

# 推荐系统系列（十一）：xDeepFM理论与实践

原创 默存 SOTA Lab 1月5日

## 前言

今天为大家带来的是2018年由中科大、北邮、微软联合推出的xDeepFM（eXtreme Deep Factorization Machine）模型[1]，该模型的主要贡献在于，基于vector-wise的模式提出了新的显式交叉高阶特征的方法，并且与DCN [2]一样，能够构造有限阶交叉特征。虽然xDeepFM在名称上与DeepFM [3]相似，但其主要对比的是DCN模型。

在进行模型分析与对比之前，首先为大家介绍一下什么是vector-wise模式。与vector-wise概念相对应的是bit-wise，在最开始的FM模型当中，通过特征隐向量之间的点积来表征特征之间的交叉组合。特征交叉参与运算的最小单位为向量，且同一隐向量内的元素并不会交叉乘积，这种方式称为vector-wise。后续FM的衍生模型，尤其是引入DNN模块后，常见的做法是，将embedding之后的特征向量拼接到一起，然后送入后续的DNN结构模拟特征交叉的过程。这种方式与vector-wise的区别在于，各特征向量concat在一起成为一个向量，抹去了不同特征向量的概念，后续模块计算时，对于同一特征向量内的元素会有交互计算的现象出现，这种方式称为bit-wise。将常见的bit-wise方式改为vector-wise，使模型与FM思想更贴切，这也是xDeepFM的Motivation之一。这里需要提醒的是，vector-wise的方式其实在之前介绍的PNN、NFM、AFM与DeepFM中都有使用，但是并没有单独拎出来说明清楚。

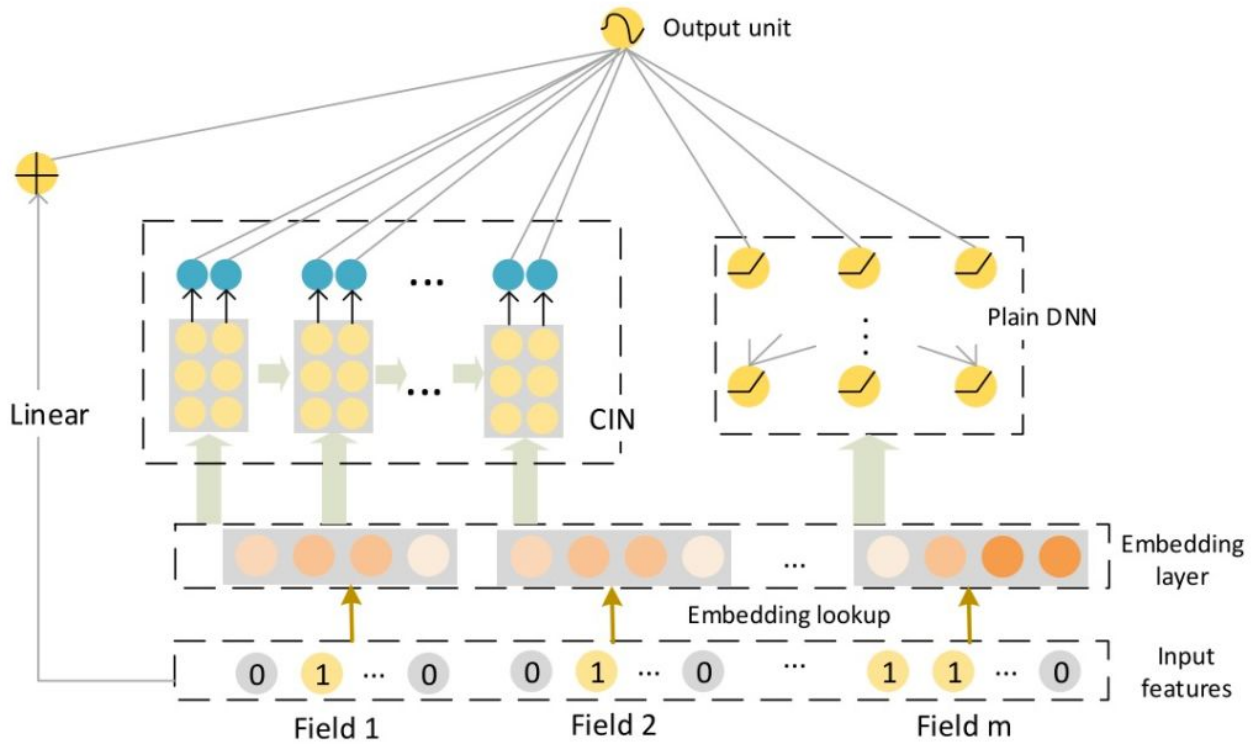
关于xDeepFM的另一个贡献点“有限阶数特征交叉”，所具备的优点之前也提到过。简单DNN结构虽然说能够隐式交叉特征，学习任意函数的表示，但是我们并不清楚其最大特征交叉阶数为多少，简单来说就是无法得知哪些特征交叉可以得到最佳效果，而显式指定阶数交叉特征能够在一定程度缓解这些问题。

在论文原文中，作者对DCN模型的Cross Network进行分析，并从数学层面证明了Cross Network最终学习到的是一种特定形式的表示，认为该结构不利于模型充分表征交叉特征。个人理解，Cross Network之所以采用该形式，是因为其精简参数，在实现显式特征交叉的同时控制计算复杂度。二者孰优孰劣，仍需要在实际业务场景中对比试验方能判定。

## 分析

### 1. 模型结构

xDeepFM的模型结构如下所示，下面将对模型的四个模块详细讲解。



**Figure 5: The architecture of xDeepFM.**

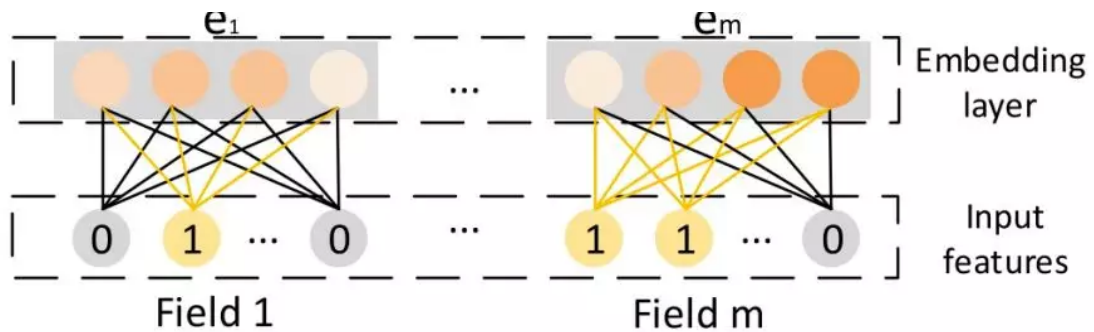
SOTA Lab

### 1.1 Embedding Layer

这个部分无需多言，将离散特征进行embedding，表示成低维稠密向量。

If the field is univalent, the feature embedding is used as the field embedding.

If the field is multivalent, the sum of feature embedding is used as the field embedding.



**Figure 1: The field embedding layer. The dimension of embedding in this example is 4.**

SOTA Lab

## 1.2 Compressed Interaction Network (CIN)

与DCN的bit-wise fashion不同的是，xDeepFM的CIN中采用vector-wise fashion对特征向量进行交叉。将embedding矩阵表示为  $X^0 \in \mathbb{R}^{m \times D}$ ， $X^0$  第  $i$  行表示第  $i$  个Field对应的特征embedding向量，共有  $m$  个  $D$  维的embedding vector。

CIN是多层输出结构，每层都会产生一个中间结果矩阵，将第  $k$  层产生的矩阵表示为  $X^k \in \mathbb{R}^{H_k \times D}$ ，令  $H_0 = m$ ，其中  $H_k$  表示第  $k$  层特征向量数。

接下来，让我们详细的分析下计算过程，公式如下：

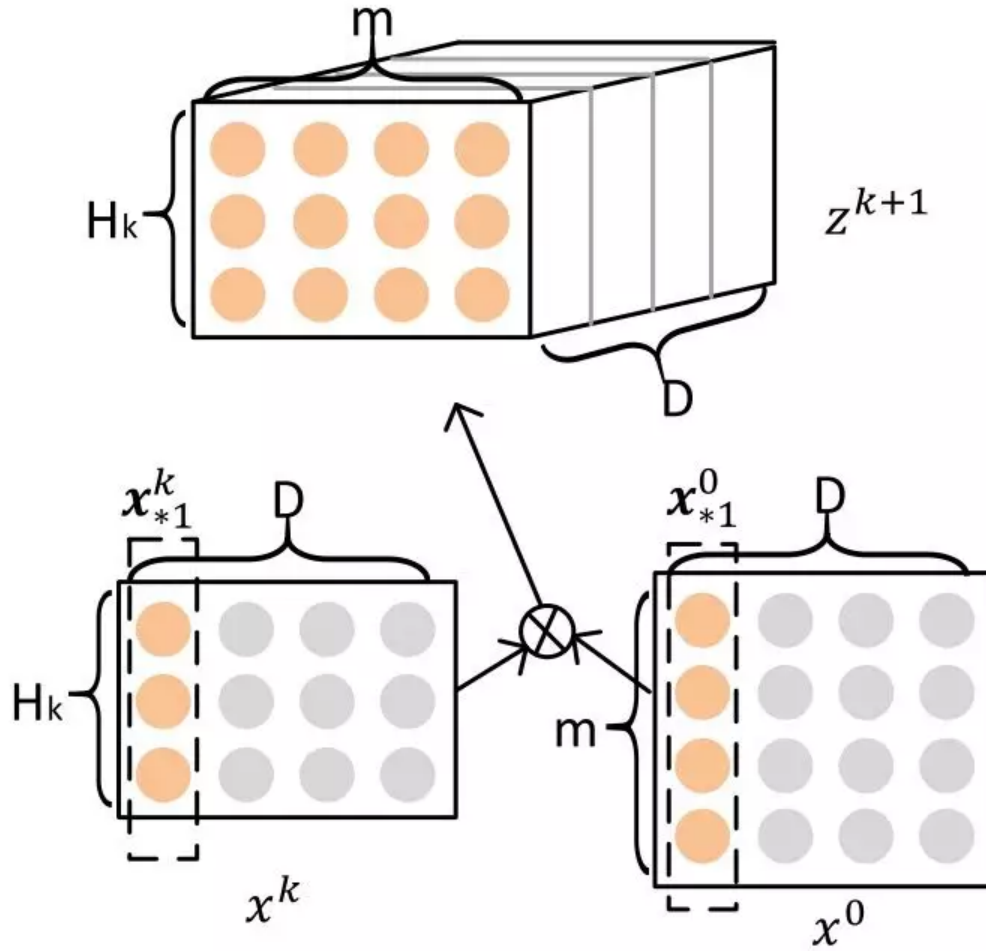
$$X_{h,*}^{k+1} = \sum_{i=1}^{H_k} \sum_{j=1}^m W_{i,j}^{k+1,h} (X_{i,*}^k \circ X_{j,*}^0) \quad (1)$$


其中， $X_{h,*}^{k+1} \in \mathbb{R}^D$  表示  $X^{k+1}$  中第  $h$  行特征向量，同理， $X_{i,*}^k$  表示CIN第  $k$  层输出矩阵的第  $i$  行特征向量， $X_{j,*}^0$  表示原始特征矩阵  $X^0$  第  $j$  行特征向量。 $W^{k+1,h} \in \mathbb{R}^{H_k \times m}$  是参数矩阵， $W_{i,j}^{k+1,h}$  是一个标量值，表示第  $i$  行第  $j$  列位置的参数。运算  $\circ$  表示哈达玛积，即向量对应元素相乘，结果仍为向量。

从计算原理上进行理解，公式（1）首先将上一层（第  $k$  层）输出的特征矩阵  $X^k$  的特征向量，与原始特征矩阵  $X^0$  的特征向量，两两之间进行哈达玛积。因为上一层有  $H_k$  个特征向量，原始特征矩阵有  $m$  个特征向量，所以运算完成会得到  $H_k \times m$  个向量。将得到的全部向量结果根据参数  $W^{k,h}$  进行加权求和，得到第  $k+1$  层的结果  $X_{h,*}^{k+1}$ 。

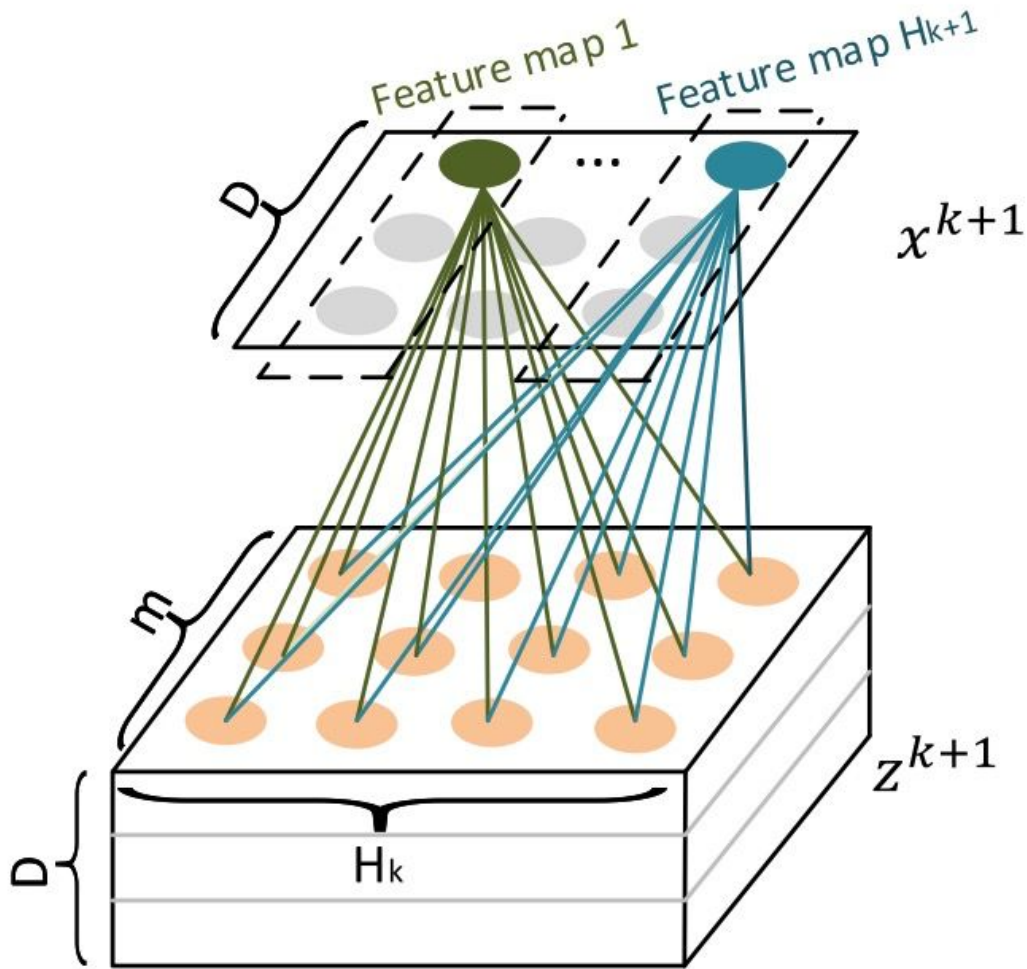
让我们换个视角观察上述计算过程， $X^{k+1}$  中每一行特征向量都是  $X^k$  与  $X^0$  特征交叉之后加权求和得到的。 $X^{k+1}$  中任意两个特征向量的不同点在于，计算过程中的加权求和参数不同。所以可以将  $X^{k+1}$  的计算拆成2个步骤：1）将  $X^k$  与  $X^0$  中的特征向量两两交叉计算，得到中间结果  $Z^{k+1}$ ，其中包含  $H_k \times m$  个  $D$  维特征向量；2）使用不同的加权参数  $W \in \mathbb{R}^{H_k \times m}$  对  $Z^{k+1}$  中每个特征向量进行加权求和，得到  $X^{k+1}$ ；

步骤1）中，计算  $Z^{k+1}$  的过程可以看做是矩阵  $X^k$  与  $X^0$  的向量外积计算，如下图（a）所示，得到的  $Z^{k+1} \in \mathbb{R}^{H_k \times m \times D}$ 。



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

步骤2) 通过不同的参数矩阵  $W \in \mathbb{R}^{H_k \times m}$  对  $Z^{k+1}$  中的向量进行加权求和，这个过程类似于CNN中的卷积操作，不同的参数矩阵便是不同的卷积核，每个卷积核与  $Z^{k+1}$  进行一次卷积计算得到一个  $D$  维的 **feature map** 向量，其中  $D$  为  $Z^{k+1}$  的通道数。最终通过  $H_{k+1}$  次不同的卷积之后，便得到  $X^{k+1} \in \mathbb{R}^{H_{k+1} \times m}$ 。通过卷积操作对上层特征矩阵  $Z^{k+1}$  进行压缩编码，这也是CIN中“compress”的由来。计算过程如图（b）所示：



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

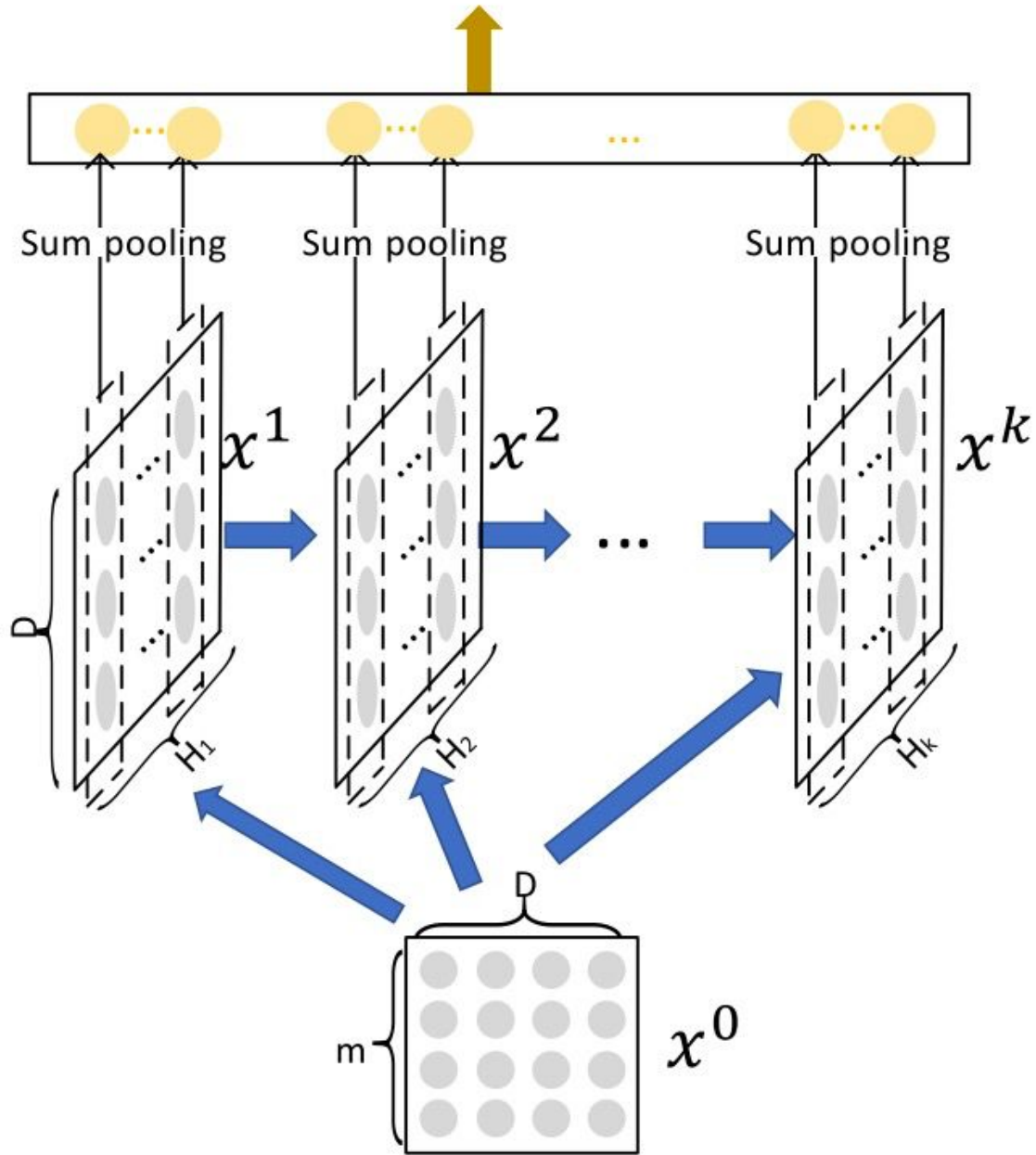
SOTA Lab

得到每层的输出之后，通过sum pooling操作将所有的特征矩阵进一步压缩为向量，作为最终的输出。

$X^k$  中含有  $H_k$  个  $D$  维feature map，对于每个feature map进行求和池化之后得到一个标量值  $p_i^k$ ，其中  $p_i^k = \sum_{j=1}^D X_{i,j}^k$ ，所以  $X^k$  产出长度为  $H_k$  的向量  $p^k = [p_1^k, p_2^k, \dots, p_{H_k}^k]$ 。同理，对于CIN中的每一个中间层都产出一个向量，将  $k$  层所有向量进行拼接，得到  $p^+ = [p^1, p^2, \dots, p^k]$ 。最终CIN的输出为， $y = 1/(1 + \exp(-(p^+)^T W^o))$ ，其中  $W^o$  是长度为  $\sum_{k=1}^{H_k}$  的参数向量。

整体如下图所示：





**(c) An overview of the CIN architecture.**

SOTA Lab

从上述分析可以看出，xDeepFM的CIN模块在  $l$  层仅包含了  $l+1$  阶交叉特征。对比DCN模型中，在  $l$  层包含了从1到  $l+1$  阶全部交叉特征。这两种方式殊途同归，都能够实现有限阶特征交叉。

### 1.2.1 复杂度分析

#### 1) 空间复杂度:

首先，第  $k$  层有  $H_k$  个  $W^{k,h} \in \mathbb{R}^{H_{k-1} \times m}$ ，所以第  $k$  层参数量为  $H_k \times H_{k-1} \times m$ ，假设CIN有  $T$  层，那么共有  $\sum_{k=1}^T H_k \times H_{k-1} \times m$  个参数。最后将各层输出池化拼接，需要  $\sum_{k=1}^T H_k$  个参数进行加权求和。所以，共有  $\sum_{k=1}^T H_k \times (1 + H_{k-1} \times m)$  个参数。

对比普通的DNN，第一层参数量为  $m \times D \times H_1$ ，后续层参数为  $\sum_{k=2}^T H_k \times H_{k-1}$ ，最后将所有输出进行加权求和，需要参数为  $H_T$ 。所以全部参数量为  $m \times D \times H_1 + H_T + \sum_{k=2}^T H_k \times H_{k-1}$ 。

不妨假设每一层的  $H_k = H$ ，那么CIN的空间复杂度为  $O(mTH^2)$ ，普通DNN的空间复杂度为  $O(mDH + TH^2)$ 。

一般来说， $m$ （特征Field个数）与  $H_k$ （每层feature map数目）都是较小的数值，所以  $W^{k,h} \in \mathbb{R}^{H_{k-1} \times m}$  在一个可接受的范围之内。但如果有一些极端情况， $m$  与  $H_k$  都比较大的时候，可以使用矩阵  $L$  阶分解的思想，将  $W^{k,h}$  表示为两个小矩阵的乘积， $W^{k,h} = U^{k,h}(V^{k,h})^T$ ，其中  $U^{k,h} \in \mathbb{R}^{H_{k-1} \times L}$ ， $V^{k,h} \in \mathbb{R}^{m \times L}$ ，并且  $L \ll H_k$  同时  $L \ll m$ 。在这种情况下，假设  $H_k = H$ ，那么  $W^{k,h}$  的参数为  $HL + mL$ 。所以CIN的空间复杂度从  $O(mTH^2)$  变为  $O(mTHL + TH^2L)$ ，而普通DNN的空间复杂度变为  $O(mDH + TH^2)$ 。

## 2) 时间复杂度:

因为CIN中每一层都需要计算  $Z^{k+1}$ ，其计算时间复杂度为  $O(mHD)$ ，而后每层都有  $H$  个feature map，所以  $T$  层的CIN计算时间复杂度为  $O(mH^2DT)$ 。容易推出，普通DNN的时间复杂度为  $O(mDH + TH^2)$ 。

CIN的主要瓶颈在于时间复杂度高。

## 1.3 DNN

在Embedding Layer之后，除了连接CIN模块，同时并行的会接入到简单的多层感知机。与Wide&Deep、DeepFM类似，设置这个模块的目的在于，隐式交叉编码可以作为显式交叉（CIN）的补充，进一步的提高模型表征能力。

## 1.4 Linear part

这个部分可以是一个简单的LR，将原始特征（未经过Embedding）作为输入，表示为： $y = \sigma(W_{linear}^T a)$ ，其中的  $a$  就是原始特征输入， $\sigma$  是激活函数。

## 1.5 模型组合

各个模块分析完毕，将1.1~1.4所有模块组合到一块，三个输出模块（Linear、CIN、DNN）统一到一起，作为模型的最终输出，公式如下：

$$\hat{y} = \sigma(W_{linear}^T a + W_{dnn}^T X_{dnn}^k + W_{cin}^T p^+ + b) \quad (2)$$

其中， $a$  是原始特征， $X_{dnn}^k$  是DNN模块的输出向量， $p^+$  是CIN模块的输出向量， $W_{linear}$ 、 $W_{dnn}$ 、 $W_{cin}$  分别是对应模块的可训练参数， $b$  为全局偏置项， $\sigma$  为激活函数。

从这个形式上来看，xDeepFM既有低阶项又有高阶项，既有显式交叉又有隐式组合，并且是基于vector-wise级别的交叉，可谓是应有尽有，算是FM类模型的完备实现了。

## 2. 与FM、DeepFM的关系

假设输入的所有特征Field都是单值的，CIN模块只有1层，并且其中的feature map只有1个，此时的xDeepFM模型基本等价于DeepFM模型。因为CIN模块只对所有的特征embedding vector进行两两哈达玛积，将得到的结果进行加权求和（DeepFM是权重恒定为1进行求和），然后再对结果进行sum pooling作为输出。结合一阶项与DNN部分，模型基本等价于DeepFM。

如果更进一步，将DNN部分去除，同时在加权求和部分固定权重为1，那么此时xDeepFM与FM模型完全等价。

## 3. 性能分析

围绕三个方面进行性能试验

- 1) CIN模块进行高阶特征交叉是否有效？
- 2) 对于推荐系统而言，同时结合显式高阶特征交叉与隐式高阶特征交叉，是否有必要？
- 3) 网络结构的设置对于xDeepFM性能有何影响？

### 3.1 实验准备

数据集使用 Criteo 、 Dianping 和 Bing News 。

**Table 1: Statistics of the evaluation datasets. M indicates million and K indicates thousand.**

Datasest	#instances	#fields	#features (sparse)
Criteo	45M	39	2.3M
Dianping	1.2M	18	230K
Bing News	5M	45	17K



评测指标使用AUC与Logloss。其中AUC主要关注的是正负样本的相对顺序，并且对非平衡数据不敏感。而Logloss关注预测值与真实值之间的相关程度，在计算广告中因为涉及到广告竞价策略，一般使用Logloss比较多。



对比模型：LR、FM、DNN、PNN（IPNN与OPNN择其优）、Wide&Deep、DCN和DeepFM。

参数设置：

`learning rate` : 0.001, 优化器: `Adam` , `batch size` : 4096

对于DNN、DCN、Wide&Deep、DeepFM与xDeepFM使用L2正则，其中  $\lambda = 0.0001$  。对于PNN使用dropout rate=0.5。

模型每层的默认节点数:（1）DNN Layer节点为400；（2）在 `Criteo` 中，CIN Layer节点200。而 `Dianping` 和 `Bing News` 中，CIN Layer节点数为100；

在实验过程中，Field embedding维度默认为10。

### 3.2 问题一

所有对比模型都优于FM，说明特征高阶交叉的必要性。但所有模型都低于CIN，这也就凸显出了CIN结构（显式高阶交叉）的优越性，论证了CIN模块进行高阶特征交叉是有效的。

**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	<b>0.8012</b>	0.4493	3
Dianping			
FM	0.8165	0.3558	-
DNN	0.8318	0.3382	3
CrossNet	0.8283	0.3404	2
CIN	<b>0.8576</b>	<b>0.3225</b>	2
Bing News			
FM	0.8223	0.2779	-
DNN	0.8366	0.273	2
CrossNet	0.8304	0.2765	6
CIN	<b>0.8377</b>	<b>0.2662</b>	5

### 3.3 问题二

上述对比试验仅说明了CIN的有效性，但是在xDeepFM中结合了CIN与DNN两个部分，这种结合是否能带来提升呢？答案是肯定的，通过下表对比结果可以看出xDeepFM这种双形态结合的有效性。

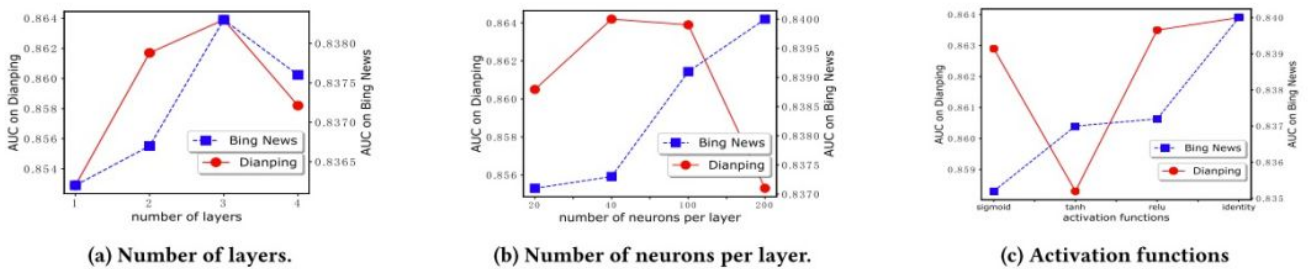
**Table 3: Overall performance of different models on Criteo, Dianping and Bing News datasets. The column *Depth* presents the best setting for network depth with a format of (cross layers, DNN layers).**

Model name	Criteo			Dianping			Bing News		
	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	0.2673	-, 3
xDeepFM	<b>0.8052</b>	<b>0.4418</b>	3, 2	<b>0.8639</b>	<b>0.3156</b>	3, 3	<b>0.8400</b>	<b>0.2649</b>	3, 2

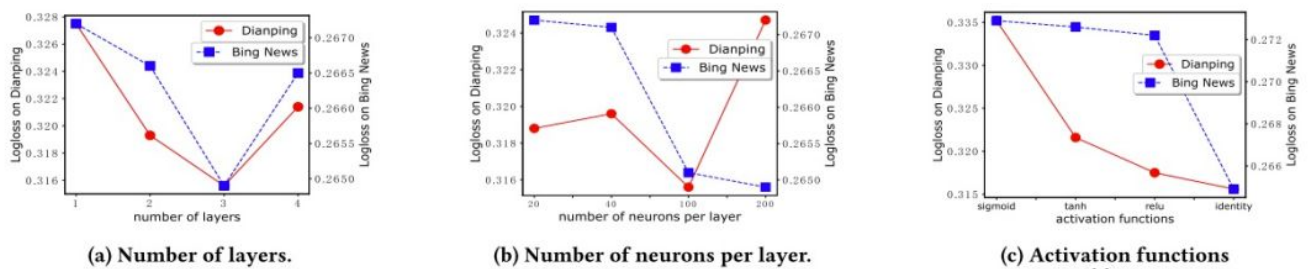
### 3.4 问题三

这个环节主要探讨超参数带来的影响，1）隐藏层层数；2）每层的节点数；3）激活函数；在对比过程中，固定最佳的DNN参数，仅调节CIN模块参数。

xDeepFM的表现可简单通过下图可得出结论，就不做过多剖析了。



**Figure 6: Impact of network hyper-parameters on AUC performance.**



**Figure 7: Impact of network hyper-parameters on Logloss performance.**

## 实践

仍然使用 *MovieLens100K* 作为数据集，核心代码如下。

参数含义：

`vec_dim` : 代表embedding vector维度

`field_lens` : list结构，其中每个元素代表对应Field有多少取值。例如gender有两个取值，那么其对应的元素为2

`cin_layer_num` : cin network 层数

`dnn_layers` : list结构，其中每个元素对应DNN部分节点数目

`lr` : 学习率

```
class XDeepFM(object):

    def __init__(self, vec_dim=None, field_lens=None, cin_layer_num=None, dnn_layers=None, lr=None,
                 dropout_rate=None):
        self.vec_dim = vec_dim
        self.field_lens = field_lens
        self.field_num = len(field_lens)
        self.feats_num = np.sum(field_lens)
        self.cin_layer_num = cin_layer_num
        self.dnn_layers = dnn_layers
        self.lr = lr
        self.dropout_rate = dropout_rate

        self._build_graph()

    def _build_graph(self):
        self.add_input()
        self.inference()

    def add_input(self):
        self.index = tf.placeholder(tf.int32, shape=[None, self.field_num], name='feat_index') # (l
        self.x = tf.placeholder(tf.float32, shape=[None, self.feats_num], name='feat_value') # (ba
        self.y = tf.placeholder(tf.float32, shape=[None], name='input_y')
        self.is_train = tf.placeholder(tf.bool)

    def cin_layer(self, x0_tensor, xk_tensor, xk_field_num, feature_map_num, name):
        with tf.variable_scope(name):
            x0 = tf.split(value=x0_tensor, num_or_size_splits=self.vec_dim*[1], axis=2)
            xk = tf.split(value=xk_tensor, num_or_size_splits=self.vec_dim*[1], axis=2)
```

```

z_tensor = tf.matmul(x0, xk, transpose_b=True) # (D, batch, m, H_k)
z_tensor = tf.reshape(z_tensor, shape=[self.vec_dim, -1, self.field_num*xk_field_num])
z_tensor = tf.transpose(z_tensor, perm=[1, 0, 2]) # (batch, D, m*H_k)

filters = tf.get_variable(name='filters', shape=[1, self.field_num*xk_field_num, feature_dim])
xk_1 = tf.nn.conv1d(z_tensor, filters=filters, stride=1, padding='VALID') # (batch, D, m*H_k)
xk_1 = tf.transpose(xk_1, perm=[0, 2, 1]) # (batch, feature_map_num, D)

return xk_1

```

```
def inference(self):
```

```
    with tf.variable_scope('first_order_part'):
```

```
        first_ord_w = tf.get_variable(name='first_ord_w', shape=[self.feats_num, 1], dtype=tf.float32)
        first_order = tf.nn.embedding_lookup(first_ord_w, self.index) # (batch, m, 1)
        first_order = tf.reduce_sum(tf.multiply(first_order, tf.expand_dims(self.x, axis=2)), axis=-1)

```

```
    with tf.variable_scope('emb_part'):
```

```
        embed_matrix = tf.get_variable(name='second_ord_v', shape=[self.feats_num, self.vec_dim])
        embed_v = tf.nn.embedding_lookup(embed_matrix, self.index) # (batch, m, D)

        embed_x = tf.multiply(tf.expand_dims(self.x, axis=2), embed_v) # (batch, m, D)
        embed_x = tf.layers.dropout(embed_x, rate=self.dropout_rate, training=self.is_train) # (batch, m, D)
        node_num = self.field_num * self.vec_dim
        embed_x = tf.reshape(embed_x, shape=[-1, node_num]) # (batch, node_num)

```

```
    with tf.variable_scope('cin_part'):
```

```
        cross_tensors = []
        x0_tensor = tf.reshape(embed_x, shape=[-1, self.field_num, self.vec_dim]) # (batch, m, D)
        cross_tensors.append(x0_tensor)
        field_nums = []
        field_nums.append(int(self.field_num))
        for i, layer_num in enumerate(self.cin_layer_num):
            xk_tensor = self.cin_layer(x0_tensor, cross_tensors[-1], field_nums[-1], layer_num)
            cross_tensors.append(xk_tensor)
            field_nums.append(layer_num)
        p_vec = [tf.reduce_sum(x, axis=2) for x in cross_tensors]
        cin = tf.concat(p_vec, axis=1)
        cin_lens = np.sum(field_nums)

```

```
    with tf.variable_scope('dnn_part'):
```

```
        dnn = embed_x
        in_num = node_num
        for i in range(len(self.dnn_layers)):
            out_num = self.dnn_layers[i]
            w = tf.get_variable(name='w_%d'%i, shape=[in_num, out_num], dtype=tf.float32)

```

```
b = tf.get_variable(name='b_%d'%i, shape=[out_num], dtype=tf.float32)
dnn = tf.matmul(dnn, w) + b
dnn = tf.layers.dropout(tf.nn.relu(dnn), rate=self.dropout_rate, training=self.is_training)
in_num = out_num

with tf.variable_scope('output_part'):
    output = tf.concat([first_order, cin, dnn], axis=1)
    global_w = tf.get_variable(name='global_w', shape=[self.field_num+cin_lens+in_num, 1],
                                dtype=tf.float32)
    global_b = tf.get_variable(name='global_b', shape=[1], dtype=tf.float32)
    self.y_logits = tf.matmul(output, global_w) + global_b

self.y_hat = tf.nn.sigmoid(self.y_logits)
self.pred_label = tf.cast(self.y_hat > 0.5, tf.int32)

self.loss = -tf.reduce_mean(self.y*tf.log(self.y_hat+1e-8) + (1-self.y)*tf.log(1-self.y_hat))
self.train_op = tf.train.AdamOptimizer(self.lr).minimize(self.loss)
```

## reference

- [1] Lian, Jianxun, et al. "xdeepfm: Combining explicit and implicit feature interactions for recommender systems." *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018.
- [2] Wang, R., Fu, B., Fu, G., & Wang, M. (2017, August). Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17* (p. 12). ACM.
- [3] Guo, Huifeng, et al. "DeepFM: a factorization-machine based neural network for CTR prediction." *arXiv preprint arXiv:1703.04247* (2017).
- [4] <https://github.com/Leavingseason/xDeepFM>
- [5] <https://zhuanlan.zhihu.com/p/57162373>

专注知识分享，欢迎关注 SOTA Lab~