算法大佬看了流泪，为什么这么好的CTR预估总结之前没分享(上篇)

原创 Eric陈健锋 炼丹笔记 昨天

收录于话题

#搜索推荐前沿算法 42 #搜索推荐基础知识 12

↑↑↑关注后"星标"**炼丹笔记**

炼丹笔记干货

作者：特邀资深炼丹师Eric陈健锋

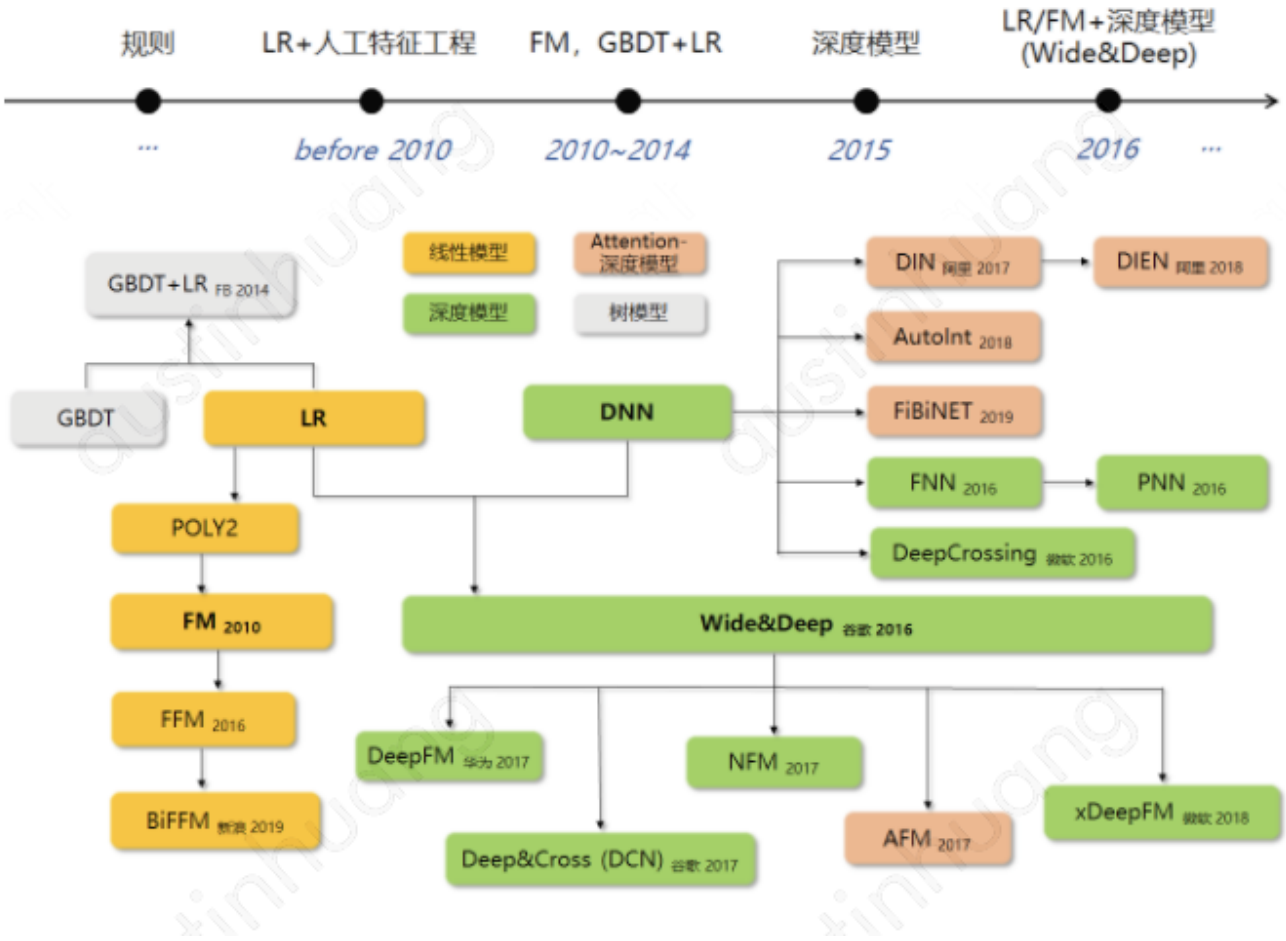
排版：十方导语

导语

在广告、推荐系统CTR预估问题上，早期的完全规则方法被过渡到以LR为代表的机器学习方法，为了充分发挥组合特征的价值，在相当长一段时间里，业界热衷于使用LR+人工特征工程。但人工组合特征成本高昂，在不同任务上也难以复用。2010年FM因子分解方法的出现解决了人工组合特征的困境，2014年Facebook提出的GBDT+LR也给出了一种利用树模型特点构建组合特征的思路。不过随着深度学习的崛起，2015年以后，借助非线性自动组合特征能力的深度模型，开始成为业内的主流。从经典DNN到结合浅层的Wide&Deep，用于CTR预估的深度模型在近些年间百花盛开，各种交叉特征建模方法层出不穷，Attention机制也从其他研究领域引入，帮助更好的适应业务，提升模型的解释性。在这进化路线之下，核心问题离不开解决数据高维稀疏难题，自动化组合特征，模型可解释。我们梳理了近些年CTR预估问题中有代表性的模型研究/应用成果，并对部分经典模型的实现原理进行详细剖析，落成文字作为学习过程的记录。

目录

00 CTR预估模型进化路线



01 从LR到FM/FFM

1.1 LR与多项式模型

LR（Logistic Regression）是机器学习中的一种线性分类模型，由于其简单、高效、易并行的特点，在实际生产环境中被广泛地应用。假设讨论互联网广告场景下的广告点击率预估问题，给定年龄、性别、教育程度、兴趣类目等用户侧特征，上下文特征，以及广告id，商品类型等广告侧特征，预测用户对于该广告的点击概率p(label=1 | user, context, ad)。针对特征的表示，通常做法是对各类特征进行one-hot编码，对于类目型或离散型数值特征很好理解，对于连续型数值特征，例如收入，一般会按区间分段方式将其离散化再进行one-hot编码。最后被表示成n维的0-1特征向量x，如式1，然后输入到分类模型。作为线性分类模型，LR实际上可拆解成线性回归和逻辑变换两部分。如式2，w是模型参数，利用w对特征向量x求加权和，得到回归值y，这是线性回归。然后再对y进行逻辑变换（sigmoid函数），如式3，得到点击广告（即label为1）的概率值。下文讲述的主要是关于公式2的迭代演进。

$$x = [\underbrace{0,0,1,0}_{\text{age}},\underbrace{0,1}_{\text{gender}},\underbrace{0,0,0,1,0,0,0}_{\text{education}},\ldots,\underbrace{0,1,0,1,0,0,}_{\text{interests}}\ldots,\underbrace{0,0,1,0,0}_{\text{product type}}] \quad (1)$$

$$\hat{y}(x) = w_0 + \sum_{i=1}^{n} w_i x_i \quad (2)$$

$$p(label = 1 \mid x) = sigmoid(\hat{y}(x)) = \frac{1}{1 + e^{-\hat{y}(x)}} \quad (3)$$

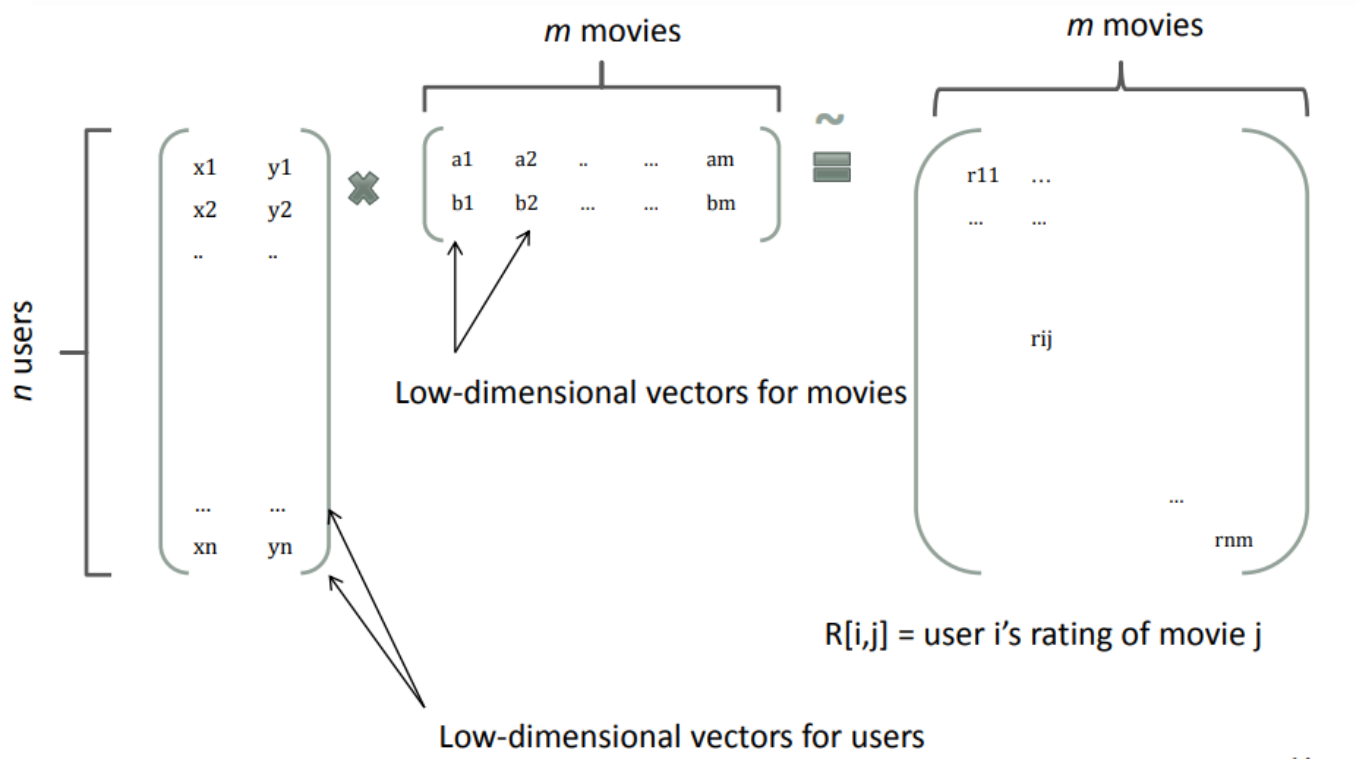
从式2来看，每一维特征是相互独立地作用在回归值上，即每一维特征与label的相关性是被单独衡量的。而根据实际样本数据观察，我们会发现对某些特征进行组合以后，组合特征会与label有更强的相关性。例如用户性别与广告商品类型的组合，[性别=“女性”，商品类型=“粉底液”]，或者[性别=“男性”，商品类型=“剃须刀”]，这些特征元组对广告的点击有更强的正相关信号。要引入这种组合特征信号，一种直观的想法是多项式模型，这里我们仅讨论二阶的情况，模型表达式如下

$$\hat{y}_{Poly2}(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} w_{ij} x_i x_j$$

特征xi和xj的两两组合用x_ix_j表示，可见只有当x_i和x_j都为非零值时，组合特征（或称交叉项）在多项式模型中才有意义。在真实业务场景下，样本特征通常高维稀疏，而组合特征xixj的稀疏程度则更为严重。在这种极度稀疏的情形下，模型很难准确地学习出参数w_{ij}。说通俗点，就是巧妇难为无米之炊。因为特征数据十分稀疏，模型在训练样本中从未见过有效的交叉项x_ix_j，无法学习这个交叉项和label之间的关系。而参数无法准确学习，必然会严重影响模型效果。

1.2 FM模型

在多项式模型里，高维稀疏数据使得交叉项参数学习非常困难，那么有没有可能通过低维稠密化解决？答案是肯定的，这就是矩阵分解的思想，我们借用CMU的FM课件[1]插图来简要说明



假设在电影推荐系统场景，系统只收集到少数的用户电影评分样本，R是用户对电影的评分矩阵，任务是预测用户i对各部电影j的评分值R_{ij}，然后为其推荐按评分值从高到低排序的前N部未观影过的电影。矩阵分解的思想是，假设我们可以用k个维度来完全描述一部电影的所有属性，那么每部电影都可以表示成一个k维向量，对于用户而言，对电影的每个维度会有不同的偏好程度，因此每个用户也可以表示成一个k维向量。k是系统超参数，以k=2作为示例，[a1, b1]是电影1的向量表示，[x1, y1]是用户1的向量表示，两个向量的点积结果r11是用户1对电影1的预测评分，而插图是这一过程的矩阵计算形式。可以看出，n*m的评分矩阵R被分解成两个矩阵的乘积，分解后的n*k用户矩阵和k*m电影矩阵是模型的参数，最后可通过最小化R_{ij}的均方误差进行求解。这里的k维向量又被称为隐向量，一般地k<<n，k<<m，于是我们通过低维稠密隐向量将用户的偏好和item（电影）的属性联系了起来，因此这种方法也被称为隐因子模型（Latent Factor Model）。

那么多项式模型的交叉项参数w_{ij}是否也可以通过隐向量分解来解决高维稀疏问题？这时候就是FM因子分解机（Factorization Machines）[2]的横空出世，它由Konstanz大学Steffen Rendle于2010年提出，模型方程如下

$$\hat{y}_{FM}(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle v_i, v_j \rangle x_i x_j$$

向量v_i，v_j分别是特征x_i，x_j对应的低维稠密隐向量，隐向量长度为k（k<<n），<·, ·>表示向量的点积运算。原来我们要求训练数据中交叉项xixj有足够多的共现样本时，模型参数wij才能有效学习，但这里将参数矩阵w通过隐向量分解后，问题就迎刃而解了，因为任意包含x_i或x_j的交叉项，都可用于参数向量vi或vj

的学习。例如样本1（地域=“北京”，商品类型=“羽绒服”），样本2（地域=“哈尔滨”，商品类型=“羽绒服”），可以同时用于优化“北京”、“哈尔滨”、“羽绒服”对应的特征隐向量表示。另外，模型的参数空间也从原始二项式模型的O(n^2)压缩到O(kn)。

直观上看，模型的计算复杂度是O(kn^2)，但经过如下对交叉项的优化，复杂度可以降低到O(kn)。

$$
\begin{aligned}
&\sum_{i=1}^{n}\sum_{j=i+1}^{n}\langle \mathbf{v}_i, \mathbf{v}_j\rangle x_i x_j \\
=&\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\langle \mathbf{v}_i, \mathbf{v}_j\rangle x_i x_j - \frac{1}{2}\sum_{i=1}^{n}\langle \mathbf{v}_i, \mathbf{v}_i\rangle x_i x_i \\
=&\frac{1}{2}\left(\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{f=1}^{k} v_{i,f}\, v_{j,f}\, x_i x_j - \sum_{i=1}^{n}\sum_{f=1}^{k} v_{i,f}\, v_{i,f}\, x_i x_i\right) \\
=&\frac{1}{2}\sum_{f=1}^{k}\left(\left(\sum_{i=1}^{n} v_{i,f}\, x_i\right)\left(\sum_{j=1}^{n} v_{j,f}\, x_j\right) - \sum_{i=1}^{n} v_{i,f}^2\, x_i^2\right) \\
=&\frac{1}{2}\sum_{f=1}^{k}\left(\left(\sum_{i=1}^{n} v_{i,f}\, x_i\right)^2 - \sum_{i=1}^{n} v_{i,f}^2\, x_i^2\right)
\end{aligned}
$$

经过化简，FM模型的最终表达式为

$$
\hat{y}(\mathbf{x}) = w_0 + \sum_{j=1}^{p} w_j x_j + \frac{1}{2}\sum_{f=1}^{k}\left[\left(\sum_{j=1}^{p} v_{j,f}\, x_j\right)^2 - \sum_{j=1}^{p} v_{j,f}^2\, x_j^2\right].
$$

梯度计算式为

$$
\frac{\partial}{\partial\theta}\hat{y}(\mathbf{x}) = \begin{cases} 1, & \text{if } \theta \text{ is } w_0 \\ x_i, & \text{if } \theta \text{ is } w_i \\ x_i \sum_{j=1}^{n} v_{j,f} x_j - v_{i,f} x_i^2, & \text{if } \theta \text{ is } v_{i,f} \end{cases}
$$

由于求和项 $\sum_{j=1}^{n} v_{j,f} x_j$ 与i无关，可以在每轮迭代中对所有的f提前计算，因此每个参数的梯度计算和更新可以在O(1)时间内完成。模型总共有kn+n+1个参数，所以模型训练的复杂度也是O(kn)。

可见，FM不仅通过矩阵分解的思想巧妙地解决了高维稀疏数据学习困难的问题，模型的参数空间以及模型的训练、预测时间都是线性复杂度的，是一种非常高效优雅的模型。

1.3 FFM模型

我们知道，不同特征是按域（field）来划分的，如下表的两个样本，“粉底液”、“剃须刀”属于“商品类型”field，“飞利浦”、“兰蔻”属于“品牌商”field，“男性”、“女性”属于“性别”field。在FM模型中，所有特征共享同一个隐空间（latent space），每一维特征xi仅被映射成一个隐向量vi去计算与其他特征的相互作用关系。这种做法实际上缺乏了对特征域信息的利用，以商品类型“粉底液”为例，FM模型的交叉项系数为<$v_{粉底液}$, $v_{兰蔻}$>，<$v_{粉底液}$, $v_{女性}$>，它使用了同一个隐向量$v_{粉底液}$去学习“粉底液”和“兰蔻”以及“粉底液”和“女性”的隐含关系。而“粉底液”对于品牌商域特征和对于性别域特征的作用关系可能是不一样的。

label（是否点击）	商品类型	品牌商	性别
1	粉底液	兰蔻	女性
0	剃须刀	飞利浦	女性

为了将不同特征域的差异信息显式引入到模型中，台湾大学阮毓钦（YuChin Juan）等人提出了域感知的改进版FM，称为FFM（Field-aware Factorization Machines）模型[3]。模型的数学形式如下

$$\hat{y}_{FFM}(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left\langle v_{i,f_j}, v_{j,f_i} \right\rangle x_i x_j$$

其中，f_j是特征x_j所属的field，v_i，f_j是特征x_i与特征x_j交叉时的隐向量。一次项和偏置项形式不变，以第一条样本为例，二次项在FM和FFM中的差异如下

FM的二次项：

$$<v_{粉底液}, v_{兰蔻}>\cdot 1\cdot 1 + <v_{粉底液}, v_{女性}>\cdot 1\cdot 1 + <v_{兰蔻}, v_{女性}>\cdot 1\cdot 1$$

FFM的二次项：

$$<v_{粉底液，品牌商}, v_{兰蔻，商品类型}>\cdot 1\cdot 1 + <v_{粉底液，性别}, v_{女性，商品类型}>\cdot 1\cdot 1 + <v_{兰蔻，性别}, v_{女性，品牌商}>\cdot 1\cdot 1$$

式中1为特征x_i（或x_j）的取值，在FM二次项中，每个特征仅有一个隐向量来表示，而FFM二次项中，每个特征有若干隐向量表示。可见，与FM相比，FFM划分了若干个隐空间，隐空间的数量等于field数量。在特

定的field下，特征被映射成不同的隐向量，这种特性被描述为Field-aware。反过来看，FM实际上等价于field数量为1的FFM，是FFM的特例。

FM

	k				
v_1	0.2	0.1	0.4	0.8	0.1
v_2	0.4	0.2	0.3	0.3	0.5
⋮	0.2	0.4	0.7	0.6	0.1
	0.5	0.7	0.4	0.8	0.4
v_n	0.3	0.1	0.5	0.4	0.2

FFM

	k				
$v_{1,f1}$	0.2	0.1	0.4	0.8	0.1
$v_{1,f2}$	0.4	0.2	0.3	0.3	0.5
⋮	0.2	0.4	0.7	0.6	0.1
$v_{2,f1}$	0.4	0.2	0.3	0.3	0.5
$v_{2,f2}$	0.2	0.4	0.7	0.6	0.1
⋮	0.5	0.7	0.4	0.8	0.4
⋮					
$v_{n,f1}$	0.2	0.4	0.7	0.6	0.1
$v_{n,f2}$	0.5	0.7	0.4	0.8	0.4
⋮	0.3	0.1	0.5	0.4	0.2

$field\ num$

假设隐向量的长度为k，特征fields数量为f，FFM的参数空间大小为fkn+n+1。由于二次项不能化简，FFM的训练和预测时间复杂度为O(kn^2)。

$$
\begin{aligned}
&\sum_{i=1}^{n}\sum_{j=i+1}^{n}\left\langle \mathrm{v}_{i,f_j}, \mathrm{v}_{j,f_i}\right\rangle x_i x_j \\
&= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\left\langle \mathrm{v}_{i,f_j}, \mathrm{v}_{j,f_i}\right\rangle x_i x_j - \frac{1}{2}\sum_{i=1}^{n}\left\langle \mathrm{v}_{i,f_i}, \mathrm{v}_{i,f_i}\right\rangle x_i x_i \\
&= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{f=1}^{k} v_{i,f_j,f} v_{j,f_i,f} x_i x_j - \frac{1}{2}\sum_{i=1}^{n}\sum_{f=1}^{k} v_{i,f_i,f} v_{i,f_i,f} x_i x_i \\
&= \frac{1}{2}\sum_{f=1}^{k}\left(\sum_{i=1}^{n}\sum_{j=1}^{n} v_{i,f_j,f} v_{j,f_i,f} x_i x_j - \sum_{i=1}^{n} v_{i,f_i,f}^2 x_i^2\right)
\end{aligned}
$$

最后，关于FM和FFM的介绍，美团技术团队的博客文章深入FFM原理与实践[5]，非常清晰通俗，是很好的阅读材料，入门进阶必看👆。

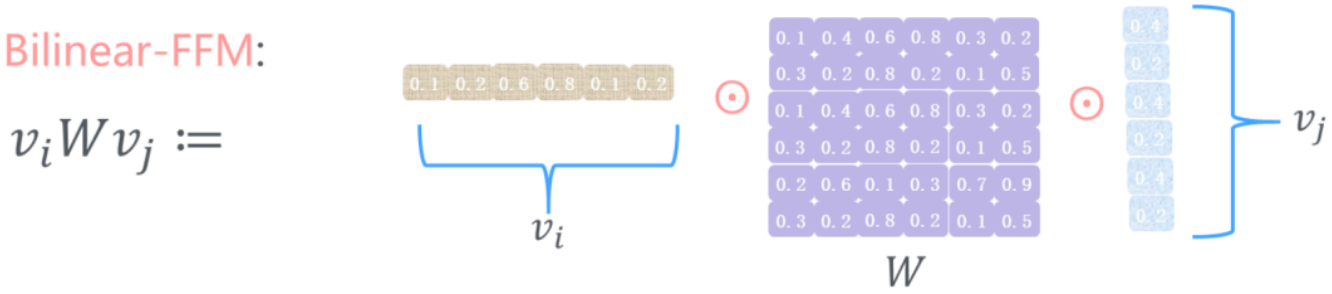
1.4 双线性FFM模型

FFM的二次项参数量是FM的fields数量倍，实际应用中面临内存消耗大的困境。为此，任职于新浪微博的张俊林博士提出了新的解决方案，称为双线性FFM模型[4]，模型方程如下

$$\hat{y}_{BiFFM}(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n}\sum_{j=i+1}^{n} v_i W v_j x_i x_j$$

双线性FFM的二次项和FM类似，仍然是将所有特征映射到同一隐空间。但不同于直接对隐向量做点积，为了增强模型的表达能力，双线性FFM在FM基础上增加了k×k参数矩阵W。这里有三种不同的类型，第一

种是所有特征共享一个矩阵W，引入的参数量为k×k；第二种是一个Field一个Wi，引入的参数量为f×k×k；第三种是一个Field组合一个Wij，引入的参数量为f×f×k×k。



下表是双线性FFM在Criteo和Avazu两个公开数据集上的对比实验，结果表明，在内存消耗远小于FFM的情况下，双线性FFM效果接近于FFM，第三种类型的效果优于第一、第二种类型。一般情况下，在加入LayerNorm后，效果还有一定的提升。

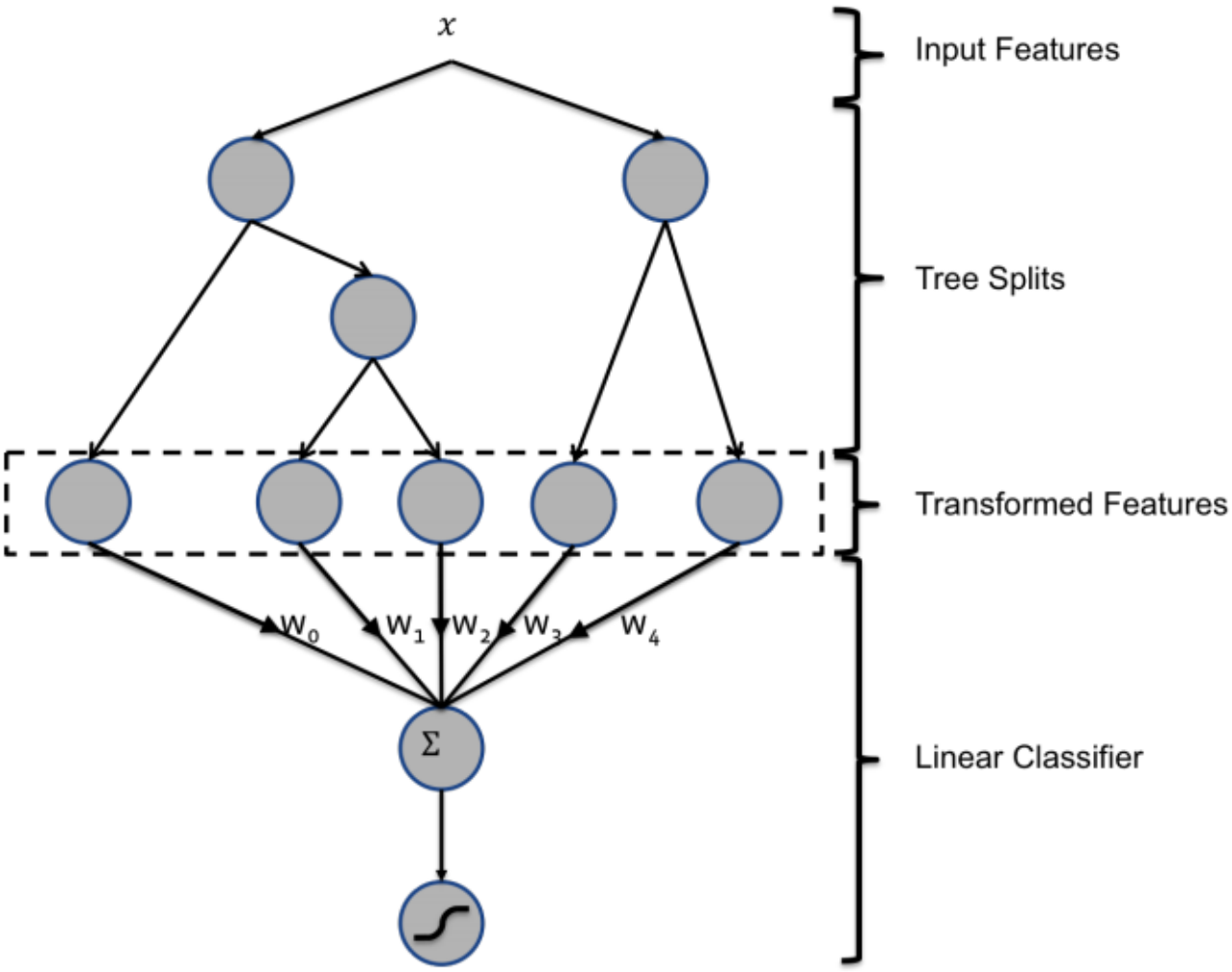
	Criteo		Avazu	
Model Name	AUC	Logloss	AUC	Logloss
LR	0.7808	0.4681	0.7633	0.3891
FM	0.7923	0.4584	0.7745	0.3832
FFM	0.8001	0.4525	**0.7795**	**0.3810**
Bi-FFM-ALL	0.7935	0.4573	0.7754	0.3830
Bi-FFM-EACH	0.7963	0.4550	0.7738	0.3838
Bi-FFM-INTER	0.7995	0.4525	**0.7781**	**0.3820**
Bi-FFM-ALL-LN	0.8004	0.4518	0.7763	0.3837
Bi-FFM-EACH-LN	0.8007	0.4511	0.7745	0.3843
Bi-FFM-INTER-LN	**0.8035**	**0.4484**	0.7765	0.3829

附：不同模型的参数规模和时间复杂度对比：

	参数规模	时间复杂度
LR	O(n)	O(n)
Poly2	O(n^2)	O(n^2)
FM	O(kn)	O(kn)
FFM	O(fkn)	O(kn^2)
双线性FFM	O(kn)*	暂未公开

* 假定特征fields数量f远小于n，k远小于n。

02 GBDT+LR



在FM/FFM之外，2014年Facebook提出的GBDT+LR[6]是实现用模型自动化特征工程的一项经典之作。GBDT+LR是级联模型，主要思路是先训练一个GBDT模型，然后利用训练好的GBDT对输入样本进行特征变换，最后把变换得到的特征向量作为LR模型的输入，对LR模型进行训练。举个具体例子，上图是由两颗子树构成的GBDT，将样本x输入，假设经过自上而下的节点判别，左边子树落入到第二个叶子节点，右边子树落入到第一个叶子节点，那么两颗子树分别得到向量[0, 1, 0]和[1, 0]，将各子树的输出向量进行concat，得到的[0, 1, 0, 1, 0]就是由GBDT变换得到的特征向量，最后将此向量作为LR模型的输入，训练LR的权重w0, w1，...。在此过程中，从根节点到叶子节点的遍历路径，就是一种特征组合，LR模型参数wi，对应一种特征组合的权重。

Facebook的论文实验证明了GBDT+LR有显著的提升，相比仅使用LR或Trees模型，loss降低了3%左右。

Table 1: Logistic Regression (LR) and boosted decision trees (Trees) make a powerful combination. We evaluate them by their Normalized Entropy (NE) relative to that of the Trees only model.

Model Structure	NE (relative to Trees only)
LR + Trees	96.58%
LR only	99.43%
Trees only	100% (reference)

虽然如此，GBDT+LR也有较明显的短板，例如，

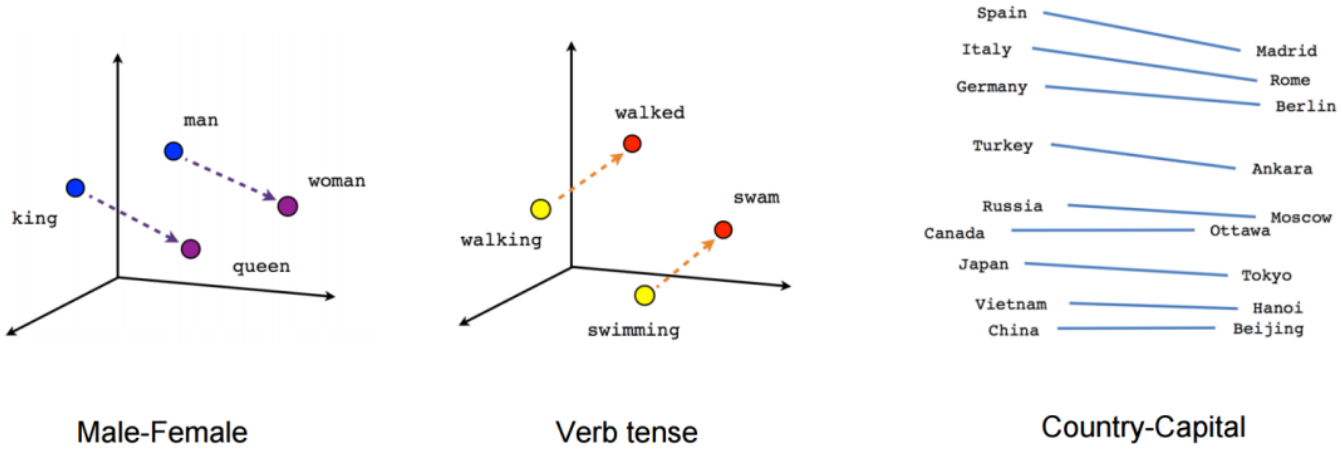
1）树模型在节点分裂时需要遍历特征集，通常不适用于海量离散ID类特征场景

2）引入级联模型误差，无法做end-to-end学习

03 深度模型

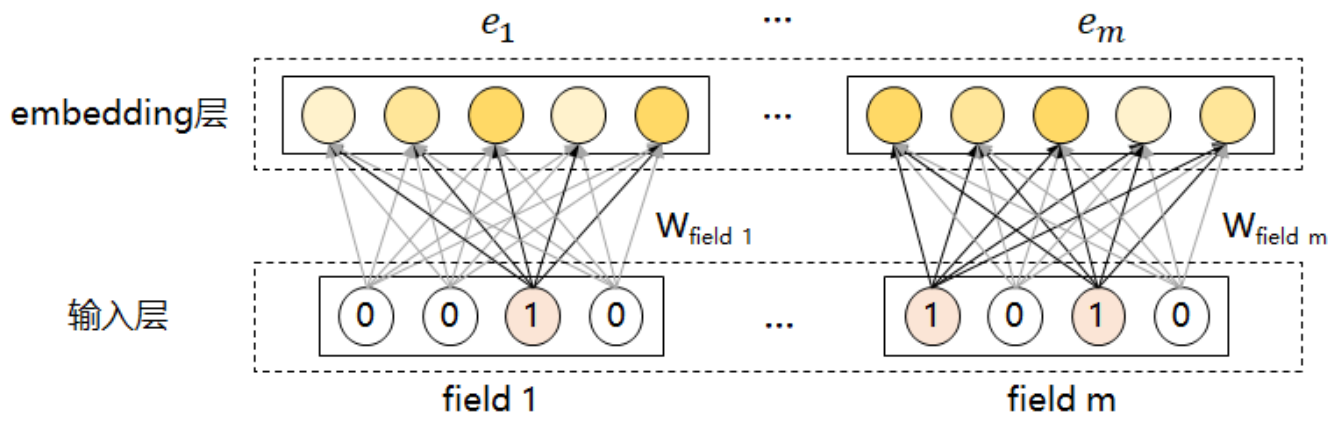
FM/FFM等变体模型，本质上是在解决高维稀疏数据的特征组合学习问题。解决问题的核心思路是，将高维空间转变成低维，在低维空间内寻找特征的稠密表示（这一过程也被称为“低维稠密化”），最后利用低维稠密向量的交互运算达到特征组合的目的。不过FM系列依然属于线性模型，对于非线性数据的拟合还略显不足，而GBDT+LR虽然是非线性建模，但它不适用于海量离线特征场景，也无法进行端到端学习。在实际业务中，我们通常面对的是非线性数据，同时离散id类特征也很重要，例如广告id，商品id等。因此，深度模型是另一种更好的解决方案。对非线性数据的拟合能力，一般体现在对高阶特征的建模能力上。深度模型通过全连接的多层感知器，能自动学习出特征之间有价值的高阶交叉信息，对高阶组合特征信号有很强的捕获力，同时非常适合进行端到端建模。这一节将从深度模型中的特征表示开始讲起，然后介绍业界经典的DNN网络结构，最后谈论宽深度模型的演变和显式高阶组合特征的设计原理。

3.1 特征的嵌入向量表示



Male-Female　　Verb tense　　Country-Capital

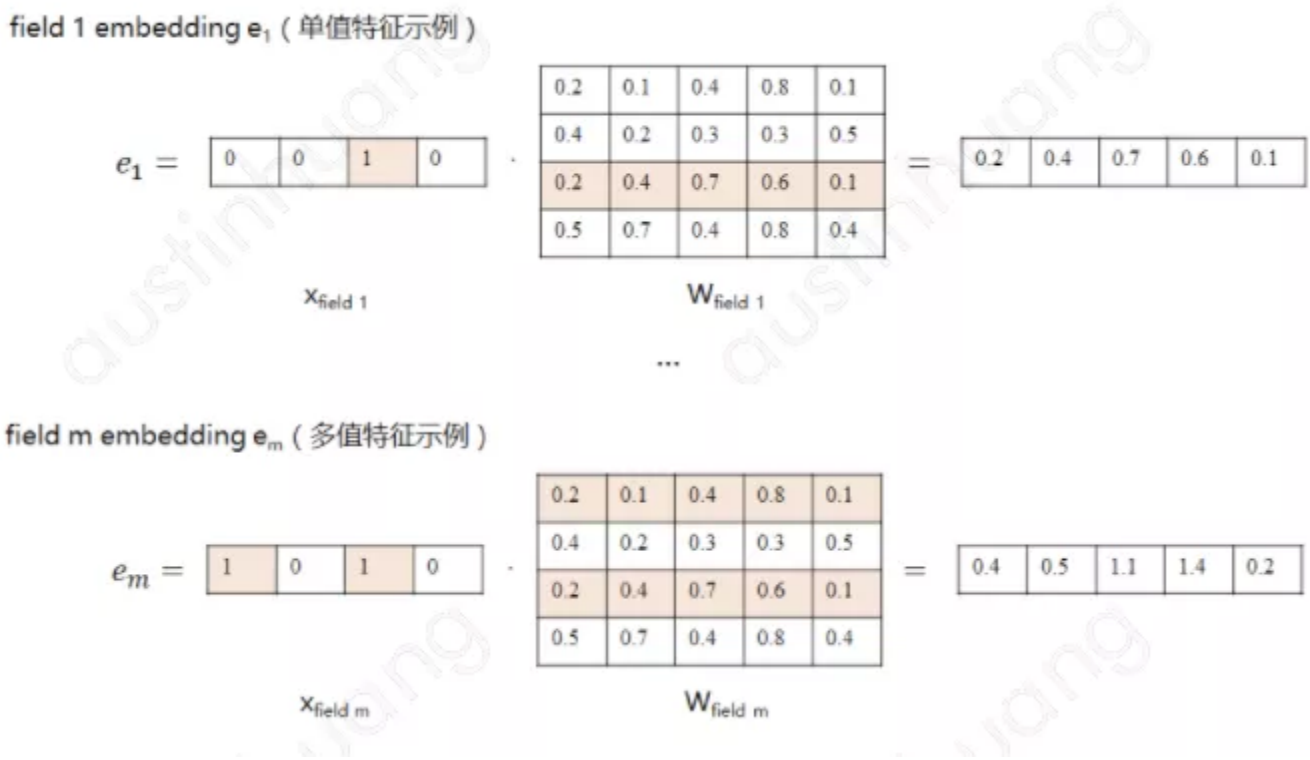
自2013年Google Mikolov的开山之作word2vec算法[7]提出以来，嵌入向量被广泛应用于自然语言处理领域用于词的表示。word2vec将自然语言中的单词从原来常规的one-hot表示变为低维稠密的词向量分布式表示（distributed representation，或通俗理解为低维实数向量），不仅有效降低了输入词维度，并且有趣的是经过训练后的词向量，在低维空间中有令人惊奇的几何关系，例如$V_{国王} \approx V_{男人} - V_{女人} + V_{皇后}$。这种分布式表示也被称作embedding（或嵌入向量），除了在自然语言处理上被大量使用，embedding也快速地在其他适用深度学习的领域或任务上（例如推荐系统、广告点击率预估等）流行，用于物品、用户或相关特征的表示。



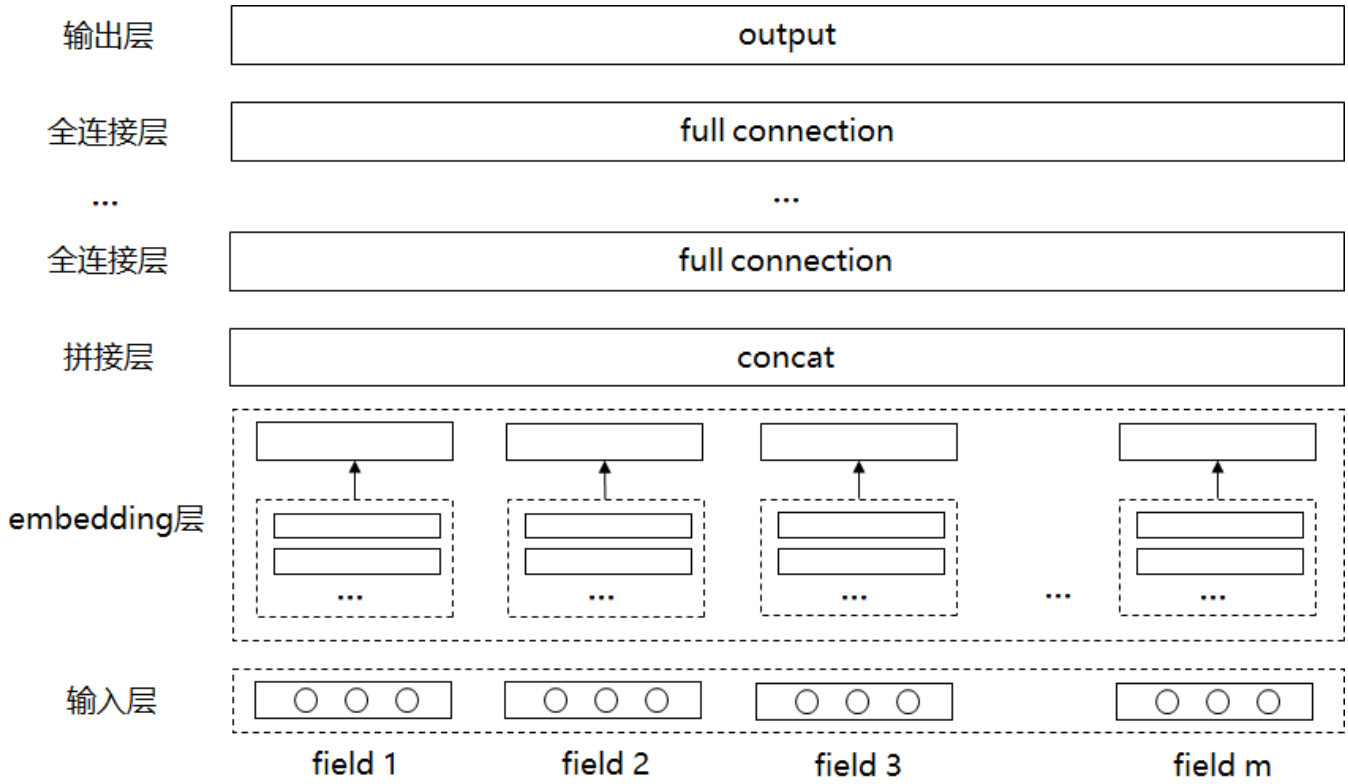
在神经网络模型中，将特征映射成embedding的操作（这种操作也叫做embedding lookup）可以看作是对输入层的原始one-hot（或多值特征对应的multi-hot）特征向量加上全连接线性变换，用数学符号可以表示成特征向量x和参数矩阵W的乘法运算。在点击率预估等任务上，一般以特征field的embedding作为神经网络的embedding层输出，对于单值特征，field embedding等于特征的embedding；对于多值特征，field embedding等于多个特征embedding的求和结果。当然，求和是对多值特征embedding的sum pooling操作，在具体应用上，也可以使用mean/max/k-max pooling代替。

在实际应用中，特征embedding已经基本成为解决高维稀疏特征问题的通用操作。它和FM系列模型中的特征隐向量是相似的，本质都是特征的低维稠密表示，只是在不同的模型中，低维稠密特征向量有不同的

应用形式。在FM模型中，取两两特征隐向量的点积作为交叉特征的权重；在深度模型中，将各特征field embedding拼接后输入到全连接层。



3.2 经典DNN网络框架

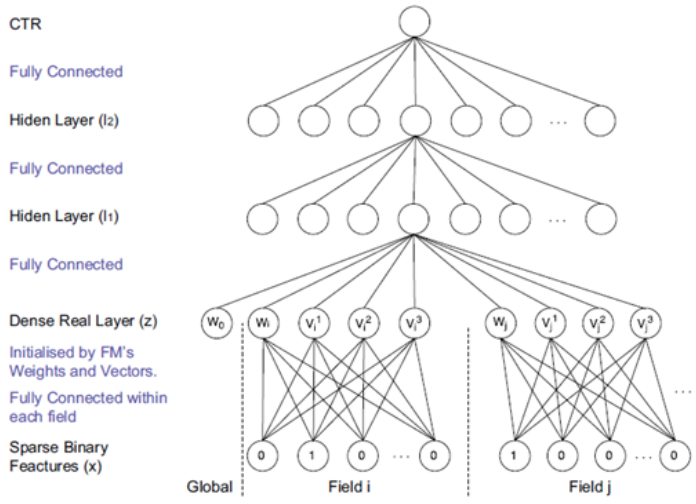


当前业界通用的深度神经网络模型结构，大抵可以划分为输入层、embedding层、拼接层、若干全连接层和输出层。输入层和embedding层，上文已经有过介绍，输入层是按特征field划分的one-hot（或multi-hot）离散特征向量表示，而图中的embedding层实际包含lookup和pooling两个操作。拼接层是对embedding层输出的m个长度分别为d_i（i=1,2,...,m）的稠密向量拼成一个长度为Σd_i的稠密向量。拼接后的向量经过若干全连接层（或称多层感知器MLP）后，最终在输出层得到回归值或分类概率值。经典DNN网络结构具有天然的自动学习交叉特征信息的能力，然而从特征embedding向量拼接和前向全连接的计算特点来看，这种能力更多是限于对隐式元素级（bit-wise）的交叉表示建模上。而经验上，特征间显式的向量级交叉信息（vector-wise）具有更直接的描述特征关系的能力，有助于使模型获得更强的交叉特征信号，降低模型学习难度。

3.3 DNN框架下的FNN、PNN与DeepCrossing模型

在经典DNN框架下，FNN、PNN以及DeepCrossing这三个是在推荐系统特征交叉建模方面比较引人关注的模型。FNN（Factorisation-machine supported Neural Networks）[8]和PNN（Product-based Neural Network）[9]是较早提出的引入了向量级交叉特征信息的模型，它们分别由Weinan Zhang和Yanru Qu于2016年提出。DeepCrossing模型[10]则是一种引入残差网络的改进DNN模型，由微软在2016年提出。

FNN模型



FNN

FNN使用了预训练的FM隐向量作为DNN第一层全连接层的输入，即左图中的z层，表示如下，

$$z = (w_0, z_1, z_2, ...z_i, ..., z_n)$$

$$z_i = W_0^i \cdot x[\text{start}_i : \text{end}_i] = (w_i, v_i^1, v_i^2, \ldots, v_i^K)$$

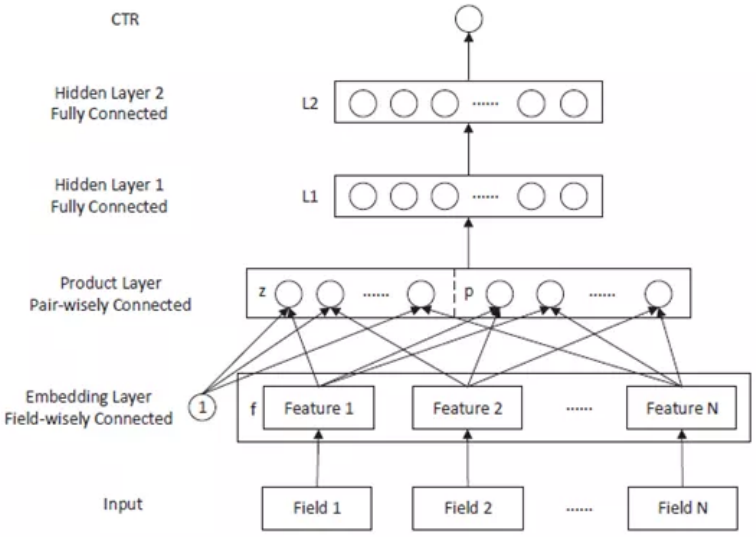
FM预训练

$$y_{\text{FM}}(x) := \text{sigmoid}\Big(w_0 + \sum_{i=1}^{N} w_i x_i + \sum_{i=1}^{N}\sum_{j=i+1}^{N} \langle v_i, v_j \rangle x_i x_j\Big)$$

$start_i$和end_i分别是第i个field的首末特征索引，W_0^i是$(K+1)\times(end_i - start_i + 1)$维的参数矩阵，由FM模型预训练得到。由于FM模型本身包含交叉特征隐向量点乘，因此这里认为FNN模型实际上引入了向量级交叉特征信息。对于上面的表达式，需要说明的是，FNN中的z_i下标i表示第i个特征field，而FM公式中的下标i表示第i个特征。FNN原文符号表示时假设每个特征field是单值的，但对于多值特征field，FNN模型仍然适用。

不过，由于FNN的初始embedding是由FM模型预训练得到的，这样的embedding初始化质量要取决于FM模型，引入了一些外部依赖。另外，FNN的z层向量输入到MLP全连接层，接受的是“加法”操作，而PNN论文作者认为这样的“加法”操作可能不如向量“乘法”操作能进一步的建模不同特征filed之间的局部关系。

PNN模型



PNN

PNN在embedding层和MLP全连接隐层之间增加了一个乘积层（product layer），用于更直接的建模两两特征的交互作用关系。乘积层包含z向量和p向量两部分，z向量由常数“1”向量和特征embedding相乘得到，因此z向量部分实际相当于特征embedding的直接拼接。p向量部分是PNN的核心，它是两两特征embedding进行“乘法”操作后的产物。论文作者给出了两种“乘法”操作，分别是内积型乘法和外积型乘法，使用不同“乘法”得到的变体模型被分别命名为IPNN（Inner Product-based Neural Network）和

OPNN（Outer Product-based Neural Network）。论文在两个公开数据集上对不同的模型进行了对比，从实验结果看，PNN模型，包括IPNN、OPNN、PNN*（将inner product和outer product进行拼接），效果都要优于FNN。另外也能看到，基于深度神经网络的模型，效果普遍优于LR、FM线性模型，这说明非线性高阶特征建模有很重要的提升作用。

TABLE I: Overall Performance on the Criteo Dataset.

Model	AUC	Log Loss	RMSE	RIG
LR	71.48%	0.1334	9.362e-4	6.680e-2
FM	72.20%	0.1324	9.284e-4	7.436e-2
FNN	75.66%	0.1283	9.030e-4	1.024e-1
CCPM	76.71%	0.1269	8.938e-4	1.124e-1
IPNN	**77.79%**	**0.1252**	**8.803e-4**	**1.243e-1**
OPNN	77.54%	0.1257	8.846e-4	1.211e-1
PNN*	77.00%	0.1270	8.988e-4	1.118e-1

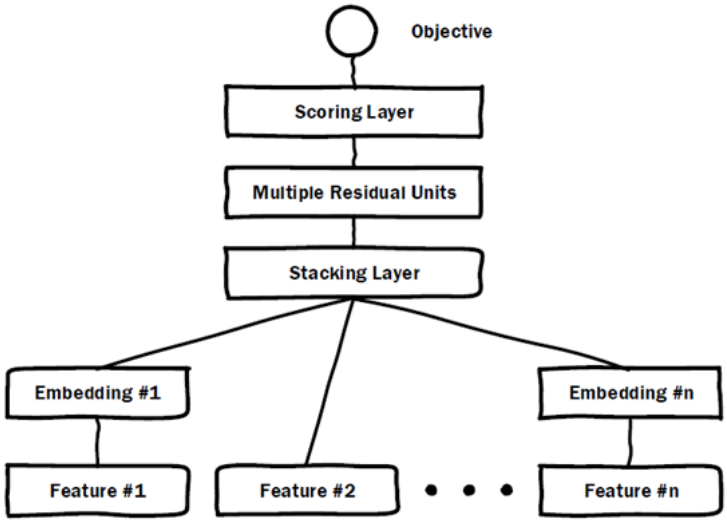
TABLE II: Overall Performance on the iPinYou Dataset.

Model	AUC	Log Loss	RMSE	RIG
LR	73.43%	5.581e-3	5.350e-07	7.353e-2
FM	75.52%	5.504e-3	5.343e-07	8.635e-2
FNN	76.19%	5.443e-3	5.285e-07	9.635e-2
CCPM	76.38%	5.522e-3	5.343e-07	8.335e-2
IPNN	79.14%	5.195e-3	4.851e-07	1.376e-1
OPNN	**81.74%**	5.211e-3	5.293e-07	1.349e-1
PNN*	76.61%	**4.975e-3**	**4.819e-07**	**1.740e-1**

注：CCPM（Convolutional Click Prediction Model）是基于卷积神经网络的点击率预测模型，CIKM 2015。

最后，PNN作者在论文中也提出了一些思考，他们认为乘积层实质上是一系列的加法和乘法操作，文中具体的内积型和外积型只是其中两种实施方案，实际上还可以定义其他更通用或更复杂的乘积层，以更充分的挖掘特征间的交互关系。以逻辑电路为例做类比，“加法”操作类似于逻辑“OR”，乘法操作类似于逻辑“AND”，而对于web应用领域，这些经过人为定义的用户、item类目型特征，相比像CV领域的图片像素等原始特征，有更高抽象的概念和含义，因此他们认为在特征交互中引入乘法操作，与加法相辅相成，能更有效的建模这些高抽象特征。

DeepCrossing模型



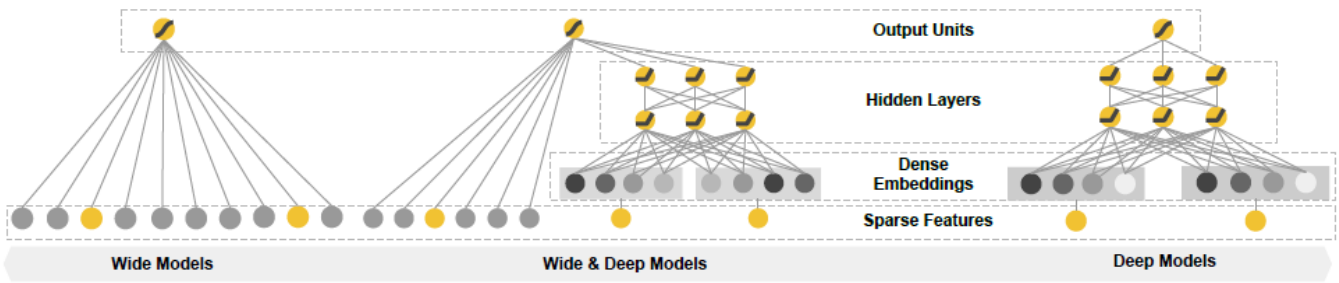
DeepCrossing

DeepCrossing模型结构如上图，Stacking层负责将各特征embedding拼接成一个向量，随后使用级联的多个残差单元来代替传统MLP。引入残差单元是DeepCrossing的主要改进点，好处是可以使用更深的网络层数，建模更高阶的特征，增强模型的表达能力。

FNN、PNN和DeepCrossing模型都是在经典DND框架下演化而来的，它们都有效的提高了交叉特征的建模能力。不过，这些模型都偏重于挖掘高阶交叉特征，对低阶交叉特征信息利用不足，下文介绍的Wide&Deep框架及其系列的衍生模型，将弥补这种缺陷，同时兼顾高阶和低阶交叉特征。

3.4 Wide&Deep框架及其衍生模型

对推荐系统的排序模型而言，定性地说，我们通常追求模型的记忆能力和泛化能力。一方面我们希望模型能准确记忆不同特征组合对预测目标的影响，以便依照已有的用户画像以及用户过去发生的行为使系统获得精准推荐能力，准确触达用户兴趣。另一方面，希望模型拥有一定的泛化能力，对未见过的或极少出现的特征组合，同样给出良好的预测结果，提高推荐内容的多样性。我们知道，线性模型以浅层形式直接学习稀疏组合特征权重，对训练数据中出现过的组合特征具有很好的记忆能力。而深度模型，稀疏特征被映射成低维稠密embedding向量，随后在深层全连接网络中获得充分交互，对具体的特征组合的“记忆”能力会减弱，但换来了更好的泛化效果。



基于这些想法，Google工程师在2016年提出了一种线性模型和深度模型的联合学习框架——Wide&Deep模型[11]，期望通过联合训练的方式，同时获得线性模型（Wide）的记忆能力和深度模型（Deep）的泛化能力，提高模型的预测准确性和多样性。Wide&Deep模型框架如上图（中间）所示，模型的左侧是输入单元和输出单元直接连接的线性模型，即方程形式是y=wx+b，输入单元接收的是未经嵌入的高维稀疏交叉特征向量。模型的右侧是一个DNN网络，输入层的稀疏特征先映射成低维稠密embedding，经过拼接后再馈入到隐层全连接网络进行前向计算。最终模型以左侧Wide部分输出和右侧Deep部分输出的加权和，作为预测事件发生的概率函数的变量，概率表达式如下

$$P(Y=1|\mathbf{x})=\sigma(\mathbf{w}_{wide}^{T}[\mathbf{x},\phi(\mathbf{x})]+\mathbf{w}_{deep}^{T}a^{(l_f)}+b)$$

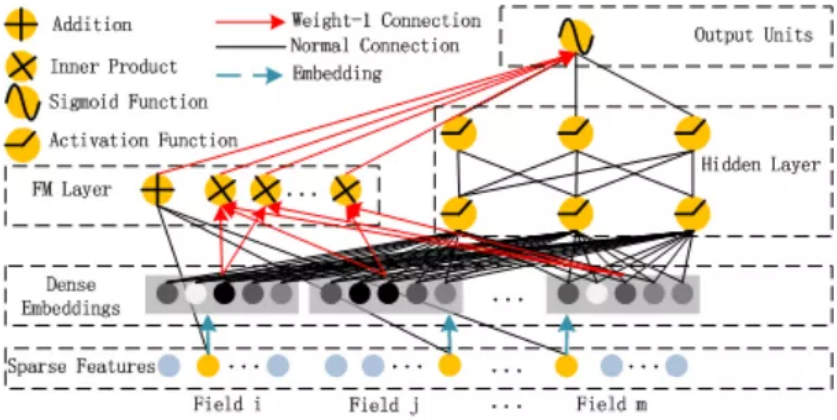
式中，$\phi(x)$表示原始特征x的交叉组合，$a^{(lf)}$是Deep部分最后一层的激活值。从表达式来看，如果去掉右侧的Deep部分，模型将退化成一个LR模型。整体上，Wide&Deep模型并不复杂，不过其提出的创新点更多在于Wide模型和Deep模型的联合训练上。相比传统的ensemble模型集成形式，即各模型独立训练并且每个模型通常需要足够大的参数规模以确保模型的表达能力，Wide&Deep模型中，Wide部分只需考虑加入少量的强交叉特征，用很少的参数来弥补Deep模型记忆力的不足，实现更好的效果。

Model	Offline AUC	Online Acquisition Gain
Wide (control)	0.726	0%
Deep	0.722	+2.9%
Wide & Deep	0.728	+3.9%

Wide&Deep模型在Google应用商店的App推荐业务上进行了实际测试，相比仅使用Wide模型，使用Wide&Deep模型App安装率相对提升了3.9%，提升幅度比仅使用Deep模型也高1%。除了在Google内部业务获得应用，Wide&Deep作为一种新的模型框架，也逐渐在业内流行。并且在原来基础上，Wide部分和Deep部分被不断优化改进，或者加入第三部分更精巧设计的子网络，衍生出各种不同变体模型。

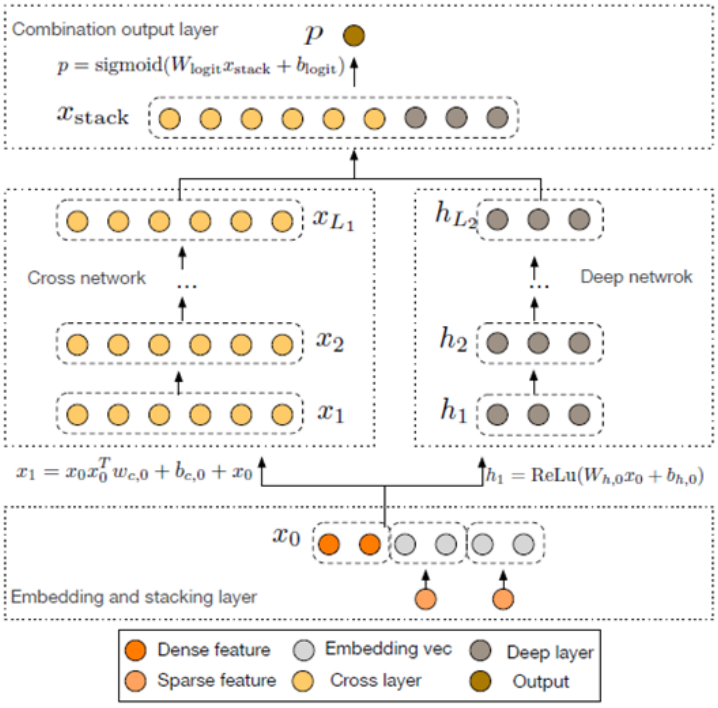
3.4.1 Wide部分的改进

衍生1：DeepFM模型



Wide&Deep模型中Wide部分和Deep部分的特征输入是不同的，Wide部分的输入还需要依赖人工特征工程来挑选有效特征，为了减少特征工程依赖，华为工程师在2017年提出了DeepFM模型[12]。DeepFM仍然由Wide部分和Deep部分构成，不过Wide部分使用了FM来构建，利用FM自动学习二阶交叉特征的能力，代替需要人工交叉特征的LR。另外，Wide部分和Deep部分的底层输入特征以及特征embedding向量完全共享，能实现端到端学习。

衍生2：Deep&Cross（DCN）模型



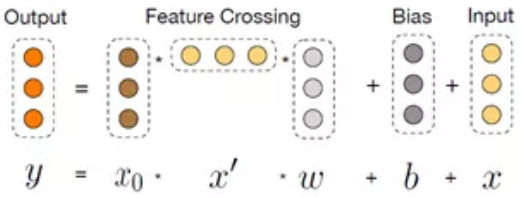
除了用FM代替LR以外，2017年发表的Deep&Cross Network（DCN）模型[13]，提出了一种新的Cross网络来建模Wide部分，可以用更高效的方式实现更高阶的交叉。

$$p = sigmoid\underbrace{\left(W_{logit} x_{stack} + b_{logit}\right)}_{g_l(\mathrm{x}_0) + deep(\mathrm{x}_0)}$$

根据DCN的模型结构图，我们可以把DCN的输出端点，即sigmoid节点的输入拆解成三个部分：由cross网络贡献的$g_l(x_0)$（l为交叉网络的层数），由deep网络贡献的$deep(x_0)$和偏置项b_{logit}。这里的$deep(x_0)$和偏置项b_{logit}不再说明，我们主要关心cross网络$g_l(x_0)$是如何高效实现高阶交叉的。下面我们先来看cross网络的特征交叉是怎么实现的，然后再看为什么高效。

- 怎么实现？

在cross网络中，一个cross layer的运算操作如下，



为了简化计算，参考论文分析省略常数项，我们讨论下面的交叉迭代公式

$$\mathrm{x}_{i+1} = \mathrm{x}_0 {\mathrm{x}_i}^T \mathrm{w}_i + \mathrm{x}_i$$

约定符号x_j^i，w_j^i的下标i表示第i层，上标j表示向量的元素索引，假设经过embedding拼接后的特征向量维度为3，在这种约定下，第一层的输入向量和参数向量分别为$x_0=[x_0^1, x_0^2, x_0^3]$，$w_0=[w_0^1, w_0^2, w_0^3]$。根据上式，不难发现，对于层数为l的cross网络，可以实现的最高交叉阶数为l+1。这里给出2阶cross（l=1）和3阶cross（l=2）的计算示例

2阶cross

$$\begin{aligned}
\mathrm{x}_1 &= \mathrm{x}_0 {\mathrm{x}_0}^T \mathrm{w}_0 + \mathrm{x}_0 \\
&= \begin{bmatrix} x_0^1 \\ x_0^2 \\ x_0^3 \end{bmatrix} \begin{bmatrix} x_0^1 & x_0^2 & x_0^3 \end{bmatrix} \begin{bmatrix} w_0^1 \\ w_0^2 \\ w_0^3 \end{bmatrix} + \begin{bmatrix} x_0^1 \\ x_0^2 \\ x_0^3 \end{bmatrix} \\
&= \begin{bmatrix} x_0^1 x_0^1 & x_0^2 x_0^1 & x_0^3 x_0^1 \\ x_0^1 x_0^2 & x_0^2 x_0^2 & x_0^3 x_0^2 \\ x_0^1 x_0^3 & x_0^2 x_0^3 & x_0^3 x_0^3 \end{bmatrix} \begin{bmatrix} w_0^1 \\ w_0^2 \\ w_0^3 \end{bmatrix} + \begin{bmatrix} x_0^1 \\ x_0^2 \\ x_0^3 \end{bmatrix} \\
&= \begin{bmatrix} w_0^1 x_0^1 x_0^1 + w_0^2 x_0^2 x_0^1 + w_0^3 x_0^3 x_0^1 + x_0^1 \\ w_0^1 x_0^1 x_0^2 + w_0^2 x_0^2 x_0^2 + w_0^3 x_0^3 x_0^2 + x_0^2 \\ w_0^1 x_0^1 x_0^3 + w_0^2 x_0^2 x_0^3 + w_0^3 x_0^3 x_0^3 + x_0^3 \end{bmatrix}
\end{aligned}$$

$$\begin{aligned}
g_l(\mathrm{x}_0) &= {\mathrm{x}_1}^T \mathrm{w}_1 \\
&= \begin{bmatrix} w_0^1 x_0^1 x_0^1 + w_0^2 x_0^2 x_0^1 + w_0^3 x_0^3 x_0^1 + x_0^1 \\ w_0^1 x_0^1 x_0^2 + w_0^2 x_0^2 x_0^2 + w_0^3 x_0^3 x_0^2 + x_0^2 \\ w_0^1 x_0^1 x_0^3 + w_0^2 x_0^2 x_0^3 + w_0^3 x_0^3 x_0^3 + x_0^3 \end{bmatrix} \begin{bmatrix} w_1^1 \\ w_1^2 \\ w_1^3 \end{bmatrix} \\
&= w_1^1 w_0^1 x_0^1 x_0^1 + w_1^1 w_0^2 x_0^2 x_0^1 + w_1^1 w_0^3 x_0^3 x_0^1 + w_1^1 x_0^1 \\
&\quad + w_1^2 w_0^1 x_0^1 x_0^2 + w_1^2 w_0^2 x_0^2 x_0^2 + w_1^2 w_0^3 x_0^3 x_0^2 + w_1^2 x_0^2 \\
&\quad + w_1^3 w_0^1 x_0^1 x_0^3 + w_1^3 w_0^2 x_0^2 x_0^3 + w_1^3 w_0^3 x_0^3 x_0^3 + w_1^3 x_0^3
\end{aligned}$$

3阶cross

$$
\begin{aligned}
x_2 &= x_0 x_1^T w_1 + x_1 \\
&= \begin{bmatrix} x_0^1 \\ x_0^2 \\ x_0^3 \end{bmatrix} \begin{bmatrix} x_1^1 & x_1^2 & x_1^3 \end{bmatrix} \begin{bmatrix} w_1^1 \\ w_1^2 \\ w_1^3 \end{bmatrix} + \begin{bmatrix} x_1^1 \\ x_1^2 \\ x_1^3 \end{bmatrix} \\
&= \begin{bmatrix} x_1^1 x_0^1 & x_1^2 x_0^1 & x_1^3 x_0^1 \\ x_1^1 x_0^2 & x_1^2 x_0^2 & x_1^3 x_0^2 \\ x_1^1 x_0^3 & x_1^2 x_0^3 & x_1^3 x_0^3 \end{bmatrix} \begin{bmatrix} w_1^1 \\ w_1^2 \\ w_1^3 \end{bmatrix} + \begin{bmatrix} x_1^1 \\ x_1^2 \\ x_1^3 \end{bmatrix} \\
&= \begin{bmatrix} w_1^1 x_1^1 x_0^1 + w_1^2 x_1^2 x_0^1 + w_1^3 x_1^3 x_0^1 + x_1^1 \\ w_1^1 x_1^1 x_0^2 + w_1^2 x_1^2 x_0^2 + w_1^3 x_1^3 x_0^2 + x_1^2 \\ w_1^1 x_1^1 x_0^3 + w_1^2 x_1^2 x_0^3 + w_1^3 x_1^3 x_0^3 + x_1^3 \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
g_l(x_0) &= x_2^T w_2 \\
&= \begin{bmatrix} w_1^1 x_1^1 x_0^1 + w_1^2 x_1^2 x_0^1 + w_1^3 x_1^3 x_0^1 + x_1^1 \\ w_1^1 x_1^1 x_0^2 + w_1^2 x_1^2 x_0^2 + w_1^3 x_1^3 x_0^2 + x_1^2 \\ w_1^1 x_1^1 x_0^3 + w_1^2 x_1^2 x_0^3 + w_1^3 x_1^3 x_0^3 + x_1^3 \end{bmatrix} \begin{bmatrix} w_2^1 \\ w_2^2 \\ w_2^3 \end{bmatrix} \\
&= w_2^1 w_1^1 x_1^1 x_0^1 + w_2^1 w_1^2 x_1^2 x_0^1 + w_2^1 w_1^3 x_1^3 x_0^1 + w_2^1 x_1^1 \\
&\quad + w_2^2 w_1^1 x_1^1 x_0^2 + w_2^2 w_1^2 x_1^2 x_0^2 + w_2^2 w_1^3 x_1^3 x_0^2 + w_2^2 x_1^2 \\
&\quad + w_2^3 w_1^1 x_1^1 x_0^3 + w_2^3 w_1^2 x_1^2 x_0^3 + w_2^3 w_1^3 x_1^3 x_0^3 + w_2^3 x_1^3 \\
&= w_2^1 w_1^1 (w_0^1 x_0^1 x_0^1 + w_0^2 x_0^2 x_0^1 + w_0^3 x_0^3 x_0^1 + x_0^1) x_0^1 \\
&\quad + w_2^1 w_1^2 (w_0^1 x_0^1 x_0^2 + w_0^2 x_0^2 x_0^2 + w_0^3 x_0^3 x_0^2 + x_0^2) x_0^1 \\
&\quad + w_2^1 w_1^3 (w_0^1 x_0^1 x_0^3 + w_0^2 x_0^2 x_0^3 + w_0^3 x_0^3 x_0^3 + x_0^3) x_0^1 \\
&\quad + w_2^1 (w_0^1 x_0^1 x_0^1 + w_0^2 x_0^2 x_0^1 + w_0^3 x_0^3 x_0^1 + x_0^1) \\
&\quad + w_2^2 w_1^1 (w_0^1 x_0^1 x_0^1 + w_0^2 x_0^2 x_0^1 + w_0^3 x_0^3 x_0^1 + x_0^1) x_0^2 \\
&\quad + w_2^2 w_1^2 (w_0^1 x_0^1 x_0^2 + w_0^2 x_0^2 x_0^2 + w_0^3 x_0^3 x_0^2 + x_0^2) x_0^2 \\
&\quad + w_2^2 w_1^3 (w_0^1 x_0^1 x_0^3 + w_0^2 x_0^2 x_0^3 + w_0^3 x_0^3 x_0^3 + x_0^3) x_0^2 \\
&\quad + w_2^2 (w_0^1 x_0^1 x_0^2 + w_0^2 x_0^2 x_0^2 + w_0^3 x_0^3 x_0^2 + x_0^2) \\
&\quad + w_2^3 w_1^1 (w_0^1 x_0^1 x_0^1 + w_0^2 x_0^2 x_0^1 + w_0^3 x_0^3 x_0^1 + x_0^1) x_0^3 \\
&\quad + w_2^3 w_1^2 (w_0^1 x_0^1 x_0^2 + w_0^2 x_0^2 x_0^2 + w_0^3 x_0^3 x_0^2 + x_0^2) x_0^3 \\
&\quad + w_2^3 w_1^3 (w_0^1 x_0^1 x_0^3 + w_0^2 x_0^2 x_0^3 + w_0^3 x_0^3 x_0^3 + x_0^3) x_0^3 \\
&\quad + w_2^3 (w_0^1 x_0^1 x_0^3 + w_0^2 x_0^2 x_0^3 + w_0^3 x_0^3 x_0^3 + x_0^3)
\end{aligned}
$$

从$g_l(x0)$的展开式，可以更直观的看到每一个交叉项$x_0^i x_0^j x_0^k$（通常的写法是$x_i x_j x_k$，下文回归这种写法）最终都被分配到一个关于参数集$\{w_m^i\}_{m=0\sim l}$、$\{w_m^j\}_{m=0\sim l}$、$\{w_m^k\}_{m=0\sim l}$乘积和的系数。以l=2时的交叉项$x_1x_2x_3$为例，它的系数是

$$
w_2^1 w_1^2 w_0^3 + w_2^1 w_1^3 w_0^2 + w_2^2 w_1^1 w_0^3 + w_2^2 w_1^3 w_0^1 + w_2^3 w_1^1 w_0^2 + w_2^3 w_1^2 w_0^1
$$

（4）

对于任意阶数的情况，$g_l(x0)$可以表示成如下形式

$$
\left\{ \sum_{\alpha} c_{\alpha}(w_0, \ldots, w_l) x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_d^{\alpha_d} \;\middle|\; 0 \le |\alpha| \le l+1, \alpha \in \mathbb{N}^d \right\}
$$

（5）

式中，x_1，x_2，…，x_d表示d维的输入特征向量，即$x_0=[x_1, x_2, ..., x_d]$；α_1，α_2，…，α_d分别是对应特征项的幂数，$|\alpha|=\Sigma\alpha_i$；$c_\alpha(w_0, ..., w_l)$是交叉项$x_1^{\alpha_1}x_2^{\alpha_2}...x_d^{\alpha_d}$的系数。DCN论文给出了系数表达式$c_\alpha(w_0, ..., w_l)$的详细数学推导，本文不再深入讨论。这里简单以交叉项$x_1x_2x_3$为例，系数式（4）可以形式化为 $c_\alpha = \sum_{i,j,k \in P_\alpha} w_0^{(i)} w_1^{(j)} w_2^{(k)}$ ，其中P_α是特征索引（1, 2, 3）的全排列集合。

至此可见，cross网络是以元素级bit-wise方式来建模高阶交叉特征，并且不同于dnn网络，cross网络的高阶交叉是显式的。论文还指出，cross网络实际上是n阶多元多项式的一种高效实现。

- 为什么高效？

用$P_n(x)$表示n阶多元多项式（multivariate polynomial），

$$P_n(\mathbf{x}) = \left\{ \sum_{\boldsymbol{\alpha}} w_{\boldsymbol{\alpha}} x_1^{\alpha_1} x_2^{\alpha_2} \dots x_d^{\alpha_d} \;\middle|\; 0 \le |\boldsymbol{\alpha}| \le n, \boldsymbol{\alpha} \in \mathbb{N}^d \right\}$$

（6）

式中，w_{α}是交叉项的权重系数，对于d维特征，要实现最高n阶的交叉，需要的参数空间为$O(d^n)$，是n次方阶复杂度。

观察可知，cross网络$g_i(x0)$，式（5），和$P_n(x)$，式（6），表达的是同样的多项式，只是交叉项的系数不一样。我们现在来看，cross网络$g_i(x0)$实现的n阶多元多项式需要多大的参数规模呢？前文说了，实现n阶需要的网络层数为n-1，对于d维特征，cross网络需要的参数数量为d×(n-1)×2，因此空间复杂度为O(d)，只需要线性复杂度。

cross网络参数复杂度的大幅降低，其实是得益于参数共享的思想，下面参考原著和FM作类比。在FM模型中，特征xi用隐向量vi来表示，交叉项xixj的权重由内积<vi, vj>计算。而cross网络中，特征xi和参数集$\{w_k^i\}_{k=0\sim l}$关联，交叉项xixj的权重由参数集

$\{w_k^i\}_{k=0\sim l}$和$\{w_k^j\}_{k=0\sim l}$计算。这二者背后都是参数共享思想，减小参数规模，同时解决数据稀疏带来的权重学习困难问题，使模型学习更有效。

最后，实际上cross网络的时间和空间复杂度都是关于输入维度d的线性量，相对deep部分并没有增加额外的复杂度，因此DCN模型在整体效率上是可以比拟传统DNN模型的。

3.4.2 Deep部分的改进

衍生3：NFM/AFM模型

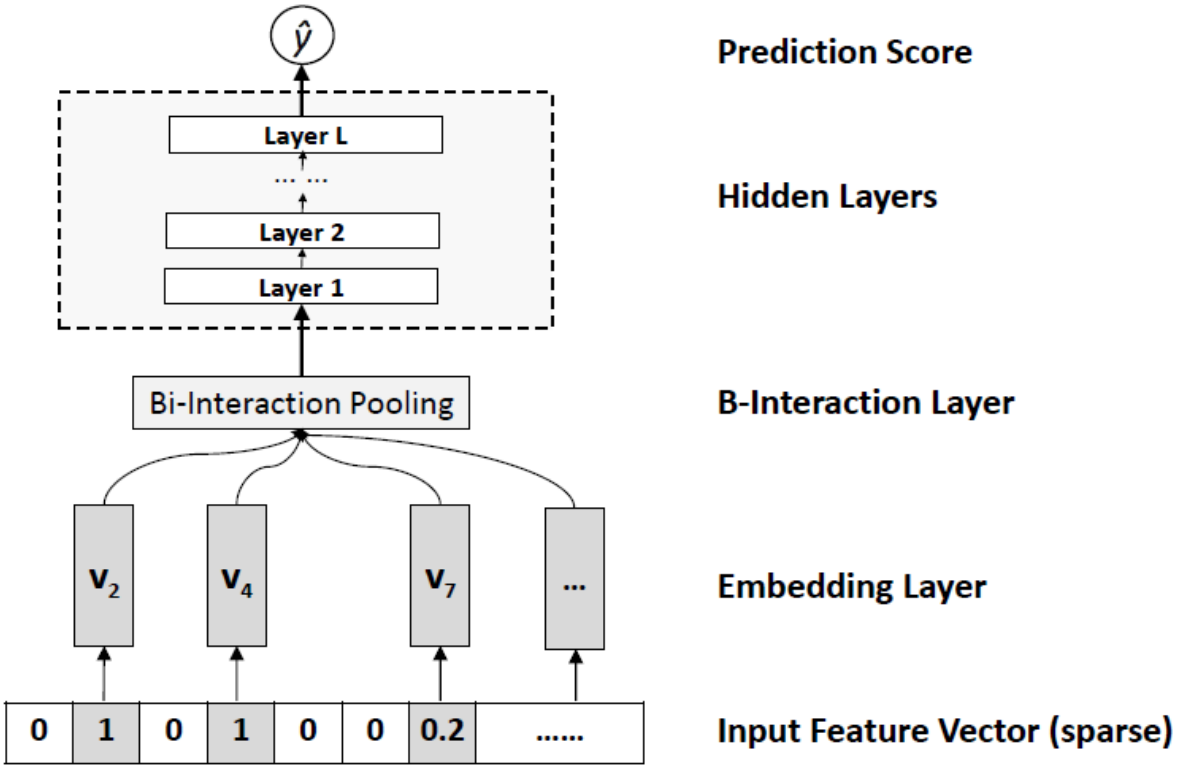


Figure 2: Neural Factorization Machines model (the first-order linear regression part is not shown for clarity).

对Deep部分的改进，Neural Factorization Machines（NFM）模型[14]是一个比较有效的方案，由Xiangnan He等人于2017年提出。NFM作者认为，经典DNN框架中的concat层只是把各特征embedding拼接成一个向量，并没有在底层利用好特征交互信息，只能依赖后面的MLP全连接层来建模特征之间的相互关系。但由于深度网络有梯度消失、梯度爆炸和过拟合等问题，DNN并不是很好优化，所以最终会导致模型效果有损失。所以NFM的主要改进点是，引入Bi-Interaction Pooling层，替代经典DNN的concat层，在底层增加足够的特征交互信息后，再馈入到MLP网络做进一步的高阶非线性建模。

NFM改进后的Deep部分如上图所示，Bi-Interaction Pooling算子定义如下

$$f_{BI}(\mathcal{V}_x) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} x_i \mathbf{v}_i \odot x_j \mathbf{v}_j,$$

其中$V_x = \{x_1v_1, ..., x_nv_n\}$表示特征的embedding集合，符号“⊙”表示向量的哈达玛积，即$[a_1, a_2, a_3] \odot [b_1, b_2, b_3] = [a_1b_1, a_2b_2, a_3b_3]$。Bi-Interaction Pooling将两两特征embedding做哈达玛积交互后，再通过求和的方式压缩成一个向量。经过化简后，上式的计算复杂度可以从$O(kn^2)$优化到$O(kn)$：

$$f_{BI}(\mathcal{V}_x) = \frac{1}{2}\left[(\sum_{i=1}^{n} x_i \mathbf{v}_i)^2 - \sum_{i=1}^{n}(x_i \mathbf{v}_i)^2\right]$$

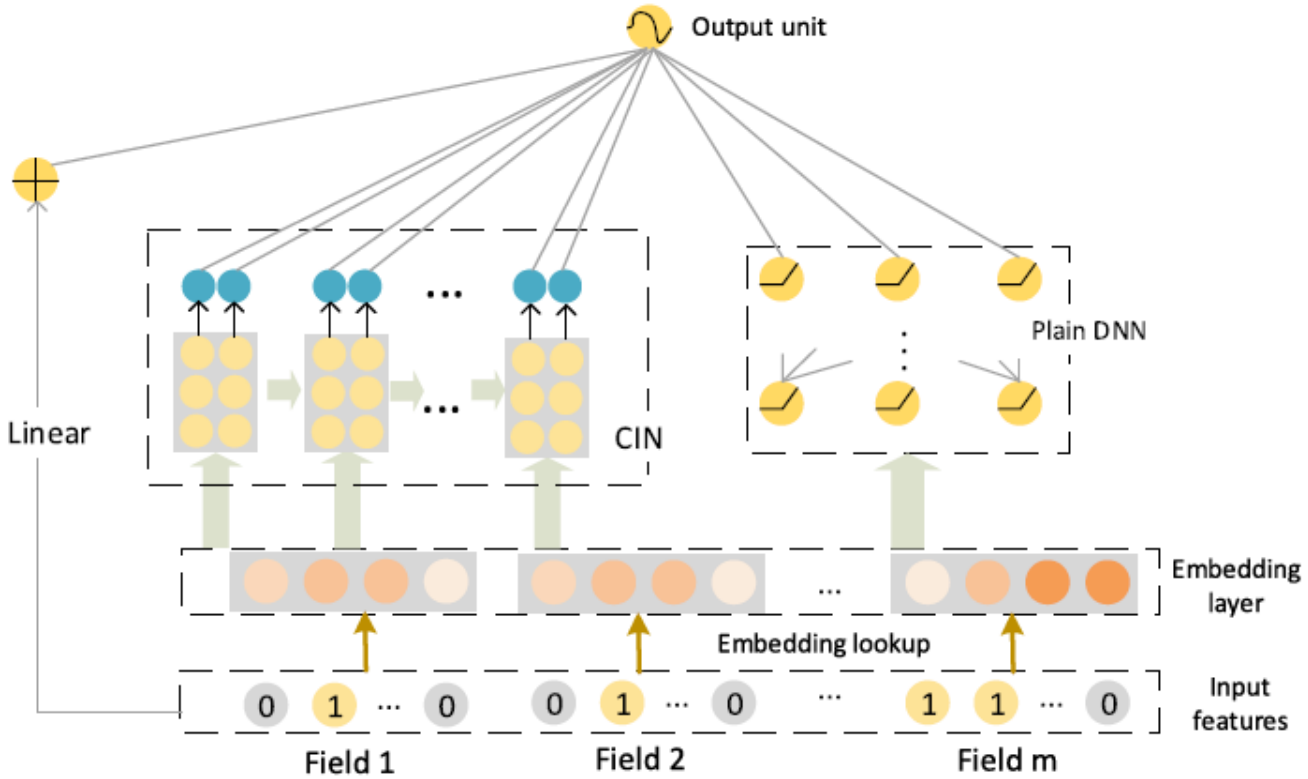
加入Wide线性部分后，NFM模型的完整表达式为

$$\hat{y}_{NFM}(\mathbf{x}) = w_0 + \sum_{i=1}^{n} w_i x_i + \mathbf{h}^T \sigma_L(\mathbf{W}_L(\ldots\sigma_1(\mathbf{W}_1 f_{BI}(\mathcal{V}_x) + \mathbf{b}_1)\ldots) + \mathbf{b}_L),$$

介绍完NFM，我们想在此处先简单地把另一个也是对Deep部分进行改进的模型——AFM模型（Attentional Factorization Machines）[15]也引出场。这两个模型出自同一个团队，AFM模型名字带有“Attentional”，给人的直觉AFM应该是在NFM的基础上改进的，但实际上这两个模型是各有所长的。不同于NFM将Bi-Interaction Pooling两两向量交互后直接（等权）求和压缩成一个向量，AFM引入了attention权重因子来对交互后的向量进行加权求和压缩，增强了二阶交叉的表达能力。不过，压缩后的向量不再经过MLP，而是直接一层full connection输出到prediction score，这就将模型限制在只表达二阶交叉特征，缺少更高阶信息的建模。尽管如此，AFM对特征交互的“attention-based”pooling思路仍然非常有借鉴意义和应用价值，作为基于注意力机制模型的一种，我们将在下一部分“2.3 引入注意力机制：提高模型自适应能力与可解释性”，再展开对AFM的介绍。

3.4.3 引入新的子网络

衍生4：xDeepFM模型



xDeepFM模型[16]，全称eXtreme Deep Factorization Machine，是微软研究人员在Deep&Cross（DCN）模型基础上进行研究和改进后提出的。xDeepFM的主要创新点是，在传统Wide部分和Deep部分之外，提出了一种新的特征交叉子网络，Compressed Interaction Network（CIN），用于显式地以向量级vector-wise方式建模高阶交叉特征。上图是xDeepFM的模型网络结构，模型输出节点的数学表达式为

$$\hat{y} = \sigma(\mathbf{w}_{linear}^T \mathbf{a} + \mathbf{w}_{dnn}^T \mathbf{x}_{dnn}^k + \mathbf{w}_{cin}^T \mathbf{p}^+ + b)$$

式中，a为原始离散特征向量（raw feature），x_{dnn}^k、p^+分别是DNN和CIN模块的输出。不难得知，由于包含了线性模块（Wide）、DNN模块（Deep）和CIN模块，xDeepFM模型是同时具备显式和隐式特征交叉的建模能力的。不过这里同样不再介绍线性和DNN模块，下文主要讨论子网络CIN的由来、实现方式和特点，以及论文的实验效果。

xDeepFM论文认为，Deep&Cross中的cross网络存在两点不足，1）cross网络是限定在一种特殊形式下实现高阶交叉特征的，即cross网络中每一层的输出都是x_0的标量乘积，$\alpha^i x_0$；2）cross网络是以bit-wise形式交叉的，即便同一个field embedding向量下的元素也会相互影响。

关于第1点，论文给出了证明过程

PROOF. When k=1, according to the associative law and distributive law for matrix multiplication, we have:

$$\begin{aligned}\mathbf{x}_1 &= \mathbf{x}_0(\mathbf{x}_0^T\mathbf{w}_1) + \mathbf{x}_0 \\ &= \mathbf{x}_0(\mathbf{x}_0^T\mathbf{w}_1 + 1) \\ &= \alpha^1\mathbf{x}_0\end{aligned} \tag{4}$$

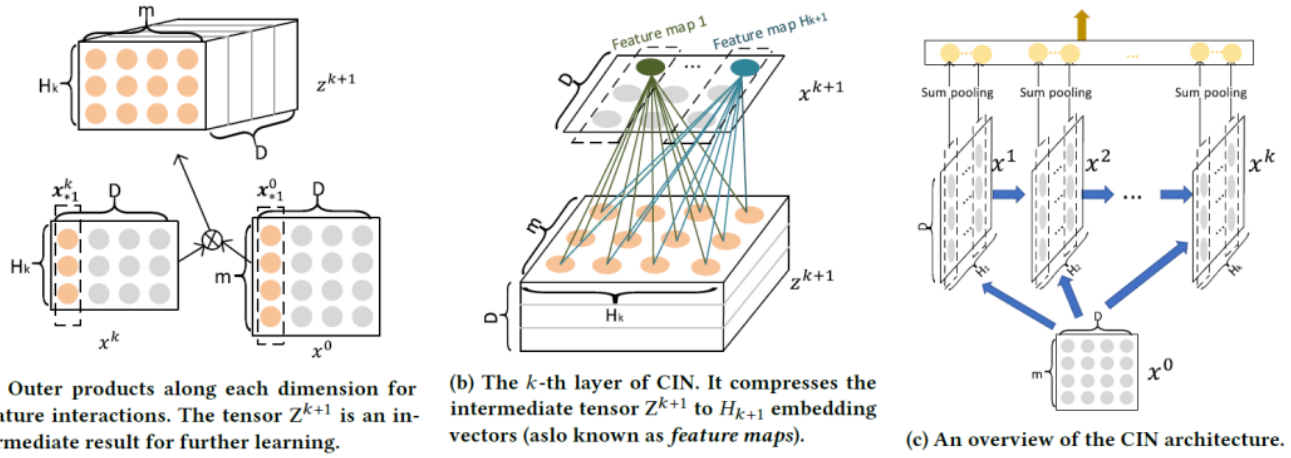
where the scalar $\alpha^1 = \mathbf{x}_0^T\mathbf{w}_1 + 1$ is actually a linear regression of $\mathbf{x}_0$. Thus, $\mathbf{x}_1$ is a scalar multiple of $\mathbf{x}_0$. Suppose the scalar multiple statement holds for k=i. For k=$i+1$, we have :

$$\begin{aligned}\mathbf{x}_{i+1} &= \mathbf{x}_0\mathbf{x}_i^T\mathbf{w}_{i+1} + \mathbf{x}_i \\ &= \mathbf{x}_0((\alpha^i\mathbf{x}_0)^T\mathbf{w}_{i+1}) + \alpha^i\mathbf{x}_0 \\ &= \alpha^{i+1}\mathbf{x}_0\end{aligned} \tag{5}$$

where, $\alpha^{i+1} = \alpha^i(\mathbf{x}_0^T\mathbf{w}_{i+1} + 1)$ is a scalar. Thus $\mathbf{x}_{i+1}$ is still a scalar multiple of $\mathbf{x}_0$. By induction hypothesis, the output of cross network $\mathbf{x}_k$ is a scalar multiple of $\mathbf{x}_0$. □

注：虽然cross网络的输出是x_0的标量乘积，但并不意味输出是x_0的线性量，因为标量系数α^i是与x_0相关的，它并非仅由模型自身的参数来定义。

针对以上两个缺点，xDeepFM提出了新的解决方案，CIN。



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

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

(c) An overview of the CIN architecture.

论文给出的CIN实现原理图如上，a和b是CIN第k层的计算过程，数学形式是

$$X_{h,*}^{k} = \sum_{i=1}^{H_{k-1}} \sum_{j=1}^{m} W_{ij}^{k,h} (X_{i,*}^{k-1} \circ X_{j,*}^{0})$$

式中，$1 \le h \le H_k$，$W^{k,h}$是一个$H_{k-1} \times m$的参数矩阵，符号"∘"表示向量的哈达玛积，即$[a_1, a_2, a_3] \circ [b_1, b_2, b_3] = [a_1b_1, a_2b_2, a_3b_3]$。每一层$X^k$都由前一层$X^{k-1}$和$X^0$进行vector-wise的交互得到，如果用T表示网络深度，那么CIN最高可以实现T+1阶特征交叉。图c是CIN网络的整体架构，网络的最终输出p^+，是每层的X^k沿embedding维度sum pooling后再拼接的向量，维度是所有隐层的feature map个数之和，即$\Sigma_{i=1 \sim T} H_i$。

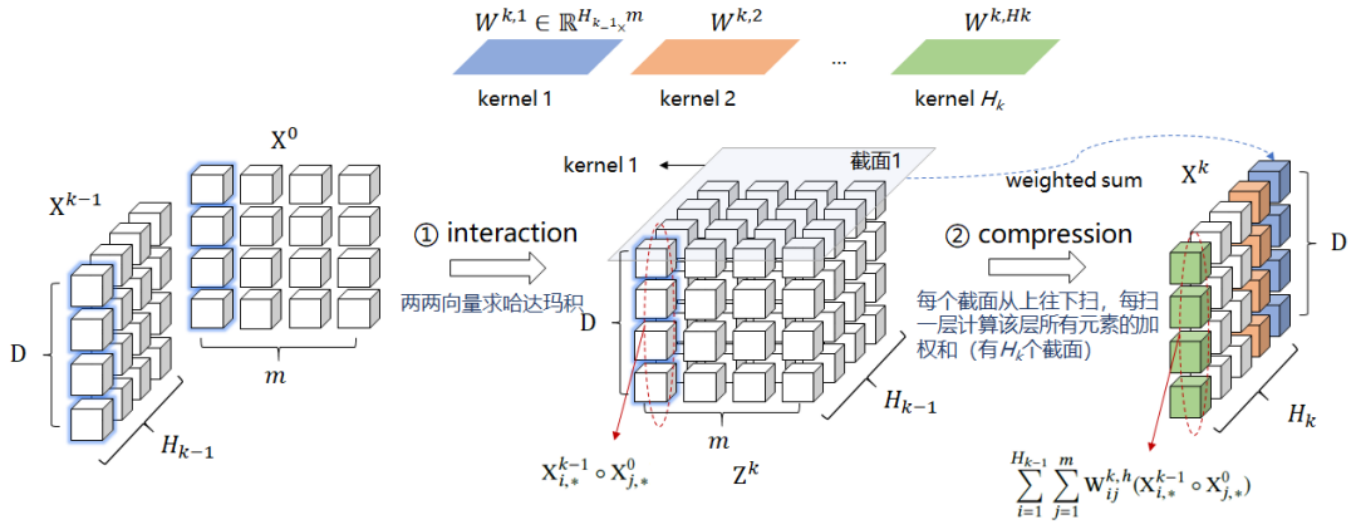
$$p^{+} = [p^{1}, p^{2}, \ldots, p^{T}] \in \mathbb{R}^{\sum_{i=1}^{T} H_i}.$$

其中，

$$p^{k} = [p_1^k, p_2^k, \ldots, p_{H_k}^k]$$

$$p_i^k = \sum_{j=1}^{D} X_{i,j}^k$$

初次读xDeepFM论文，我就有一个困惑，为什么说CIN特征交叉建模是vector-wise的呢？不知道大家是否会有同样的疑惑，个人认为论文里的图片和公式，并没有很直观的体现出vector-wise。所以，如果我们把图a和b所示的张量在三维空间里旋转一下，用另一种角度来看，或许能帮助更直观的理解。



CIN第k层的计算过程

上面这张图我们也把CIN第k层的计算拆解成两步，第一步是对X^0和X^{k-1}中的两两embedding向量做哈达玛积，得到一个中间张量Zk，这一步实际就是对两两特征做交互，所以称为interaction。第二步是使用

Hk个截面（类似CNN中的“kernel”），每个截面对中间张量Z^k从上往下扫，每扫一层计算该层所有元素的加权和，一个截面扫完D层得到Xk的一个embedding向量（类似CNN中的“feature map”），Hk个截面扫完得到Hk个embedding向量。这一步实际是用kernel将中间张量压缩成一个embedding向量，所以我们称此过程为compression。这也是整个子网络被命名为*Compressed Interaction Network*（CIN）的原因。从这两步计算过程可以看到，特征交互是在vector-wise级别上进行的（哈达玛积），尽管中间经过压缩，每一个隐层仍然保留着embedding vector的形式，所以说CIN实现的是vector-wise的特征交叉。

注：假设CIN每个隐层的feature map数量都为H，特征field数量为m，那么深度为T层的CIN，空间复杂度为$O(mTH^2)$，由于m和H通常不会很大，所以认为这种复杂度是可接受的（对应的DNN模块空间复杂度为$O(mDH+TH^2)$）。而CIN的时间复杂度为$O(mH^2DT)$，对应的DNN模块时间复杂度为$O(mHD+H^2T)$，相比之下，时间复杂度上CIN稍有不利。

最后，给出论文中的实验效果：

Table 2: Performance of individual models on the Criteo, Dianping, and Bing News datasets. Column *Depth* indicates the best network depth for each model.

Model name	AUC	Logloss	Depth
Criteo			
FM	0.7900	0.4592	-
DNN	0.7993	0.4491	2
CrossNet	0.7961	0.4508	3
CIN	**0.8012**	0.4493	3
Dianping			
FM	0.8165	0.3558	-
DNN	0.8318	0.3382	3
CrossNet	0.8283	0.3404	2
CIN	**0.8576**	**0.3225**	2
Bing News			
FM	0.8223	0.2779	-
DNN	0.8366	0.273	2
CrossNet	0.8304	0.2765	6
CIN	**0.8377**	**0.2662**	5

Table 3: Overall performance of different models on Criteo, Dianping and Bing News datasets. The column *Depth* presents the best setting for network depth with a format of (cross layers, DNN layers).

	Criteo			Dianping			Bing News		
Model name	AUC	Logloss	Depth	AUC	Logloss	Depth	AUC	Logloss	Depth
LR	0.7577	0.4854	-,-	0.8018	0.3608	-,-	0.7988	0.2950	-,-
FM	0.7900	0.4592	-,-	0.8165	0.3558	-,-	0.8223	0.2779	-,-
DNN	0.7993	0.4491	-,2	0.8318	0.3382	-,3	0.8366	0.2730	-,2
DCN	0.8026	0.4467	2,2	0.8391	0.3379	4,3	0.8379	0.2677	2,2
Wide&Deep	0.8000	0.4490	-,3	0.8361	0.3364	-,2	0.8377	0.2668	-,2
PNN	0.8038	0.4927	-,2	0.8445	0.3424	-,3	0.8321	0.2775	-,3
DeepFM	0.8025	0.4468	-,2	0.8481	0.3333	-,2	0.8376	0.2671	-,3
xDeepFM	**0.8052**	**0.4418**	3,2	**0.8639**	**0.3156**	3,3	**0.8400**	**0.2649**	3,2

注：CIN每个隐层的feature map数量设置为Criteo 200，Dianping/Bing News 100。

在Criteo，Dianping，Bing News三个数据集上，单独交叉模块CIN的效果要一致优于FM、DNN、CrossNet（DCN中的Cross模块），结合线性和DNN模块后的xDeepFM效果要显著优于前文介绍的其他

模型。

干货过多，一次看完很难全部吸收，“引入注意力机制：提高模型自适应能力与可解释性”这一节将于下篇分享，请持续关注炼丹笔记，这里有最前沿最优质最全最专业的学习资料，我们成立该公众号的初衷，就是让大家平等快速的获取想要的知识，希望大家都能通过该公众号提升自己。

收录于话题 #搜索推荐前沿算法·42个

上一篇 下一篇

喜欢此内容的人还喜欢

我用特征工程+LR超过了xDeepFM！

kaggle竞赛宝典

华为DIGIX CTR赛题冠军分享

Coggle数据科学

我们与Datawhale的故事！

Datawhale