推荐系统入门系列(八)-xDeepFM理论与实战

何无涯 何无涯的技术小屋 7月16日

只要功夫深，铁杵磨成针。

—— 佚名

一、xDeepFM算法思想

xDeepFM原始论文：《xDeepFM: Combining Explicit and Implicit Feature Interactions for Recommender Systems》：https://arxiv.org/abs/1803.05170。

前面的文章中说到，DCN为了**显示地**、**自动地**构造**有限高阶**特征，引入Cross Network取代了Wide&Deep中的Wide层。不过，有一点需要注意，DCN的Cross Network接在Embedding层之后，虽然可以**显示的构造高阶特征**，但是它是以**bit-wise**的方式。例如，Age Field对应嵌入向量<a1,b1,c1>，Occupation Field对应嵌入向量<a2,b2,c2>，在Cross Network中，向量<a1,b1,c1>和向量<a2,b2,c2>会拼接后直接作为输入，即它意识不到Field Vector的概念。Cross Network以bit为最细粒度，而FM以向量为最细粒度学习相关性，即vector-wise。xDeepFM的动机，正是将FM的vector-wise的思想引入Cross部分。

二、xDeepFM模型结构

xDeepFM的整体结构如图1所示，基本框架依然类似于Wide&Deep模型，其中Linear、Plain DNN分别类似于Wide和Deep部分，而CIN部分正是xDeepFM的重点。

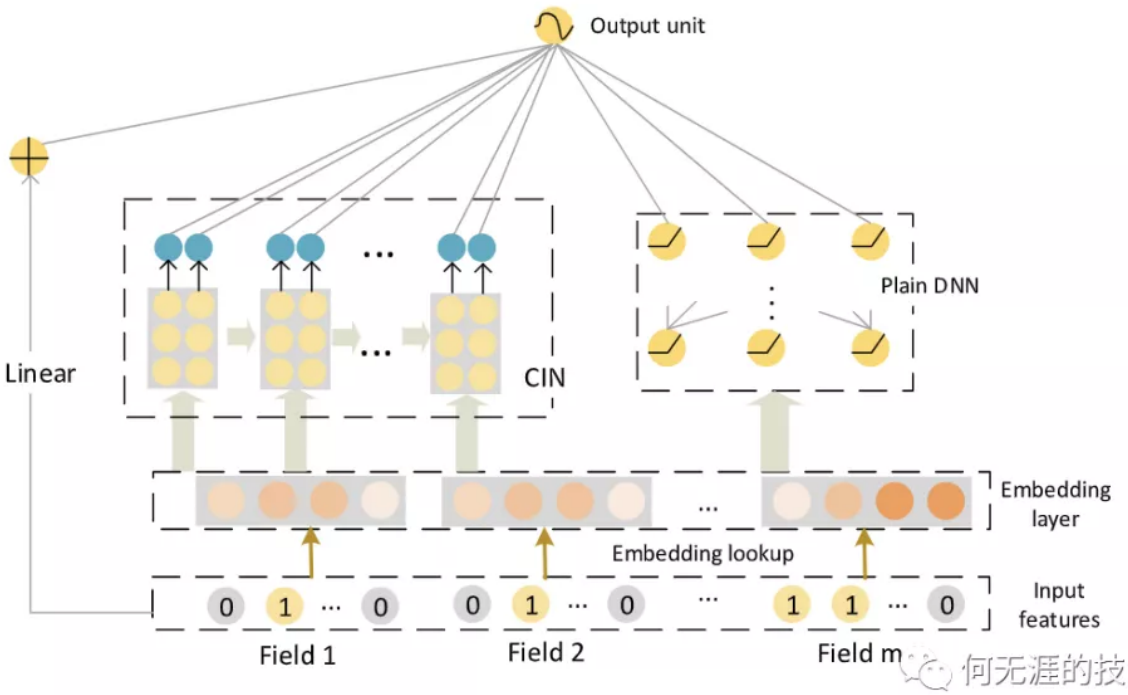


图1 xDeepFM网络结构图

接下来将详细介绍CIN这一部分是怎么干的。CIN，全称是Compressed Interaction Network，如图2所示，CIN层的输入来自Embedding层，假设有m个field，每个field的embedding vector维度为D，则输入矩阵可表示为X0，为m*D。

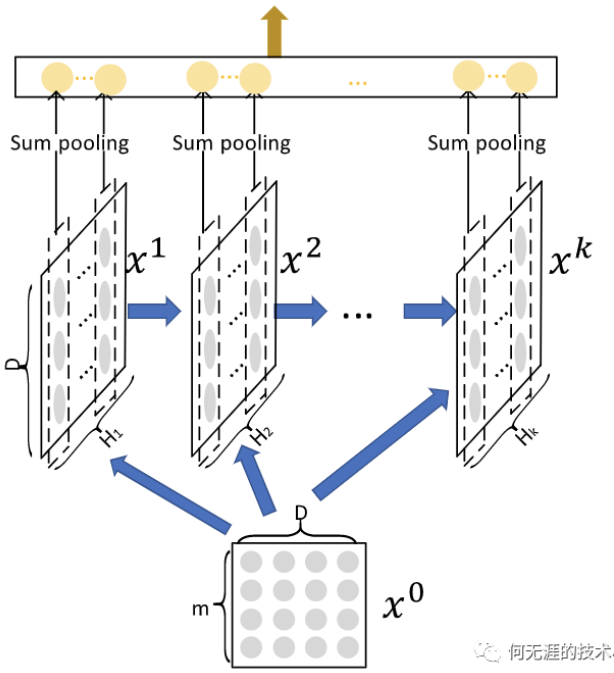


图2 CIN模型结构图

粗略一看，CIN的结构有k层，然后都做了sum pooling，那么具体的CIN层做了什么呢?

令 $\boldsymbol{X}^k \in \mathbb{R}^{H_k * D}$ 表示第 k 层的输出，其中 H_k 表示第 k 层的vector个数，vecor维度始终为 D ，保持和输入层一致。具体地，第 k 层每个vector的计算方式为：

$$\boldsymbol{X}_{h,*}^k = \sum_{i=1}^{H_{k-1}} \sum_{j=1}^{m} \boldsymbol{W}_{ij}^{k,h} (\boldsymbol{X}_{i,*}^{k-1} \circ \boldsymbol{X}_{j,*}^0) \in \mathbb{R}^{1*D}, \quad where \ 1 \le h \le H_k \quad (1)$$

其中 $\boldsymbol{W}^{k,h} \in \mathbb{R}^{H_{k-1}*m}$ 表示第 k 层的第 h 个vector的权重矩阵， $\circ$ 表示Hadamard乘积，即逐元素乘，例如 $< a_1, b_1, c_1 > \circ < a_2, b_2, c_2 > = < a_1 b_1, a_2 b_2, a_3 b_3 >$ 。

看懂了这个计算公式，就理解了图2的CIN结构，我们先看这个公式到底干了什么：

1. 取**前一层** $\boldsymbol{X}^{k-1} \in \mathbb{R}^{H_{k-1}*D}$ 中的 H_{k-1} 个vector，与**输入层** $\boldsymbol{X}^0 \in \mathbb{R}^{m*D}$ 中的 m 个vector，进行两两Hadamard乘积运算，得到 $H_{k-1} * m$ 个 vector，然后加权求和。
2. 第 k 层的不同vector区别在于，对这 $H_{k-1} * m$ 个 vector 求和的权重矩阵不同。 H_k 即对应有多少个不同的权重矩阵 $\boldsymbol{W}^k$ ，是一个可以调整的超参。

为什么这么设计呢，好处是什么？CIN与DCN中Cross层的设计动机是相似的，Cross层的input也是前一层与输出层。这种做法的优点是：**有限高阶、自动叉乘、参数共享。**

再来看看CIN与Cross的几个主要差异：

1. Cross是bit-wise的，而CIN是vector-wise的

2.在第l层，Cross包含从1阶～（l+1）阶的所有组合特征，而CIN只包含（l+1）阶的组合特征。相应的Cross在输出层输出全部结果，而CIN在每层都输出中间结果。

其中造成差异2的原因是，Cross层计算公式中除了与CIN一样包含"上一层与输入层的x乘"外，会额外"+输入层"，这是涵盖所有阶特征的不同策略。

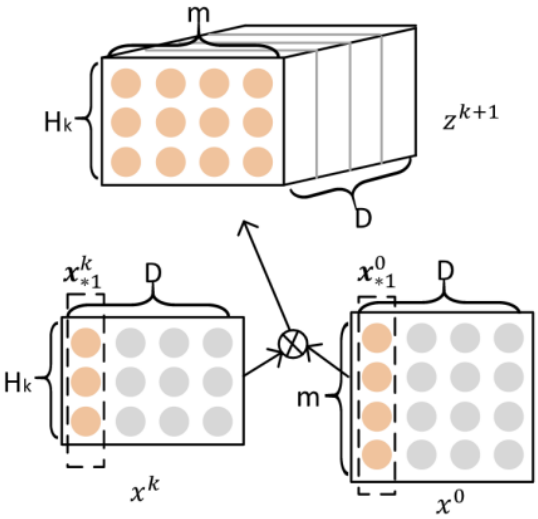
注意图2的CIN结构，可以思考两个问题，这涉及到CIN的另一位亲戚FM：

1. 每层通过**sum pooling**对vector的元素加和输出，这么做的意义或合理性？可以设想，如果CIN只有1层， 只有m个vector，即 $H_1 = m$ ，且加和的权重矩阵恒等于1，即 $\boldsymbol{W}^1 = \mathbf{1}$ ，那么sum pooling的输出结果，就是一系列的两两向量内积之和，即标准的FM（不考虑一阶与偏置）。
2. 除了第1层，中间层的这种基于**vector高阶组合**有什么物理意义？回顾FM，虽然是二阶的，但可以扩展到多阶，例如考虑三阶FM，是对三个嵌入向量作Hadamard乘再对得到的vector作sum，CIN基于vector-wise的高阶组合再作sum pooling与之是类似的，这也是模型名字"**eXtreme Deep Factorization Machine** (xDeepFM) "的由来。

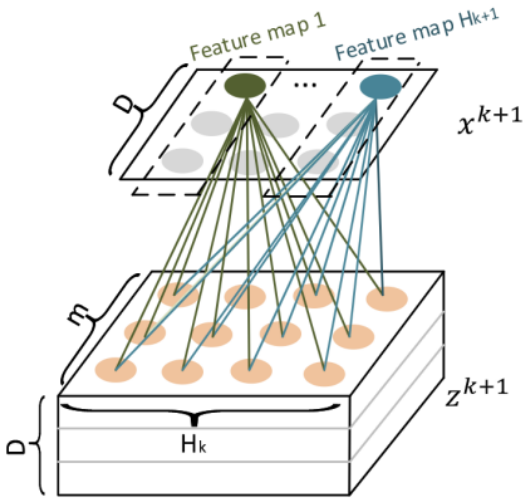
为什么取名CIN呢?

CIN名字由来与它特定的计算方式有关，不感兴趣的读者可以直接跳过这部分，不影响模型理解。回顾式（1），同层不同vector的区别仅仅在于不同的加和权重矩阵 $\boldsymbol{W}$ ，我们可以提前计算好两两向量间Hadamard乘的结果。

具体的方式如下图所示，首先如图 a 计算中间结果—— tensor $\boldsymbol{Z}^{k+1}$ ，然后使用权重矩阵 $\boldsymbol{W}^{k,i} \in \mathbb{R}^{H_k * m}$ 顺着tensor的维度 D ，逐层相乘加和，得到 k+1 层的第 i 个vector，如图 b 所示。如果把 $\boldsymbol{W}$ 看成filter，这和CNN的方式很像。可以看到，最后 $\boldsymbol{Z}^{k+1}$ 被压缩成了一个矩阵，这是名字中"Compressed"的由来。



(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*).

复杂度分析

假设CIN和DNN每层神经元/向量个数都为 H ，网络深度为 T 。那么CIN的参数空间复杂度为 $O(mTH^2)$ ，普通的DNN为 $O(mDH+TH^2)$ ，CIN的空间复杂度与输入维度 D 无关，此外，如果有必要，CIN还可以对权重矩阵 $\boldsymbol{W}$ 进行 L 阶矩阵分解从而能降低空间复杂度。

CIN的时间复杂度就不容乐观了，按照上面介绍的计算方式为 $O(mH^2DT)$ ，而DNN为 $O(mDH+TH^2)$ ，**时间复杂度会是CIN的一个主要痛点。**

三、xDeepFM代码实现

xDeepFM的PyTorch实现如下：

```
class ExtremeDeepFactorizationMachineModel(torch.nn.Module):
    """
    A pytorch implementation of xDeepFM.
    Reference:
        J Lian, et al. xDeepFM: Combining Explicit and Implicit Feature Inter
    """

    def __init__(self, field_dims, embed_dim, mlp_dims, dropout, cross_layer_
        super().__init__()
        self.embedding = FeaturesEmbedding(field_dims, embed_dim)
        self.embed_output_dim = len(field_dims) * embed_dim
        self.cin = CompressedInteractionNetwork(len(field_dims), cross_layer_
        self.mlp = MultiLayerPerceptron(self.embed_output_dim, mlp_dims, drop
        self.linear = FeaturesLinear(field_dims)

    def forward(self, x):
        """
        :param x: Long tensor of size ``(batch_size, num_fields)``
        """
        embed_x = self.embedding(x)
        x = self.linear(x) + self.cin(embed_x) + self.mlp(embed_x.view(-1, se
        return torch.sigmoid(x.squeeze(1))
```

完整的代码可以参考我的github: https://github.com/yyHaker/RecommendationSystem。

小结：xDeepFM将基于Field的vector-wise思想引入Cross，并且保留了Cross的优势，模型结构也很优雅，实验效果提升也明显。如果是DeepFM只是“Deep&FM”，那么xDeepFM就真正做到了“Deep” Factorization Machine。但是xDeepFM的时间复杂度有点高，在工业应用落地会有一个性能瓶颈，需要重点优化。