime?
一般说来，如果 item 数目不多，比如不超过十万，而且不显著增长的话，就用 item-based 好了。为何？如@wuzh670 所说，如果 item 数目不多+不显著增长，说明 item 之间的关系在一段时间内相对稳定(对比 user 之间关系)，对于实时更新 item-similarity 需求就降低很多，推荐系统效率提高很多，故用 item-based 会明智些。
反之，当 item 数目很多，建议用 user-base。当然，实践中具体情况具体分析。如下图所示（摘自项亮的《推荐系统实践》一书）：



2.4 基于邻域的算法 61

表2-11 UserCF和ItemCF优缺点的对比

	UserCF	ItemCF
性能	适用于用户较少的场合，如果用户很多，计算用户相似度矩阵代价很大	适用于物品数明显小于用户数的场合，如果物品很多（网页），计算物品相似度矩阵代价很大
领域	时效性较强，用户个性化兴趣不太明显的领域	长尾物品丰富，用户个性化需求强烈的领域
实时性	用户有新行为，不一定造成推荐结果的立即变化	用户有新行为，一定会导致推荐结果的实时变化
冷启动	在新用户对很少的物品产生行为后，不能立即对他进行个性化推荐，因为用户相似度表是每隔一段时间离线计算的	新用户只要对一个物品产生行为，就可以给他推荐和该物品相关的其他物品
	新物品上线后一段时间，一旦有用户对物品产生行为，就可以将新物品推荐给和对它产生行为的用户兴趣相似的其他用户	但没有办法在不离线更新物品相似度表的情况下将新物品推荐给用户
推荐理由	很难提供令用户信服的推荐解释	利用用户的历史行为给用户做推荐解释，可以令用户比较信服

3 参考文献
1 推荐引擎算法学习导论：协同过滤、聚类、分类：
https://blog.csdn.net/v_july_v/article/details/7184318
2 协同过滤推荐算法和基于内容的过滤算法：https://time.geekbang.org/article/1947

2、搜索和推荐中的精度和召回(recall)分别是什么意思？

解析：
精度/精确率，和召回率是广泛用于信息检索和统计学分类领域的两个度量值，用来评价结果的质量。
其中精度是检索出相关文档数与检索出的文档总数的比率，衡量的是检索系统的查准率；
召回率是指检索出的相关文档数和文档库中所有的相关文档数的比率，衡量的是检索系统的查全率。
一般来说，Precision 就是检索出来的条目（比如：文档、网页等）有多少是准确的，Recall 就是所有准确的条目有多少被检索出来了。
正确率、召回率和 F 值是在鱼龙混杂的环境中，选出目标的重要评价指标。不妨看看这些指标的定义先：

1. 精确率 = 提取出的正确信息条数 / 提取出的信息条数
2. 召回率 = 提取出的正确信息条数 / 样本中的信息条数

顺便说一句，如果两者取值在 0 和 1 之间，数值越接近 1，查准率或查全率就越高。比如定义：F 值 = 正确率 * 召回率 * 2 / (正确率 + 召回率) （F 值即为正确率和召回率的调和平

均值）

$$k\text{精度}(precision\ at\ k) = \frac{\text{所推荐的}k\text{个物品中相关物品的个数}}{k}$$

$$k\text{召回}(recall\ at\ k) = \frac{\text{所推荐的}k\text{个物品中相关物品的个数}}{\text{所有相关物品的个数}}$$

这就好比推荐系统根据你的喜好，推荐了 10 个商品，其中真正相关的是 5 个商品。在所有商品当中，相关的商品一共有 20 个，那么

k 精度 = 5 / 10

k 召回 = 5 / 20

咱们再看下先第二个例子。比如搜：北京大学，有三个网页被搜索到了：

a. 北京大学保安考上研究生

b. 北京互联网工作招聘

c. 大学生活是什么样的

其中只有 a 是被正确搜索到的，其他两个其实是和用户搜索词无关，而事实上数据库里还有这种网页：

d. 北大开学季

e. 未名湖的景色

这两个没被搜索到，但 d、e 和“北京大学”的相关度是超过 b、c 的，也就是应该被搜索（被召回）到的却没有显示在结果里，即：

精确率 = (a) / (a + b + c)

召回率 = (a) / (a + d + e)

不妨再看第三个例子：某池塘有 1400 条鲤鱼，300 只虾，300 只鳖。现在以捕鲤鱼为目的。撒一大网，逮着了 700 条鲤鱼，200 只虾，100 只鳖。那么，这些指标分别如下：

代表查准率的正确率 = 700 / (700 + 200 + 100) = 70%

代表查全率的召回率 = 700 / 1400 = 50%

F 值 = 70% * 50% * 2 / (70% + 50%) = 58.3%

不妨看看如果把池子里的所有的鲤鱼、虾和鳖都一网打尽，这些指标又有何变化：

正确率 = 1400 / (1400 + 300 + 300) = 70%

召回率 = 1400 / 1400 = 100%

F 值 = 70% * 100% * 2 / (70% + 100%) = 82.35%

由此可见，正确率是评估捕获的成果中目标成果所占得比例；召回率，顾名思义，就是从关注领域中，召回目标类别的比例；而 F 值，则是综合这二者指标的评估指标，用于综合反映整体的指标。

当然希望检索结果 Precision 越高越好，同时 Recall 也越高越好，但事实上这两者在某些情况下有矛盾的。比如极端情况下，我们只搜索出了一个结果，且是准确的，那么 Precision 就是 100%，但是 Recall 就很低；而如果我们把所有结果都返回，那么比如 Recall 是 100%，但是 Precision 就会很低。因此在不同的场合中需要自己判断希望 Precision 比较高或是 Recall 比较高。如果是做实验研究，可以绘制 Precision-Recall 曲线来帮助分析。

3、推荐系统有哪些常用的评价标准

解析：

常用的评价标准：

一类是线上的评测，比如通过点击率、网站流量、A/B test 等判断。这类评价标准在这里就不细说了，因为它们并不能参与到线下训练模型和选择模型的过程当中。

第二类是线下评测。评测标准很多，我挑几个常用的。我就拿给用户推荐阅读相关链接来举例

好了。
1. 精度 Precision：P(k)
P(k) = c/k
我们给某个用户推荐了 k 个链接，他 / 她点击了其中的 c 个链接，那么精度就是 c/k。
2. 平均精度 Average Precision: ap@n

$$ap@n = \sum_{k=1}^{n} \frac{P(k)}{\min(m,n)}$$

n 是被预测的链接的总数，m 是用户点击的链接的总数。
例子 1：　我们一共推荐了 10 个链接，用户实际上点击了我们推荐当中的第 1 个和第 4 个链接，以及另外两个其他的链接，那么对于这个用户，
ap@10=(1/1+2/4)/4≈0.38
例子 2：　我们一共推荐了 10 个链接，用户实际上点击了我们推荐当中的第 2 个,第 3 个和第 5 个链接，以及另外三个其他的链接，那么对于这个用户，
ap@10=(1/2+2/3+3/5)/6≈0.29
例子 3：　我们一共推荐了 10 个链接，用户实际上点击了我们推荐当中的第 2 个,第 7 个，此外没有点击其他联系，那么对于这个用户，
ap@10=(1/2+2/7)/2≈0.39
例子 4：　我们一共推荐了 5 个链接，用户实际上点击了我们推荐当中的第 1 个,第 2 个和第 4 个，以及另外 6 个其他链接，那么对于这个用户，
ap@5=(1/1+2/2+3/4)/5≈0.55
3. 平均精度均值 Mean Average Precision: MAP@n
MAP 计算的是 N 个用户的平均精度的均值。

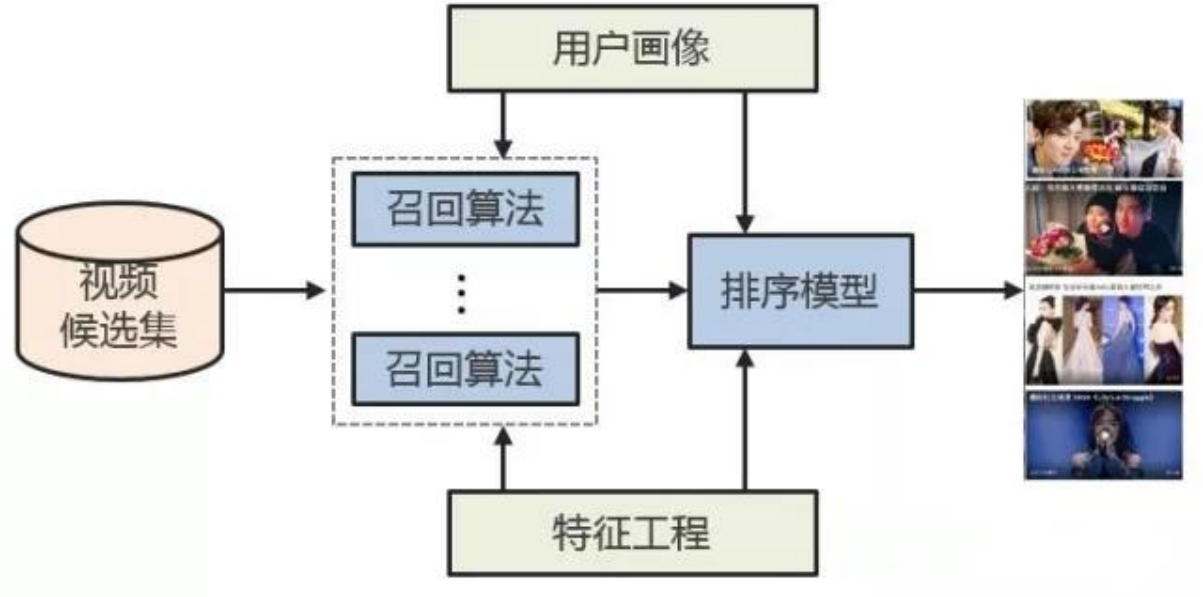
$$MAP@n = \sum_{i=1}^{N} \frac{(ap@n)_i}{N}$$

这个 N 是用户数量。
比如说我们三个用户甲、乙、丙分别推荐了 10 个链接，
甲点击了我们推荐当中的第 1 个和第 4 个链接，以及另外两个其他的链接，那么
(ap@10)1=(1/1+2/4)/4≈0.38.
乙点击了我们推荐当中的第 3 个链接，以及另外一个其他的链接，那么
(ap@10)2=(1/3)/2≈0.17.
丙点击了我们推荐当中的第 1 个链接，第 7 个链接，以及另外三个其他的链接，那么
(ap@10)3=(1/1+2/7)/5≈0.26.
那么这个模型的平均精度均值
MAP@10 = (0.38+0.17+0.26)/3 ≈ 0.27
更多请参考：http://sofasofa.io/forum_main_post.php?postid=1000292

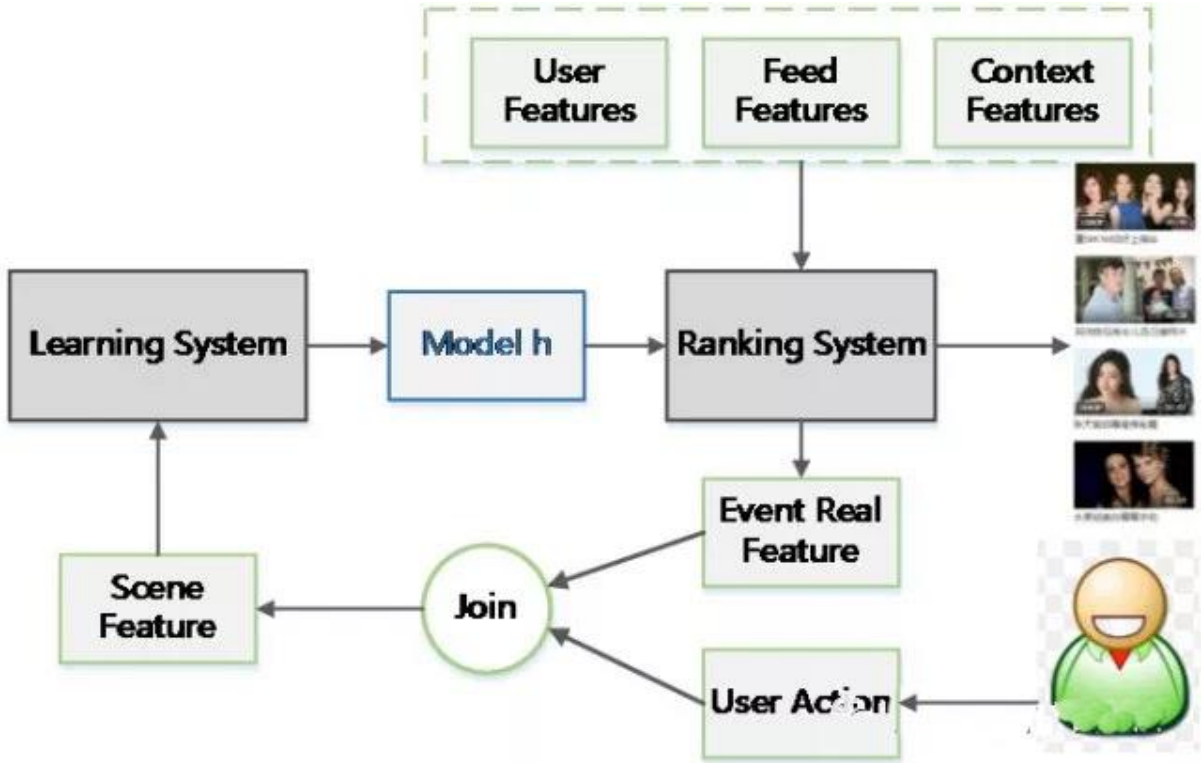
4、请详细聊聊推荐系统里的排序算法

解析:
一、个性化推荐
在海量的内容在满足了我们需求的同时，也使我们寻找所需内容更加困难，在这种情况下个性化推荐应运而生。
在当前这个移动互联网时代，除了专业内容的丰富，UGC 内容更是爆发式发展，每个用户既是内容的消费者，也成为了内容的创造者。这些海量的内容在满足了我们需求的同时，也使我们寻找所需内容更加困难，在这种情况下个性化推荐应运而生。
个性化推荐是在大数据分析和人工智能技术的基础上，通过研究用户的兴趣偏好，进行个性化计算，从而给用户提供高质量的个性化内容，解决信息过载的问题，更好的满足用户的需求。
二、爱奇艺推荐系统介绍

我们的推荐系统主要分为两个阶段，召回阶段和排序阶段。
召回阶段根据用户的兴趣和历史行为，同千万级的视频库中挑选出一个小的候选集（几百到几千个视频）。这些候选都是用户感兴趣的内容，排序阶段在此基础上进行更精准的计算，能够给每一个视频进行精确打分，进而从成千上万的候选中选出用户最感兴趣的少量高质量内容（十几个视频）。



推荐系统的整体结构如图所示，各个模块的作用如下：
i)用户画像：包含用户的人群属性、历史行为、兴趣内容和偏好倾向等多维度的分析，是个性化的基石
ii)特征工程：包含了了视频的类别属性，内容分析，人群偏好和统计特征等全方位的描绘和度量，是视频内容和质量分析的基础
iii)召回算法：包含了多个通道的召回模型，比如协同过滤，主题模型，内容召回和 SNS 等通道，能够从视频库中选出多样性的偏好内容
iv)排序模型：对多个召回通道的内容进行同一个打分排序，选出最优的少量结果。
除了这些之外推荐系统还兼顾了推荐结果的多样性，新鲜度，逼格和惊喜度等多个维度，更能够满足用户多样性的需求。
三、推荐排序系统架构
在召回阶段，多个通道的召回的内容是不具有可比性的，并且因为数据量太大也难以进行更加精确的偏好和质量评估，因此需要在排序阶段对召回结果进行统一的准确的打分排序。
用户对视频的满意度是有很多维度因子来决定的，这些因子在用户满意度中的重要性也各不相同，甚至各个因子之间还有多层依赖关系，人为制定复杂的规则既难以达到好的效果，又不具有可维护性，这就需要借助机器学习的方法，使用机器学习模型来综合多方面的因子进行排序。



排序系统的架构如图所示，主要由用户行为收集，特征填充，训练样本筛选，模型训练，在线预测排序等多个模块组成。

机器学习的主体流程是比较通用的，设计架构并不需要复杂的理论，更多的是需要对细节，数据流和架构逻辑的仔细推敲。

这个架构设计吸取了以前的经验和教训，在通用机器学习的架构基础上解决了两个问题：

A 训练预测的一致性

机器学习模型在训练和预测之间的差异会对模型的准确性产生很大的影响，尤其是模型训练与在线服务时特征不一致，比如用户对推荐结果的反馈会实时影响到用户的偏好特征，在训练的时候用户特征的状态已经发生了变化，模型如果依据这个时候的用户特征就会产生非常大的误差。

我们的解决办法是，将在线服务时的特征保存下来，然后填充到收集的用户行为样本中，这样就保证了训练和预测特征的一致性。

B 持续迭代

互联网产品持续迭代上线是常态，在架构设计的时候，数据准备，模型训练和在线服务都必须能够对持续迭代有良好的支持。

我们的解决方案是，数据准备和模型训练各阶段解耦，并且策略配置化，这种架构使模型测试变得非常简单，可以快速并行多个迭代测试。

四、推荐机器学习排序算法演进

4.1 上古时期

我们第一次上线机器学习排序模型时，选用了比较简单的 Logistic Regression，将重点放到架构设计上，尽量保证架构的正确性。除此之外，LR 模型的解释性强，方便 debug，并且通过特征权重可以解释推荐的内容，找到模型的不足之处。

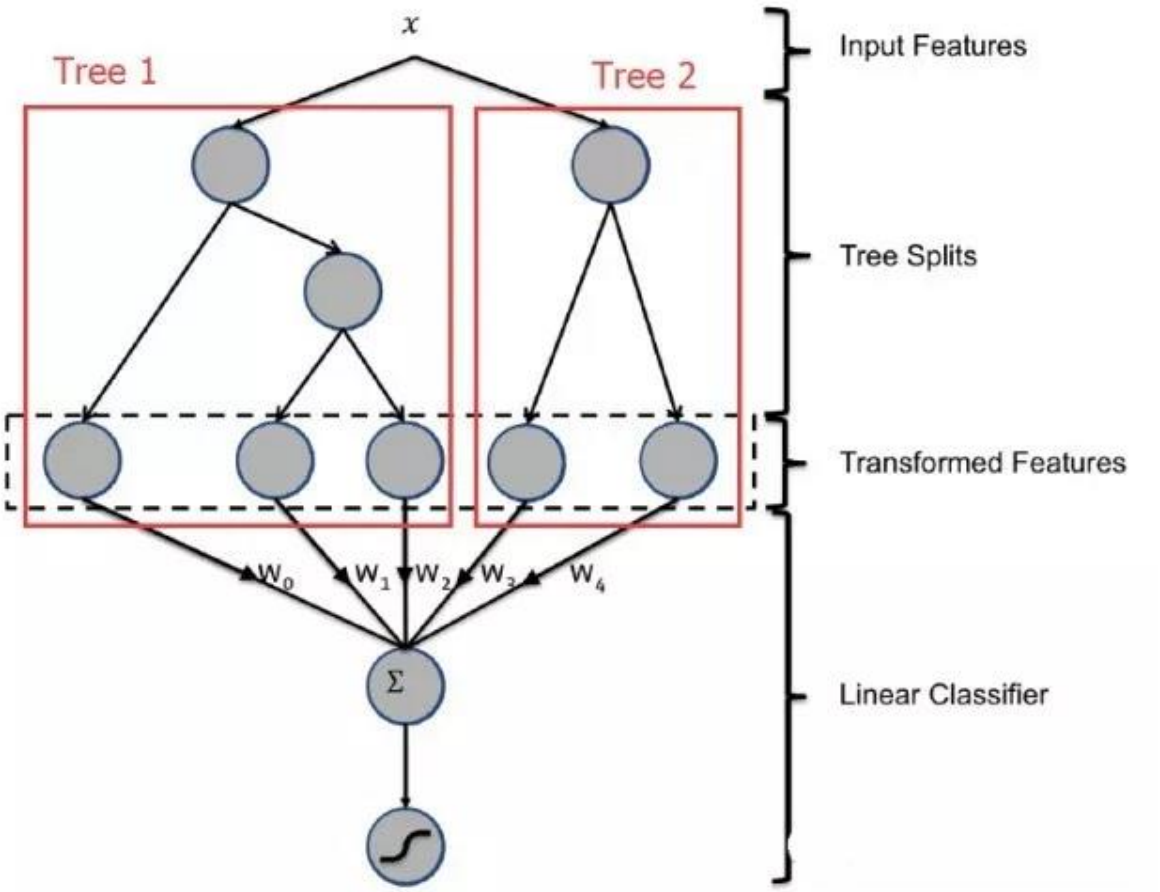
在模型训练之前，我们首先解决的是评测指标和优化目标的问题。

评测指标（metrics）

线上效果的评测指标需要与长远目标相匹配，比如使用用户的投入程度和活跃度等。在我们的实验中，业界流行的 CTR 并不是一个好的评测指标，它会更偏向于较短的视频，标题党和低俗内容。

离线评测指标是按照业务来定制的，以便与在线评测指标匹配，这样在离线阶段就能够淘汰掉

无效策略，避免浪费线上流量。
优化目标（objective）
机器学习会按照优化目标求解最优解，如果优化目标有偏差，得到的模型也存在偏差，并且在迭代中模型会不断地向这个偏差的方向学习，偏差会更加严重。
我们的方法是给样本添加权重，并且将样本权重加到 loss function 中，使得优化目标与评测指标尽可能的一致，达到控制模型的目的。
LR 是个线性分类模型，要求输入是线性独立特征。我们使用的稠密的特征（维度在几十到几百之间）往往都是非线性的，并且具有依赖性，因此需要对特征进行转换。
特征转换需要对特征的分布，特征与 label 的关系进行分析，然后采用合适的转换方法。我们用到的有以下几种：Polynomial Transformation，Logarithmic or Exponential Transformation，Interaction Transformation 和 Cumulative Distribution Function 等。
虽然 LR 模型简单，解释性强，不过在特征逐渐增多的情况下，劣势也是显而易见的。
1. 特征都需要人工进行转换为线性特征，十分消耗人力，并且质量不能保证
2. 特征两两作 Interaction 的情况下，模型预测复杂度是。在 100 维稠密特征的情况下，就会有组合出 10000 维的特征。复杂度高，增加特征困难
3. 三个以上的特征进行 Interaction 几乎是不可行的
4.2 中古时期
为了解决 LR 存在的上述问题，我们把模型升级为 Facebook 的 GBDT+LR 模型，模型结构如图所示。

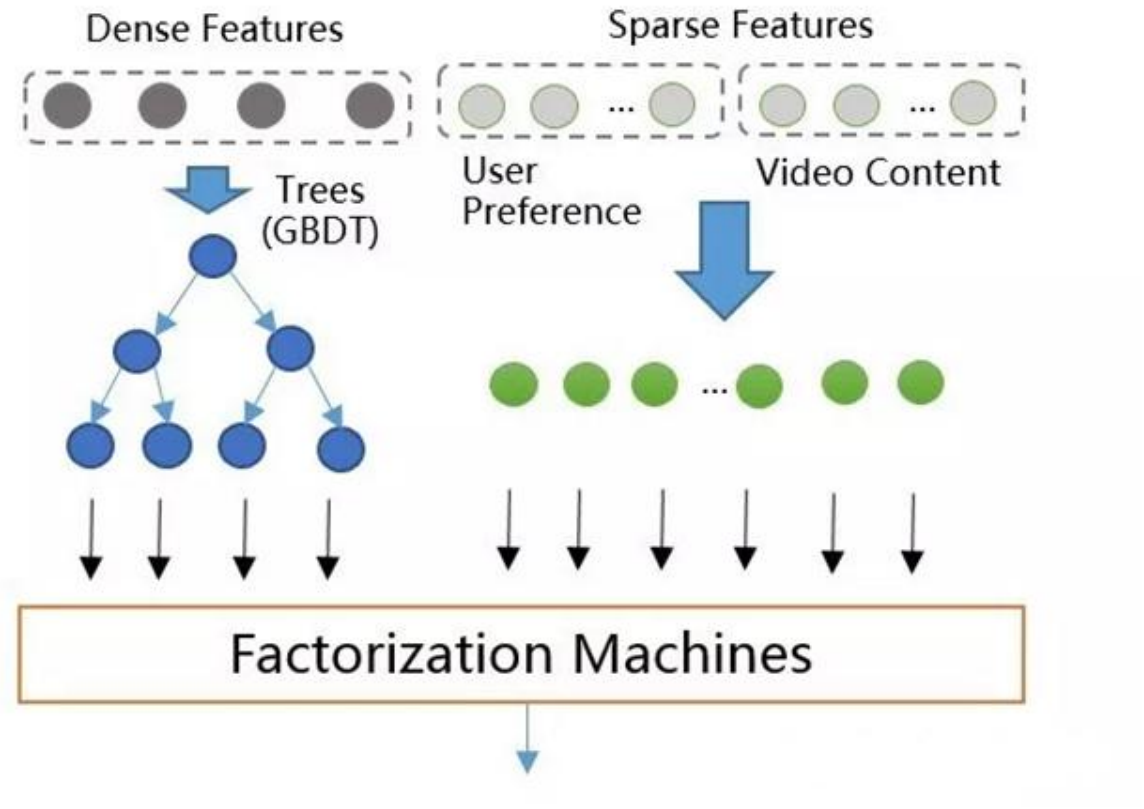


GBDT 是基于 Boosting 思想的 ensemble 模型，由多颗决策树组成，具有以下优点：
1)对输入特征的分布没有要求
2)根据熵增益自动进行特征转换、特征组合、特征选择和离散化，得到高维的组合特征，省去了人工转换的过程，并且支持了多个特征的 Interaction

3)预测复杂度与特征个数无关
假设特征个数 n=160 决策数个数 k=50，树的深度 d=6，两代模型的预测复杂度对比如下，升级之后模型复杂度降低到原来的 2.72%

Model Structure	Complexity	Complexity
LR	n(n-1) / 2 + n	12880
GBDT+LR	k * d + k	350
rate		2.72%

GBDT 与 LR 的 stacking 模型相对于只用 GBDT 会有略微的提升，更大的好处是防止 GBDT 过拟合。升级为 GBDT+LR 后，线上效果提升了约 5%，并且因为省去了对新特征进行人工转换的步骤，增加特征的迭代测试也更容易了。
4.3 近代历史
GBDT+LR 排序模型中输入特征维度为几百维，都是稠密的通用特征。
这种特征的泛化能力良好，但是记忆能力比较差，所以需要增加高维的（百万维以上）内容特征来增强推荐的记忆能力，包括视频 ID，标签，主题等特征。
GBDT 是不支持高维稀疏特征的，如果将高维特征加到 LR 中，一方面需要人工组合高维特征，另一方面模型维度和计算复杂度会是 O（N^2）级别的增长。所以设计了 GBDT+FM 的模型如图所示，采用 Factorization Machines 模型替换 LR。



Factorization Machines（FM）模型如下所示：
模型公式

$$\hat{y}(\mathbf{x}) := w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j$$

具有以下几个优点
①前两项为一个线性模型，相当于 LR 模型的作用
②第三项为一个二次交叉项，能够自动对特征进行交叉组合
③通过增加隐向量，模型训练和预测的计算复杂度降为了 O(N)
④支持稀疏特征
几个优点，使的 GBDT+FM 具有了良好的稀疏特征支持，FM 使用 GBDT 的叶子结点和稀疏特征（内容特征）作为输入，模型结构示意图如下，GBDT+FM 模型上线后相比 GBDT+LR 在各项指标的效果提升在 4%~6%之间。
典型的 FM 模型中使用 user id 作为用户特征，这会导致模型维度迅速增大，并且只能覆盖部分热门用户，泛化能力比较差。在此我们使用用户的观看历史以及兴趣标签代替 user id，降低了特征维度，并且因为用户兴趣是可以复用的，同时也提高了对应特征的泛化能力。
我们主要尝试使用了 L-BFGS、SGD 和 FTRL（Follow-the-regularized-Leader）三种优化算法进行求解：
SGD 和 L-BFGS 效果相差不大，L-BFGS 的效果与参数初始化关系紧密
FTRL，较 SGD 有以下优势：
a)带有 L1 正则，学习的特征更加稀疏
b)使用累计的梯度，加速收敛
c)根据特征在样本的出现频率确定该特征学习率，保证每个特征有充分的学习
FM 模型中的特征出现的频次相差很大，FTRL 能够保证每个特征都能得到充分的学习，更适合

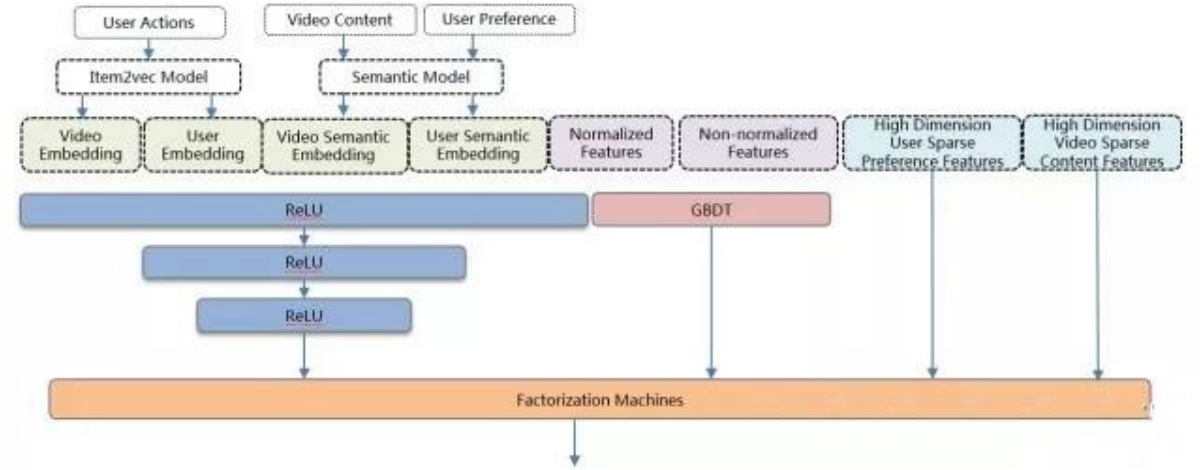
稀疏特征。线上测试表明，在稀疏特征下 FTRL 比 SGD 有 4.5%的效果提升。

4.4 当代模型

GBDT+FM 模型，对 embedding 等具有结构信息的深度特征利用不充分，而深度学习（Deep Neural Network）能够对嵌入式（embedding）特征和普通稠密特征进行学习，抽取出深层信息，提高模型的准确性，并已经成功应用到众多机器学习领域。因此我们将 DNN 引入到排序模型中，提高排序整体质量。

DNN+GBDT+FM 的 ensemble 模型架构如图所示，FM 层作为模型的最后一层，即融合层，其输入由三部分组成：DNN 的最后一层隐藏层、GBDT 的输出叶子节点、高维稀疏特征。

DNN+GBDT+FM 的 ensemble 模型架构介绍如下所示，该模型上线后相对于 GBDT+FM 有 4%的效果提升。



DNN 模型

使用全连接网络，共三个隐藏层。

隐藏节点数目分别为 1024，512 和 256。

预训练好的用户和视频的 Embedding 向量，包含基于用户行为以及基于语义内容的两种 Embedding。

DNN 能从具有良好数学分布的特征中抽取深层信息，比如 embedding 特征，归一化后统计特征等等。

虽然 DNN 并不要求特征必须归一化，不过测试发现有些特征因为 outlier 的波动范围过大，会导致 DNN 效果下降。

GBDT 模型

单独进行训练，输入包含归一化和未归一化的稠密特征。

能处理未归一化的连续和离散特征。

能根据熵增益自动对输入特征进行离散和组合。

FM 融合层

FM 模型与 DNN 模型作为同一个网络同时训练。

将 DNN 特征，GBDT 输出和稀疏特征进行融合并交叉。

使用分布式的 TensorFlow 进行训练

使用基于 TensorFlow Serving 的微服务进行在线预测

DNN+GBDT+FM 的 ensemble 模型使用的是 Adam 优化器。Adam 结合了 The Adaptive Gradient Algorithm（AdaGrad）和 Root Mean Square Propagation（RMSProp）算法。具有更优的收敛速率，每个变量有独自的下降步长，整体下降步长会根据当前梯度进行调节，能够适应带噪音的数据。实验测试了多种优化器，Adam 的效果是最优的。

4.5 工业界 DNN ranking 现状

1、Youtube 于 2016 年推出 DNN 排序算法。

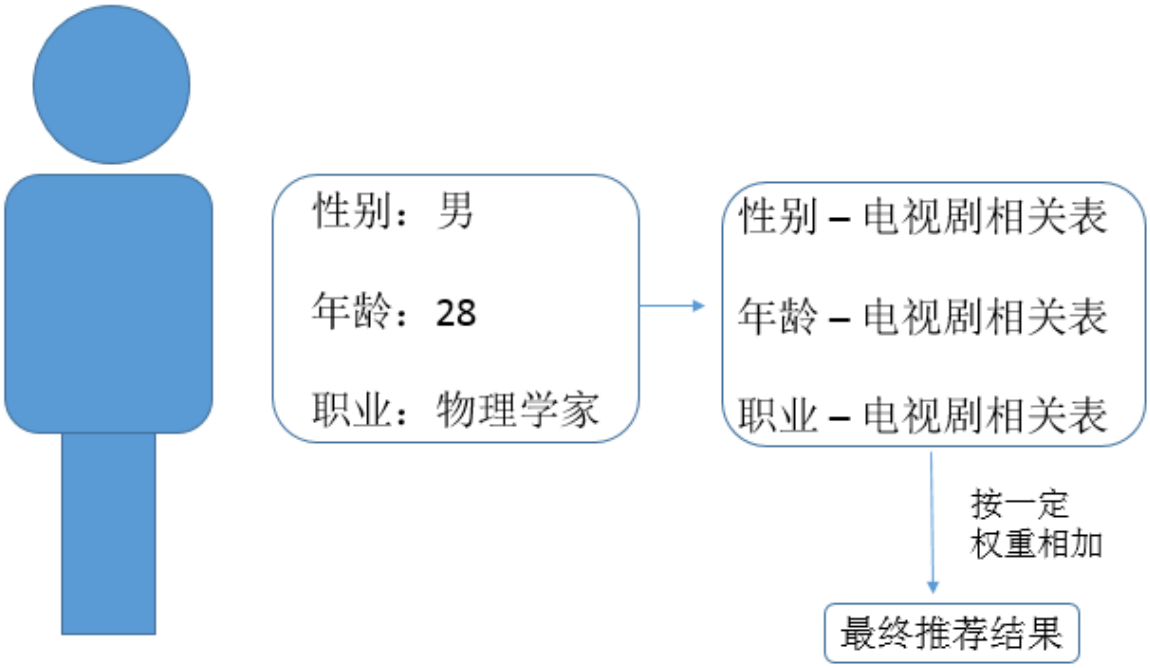
2、上海交通大学和 UCL 于 2016 年推出 Product-based Neural Network（PNN）网络进行用户

点击预测。PNN 相当于在 DNN 层做了特征交叉，我们的做法是把特征交叉交给 FM 去做，DNN 专注于深层信息的提取。
3、Google 于 2016 年推出 Wide And Deep Model，这个也是我们当前模型的基础，在此基础上使用 FM 替换了 Cross Feature LR，简化了计算复杂度，提高交叉的泛化能力。
阿里今年使用 attention 机制推出了 Deep Interest Network（DIN）进行商品点击率预估，优化 embedding 向量的准确性，值得借鉴。
五、总结
推荐系统的排序是一个经典的机器学习场景，对于推荐结果影响也十分重大，除了对模型算法的精益求精之外，更需要对业务的特征，工程的架构，数据处理的细节和 pipeline 的流程进行仔细推敲和深入的优化。
Ranking 引入 DNN 仅仅是个开始，后续还需要在模型架构，Embedding 特征，多样性，冷启动和多目标学习中做更多的尝试，提供更准确，更人性化的推荐，优化用户体验。

5、怎么解决推荐系统中的冷启动问题?

解析:
1.冷启动问题定义
推荐系统需要根据用户的历史行为和兴趣预测用户未来的行为和兴趣，对于 BAT 这类大公司来说，它们已经积累了大量的用户数据，不发愁。但是对于很多做纯粹推荐系统的网站或者很多在开始阶段就希望有个性化推荐应用的网站来说，如何在对用户一无所知（即没有用户行为数据）的情况下进行最有效的推荐呢？这就衍生了冷启动问题。
2.冷启动的分类
冷启动问题主要分为 3 类:
用户冷启动，即如何给新用户做个性化推荐
物品冷启动，即如何将新的物品推荐给可能对它感兴趣的用户
系统冷启动，即如何在一个新开发的网站（没有用户，没有用户行为，只有部分物品信息）上设计个性化推荐系统，从而在网站刚发布时就让用户体会到个性化推荐
3.冷启动问题的解决方案
3.1 提供非个性化的推荐
最简单的例子就是提供热门排行榜，可以给用户推荐热门排行榜，等到用户数据收集到一定的时候，再切换为个性化推荐。
关于热门排行榜解决推荐问题的理论测试，可以参考着篇文章 Performance of recommender algorithms on top-n recommendation tasks.
并且 Netflix 的研究也表明新用户在冷启动阶段确实是更倾向于热门排行榜的，老用户会更加需要长尾推荐。
3.2 利用用户注册信息
用户的注册信息主要分为 3 种:
1）人口统计学信息，包括年龄、性别、职业、民族、学历和居住地
2）用户兴趣的描述，部分网站会让用户用文字来描述兴趣
3）从其他网站导入的用户站外行为，比如用户利用社交网站账号登录，就可以在获得用户授权的情况下导入用户在该社交网站的部分行为数据和社交网络数据
这种个性化的粒度很粗，假设性别作为一个粒度来推荐，那么所有刚注册的女性看到的都是同样的结果，但是相对于男女不区分的方式，这种推荐精度已经大大提高了。
推荐流程基本如下:
1. 获取用户的注册信息
2. 根据用户的注册信息对用户分类
3. 给用户推荐他所属分类中用户喜欢的物品

下面便是一个利用用户的注册信息进行推荐的例子：



3.3 选择合适的物品启动用户的兴趣

用户在登录时对一些物品进行反馈，收集用户对这些物品的兴趣信息，然后给用户推荐那些和这些物品相似的物品。

一般来说，能够用来启动用户兴趣的物品需要具有以下特点：

1. 比较热门，如果要让用户对物品进行反馈，前提是用户得知道这是什么东西；
2. 具有代表性和区分性，启动用户兴趣的物品不能是大众化或老少咸宜的，因为这样的物品对用户的兴趣没有区分性；
3. 启动物品集合需要有多样性，在冷启动时，我们不知道用户的兴趣，而用户兴趣的可能性非常多，为了匹配多样的兴趣，我们需要提供具有很高覆盖率的启动物品集合，这些物品能覆盖几乎所有主流的用户兴趣。

3.4 利用物品的内容信息

用来解决物品的冷启动问题，即如何将新加入的物品推荐给对它感兴趣的用户。物品冷启动问题在新闻网站等时效性很强的网站中非常重要，因为这些网站时时刻刻都有新物品加入，而且每个物品必须能够再第一时间展现给用户，否则经过一段时间后，物品的价值就大大降低了。针对协同过滤的两种推荐算法——userCF 算法、itemCF 算法来分别了解一下物品冷启动的问题。

userCF 算法

针对推荐列表并不是给用户展示内容的唯一列表（大多网站都是这样的）的网站

当新物品加入时，总会有用户通过某些途径看到，那么当一个用户对其产生反馈后，和他历史兴趣相似的用户的推荐列表中就有可能出现该物品，从而更多的人对该物品做出反馈，导致更多的人的推荐列表中出现该物品。因此，该物品就能不断扩散开来，从而逐步展示到对它感兴趣用户的推荐列表中

针对推荐列表是用户获取信息的主要途径（例如豆瓣网络电台）的网站

userCF 算法就需要解决第一推动力的问题，即第一个用户从哪儿发现新物品。最简单的方法是将新的物品随机战士给用户，但是太不个性化。因此可以考虑利用物品的内容信息，将新物品先投放给曾经喜欢过和它内容相似的其他物品的用户

itemCF 算法

对 itemCF 算法来说，物品冷启动就是很严重的问题了。因为该算法的基础是通过用户对物品

产生的行为来计算物品之间的相似度，当新物品还未展示给用户时，用户就无法产生行为。为此，只能利用物品的内容信息计算物品的相关程度。基本思路就是将物品转换成关键词向量，通过计算向量之间的相似度（例如计算余弦相似度），得到物品的相关程度。
下表列出了常见物品的内容信息：

表3-4　常见物品的内容信息

图书	标题、作者、出版社、出版年代、丛书名、目录、正文
论文	标题、作者、作者单位、关键字、分类、摘要、正文
电影	标题、导演、演员、编剧、类别、剧情简介、发行公司
新闻	标题、正文、来源、作者
微博	作者、内容、评论

3.5 采用专家标注
很多系统在建立的时候，既没有用户的行为数据，也没有充足的物品内容信息来计算物品相似度。这种情况下，很多系统都利用专家进行标注。
代表系统：个性化网络电台 Pandora、电影推荐网站 Jinni
以 Pandora 电台为例，Pandora 雇用了一批音乐人对几万名歌手的歌曲进行各个维度的标注，最终选定了 400 多个特征。每首歌都可以标识为一个 400 维的向量，然后通过常见的向量相似度算法计算出歌曲的相似度。
以上均为项亮一书《推荐系统实践》中描述到的方法，下面再介绍两种方法。
3.6 利用用户在其他地方已经沉淀的数据进行冷启动
以 QQ 音乐举例：
QQ 音乐的猜你喜欢电台想要去猜测第一次使用 QQ 音乐的用户的口味偏好，一大优势是可以利用其它腾讯平台的数据，比如在 QQ 空间关注了谁，在腾讯微博关注了谁，更进一步，比如在腾讯视频刚刚看了一部动漫，那么如果 QQ 音乐推荐了这部动漫里的歌曲，用户会觉得很人性化。这就是利用用户在其它平台已有的数据。
再比如今日头条：
它是在用户通过新浪微博等社交网站登录之后，获取用户的关注列表，并且爬取用户最近参与互动的 feed（转发/评论等），对其进行语义分析，从而获取用户的偏好。
所以这种方法的前提是，引导用户通过社交网络账号登录，这样一方面可以降低注册成本提高转化率；另一方面可以获取用户的社交网络信息，解决冷启动问题。
3.7 利用用户的手机等兴趣偏好进行冷启动
Android 手机开放的比较高，所以在安装自己的 app 时，就可以顺路了解下手机上还安装了什么其他的 app。比如一个用户安装了美丽说、蘑菇街、辣妈帮、大姨妈等应用，就可以判定这是女性了，更进一步还可以判定是备孕还是少女。
目前读取用户安装的应用这部分功能除了 app 应用商店之外，一些新闻类、视频类的应用也在做，对于解决冷启动问题有很好的帮助。

6、请聊聊你所了解的推荐系统算法

解析：
推荐系统算法如果根据推荐的依据进行划分，有如下三大类算法：
一、Content-based recommenders：推荐和用户曾经喜欢的商品相似的商品。 主要是基于商品属性信息和用户画像信息的对比。核心问题是如何刻画商品属性和用户画像以及效用的度量。方法包括：
1.1 Heuristic-based method：对于特征维度的构建，例如基于关键字提取的方法，使用 TF-IDF 等指标提取关键字作为特征。对于效用的度量，例如使用启发式 cosine 相似性指标，衡量商品特征和用户画像的相似性，似性越高，效用越大。
1.2 Machine learning-based mehod：对于特征维度的构建，使用机器学习算法来构建用户和

商品的特征维度。例如建模商品属于某个类别的概率，得到商品的刻画属性。对于效用的度量，直接使用机器学习算法拟合效用函数。
二、Collaborative recommenders: 推荐和用户有相似品味和偏好的用户喜欢过的商品。主要是基于用户和商品历史交互行为信息，包括显示的和隐式的。协同过滤方法进一步细分为:
2.1 Memory-based CF: 基于内存的协同过滤方法。直接对 User-Item 矩阵进行研究。通过启发式的方法来进行推荐。核心要素包括相似性度量和推荐策略。相似性度量包括 Pearson 或 Cosine 等；而最简单的推荐方法是基于大多数的推荐策略。
User-based CF: 推荐给特定用户列表中还没有发生过行为、而在相似用户列表中产生过行为的高频商品。
Item-based CF: 推荐给特定用户列表中还没有发生过行为、并且和已经发生过行为的商品相似的商品。
2.2 Model-based CF: 基于模型的协同过滤方法。主要是运用机器学习的思想来进行推荐。主要包括:
基于流形学习的矩阵降维/分解算法: SVD、FunkSVD、BiasSVD、SVD++、NMF 等。
基于表示学习的深度学习算法：MLP、CNN、AutoEncoder、RNN 等。
基于图/网络模型的算法：MDP-based CF、Bayesian Belief nets CF、CTR(协同主题回归，将概率矩阵分解和主题模型结合应用于推荐系统)等。
其它：包括基于聚类的 CF、稀疏因子分析 CF、隐语义分析 CF 等等。
2.3 Hybrid CF: 结合多种方式的 CF 算法。如 Content-based CF、Content-boosted CF 或者结合 Memory-based 和 Model-based CF 混合方法。

Table 2: Overview of collaborative filtering techniques.

CF categories	Representative techniques	Main advantages	Main shortcomings
Memory-based CF	*Neighbor-based CF (item-based/user-based CF algorithms with Pearson/vector cosine correlation) *Item-based/user-based top-*N* recommendations	*easy implementation *new data can be added easily and incrementally *need not consider the content of the items being recommended *scale well with co-rated items	*are dependent on human ratings *performance decrease when data are sparse *cannot recommend for new users and items *have limited scalability for large datasets
Model-based CF	*Bayesian belief nets CF *clustering CF *MDP-based CF *latent semantic CF *sparse factor analysis *CF using dimensionality reduction techniques, for example, *SVD*, *PCA*	*better address the sparsity, scalability and other problems *improve prediction performance *give an intuitive rationale for recommendations	*expensive model-building *have trade-off between prediction performance and scalability *lose useful information for dimensionality reduction techniques
Hybrid recommenders	*content-based CF recommender, for example, *Fab* *content-boosted CF *hybrid CF combining memory-based and model-based *CF* algorithms, for example, Personality Diagnosis	*overcome limitations of CF and content-based or other recommenders *improve prediction performance *overcome CF problems such as sparsity and gray sheep	*have increased complexity and expense for implementation *need external information that usually not available

三、Hybrid approaches: 混合方法。综合集成上述两种方法。
当前推荐算法主要是基于内容(CB)、协同过滤(CF)、混合算法。基于内容的推荐依靠用户 profile 和 item 的描述做推荐。CF 基于过去的的表现和行为推荐。由于种种原因，收集过去的行为比收集用户画像要容易，但 CF 又有他的局限性，当打分（rating）很稀疏时，预测精度会下降很厉害，同时，新产品的冷启动也是 CF 的问题。因此，近年来，混合方法应用比较广。

Recommendation Approach	Recommendation Technique	
	Heuristic-based	Model-based
Content-based	Commonly used techniques: • TF-IDF (information retrieval) • Clustering Representative research examples: • Lang 1995 • Balabanovic & Shoham 1997 • Pazzani & Billsus 1997	Commonly used techniques: • Bayesian classifiers • Clustering • Decision trees • Artificial neural networks Representative research examples: • Pazzani & Billsus 1997 • Mooney et al. 1998 • Mooney & Roy 1999 • Billsus & Pazzani 1999, 2000 • Zhang et al. 2002
Collaborative	Commonly used techniques: • Nearest neighbor (cosine, correlation) • Clustering • Graph theory Representative research examples: • Resnick et al. 1994 • Hill et al. 1995 • Shardanand & Maes 1995 • Breese et al. 1998 • Nakamura & Abe 1998 • Aggarwal et al. 1999 • Delgado & Ishii 1999 • Pennock & Horwitz 1999 • Sarwar et al. 2001	Commonly used techniques: • Bayesian networks • Clustering • Artificial neural networks • Linear regression • Probablistic models Representative research examples: • Billsus & Pazzani 1998 • Breese et al. 1998 • Ungar & Foster 1998 • Chien & George 1999 • Getoor & Sahami 1999 • Pennock & Horwitz 1999 • Goldberg et al. 2001 • Kumar et al. 2001 • Pavlov & Pennock 2002 • Shani et al. 2002 • Yu et al. 2002, 2004 • Hofmann 2003, 2004 • Marlin 2003 • Si & Jin 2003
Hybrid	Combining content-based and collaborative components using: • Linear combination of predicted ratings • Various voting schemes • Incorporating one component as a part of the heuristic for the other Representative research examples: • Balabanovic & Shoham 1997 • Claypool et al. 1999 • Good et al. 1999 • Pazzani 1999 • Billsus & Pazzani 2000 • Tran & Cohen 2000 • Melville et al. 2002	Combining content-based and collaborative components by: • Incorporating one component as a part of the model for the other • Building one unifying model Representative research examples: • Basu et al. 1998 • Condliff et al. 1999 • Soboroff & Nicholas 1999 • Ansari et al. 2000 • Popescul et al. 2001 • Schein et al. 2002

7、了解隐语义模型在推荐系统中的应用的么？

解析:
本篇记录在学习隐语义模型的一些总结，隐语义建模；隐语义模型的核心思想；隐语义模型在推荐系统中的应用；隐语义模型与推荐系统的关系；工程中常用的矩阵分解方法
前言
推荐系统中一个重要的分支，隐语义建模。隐语义模型 LFM：Latent Factor Model，其核心思想就是通过隐含特征联系用户兴趣和物品。
过程分为三个部分，将物品映射到隐含分类，确定用户对隐含分类的兴趣，然后选择用户感兴趣的分类中的物品推荐给用户。它是基于用户行为统计的自动聚类。
隐语义模型在 Top-N 推荐中的应用十分广泛。常用的隐语义模型，LSA(Latent Semantic Analysis)，LDA(Latent Dirichlet Allocation)，主题模型(Topic Model)，矩阵分解(Matrix

Factorization)等等。
这些模型是本质上是相通的，目的就是找出潜在的主题或分类。这些技术一开始实在文本挖掘中提出来的。从 netflix 推荐大赛之后，这些技术被用在推荐领域，并且得到明显的效果。比如，在推荐系统中，能够用这些模型将用户的行为（用户的偏好）对 item 进行自动聚类，也就是把 item 划分到不同类别/主题，这些主题/类别可以理解为用户的兴趣。
凭借这种思想，著名的推荐领域大神 Yehuda Koren 更是凭借矩阵分解模型勇夺 Netflix Prize 推荐比赛冠军，以矩阵分解为基础，Yehuda Koren 在数据挖掘和机器学习相关的国际顶级会议(SIGIR,SIGKDD,RecSys 等)发表了很多文章，将矩阵分解模型的优势发挥得淋漓尽致。实验结果表明，在个性化推荐中使用矩阵分解模型要明显优于传统的基于邻域的协同过滤(又称基于记忆的协同过滤)方法，如 UserCF、ItemCF 等，这也使得矩阵分解成为了之前个性化推荐研究领域中的主流模型。
基本算法
与 UserCF 和 ItemCF 对比
首先通过一个例子来理解一下这个模型，比如说有两个用户 A 和 B，目前有用户的阅读列表，用户 A 的兴趣涉及侦探小说，科普图书以及一些计算机技术书，而用户 B 的兴趣比较集中在数学和机器学习方面。
那么如何给 A 和 B 推荐图书呢？
对于 UserCF，首先需要找到和他们看了同样书的其他用户(兴趣相似的用户)，然后在给他们推荐那些用户喜欢的其他书。
对于 ItemCF,需要给他们推荐和他们已经看的书相似的书，比如用户 B 看了很多数据挖掘方面的书，那么可以给他推荐机器学习或者模式识别方面的书。
还有一种方法，可以对书和物品的兴趣进行分类。对于某个用户，首先得到他的兴趣分类，然后从分类中挑选他可能喜欢的物品。
基于兴趣分类的方法大概需要解决的问题：
如何给物品进行分类？
如何确定用户对哪些类的物品感兴趣，以及感兴趣的程度？
对于一个给定的类，选择哪些属于这个类的物品推荐给用户，以及如何确定这些物品在一个类中的权重？
对于第一个问题最简单的解决方案是找编辑给物品打标签，或者采用豆瓣的方式，让用户自己打标签。这样做的确定是，这种打标签不自动发现类比，还有一点就是编辑的意见并不能达标各种用户的意见，比如一些书目的划分是模棱两可的。这就涉及到分类的粒度不易控制的问题。另外，编辑很难决定一个物品在某一个分类中的权重，比如编辑可以很容易的决定《数据挖掘导论》属于挖掘类的图书，但是这本书在这类书中的定位是什么样的，编辑就很难给出一个准确的数字来标示。
为了解决上面的问题，研究人员提出：为什么我们不从数据出发，自动地找到那些类，然后进行个性化推荐，隐语义分析技术因为采取基于用户行为统计的自动聚类，较好地解决了上面的问题。隐语义分析技术从诞生到今天产生了很多著名的模型和方法，其中和推荐技术相关的有 pLSA，LDA，隐含类别模型（latent class model），隐含主题模型（latent topic model），矩阵分解（matrix factorization）。
隐语义模型的应用
向用户推荐物品
在推荐系统中，可以通过隐含语义模型将用户（user）和物品（item）自动分类，这些类别是自动生成的，这些类比也可以叫做“隐含的分类”，或许我们看不懂分类后的结果。但是采用这种技术后，每个用户或者物品会被分到多个类别中，属于某个类别的权重会被计算出来。
假设现在有一个大小为 m×n 的评分矩阵 V，包含了 m 个用户对 n 个物品的评分，评分从 0 到 5，值越大代表越喜欢，0 代表没有打分。设定共有 r 个隐含的分类。通过一些方法，将 V 展开为两个相乘的矩阵：

V = W*H
其中，W 的大小为 m×r，H 的大小为 r×n。在隐语义模型中，W(i,j)被解释为用户 i 属于类别 j 的权重，H(a,b)被解释为物品 b 属于类别 a 的的权重。
如果用户 u 对物品 i 没有评分，可以将这个评分 r(u,i)预测为：
r(u,i) = sum(W(i, :) .* H(:, i)) // 向量点乘
据此可以构建一个推荐系统。
文本分类
隐语义模型的另一个常见的应用领域是文本分类，这个类似于上面的推荐系统。在文本分类领域，将文档和词用一个矩阵表示，如常见的词袋模型，文档-词矩阵。我们将数据集中的一堆文本构造成文档-词矩阵 V，如果共有 m 个文本，n 个单词，那么 V 的大小为 m*n，V（i,j）表示文档 i 中出现单词 j 的次数。
设定共有 r 个隐含的分类。通过一些方法，将 V 展开为两个相乘的矩阵：
V = W*H
其中，W 的大小为 m×r，H 的大小为 r×n。在隐语义模型中，W(i,j)被解释为文档 i 属于类别 j 的权重，H(a,b)被解释为单词 b 属于类别 a 的的权重。
对于一个文档，其权重最大的类别被看作是该文档的类别。由于设定共有 r 个隐含的分类，分类结果也是 r 个份分类。
采用矩阵分解技术发现两种实体见潜在的特征
使用矩阵分解来预测评分的思想来源于，我们可以通过矩阵分解来发现一些用户打分的潜在特征，比如两个人都喜欢某一演员，那他们就倾向于给该演员演的电影打高分；或者两个都喜欢动作片，假如我们能够发现这些特征，我们就能够预测特定用户对特定电影的打分。
为了发现不同的特征，我们假设特征的数量少于用户和电影的数量，矩阵分解算法的数学理论基础是矩阵的行列变换。在《线性代数》中，我们知道矩阵 A 进行行行变换相当于 A 左乘以一个矩阵，矩阵 A 进行列变换等价于矩阵 A 右乘以一个矩阵，因此矩阵 A 可以进行矩阵分解，一方面可以起到降低数据维度的作用，另一方面也可以找到潜在的特征。
矩阵分解可以开来非常好的结果，而且可以充分地考虑各种因素的影响，有非常好的扩展性，因为要考虑各种因素的综合作用，往往需要构造 cost function 来将矩阵分解转换为优化问题。根据要考虑的因素为优化问题其他限制条件，然后通过迭代的方法进行矩阵分解，原来评分矩阵中的 missing value 可以通过分解后得到的矩阵求得。
pureSVD
其实，矩阵分解的核心是将一个非常稀疏的评分矩阵分解为两个矩阵，一个表示用户 user 的特性，一个表示 item 的特性，将两个矩阵各自取一行一列向量做内积就可以得到对应评分。那么如何将一个矩阵分解为两个矩阵就是唯一的问题啦。
这里可以借用在线性代数和数值分析中学到的各种矩阵分解方法，如 QR, Jordan，三角分解，SVD。这里说明下 svd 方法的大概意思：将一个任意实矩阵分解为桑格矩阵 U,S,V，其中 U , V 是两个正交矩阵，称为左右奇异矩阵，S 是个对称阵，称为奇异值矩阵。
Latent Factor Model
这是真正的矩阵分解算法，经过实践检验，具有非常高的准确性和易扩展性。正如上面提到的，实现推荐系统结果的目标是将那个稀疏的评分矩阵分解成为两个矩阵，一个表示 user 的 feature，一个表示 item 的 feature，然后做内积得到预测。
当然要实现满足一定约束条件的矩阵分解，不像上面的 pureSVD 那么容易，需要构造一个优化问题，用一些复杂的算法求最优化问题。而这些最优化问题往往是 NP 问题，只有局部最优解。首先构造一个优化目标函数，考虑到最后的评价指标是预测分值与实际分值之间的误差的平方（RMSE）,所以构造的目标函数也是误差的平方的函数。为什么这样的算法可以得到更优的结果呢？因为算法可以很容易地扩展很多的 feature 进来，更加出色地考虑了多种影响推荐效果的实实在在的因素。
Biases 用户评分的偏见：

因为有的用户总是会打出比别人高的分，或者说有的用户他的评价尺度比较宽松，同样有的 item 总是被打高分，所以在真正实
践中，构造目标函数需要增加几项：所有评分的平均值 miu,user 的偏见分数，item 的偏见分数。
implicit feedback 隐式反馈数据
用户在使用 web 应用的时候，会产生大量的行为，充分挖掘这部分的价值，将会很好的提升推荐的效果，利用用户的隐式反
馈，相当于充分的利用评分矩阵中的 missing value 的价值，用户在页面的停留时间，检索，浏览，点击等等各种行为都可
以建模到目标函数中。在看到的论文中，这里，可以将简单的用一个 boolean 来描述一种行为有还是没有。补过在使用的使用
需要进行归一化处理。
User-associated attributes
基于用户的社会化属性进行推荐也是一种很基本的推荐，当然也可以考虑到目标函数中。
Temporal dynamics
用户的兴趣包括长期和短期，动态地考虑一段时间内用户的兴趣是很有必要的。
Confidence level
因为在处理上述的因素的时候，很多都是比较主观的，所以需要给每个因素添加一个置信权重，以平衡整个结果。
通过构造出这个目标函数，然后添加相应的约束条件，接下来的问题就是求解这个优化问题，通常比较好的方法是 随机梯度下降算法（Stochastic gradient desent）
这种方法在学术上主流的方法，参考 http:research.yahoo.com/Yehuda_Koren 以及上海交大在 kddcup 的论文集开源系统 http://apex.sjtu.edu.cn/apex_wiki/svdfeature
非负矩阵分解
基本原理：NMF，非负矩阵分解，它的目标很明确，就是将大矩阵分解为两个小矩阵，使得这两个小矩阵相乘后能够还原到大矩阵。而非负表示分解的矩阵都不包含负值。 从应用的角度来说，矩阵分解能够用于发现两种实体间的潜在特征，一个最常见的应用就是协同过滤中的预测打分值，而从协同过滤的这个角度来说，非负也很容易理解：打分都是正的，不会出现负值。
考虑到 svd 或者 latent factor model 会得到负值的情况，所以非负矩阵分解的物理意义比较明确。同样的道理，NMF 也是将评分矩阵的转置矩阵分解成两个矩阵。*不同的是这两个矩阵有着和上面的矩阵不同的物理意义。 其中一个是基矩阵 W，另一个是投影矩阵 H，即 R（nm）=W(nr)H(r*m)。
W：每一列包含一个基向量，这组基向量构成一个 r 维的空间。
H： 每一列则近似为原始数据对应的列向量在该 r 维空间的投影。
做这个分解的方法也是构造一个目标函数和一些约束条件，然后用梯度下降的算法计算得到一个局部最优解。这种方法的大概思路
1 将评分矩阵转置然后分解称为两个矩阵 W 和 H。
2 根据基矩阵 W，可以计算得到目标用户评分向量 a 对基矩阵 W 的投影向量 h.
3 计算投影向量 h 与投影矩阵 H 各行之间的欧氏距离，将其中距离最小的前 k 个用户组成目标用户 a 的最近邻集合。
4 然后用皮尔逊讲最近邻集合中的数据进行加权计算，然后拍下进行推荐。
这种方法的思路和上面的两种还是不同相同的，主要是在计算目标用户的最近邻集合，主要思想还是 knn,只是在中间用了矩阵分解的方法降低了计算的时间复杂度。
参考：blog.csdn.net/zyvscc/article/details/7551842
www.cnblogs.com/LeftNotEasy/archive/2011/01/19/svd-and-applications.html

8、如何通俗理解奇异值分解

解析：

特征值和奇异值在大部分人的印象中，往往是停留在纯粹的数学计算中。而且线性代数或者矩阵论里面，也很少讲任何跟特征值与奇异值有关的应用背景。

奇异值分解是一个有着很明显的物理意义的一种方法，它可以将一个比较复杂的矩阵用更小更简单的几个子矩阵的相乘来表示，这些小矩阵描述的是矩阵的重要的特性。就像是描述一个人一样，给别人描述说这个人长得浓眉大眼，方脸，络腮胡，而且带个黑框的眼镜，这样寥寥的几个特征，就让别人脑海里面就有一个较为清楚的认识，实际上，人脸上的特征是有着无数种的，之所以能这么描述，是因为人天生就有着非常好的抽取重要特征的能力，让机器学会抽取重要的特征，SVD是一个重要的方法。

在机器学习领域，有相当多的应用与奇异值都可以扯上关系，比如做feature reduction的PCA，做数据压缩（以图像压缩为代表）的算法，还有做搜索引擎语义层次检索的LSI（Latent Semantic Indexing）

一、特征值与奇异值

特征值分解和奇异值分解在机器学习领域都是属于满地可见的方法。两者有着很紧密的关系，接下来会谈到特征值分解和奇异值分解的目的都是一样，就是提取出一个矩阵最重要的特征。先谈特征值分解。

1.1 特征值

如果说一个向量v是方阵A的特征向量，将一定可以表示成下面的形式：

$$Av = \lambda v$$

这时候λ就被称为特征向量v对应的特征值，一个矩阵的一组特征向量是一组正交向量。特征值分解是将一个矩阵分解成下面的形式：

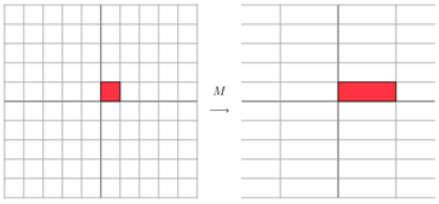
$$A = Q\Sigma Q^{-1}$$

其中Q是这个矩阵A的特征向量组成的矩阵，Σ是一个对角阵，每一个对角线上的元素就是一个特征值。我这里引用了一些参考文献中的内容来说明一下。

首先，要明确的是，一个矩阵其实就是一个线性变换，因为一个矩阵乘以一个向量后得到的向量，其实就相当于将这个向量进行了线性变换。比如说下面的一个矩阵：

$$M = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}$$

它其实对应的线性变换是下面的形式：



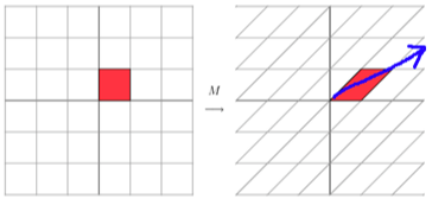
因为这个矩阵M乘以一个向量(x,y)的结果是：

$$\begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3x \\ y \end{bmatrix}$$

上面的矩阵是对称的，所以这个变换是一个对x，y轴的方向一个拉伸变换（每一个对角线上的元素将会对一个维度进行拉伸变换，当值>1时，是拉长，当值<1时时缩短），当矩阵不是对称的时候，假如说矩阵是下面的样子：

$$M = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

它所描述的变换是下面的样子：



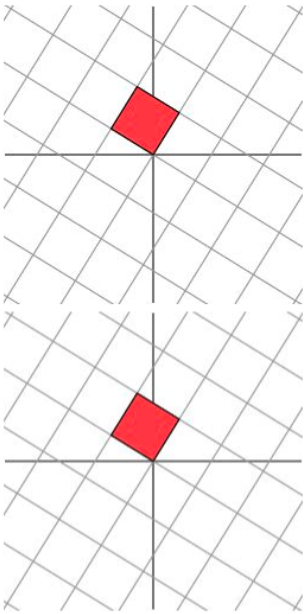
这其实是在平面上对一个轴进行的拉伸变换（如蓝色的箭头所示），在图中，蓝色的箭头是一个最主要的变化方向（变化方向可能有不止一个），如果我们想要描述好一个变换，那我们就描述好这个变换主要的变化方向就好了。反过头来看看之前特征值分解的式子，分解得到的Σ矩阵是一个对角阵，里面的特征值是由大到小排列的，这些特征值所对应的特征向量就是描述这个矩阵变化方向（从主要的变化到次要的变化排列）。

考虑更一般的非对称矩阵

$$M = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

很遗憾，此时我们再也找不到一组网格，使得矩阵作用在该网格上之后只有拉伸变换（找不到背后的数学原因是对一般非对称矩阵无法保证在实数域上可对角化，不明白也不要在意）。

我们退而求其次，找一组网格，使得矩阵作用在该网格上之后允许有拉伸变换和旋转变换，但要保证变换后的网格依旧互相垂直，这是可以做到的，如下图所示。



简言之，当矩阵是高维的情况下，那么这个矩阵就是高维空间下的一个线性变换，这个变换也同样有很多的变换方向，我们通过特征值分解得到的前N个特征向量，那么就对应了这个矩阵最主要的N个变化方向。我们利用这前N个变化方向，就可以近似这个矩阵（变换）。

也就是之前说的：提取这个矩阵最重要的特征。总结一下，特征值分解可以得到特征值与特征向量，特征值表示的是这个特征到底有多重要，而特征向量表示这个特征是什么，可以将每一个特征向量理解为一个线性的子空间，我们可以利用这些线性的子空间干很多的事情。不过，特征值分解也有很多的局限，比如说变换的矩阵必须是方阵。

下面我们就可以自然过渡到奇异值分解的引入。

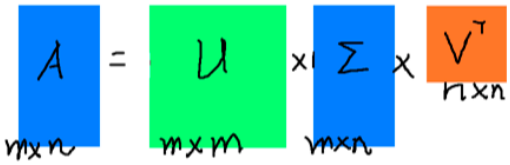
1.2 奇异值

下面谈谈奇异值分解。特征值分解是一个提取矩阵特征很不错的方法，但是它只是对方阵而言的，在现实的世界中，我们看到的大部分矩阵都不是方阵，比如说有N个学生，每个学生有M科成绩，这样形成的一个N * M的矩阵就不可能是方阵，我们怎样才能描述这样普通的矩阵呢的重要特征呢？

奇异值分解可以用来干这个事情，奇异值分解是一个能适用于任意的矩阵的一种分解的方法：

$$A = U\Sigma V^T$$

假设A是一个N * M的矩阵，那么得到的U是一个N * N的方阵（里面的向量是正交的，U里面的向量称为左奇异向量），Σ是一个N * M的矩阵（除了对角线的元素都是0，对角线上的元素称为奇异值），V'（V的转置)是一个N * N的矩阵，里面的向量也是正交的，V里面的向量称为右奇异向量），从图片来反映几个相乘的矩阵的大小可得下面的图片



那么奇异值和特征值是怎么对应起来的呢？首先，我们将一个矩阵A的转置 * A，将会得到一个方阵，我们用这个方阵求特征值可以得到：

$$(A^T A)v_i = \lambda_i v_i$$

这里得到的v，就是我们上面的右奇异向量。此外我们还可以得到：

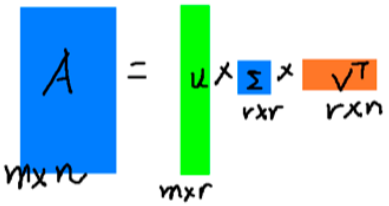
$$\sigma_i = \sqrt{\lambda_i}$$

$$u_i = \frac{1}{\sigma_i} A v_i$$

这里的σ就是上面说的奇异值，u就是上面说的左奇异向量。奇异值σ跟特征值类似，在矩阵Σ中也是从大到小排列，而且σ的减少特别的快，在很多情况下，前10%甚至1%的奇异值的和就占了全部的奇异值之和的99%以上了。也就是说，我们也可以用前r大的奇异值来近似描述矩阵，这里定义一下部分奇异值分解：

$$A_{m\times n} \approx U_{m\times r}\Sigma_{r\times r}V^T{}_{r\times n}$$

r是一个远小于m、n的数，这样矩阵的乘法看起来像是下面的样子：

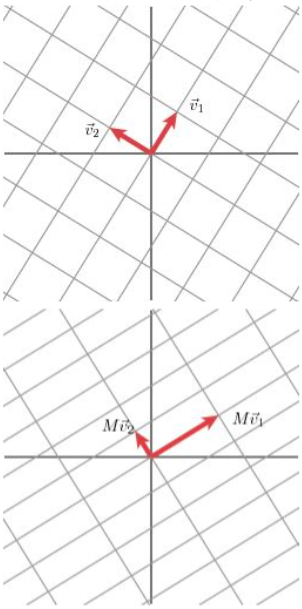


右边的三个矩阵相乘的结果将会是一个接近于A的矩阵，在这儿，r越接近于n，则相乘的结果越接近于A。而这三个矩阵的面积之和（在存储观点来说，矩阵面积越小，存储量就越小）要远远小于原始的矩阵A，我们如果想要压缩空间来表示原矩阵A，我们存下这里的三个矩阵：U、Σ、V就好了。

说句大白话，称作「奇异值」可能无法顾名思义迅速理解其本质，那咱们换个说法，称作「主特征值」，你可能就迅速了然了。

而奇异值分解的几何含义为：对于任何的一个矩阵，我们要找到一组两两正交单位向量序列，使得矩阵作用在此向量序列上后得到新的向量序列保持两两正交。

继续拿1.1节的例子进一步阐述，奇异值的几何含义为：这组变换后的新的向量序列的长度。



当矩阵M作用在正交单位向量v_2和v_2上之后，得到Mv_1和Mv_2也是正交的。令u_1和u_2分别是Mv_1和Mv_2方向上的单位向量，即$Mv_1 = \sigma_1 u_1$，$Mv_2 = \sigma_2 u_2$，写在一起就是$M\begin{bmatrix} v_1 & v_2 \end{bmatrix} = \begin{bmatrix} \sigma_1 u_1 & \sigma_2 u_2 \end{bmatrix}$，整理得：

$$M = M\begin{bmatrix} v_1 & v_2 \end{bmatrix}\begin{bmatrix} v_1^{\mathrm{T}} \\ v_2^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} \sigma_1 u_1 & \sigma_2 u_2 \end{bmatrix}\begin{bmatrix} v_1^{\mathrm{T}} \\ v_2^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} u_1 & u_2 \end{bmatrix}\begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix}\begin{bmatrix} v_1^{\mathrm{T}} \\ v_2^{\mathrm{T}} \end{bmatrix}$$

这样就得到矩阵M的奇异值分解。奇异值σ_1和σ_2分别是Mv_1和Mv_2的长度。很容易可以把结论推广到一般n维情形。

现在咱们给出一个更简洁更直观的奇异值的几何意义。先来一段线性代数的推导。

假设矩阵A的奇异值分解为

$$A = \begin{bmatrix} u_1 & u_2 \end{bmatrix}\begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} v_1^{\mathrm{T}} \\ v_2^{\mathrm{T}} \end{bmatrix}$$

其中u_1，u_2，v_1，v_2是二维平面的向量。根据奇异值分解的性质，u_1，u_2线性无关，v_1，v_2线性无关。那么对二维平面上任意的向量x，都可以表示为：$x = \xi_1 v_1 + \xi_2 v_2$。

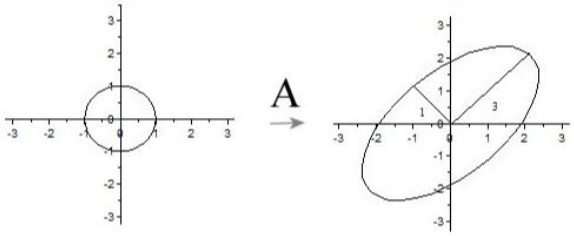
当A作用在x上时，

$$y = Ax = A[v_1 \quad v_2]\begin{bmatrix} \xi_1 \\ \xi_2 \end{bmatrix} = \begin{bmatrix} u_1 & u_2 \end{bmatrix}\begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} v_1^{\mathrm{T}} \\ v_2^{\mathrm{T}} \end{bmatrix}[v_1 \quad v_2]\begin{bmatrix} \xi_1 \\ \xi_2 \end{bmatrix} = 3\xi_1 u_1 + \xi_2 u_2$$

令$\eta_1 = 3\xi_1$，$\eta_2 = \xi_2$，我们可以得出结论：如果x是在单位圆$\xi_1^2 + \xi_2^2 = 1$上，那么y正好在椭圆$\eta_1^2/3^2 + \eta_2^2/1^2 = 1$上。这表明：矩阵A将二维平面中单位圆变换成椭圆，而两个奇异值正好是椭圆的两个半轴长，长轴所在的直线是$\mathrm{span}\{u_1\}$，短轴所在的直线是$\mathrm{span}\{u_2\}$。

推广到一般情形：一般矩阵A将单位球$\|x\|_2 = 1$变换为超椭球面$E_m = \{y \in \mathbf{C}^m : y = Ax,\ x \in \mathbf{C}^n,\ \|x\|_2 = 1\}$

那么矩阵A的每个奇异值恰好就是超椭球的每条半轴长度。



奇异值的计算是一个难题，是一个O(N^3)的算法。在单机的情况下当然是没问题的，matlab在一秒钟内就可以算出1000 * 1000的矩阵的所有奇异值，但是当矩阵的规模增长的时候，计算的复杂度呈3次方增长，就需要并行计算参与了。Google的吴军老师在数学之美系列谈到SVD的时候，说起Google实现了SVD的并行化算法，说这是对人类的一个贡献，但是也没有给出具体的计算规模，也没有给出太多有价值的信息。

其实SVD还是可以用并行的方式去实现的，在解大规模的矩阵的时候，一般使用迭代的方法，当矩阵的规模很大（比如说上亿）的时候，迭代的次数也可能会上亿次，如果使用Map-Reduce框架去解，则每次Map-Reduce完成的时候，都会涉及到写文件、读文件的操作。个人猜测Google云计算体系中除了Map-Reduce以外应该还有类似于MPI的计算模型，也就是节点之间是保持通信，数据是常驻在内存中的，这种计算模型比Map-Reduce在解决迭代次数非常多的时候，要快了很多倍。

Lanczos迭代就是一种解对称方阵部分特征值的方法（之前谈到了，解A’ * A得到的对称方阵的特征值就是解A的右奇异向量），是将一个对称的方程化为一个三对角矩阵再进行求解。按网上的一些文献来看，Google应该是用这种方法去做的奇异值分解的。请见Wikipedia上面的一些引用的论文，如果理解了那些论文，也“几乎”可以做出一个SVD了。

二、奇异值的直观应用

2.1 女神图片压缩

下面，咱们从女神上野树里（Ueno Juri）的一张像素为高度450*宽度333的照片，来直观理解奇异值在物理上到底代表什么意义（请屏幕前的痴汉暂停舔屏）。

我们都知道，图片实际上对应着一个矩阵，矩阵的大小就是像素大小，比如这张图对应的矩阵阶数就是450*333，矩阵上每个元素的数值对应着像素值。我们记这个像素矩阵为A

我们都知道，图片实际上对应着一个矩阵，矩阵的大小就是像素大小，比如这张图对应的矩阵阶数就是450*333，矩阵上每个元素的数值对应着像素值。我们记这个像素矩阵为A

现在我们对矩阵A进行奇异值分解。直观上，奇异值分解将矩阵分解成若干个秩一矩阵之和，用公式表示就是：

$$A = \sigma_1 u_1 v_1^{\mathrm{T}} + \sigma_2 u_2 v_2^{\mathrm{T}} + \ldots + \sigma_r u_r v_r^{\mathrm{T}} \tag{1}$$

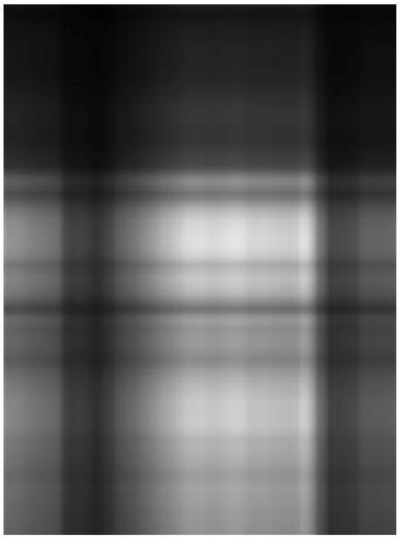
其中等式右边每一项前的系数σ就是奇异值，u和v分别表示列向量，秩一矩阵的意思是矩阵秩为1。注意到每一项uv^{T}都是秩为1的矩阵。我们假定奇异值满足（奇异值大于0是个重要的性质，但这里先别在意）

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$$

如果不满足的话重新排列顺序即可，这无非是编号顺序的问题。

既然奇异值有从大到小排列的顺序，我们自然要问，如果只保留大的奇异值，舍去较小的奇异值，这样(1)式里的等式自然不再成立，那会得到怎样的矩阵——也就是图像？

令$A_1 = \sigma_1 u_1 v_1^{\mathrm{T}}$，这只保留(1)中等式右边第一项，然后作图：



结果就是完全看不清是啥……我们试着多增加几项进来：$A_5 = \sigma_1 u_1 v_1^{\mathrm{T}} + \sigma_2 u_2 v_2^{\mathrm{T}} + \sigma_3 u_3 v_3^{\mathrm{T}} + \sigma_4 u_4 v_4^{\mathrm{T}} + \sigma_5 u_5 v_5^{\mathrm{T}}$
再作图

结果就是完全看不清是啥……我们试着多增加几项进来：$A_5 = \sigma_1 u_1 v_1^{\mathrm{T}} + \sigma_2 u_2 v_2^{\mathrm{T}} + \sigma_3 u_3 v_3^{\mathrm{T}} + \sigma_4 u_4 v_4^{\mathrm{T}} + \sigma_5 u_5 v_5^{\mathrm{T}}$
再作图

隐约可以辨别这是短发伽椰子的脸……但还是很模糊，毕竟我们只取了5个奇异值而已。下面我们取20个奇异值试试，也就是(1)式等式右边取前20项构成A_{20}

虽然还有些马赛克般的模糊，但我们总算能辨别出这是Juri酱的脸。当我们取到(1)式等式右边前50项时：

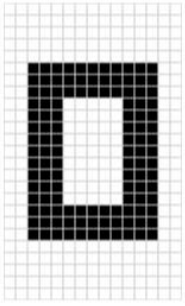
我们得到和原图差别不大的图像。也就是说当k从1不断增大时，A_k不断的逼近A。让我们回到公式(1)　　$A = \sigma_1 u_1 v_1^{\mathrm{T}} + \sigma_2 u_2 v_2^{\mathrm{T}} + \ldots + \sigma_r u_r v_r^{\mathrm{T}}$

矩阵A表示一个450*333的矩阵，需要保存$450 \times 333 = 149850$个元素的值。等式右边u和v分别是450*1和333*1的向量，每一项有1+450+333=784个元素。如果我们要存储很多高清的图片，而又受限于存储空间的限制，在尽可能保证图像可被识别的精度的前提下，我们可以保留奇异值较大的若干项，舍去奇异值较小的项即可。例如在上面的例子中，如果我们只保留奇异值分解的前50项，则需要存储的元素为$784 \times 50 = 39200$，和存储原始矩阵A相比，存储量仅为后者的26%。

奇异值往往对应着矩阵中隐含的重要信息，且重要性和奇异值大小正相关。每个矩阵A都可以表示为一系列秩为1的“小矩阵”之和，而奇异值则衡量了这些“小矩阵”对于A的权重。

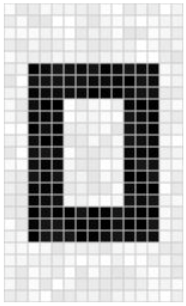
2.2 图像去噪

在图像处理领域，奇异值不仅可以应用在数据压缩上，还可以对图像去噪。如果一副图像包含噪声，我们有理由相信那些较小的奇异值就是由于噪声引起的。当我们强行令这些较小的奇异值为0时，就可以去除图片中的噪声。如下是一张25*15的图像

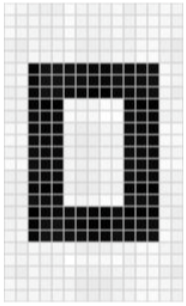


但往往我们只能得到如下带有噪声的图像（和无噪声图像相比，下图的部分白格子中带有灰色）：

但往往我们只能得到如下带有噪声的图像（和无噪声图像相比，下图的部分白格子中带有灰色）：



通过奇异值分解，我们发现矩阵的奇异值从大到小分别为：14.15，4.67，3.00，0.21，……，0.05。除了前3个奇异值较大以外，其余奇异值相比之下都很小。强行令这些小奇异值为0，然后只用前3个奇异值构造新的矩阵，得到



可以明显看出噪声减少了（白格子上灰白相间的图案减少了）。

奇异值分解还广泛的用于主成分分析（Principle Component Analysis，简称PCA）和推荐系统（如Netflex的电影推荐系统）等。在这些应用领域，奇异值也有相应的意义。

参考文献

1 https://www.cnblogs.com/LeftNotEasy/archive/2011/01/19/svd-and-applications.html
2 https://www.zhihu.com/question/22237507
3 We Recommend a Singular Value Decomposition（Feature Column from the AMS）

9、请通俗的解释下什么叫张量分解?

解析：

在介绍张量分解（tensor decomposition）之前，我们可能需要先简单地了解一下张量是什么，然后再考虑张量分解有什么用途，并如何像稀疏矩阵分解（matrix decomposition/factorization）一样来对稀疏张量进行分解。

从初中到大学，我们接触最多的可能只是标量（scalar）、向量（vector）和矩阵（matrix），而张量则不那么常见，但实际上，标量是第0阶张量，向量是第1阶张量，矩阵是第2阶张量，第3阶或阶数更高的张量被称为高阶张量（higher-order tensor），一般提到的张量都是特指高阶张量，这在接下来的叙述中也不例外。

我们也知道，在一个矩阵中，某一元素的位置可以说成“第i行第j列”的形式，要表达某一元素的位置需要两个索引构成的组合(i,j)，类似地，在一个第3阶张量里面，表达某一元素的位置需要三个索引构成的组合(i,j,k) 。在处理稀疏矩阵和稀疏张量时，用索引来标记元素的位置会带来很多便利。

另外，阶数$d \geq 3$的张量可以理解为矩阵的d维泛化，在这里，阶数d其实就是空间维度（spatial dimension），张量可以被视为多维数组。

张量分解从本质上来说是矩阵分解的高阶泛化。对矩阵分解有所了解的读者可能知道，矩阵分解有三个很明显的用途，即降维处理、缺失数据填补（或者说成“稀疏数据填补”）和隐性关系挖掘，其实张量分解也能够很好地满足这些用途。因此，在介绍张量分解之前，我们先来看看相对简单的矩阵分解，并掌握稀疏矩阵的分解过程。

1 推荐系统中常用的矩阵分解

在我们常见的推荐系统（如商品推荐系统、电影推荐系统等）中，给定一个大小为$m \times n$的评分矩阵R，元素r_{ij}表示用户（user）i对项（item，如电影、商品等）j的评分值，如1~5分不等。

当矩阵R的秩为$k = rank\,(R) \ll \min\,(m, n)$，并且能够写成如下形式

$$R = UV^T$$

其中，U是大小为$m \times k$的矩阵（用户因子矩阵，user-factor matrix），V是大小为$n \times k$的矩阵（项因子矩阵，item-factor matrix）。这一过程就是矩阵分解。

当$k < rank\,(R)$时，我们可以将矩阵分解的过程看作是一个低秩逼近问题（low-rank approximation problem），原来的分解过程则变成

$$R \approx UV^T$$

和前面相同，U是大小为$m \times k$的矩阵（用户因子矩阵，user-factor matrix），V是大小为$n \times k$的矩阵（项因子矩阵，item-factor matrix）。在这个低秩逼近问题中，可以很明显得看出整体误差为残差矩阵$R - UV^T$中所有元素的平方和，即$||R - UV^T||^2$（这种写法含义是矩阵F-范数的平方，等价于矩阵中所有元素的平方和）。

如果简单地使总的误差最小，那么，可以将矩阵分解的逼近问题转化为一个无约束的优化问题，即

$$\min J = \frac{1}{2}||R - UV^T||^2$$

在实际应用中，这里的评分矩阵R往往是一个稀疏矩阵，即很多位置上的元素是空缺的，或者说根本不存在。试想一下，如果有10000个用户，同时存在10000部电影，如果我们需要构造一个评分矩阵，难道每个用户都要把每部电影都看一遍才知道用户的偏好吗？其实不是，我们只需要知道每个用户仅有的一些评分就可以利用矩阵分解来估计用户的偏好，并最终推荐用户可能喜欢的电影。

在这里，我们将矩阵R中存在评分的位置记为(i, j)，所有观测到的位置索引记作集合S，其中，用户的索引为$i \in \{1, 2, \ldots, m\}$，项的索引为$j \in \{1, 2, \ldots, n\}$，需要注意的是，推荐系统中的矩阵分解对原矩阵R有一定的要求，即矩阵的每行和每列至少有一个元素。此时，任意位置(i, j)所对应的评分估计值为$\hat{r}_{ij} = \left(UV^T\right)_{ij} = \sum_{q=1}^{k} u_{iq} \cdot v_{jq}$。

则原来的优化问题等价于

$$\min J = \frac{1}{2} \sum_{(i,j) \in S} e_{ij}^2 = \frac{1}{2} \sum_{(i,j) \in S} \left(r_{ij} - \sum_{q=1}^{k} u_{iq} \cdot v_{jq}\right)^2$$

对目标函数J中的u_{iq}和v_{jq}求偏导数，得

$$\frac{\partial J}{\partial v_{jq}} = \sum_{i:(i,j) \in S} \left(r_{ij} - \sum_{q=1}^{k} u_{iq} v_{jq}\right)(-u_{iq})$$

;

$$\frac{\partial J}{\partial u_{iq}} = -\sum_{j:(i,j) \in S} e_{ij} v_{jq}$$

这里，可以将两个偏导数分别简写为$\frac{\partial J}{\partial u_{iq}} = -\sum_{j:(i,j) \in S} e_{ij} v_{jq}$和$\frac{\partial J}{\partial v_{jq}} = -\sum_{i:(i,j) \in S} e_{ij} u_{iq}$，其中，$j \in \{1, 2, \ldots, n\}$，$q \in \{1, 2, \ldots, k\}$，$u_{iq} \Leftarrow u_{iq} + \alpha \sum_{j:(i,j) \in S} e_{ij} v_{jq}$。

根据梯度下降（gradient descent）方法，u_{iq}和v_{jq}在每次迭代过程中的更新公式为

$$u_{iq} \Leftarrow u_{iq} + \alpha \sum_{i:(i,j) \in S} e_{ij} v_{jq} \quad v_{jq} \Leftarrow v_{jq} + \alpha \sum_{i:(i,j) \in S} e_{ij} u_{iq}$$

这里的$\alpha > 0$表示梯度下降的步长，又称为学习率（learning rate），另外，更新公式中的求和项下标$j:(i,j)\in S$和$i:(i,j)\in S$分别表示向量$R(i,:)$和$R(:,j)$上所有非零元素的位置索引构成的集合。

2 隐性因子模型

上面已经简单地介绍了矩阵分解的原理，对推荐系统有了解的读者可能对上面这一过程并不陌生，上面的矩阵分解在推荐系统中常常被称为隐性因子模型（latent factor model, LFM），其实张量分解与上述过程非常相似，为了便于读者初步地理解张量分解，这里将会沿用矩阵分解类似的推导过程。

定义一个关于用户（user）i在环境（context，如时间，注意：这里将context翻译成“环境”不一定准确，在隐性语义分析中常常理解为“语境”或“上下文”）c下对项（item）j的评分为r_{ijc}，评分张量的大小为$m\times n\times d$，所有观测到位置索引仍然记作集合S，其中，用户的索引为$i\in\{1,2,\ldots,m\}$，项的索引为$j\in\{1,2,\ldots,n\}$，环境的索引为$c\in\{1,2,\ldots,d\}$。

Charu C. Aggarwal在其著作《Recommender systems》中给出了一个特殊的张量分解结构，即大小为$m\times n\times d$的评分张量$\mathcal{R}$分解后会得到三个矩阵，这三个矩阵分别是：大小为$m\times k$的用户因子矩阵U（user-factor matrix）、大小$n\times k$为的项因子矩阵V（item-factor matrix）和大小为$d\times k$的环境因子矩阵W（context-factor matrix），这种分解结构是隐性因子模型的一种高阶泛化。

此时，第3阶张量$\mathcal{R}$上任意位置(i,j,c)所对应的评分估计值为

$$\hat{r}_{ijc}=\left(UV^T\right)_{ic}+\left(UW^T\right)_{ic}+\left(VW^T\right)_{jc}$$

即

$$\hat{r}_{ijc}=\sum_{q=1}^{k}\left(u_{iq}v_{jq}+u_{iq}w_{cq}+v_{jq}w_{cq}\right)$$

与矩阵分解中的低秩逼近问题相似，评分张量分解的逼近问题为

$$\min J=\frac{1}{2}\sum_{(i,j,c)\in S}e_{ijc}^2=\frac{1}{2}\sum_{(i,j,c)\in S}\left(r_{ijc}-\sum_{q=1}^{k}\left(u_{iq}v_{jq}+u_{iq}w_{cq}+v_{jq}w_{cq}\right)\right)^2$$

对目标函数J中的u_{iq}、v_{jq}和w_{cq}求偏导数，得

$$\frac{\partial J}{\partial u_{iq}}=-\sum_{j,c:(i,j,c)\in S}e_{ijc}\cdot\left(v_{jq}+w_{cq}\right);$$

$$\frac{\partial J}{\partial v_{jq}}=-\sum_{i,c:(i,j,c)\in S}e_{ijc}\cdot\left(u_{iq}+w_{cq}\right);$$

$$\frac{\partial J}{\partial w_{cq}}=-\sum_{i,j:(i,j,c)\in S}e_{ijc}\cdot\left(u_{iq}+v_{jq}\right)$$

根据梯度下降方法，u_{iq}、v_{jq}和w_{cq}在每次迭代过程中的更新公式为

$$u_{iq}\Leftarrow u_{iq}+\alpha\sum_{j,c:(i,j,c)\in S}e_{ijc}\cdot\left(v_{jq}+w_{cq}\right);$$

$$v_{jq}\Leftarrow v_{jq}+\alpha\sum_{i,c:(i,j,c)\in S}e_{ijc}\cdot\left(u_{iq}+w_{cq}\right);$$

$$w_{cq}\Leftarrow w_{cq}+\alpha\sum_{i,j:(i,j,c)\in S}e_{ijc}\cdot\left(u_{iq}+v_{jq}\right).$$

需要注意的是，更新公式中的求和项下标$j,c:(i,j,c)\in S$、$i,c:(i,j,c)\in S$和$i,j:(i,j,c)\in S$分别表示矩阵$\mathcal{R}(i,:,:)$、$\mathcal{R}(:,j,:)$和$\mathcal{R}(:,:,c)$上所有非零元素的位置索引构成的集合。

介绍到这里，可能部分读者会疑问，这里介绍的张量分解过程为何与我们常在大量文献中见到的Tucker张量分解和CP张量分解（可认为是Tucker张量分解的特例）不一样呢？接下来我们来看看采用Tucker分解的情况是怎样的。

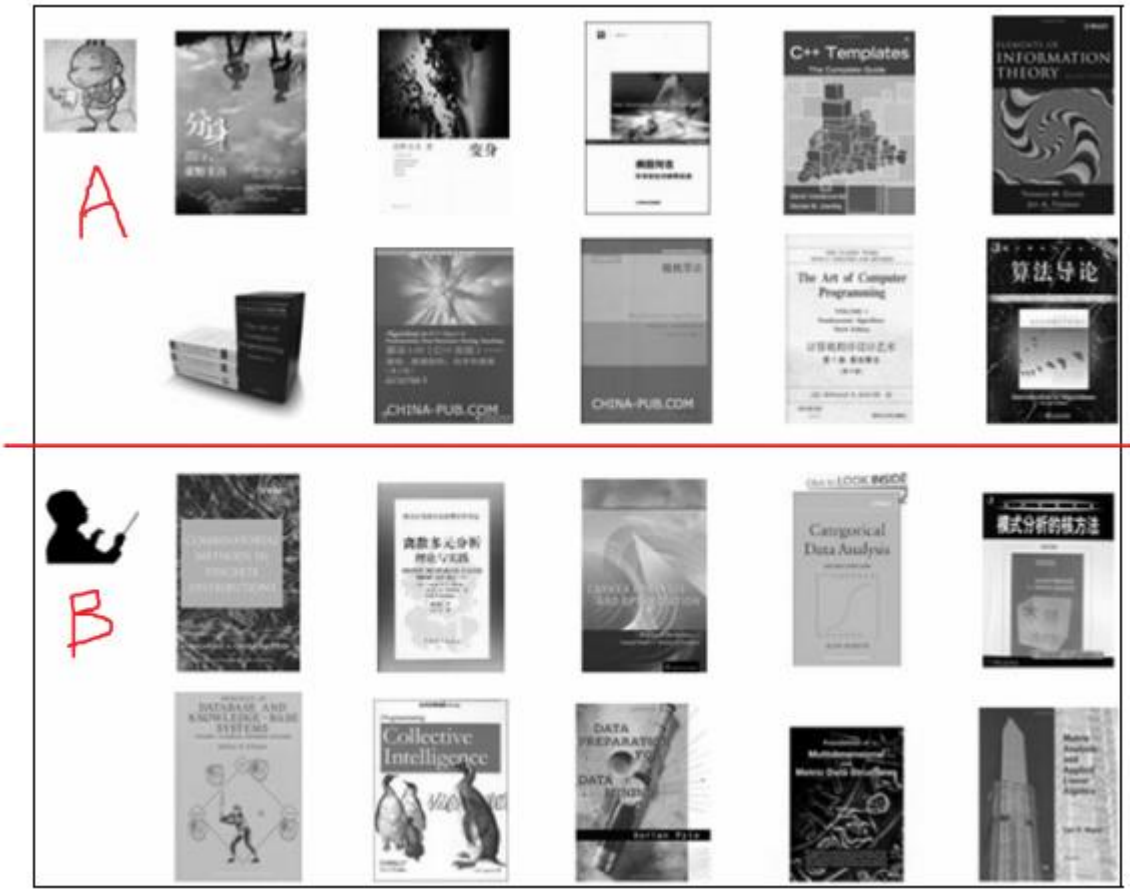
参考文献 1

10、请说说隐语义模型 LFM 背后的原理

一、隐语义模型的基本思想

隐语义模型是近年来推荐系统领域较为热门的话题，它主要是根据隐含特征将用户与物品联系起来。

现从简单例子出发介绍隐语义模型的基本思想。假设用户 A 喜欢《算法导论》，用户 B 喜欢《离散多元分析》，现在小编要对用户 A 和用户 B 推荐其他书籍。



基于 UserCF(基于用户的协同过滤)，找到与他们偏好相似的用户，将相似用户偏好的书籍推荐给他们；

基于 ItemCF(基于物品的协同过滤)，找到与他们 当前偏好书籍相似的其他书籍，推荐给他们。

其实还有一种思路，就是根据用户的当前偏好信息，得到用户的兴趣偏好，将该类兴趣对应的物品推荐给当前用户。比 如，用户 A 喜欢的《算法导论》属于计算机类的书籍，那我们可以将其他的计算机类书籍推荐给用户 A；用户 B 喜欢的是数学类数据，可将《微积分概念发展史》等这类文字作品推荐给用户 B。这就是隐语义模型，依据“兴趣”这一隐含特征将用户与物品进行连接，需要说明的是此处的“兴趣”其实是对物品类型的一个分类而已。

说白了，就是找出隐含的特征、关联，然后根据做关联推荐。

二、隐语义模型的数学理解

我们从数学角度来理解隐语义模型。如下图所示，R 矩阵是用户对物品的偏好信息（Rij 表示的是 user i 对 item j 的兴趣度），P 矩阵是用户对各物品类别的一个偏好信息（Pij 表示的是 user i 对 class j 的兴趣度），Q 矩阵是各物品所归属的的物品类别的信息（Qij 表示的是 item j 在 class i 中的权重）。

隐语义模型就是要将矩阵 R 分解为矩阵 P 和矩阵 Q 的乘积，即通过矩阵中的物品类别(class)将用户 user 和物品 item 联系起来。实际 上我们需要根据用户当前的物品偏好信息 R 进行计算，从而得到对应的矩阵 P 和矩阵 Q。

	item 1	item 2	item 3	item 4
user 1	R11	R12	R13	R14
user 2	R21	R22	R23	R24
user 3	R31	R32	R33	R34

R

=

	class 1	class 2	class 3
user 1	P11	P12	P13
user 2	P21	P22	P23
user 3	P31	P32	P33

P

×

	item 1	item 2	item 3	item 4
class 1	Q11	Q12	Q13	Q14
class 2	Q21	Q22	Q23	Q24
class 3	Q31	Q32	Q33	Q34

Q

三、隐语义模型所解决的问题

从上面的陈述我们可以知道，要想实现隐语义模型，我们需解决以下问题：

如何对物品进行分类，分成几类？

如何确定用户对哪些物品类别有兴趣，兴趣程度如何？

对于一个给定的类，选择这个类中的哪些物品进行推荐，如何确定物品在某个类别中的权重？

对于第一个问题，就是找对应的编辑人进行人工分类，但是人工分类会存在以下问题：

当前的人工分类不能代表用户的意见：比如一本《算法导论》，从编辑人员的角度看会属于计算机类，但从用户来看，可能会归到数学类；难以把握分类的粒度：仍以《算法导论》为例，可分类得粗一点，属于计算机类，也可分得再细一点，属于计算机类别中的算法类；

难以给一个物品多个类别：一本小说可归为文学类，或言情类等；

难以给出多维度的分类：对物品的分类可从多个角度进行，比如一本书可从内容进行分类，也可从作者角度进行分类；

难以确定一个物品在某一分类中的权重：一个物品可能属于多个类别，但是权重不同。

基于以上局限性，显然我们不能靠由个人的主观想法建立起来的分类标准对整个平台用户喜好进行标准化。

隐语义模型是从用户的偏好数据出发进行个性推荐的，即基于用户的行为统计进行自动聚类，所以能解决以上提到的 5 个问题：

隐语义模型是基于用户的行为数据进行自动聚类的，能反应用户对物品的分类意见；

我们可以指定将物品聚类的类别数 k，k 越大，则粒度越细；

隐语义模型能计算出物品在各个类别中的权重，这是根据用户的行为数据统计的，不会只将其归到一类中；

隐语义模型得到的物品类别不是基于同一个维度的，维度是由用户的共同兴趣决定的。

四、隐语义模型的样本问题

隐语义模型在显性反馈数据（也就是评分数据）上能解决评分预测问题并达到了很好的精度。不过推荐系统主要讨论的是隐性反馈数据集，这种数据集的特点 是只有正样本（用户喜欢什么物品），而没有负样本（用户对什么物品不感兴趣）。

那么，在隐性反馈数据集上应用隐语义模型解决推荐的第一个关键问题就是如何给每个用户生成负样本。我们发现对负样本采样时应该 遵循以下原则：

对每个用户，要保证正负样本的平衡（数目相似）；

每个用户采样负样本时，要选取那些很热门，而用户却没有行为的物品：一般认为，很热门而用户却没有行为更加代表用户对这个物品不感兴趣。因为对于冷门的物 品，用户可能是压根没在网站中发现这个物品，所以谈不上是否感兴趣；

五、隐语义模型的推导思路

隐语义模型是根据如下公式来计算用户 U 对物品 I 的兴趣度。

$$R_{UI} = P_U Q_I = \sum_{k=1}^{K} P_{U,k} Q_{k,I}$$

其中，隐语义模型会把物品分成 K 个类型，这个是我们根据经验和业务知识进行反复尝试决定的，p(u,k)表示用户 u 对于第 k 个分类的喜爱程度(1 < k <= K)，q(k, i)表示物品 i 属于第 k 个分类的权重(1 < k <= K)。

现在我们讨论下如何计算矩阵 P 和矩阵 Q 中的参数值。一般做法就是最优化损失函数来求参数。损失函数如下所示:

$$C = \sum_{(U,I)\in K} (R_{UI} - \hat{R}_{UI})^2 = \sum_{(U,I)\in K} (R_{UI} - \sum_{k=1}^{K} P_{U,k} Q_{k,I})^2 + \lambda \|P_U\|^2 + \lambda \|Q_I\|^2$$

上式中的 $\lambda \|P_U\|^2 + \lambda \|Q_I\|^2$ 是用来防止过拟合的正则化项，λ 需要根据具体应用场景反复实验得到。损失函数的意义是用户 u 对物品 i 的真实喜爱程度与推算出来的喜爱程度的均方根误差，通俗来说就是真实的喜爱程度与推算的喜爱程度的误差，要使模型最合理当然就是使这个误差达到最小值。公式中最后两项是惩罚因子，用来防止分类数取得过大而使误 差减少的不合理做法的发生，λ 参数是一个常数，需要根据经验和业务知识进行反复尝试决定的。损失函数的优化使用随机梯度下降算法：1）对两组未知数求偏导数

$$\frac{\partial C}{\partial P_{Uk}} = -2(R_{UI} - \sum_{k=1}^{K} P_{U,k} Q_{k,I}) Q_{kI} + 2\lambda P_{Uk}$$

$$\frac{\partial C}{\partial Q_{kI}} = -2(R_{UI} - \sum_{k=1}^{K} P_{U,k} Q_{k,I}) P_{Uk} + 2\lambda Q_{kI}$$

2）根据随机梯度下降法得到递推公式

$$P_{Uk} = P_{Uk} + \alpha((R_{UI} - \sum_{k=1}^{K} P_{U,k} Q_{k,I}) Q_{kI} - \lambda P_{Uk})$$

$$Q_{kI} = Q_{kI} + \alpha((R_{UI} - \sum_{k=1}^{K} P_{U,k} Q_{k,I}) P_{Uk} - \lambda Q_{kI})$$

其中 α 是在梯度下降的过程中的步长(也可以称作学习速率)，这个值不宜过大也不宜过小，过大会产生震荡而导致很难求得最小值，过小会造成计算速度下降，需要经过试验得到最合适的值。最终会求得每个用户对于每个隐分类的喜爱程度矩阵 P 和每个物品与每个隐分类的匹配程度矩阵 Q。

在用户对物品的偏好信息矩阵 R 中，通过迭代可以求得每个用户对每个物品的喜爱程度,选取喜爱程度最高而且用户没有反馈过的物品进行推荐。

在隐语义模型中，重要的参数有以下 4 个:

1）隐分类的个数 F;

2）梯度下降过程中的步长(学习速率)α;

3）损失函数中的惩罚因子 λ;

4）正反馈样本数和负反馈样本数的比例 ratio；这四项参数需要在试验过程中获得最合适的值 1）3）4）这三项需要根据推荐系统的准确率、召回率、覆盖率及流行度作为参考，而 2）步长 α 要参考模型的训练效率。

六、优缺点分析

隐语义模型在实际使用中有一个困难，那就是它很难实现实时推荐。经典的隐语义模型每次训练时都需要扫描所有的用户行为记录，这样才能计算出用户对于 每个隐分类的喜爱程度矩阵 P 和每个物品与每个隐分类的匹配程度矩阵 Q。而且隐语义模型的训练需要在用户行为记录上

反复迭代才能获得比较好的性能，因此 LFM 的每次训练都很耗时，一般在实际应用中只能每天训练一次，并且计算出所有用户的推荐结果。从而隐语义模型不能因为用户行为的变化实时地调整推荐结果 来满足用户最近的行为。

七、参考文献

1 https://www.jianshu.com/p/7b6bb28c1753?tdsourcetag=s_pcqq_aiomsg

2

11、请详细说说你对 Learning to rank 的通俗理解

引言

我们正处在一个知识爆炸的时代，伴随着信息量的剧增和人工智能的蓬勃发展，互联网公司越发具有强烈的个性化、智能化信息展示的需求。而信息展示个性化的典型应用主要包括搜索列表、推荐列表、广告展示等等。

很多人不知道的是，看似简单的个性化信息展示背后，涉及大量的数据、算法以及工程架构技术，这些足以让大部分互联网公司望而却步。究其根本原因，个性化信息展示背后的技术是排序学习问题（Learning to Rank）。市面上大部分关于排序学习的文章，要么偏算法、要么偏工程。虽然算法方面有一些系统性的介绍文章，但往往对读者的数学能力要求比较高，也比较偏学术，对于非算法同学来说门槛非常高。而大部分工程方面的文章又比较粗糙，基本上停留在 Google 的 Two-Phase Scheme 阶段，从工程实施的角度来说，远远还不够具体。

对于那些由系统开发工程师负责在线排序架构的团队来说，本文会采用通俗的例子和类比方式来阐述算法部分，希望能够帮助大家更好地理解和掌握排序学习的核心概念。如果是算法工程师团队的同学，可以忽略算法部分的内容。本文的架构部分阐述了美团点评到店餐饮业务线上运行的系统，可以作为在线排序系统架构设计的参考原型直接使用。该架构在服务治理、分层设计的理念，对于保障在线排序架构的高性能、高可用性、易维护性也具有一定的参考价值。包括很多具体环节的实施方案也可以直接进行借鉴，例如流量分桶、流量分级、特征模型、级联模型等等。

总之，让开发工程师能够理解排序学习算法方面的核心概念，并为在线架构实施提供细颗粒度的参考架构，是本文的重要目标。

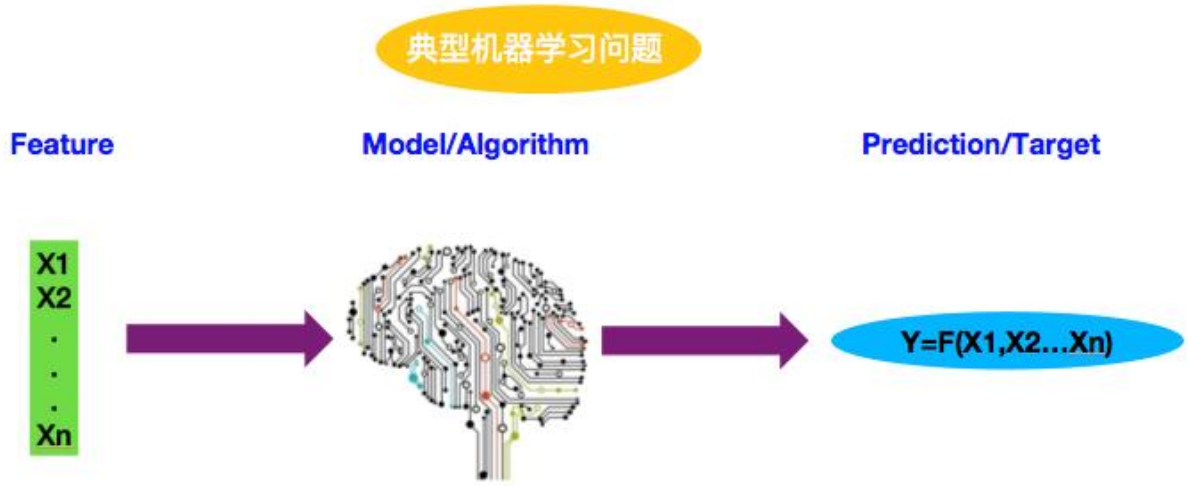
算法部分

机器学习涉及优化理论、统计学、数值计算等多个领域。这给那些希望学习机器学习核心概念的系统开发工程师带来了很大的障碍。不过，复杂的概念背后往往蕴藏着朴素的道理。本节将尝试采用通俗的例子和类比方式，来对机器学习和排序学习的一些核心概念进行揭秘。

机器学习

什么是机器学习?

典型的机器学习问题，如下图所示:

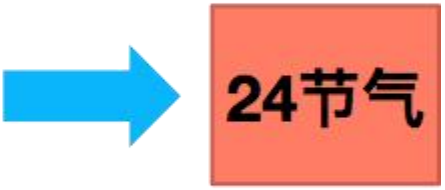


机器学习模型或算法（Model/Algorithm）会根据观察到的特征值（Feature）进行预测，给出预测结果或者目标（Prediction/Target)。这就像是一个函数计算过程，对于特定 X 值（Feature)，算法模型就像是函数，最终的预测结果是 Y 值。不难理解，机器学习的核心问题就是如何得到预测函数。

Wikipedia 的对机器学习定义如下:

“Machine learning is a subset of artificial intelligence in the field of computer science that often uses statistical techniques to give computers the ability to learn with data, without being explicitly programmed.”

机器学习的最重要本质是从数据中学习，得到预测函数。人类的思考过程以及判断能力本质上也是一种函数处理。从数据或者经验中学习，对于人类来说是一件再平常不过的事情了。例如人们通过观察太阳照射物体影子的长短而发明了日晷，从而具备了计时和制定节气的能力。古埃及人通过尼罗河水的涨落发明了古埃及历法。



又比如人们通过观察月亮形状的变化而发明了阴历。

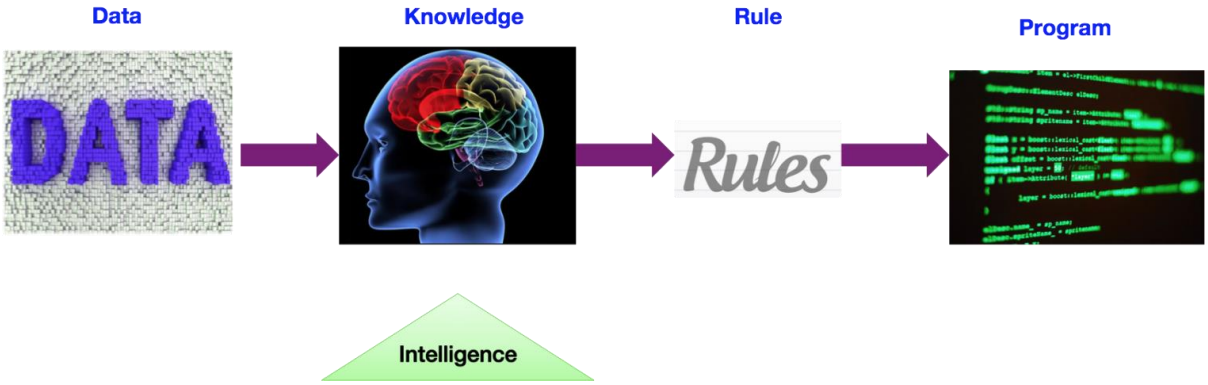
如果机器能够像人一样具备从数据中学习的能力，从某种意义上讲，就具备了一定的“智能”。现在需要回答的两个问题就是：

- 到底什么是“智能”？
- 如何让机器具备智能？

什么是智能？

在回答这个问题之前，我们先看看传统的编程模式为什么不能称之为“智能”。传统的编程模式如下图所示，它一般要经历如下几个阶段：

- 人类通过观察数据（Data）总结经验，转化成知识（Knowledge）。
- 人类将知识转化成规则（Rule）。
- 工程师将规则转化成计算机程序（Program）。



在这种编程模式下，如果一个问题被规则覆盖，那么计算机程序就能处理。对于规则不能覆盖的问题，只能由人类来重新思考，制定新规则来解决。所以在这里“智能”角色主要由人类来承担。人类负责解决新问题，所以传统的程序本身不能称之为“智能”。

所以，“智能”的一个核心要素就是“举一反三”。

如何让机器具备智能？

在讨论这个问题之前，可以先回顾一下人类是怎么掌握“举一反三”的能力的？基本流程如下：

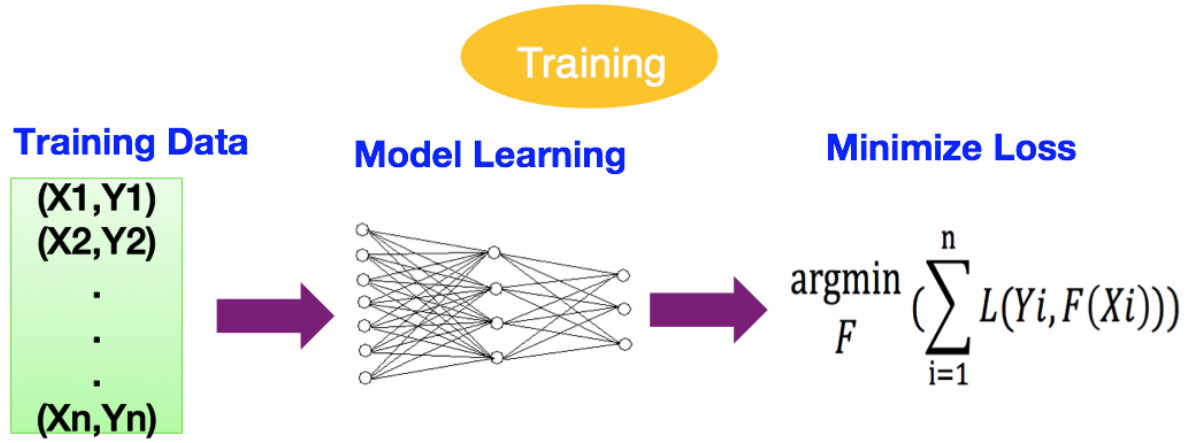
- 老师给学生一些题目，指导学生如何解题。学生努力掌握解题思路，尽可能让自己的答案和老师给出的答案一致。
- 学生需要通过一些考试来证明自己具备“举一反三”的能力。如果通过了这些考试，学生将获得毕业证书或者资格从业证书。
- 学生变成一个从业者之后将会面临并且处理很多之前没有碰到过的新问题。

机器学习专家从人类的学习过程中获得灵感，通过三个阶段让机器具备“举一反三”的能力。这三个阶段是：训练阶段（Training）、测试阶段（Testing）、推导阶段（Inference）。下面逐一进行介绍。

训练阶段

训练阶段如下图所示：

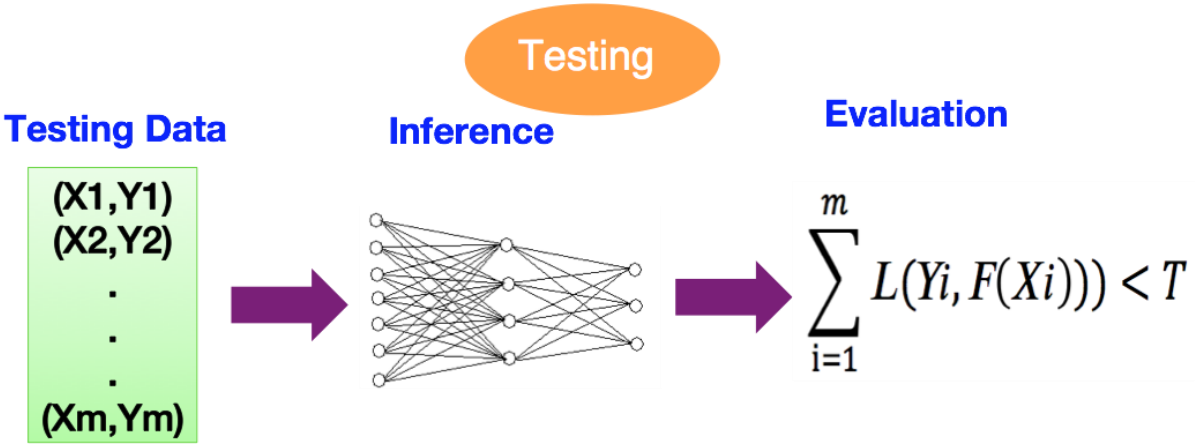
- 人类给机器学习模型一些训练样本（X，Y），X 代表特征，Y 代表目标值。这就好比老师教学生解题，X 代表题目，Y 代表标准答案。
- 机器学习模型尝试想出一种方法解题。
- 在训练阶段，机器学习的目标就是让损失函数值最小。类比学生尝试让自己解题的答案和老师给的标准答案差别最小，思路如出一辙。



测试阶段

测试阶段如下图所示：

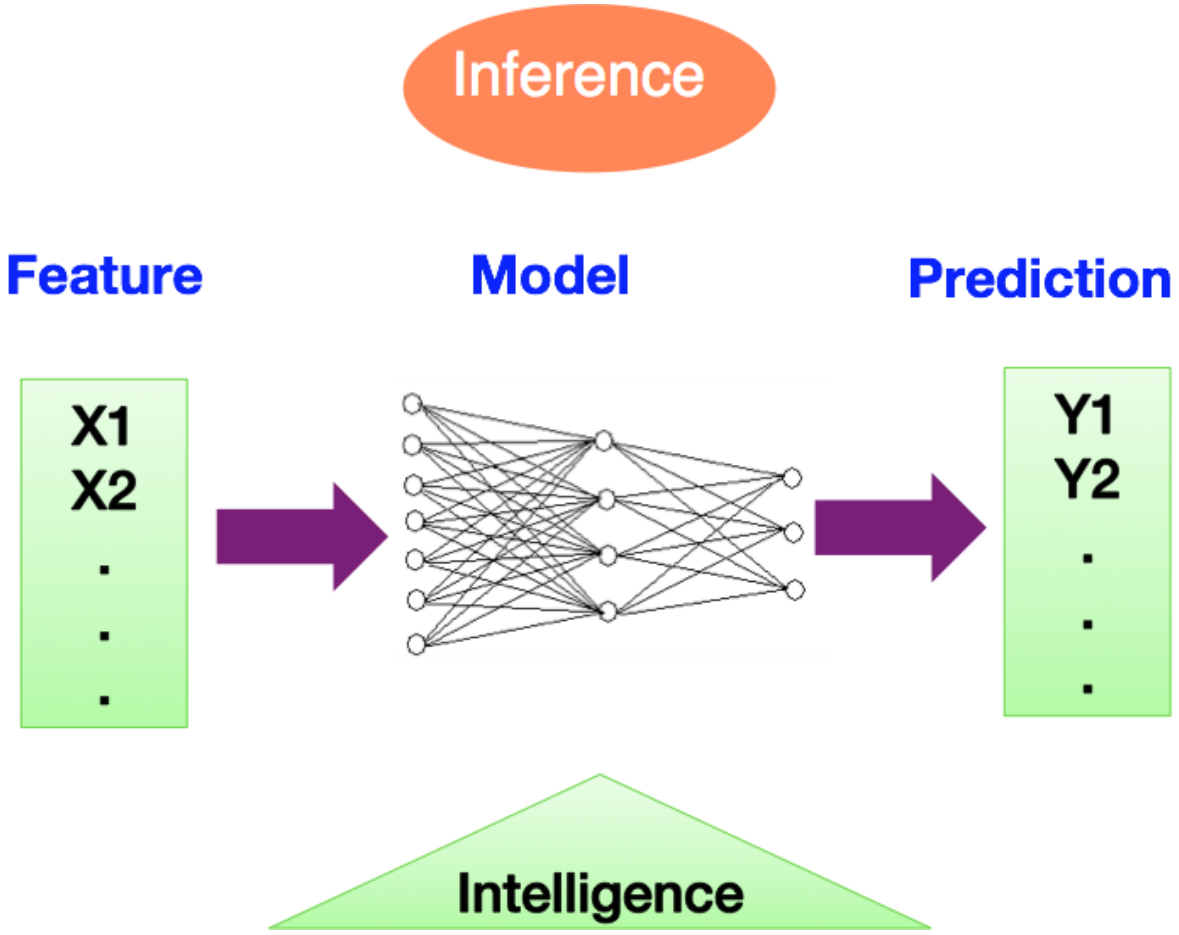
- 人类给训练好的模型一批完全不同的测试样本（X，Y）。这就好比学生拿到考试试卷。
- 模型进行推导。这个过程就像学生正在考试答题。
- 人类要求测试样本的总损失函数值低于设定的最低目标值。这就像学校要求学生的考试成绩必须及格一样。



推导阶段

推导阶段如下图所示：

- 在这个阶段机器学习模型只能拿到特征值 X，而没有目标值。这就像工作中，人们只是在解决一个个的问题，但不知道正确的结果到底是什么。
- 在推导阶段，机器学习的目标就是预测，给出目标值。



排序学习

什么是排序学习?

Wikipedia 的对排序学习的定义如下:

“Learning to rank is the application of machine learning, typically supervised, semi-supervised or reinforcement learning, in the construction of ranking models for information retrieval systems. Training data consists of lists of items with some partial order specified between items in each list. This order is typically induced by giving a numerical or ordinal score or a binary judgment (e.g. “relevant” or “not relevant”) for each item. The ranking model’s purpose is to rank, i.e. produce a permutation of items in new, unseen lists in a way which is “similar” to rankings in the training data in some sense.”

排序学习是机器学习在信息检索系统里的应用，其目标是构建一个排序模型用于对列表进行排序。排序学习的典型应用包括搜索列表、推荐列表和广告列表等等。

列表排序的目标是对多个条目进行排序，这就意味着它的目标值是有结构的。与单值回归和单值分类相比，结构化目标要求解决两个被广泛提起的概念:

- 列表评价指标

- 列表训练算法

列表评价指标

以关键词搜索返回文章列表为例，这里先分析一下列表评价指标要解决什么挑战。

- 第一个挑战就是定义文章与关键词之间的相关度，这决定了一篇文章在列表中的位置，相关度越高排序就应该越靠前。
- 第二个挑战是当列表中某些文章没有排在正确的位置时候，如何给整个列表打分。举个例子来说，假如对于某个关键词，按照相关性的高低正确排序，文档 1、2、3、4、5 应该依次排在前 5 位。现在的挑战就是，如何评估“2，1，3，4，5”和“1，2，5，4，3”这两个列表的优劣呢？

列表排序的评价指标体系总来的来说经历了三个阶段，分别是 Precision and Recall、Discounted Cumulative Gain(DCG)和 Expected Reciprocal Rank(ERR)。我们逐一进行讲解。

Precision and Recall(P-R)

本评价体系通过准确率（Precision）和召回率（Recall）两个指标来共同衡量列表的排序质量。对于一个请求关键词，所有的文档被标记为相关和不相关两种。

Precision 的定义如下：

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

Recall 的定义如下：

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

举个列子来说，对于某个请求关键词，有 200 篇文章实际相关。某个排序算法只认为 100 篇文章是相关的，而这 100 篇文章里面，真正相关的文章只有 80 篇。按照以上定义：

- 准确率=80 / 100=0.8
- 召回率=80 / 200=0.4。

Discounted Cumulative Gain(DCG)

P-R 的有两个明显缺点：

- 所有文章只被分为相关和不相关两档，分类显然太粗糙。
- 没有考虑位置因素。

DCG 解决了这两个问题。对于一个关键词，所有的文档可以分为多个相关性级别，这里以 rel1，rel2…来表示。文章相关性对整个列表评价指标的贡献随着位置的增加而对数衰减，位置越靠后，衰减越严重。基于 DCG 评价指标，列表前 p 个文档的评价指标定义如下：

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

对于排序引擎而言，不同请求的结果列表长度往往不相同。当比较不同排序引擎的综合排序性能时，不同长度请求之间的 DCG 指标的可比性不高。目前在工业界常用的是 Normalized DCG(nDCG)，它假定能够获取到某个请求的前 p 个位置的完美排序列表，这个完美列表的分值称为 Ideal DCG(IDCG)，nDCG 等于 DCG 与 IDCG 比值。所以 nDCG 是一个在 0 到 1 之间的值。

nDCG 的定义如下：

$$\mathrm{nDCG_p} = \frac{DCG_p}{IDCG_p}$$

IDCG 的定义如下：

$$\mathrm{IDCG_p} = \sum_{i=1}^{|REL|} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

|REL|代表按照相关性排序好的最多到位置 p 的结果列表。

Expected Reciprocal Rank(ERR)

与 DCG 相比，除了考虑位置衰减和允许多种相关级别（以 R1，R2，R3…来表示）以外，ERR 更进了一步，还考虑了排在文档之前所有文档的相关性。举个例子来说，文档 A 非常相关，排在第 5 位。如果排在前面的 4 个文档相关度都不高，那么文档 A 对列表的贡献就很大。反过来，如果前面 4 个文档相关度很大，已经完全解决了用户的搜索需求，用户根本就不会点击第 5 个位置的文档，那么文档 A 对列表的贡献就不大。

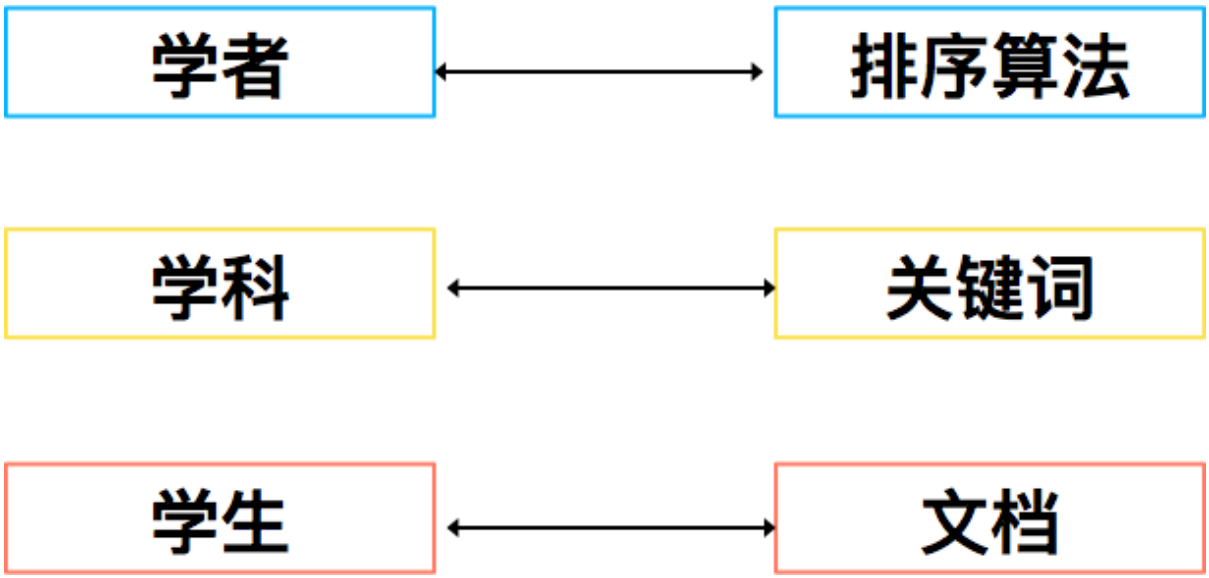
ERR 的定义如下：

$$ERR = \sum_{r=1}^{n} \frac{1}{r} \prod_{i=1}^{r-1} (1-R_i) R_r$$

列表训练算法

做列表排序的工程师们经常听到诸如 Pointwise、Pairwise 和 Listwise 的概念。这些是什么东西呢，背后的原理又是什么呢？这里将逐一解密。

仍然以关键词搜索文章为例，排序学习算法的目标是为给定的关键词对文章列表进行排序。做为类比，假定有一个学者想要对各科学生排名进行预测。各种角色的对应关系如下：



首先我们要告诉学者每个学生的各种属性，这就像我们要告诉排序算法文档特征。对于目标值，我们却有三种方式来告诉学者：

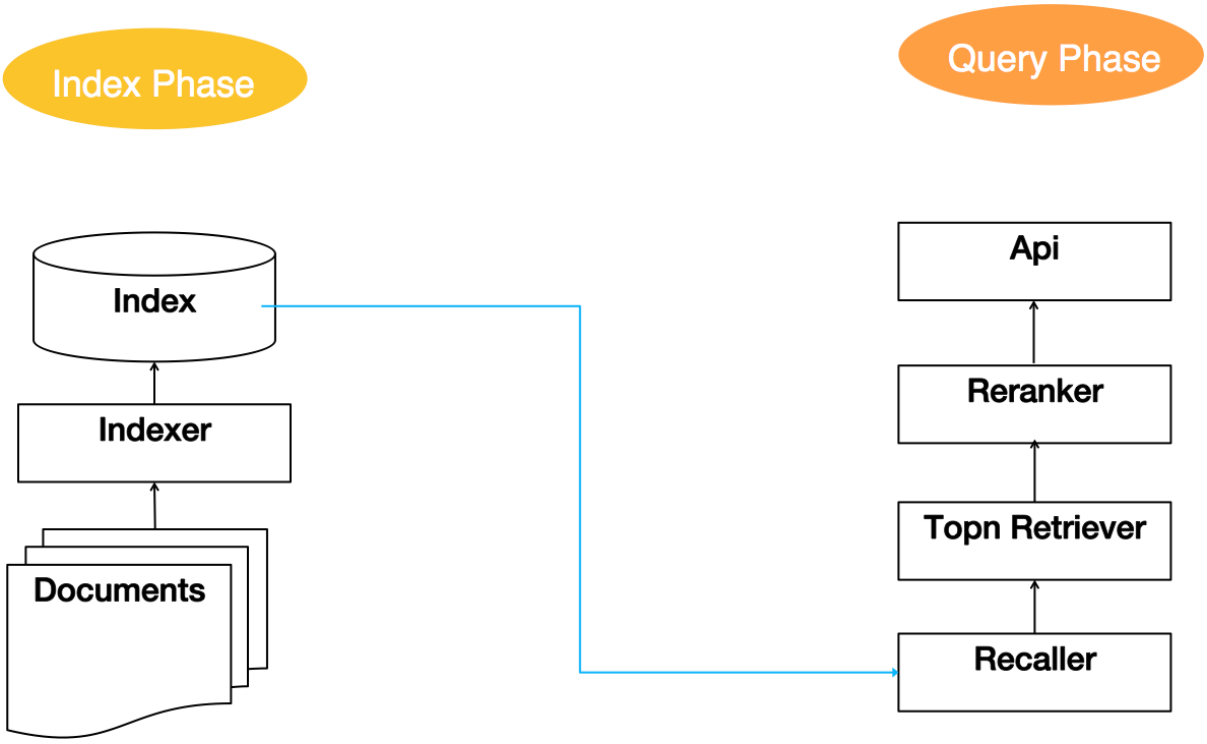
- 对于每个学科，我们可以告诉学者每个学生的成绩。比较每个学生的成绩，学者当然可以算出每个学生的最终排名。这种训练方法被称为 Pointwise。对于 Pointwise 算法，如果最终预测目标是一个实数值，就是一个回归问题。如果目标是概率预测，这就是一个分类问题，例如 CTR 预估。
- 对于每个学科，我们可以告诉学者任意两个学生的相互排名。根据学生之间排名的情况，学者也可以算出每个学生的最终排名。这种训练方法被称为 Pairwise。Pairwise 算法的目标是减少逆序的数量，所以是个二分类问题。
- 对于每个学科，我们可以直接告诉学者所有学生的整体排名。这种训练方法被称为 Listwise。Listwise 算法的目标往往是直接优化 nDCG、ERR 等评价指标。

这三种方法表面看上去有点像玩文字游戏，但是背后却是工程师和科学家们不断探索的结果。最直观的方案是 Pointwise 算法，例如对于广告 CTR 预估，在训练阶段需要标注某个文档的点击概率，这相对来说容易。Pairwise 算法一个重要分支是 Lambda 系列，包括 LambdaRank、

LambdaMart 等，它的核心思想是：很多时候我们很难直接计算损失函数的值，但却很容易计算损失函数梯度（Gradient）。这意味着我们很难计算整个列表的 nDCG 和 ERR 等指标，但却很容易知道某个文档应该排的更靠前还是靠后。Listwise 算法往往效果最好，但是如何为每个请求对所有文档进行标注是一个巨大的挑战。

在线排序架构

典型的信息检索包含两个阶段：索引阶段和查询阶段。这两个阶段的流程以及相互关系可以用下图来表示：



索引阶段的工作是由索引器（Indexer）读取文档（Documents）构建索引（Index）。

查询阶段读取索引做为召回，然后交给 Topn Retriever 进行粗排，在粗排后的结果里面将前 n 个文档传给 Reranker 进行精排。这样一个召回、粗排、精排的架构最初是由 Google 提出来的，也被称为“Two-Phase Scheme”。

索引部分属于离线阶段，这里重点讲述在线排序阶段，即查询阶段。

三大挑战

在线排序架构主要面临三方面的挑战：特征、模型和召回。

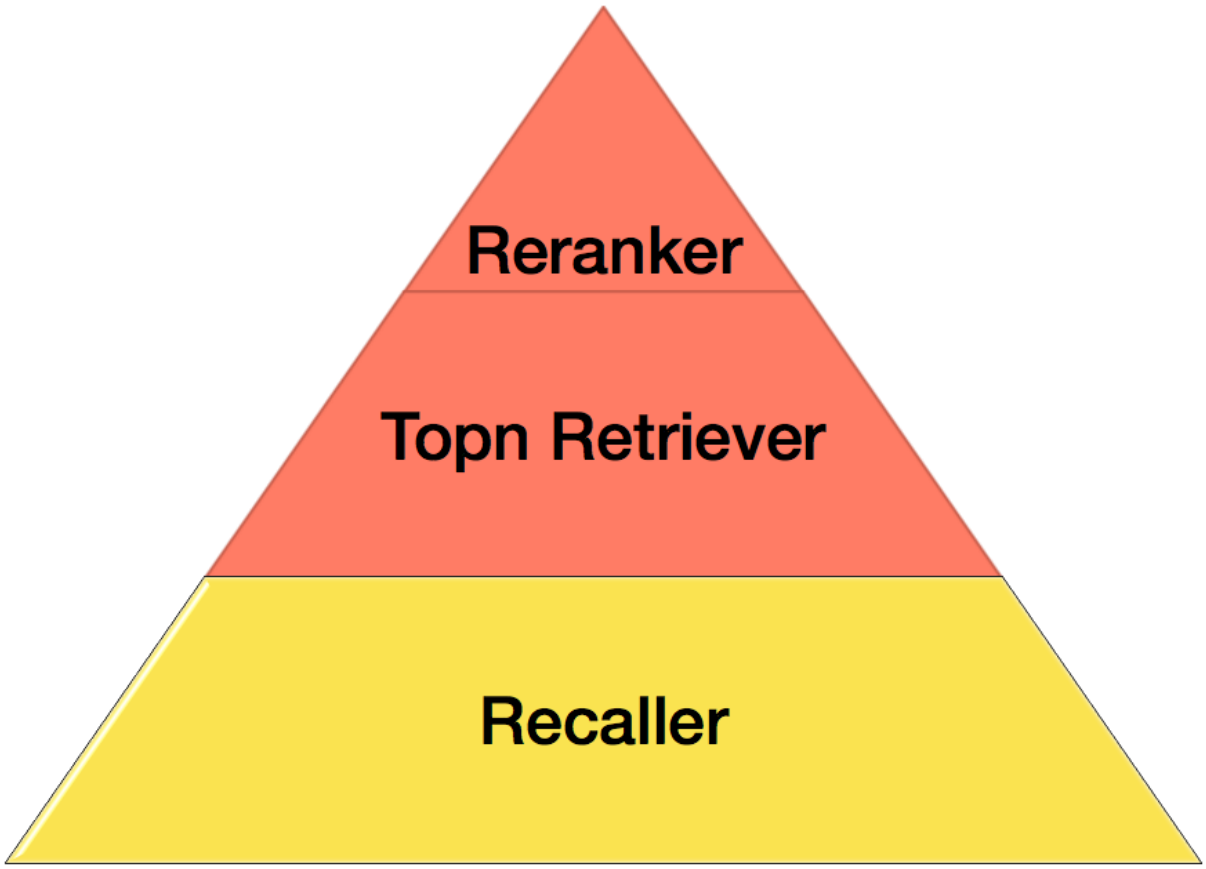
- 特征挑战包括特征添加、特征算子、特征归一化、特征离散化、特征获取、特征服务治理等。
- 模型挑战包括基础模型完备性、级联模型、复合目标、A/B 实验支持、模型热加载等。
- 召回挑战包括关键词召回、LBS 召回、推荐召回、粗排召回等。

三大挑战内部包含了非常多更细粒度的挑战，孤立地解决每个挑战显然不是好思路。在线排序作为一个被广泛使用的架构值得采用领域模型进行统一解决。Domain-driven design(DDD)的三个原则分别是：领域聚焦、边界清晰、持续集成。

基于以上分析，我们构建了三个在线排序领域模型：召回治理、特征服务治理和在线排序分层模型。

召回治理

经典的 Two-Phase Scheme 架构如下图所示，查询阶段应该包含：召回、粗排和精排。但从领域架构设计的角度来讲，粗排对于精排而言也是一种召回。和基于传统的文本搜索不同，美团点评这样的 O2O 公司需要考虑地理位置和距离等因素，所以基于 LBS 的召回也是一种召回。与搜索不同，推荐召回往往基于协同过滤来完成的，例如 User-Based CF 和 Item-Based CF。

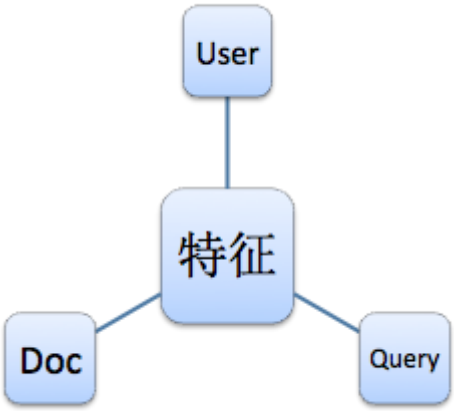


综上所述，召回总体而言分成四大类：

- 关键词召回，我们采用 Elasticsearch 解决方案。
- 距离召回，我们采用 K-D tree 的解决方案。
- 粗排召回。
- 推荐类召回。

特征服务治理

传统的视角认为特征服务应该分为用户特征（User）、查询特征（Query）和文档特征（Doc），如下图：

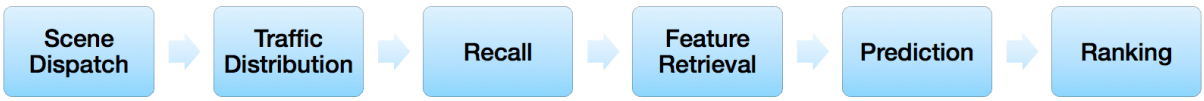


这是比较纯粹的业务视角，并不满足 DDD 的领域架构设计思路。由于特征数量巨大，我们没有办法为每个特征设计一套方案，但是我们可以把特征进行归类，为几类特征分别设计解决方案。每类技术方案需要统一考虑性能、可用性、存储等因素。从领域视角，特征服务包含四大类：

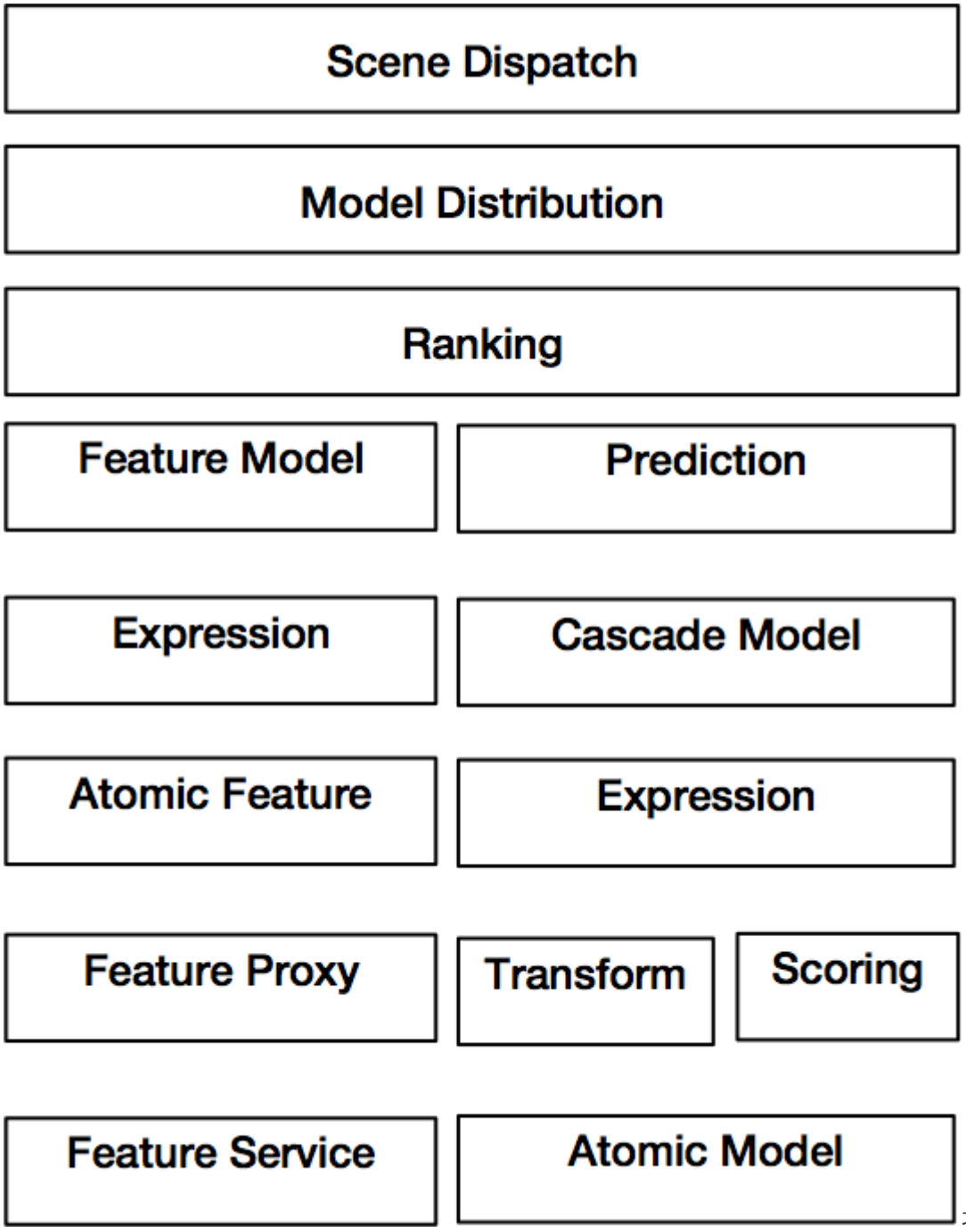
- 列表类特征。一次请求要求返回实体列表特征，即多个实体，每个实体多个特征。这种特征建议采用内存列表服务，一次性返回所有请求实体的所有特征。避免多次请求，从而导致数量暴增，造成系统雪崩。
- 实体特征。一次请求返回单实体的多个特征。建议采用 Redis、Tair 等支持多级的 Key-Value 服务中间件提供服务。
- 上下文特征。包括召回静态分、城市、Query 特征等。这些特征直接放在请求内存里面。
- 相似性特征。有些特征是通过计算个体和列表、列表和列表的相似度而得到的。建议提供单独的内存计算服务，避免这类特征的计算影响在线排序性能。本质上这是一种计算转移的设计。

在线排序分层模型

如下图所示，典型的排序流程包含六个步骤：场景分发（Scene Dispatch）、流量分配（Traffic Distribution）、召回（Recall）、特征抽取（Feature Retrieval）、预测（Prediction）、排序（Ranking）等等。

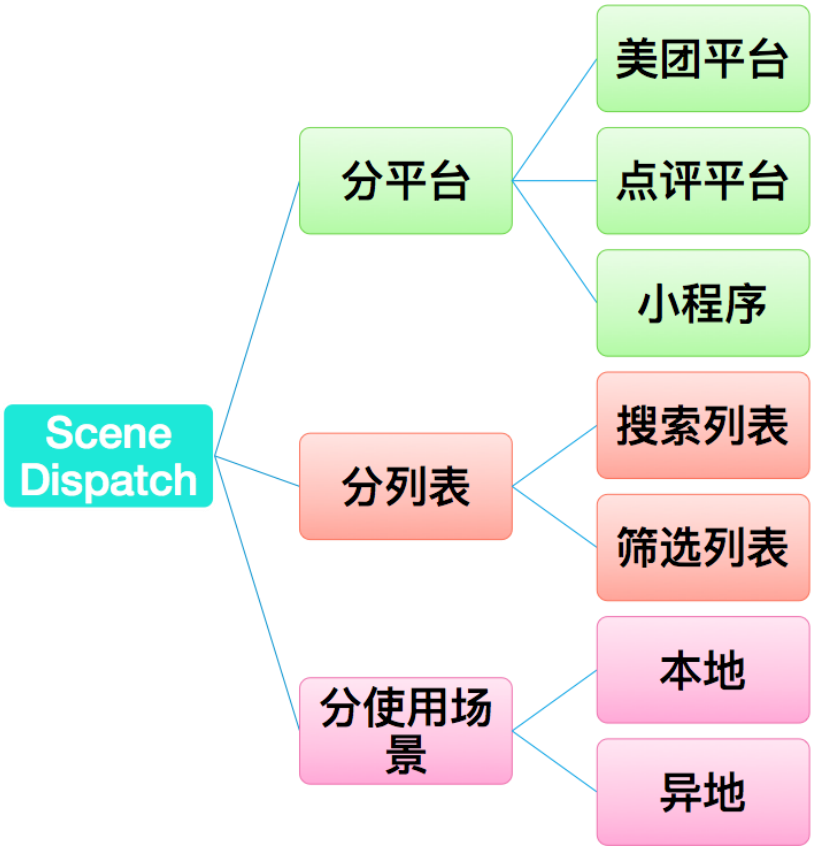


按照 DDD 的设计原则，我们设计了如下在线排序分层模型，包括：场景分发（Scene Dispatch）、模型分发（Model Distribution）、排序（Ranking）、特征管道（Feature Pipeline）、预测管道（Prediction Pipeline）。我们将逐一进行介绍。



场景分发（Scene Dispatch）

场景分发一般是指业务类型的分发。对于美团点评而言包括：分平台、分列表、分使用场景等。如下图所示：



模型分发（Model Distribution）

模型分发的目标是把在线流量分配给不同的实验模型，具体而言要实现三个功能：

- 为模型迭代提供在线流量，负责线上效果收集、验证等。
- A/B 测试，确保不同模型之间流量的稳定、独立和互斥、确保效果归属唯一。
- 确保与其他层的实验流量的正交性。

流量的定义是模型分发的一个基础问题。典型的流量包括：访问、用户和设备。

如何让一个流量稳定地映射到特定模型上面，流量之间是否有级别？这些是模型分发需要重点解决的问题。

流量分桶原理

采用如下步骤将流量分配到具体模型上面去：

- 把所有流量分成 N 个桶。
- 每个具体的流量 Hash 到某个桶里面去。
- 给每个模型一定的配额，也就是每个策略模型占据对应比例的流量桶。
- 所有策略模型流量配额总和为 100%。
- 当流量和模型落到同一个桶的时候，该模型拥有该流量。

举个例子来说，如上图所示，所有流量分为 32 个桶，A、B、C 三个模型分别拥有 37.5%、25%和 37.5%的配额。对应的，A、B、C 应该占据 12、8 和 12 个桶。

为了确保模型和流量的正交性，模型和流量的 Hash Key 采用不同的前缀。

流量分级

每个团队的模型分级策略并不相同，这里只给出一个建议模型流量分级：

- 基线流量。本流量用于与其他流量进行对比，以确定新模型的效果是否高于基准线，低于基准线的模型要快速下线。另外，主力流量相对基线流量的效果提升也是衡量算法团队贡献的重要指标。
- 实验流量。该流量主要用于新实验模型。该流量大小设计要注意两点：第一不能太大而伤害线上效果；第二不能太小，流量太小会导致方差太大，不利于做正确的效果判断。
- 潜力流量。如果实验流量在一定周期内效果比较好，可以升级到潜力流量。潜力流量主要是要解决实验流量方差大带来的问题。
- 主力流量。主力流量只有一个，即稳定运行效果最好的流量。如果某个潜力流量长期好于其他潜力流量和主力流量，就可以考虑把这个潜力流量升级为主力流量。

做实验的过程中，需要避免新实验流量对老模型流量的冲击。流量群体对于新模型会有一定的适应期，而适应期相对于稳定期的效果一般会差一点。如果因为新实验的上线而导致整个流量群体的模型都更改了，从统计学的角度讲，模型之间的对比关系没有变化。但这可能会影响整个大盘的效果，成本很高。

为了解决这个问题，我们的流量分桶模型优先为模型列表前面的模型分配流量，实验模型尽量放在列表尾端。这样实验模型的频繁上下线不影响主力和潜力流量的用户群体。当然当发生模型流量升级的时候，很多流量用户的服务模型都会更改。这种情况并不是问题，因为一方面我们在尝试让更多用户使用更好的模型，另一方面固定让一部分用户长期使用实验流量也是不公平的事情。

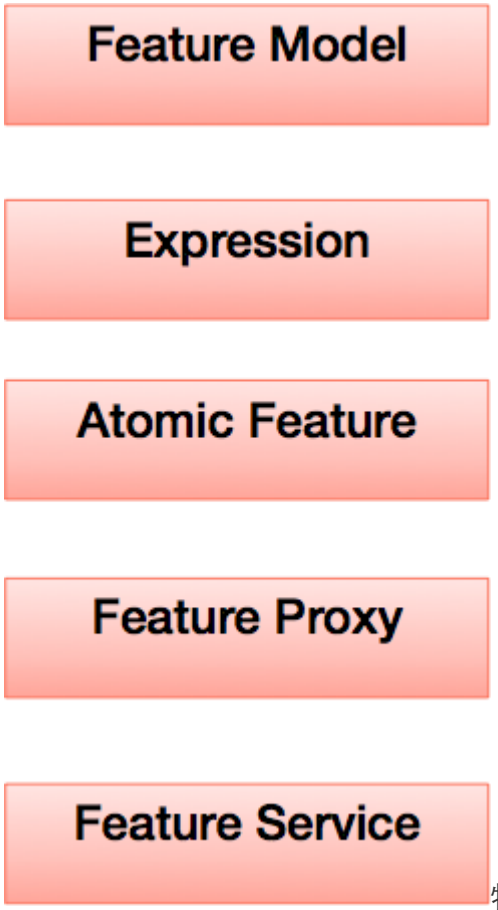
排序（Ranking）

排序模块是特征模块和预测模块的容器，它的主要职责如下：

- 获取所有列表实体进行预测所需特征。
- 将特征交给预测模块进行预测。
- 对所有列表实体按照预测值进行排序。

特征管道（Feature Pipeline）

特征管道包含特征模型（Feature Model）、表达式（Expression）、原子特征（Atomic Feature）、 特征服务代理（Feature Proxy）、特征服务（Feature Service）。如下图所示：



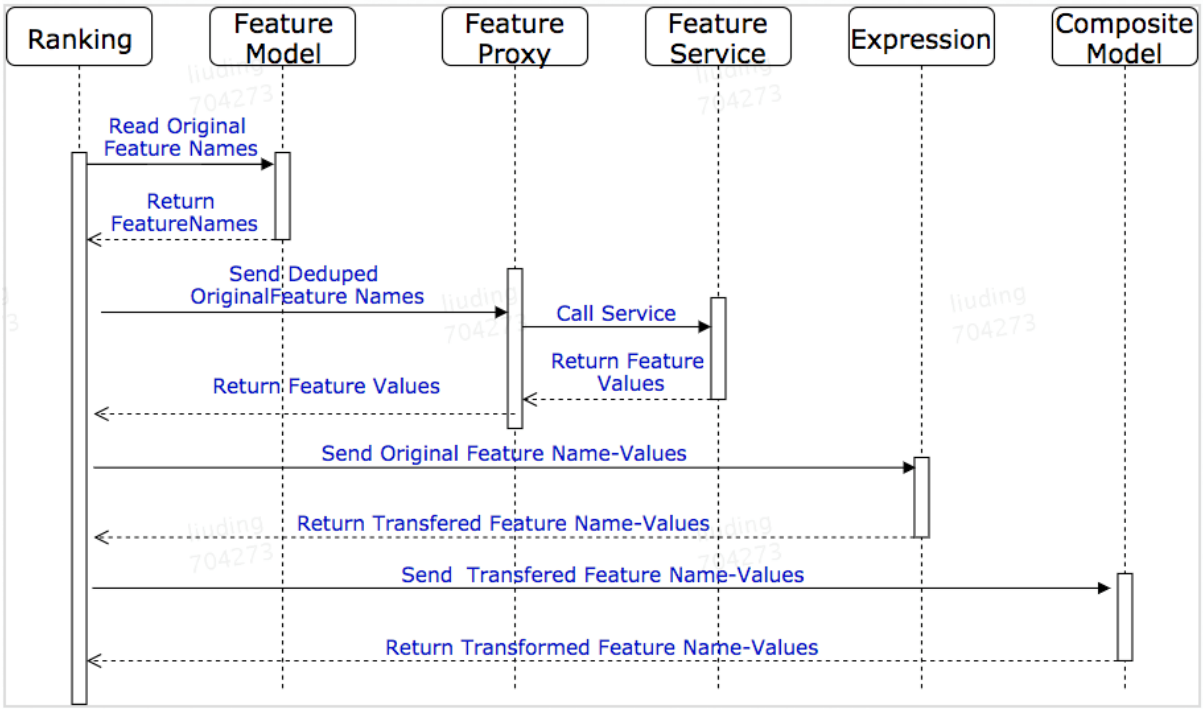
特征管道要解决两个核心问题：

- 给定特征名获取对应特征值。这个过程很复杂，要完成特征名->特征服务->特征类->特征值的转化过程。
- 特征算子问题。模型使用的特征往往是对多个原子特征进行复合运算后的结果。另外，特征离散化、归一化也是特征算子需要解决的问题。

完整的特征获取流程如下图所示，具体的流程如下：

- Ranking 模块从 FeatureModel 里面读取所有的原始特征名。
- Ranking 将所有的原始特征名交给 Feature Proxy。
- Feature Proxy 根据特征名的标识去调用对应的 Feature Service，并将原始特征值返回给 Ranking 模块。

- Ranking 模块通过 Expression 将原始特征转化成复合特征。
- Ranking 模块将所有特征交给级联模型做进一步的转换。



特征模型（Feature Model）

我们把所有与特征获取和特征算子的相关信息都放在一个类里面，这个类就是 FeatureModel，定义如下：

```
//包含特征获取和特征算子计算所需的 meta 信息
public class FeatureModel {
    //这是真正用在 Prediction 里面的特征名
    private String featureName;
    //通过表达式将多种原子特征组合成复合特征。
    private IExpression expression;
    //这些特征名是真正交给特征服务代理(Feature Proxy)去从服务端获取特征值的特征名集合。
    private Set<String> originalFeatureNames;
```

```
        //用于指示特征是否需要被级联模型转换
        private boolean isTransformedFeature;
        //是否为 one-hot 特征
        private boolean isOneHotIdFeature;
        //不同 one-hot 特征之间往往共享相同的原始特征，这个变量>用于标识原始特征名。
        private String oneHotIdKey;
        //表明本特征是否需要归一化
        private boolean isNormalized;
    }
```

表达式（Expression）

表达式的目的是为了将多种原始特征转换成一个新特征，或者对单个原始特征进行运算符转换。我们采用前缀法表达式（Polish Notation）来表示特征算子运算。例如表达式(5-6)*7 的前缀表达式为* - 5 6 7。

复合特征需要指定如下分隔符：

- 复合特征前缀。区别于其他类型特征，我们以“$”表示该特征为复合特征。
- 表达式各元素之间的分隔符，采用“_”来标识。
- 用“O”表示运算符前缀。
- 用“C”表示常数前缀。
- 用“V”表示变量前缀。

例如：表达式 v1 + 14.2 + (2*(v2+v3)) 将被表示为 $O+_O+_Vv1_C14.2_O*_C2_O+_Vv2_Vv3

原子特征（Atomic Feature）

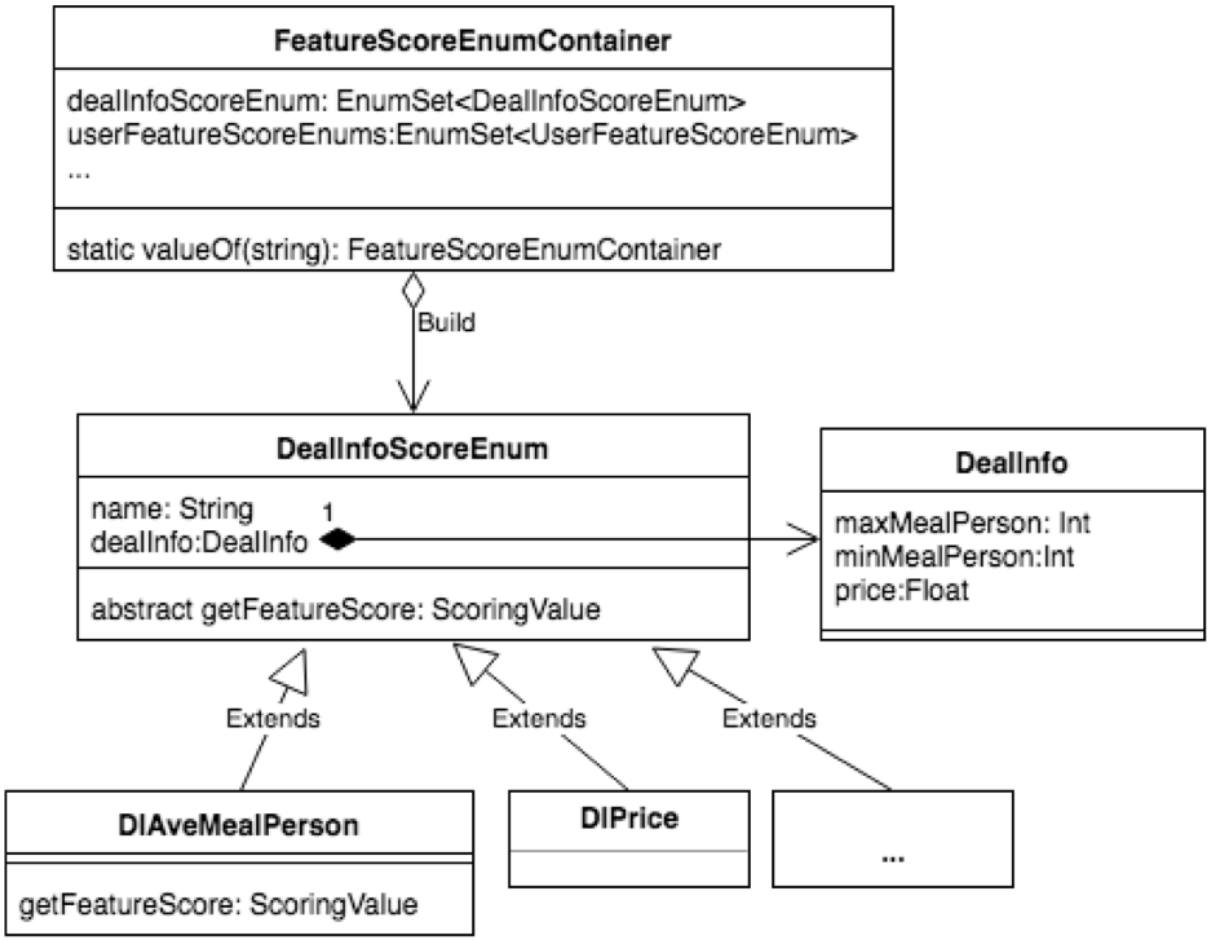
原子特征（或者说原始特征）包含特征名和特征值两个部分。原子特征的读取需要由 4 种实体类共同完成：

- POJO 用于存储特征原始值。例如 DealInfo 保存了所有与 Deal 实体相关的特征值，包括 Price、maxMealPerson、minMealPerson 等等。

- ScoringValue 用于存储从 POJO 中返回的特征值。特征值包含三种基本类型，即数量型（Quantity）、序数型（Ordinal）、类别型（Categorical）。
- ScoreEnum 实现特征名到特征值的映射。每类原子特征对应于一个 ScoreEnum 类，特征名通过反射（Reflection）的方式来构建对应的 ScoreEnum 类。ScoreEnum 类和 POJO 一起共同实现特征值的读取。
- FeatureScoreEnumContainer 用于保存一个算法模型所需所有特征的 ScoreEnum。

一个典型的例子如下图所示：

- DealInfo 是 POJO 类。
- DealInfoScoreEnum 是一个 ScoreEnum 基类。对应于平均用餐人数特征、价格等特征，我们定义了 DIAveMealPerson 和 DIPrice 的具体 ScoreEnum 类。
- FeatureScoreEnumContainer 用于存储某个模型的所有特征的 ScoreEnum。



复杂系统设计需要充分的利用语言特性和设计模式。建议的优化点有三个：

- 为每个原子特征定义一个 ScoreEnum 类会导致类数量暴增。优化方式是 ScoreEnum 基类定义为 Enum 类型，每个具体特征类为一个枚举值，枚举值继承并实现枚举类的方法。
- FeatureScoreEnumContainer 采用 Build 设计模式将一个算法模型的所需特征转换成 ScoreEnum 集合。

- ScoreEnum 从 POJO 类中读取具体特征值采用的是 Command 模式。

这里稍微介绍一下 Command 设计模式。Command 模式的核心思想是需求方只要求拿到相关信息，不关心谁提供以及怎么提供。具体的提供方接受需求方的需求，负责将结果交给需求方。

在特征读取里面，需求方是模型，模型仅仅提供一个特征名（FeatureName），不关心怎么读取对应的特征值。具体的 ScoreEnum 类是具体的提供方，具体的 ScoreEnum 从 POJO 里面读取特定的特征值，并转换成 ScoringValue 交给模型。

特征服务代理（Feature Proxy）

特征服务代理负责远程特征获取实施，具体的过程包括:

- 每一大类特征或者一种特征服务有一个 FeatureProxy，具体的 FeatureProxy 负责向特征服务发起请求并获取 POJO 类。
- 所有的 FeatureProxy 注册到 FeatureServiceContainer 类。
- 在具体的一次特征获取中，FeatureServiceContainer 根据 FeatureName 的前缀负责将特征获取分配到不同的 FeatureProxy 类里面。
- FeatureProxy 根据指定的实体 ID 列表和特征名从特征服务中读取 POJO 列表。只有对应 ID 的指定特征名（keys）的特征值才会被赋值给 POJO。这就最大限度地降低了网络读取的成本。

预测管道（Prediction Pipeline）

预测管道包含：预测（Prediction）、级联模型（Cascade Model）、表达式（Expression）、特征转换（Transform）、计分（Scoring）和原子模型（Atomic Model）。

预测（Prediction）

预测本质上是对模型的封装。它负责将每个列表实体的特征转化成模型需要的输入格式，让模型进行预测。

级联模型（Cascade Model）

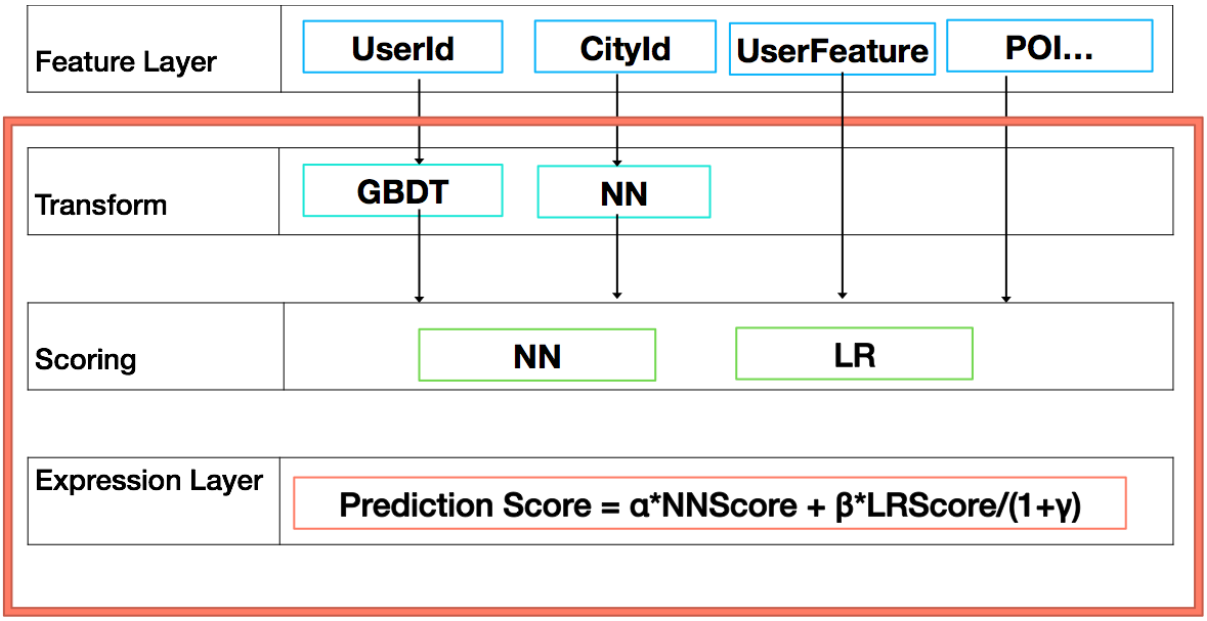
我们构建级联模型主要是基于两方面的观察:

- 基于 Facebook 的《Practical Lessons from Predicting Clicks on Ads at Facebook》的 Xgboost+LR 以及最近很热门的 Wide&Deep 表明，对一些特征，特别是 ID 类特征通过树模型或者 NN 模型进行转化，把转化后的值做为特征值交给预测模型进行预测，往往会能实现更好的效果。

- 有些训练目标是复合目标，每个子目标需要通过不同的模型进行预测。这些子目标之间通过一个简单的表达式计算出最终的预测结果。

举例如下图所示，我们自上而下进行讲解：

- 该模型有 UserId、CityId、UserFeature、POI 等特征。
- UserId 和 CityId 特征分别通过 GBDT 和 NN 模型进行转换（Transform）。
- 转换后的特征和 UserFeature、POI 等原始特征一起交给 NN 和 LR 模型进行算分（Scoring）。
- 最终的预测分值通过表达式 Prediction Score = α*NNScore + β*LRScore/(1+γ)来完成。表达式中的 α 、β 和 γ 是事先设置好的值。



原子模型（Atomic Model）

在这里原子模型指的是一种原子计算拓扑结构，比如线性模型、树模型和网络模型。

常用的模型像 Logistic Regression 和 Linear Regression 都是线性模型。GBDT、Random Forest 都是树模型。MLP、CNN、RNN 都是网络模型。

这里定义的原子模型主要的目的是为了工程实施的便利。一个模型被认定为原子模型有如下两个原因：

- 该模型经常做为独立预测模型被使用。
- 该模型有比较完整的实现代码。

总结

本文总结了作者在美团点评解决到店餐饮个性化信息展示的相关经验，从算法和架构两方面进行阐述。在算法部分，文章采用通俗的例子和类比方式进行讲解，希望非算法工程师能够理解

关键的算法概念。架构部分比较详细地阐述了到店餐饮的排序架构。

根据我们所掌握的知识，特征治理和召回治理的思路是一种全新的视角，这对于架构排序系统设计有很大的帮助。这种思考方式同样也适用于其他领域模型的构建。与 Google 提供的经典 Two-Phase Scheme 架构相比，在线排序分层模型提供了更细颗粒度的抽象原型。该原型细致的阐述了包括分流、A/B 测试、特征获取、特征算子、级联模型等一系列经典排序架构问题。同时该原型模型由于采用了分层和层内功能聚焦的思路，所以它比较完美地体现了 DDD 的三大设计原则，即领域聚焦、边界清晰、持续集成。

作者简介

- 刘丁，目前负责到店餐饮算法策略方向，推进 AI 在到店餐饮各领域的应用。2014 年加入美团，先后负责美团推荐系统、智能筛选系统架构、美团广告系统的架构和上线、完成美团广告运营平台的搭建。曾就职于 Amazon、TripAdvisor 等公司。

参考文章:

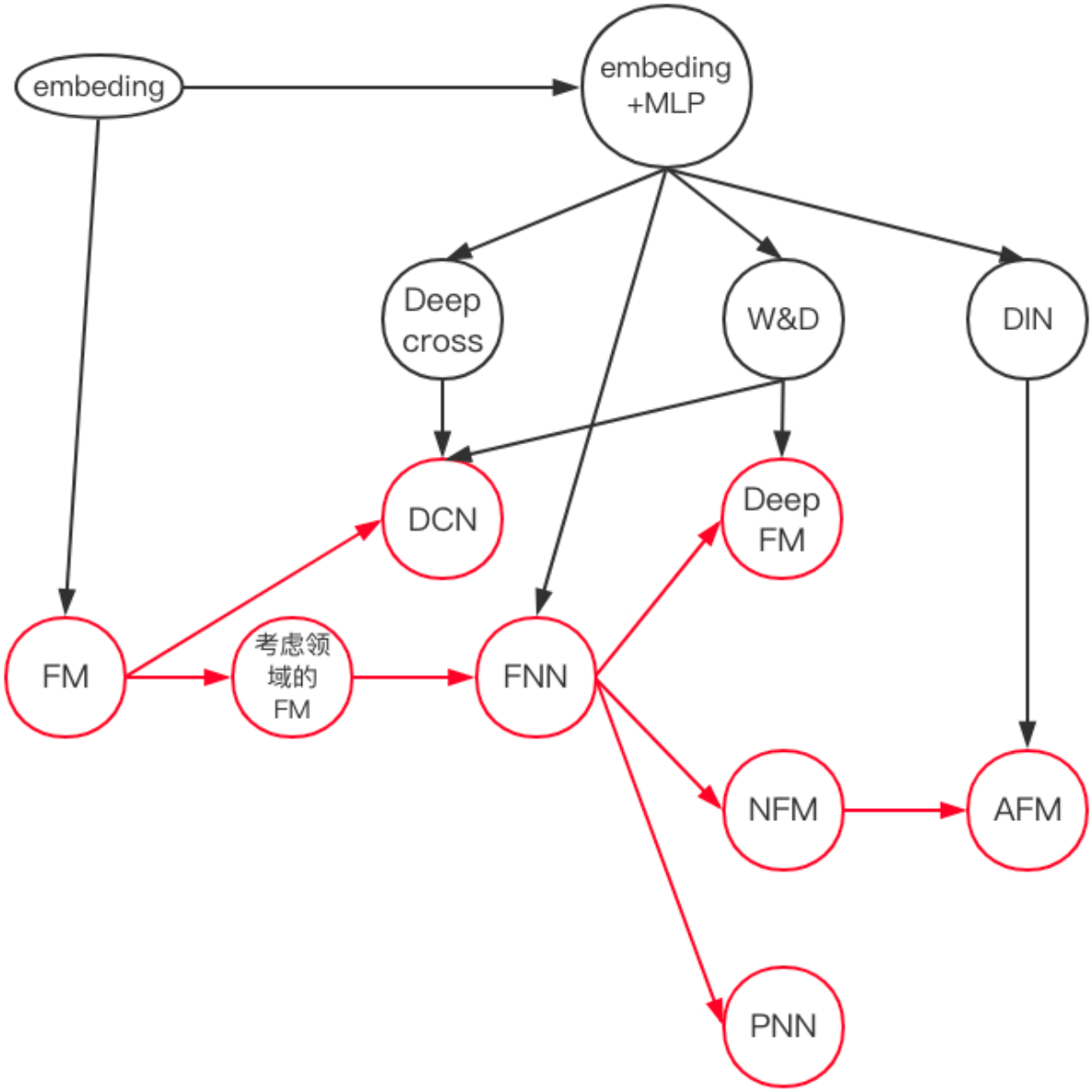
- [1]Gamma E, Helm R, Johnson R, et al. Design Patterns-Elements of Reusable Object-Oriented Software. Machinery Industry, 2003
- [2]Wikipedia,.
- [3]Wikipedia,.
- [4]Wikipedia,.
- [5]Wikipedia,.
- [6]Wikipedia,.
- [7]Wikipedia,.
- [8]Wikipedia,.
- [9]百度百科,.
- [10]百度百科,.
- [11]Xinran H, Junfeng P, Ou J, et al.
- [12]Olivier C, Donald M, Ya Z, Pierre G.
- [13]Heng-Tze C, Levent K, et al.

12、从 FM 推演各深度学习 CTR 预估模型

解析:

作者： 龙心尘 && 寒小阳
时间：2018 年 7 月。
出处：https://blog.csdn.net/longxinchen_ml/article/details/81031736
https://blog.csdn.net/han_xiaoyang/article/details/81031961

注：本文根据作者在公司内训讲稿整理而成。
多年以后，当资深算法专家们看着无缝对接用户需求的广告收入节节攀升时，他们可能会想起自己之前痛苦推导 FM 与深度学习公式的某个夜晚……——题记
本文的 PDF 版本、代码实现和数据可以在我的 github 取到。
一、引言
点击率(click-through rate, CTR)是互联网公司进行流量分配的核心依据之一。比如互联网广告平台，为了精细化权衡和保障用户、广告、平台三方的利益，准确的 CTR 预估是不可或缺的。CTR 预估技术从传统的逻辑回归，到近两年大火的深度学习，新的算法层出不穷：DeepFM, NFM, DIN, AFM, DCN……
然而，相关的综述文章不少，但碎片罗列的居多，模型之间内在的联系和演化思路如何揭示？怎样才能迅速 get 到新模型的创新点和适用场景，快速提高新论文速度，节约理解、复现模型的成本？这些都是亟待解决的问题。
我们认为，从 FM 及其与神经网络的结合出发，能够迅速贯穿很多深度学习 CTR 预估网络的思路，从而更好地理解和应用模型。
本文的思路与方法
我们试图从原理上进行推导、理解各个深度 CTR 预估模型之间的相互关系，知其然也知其所以然。（以下的分析与拆解角度，是一种我们尝试的理解视角，并不是唯一的理解方式）
推演的核心思路：“通过设计网络结构进行组合特征的挖掘”。
具体来说有两条：其一是从 FM 开始推演其在深度学习上的各种推广（对应下图的红线），另一条是从 embedding+MLP 自身的演进特点结合 CTR 预估本身的业务场景进行推演（对应下图黑线部分）。



为了便于理解，我们简化了数据案例——只考虑离散特征数据的建模，以分析不同神经网络在处理相同业务问题时的不同思路。

同时，我们将各典型论文不同风格的神经网络结构图统一按照计算图来绘制，以便于对比不同模型。

二、FM：降维版本的特征二阶组合

CTR 预估本质是一个二分类问题，以移动端展示广告推荐为例，依据日志中的用户侧的信息（比如年龄，性别，国籍，手机上安装的 app 列表）、广告侧的信息（广告 id，广告类别，广告标题等）、上下文侧信息（渠道 id 等），去建模预测用户是否会点击该广告。

FM 出现之前的传统的处理方法是人工特征工程加上线性模型（如逻辑回归 Logistic Regression）。为了提高模型效果，关键技术是找到到用户点击行为背后隐含的特征组合。如男性、大学生用户往往会点击游戏类广告，因此“男性且是大学生且是游戏类”的特征组合就是一个关键特征。但这本质仍是线性模型，其假设函数表示成内积形式一般为：

$$y_{linear} = \sigma\left(\langle \vec{w}, \vec{x} \rangle\right)$$

其中

$\vec{x}$ 为特征向量，
$\vec{w}$ 为权重向量，σ()为 sigmoid 函数。
但是人工进行特征组合通常会存在诸多困难，如特征爆炸、特征难以被识别、组合特征难以设计等。
为了让模型自动地考虑特征之间的二阶组合信息，线性模型推广为二阶多项式（2d−Polynomial 2d-Polynomial2d−Polynomial）模型：

$$y_{poly} = \sigma\left(\langle \vec{w}, \vec{x} \rangle + \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \cdot x_i \cdot x_j\right)$$

其实就是对特征两两相乘（组合）构成新特征(离散化之后其实就是“且”操作)，并对每个新特征分配独立的权重，通过机器学习来自动得到这些权重。将其写成矩阵形式为：

$$y_{poly} = \sigma\left(\vec{w}^T \cdot \vec{x} + \vec{x}^T \cdot W^{(2)} \cdot \vec{x}\right)$$

其中 $W^{(2)}$ 为二阶特征组合的权重矩阵，是对称矩阵。而这个矩阵参数非常多，为 $O(n^2)$。为了降低该矩阵的维度，可以将其因子分解（Factorization FactorizationFactorization）为两个低维（比如 n*k n*kn*k）矩阵的相乘。则此时 W WW 矩阵的参数就大幅降低，为 O(nk)。公式如下：

$$W^{(2)} = W^T \cdot W$$

这就是 Rendle RendleRendle 等在 2010 年提出因子分解机（Factorization Machines，FM）的名字的由来。FM 的矩阵形式公式如下：

$$y_{FM} = \sigma\left(\vec{w}^T \cdot \vec{x} + \vec{x}^T \cdot W^T \cdot W \cdot \vec{x}\right)$$

将其写成内积的形式：

$$y_{FM} = \sigma(\langle \vec{w}, \vec{x} \rangle + \langle W \cdot \vec{x}, W \cdot \vec{x} \rangle)$$

利用 $\langle \sum_{i=1}^{n} \vec{a_i}, \sum_{i=1}^{n} \vec{a_i} \rangle = \sum_{i=1}^{n} \sum_{j=1}^{n} \langle \vec{a_i}, \vec{a_j} \rangle$，可以将上式进一步改写成求和式的形式：

$$y_{FM} = \sigma\left(\langle \vec{w}, \vec{x} \rangle + \sum_{i=1}^{n} \sum_{j=1}^{n} \langle x_i \cdot \vec{v}_i, x_j \cdot \vec{v}_j \rangle\right)$$

其中 $\overrightarrow{vi}$ 向量是矩阵 W WW 的第 i ii 列。为了去除重复项与特征平方项，上式可以进一步改写成更为常见的 FM 公式：

$$y_{FM} = \sigma\left(\langle \vec{w}, \vec{x} \rangle + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \vec{v}_i, \vec{v}_j \rangle x_i \cdot x_j\right)$$

对比二阶多项式模型，FM 模型中特征两两相乘（组合）的权重是相互不独立的，它是一种参数较少但表达力强的模型。
在给出 FM 的 TensorFlow 代码实现之前，值得一提的是 FM 通过内积进行无重复项与特征平方项的特征组合过程使用了一个小 trick，就是：

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} x_i x_j = 1/2 \times [(\sum_{i=1}^{n} x_i)^2 - \sum_{i=1}^{n} x_i^2]$$

```
class FM(Model):
    def __init__(self, input_dim=None, output_dim=1, factor_order=10, init_path=None,
opt_algo='gd', learning_rate=1e-2,
                 l2_w=0, l2_v=0, random_seed=None):
        Model.__init__(self)
        # 一次、二次交叉、偏置项
        init_vars = [('w', [input_dim, output_dim], 'xavier', dtype),
                     ('v', [input_dim, factor_order], 'xavier', dtype),
                     ('b', [output_dim], 'zero', dtype)]
        self.graph = tf.Graph()
        with self.graph.as_default():
            if random_seed is not None:
                tf.set_random_seed(random_seed)
            self.X = tf.sparse_placeholder(dtype)
            self.y = tf.placeholder(dtype)
            self.vars = init_var_map(init_vars, init_path)
            w = self.vars['w']
            v = self.vars['v']
            b = self.vars['b']
```

```
            # [(x1+x2+x3)^2 - (x1^2+x2^2+x3^2)]/2
            #  先计算所有的交叉项，再减去平方项(自己和自己相乘)
            X_square = tf.SparseTensor(self.X.indices, tf.square(self.X.values),
tf.to_int64(tf.shape(self.X)))
            xv = tf.square(tf.sparse_tensor_dense_matmul(self.X, v))
            p = 0.5 * tf.reshape(
                tf.reduce_sum(xv - tf.sparse_tensor_dense_matmul(X_square, tf.square(v)), 1),
                [-1, output_dim])
            xw = tf.sparse_tensor_dense_matmul(self.X, w)
            logits = tf.reshape(xw + b + p, [-1])
            self.y_prob = tf.sigmoid(logits)
            self.loss = tf.reduce_mean(
                tf.nn.sigmoid_cross_entropy_with_logits(logits=logits, labels=self.y)) + \
                            l2_w * tf.nn.l2_loss(xw) + \
                            l2_v * tf.nn.l2_loss(xv)
            self.optimizer = get_optimizer(opt_algo, learning_rate, self.loss)
            #GPU 设定
            config = tf.ConfigProto()
            config.gpu_options.allow_growth = True
            self.sess = tf.Session(config=config)
            #  图中所有 variable 初始化
            tf.global_variables_initializer().run(session=self.sess)
```

三、用神经网络的视角看 FM：嵌入后再进行内积

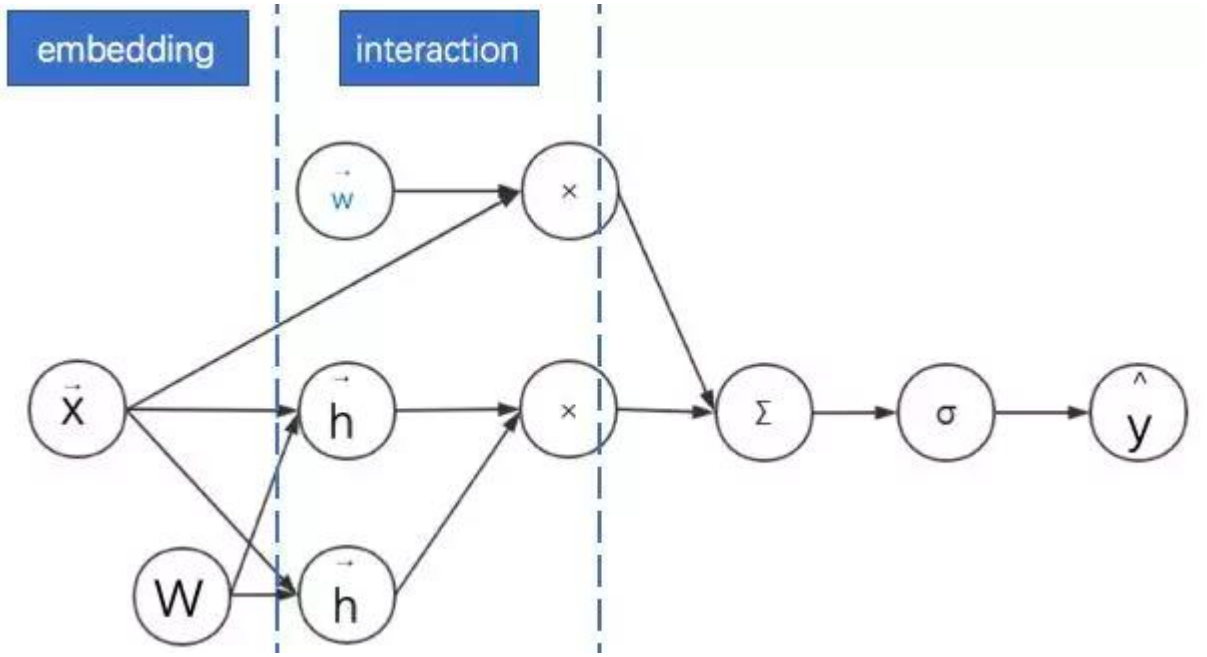
我们观察 FM 公式的矩阵内积形式：

$$y_{FM} = \sigma(\langle \vec{w}, \vec{x} \rangle + \langle W \cdot \vec{x}, W \cdot \vec{x} \rangle)$$

发现 $W \cdot \vec{x}$ 部分就是将离散系数特征通过矩阵乘法降维成一个低维稠密向量。这个过程对神经网络来说就叫做嵌入（embedding）。所以用神经网络视角来看：

a) FM首先是对离散特征进行嵌入。
b) 之后通过对嵌入后的稠密向量进行内积来进行二阶特征组合。
c) 最后再与线性模型的结果求和进而得到预估点击率。

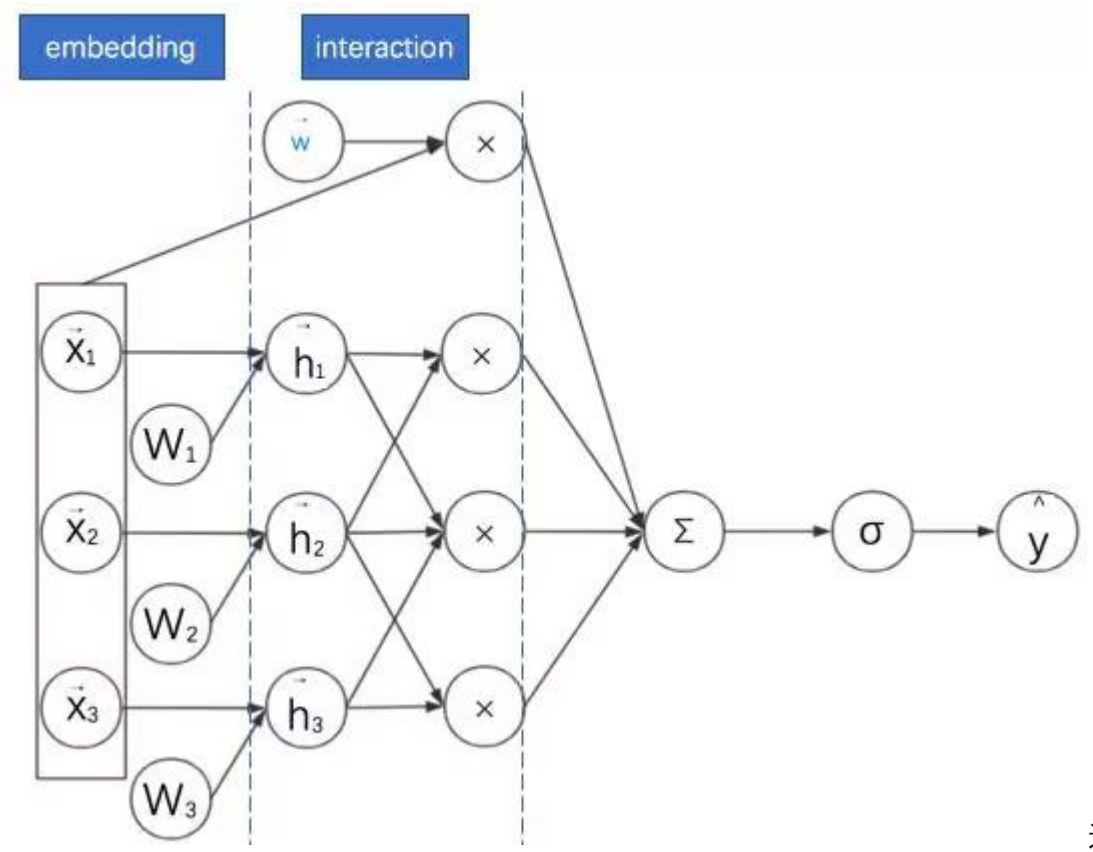
其示意图如下。为了表述清晰，我们绘制的是神经网络计算图而不是网络结构图——在网络结构图中增加了权重W位置。



四、FM 的实际应用：考虑领域信息

广告点击率预估模型中的特征以分领域的离散特征为主，如：广告类别、用户职业、手机 APP 列表等。由于连续特征比较好处理，为了简化起见，本文只考虑同时存在不同领域的离散特征的情形。处理离散特征的常见方法是通过独热（one-hot）编码转换为一系列二值特征向量。然后将这些高维稀疏特征通过嵌入（embedding）转换为低维连续特征。前面已经说明 FM 中间的一个核心步骤就是嵌入，但这个嵌入过程没有考虑领域信息。这使得同领域内的特征也被当做不同领域特征进行两两组合了。

其实可以将特征具有领域关系的特点作为先验知识加入到神经网络的设计中去：同领域的特征嵌入后直接求和作为一个整体嵌入向量，进而与其他领域的整体嵌入向量进行两两组合。而这个先嵌入后求和的过程，就是一个单领域的小离散特征向量乘以矩阵的过程。此时 FM 的过程变为：对不同领域的离散特征分别进行嵌入，之后再进行二阶特征的向量内积。其计算图图如下所示：



这样考虑其实是给 FM 增加了一个正则：考虑了领域内的信息的相似性。而且还有一个附加的好处，这些嵌入后的同领域特征可以拼接起来作为更深的神经网络的输入，达到降维的目的。接下来我们将反复看到这种处理方式。

此处需要注意，这与“基于领域的因子分解机”（Field-aware Factorization Machines，FFM）有区别。FFM 也是 FM 的另一种变体，也考虑了领域信息。但其不同点是同一个特征与不同领域进行特征组合时，其对应的嵌入向量是不同的。本文不考虑 FFM 的作用机制。

经过这些改进的 FM 终究还是浅层网络，它的表现力仍然有限。为了增加模型的表现力(model capacity)，一个自然的想法就是将该浅层网络不断“深化”。

五、embedding+MLP：深度学习 CTR 预估的通用框架

embedding+MLP 是对于分领域离散特征进行深度学习 CTR 预估的通用框架。深度学习在特征组合挖掘(特征学习)方面具有很大的优势。比如以 CNN 为代表的深度网络主要用于图像、语音等稠密特征上的学习，以 W2V、RNN 为代表的深度网络主要用于文本的同质化、序列化高维稀疏特征的学习。CTR 预估的主要场景是对离散且有具体领域的特征进行学习，所以其深度网络结构也不同于 CNN 与 RNN。

具体来说， embedding+MLP 的过程如下：

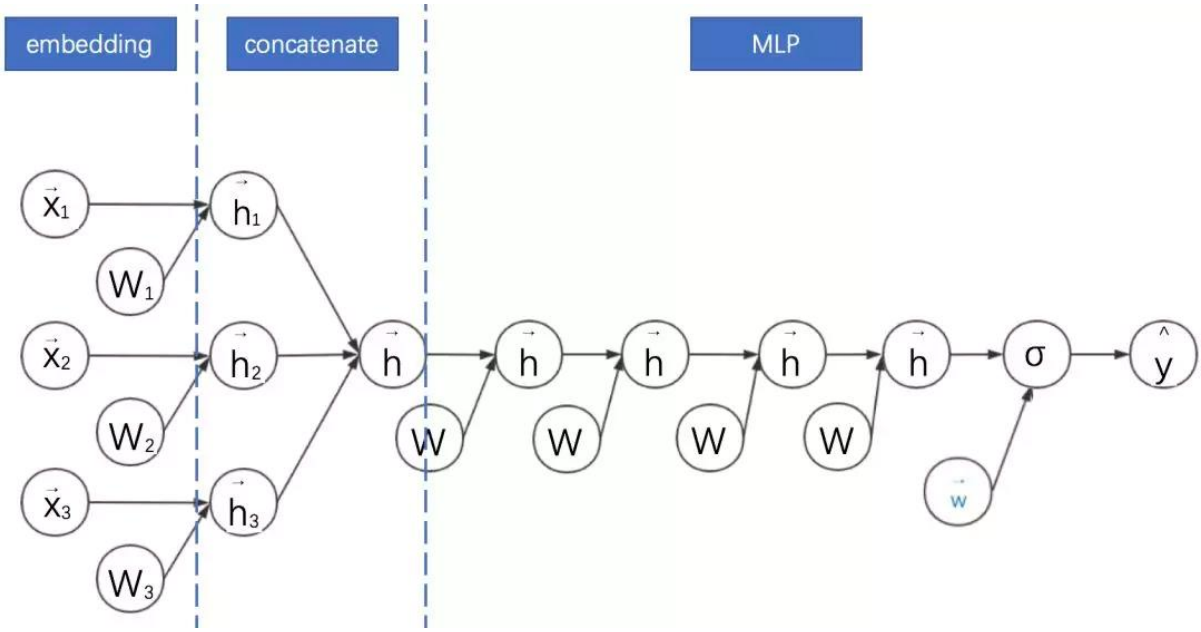
①对不同领域的 one-hot 特征进行嵌入（embedding），使其降维成低维度稠密特征。

②然后将这些特征向量拼接（concatenate）成一个隐含层。

③之后再不断堆叠全连接层，也就是多层感知机(Multilayer Perceptron, MLP，有时也叫作前馈神经网络)。

④最终输出预测的点击率。

其示意图如下：



embedding+MLP 的缺点是只学习高阶特征组合，对于低阶或者手动的特征组合不够兼容，而且参数较多，学习较困难。

六、FNN:FM 与 MLP 的串联结合

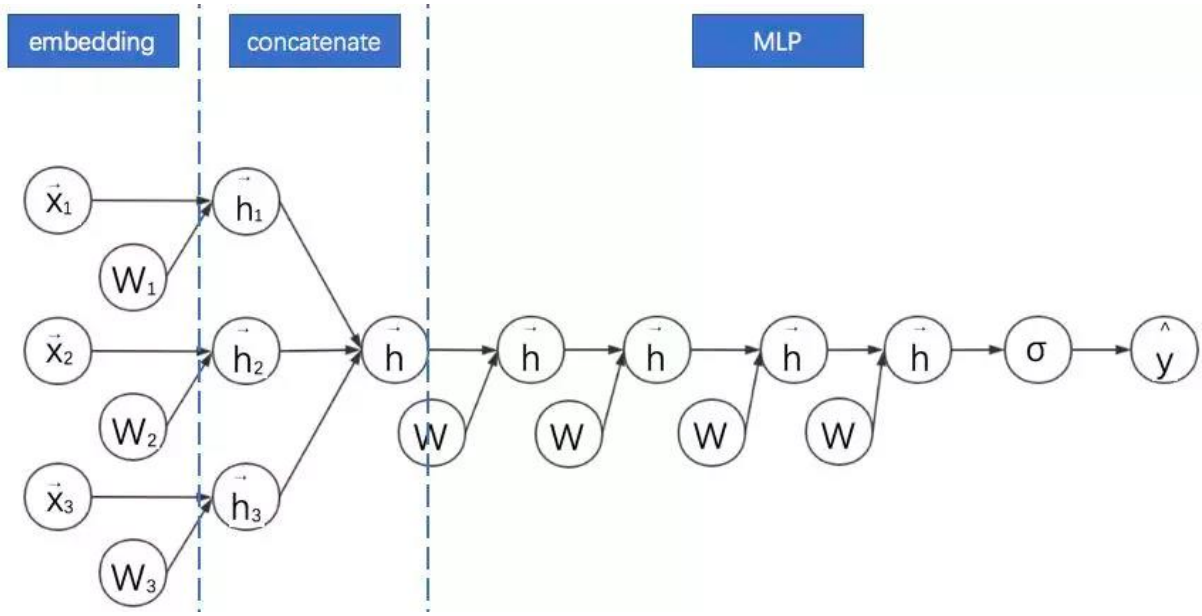
Weinan Zhang 等在 2016 年提出的因子分解机神经网络(Factorisation Machine supported Neural Network，FNN)将考 FM 与 MLP 进行了结合。它有着十分显著的特点：

采用 FM 预训练得到的隐含层及其权重作为神经网络的第一层的初始值，之后再不断堆叠全连接层，最终输出预测的点击率。

可以将 FNN 理解成一种特殊的 embedding+MLP，其要求第一层嵌入后的各领域特征维度一致，并且嵌入权重的初始化是 FM 预训练好的。

这不是一个端到端的训练过程，有贪心训练的思路。而且如果不考虑预训练过程，模型网络结构也没有考虑低阶特征组合。

其计算图如下所示：



通过观察 FFN 的计算图可以看出其与 embedding+MLP 确实非常像。**不过此处省略了 FNN 的 FM 部分的线性模块。**这种省略为了更好地进行两个模型的对比。接下来的计算图我们都会省略线性模块。

此处附上 FNN 的代码实现：

```
class FNN(Model):
    def __init__(self, field_sizes=None, embed_size=10, layer_sizes=None, layer_acts=None,
drop_out=None,
                 embed_l2=None, layer_l2=None, init_path=None, opt_algo='gd',
learning_rate=1e-2, random_seed=None):
        Model.__init__(self)
        init_vars = []
        num_inputs = len(field_sizes)
        for i in range(num_inputs):
            init_vars.append(('embed_%d' % i, [field_sizes[i], embed_size], 'xavier', dtype))
        node_in = num_inputs * embed_size
        for i in range(len(layer_sizes)):
            init_vars.append(('w%d' % i, [node_in, layer_sizes[i]], 'xavier', dtype))
            init_vars.append(('b%d' % i, [layer_sizes[i]], 'zero', dtype))
            node_in = layer_sizes[i]
        self.graph = tf.Graph()
        with self.graph.as_default():
            if random_seed is not None:
                tf.set_random_seed(random_seed)
            self.X = [tf.sparse_placeholder(dtype) for i in range(num_inputs)]
            self.y = tf.placeholder(dtype)
            self.keep_prob_train = 1 - np.array(drop_out)
            self.keep_prob_test = np.ones_like(drop_out)
            self.layer_keeps = tf.placeholder(dtype)
            self.vars = init_var_map(init_vars, init_path)
            w0 = [self.vars['embed_%d' % i] for i in range(num_inputs)]
            xw = tf.concat([tf.sparse_tensor_dense_matmul(self.X[i], w0[i]) for i in
range(num_inputs)], 1)
            l = xw
                    #全连接部分
            for i in range(len(layer_sizes)):
                wi = self.vars['w%d' % i]
                bi = self.vars['b%d' % i]
                print(l.shape, wi.shape, bi.shape)
                l = tf.nn.dropout(
                    activate(
                        tf.matmul(l, wi) + bi,
                        layer_acts[i]),
                    self.layer_keeps[i])
            l = tf.squeeze(l)
            self.y_prob = tf.sigmoid(l)
            self.loss = tf.reduce_mean(
                tf.nn.sigmoid_cross_entropy_with_logits(logits=l, labels=self.y))
            if layer_l2 is not None:
                self.loss += embed_l2 * tf.nn.l2_loss(xw)
                for i in range(len(layer_sizes)):
                    wi = self.vars['w%d' % i]
```

```
                self.loss += layer_l2[i] * tf.nn.l2_loss(wi)
        self.optimizer = get_optimizer(opt_algo, learning_rate, self.loss)
        config = tf.ConfigProto()
        config.gpu_options.allow_growth = True
        self.sess = tf.Session(config=config)
        tf.global_variables_initializer().run(session=self.sess)
```

七、DeepFM: FM 与 MLP 的并联结合

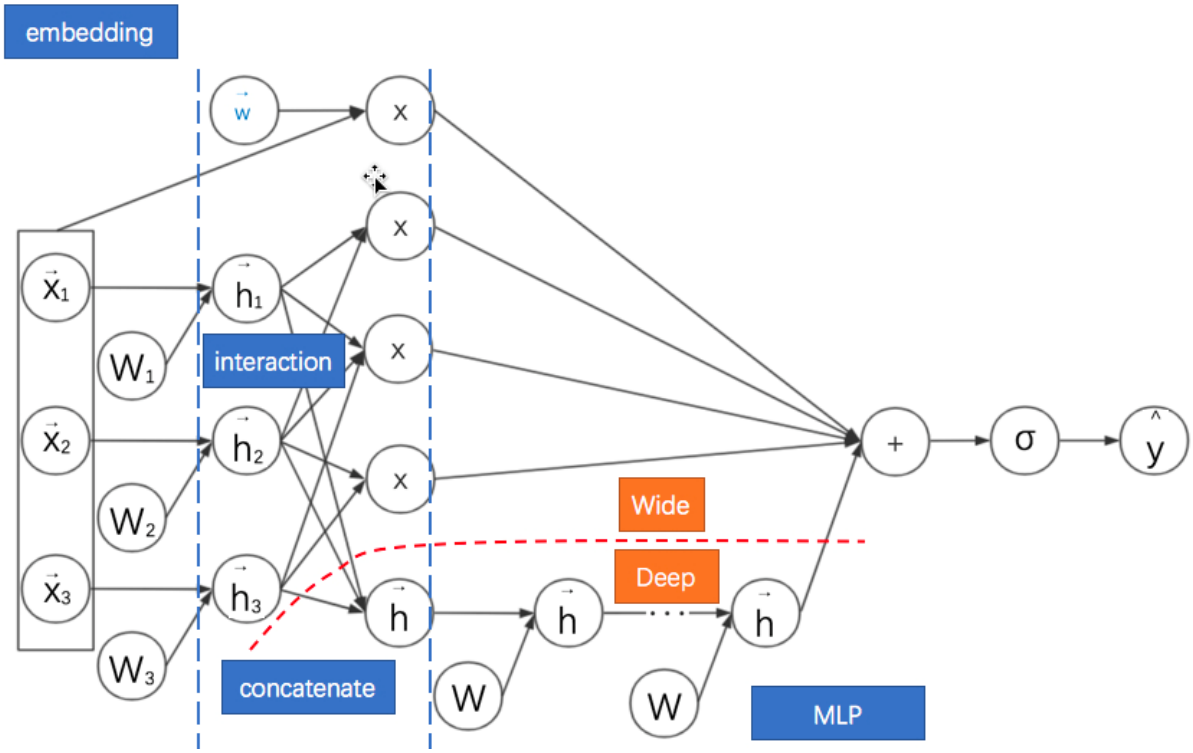
针对 FNN 需要预训练的问题，Huifeng Guo 等提出了深度因子分解机模型（Deep Factorisation Machine, DeepFM, 2017）。该模型的特点是：

i) 不需要预训练。

ii)将考虑领域信息的 FM 部分与 MLP 部分并联起来（借用初中电路的术语），其实就是对两个模型进行联合训练。

iii)考虑领域信息的 FM 部分的嵌入向量拼接起来作为 MLP 部分的输入特征，也就是是两个模型共享嵌入后的特征。

其计算图如下所示：



通过观察 DeepFM 的计算图可以看出红色虚线以上部分其实就是 FM 部分，虚线以下就是 MLP 部分。

此处附上 DeepFM 的代码实现：

```
def model_fn(features, labels, mode, params):
    """Bulid Model function f(x) for Estimator."""
    #------超参数的设定----
    field_size = params["field_size"]
    feature_size = params["feature_size"]
    embedding_size = params["embedding_size"]
    l2_reg = params["l2_reg"]
    learning_rate = params["learning_rate"]
    #batch_norm_decay = params["batch_norm_decay"]
    #optimizer = params["optimizer"]
```

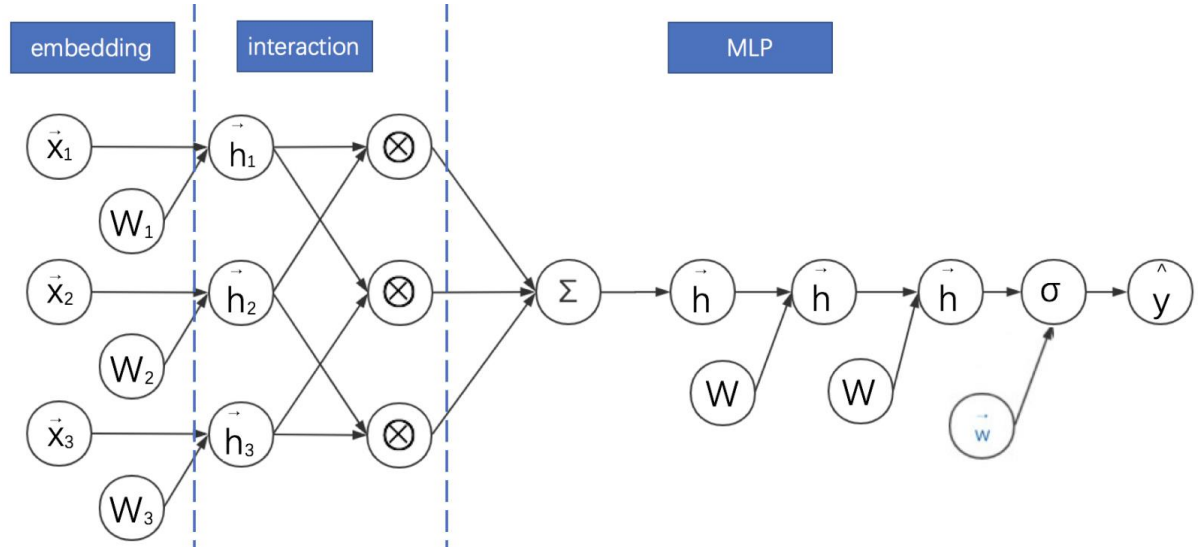
```
        layers = map(int, params["deep_layers"].split(','))
        dropout = map(float, params["dropout"].split(','))
        #------权重------
        FM_B = tf.get_variable(name='fm_bias', shape=[1], initializer=tf.constant_initializer(0.0))
        FM_W = tf.get_variable(name='fm_w', shape=[feature_size],
initializer=tf.glorot_normal_initializer())
        # F
        FM_V = tf.get_variable(name='fm_v', shape=[feature_size, embedding_size],
initializer=tf.glorot_normal_initializer())
        # F * E
        #------build feaure-------
        feat_ids    = features['feat_ids']
        feat_ids = tf.reshape(feat_ids,shape=[-1,field_size]) # None * f/K * K
        feat_vals = features['feat_vals']
        feat_vals = tf.reshape(feat_vals,shape=[-1,field_size]) # None * f/K * K
        #------build f(x)------
        with tf.variable_scope("First-order"):
            feat_wgts = tf.nn.embedding_lookup(FM_W, feat_ids) # None * f/K * K
            y_w = tf.reduce_sum(tf.multiply(feat_wgts, feat_vals),1)
        with tf.variable_scope("Second-order"):
            embeddings = tf.nn.embedding_lookup(FM_V, feat_ids) # None * f/K * K * E
            feat_vals = tf.reshape(feat_vals, shape=[-1, field_size, 1]) # None * f/K * K * 1  ?
            embeddings = tf.multiply(embeddings, feat_vals) #vij*xi
            sum_square = tf.square(tf.reduce_sum(embeddings,1)) # None * K * E
            square_sum = tf.reduce_sum(tf.square(embeddings),1)
            y_v = 0.5*tf.reduce_sum(tf.subtract(sum_square, square_sum),1)              # None * 1
        with tf.variable_scope("Deep-part"):
            if FLAGS.batch_norm:
                #normalizer_fn = tf.contrib.layers.batch_norm
                #normalizer_fn = tf.layers.batch_normalization
                if mode == tf.estimator.ModeKeys.TRAIN:
                    train_phase = True
                    #normalizer_params = {'decay': batch_norm_decay, 'center': True, 'scale':
True, 'updates_collections': None, 'is_training': True, 'reuse': None}
                else:
                    train_phase = False
                    #normalizer_params = {'decay': batch_norm_decay, 'center': True, 'scale':
True, 'updates_collections': None, 'is_training': False, 'reuse': True}
            else:
                normalizer_fn = None
                normalizer_params = None
            deep_inputs = tf.reshape(embeddings,shape=[-1,field_size*embedding_size]) # None
* (F*K)
            for i in range(len(layers)):
                #if FLAGS.batch_norm:
                #       deep_inputs = batch_norm_layer(deep_inputs, train_phase=train_phase,
scope_bn='bn_%d' %i)
```

```
                    #normalizer_params.update({'scope': 'bn_%d' %i})
                deep_inputs = tf.contrib.layers.fully_connected(inputs=deep_inputs,
num_outputs=layers[i], \
                    #normalizer_fn=normalizer_fn, normalizer_params=normalizer_params, \
                    weights_regularizer=tf.contrib.layers.l2_regularizer(l2_reg), scope='mlp%d' % i)
                if FLAGS.batch_norm:
                    deep_inputs = batch_norm_layer(deep_inputs, train_phase=train_phase,
scope_bn='bn_%d' %i)    #放在 RELU 之后  https://github.com/ducha-aiki/caffenet-
benchmark/blob/master/batchnorm.md#bn----before-or-after-relu
                if mode == tf.estimator.ModeKeys.TRAIN:
                    deep_inputs = tf.nn.dropout(deep_inputs, keep_prob=dropout[i])
#Apply Dropout after all BN layers and set dropout=0.8(drop_ratio=0.2)
                    #deep_inputs = tf.layers.dropout(inputs=deep_inputs, rate=dropout[i],
training=mode == tf.estimator.ModeKeys.TRAIN)
            y_deep = tf.contrib.layers.fully_connected(inputs=deep_inputs, num_outputs=1,
activation_fn=tf.identity, \
                    weights_regularizer=tf.contrib.layers.l2_regularizer(l2_reg), scope='deep_out')
            y_d = tf.reshape(y_deep,shape=[-1])
            #sig_wgts = tf.get_variable(name='sigmoid_weights', shape=[layers[-1]],
initializer=tf.glorot_normal_initializer())
            #sig_bias = tf.get_variable(name='sigmoid_bias', shape=[1],
initializer=tf.constant_initializer(0.0))
            #deep_out = tf.nn.xw_plus_b(deep_inputs,sig_wgts,sig_bias,name='deep_out')
        with tf.variable_scope("DeepFM-out"):
            #y_bias = FM_B * tf.ones_like(labels, dtype=tf.float32)  # None * 1  warning;这里不能
用 label，否则调用 predict/export 函数会出错，train/evaluate 正常；初步判断 estimator 做了
优化，用不到 label 时不传
            y_bias = FM_B * tf.ones_like(y_d, dtype=tf.float32)      # None * 1
            y = y_bias + y_w + y_v + y_d
            pred = tf.sigmoid(y)
        predictions={"prob": pred}
        export_outputs =
{tf.saved_model.signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY:
tf.estimator.export.PredictOutput(predictions)}
        # Provide an estimator spec for `ModeKeys.PREDICT`
        if mode == tf.estimator.ModeKeys.PREDICT:
            return tf.estimator.EstimatorSpec(
                    mode=mode,
                    predictions=predictions,
                    export_outputs=export_outputs)
        #------bulid loss------
        loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=y, labels=labels)) + \
            l2_reg * tf.nn.l2_loss(FM_W) + \
            l2_reg * tf.nn.l2_loss(FM_V) #+ \ l2_reg * tf.nn.l2_loss(sig_wgts)
        # Provide an estimator spec for `ModeKeys.EVAL`
        eval_metric_ops = {
            "auc": tf.metrics.auc(labels, pred)
```

```
    }
    if mode == tf.estimator.ModeKeys.EVAL:
        return tf.estimator.EstimatorSpec(
                    mode=mode,
                    predictions=predictions,
                    loss=loss,
                    eval_metric_ops=eval_metric_ops)
    #------bulid optimizer------
    if FLAGS.optimizer == 'Adam':
        optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate, beta1=0.9,
beta2=0.999, epsilon=1e-8)
    elif FLAGS.optimizer == 'Adagrad':
        optimizer = tf.train.AdagradOptimizer(learning_rate=learning_rate,
initial_accumulator_value=1e-8)
    elif FLAGS.optimizer == 'Momentum':
        optimizer = tf.train.MomentumOptimizer(learning_rate=learning_rate,
momentum=0.95)
    elif FLAGS.optimizer == 'ftrl':
        optimizer = tf.train.FtrlOptimizer(learning_rate)
    train_op = optimizer.minimize(loss, global_step=tf.train.get_global_step())
    # Provide an estimator spec for `ModeKeys.TRAIN` modes
    if mode == tf.estimator.ModeKeys.TRAIN:
        return tf.estimator.EstimatorSpec(
                    mode=mode,
                    predictions=predictions,
                    loss=loss,
                    train_op=train_op)
```

八、NFM:通过逐元素乘法延迟 FM 的实现过程

我们再回到考虑领域信息的 FM，它仍有改进的空间。因为以上这些网络的 FM 部分都是只进行嵌入向量的两两内积后直接求和，没有充分利用二阶特征组合的信息。Xiangnan He 等在 2017 年提出了神经网络因子分解机（Neural Factorization Machines，NFM）对此作出了改进。其计算图如下所示：



NFM 的基本特点是：

1 利用二阶交互池化层（Bi-Interaction Pooling）对 FM 嵌入后的向量两两进行元素级别的乘法，形成同维度的向量求和后作为前馈神经网络的输入。计算图中用圈乘⊗ \bigotimes⊗ 表示逐元素乘法运算。

2 NFM 与 DeepFM 的区别是没有单独的 FM 的浅层网络进行联合训练，而是将其整合后直接输出给前馈神经网络。

3 当 MLP 的全连接层都是恒等变换且最后一层参数全为 1 时，NFM 就退化成了 FM。可见，NFM 是 FM 的推广，它推迟了 FM 的实现过程，并在其中加入了更多非线性运算。

4 另一方面，我们观察计算图会发现 NFM 与 FNN 非常相似。它们的主要区别是 NFM 在 embedding 之后对特征进行了两两逐元素乘法。因为逐元素相乘的向量维数不变，之后对这些向量求和的维数仍然与 embedding 的维数一致。因此输入到 MLP 的参数比起直接 concatenate 的 FNN 更少。

此处附上 NFM 的代码实现，完整数据和代码请参考网盘：

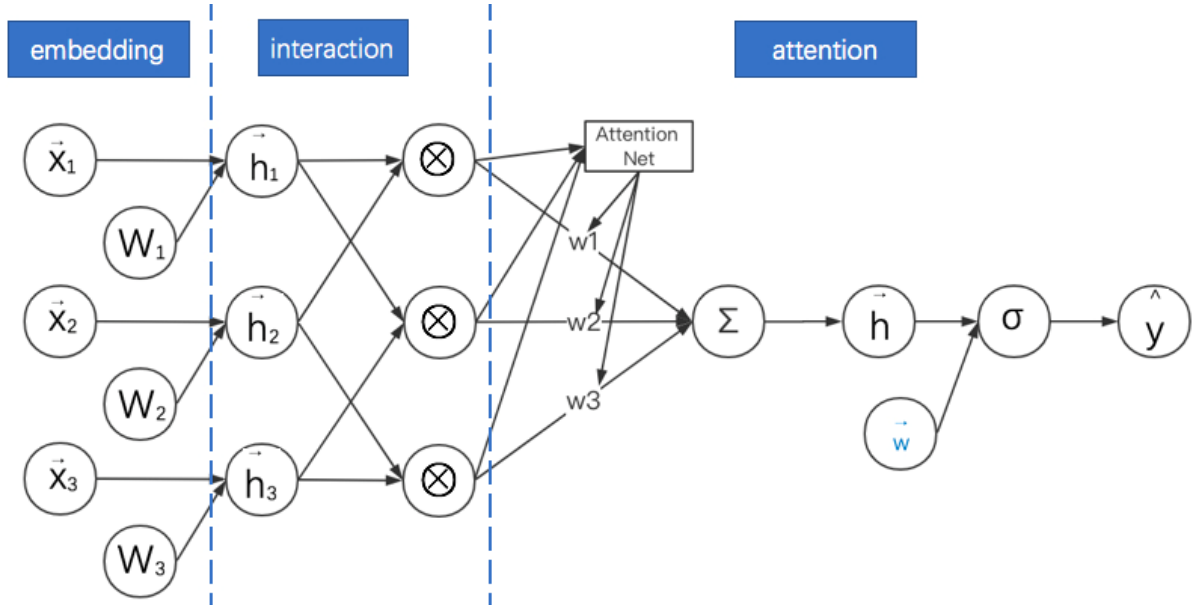
```
def model_fn(features, labels, mode, params):
    """Bulid Model function f(x) for Estimator."""
    #------hyperparameters----
    field_size = params["field_size"]
    feature_size = params["feature_size"]
    embedding_size = params["embedding_size"]
    l2_reg = params["l2_reg"]
    learning_rate = params["learning_rate"]
    #optimizer = params["optimizer"]
    layers = map(int, params["deep_layers"].split(','))
    dropout = map(float, params["dropout"].split(','))
    #------bulid weights------
    Global_Bias = tf.get_variable(name='bias', shape=[1], initializer=tf.constant_initializer(0.0))
    Feat_Bias = tf.get_variable(name='linear', shape=[feature_size],
initializer=tf.glorot_normal_initializer())
    Feat_Emb = tf.get_variable(name='emb', shape=[feature_size,embedding_size],
initializer=tf.glorot_normal_initializer())
    #------build feaure-------
    feat_ids  = features['feat_ids']
    feat_ids = tf.reshape(feat_ids,shape=[-1,field_size])
    feat_vals = features['feat_vals']
    feat_vals = tf.reshape(feat_vals,shape=[-1,field_size])
    #------build f(x)------
    with tf.variable_scope("Linear-part"):
        feat_wgts = tf.nn.embedding_lookup(Feat_Bias, feat_ids)                    # None * F
* 1
        y_linear = tf.reduce_sum(tf.multiply(feat_wgts, feat_vals),1)
    with tf.variable_scope("BiInter-part"):
        embeddings = tf.nn.embedding_lookup(Feat_Emb, feat_ids)                    #
None * F * K
        feat_vals = tf.reshape(feat_vals, shape=[-1, field_size, 1])
        embeddings = tf.multiply(embeddings, feat_vals)
# vij * xi
        sum_square_emb = tf.square(tf.reduce_sum(embeddings,1))
        square_sum_emb = tf.reduce_sum(tf.square(embeddings),1)
```

```
        deep_inputs = 0.5*tf.subtract(sum_square_emb, square_sum_emb)              # None * K
    with tf.variable_scope("Deep-part"):
        if mode == tf.estimator.ModeKeys.TRAIN:
            train_phase = True
        else:
            train_phase = False
        if mode == tf.estimator.ModeKeys.TRAIN:
            deep_inputs = tf.nn.dropout(deep_inputs, keep_prob=dropout[0])
# None * K
        for i in range(len(layers)):
            deep_inputs = tf.contrib.layers.fully_connected(inputs=deep_inputs,
num_outputs=layers[i], \
                weights_regularizer=tf.contrib.layers.l2_regularizer(l2_reg), scope='mlp%d' % i)
            if FLAGS.batch_norm:
                deep_inputs = batch_norm_layer(deep_inputs, train_phase=train_phase,
scope_bn='bn_%d' %i)    #放在 RELU 之后  https://github.com/ducha-aiki/caffenet-
benchmark/blob/master/batchnorm.md#bn----before-or-after-relu
            if mode == tf.estimator.ModeKeys.TRAIN:
                deep_inputs = tf.nn.dropout(deep_inputs, keep_prob=dropout[i])
#Apply Dropout after all BN layers and set dropout=0.8(drop_ratio=0.2)
                #deep_inputs = tf.layers.dropout(inputs=deep_inputs, rate=dropout[i],
training=mode == tf.estimator.ModeKeys.TRAIN)
        y_deep = tf.contrib.layers.fully_connected(inputs=deep_inputs, num_outputs=1,
activation_fn=tf.identity, \
            weights_regularizer=tf.contrib.layers.l2_regularizer(l2_reg), scope='deep_out')
        y_d = tf.reshape(y_deep,shape=[-1])
    with tf.variable_scope("NFM-out"):
        #y_bias = Global_Bias * tf.ones_like(labels, dtype=tf.float32)  # None * 1  warning;这
里不能用 label，否则调用 predict/export 函数会出错，train/evaluate 正常；初步判断
estimator 做了优化，用不到 label 时不传
        y_bias = Global_Bias * tf.ones_like(y_d, dtype=tf.float32)                 # None * 1
        y = y_bias + y_linear + y_d
        pred = tf.sigmoid(y)
    predictions={"prob": pred}
    export_outputs =
{tf.saved_model.signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY:
tf.estimator.export.PredictOutput(predictions)}
    # Provide an estimator spec for `ModeKeys.PREDICT`
    if mode == tf.estimator.ModeKeys.PREDICT:
        return tf.estimator.EstimatorSpec(
            mode=mode,
            predictions=predictions,
            export_outputs=export_outputs)
    #------bulid loss------
    loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=y, labels=labels)) + \
        l2_reg * tf.nn.l2_loss(Feat_Bias) + l2_reg * tf.nn.l2_loss(Feat_Emb)
    # Provide an estimator spec for `ModeKeys.EVAL`
```

```
    eval_metric_ops = {
        "auc": tf.metrics.auc(labels, pred)
    }
    if mode == tf.estimator.ModeKeys.EVAL:
        return tf.estimator.EstimatorSpec(
                    mode=mode,
                    predictions=predictions,
                    loss=loss,
                    eval_metric_ops=eval_metric_ops)
    #------bulid optimizer------
    if FLAGS.optimizer == 'Adam':
        optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate, beta1=0.9,
beta2=0.999, epsilon=1e-8)
    elif FLAGS.optimizer == 'Adagrad':
        optimizer = tf.train.AdagradOptimizer(learning_rate=learning_rate,
initial_accumulator_value=1e-8)
    elif FLAGS.optimizer == 'Momentum':
        optimizer = tf.train.MomentumOptimizer(learning_rate=learning_rate,
momentum=0.95)
    elif FLAGS.optimizer == 'ftrl':
        optimizer = tf.train.FtrlOptimizer(learning_rate)
    train_op = optimizer.minimize(loss, global_step=tf.train.get_global_step())
    # Provide an estimator spec for `ModeKeys.TRAIN` modes
    if mode == tf.estimator.ModeKeys.TRAIN:
        return tf.estimator.EstimatorSpec(
                    mode=mode,
                    predictions=predictions,
                    loss=loss,
                    train_op=train_op)
```

九、AFM: 对简化版 NFM 进行加权求和

NFM 的主要创新点是在 FM 过程中添加了逐元素相乘的运算来增加模型的复杂度。但没有在此基础上添加更复杂的运算过程，比如对加权求和。Jun Xiao 等在 2017 年提出了注意力因子分解模型（Attentional Factorization Machine，AFM）就是在这个方向上的改进。其计算图如下所示：



AFM 的特点是：

AFM 与 NFM 都是致力于充分利用二阶特征组合的信息，对嵌入后的向量两两进行逐元素乘法，形成同维度的向量。而且 AFM 没有 MLP 部分。

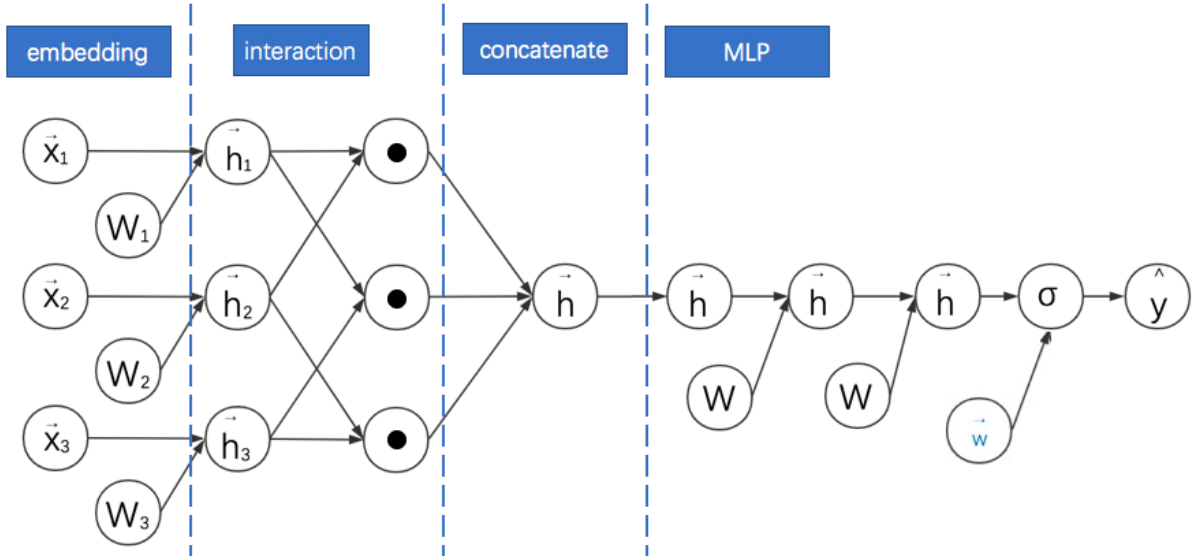
AFM 通过在逐元素乘法之后形成的向量进行加权求和，而且权重是基于网络自身来产生的。其方法是引入一个注意力子网络（Attention Net）。

当权重都相等时，AFM 退化成无全连接层的 NFM。

“注意力子网络”的主要操作是进行矩阵乘法，其最终输出结果为 softmax，以保证各分量的权重本身是一个概率分布。

11.PNN:通过改进向量乘法运算延迟 FM 的实现过程

再回到 FM。既然 AFM、NFM 可以通过添加逐元素乘法的运算来增加模型的复杂度，那向量乘法有这么多，可否用其他的方法增加 FM 复杂度？答案是可以的。Huifeng Guo 等在 2016 年提出了基于向量积的神经网络（Product-based Neural Networks，PNN）就是一个典型例子。其简化计算图如下所示：



对比之前模型的计算图，我们可以发现 PNN 的基本特点是：

1 利用二阶向量积层（Pair-wisely Connected Product Layer）对 FM 嵌入后的向量两两进行向量积，形成的结果作为之后 MLP 的输入。计算图中用圆点•表示向量积运算。PNN 采用的向量积有内积与外积两种形式。

2 需要说明的是，本计算图中省略了 PNN 中向量与常数 1 进行的乘法运算。这部分其实

与 FNN 类似，不是 PNN 的主要创新点。故在此图中省略。

3 对于内积形式的 PNN，因为两个向量相乘的结果为标量，可以直接把各个标量“拼接”成一个大向量，就可以作为 MLP 的输入了。

4 当 MLP 的全连接层都是恒等变换且最后一层参数全为 1 时，内积形式的 PNN 就退化成了 FM。

5 对于外积形式的 PNN，因为两个向量相乘相当于列向量与行向量进行矩阵相乘，得到的结果为一个矩阵。各个矩阵向之前内积形式的操作一样直接拼接起来维数太多，论文的简化方案是直接对各个矩阵进行求和，得到的新矩阵（可以理解成之后对其拉长成向量）就直接作为 MLP 的输入。

6 观察计算图发现外积形式的 PNN 与 NFM 很像，其实就是 PNN 把 NFM 的逐元素乘法换成了外积。

此处分别附上 PNN 的内积与外积形式代码。

```
class PNN1(Model):
    def __init__(self, field_sizes=None, embed_size=10, layer_sizes=None, layer_acts=None,
drop_out=None,
                 embed_l2=None, layer_l2=None, init_path=None, opt_algo='gd',
learning_rate=1e-2, random_seed=None):
        Model.__init__(self)
        init_vars = []
        num_inputs = len(field_sizes)
        for i in range(num_inputs):
            init_vars.append(('embed_%d' % i, [field_sizes[i], embed_size], 'xavier', dtype))
        num_pairs = int(num_inputs * (num_inputs - 1) / 2)
        node_in = num_inputs * embed_size + num_pairs
        # node_in = num_inputs * (embed_size + num_inputs)
        for i in range(len(layer_sizes)):
            init_vars.append(('w%d' % i, [node_in, layer_sizes[i]], 'xavier', dtype))
            init_vars.append(('b%d' % i, [layer_sizes[i]], 'zero', dtype))
            node_in = layer_sizes[i]
        self.graph = tf.Graph()
        with self.graph.as_default():
            if random_seed is not None:
                tf.set_random_seed(random_seed)
            self.X = [tf.sparse_placeholder(dtype) for i in range(num_inputs)]
            self.y = tf.placeholder(dtype)
            self.keep_prob_train = 1 - np.array(drop_out)
            self.keep_prob_test = np.ones_like(drop_out)
            self.layer_keeps = tf.placeholder(dtype)
            self.vars = init_var_map(init_vars, init_path)
            w0 = [self.vars['embed_%d' % i] for i in range(num_inputs)]
            xw = tf.concat([tf.sparse_tensor_dense_matmul(self.X[i], w0[i]) for i in
range(num_inputs)], 1)
            xw3d = tf.reshape(xw, [-1, num_inputs, embed_size])
            row = []
            col = []
            for i in range(num_inputs-1):
                for j in range(i+1, num_inputs):
```

```
            row.append(i)
            col.append(j)
    # batch * pair * k
    p = tf.transpose(
        # pair * batch * k
        tf.gather(
            # num * batch * k
            tf.transpose(
                xw3d, [1, 0, 2]),
            row),
        [1, 0, 2])
    # batch * pair * k
    q = tf.transpose(
        tf.gather(
            tf.transpose(
                xw3d, [1, 0, 2]),
            col),
        [1, 0, 2])
    p = tf.reshape(p, [-1, num_pairs, embed_size])
    q = tf.reshape(q, [-1, num_pairs, embed_size])
    ip = tf.reshape(tf.reduce_sum(p * q, [-1]), [-1, num_pairs])
    # simple but redundant
    # batch * n * 1 * k, batch * 1 * n * k
    # ip = tf.reshape(
    #         tf.reduce_sum(
    #             tf.expand_dims(xw3d, 2) *
    #             tf.expand_dims(xw3d, 1),
    #             3),
    #         [-1, num_inputs**2])
    l = tf.concat([xw, ip], 1)
    for i in range(len(layer_sizes)):
        wi = self.vars['w%d' % i]
        bi = self.vars['b%d' % i]
        l = tf.nn.dropout(
            activate(
                tf.matmul(l, wi) + bi,
                layer_acts[i]),
            self.layer_keeps[i])
    l = tf.squeeze(l)
    self.y_prob = tf.sigmoid(l)
    self.loss = tf.reduce_mean(
        tf.nn.sigmoid_cross_entropy_with_logits(logits=l, labels=self.y))
    if layer_l2 is not None:
        self.loss += embed_l2 * tf.nn.l2_loss(xw)
        for i in range(len(layer_sizes)):
            wi = self.vars['w%d' % i]
            self.loss += layer_l2[i] * tf.nn.l2_loss(wi)
```

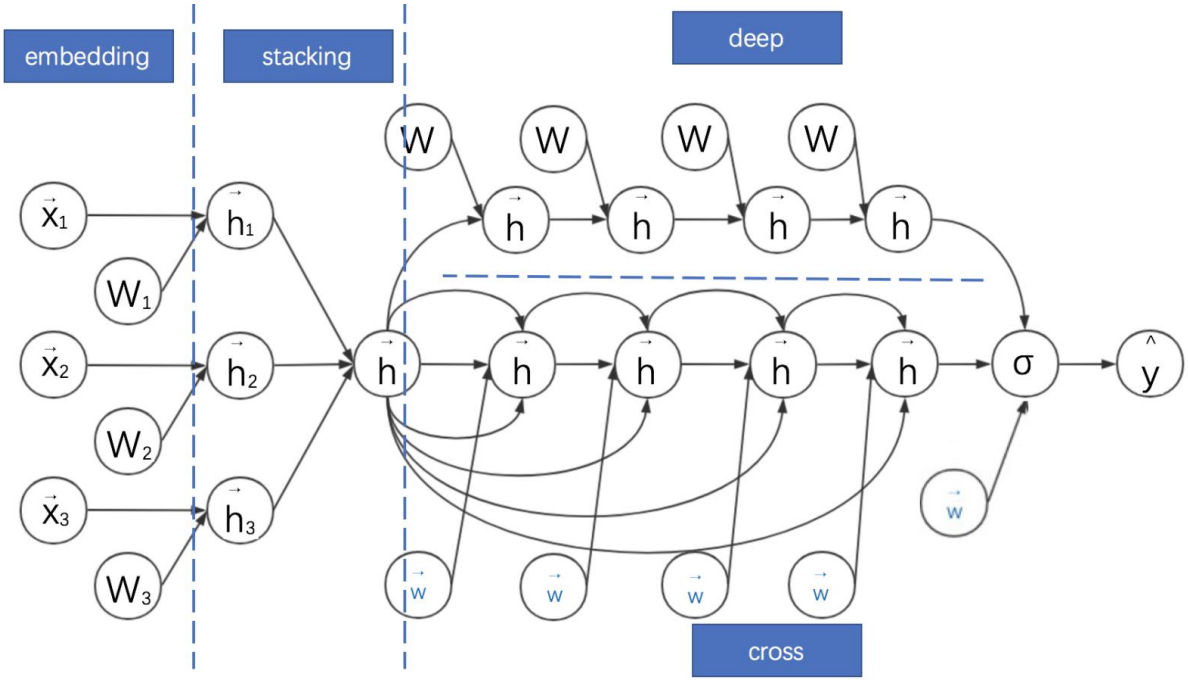
```
                self.optimizer = get_optimizer(opt_algo, learning_rate, self.loss)
                config = tf.ConfigProto()
                config.gpu_options.allow_growth = True
                self.sess = tf.Session(config=config)
                tf.global_variables_initializer().run(session=self.sess)
class PNN2(Model):
    def __init__(self, field_sizes=None, embed_size=10, layer_sizes=None, layer_acts=None,
drop_out=None,
                 embed_l2=None, layer_l2=None, init_path=None, opt_algo='gd',
learning_rate=1e-2, random_seed=None,
                 layer_norm=True):
        Model.__init__(self)
        init_vars = []
        num_inputs = len(field_sizes)
        for i in range(num_inputs):
            init_vars.append(('embed_%d' % i, [field_sizes[i], embed_size], 'xavier', dtype))
        num_pairs = int(num_inputs * (num_inputs - 1) / 2)
        node_in = num_inputs * embed_size + num_pairs
        init_vars.append(('kernel', [embed_size, num_pairs, embed_size], 'xavier', dtype))
        for i in range(len(layer_sizes)):
            init_vars.append(('w%d' % i, [node_in, layer_sizes[i]], 'xavier', dtype))
            init_vars.append(('b%d' % i, [layer_sizes[i]], 'zero',    dtype))
            node_in = layer_sizes[i]
        self.graph = tf.Graph()
        with self.graph.as_default():
            if random_seed is not None:
                tf.set_random_seed(random_seed)
            self.X = [tf.sparse_placeholder(dtype) for i in range(num_inputs)]
            self.y = tf.placeholder(dtype)
            self.keep_prob_train = 1 - np.array(drop_out)
            self.keep_prob_test = np.ones_like(drop_out)
            self.layer_keeps = tf.placeholder(dtype)
            self.vars = init_var_map(init_vars, init_path)
            w0 = [self.vars['embed_%d' % i] for i in range(num_inputs)]
            xw = tf.concat([tf.sparse_tensor_dense_matmul(self.X[i], w0[i]) for i in
range(num_inputs)], 1)
            xw3d = tf.reshape(xw, [-1, num_inputs, embed_size])
            row = []
            col = []
            for i in range(num_inputs - 1):
                for j in range(i + 1, num_inputs):
                    row.append(i)
                    col.append(j)
            # batch * pair * k
            p = tf.transpose(
                # pair * batch * k
                tf.gather(
```

```
            # num * batch * k
            tf.transpose(
                xw3d, [1, 0, 2]),
            row),
        [1, 0, 2])
# batch * pair * k
q = tf.transpose(
        tf.gather(
            tf.transpose(
                xw3d, [1, 0, 2]),
            col),
        [1, 0, 2])
# b * p * k
p = tf.reshape(p, [-1, num_pairs, embed_size])
# b * p * k
q = tf.reshape(q, [-1, num_pairs, embed_size])
# k * p * k
k = self.vars['kernel']
# batch * 1 * pair * k
p = tf.expand_dims(p, 1)
# batch * pair
kp = tf.reduce_sum(
        # batch * pair * k
        tf.multiply(
            # batch * pair * k
            tf.transpose(
                # batch * k * pair
                tf.reduce_sum(
                    # batch * k * pair * k
                    tf.multiply(
                        p, k),
                    -1),
                [0, 2, 1]),
            q),
        -1)
#
# if layer_norm:
#         # x_mean, x_var = tf.nn.moments(xw, [1], keep_dims=True)
#         # xw = (xw - x_mean) / tf.sqrt(x_var)
#         # x_g = tf.Variable(tf.ones([num_inputs * embed_size]), name='x_g')
#         # x_b = tf.Variable(tf.zeros([num_inputs * embed_size]), name='x_b')
#         # x_g = tf.Print(x_g, [x_g[:10], x_b])
#         # xw = xw * x_g + x_b
#         p_mean, p_var = tf.nn.moments(op, [1], keep_dims=True)
#         op = (op - p_mean) / tf.sqrt(p_var)
#         p_g = tf.Variable(tf.ones([embed_size**2]), name='p_g')
#         p_b = tf.Variable(tf.zeros([embed_size**2]), name='p_b')
```

```
#         # p_g = tf.Print(p_g, [p_g[:10], p_b])
#         op = op * p_g + p_b
l = tf.concat([xw, kp], 1)
for i in range(len(layer_sizes)):
    wi = self.vars['w%d' % i]
    bi = self.vars['b%d' % i]
    l = tf.nn.dropout(
        activate(
            tf.matmul(l, wi) + bi,
            layer_acts[i]),
        self.layer_keeps[i])
l = tf.squeeze(l)
self.y_prob = tf.sigmoid(l)
self.loss = tf.reduce_mean(
    tf.nn.sigmoid_cross_entropy_with_logits(logits=l, labels=self.y))
if layer_l2 is not None:
    self.loss += embed_l2 * tf.nn.l2_loss(xw)#tf.concat(w0, 0))
    for i in range(len(layer_sizes)):
        wi = self.vars['w%d' % i]
        self.loss += layer_l2[i] * tf.nn.l2_loss(wi)
self.optimizer = get_optimizer(opt_algo, learning_rate, self.loss)
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
self.sess = tf.Session(config=config)
tf.global_variables_initializer().run(session=self.sess)
```

十一、DCN:高阶 FM 的降维实现

以上的 FM 推广形式，主要是对 FM 进行二阶特征组合。高阶特征组合是通过 MLP 实现的。但这两种实现方式是有很大不同的，**FM 更多是通过向量 embedding 之间的内积来实现，而 MLP 则是在向量 embedding 之后一层一层进行权重矩阵乘法实现。**可否直接将 FM 的过程在高阶特征组合上进行推广？答案是可以的。Ruoxi Wang 等在 2017 提出的深度与交叉神经网络（Deep & Cross Network，DCN）就是在这个方向进行改进的。DCN 的计算图如下：



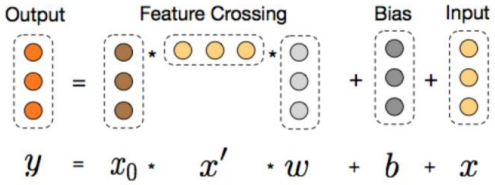
DCN的特点如下：

①Deep部分就是普通的MLP网络，主要是全连接。

②与DeepFM类似，DCN是由embedding+MLP部分与cross部分进行联合训练的。Cross部分是对FM部分的推广。

③cross部分的公式如下：

$$\mathbf{x}_{l+1} = \mathbf{x}_0 \mathbf{x}_l^T \mathbf{w}_l + \mathbf{b}_l + \mathbf{x}_l = f(\mathbf{x}_l, \mathbf{w}_l, \mathbf{b}_l) + \mathbf{x}_l, \ \ \mathbf{x}_{l+1} = \mathbf{x}_0 \mathbf{x}_l^T \mathbf{w}_l + \mathbf{b}_l + \mathbf{x}_l = f(\mathbf{x}_l, \mathbf{w}_l, \mathbf{b}_l) + \mathbf{x}_l,$$



$$y = x_0 \cdot x' \cdot w + b + x$$

④可以证明，cross网络是FM的过程在高阶特征组合的推广。完全的证明需要一些公式推导，感兴趣的同学可以直接参考原论文的附录。

⑤而用简单的公式证明可以得到一个很重要的结论：只有两层且第一层与最后一层权重参数相等时的Cross网络与简化版FM等价。

⑥此处对应简化版的FM视角是将拼接好的稠密向量作为输入向量，且不做领域方面的区分（但产生这些稠密向量的过程是考虑领域信息的，相对全特征维度的全连接层减少了大量参数，可以视作稀疏链接思想的体现）。而且之后进行embedding权重矩阵W只有一列——是退化成列向量的情形。

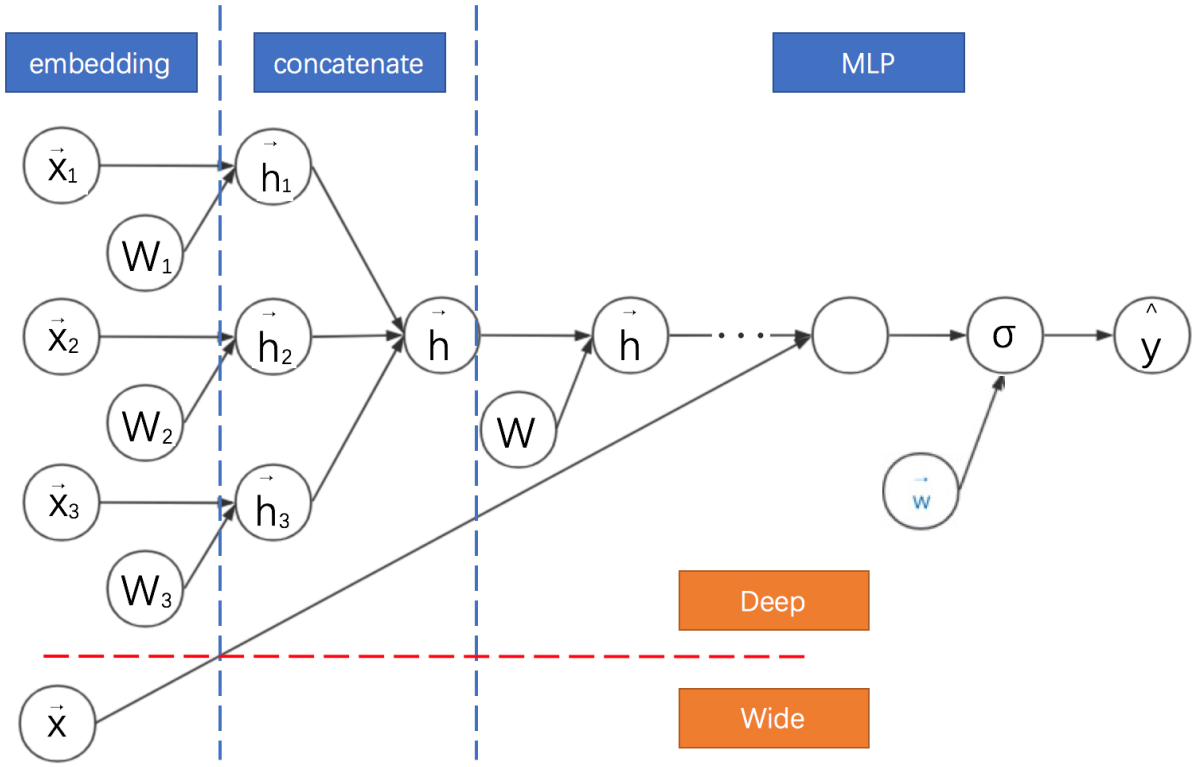
⑦与MLP网络相比，Cross部分在增加高阶特征组合的同时减少了参数的个数，并省去了非线性激活函数。

十二、Wide&Deep: DeepFM与DCN的基础框架

开篇已经提到，本文思路有两条主线。到此为止已经将基于FM的主线介绍基本完毕。接下来将串讲从embedding+MLP自身的演进特点的CTR预估模型主线，而这条思路与我们之前的FM思路同样有千丝万缕的联系。

Google在2016年提出的宽度与深度模型（Wide&Deep）在深度学习CTR预估模型中占有非常重要的位置，它奠定了之后基于深度学习的广告点击率预估模型的框架。

Wide&Deep将深度模型与线性模型进行联合训练，二者的结果求和输出为最终点击率。其计算图如下：



我们将 Wide&Deep 的计算图与之前的模型进行对比可知：

1 Wide&Deep 是前面介绍模型 DeepFM 与 DCN 的基础框架。这些模型均采用神经网络联合训练的思路，对神经网络进行并联。

2 DeepFM、DCN 与 Wide&Deep 的 Deep 部分都是 MLP。

3 Wide&Deep 的 Wide 部分是逻辑回归，可以手动设计组合特征。

4 DeepFM 的 Wide 部分是 FM，DCN 的 Wide 部分是 Cross 网络，二者均不强求手动设计特征。但此时都与字面意义上的 Wide 有一定差异，因为均共享了降维后的嵌入特征。

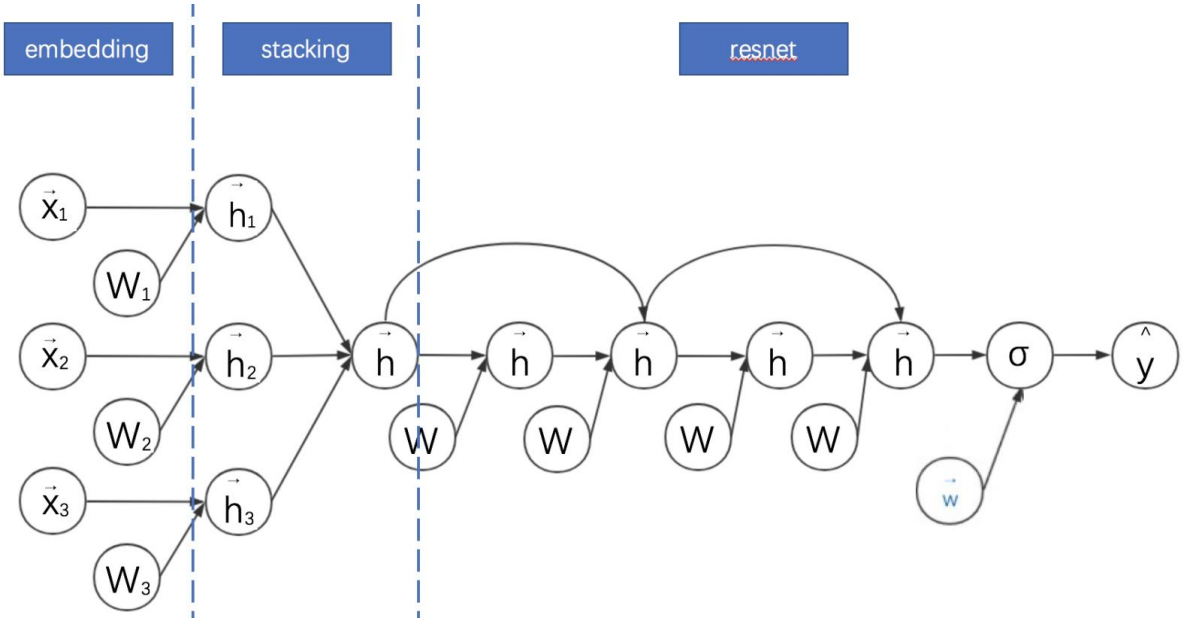
此处附上 Wide&Deep 的代码实现：

```
def get_model(model_type, model_dir):
    print("Model directory = %s" % model_dir)
    #  对 checkpoint 去做设定
    runconfig = tf.contrib.learn.RunConfig(
        save_checkpoints_secs=None,
        save_checkpoints_steps = 100,
    )
    m = None
    #  宽模型
    if model_type == 'WIDE':
        m = tf.contrib.learn.LinearClassifier(
            model_dir=model_dir,
            feature_columns=wide_columns)
    #  深度模型
    if model_type == 'DEEP':
        m = tf.contrib.learn.DNNClassifier(
            model_dir=model_dir,
            feature_columns=deep_columns,
            hidden_units=[100, 50, 25])
    #  宽度深度模型
```

```
    if model_type == 'WIDE_AND_DEEP':
        m = tf.contrib.learn.DNNLinearCombinedClassifier(
            model_dir=model_dir,
            linear_feature_columns=wide_columns,
            dnn_feature_columns=deep_columns,
            dnn_hidden_units=[100, 70, 50, 25],
            config=runconfig)
    print('estimator built')
    return m
```

十三、Deep Cross: DCN 由其残差网络思想进化

由 K. He 等提出的深度残差网络能够大大加深神经网络的深度，同时不会引起退化的问题，显著提高了模型的精度。Ying Shan 等将该思路应用到广告点击率预估模型中，提出深度交叉模型（DeepCross,2016 Deep Cross, 2016DeepCross,2016）。Deep Cross 的计算图如下：



将 Deep Cross 与之前的模型对比，可以发现其特点是：

①对 embedding+MLP 的改进主要是 MLP 部分增加跳跃连接成为残差网络。

②Deep Cross 与传统的残差网络的区别主要是没有采用卷积操作。其中一个原因是在广告点击率预估领域，特征不具备平移不变性。

③DCN 其实是从 Deep Cross 进化出来的版本。DCN 相对 Deep Cross 的主要贡献是解耦了 Deep 与 Cross（特征交叉）部分。

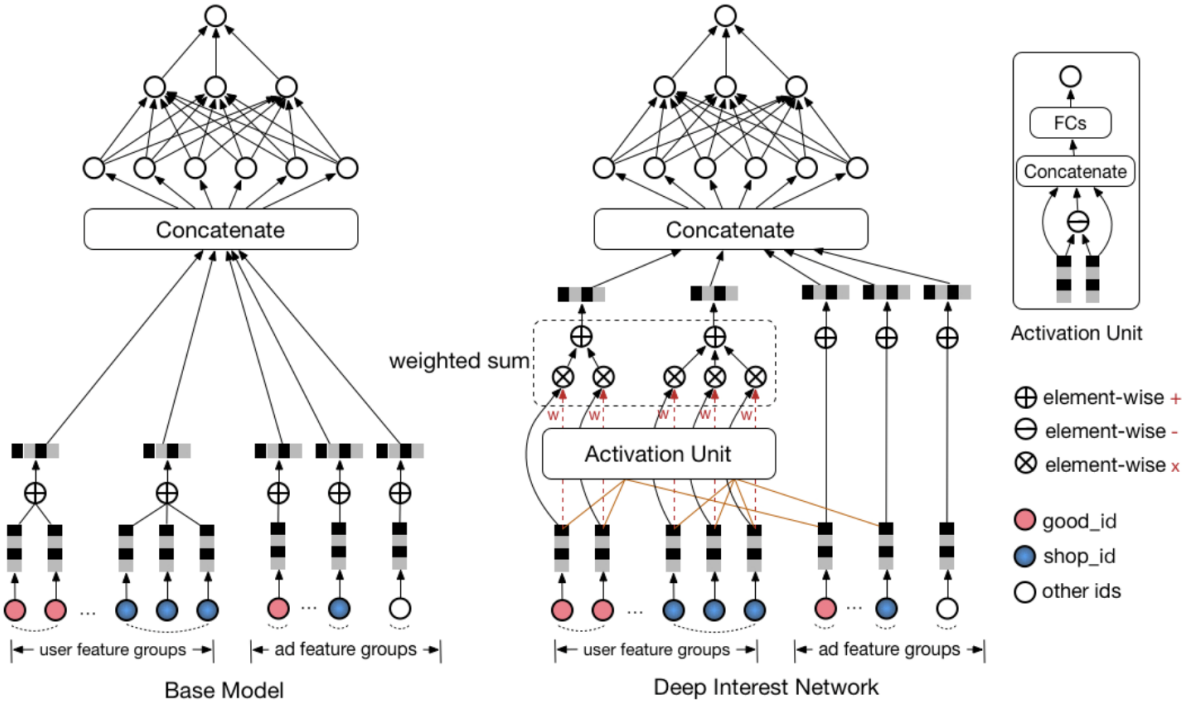
④因此 DCN 中的 Cross 部分可以理解为残差网络的变体：其将 Deep Cross 的跨越链接缩短为只有一层，而全连接部分改为与权重向量和输入向量的内积。

十四、DIN:对同领域历史信息引入注意力机制的 MLP

以上神经网络对同领域离散特征的处理基本是将其嵌入后直接求和，这在一般情况下没太大问题。但其实可以做得更加精细。比如对于历史统计类特征。以用户历史浏览的商户 id 为例，假设用户历史浏览了 10 个商户，这些商户 id 的常规处理方法是作为同一个领域的特征嵌入后直接求和得到一个嵌入向量。但这 10 个商户只有一两个商户与当前被预测的广告所在的商户相似，其他商户关系不大。增加这两个商户在求和过程中的权重，应该能够更好地提高模型的表现力。而增加求和权重的思路就是典型的注意力机制思路。

由 **Bahdanau et al. (2015) **引入的现代注意力机制，本质上是加权平均(权重是模型根据数据学习出来的)，其在机器翻译上应用得非常成功。受注意力机制的启发，Guorui Zhou 等在 2017 年提出了深度兴趣网络(Deep Interest Network，DIN)。DIN 主要关注用户在同一领域的历

史行为特征，如浏览了多个商家、多个商品等。DIN 可以对这些特征分配不同的权重进行求和。其网络结构图如下：



a) 此处采用原论文的结构图，表示起来更清晰。

b) DIN 考虑对同一领域的历史特征进行加权求和，以加强其感兴趣的特征的影响。

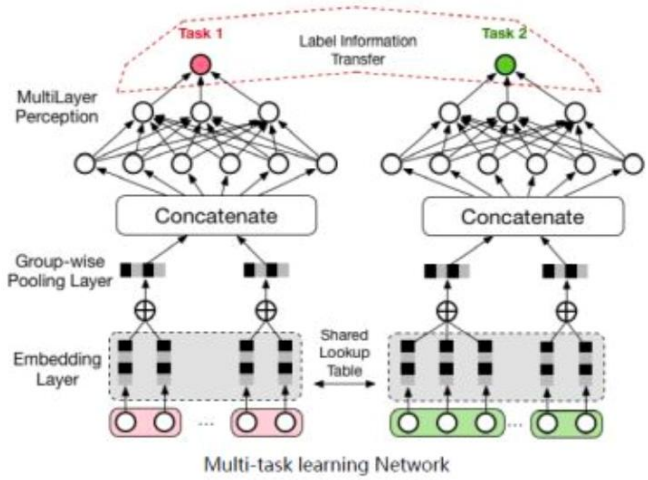
c) 用户的每个领域的历史特征权重则由该历史特征及其对应备选广告特征通过一个子网络得到。即用户历史浏览的商户特征与当前浏览商户特征对应，历史浏览的商品特征与当前浏览商品特征对应。

d) 权重子网络主要包括特征之间的元素级别的乘法、加法和全连接等操作。

e) AFM 也引入了注意力机制。但是 AFM 是将注意力机制与 FM 同领域特征求和之后进行结合，DIN 直接是将注意力机制与同领域特征求和之前进行结合。

十五、多任务视角：信息的迁移与补充

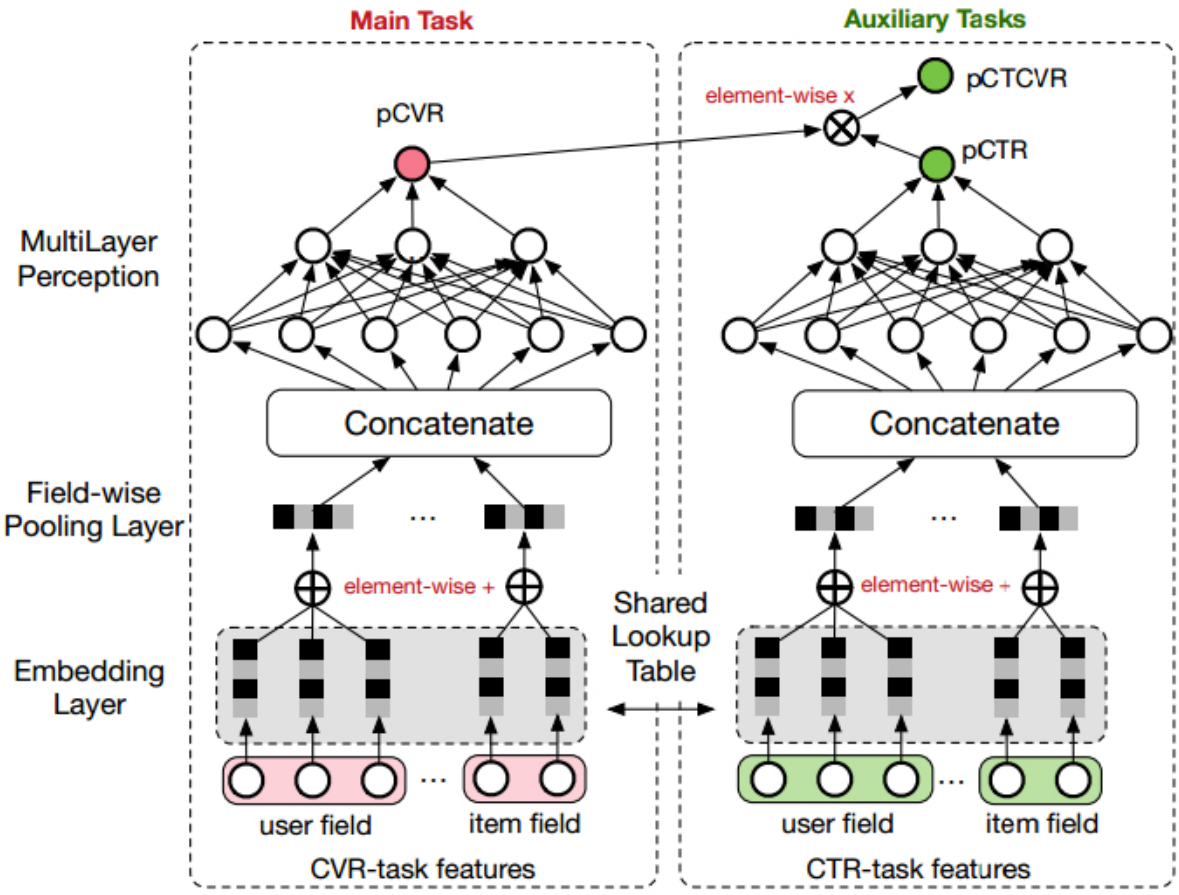
对于数据驱动的解决方案而言，数据和模型同样重要，数据(特征)通常决定了效果的上限，各式各样的模型会以不同的方式去逼近这个上限。而所有算法应用的老司机都知道很多场景下，如果有更多的数据进行模型训练，效果一般都能显著得到提高。广告也是一样的场景，在很多电商的平台上会有很多不同场景的广告位，每个场景蕴含了用户的不同兴趣的表达，这些信息的汇总与融合可以带来最后效果的提升。但是将不同场景的数据直接进行合并用来训练(ctr/cvr)模型，结果很多时候并不是很乐观，仔细想想也是合理的，不同场景下的样本分布存在差异，直接对样本累加会影响分布导致效果负向。



而深度学习发展，使得信息的融合与应用有了更好的进展，用 Multi−task Multi-taskMulti−task learning(MTL) learning(MTL)learning(MTL)的方式可以很漂亮的解决上面提到的问题。我们不直接对样本进行累加和训练，而是像上图所示，把两个场景分为两个 task，即分为两个子网络。对单个网络而言，底层的 embedding 层的表达受限于单场景的数据量，很可能学习不充分。而上图这样的网络结合，使得整个训练过程有了表示学习的共享（Shared Lookup Table），这种共享有助于大样本的子任务帮助小样本的子任务，使得底层的表达学习更加充分。
DeepFM 和 DCN 也用到了这个思路！只是它们是对同一任务的不同模型进行结合，而多任务学习是对不同任务的不同模型进行结合。而且，我们可以玩得更加复杂。
Multi-task learning(MTL)整个结构的上层的不同的 task 的子网络是不一样的，这样每个子网络可以各自去拟合自己 task 对应的概念分布。并且，取决于问题与场景的相似性和复杂度，可以把底层的表达学习，从简单的共享 embedding 到共享一些层次的表达。极端的情况是我们可以直接共享所有的表达学习(representation learning)部分，而只接不同的网络 head 来完成不一样的任务。
这样带来的另外一个好处是，不同的 task 可以共享一部分计算，从而实现计算的加速。
值得一提的另一篇 paper 是阿里妈妈团队提出的“完整空间多任务模型”（Entire Space Multi-Task Model，ESMM），也是很典型的多任务学习和信息补充思路，这篇 paper 解决的问题不是 ctr(点击率)预估而是 cvr(转化率)预估，传统 CVR 预估模型会有比较明显的样本选择偏差（sample selection bias）和训练数据过于稀疏（data sparsity ）的问题，而 ESMM 模型利用用户行为序列数据，在完整的样本数据空间同时学习点击率和转化率（post-view clickthrough&conversion rate，CTCVR），在一定程度上解决了这个问题。
在电商的场景下，用户的决策过程很可能是这样的，在观察到系统展现的推荐商品列表后，点击自己感兴趣的商品，进而产生购买行为。所以用户行为遵循这样一个决策顺序：impression → click → conversion。CVR 模型旨在预估用户在观察到曝光商品进而点击到商品详情页之后购买此商品的概率，即 pCVR = p(conversion|click,impression)。
预估点击率 pCTR，预估点击下单率 pCVR 和预估点击与下单率 pCTCVR 关系如下。

$$\underbrace{p(y=1,z=1|\boldsymbol{x})}_{pCTCVR}=\underbrace{p(y=1|\boldsymbol{x})}_{pCTR}\times\underbrace{p(z=1|y=1,\boldsymbol{x})}_{pCVR}$$

传统的 CVR 预估任务通常采用类似于 CTR 预估的技术进行建模。但是不同于 CTR 预估任务的是，这个场景面临一些特有的挑战：1）样本选择偏差；2）训练数据稀疏；3）延迟反馈等。
![图](http://ww1.sinaimg.cn/large/b57cc2efly1fxccmw03jwj20g306qjrq.jpg
ESMM 模型提出了下述的网络结构进行问题建模



EMMS的特点是：

①在整个样本空间建模。pCVR 可以在先估计出pCTR 和pCTCVR之后计算得出，如下述公式。从原理上看，相当于分别单独训练两个模型拟合出pCTR 和pCTCVR，进而计算得到pCVR。

$$p(z=1|y=1,\boldsymbol{x})=\frac{p(y=1,z=1|\boldsymbol{x})}{p(y=1|\boldsymbol{x})}$$

注意到pCTR 和pCTCVR是在整个样本空间上建模得到的，pCVR 只是一个中间变量。因此，ESMM模型是在整个样本空间建模，而不像传统CVR预估模型那样只在点击样本空间建模。

②特征表示层共享。ESMM模型借鉴迁移学习和multi-task learning的思路，在两个子网络的embedding层共享特征表示词典。embedding层的表达参数占了整个网络参数的绝大部分，参数量大，需要大量的训练样本才能学习充分。显然CTR任务的训练样本量要大大超过CVR任务的训练样本量，ESMM模型中特征表示共享的机制能够使得CVR子任务也能够从只有展现没有点击的样本中学习，从而在一定程度上缓解训练数据稀疏性问题。

十六、各种模型的对比和总结

前面介绍了各种基于深度学习的广告点击率预估算法模型，针对不同的问题、基于不同的思路，不同的模型有各自的特点。各个模型具体关系比较如下表1所示：

表 1. 各模型对比

	Embedding+MLP	FM	Attention	Resnet
Wide&Deep	√	×	×	×
FNN	√	√	×	×
DeepFM	√	√	×	×
NFM	√	√	×	×
DIN	√	×	√	×
AFM	×	√	√	×
Deep Cross	√	×	×	√
DCN	√	√	×	√

本文从开篇就说明这些模型推演的核心思路是“通过设计网络结构进行组合特征的挖掘”，其在各个模型的实现方式如下：

FM 其实是对嵌入特征进行两两内积实现特征二阶组合；FNN 在 FM 基础上引入了 MLP；

DeepFM 通过联合训练、嵌入特征共享来兼顾 FM 部分与 MLP 部分不同的特征组合机制；

NFM、PNN 则是通过改造向量积的方式来延迟 FM 的实现过程，在其中添加非线性成分来提升

模型表现力；
AFM 更进一步，直接通过子网络来对嵌入向量的两两逐元素乘积进行加权求和，以实现不同组合的差异化，也是一种延迟 FM 实现的方式；
DCN 则是将 FM 进行高阶特征组合的方向上进行推广，并结合 MLP 的全连接式的高阶特征组合机制；
Wide&Deep 是兼容手工特征组合与 MLP 的特征组合方式，是许多模型的基础框架；
Deep Cross 是引入残差网络机制的前馈神经网络，给高维的 MLP 特征组合增加了低维的特征组合形式，启发了 DCN；
DIN 则是对用户侧的某历史特征和广告侧的同领域特征进行组合，组合成的权重反过来重新影响用户侧的该领域各历史特征的求和过程；
多任务视角则是更加宏观的思路，结合不同任务（而不仅是同任务的不同模型）对特征的组合过程，以提高模型的泛化能力。
当然，广告点击率预估深度学习模型还有很多，比如 Jie Zhu 提出的基于决策树的神经网络（Deep Embedding Forest）将深度学习与树型模型结合起来。如果数据特征存在图像或者大量文本相关特征，传统的卷积神经网络、循环神经网络均可以结合到广告点击率预估的场景中。各个深度模型都有相应的特点，限于篇幅，我们就不再赘述了。

十七、后记

目前深度学习的算法层出不穷，看论文确实有些应接不暇。我们的经验有两点：要有充分的生产实践经验，同时要有扎实的算法理论基础。很多论文的亮点其实是来自于实际做工程的经验。也辛亏笔者一直都在生产一线并带领算法团队进行工程研发（当然也因此荒废了近 2 年的博客，T△T ），积淀了一些特征工程、模型训练的经验，才勉强跟得上新论文。比如 DIN“对用户侧的某领域历史特征基于广告侧的同领域特征进行加权求和”的思想，其实与传统机器学习对强业务相关特征进行针对性特征组合的特征工程思路比较相似。另一方面，对深度学习的经典、前沿方法的熟悉也很重要。从前面我们的串讲也能够看出，CTR 预估作为一个业务特点很强的场景，在应用深度学习的道路上，也充分借鉴了注意力机制、残差网络、联合训练、多任务学习等经典的深度学习方法。了解博主的朋友也知道我们一直推崇理论与实践相结合的思路，我们自身对这条经验也非常受用。当然，计算广告是一个很深的领域，自己研究尚浅，串讲难免存在纰漏。欢迎大家指出问题，共同交流学习。

十八、参考文献

陈巧红,余仕敏,贾宇波. 广告点击率预估技术综述[J]. 浙江理工大学学报. 2015(11).
纪文迪,王晓玲,周傲英. 广告点击率估算技术综述[J]. 华东师范大学学报(自然科学版). 2013(03).
Rendle S. Factorization machines. Data Mining (ICDM), 2010 IEEE 10th International Conference on. 2010.
Heng-Tze Cheng and Levent Koc. Wide & deep learning for recommender systems. In Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, pages 7–10. ACM, 2016.
Weinan Zhang, Tianming Du, and Jun Wang. Deep learning over multi-field categorical data - - A case study on user response prediction. In ECIR, 2016.
Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. arXiv preprint arXiv:1703.04247 (2017).
Xiangnan He and Tat-Seng Chua. Neural Factorization Machines for Sparse Predictive Analytics SIGIR. 355–364. 2017.
Guorui Zhou, Chengru Song, Xiaoqiang Zhu, Xiao Ma, Yanghui Yan, Xingya Dai, Han Zhu, Junqi Jin, Han Li, and Kun Gai. 2017. Deep Interest Network for Click-Through Rate Prediction. arXiv preprint arXiv:1706.06978 (2017).
J. Xiao, H. Ye, X. He, H. Zhang, F. Wu, and T.-S. Chua. Attentional factorization machines:

Learning the weight of feature interactions via attention networks. In IJCAI, 2017.
Ying Shan, T Ryan Hoens, Jian Jiao, Haijing Wang, Dong Yu, and JC Mao. 2016. Deep Crossing: Web-Scale Modeling without Manually Cra ed Combinatorial Features. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 255–262.
Wang, R., Fu, B., Fu, G., Wang, M.: Deep & cross network for ad click predictions. In: Proceedings of the ADKDD 17. pp. 12:1–12:7 (2017).
Ying Shan, T Ryan Hoens, et al. Deep crossing: Web-scale modeling without manually crafted combinatorial features. KDD '16. ACM, 2016.
Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In Proceedings of the 10th ACM Conference on Recommender Systems, pages 191–198. ACM, 2016.
Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385 (2015).

13、请简述什么是 wide&deep 模型

解析:
本题解析来源：https://mp.weixin.qq.com/s/5wdGemYBtkYUvq1n5ioyFg ，
https://mp.weixin.qq.com/s/CkgTvwIRg8R9RX-qSVKZUA
名词解释
1.1 Memorization 和 Generalization
Google Wide&Deep 论文中，通篇都是这两个词，必须搞懂是怎么回事!
这个是从人类的认知学习过程中演化来的。人类的大脑很复杂，它可以记忆 (memorize) 下每天发生的事情（麻雀可以飞，鸽子可以飞）然后泛化 (generalize) 这些知识到之前没有看到过的东西（有翅膀的动物都能飞)。但是泛化的规则有时候不是特别的准，有时候会出错（有翅膀的动物都能飞吗)。那怎么办那，没关系，记忆 (memorization) 可以修正泛化的规则 (generalized rules)，叫做特例（企鹅有翅膀，但是不能飞)。
这就是 Memorization 和 Generalization 的来由或者说含义。
Wide&Deep Mode 就是希望计算机可以像人脑一样，可以同时发挥 memorization 和 generalization 的作用。--Heng-Tze Cheng(Wide&Deep 作者)
1.2 Wide 和 Deep
同样，这两个词也是通篇出现，究竟什么意思你明白了没?
其实，Wide 也是一种特殊的神经网络，他的输入直接和输出相连。属于广义线性模型的范畴。Deep 就是指 Deep Neural Network，这个很好理解。Wide Linear Model 用于 memorization；Deep Neural Network 用于 generalization。左侧是 Wide-only，右侧是 Deep-only，中间是 Wide & Deep：

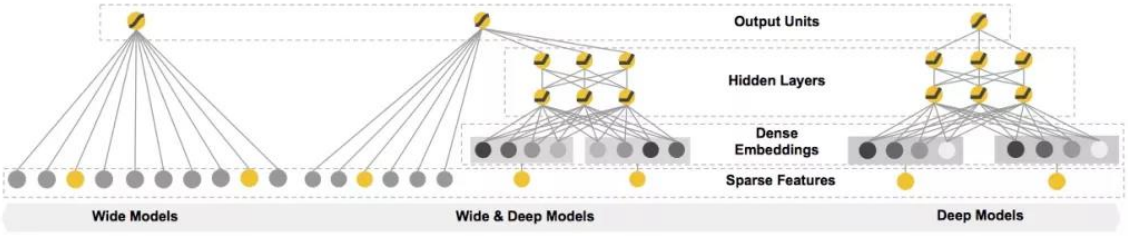


Figure 1: The spectrum of Wide & Deep models.

1.3 Cross-product transformation

Wide 中不断提到这样一种变换用来生成组合特征，也必须搞懂才行哦。它的定义如下：

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

k 表示第 k 个组合特征。i 表示输入 X 的第 i 维特征。C_ki 表示这个第 i 维度特征是否要参与第 k 个组合特征的构造。d 表示输入 X 的维度。那么到底有哪些维度特征要参与构造组合特征那？这个是你之前自己定好的，在公式中没有体现。

饶了一大圈，整这么一个复杂的公式，其实就是我们之前一直在说的 one-hot 之后的组合特征：仅仅在输入样本 X 中的特征 gender=female 和特征 language=en 同时为 1，新的组合特征 AND(gender=female, language=en)才为 1。所以只要把两个特征的值相乘就可以了。

Cross-product transformation 可以在二值特征中学习到组合特征，并且为模型增加非线性

2. Wide & Deep Model

Memorization：之前大规模稀疏输入的处理是：通过线性模型 + 特征交叉。所带来的 Memorization 以及记忆能力非常有效和可解释。但是 Generalization（泛化能力）需要更多的人工特征工程。

Generalization：相比之下，DNN 几乎不需要特征工程。通过对低纬度的 dense embedding 进行组合可以学习到更深层次的隐藏特征。但是，缺点是有点 over-generalize（过度泛化）。推荐系统中表现为：会给用户推荐不是那么相关的物品，尤其是 user-item 矩阵比较稀疏并且是 high-rank（高秩矩阵）

两者区别：Memorization 趋向于更加保守，推荐用户之前有过行为的 items。相比之下，generalization 更加趋向于提高推荐系统的多样性（diversity）。

Wide & Deep:Wide & Deep 包括两部分：线性模型 + DNN 部分。结合上面两者的优点，平衡 memorization 和 generalization。原因：综合 memorization 和 generalizatio 的优点，服务于推荐系统。相比于 wide-only 和 deep-only 的模型，wide & deep 提升显著（这么比较脸是不是有点大）

2.1 推荐系统

2.1.1 介绍推荐系统分为两种： CF-Based（协同过滤）、Content-Based（基于内容的推荐）

协同过滤 (collaborative filtering) 就是指基于用户的推荐，用户 A 和 B 比较相似，那么 A 喜欢的 B 也可能喜欢

基于内容推荐是指物品 item1 和 item2 比较相似，那么喜欢 item1 的用户多半也喜欢 item2

2.1.2 线性模型

大规模的在线推荐系统中，logistic regression 应用非常广泛，因为其 简单、易扩展、可解释性。

LR 的输入多半是二值化后的 one-hot 稀疏特征。Memorization 可以通过在稀疏特征上 cross-product transformations 来实现，比如：AND(user_installed_app=QQ, impression_app=WeChat)，当特征 user_installed_app=QQ，和特征 impression_app=WeChat 取值都为 1 的时候，组合特征 AND(user_installed_app=QQ, impression_app=WeChat) 的取

值才为 1，否则为 0。
推荐系统可以看成是一个 search ranking 问题，根据 query 得到 items 候选列表，然后对 items 通过 ranking 算法排序，得到最终的推荐列表。Wide & Deep 模型是用来解决 ranking 问题的。
如果仅仅使用线性模型：无法学习到训练集中没有的 query-item 特征组合。Embedding-based Model 可以解决这个问题。
2.1.3 Embedding-Based
FM 和 DNN 都算是这样的模型，可以在很少的特征工程情况下，通过学习一个低纬度的 embedding vector 来学习训练集中从未见过的组合特征。
FM 和 DNN 的缺点在于： 当 query-item 矩阵是稀疏并且是 high-rank 的时候（比如 user 有特殊的爱好，或 item 比较小众），很难非常效率的学习出低维度的表示。这种情况下，大部分的 query-item 都没有什么关系。但是 dense embedding 会导致几乎所有的 query-item 预测值都是非 0 的，这就导致了推荐过度泛化，会推荐一些不那么相关的物品。
相反，linear model 却可以通过 cross-product transformation 来记住这些 exception rules，而且仅仅使用了非常少的参数。
总结一下：线性模型无法学习到训练集中未出现的组合特征；FM 或 DNN 通过学习 embedding vector 虽然可以学习到训练集中未出现的组合特征，但是会过度泛化。
Wide & Deep Model 通过组合这两部分，解决了这些问题。
2.1.4 工作流程
总的来说， 推荐系统 = Retrieval + Ranking
推荐系统工作流程如下：

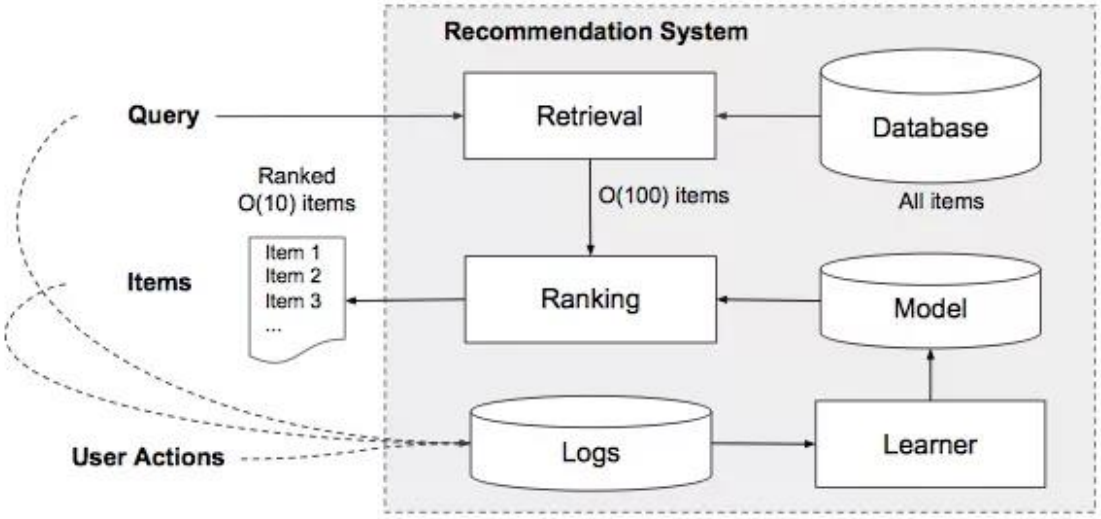


Figure 2: Overview of the recommender system.

想象这样一个实际情况：我们打开 Google APP store，首页展示给我们一些 APP，我们点击或者下载或者购买了其中一个 APP。 在这样一个流程中，推荐系统是如何工作的那？
我们对比上面的图一点点来说：
Query:当我们打开 APP Store 的时候，就产生了一次 Query，它包含两部分的特征：User features, contextual features。UserFeatures 包括性别、年龄等人口统计特征，ContextualFeatures 包括设备、时间等上下文特征。
Items:APP store 接着展示给我们一系列的 app，这些 app 就是推荐系统针对我们的 Query 给出的推荐。这个也被叫做 impression。
User Actions:针对推荐给你的任何一个 APP，我们都可以点击、下载、购买等操作。也就是说推荐给你的 APP，你产生了某种行为。这不正是我们的最终目的吗！

Logs:Logs = Query + Impression + UserAction 查询、展示列表、操作会被记录到 logs 中作为训练数据 给 Learner 来学习。
Retrieval：假如让你来想一个最简单的推荐系统，针对这一次 Query，来给出推荐列表。你能想到的最简单，最暴力的做法是什么那？
给数据库中所有的 APP 都打一个分数，然后按照分数从高到低返回前 N 个（比如说前 100 个）
但是有个问题，这样数据库中的 APP 实在是太多了，为了保证响应时间，这样做太慢了！
Retrieval 就是用来解决这个问题的。它会利用机器学习模型和一些人为定义的规则，来返回最匹配当前 Query 的一个小的 items 集合，这个集合就是最终的推荐列表的候选集。
Ranking：今天的主角 Wide&Deep Model 就是用来做这个事情的啦。
前面 Learner 学习到了一个 Model，利用这个 Model 对 Retrieval 给出的候选集 APP 打分！并按照打分从高到低来排序，并返回前 10 个 APP 作为最终的推荐结果展示给用户。
Retrieval system 完了之后，就是 Ranking system。Retrieval 减小了候选 items 池，Ranking system 要做的就是对比当前的 Query，对 Candidate pool 里面的所有 item 打分！
得分 score 表示成 P(y|x), 表示的是一个条件概率。y 是 label，表示 user 可以采取的 action，比如点击或者购买。x 表示输入，特征包括：
User features（年龄、性别、语言、民族等）
Contextual features(上下文特征：设备，时间等)
Impression features（展示特征：app age、app 的历史统计信息等）
2.2 Wide Part

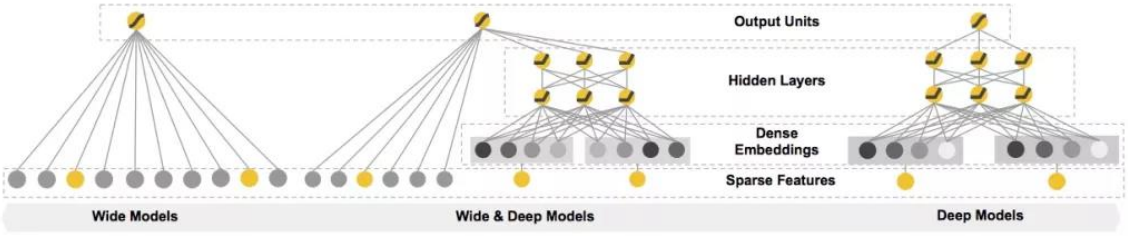


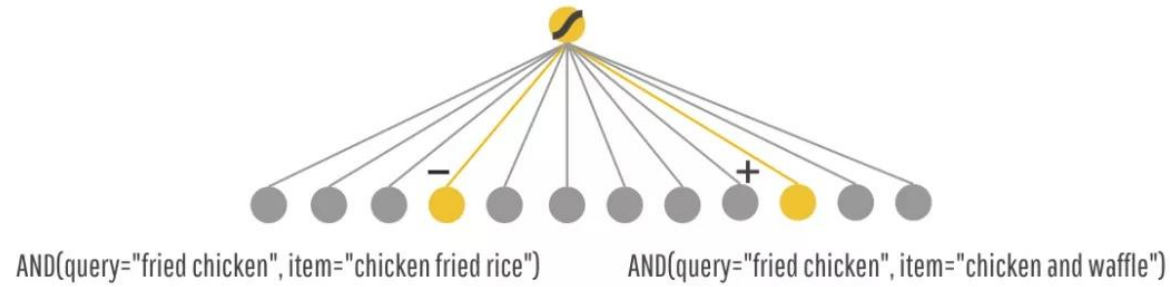
Figure 1: The spectrum of Wide & Deep models.

Wide 部分其实是一个广义的线性模型，就是一个泛化的形如 $y = w^T x + b$ 的线性模型，就如上图左边部分所展示的一样。y 是我们要预测的结果，x 是特征，它是一个 d 维的向量 $x = [x_1, x_2, \cdots, x_d]$。这里的 d 是特征的数量。同样 w 也是一个 d 维的权重向量 $w = [w_1, w_2, \cdots, w_d]$，b 呢则是偏移量。这些我们在之前线性回归的模型当中曾经都介绍过，大家应该也都不陌生。
特征包含两个部分，一种是原始数据直接拿过来的数据 raw input，另外一种是我们经过特征转化之后得到的特征 cross-product transformation。最重要的一种特征转化方式就是交叉组合，交叉组合可以定义成如下形式：

$$\phi_k(x) = \Pi_{i=1}^{d} x_i^{c_{ki}} \quad c_{ki} \in \{0, 1\}$$

这里的 c_{ki} 是一个 bool 型的变量，表示的是第 i 个特征的第 k 种转化函数 ϕ_k 的结果。由于使用的是乘积的形式，只有所有项都为真，最终的结果才是 1，否则是 0。比如"AND(gender=female,language=en)"这就是一个交叉特征，只有当用户的性别为女，并且使用的语言为英文同时成立，这个特征的结果才会是 1。通过这种方式我们可以捕捉到特征之间的交互，以及为线性模型加入非线性的特征。

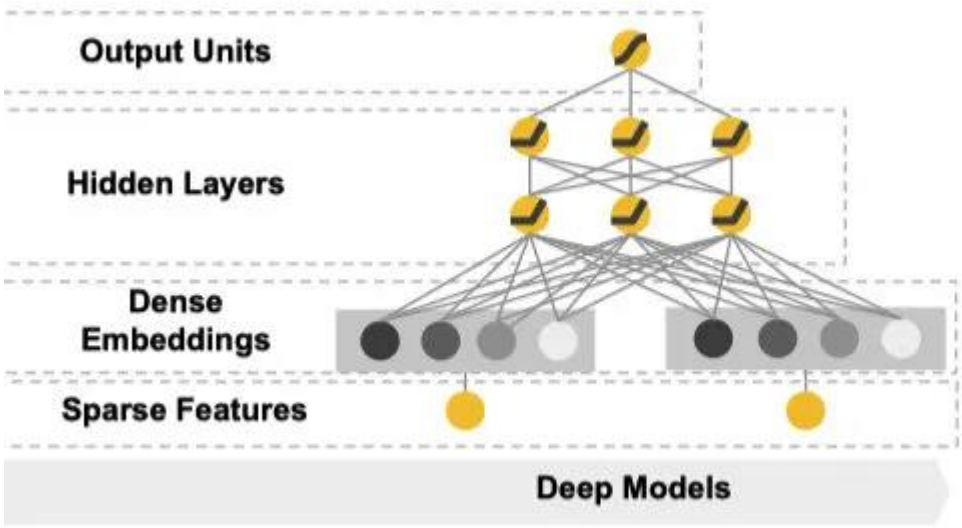
接下来我们用同一个例子来说明：你给 model 一个 query（你想吃的美食），model 返回给你一个美食，然后你购买 / 消费了这个推荐。 也就是说，推荐系统其实要学习的是这样一个条件概率： P(consumption | query, item)



Wide Part 可以对一些特例进行 memorization。比如 AND(query="fried chicken", item="chicken fried rice") 虽然从字符角度来看很接近，但是实际上完全不同的东西，那么 Wide 就可以记住这个组合是不好的，是一个特例，下次当你再点炸鸡的时候，就不会推荐给你鸡肉炒米饭了。

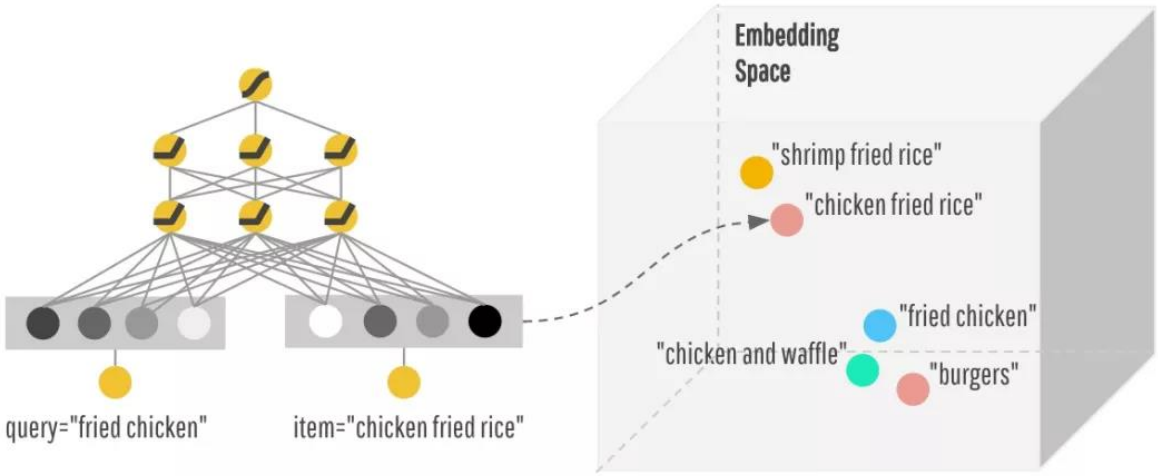
2.3 Deep Part

Deep 部分是一个前馈神经网络，也就是上图当中的右侧部分。



我们观察一下这张图会发现很多细节，比如它的输入是一个 sparse 的 feature，可以简单理解成 multihot 的数组。这个输入会在神经网络的第一层转化成一个低维度的 embedding，然后神经网络训练的是这个 embedding。这个模块主要是被设计用来处理一些类别特征，比如说 item 的类目，用户的性别等等。和传统意义上的 one-hot 方法相比，embedding 的方式用一个向量来表示一个离散型的变量，它的表达能力更强，并且这个向量的值是让模型自己学习的，因此泛化能力也大大提升。这也是深度神经网络当中常见的做法。

继续套用上面的例子。

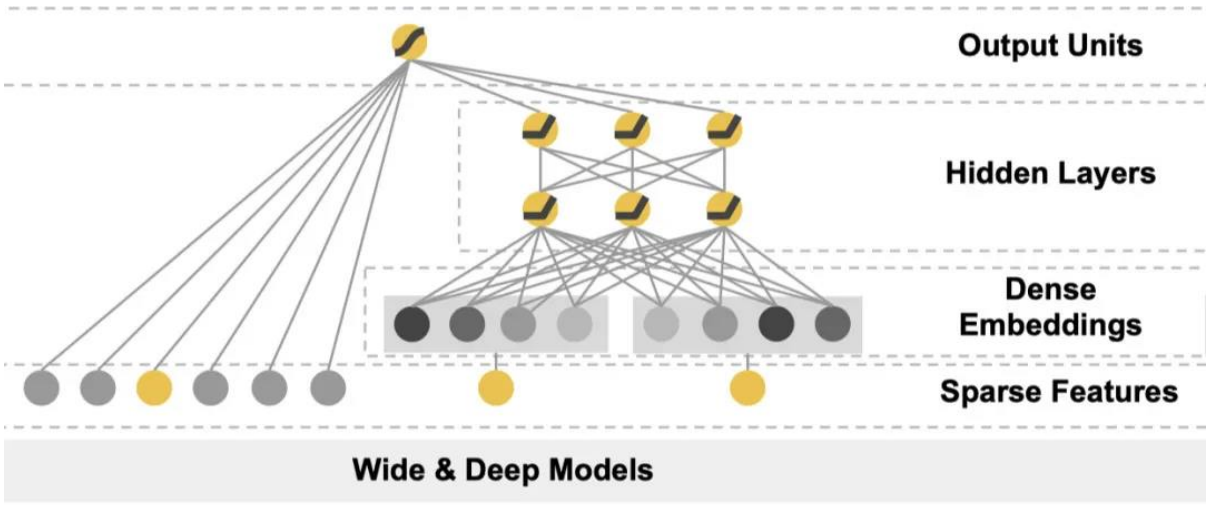


Deep Part 通过学习一个低纬度的 dense representation（也叫做 embedding vector）对于每一个 query 和 item，来 泛化 给你推荐一些字符上看起来不那么相关，但是你可能也是需要的。比如说：你想要炸鸡，Embedding Space 中，炸鸡和汉堡很接近，所以也会给你推荐汉堡。

Embedding vectors 被随机初始化，并根据最终的 loss 来反向训练更新。这些低维度的 dense embedding vectors 被作为第一个隐藏层的输入。隐藏层的激活函数通常使用 ReLU。

2.4 模型训练

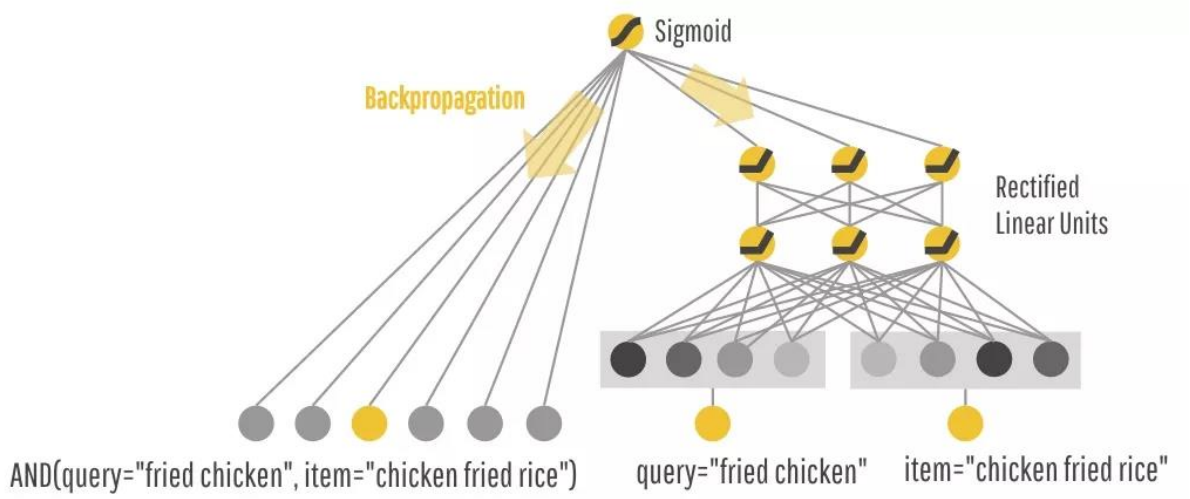
Wide 部分和 Deep 部分都有了之后，通过加权的方式合并在一起。这也就是上图当中的中间部分。



最上层输出之前其实是一个 sigmoid 层或者是一个 linear 层，就是一个简单的线性累加。英文叫做 joint，paper 当中还列举了 joint 和 ensemble 的区别，对于 ensemble 模型来说，它的每一个部分是独立训练的。而 joint 模型当中的不同部分是联合训练的。ensemble 模型当中的每一个部分的参数是互不影响的，但是对于 joint 模型而言，它当中的参数是同时训练的。

这样带来的结果是，由于训练对于每个部分是分开的，所以每一个子模型的参数空间都很大，这样才能获得比较好的效果。而 joint 训练的方式则没有这个问题，我们把线性部分和深度学习的部分分开，可以互补它们之间的缺陷，从而达到更好的效果，并且也不用人为地扩大训练参数的数量。

再次回到那个例子。原始的稀疏特征，在两个组件中都会用到，比如 query="fried chicken" item="chicken fried rice":



在训练的时候，根据最终的 loss 计算出 gradient，反向传播到 Wide 和 Deep 两部分中，分别训练自己的参数。也就是说，两个模块是一起训练的，注意这不是模型融合。
Wide 部分中的组合特征可以 记住 那些稀疏的，特定的 rules
Deep 部分通过 Embedding 来 泛化 推荐一些相似的 items
Wide 模块通过组合特征可以很效率的学习一些特定的组合，但是这也导致了他并不能学习到训练集中没有出现的组合特征。所幸，Deep 模块弥补了这个缺点。另外，因为是一起训练的，wide 和 deep 的 size 都减小了。wide 组件只需要填补 deep 组件的不足就行了，所以需要比较少的 cross-product feature transformations，而不是 full-size wide Model。
论文中的实现：
训练方法是用 mini-batch stochastic optimization。
Wide 组件是用 FTRL（Follow-the-regularized-leader） + L1 正则化学习。
Deep 组件是用 AdaGrad 来学习。
3. 系统实现

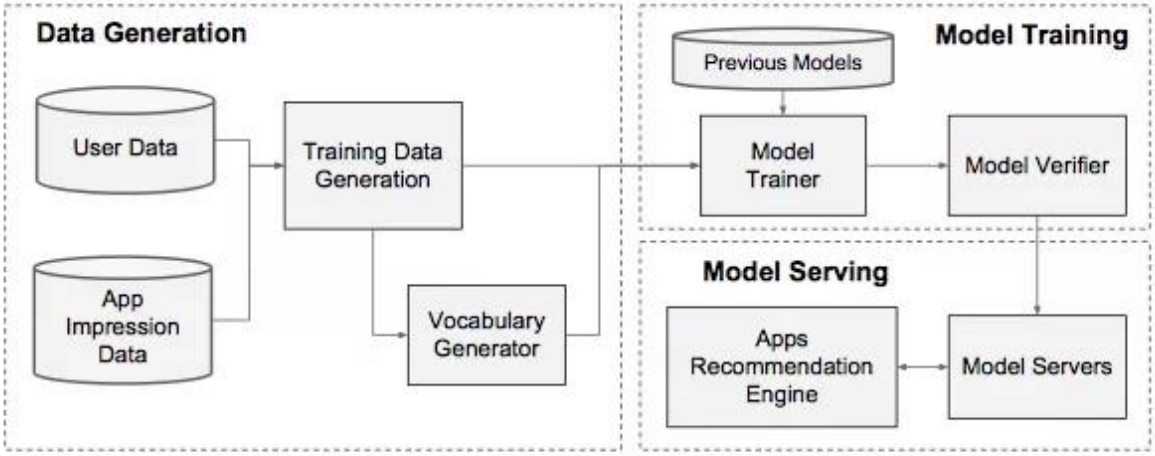


Figure 3: Apps recommendation pipeline overview.

3.1 训练数据生成
请大家一定格外的关注训练数据到底是什么？这对于理解推荐系统到底是怎么回事很重要。先给出结论：
一次展示中的一个 Item 就是一条样本。
样本的 label 要根据实际的业务需求来定，比如 APP Store 中想要提高 APP 的下载率，那么就以这次展示的这个 Item 中用户有没有下载，作为 label。下载了 label 为 1，否则为 0.说白了，模型需要预测，在当前 Query 的条件下，对于这个 Item，用户下载的条件概率。

离散特征 map 成 id
过滤掉出现次数少于设定阈值的离散特征取值，然后把这些全部 map 成一个 ID。离散特征取值少，就直接编号。多的话可能要 Hash
连续特征通过分位数规范化到 [0,1]
先把所有的值分成 n 份，那么属于第 i 部分的值规范化之后的值为 (i - 1)/(n - 1)。
3.2 模型训练

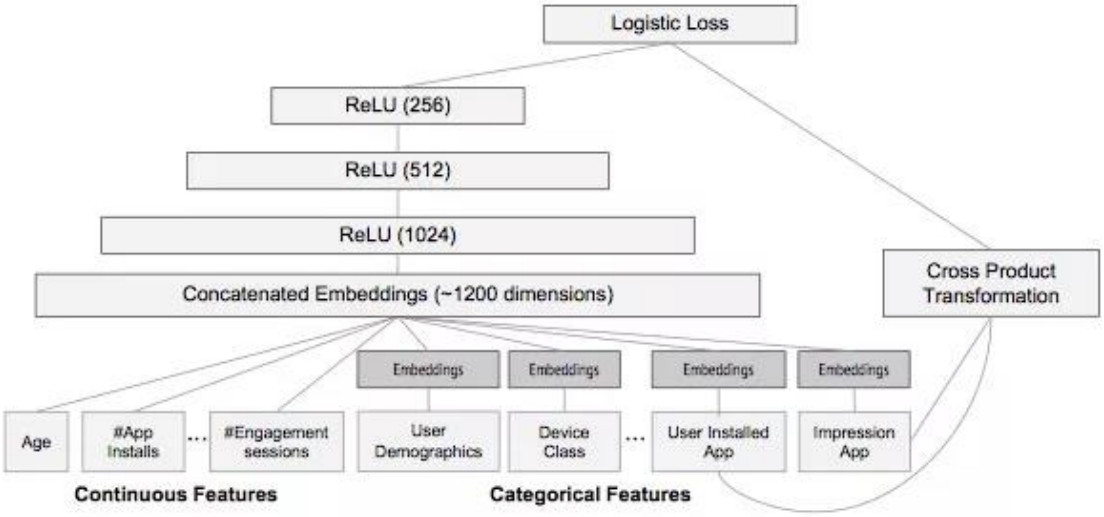


Figure 4: Wide & Deep model structure for apps recommendation.

Deep 部分使用的特征：连续特征 Embedding 后的离散特征，Item 特征
Wide 部分使用的特征：Cross Product Transformation 生成的组合特征
但是，官方给出的示例代码中，Wide 部分还使用了离散特征（没有 one-hot）。也有大佬说不用特征交叉效果也很好，这个大家在实际项目中就以实验为准吧。
每当有新的数据到达的时候，就要重新训练。如果每次都从头开始会非常耗时，Google 给出的解决办法是：实现了 warm-starting system，它可以用之前模型的 embeddings 和 线性模型的 weights 来初始化新的模型。
Embedding 维度大小的建议：Wide&Deep 的作者指出，从经验上来讲 Embedding 层的维度大小可以用如下公式来确定：
$k\sqrt[4]{n}$，n 是原始维度上特征不同取值的个数；k 是一个常数，通常小于 10.
3.3 线上使用
模型被部署之后。每一次请求，服务器会收到一系列的 app 候选集（从 app retrival system 输出的）以及 user features（用于为每一个 app 打分）。然后，模型会把 APP 按照 score 排序，并展示给 user，按照这个顺序展示。score 就是对于 wide & deep 模型的一次 forward pass。为了控制每一次 request 响应时间在 10ms 内，引入了并行化技术。将 app 候选集分成多个小的 batches，并行化预测 score。
4. 适用范围
Wide & Deep Model 适用于输入非常稀疏的大规模分类或回归问题。比如推荐系统、search、ranking 问题。输入稀疏通常是由离散特征有非常非常多个可能的取值造成的，one-hot 之后维度非常大。
5. 优缺点
缺点：Wide 部分还是需要人为的特征工程。
优点：实现了对 memorization 和 generalization 的统一建模。
6. 代码实践

代码放到 github 上了： https://github.com/gutouyu/ML_CIA/tree/master/Wide%26Deep
数据集：https://archive.ics.uci.edu/ml/machine-learning-databases/adult 代码主要包括两部分：Wide Linear Model 和 Wide & Deep Model。
数据集长这样，最后一行是 label，预测收入是否超过 5 万美元，二分类问题。

Column Name	Type	Description
age	Continuous	The age of the individual
workclass	Categorical	The type of employer the individual has (government, military, private, etc.).
fnlwgt	Continuous	The number of people the census takers believe that observation represents (sample weight). Final weight will not be used.
education	Categorical	The highest level of education achieved for that individual.
education_num	Continuous	The highest level of education in numerical form.
marital_status	Categorical	Marital status of the individual.
occupation	Categorical	The occupation of the individual.
relationship	Categorical	Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
race	Categorical	Amer-Indian-Eskimo, Asian-Pac- Islander, Black, White, Other.
gender	Categorical	Female, Male.
capital_gain	Continuous	Capital gains recorded.
capital_loss	Continuous	Capital Losses recorded.
hours_per_week	Continuous	Hours worked per week.
native_country	Categorical	Country of origin of the individual.
income_bracket	Categorical	">50K" or "<=50K", meaning whether the person makes more than $50,000 annually.

6.1 Wide Linear Model
离散特征 处理分为两种情况：知道所有的不同取值，而且取值不多。
tf.feature_column.categorical_column_with_vocabulary_list 不知道所有不同取值，或者取值非常多。tf.feature_column.categorical_column_with_hash_bucket

```
## 3.1 Base Categorical Feature Columns
# 如果我们知道所有的取值，并且取值不是很多
relationship = tf.feature_column.categorical_column_with_vocabulary_list(
    'relationship', [
        'Husband', 'Not-in-family', 'Wife', 'Own-child', 'Unmarried',
        'Other-relative'
    ]
)
# 如果不知道有多少取值
occupation = tf.feature_column.categorical_column_with_hash_bucket(
    'occupation', hash_bucket_size=1000
)
```

原始连续特征：tf.feature_column.numeric_column

```
# 3.2 Base Continuous Feature Columns
age = tf.feature_column.numeric_column('age')
education_num = tf.feature_column.numeric_column('education_num')
capital_gain = tf.feature_column.numeric_column('capital_gain')
capital_loss = tf.feature_column.numeric_column('capital_loss')
hours_per_week = tf.feature_column.numeric_column('hours_per_week')
```

规范化到 [0,1] 的连续特征：tf.feature_column.bucketized_column

```
# 3.2.1 连续特征离散化
# 之所以这么做是因为：有些时候连续特征和label之间不是线性的关系。
# bucketization 装桶
# 10个边界，11个桶
age_buckets = tf.feature_column.bucketized_column(
    age, boundaries=[18, 25, 30, 35, 40, 45, 50, 55, 60, 65])
```

组合特征 / 交叉特征：tf.feature_column.crossed_column

```
# 3.3 组合特征/交叉特征
education_x_occupation = tf.feature_column.crossed_column(
    ['education', 'occupation'], hash_bucket_size=1000)

age_buckets_x_education_x_occupation = tf.feature_column.crossed_column(
    [age_buckets, 'education', 'occupation'], hash_bucket_size=1000
)
```

组装模型：这里主要用了离散特征 + 组合特征

```
# 4. 模型
"""
之前的特征:
1. CategoricalColumn
2. NumericalColumn
3. BucketizedColumn
4. CrossedColumn
这些特征都是FeatureColumn的子类，可以放到一起
"""
base_columns = [
    education, marital_status, relationship, workclass, occupation,
    age_buckets,
]

crossed_column = [
    tf.feature_column.crossed_column(
        ['education', 'occupation'], hash_bucket_size=1000
    ),
    tf.feature_column.crossed_column(
        [age_buckets, 'education', 'occupation'], hash_bucket_size=1000
    )
]

model_dir = "./model/wide_component"
model = tf.estimator.LinearClassifier(
    model_dir=model_dir, feature_columns=base_columns + crossed_column
)
```

训练 & 评估：

```
# 5. Train & Evaluate & Predict
model.train(input_fn=lambda: input_fn(data_file=train_file, num_epochs=1, shuffle=True, batch_size=512))
results = model.evaluate(input_fn=lambda: input_fn(val_file, 1, False, 512))
for key in sorted(results):
    print("{0:20}: {1:.4f}".format(key, results[key]))
```

运行截图：

```
Parsing ./data/adult.data
accuracy            : 0.8434
accuracy_baseline   : 0.7592
auc                 : 0.8940
auc_precision_recall: 0.7230
average_loss        : 0.3402
global_step         : 192.0000
label/mean          : 0.2408
loss                : 173.0865
prediction/mean     : 0.2405
```

6.2 Wide & Deep Model

Deep 部分用的特征： 未处理的连续特征 + Embedding(离散特征)在 Wide 的基础上，增加 Deep 部分：离散特征 embedding 之后，和连续特征串联。

```
# 3. The Deep Model: Neural Network with Embeddings
"""
1. Sparse Features -> Embedding vector -> 串联(Embedding vector, 连续特征) -> 输入到Hidden Layer
2. Embedding Values随机初始化
3. 另外一种处理离散特征的方法是：one-hot or multi-hot representation. 但是仅仅适用于维度较低的，embedding是更加通用的做法
4. embedding_column(embedding);indicator_column(multi-hot);
"""
deep_columns = [
    age,
    education_num,
    capital_gain,
    capital_loss,
    hours_per_week,
    tf.feature_column.indicator_column(workclass),
    tf.feature_column.indicator_column(education),
    tf.feature_column.indicator_column(marital_status),
    tf.feature_column.indicator_column(relationship),

    # To show an example of embedding
    tf.feature_column.embedding_column(occupation, dimension=8)
]
```

组合 Wide & Deep：DNNLinearCombinedClassifier

```
# 4. Combine Wide & Deep
model = tf.estimator.DNNLinearCombinedClassifier(
    model_dir = model_dir,
    linear_feature_columns=base_columns + crossed_columns,
    dnn_feature_columns=deep_columns,
    dnn_hidden_units=[100,50]
)
```

训练 & 评估：

```
for n in range(train_epochs // epochs_per_eval):
    model.train(input_fn=lambda: input_fn(train_file, epochs_per_eval, True, batch_size))
    results = model.evaluate(input_fn=lambda: input_fn(
        test_file, 1, False, batch_size))

    # Display Eval results
    print("Results at epoch {0}".format((n+1) * epochs_per_eval))
    print('-'*30)

    for key in sorted(results):
        print("{0:20}: {1:.4f}".format(key, results[key]))
```

运行结果：

```
Parsing ./data/adult.test
Results at epoch 2
------------------------------
accuracy            : 0.8486
accuracy_baseline   : 0.7638
auc                 : 0.8931
auc_precision_recall: 0.7550
average_loss        : 0.3385
global_step         : 8145.0000
label/mean          : 0.2362
loss                : 13.5087
prediction/mean     : 0.2346
Parsing ./data/adult.data
Parsing ./data/adult.test
Results at epoch 4
------------------------------
accuracy            : 0.8457
accuracy_baseline   : 0.7638
auc                 : 0.8955
auc_precision_recall: 0.7421
average_loss        : 0.3433
global_step         : 9774.0000
label/mean          : 0.2362
loss                : 13.6987
prediction/mean     : 0.2429
Parsing ./data/adult.data
Parsing ./data/adult.test
Results at epoch 6
------------------------------
accuracy            : 0.8476
accuracy_baseline   : 0.7638
auc                 : 0.8954
auc_precision_recall: 0.7495
average_loss        : 0.3367
global_step         : 11403.0000
label/mean          : 0.2362
loss                : 13.4349
prediction/mean     : 0.2404
```

Reference
1 Wide & Deep Learning for Recommender Systems
2 Google AI Blog Wide & Deep Learning: Better Together with TensorFlow
https://ai.googleblog.com/2016/06/wide-deep-learning-better-together-with.html
3 TensorFlow Linear Model Tutorial https://www.tensorflow.org/tutorials/wide
4 TensorFlow Wide & Deep Learning Tutorial
https://www.tensorflow.org/tutorials/wide_and_deep
5 TensorFlow 数据集和估算器介绍 http://developers.googleblog.cn/2017/09/tensorflow.html
6 absl

14、推荐系统评估(从用户\内容提供方\网站角度)

解析：

从用户角度： 满足需求 获取快乐 扩展视野
内容提供方： 获取长尾流量 获得互动和认可 获得收益
网站： 留住用户 实现商业目标

15、推荐系统常用评估指标

解析：

准确性
满意度
覆盖率
多样性
新颖性
惊喜度
信任度
实时性
鲁棒性
可扩展性
商业目标
用户留存

16、推荐系统准确性(学术界)的评估函数

- 评分预测
  - RMSE: 均方根误差

$$RMSE = \sqrt{\frac{\sum_{u,ieT}(r_{ui} - \hat{r}_{ui})^2}{|T|}}$$

  - MAE: 平均绝对误差

$$MAE = \frac{\sum_{u,i \in T} |r_{ui} - \hat{r}_{ui}|}{|T|}$$

- topN推荐
  - Recall: 召回率

$$Recall = \frac{\sum_{\mu \in U} |R(\mu) \bigcap T(\mu)|}{\sum_{\mu \in U} |T(\mu)|}$$

  - Precision: 准确率

$$Precision = \frac{\sum_{\mu \in U} |R(u) \bigcap T(u)|}{\sum_{\mu \in U} |R(u)|}$$

17、推荐系统多样性&新颖性&惊喜性

多样性: 推荐列表中两两物品的不相似性

新颖性: 未曾关注的类别\作者; 推荐结果的平均流行度

惊喜性: 历史不相似(惊)但很满意(喜)

18、解释 Exploitation & Exploration

Exploitation: 选择现在可能最佳的方案(可能会让推荐的结果范围越来越小)
Exploration: 选择现在不确定的一些方案,但未来可能会有高收益的方案(会让推荐的面越来越广)
在做两类决策的过程中,不断更新对所有决策的不确定性的认知,优化长期的目标函数

19、Bandit 算法-原理

- Thompson Sampling: 每个臂维护一个beta(wins,lose)分布,每次用现有的beta分布产生一个随机数,选择随机数最大的臂.
  - 假设每个臂是否产生收益,其背后有一个概率分布,产生收益的概率为 `P`
  - 我们不断地实验,去估计出一个置信度较高的"概率 `P` 的概率分布"就能近似解决这个问题了
  - 假设概率 `P` 的概率分布符合 `beta(wins, lose)`分布 ,它有两个参数: `wins` , `lose`
  - 每个臂都维护一个beta分布的参数,每次试验后,选中一个臂,摇一下,有收益则该臂的 `wins` 增加1**,否则该臂的 `lose` 增加1**
  - 每次选择臂的方式是: 用每个臂现有的beta分布产生一个随机数b,选择所有臂产生的随机数中最大的那个臂去摇.
- Upper Confidence Bound(置信区间上界): 均值越大,标准差越小,被选中的概率会越来越大
  - 初始化: 先对每一个臂都试一遍
  - 按照如下公式计算每个臂的分数,然后选择分数最大的臂作为选择

$$\bar{x}_j(t) + \sqrt{\frac{2 \ln t}{T_{j,t}}}$$

  - 观察选择结果,更新t和$T_{j,t}$,其中加号前面是这个臂到目前的 收益均值 ,后面的叫做 `bonus` ,本质上是均值的标准差,t是目前的试验次数,$T_{j,t}$是这个臂被试次数
- Epsilon-Greedy: 以1-epsilon的概率选取当前收益最大的臂,以epsilon的概率随机选取一个臂. `epsilon` 的值可以控制对 `Exploit` 和 `Explore` 的偏好程度。越接近0，越靠近 `Exploit`
  - 选择一个(0,1)之间较小的数作为 `epsilon`
  - 每次以概率 `epsilon` 做一件事: 所有臂中随机选择一个
  - 每次以概率 `1-epsilon` 选择截止到目前,平均收益最大的那个臂
- 朴素Bandit算法: 先随机试若干次,计算每个臂的平均收益,一直选均值最大的那个臂.

20、什么是多层重叠试验框架?

保留单层实验框架易用,快速的优点的同时,增加可扩展性,灵活性,健壮性.

核心思路: 将参数划分到 N 个子集,每个子集都关联一个实验层,每个请求会被 N 个实验处理,同一个参数不能出现在多个层中.

21、GCN 方法分为哪两类?

基于谱的方法，通过从图信号处理的角度引入滤波器来定义图卷积，其中图卷积运算被解释为从图信号中去除噪声 ，包括 Spectral CNN、Chebyshev Spectral CNN (ChebNet)、First order of ChebNet (1stChebNet) 和 Adaptive Graph Convolution Network (AGCN)

基于空间的方法将图卷积表征为聚合来自近邻的特征信息。虽然 GCN 在节点级别上运行，但是图池化模块可以与 GCN 层交替，将图粗粒化为高级子结构 ，包括 Recurrent-based Spatial GCN 和 Composition Based Spatial GCN

22、什么是图卷积神经网络?

图卷积神经网络（Graph Convolutional Network）是一种能对图数据进行深度学习的方法

图卷积算子：

$$h_i^{(l+1)} = \sigma(\sum_{j \in N_i} \frac{1}{c_{ij}} h_j^l w_{R_j}^l)$$

h_i^l 节点i在第l层的特征表达

c_{ij} 归一化因子

N_i 节点i的邻居

R_i 节点i的类型

w_{R_i} R_i类型节点的变换权重参数

23、如何理解图卷积算法?

解析：

1）发射（send）每一个节点将自身的特征信息经过变换后发送给邻居节点。这一步是在对节点的特征信息进行抽取变换

2）接收（receive）每个节点将邻居节点的特征信息聚集起来。这一步是在对节点的局部结构信息进行融合

3）变换（transform）把前面的信息聚集之后做非线性变换，增加模型的表达能力

24、GCN 有哪些特征?

1. GCN 是对卷积神经网络在 graph domain 上的自然推广
2. 它能同时对节点特征信息与结构信息进行端对端学习，是目前对图数据学习任务的最佳选择
3. 图卷积适用性极广，适用于任意拓扑结构的节点与图
4. 在节点分类与边预测等任务上，在公开数据集上效果要远远优于其他方法

25、已知基于数据驱动的机器学习和优化技术在单场景内的 A/B 测试上，点击率、转化率、成交额、单价都取得了不错的效果。 但是，目前各个场景之间是完全独立优化的，这样会带来哪些比较严重的问题 ？

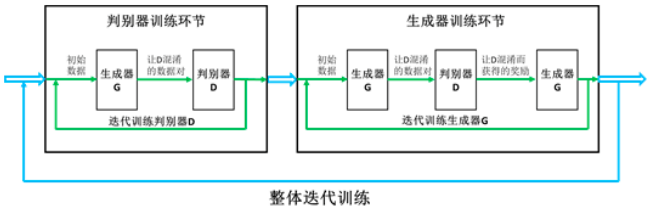
1. 不同场景的商品排序仅考虑自身，会导致 用户的购物体验是不连贯或者雷同的 。例如：从冰箱的详情页进入店铺，却展示手机；各个场景都展现趋同，都包含太多的 U2I（点击或成交过的商品）；
2. 多场景之间是博弈（竞争）关系， 期望每个场景的提升带来整体提升这一点是无法保证的 。很有可能一个场景的提升会导致其他场景的下降，更可怕的是某个场景带来的提升甚至小于其他场景更大的下降。这并非是不可能的，这种情况下，单场景的 A/B 测试就显得没那么有意义，单场景的优化也会存在明显的问题。

26、什么是多场景联合排序算法 ？

多场景联合排序算法，旨在提升整体指标。我们将多场景的排序问题看成一个完全合作的、部分可观测的多智能体序列决策问题，利用 Multi-Agent Reinforcement Learning 的方法来尝试着对问题进行建模。该模型以各个场景为 Agent，让各个场景不同的排序策略共享同一个目标，同时在一个场景的排序结果会考虑该用户在其他场景的行为和反馈。这样使得各个场景的排序策略由独立转变为合作与共赢。由于我们想要使用用户在所有场景的行为，而 DRQN 中的 RNN 网络可以记住历史信息，同时利用 DPG 对连续状态与连续动作空间进行探索，因此我们算法取名 MA-RDPG

27、IRGAN 框架具体是什么?

IRGAN 的训练数据必须满足一定的格式要求。比如，每条训练数据要包括一个查询语句 query 以及多个（个数大于 2）与查询相关的网页数据。此外，每条训练数据还要在多个网页数据中标明哪个与本条 query 最为相关。记与当前查询最为相关的网页为 doc_+，其他网页信息为 doc_-，那么训练集的格式即为 (query, doc_+, doc_-, doc_-, … ,doc_-)。



如图，IRGAN 的训练过程可分为判别器的训练和生成器的训练两部分。在训练判别器时，生成器首先会拿到形如 (query, doc_+, doc_-, doc_-, … ,doc_-) 的初始数据。为了最大程度的欺骗判别器，生成器会从诸多 doc_-中挑选一个它认为与查询最相关的网页数据，并将这个 doc_-与 doc_+（实际上最相关的网页数据）拼在一起形成新的数据对 (query, doc_+, doc_-)。判别器会对生成器产生的数据对做出选择：究竟哪个网页与查询最相关。如果判别器选择错误（换言之，被生成器骗到了），那么它将会被惩罚，从而得到训练。

28、推荐页与搜索页特性有什么不同?

搜索带着 query 来的，结果与之相关性越高越好，不用太关心结果的多样性 推荐页用户没有明确的目的，但是有兴趣偏好和对结果的多样性需求，推荐既要准确又要多样化。

29、推荐系统常用的评价指标有哪些?

- 准确率：准确率表示的是分类正确的样本数占样本总数的比例

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$$

- 精确率:表示预测结果中，预测为正样本的样本中，正确预测为正样本的概率

$$Precision = \frac{TP}{TP+FP}$$

- 召回率:表示在原始样本的正样本中，最后被正确预测为正样本的概率

$$Recall = \frac{TP}{TP+FN}$$

- F1值：折中精确率和召回率的结果

$$F_{1-score} = \frac{2*Recall*Precision}{Recall+Precision}$$

- AUC:横轴为FP，纵轴为TP
- ROC:AUC底线的面积
- Hit Ratio:一种常用的衡量召回率的指标

$$HR@K = \frac{Numberd\,fHits@K}{GT}$$

分母是所有的测试集合，分子式每个用户top-K推荐列表中属于测试集合的个数的总和。

- AP:用户u, 给他推荐一些物品，那么 u 的平均准确率定义为：

$$\frac{1}{\Omega_u}\sum_{i\in\Omega_u}\frac{\sum_{j\in\Omega_u}h(p_{uj}-p_{ui})+1}{p_{ui}}$$

Ω_u表示ground-truth的结果，p_{ui}表示物品在推荐列表的位置，$p_{uj} < p_{ui}$表示j在物品列表中排在i之前

- MAP:所有用户 u 的AP再取均值(mean)

$$MAP = \frac{\sum_{u\in U}AP_u}{|U^{te}|}$$

- CG(Cummulative Gain) ,在推荐系统中，CG即将每个推荐结果相关性(relevance)的分值累加后作为整个推荐列表(list)的得分

$$CG_k = \sum_{i=1}^{k} rel_i$$

， rel-i 表示处于位置 i 的推荐结果的相关性，k 表示所要考察的推荐列表的大小

- DCG :CG的一个缺点是没有考虑每个推荐结果处于不同位置对整个推荐效果的影响，例如我们总是希望相关性高的结果应排在前面。显然，如果相关性低的结果排在靠前的位置会严重影响用户体验， 所以在CG的基础上引入位置影响因素，即DCG(Discounted Cummulative Gain), “Discounted”有打折，折扣的意思，这里指的是对于排名靠后推荐结果的推荐效果进行“打折处理”:

$$DCG_k = \sum_{i=1}^{k}\frac{2^{rel_i}-1}{log_2(i+1)}$$

- NDCG:DCG仍然有其局限之处，即不同的推荐列表之间，很难进行横向的评估。 那么不同用户的推荐列表的评估分数就需要进行归一化，也即NDCG(Normalized Discounted Cummulative Gain)。

$$NDCG@k = \frac{\sum_{u \in \mathcal{U}^{te}} NDCG_u@k}{|\mathcal{U}^{te}|}$$

- MRR:

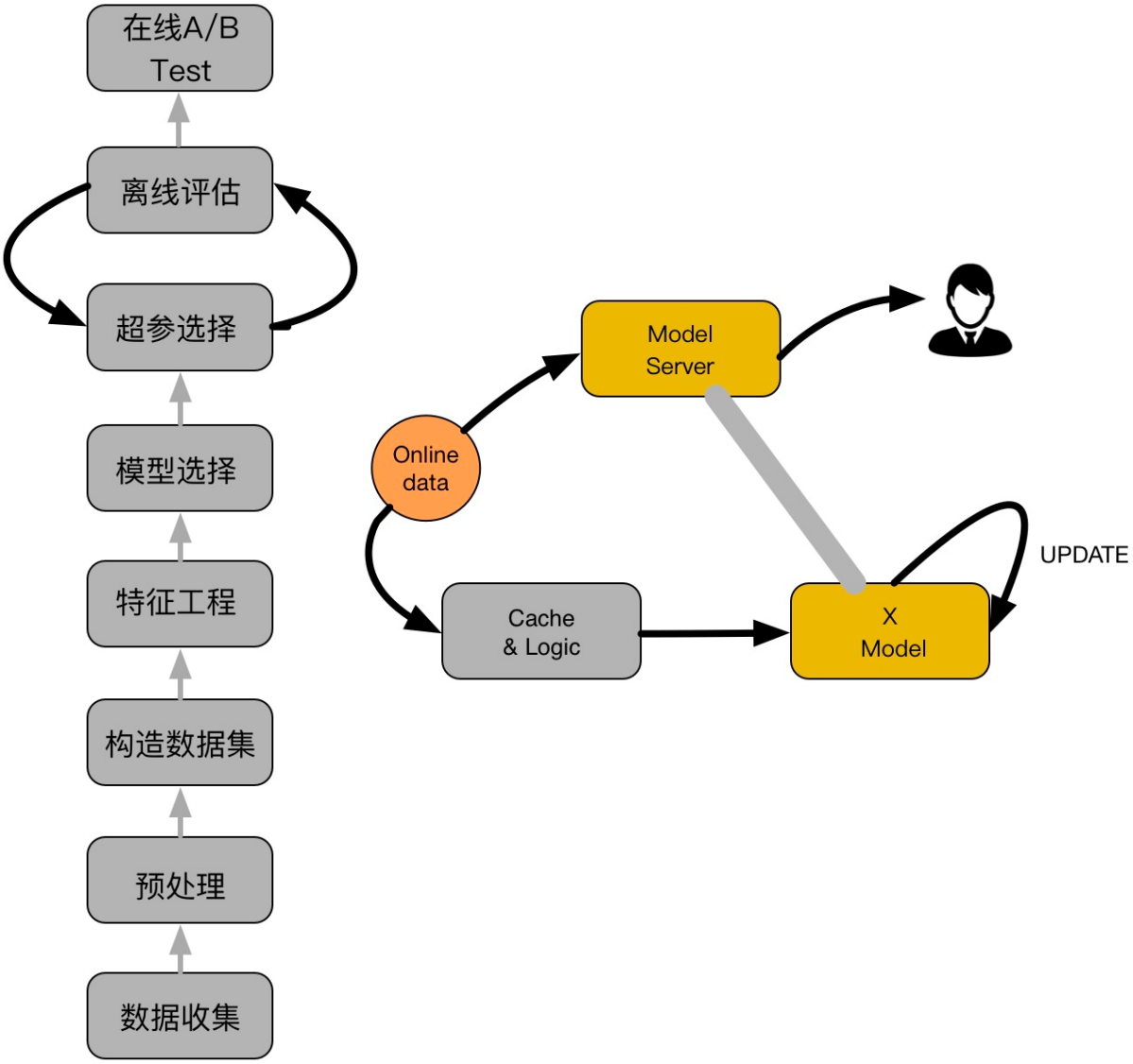
  $$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i}$$

  |Q|是用户的个数，ranki是对于第i个用户，推荐列表中第一个在ground-truth结果中的item所在的排列位置
- ILS:衡量推荐列表多样性的指标

$$ILS(L) = \frac{\sum_{b_i \in L}\sum_{b_j \in L, b_j \neq b_i} S(b_i, b_j)}{\sum_{b_i \in L}\sum_{b_j \in L, b_j \neq b_i} 1}$$

如果S(bi,bj)计算的是i和j两个物品的相似性，如果推荐列表中物品越不相似，ILS越小，那么推荐结果的多样性越好

30、请画出点击率预估流程图，分为那两部分?



主要包括两大部分：离线部分、在线部分，其中离线部分目标主要是训练出可用模型，而在线部分则考

虑模型上线后，性能可能随时间而出现下降，弱出现这种情况，可选择使用 Online-Learning 来在线更新模型

31、怎么理解 FFM？使用 FFM 有哪些需要注意的地方

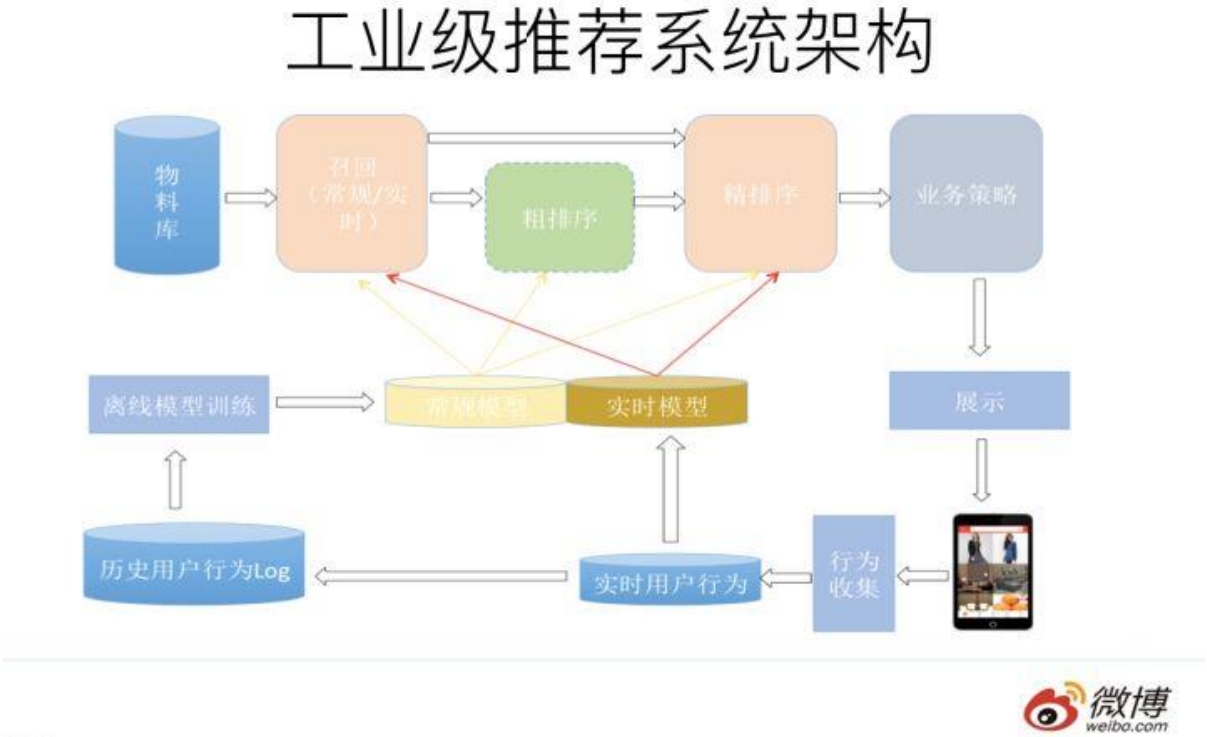
解析：
本题解析作者张俊林，来源链接：https://zhuanlan.zhihu.com/p/59528983
美剧《权力的游戏》中，宗教刺客团体“无面者”成员之一的贾昆，在他和艾莉亚·史塔克分别之际，将一枚特殊的小硬币交给了艾莉亚，并教会艾莉亚一句话，原话是这样的：“Valar Morghulis （凡人皆有一死）
Valar Dohaeris （凡人皆须侍奉）”

但他并未详细解释硬币的用途和这句话的意思。硬币是用来做何用的呢？读完这篇文章你自会知晓答案。至于这句话，前半句好理解，就是字面意思，后半句比较深奥，我也不太懂，我猜大概跟老子说的“天地不仁，以万物为刍狗”意思有点类似吧。

其实仔细想想，上面这句话送给我们见到的各式算法模型也蛮适合，某个算法甲在某年某月被某人乙发明出来，我们发明出来它，当然也没有事先征求它个人的意见。这就仿佛，我们每个人，明知道这世间事，十有九苦，不也没有征求孩子们的个人意愿，他们，也包括我们自己，就这么硬生生地被拉入充斥各色人等的人世间，红尘翻滚，遍尝疾苦，作为补偿，生活也许会回馈些许短暂的欢乐，以让我们继续有动力去面对新的难题。

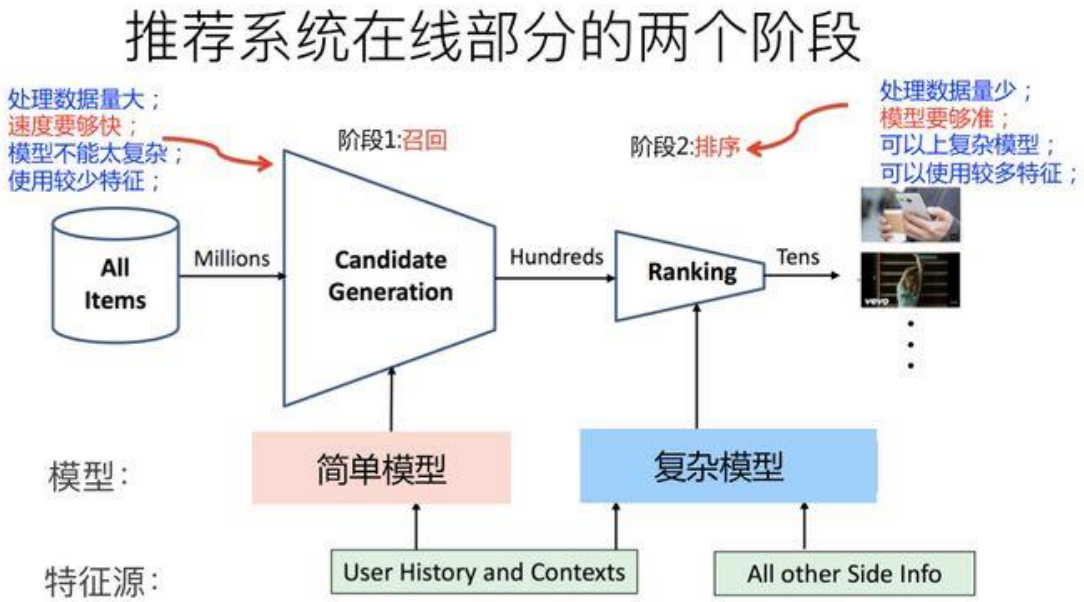
你再想想，算法模型的诞生不也一样吗？它们出生后，如果人们觉得它有用，那它就安心本分地做好自己的工作，好好地服务你我他，如果哪天冒出更活力四射青春逼人的新模型，你觉得它没什么大用了，不也弃之如敝履吗？我们每个人不也是他人疾苦的来源之一吗？

正所谓:
模型皆有一死;
模型皆须侍奉。
好像也挺有道理的，你说是吗?
我们说回主题，这篇文章介绍如何用 FFM 模型来做推荐系统的统一召回。算是召回模型系列四篇的第二篇，之前在“推荐系统召回四模型之：全能的 FM 模型”中，介绍了一些基本知识，以及如何用 FM 模型做统一召回，又及，FM 模型是否可以做一体化的单阶段推荐模型。
本文为了能够看起来也独立成篇，所以很多前篇文章的基础知识点，仍然会保留。如果比较熟悉的读者，可以直接跳到“用 FM/FFM 模型做召回意味着什么”部分看起。
这个召回模型系列文章的缘起，来自于我对下列两个不太符合常规做法的推荐技术问题的思考，下面再次复述一遍:
第一个问题：我们知道在个性化推荐系统里，第一个环节一般是召回阶段，而召回阶段工业界目前常规的做法是多路召回，每一路召回可能采取一个不同的策略。那么打破常规的思考之一是：是否我们能够使用一个统一的模型，将多路召回改造成单模型单路召回策略？如果不能，那是为什么？如果能，怎么做才可以？这样做有什么好处和坏处?
第二个问题：我们同样知道，目前实用化的工业界的推荐系统通常由两个环节构成，召回阶段和排序阶段，那么为什么要这么划分？它们各自的职责是什么？打破常规的另外一个思考是：是否存在一个模型，这个模型可以将召回阶段和排序阶段统一起来，就是把两阶段推荐环节改成单模型单环节推荐流程？就是说靠一个模型一个阶段把传统的两阶段推荐系统做的事情一步到位做完？如果不能，为什么不能？如果能，怎么做才可以？什么样的模型才能担当起这种重任呢？而在现实世界里是否存在这个模型？这个思路真的可行吗?
本文主要探讨 FFM 模型是否能够解决上述两个问题。我仍然会先简单介绍下推荐系统整体架构以及多路召回的基本模式，然后说明下 FFM 模型基本思想，之后探讨 FFM 模型是否能够解决上面提到的两个非常规问题，如果能，该怎么解决？纯属个人思考，非经验分享，天马行空，谬误难免。
工业界推荐系统整体架构是怎样的



一个典型的工业级推荐系统整体架构可以参考上图，一般分为在线部分，近线部分和离线部分。

对于在线部分来说，一般要经历几个阶段。首先通过召回环节，将给用户推荐的物品降到千以下规模；如果召回阶段返回的物品还是太多，可以加入粗排阶段，这个阶段是可选的，粗排可以通过一些简单排序模型进一步减少往后续环节传递的物品；再往后是精排阶段，这里可以使用复杂的模型来对少量物品精准排序。对某个用户来说，即使精排推荐结果出来了，一般并不会直接展示给用户，可能还要上一些业务策略，比如去已读，推荐多样化，加入广告等各种业务策略。之后形成最终推荐结果，将结果展示给用户。
对于近线部分来说，主要目的是实时收集用户行为反馈，并选择训练实例，实时抽取拼接特征，并近乎实时地更新在线推荐模型。这样做的好处是用户的最新兴趣能够近乎实时地体现到推荐结果里。
对于离线部分而言，通过对线上用户点击日志的存储和清理，整理离线训练数据，并周期性地更新推荐模型。对于超大规模数据和机器学习模型来说，往往需要高效地分布式机器学习平台来对离线训练进行支持。



因为粗排是可选的，对于大多数推荐系统来说，通常在线部分的主体分为两个阶段就够，第一个阶段是召回，第二个阶段是排序。因为个性化推荐需要给每个用户展现不同的信息流或者物品流，而对于每个用户来说，可供推荐的物品，在具备一定规模的公司里，是百万到千万级别，甚至上亿。所以对于每一个用户，如果对于千万级别物品都使用先进的模型挨个进行排序打分，明显速度上是算不过来的，资源投入考虑这么做也不划算。从这里可以看出，召回阶段的主要职责是：从千万量级的候选物品里，采取简单模型将推荐物品候选集合快速筛减到千级别甚至百级别，这样将候选集合数量降下来，之后在排序阶段就可以上一些复杂模型，细致地对候选集进行个性化排序。
从上面在线推荐两阶段任务的划分，我们可以看出，召回阶段因为需要计算的候选集合太大，所以要想速度快，就只能上简单模型，使用少量特征，保证泛化能力，尽量让用户感兴趣的物品在这个阶段能够找回来；而排序阶段核心目标是要精准，因为它处理的物品数据量小，所以可以采用尽可能多的特征，使用比较复杂的模型，一切以精准为目标。
多路召回怎么做
目前工业界推荐系统的召回阶段一般是怎么做的呢？可以用一句江湖气很重的话来总结，请您系好安全带坐稳，怕吓到您，这句话就是：“一只穿云箭，千军万马来相见”。听起来霸气十足是吧？我估计看过古惑仔电影的都熟悉这句话，黑帮集结打群架的时候喜欢引用这句名言，以

增加气势，自己给自己打气。如果和推荐系统对应起来理解，这里的“穿云箭”就是召回系统，而千军万马就是各路花式召回策略。

目前工业界的推荐系统，在召回阶段，一般都采取多路召回策略。上图展示了一个简化版本的例子，以微博信息流排序为例，不同业务召回路数不太一样，但是常用的召回策略，基本都会包含，比如兴趣标签，兴趣 Topic，兴趣实体，协同过滤，热门，相同地域等，多者几十路召回，少者也有 7 / 8 路召回。

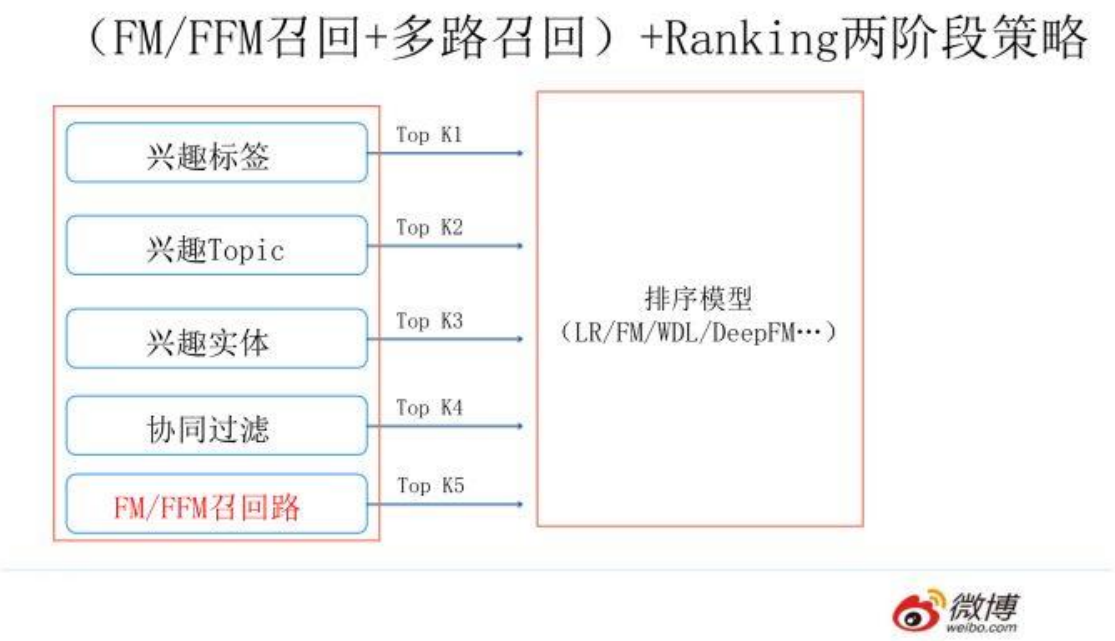
对于每一路召回，会拉回 K 条相关物料，这个 K 值是个超参，需要通过线上 AB 测试来确定合理的取值范围。如果你对算法敏感的话，会发现这里有个潜在的问题，如果召回路数太多，对应的超参就多，这些超参组合空间很大，如何设定合理的各路召回数量是个问题。另外，如果是多路召回，这个超参往往不太可能是用户个性化的，而是对于所有用户，每一路拉回的数量都是固定的，这里明显有优化空间。按理说，不同用户也许对于每一路内容感兴趣程度是不一样的，更感兴趣的那一路就应该多召回一些，所以如果能把这些超参改为个性化配置是很好的，但是多路召回策略下，虽然也不是不能做，但是即使做，看起来还是很 Trick 的。有什么好办法能解决这个问题吗？有，本文后面会讲。

用 FM/FFM 模型做召回意味着什么

本文的主体是春节期间写完的，但是这个部分是最近加进来的，因为我发现第一篇介绍 FM 模型做召回的文章发出去后，很多人反馈有些地方看不懂。这让我一度感到 FM 那篇文章写得有些失败，通俗易懂一直是我写文章的首要追求，有时候宁肯为了保证这点，牺牲掉很多公式的表达。因为据说每加进一个公式，读者就会跑掉一半。每当我写到公式部分的时候，都会下意识地数数公式数量，当发现读者都已经跑光了的时候，我一般回头会把公式都默默地删掉。说远了，这里增加这个部分，是考虑到上篇介绍 FM 做召回的文章读者的疑惑，先做些背景说明，以增进理解。

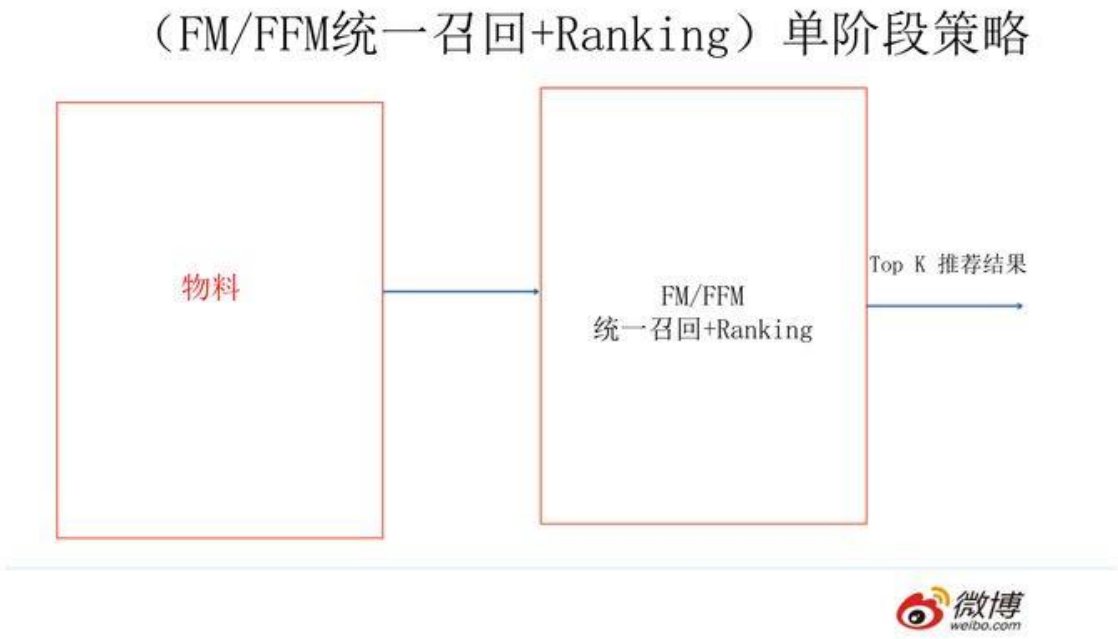
首先，第一个问题是：FM/FFM 模型一般是作为典型的 Ranking 阶段的模型，怎么理解用它来做召回这件事情呢？

上图是目前最常见的多路召回+Ranking 的两阶段推荐策略，上面讲过不多说。



如果我们想把一般用来做排序的模型，比如 FM、FFM，用来做召回模型。那么一种比较温柔地做法，参考上图，就是用模型作为一路新的召回策略，或者用模型召回代替掉原先多路召回的一路或者几路召回，这是温柔派的做法。

你说了，我生性比较凶猛，堪比猛禽，那么怎么做呢？像上图这么做，用 FM/FFM 模型代替掉所有原先的多路召回策略。本文明显主要目的之一是这个，从这里好像你很容易推断我也是猛禽派的代表是吧？

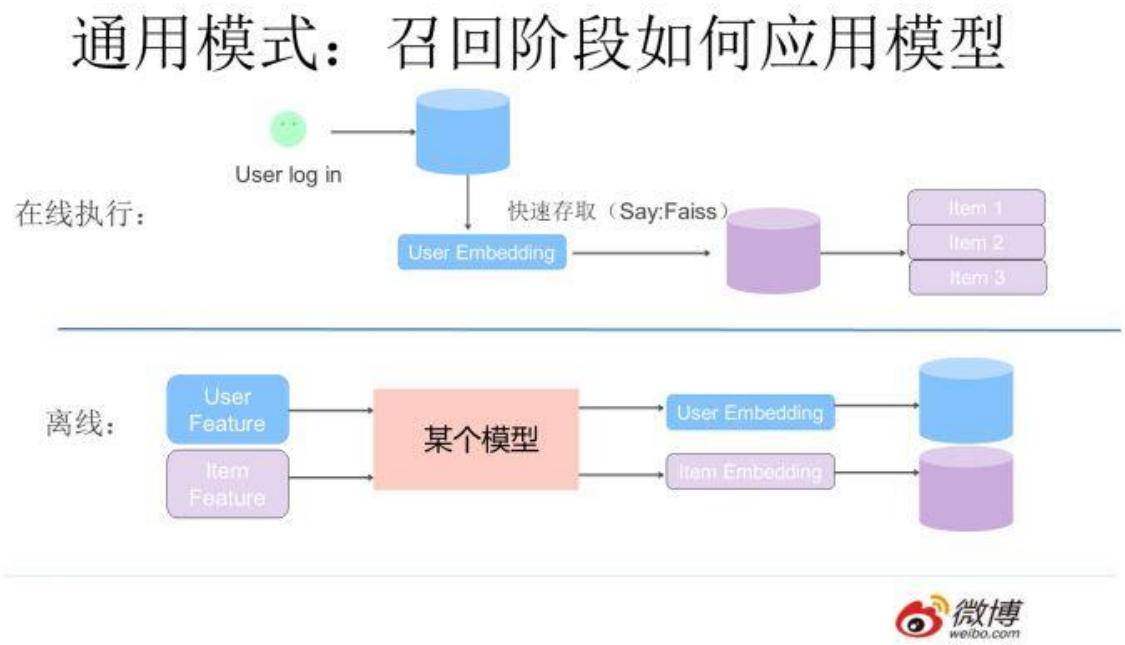


你又说了，刚才那个做法也还算温柔吧，我属于飞越疯人院那种风格的，是不是可以做得更生猛些？可以啊，向上图这么做推荐，就是用一个模型把召回和排序两个阶段的事情全做掉。这也是本文要探讨的另外一个要点。嗯，此时，聪明的你肯定又推论出了什么真理是吧？

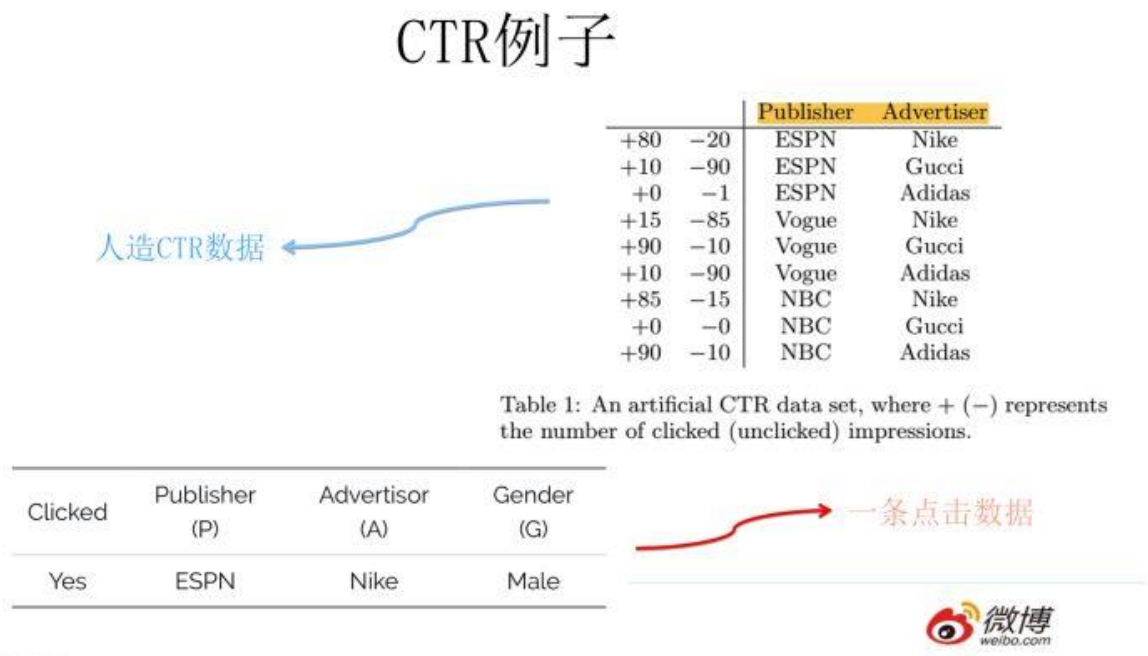
上面说的是有困惑的第一个点，可能我之前没讲明白用排序模型做召回是什么意思；第二个问题是：FM/FFM 做排序的话，大家都很熟悉怎么用了，那么用这些模型做召回，对它有什么和做排序不同的要求吗？其实把 FM/FFM 模型用在召回和 Ranking 这两个不同阶段，差别还是蛮大的。

如果是在排序阶段使用 FM/FFM 或者其他模型，因为此时用户已知，要排序的具体是哪篇文章也知道（通过召回阶段拉回来少量的文章），都在模型面前摆着，此时模型的任务是要判断用户是否对某篇文章感兴趣，所以用户特征和物料特征可以同时作为模型的输入。而如果是在召

回阶段使用 FM/FFM 模型，首先面临的问题是：我们现在只知道是哪个用户在浏览，用户特征都是全的，但是面对的判断对象是千万量级的文章，汪洋大海，人民战争，而不是某篇具体的文章。模型的任务是：只拿着用户特征，去千万量级的文章库里找出一小批用户可能感兴趣的文章出来，而且速度要够快。这个要求就稍微刁钻一些。
这是它们最大的不同，一个不同是召回阶段要计算的数据量巨大；另外一个不同是貌似我们手头只有用户特征，此时如何应用模型呢？所以你可以看出来，在召回阶段，使用某个具体的模型，比排序阶段使用这个模型的应用条件更苛刻，需要满足一定的模式，才能把某个模型用到召回阶段。那么，怎么做呢？

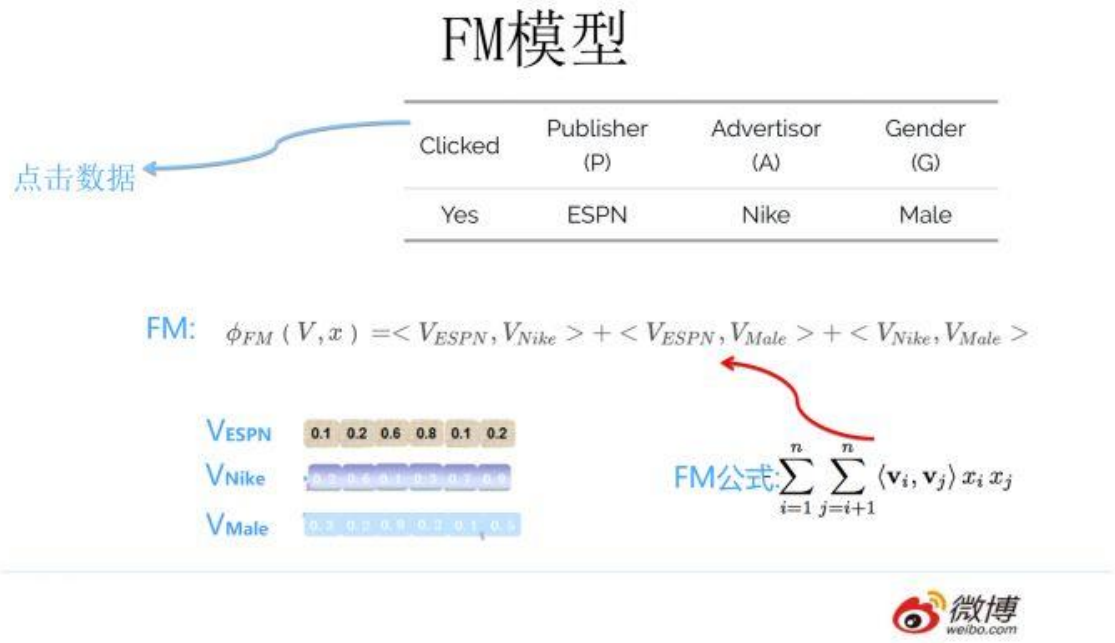


上图展示了一个通用的在召回阶段使用模型的思路，尽管具体采用的模型不同，但是基本都是在这个框架下运转的：因为用户特征和物料特征不能同时作为模型的输入，那么我们需要对它们分别处理。具体做法是，采用某个模型，离线把用户特征打包成用户 embedding，代表用户兴趣向量；同时可以离线或者近乎在线地把物料特征也单独打包，打成物料 embedding，需要将两类特征分离。
在使用模型的时候，对于每个用户以及每个物品，我们可以按照上述方法，将每个用户的兴趣向量离线算好，存入在线数据库中比如 Redis（用户 ID 及其对应的 embedding），把物品的向量逐一离线算好，存入 Faiss(Facebook 开源的 embedding 高效匹配库)数据库中。
当用户登陆或者刷新页面时，可以根据用户 ID 取出其对应的兴趣向量 embedding，然后和 Faiss 中存储的物料 embedding 做内积/Cosine 等不同类型的计算，按照得分由高到低返回得分 Top K 的物料作为召回结果。提交给第二阶段的排序模型进行进一步的排序。
所以你看到了，在召回阶段是如何使用模型的：首先用户特征和物品特征需要分离打包，这个包怎么打？才能符合 FM/FFM 的计算原则？这是一个问题。然后通过 Faiss 解决数据量太大计算速度慢的问题，所以速度问题可以认为已经被解决了。剩下的问题就是 Faiss 的对用户兴趣 embedding 和物料 embedding 做内积计算，这种计算结果，是否符合 FM/FFM 模型的计算原则？或者其它模型的计算原则？这个也是关键。想明白上述一个问题一个关键，那么完全可以采用新模型来做这个事情。
此为背景解释，下面进入模型相关内容。
什么是 FFM 模型
FFM 的全称是 Field-aware FM，直观翻译过来，就是能够意识到特征域(Field)的存在的 FM 模型。那么 FFM 模型是有第六感吗？它怎么能够感知到特征域的存在呢？这里先不解释，后面会说明。我们先看个例子。



上图是一个人造的广告 CTR 的数据例子，代表的意思是在某个网站（Publisher）上刊登一则广告（Advertiser），某个用户（用户性别特征 Gender）是否会点击某条广告的数据。这个例子中假设包含三个特征域（Field）:网站 Publisher(可能的特征值是 ESPN、Vogue、NBC)、广告（可能的特征值是 Nike、Adidas、Gucci）和性别特征 Gender(可能的特征值是 Male、Female)。由这个例子可以看出组合特征的重要性：如果在体育网站 ESPN 上发布 Nike 的广告，那么 100 次展现，80 次会被点击，而 20 次不会被点击。意味着组合特征（Publisher="ESPN" and Advertiser="Nike"）是个很强的预测用户是否点击的二阶组合特征。上图同时展示了一条用户点击记录。

我们用这个例子来说明 FFM 的基本思想，FM 模型可以看做是 FFM 模型的一个特例，所以在说明 FFM 模型思想之前，我们先用上述例子说明 FM 的思想，然后通过和 FM 模型的对比，很容易理解 FFM 模型的基本思路。



FM 模型在做二阶特征组合的时候，对于每个二阶组合特征的权重，是根据对应两个特征的 Embedding 向量内积，来作为这个组合特征重要性的指示。当训练好 FM 模型后，每个特征都

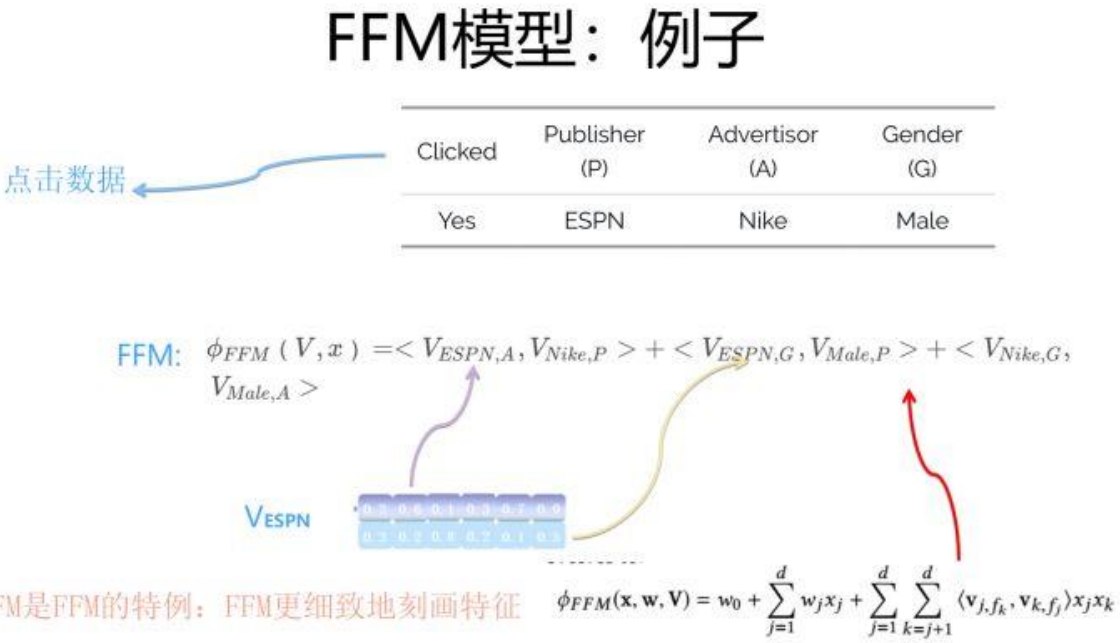
可以学会一个特征 embedding 向量，参考上图。当做预测的时候，比如我们接收到上面例子的数据，需要预测用户是否会点击这条广告，则对三个特征做两两组合，每个组合特征的权重，可以根据两个对应的特征 embedding 内积求得，对所有组合特征求和后，套接 Sigmoid 函数即可做出二分类预测。

对于 FM 模型来说，每个特征学会唯一的一个特征 embedding 向量，注意，在这里，和 FFM 的最大不同点冒出来了。为了更容易向 FFM 模型理解过渡，我们可以这么理解 FM 模型中的某个特征的 embedding：拿 Vespn 这个特征作为例子，当这个特征和其它特征域的某个特征进行二阶特征组合的时候，不论哪个特征域的特征和 Vespn 特征进行组合，Vespn 这个特征都反复使用同一个特征 embedding 去做内积，所以可以理解为 Vespn 这个特征在和不同特征域特征进行组合的时候，共享了同一个特征向量。

沿着这个思路思考，我会问出一个问题：我们可以改进下 FM 模型吗？怎么改进？下图给个提示。

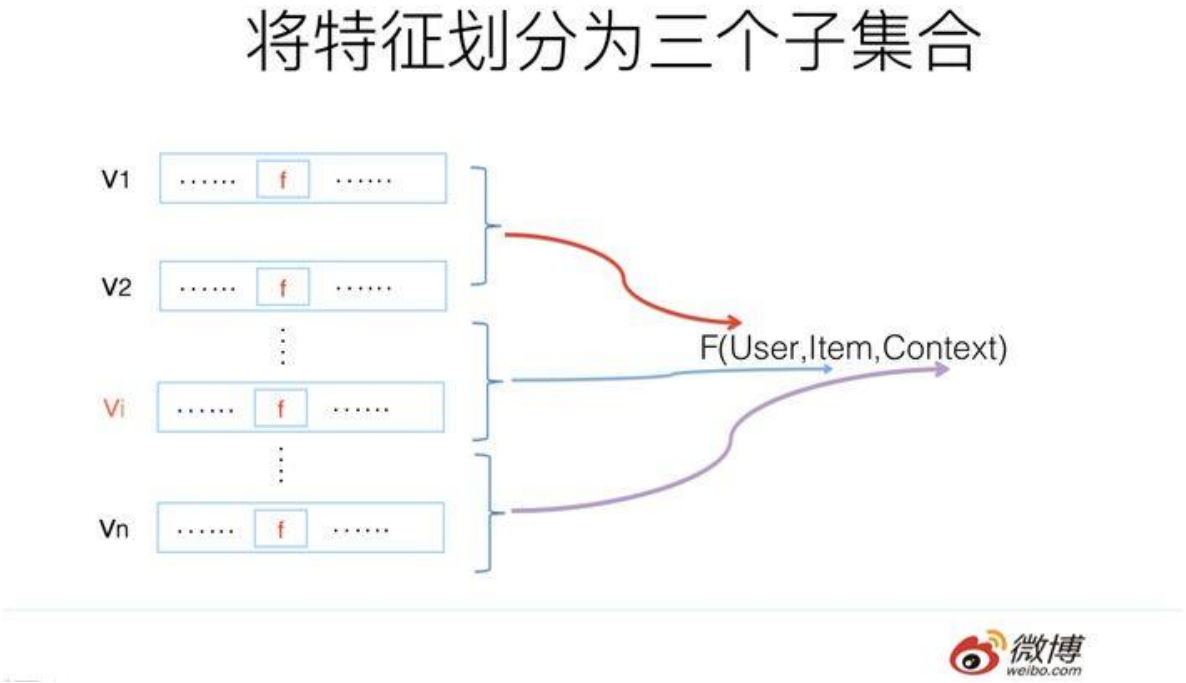
如果你对算法敏感的话，你可以这么回答我：既然 FM 模型的某个特征，在和任意其它特征域

的特征进行组合求权重的时候，共享了同一个特征向量。那么，如果我们把这个事情做地更细致些，比如 Vespn 这个特征，当它和 Nike（所属特征域 Advertiser）组合的时候用一个特征 embedding 向量，而当它和 Male(所属特征域 Gendor)组合的时候，使用另外一个特征 embedding 向量，这样是否在描述特征组合的时候更细腻一些？也就是说，当 Vespn 这个特征和属于 Advertiser 这个域的特征进行组合的时候，用一个特征 embedding；和属于 Gendor 这个特征域的特征进行组合的时候，用另外一个特征 embedding。这意味着，如果有 F 个特征域，那么每个特征由 FM 模型的一个 k 维特征 embedding，拓展成了（F-1）个 k 维特征 embedding。之所以是 F-1，而不是 F，是因为特征不和自己组合，所以不用考虑自己。
这样行吗？
嗯，你说的很有道理，是的，这其实就是 FFM 模型的基本思想。所以从上面两个图的示意可以看出，为何说 FM 模型是 FFM 模型的特例。



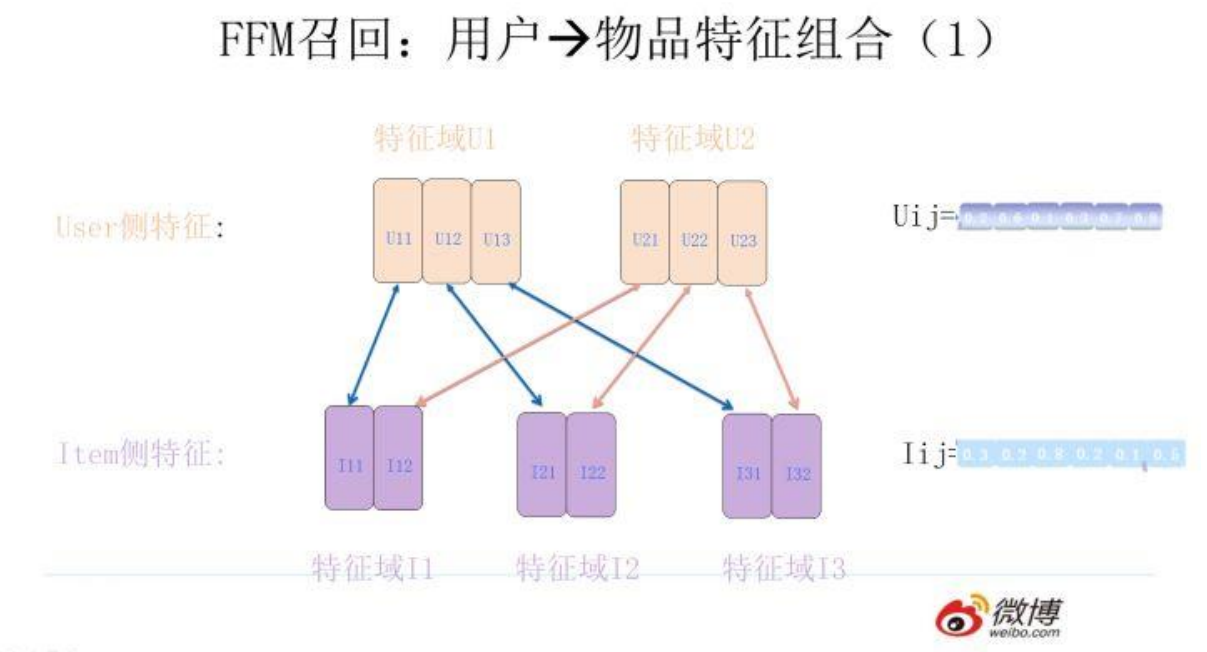
我们再回头看下刚才那个点击数据的例子，看看在 FFM 场景下是怎样应用的，上图展示了这个过程。因为这个例子有三个特征域，所以 Vespn 有两个特征 embedding，当和 Nike 特征组合的时候，用的是针对 Advertisor 这个特征域的 embedding 去做内积；而当和 Male 这个特征组合的时候，则用的是针对 Gendor 这个特征域的 embedding 去做内积。同理，Nike 和 Male 这两个特征也是根据和它组合特征所属特征域的不同，采用不同的特征向量去做内积。而两两特征组合这个事情的做法，FFM 和 FM 则是完全相同的，区别就是每个特征对应的特征 embedding 个数不同。FM 每个特征只有一个共享的 embedding 向量，而对于 FFM 的一个特征，则有（F-1）个特征 embedding 向量，用于和不同的特征域特征组合时使用。
从上面的模型演化过程，你可以体会到，为何这篇文章的标题将 FFM 模型称为笨重，它笨重在哪里？说它笨重，是和 FM 模型相比较而言的。我们可以推出，假设模型具有 n 个特征，那么 FM 模型的参数量是 n*k（暂时忽略掉一阶特征的参数），其中 k 是特征向量大小。而 FFM 模型的参数量呢？因为每个特征具有（F-1）个 k 维特征向量，所以它的模型参数量是(F-1)*n*k，也就是说参数量比 FM 模型扩充了(F-1)倍。这意味着，如果我们的任务有 100 个特征域，FFM 模型的参数量就是 FM 模型的大约 100 倍。这其实是很恐怖的，因为现实任务中，特征数量 n 是个很大的数值，特征域几十上百也很常见。另外，我们在上一篇介绍 FM 模型的文章里也讲过，FM 模型可以通过公式改写，把本来看着是 n 的平方的计算复杂度，降低到
。而 FFM 无法做类似的改写，所以它的计算复杂度是
，这明显在计算速度上也比 FM 模型慢得多。所以，无论是急剧膨胀的参数量，还是变慢的计算速度，无论从哪个角度看，相对 FM 模型，FFM 模型是略显笨重的。

正因为 FFM 模型参数量太大，所以在训练 FFM 模型的时候，很容易过拟合，需要采取早停等防止过拟合的手段。而根据经验，FFM 模型的 k 值可以取得小一些，一般在几千万训练数据规模下，取 8 到 10 能取得较好的效果，当然，k 具体取哪个数值，这其实跟具体训练数据规模大小有关系，理论上，训练数据集合越大，越不容易过拟合，这个 k 值可以设置得越大些。
上面是对 FFM 模型基本思想的说明，下面我们讨论如何用 FFM 模型做召回。
如何用 FFM 做召回模型
如果要做一个实用化的统一召回模型，要考虑的因素有很多，比如 Context 上下文特征怎么处理，一阶项特征怎么加入等。为了能够更清楚地说明，我们先从简易模型说起，然后逐步加入必须应该考虑的元素，最后形成一个实用化的 FFM 版本的召回模型。不论是简化版本 FFM 召回模型，还是完全化版本，首先都需要先做如下两件事情：
第一，离线训练。这个过程跟在排序阶段采用 FFM 模型的离线训练过程是一样的，比如可以使用线上收集到的用户点击数据来作为训练数据，线下训练一个完整的 FFM 模型。在召回阶段，我们想要的其实是：每个特征和这个特征对应的训练好的（F-1）个 embedding 向量。这个可以存好待用。



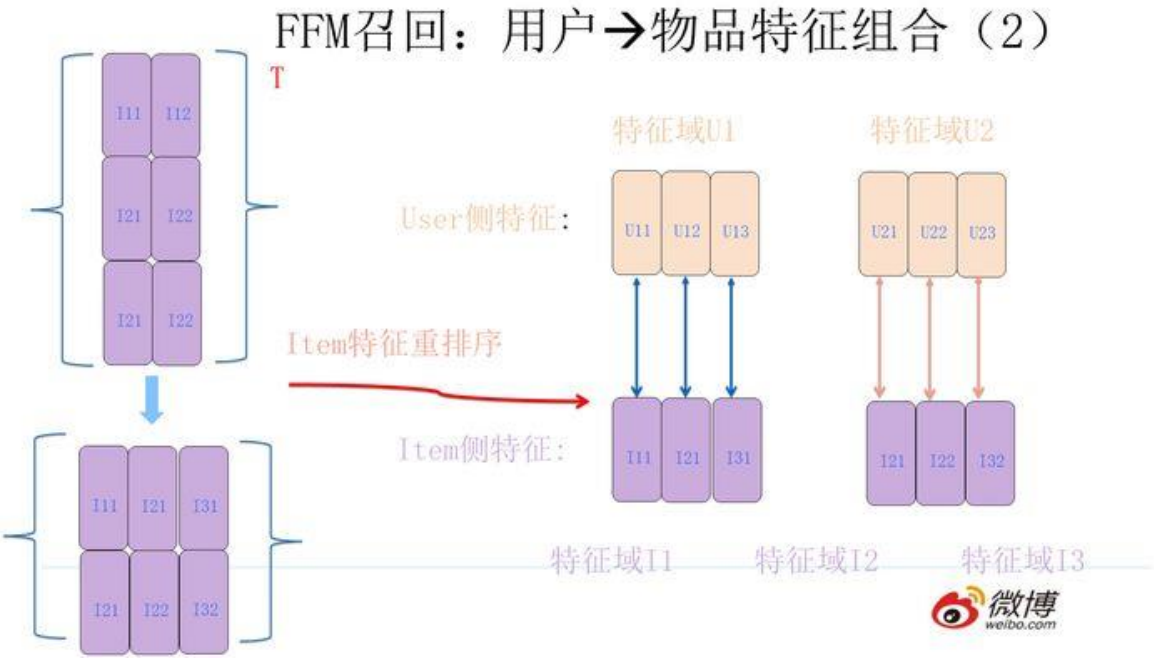
第二，如果将推荐系统做个很高层级的抽象的话，可以表达成学习如下形式的映射函数：
意思是，我们利用用户（User）相关的特征，物品(Item)相关的特征，以及上下文特征（Context,比如何时何地用的什么牌子手机登陆等等）学习一个映射函数 F。学好这个函数后，当以后新碰到一个 Item，我们把用户特征，物品特征以及用户碰到这个物品时的上下文特征输入 F 函数，F 函数会告诉我们用户是否对这个物品感兴趣。如果他感兴趣，就可以把这个 Item 作为推荐结果推送给用户。
说了这么多，第二个我们需要做的事情是：把特征域划分为三个子集合，用户相关特征集合，物品相关特征集合以及上下文相关的特征集合。而用户历史行为类特征，比如用户过去点击物品的特征，可以当作描述用户兴趣的特征，放入用户相关特征集合内。至于为何要这么划分，后面会讲。做完上述两项基础工作，我们可以试着用 FFM 模型来做召回了。
简易版 FFM 召回模型
我们先来尝试着构建一个简易版的 FFM 召回模型。在本文前面，我新增加了一节内容，专门叙述了如果想要使用类似 FM/FFM 这种排序模型来做召回，面临哪些约束，以及要解决的一个问题和一个关键点。那么如果你现在的任务是使用 FFM 模型来做召回，这个问题以及关键点怎么解决？建议你可以想想。下面是我思考的方案。
1.1 问题：如何根据 FFM 计算原则构建用户 Embedding 以及物品 Embedding

上文简单叙述过，用排序模型做召回的特点。其实，你可以这么理解：把 FM/FFM 等模型用来做召回，看做原先的“召回+排序”两阶段过程中的第二个过程前置，放到召回阶段来做排序。它本质上其实就是希望同时将两阶段过程用同一个阶段吸收掉。
只是因为召回阶段面临的待处理物料数量太大，所以依赖一种高效的计算模式，而这个目前看不是问题，成熟的方案就是 Faiss 的 Embedding 匹配的模式，速度应该是足够实用化的。
所以问题就转换成了：如何根据某个模型的计算标准，打出对应的用户侧 Embedding，以及物品侧的 Embedding。于是，我们可以将召回阶段的 FM/FFM 或者其它模型看成是一种受约束的排序过程，这里的“约束”，指的是需要明确将 FM/FFM 召回模型划分为两个阶段：首先需要离线将用户侧特征和物品侧特征进行分离编码，然后在线快速 embedding 匹配的时候完成模型计算过程。这不像传统的排序阶段使用 FM/FFM 模型，此时，两侧特征可以同时作为模型的输入，明显更灵活，受约束更小。所以，我们可以把召回阶段采用这种排序模型看成一种受约束的排序。我们的主题是利用 FFM 模型做召回。于是问题转换成了：如何根据 FFM 模型的计算原则，打出对应的用户侧 Embedding，以及物品侧 Embedding。怎么做呢？
我用一个极度简化的例子来说明这个过程：假设在这个例子中，我们只使用五个特征域，用户侧采用两个特征域 U1 和 U2，而物品侧采用三个特征域 I1,I2 和 I3。当面对具体数据实例的时候，对应特征域下会有一个对应的特征值存在。对于某个具体的特征值 f1 来说，根据 FFM 的计算原则，它在离线训练阶段会学会 4 个对应的 embedding 向量，分别在这个特征和其它特征域的特征进行特征组合的时候使用。

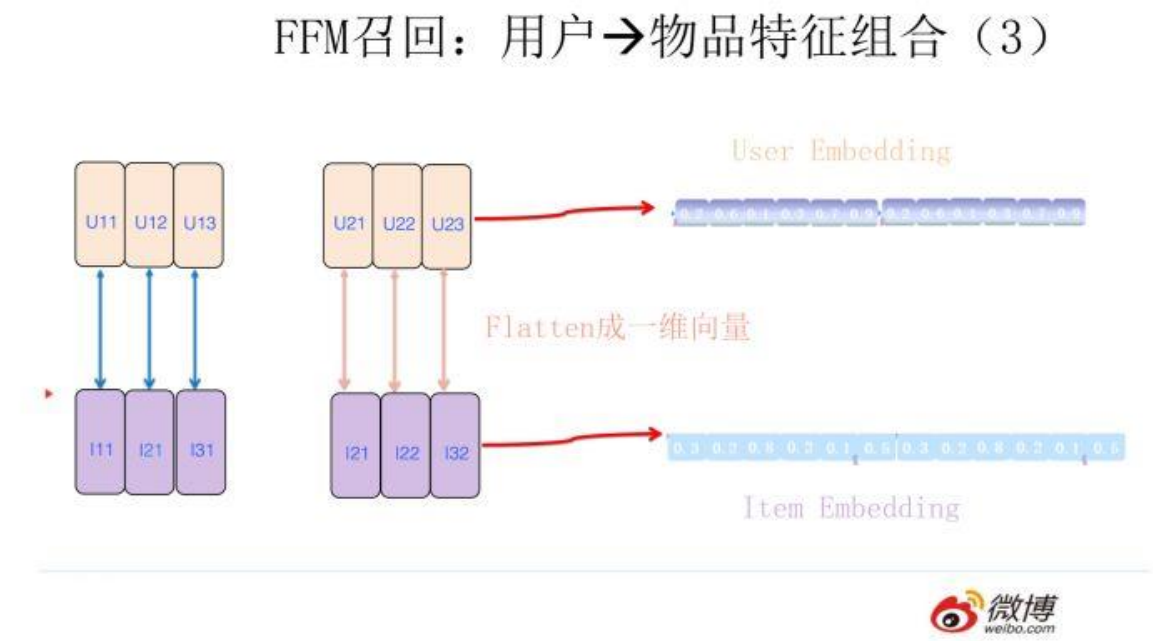


首先，要意识到，我们当前的任务是希望通过 FFM 模型来做用户任意特征和物品任意特征的组合。
对于用户侧的两个特征，我们取出它们分别用于和对应的三个物品侧特征域组合时要用的 embedding 向量。比如对于 U1 来说，我们分别将这三个特征 embedding 称为 U11/U12/U13，U11 的两个下标数字的含义是：这是第 1 个用户侧的特征域 U1 和第 1 个物品侧特征域 I1 进行组合时使用的特征 embedding。U12 则是第 1 个用户侧的特征域 U1 和第 2 个物品侧特征域 I2 进行组合时使用的特征 embedding。如此处理，于是每个用户侧的特征取出三个特征向量，每个物品侧的特征取出两个特征向量。形成上图的结构。
根据 FFM 的计算规则，如果我们希望计算用户侧和物品侧的两两特征组合，需要将特征向量求内积时的对应关系建立起来，图中箭头标出了对应关系。你可能看着有点乱，但是对应关系里面隐藏着一个规律，你可以找找这个规律看。提示下：你可以看看 U 和 I 特征向量下标编号，有什么规律性的对应关系吗？U12 < -->I21、U23 < -->I32……，嗯，我估计你看出来了，

规律就是 < Uij,Iji>。

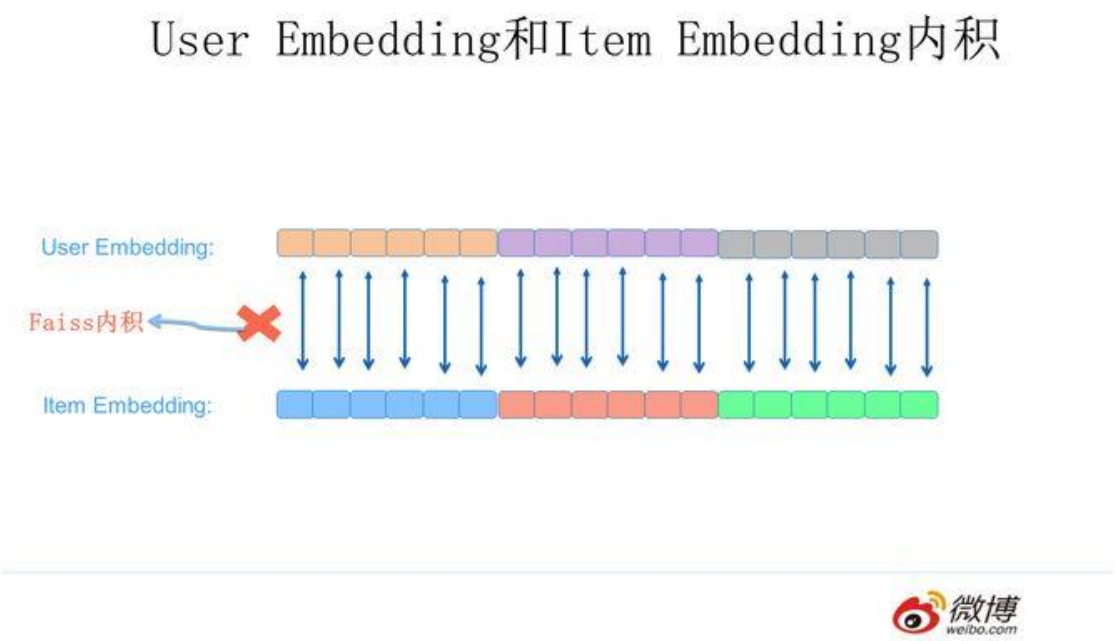


上面那张图的特征向量之间的对应关系，看着确实有点让人眼花缭乱，那么能否让它们的对应关系看上去更简洁直接一些呢？很简单，只需要把物品侧的特征向量重新排下顺序即可。这个重排序的过程，可以看做是：对原先顺序排列的物品侧特征向量矩阵，做了一个转置操作。这样，每个物品侧的特征向量，就和需要求内积的对应用户侧特征向量，形成了整齐的一一对齐的效果了。具体过程参考上图。



我们讲过，模型做召回，要解决的问题是：如何利用 FFM 原则打出对应的用户侧 embedding 和物品侧 embedding。前面两段所讲的，是根据 FFM 原则，对应的特征向量应该如何对齐的过程，而如果向量对齐后，怎么打出两个 embedding 向量？很简单，把刚才对齐的二维向量拉平，顺序 concat 连接，就形成了展开的一维的用户 embedding 和物品 embedding。然后，我们可以把每个物品的 embedding 离线存入 Faiss，用户 embedding 离线算好，放在内存数据库中。当用户登录或者刷新时，在线根据用户的 embedding 向量，通过 Faiss 的快速查询功能，根据内积往回拉取 top K 物品，返回的物品就是根据 FFM 模型计算得分最高的推荐结果。

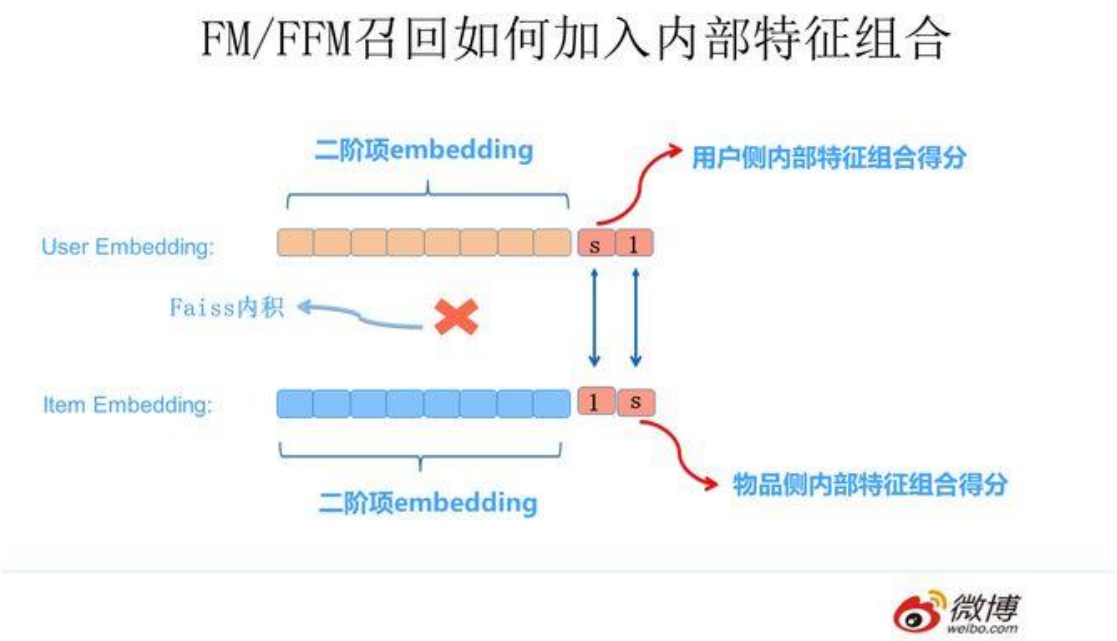
1.2 关键点：用户 Embedding 和物品 Embedding 内积计算符合 FFM 计算原则吗
这样，其实就已经完成了一个简易版本的 FFM 召回模型。我们考虑下之前说的关键点：两个拉长版本的 User Embeding 和 Item Embedding，通过 Faiss 内积计算，最后的得分，是否和标准的 FFM 计算结果等价？



两者很显然是等价的，< U,I>内积的操作是两个长向量对应位的数值相乘，然后求和，所以拉长向量匹配版本和分拆成子项分别求内积再求和，数值是一样的，从上图示例可以很容易看出这一点。从上述说明可以看出，此时我们获得了一个基础版本的 FFM 召回模型，这个版本的召回模型，只考虑了 U 和 I 特征的相互组合，其它的因素还没考虑。
此时应该回头再想想我们的标题：沉重的 FFM。为什么我说 FFM 沉重呢？你可以算算这个拉平的 embedding 向量的长度。假设在我们的实际任务中，用户侧有 50 个特征域（M=50），物品侧有 50 个特征域(N=50)，每个特征向量的大小 k=10，可以很容易推断出用户和物品的 embedding 长度，它的 size=M*N*K=50*50*10=25000，两万五千，“苦不苦，想想红军两万五，累不累，想想革命老前辈”，如果把一个数值位换成一里地，那快赶上长征的距离了。而这对于 Faiss 来说，如果物品库比较大，速度明显是跟不上的。
一种直观减小 embedding 长度的方法是把 k 值往小放，比如 k=2 或者 4。如果只是使用 FFM 模型做召回，这个策略是可行的，反正召回阶段不用特别准，推荐结果的准确性靠第二个排序阶段来来保证，召回阶段原则上能把好的物料找回来即可。即使这样，embedding size=50*50*2=5000，长度也还是很长，虽说比不上长征的里程，但是明显比苏小妹的脸还是要长的。
另外一种思路是把特征域数量降下来，比如 M=N=10，就是说用户和物品两侧各有 10 个特征域，这样的话 embedding size=10*10*2=200。嗯，这个基本可以实用化了。如果只是将 FFM 用来做召回，虽说受限严重，但这么做，也不是不可以。
但是，我希望 FFM 不仅能够不受特征域数量限制地做召回，而且最好它还能一阶段地把排序也做掉，所以靠上面两个手段，是不能从根本上解决问题的。有什么加速策略吗？我想了两个方法，后面会分别介绍。
我们先把速度问题往后放一放，等会再谈。先一步一步优化这个 FFM 召回模型。上面介绍的 FFM 召回模型，只是个简易版本，和标准 FFM 模型相比，很多内容它还没有考虑进来，比如用户侧或物品侧内部特征组合问题，一阶项如何引入的问题以及如何融入场景上下文特征问题，如果再将这三者引入，此时应该怎么做呢？
2.加入用户侧及物品侧内部特征组合

上小节所述内容，本质上是在计算用户侧任意特征和物品侧任意特征之间的两两特征组合。到这里你发现，如果希望在召回阶段完整复现 FFM 模型，还需要考虑用户侧内部两两特征组合，以及物品侧内部两两特征组合。

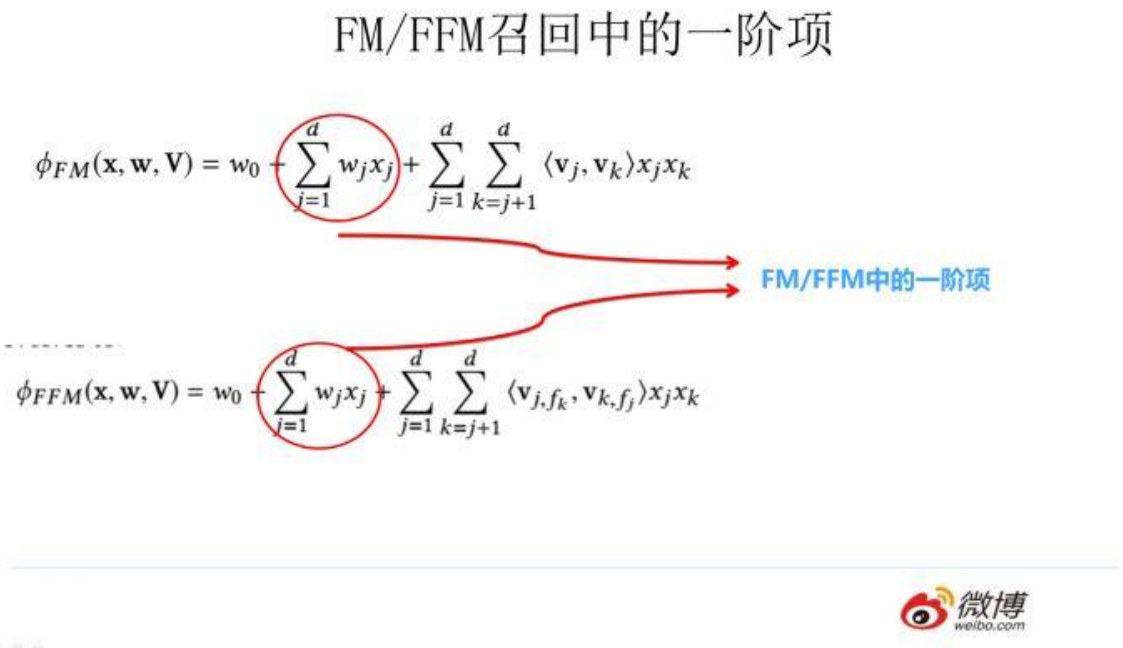
至于用户侧或者物品侧内部的两两特征组合的计算方法，可以仿照上述计算用户侧和物品侧特征组合的方法，也可以按照标准的 FFM 计算流程计算，总之方式比较灵活。关键的问题是：假设用户侧的内部特征两两组合得分 Score(User_i*User_j)及 Score(Iem_i*Item_j)算出来后，如何把它们集成进入那两个长长的用户 embedding 和物品 embedding 中？



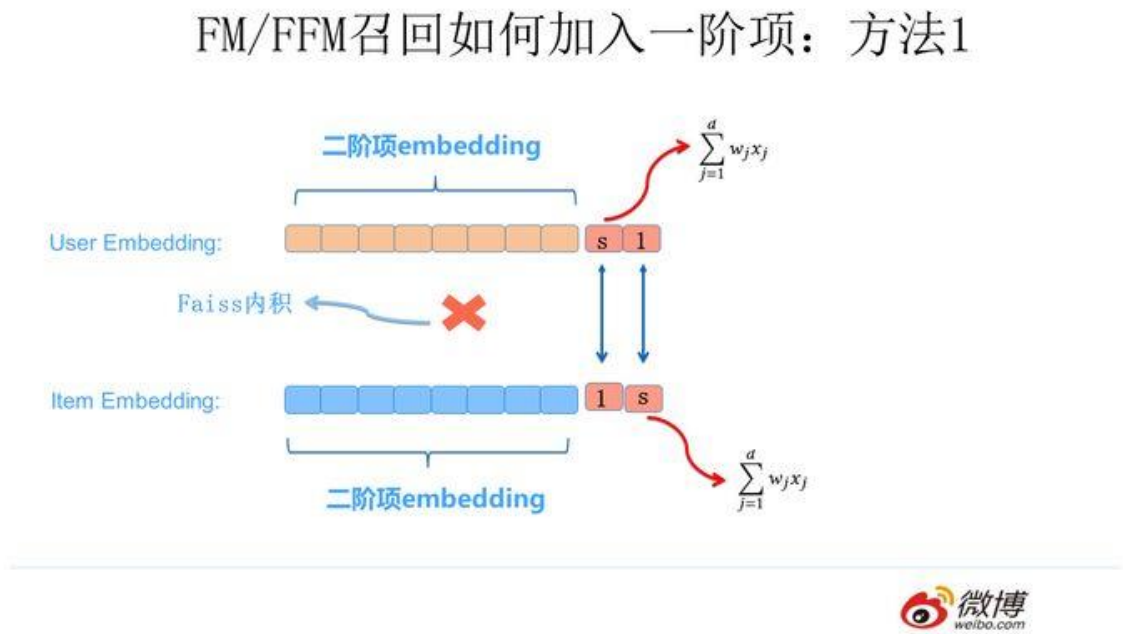
可以如上图所示去做，在用户的二阶项 embedding 后添加两位：一位就是用户侧内部特征组合得分，在对应的物品侧位置，增加一位，值设置为 1。这样的话，在 Faiss 做内积的过程中，就将用户侧内部特征组合得分计入；类似地，在物品侧也可以如此炮制。这样就将 U 和 I 的内部特征组合融入 FFM 召回模型中了，FM 模型也是一样的道理。

理论上来说，如果是只用 FM/FFM 模型做召回，用户侧内部的特征组合对于返回结果排序没有影响，所以可以不用加入。物品侧内部特征之间的特征组合可能会对返回的物品排序结果有影响，可以考虑引入这种做法，把它统一加进去。而如果是希望用 FM/FFM 模型一阶段地替代掉“多路召回+Ranking”的两阶段模式，则可以考虑完全复现 FM/FFM 模型，如此，应将两侧的内部特征组合都考虑进去。（本小节内容是最近新加入的，这一部分的做法及使用场景是在最近的讨论中，微博机器学习团队佘青云同学想出来的，在此表示感谢）

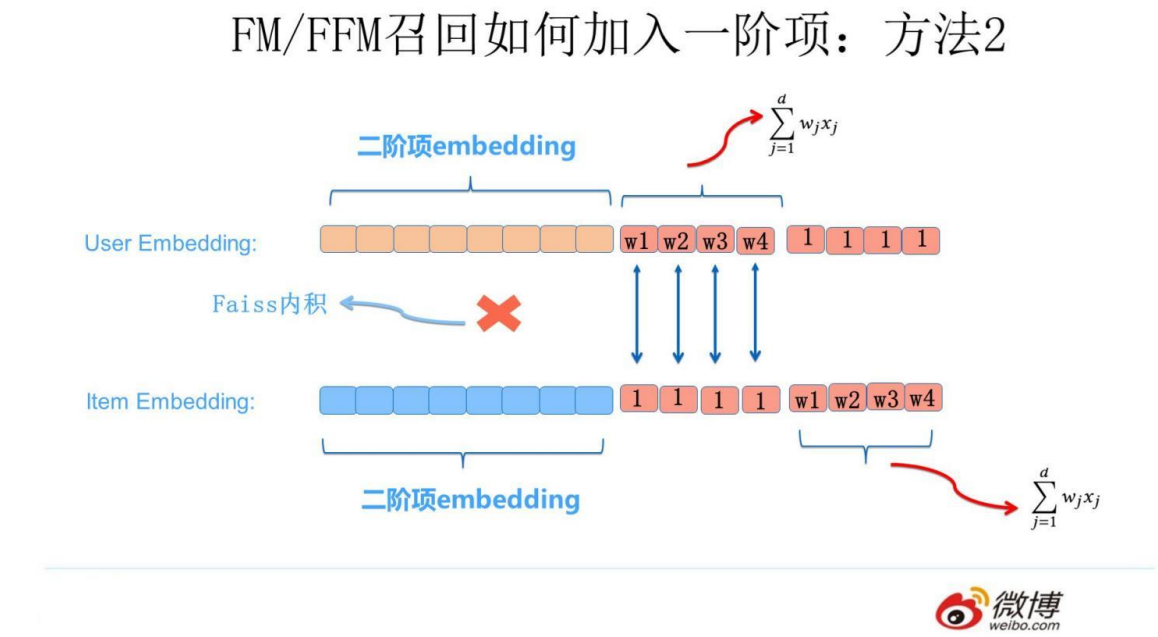
3.如何加入一阶项



我们知道，标准的 FM/FFM 公式中是包含一阶项的，也就是 LR 模型。如果我们根据上节方法所述，做出了用户侧和物品侧的二阶项 embedding，此时，想要把一阶项加入 FM/FFM 召回模型，应该怎么做呢？



其实很简单，上图展示了一种做法，在用户侧的 embedding 中增加两位，第一位是属于用户特征域的特征对应的一阶项累加和，相应地，在物品侧对应位置增加一位，设置值为 1，这样在 Faiss 求内积的过程中，就把用户侧的一阶项引入了。类似地，也可以如此加入物品侧的一阶项。



还有一种做法，如上图所示，不做用户侧和物品侧的一阶项求和，而是直接将用户侧及物品侧对应特征的一阶权重拼接到二阶项的 embedding 后。同样的，对应的物品侧或用户侧相应位置设置为 1。这样，也可以在 Faiss 求内积过程中，把一阶项算入得分中。

微博在业务中的实践表明，如果采取 FM 召回模型，对于有些应用来说，一阶项对于最终效果有明显影响，所以在用 FM/FFM 做召回的时候，是需要将一阶项考虑进去的，这可能是个别一阶特征比较重要导致的。我们在 Criteo 数据集合的实验结果也证明：如果是 FM 模型，一阶项是有用的，去掉一阶项，只保留二阶项，AUC 大约会掉 1 个绝对百分点，对于 CTR 来说，这个差距还是很明显的；而如果是采用 DeepFM 模型，则 FM 部分是否保留一阶项对最终结果没有什么影响，这说明 DNN 的隐层有效地将一阶项的作用吸收掉了。（这一小节也是最近新加入的，感谢微博机器学习团队黄通文同学补充的 Criteo 实验数据，以及马柏樟/邸海波同学在微博正文页推荐业务中测试 FM 统一召回模型时，提供的业务数据表现和一些建议）

4.如何加入场景上下文特征

我们上面说过，抽象的推荐系统除了用户特征及物品特征外，还有一类重要特征，就是用户发生行为的场景上下文特征（比如什么时间在什么地方用的什么设备在刷新），而上面逐步改进版本的 FFM 召回模型还没有考虑这一块。

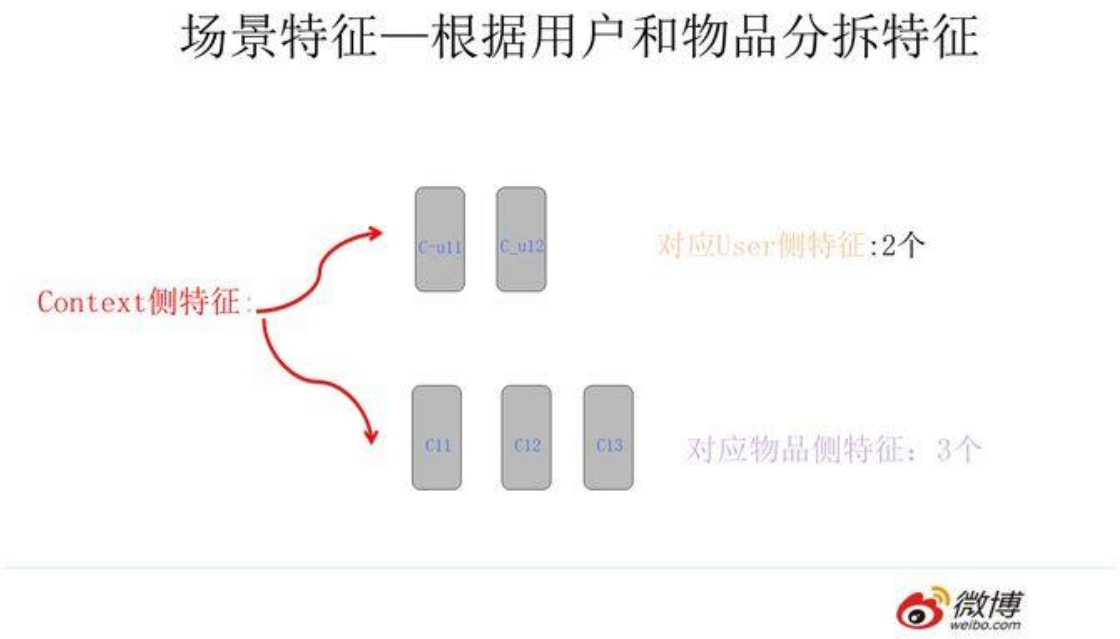
之所以把上下文特征单独拎出来，是因为它有自己的特点，有些上下文特征是近乎实时变化的，比如刷新微博的时间，再比如对于美团嘀嘀这种对地理位置特别敏感的应用，用户所处的地点可能随时也在变化，而这种变化在召回阶段就需要体现出来。所以，上下文特征是不太可能像用户特征离线算好存起来直接使用的，而是用户在每一次刷新可能都需要重新捕获当前的特征值。动态性强是它的特点。

而考虑进来上下文特征，如果我们希望构造和标准的 FFM 等价的召回模型，就需要多考虑两个问题：

问题一：既然部分上下文特征可能是实时变化的，无法离线算好，那么怎么实时地将它融入上文所述的 FFM 召回计算框架里？

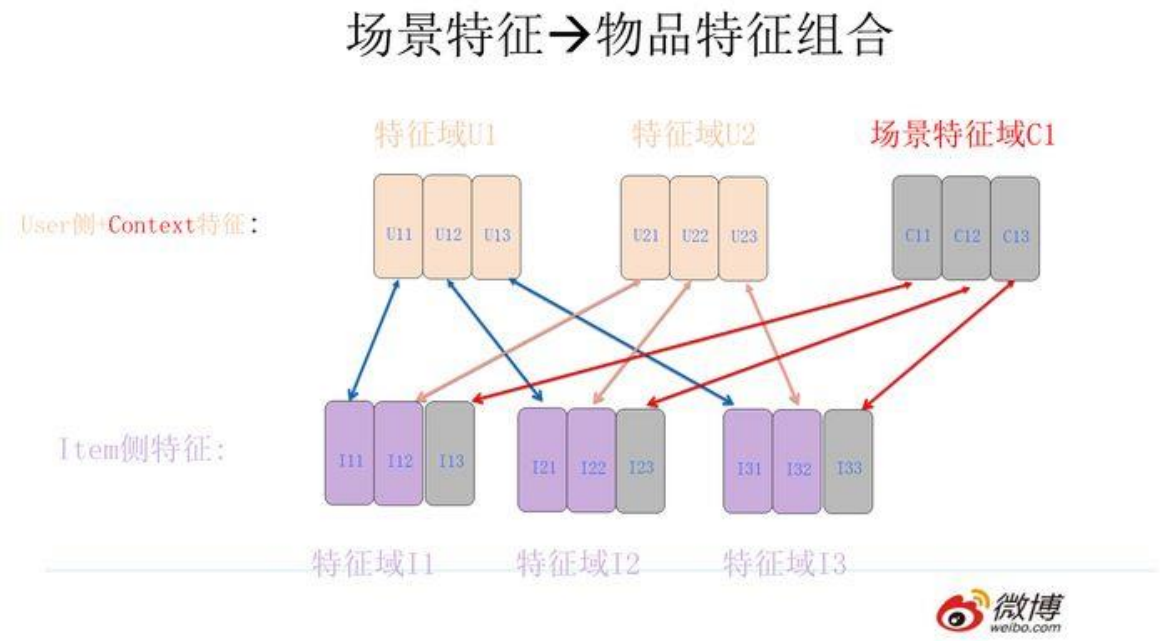
问题二：我们需要考虑上下文特征 C 和用户特征 U 之间的特征组合，也需要考虑 C 和物品特征 I 之间的特征组合。上下文特征有时是非常强的特征。那么，如何做能够将这两对特征组合考虑进来呢？

我们可以这么做：



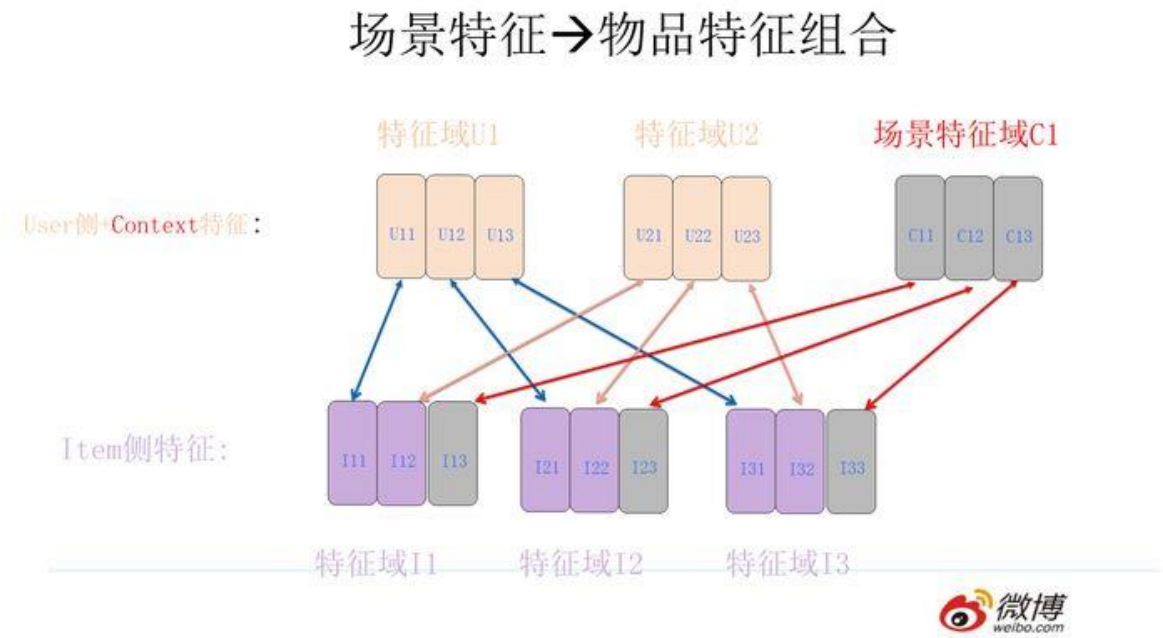
首先，由于上下文特征的动态性，所以给定用户 UID 后，可以在线查询某个上下文特征对应的（F-1）个 embedding 向量，F 是任务特征域的个数。这（F-1）个特征向量可以分成三组：一组是用于拿 Context 特征和用户特征域的特征进行特征组合用的，在我们上面给的例子里，有两个；第二组是拿 Context 特征和物品特征域的特征进行特征组合用的，我们的例子里这个数目是三；第三组是 Context 特征用于自身内部特征组合用的，这个我们先忽略，因为它的做法和上文所述的用户侧及物品侧求内部特征组合的做法是一样的。

为了简化说明，我们假设只有一个 Context 特征，于是它对应了（6-1）=5 个 embedding 向量，其中 2 个是用于和用户侧特征进行组合的，3 个是用于和物品侧特征进行组合的。我们把它们拆分成两组，如上图所示。



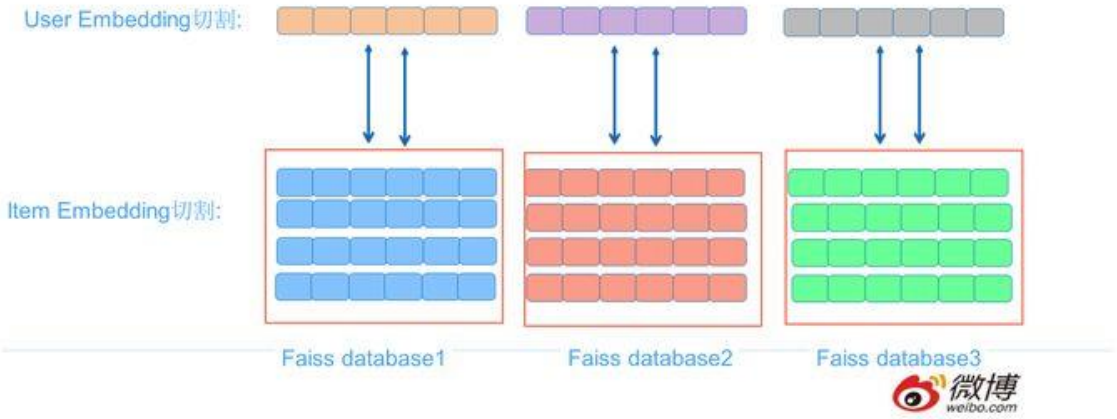
然后，我们来计算上下文特征和用户侧特征如何进行特征组合。如上图所示，其实这个过程和上文讲的用户侧与物品侧的 FFM 特征组合过程是一样的。物品侧和上下文侧特征找到对应的 embedding 向量做内积计算即可。这里不展开讲，如果不理解的话再回头看下上面的叙述。因为这两类特征都在用户发生访问行为的时候能获得，不依赖和物品发生关系，所以这个过程可

以在用户侧在线计算完成。
这个内积数值代表用户特征和上下文特征的二阶特征组合得分，算好备用。



再然后，我们来计算上下文特征和物品侧特征的特征组合，如上图所示。其实很好理解，就相当于在做用户侧特征与物品侧特征组合的时候，在用户侧新加入了几个特征，无非这几个特征是 Context 特征，而其实不是用户侧的特征，但是做法是完全一样的。这样，就可以将 Context 特征打入用户侧 embedding 以及物品侧 embedding，于是 Context 和物品的特征组合问题就解决了。
利用这个用户侧 embedding，用 Faiss 通过内积方式取出 Top K 物品。通过这种方式取出的物品同时考虑到了用户和物品的特征组合< U,I>，以及上下文和物品的特征组合< C,I>。
假设返回的 Top K 物品都带有内积的得分 Score1，再考虑上一步< U,C>的得分 Score，将两者相加对物品重排序（< U,C>因为跟物品无关，所以其实不影响物品排序，如果是召回阶段使用 FM/FFM，是可以不考虑引入的），就得到了最终结果。而这个最终结果，在遵循 FFM 计算原则的基础上，考虑了 U/I/C 两两之间的特征组合。当然，我们可以把上面说的一阶项以及< U,U>/ < I,I>内部特征组合也融入这个系统。
于是我们通过这种手段，构造出了一个完整的 FFM 召回模型。这个召回模型通过构造 user embedding，Context embedding 和 Item embedding，以及充分利用类似 Faiss 这种高效 embedding 计算框架，就构造了高效执行的和 FFM 计算完全等价的召回系统。前文提过，FFM 按照上述方法做，打出来的两个 embedding 长度太长，可能影响 Faiss 的效率。下面提供两个可能的提速方案。
5.沉重的 FFM：并行拉取提速策略

并行拉取策略

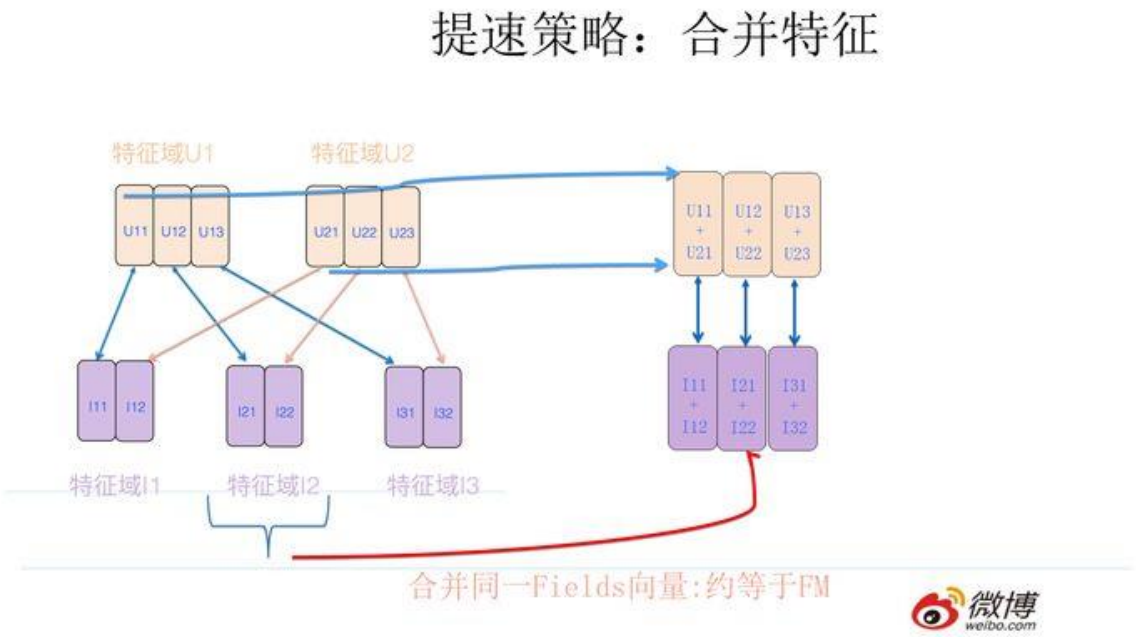


如果用上述方法做 FFM 召回模型，有可能被拉平的用户 embedding 以及物品 embedding 长度太长，这会导致 Faiss 提取速度变慢，以致这个方法因为速度太慢而变得不可行。那么一种比较直接的提速想法就是：把太长的用户 embedding 打断成连续片段，物品 embedding 也相应地打断，同一个物品的 embedding 片段分别存在不同的 Faiss 数据库中，这样由于减少了 embedding 的长度，所以会极大加快 Faiss 的提取速度。

在结果返回时，对每个 User Embedding 片段拉回的 Item 子集合进行合并，同一个物品，把各自的片段内积得分累加，就得到了这个物品相对用户的 FFM 最终得分，很容易推断，这种片段得分累加策略，和作为整体计算长向量内积，两者得分是相同的。按照这个得分对返回的物品重排序，于是就得到了最终计算结果。这是一种典型的并行策略。

虽然，理论上，这个方案能够处理相当长的 embedding 匹配问题。但是，这个方案有个问题：并不能保证返回结果的最终排序和真实排序是一致的。因为有可能某个综合总得分较高的物品没有被从任何一个 Faiss 子数据库拉回来，比如这个物品每个片段的得分都不太高也不太低的情况，是可能发生这种漏召回的情况的。

6.沉重的 FFM：（FM+FFM）混合提速策略



本系列文章中，上篇在介绍 FM 召回模型的时候，可以看出，它的一个特别简洁的方式是把用户侧的特征 embedding 累加，以及物品侧的特征 embedding 累加，所以 FM 打出来的两个 embedding 长度，只跟 k 相关，跟特征数目没关系，无论多少特征，embedding size 恒等于 k。所以看着特别简洁，效率也高。

那么 FFM 是否能够参照 FM 的思路，把一部分特征的 embedding 累加起来，通过这种方式来减小用户侧或物品侧的 embedding 大小呢？我觉得，结论是这样的：如果你坚持做一个原汁原味版的 FFM，是不可能存在类似的特征合并的，因为用户侧和物品侧的做内积的 embedding 向量都是一一对应的，且无公共因子项可提出，所以没有可能进行特征 embedding 合并。

但是，如果我们不是原教旨 FFM 主义分子，一定坚持计算过程完全符合 FFM 计算原则，那么这个事情还是可以做的。参考上图，我觉得可以这么做：不同用户侧的特征，对应 Fileds 的向量直接累加；而在物品侧，则是属于同一个特征域的向量直接累加。这样可以保证用户 embedding 和物品 embedding 大小一致。这样的话，用户侧和物品侧的 embedding size=M*K，比如 M=50,K=10，那么长度是 500，这样的长度还是可以把速度做起来的。

如果参照上面的做法，这其实等价于做了这么个事情：用户侧的特征仍然坚持了 FFM 的计算原则，就是每个特征针对其它不同特征域的组合，采用了不同的特征向量；但是，物品侧的特征向量，因为同一个特征域的（F-1）个特征域合并成一个，类似于这里采取的是 FM 的特征 embedding 思路。所以，这个方法看上去貌似是一个处于 FFM 和 FM 模型之间的一种混合模型。至于效果的话，我估计应该比 FM 好，比 FFM 不如，很可能也介于两者之间。当然，这只是我的分析结论。实际效果如何要通过实验来证明。上面是按照合并物品侧的同一个特征域的特征向量角度来做的。完全也可以反过来，就是去合并用户侧的同一个特征域的特征向量。而如果是那样，则 embedding size=N*K。

好了，经过了一系列补充特性，以及一些性能优化方案，我们就得到了一个完整版本的 FFM 召回模型。上面所讲都是说如何用 FFM 模型来做召回，那么下面我们开始探讨本文开头提出的第一个问题：如何用 FFM 召回模型统一多路召回策略？

如何利用 FFM 模型做统一的召回模型

上文书提到过，目前工业界推荐系统在召回阶段，大多数采用了多路召回策略，比如典型的召回路有：基于用户兴趣标签的召回；基于协同过滤的召回；基于热点的召回；基于地域的召回；基于 Topic 的召回；基于命名实体的召回等等，除此外还有很多其它类型的召回路。

现在我们来探讨下第一个问题：在召回阶段，能否用一个统一的模型把多路召回招安？就是说

改造成利用单个模型，单路召回的模式？具体到这篇文章，就是说能否利用 FFM 模型来把多路召回统一起来？在回答上述问题之前，我估计你会提出疑问：目前大家用多路召回用的好好的，为啥要多此一举，用一个模型把多路召回统一起来呢？这个问题非常好，我们确实应该先看这么做的必要性。

1.统一召回和多路召回优缺点比较我们先来说明下统一召回和多路召回各自的优缺点，我觉得使用统一召回模式，相对多路召回有如下优点：

首先，采用多路召回，每一路召回因为采取的策略或者模型不同，所以各自的召回模型得分不可比较，比如利用协同过滤召回找到的候选 Item 得分，与基于兴趣标签这一路召回找到的候选 Item 得分，完全是不可比较的。这也是为何要用第二阶段 Ranking 来将分数统一的原因。而如果采取统一的召回模型，比如 FM/FFM 模型，那么不论候选项 Item 来自于哪里，它们在召回阶段的得分是完全可比的。

其次，貌似在目前“召回+Ranking”两阶段推荐模型下，多路召回分数不可比这个问题不是特别大，因为我们可以依靠 Ranking 阶段来让它们可比即可。但是其实多路召回分数不可比会直接引发一个问题：对于每一路召回，我们应该返回多少个 Item 是合适的呢？如果在多路召回模式下，这个问题就很难解决。既然分数不可比，那么每一路召回多少候选项 K 就成为了超参，需要不断调整这个参数上线做 AB 测试，才能找到合适的数值。

而如果召回路数特别多，于是每一路召回带有一个超参 K，就是这一路召回多少条候选项，这样的超参组合空间是非常大的。所以到底哪一组超参是最优的，就很难定。其实现实情况中，很多时候这个超参都是拍脑袋上线测试，找到最优的超参组合概率是很低的。而如果假设我们统一用 FM/FFM 模型来做召回，其实就不存在上面这个问题。这样，我们可以在召回阶段做到更好的个性化，比如有的用户喜欢看热门的内容，那么热门内容在召回阶段返回的比例就高，而其它内容返回比例就低。所以，可以认为各路召回的这组超参数就完全依靠 FM 模型调整成个性化的了，很明显这是使用单路单模型做召回的一个特别明显的好处。

再次，对于工业界大型的推荐系统来说，有极大的可能做召回的技术人员和做 Ranking 的技术人员是两拨人。这里隐含着一个潜在可能会发生的问题，比如召回阶段新增了一路召回，但是做 Ranking 的哥们不知道这个事情，在 Ranking 的时候没有把能体现新增召回路特性的特征加到 Ranking 阶段的特征中。这样体现出来的效果是：新增召回路看上去没什么用，因为即使你找回来了，而且用户真的可能点击，但是在排序阶段死活排不上去。也就是说，在召回和排序之间可能存在信息鸿沟的问题，因为目前召回和排序两者的表达模式差异很大，排序阶段以特征为表达方式，召回则以“路 / 策略 / 具体模型”为表达方式，两者之间差异很大，是比较容易产生上述现象的。

但是如果我们采用 FM/FFM 模型来做召回的话，新增一路召回就转化为新增特征的问题，而这一点和 Ranking 阶段在表现形式上是相同的，对于召回和排序两个阶段来说，两者都转化成了新增特征问题，所以两个阶段的改进语言体系统一，就不太容易出现上述现象。

上面三点，是我能想到的采用统一召回模型，相对多路召回的几个好处。但是是不是多路召回一定不如统一召回呢？其实也不是，很明显多路召回这种策略，上线一个新召回方式比较灵活，对线上的召回系统影响很小，因为不同路召回之间没有耦合关系。但是如果采用统一召回，当想新增一种召回方式的时候，表现为新增一种或者几种特征，可能需要完全重新训练一个新的 FM/FFM 模型，整个召回系统重新部署上线，灵活性比多路召回要差。

上面讲的是必要性，讲完了必要性，我们下面探讨如何把多路召回改造成单路召回。

2.如何将多路召回融入 FFM 召回模型

其实，用 FFM 模型统一多路召回，和 FM 模型统一多路召回，基本是一样的，只有些许不同。我们以目前不同类型推荐系统中共性的一些召回策略来说明这个问题，以信息流推荐为例子，传统的多路召回阶段通常包含以下策略：协同过滤，兴趣分类，兴趣标签，兴趣 Topic，兴趣实体，热门物品，相同地域等。这些不同角度的召回策略都是较为常见的。

我们再将上述不同的召回路分为两大类，可以把协同过滤作为一类，其它的作为一类，协同过滤相对复杂，我们先说下其它类别。

对于比如兴趣分类，兴趣标签，热门，地域等召回策略，要把这些召回渠道统一到 FM/FFM 模型相对直观，只需要在训练 FM/FFM 模型的时候，针对每一路的特性，在用户特征端和物品特征端新增对应特征即可。比如对于地域策略，我们可以把物品所属地域（比如微博所提到的地域）和用户的感兴趣地域都作为特征加入 FM/FFM 模型即可。兴趣标签，Topic，兴趣实体等都是类似的。所以大多数情况下，在多路召回模式下你加入新的一路召回，在 FM/FFM 统一召回策略下，对应地转化成了新增特征的方式。

然后我们再说协同过滤这路召回。其实本质上也是将一路召回转化为新加特征的模式。我们以前提到过：本质上 MF 模型这种典型的协同过滤策略，是 FM 模型的一个特例，而 FM 模型又是 FFM 模型的特例，所以其实 MF 模型也是 FFM 模型的特例。MF 可以看作在 FM/FFM 模型里只有 User ID 和 Item ID 这两类（Fields）特征的情形。意思是说，如果我们将 user ID 和 Item ID 作为特征放入 FFM 模型中进行训练，那么 FFM 模型本身就是包含了协同过滤的思想的。

当然，对于超大规模的网站，用户以亿计，物品可能也在千万级别，如果直接把 ID 引入特征可能会面临一些工程效率问题以及数据稀疏的问题。

FM 要想把 ID 特征融入，应该是可行的，因为毕竟每个特征只需要学习一个 k 维大小特征向量，虽然 ID 数量大，但是总还是能接受。但是，如果是在 FFM 召回模型中融入 ID 特征，你会发现这里有个严重的问题：因为每个特征要包含（F-1）个 k 维特征向量，这对于 FFM 来说，ID 特征会有超量的参数需要学习。比如假设 F=101,k=10，UID 有 1 亿个不同 ID。这意味着光 UID 特征，就需要 1000 亿参数，这个……估计你会被吓退。所以，感觉 FFM 是很难把协同特征引入的，除非，事先通过其它方法对 ID 进行协同 embedding 编码，在 FFM 中直接使用，而不作为它的参数。否则，这在参数量以及存储量上来说，是很难做到的。

在具体实施统一多路召回的时候，可以沿着这个路径逐步替换线上的多路召回：先用 FM/FFM 模型替换一路召回，线上替换掉；再新加入某路特征，这样上线，就替换掉了两路召回；如此往复逐渐把每一路召回统一到一个模型里。这是比较稳的一种替换方案。当然如果你是个猛人，直接用完整的 FFM 召回模型一步替换掉线上的各路召回，也，未尝不可。只要小流量 AB 测试做好也没啥。

FFM 模型能将召回和排序阶段一体化吗

我们在前文讲过，召回和排序各司其职。召回主要考虑泛化性并把候选物品集合数量降下来；排序则主要负责根据用户特征 / 物品特征 / 上下文特征对物品进行精准排名。
那么，我们现在可以来审视下本文开头提出的第二个问题了：FFM 模型能否将常见的两阶段模型一体化？即是否能将实用化的推荐系统通过 FFM 召回模型简化为单阶段模型？意思是推荐系统是否能够只保留 FFM 召回这个模块，绕过后续的排序阶段，FFM 召回模块按照得分排序直接作为推荐结果返回。我们可以这么做吗？
这取决于 FFM 召回模型是否能够一并把原先两阶段模型的两个职责都能承担下来。这句话的意思是说，FFM 召回模型如果直接输出推荐结果，那么它的速度是否足够快？另外，它的精准程度是否可以跟两阶段模型相媲美？不会因为少了第二阶段的专门排序环节，而导致推荐效果变差？如果上面两个问题的答案都是肯定的，那么很明显 FFM 模型就能够将现有的两阶段推荐过程一体化。在本系列的第一篇介绍 FM 召回模型的文章里，分析结论是：FM 模型无论在推荐精准性，还是推荐速度方面，应该是能够同时承载两阶段模型的功能的。
那么 FFM 召回模型也可以担任类似的重任吗？我的答案是：It Depends。要看情况，跟应用的复杂情况有关。
如果从推荐的精准性角度考虑，假设我们能够把排序阶段的特征都引入 FFM 召回模型，那么应该能够得到等价的排序结果，这个很好理解，因为这等于你在召回部分复制了一个完全相同的 FFM 排序模型，类似于把排序功能前置到了召回阶段，所以推荐精准度基本等价。
看着好像这个事情是能做的是吧？其实不然。
在前文我们分析过如何用 FFM 模型来做召回模型，你会再次发现 FFM 模型的特性，就是太沉重。这种“沉重性”在召回阶段，表现为：用 FFM 模型打出来的用户 Embedding 长度太长，如果用户侧有 M 个特征域，物品侧有 N 个特征域，单个特征 embedding 向量大小为 K,先不考虑上下文特征域，打出来的用户 Embedding size=M*N*K。而这个长度是很容易失控的。
如果这个长度太长，意味着单机版本的 Faiss 速度肯定是跟不上的，那这个事情就得搁浅。而如果长度可以接受，Faiss 速度 OK，那么这事情就能成。所以关键是这个 M*N*K 到底有多长。于是问题转换成了：M、N 和 K，各自大约有多大？
我们拿一个工业级的 CTR 数据 Criteo 来说明（4500 万数据，39 个特征域，为了好计算，我们假设是 40 个特征域）。先说 K，这是单个特征向量的大小，在 Criteo 这种工业级的数据规模下，实验证明，K=8 效果最好。如果 FFM 模型只是用在召回阶段，后面还会再接上排序模型，也就是两阶段模式，k 主观随意设置小点，比如 2 到 4，问题不太大，因为推荐的精准性还可以依赖排序模型来保证。而现在我们对 FFM 模型的期待更多，希望它一步把排序也做掉，于是这个 k 就不能调小，就得是 8，否则推荐效果受影响。
再来说 M 和 N，我们假设仍然是这个数据集，它有 40 个域，我们再假设这些特征域在用户侧和物品侧平分，就是:M=N=20。
于是我们可以算出，如果用 FFM 模型来做 Criteo 数据的召回模型，打出来的用户侧 embedding 大小为：M*N*k=20*20*8=3200。如果采用单机版本的 Faiss 做，这速度估计是跟不上的。如果采用上文讲的对用户侧 embedding 分布式切割的思路，比如把这个 embedding 切成 10 份，那么速度应该是能接受的，但是前面也说过，这可能对推荐精度有损失。
当然，我们也可以采取上文提到的（FM+FFM）嫁接版本来做，如果是这样，打出来的用户侧 embedding size=M*K 或 N*K。对应 Criteo 数据来说，这个长度就是 160，这对于 Faiss 来说，速度绝对不是问题，所以是可以充当一体化模型的。但是效果估计比不了原汁原味版本的 FFM。
另外，如果排序包含 ID 特征，估计 FFM 召回模型也比较难以承担这个重任。
从上面这个实际例子来看，是否能使用 FFM 模型来做一体化推荐模型？这个问题的答案其实取决于任务复杂度，也就是特征域的个数，很明显结论是：如果特征域数量比较少，那么 FFM 模型是可行的，如果特征域数量比较多，则这事情做不了。除非，你愿意采取 embedding 分段切割模式损失精度，或者采取（FM+FFM）嫁接版本，而这也可能会损失精度。
当然，上面都是分析结果，并非实测，所以不能确定实际应用起来也能达到上述理论分析的效

果。
尾声
在本文开头，我提到过一枚硬币的故事。贾昆非常看好艾莉亚，认为她会成为最出色的无面者，也即刺客，想带走艾莉亚去狭海对面，但被艾莉亚拒绝了。贾昆给了艾莉亚一枚旧硬币，并告诉她，如果有朝一日她要找他，可以把这枚旧硬币交给布拉佛斯的任何一个人，并对他说“Valar Morghulis”。
两人自此分别，长久未见。在此期间，少女艾莉亚四处漂泊，历经苦厄，当有一天，发现家国残破，亲人或离散无音讯，或死亡不可追。世界虽大，无处容身，想起了贾昆曾经说过的话，于是远涉重洋来到传说中的布拉佛斯，在码头，她将硬币递给一位船夫，并说：“凡人皆有一死”。船夫应答：凡人皆须侍奉，之后将艾莉亚带到布拉夫斯的黑白之院。开门的是一位须发皆白的老人，一番谈话后，当老人撕下面具的时候，露出了贾昆的真面容。艾莉亚惊诧疑惑地问到：“你究竟是谁？”贾昆缓声回答：“无名之辈（No One），而这也是你的宿命”。从跨入黑白之院的大门起，艾莉亚步入了只属于她自己的，历尽艰险，追求成为“No One”的无尽宿命中。
谜底揭晓，故事也讲完了。
如果一句话归纳本文头尾故事的话，正所谓：
少年心事当拿云，谁念幽寒坐鸣呃。

32、什么是 DSSM?有什么优缺点?

DSSM（Deep Structured Semantic Models）的原理很简单，通过搜索引擎里 Query 和 Title 的海量的点击曝光日志，用 DNN 把 Query 和 Title 表达为低纬语义向量，并通过 cosine 距离来计算两个语义向量的距离，最终训练出语义相似度模型。该模型既可以用来预测两个句子的语义相似度，又可以获得某句子的低纬语义向量表达。
优点:DSSM 用字向量作为输入既可以减少切词的依赖，又可以提高模型的范化能力，因为每个汉字所能表达的语义是可以复用的。另一方面，传统的输入层是用 Embedding 的方式（如 Word2Vec 的词向量）或者主题模型的方式（如 LDA 的主题向量）来直接做词的映射，再把各个词的向量累加或者拼接起来，由于 Word2Vec 和 LDA 都是无监督的训练，这样会给整个模型引入误差，DSSM 采用统一的有监督训练，不需要在中间过程做无监督模型的映射，因此精准度会比较高。
缺点：DSSM 采用词袋模型（BOW），因此丧失了语序信息和上下文信息。另一方面，DSSM 采用弱监督、端到端的模型，预测结果不可控。

33、请简要说说 Wide&Deep、DeepFM 这些模型之间的联系与区别

作者：王喆

链接：https://www.zhihu.com/question/20830906/answer/681688041

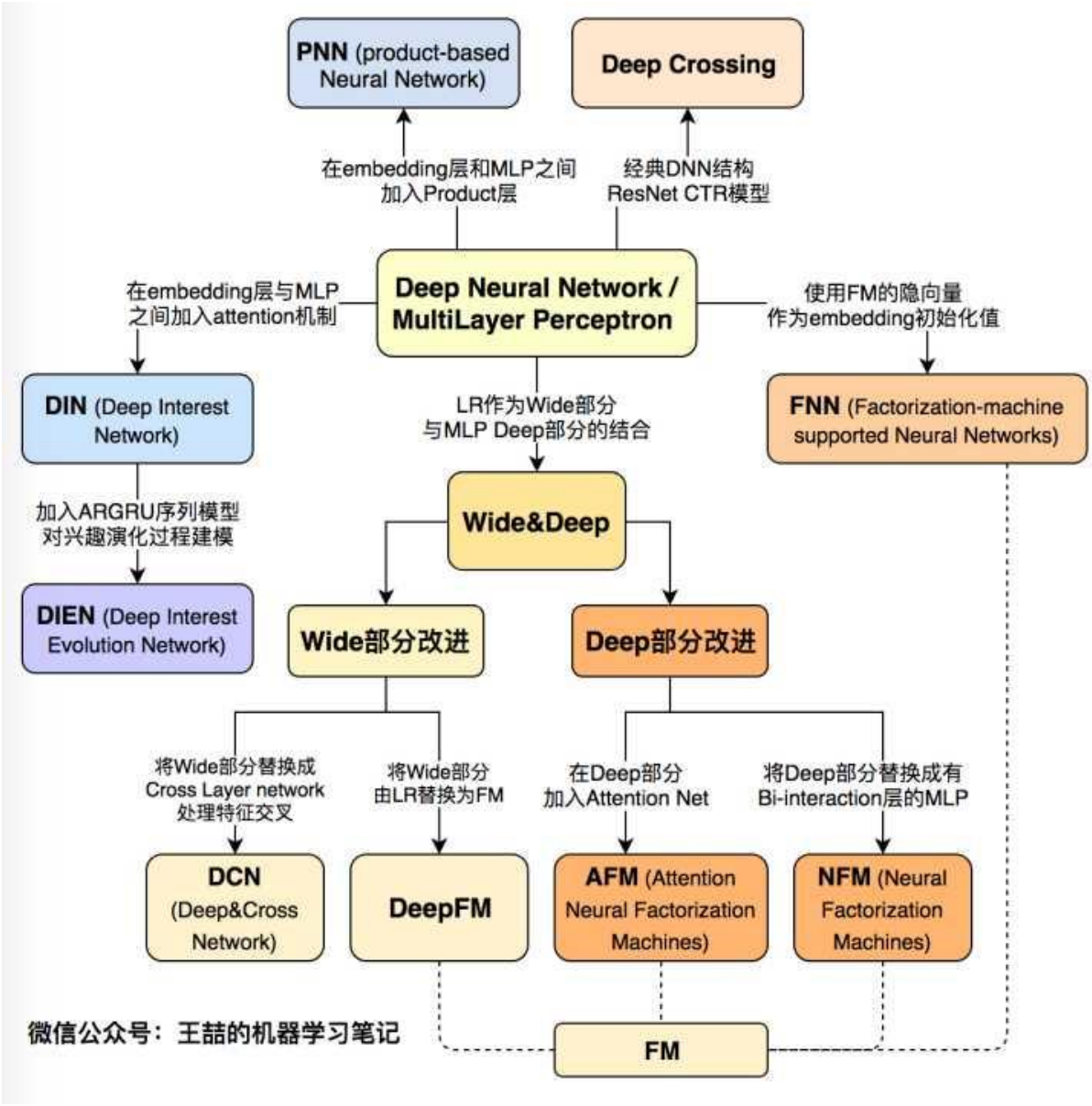
来源：知乎

这个问题是 2013 年提问的，时间来到 2019 年，deep learning 不仅能够来做推荐系统，而且已经成为推荐系统的主流了。我在我的知乎专栏和微信公众号：王喆的机器学习笔记（wangzhenotes）中从 CTR 预估的角度总结了最近三年来主流的深度学习推荐系统的演化图，希望能够帮助到大家。

随着微软的 Deep Crossing，Google 的 Wide&Deep，以及 FNN，PNN 等一大批优秀的深度学习 CTR 预估模型在 2016 年被提出，计算广告和推荐系统领域全面进入了深度学习时代，本文总结了广告、推荐领域最为流行的 10 个深度学习 CTR 模型的结构特点，构建了它们之间的演化图谱。选择模型的标准尽量遵循下面三个原则：

1. 模型的在业界影响力较大的；
2. 已经被 Google，微软，阿里等知名互联网公司成功应用的；
3. 工程导向的，而不是仅用实验数据验证或学术创新用的。

下面首先列出这张深度学习 CTR 模型的演化图谱，再对其进行逐一介绍：


PNN (product-based Neural Network)
Deep Crossing
在embedding层和MLP之间加入Product层
经典DNN结构 ResNet CTR模型
Deep Neural Network / MultiLayer Perceptron
在embedding层与MLP之间加入attention机制
使用FM的隐向量作为embedding初始化值
DIN (Deep Interest Network)
LR作为Wide部分与MLP Deep部分的结合
FNN (Factorization-machine supported Neural Networks)
加入ARGRU序列模型对兴趣演化过程建模
Wide&Deep
DIEN (Deep Interest Evolution Network)
Wide部分改进
Deep部分改进
将Wide部分替换成Cross Layer network处理特征交叉
将Wide部分由LR替换为FM
在Deep部分加入Attention Net
将Deep部分替换成有Bi-interaction层的MLP
DCN (Deep&Cross Network)
DeepFM
AFM (Attention Neural Factorization Machines)
NFM (Neural Factorization Machines)
FM
微信公众号：王喆的机器学习笔记

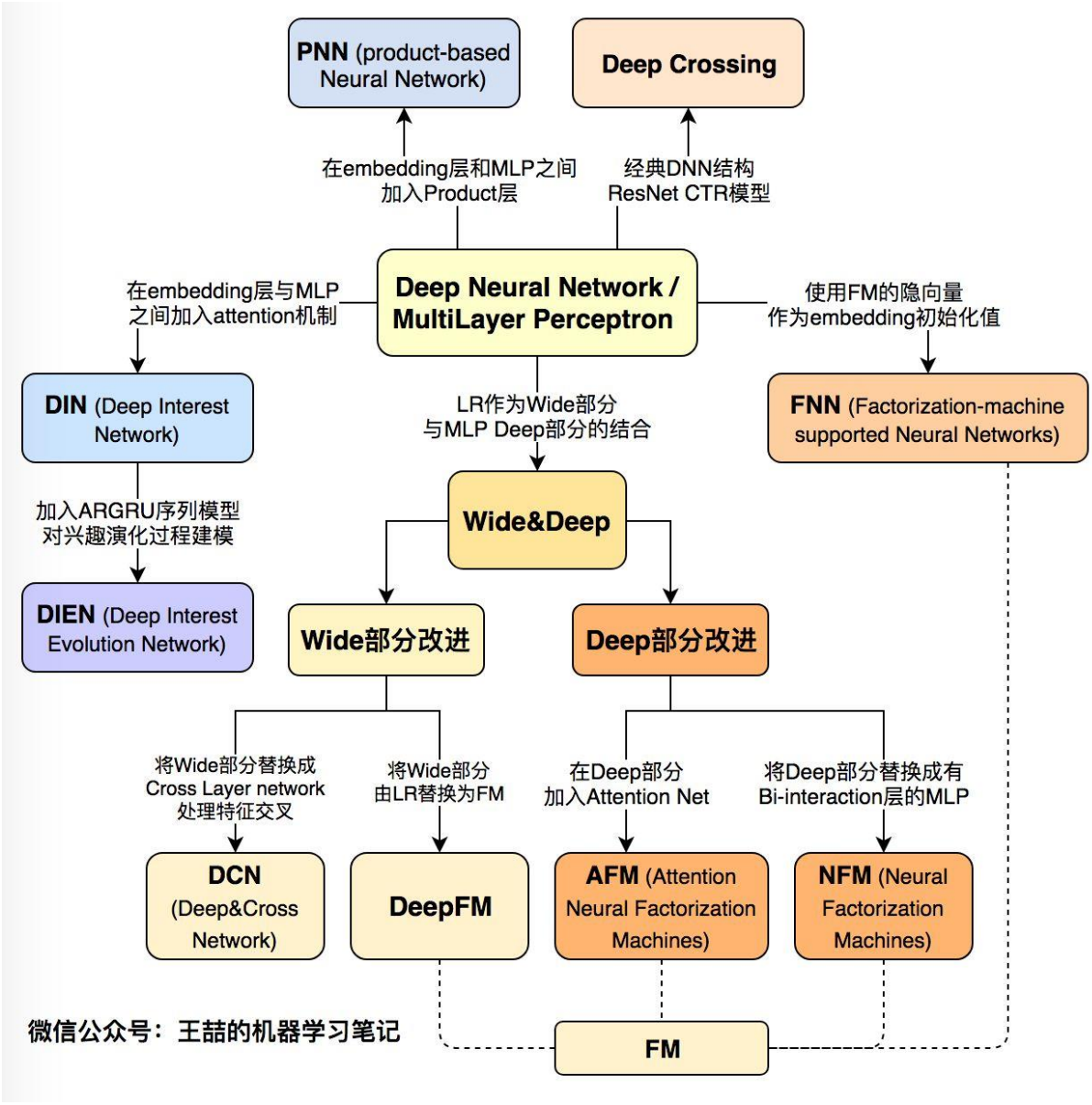
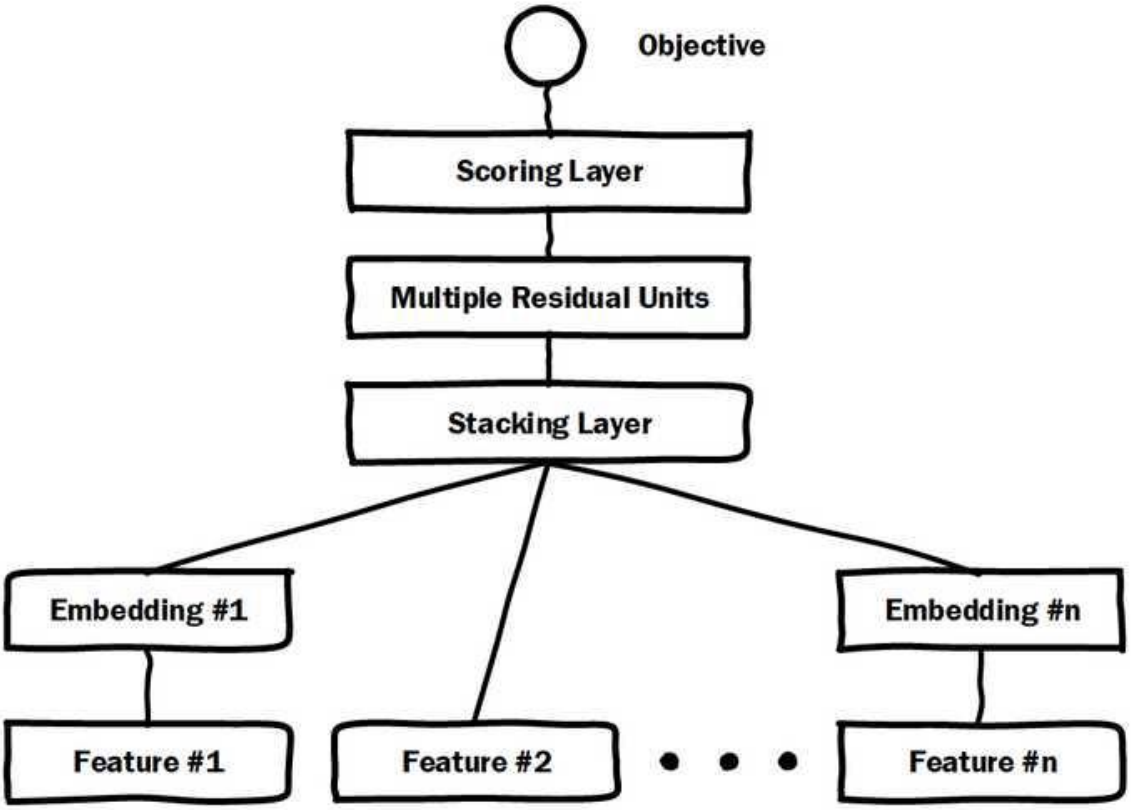


图 1 深度学习 CTR 模型演化图谱一、微软 Deep Crossing（2016 年）——深度学习 CTR 模型的 base model


Objective
Scoring Layer
Multiple Residual Units
Stacking Layer
Embedding #1
Embedding #n
Feature #1
Feature #2
Feature #n

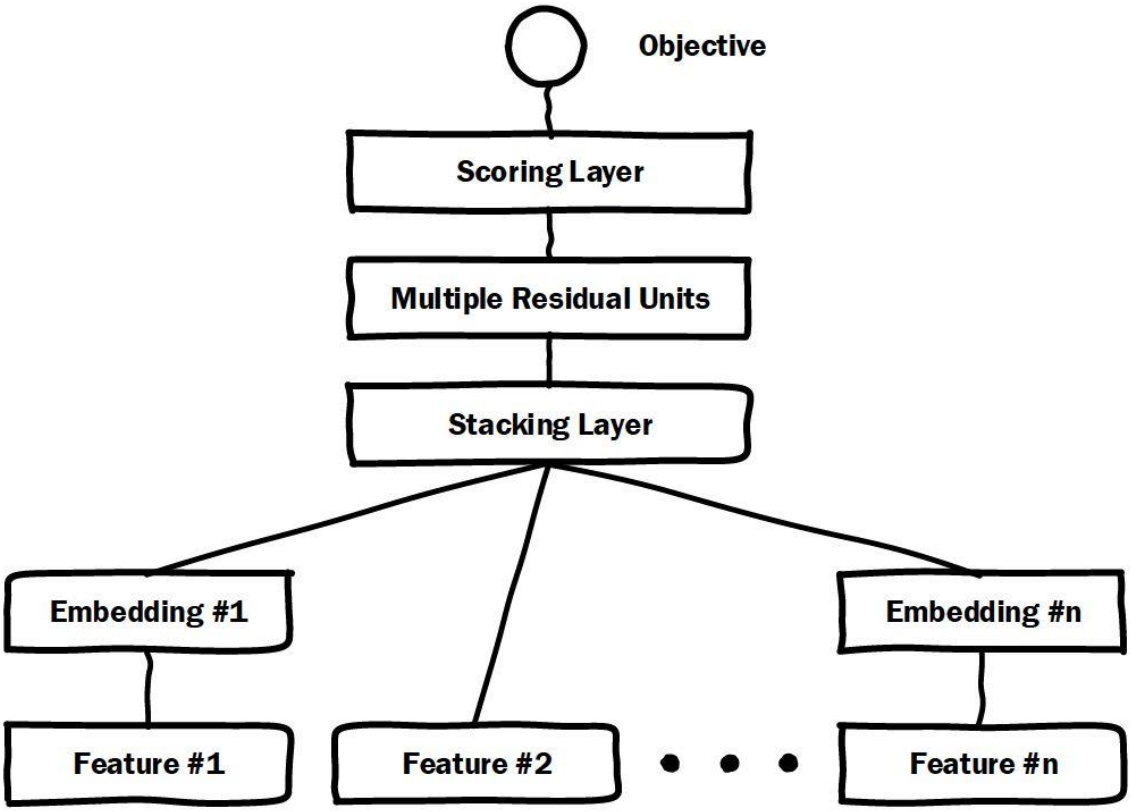
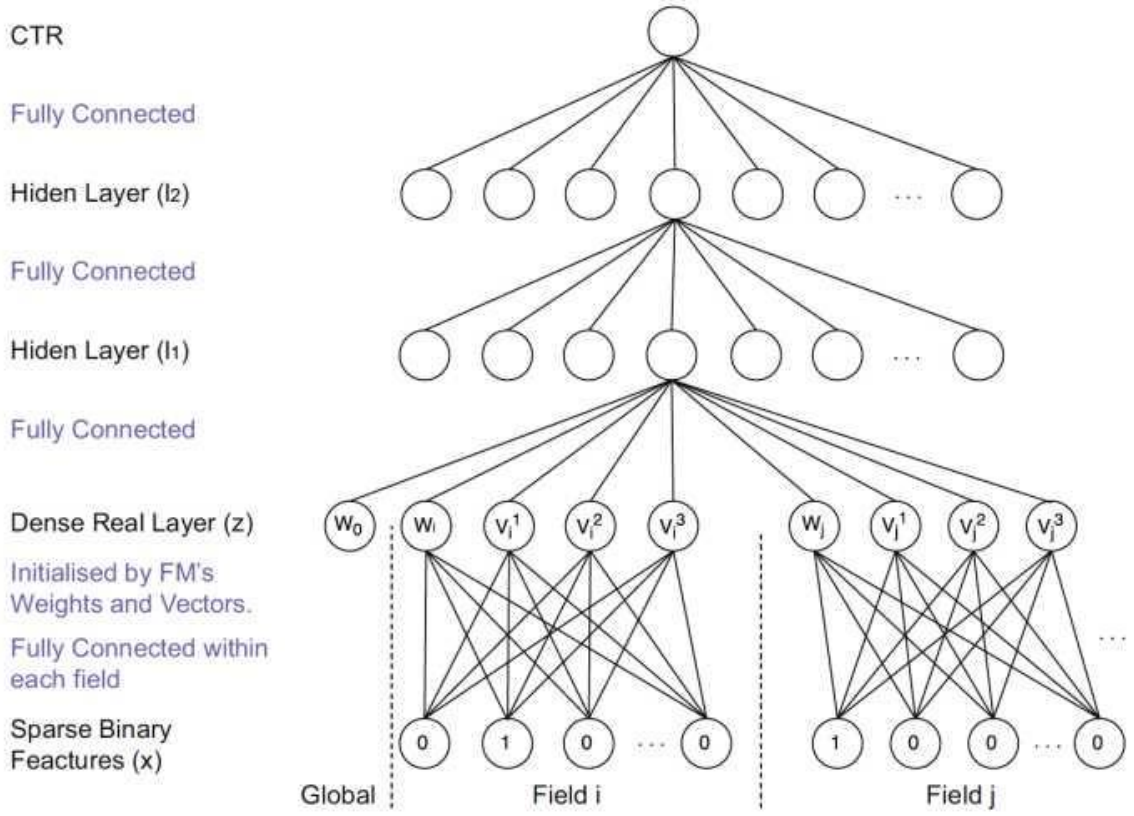

Objective
Scoring Layer
Multiple Residual Units
Stacking Layer
Embedding #1
Embedding #n
Feature #1
Feature #2
Feature #n

图 2 微软 Deep Crossing 模型架构图微软于 2016 年提出的 Deep Crossing 可以说是深度学习 CTR 模型的最典型和基础性的模型。如图 2 的模型结构图所示，它涵盖了深度 CTR 模型最典型的要素，即通过加入 embedding 层将稀疏特征转化为低维稠密特征，用 stacking layer，或者叫做 concat layer 将分段的特征向量连接起来，再通过多层神经网络完成特征的组合、转换，最终用 scoring layer 完成 CTR 的计算。跟经典 DNN 有所不同的是，Deep crossing 采用的 multilayer perceptron 是由残差网络组成的，这无疑得益于 MSRA 著名研究员 DOC_LINK 何恺明提出的著名的 152 层 ResNet。

论文:

二、FNN（2016 年）——用 FM 的完成 Embedding 初始化



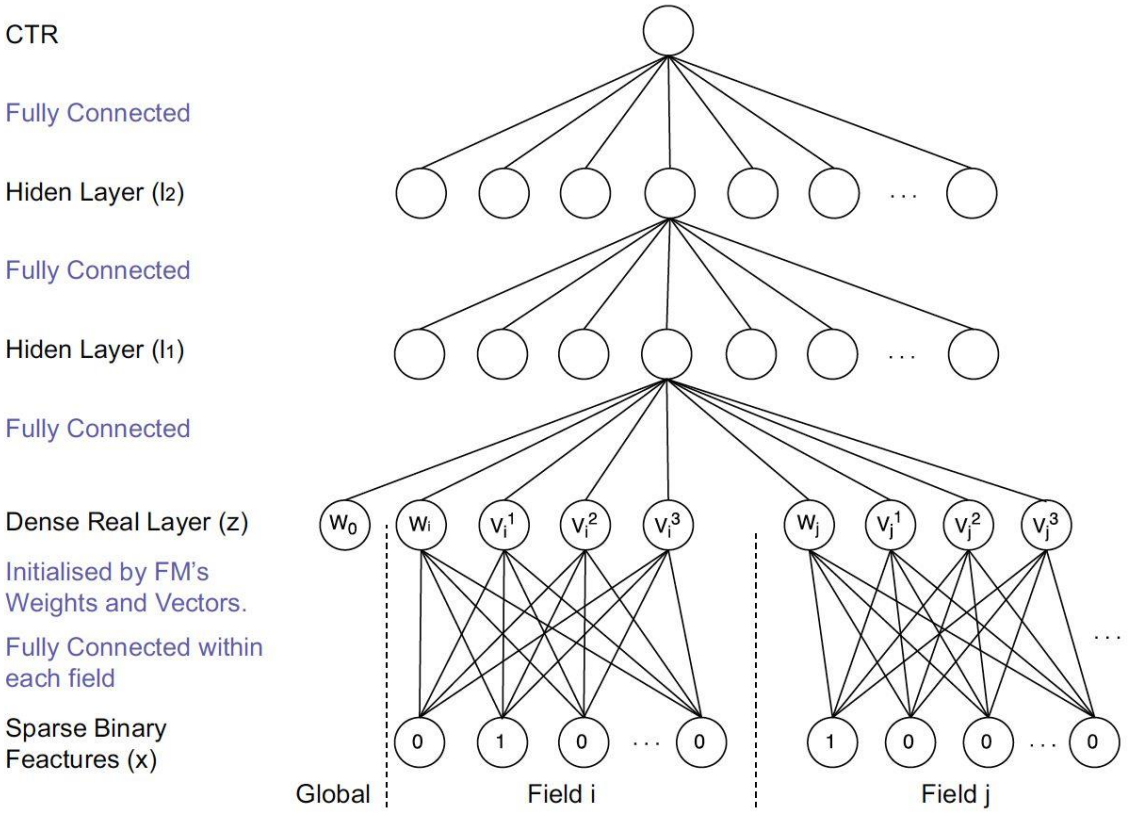


图 3 FNN 模型架构图 FNN 相比 Deep Crossing 的创新在于使用 FM 的隐层向量作为 user 和 item 的 Embedding，从而避免了完全从随机状态训练 Embedding。由于 id 类特征大量采用 one-hot 的编码方式，导致其维度极大，向量极稀疏，所以 Embedding 层与输入层的连接极多，梯度下降的效率很低，这大大增加了模型的训练时间和 Embedding 的不稳定性，使用 pre train 的方法完成 Embedding 层的训练，无疑是降低深度学习模型复杂度和训练不稳定性的有效工程经验。

论文:

三、PNN (2016 年)——丰富特征交叉的方式

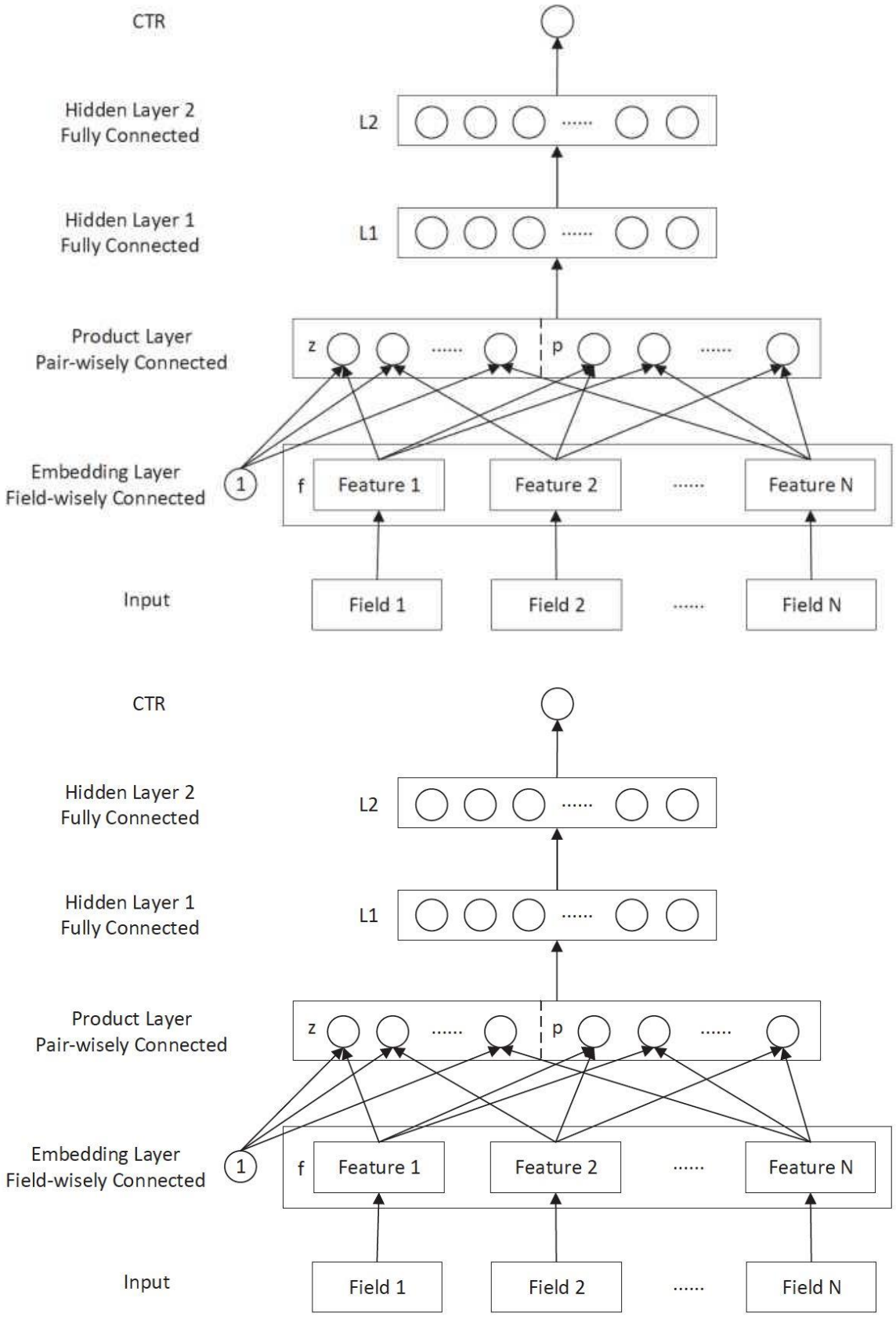
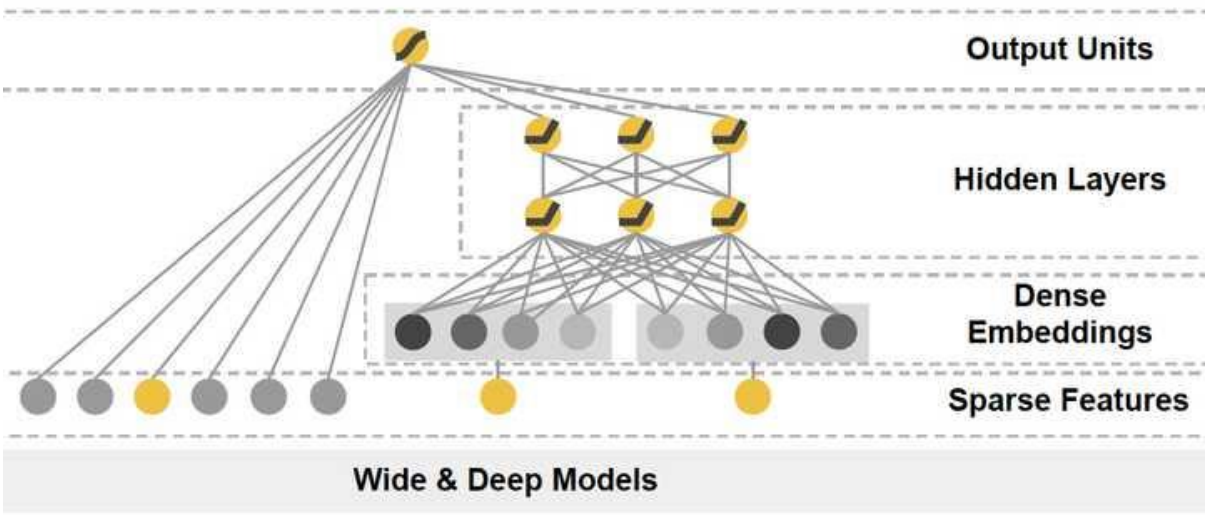


图 4 PNN 模型架构图 PNN 的全称是 Product-based Neural Network，PNN 的关键在于在 embedding 层和 DOC_LINK 全连接层之间加入了 Product DOC_LINK layer。传统的 DNN 是直

接通过多层全连接层完成特征的交叉和组合的，但这样的方式缺乏一定的“针对性”。首先全连接层并没有针对不同特征域之间进行交叉；其次，全连接层的操作也并不是直接针对特征交叉设计的。但在实际问题中，特征交叉的重要性不言而喻，比如年龄与性别的交叉是非常重要的分组特征，包含了大量高价值的信息，我们急需深度学习网络能够有针对性的结构能够表征这些信息。因此 PNN 通过加入 Product layer 完成了针对性的特征交叉，其 product 操作在不同特征域之间进行特征组合。并定义了 inner product，outer product 等多种 product 的操作捕捉不同的交叉信息，增强模型表征不同数据模式的能力 。

论文:

四、Google Wide&Deep（2016 年）——记忆能力和泛化能力的综合权衡



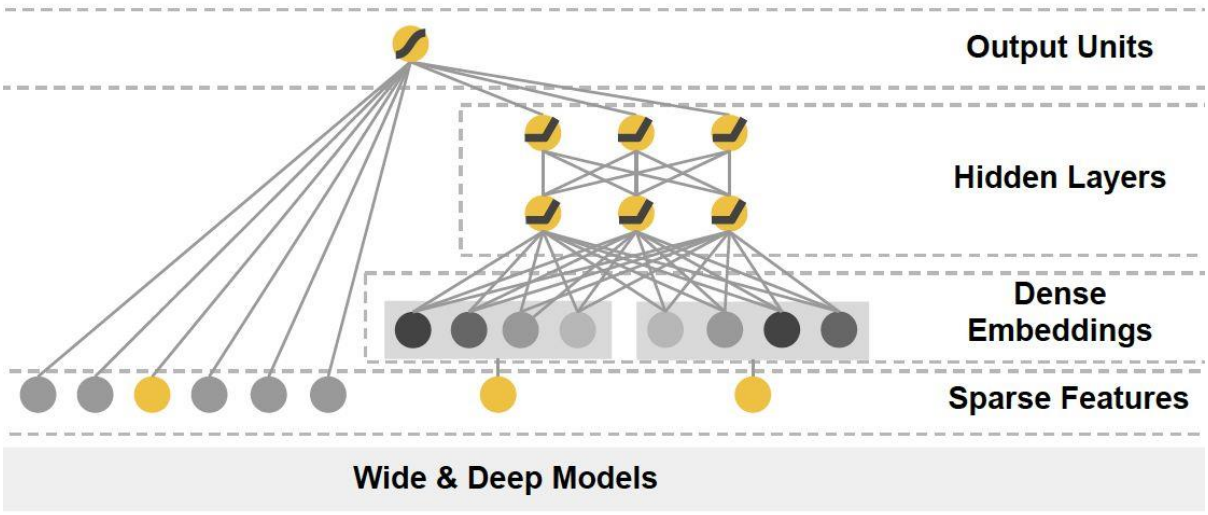


图 5 Google Wide&Deep 模型架构图 Google Wide&Deep 模型的主要思路正如其名，把单输入层的 Wide 部分和经过多层 DOC_LINK 感知机的 Deep 部分连接起来，一起输入最终的输出层。其中 Wide 部分的主要作用是让模型具有记忆性（Memorization），单层的 Wide 部分善于处理大量稀疏的 id 类特征，便于让模型直接“记住”用户的大量历史信息；Deep

部分的主要作用是让模型具有“DOC_LINK 泛化性”（Generalization），利用 DNN 表达能力强的特点，挖掘藏在特征后面的数据模式。最终利用 LR 输出层将 Wide 部分和 Deep 部分组合起来，形成统一的模型。Wide&Deep 对之后模型的影响在于——大量深度学习模型采用了两部分甚至多部分组合的形式，利用不同网络结构挖掘不同的信息后进行组合，充分利用和结合了不同网络结构的特点。

论文:

五、华为 DeepFM (2017 年)——用 FM 代替 Wide 部分

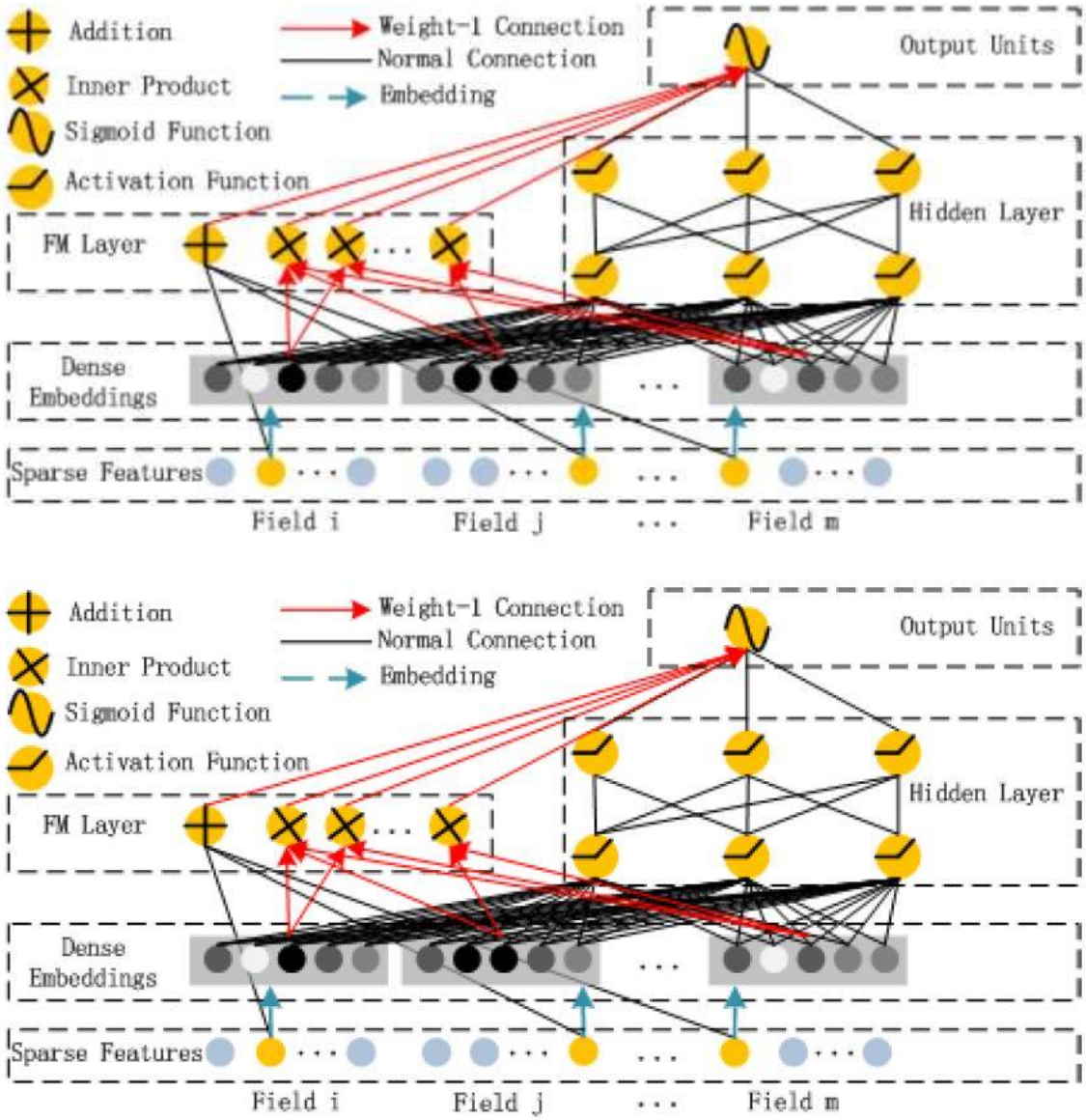
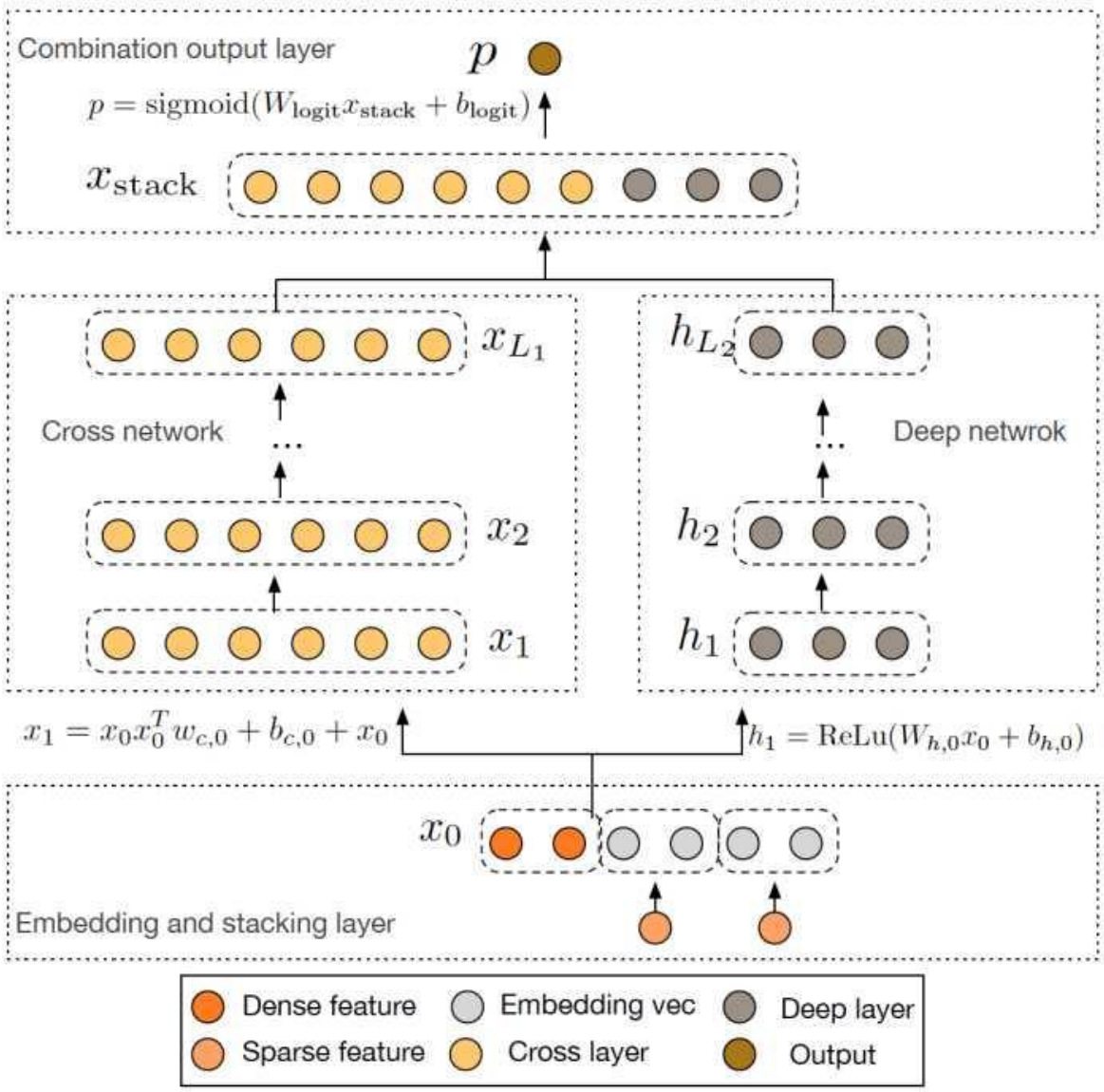


图 6 华为 DeepFM 模型架构图在 Wide&Deep 之后，诸多模型延续了双网络组合的结构，DeepFM 就是其中之一。DeepFM 对 Wide&Deep 的改进之处在于，它用 FM 替换掉了原来的 Wide 部分，加强了浅层网络部分特征组合的能力。事实上，由于 FM 本身就是由一阶部分和二阶部分组成的，DeepFM 相当于同时组合了原 Wide 部分+二阶特征交叉部分+Deep 部分三种结构，无疑进一步增强了模型的表达能力。

论文:

六、Google Deep&Cross（2017 年）——使用 Cross 网络代替 Wide 部分



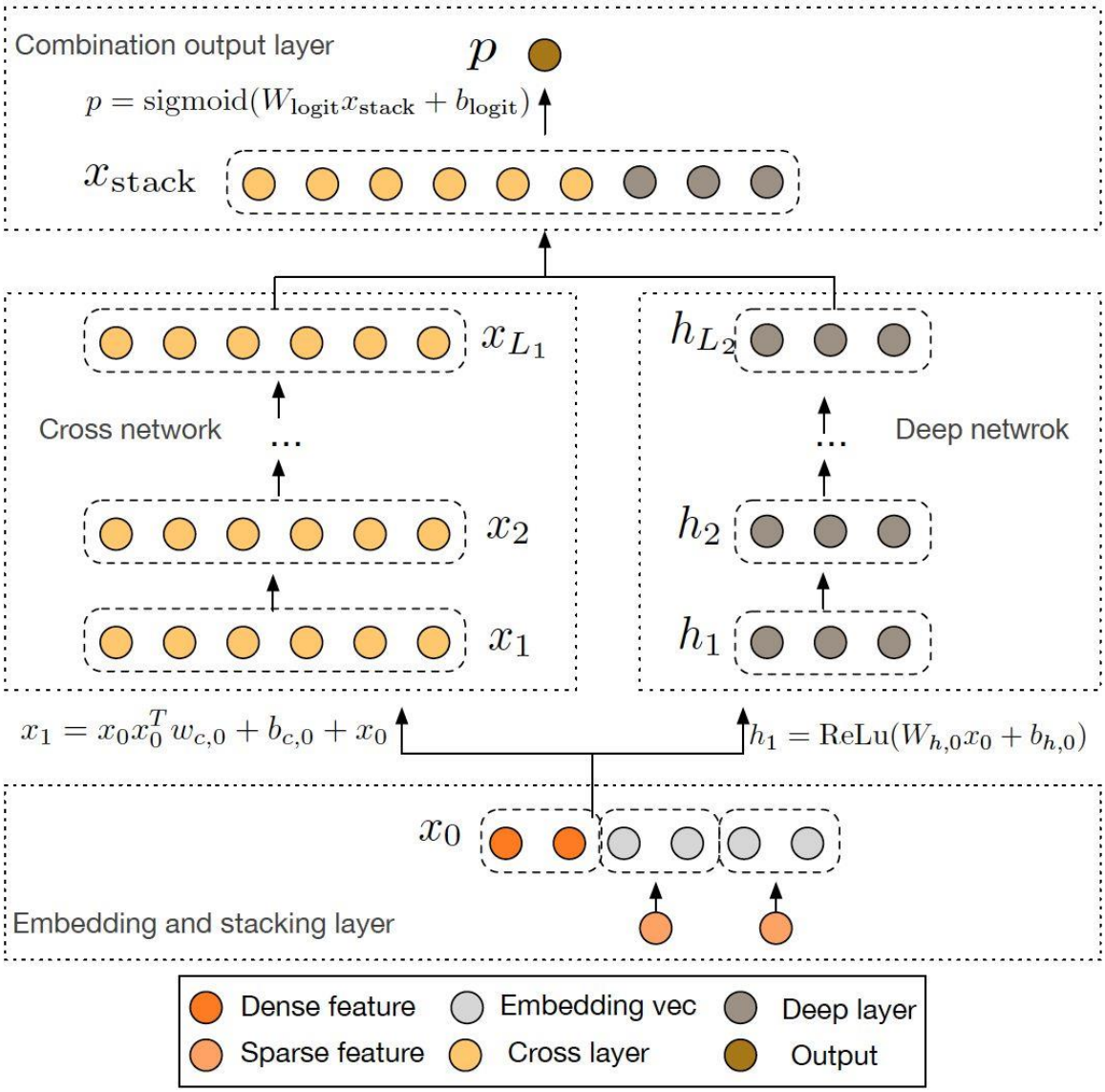


图 7 Google Deep Cross Network 模型架构图 Google 2017 年发表的 Deep&Cross Network（DCN）同样是对 Wide&Deep 的进一步改进，主要的思路使用 Cross 网络替代了原来的 Wide 部分。其中设计 Cross 网络的基本动机是为了增加特征之间的交互力度，使用多层 cross layer 对输入向量进行特征交叉。单层 cross layer 的基本操作是将 cross layer 的输入向量 xl 与原始的输入向量 x0 进行交叉，并加入 bias 向量和原始 xl 输入向量。DCN 本质上还是对 Wide&Deep Wide 部分表达能力不足的问题进行改进，与 DeepFM 的思路非常类似。

论文:

七、NFM（2017 年）——对 Deep 部分的改进

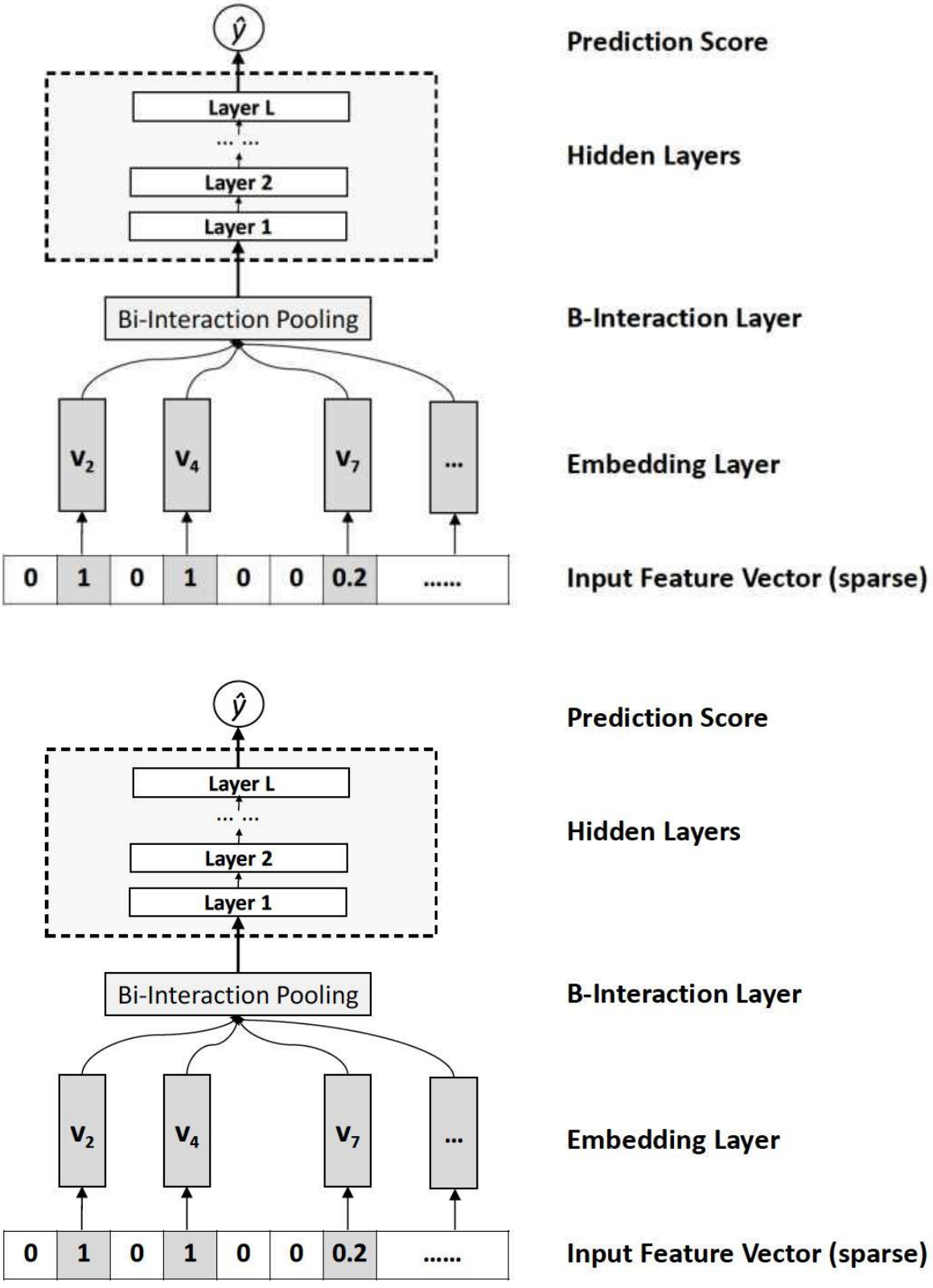


图 8 NFM 的深度网络部分模型架构图相对于 DeepFM 和 DCN 对于 Wide&Deep Wide 部分的改进，NFM 可以看作是对 Deep 部分的改进。NFM 的全称是 Neural Factorization Machines，如果我们从深度学习网络架构的角度看待 FM，FM 也可以看作是由单层 LR 与二阶特征交叉组成的 Wide&Deep 的架构，与经典 W&D 的不同之处仅在于 Deep 部分变成了二阶隐向量相乘

的形式。再进一步，NFM 从修改 FM 二阶部分的角度出发，用一个带 Bi-interaction Pooling 层的 DNN 替换了 FM 的特征交叉部分，形成了独特的 Wide&Deep 架构。其中 Bi-interaction Pooling 可以看作是不同特征 embedding 的 element-wise product 的形式。这也是 NFM 相比 Google Wide&Deep 的创新之处。

论文：

八、AFM（2017 年）——引入 Attention 机制的 FM

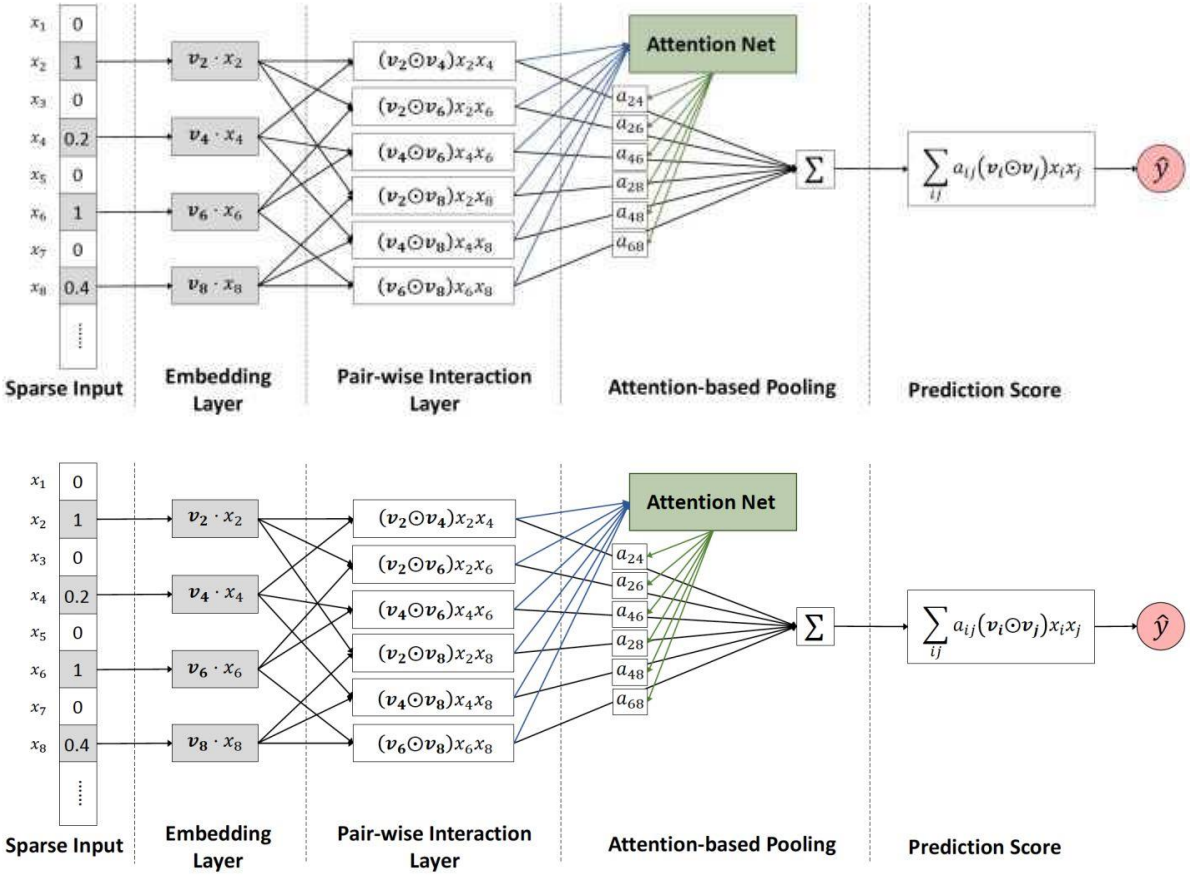


图 9 AFM 模型架构图 AFM 的全称是 Attentional Factorization Machines，通过前面的介绍我们很清楚的知道，FM 其实就是经典的 Wide&Deep 结构，其中 Wide 部分是 FM 的一阶部分，Deep 部分是 FM 的二阶部分，而 AFM 顾名思义，就是引入 Attention 机制的 FM，具体到模型结构上，AFM 其实是对 FM 的二阶部分的每个交叉特征赋予了权重，这个权重控制了交叉特征对最后结果的影响，也就非常类似于 NLP 领域的 DOC_LINK 注意力机制（Attention Mechanism）。为了训练 Attention 权重，AFM 加入了 Attention Net，利用 Attention Net 训练好 Attention 权重后，再反向作用于 FM 二阶交叉特征之上，使 FM 获得根据样本特点调整特征权重的能力。

论文：

九、阿里 DIN（2018 年）——阿里加入 Attention 机制的深度学习网络

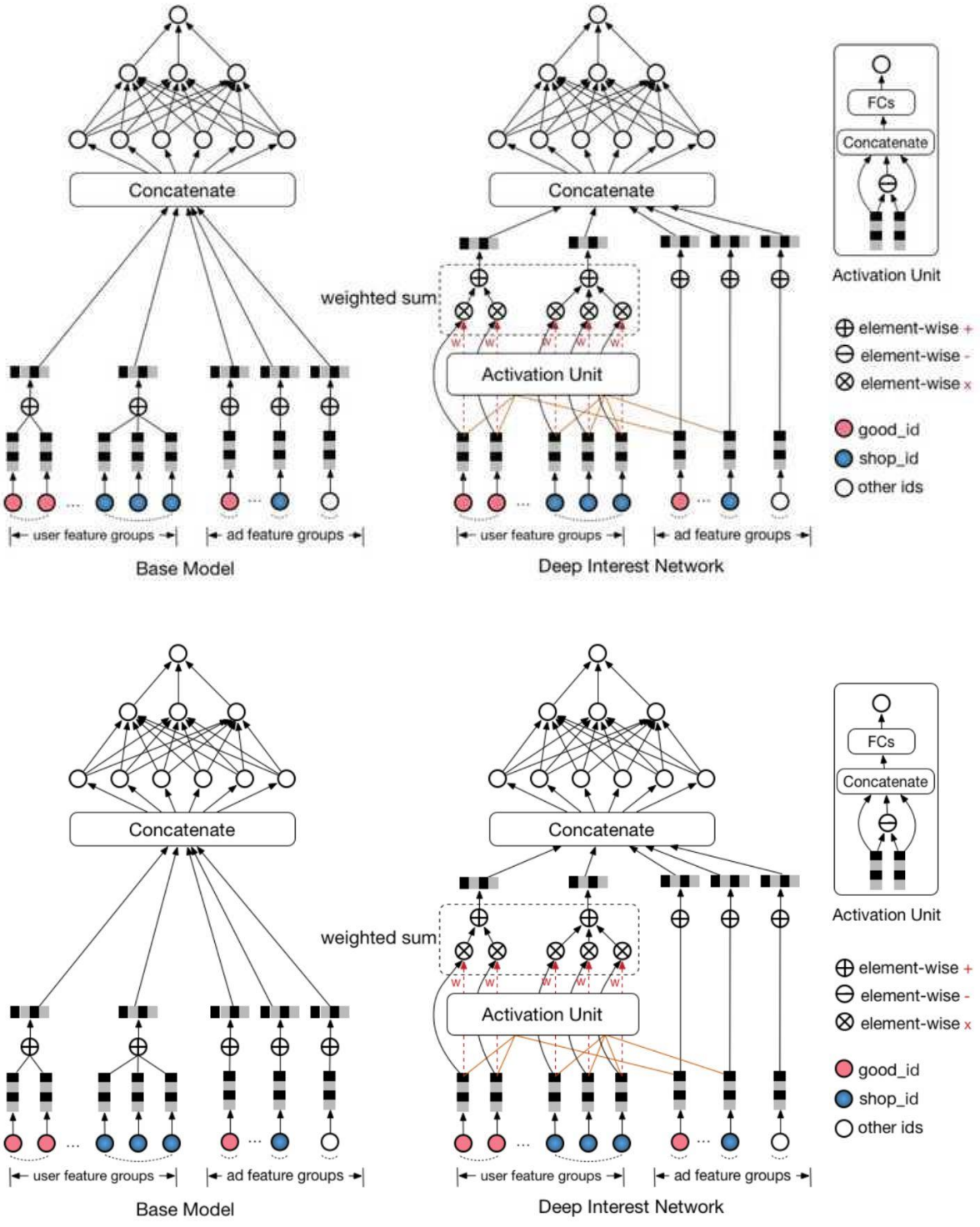
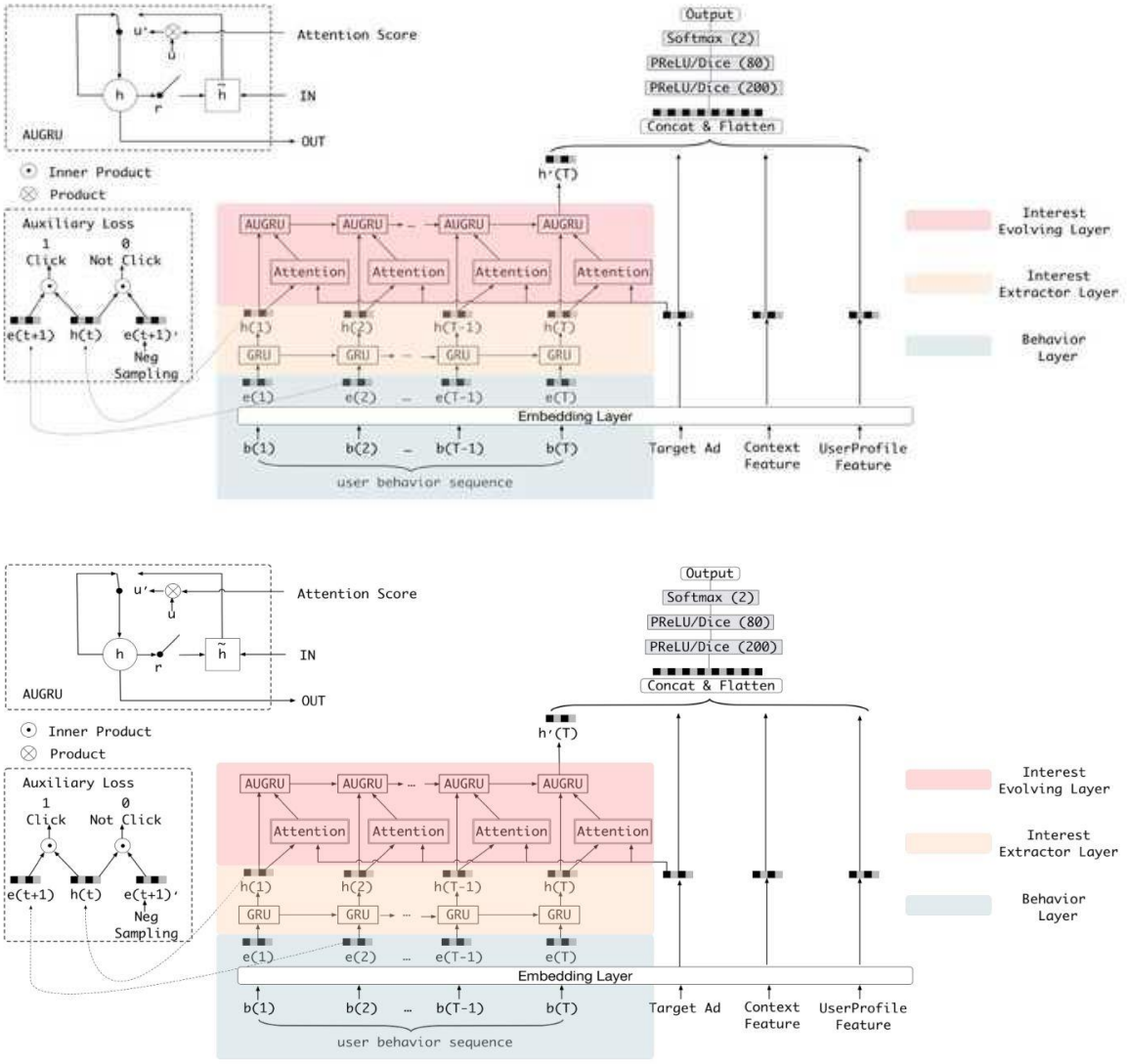


图 10 阿里 DIN 模型与 Base 模型的架构图 AFM 在 FM 中加入了 Attention 机制，2018 年，阿里巴巴正式提出了融合了 Attention 机制的深度学习模型——Deep Interest Network。与 AFM 将 Attention 与 FM 结合不同的是，DIN 将 Attention 机制作用于深度神经网络，在模型的 embedding layer 和 concatenate layer 之间加入了 attention unit，使模型能够根据候选商品的不同，调整不同特征的权重。

论文：

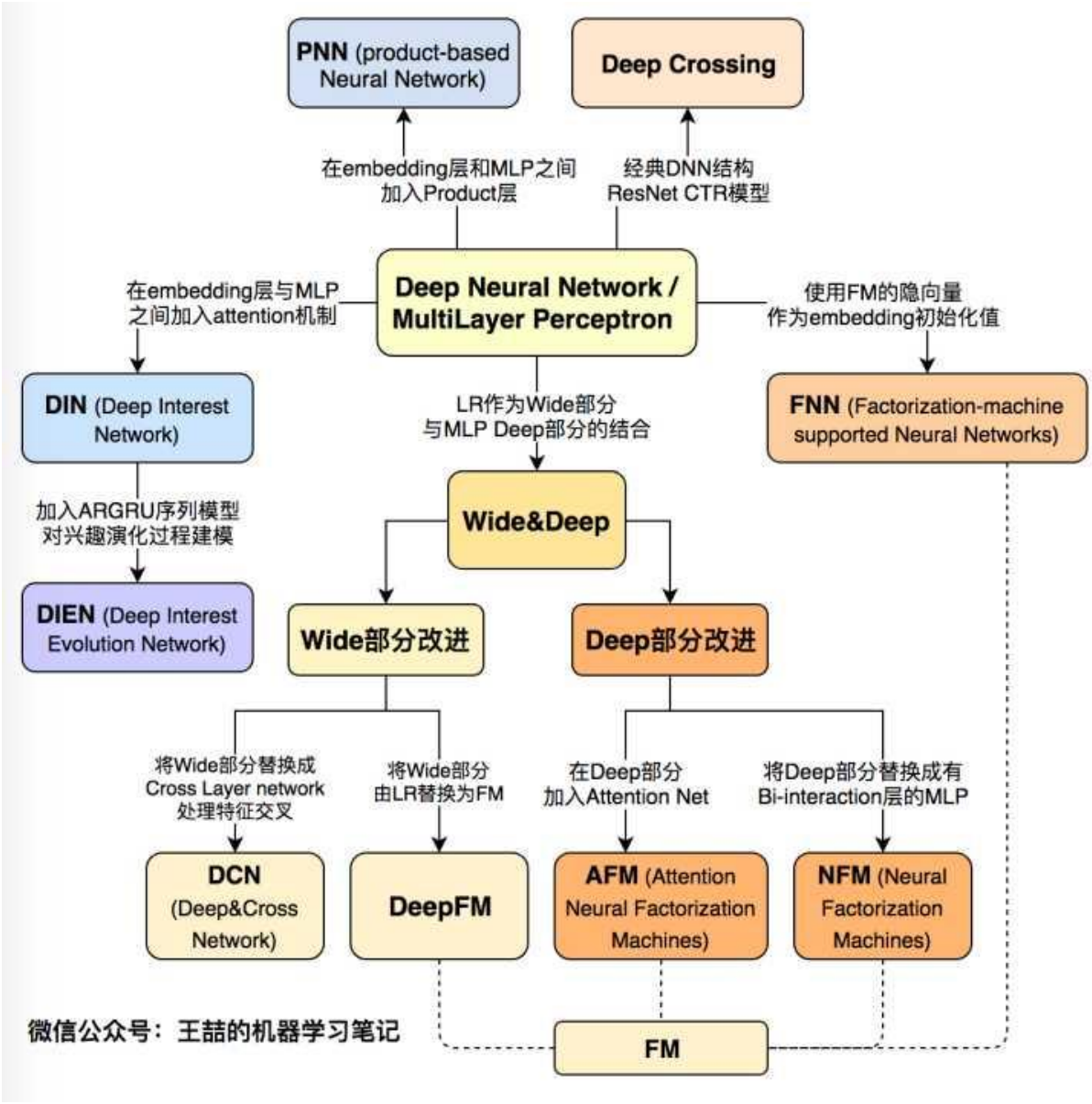
十、阿里 DIEN（2018 年）——DIN 的“进化”

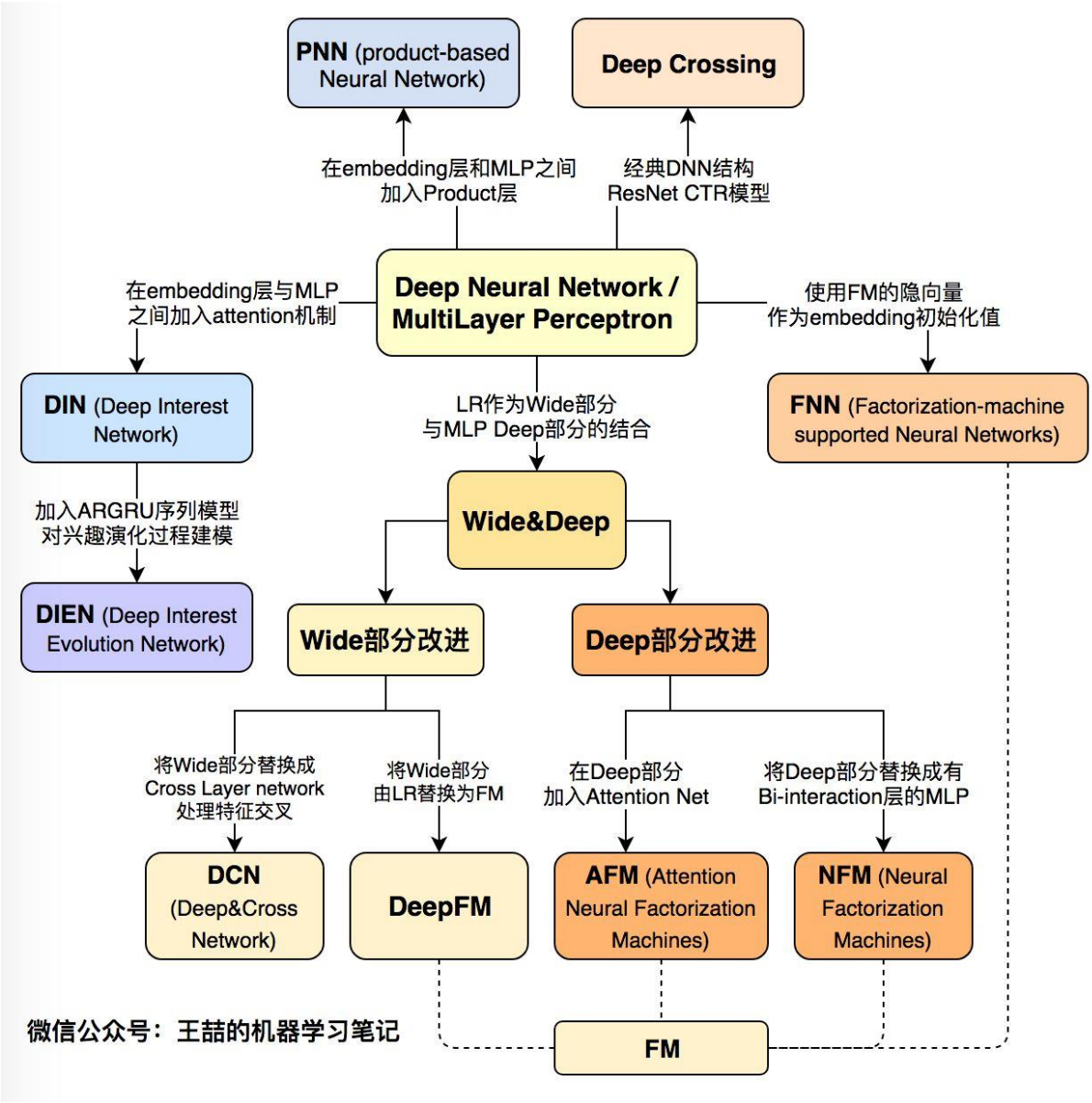


阿里 DIEN 模型架构图 DIEN 的全称为 Deep Interest Evolution Network，它不仅是对 DIN 的进一步“进化”，更重要的是 DIEN 通过引入序列模型 AUGRU 模拟了用户兴趣进化的过程。具体来讲模型的主要特点是在 Embedding layer 和 Concatenate layer 之间加入了生成兴趣的 Interest Extractor Layer 和模拟兴趣演化的 Interest Evolving layer。其中 Interest Extractor Layer 使用了 DIN 的结构抽取了每一个时间片内用户的兴趣，Interest Evolving layer 则利用序列模型 AUGRU 的结构将不同时间的用户兴趣串联起来，形成兴趣进化的链条。最终再把当前时刻的“兴趣向量”输入上层的多层全连接网络，与其他特征一起进行最终的 CTR 预估。

论文:

总结—— CTR 模型的深度学习时代


PNN (product-based Neural Network)
Deep Crossing
在embedding层和MLP之间加入Product层
经典DNN结构 ResNet CTR模型
Deep Neural Network / MultiLayer Perceptron
在embedding层与MLP之间加入attention机制
使用FM的隐向量作为embedding初始化值
DIN (Deep Interest Network)
LR作为Wide部分与MLP Deep部分的结合
FNN (Factorization-machine supported Neural Networks)
加入ARGRU序列模型对兴趣演化过程建模
Wide&Deep
DIEN (Deep Interest Evolution Network)
Wide部分改进
Deep部分改进
将Wide部分替换成Cross Layer network处理特征交叉
将Wide部分由LR替换为FM
在Deep部分加入Attention Net
将Deep部分替换成有Bi-interaction层的MLP
DCN (Deep&Cross Network)
DeepFM
AFM (Attention Neural Factorization Machines)
NFM (Neural Factorization Machines)
微信公众号：王喆的机器学习笔记
FM



文章的最后，我再次强调这张深度学习 CTR 模型演化图，可以毫不夸张的说，这张演化图包括了近年来所有主流的深度学习 CTR 模型的结构特点以及它们之间的演化关系。希望能够帮助推荐、广告、搜索领域的算法工程师们建立起完整的知识体系，能够驾轻就熟的针对业务特点应用并比较不同模型的效果，从而用最适合当前数据模式的模型驱动公司业务。

结合自己的工作经验，关于深度学习模型我想再分享两点内容：

1. 没有银弹。从来没有一个深度学习模型能够在所有数据集上都表现最优，特别是推荐、广告领域，各家的数据集，数据 pattern、业务领域差异巨大，不存在能够解决一切问题的“银弹”模型。比如，阿里的 DIEN 对于数据质量、用户整个 life cycle 行为完整性的要求很高，如果在某些 DSP 场景下运用这个模型，预计不会收到良好的效果。再比如 Google 的 Deep&Cross，我们也要考虑自己的数据集需不需要如此复杂的特征交叉方式，在一些百万量级的数据集上，也许浅层神经网络的表现更好。

2. 算法工程师永远要在理想和现实间做 trade off。有一种思想要避免，就是我为了要上某个

模型就要强转团队的，强买某些硬件设备。模型的更新过程一定是迭代的，一定是从简单到复杂的，一定是你证明了某个模型是 work 的，然后在此基础上做改进的。这也是我们要熟悉所有模型演化关系的原因。

就在我们熟悉这些已有模型的时候，深度学习 CTR 模型的发展从没有停下它的脚步。从阿里的多模态、多目标的深度学习模型，到 YouTube 基于 RNN 等序列模型的推荐系统，再到 Airbnb 使用 Embedding 技术构建的搜索推荐模型，深度学习的应用不仅越来越广泛，而且得到了越来越快的进化。在今后的专栏文章中，我们不仅会更深入的介绍深度学习 CTR 模型的知识，而且会更加关注深度学习 CTR 模型的应用与落地，期待与大家一同学习。

34、Join training 和 ensemble training 有什么区别?

（1） ensemble 中每个模型需要单独训练，并且各个模型之间是相互独立的，模型之间互相不感知，当预测样本时，每个模型的结果用于投票，最后选择得票最多的结果。而 join train 这种方式模型之间不是独立的，是相互影响的，可以同时优化模型的参数。

（2） ensemble 的方式中往往要求存在很多模型，这样就需要更多的数据集和数据特征，才能取得比较好的效果，模型的增多导致难以训练，不利于迭代。而在 wide&deep 中，只需要两个模型，训练简单，可以很快的迭代模型。

35、请简述 DeepFM 模型的原理

本题解析来源：https://www.jianshu.com/p/6f1c2643d31b

1、背景特征组合的挑战

对于一个基于 CTR 预估的推荐系统，最重要的是学习到用户点击行为背后隐含的特征组合。在不同的推荐场景中，低阶组合特征或者高阶组合特征可能都会对最终的 CTR 产生影响。

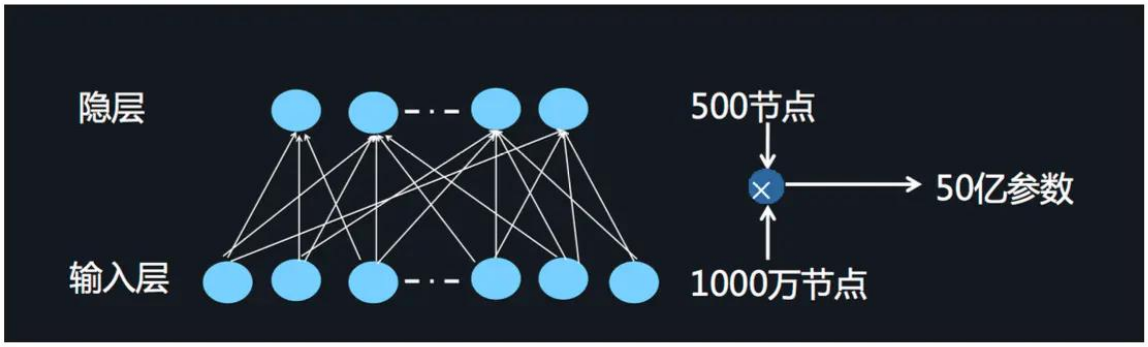
之前介绍的因子分解机(Factorization Machines, FM)通过对于每一维特征的隐变量内积来提取特征组合。最终的结果也非常好。但是，虽然理论上来讲 FM 可以对高阶特征组合进行建模，但实际上因为计算复杂度的原因一般都只用到了二阶特征组合。那么对于高阶的特征组合来说，我们很自然的想法，通过多层的神经网络即 DNN 去解决。

DNN 的局限

下面的图片来自于张俊林教授在 AI 大会上所使用的 PPT。

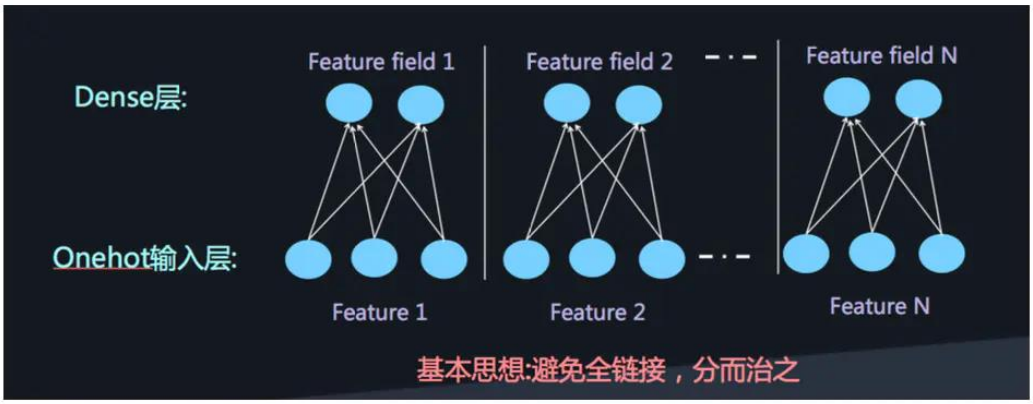
我们之前也介绍过了，对于离散特征的处理，我们使用的是将特征转换成为 one-hot 的形式，但是将 One-hot 类型的特征输入到 DNN 中，会导致网络参数太多：

- Onehot作为DNN输入的问题：CTR预估任务里不可行

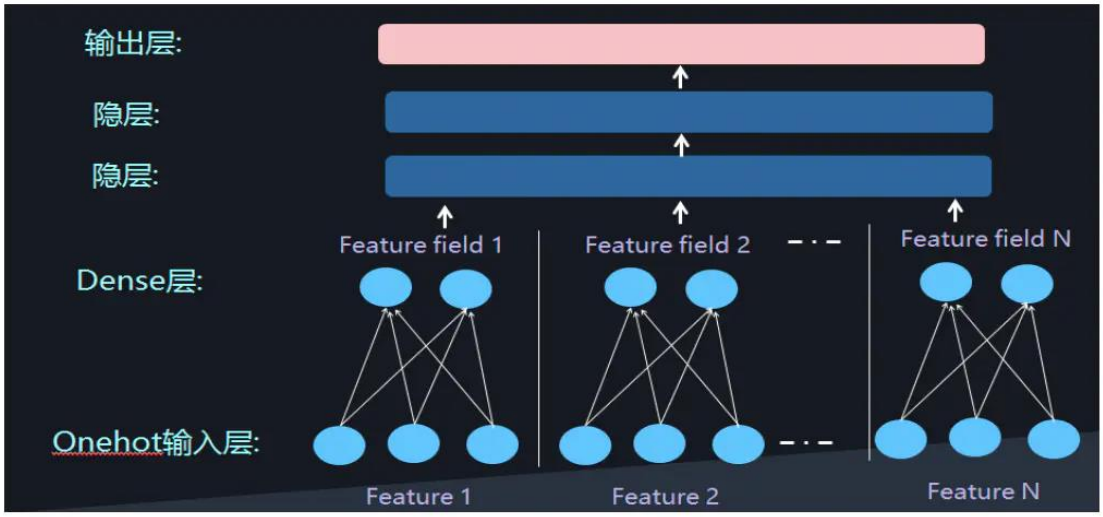


如何解决这个问题呢，类似于 FFM 中的思想，将特征分为不同的 field：

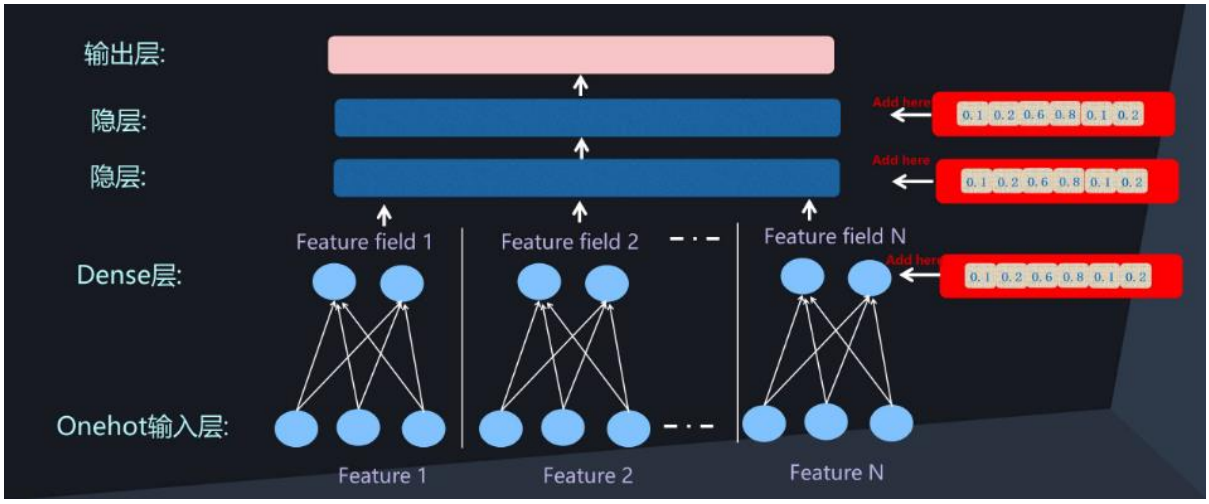
- 解决思路：从OneHot到Dense Vector



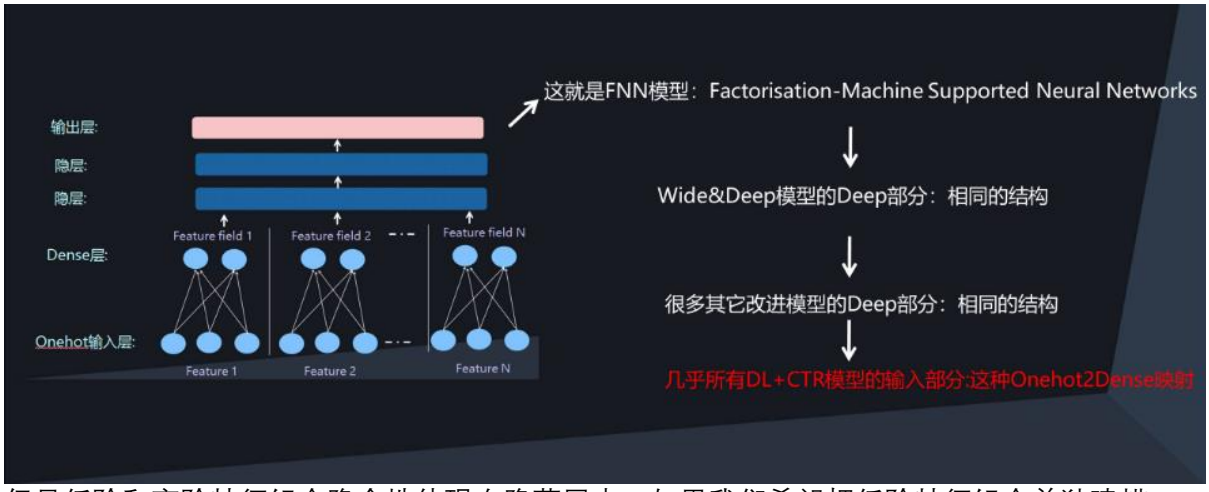
再加两层的全链接层，让 Dense Vector 进行组合，那么高阶特征的组合就出来了



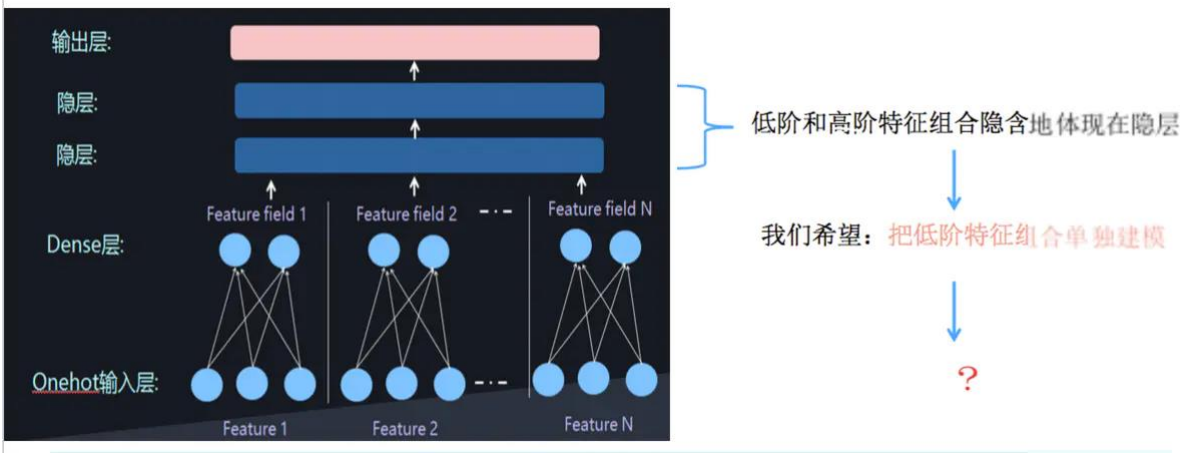
当然，再加入连续特征的话



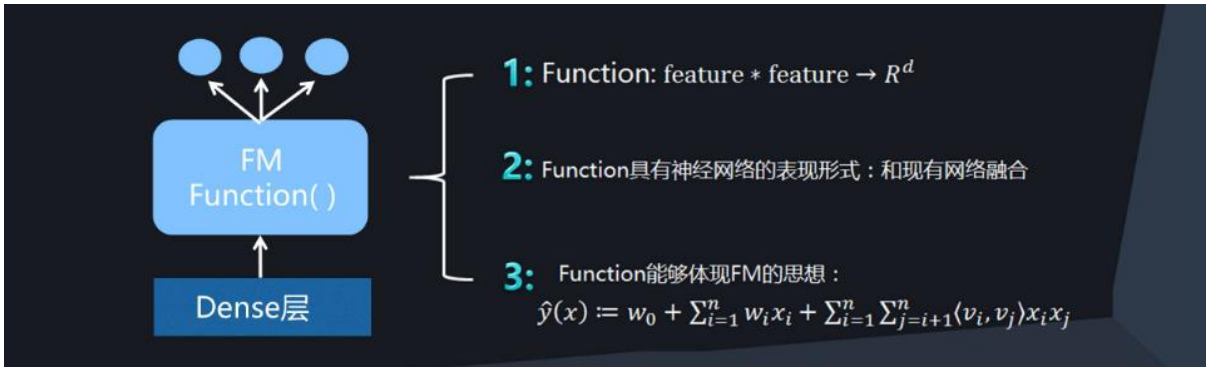
就是通用的深层模型机构了



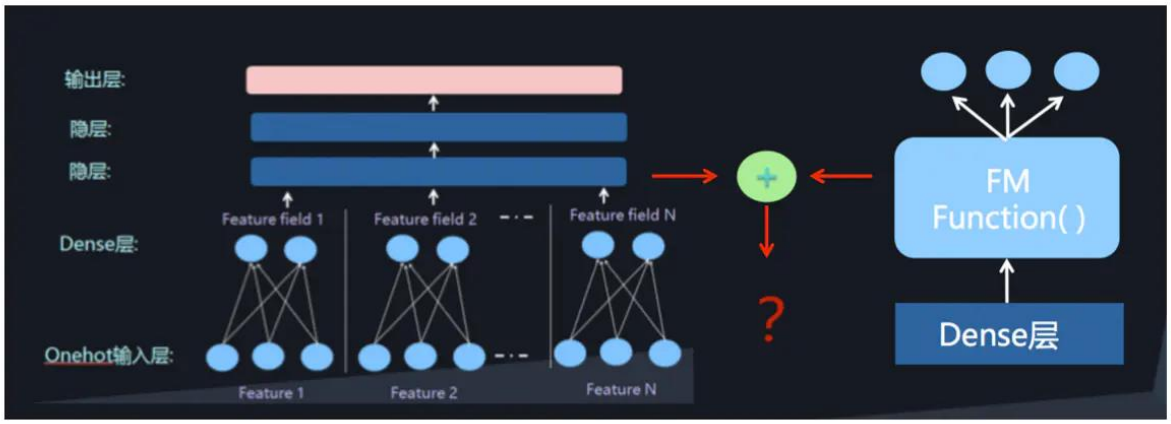
但是低阶和高阶特征组合隐含地体现在隐藏层中，如果我们希望把低阶特征组合单独建模



首先，定义

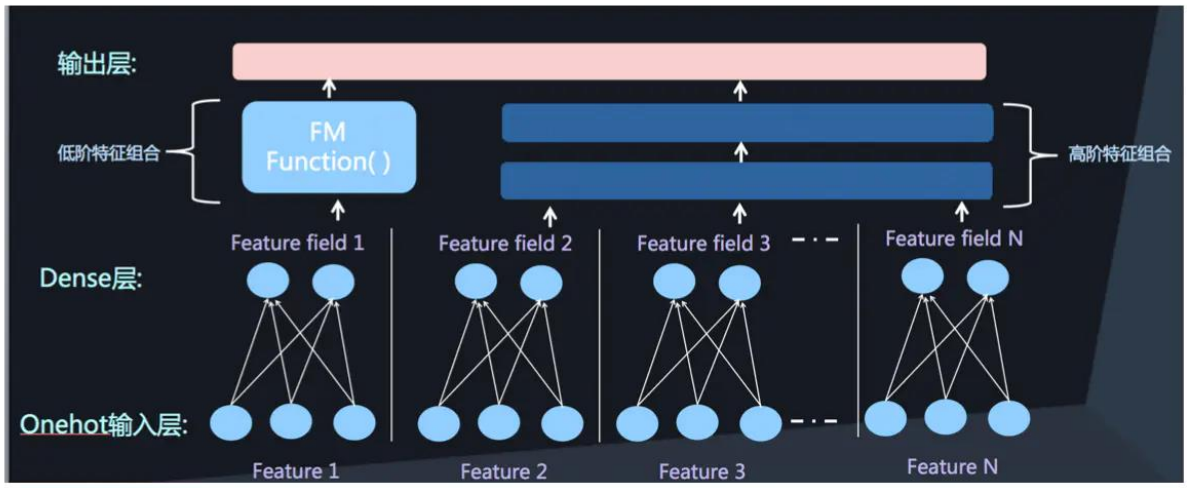


然后，将 DNN 与 FM 进行一个合理的融合：

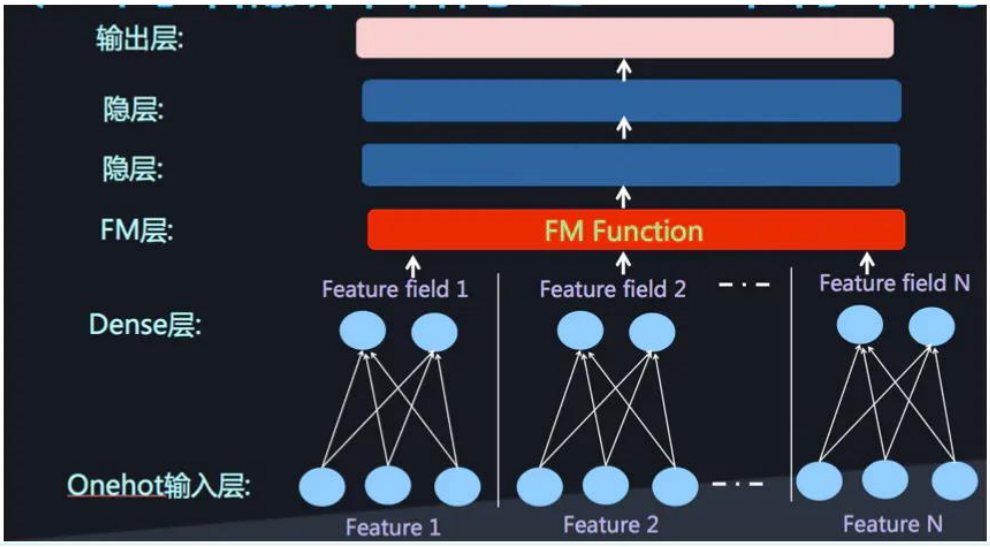


二者的融合总的来说有两种形式，一是并行结构，二是串行结构

典型网络融合结构之一：并行结构



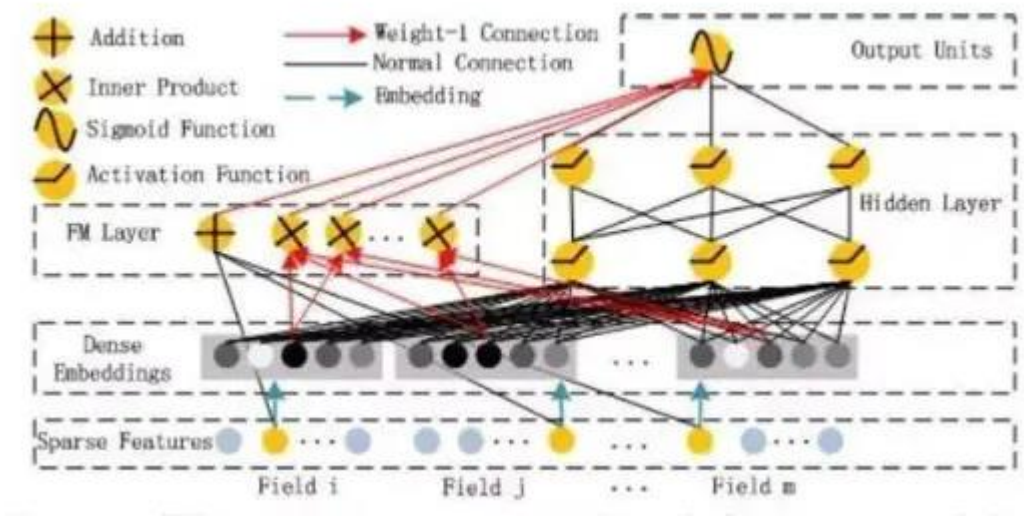
典型网络融合结构之二：串行结构



而我们今天要讲到的 DeepFM，就是并行结构中的一种典型代表。

2、DeepFM 模型

我们先来看一下 DeepFM 的模型结构：



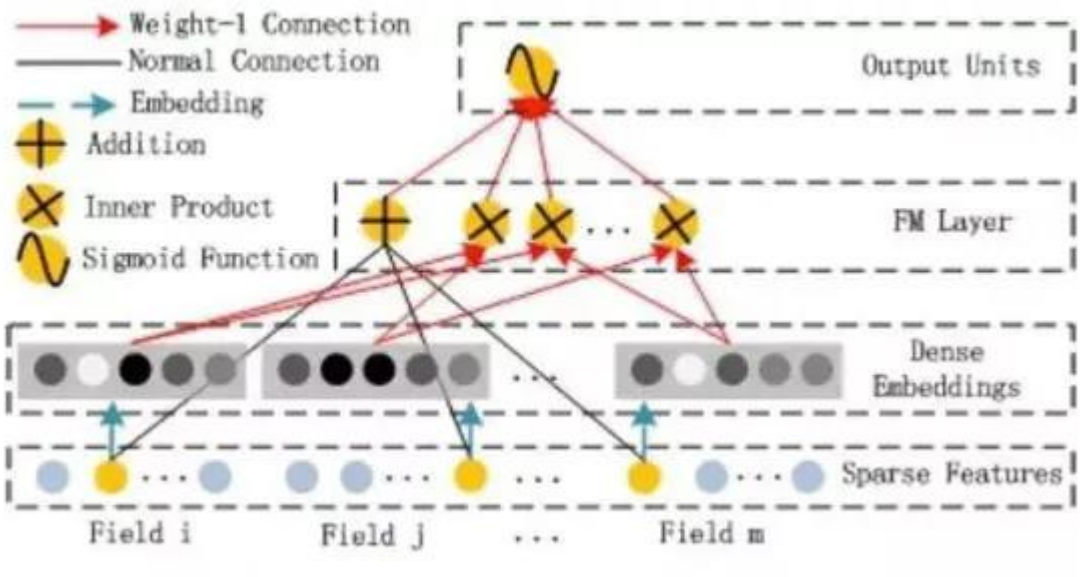
DeepFM

包含两部分：神经网络部分与因子分解机部分，分别负责低阶特征的提取和高阶特征的提取。这两部分共享同样的输入。DeepFM 的预测结果可以写为：

$$\hat{y} = sigmoid(y_{FM} + y_{DNN}),$$

FM 部分

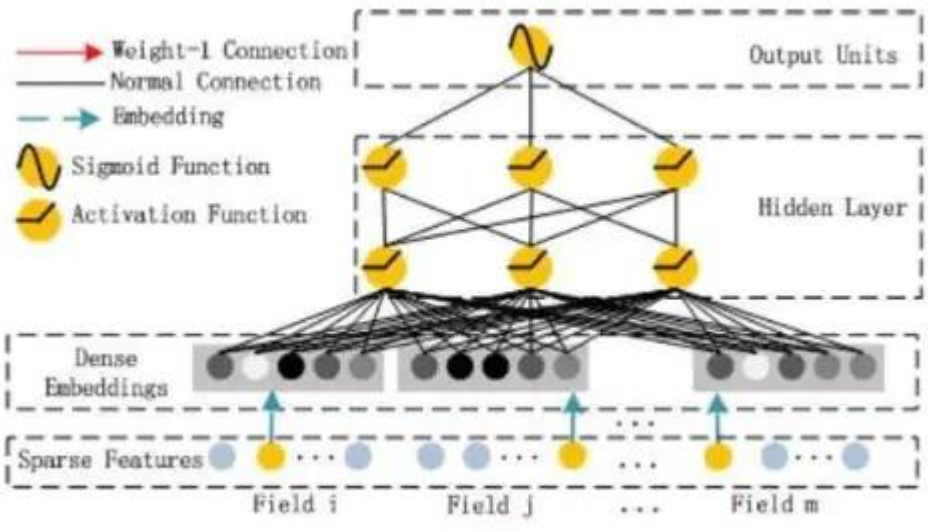
FM 部分的详细结构如下：



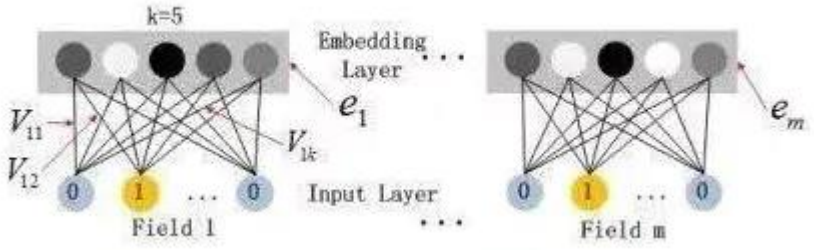
FM 部分是一个因子分解机。关于因子分解机可以参阅文章[Rendle, 2010] Steffen Rendle. Factorization machines. In ICDM, 2010.。因为引入了隐变量的原因，对于几乎不出现或者很少出现的隐变量，FM 也可以很好的学习。
FM 的输出公式为：

$$y_{FM} =< w, x > + \sum_{j_1=1}^{d} \sum_{j_2=j_1+1}^{d} < V_i, V_j > x_{j_1} \cdot x_{j_2}$$

深度部分



深度部分是一个前馈神经网络。与图像或者语音这类输入不同，图像语音的输入一般是连续而且密集的，然而用于 CTR 的输入一般是及其稀疏的。因此需要重新设计网络结构。具体实现中为，在第一层隐含层之前，引入一个嵌入层来完成将输入向量压缩到低维稠密向量。



嵌入层(embedding layer)的结构如上图所示。

当前网络结构有两个有趣的特性

1）尽管不同 field 的输入长度不同，但是 embedding 之后向量的长度均为 K。2)在 FM 里得到的隐变量 Vik 现在作为了嵌入层网络的权重。

2）这里的第二点如何理解呢，假设我们的 k=5，首先，对于输入的一条记录，同一个 field 只有一个位置是 1，那么在由输入得到 dense vector 的过程中，输入层只有一个神经元起作用，得到的 dense vector 其实就是输入层到 embedding 层该神经元相连的五条线的权重，即 vi1，vi2，vi3，vi4，vi5。这五个值组合起来就是我们在 FM 中所提到的 Vi。在 FM 部分和 DNN 部分，这一块是共享权重的，对同一个特征来说，得到的 Vi 是相同的。

36、Collaborative Knowledge Base Embedding 使用哪三种知识的学习?

解析：

结构化知识学习：TransR。TransR是一种基于距离的翻译模型，可以学习得到知识实体的向量表示

文本知识学习：去噪自编码器。去噪自编码器可以学习得到文本的一种泛化能力较强的向量表示

图像知识学习：卷积-反卷积自编码器。卷积-反卷积自编码器可以得到图像的一种泛化能力较强的向量表示

37、怎样将知识图谱引入推荐系统 ？

基于特征的知识图谱辅助推荐，核心是知识图谱特征学习的引入。一般而言，知识图谱是一个由三元组<头节点，关系，尾节点> 组成的异构网络。由于知识图谱天然的高维性和异构性，首先使用知识图谱特征学习对其进行处理，从而得到实体和关系的低维稠密向量表示。这些低维的向量表示可以较为自然地与推荐系统进行结合和交互。

基于结构的推荐模型，更加直接地使用知识图谱的结构特征。具体来说，对于知识图谱中的每一个实体，我们都进行宽度优先搜索来获取其在知识图谱中的多跳关联实体从中得到推荐结果。

38、普通的逻辑回归能否用于大规模的广告点击率预估 ，为什么?

不能 第一，数据量太大。传统的逻辑回归参数训练过程都依靠牛顿法（Newton's Method）或者 L-BFGS 等算法。这些算法并不太容易在大规模数据上得以处理。

第二，不太容易得到比较稀疏（Sparse）的答案（Solution）。也就是说，虽然数据中特征的总数很多，但是对于单个数据点来说，有效特征是有限而且稀疏的。

39、FTRL 在准备训练数据（特征工程）和训练模型时有哪些 trick ？

（1）特征工程

特征预处理：ID化、离散化、归一化等；
特征选择：方差、变异系数、相关系数、Information Gain、Information Gain-Ratio、IV值等；
特征交叉和组合特征：根据特征具有的业务属性特征交叉，利用FM算法、GBDT算法做高维组合特征等。
(2) Subsampling Training Data

正样本全采样，负样本使用一个比例r采样，并在模型训练的时候，对负样本的更新梯度乘以权重1/r;
负采样的方式：随机负采样、Negative sampling、邻近负采样、skip above负采样等。
(3) 在线丢弃训练数据中很少出现的特征(probabilistic feature inclusion)

Poisson Inclusion：对某一维度特征所来的训练样本，以p的概率接受并更新模型；
Bloom Filter Inclusion：用bloom filter从概率上做某一特征出现k次才更新。

40、阿里最新开源的 X-Deep Learning 为 Online Learning 提供了哪些解决方案?

去ID化的稀疏特征学习：传统的机器学习框架一般要求对稀疏特征进行ID化表征（从0开始紧凑编码），以此来保证训练的高效性。XDL则允许直接以原始的特征进行训练，大幅简化了特征工程的复杂度，极大地增加了全链路数据处理效率，这一特性在实时在线学习场景下显得更加有意义。

实时特征频控：用户可以设置一个特征过滤的阈值，例如出现次数大于N次的特征才纳入模型训练，系统会自动的采用自动概率丢弃的算法进行特征选择，这样可以大幅降低无效超低频特征在模型中的空间占用。

过期特征淘汰：长周期的在线学习时，用户也可以通过打开过期特征淘汰功能，系统会自动的对影响力弱且长周期没有碰触到的特征参数进行自动淘汰

41、特征交叉(特征组合)方式有哪些?

1.Dense特征组合
将一个特征与其本身或其他特征相乘(称为特征组合)(二阶或者高阶);
两个特征相除;
对连续特征进行分桶,以分为多个区间分箱。
2.ID特征之间的组合
笛卡尔积:假如拥有一个特征A,A有两个可能值{A1,A2}。拥有一个特征B,存在{B1,B2}等可能值。然后,A&B之间的交叉特征如下:{(A1,B1),(A1,B2),(A2,B1),(A2,B2)},比如经纬度,一个更好地诠释好的交叉特征的实例是类似于(经度,纬度)。一个相同的经度对应了地图上很多的地方,纬度也是一样。但是一旦你将经度和纬度组合到一起,它们就代表了地理上特定的一块区域,区域中每一部分是拥有着类似的特性。

42、特征选择的方法有哪些?

Filter:过滤法,按照发散性或者相关性对各个特征进行行行评分,设定阈值或者待选择阈值的个数,选择特征。

Wrapper:包装法,根据目目标函数(通常是预测效果评分),每次选择若干干特征,或者排除若干干特征。

Embedded:嵌入入法,先使用用某些机器器学习的算法和模型进行行行训练,得到各个特征的权值系数,根据系数从大大到小小选择特征。类似于Filter方方法,但是是通过训练来确定特征的优劣。

43、简述 Multi-task learning(MLT)多任务学习

在机器学习中，我们通常关心优化某一特定指标，不管这个指标是一个标准值，还是企业 KPI。为了达到这个目标，我们训练单一模型或多个模型集合来完成指定得任务。然后，我们通过精细调参，来改进模型直至性能不再提升。尽管这样做可以针对一个任务得到一个可接受得性能，但是我们可能忽略了一些信息，这些信息有助于在我们关心的指标上做得更好。具体来说，这些信息就是相关任务的监督数据。通过在相关任务间共享表示信息，我们的模型在原始任务上泛化性能更好。这种方法称为多任务学习（Multi-Task Learning）

44、如何离线评价召回阶段各种模型算法的好坏？由于没有明确的召回预期值，所以无论 rmse 还是 auc 都不知道该怎么做?

答：召回最直接的评估就是召回率，也就是召回集里正样本的比例；也可以不同的召回算法+同一个排序算法，还是用排序之后的 AUC 和 RMSE 来评估。

45、召回分支的作用是什么?

答：快速帮助用户找到可能感兴趣的候选物品；减少排序模型的候选输入，降低系统 RT。

46、如何保证离线特征与线上特征的一致性?

答：将在线预测时的特征保存下来并保存该样本的时间戳，线下训练时根据样本时间戳进行特征 join，以保证训练和预测阶段的特征一致性。

47、深度排序模型比如 deepfm 和传统的排序模型比如 gbdt、lr 的区别在哪，推深度排序模型主要为了解决传统排序模型存在的哪些问题?

答：重点区别在于高维特征的学习和抽取以及模型的拟合能力上，使用深度模型也是为了解决这些问题。

48、Thompson(汤普森)采样为何有效

本题解析来源：https://www.cnblogs.com/gczr/p/11220187.html
如果想理解汤普森采样算法，就必须先熟悉了解贝塔分布。
一、Beta(贝塔)分布
Beta 分布是一个定义在[0,1]区间上的连续概率分布族，它有两个正值参数，称为形状参数，一般用 α 和 β 表示。
Beta 分布的概率密度函数形式如下：

$$
\begin{aligned}
f(x;\alpha,\beta) &= \text{constant} \cdot x^{\alpha-1}(1-x)^{\beta-1} \\
&= \frac{x^{\alpha-1}(1-x)^{\beta-1}}{\int_0^1 u^{\alpha-1}(1-u)^{\beta-1}\,du} \\
&= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1} \\
&= \frac{1}{\mathrm{B}(\alpha,\beta)} x^{\alpha-1}(1-x)^{\beta-1}
\end{aligned}
$$

这里的 Γ 表示 gamma 函数。

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t}\,dt$$

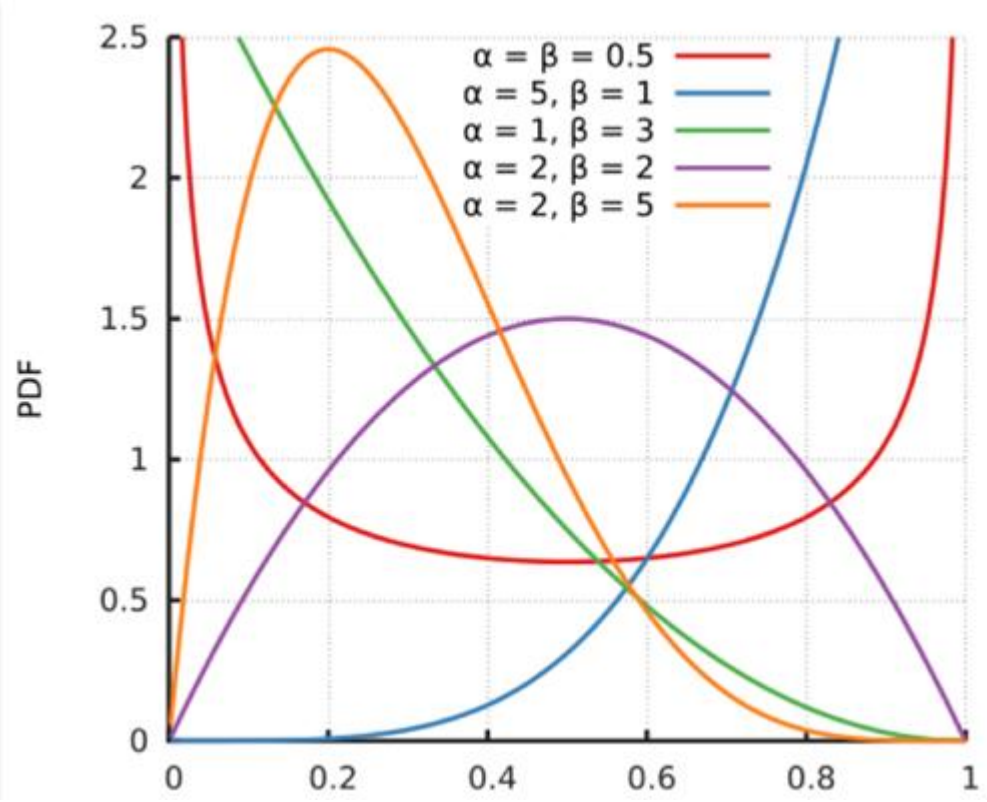
Beta 分布的均值是:

$$\frac{\alpha}{\alpha+\beta}$$

方差:

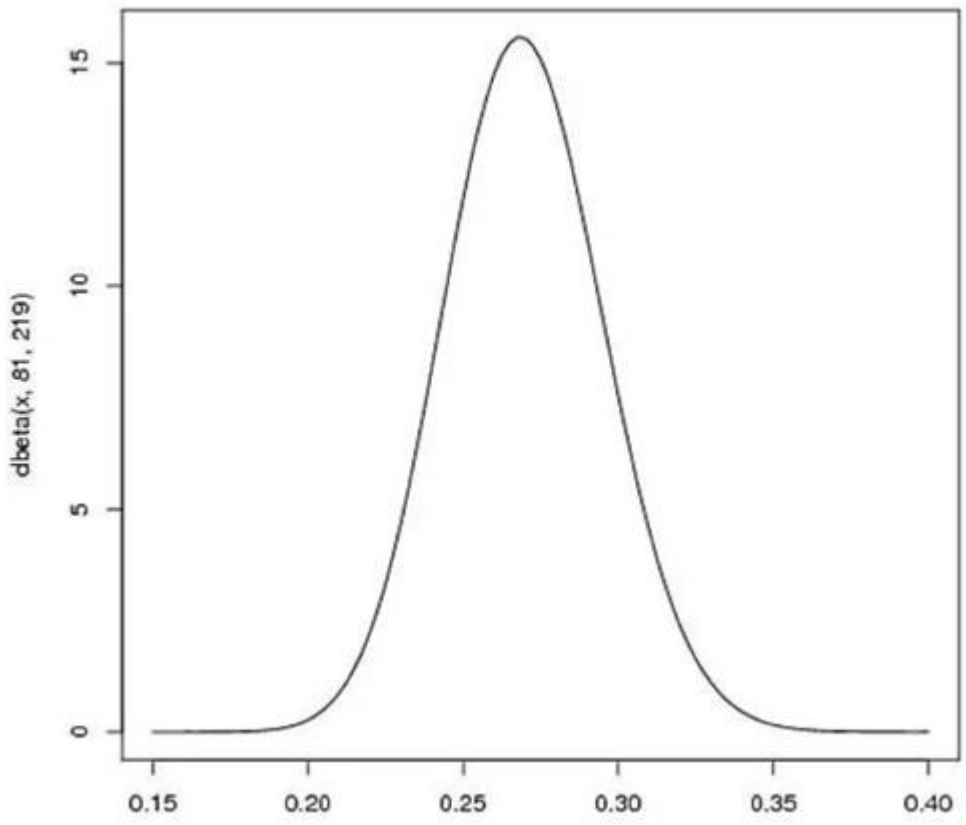
$$\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$$

Beta 分布的图形(概率密度函数):



从 Beta 分布的概率密度函数的图形我们可以看出，Beta 分布有很多种形状，但都是在 0-1 区间内，因此 Beta 分布可以描述各种 0-1 区间内的形状（事件）。因此，它特别适合为某件事发生或者成功

的概率建模。
同时，当 α=1，β=1 的时候，它就是一个均匀分布。
贝塔分布主要有 α 和 β 两个参数，这两个参数决定了分布的形状，从上图及其均值和方差的公式可以看出：
1）α/(α+β)也就是均值，其越大，概率密度分布的中心位置越靠近 1，依据此概率分布产生的随机数也多说都靠近 1，反之则都靠近 0。
2）α+β 越大，则分布越窄，也就是集中度越高，这样产生的随机数更接近中心位置，从方差公式上也能看出来。
二、举例理解 Beta 分布
贝塔分布可以看作是一个概率的分布，当我们不知道一个东西的具体概率是多少时，它给出了所有概率出现的可能性大小，可以理解为概率的概率分布。
以棒球为例子。棒球运动的一个指标就是棒球击球率，就是用一个运动员击中的球数除以总的击球数，一般认为 0.27 是一个平均的击球水平，如果击球率达到 0.3 就会认为非常优秀了。如果我们要预测一个棒球运动员，他整个赛季的棒球击球率，怎么做呢？
你可以直接计算他目前的棒球击球率，用击中数除以击球数。但是，这在赛季开始阶段时是很不合理的。假如这个运动员就打了一次，还中了，那么他的击球率就是 100%；如果没中，那么就是 0%，甚至打 5、6 次的时候，也可能运气爆棚全中击球率 100%，或者运气很糟击球率 0%，所以这样计算出来的击球率是不合理也是不准确的。
为什么呢？
当运动员首次击球没中时，没人认为他整个赛季都会一次不中，所以击球率不可能为 0。因为我们有先验期望，根据历史信息，我们知道击球率一般会在 0.215 到 0.36 之间。如果一个运动员一开始打了几次没中，那么我们知道他可能最终成绩会比平均稍微差一点，但是一般不可能会偏离上述区间，更不可能为 0。
如何解决呢？
一个最好的方法来表示这些先验期望（统计中称为先验（prior））就是贝塔分布，表示在运动员打球之前，我们就对他的击球率有了一个大概范围的预测。假设我们预计运动员整个赛季的击球率平均值大概是 0.27 左右，范围大概是在 0.21 到 0.35 之间。那么用贝塔分布来表示，我们可以取参数 α=81，β=219，因为 α/(α+β)=0.27，图形分布也主要集中在 0.21~0.35 之间，非常符合经验值，也就是我们在不知道这个运动员真正击球水平的情况下，我们先给一个平均的击球率的分布。

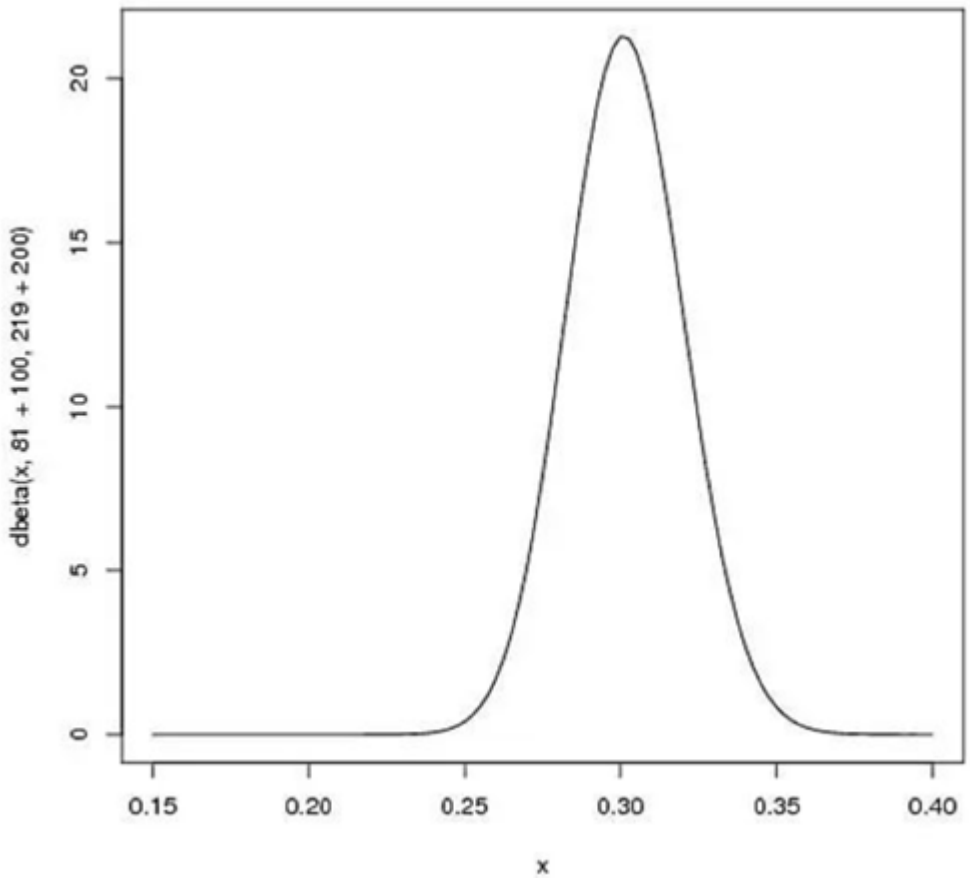


假设运动员一次击中，那么现在他本赛季的记录是“1 次打中；1 次打击”。那么我们更新我们的概率分布，让概率曲线做一些移动来反应我们的新信息。

Beta(α0+hits，β0+misses)

注：α0，β0 是初始化参数，也就是本例中的 81，219。hits 表示击中的次数，misses 表示未击中的次数。击中一次，则新的贝塔分布为 Beta(81+1,219)，一次并不能反映太大问题，所以在图形上变化也不大，不画示意图了。然而，随着整个赛季运动员逐渐进行比赛，这个曲线也会逐渐移动以匹配最新的数据。由于我们拥有了更多的数据，因此曲线（击球率范围）会逐渐变窄。假设赛季过半时，运动员一共打了 300 次，其中击中 100 次。

那么新的贝塔分布是 Beta(81+100,219+200)，如下图：



可以看出，曲线更窄而且往右移动了（击球率更高），由此我们对于运动员的击球率有了更好的了解。新的贝塔分布的期望值为 0.303，比直接计算 100/(100+200)=0.333 要低，是比赛季开始时的预计 0.27 要高，所以贝塔分布能够抛出掉一些偶然因素，比直接计算击球率更能客观反映球员的击球水平。

总结：这个公式就相当于给运动员的击中次数添加了“初始值”，相当于在赛季开始前，运动员已经有 81 次击中 219 次不中的记录。 因此，在我们事先不知道概率是什么但又有一些合理的猜测时，贝塔分布能够很好地表示为一个概率的分布。

三、汤普森采样

汤普森采样的背后原理正是上述所讲的 Beta 分布，你把贝塔分布的 a 参数看成是推荐后用户点击的次数，把分布的 b 参数看成是推荐后用户未点击的次数，则汤普森采样过程如下：

1、取出每一个候选对应的参数 a 和 b；

2、为每个候选用 a 和 b 作为参数，用贝塔分布产生一个随机数；

3、按照随机数排序，输出最大值对应的候选；

4、观察用户反馈，如果用户点击则将对应候选的 a 加 1，否则 b 加 1；

注：实际上在推荐系统中，要为每一个用户都保存一套参数，比如候选有 m 个，用户有 n 个，那么就要保存 2 m n 个参数。

汤普森采样为什么有效呢？

1）如果一个候选被选中的次数很多，也就是 a+b 很大了，它的分布会很窄，换句话说这个候选的收益已经非常确定了，就是说不管分布中心接近 0 还是 1 都几乎比较确定了。用它产生随机数，基本上就在中心位置附近，接近平均收益。

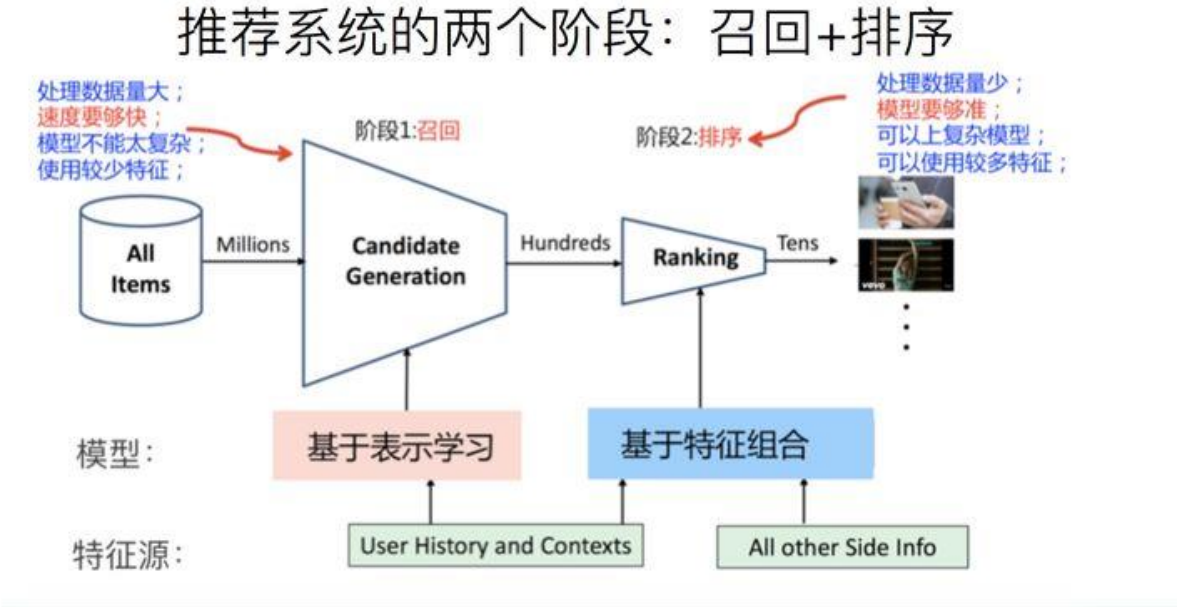
2）如果一个候选不但 a+b 很大，即分布很窄，而且 a/(a+b) 也很大，接近 1，那就确定这是个好的候选项，平均收益很好，每次选择很占优势，就进入利用阶段。反之则有可能平均分布比较接近与 0，几乎再无出头之日。

3）如果一个候选的 a+b 很小，分布很宽，也就是没有被选择太多次，说明这个候选是好是

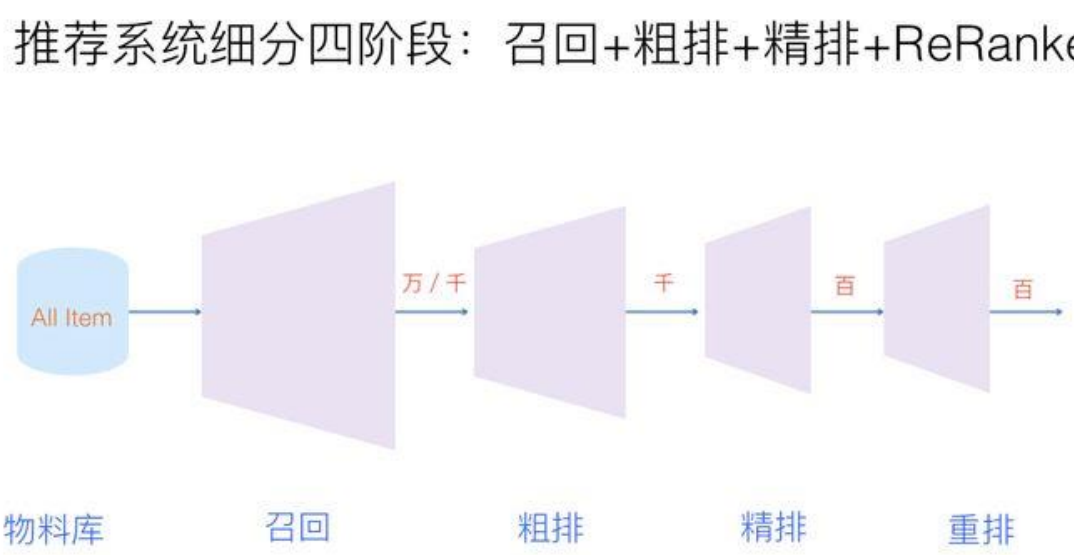
坏还不太确定，那么分布就是跳跃的，这次可能好，下次就可能坏，也就是还有机会存在，没有完全抛弃。那么用它产生随机数就有可能得到一个较大的随机数，在排序时被优先输出，这就起到了前面说的探索作用。
python 代码实现：
choice = numpy.argmax(pymc.rbeta(1 + self.wins, 1 + self.trials - self.wins))

49、请说说推荐系统技术的演进趋势：从召回到排序再到重排

推荐系统技术，总体而言，与 NLP 和图像领域比，发展速度不算太快。不过最近两年，由于深度学习等一些新技术的引入，总体还是表现出了一些比较明显的技术发展趋势。这篇文章试图从推荐系统几个环节，以及不同的技术角度，来对目前推荐技术的比较彰显的技术趋势做个归纳。个人判断较多，偏颇难免，所以还请谨慎参考。
在写技术趋势前，照例还是对推荐系统的宏观架构做个简单说明，以免读者迷失在技术细节中。



实际的工业推荐系统，如果粗分的化，经常讲的有两个阶段。首先是召回，主要根据用户部分特征，从海量的物品库里，快速找回一小部分用户潜在感兴趣的物品，然后交给排序环节，排序环节可以融入较多特征，使用复杂模型，来精准地做个性化推荐。召回强调快，排序强调准。当然，这是传统角度看推荐这个事情。
但是，如果我们更细致地看实用的推荐系统，一般会有四个环节，如下图所示：



四个环节分别是：召回、粗排、精排和重排。召回目的如上所述；有时候因为每个用户召回环节返回的物品数量还是太多，怕排序环节速度跟不上，所以可以在召回和精排之间加入一个粗排环节，通过少量用户和物品特征，简单模型，来对召回的结果进行个粗略的排序，在保证一定精准的前提下，进一步减少往后传送的物品数量，粗排往往是可选的，可用可不同，跟场景有关。之后，是精排环节，使用你能想到的任何特征，可以上你能承受速度极限的复杂模型，尽量精准地对物品进行个性化排序。排序完成后，传给重排环节，传统地看，这里往往会上各种技术及业务策略，比如去已读、去重、打散、多样性保证、固定类型物品插入等等，主要是技术产品策略主导或者为了改进用户体验的。
那么，每个环节，从技术发展的角度看，都各自有怎样的发展趋势呢？下面我们分头说明。

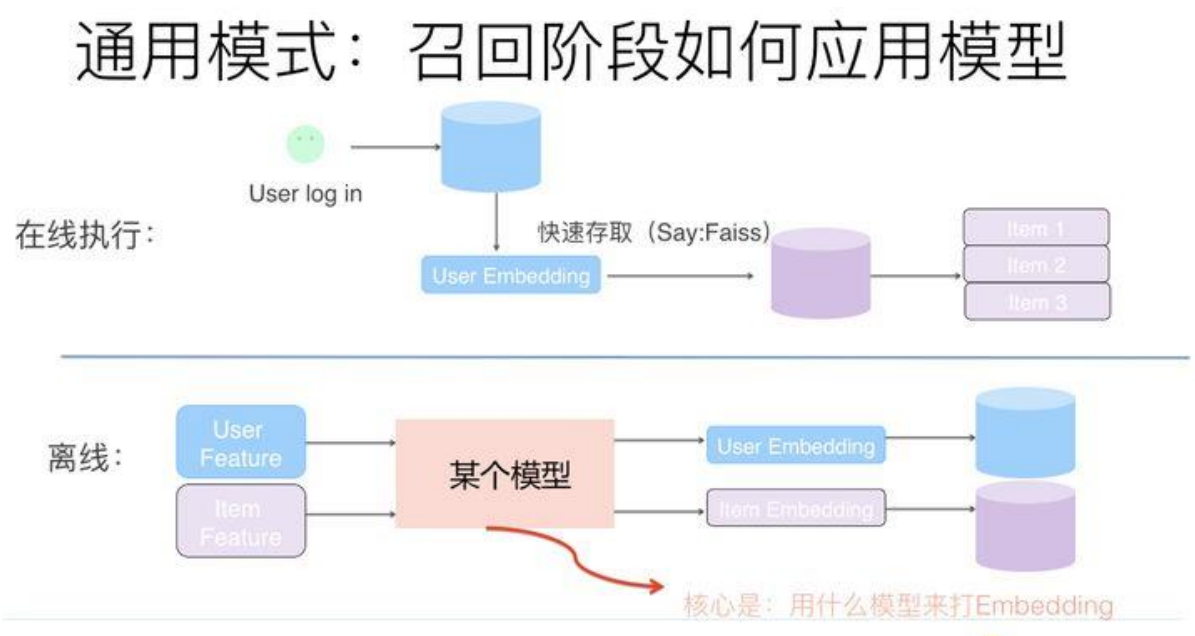
召回技术演进趋势

推荐系统的召回阶段是很关键的一个环节，但是客观的说，传统地看，这个环节，技术含量是不太高的，偏向策略型导向，往往灵机一动，就能想到一个策略，增加一路新的召回。你在网上搜，发现讲推荐模型的，95%是讲排序阶段的模型，讲召回的别说模型，讲它本身的都很少，这与它的策略导向有关系，大家觉得没什么好讲的。总体而言，召回环节的有监督模型化以及一切 Embedding 化，这是两个相辅相成的总体发展趋势。而打 embedding 的具体方法，则可以有各种选择，比如下面介绍的几个技术发展趋势，可以理解为不同的给用户和物品打 embedding 的不同方法而已。

模型召回

传统的标准召回结构一般是多路召回，如上图所示。如果我们根据召回路是否有用户个性化因素存在来划分，可以分成两大类：一类是无个性化因素的召回路，比如热门商品或者热门文章或者历史点击率高的物料的召回；另外一类是包含个性化因素的召回路，比如用户兴趣标签召回。我们应该怎么看待包含个性化因素的召回路呢？其实吧，你可以这么看，可以把某个召回路看作是：单特征模型排序的排序结果。意思是，可以把某路召回，看成是某个排序模型的排序结果，只不过，这个排序模型，在用户侧和物品侧只用了一个特征。比如说，标签召回，其实就是用用户兴趣标签和物品标签进行排序的单特征排序结果；再比如协同召回，可以看成是只包含 UID 和 ItemID 的两个特征的排序结果…诸如此类。我们应该统一从排序的角度来看待推荐系统的各个环节，这样可能会更好理解本文所讲述的一些技术。

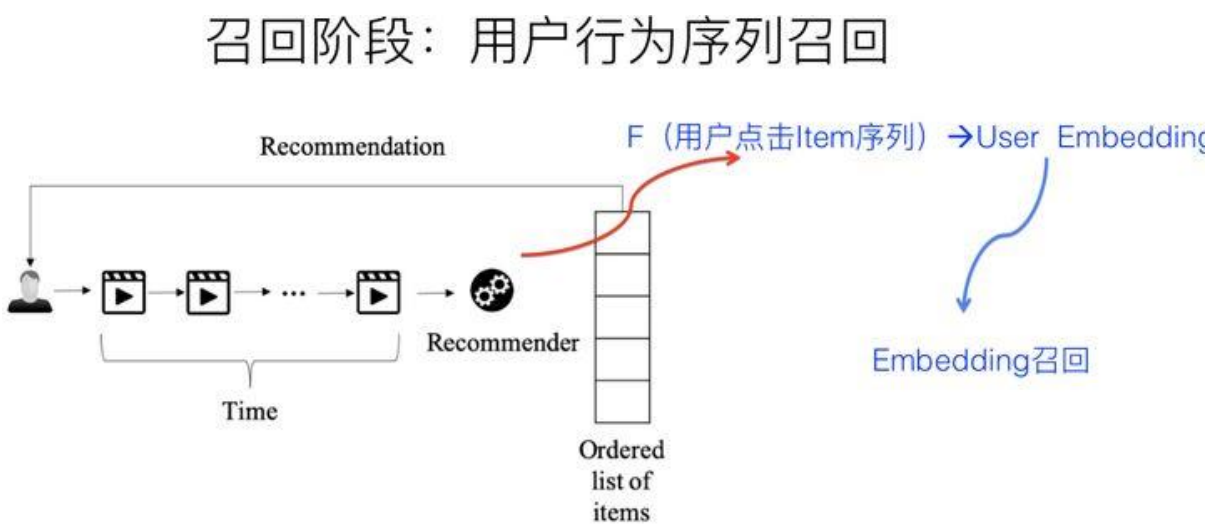
如果我们换做上面的角度看待有个性化因素召回路，那么在召回阶段引入模型，就是自然而然的一个拓展结果：无非是把单特征排序，拓展成多特征排序的模型而已；而多路召回，则可以通过引入多特征，被融入到独立的召回模型中，找到它的替代品。如此而已。所以，随着技术的发展，在 embedding 基础上的模型化召回，必然是个符合技术发展潮流的方向。



那么如何在召回阶段利用模型来代替多路召回呢？上图展示了一个抽象的模型召回的通用架构，核心思想是：将用户特征和物品特征分离，各自通过某个具体的模型，分别打出用户 Embedding 以及物品 Embedding。在线上，可以根据用户兴趣 Embedding，采用类似 Faiss 等高效 Embedding 检索工具，快速找出和用户兴趣匹配的物品，这样就等于做出了利用多特征融合的召回模型了。理论上来说，任何你能见到的有监督模型，都可以用来做这个召回模型，比如 FM / FFM / DNN 等，常说的所谓"双塔"模型，指的其实是用户侧和物品侧特征分离分别打 Embedding 的结构而已，并非具体的模型。
模型召回具备自己独有的好处和优势，比如多路召回每路截断条数的超参个性化问题等会自然被消解掉。当然，它也会带来自己的问题，比较典型的是召回内容头部问题，因为之前多路，每路召回个数靠硬性截断，可以根据需要，保证你想要召回的，总能通过某一路拉回来；而由于换成了模型召回，面向海量物料库，排在前列得分高的可能聚集在几个物料分布比较多的头部领域。解决这个问题的方法包括比如训练数据对头部领域的降采样，减少某些领域主导，以及在模型角度鼓励多样性等不同的方法。
另外一点值得注意的是：如果在召回阶段使用模型召回，理论上也应该同步采用和排序模型相同的优化目标，尤其是如果排序阶段采用多目标优化的情况下，召回模型也应该对应采取相同的多目标优化。同理，如果整个流程中包含粗排模块，粗排也应该采用和精排相同的多目标优化，几个环节优化目标应保持一致。因为召回和粗排是精排的前置环节，否则，如果优化目标不一致，很可能会出现高质量精排目标，在前置环节就被过滤掉的可能，影响整体效果。

典型工作：
FM 模型召回：推荐系统召回四模型之：全能的 FM 模型
DNN 双塔召回：Sampling-Bias-Corrected Neural Modeling for Large Corpus Item Recommendations

用户行为序列召回
用户在使用 APP 或者网站的时候，一般会产生一些针对物品的行为，比如点击一些感兴趣的物品，收藏或者互动行为，或者是购买商品等。而一般用户之所以会对物品发生行为，往往意味着这些物品是符合用户兴趣的，而不同类型的行为，可能代表了不同程度的兴趣。比如购买就是比点击更能表征用户兴趣的行为。



趋势：用户行为序列模型召回（无监督 / 有监督）

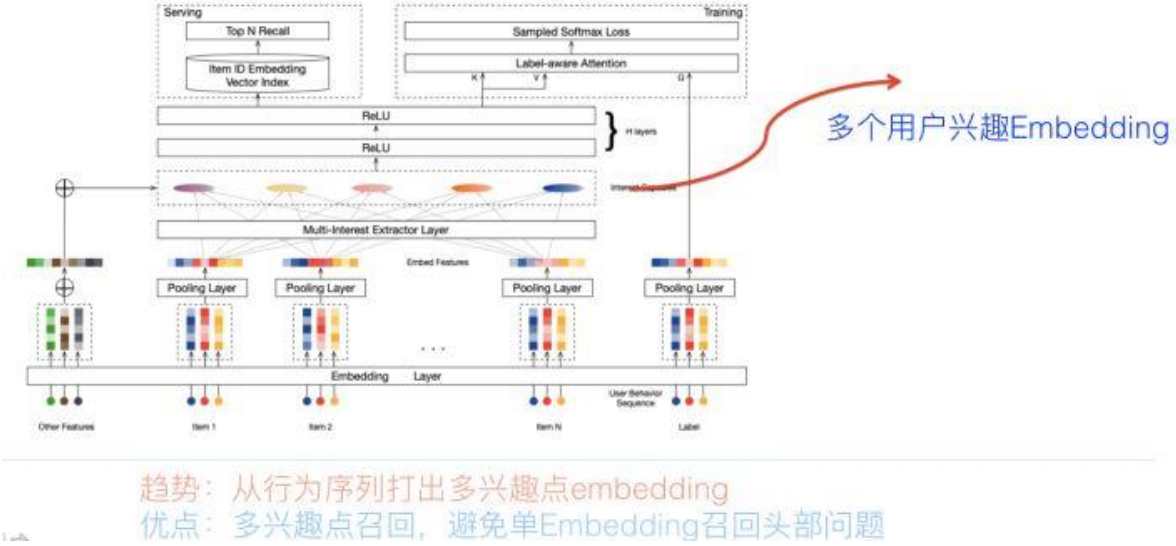
而用户行为过的物品序列，其实是具备表征用户兴趣的非常有价值的信息，而且这种兴趣表征，是

细粒度的用户兴趣，所以对于刻画用户兴趣具备特别的价值。利用用户行为过的物品序列，来表征用户兴趣，具备很好的实用价值。
如果我们抽象地来看的话，利用用户行为过的物品序列对用户兴趣建模，本质上就是这么个过程：输入是用户行为过的物品序列，可以只用物品 ID 表征，也可以融入物品的 Side Information 比如名称，描述，图片等，现在我们需要一个函数 Fun，这个函数以这些物品为输入，需要通过一定的方法把这些进行糅合到一个 embedding 里，而这个糅合好的 embedding，就代表了用户兴趣。无论是在召回过程，还是排序过程，都可以融入用户行为序列。在召回阶段，我们可以用用户兴趣 Embedding 采取向量召回，而在排序阶段，这个 embedding 则可以作为用户侧的特征。
所以，核心在于：这个物品聚合函数 Fun 如何定义的问题。这里需要注意的一点是：用户行为序列中的物品，是有时间顺序的。理论上，任何能够体现时序特点或特征局部性关联的模型，都比较适合应用在这里，典型的比如 CNN、RNN、Transformer 等，都比较适合用来集成用户行为序列信息。而目前的很多试验结果证明，GRU（RNN 的变体模型）可能是聚合用户行为序列效果最好又比较简单的模型。当然，RNN 不能并行的低效率，那是另外一个问题。
在召回阶段，如何根据用户行为序列打 embedding，可以采取有监督的模型，比如 Next Item Prediction 的预测方式即可；也可以采用无监督的方式，比如物品只要能打出 embedding，就能无监督集成用户行为序列内容，例如 Sum Pooling。而排序侧，必然是有监督的模式，需要注意的是：排序侧表征用户特征的时候，可以只用用户行为过的物品序列，也可以混合用户其它特征，比如群体属性特征等一起来表征用户兴趣，方式比较灵活。比如 DIEN，就是典型的采用混合模式的方法。

典型工作：
GRU：Recurrent Neural Networks with Top-k Gains for Session-based Recommendations
CNN：Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding
Transformer: Self-Attentive Sequential Recommendation

用户多兴趣拆分
上文讲了利用用户行为物品序列，打出用户兴趣 Embedding 的做法。但是，另外一个现实是：用户往往是多兴趣的，比如可能同时对娱乐、体育、收藏感兴趣。这些不同的兴趣也能从用户行为序列的物品构成上看出来，比如行为序列中大部分是娱乐类，一部分体育类，少部分收藏类等。那么能否把用户行为序列物品中，这种不同类型的用户兴趣细分，而不是都笼统地打到一个用户兴趣 Embedding 里呢？用户多兴趣拆分就是解决这类更细致刻画用户兴趣的方向。

召回阶段：用户兴趣多Embedding拆分



用户多兴趣拆分，本质上是上文所叙述的用户行为序列打 embedding 方向的一个细化，无非上文说的是：以用户行为序列物品作为输入，通过一些能体现时序特点的模型，映射成一个用户兴趣 embedding。而用户多兴趣拆分，输入是一样的，输出不同，无非由输出单独一个用户 embedding，换成输出多个用户兴趣 embedding 而已。虽说道理如此，但是在具体技术使用方向上却不太一样，对于单用户兴趣 embedding 来说，只需要考虑信息有效集成即可；而对于多用户兴趣拆分来说，需要多做些事情，多做什么事情呢？本质上，把用户行为序列打到多个 embedding 上，实际它是个类似聚类的过程，就是把不同的 Item，聚类到不同的兴趣类别里去。目前常用的拆分用户兴趣 embedding 的方法，主要是胶囊网络和 Memory Network，但是理论上，很多类似聚类的方法应该都是有效的，所以完全可以在这块替换成你自己的能产生聚类效果的方法来做。

说到这里，有同学会问了：把用户行为序列拆分到不同的 embedding 里，有这个必要吗？反正不论怎样，即使是一个 embedding，信息都已经包含到里面了，并未有什么信息损失问题呀。这个问题很好。我的个人感觉是：在召回阶段，把用户兴趣拆分成多个 embedding 是有直接价值和意义的，前面我们说过，召回阶段有时候容易碰到头部问题，就是比如通过用户兴趣 embedding 拉回来的物料，可能集中在头部优势领域中，造成弱势兴趣不太能体现出来的问题。而如果把用户兴趣进行拆分，每个兴趣 embedding 各自拉回部分相关的物料，则可以很大程度缓解召回的头部问题。所以我感觉，这种兴趣拆分，在召回阶段是很合适的，可以定向解决它面临的一些实际问题。

对于排序环节，是否有必要把用户兴趣拆分成多个，我倒觉得必要性不是太大，很难直观感受这样做背后发生作用的机理是怎样的。我能想到的，在排序环节使用多兴趣 Embedding 能发生作用的地方，好像有一个：因为我们在计算 user 对某个 item 是否感兴趣的时候，对于用户行为序列物品，往往计算目标 item 和行为序列物品的 Attention 是有帮助的，因为用户兴趣是多样的，物品 Item 的类型归属往往是唯一的，所以行为序列里面只有一部分物品和当前要判断的 Item 是类型相关的，这会对判断有作用，其它的无关物品其实没啥用，于是 Attention 就是必要的，可以减少那些无关物品对当前物品判断的影响。而当行为序列物品太多的时候，我们知道，Atttention 计算是非常耗时的操作，如果我们把这种 Attention 计算，放到聚类完的几个兴趣 embedding 维度计算，无疑能极大提升训练和预测的速度。貌似这个优点还是成立的。

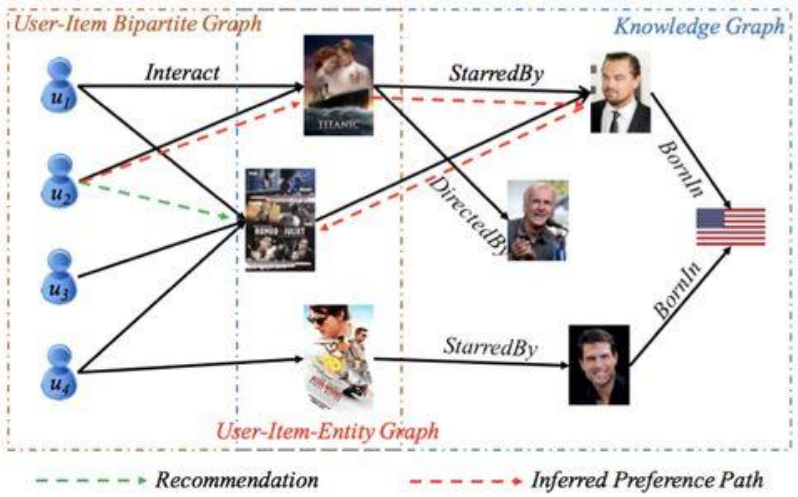
典型工作：

召回：Multi-Interest Network with Dynamic Routing for Recommendation at Tmall

排序：Practice on Long Sequential User Behavior Modeling for Click-Through Rate Prediction

知识图谱融合召回

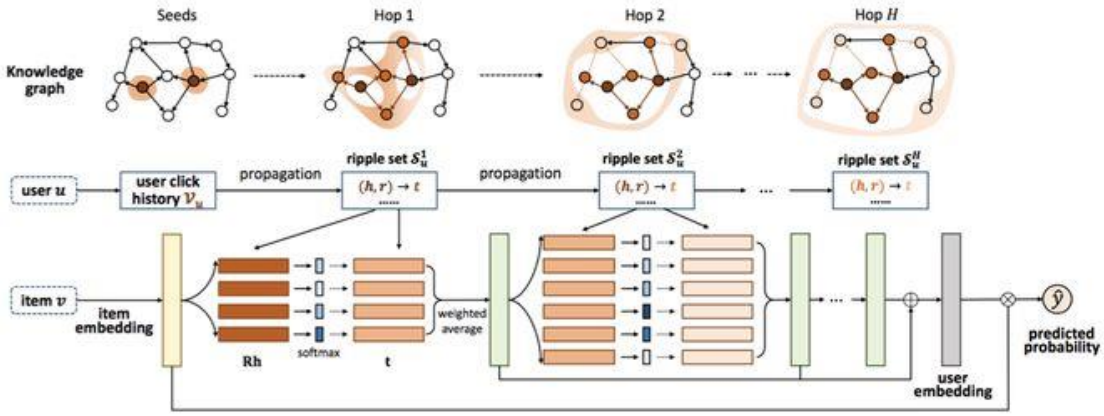
召回模型趋势：Knowledge融合



Knowledge:User-Item-Entity

推荐系统中，最核心的数据是用户对物品的行为数据，因为这直接表明了用户兴趣所在。如上图所示，如果把用户放在一侧，物品放在另一侧，若用户对某物品有行为产生，则建立一条边，这样就构建了用户-物品交互的二部图。其实，有另外一种隐藏在冰山之下的数据，那就是物品之间是有一些知识联系存在的，就是我们常说的知识图谱，而这类数据是可以考虑用来增强推荐效果的，尤其是对于用户行为数据稀疏的场景，或者冷启动场景。以上图例子说明，用户点击过电影“泰坦尼克号”，这是用户行为数据，我们知道，电影“泰坦尼克号”的主演是莱昂纳多，于是可以推荐其它由莱昂纳多主演的电影给这个用户。后面这几步操作，利用的是电影领域的知识图谱数据，通过知识图谱中的“电影 1—>主演—>电影 2”的图路径给出的推荐结果。

召回模型趋势：Knowledge融合



趋势：将知识图谱的关于User和Item的相关知识引入，增强推荐效果
现状：在排序侧其实效果一般，必要性也不太大；比较适合召回；

用于做推荐，一般有两大类知识图谱融合模式：知识图谱 Embedding 模式（KGE）及图路径模式。知识图谱 Embedding 模式首先根据 TransE 等对知识图谱进行 Embedding 化编码的工具，将节点和边转换成 Embedding 表征方式。然后根据用户行为过的物品，以及物品在知识图谱中的 Embedding 和知识图谱中其它知识 embedding 的距离，来扩展物品的信息含量，或者扩充用户行为数据，类似用已知的用户行为数据，在知识图谱辅助下进行外扩。知识图谱的 Embedding 模式在可解释性方面比较弱，因为知识之间的关联是通过 Embedding 计算出来的，不好解释为什么从这个知识跳到那个知识；而图路径模式则是根据物品属性之间的关联等人工定义好的所谓 Meta-Path，也就是人工定义的知识图谱中知识的关联和传播模式，通过中间属性来对知识传播进行路径搭建，具体例子就是上面说的“电影 1 主演电影 2”，这就是人事先定义好的 Meta-Path，也就是人把自己的经验写成规则，来利用知识图谱里的数据。图路径模式在可解释性方面效果较好，因为是人工定义的传播路径，所以非常好理解知识传播关系，但是往往实际应用效果并不好。

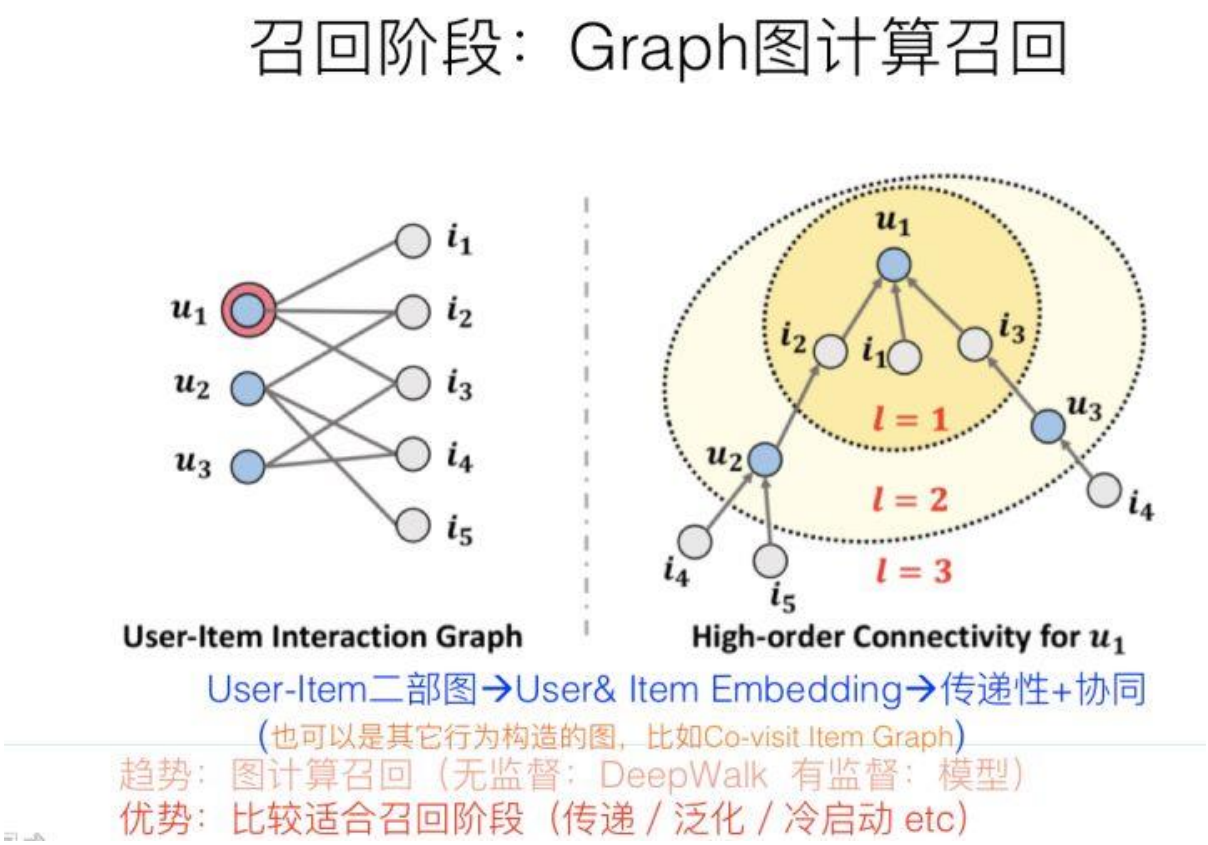
知识图谱是一种信息拓展的模式，很明显，对知识进行近距离的拓展，这可能会带来信息补充作用，但是如果拓展的比较远，或者拓展不当，反而可能会引入噪音，这个道理好理解。所以，我的感觉是，知识图谱在排序侧并不是特别好用，如果想用的化，比较适合用户行为数据非常稀疏以及用户冷启动的场景，也就是说如果用户数据太少，需要拓展，可以考虑使用它。另外，知识图谱还有一个普适性的问题，完全通用的知识图谱在特定场景下是否好用，对此我是有疑问的，而专业性的知识图谱，还有一个如何构建以及构建成本问题；而且很多时候，所谓的知识传播，是可以通过添加属性特征来解决的，比如：电影 1—>主演—>电影 2 这种知识传播路径，完全可以通过把主演作为电影这个实体的属性特征加入常规排序模型，来达到类似知识近距离传播的目的，所以感觉也不是很有必要在排序侧专门去做知识图谱拓展这种事情。

这种知识拓展，可能比较适合用在召回阶段，因为对于传统观点的召回来说，精准并不是最重要的目标，找出和用户兴趣有一定程度相关性但是又具备泛化性能的物品是召回侧的重点，所以可能知识图谱的模式更适合将知识图谱放在召回侧。

当然，知识图谱有一个独有的优势和价值，那就是对于推荐结果的可解释性；比如推荐给用户某个物品，可以在知识图谱里通过物品的关键关联路径给出合理解释，这对于推荐结果的解释性来说是很好的，因为知识图谱说到底是人编码出来让自己容易理解的一套知识体系，所以人非常容易理解其间的关系。但是，在推荐领域目前的工作中，知识图谱的可解释性往往是和图路径方法关联在一起的，而 Path 类方法，很多实验证明了，在排序角度来看，是效果最差的一类方法。所以，我觉得，应该把知识图谱的可解释性优势从具体方法中独立出来，专门用它来做推荐结果的可解释性，

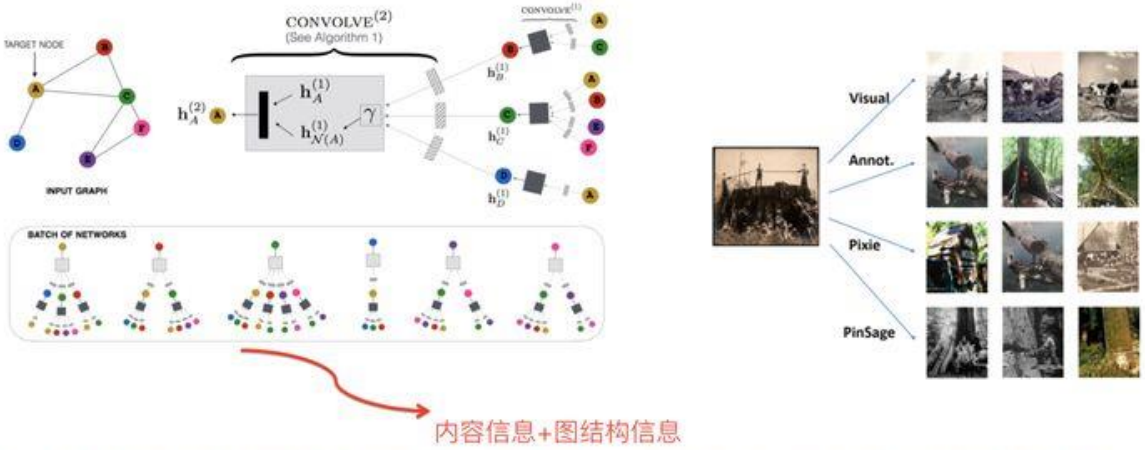
这样就能独立发挥它自身的优势；
至于如何利用知识图谱做召回，其实很直观，比如可以采取如下的无监督学习版本：例如，推荐系统里对用户感兴趣的实体比如某个或者某些明星，往往是个单独的召回路，而可以根据用户的兴趣实体，通过知识图谱的实体 Embedding 化表达后（或者直接在知识图谱节点上外扩），通过知识外扩或者可以根据 Embedding 相似性，拓展出相关实体。形成另外一路相关性弱，但是泛化能力强的 Knowledge 融合召回路。
典型工作：
1. KGAT: Knowledge Graph Attention Network for Recommendation
2. RippleNet: Propagating User Preferences on the Knowledge Graph for Recommender Systems

图神经网络模型召回



严格来说，知识图谱其实是图神经网络的一个比较特殊的具体实例，但是，知识图谱因为编码的是静态知识，而不是用户比较直接的行为数据，和具体应用距离比较远，这可能是导致两者在推荐领域表现差异的主要原因。图神经网络中的图结构，可以是上面介绍知识图谱时候说过的“用户-物品”二部图，也可以是我们常见的有向图或者无向图，图中的节点是各种不同类型的物品及用户，边往往是通过用户行为建立起来的，可以是具体用户的具体行为，也可以是所有用户的群体统计行为，比如物品 1—>物品 2 可以有边，边还可以带上权重，如果越多的用户对物品 1 进行行为后对物品 2 进行行为，则这条边的权重越大。而且对于用户或者物品来说，其属性也可以体现在图中，比如对于一个微博，它的文本内容、图片内容、发布者等等属性都可以引入到图中，比如挂接到物品上，或者建立独立的节点也是可以的，这取决于具体的做法。

召回发展趋势：Graph Neural Network



内容信息+图结构信息

趋势：图模型比较适合解决冷启动问题（内容信息+结构传递）（无监督 / 半监督 / 监督）

难点：大规模计算效率问题（图规模大 / 需要不断迭代）→GraphSAGE

图神经网络的最终目的是要通过一定技术手段，获得图中节点的 embedding 编码。最常用的 embedding 聚合工具是 CNN，对于某个图节点来说，它的输入可以有两类信息，一类是自身的属性信息，比如上面举的微博的例子；另外一类是图结构信息，就是和当前节点有直接边关联的其它节点信息。通过 CNN，可以对两类信息进行编码和聚合，形成图节点的 embedding。通过 CNN 等信息聚合器，在图节点上进行计算，并反复迭代更新图节点的 embedding，就能够最终获得可靠的图节点 embedding 信息，而这种迭代过程，其实体现的是远距离的节点将信息逐步通过图结构传递信息的过程，所以图结构是可以进行知识传递和补充的。

我们可以进一步思考下，图节点因为可以带有属性信息，比如物品的 Content 信息，所以明显这对于解决物品侧的冷启动问题有帮助；而因为它也允许知识在图中远距离进行传递，所以比如对于用户行为比较少的场景，可以形成知识传递和补充，这说明它也比较适合用于数据稀疏的推荐场景；另外一面，图中的边往往是通过用户行为构建的，而用户行为，在统计层面来看，本质上是一种协同信息，比如我们常说的“A 物品协同 B 物品”，本质上就是说很多用户行为了物品 A 后，大概率会去对物品 B 进行行为；所以图具备的一个很好的优势是：它比较便于把协同信息、用户行为信息、内容属性信息等各种异质信息在一个统一的框架里进行融合，并统一表征为 embedding 的形式，这是它独有的一个优势，做起来比较自然。另外的一个特有优势，就是信息在图中的传播性，所以对于推荐的冷启动以及数据稀疏场景应该特别有用。

因为图神经网络，最终获得的往往是图中节点的 embedding，这个 embedding，就像我们上面说的，其实融合了各种异质信息。所以它是特别适合用来做召回的，比如拿到图网络中用户的 embedding 和物品 embedding，可以直接用来做向量召回。当然，物品和用户的 embedding 也可以作为特征，引入排序模型中，这都是比较自然的。有些推荐场景也可以直接根据 embedding 计算 user to user/item to item 的推荐结果，比如看了又看这种推荐场景。

早期的图神经网络做推荐，因为需要全局信息，所以计算速度是个问题，往往图规模都非常小，不具备实战价值。而 GraphSAGE 则通过一些手段比如从临近节点进行采样等减少计算规模，加快计算速度，很多后期改进计算效率的方法都是从这个工作衍生的；而 PinSage 在 GraphSAGE 基础上（这是同一拨人做的），进一步采取大规模分布式计算，拓展了图计算的实用性，可以计算 Pinterest 的 30 亿规模节点、180 亿规模边的巨型图，并产生了较好的落地效果。所以这两个工作可以重点借鉴一下。

总体而言，图模型召回，是个很有前景的值得探索的方向。

典型工作：

GraphSAGE: Inductive Representation Learning on Large Graphs

PinSage: Graph Convolutional Neural Networks for Web-Scale Recommender Systems

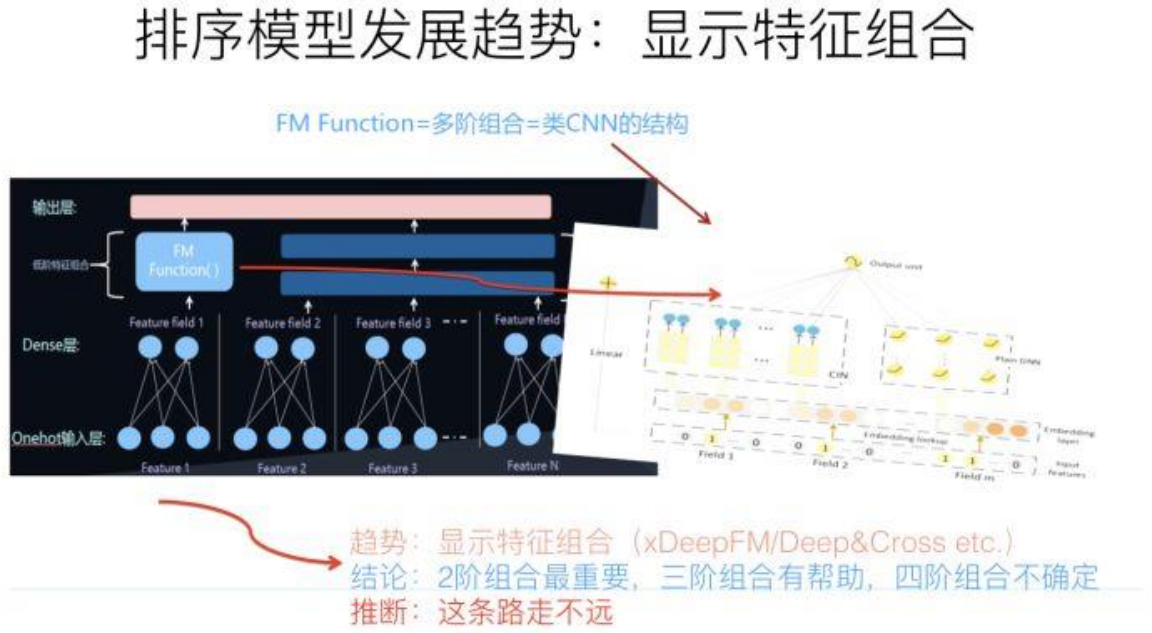
排序模型技术演进趋势

排序环节是推荐系统最关键，也是最具有技术含量的部分，目前大多数推荐技术其实都聚焦在这块。下面我们从模型表达能力、模型优化目标以及特征及信息三个角度分述推荐排序模型的技术发展趋势。

模型表达能力代表了模型是否具备充分利用有效特征及特征组合的能力，其中显示特征组合、新型特征抽取器、增强学习技术应用以及 AutoML 自动探索模型结构是这方面明显的技术进化方向；模型优化目标则体现了我们希望推荐系统去做好什么，往往跟业务目标有关联，这里我们主要从技术角度来探讨，而多目标优化以及 ListWise 最优是目前最常见的技术进化方向，ListWise 优化目标在排序阶段和重排阶段都可采用，我们把它放到重排部分去讲，这里主要介绍多目标优化；从特征和信息角度，如何采用更丰富的新类型特征，以及信息和特征的扩充及融合是主要技术进化方向，用户长短期兴趣分离、用户行为序列数据的使用、图神经网络以及多模态融合等是这方面的主要技术趋势，因为用户行为序列以及图神经网络在召回部分介绍过，这些点同样可以应用在排序部分，所以这里不再叙述这两点。

显式特征组合

如果归纳下工业界 CTR 模型的演化历史的话，你会发现，特征工程及特征组合的自动化，一直是推动实用化推荐系统技术演进最主要的方向，而且没有之一。最早的 LR 模型，基本是人工特征工程及人工进行特征组合的，简单有效但是费时费力；再发展到 LR+GBDT 的高阶特征组合自动化，以及 FM 模型的二阶特征组合自动化；再往后就是 DNN 模型的引入，纯粹的简单 DNN 模型本质上其实是在 FM 模型的特征 Embedding 化基础上，添加几层 MLP 隐层来进行隐式的特征非线性自动组合而已。所谓隐式，意思是并没有明确的网络结构对特征的二阶组合、三阶组合进行直接建模，只是通过 MLP，让不同特征发生交互，至于怎么发生交互的，怎么进行特征组合的，谁也说不清楚，这是 MLP 结构隐式特征组合的作用，当然由于 MLP 的引入，也会在特征组合时候考虑进入了特征间的非线性关系。



明白了隐式特征组合，也就明白了什么是显式特征组合。就是在模型结构中，明确设计一些子网络或者子结构，对二阶特征组合、三阶特征组合，甚至更高阶的特征组合进行表征。比如说 DeepFM，Deep 部分就是个典型的 DNN 模型，这个大家基本都会用，而 FM 部分则是明确对特征

二阶组合进行建模的子模型。这就是一个典型的显式二阶特征组合的模型。而如果进一步拓展的话，很自然想到的一个改进思路是：除了明确的把特征二阶组合做一个子结构，还可以把特征三阶特征组合，更高阶特征组合…..分别做一个模型子结构。融合这些子结构一起来做预测。这就是显式特征组合的含义，其实这条线的发展脉络是异常清晰的。典型的对高阶特征组合建模的比如 Deep& Cross、XDeepFM 模型等，就是这么个思路。

在两年多前，我一直以为这个方向是 CTR 或者推荐模型的关键所在，而且可能如何简洁融入更多特征组合是最重要且最有前景的方向。但是后来发现可能错了，目前基本对这个方向改变了看法。目前我对这个方向的看法是：这个方向确实很重要，但是未来可挖掘的潜力和空间很有限，在这条路上继续行进，应该不会走得太远。原因在于，目前基本很多经验已经证明了，显式的二阶特征组合是非常重要的，三阶特征组合对不同类型任务基本都有帮助。四阶特征组合已经说不清楚是否有用了，跟数据集有关系，有些数据集合引入显式 4 阶特征组合有帮助，有些数据集合没什么用。至于更高阶的特征组合，明确用对应的子结构建模，基本已经没什么用了，甚至是负面作用。这说明：我们在实际做事情的时候，其实显式结构把三阶特征组合引入，已经基本足够了。这是为什么说这条路继续往后走潜力不大的原因。

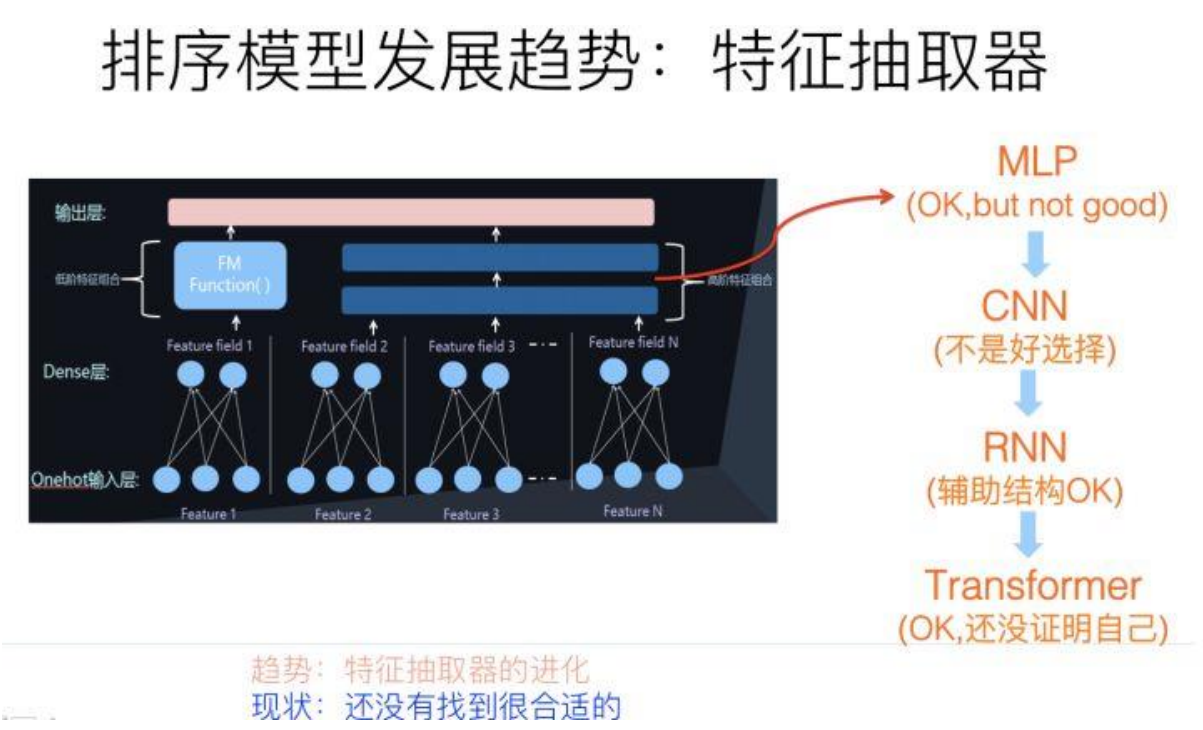
典型工作：

Deep& Cross: Deep & Cross Network for Ad Click Predictions

XDeepFM: Combining Explicit and Implicit Feature Interactions for Recommender Systems

特征抽取器的进化

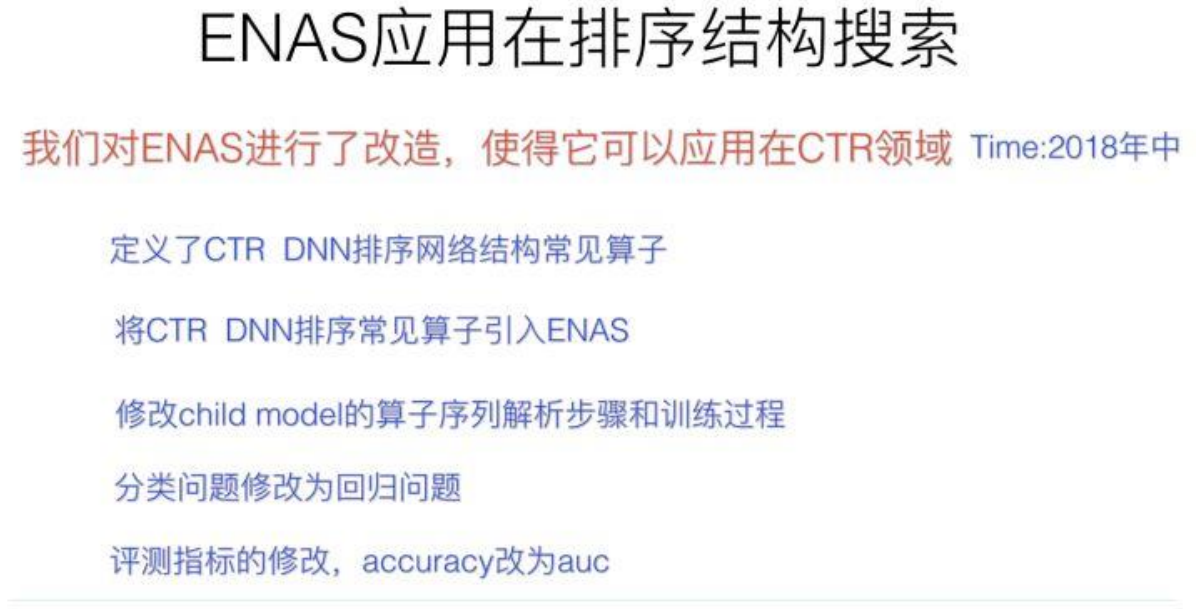
从特征抽取器的角度来看，目前主流的 DNN 排序模型，最常用的特征抽取器仍然是 MLP 结构，通常是两层或者三层的 MLP 隐层。目前也有理论研究表明：MLP 结构用来捕获特征组合，是效率比较低下的，除非把隐层神经元个数急剧放大，而这又会急剧增加参数规模。与自然语言处理和图像处理比较，推荐领域的特征抽取器仍然处于非常初级的发展阶段。所以，探寻新型特征抽取器，对于推荐模型的进化是个非常重要的发展方向。



目前其它 AI 领域里，常用的特征抽取器包括图像领域的 CNN、NLP 领域的 RNN 和 Transformer。这些新型特征抽取器，在推荐领域最近两年也逐步开始尝试使用，但是宏观地看，在推荐领域，相对 MLP 结构并未取得明显优势，这里的原因比较复杂。CNN 捕获局部特征关联是非常有效的结构，但是并不太适合做纯特征输入的推荐模型，因为推荐领域的特征之间，在输入顺序上并无必然的序列关系，基本属于人工定义随机顺序，而 CNN 处理这种远距离特征关系能力薄弱，所以并不

是特别适合用来处理特征级的推荐模型。当然，对于行为序列数据，因为本身带有序列属性，所以CNN 和 RNN 都是非常适合应用在行为序列结构上的，也是有一定应用历史的典型工具，但是对于没有序关系存在的特征来说，这两个模型的优势不能发挥出来，反而会放大各自的劣势，比如CNN 的捕获远距离特征关系能力差的弱点，以及 RNN 的不可并行处理、所以速度慢的劣势等。
Transformer 作为 NLP 领域最新型也是最有效的特征抽取器，从其工作机制来说，其实是非常适合用来做推荐的。为什么这么说呢？核心在于 Transformer 的 Multi-Head Self Attention 机制上。MHA 结构在 NLP 里面，会对输入句子中任意两个单词的相关程度作出判断，而如果把这种关系套用到推荐领域，就是通过 MHA 来对任意特征进行特征组合，而上文说过，特征组合对于推荐是个很重要的环节，所以从这个角度来说，Transformer 是特别适合来对特征组合进行建模的，一层Transformer Block 代表了特征的二阶组合，更多的 Transformer Block 代表了更高阶的特征组合。
但是，实际上如果应用 Transformer 来做推荐，其应用效果并没有体现出明显优势，甚至没有体现出什么优势，基本稍微好于或者类似于典型的 MLP 结构的效果。这意味着，可能我们需要针对推荐领域特点，对 Transformer 需要进行针对性的改造，而不是完全直接照搬 NLP 里的结构。
典型工作:
AutoInt: Automatic Feature Interaction Learning via Self-Attentive Neural Networks
DeepFM: An End-to-End Wide & Deep Learning Framework for CTR Prediction

AutoML 在推荐的应用
AutoML 在 17 年初开始出现，最近三年蓬勃发展，在比如图像领域、NLP 领域等都有非常重要的研究进展，在这些领域，目前都能通过 AutoML 找到比人设计的效果更好的模型结构。AutoML 作为算法方向最大的领域趋势之一，能否在不同领域超过人类专家的表现？这应该不是一个需要回答“会不会”的问题，而是应该回答“什么时间会”的问题。原因很简单，AutoML 通过各种基础算子的任意组合，在超大的算子组合空间内，寻找性能表现最好的模型，几乎可以达到穷举遍历的效果，而人类专家设计出来的最好的模型，无非是算子组合空间中的一个点而已，而且人类专家设计的那个模型，是最好模型的可能性是很低的。如果设计精良的 AutoML，一定可以自己找到超过目前人类专家设计的最好的那个模型，这基本不会有什么疑问，就像人类就算不是 2017 年，也会是某一年，下围棋下不过机器，道理其实是一样的，因为 AutoML 在巨大的算子组合空间里寻找最优模型，跟围棋在无穷的棋盘空间寻找胜利的盘面，本质上是一个事情。无非，现在 AutoML 的不成熟，体现在需要搜索的空间太大，比较消耗计算资源方面而已，随着技术的不断成熟，搜索成本越来越低，AutoML 在很多算法方向超过人类表现只是个时间问题。



DNN Ranking Model基本算子

类型1. Field级别的Attention算子 :MLP，SENET，Multi-Head…

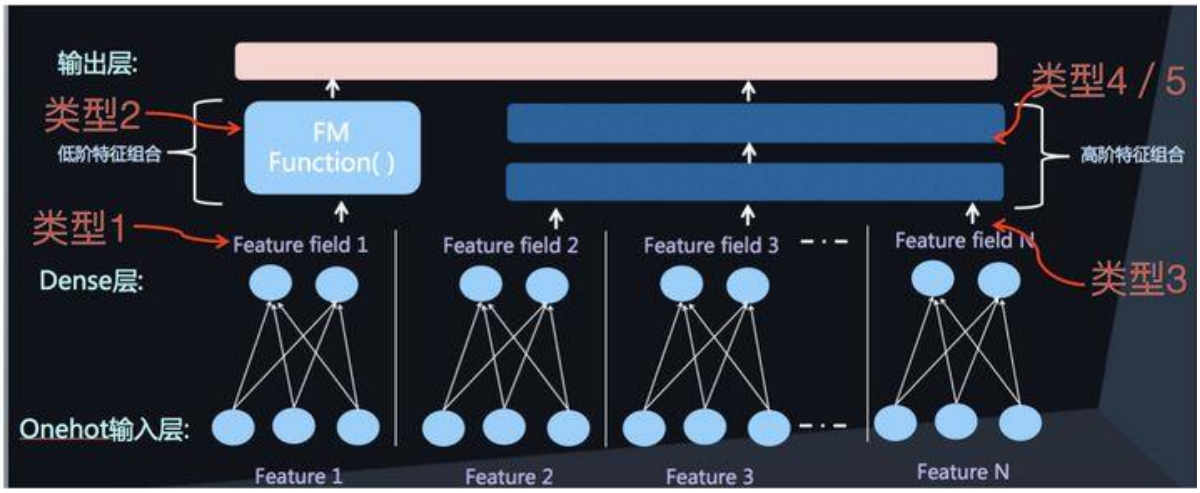
类型2. 二阶特征组合算子 :FM, HFM, CIN, Bilinear三种类型…

类型3. DNN输入算子: 一阶，二阶,一阶+二 阶,一阶+二阶n维度…

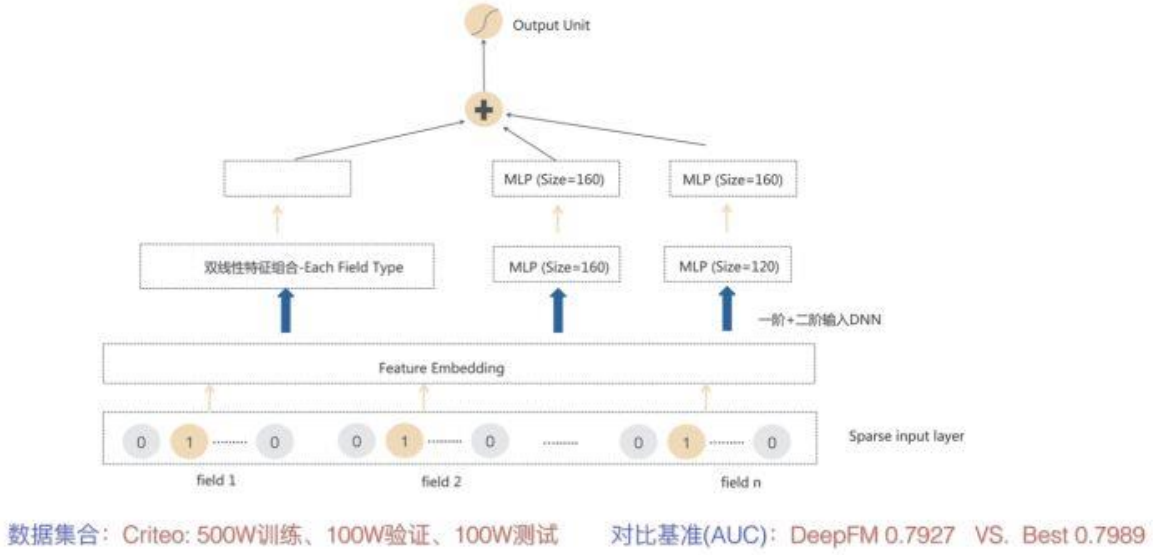
类型4. DNN的隐层个数:40-160，100-400, 400-900…

类型5. DNN的激活函数:relu, sigmoid, identity,tanh…

DNN Ranking的典型结构

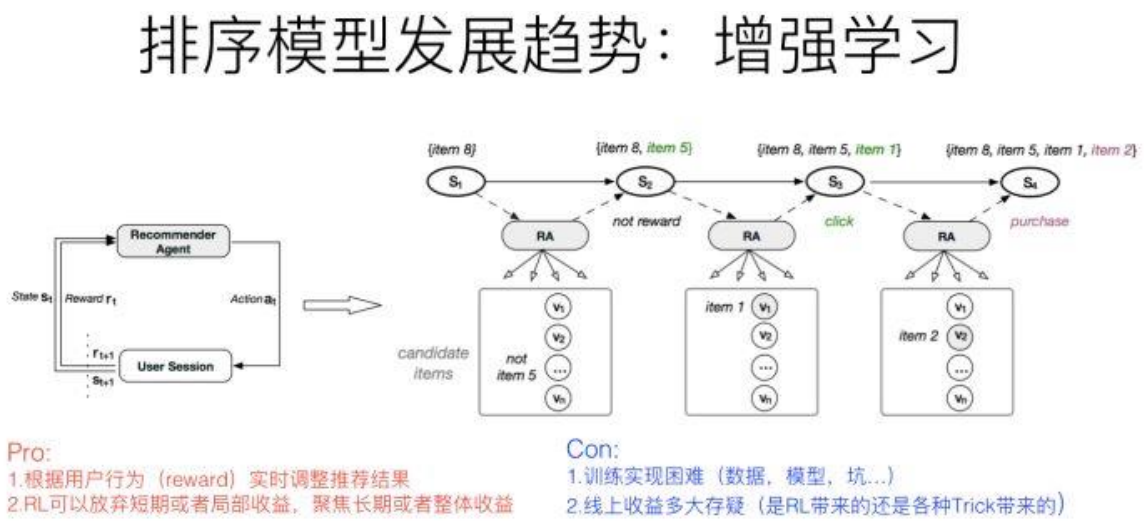


在推荐领域，采用 AutoML 做网络结构的工作还很少，这里面有很多原因。由于我一直以来特别看好这个方向，所以在 18 年的时候，我们也尝试过利用 AutoML 来自动探索推荐系统的网络结构，这里非常简略地介绍下过程及结果（参考上面三图）。我们用 ENAS 作为网络搜索工具，设计了推荐领域网络结构自动探索的尝试。ENAS 是个非常高效率的 AutoML 工具，可以做到单 GPU 半天搜索找到最优的网络结构，但是它定义的主要是 CNN 结构和 RNN 结构搜索。我们对 ENAS 进行了改造，包括算子定义，优化目标以及评价指标定义等。DNN 排序模型因为模型比较单一，所以算子是比较好找的，我们定义了推荐领域的常用算子，然后在这些算子组合空间内通过 ENAS 自动寻找效果最优的网络结构，最终找到的一个表现最好的网络结构如下图所示:



首先是特征 onehot 到 embedding 的映射，我们把这层固定住了，不作为模型结构探索因子。在特征 embedding 之上，有三个并行结构，其中两个是包含两个隐层的 MLP 结构，另外一个是特征双线性组合模块（Each Fields Type，具体含义可以参考下面的 FibiNet）。其表现超过了 DeepFM 等人工结构，但是并未超过很多。（感谢黄通文同学的具体尝试）
总体而言，目前 AutoML 来做推荐模型，还很不成熟，找出的结构相对人工设计结构效果优势也不是太明显。这与 DNN Ranking 模型比较简单，算子类型太少以及模型深度做不起来也有很大关系。但是，我相信这里可以有更进一步的工作可做。
典型工作：
ENAS 结构搜索：
双线性特征组合: FiBiNET: Combining Feature Importance and Bilinear feature Interaction for Click-Through Rate Prediction

增强学习在推荐的应用



增强学习其实是比较吻合推荐场景建模的。一般而言，增强学习有几个关键要素：状态、行为以及回报。在推荐场景下，我们可以把状态 St 定义为用户的行为历史物品集合；推荐系统可选的行为空间则是根据用户当前状态 St 推荐给用户的推荐结果列表，这里可以看出，推荐场景下，用户行为空间是巨大无比的，这制约了很多无法对巨大行为空间建模的增强学习方法的应用；而回报呢，则是用户对推荐系统给出的列表内容进行互动的行为价值，比如可以定义点击了某个物品，则回报是 1，购买了某个物品，回报是 5…诸如此类。有了这几个要素的场景定义，就可以用典型的增强学习来对推荐进行建模。

利用增强学习来做推荐系统，有几个显而易见的好处，比如：

. 比较容易对“利用-探索”（Exploitation/Exploration）建模。所谓利用，就是推荐给用户当前收益最大的物品，一般推荐模型都是优化这个目标；所谓探索，就是随机推给用户一些物品，以此来探测用户潜在感兴趣的东西。如果要进行探索，往往会牺牲推荐系统的当前总体收益，毕竟探索效率比较低，相当的通过探索渠道推给用户的物品，用户其实并不感兴趣，浪费了推荐位。但是，利用-探索的均衡，是比较容易通过调节增强学习的回报（Reward）来体现这个事情的，比较自然；
. 比较容易体现用户兴趣的动态变化。我们知道，用户兴趣有长期稳定的，也有不断变化的。而增强学习比较容易通过用户行为和反馈的物品对应的回报的重要性，而动态对推荐结果产生变化，所以是比较容易融入体现用户兴趣变化这个特点的。
. 有利于推荐系统长期收益建模。这点是增强学习做推荐最有优势的一个点。我们优化推荐系统，往往会有一些短期的目标比如增加点击率等，但是长期目标比如用户体验或者用户活跃留存等指标，一般不太好直接优化，而增强学习模型比较容易对长期收益目标来进行建模。

说了这么多优点，貌似增强学习应该重点投入去做，是吧？我的意见正好相反，觉得从实际落地角度来看，推荐系统里要尝试增强学习方法，如果你有这个冲动，最好还是抑制一下。主要原因是，貌似增强学习是技术落地投入产出比非常低的技术点。首先投入高，要想把增强学习做 work，意味着有很多大坑在等着你去踩，数据怎么做、模型怎么写、回报怎么拍，长期收益怎么定义、建模并拆解成回报……超大规模实际场景的用户和物品，增强学习这么复杂的模型，系统怎么才能真的落地并撑住流量…..很多坑在里面；其次，貌似目前看到的文献看，貌似很少见到真的把增强学习大规模推到真实线上系统，并产生很好的收益的系统。Youtube 在最近一年做了不少尝试，虽说把系统推上线了，但是收益怎样不好说。而且，从另外一个角度看，做增强学习里面还是有不少 Trick 在，那些收益到底是系统带来的，还是 Trick 带来的，真还不太好说。所以，综合而言，目前看在增强学习做推荐投入，貌似还是一笔不太合算的买卖。当然，长远看，可能还是很有潜力的，但是貌似这个潜力还需要新的技术突破去推动和挖掘。

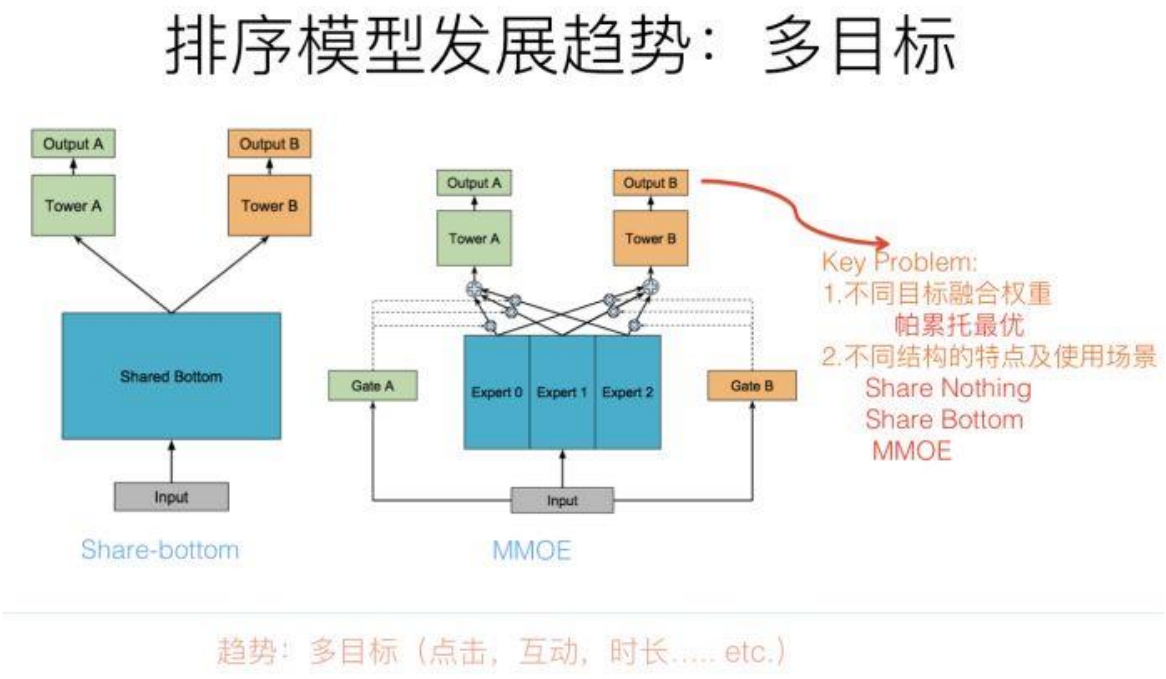
典型工作：

Youtube: Top-K Off-Policy Correction for a REINFORCE Recommender System

Youtube: Reinforcement Learning for Slate-based Recommender Systems: A Tractable Decomposition and Practical Methodology

多目标优化

推荐系统的多目标优化（点击，互动，时长等多个目标同时优化）严格来说不仅仅是趋势，而是目前很多公司的研发现状。对于推荐系统来说，不同的优化目标可能存在互相拉后腿的现象，比如互动和时长，往往拉起一个指标另外一个就会明显往下掉，而多目标旨在平衡不同目标的相互影响，尽量能够做到所有指标同步上涨，即使很难做到，也尽量做到在某个优化目标上涨的情况下，不拉低或者将尽量少拉低其它指标。多目标优化对于实用化的推荐系统起到了举足轻重的作用，这里其实是有很多工作可以做的，而如果多目标优化效果好，对于业务效果的推动作用也非常大。总而言之，多目标优化是值得推荐系统相关研发人员重点关注的技术方向。



从技术角度讲，多目标优化最关键的有两个问题。第一个问题是多个优化目标的模型结构问题；第二个问题是不同优化目标的重要性如何界定的问题。

既然存在多个优化目标，最简单直接的方式，也是目前最常用的方式是：每个

优化目标独立优化，比如点击目标训练一个模型，互动目标训练一个模型，时长目标训练一个模型，各自优化，然后每个目标独立给实例预测打分，给每个目标设定权重值，各个目标打分加权求和线性融合，或者引入权重指数及根据目标关系引入非线性融合。这是目前最常见的落地方案。因为目标之间独立优化，模型是通过分数融合来实现多目标的，所以可以把这种多目标方式称作“Share-Nothing”结构。这个结构实现和优化方式很简单。

与 Share-Nothing 结构相比，其实我们是可以让不同优化目标共享一部分参数的，一旦引入不同目标或者任务的参数共享，我们就踏入了 Transfer Learning 的领地了。那么为什么要共享参数呢？一方面出于计算效率考虑，不同目标共享结构能够提升计算效率；另外一点，假设我们有两类任务或者目标，其中一个目标的训练数据很充分，而另外一个目标的训练数据比较少；如果独立优化，训练数据少的目标可能很难获得很好的效果；如果两个任务相关性比较高的话，其实我们可以通过共享参数，达到把大训练数据任务的知识迁移给训练数据比较少的任务的目的，这样可以极大提升训练数据量比较少的任务的效果。Share-Bottom 结构是个非常典型的共享参数的多目标优化结构，核心思想是在比如网络的底层参数，所有任务共享参数，而上层网络，不同任务各自维护自己独有的一部分参数，这样就能达成通过共享参数实现知识迁移的目的。但是，Share-Bottom 结构有他的缺点：如果两个任务不那么相关的话，因为强制共享参数，所以可能任务之间相互干扰，会拉低不同目标的效果。MMOE 针对 Share-Bottom 结构的局限进行了改进，核心思想也很简单，就是把底层全部共享的参数切分成小的子网络，不同任务根据自己的特点，学习配置不同权重的小网络来进行参数共享。这样做的话，即使是两个任务不太相关，可以通过不同的配置来达到模型解耦的目的，而如果模型相关性强，可以共享更多的子网络。明显这样的组合方式更灵活，所以对于 MMOE 来说，无论是相关还是不相关的任务，它都可以达到我们想要的效果。

上面介绍的是典型的不同多目标的模型结构，各自有其适用场景和特点。而假设我们选定了模型结构，仍然存在一个很关键的问题：不同优化目标权重如何设定？当然，我们可以根据业务要求，强制制定一些权重，比如视频网站可能更重视时长或者完播率等指标，那就把这个目标权重设置大一些。但是，我们

讲过，有些任务之间的指标优化是负相关的，提升某个目标的权重，有可能造成另外一些指标的下跌。所以，如何设定不同目标权重，能够尽量减少相互之间的负面影响，就非常重要。这块貌似目前并没有特别简单实用的方案，很多实际做法做起来还是根据经验拍一些权重参数上线 AB 测试，费时费力。而如何用模型自动寻找最优权重参数组合就是一个非常有价值的方向，目前最常用的方式是采用帕累托最优的方案来进行权重组合寻优，这是从经济学引入的技术方案，未来还有很大的发展空间。

典型工作：

MMOE：Modeling Task Relationships in Multi-task Learning with Multi-gate Mixture-of-Experts

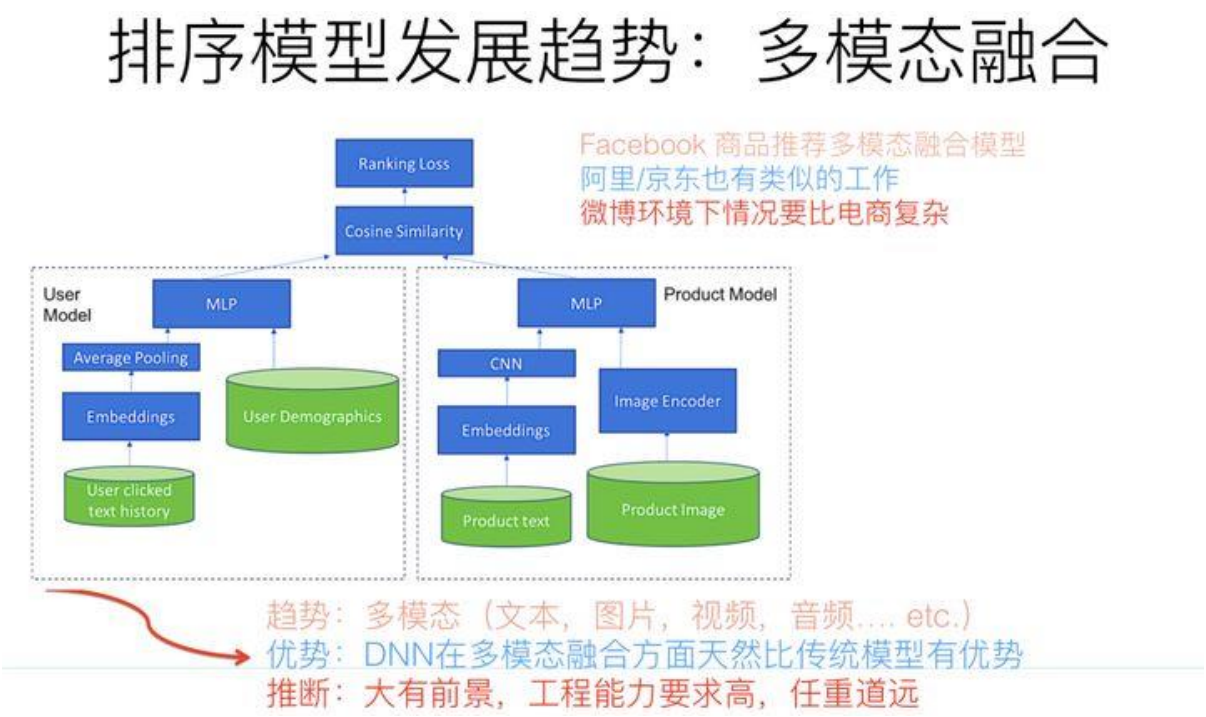
帕累托最优：A Pareto-Efficient Algorithm for Multiple Objective Optimization in E-Commerce Recommendation

多模态信息融合

所谓模态，指的是不同类型的或者模态形式的信息存在形式，比如文本、图片、视频、音频、互动行为、社交关系等，都是信息不同的存在模态形式。如果类比一下的话，就仿佛我们人类感知世界，也是用不同的感官来感知不同的信息类型的，比如视觉、听觉、味觉、触觉等等，就是接受不同模态类型的信息，而大脑会把多模态信息进行融合，来接受更全面更综合的世界知识。类似的，如何让机器学习模型能够接受不同模态类型的信息，并做知识和信息互补，更全面理解实体或者行为。这不仅仅是推荐领域的技术发现趋势，也是人工智能几乎所有方向都面临的重大发展方向，所以这个方向特别值得重视。

多模态融合，从技术手段来说，本质上是把不同模态类型的信息，通过比如

Embedding 编码，映射到统一的语义空间内，使得不同模态的信息，表达相同语义的信息完全可类比。比如说自然语言说的单词“苹果”，和一张苹果的图片，应该通过一定的技术手段，对两者进行信息编码，比如打出的 embedding，相似度是很高的，这意味着不同模态的知识映射到了相同的语义空间了。这样，你可以通过文本的苹果，比如搜索包含苹果的照片，诸如此类，可以玩出很多新花样。



在推荐场景下，多模态融合其实不是个很有难度的算法方向，大的技术框架仍然遵循目前主流的技术框架，比如 DNN Ranking。为了体现多模态集成的目标，可以在 User 侧或者 Item 侧，把多模态信息作为新的特征融入，比如加入 CNN 特征抽取器，把商品图片的特征抽取出来，作为商品侧的一种新特征，不同模态的融入，很可能意味着找到对应的特征抽取器，以新特征的方式融入，而有监督学习的学习目标会指导特征抽取器抽出那些有用的特征。所以，你可以看到，如果在推荐里融入多模态，从算法层面看，并不难，它的难点其实在它处；本质上，多模态做推荐，如果说难点的话，难在工程效率。因为目前很多模态的信息抽取器，比如图片的特征抽取，用深层 ResNet 或者 ReceptionNet，效果都很好，但是因为网络层深太深，抽取图片特征的速度问

题就是多模态落地面临的主要问题。所以，本质上，在推荐领域应用多模态，看上去其实是个工程效率问题，而非复杂的算法问题。而且，如果融合多模态的话，离开 DNN 模型，基本是不现实的。在这点上，可以比较充分体现 DNN 模型相对传统模型的绝对技术优势。

多模态信息融合，不仅仅是排序端的一个发展方向，在召回侧也是一样的，比如用用户点击过的图片，作为图片类型的新召回路，或者作为模型召回的新特征。明显这种多模态融合是贯穿了推荐领域各个技术环节的。

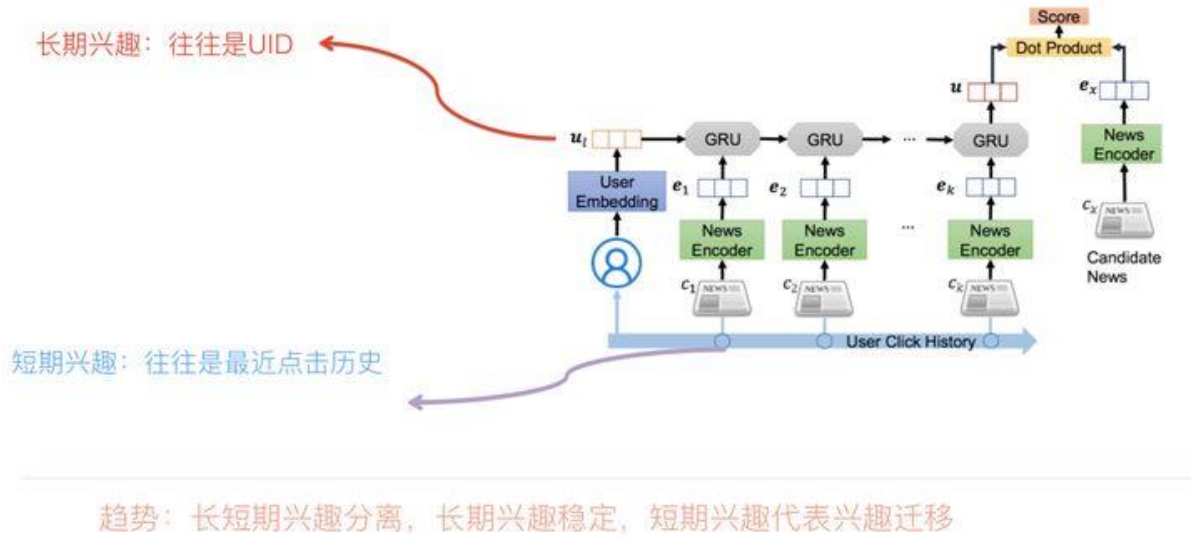
典型工作：

DNN 召回：Collaborative Multi-modal deep learning for the personalized product retrieval in Facebook Marketplace

排序：Image Matters: Visually modeling user behaviors using Advanced Model Server

长期兴趣 / 短期兴趣分离

对于推荐系统而言，准确描述用户兴趣是非常重要的。目前常用的描述用户兴趣的方式主要有两类。一类是以用户侧特征的角度来表征用户兴趣，也是最常见的；另外一类是以用户发生过行为的物品序列作为用户兴趣的表征。



我们知道，用户兴趣其实是可以继续细分的，一种典型的分法就是划分为长期兴趣和短期兴趣。长期兴趣代表用户长久的比较稳定的偏好；而短期兴趣具有不断变化等特点。两者综合，可以从稳定性和变化性这个维度来表征用户偏好。

最近推荐系统在排序侧模型的演进方向来说，把用户长期兴趣和短期兴趣分离并各自建立模型是个技术小趋势。那么用什么信息作为用户的短期兴趣表征？什么信息作为用户的长期兴趣表征呢？各自又用什么模型来集成这些信息呢？这是这个趋势的三个关键之处。

目前的常用做法是：用户短期兴趣往往使用用户点击（或购买，互动等其它行为类型）过的物品序列来表征，尤其对于比较活跃的用户，用点击序列更能体现短期的含义，因为出于工程效率的考虑，如果用户行为序列太长，往往不会都拿来使用，而是使用最近的 K 个行为序列中的物品，来表征用户兴趣，而这明显更含有短期的含义；因为点击序列具备序列性和时间属性，所以对于这类数据，用那些能够刻画序列特性或者物品局部相关性的模型比较合适，比如 RNN / CNN 和 Transformer 都比较适合用来对用户短期兴趣建模。

而用户长期兴趣如何表征呢？我们换个角度来看，其实传统的以特征作为用户兴趣表征的方法，其中部分特征就是从用户长期兴趣出发来刻画的，比如群体人群属性，是种间接刻画用户长期兴趣的方法，再比如类似用户兴趣标签，是种用用户行为序列物品的统计结果来表征用户长期兴趣的方法。这些方法当然可以用来刻画用户长期兴趣，但是往往粒度太粗，所以我们其实需要一个比较细致刻画用户长期兴趣的方式和方法。目前在对长短期兴趣分离的工作中，关于如何刻画用户长期兴趣，往往还是用非常简单的方法，就是用 UID 特征来表征用户的长期兴趣，通过训练过程对 UID 进行 Embedding 编码，以此学习到的 UID Embedding 作为用户长期兴趣表征，而用户行为序列物品作为用户短期兴趣表征。当然，UID 如果用一些其它手段比如矩阵分解获得的 Embedding 初始化，也是很有帮助的。

总而言之，用户长期兴趣和短期兴趣的分离建模，应该还是有意义的。长期兴趣目前建模方式还比较简单，这里完全可以引入一些新方法来进行进一步的兴趣刻画，而且有很大的建模空间。

典型工作：

1. Neural News Recommendation with Long- and Short-term User Representations

2. Sequence-Aware Recommendation with Long-Term and Short-Term Attention Memory Networks

---

重排技术演进趋势

在重排环节，常规的做法，这里是个策略出没之地，就是集中了各种业务和技术策略。比如为了更好的推荐体验，这里会加入去除重复、结果打散增加推荐结果的多样性、强插某种类型的推荐结果等等不同类型的策略。

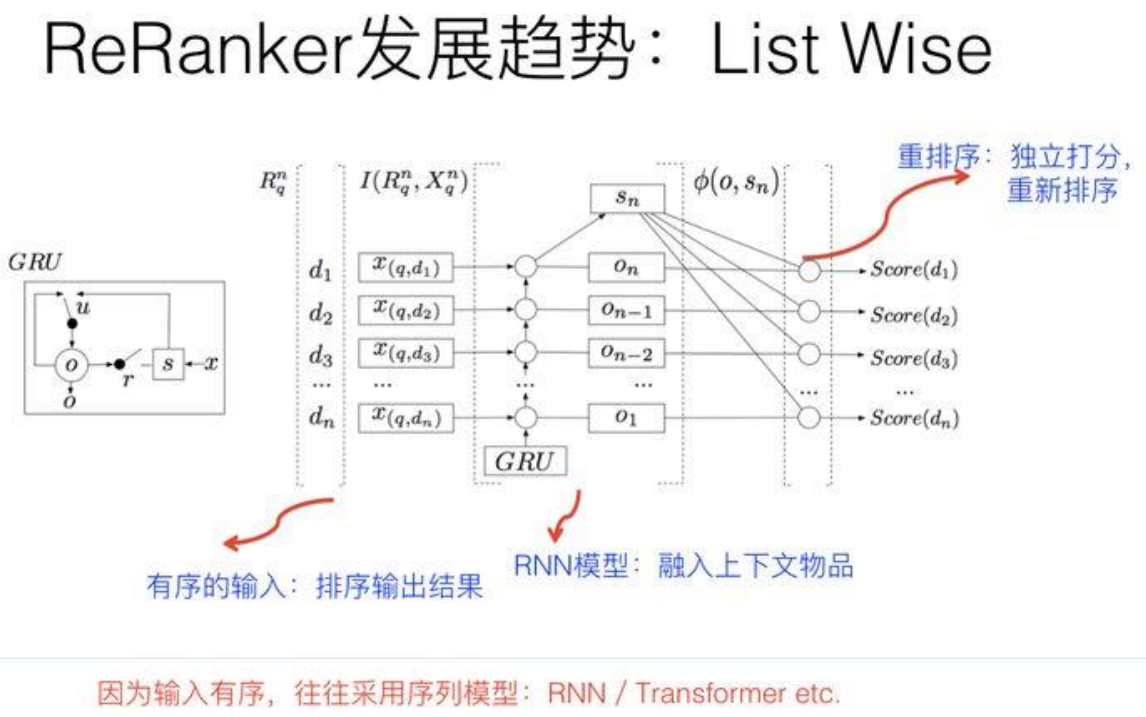
按理说，这块没什么可讲的。但是，如果从技术发展趋势角度看，重排阶段上模型，来代替各种花样的业务策略，是个总体的大趋势。

List Wise 重排序

关于 List Wise 排序，可以从两个角度来说，一个是优化目标或损失函数；一个是推荐模块的模型结构。

推荐系统里 Learning to Rank 做排序，我们知道常见的有三种优化目标：Point Wise、Pair Wise 和 List Wise。所以我们首先应该明确的一点是：List Wise 它不是指的具体的某个或者某类模型，而是指的模型的优化目标或者损失函数定义方式，理论上各种不用的模型都可以使用 List Wise 损失来进行模型训练。最简单的损失函数定义是 Point Wise，就是输入用户特征和单个物品特征，对这个物品进行打分，物品之间的排序，就是谁应该在谁前面，不用考虑。明显这种方式无论是训练还是在线推理，都非常简单直接效率高，但是它的缺点是没有考虑物品直接的关联，而这在排序中其实是有用的。Pair Wise 损失在训练模型时，直接用两个物品的顺序关系来训练模型，就是说优化目标是物品 A 排序要高于物品 B，类似这种优化目标。其实 Pair Wise 的 Loss 在推荐领域已经被非常广泛得使用，比如 BPR 损失，就是典型且非常有效的 Pair Wise 的 Loss Function，经常被使用，尤其在隐式反馈中，是非常有效的优化

目标。List Wise 的 Loss 更关注整个列表中物品顺序关系，会从列表整体中物品顺序的角度考虑，来优化模型。在推荐中，List Wise 损失函数因为训练数据的制作难，训练速度慢，在线推理速度慢等多种原因，尽管用的还比较少，但是因为更注重排序结果整体的最优性，所以也是目前很多推荐系统正在做的事情。



从模型结构上来看。因为重排序模块往往是放在精排模块之后，而精排已经对推荐物品做了比较准确的打分，所以往往重排模块的输入是精排模块的 Top 得分输出结果，也就是说，是有序的。而精排模块的打分或者排序对于重排模块来说，是非常重要的参考信息。于是，这个排序模块的输出顺序就比较重要，而能够考虑到输入的序列性的模型，自然就是重排模型的首选。我们知道，最常见的考虑时序性的模型是 RNN 和 Transformer，所以经常把这两类模型用在重排模块，这是很自然的事情。一般的做法是：排序 Top 结果的物品有序，作为 RNN 或者 Transformer 的输入，RNN 或者 Transformer 明显可以考虑在特征级别，融合当前物品上下文，也就是排序列表中其它物品，的特征，来从列表整体评估效果。RNN 或者 Transformer 每个输入对应位置经过特征融合，再

次输出预测得分，按照新预测的得分重新对物品排序，就完成了融合上下文信息，进行重新排序的目的。

尽管目前还没看到 CNN 做重排的方法，但是从机制上来说，明显 CNN 也是比较适合用来做重排环节模型的，感兴趣的同学可以试一试。当然，前面说的强化学习，也是非常适合用在 List Wise 优化的，目前也有不少相关工作出现。

典型工作：

1.Personalized Re-ranking for Recommendation

2.Learning a Deep Listwise Context Model for Ranking Refinement

好了，这篇文章又太长了，先写这么多吧。

致谢：本文的部分内容和观点，是在和一些同学的技术讨论过程中形成的，在此感谢微博机器学习团队佘青云和王志强同学的想法和建议。

编辑于 2019-12-29 12:35

50、请详细说说 FM 模型的原理

转自知乎

既然你点开这篇文章了，我假设你是在某司做推荐系统的算法工程师。这个假设的正确率我估计大约在 20%左右，因为根据我的经验，80%的算法工程师是很博爱的，只要标题里带有“模型 / 算法 / 深度学习 / 震惊/美女....”等词汇，他们都会好奇地点开

看三秒，然后失望地关掉，技术性越强的反而越容易被关掉，很可能撑不过三秒。我说得没错吧？嘿嘿。为了骗点击，关于本文标题，其实我内心冲动里最想写下的震惊部风格标题是这样的：“连女神级美女程序媛看了都震惊！FM 模型居然能够做这么大规模推荐系统的召回!!!”，然后打开文章后，文章配上的背景音乐缓缓地传来“路灯下昏黄的剪影,越走越漫长的林径……”

嗯，好吧，我承认连我自己也忍不了上面的场景，主要是这首歌我还挺喜欢的，单曲循环快半个月了，标题风格比较毁歌的意境。请收拾好您此刻看到上述标题后接近崩溃的心情，不开玩笑了。让我再次活回到幻想中，就勉强假设你是位推荐算法工程师吧，您坚持说您不是？别谦虚，您很快就是了，请立即辞职去申请相关工作……如果您真的是推荐工程师，那么首先我想揪住您问个问题：一说起推荐模型或者推荐场景下的排序模型，您脑子里第一个念头冒出的模型是哪个或哪几个？

如果你第一念头冒出来的仍然是 SVD / 矩阵分解啥的，那么明显你还停留在啃书本的阶段，实践经验不足；如果你第一念头是 LR 模型或者 GBDT 模型，这说明你是具备一定实践经验的算法工程师，但是知识更新不足。现在都 9102 年了，我们暂且把 Wide&Deep/DeepFM 这些模型抛开不提，因为在大规模场景下想要把深度推荐模型高性价比地用好发挥作用其实并不容易。我们退而求其次，如果现在您仍然不能在日常工作中至少尝试着用 FM 模型来搞事情，那只能说明一定概率下（30%到 90%？），您是在技术方面对自我没有太高要求的算法工程师，未来您的技术之路走起来，我猜可能会比较辛苦和坎坷，这里先向身处 2025 年的另一位您道声辛苦啦。这是我对您的算法工程师之路的一个预测，至于这个预测准不准，往后若干年的经历以及时间会告诉您正确答案，当然我个人觉得付出的这个代价可能有点高。

假设你第一念头是在排序阶段使用 FM 模型、GBDT+LR 模型、DNN 模型，这说明你算是紧追技术时代发展脉络的技术人员，很好。那么，单独给你准备的更专业的新问题来了，请问：树上七只猴.....嗯，跑偏了，其实我想问的是：我们日常看到的推荐系统长什么样子，我相信你脑子里很清楚，但是能否打破常规？比如下列两个不太符合常规做法的技术问题，您可以考虑考虑：

第一个问题：我们知道在个性化推荐系统里，第一个环节一般是召回阶段，而召回阶段工业界目前常规的做法是多路召回，每一路召回可能采取一个不同的策略。那么打破常规的思考之一是：**是否我们能够使用一个统一的模型，将多路召回改造成单模型**

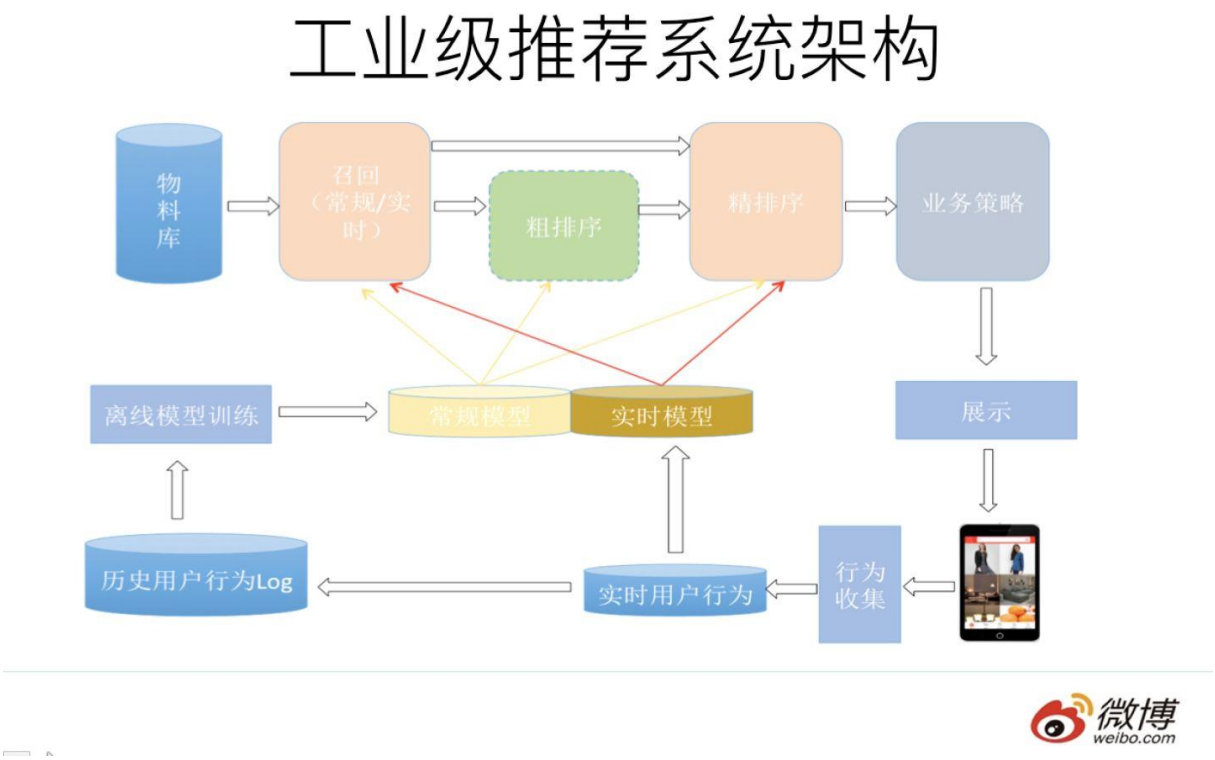
单路召回策略? 如果不能，那是为什么？如果能，怎么做才可以？这样做有什么好处和坏处？

第二个问题：我们同样知道，目前实用化的工业界的推荐系统通常由两个环节构成，召回阶段和排序阶段，那么为什么要这么划分？它们各自的职责是什么？打破常规的另外一个思考是：**是否存在一个模型，这个模型可以将召回阶段和排序阶段统一起来，就是把两阶段推荐环节改成单模型单环节推荐流程**？就是说靠一个模型一个阶段把传统的两阶段推荐系统做的事情一步到位做完？如果不能，为什么不能？如果能，怎么做才可以？什么样的模型才能担当起这种重任呢？而在现实世界里是否存在这个模型？这个思路真的可行吗？

上面列的两个非常规问题，18 年年末我自己也一直在思考，有些初步的思考结论，所以计划写四篇文章形成一个专题，主题集中在推荐系统的统一召回模型方面，也就是第一个问题，同时兼谈下第二个问题，每篇文章会介绍一个或者一类模型，本文介绍的是 FM 模型。这个系列，我春节期间写完了 3 篇，等我四篇都写完后，会陆续发出来，供感兴趣的“点开看三秒”同学参考。

不过这里需要强调一点：关于这两个问题，因为非常规，网上也也没有见到过类似的问题，说法及解决方案，所以没什么依据，文章写的只是我个人的思考结果，是否真能顺利落地以及落地效果存疑，还请谨慎参考。不过，我觉得从目前算法的发展趋势以及硬件条件的快速发展情况来看，单阶段推荐模型从理论上是可行的。我会陆续给出几个方案，建议从事中小型推荐业务的同学可以快速尝试一下。

下面进入正题，我会先简单介绍下推荐系统整体架构以及多路召回的基本模式，然后说明下 FM 模型，之后探讨 FM 模型是否能够解决上面提到的两个非常规问题。



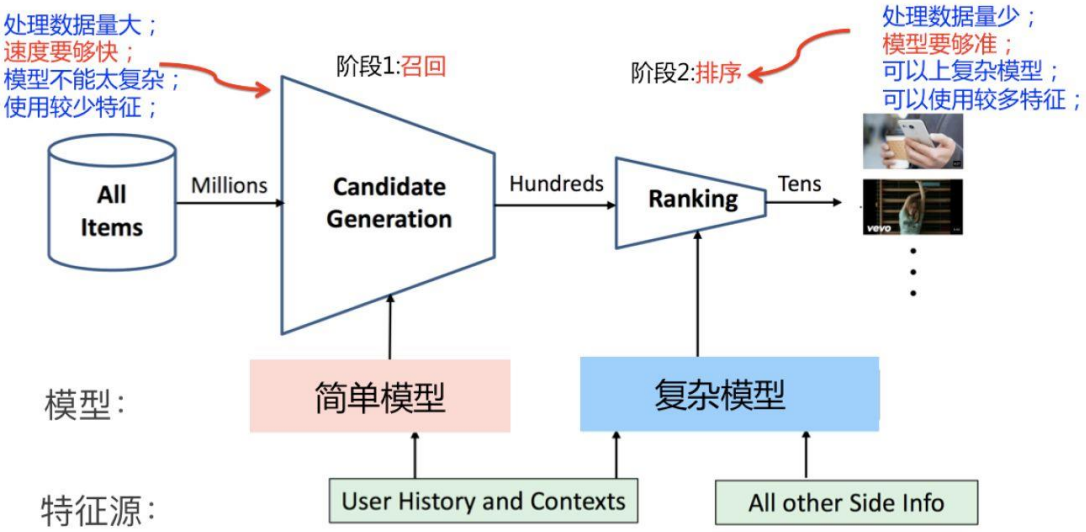
一个典型的工业级推荐系统整体架构可以参考上图，一般分为在线部分，近线部分和离线部分。

对于在线部分来说，一般要经历几个阶段。首先通过召回环节，将给用户推荐的物品降到千以下规模；如果召回阶段返回的物品还是太多，可以加入粗排阶段，这个阶段是可选的，粗排可以通过一些简单排序模型进一步减少往后续环节传递的物品；再往后是精排阶段，这里可以使用复杂的模型来对少量物品精准排序。对某个用户来说，即使精排推荐结果出来了，一般并不会直接展示给用户，可能还要上一些业务策略，比如去已读，推荐多样化，加入广告等各种业务策略。之后形成最终推荐结果，将结果展示给用户。

对于近线部分来说，主要目的是实时收集用户行为反馈，并选择训练实例，实时抽取拼接特征，并近乎实时地更新在线推荐模型。这样做的好处是用户的最新兴趣能够近乎实时地体现到推荐结果里。

对于离线部分而言，通过对线上用户点击日志的存储和清理，整理离线训练数据，并周期性地更新推荐模型。对于超大规模数据和机器学习模型来说，往往需要高效地分布式机器学习平台来对离线训练进行支持。

推荐系统在线部分的两个阶段



因为粗排是可选的，对于大多数推荐系统来说，通常在线部分的主体分为两个阶段就够，第一个阶段是召回，第二个阶段是排序。因为个性化推荐需要给每个用户展现不同的信息流或者物品流，而对于每个用户来说，可供推荐的物品，在具备一定规模的公司里，是百万到千万级别，甚至上亿。所以对于每一个用户，如果对于千万级别物品都使用先进的模型挨个进行排序打分，明显速度上是算不过来的，资源投入考虑这么做也不划算。从这里可以看出，召回阶段的主要职责是：从千万量级的候选物品里，采取简单模型将推荐物品候选集合快速筛减到千级别甚至百级别，这样将候选集合数量降下来，之后在排序阶段就可以上一些复杂模型，细致地对候选集进行个性化排序。

从上面在线推荐两阶段任务的划分，我们可以看出，召回阶段因为需要计算的候选集合太大，所以要想速度快，就只能上简单模型，使用少量特征，保证泛化能力，尽量让用户感兴趣的物品在这个阶段能够找回来；而排序阶段核心目标是要精准，因为它处理的物品数据量小，所以可以采用尽可能多的特征，使用比较复杂的模型，一切以精准为目标。

多路召回怎么做

目前工业界推荐系统的召回阶段一般是怎么做的呢？可以用一句江湖气很重的话来总结，请您系好安全带坐稳，怕吓到您，这句话就是：“一只穿云箭，千军万马来相见”。听起来霸气十足是吧？我估计看过古惑仔电影的都熟悉这句话，黑帮集结打群架的时候喜欢引用这句名言，以增加气势，自己给自己打气。如果和推荐系统对应起来理解，这里的“穿云箭”就是召回系统，而千军万马就是各路花式召回策略。

目前工业界的推荐系统，在召回阶段，一般都采取多路召回策略。上图展示了一个简化版本的例子，以微博信息流排序为例，不同业务召回路数不太一样，但是常用的召回策略，基本都会包含，比如兴趣标签，兴趣 Topic，兴趣实体，协同过滤，热门，相同地域等，多者几十路召回，少者也有 7 / 8 路召回。

对于每一路召回，会拉回 K 条相关物料，这个 K 值是个超参，需要通过线上 AB 测试来确定合理的取值范围。如果你对算法敏感的话，会发现这里有个潜在的问题，如果召回路数太多，对应的超参就多，这些超参组合空间很大，如何设定合理的各路召回数量是个问题。另外，如果是多路召回，这个超参往往不太可能是用户个性化的，而是对于所有用户，每一路拉回的数量都是固定的，这里明显有优化空间。按理说，不同用户也许对于每一路内容感兴趣程度是不一样的，更感兴趣的那一路就应该多召回一些，所以如果能把这些超参改为个性化配置是很好的，但是多路召回策略下，虽然也不是不能做，但是即使做，看起来还是很 Trick 的。有什么好办法能解决这个问题吗？有，本文后面会讲。

什么是 FM 模型

什么是 FM 模型呢？我隐约意识到这个问题在很多人看起来好像有点过于简单，因为一说起 FM，开车的朋友们估计都熟悉，比如 FM1039 交通台家喻户晓，最近应该经

常听到交通台这么提醒大家吧：“...春节返程高峰，北京市第三交通委提醒您：道路千万条，安全第一条.....”

一想到有可能很多人这么理解 FM，我的眼泪就不由自主流了下来，同时对他们在心理上有种莫名的亲切感，为什么呢？不是说“缩写不规范，亲人两行泪”么。下面我郑重地给各位介绍下，FM 英文全称是“Factorization Machine”，简称 FM 模型，中文名“因子分解机”。

FM 模型其实有些年头了，是 2010 年由 Rendle 提出的，但是真正在各大厂大规模在 CTR 预估和推荐领域广泛使用，其实也就是最近几年的事。

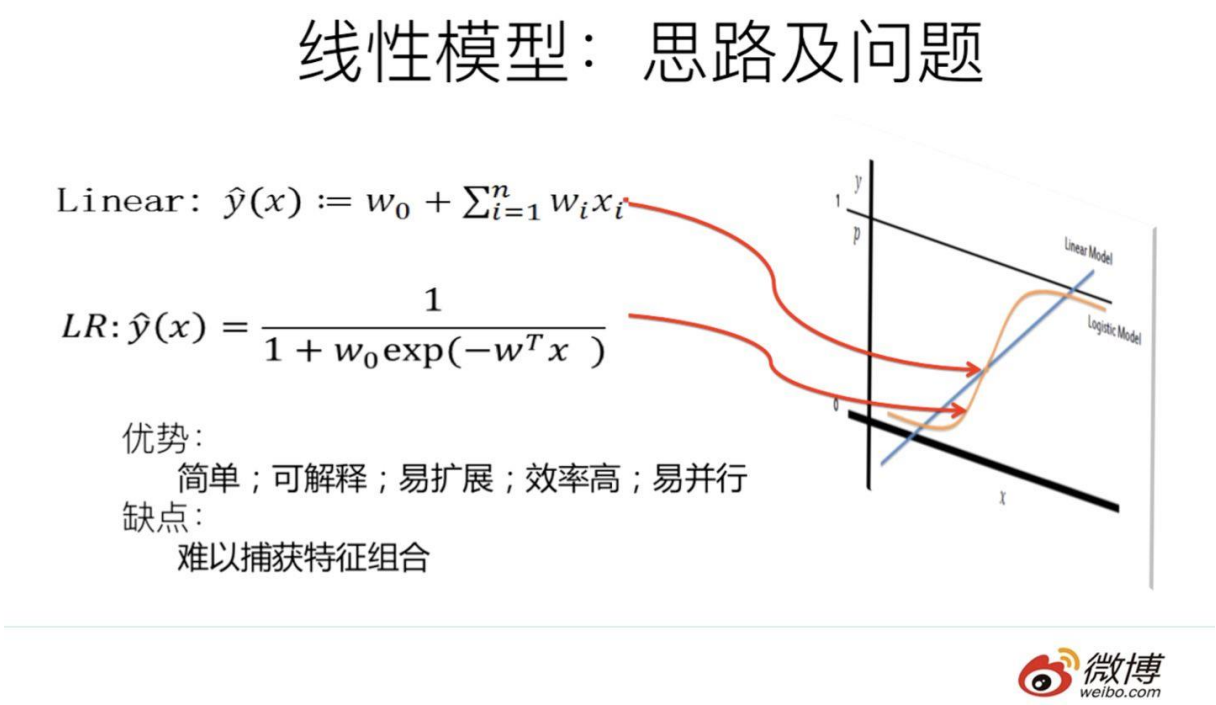
FM 模型比较简单，网上介绍的内容也比较多，细节不展开说它了。不过我给个个人判断：我觉得 FM 是推荐系统工程师应该熟练掌握和应用的必备算法，即使你看很多 DNN 版本的排序模型，你应该大多数情况会看到它的影子，原因其实很简单：特征组合对于推荐排序是非常非常重要的，而 FM 这个思路已经很简洁优雅地体现了这个思想了（主要是二阶特征组合）。DNN 模型一样离不开这个特点，而 MLP 结构是种低效率地捕获特征组合的结构，所以即使是深度模型，目前一样还离不开类似 FM 这个能够直白地直接去组合特征的部分。这是你会反复发现它的原因所在，当然也许是它本人，也许不一定是它本人，但是一定是它的变体。

既然谈到这里了，那顺手再多谈谈推荐排序模型。目前具备实用化价值的 DNN 版本的 CTR 模型一般采用 MLP 结构，看着远远落后 CV/NLP 的特征抽取器的发展水平，很容易让人产生如下感觉：CTR 的 DNN 模型还处于深度学习原始社会阶段。那这又是为什么呢？因为 CNN 的特性天然不太适合推荐排序这个场景（为什么？您可以思考一下。为了预防某些具备某种独特个性特征的同学拿个别例子说事情，我先提一句：请不要跟我说某个已有的看上去比较深的 CNN CTR 模型，你自己试过效果如何再来说。这算是我的预防性回怼或者是假设性回怼，哈哈）。RNN 作为捕捉用户行为序列，利用时间信息的辅助结构还行，但是也不太适合作为 CTR 预估或者推荐排序的主模型（为什么？您可以思考一下，关于这点，我的看法以后有机会会提）。好像剩下的选择不多了（Transformer 是很有希望的，去年年中左右，我觉得 Self attention 应该是个能很好地捕捉特征组合（包括二阶 / 三阶...多阶）的工具，于是，我们微博也尝试过用 self attention 和 transformer 作为 CTR 的主体排序模型，非业务数据测试的，当时测试效果和 DeepFM 等主流模型效果差不太多。我现在回头看，很可能是哪

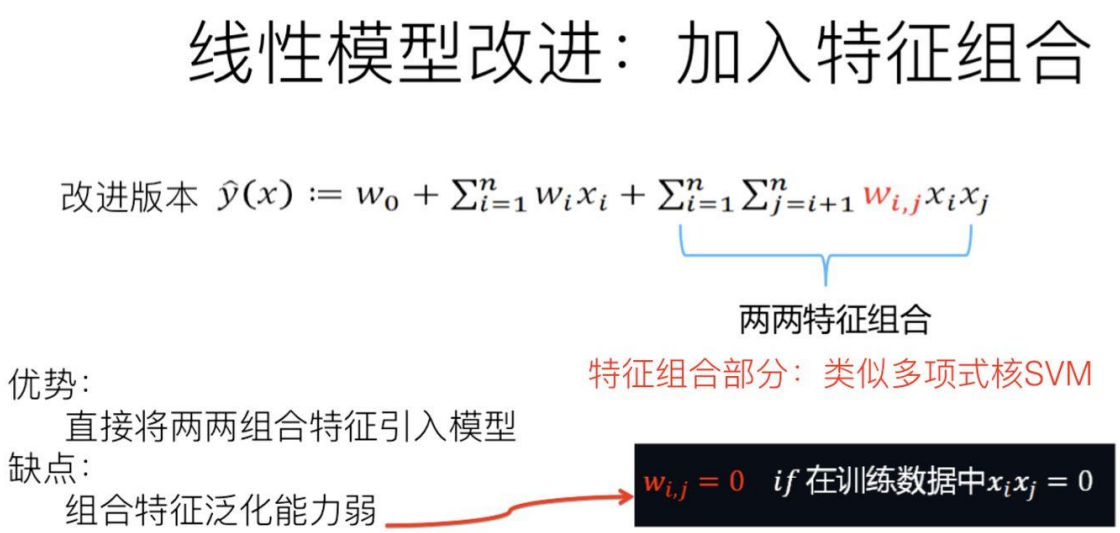
些细节没做对，当时觉得没有特别的效果优势，于是没再继续尝试这个思路。当然貌似 18 年下半年已经冒出几篇用 Transformer 做 CTR 排序模型的论文了，我个人非常看好这个 CTR 模型进化方向)，于是剩下的选择貌似只有 MLP 了，意思是：对于 CTR 或者推荐排序领域来说，不是它不想进入模型共产主义阶段，是大门关得太紧，它进不去，于是只能在 MLP 这个门槛徘徊。在深度学习大潮下，从模型角度看，确实跟很多领域比，貌似推荐领域远远落后，这个是事实。我觉得主要原因是它自身的领域特点造成的，它可能需要打造适合自身特点的 DNN 排序模型。就像图像领域里有 Resnet 时刻，NLP 里面有 Bert 时刻，我觉得推荐排序深度模型目前还没有，现在和未来也需要这个类似的高光时刻，而这需要一个针对它特性改造出的新结构，对此我是比较乐观的，我预感这个时刻一年之内还无法出现，但是很可能已经在路上，距离我们不远了。

又说远了，本来我们主题是召回，说到排序模型里去了，我往主车道走走。上面本来是要强调好好学好好用 FM 模型的。下面我从两个角度来简单介绍下 FM 模型，一个角度是从特征组合模型的进化角度来讲；另外一个角度从协同过滤模型的进化角度来讲。FM 模型就处于这两类模型进化的交汇口。

- **从 LR 到 SVM 再到 FM 模型**

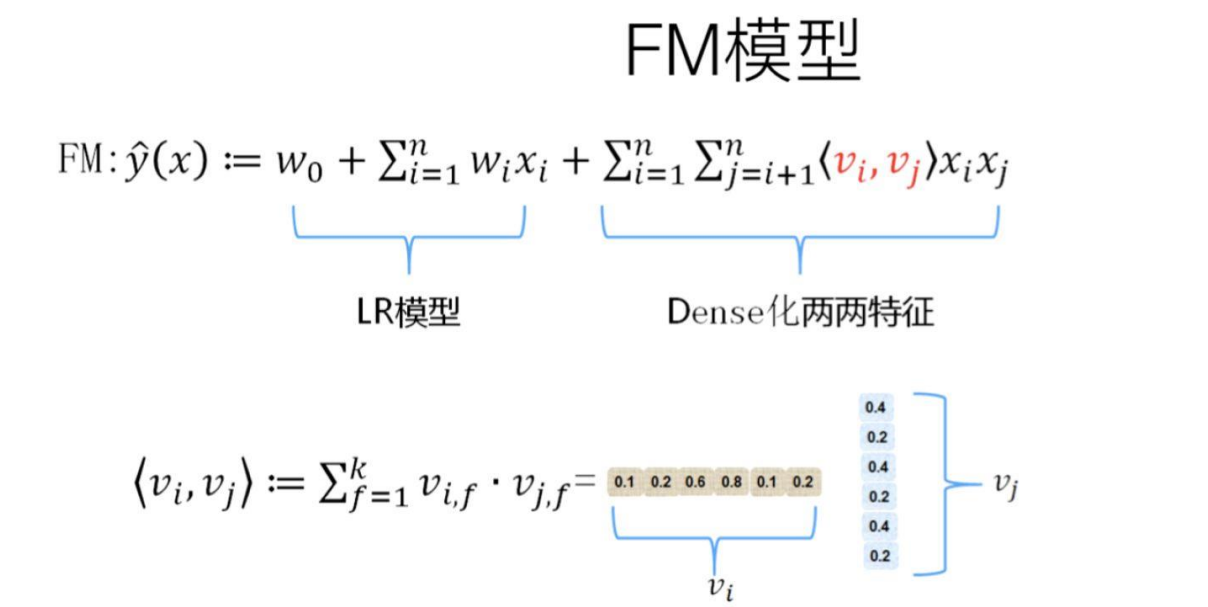


LR 模型是 CTR 预估领域早期最成功的模型，大多工业推荐排序系统采取 LR 这种“线性模型+人工特征组合引入非线性”的模式。因为 LR 模型具有简单方便易解释容易上规模等诸多好处，所以目前仍然有不少实际系统仍然采取这种模式。但是，LR 模型最大的缺陷就是人工特征工程，耗时费力费人力资源，那么能否将特征组合的能力体现在模型层面呢？

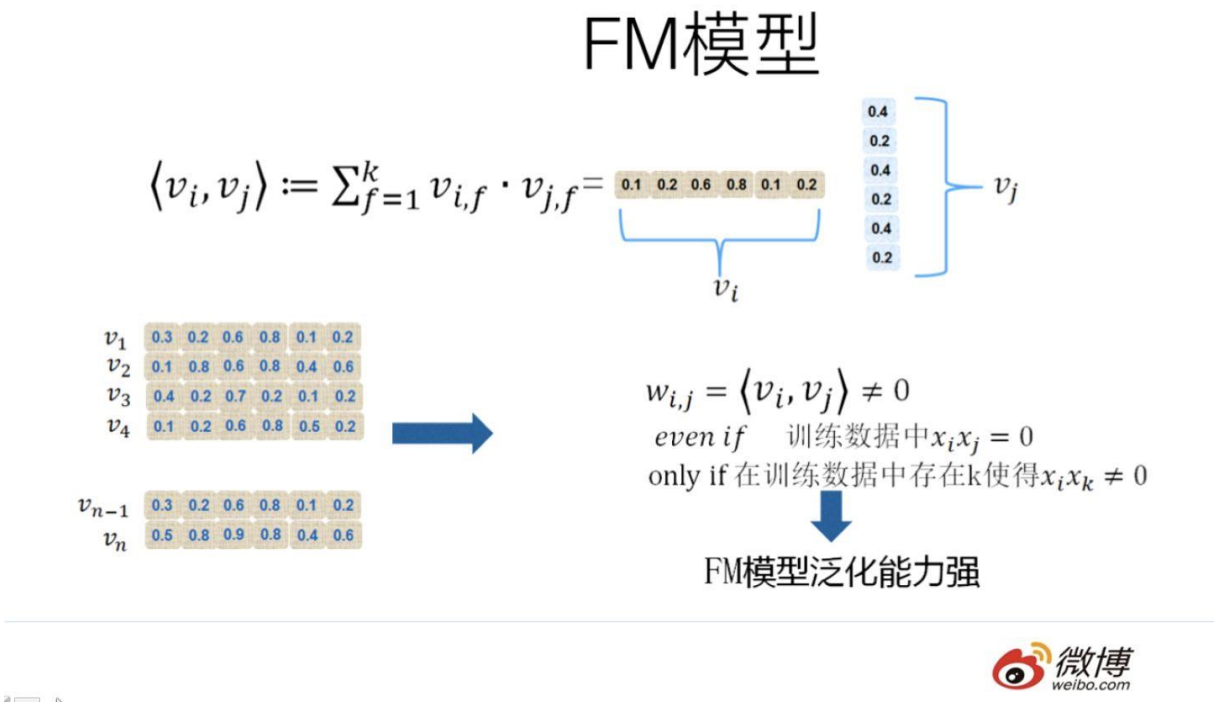


微博 weibo.com

其实想达到这一点并不难，如上图在计算公式里加入二阶特征组合即可，任意两个特征进行组合，可以将这个组合出的特征看作一个新特征，融入线性模型中。而组合特征的权重可以用来表示，和一阶特征权重一样，这个组合特征权重在训练阶段学习获得。其实这种二阶特征组合的使用方式，和多项式核 SVM 是等价的。虽然这个模型看上去貌似解决了二阶特征组合问题了，但是它有个潜在的问题：它对组合特征建模，泛化能力比较弱，尤其是在大规模稀疏特征存在的场景下，这个毛病尤其突出，比如 CTR 预估和推荐排序，这些场景的最大特点就是特征的大规模稀疏。所以上述模型并未在工业界广泛采用。那么，有什么办法能够解决这个问题吗？



于是，FM 模型此刻可以闪亮登场了。如上图所示，FM 模型也直接引入任意两个特征的二阶特征组合，和 SVM 模型最大的不同，在于特征组合权重的计算方法。FM 对于每个特征，学习一个大小为 k 的一维向量，于是，两个特征 xi 和 xj 的特征组合的权重值，通过特征对应的向量 vi 和 vj 的内积 ⟨vi,vj⟩ 来表示。这本质上是在对特征进行 embedding 化表征，和目前非常常见的各种实体 embedding 本质思想是一脉相承的，但是很明显在 FM 这么做的年代（2010 年），还没有现在能看到的各种眼花缭乱的 embedding 的形式与概念。所以 FM 作为特征 embedding，可以看作当前深度学习里各种 embedding 方法的老前辈。当然，FM 这种模式有它的前辈模型吗？有，等会会谈。其实，和目前的各种深度 DNN 排序模型比，它仅仅是少了 2 层或者 3 层 MLP 隐层，用来直接对多阶特征非线性组合建模而已，其它方面基本相同。



那么为什么说 FM 的这种特征 embedding 模式，在大规模稀疏特征应用环境下比较好用？为什么说它的泛化能力强呢？参考上图说明。即使在训练数据里两个特征并未同时在训练实例里见到过，意味着 xiandxj 一起出现的次数为 0，如果换做 SVM 的模式，是无法学会这个特征组合的权重的。但是因为 FM 是学习单个特征的 embedding，并不依赖某个特定的特征组合是否出现过，所以只要特征 xi 和其它任意特征组合出现过，那么就可以学习自己对应的 embedding 向量。于是，尽管 xiandxj 这个特征组合没有看到过，但是在预测的时候，如果看到这个新的特征组合，因为 xi 和 xj 都能学会自己对应的 embedding，所以可以通过内积算出这个新特征组合的权重。这是为何说 FM 模型泛化能力强的根本原因。

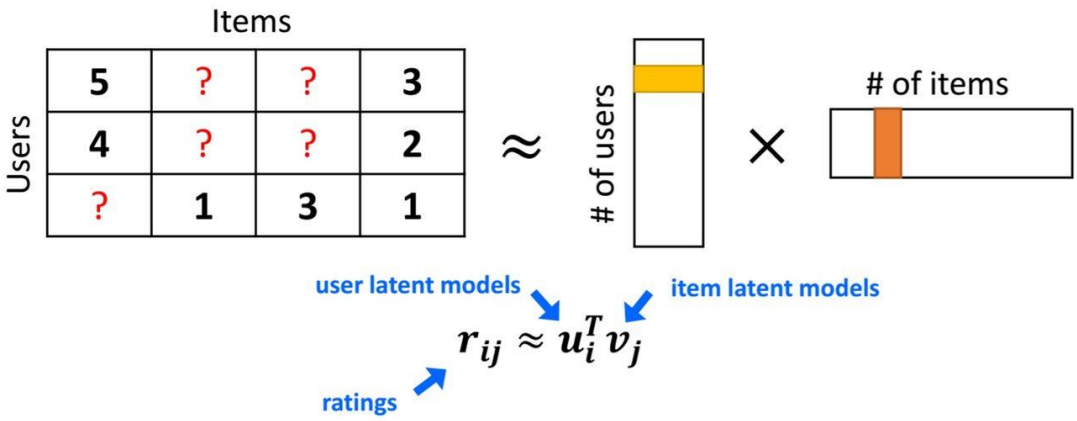
其实本质上，这也是目前很多花样的 embedding 的最核心特点，就是从 0/1 这种二值硬核匹配，切换为向量软匹配，使得原先匹配不上的，现在能在一定程度上算密切程度了，具备很好的泛化性能。

- **从 MF 到 FM 模型**

FM 我们大致应该知道是怎么个意思了，这里又突然冒出个 MF，长得跟 FM 貌似还有点像，那么 MF 又是什么呢？它跟 FM 又有什么关系？

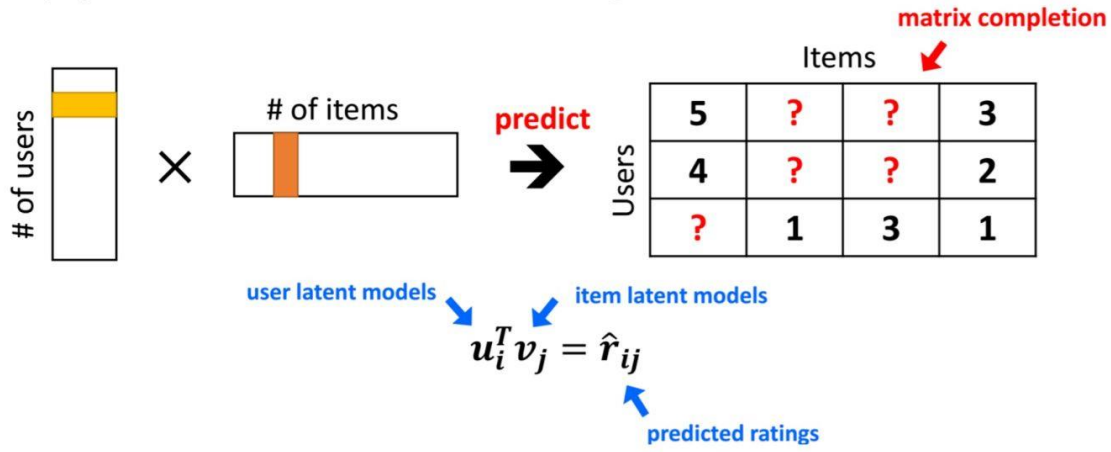
请跟我念：“打东边来了个 FM，手里提着一斤挞嘛；打西边来了个 MF，腰里别着一个喇叭；提着一斤挞嘛的 FM 想要别着喇叭的 MF 腰里的喇叭……”你要是能不打磕绊一遍念下来的话……你以为你就理解它们的错综复杂的关系了是吗？不，你就可以去学说相声了……

Matrix Factorization基本原理



MF（Matrix Factorization，矩阵分解）模型是个在推荐系统领域里资格很深的老前辈协同过滤模型了。核心思想是通过两个低维小矩阵（一个代表用户 embedding 矩阵，一个代表物品 embedding 矩阵）的乘积计算，来模拟真实用户点击或评分产生的大的协同信息稀疏矩阵，本质上是编码了用户和物品协同信息的降维模型。

Matrix Factorization基本原理

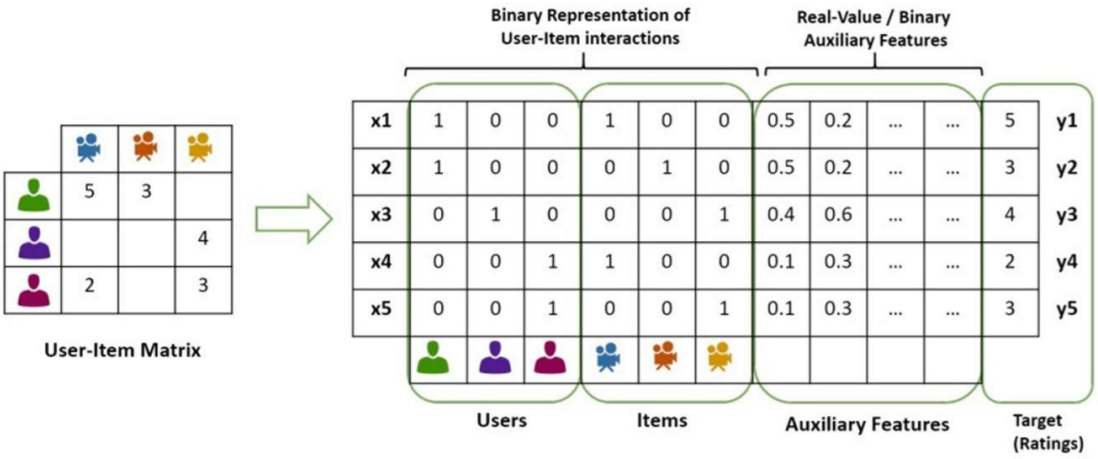


当训练完成，每个用户和物品得到对应的低维embedding表达后，如果要预测某个 $User_i$ 对 $Item_j$ 的评分的时候，只要它们做个内积计算 $\langle User_i, Item_j \rangle$ ，这个得分就是预测得分。看到这里，让你想起了什么吗?

身为推荐算法工程师，我假设你对它还是比较熟悉的，更多的就不展开说了，相关资料很多，我们重点说MF和FM的关系问题。

MF和FM不仅在名字简称上看着有点像，其实他们本质思想上也有很多相同点。那么，MF和FM究竟是怎样的关系呢?

Matrix Factorization到FM的转换



本质上，MF 模型是 FM 模型的特例，MF 可以被认为是只有 User ID 和 Item ID 这两个特征 Fields 的 FM 模型，MF 将这两类特征通过矩阵分解，来达到将这两类特征 embedding 化表达的目的。而 FM 则可以看作是 MF 模型的进一步拓展，除了 User ID 和 Item ID 这两类特征外，很多其它类型的特征，都可以进一步融入 FM 模型里，它将所有这些特征转化为 embedding 低维向量表达，并计算任意两个特征 embedding 的内积，就是特征组合的权重，如果 FM 只使用 User ID 和 Item ID，你套到 FM 公式里，看看它的预测过程和 MF 的预测过程一样吗？

从谁更早使用特征 embedding 表达这个角度来看的话，很明显，和 FM 比起来，MF 才是真正的前辈，无非是特征类型比较少而已。而 FM 继承了 MF 的特征 embedding 化表达这个优点，同时引入了更多 Side information 作为特征，将更多特征及 Side information embedding 化融入 FM 模型中。所以很明显 FM 模型更灵活，能适应更多场合的应用范围。

鉴于 MF 和 FM 以上错综复杂剪不断理还乱的关系，我推论出下面的观点（个人意见）：

其一：在你有使用 MF 做协同过滤的想法的时候，暂时压抑一下这种冲动，可以优先考虑引入 FM 来做的，而非传统的 MF，因为可以在实现等价功能的基础上，很方便地融入其它任意你想加入的特征，把手头的事情做得更丰富多彩。

其二：从实际大规模数据场景下的应用来讲，在排序阶段，绝大多数只使用 ID 信息的模型是不实用的，没有引入 Side Information，也就是除了 User ID / Item ID 外的很多其它可用特征的模型，是不具备实战价值的。原因很简单，大多数真实应用场景中，User/Item 有很多信息可用，而协同数据只是其中的一种，引入更多特征明显对于更精准地进行个性化推荐是非常有帮助的。而如果模型不支持更多特征的便捷引入，明显受限严重，很难真正实用，这也是为何矩阵分解类的方法很少看到在 Ranking 阶段使用，通常是作为一路召回形式存在的原因。

- **简单谈谈算法的效率问题**

从 FM 的原始数学公式看，因为在进行二阶（2-order）特征组合的时候，假设有 n 个不同的特征，那么二阶特征组合意味着任意两个特征都要进行交叉组合，所以可以直接推论得出：FM 的时间复杂度是 n 的平方。但是如果故事仅仅讲到这里，FM 模型是不太可能如此广泛地被工业界使用的。因为现实生活应用中的 n 往往是个非常巨大的特征数，如果 FM 是 n 平方的时间复杂度，那估计基本就没人带它玩了。

对于一个实用化模型来说，效果是否足够好只是一个方面，计算效率是否够高也很重要，这两点是一个能被广泛使用算法的一枚硬币的两面，缺其中任何一个可能都不能算是优秀的算法。如果在两者之间硬要分出谁更重要的话，怎么选？

这里插入个题外话，是关于如何做选择的。这个话题如果你深入思考的话，会发现很可能是个深奥的哲学问题。在说怎么选之前，我先复述两则关于选择的笑话，有两个版本，男版和女版的。

男版是这样的："一个兄弟跟我说他最近很困惑，有三个姑娘在追他，一直犹豫不决，到底应该选哪个当女朋友呢？一个温柔贤惠，一个聪明伶俐，另外一个肤白貌美。太难选……三天后当我再次遇到他的时候，他说他做出了选择，选了那个胸最大的！"

女版是这样的："一个姐妹跟我说她很困惑，最近有三个优秀的男人在追她，一直犹豫不决，到底应该嫁给谁呢？一个努力上进，一个高大帅气，另外一个脾气好顾家。实在太难选……三天后当我再次遇到她的时候，她说她做出了选择，选了那个最有钱的！"

参考这个模版，算法选择版应该是这样的："一个算法工程师一直犹豫不决该选哪个模型去上线，他有三个优秀算法可选，一个算法理论优雅；一个算法效果好；另外一个算法很时髦，实在太难做决定……三天后当我再遇见他的时候，他说他们算法总监让他上了那个跑得最快的！"

怎么样？生活或者工作中的选择确实是个很玄妙的哲学问题吧？这个算法版的关于选择的笑话，应该已经回答了上面那个还没给答案的问题了吧？在数据量特别大的情况下，如果在效果好和速度快之间做选择，很多时候跑得快的简单模型会胜出，这是为何 LR 模型在 CTR 预估领域一直被广泛使用的原因。

而 FFM 模型则是反例，我们在几个数据集合上测试过，FFM 模型作为排序模型，效果确实是要优于 FM 模型的，但是 FFM 模型对参数存储量要求太多，以及无法能做到 FM 的运行效率，如果中小数据规模做排序没什么问题，但是数据量一旦大起来，对资源和效率的要求会急剧升高，这是严重阻碍 FFM 模型大规模数据场景实用化的重要因素。

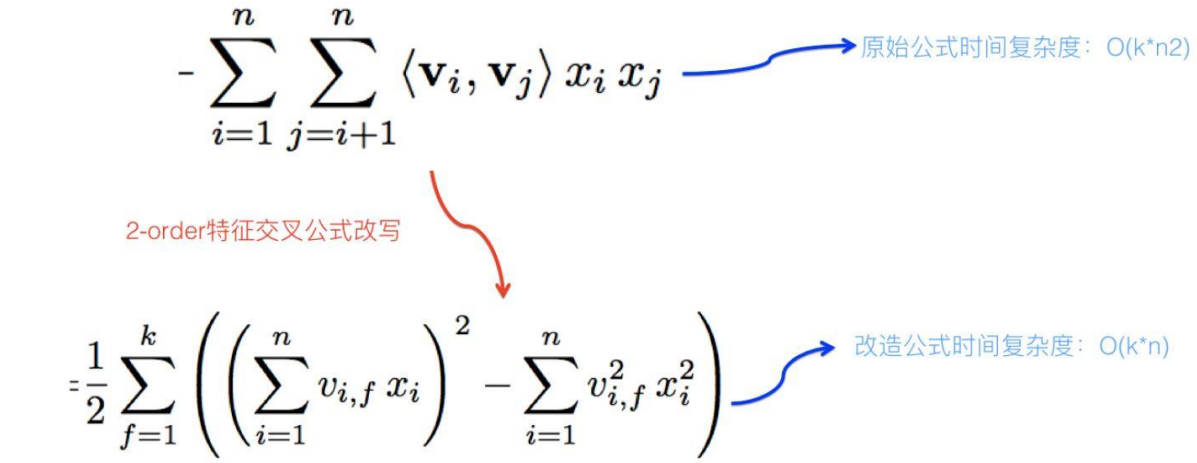
再顺手谈谈 DNN 排序模型，现在貌似看着有很多版本的 DNN 排序模型，但是考虑到上面讲的运算效率问题，你会发现太多所谓效果好的模型，其实不具备实用价值，算起来太复杂了，效果好得又很有限，超大规模训练或者在线 Serving 速度根本跟不上。除非，你们公司有具备相当强悍实力的工程团队，能够进行超大数据规模下的大规模性能优化，那当我上面这句话没说。

我对排序模型，如果你打算推上线真用起来的话，建议是，沿着这个序列尝试：FM-->DeepFM。

你看着路径有点短是吗？确实比较短。如果 DeepFM 做不出效果，别再试着去尝试更复杂的模型了，还是多从其它方面考虑考虑优化方案为好。有些复杂些的模型，也许效果确实好一些，在个别大公司也许真做上线了，但是很可能效果好不是算法的功劳，是工程能力强等多个因素共同导致的，人家能做，你未必做的了。至于被广泛尝试的 Wide &Deep，我个人对它有偏见，所以直接被我跳过了。当然，如果你原始线上版本是 LR，是可以直接先尝试 Wide&Deep 的，但是即使如此，要我做升级方案，我给的建议会是这个序列：LR—>FM-->DeepFM—>干点其他的。

- **如何优化 FM 的计算效率**

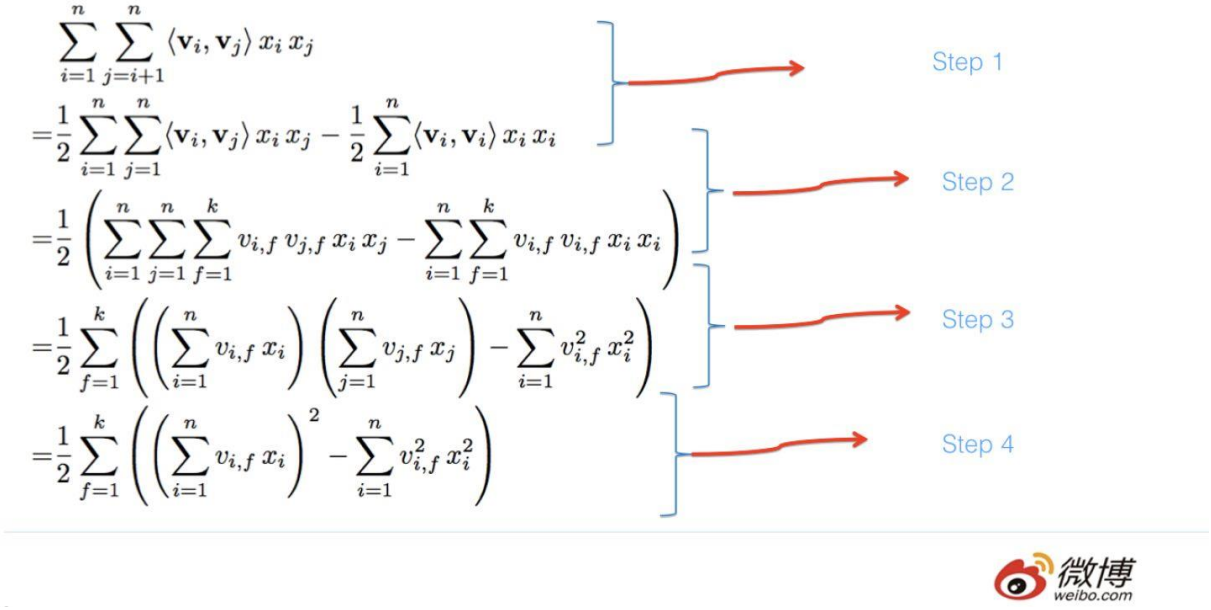
如何提升FM计算效率



再说回来，FM如今被广泛采用并成功替代LR模型的一个关键所在是：它可以通过数学公式改写，把表面貌似是 $O(k*n^2)$ 的复杂度降低到 $O(k*n)$ ，其中n是特征数量，k是特征的embedding size，这样就将FM模型改成了和LR类似和特征数量n成线性规模的时间复杂度了，这点非常好。

那么，如何改写原始的FM数学公式，让其复杂度降下来呢？因为原始论文在推导的时候没有给出详细说明，我相信不少人看完估计有点懵，所以这里简单解释下推导过程，数学公式帕金森病患者可以直接跳过下面内容往后看，这并不影响你理解本文的主旨。

FM公式是怎么改写的？概览



上图展示了整个推导过程，我相信如果数学基础不太扎实的同学看着会有点头疼，转换包括四个步骤，下面分步骤解释下。

FM公式是怎么改写的？Step by Step

$$\sum_{i=1}^{n}\sum_{j=i+1}^{n}\langle \mathbf{v}_i, \mathbf{v}_j\rangle x_i x_j$$

$$=\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\langle \mathbf{v}_i, \mathbf{v}_j\rangle x_i x_j - \frac{1}{2}\sum_{i=1}^{n}\langle \mathbf{v}_i, \mathbf{v}_i\rangle x_i x_i$$

原先<Vi,Vi>是不算的，现在算进来了
所以把这个减掉

j的下标变换后，会多算两个东西：
1.<Vi,Vj>和<Vj，Vi>都算一次，所以多算了1次
2.原先<Vi,Vi>是不算的，现在算进来了

<Vi,Vj>和<Vj，Vi>都算一次，所以多算了1次
所以这里除以2，把多算的那次消掉

Step 1

第一个改写步骤及为何这么改写参考上图，比较直观，不解释了；

FM公式是怎么改写的？Step by Step

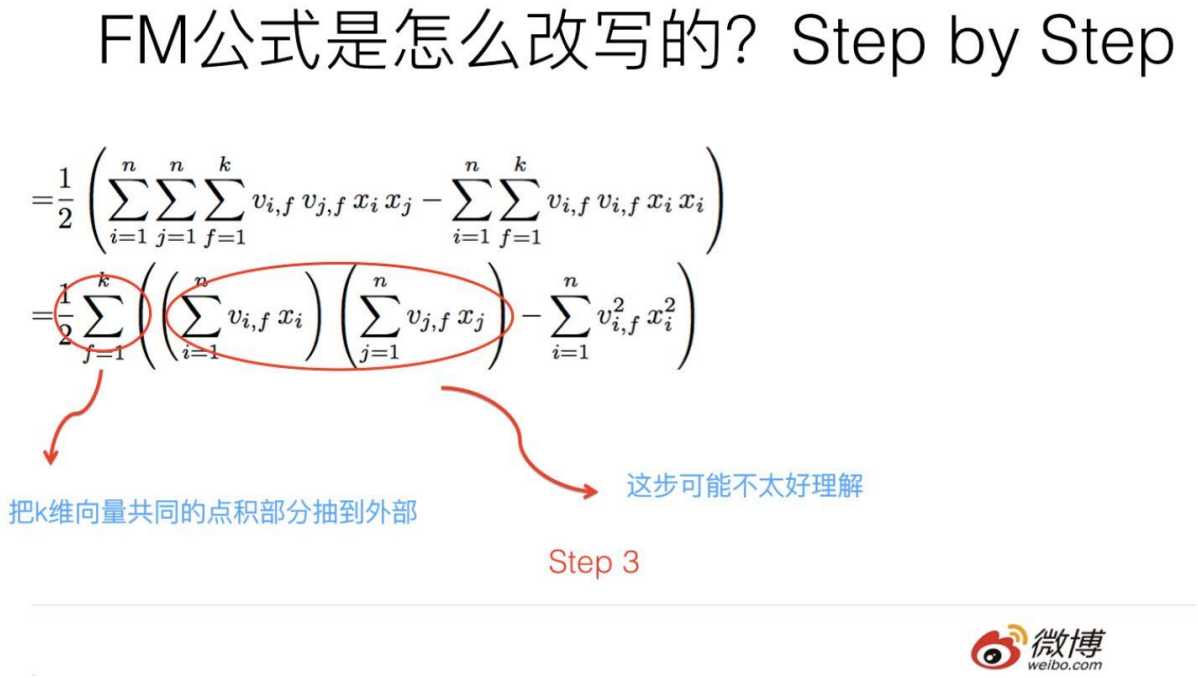
$$=\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\langle \mathbf{v}_i, \mathbf{v}_j\rangle x_i x_j - \frac{1}{2}\sum_{i=1}^{n}\langle \mathbf{v}_i, \mathbf{v}_i\rangle x_i x_i$$

$$=\frac{1}{2}\left(\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{f=1}^{k} v_{i,f}\, v_{j,f}\, x_i x_j - \sum_{i=1}^{n}\sum_{f=1}^{k} v_{i,f}\, v_{i,f}\, x_i x_i\right)$$

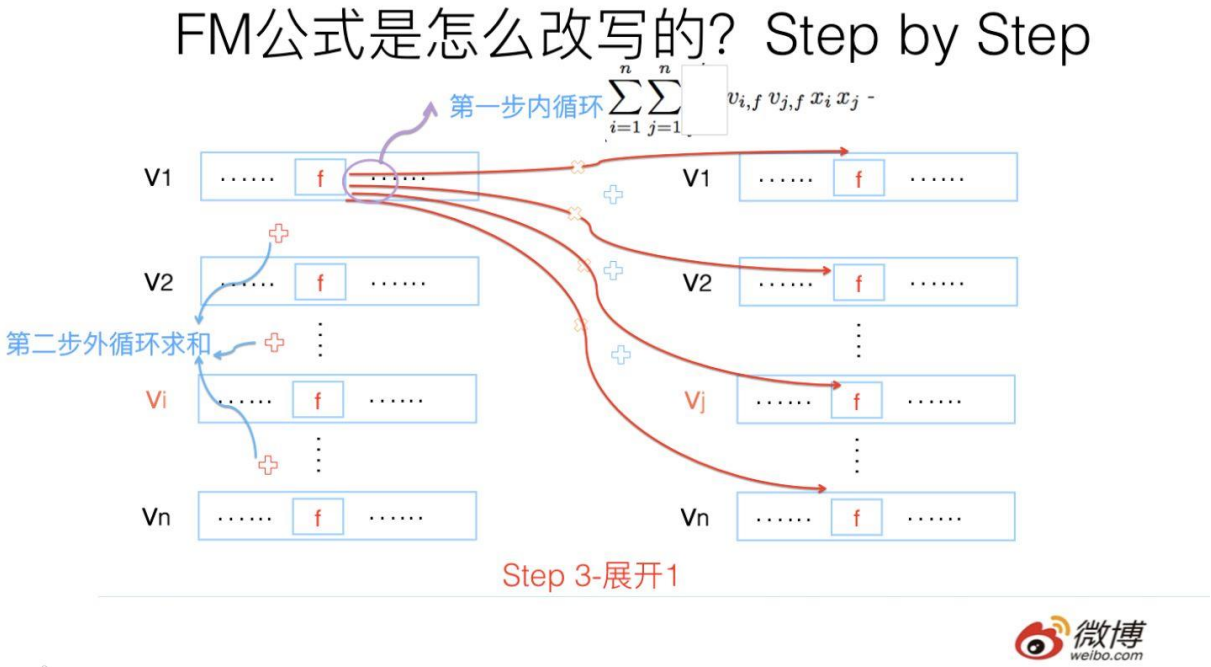
Vi和Vj是k维向量，把点积公式展开而已，这个好理解

Step 2

第二步转换更简单，更不用解释了。

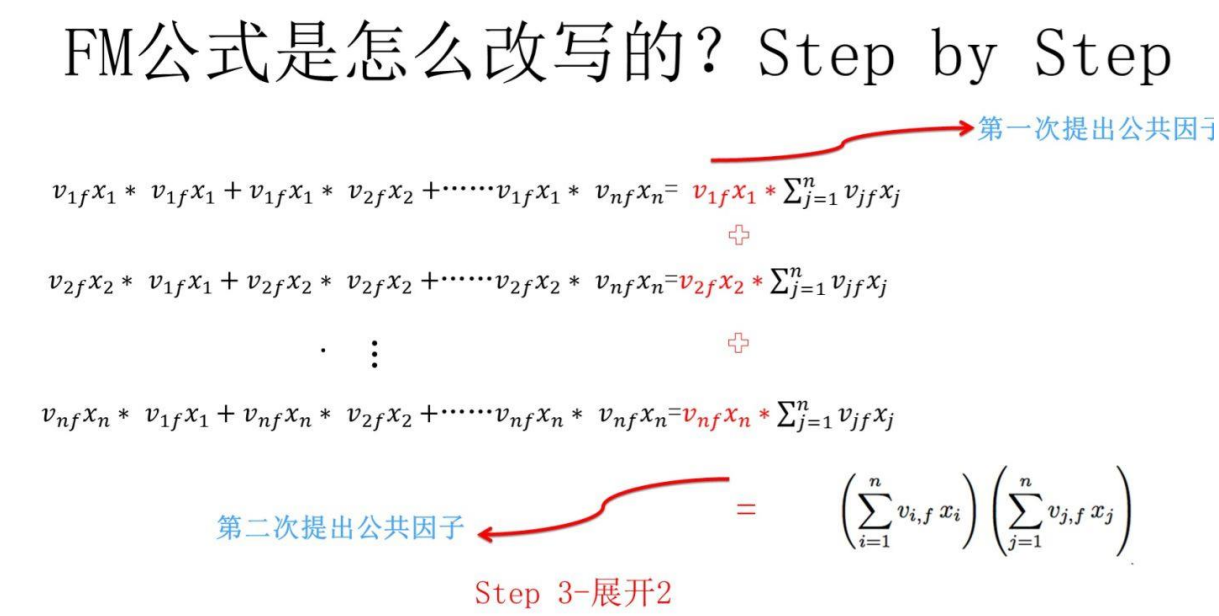


第三步转换不是太直观，可能需要简单推导一下，很多人可能会卡在这一步，所以这里解释解释。

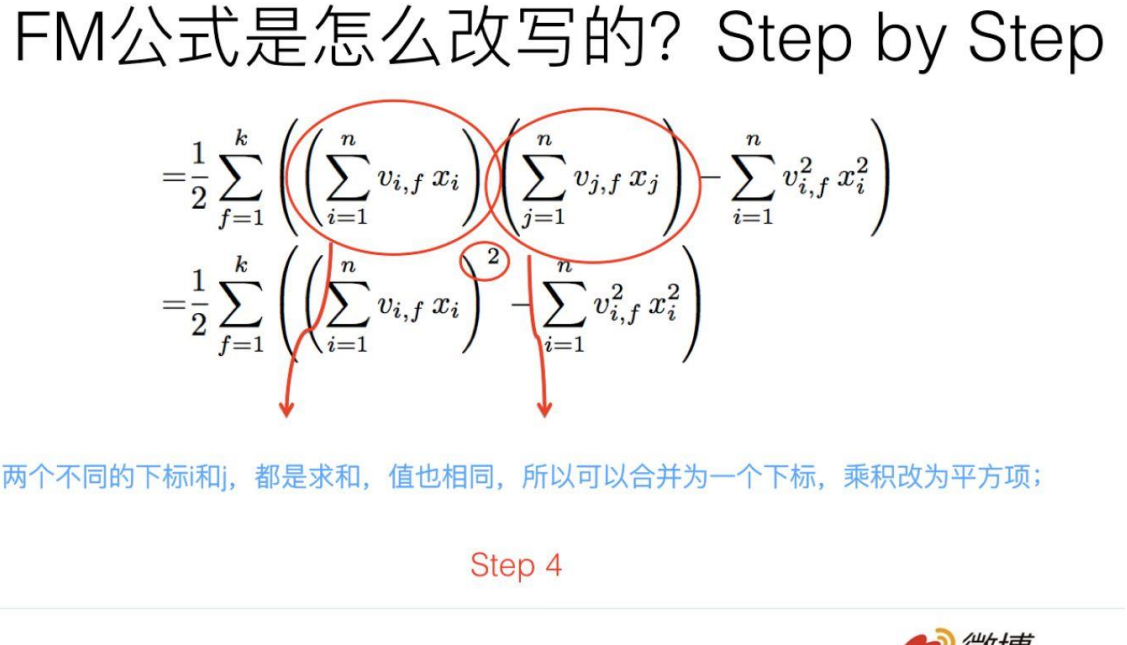


其实吧，如果把 k 维特征向量内积求和公式抽到最外边后，公式就转成了上图这个公式了（不考虑最外边 k 维求和过程的情况下）。它有两层循环，内循环其实就是指定某个特征的第 f 位（这个 f 是由最外层那个 k 指定的）后，和其它任意特征对应向量的第 f 位值相乘求和；而外循环则是遍历每个的第 f 位做循环求和。这样就完成了指定某个

特征位 f 后的特征组合计算过程。最外层的 k 维循环则依此轮循第 f 位，于是就算完了步骤三的特征组合。



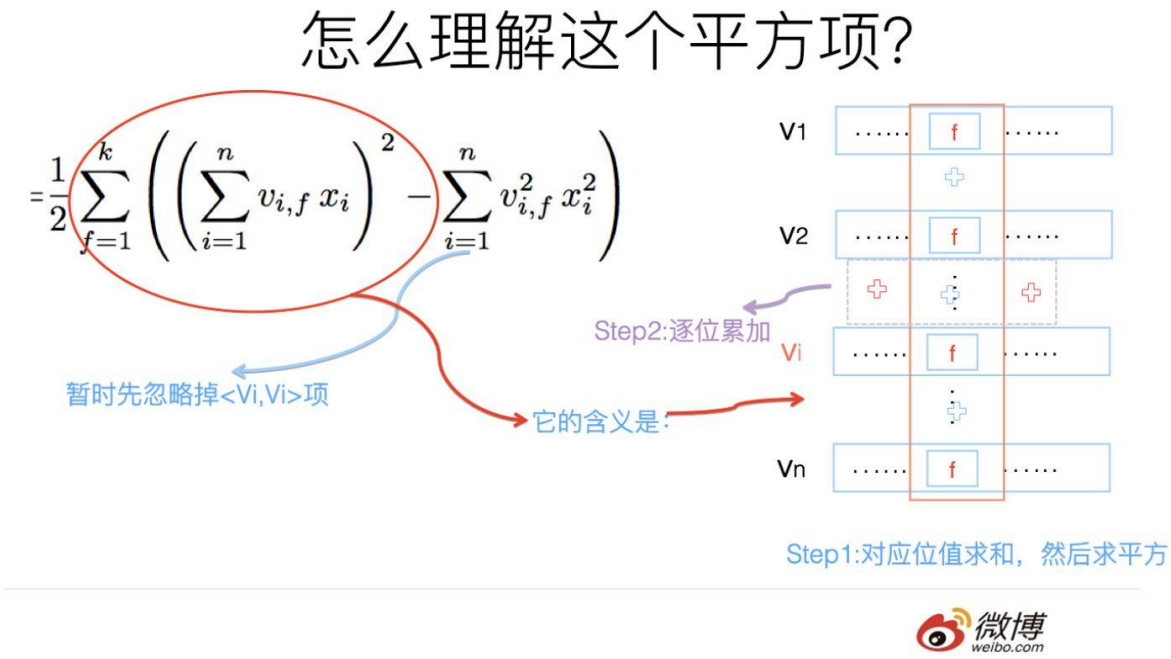
对上一页公式图片展示过程用公式方式，再一次改写（参考上图），其实就是两次提取公共因子而已，这下应该明白了吧？要是还不明白，那您的诊断结果是数学公式帕金森晚期，跟我一个毛病，咱俩病友同病相怜，我也没辙了。



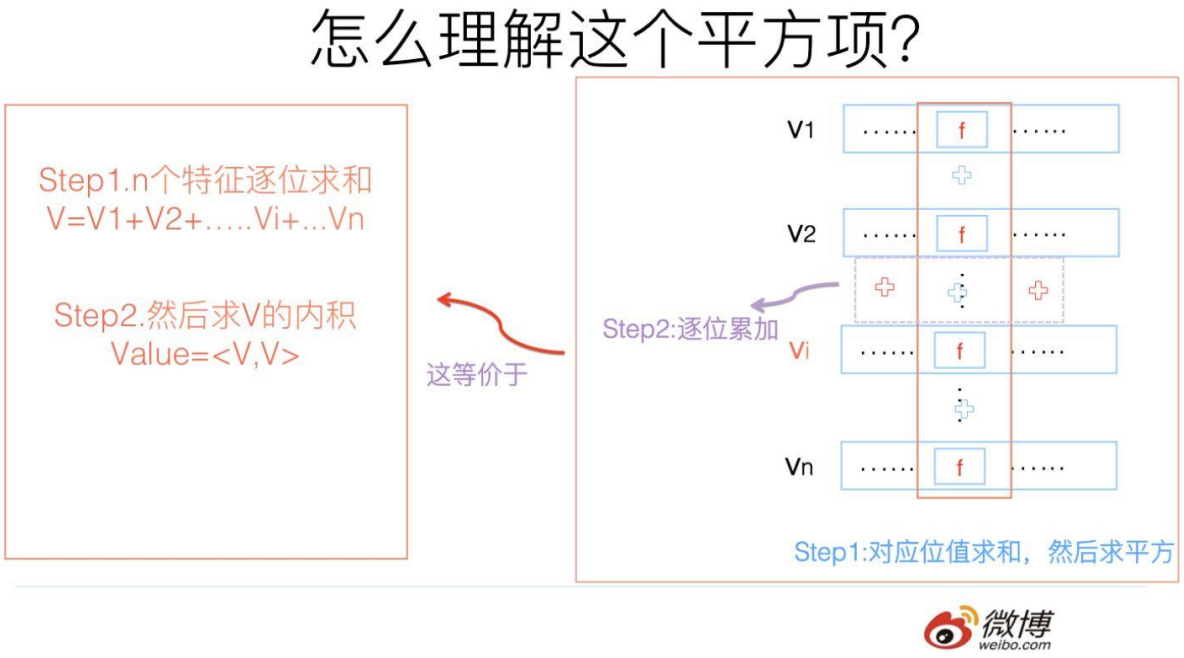
第四步公式变换，意思参考上图，这步也很直白，不解释。

于是，通过上述四步的公式改写，可以看出在实现 FM 模型时，时间复杂度就降低到了 O(k*n) 了，而虽说看上去 n 还有点大，但是其实真实的推荐数据的特征值是极为稀疏的，就是说大量 xi 其实取值是 0，意味着真正需要计算的特征数 n 是远远小于总特征数目 n 的，无疑这会进一步极大加快 FM 的运算效率。

这里需要强调下改写之后的 FM 公式的第一个平方项，怎么理解这个平方项的含义呢？这里其实蕴含了后面要讲的使用 FM 模型统一多路召回的基本思想，所以这里特殊提示一下。



参考上图，你体会下这个计算过程。它其实等价于什么？



这个平方项，它等价于将 FM 的所有特征项的 embedding 向量累加，之后求内积。我再问下之前问过的问题：“我们怎样利用 FM 模型做统一的召回？”这个平方项的含义对你有启发吗？你可以仔细想想它们之间的关联。

上文书提到过，目前工业界推荐系统在召回阶段，大多数采用了多路召回策略，比如典型的召回路有：基于用户兴趣标签的召回；基于协同过滤的召回；基于热点的召回；基于地域的召回；基于 Topic 的召回；基于命名实体的召回等等，除此外还有很多其它类型的召回路。

现在我们来探讨下第一个问题：在召回阶段，能否用一个统一的模型把多路召回招安？就是说改造成利用单个模型，单路召回的模式？具体到这篇文章，就是说能否利用 FM 模型来把多路召回统一起来？

在回答上述问题之前，我估计你会提出疑问：目前大家用多路召回用的好好的，为啥要多此一举，用一个模型把多路召回统一起来呢？这个问题非常好，我们确实应该先看这么做的必要性。

统一召回和多路召回优缺点比较

我们先来说明下统一召回和多路召回各自的优缺点，我觉得使用统一召回模式，相对多路召回有如下优点：

首先，采用多路召回，每一路召回因为采取的策略或者模型不同，所以各自的召回模型得分不可比较，比如利用协同过滤召回找到的候选 Item 得分，与基于兴趣标签这一路召回找到的候选 Item 得分，完全是不可比较的。这也是为何要用第二阶段 Ranking 来将分数统一的原因。而如果采取统一的召回模型，比如 FM 模型，那么不论候选项 Item 来自于哪里，它们在召回阶段的得分是完全可比的。

其次，貌似在目前“召回+Ranking”两阶段推荐模型下，多路召回分数不可比这个问题不是特别大，因为我们可以依靠 Ranking 阶段来让它们可比即可。但是其实多路召回分数不可比会直接引发一个问题：对于每一路召回，我们应该返回多少个 Item 是合适的呢？如果在多路召回模式下，这个问题就很难解决。既然分数不可比，那么每一路召回多少候选项 K 就成为了超参，需要不断调整这个参数上线做 AB 测试，才能找到合适的数值。而如果召回路数特别多，于是每一路召回带有一个超参 K，就是这一路召回多少条候选项，这样的超参组合空间是非常大的。所以到底哪一组超参是最优的，就很难定。其实现实情况中，很多时候这个超参都是拍脑袋上线测试，找到最优的超参组合概率是很低的。

而如果假设我们统一用 FM 模型来做召回，其实就不存在上面这个问题。这样，我们可以在召回阶段做到更好的个性化，比如有的用户喜欢看热门的内容，那么热门内容在召回阶段返回的比例就高，而其它内容返回比例就低。所以，可以认为各路召回的这组超参数就完全依靠 FM 模型调整成个性化的了，很明显这是使用单路单模型做召回的一个特别明显的好处。

再次，对于工业界大型的推荐系统来说，有极大的可能做召回的技术人员和做 Ranking 的技术人员是两拨人。这里隐含着一个潜在可能会发生的问题，比如召回阶段新增了一路召回，但是做 Ranking 的哥们不知道这个事情，在 Ranking 的时候没有把能体现新增召回路特性的特征加到 Ranking 阶段的特征中。这样体现出来的效果是：新增召回路看上去没什么用，因为即使你找回来了，而且用户真的可能点击，但是在排序阶段死活排不上去。也就是说，在召回和排序之间可能存在信息鸿沟的问

题，因为目前召回和排序两者的表达模式差异很大，排序阶段以特征为表达方式，召回则以“路 / 策略 / 具体模型”为表达方式，两者之间差异很大，是比较容易产生上述现象的。

但是如果我们采用 FM 模型来做召回的话，新增一路召回就转化为新增特征的问题，而这一点和 Ranking 阶段在表现形式上是相同的，对于召回和排序两个阶段来说，两者都转化成了新增特征问题，所以两个阶段的改进语言体系统一，就不太容易出现上述现象。

上面三点，是我能想到的采用统一召回模型，相对多路召回的几个好处。但是是不是多路召回一定不如统一召回呢？其实也不是，很明显多路召回这种策略，上线一个新召回方式比较灵活，对线上的召回系统影响很小，因为不同路召回之间没有耦合关系。但是如果采用统一召回，当想新增一种召回方式的时候，表现为新增一种或者几种特征，可能需要完全重新训练一个新的 FM 模型，整个召回系统重新部署上线，灵活性比多路召回要差。

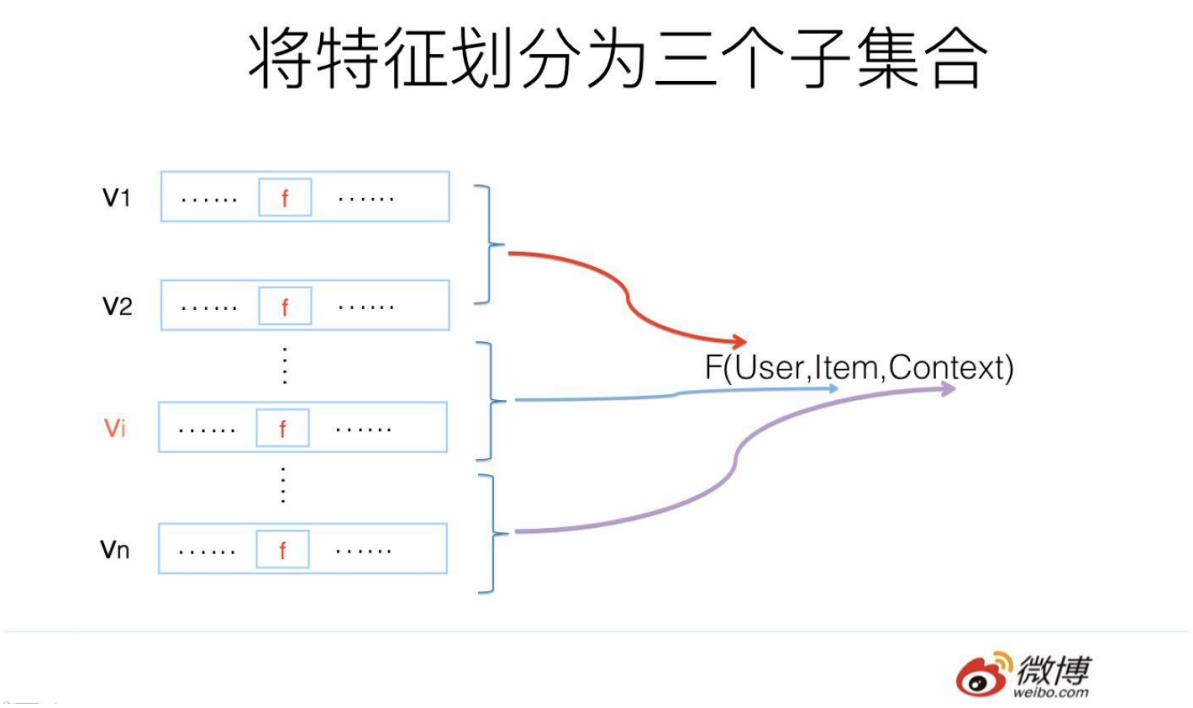
上面讲的是必要性，讲完了必要性，我们下面先探讨如何用 FM 模型做召回，然后再讨论如何把多路召回改造成单路召回，这其实是两个不同的问题。

如何用 FM 模型做召回模型

如果要做一个实用化的统一召回模型，要考虑的因素有很多，比如 Context 上下文特征怎么处理，实时反馈特征怎么加入等。为了能够更清楚地说明，我们先从极简模型说起，然后逐步加入必须应该考虑的元素，最后形成一个实用化的统一召回模型。

不论是简化版本 FM 召回模型，还是复杂版本，首先都需要做如下两件事情：

第一，离线训练。这个过程跟在排序阶段采用 FM 模型的离线训练过程是一样的，比如可以使用线上收集到的用户点击数据来作为训练数据，线下训练一个完整的 FM 模型。在召回阶段，我们想要的其实是：每个特征和这个特征对应的训练好的 embedding 向量。这个可以存好待用。



第二，如果将推荐系统做个很高层级的抽象的话，可以表达成学习如下形式的映射函数：

y=F(User,Item,Context)

意思是，我们利用用户（User）相关的特征，物品(Item)相关的特征，以及上下文特征（Context,比如何时何地用的什么牌子手机登陆等等）学习一个映射函数 F。学好这个函数后，当以后新碰到一个 Item，我们把用户特征，物品特征以及用户碰到这个物品时的上下文特征输入 F 函数，F 函数会告诉我们用户是否对这个物品感兴趣。如果他感兴趣，就可以把这个 Item 作为推荐结果推送给用户。

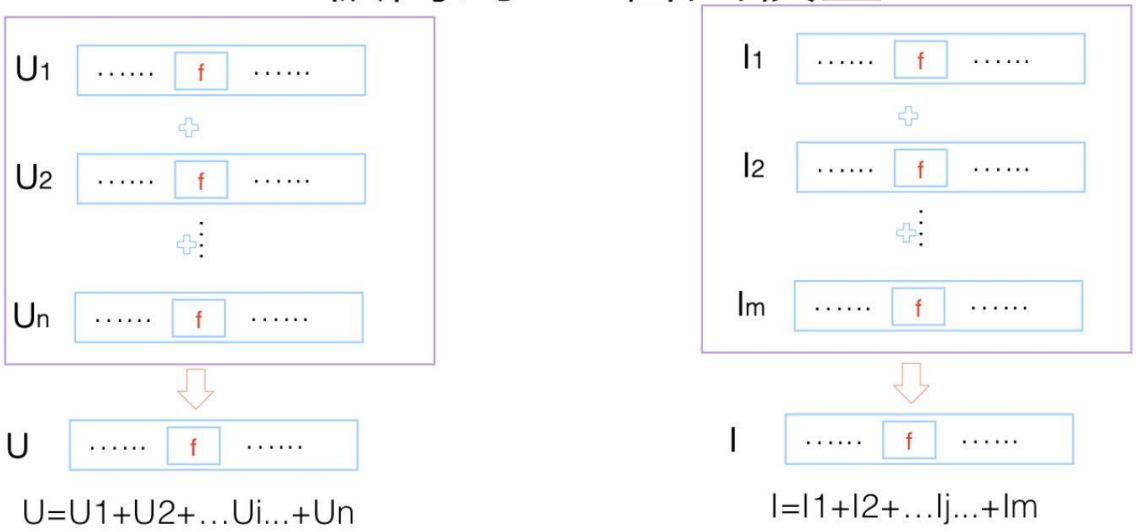
说了这么多，第二个我们需要做的事情是：把特征划分为三个子集合，用户相关特征集合，物品相关特征集合以及上下文相关的特征集合。而用户历史行为类特征，比如用户过去点击物品的特征，可以当作描述用户兴趣的特征，放入用户相关特征集合内。至于为何要这么划分，后面会讲。

做完上述两项基础工作，我们可以试着用 FM 模型来做召回了。

- **极简版 FM 召回模型**

我们先来构建一个极简的 FM 召回模型，首先，我们先不考虑上下文特征，晚点再说。

极简的FM召回模型



U=U1+U2+…Ui…+Un

I=I1+I2+…Ij…+Im

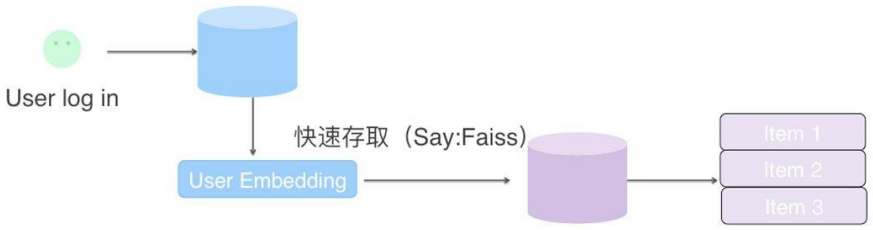
第一步，对于某个用户，我们可以把属于这个用户子集合的特征，查询离线训练好的 FM 模型对应的特征 embedding 向量，然后将 n 个用户子集合的特征 embedding 向量累加，形成用户兴趣向量 U，这个向量维度和每个特征的维度是相同的。

类似的，我们也可以把每个物品，其对应的物品子集合的特征，查询离线训练好的 FM 模型对应的特征 embedding 向量，然后将 m 个物品子集合的特征 embedding 向量累加，形成物品向量 I，这个向量维度和每个特征的维度也是是相同的。

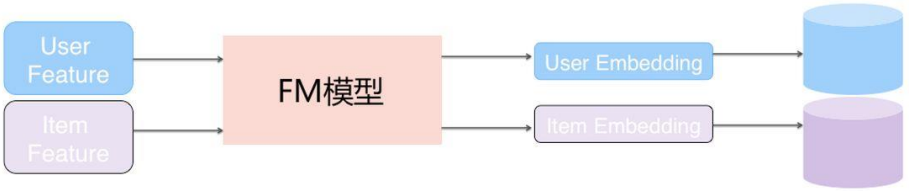
对于极简版 FM 召回模型来说，用户兴趣向量 U 可以离线算好，然后更新线上的对应内容；物品兴趣向量 I 可以类似离线计算或者近在线计算，问题都不大。

极简的FM召回模型

在线执行：



离线：



第二步，对于每个用户以及每个物品，我们可以利用步骤一中的方法，将每个用户的兴趣向量离线算好，存入在线数据库中比如 Redis（用户 ID 及其对应的 embedding），把物品的向量逐一离线算好，存入 Faiss(Facebook 开源的 embedding 高效匹配库)数据库中。

当用户登陆或者刷新页面时，可以根据用户 ID 取出其对应的兴趣向量 embedding，然后和 Faiss 中存储的物料 embedding 做内积计算，按照得分由高到低返回得分 Top K 的物料作为召回结果。提交给第二阶段的排序模型进行进一步的排序。这里 Faiss 的查询速度至关重要，至于这点，后面我们会单独说明。

这样就完成了一个极简版本 FM 召回模型。但是这个版本的 FM 召回模型存在两个问题。

问题一：首先我们需要问自己，这种累加用户 embedding 特征向量以及累加物品 embedding 特征向量，之后做向量内积。这种算法符合 FM 模型的原则吗？和常规的 FM 模型是否等价？

我们来分析一下。这种做法其实是在做用户特征集合 U 和物品特征集合 I 之间两两特征组合，是符合 FM 的特征组合原则的，考虑下列公式是否等价就可以明白了：

$\langle \sum_i U_i, \sum_j I_j \rangle$ (公式1)

$\sum_i \sum_j \langle U_i, I_j \rangle$ （公式2）

其实两者是等价的，建议您可以推导一下（这其实不就是上面在介绍FM公式改写的第三步转换吗？当然，跟完全版本的FM比，我们没有考虑U和I特征集合内部任意两个特征的组合，等会会说这个问题）。

也可以这么思考问题：在上文我们说过，FM为了提升计算效率，对公式进行了改写，改写后的高效计算公式的第一个平方项其实等价于：把所有特征embedding向量逐位累加成一个求和向量V，然后自己和自己做个内积操作<V,V>。这样等价于根据FM的原则计算了任意两个特征的二阶特征组合了。而上面描述的方法，和标准的FM的做法其实是一样的，区别无非是将特征集合划分为两个子集合U和I，分别代表用户相关特征及物品相关特征。而上述做法其实等价于在用户特征和物品特征之间做两两特征组合，只是少了U内部之间特征，及I内部特征之间的特征组合而已。一般而言，其实我们不需要做U内部特征之间以及I内部特征之间的特征组合，对最终效果影响很小。于是，沿着这个思考路径，我们也可以推导出上述做法基本和FM标准计算过程是等价的。

第二个问题是：这个版本FM是个简化版本模型，因为它没考虑场景上下文特征，那么如果再将上下文特征引入，此时应该怎么做呢？

- **加入场景上下文特征**

上面叙述了如何根据 FM 模型做一个极简版本的召回模型，之所以说极简，因为我们上面说过，抽象的推荐系统除了用户特征及物品特征外，还有一类重要特征，就是用户发生行为的场景上下文特征（比如什么时间在什么地方用的什么设备在刷新），而上面版本的召回模型并没有考虑这一块。

之所以把上下文特征单独拎出来，是因为它有自己的特点，有些上下文特征是近乎实时变化的，比如刷新微博的时间，再比如对于美团嘀嘀这种对地理位置特别敏感的应用，用户所处的地点可能随时也在变化，而这种变化在召回阶段就需要体现出来。所以，上下文特征是不太可能像用户特征离线算好存起来直接使用的，而是用户在每一次刷新可能都需要重新捕获当前的特征值。动态性强是它的特点。

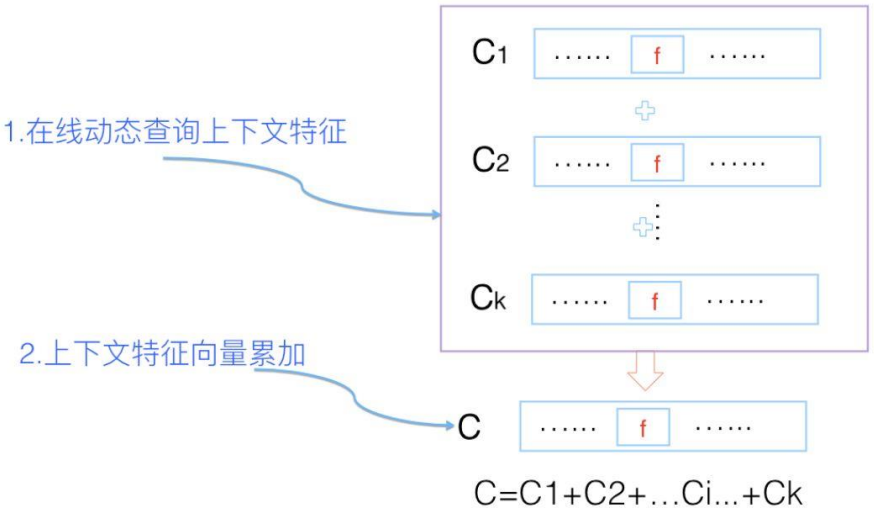
而考虑进来上下文特征，如果我们希望构造和标准的 FM 等价的召回模型，就需要多考虑两个问题：

问题一：既然部分上下文特征可能是实时变化的，无法离线算好，那么怎么融入上文所述的召回计算框架里？

问题二：我们需要考虑上下文特征 C 和用户特征 U 之间的特征组合，也需要考虑 C 和物品特征 I 之间的特征组合。上下文特征有时是非常强的特征。那么，如何做能够将这两对特征组合考虑进来呢？

我们可以这么做：

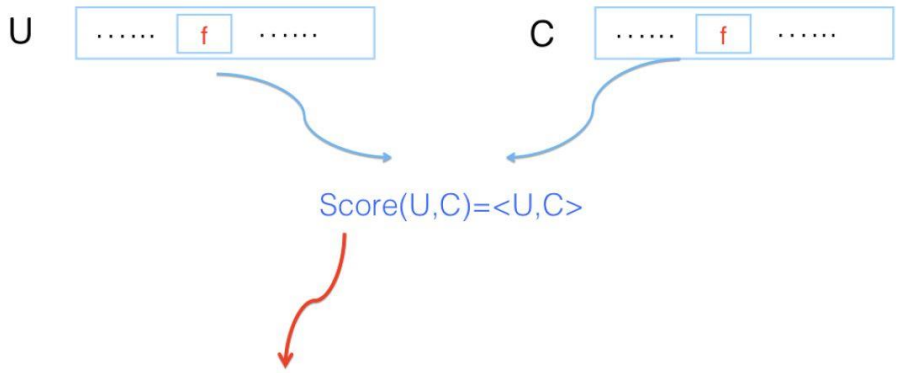
融入动态场景特征的召回模型-Step1



C=C1+C2+…Ci…+Ck

首先，由于上下文特征的动态性，所以给定用户 UID 后，可以在线查询某个上下文特征对应的 embedding 向量，然后所有上下文向量求和得到综合的上下文向量 C。这个过程其实和 U 及 I 的累加过程是一样的，区别无非是上下文特征需要在线实时计算。而一般而言，场景上下文特征数都不多，所以在线计算，速度方面应可接受。

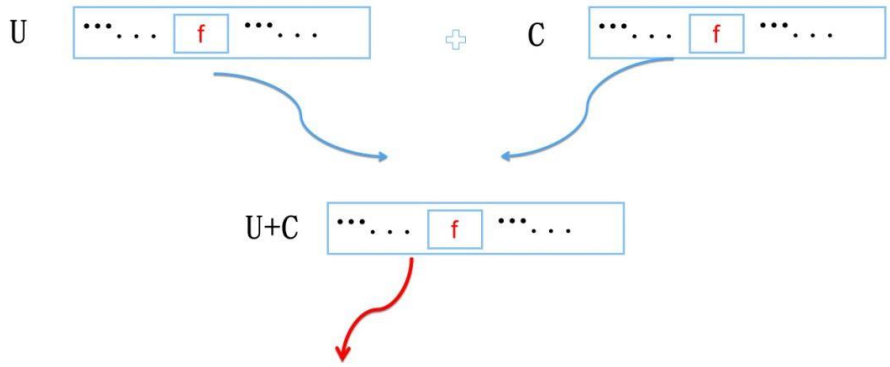
融入动态场景特征的召回模型-Step2



计算U和C的内积，代表用户特征和上下文特征的特征组合分数

然后，将在线算好的上下文向量 C 和这个用户的事先算好存起来的用户兴趣向量 U 进行内积计算 Score=<U,C>。这个数值代表用户特征和上下文特征的二阶特征组合得分，算好备用。至于为何这个得分能够代表 FM 中的两者（U 和 C）的特征组合，其实道理和上面讲的 U 和 I 做特征组合道理是一样的。

融入动态场景特征的召回模型-Step3



再然后，将 U 和 C 向量累加求和，利用（U+C）去 Faiss 通过内积方式取出 Top K 物品，这个过程和极简版是一样的，无非查询向量由 U 换成了（U+C）。通过这种方式取出的物品同时考虑到了用户和物品的特征组合<U,I>，以及上下文和物品的特征组合<C,I>。道理和之前讲的内容是类似的。

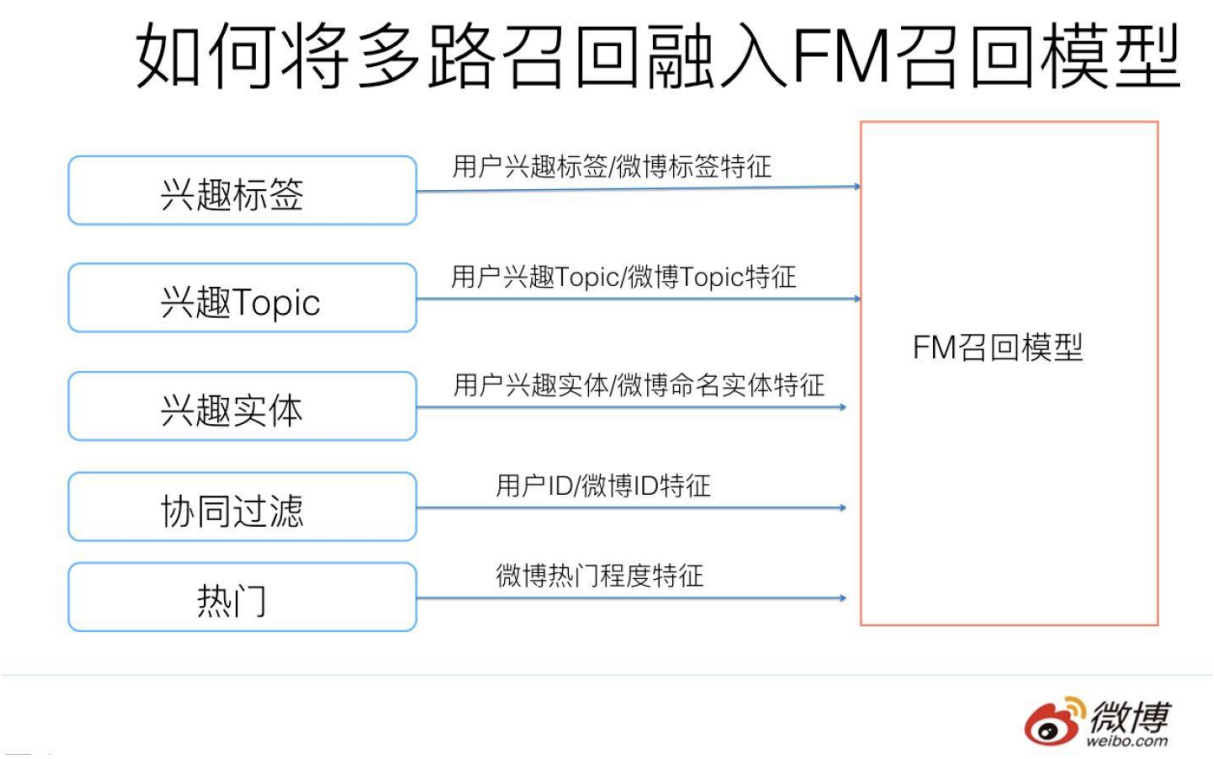
假设返回的 Top K 物品都带有内积的得分 Score1，再考虑上一步<U,C>的得分 Score，将两者相加对物品重排序（<U,C>因为跟物品无关，所以其实不影响物品排序，但是会影响最终得分，FM 最外边的 Sigmoid 输出可能会因为加入这个得分而发生变化），就得到了最终结果，而这个最终结果考虑了 U/I/C 两两之间的特征组合。

于是我们通过这种手段，构造出了一个完整的 FM 召回模型。这个召回模型通过构造 user embedding，Context embedding 和 Item embedding，以及充分利用类似 Faiss 这种高效 embedding 计算框架，就构造了高效执行的和 FM 计算等价的召回系统。

如何将多路召回融入 FM 召回模型

上文所述是如何利用 FM 模型来做召回，下面我们讨论下如何将多路召回统一到 FM 召回模型里来。

我们以目前不同类型推荐系统中共性的一些召回策略来说明这个问题，以信息流推荐为例子，传统的多路召回阶段通常包含以下策略：协同过滤，兴趣分类，兴趣标签，兴趣 Topic，兴趣实体，热门物品，相同地域等。这些不同角度的召回策略都是较为常见的。



我们再将上述不同的召回路分为两大类，可以把协同过滤作为一类，其它的作为一类，协同过滤相对复杂，我们先说下其它类别。

对于比如兴趣分类，兴趣标签，热门，地域等召回策略，要把这些召回渠道统一到 FM 模型相对直观，只需要在训练 FM 模型的时候，针对每一路的特性，在用户特征端和

物品特征端新增对应特征即可。比如对于地域策略，我们可以把物品所属地域（比如微博所提到的地域）和用户的感兴趣地域都作为特征加入 FM 模型即可。兴趣标签，Topic，兴趣实体等都是类似的。所以大多数情况下，在多路召回模式下你加入新的一路召回，在 FM 统一召回策略下，对应地转化成了新增特征的方式。

然后我们再说协同过滤这路召回。其实本质上也是将一路召回转化为新加特征的模式。我们上文在介绍 FM 模型和 MF 模型关系的时候提到过：本质上 MF 模型这种典型的协同过滤策略，是 FM 模型的一个特例，可以看作在 FM 模型里只有 User ID 和 Item ID 这两类（Fields）特征的情形。意思是说，如果我们将 user ID 和 Item ID 作为特征放入 FM 模型中进行训练，那么 FM 模型本身就是包含了协同过滤的思想的。当然，对于超大规模的网站，用户以亿计，物品可能也在千万级别，如果直接把 ID 引入特征可能会面临一些工程效率问题以及数据稀疏的问题。对于这个问题，我们可以采取类似在排序阶段引入 ID 时的 ID 哈希等降维技巧来进行解决。

所以综合来看，在多路召回下的每一路召回策略，绝大多数情况下，可以在 FM 召回模型模式中转化为新增特征的方式。

在具体实施的时候，可以沿着这个路径逐步替换线上的多路召回：先用 FM 模型替换一路召回，线上替换掉；再新加入某路特征，这样上线，就替换掉了两路召回；如此往复逐渐把每一路召回统一到一个模型里。这是比较稳的一种替换方案。当然如果你是个猛人，直接用完整的 FM 召回模型一步替换掉线上的各路召回，也，未尝不可。只要小流量 AB 测试做好也没啥。

FM 模型能否将召回和排序阶段一体化

前文有述，之所以目前常见的工业推荐系统会分为召回排序两个阶段，是因为这两个阶段各司其职，职责分明。召回主要考虑泛化性并把候选物品集合数量降下来；排序则主要负责根据用户特征 / 物品特征 / 上下文特征对物品进行精准排名。

那么，我们现在可以来审视下本文开头提出的第二个问题了：FM 模型能否将常见的两阶段模型一体化？即是否能将实用化的推荐系统通过 FM 召回模型简化为单阶段模型？意思是推荐系统是否能够只保留 FM 召回这个模块，扔掉后续的排序阶段，FM 召回按照得分排序直接作为推荐结果返回。我们可以这么做吗？

这取决于 FM 召回模型是否能够一并把原先两阶段模型的两个职责都能承担下来。这句话的意思是说，FM 召回模型如果直接输出推荐结果，那么它的速度是否足够快？另外，它的精准程度是否可以跟两阶段模型相媲美？不会因为少了第二阶段的专门排序环节，而导致推荐效果变差？如果上面两个问题的答案都是肯定的，那么很明显 FM 模型就能够将现有的两阶段推荐过程一体化。

我们分头来分析这个问题的答案：准确性和速度。先从推荐精准度来说明，因为如果精准度没有办法维持，那么速度再快也没什么意义。

所以现在的第一个子问题是：FM 召回模型推荐结果的质量，是否能够和召回+排序两阶段模式接近？

我们假设一个是 FM 统一召回模型直接输出排序结果；而对比模型是目前常见的多路召回+FM 模型排序的配置。从上文分析可以看出，尽管 FM 召回模型为了速度够快，做了一些模型的变形，但是如果对比的两阶段模型中的排序阶段也采取 FM 模型的话，我们很容易推理得到如下结论：如果 FM 召回模型采用的特征和两阶段模型的 FM 排序模型采用相同的特征，那么两者的推荐效果是等价的。这意味着：只要目前的多路召回都能通过转化为特征的方式加入 FM 召回模型，而且 FM 排序阶段采用的特征在 FM 召回模型都采用。那么两者推荐效果是类似的。这意味着，从理论上说，是可以把两阶段模型简化为一阶段模型的。

既然推理的结论是推荐效果可以保证，那么我们再来看第二个问题：只用 FM 召回模型做推荐，速度是否足够快？

我们假设召回阶段 FM 模型对 User embedding 和 Item embedding 的匹配过程采用 Facebook 的 Faiss 系统，其速度快慢与两个因素有关系：

1. 物品库中存储的 Item 数量多少，Item 数量越多越慢；

2. embedding 大小，embedding size 越大，速度越慢；

微博机器学习团队 18 年将 Faiss 改造成了分布式版本，并在业务易用性方面增加了些新功能，之前我们测试的查询效率是：假设物品库中存储 100 万条微博 embedding 数据，而 embedding size=300 的时候，TPS 在 600 左右，平均每次查询小于 13 毫秒。而当库中微博数量增长到 200 万条，embedding size=300 的时候，TPS 在 400

左右，平均查询时间小于 20 毫秒。这意味着如果是百万量级的物品库，embedding size 在百级别，一般而言，通过 Faiss 做 embedding 召回速度是足够实用化的。如果物品库大至千万量级，理论上可以通过增加 Faiss 的并行性，以及减少 embedding size 来获得可以接受的召回速度。

当然，上面测试的是纯粹的 Faiss 查询速度，而事实上，我们需要在合并用户特征 embedding 的时候，查询用户特征对应的 embedding 数据，而这块问题也不太大，因为绝大多数用户特征是静态的，可以线下合并进入用户 embedding，Context 特征和实时特征需要线上在线查询对应的 embedding，而这些特征数量占比不算太大，所以速度应该不会被拖得太慢。

综上所述，FM 召回模型从理论分析角度，其无论在实用速度方面，还是推荐效果方面，应该能够承载目前“多路召回+FM 排序”两阶段推荐模式的速度及效果两方面功能，所以推论它是可以将推荐系统改造成单模型单阶段模式的。

当然，上面都是分析结果，并非实测，所以不能确定实际应用起来也能达到上述理论分析的效果。

总结

最后我简单总结一下，目前看貌似利用 FM 模型可以做下面两个事情：

首先，我们可以利用 FM 模型将传统的多路召回策略，改为单模型单召回的策略，传

统的新增一路召回，可以转换为给 FM 召回模型新增特征的方式；

其次，理论上，我们貌似可以用一个 FM 召回模型，来做掉传统的“多路召回+排序”的两项工作，可行的原因上文有分析。

这是本文的主要内容，谢谢观看。本文开头说过，关于统一召回模型，我打算写一个四篇系列，后面会逐步介绍其它三类模型，它们是谁呢？它们可以用来做统一的召回模型吗？如果能，怎么做？如果不能，又是为什么？它们可以替代掉两阶段推荐模型，一步到位做推荐吗？这些问题的答案会是什么呢？.....

关于上面这一系列《走近科学》风格的问题，因为我第一遍写完的时间比较早，过了好一阵子翻开来看，当读到上面这些问题的时候，自己都把自己吸引住了，对着问题思考了半天。这说明什么？说明这些问题真的很有吸引力是吗？其实不是，这只能说明过去时间太长了，连我自己都记不清这些问题的答案是什么了，哈哈。

我其实是个直率的人，要不，还是先主动剧透掉下篇文章的标题吧：

“连女神级程序媛美女看了都震惊！FFM 模型居然能够做这么大规模推荐系统的召回！”背景音乐准备配置：房东的猫之《云烟成雨》.....

那么，女神级程序员到底是谁？她到底有多神？她到底有多美？她到底是不是女性？预知详情，请听下回分解。

致谢：感谢过去一段时间内，陆续和我讨论统一多路召回模型及一体化推荐模型的微博机器学习团队的雪逸龙，黄通文，佘青云，张童，邸海波等同学，以及冯凯同学提供的 Faiss 性能测试数据。也许你们并不想让自己的名字出现在这种风格的文章后边，但是其实仔细想想，能看到这里的人估计少得可怜，所以别介意想开点，哈哈。

51、有用过 FFM 和 DeepFFM 没，说说使用经验

本文是我 18 年 12 月在 AICon（全球人工智能与机器学习大会）的讲稿，感谢极客邦的同学，几乎一字不落地把演讲内容转成了文字，我在这个基础上简单修正了一些内容。/

大家好，先简单自我介绍一下，我叫张俊林，目前在微博机器学习团队负责 AI Lab 的日常工作，主要是推动一些业界新技术应用到业务里去。

今天我们主要介绍一下，FFM 模型和它的深度学习模型版本，以及我们在这个基础上改造的两个模型。这页 PPT 是我要跟大家分享的一个提纲。

首先，我会给大家介绍一下大规模推荐系统的整个流程框架和其中一些比较核心的技术点。

第二，我们知道，做推荐系统的 Rank 模型会面临一个选择：是用传统模型还是用目前比较火的深度学习模型。所以我会简单地介绍一下这两大类模型的发展历程，包括它们各自的特点。

第三，传统的线性模型，我们聚焦在 FFM 模型上。FFM 模型有它的优点，也有它的问题，所以针对它的问题，我会介绍一个我们提出的一个改进的版本，我把它叫做“双线性 FFM 模型”。

另外，我也会大致介绍一下，典型的深度学习模型有哪些，他们各自的特点是什么、使用场景是什么。

最后，我会介绍一下 DeepFFM，这可以认为是我们针对 FFM 做的神经网络版本的进一步改进版本，至于它是怎么做的，效果怎么样，我在后面会重点介绍。

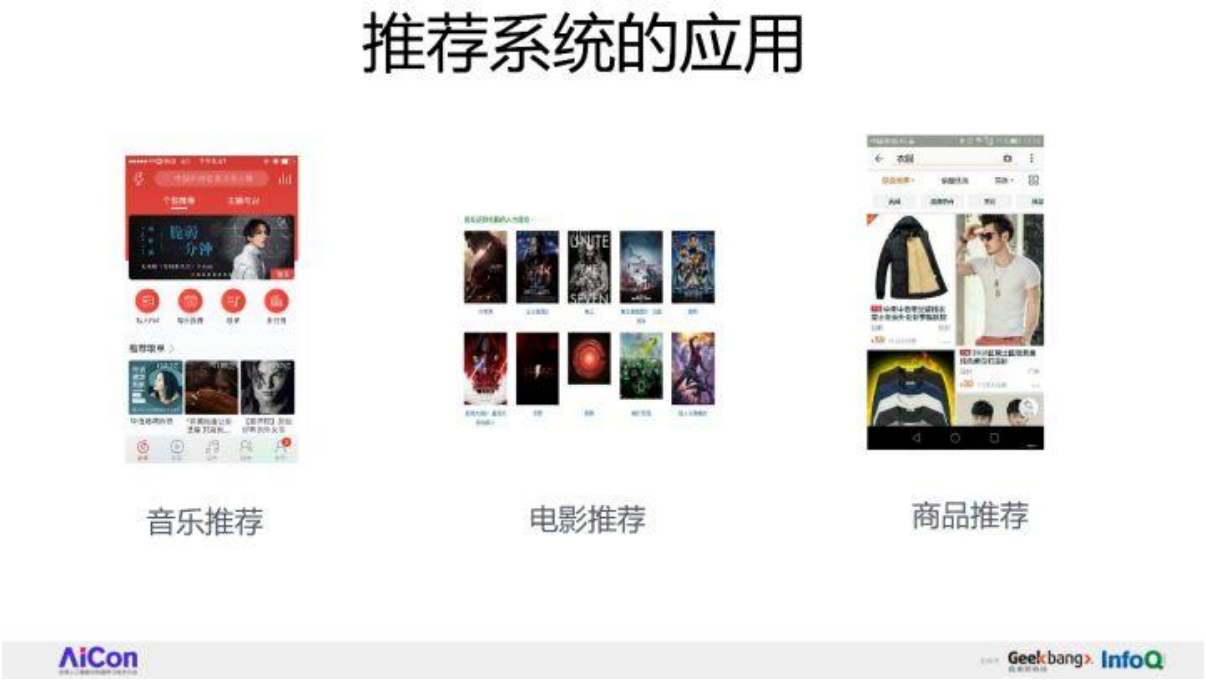
大规模推荐系统介绍

首先进入第一部分，大规模推荐系统的介绍。

推荐系统大家实际都非常熟悉了，因为大家现在都用手机 App，我觉得只要是

上网的话，70%的时间你一定会受到推荐系统的影响，只不过你可能没有意识到它的存在。

举几个典型的例子：网易云音乐的音乐推荐，我个人觉得网易音乐的推荐做得是相当不错的；此外还有豆瓣的电影推荐、阿里商品推荐等等，我这里只列了几个比较典型的例子，主要是大家可能平常比较关注的一些场景。



以微博为例：常见推荐系统场景

因为我来自于微博，所以我简单地介绍一下，在微博有哪些场景是应用推荐、CTR 以及排序任务的。

典型的有三个场景。你如果刷微博就会看到，第一个就是关系流 Feed 排序。最早的关注流 Feed 排序是按时间排序的，但后来因为国外的 Facebook、Twitter 都陆续上了机器学习的 ranking 机制，所以说现在微博的关系流实际是加了机器学习排序的。可是时间因素在其中还是起着很关键的作用，这种场景对于机器学习模型虽然有使用，但是有限制，限制就是排序对时间权重的依赖还是比较高。

另外一个是热门流。这个是比较典型的、完全个性化的一个流，因为它给用户的推荐是不需要用户关注的，只要认为通过用户过去的点击行为，判断其可能对什么感兴趣，就会给用户推荐，这是一个完全个性化的信息流。

此外，还有正文页的推荐。用户点开一个微博进去之后，下面会推荐一个用户可能感兴趣的微博，这是一个附属的推荐场景。

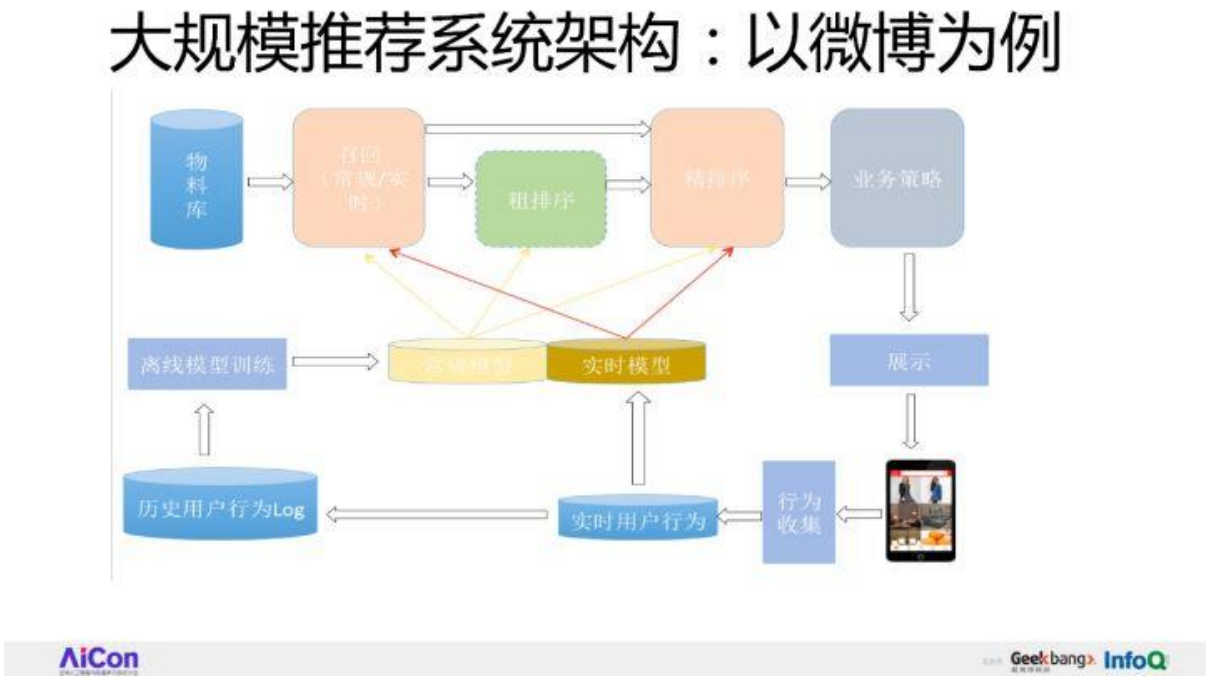
这三个场景在微博上是典型的信息流排序，或者是推荐的使用场景。

以微博为例：大规模推荐系统架构

我们以微博的推荐系统为例，来简单介绍一下：现在工业级的大规模推荐系统是怎么来做推荐架构和推荐流程的。

尽管我说是以微博为例，但是我可以肯定的说，目前工业界 90%的推荐系统就是这个结构，只不过可能在有些具体算法和实施方式上有些区别，但是架子大都是这个架子。

我简单捋一下这个过程。



首先它分为两大部分，最上面那部分是在线推荐部分，底下部分是离线部分。

先说在线部分。我们习惯把微博叫做物料库，实际就是 item，物料库的规模会比较大，一般会先过召回模块，为什么要走召回？做过推荐的人都很清楚：要给微博的每一个用户在热门流推出个性化的信息流，假设一天新发的微博条数以亿计算，再加上历史的条数，任何一个人登上微博去刷热门流，都需要算几亿条微博给这一个人。如果不增加召回环节，直接部署排序模型，这个计算量是很巨大的，速度上根本算不过来。

召回的目的很简单，就是把个性化的推荐 item 数目降下来。召回等于说把用户可能感兴趣的物料进行一些缩减，缩到一定的数量范围里面，但是还要跟用户兴趣相关。如果如果召回 Item 的数量还是比较多，可以部署一个粗排模块，用简单的排序模型再筛选一下，Item 数量就可以再往下减一减，然后进入精排环节。因为到这一步已经经过两轮筛选，对于某个用户来说，剩下的物料已经不多了。所谓精排的意思就是：在此基础上可以加一些复杂模型，精制地给把少量微博根据用户兴趣排一排序，把用户真正感兴趣的内容排到前面去。

之后还有业务逻辑。比如说要把已读的微博内容过滤掉，需要考虑推荐结果的多样性以及其它各方面的业务逻辑。

我们会捕捉用户行为，举个例子，用户看过哪些微博、反馈过哪些微博、互动过哪些微博，这些行为就被收集起来，对于实时的用户行为，我们一般会部署一个实时的模型，可以实时地更新在线模型，这体现在：召回可以改成实时的模式，包括 ranking 也可以改造成实时的模式。

所谓实时的意思就是：用户刷了一条微博，或者互动了一条微博，立刻就在刷出下一条微博的时候，体现出用户刚才这个行为了。

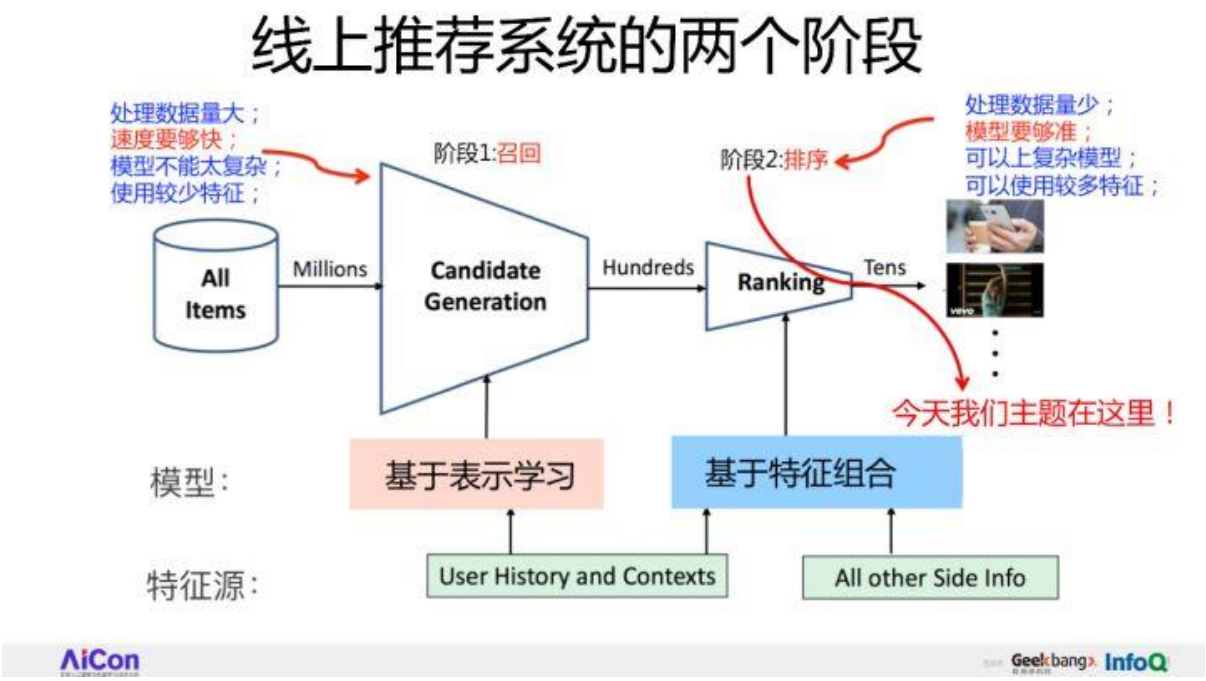
当然还会计算离线模型，因为这个模型它的训练数据更充分、更精准。通过这种方式，我们会定期地更新线上的这三个模型，这就是典型的工业界做大规模推荐系统的整体流程。

正如我上面所讲的，这个框架是大多数做推荐的公司的推荐业务基本框架，只不过可能几个关键环节使用的具体技术方法不同而已。

线上推荐系统的两个阶段

现在我们来细致地看一下在线部分。

我刚才说了三个过程：召回、粗排、精排，但是最常见的还是两个阶段的，就这张图展示的。



第一阶段，召回，第二阶段，排序。我再重复一下：召回是用来做什么的，它的特点是什么，ranking 在干什么，我觉得搞推荐系统，先要把这些事搞明白。

召回阶段

我刚刚也提到了，首先是因为面临的侯选数据集非常大，而最根本的要求是速度快，因为要求速度快，所以就不能部署太复杂的模型。另外要使用少量的特征，这是召回阶段的特性。召回阶段要掌握一点：怎么快怎么来，但是也要兼顾用户兴趣。简单来说，召回会把大量的物料减到几百上千的量级，然后扔给后面的 ranking 阶段。

排序阶段

它和召回阶段的特性完全不一样，ranking 阶段只有一点需要记住：模型要够准，这是它的根本。此外，因为这一阶段处理的数据量比较少了，所以就可以部署复杂模型，就可以使用我能想到的任意有用的特征，但归根结底是为了一件事：怎么排得准。

排序模型：工业界算法的演进路线

CTR 模型、推荐模型发展的历史很久了，举个例子，百度部署广告系统，同时也部署了 CTR 的预估系统，到目前为止应该有十几到二十年的经验了。那么我们要归纳一下，公司做推荐的话，模型是按照一个什么样的轨迹发展的。下面 PPT 展示了这个发展过程。

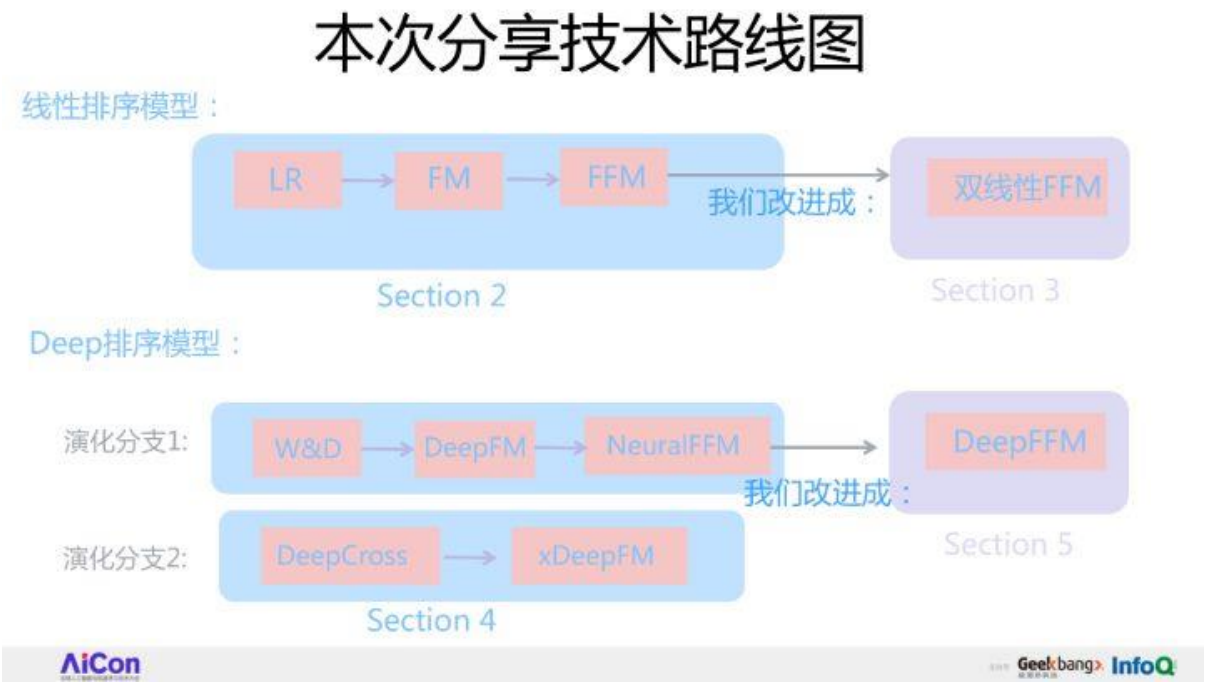
最早的是规则。什么叫规则？这个规则可能和你想的规则不太一样，比如说给用户推荐最热门的内容，这是一种规则，此外还可以添加很多其他的规则。规则的好处是什么？特别简单，如果想通过一个规则去做推荐，三天就能上线，效果也不会特别差，训练速度快，而且还可能不需要有监督地去训练。但是如果后来的规则越来越多的话，问题就出现了，它们会相互冲突，系统的综合效果，很难往上提升，因为对于系统来说，很难有个明确的优化目标，这是问题所在。

在早期的规则推荐之后，业内一般会用 LR，也就是逻辑回归。LR 之后，一般就是 LR 加 GBDT。所有的 CTR 模型，它的核心就是有效特征的选择，以及有效的特征组合的发现和利用。所以，怎么有效解决特征组合的问题，是个引领技术发展的纲领，CTR 或者排序模型的发展路径就按照这个方向发展。

LR 的特性是可以人工做特征组合，但是人工做特征组合有个问题：需要投入相当大的人力才可以做好。那么 GBDT 相对 LR 来比的话，有什么好处？GBDT 可以**半自动化**地做一些特征组合，于是 LR 后面大家就用 LR+GBDT 模型，能够半自动地做特征组合了，不完全依赖人工。

再往后发展就是 FM。FM 跟 LR+GBDT 区别又是什么？它可以**全自动化**地做特征组合。那么从特征组合的角度讲，又有什么新的特点呢？很简单，我们用 FM 的时候，因为一般因为计算量的问题，只做二阶特征组合。那么什么叫二阶特征组合？很好理解，举个例子，比如两个特征，一个特征是性别，假设“性别=女”，另外一个特征是时间，假设“时间=双十一”，这两个特征如果组合到一起，你会发现是一个非常强的指示，是用户会不会买东西的一个特征，这就叫二阶组合特征，因为有两个单特征进行组合。

再往后，也就是现在这个阶段，大家都在讲 DNN 排序模型。那么 DNN 相对 FM 有什么好处？除了一阶和二阶特征外，它可以捕获三阶特征、四阶特征、五阶特征等更高阶的特征组合； FM 一般来说很难捕获高阶的特征，DNN 典型的特点就是可以捕获更高阶的特征。按照这个路线往后捋，你要把握核心的一点是：特征组合自动化，包括更高阶的特征怎么融合进去，这是 CTR 模型进化的总体方向。



通过上面这个 PPT，我再把后面要讲的内容捋一捋：

- 第一部分我会先介绍下排序模型的发展；

- 第二部分会介绍一下经典的线性排序模型有哪些，特性是什么；

- 第三部分会讲我们针对 FFM 提出的改进的“双线性 FFM”模型；

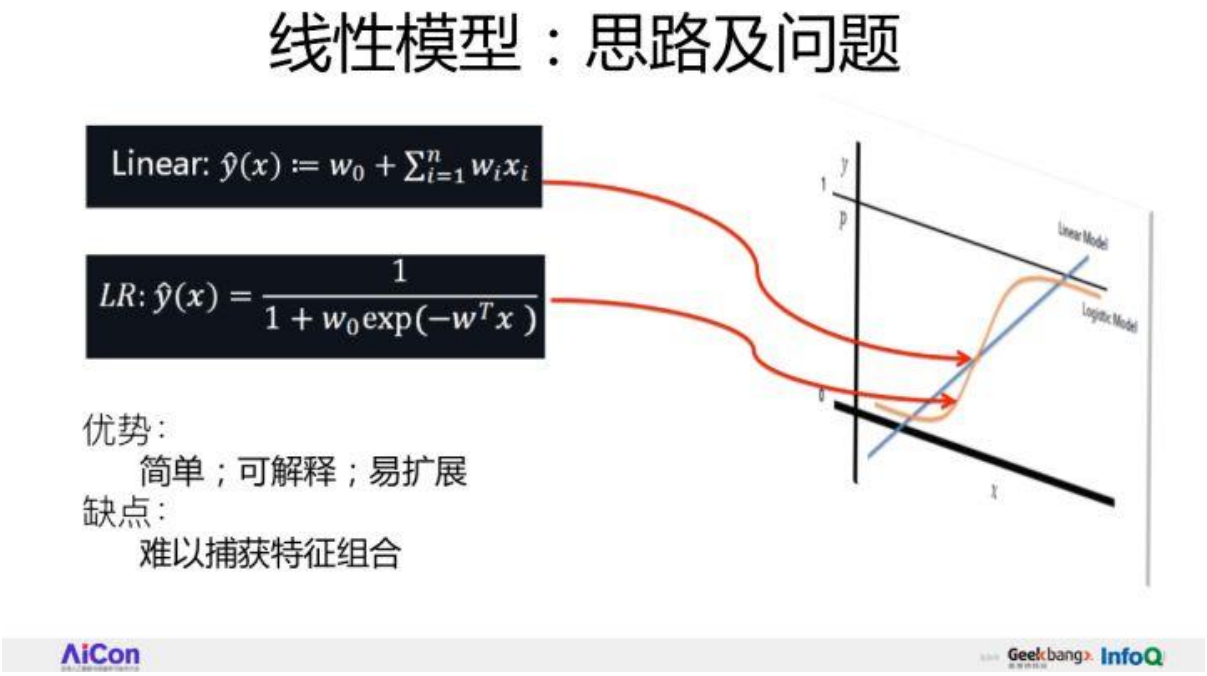
- 第四部分，我们讲一下 ranking 模型的发展趋势。目前来说，有两种演化的分支，一个就是以 W&D 做为起点的演化分支，另外一个以 Deep&Cross 做为起点的演化分支；

- 第五部分，也是最后一部分，我们对排序模型第一个演化分支也做了一个改进模型。核心思想是，FFM 模型怎么做成深度网络版本的，我们给出了一个新的改进版本的神经网络 FFM 模型。

线性排序模型：从 LR 到 FFM 模型

我先介绍一下线性模型。大家可能都有些机器学习的基础，这应该是有监督模型中最简单的一个。

所谓线性模型，比如说要做 CTR，很好理解，看这个公式：

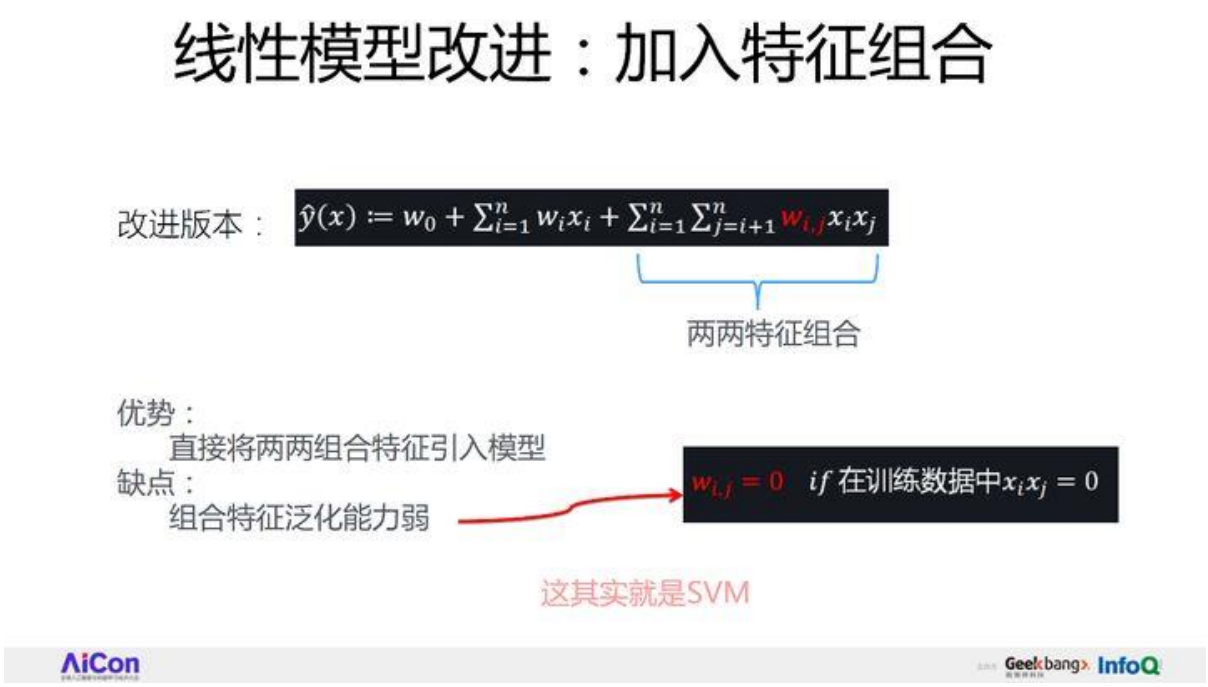


xi 就是某个特征值，意思要给每个特征学习一个对应的权重：wi，最终的模型预测值就是所有的特征值乘以这个权重，加起来求和，这就是最简单的线性模型。

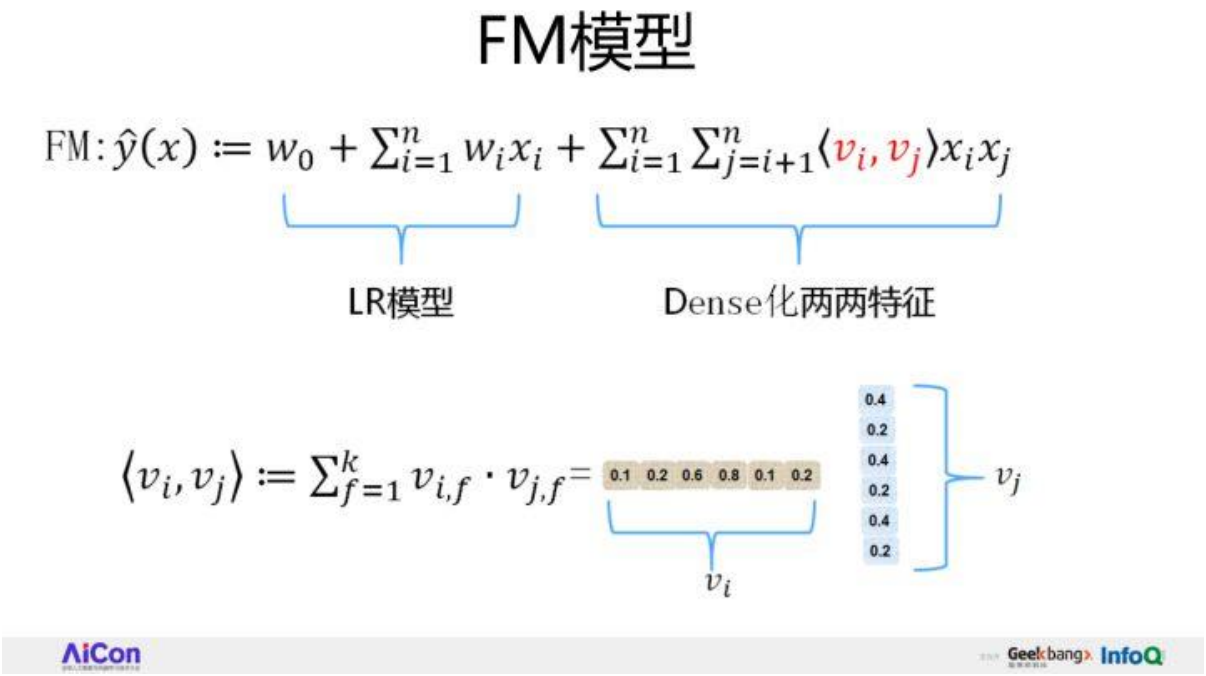
一般我们做 LR，会在上面求和基础上，套一个 sigmoid 函数，也就是上图中黄色的曲线，因为线性模型取值范围已经不可控，可以无限大，所以不好用。通过 S 形函数把它压到 0 和 1 之间，我就可以判断，是正面的结果还是负面的结果，LR 就是这样的过程。

那么线性模型有什么优点和缺点呢？首先它简单，所以好理解，上线快，速度快，这是它典型的优点。尽管我们到了深度学习时代了，但事实上，很多公司还在用 LR，因为它确实很好用。但它也有问题，我刚才讲了一句话：CTR 模型的核心是特征组合，但是你从上图的公式里看不到任何特征组合的迹象：单个特征乘以权重，特征间的关系没有被考虑，这就是 LR 最大的一个缺点。线性模型有这个问题，它不能够捕获特征组合，所以要改造一下这个公式，把特征组合揉进来。

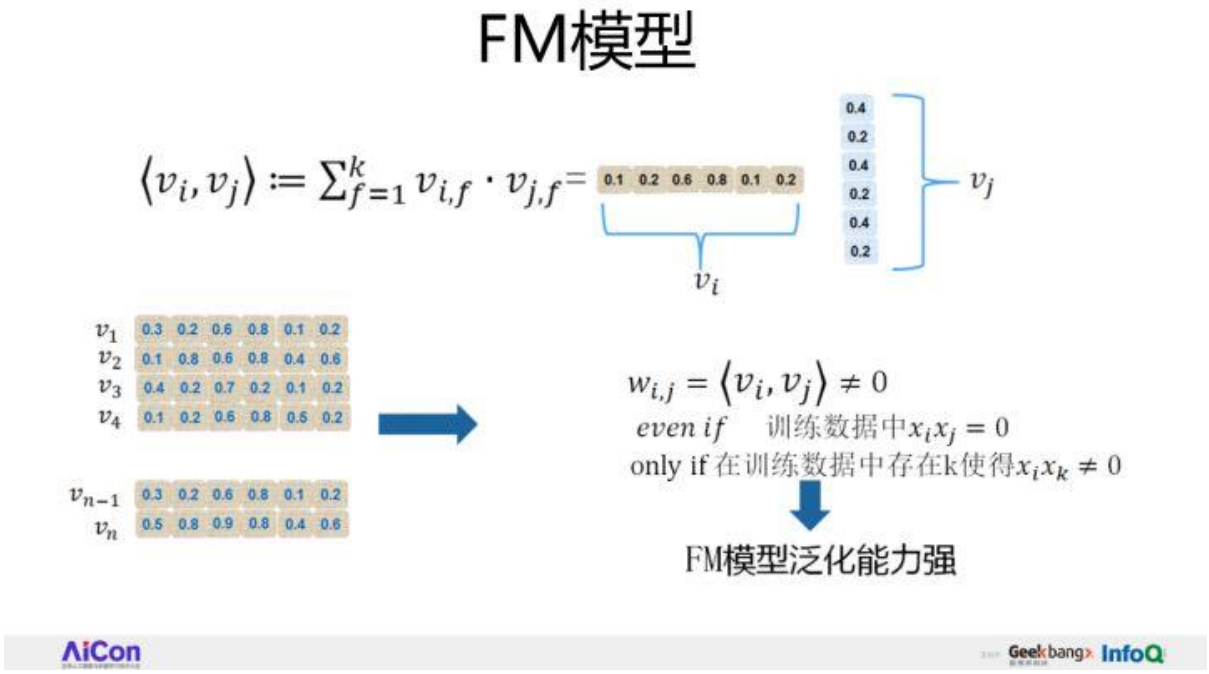
特征组合要怎么组合？任意两个特征的组合，可以把一个特征组合当作一个新特征，但既然它是新特征，它也要学个权重，下图公式中标红的 **wi,j** 就是这个特征组合的权重。



这样，我们就能把组合特征显式地、简单的揉进了这个公式，所以它的好处是：现在能够捕获两两特征组合。其实这么做，它也有问题，刚才我们这么改过来的模型本质是 SVM，它的问题是特征组合的泛化能力弱，于是我们可以进一步对它进行如下修改。



上面 PPT 中的公式，前面还是一样，就是 LR 模型，后面的也是体现了任意两个特征组合，和刚才公式唯一的区别在：**原先的 wi,j，换成了 vi 和 vj 的点积。**vi 和 vj 又是什么含义呢？vi 的意思是：对于 xi 这个特征来说它会学到一个 embedding 向量，特征组合权重是通过两个单特征各自的 embedding 的内积呈现的，因为它内积完就是个数值，可以代表它的权重，这其实就是 FM 模型。

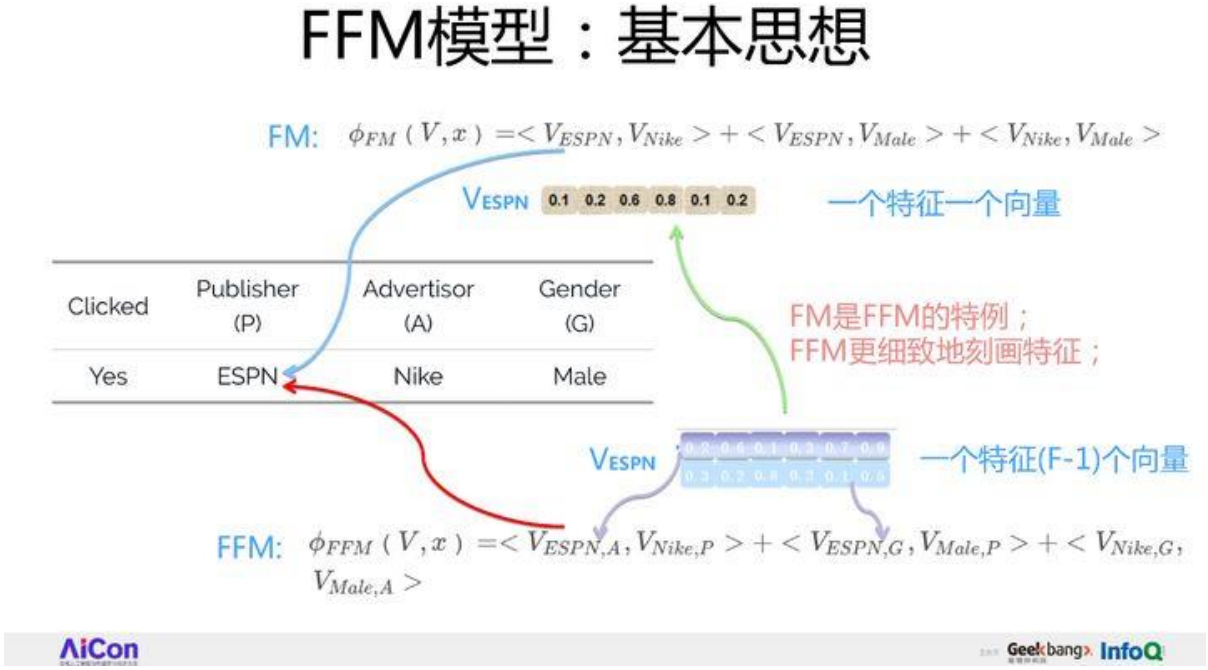


SVM 泛化能力弱，FM 的泛化能力强。

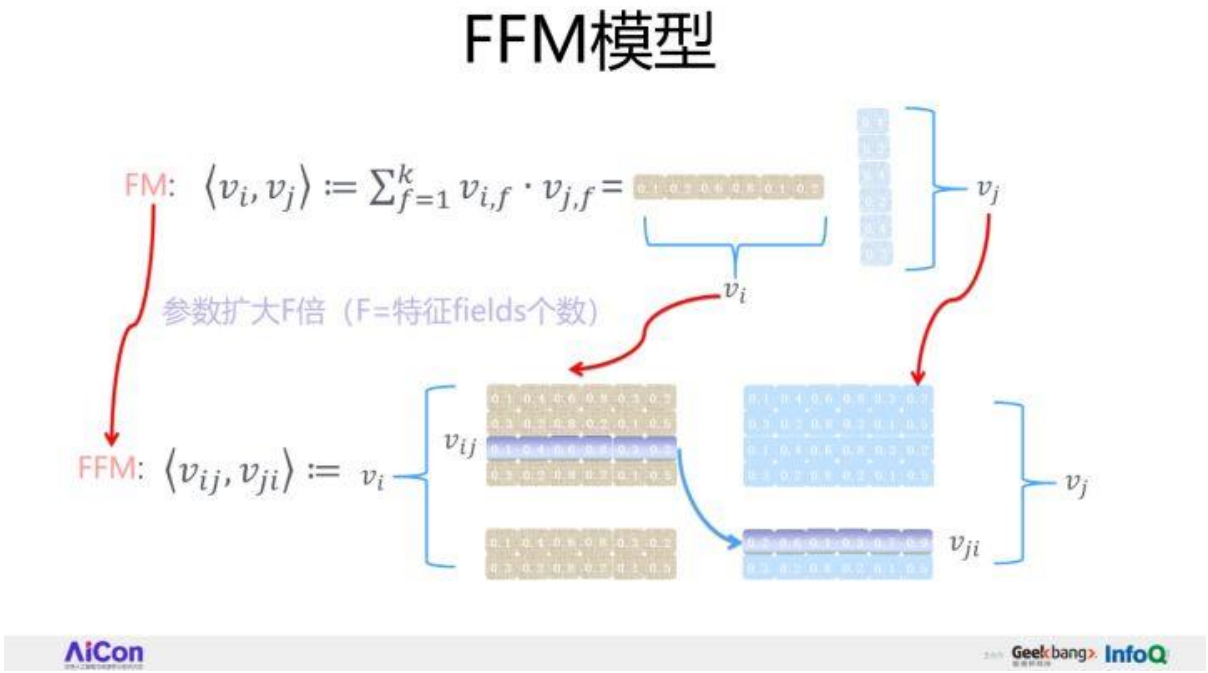
在 FM 之后，我们再改进一个新版本，就是 FFM 模型。我先定性地说下这个模型：它的效果比 FM 好，但是问题在于，参数量太大。

首先， FFM 模型的特性是什么？

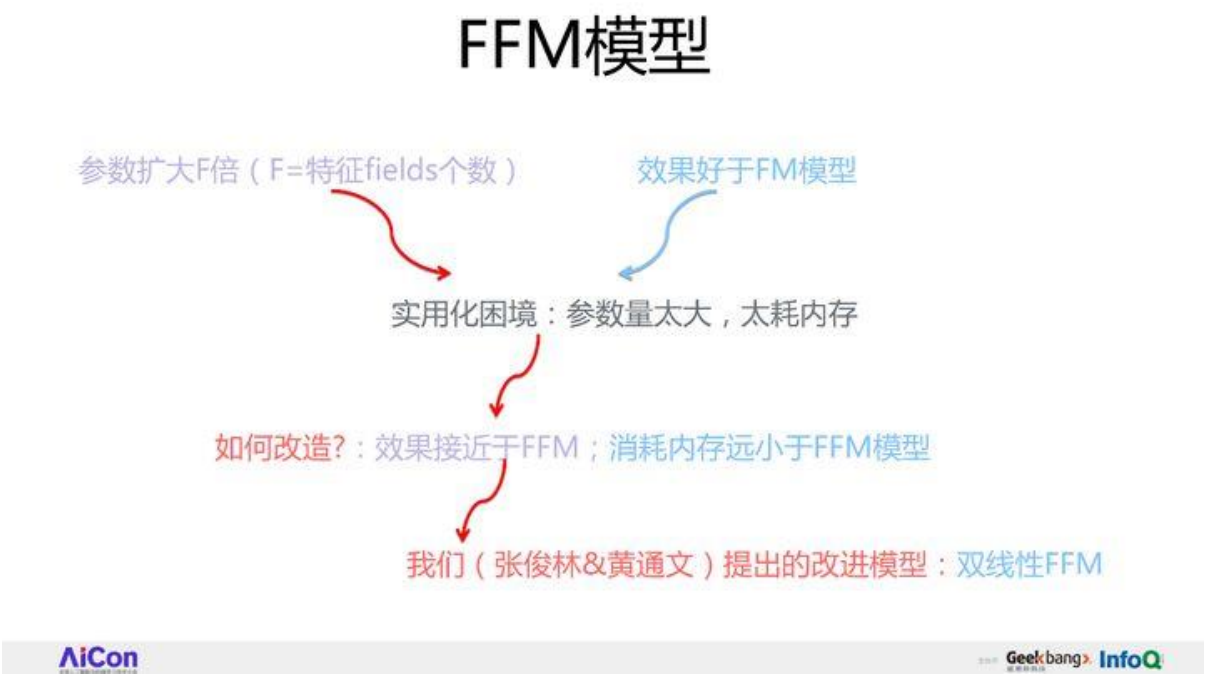
举例来说，有三个特征 fields，这是一个在线投广告的应用场景，比如说我要往 ESPN 这个网站投广告，第一个特征 Publisher 是要投放广告的网站是谁，第二个特征是这个广告主（Advertisor）是谁，本例中假设是 Nike；第三个特征，阅读者（Gender）是谁，阅读者的性别是男性（male)；那么他会不会点击？这个例子中是会点击；那么我们在这个例子上再想，FFM 在做什么事？



首先这个公式做了任意两个特征组合，它的特性是要把任意一个特征学成 embedding，这就是刚才讲的 FM 的做法，那么 FFM 是怎么做的呢？



FFM 是 FM 的一个特例，它更细致地刻画了这个特征。首先它做了任意两个特征组合，但是区别在于，怎么刻划这个特征？FM 只有一个向量，但 FFM 现在有两个向量，也就意味着同一个特征，要和不同的 fields 进行组合的时候，会用不同的 embedding 去组合，它的参数量更多。对于一个特征来说，原先是一个 vector，现在会拓成 F 个 vector，F 是特征 fields 的个数，只要有跟其它特征的任意组合，就有一个 vector 来代表，这就是 FFM 的基本思想。

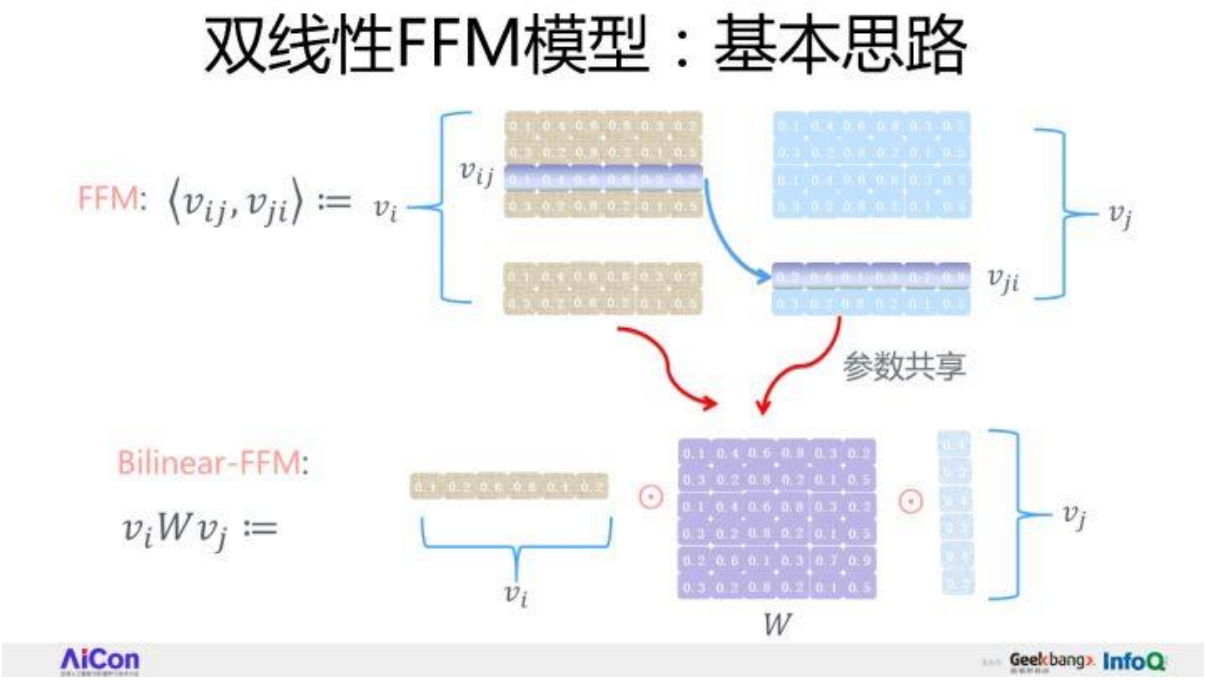


对于 FFM 的某个特征来说，会构造 F 个 vector，来和任意其他的 fields 组合的时候，各自用各自的。它有什么特点呢？首先，FFM 相对 FM 来说，参数量扩

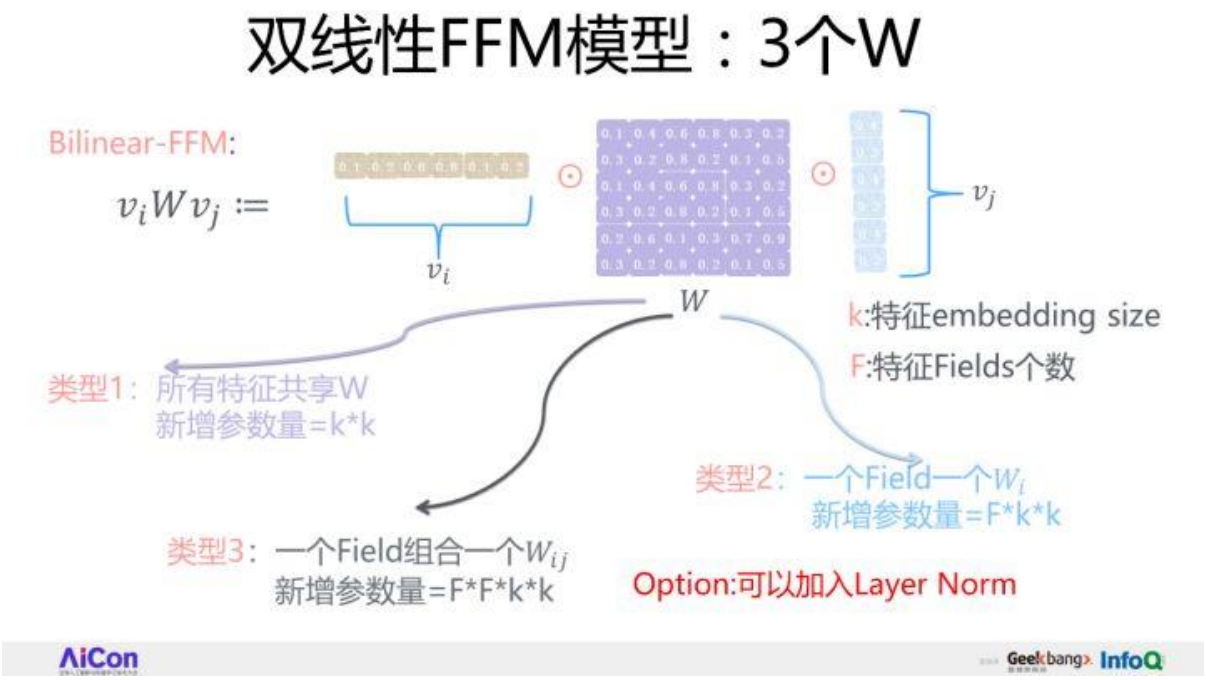
大了 F 倍，效果比 FM 好，但是要真的想把它用到现实场景中是有问题的，而问题同样在于参数量太大。参数量太大导致做起来特别耗内存，特别慢，所以我们的改进目标是把 FFM 模型的参数量降下来，并且效果又能达到 FFM 的效果。于是我们改了一个新模型，我把它叫“双线性 FFM 模型”。

FFM 模型改进版：双线性 FFM（Bilinear-FFM）模型

下面我们介绍一下双线性 FFM 怎么做的，这是 18 年 5 月份左右，我们（张俊林&黄通文）在改进 FFM 的过程中尝试出的一个有效的改进模型。这个图基本就显示原理了：



因为每个特征现在有 F 个 vector 来表示它的参数空间，每个特征都需要跟上 F 个 vector，一般 CTR 任务中的特征数量是非常大的，所以 FFM 的参数量就异常地大。能不能把跟每个特征走的参数矩阵，抽出它们的共性，所有特征大家一起共享地来用这个参数？如果你一起用的话，就能够共享这个参数矩阵，你就能把参数量降下来，这就是我们讲的双线性 FFM 的核心思想。vi,vj 还是跟 FM 一样，还是用一个 vector 来表达，但是把两个特征交互的信息放在共享参数里面去学，这就是双线性 FFM 的核心思想。



我刚才讲了一个简单的思路，理论上里面还有些变化点，比如这个共享参数矩阵 W 怎么设计？这是有些学问在里面的。我们可以有三种选择。

类型 1，不论有多少个特征，大家都共享同一个 W，这是参数量最小的一种形式。W 的参数量是 K×K，K 就是特征 embedding 的 size。

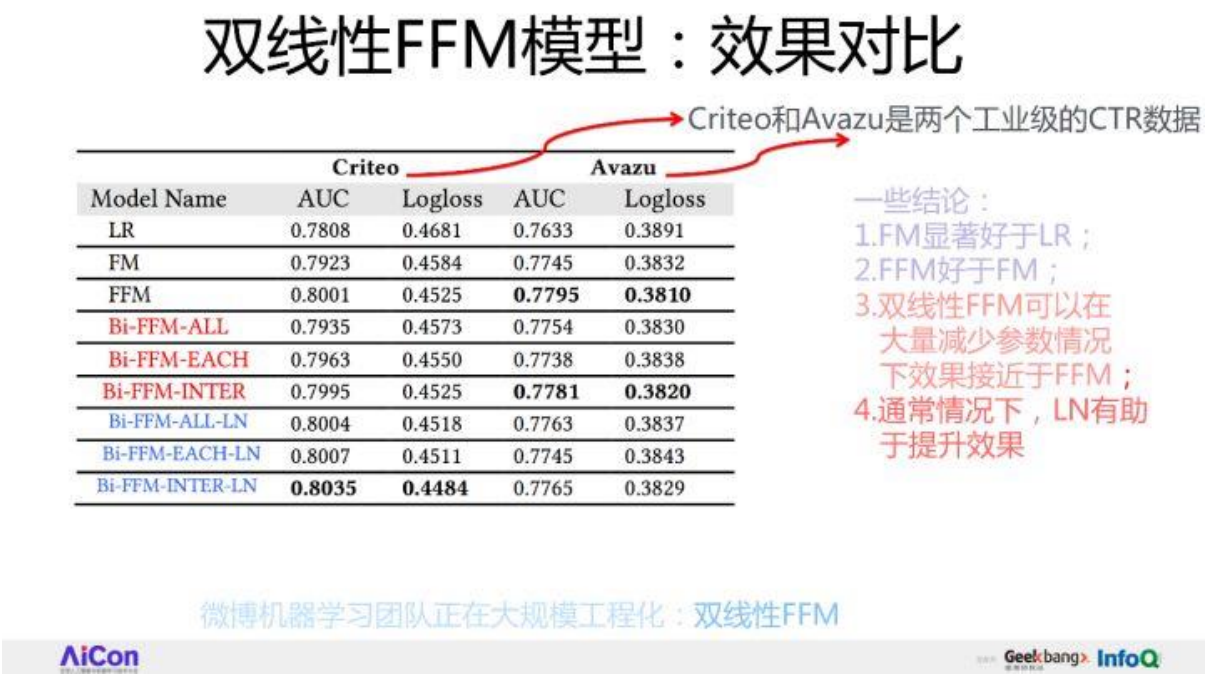
还有什么改进的方法吗？能不能每个 fields 给一个不同的 W 呢？应该是可以的，我们在类型 2 就用到了这个方法，有 12 个 Fields，就有 12 个 W，每个 Fields，各自学各自的 W。

还可以拓展，就是类型 3。我们知道，Fields 还可以组合，有 12 个不同 Fields，就有 12×12 种 Fields 间的组合，如果每个组合给一个 W，这就能更加细化地描述特征组合了。

所以这里可以有有三种不同的 W 定义。此外还可以有一个变化点，即可以加入 Layer Norm，我们试过，加入 Layer Norm 影响比较大。

我们看一下结果。

暂时不说刚才几个改进模型，先说几个基础模型：LR、FM、FFM。这是我们在 Criteo 和 Avazu 两个数据集上做的实验，这两个数据集是工业级，大约在四千到四千五百万的 CTR 数据。



先说 LR 和 FM，可以从数据里面看出来，FM 是显著优于 LR 的，这两个数据都是这样。首先要确认一点，不要期望太高。你期待提了五个点、十个点，在 CTR 里，尤其是这两个数据集合，你是见不到这种现象的，在很多大规模数据集合里，AUC 能升一个点就是非常显著的一个增加。FFM 跟 FM 比，AUC 从 0.7923 到 0.8001，也是显著提升的，这是这三个模型的特性。

上面说过，双线性 FFM 的共享参数矩阵 W 有三种类型。这三个类型中哪个效果好？双线性 FFM 的 AUC 达到了 0.7995，Fields 组合效果是最好的，接近于 FFM，但稍微低一点。还有个变化点，可以加入 Layer Norm，同样加到了那三个模型里面去，最好的效果达到了 0.8035，已经超过 FFM，而且效果比较显著。

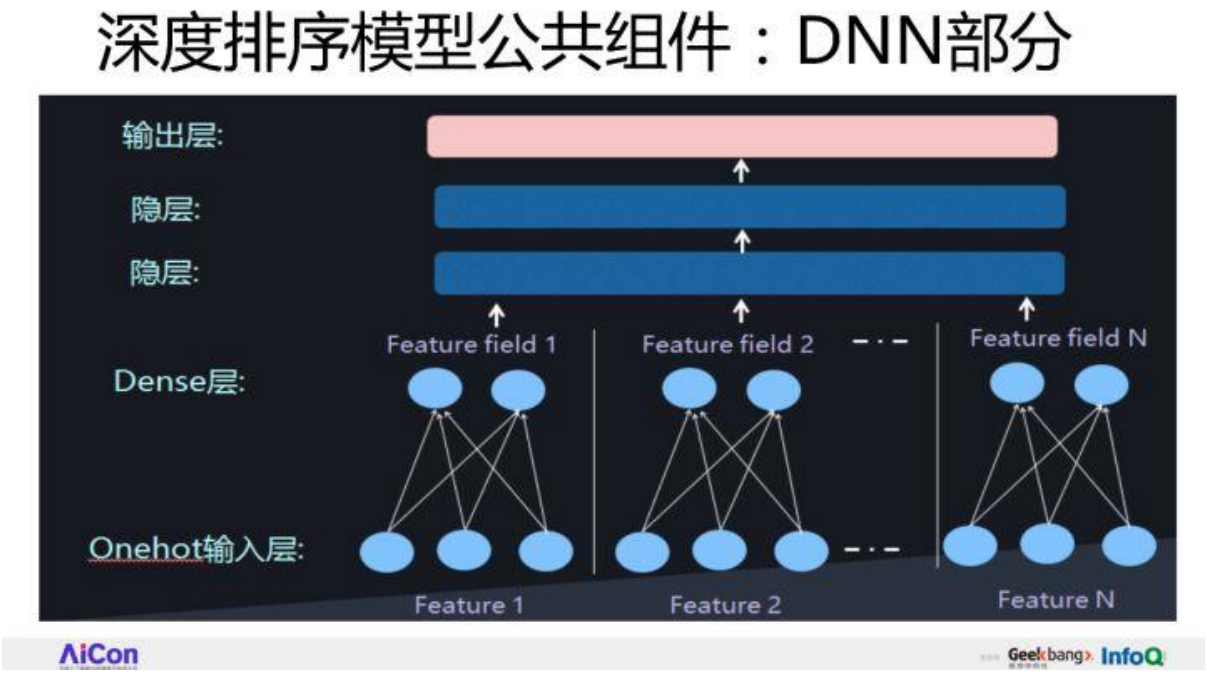
所以我们可以下结论：双线性 FFM 的推荐效果是可以达到与 FFM 相当，或者说超过 FFM 的性能的，而且 Layer Norm 对这个事情有明显的提升作用。

我们来估算一下，改进的双线性 FFM 模型，它的参数量跟 FFM 比是什么情况？如果说我们用 Criteo 这个 4500 万的数据集，它有 230 万个特征，39 个 Fields，假设 embedding size 是 10，如果用 FFM 就会有 8.97 亿的参数量，而用双线性 FFM，FM 部分是大概 2300 万的参数，刚才三个改进模型中，类型一 100 个参数，类型二 3900 个参数，类型三 15 万参数，与 FFM 相比，参数差了 38 倍，但性能两者是相当的，这就是这个模型的价值所在。

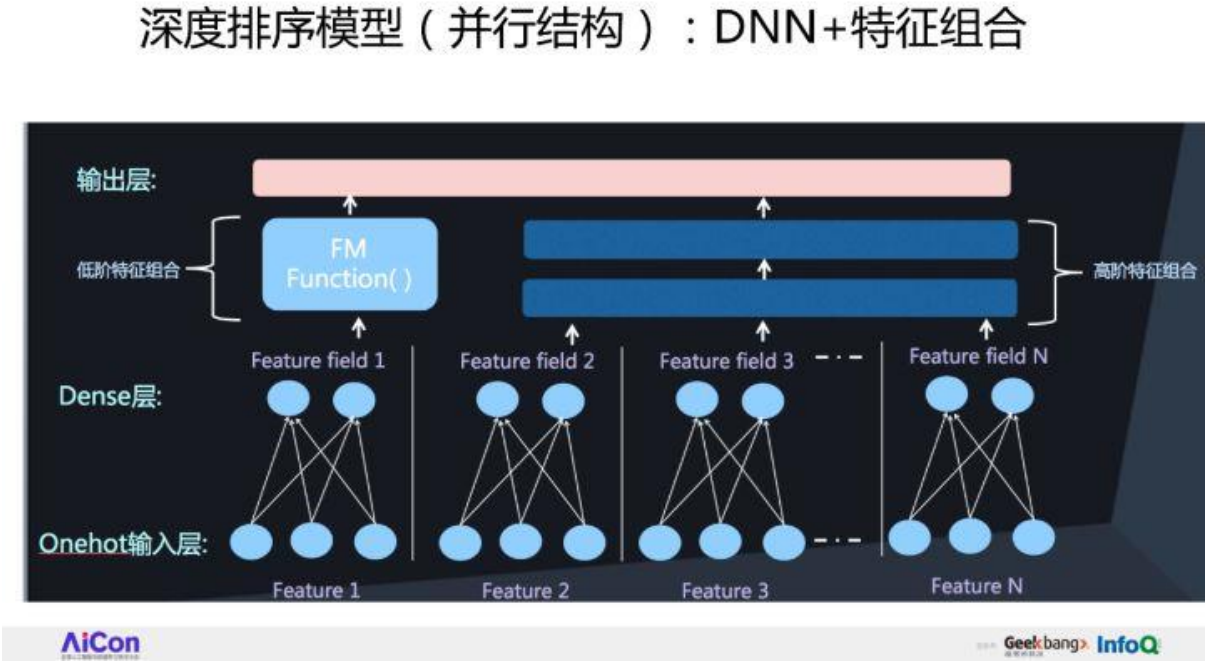
总结一下双线性 FFM 模型：它的性能接近于 FFM，但是参数量是 FFM 模型的 2.6%，这是它的优点所在。

深度排序模型：从 Wide&Deep 到 XDeepFM 模型

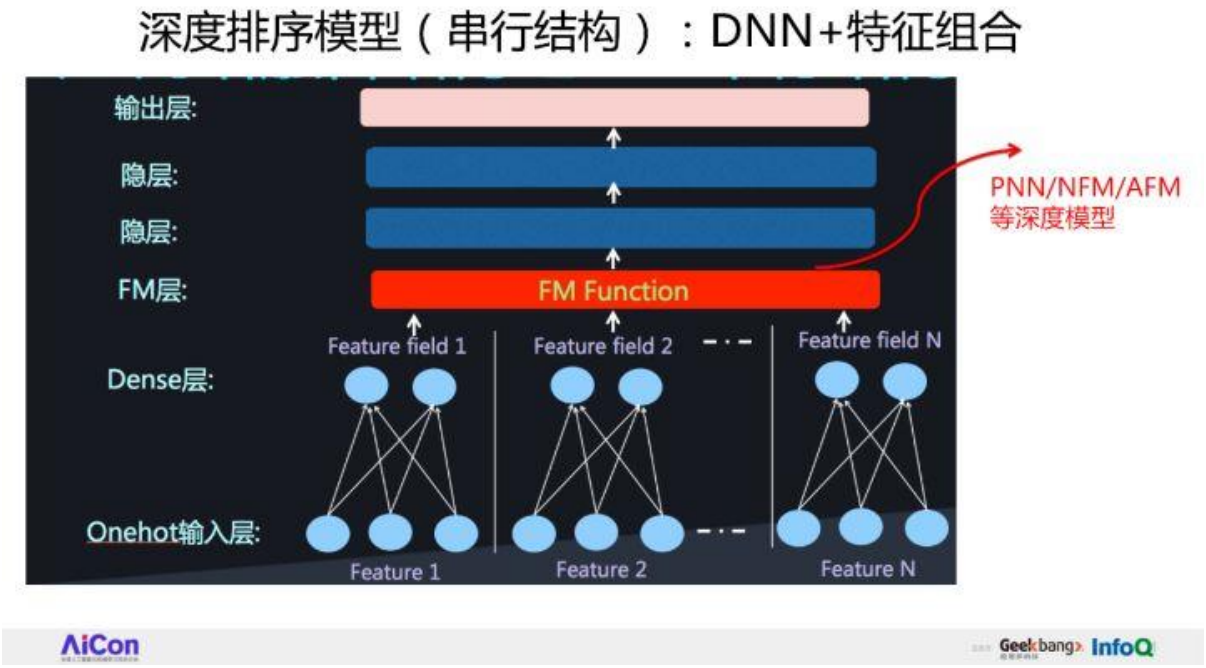
下面我介绍一下深度模型，首先介绍一下它的发展历程。



所有的深度学习，做 CTR 的模型时都会有 DNN 的部分，没有例外。什么含义呢？特征输进去，然后把它转换成 embedding，上面套两个隐层进行预测，这是所有模型公有的一部分。



我把现在这个深度 CTR 模型会分成了两大类。从结构来说，第一类我把它叫并行结构，它有 DNN 结构外的另外一个组件，我管它叫 FM Function，它捕捉特征的两两组合，两者关系看上去是个并行的，所以我把它叫并行结构。



除了并行还能怎么修改这个结构？可以把它搞成串行的，前面一样是 onehot 到 embedding 特征编码，然后用 FM Function 做二阶特征组合，上面套两个隐层做多阶特征捕获，这是串行结构。典型的模型包括：PNN、NFM、AFM 都属于这种结构。

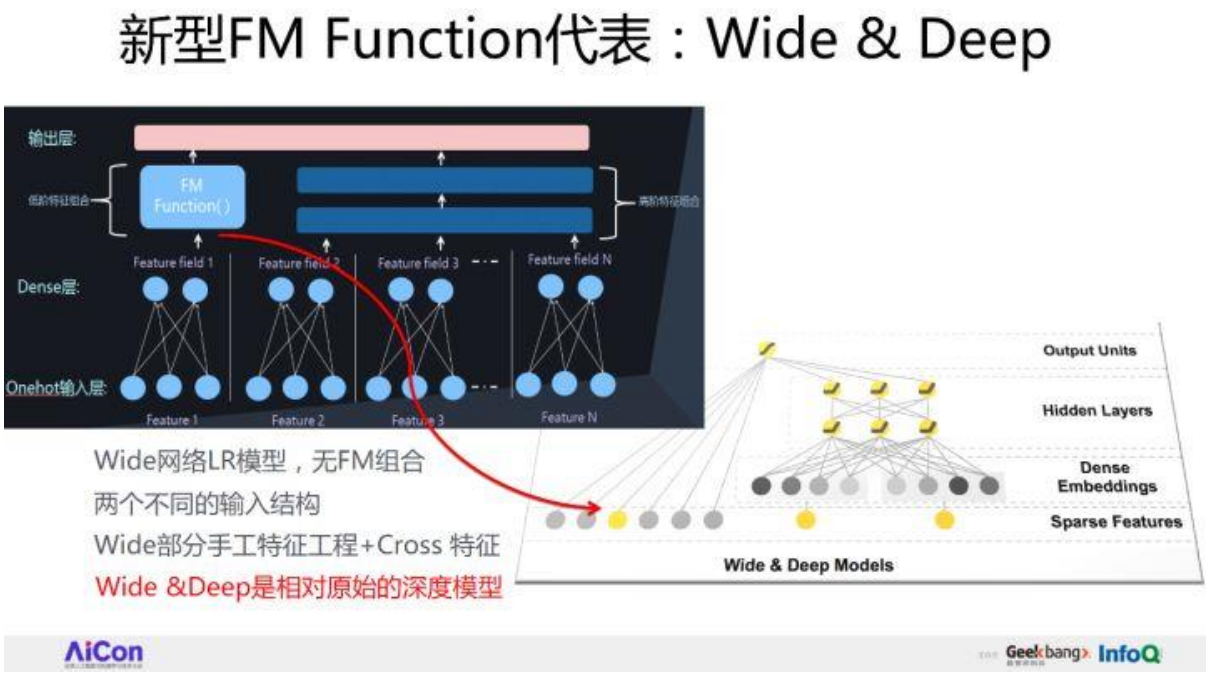
如果再归纳一下，你会发现深度排序模型，现在有朝两个研究路线走的趋势。这两条演进路线怎么走的呢？

第一条路线，提出新型的 FM Function，就是怎么能够设计一个新的 FM Function 结构，来更有效地捕获二阶特征组合，比如说典型的模型包括 Wide&Deep，DeepFM，NeuralFFM 等，就是用来做这个的。

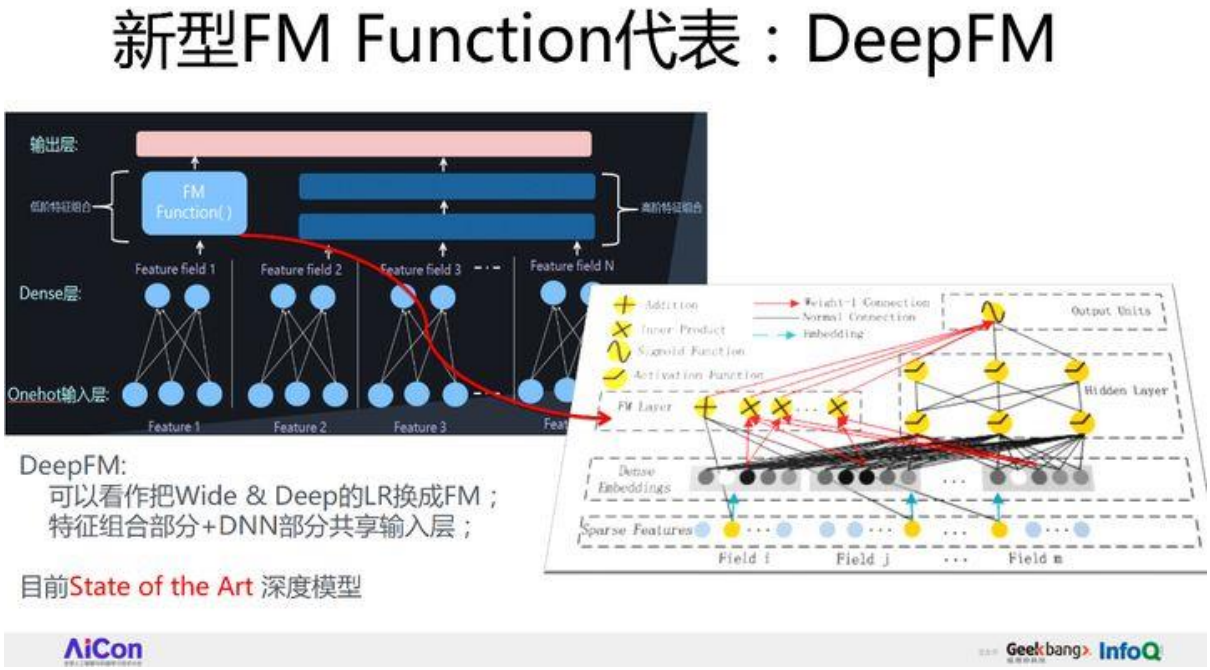
第二条演进路线，就是显式地对二阶、三阶、四阶…K 阶组合进行建模。目前的研究结论是这样的：对 CTR 捕获二、三、四阶都有正向收益，再捕获五阶以上就没什么用了。典型的代表模型是 DeepCross、xDeepFM。

针对这两条演进路线，我会各自介绍两到三个代表系统。

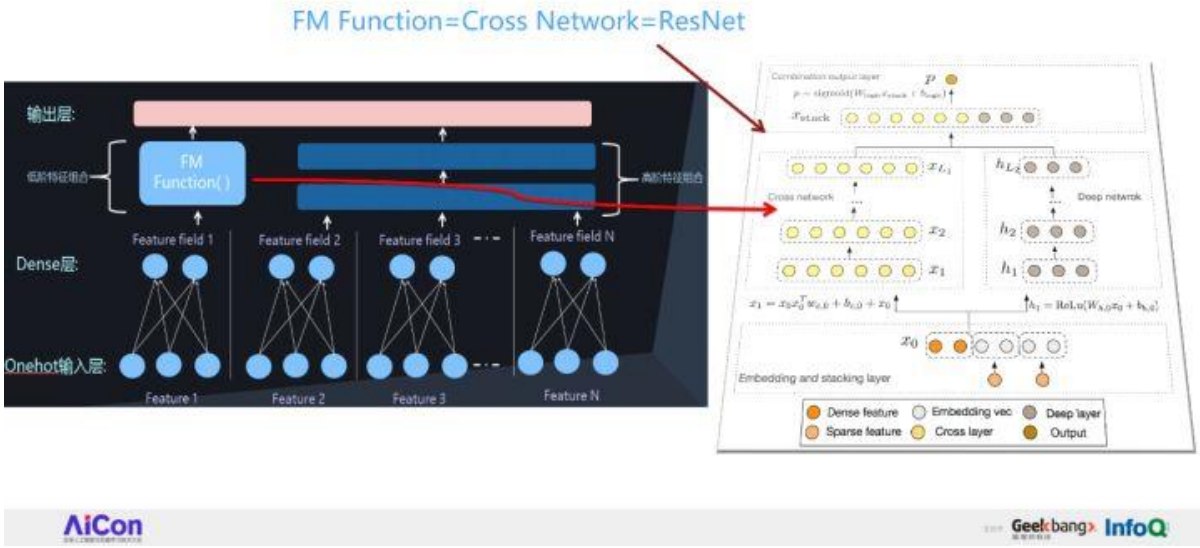
第一个代表系统：Wide&Deep，这个系统我相信大家都听说过。



Wide&Deep 的结构是什么呢？实际上就是我画的并行结构，右边就是 DNN 部分，左边的 FM Function 用的是线性回归。我个人认为，Wide&Deep 是相对原始的模型，LR 有的问题它也有，特征组合需要人去设计，Wide&Deep 也有这样的问题。

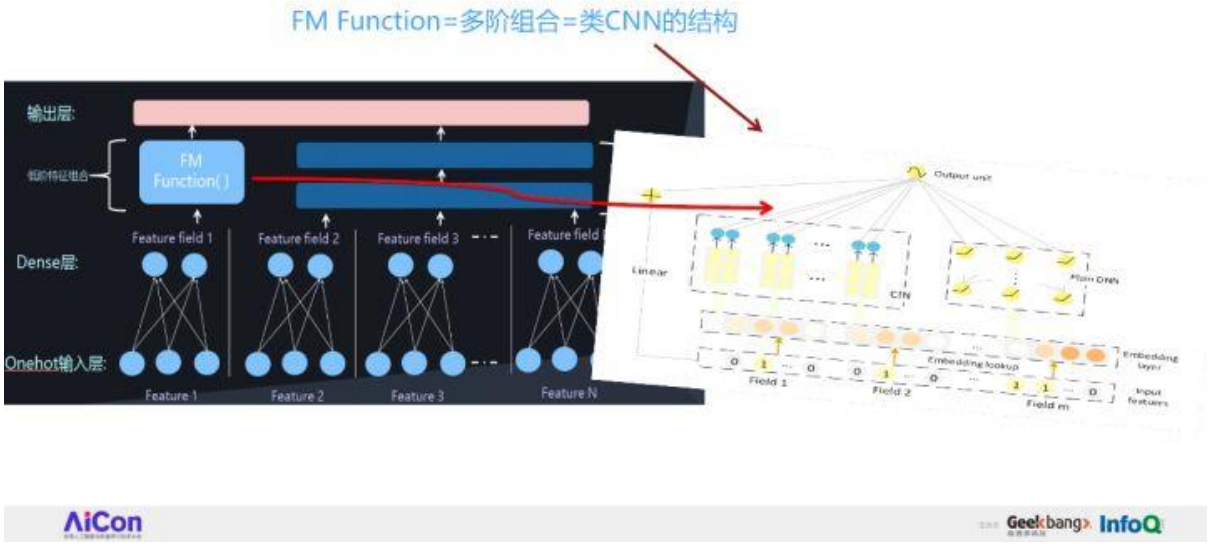


我们可以改进一下。DeepFM 模型，它相对 Wide&Deep 做出了什么改进呢？很简单，其实就是把 FM Function 的 LR 换成了 FM，就能自动做特征组合了，这就是 DeepFM。如果想部署深度模型，我建议可以考虑这个模型，这是目前效果最好的基准模型之一。而且我认为 DeepFM 是个目前技术发展阶段，完备的深度 CTR 模型。所谓完备，是指的里面的任意一个构件都有用，都不能少，但是如果再加新东西，感觉意义又没那么大，或者太复杂了工程化有难度。完备是从这个角度说的。



第二条路线的两个代表是：Deep& Cross 和 xDeepFM。Deep&Cross 用来做什么？显式地做高阶特征组合。就是说设计几层神经网络结构，每一层代表其不同阶的组合，最下面是二阶组合，再套一层，三阶组合，四阶组合，一层一层往上套，这就叫显式地捕获高阶特征组合，Deep&Cross 是最开始做这个的。

显示高阶组合：xDeepFM



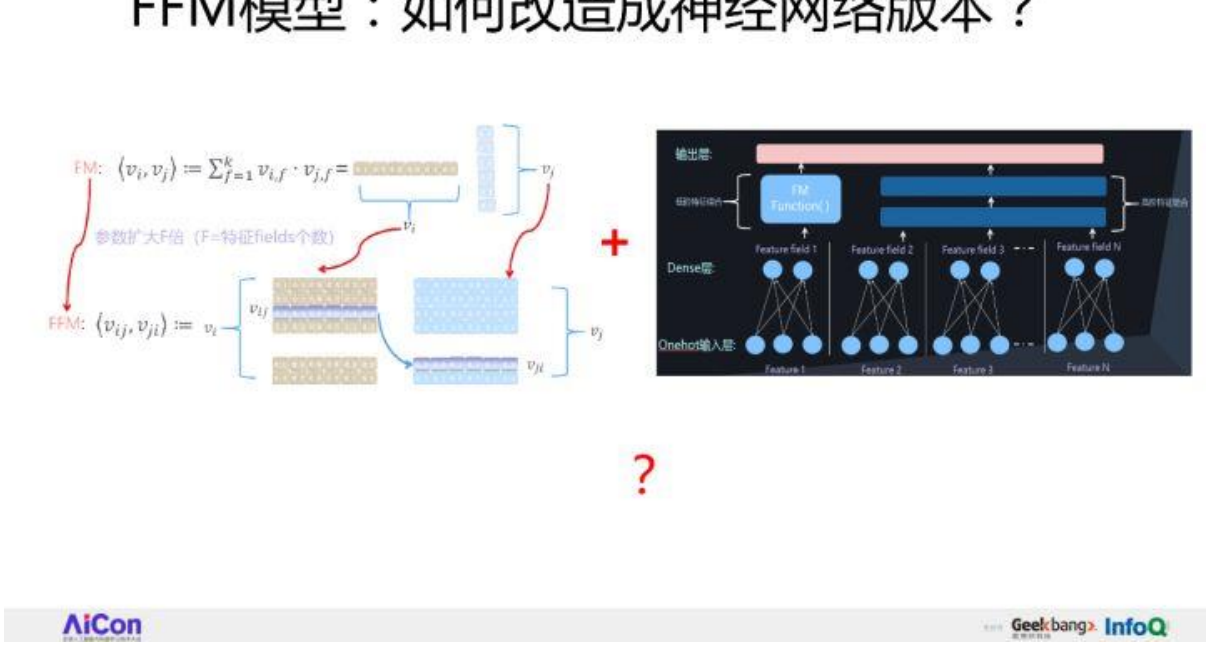
xDeepFM 是微软 2018 年发的一篇新论文，它是用来把二阶、三阶、四阶组合一层一层做出来，但无非它用的是类 CNN 的方式来做这个事的。这是第二个路线的两个代表。尽管这个符合模型发展趋势，我个人认为这种模型太复杂，真正部署上线成本比较高，不是优选方案。

接着来讲一下，我对深度 CTR 模型的个人看法。

首先可以看到，现在所有的模型结构趋同，要么并行，要么串行，几乎没有例外，这其实是能侧面反映出一些问题的，至于是什么你可以考虑考虑；第二，输入问题基本解决了，基本都是 onehot 和 embedding 做映射。模型的核心所在是二阶特征组合怎么设计网络结构。几乎所有的方法，或者大多数方法变化点集中在这。另外就是通过多层网络，显式地做二、三、四阶，这是目前的一个趋势。再有，多模态融合肯定也是个大趋势，在这个场景下，深度学习是有其不可替代的作用的。为什么这样？你可以考虑一下，其实原因很简单。

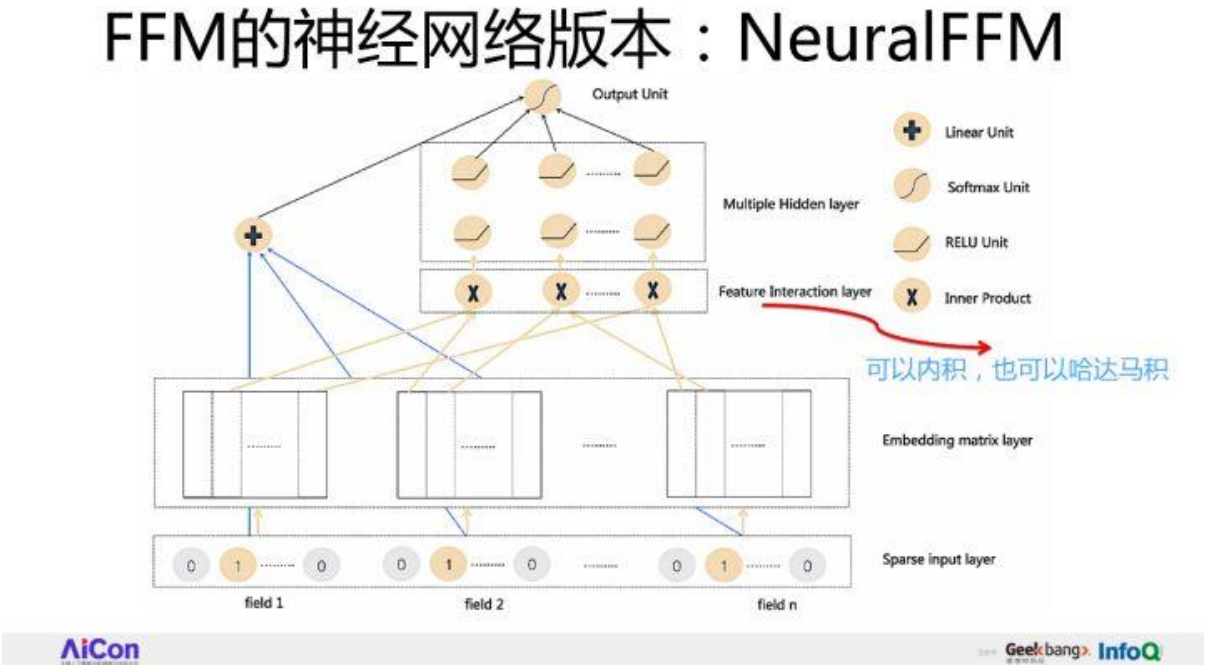
DeepFFM 模型

最后介绍一个改进版本的 DeepFFM。



首先我抛给大家一个问题：刚才我们讲了，FFM 模型有它的对应版本，是 DeepFM 模型，那么问题来了：能不能设计一个模型神经网络版的 FFM？肯定是可以的。

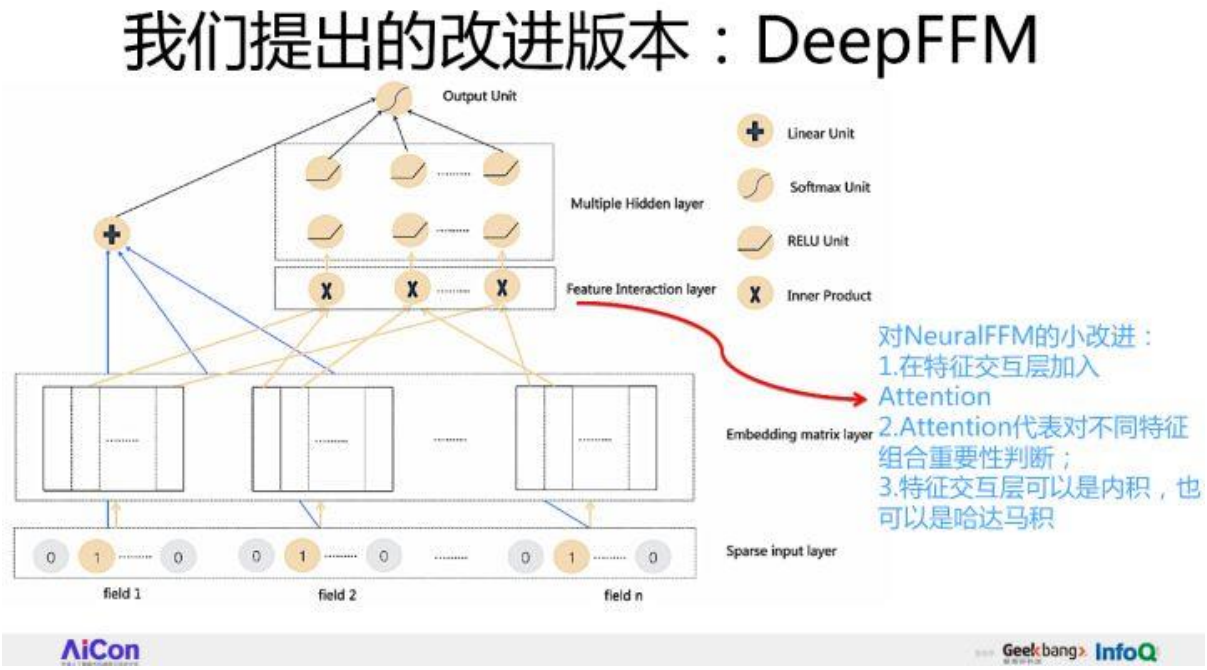
我先给出一个前导模型：NeuralFFM，这是 2017 年南京大学杨毅等人提的，他们在参加腾讯的竞赛中提出了这个模型，这个模型效果比较好，是单模型的第三名。我们对这个模型做了一个改进，把它叫做 DeepFFM。



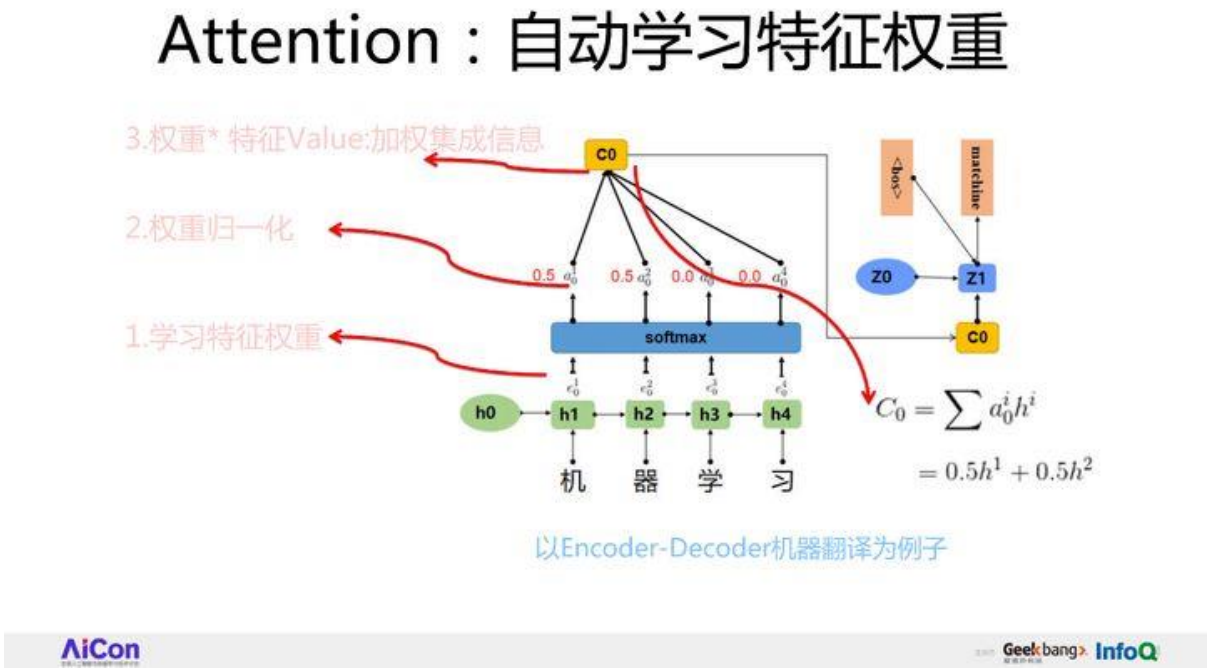
先说一下 NeuralFFM 模型怎么做的，这是我刚才抛给大家问题的解答。

要做一个神经网络版的 FFM 模型该怎么做？先把它想成 DeepFM，再想怎么把 FM 部分改成 FFM。如果是 DeepFM，一个特征、一个 vector，然后在上面做两两特征组合，这就是 DeepFM 的思路。FFM 区别在哪里？一个特征现在不是一个 vector，是 F 个 vector，在两两做特征组合的时候，需要做特征交叉，这不就是深度的 FFM 模型吗？很简单，NeuralFFM 就是这个思想。

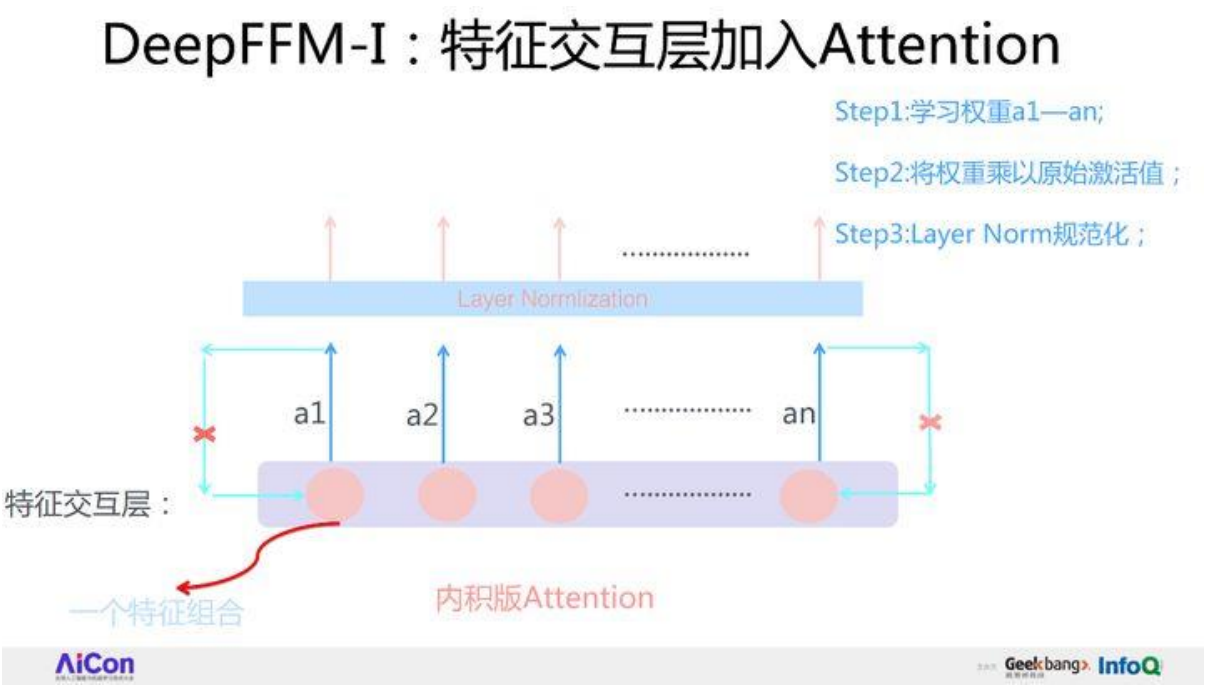
但是这里有个变化点。两个特征进行组合的时候，可以用内积做，也可以用哈达马积做，这是有区别的。什么是内积呢？就是两个向量，每个位置对应的相乘求和，表示一个数值；哈达马积怎么做的呢？它比内积少做了一步，把对应的位置相乘，乘完之后不进行求和。



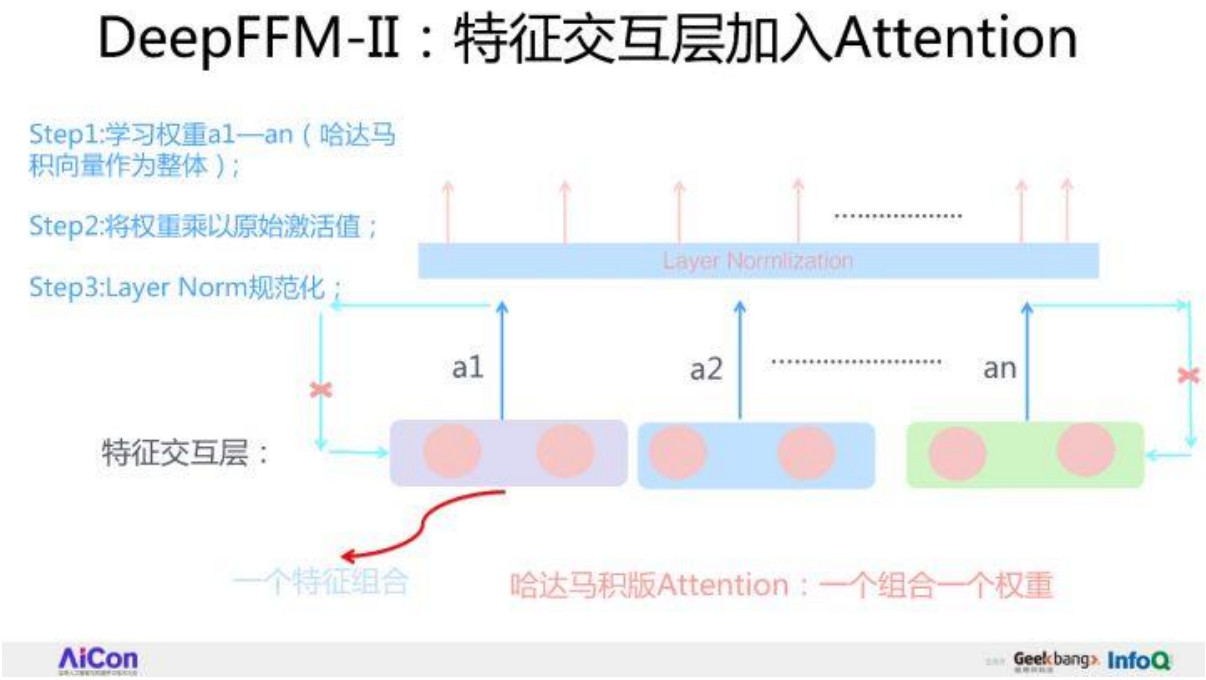
我们这个 DeepFFM 是怎么改进的呢？在特征交互层，也就是两两特征在这组合，我们加了一个 Attention，也就是一个注意力模型进去。两两特征组合有很多，但是有的组合特征比较重要，有的没那么重要，怎么体现这个思想？给每个特征组加个权重就可以。怎么给权重？加个 Attention 就可以了。这就是我们 DeepFFM 的核心思想。当然，我本人认为这种给特征组合权重的方式意义不大，所以就简单试了试，包括 AFM，也是类似思想，所以意义也不大，个人意见。我自黑起来是有很高水平的，不过这次是真心话。



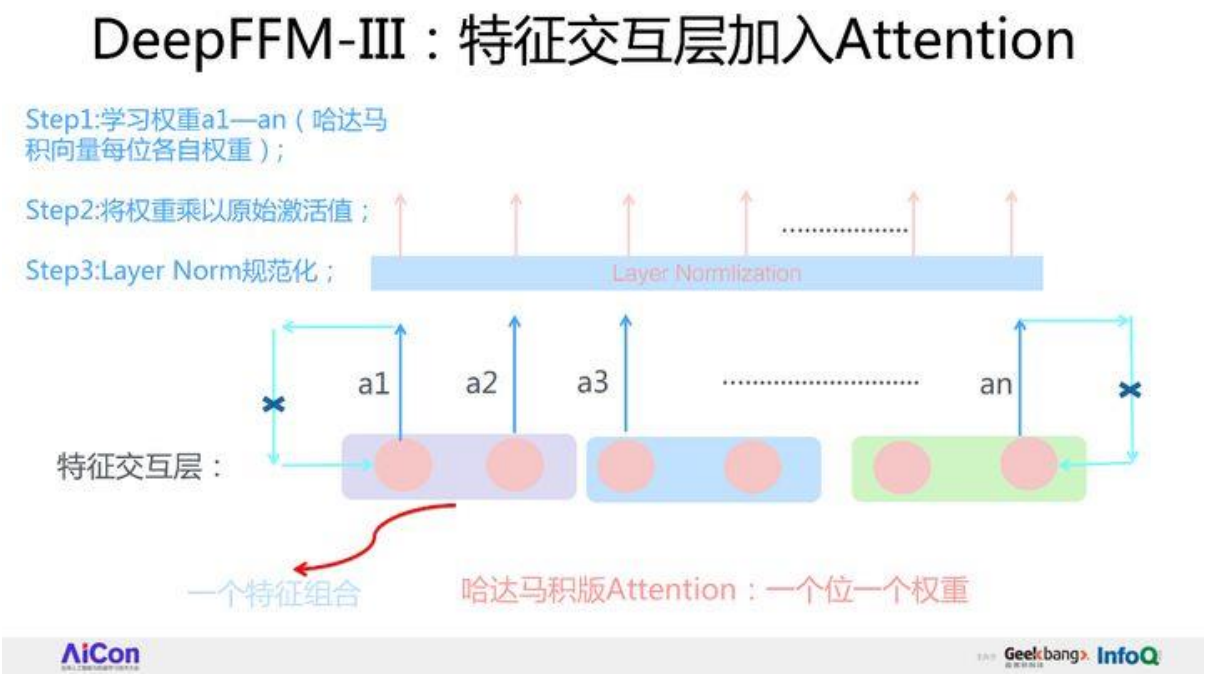
至于什么是 Attention，时间原因不介绍了，不懂的可以自己去查。Attention 是个必备基础知识，无论在推荐还是 NLP 里都是这样。



再回头讲，我们是怎么改进的，也就是怎么给特征组合做 Attention 的。在特征交互层，我们先给出个内积版 Attention，这是一个特征组合，组合完是一个值，我给每个两两特征组合的值都加个权重 a1-an，加完权重之后，再把权重乘到值里，再加上 Layer Norm，往上面 DNN 的隐层去走，这是第一个改进版本。



有没有新的改进版本？肯定有，前面说过，不仅有内积，还有哈达马积。哈达马积是说两个特征组合完之后是个向量，这是一个特征组合，里面有若干个神经元，但它整体代表一个特征组合。我给组合完的 vector 整体加个权重 a1-an，每个都加权重，一样再乘回来，上面再套 LN，这就是哈达马积版本。



那么，能不能继续改进？我刚才是把它当作一个整体求 Attention，还可以把每一位拆开，每个位都加 Attention，每个位上面套层 LN，这就是第三个版本。

来看一看实验结果。先看刚才讲的几个深度典型模型的代表是什么效果？

DeepFM，我们实际测过，想改造其它模型把指标做得比它高得明显，这是很不容易的；xDeepFM 是 3 个基础模型中效果最好的；NeuralFFM 和 DeepFM 比，在这个数据集上并没有很大提高，比原来稍有一点提升，但是在 Avazu 上是有明显提升的；DeepFFM 什么效果呢？下面标红这三个，尤其是第二个模型做到 0.8091，也就是说在所有的模型里面效果最好的，当然，提升也比较有限在 Avazu 来说也是效果最好的。

总结

最后总结一下。

今天讲了大规模推荐系统是怎么构成的：包括召回加 ranking，我们的主要内容放在 ranking 阶段；之后，我回顾了一下线性模型，重点提到了双线性模型，它的特点是可以在参数量极小的情况下，性能达到类似于 FFM 的效果；然后，我给大家分享两个深度模型的演进路线，以及改进版本的 DeepFFm，效果目前来看还是比较好的。当然，我个人不认为 FFM 模型是个好的实用化的往线上部署的模型，当然，这是另外一个话题。

谢谢大家。

52 面经　vivo（推荐算法工程师）、美团（后端）、快手

01VIVO 面试情况 +面试题总结

vivo（北京推荐部门）：行测 + 笔试（3 道算法题，20 道开发岗八股文）+ 一轮技术面 + 一轮 HR 面

行测：不用花太多时间刻意准备，但是有必要在做之前刷几套，注意不要做公务员的行测，据说好多厂的行测都是来自于某个题库，我自己找到的是 职题库 app， 不知道牛客上会不会有免费的。

笔试：

算法题：1. 简单的 dp，类似于跳台阶那题。2. 滑动窗口，同向双指针相关。3. 多重背包问题，也可用 dfs 给出暴力解。

选择题：涉及计算机底层原理，代码段分析，计算机网络，都是特别特别基础的问题，如果专注算法，这方面比较薄弱也没事，只要是科班，凭借直觉也能做出来。

技术面（35 分钟）：vivo 的技术面不难，
1、请介绍一下你简历上的所有项目。
2. 二叉树最小深度
3. 股票系列 II ，都是很熟的题目，8 分钟就都 ac 了，
最后“不错，你有什么想问我的？”
总结：vivo 的 HR 面特别关键呀，我的很多小伙伴都是在这一趴被挂的，毕竟技术面试只有一轮。
02 快手面试情况 +面试题总结
快手（社区科学部）：3 轮技术面 + HR 面
① 技术一面
一、 自我介绍
二、 项目深挖
1. 说一下 DIEN 的原理是什么，有什么创新点。
2. ESSM 的原理和基本思想是什么？
3. 为什么使用这个模型而不是其他模型？
4. GRU 通过哪两个方式解决长程依赖问题？
5. FM 的时间复杂度是什么？
6. 设想一下工程场景，如果使用 FM 你会怎么做来实现线上的高效推荐？
7. AUC 是如何绘制的？
8. 在工程上如何计算 AUC ？
三、算法题
1. 翻转字符串
2. 三数和
3. 一道从来没见过的题目，没写出来，后续也没找到。所以如果碰到会的题，一定要放慢做题速度，做太快会提升面试官对你的谜之期望...
四、反问
② 技术二面
一、自我介绍
二、项目深挖
1. 你这个数据集每一个优化目标的正负样本比例是什么？
2. 你是根据什么选择 PLE 的 shared-expert 个数？各个任务的 gates 权重分布大概是什么样的，据此你能得出什么结论？
3. 我看到你为每个任务加入了 FM 来增加记忆能力避免过度泛化，这个想法不错，但是这样做难道不会导致 Embedding 层的更新出现冲突吗？
4. 你用了什么优化算法？对于一个冷启动的短视频和一个热门短视频，如果使用 Nadam 进行学习，最终二者的特征向量模长有什么区别？
5. 我看到你使用了 预训练的 Embedding 经行初始化，但是训练网络的时候依然继续训练了初始化的 Embedding，这和直接随机初始化共同训练有什么不同吗？要知道，你预训练的时候同样间接使用了标签信息，我觉得效果不会有太大不同，你觉得呢？
6. 我看到你使用了矩阵分解作为预训练 Embedding 的一部分，为什么使用矩阵分解呢？为什么不使用监督学习的方式来训练呢？
三、算法题：（完全手撕，面试官布置了几个任务，一个个完成，核心是链表翻转）
四、智力题：一条绳子切三段，构成一个三角形的概率。
五、反问
③ 技术三面
一、自我介绍
二、边聊天边考察：
1. 你项目的创新点是什么？
2. 你觉得 DIEN 这篇论文写的怎么样？
（我想可能需要我有评判性思维，所以好坏都说了一些，尤其是我觉得有问题的着重说了一下，因

为 DIEN 是阿里妈妈团队的，当时觉得不能夸的太狠，毕竟是其他公司的论文，结果说完面试官反手说“哦，我叫 XXX， 这篇论文是我的成果”我直接当场社死(╯□╰))
3. 你为什么喜欢推荐系统呢？你对待工作和生活的态度是什么？
4. 我给你介绍一下我们部门的情况吧， blabla...
5. 我给你说一下我们现在在做的事情吧。
6. 我担心你的工程能力，你能说说你平时怎么积累工程能力的吗？你能说说序列模型是怎么在工程上高效实现的吗？
三、智力题：题目太长了，细节记不清楚了，是一道策略制定的题目。大概内容是，两个人要协作猜扑克，一个人抽五张牌，可以按照某种顺序与某种规律打出前四张，另一个人需要根据前四张牌的顺序和花色的排列组合猜出最后一张牌的花色和点数，牌组里不存在大小王。