【论文导读】KDD2018｜xDeepFM---采用显式的高阶特征交互网络CIN

原创 潜心 推荐算法的小齿轮 8月21日

收录于话题
#推荐 2319 #CTR 19 #Github 98 #推荐系统论文与复现 18

前言

xDeepFM（eXtreme Deep Factorization Machine）模型是2018年由中科大、北邮、微软研究院在KDD上联合提出的模型（论文：“xDeepFM: Combining Explicit and Implicit Feature Interactions for Recommender Systems”）。该模型**「最主要的贡献」**就是提出了CIN（Compressed Interaction Network）网络结构。该网络提出的目的是**「为了提高特征交互的能力」**。

本篇文章约2.5k字，预计阅读12分钟。

1. 研究问题

DNN有出众的学习能力，但是学习到的特征交互都是**「隐式」**的，即并不知道模型进行哪些特征交叉。并且神经元的交互是**「bit-wise level」**。

> 文章提出两个概念的特征交叉：bit-wise level和vector-wise level。bit-wiselevel指的就是神经网络中节点之间的交互（DCN模型的Cross Network交互类型也是bit-wise level，文章具体提到了DCN，可以看一下原文描述）；vector-wise level指的是embeddingh向量之间的特征交叉（例如FM、DeepFM、PNN的特征交叉）。

因此本文主要是针对DCN的Cross Network的特征交叉是bit-wise level，对其进行了改进，**「使用CIN对Cross Network进行替代，CIN是显示的高阶特征交互，并且是vector-wise level」**。

2. 模型结构

以下是xDeepFM的总体结构，主要由三部分组成：Linear、CIN与DNN。

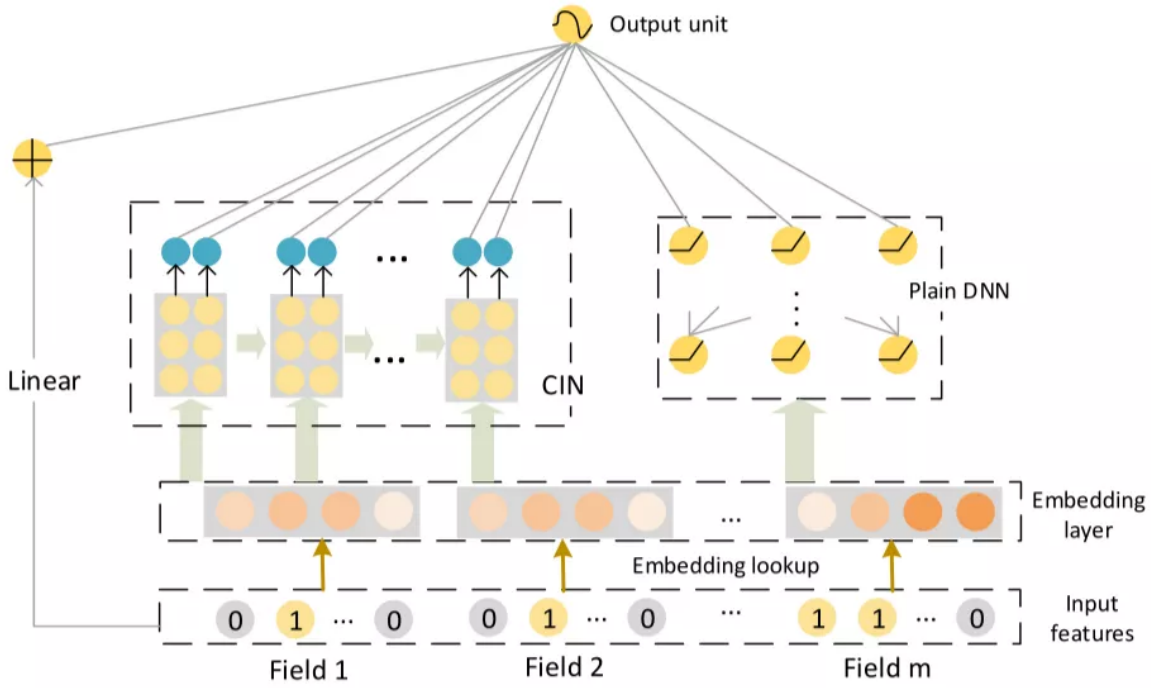


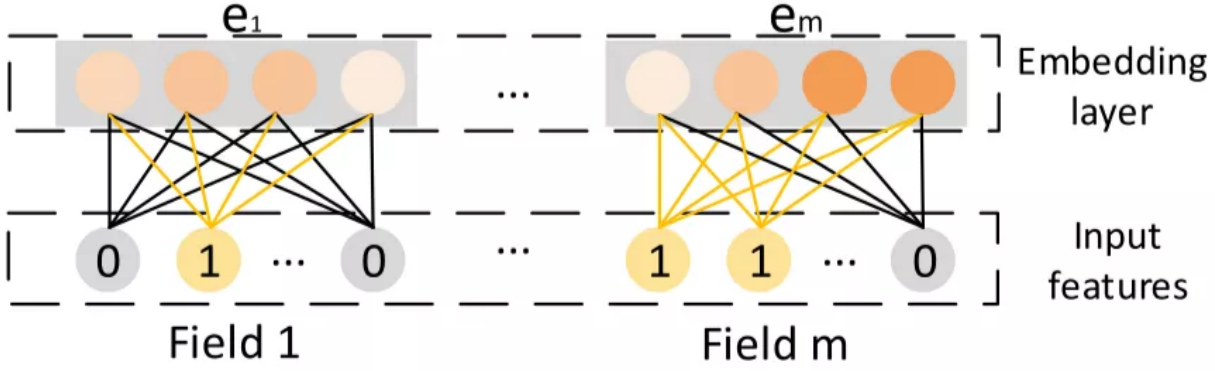
Figure 5: The architecture of xDeepFM.

2.1 Linear层

直接使用原生特征（one-hot编码）进行二分类任务。

2.2 Embedding层

与所有经典的基于神经网络的推荐相同，Embedding层都是将多个领域（field）组成的高维稀疏分类特征通过神经网络嵌入到低维密集特征。这也可以看作是一种向量（「vector-wise」）之间的交互。具体过程如下图所示：



为了后文叙述方便，作者将所有的embedding向量作为一个矩阵$\mathbf{X}^0 \in \mathbb{R}^{m \times D}$，公式化为：

$$\mathbf{X}^0 = [\mathbf{X}^0_{1,*}, \mathbf{X}^0_{2,*}, \dots, \mathbf{X}^0_{m,*}]^T$$

其中m表示field的数量，$\mathbf{X}^0_{i,*} = \mathbf{e}_i \in \mathbb{R}^D$表示embedding特征，$D$为embedding维度。

2.3 CIN网络

「输入：」 CIN网络的输入即Embedding的输出，$\mathbf{X}^0 \in \mathbb{R}^{m \times D}$；

「输出：」 CIN的第k层的输出，$\mathbf{X}^k \in \mathbb{R}^{H_k \times D}$，其中$H_k$为第$k$层特征向量的数量，且$H_0 = m$；

「计算：」 对于$\mathbf{X}^k_{h,*}$的计算方式如下：

$$\mathbf{X}^k_{h,*} = \sum_{i=1}^{H_{k-1}} \sum_{j=1}^{m} \mathbf{W}^{k,h}_{ij} \left(\mathbf{X}^{k-1}_{i,*} \circ \mathbf{X}^0_{j,*} \right)$$

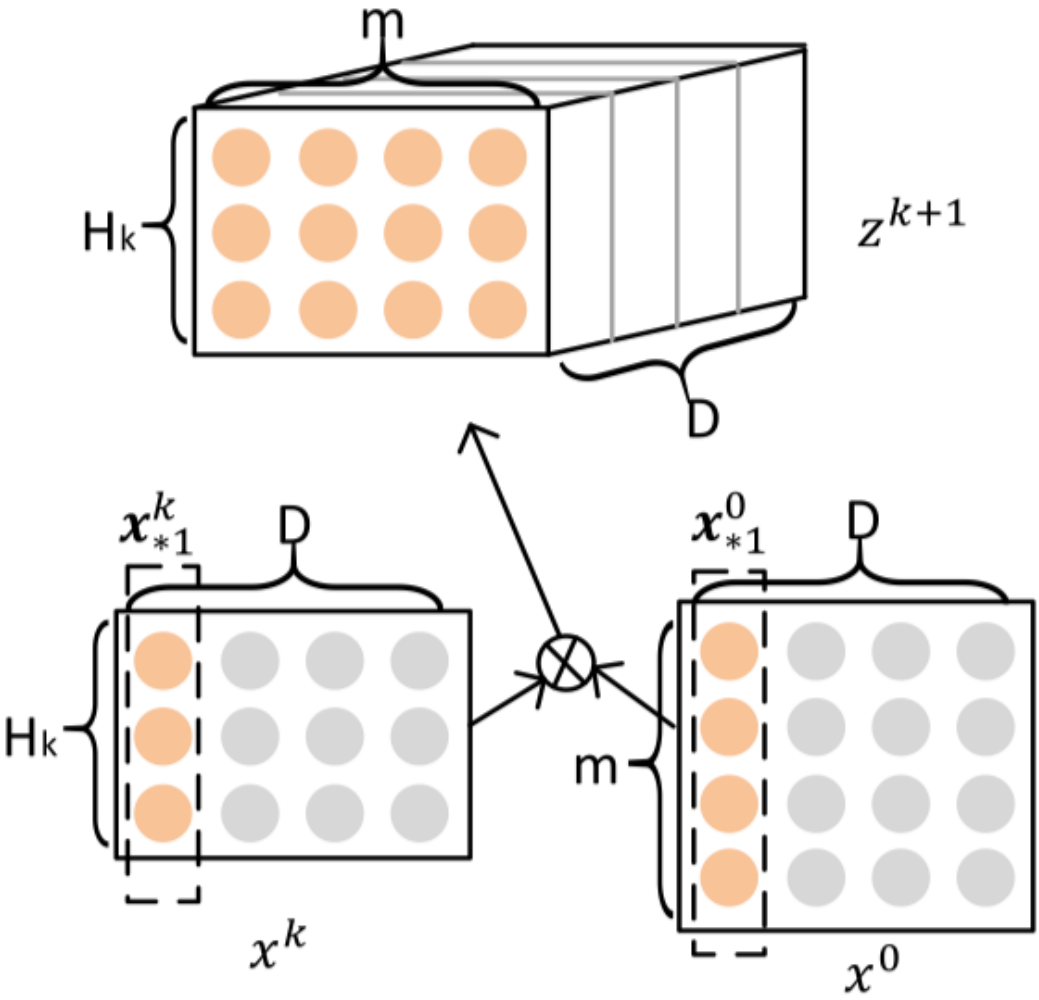
其中$1 \le h \le H_k$，$\mathbf{W}^{k,h} \in \mathbb{R}^{H_{k-1} \times m}$是第$h$个特征向量的参数矩阵，$\circ$表示一个哈达玛积（Hadamard product），即**「两个向量对应元素相乘」**。最终$\mathbf{X}^k = [\mathbf{X}^k_1, \mathbf{X}^k_2, \dots, \mathbf{X}^k_{H_k}]$。

「分析：」 由于$\mathbf{X}^k$是通过$\mathbf{X}^{k-1}$与$\mathbf{X}^0$的交互得到，因此特征交互是一种显示的方式，且交互的程度随着层的深度而增加。这里作者指出，CIN与RNN非常类似：下一个输出的隐藏层都依赖于当前最后一个隐藏层和一个额外的输入（embedding），且整个结构**「与CNN有着密切的联系」**。

具体分析

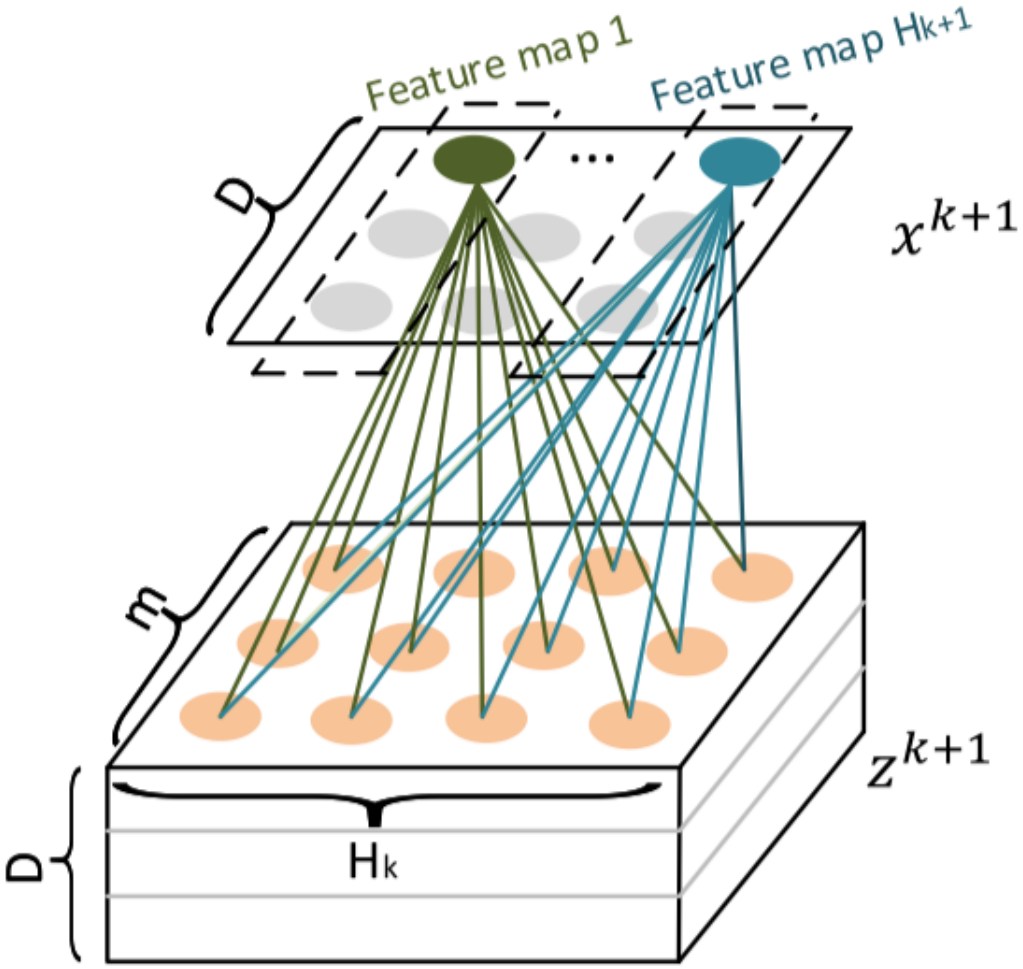
对于CIN的计算方式，可以**「以CNN的角度」**进行分析：【以下得到第$k+1$层的输出，$\mathbf{X}^{k+1} \in \mathbb{R}^{H_{k+1} \times D}$】

a）外积操作：引入一个过渡张量$\mathbf{Z}^{k+1} \in \mathbb{R}^{H_k \times m \times D}$，它是隐藏层$\mathbf{X}^k \in \mathbb{R}^{H_k \times D}$与输入$\mathbf{X}^0 \in \mathbb{R}^{m \times D}$的外积，如下图所示：



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

b）然后我们可以将整个$\mathbf{Z}^{k+1} \in \mathbb{R}^{H_k \times m \times D}$可以视为图片，$\mathbf{W}^{k+1,h} \in \mathbb{R}^{H_k \times m}$【注：原文写的是$k$，但这里得到的是下一层，所以应该是$k+1$】视为过滤器（filter）【**「重点：」** 这里$1 \leq h \leq H_{k+1}$，因此这是单个过滤器，对于所有的过滤器应该有H_{k+1}个，所以才会得到下图的$\mathbf{X}^{k+1} \in \mathbb{R}^{H_{k+1} \times D}$】。如下图所示，过滤器沿着Embedding维度（D）滑动。然后得到隐藏向量$\mathbf{X}^{k+1}_{i,*} \in \mathbb{R}^{D}$，这被称为一个**「特征映射」**（feature map）。



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

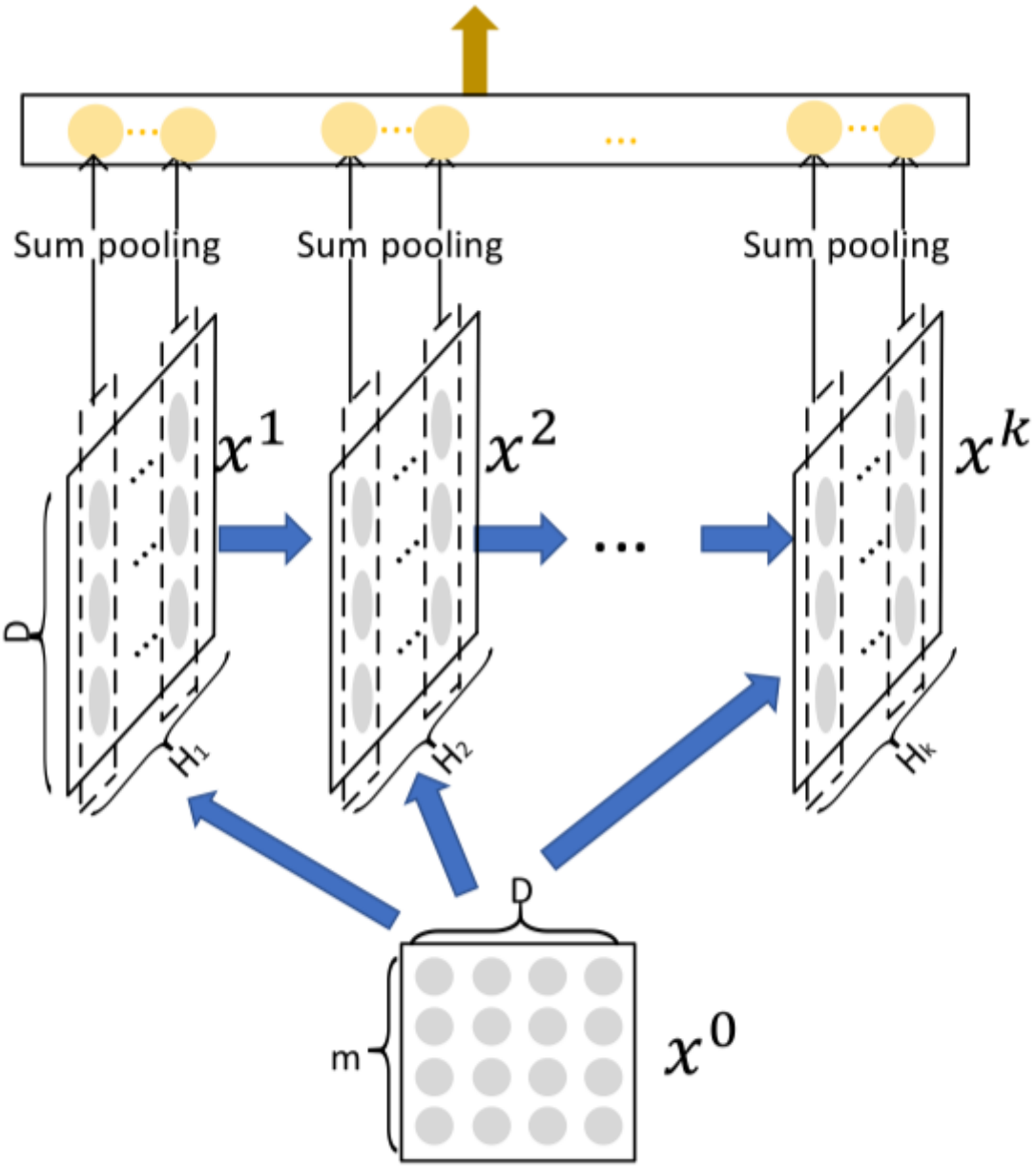
因此$\mathbf{X}^k$是H^k个**「不同特征映射的集合」**。CIN网络中的压缩（compressed）指的便是第k个隐藏层将$H_{k-1} \times m$向量空间压缩至H_k向量。

c）下图是整个CIN的结构，定义T为整个网络的深度。每个隐藏层为$\mathbf{X}^k, k \in [1, T]$，对于第$k$层，将所有的特征映射进行一个池化操作（sum pooling）【例如对上图Feature map 1向量进行一个累加】：

$$p_i^k = \sum_{j=1}^{D} \mathbf{X}_{i,j}^k \quad i \in [1, H_k]$$

因此便得到一个池化向量$\mathbf{p}^k = [p_1^k, p_2^k, \ldots, p_{H_k}^k]$对于第$k$隐藏层。

最后在对于所有的隐藏层的池化向量进行一个拼接：$\mathbf{p}^+ = [\mathbf{p}^1, \mathbf{p}^2, \ldots, \mathbf{p}^T] \in \mathbb{R}^{\sum_{i=1}^{T} H_i}$



(c) An overview of the CIN architecture.

【注】：此时文章没有提到如何来确定$H_1, H_2, \ldots, H_T$的大小，其实是**一组超参数**。

CIN复杂度分析

「空间复杂度」

主要的学习参数就是$\mathbf{W}^{k,h}$，经过上述分析，第k层的共有$H_k \times H_{k-1} \times m$个参数。假设包括CIN经过一个二元分类任务，那么CIN总共的学习参数为：

$$\sum_{k=1}^{T} H_k \times (1 + H_{k-1} \times m)$$

为了和DNN做一个对比，T层的DNN的学习参数应该为 $m \times D \times H_1 + \sum_{k=2}^{T} H_{k-1} \times H_k + H_T$，参数会随着$D$的增加而增加。对于CIN，通常$m$与$H_k$不会很大，因此$\mathbf{W}^{k,h}$的参数量是可以接受的。【这是为了说明CIN的可行性】

当然为了降低参数数量，作者还提到可以将$\mathbf{W}^{k,h}$进行分解为$\mathbf{U}^{k,h} \in \mathbb{R}^{H_{k-1} \times L}$和$\mathbf{V}^{k,h} \in \mathbb{R}^{m \times L}$【类似于FM】：

$$\mathbf{W}^{k,h} = \mathbf{U}^{k,h} \left(\mathbf{V}^{k,h}\right)^T$$

且$L \ll H \text{ and } L \ll m$。如果假设每层的特征映射数量相同，为$H$，那么这样就可以将空间复杂度从$O(mTH^2)$降低到$O(MTHL + TH^2L)$。

「时间复杂度」

计算$\mathbf{Z}^{k+1}$的时间复杂度为$O(mHD)$，那么对于T层CIN的总时间复杂度为$O(mH^2DT)$。对于DNN，时间复杂度为$O(mHD + H^2T)$，因此文章指出CIN的缺陷是时间复杂度比DNN高。

优点

（1）交互是向量的交互，不是（bit-wise）的交互；

（2）高阶特征交互是显示的；

（3）网络的复杂性不会随着交互程度的增加而呈指数增长；

2.4 联合DNN

由于CIN学习的是高阶显示特征，DNN学习高阶隐式特征。因此将两者结合，会使模型更加的健壮。该模型被称为xDeepFM，因为模型

- 包含了低阶和高阶特征的交互；
- 包含了隐式特征交互和显示特征交互；

最后输出单元为：

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

其中$\mathbf{a}$表示原生特征，$\mathbf{x}_{dnn}^{k}, \mathbf{p}^{+}$表示DNN、CIN的输出。

对于二元分类，损失函数为对数损失：

$$\mathcal{L}=-\frac{1}{N} \sum_{i=1}^{N} y_{i} \log \hat{y}_{i}+\left(1-y_{i}\right) \log \left(1-\hat{y}_{i}\right)$$

N为总样本数量，优化目标来最小化目标函数（目标函数即是损失函数+正则化）：

$$\mathcal{J}=\mathcal{L}+\lambda_{*}\|\Theta\|$$

2.5 与FM、DeepFM的关系

对于xDeepFM模型，我们假设CIN的深度与特征映射的数量都为1，则**「xDeepFM就相当于DeepFM的一个泛化」**。当进一步删除DNN部分，并且对于特征映射使用一个sum filter，那xDeepFM就将为一个传统的FM模型。【联想之前$\mathbf{W}^{k,h}$的分解】

3. 代码复现

xDeepFM模型最主要的部分就是CIN网络结构。关于$\mathbf{X}_{i,*}^{k-1} \circ \mathbf{X}_{j,*}^{0}$的计算不能直接按照公式进行编程。**「因为两者每层数量不同，只能使用for循环进行计算」**，那样就极大的降低了模型的训练效率。参考了浅梦大佬deepctr中CIN的编写，**「通过将$\mathbf{X}_{i}^{k}$在embedding维度上进行分割」**，然后再通过矩阵运算依旧能达到原有的目的。（原文开源代码可能也是这样写的）

```
class CIN(layers.Layer):
    """
    CIN part
    """

    def __init__(self, cin_size, l2_reg):
        super(CIN, self).__init__()
        self.cin_size = cin_size

    def build(self, input_shape):
        self.embedding_nums = input_shape[1]
        self.field_nums = list(self.cin_size)
        self.field_nums.insert(0, self.embedding_nums)
        # filters
        self.cin_W = {
            'CIN_W_' + str(i): self.add_weight(
                name='CIN_W_' + str(i),
```

```
                shape=(1, self.field_nums[0] * self.field_nums[i], self.field_nums[i + 1]),
                initializer='random_uniform',
                regularizer=l2(0.00001),
                trainable=True)
            for i in range(len(self.field_nums) - 1)
        }

    def call(self, inputs, **kwargs):
        dim = inputs.shape[-1]
        hidden_layers_results = [inputs]
        split_X_0 = tf.split(hidden_layers_results[0], dim, 2)
        for idx, size in enumerate(self.cin_size):
            split_X_K = tf.split(hidden_layers_results[-1], dim, 2)
            result_1 = tf.matmul(split_X_0, split_X_K, transpose_b=True)
            result_2 = tf.reshape(result_1, shape=[dim, -1, self.embedding_nums * self.field_nums
            result_3 = tf.transpose(result_2, perm=[1, 0, 2])
            result_4 = tf.nn.conv1d(input=result_3, filters=self.cin_W['CIN_W_' + str(idx)], stri
                                    padding='VALID')
            result_5 = tf.transpose(result_4, perm=[0, 2, 1])
            hidden_layers_results.append(result_5)

        final_results = hidden_layers_results[1:]
        result = tf.concat(final_results, axis=1)
        result = tf.reduce_sum(result,  axis=-1)
        return result
```

具体代码见：https://github.com/ZiyaoGeng/Recommender-System-with-TF2.0，或直接点击阅读原文。

4. 总结

xDeepFM最主要的内容是CIN网络结构，是对特征交互的创新。

往期精彩回顾

Github开源项目2.0---使用TF2.0对经典推荐论文进行复现【持续更新中...】

【论文导读】浅谈胶囊网络与动态路由算法