推荐系统入门系列(六)-深度排序模型之串型结构AFM、NFM、PNN

何无涯 何无涯的技术小屋 7月14日

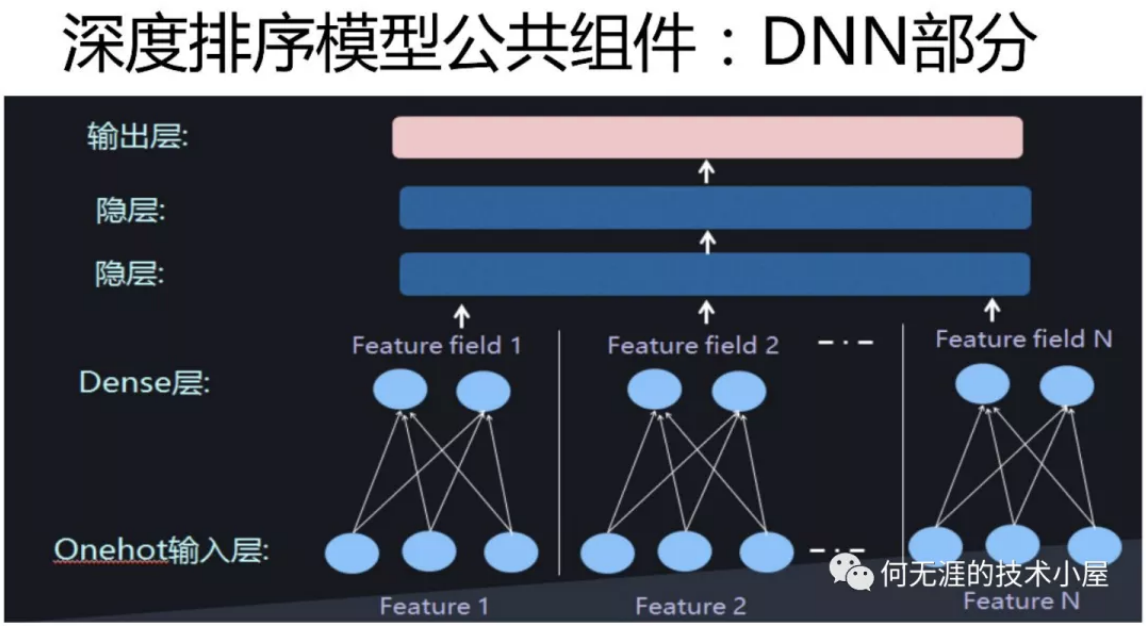
一日一钱,十日十钱。绳锯木断,水滴石穿。

—— 班固

一、深度排序模型分类

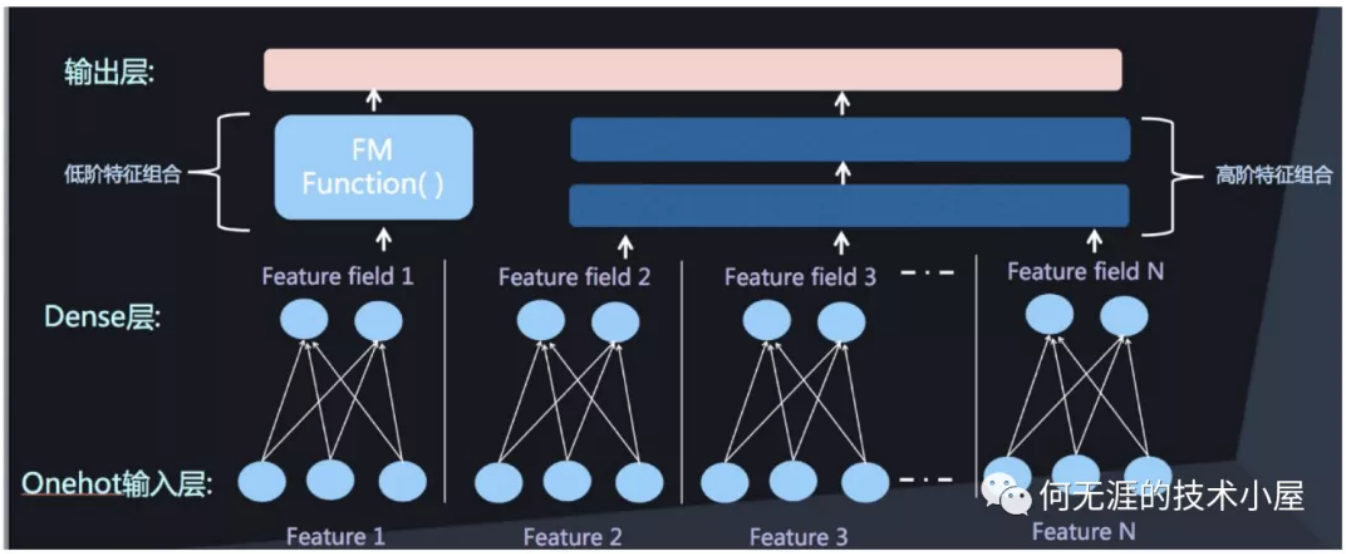
在CTR预估中，为了解决稀疏特征的问题，学者们提出了FM模型来建模特征之间的交互关系，但是FM模型只能表达特征之间的两两组合之间的关或者说是低阶特征组合，无法建模特征之间的更深层次的交互关系，因此学者们通过DNN来建模更高阶的特征之间的关系。

因此，FM和深度神经网络DNN的结合就成为了CTR预估问题以及深度排序模型中主流的方法。深度排序模型的模型结构中都有深度学习的DNN部分，什么意思呢？也就是说特征输入模型中，首先将它转换成embedding，然后再在上面套两个隐层进行预测，这是所有深度排序模型公有的一部分，几乎无一例外，如下图所示：

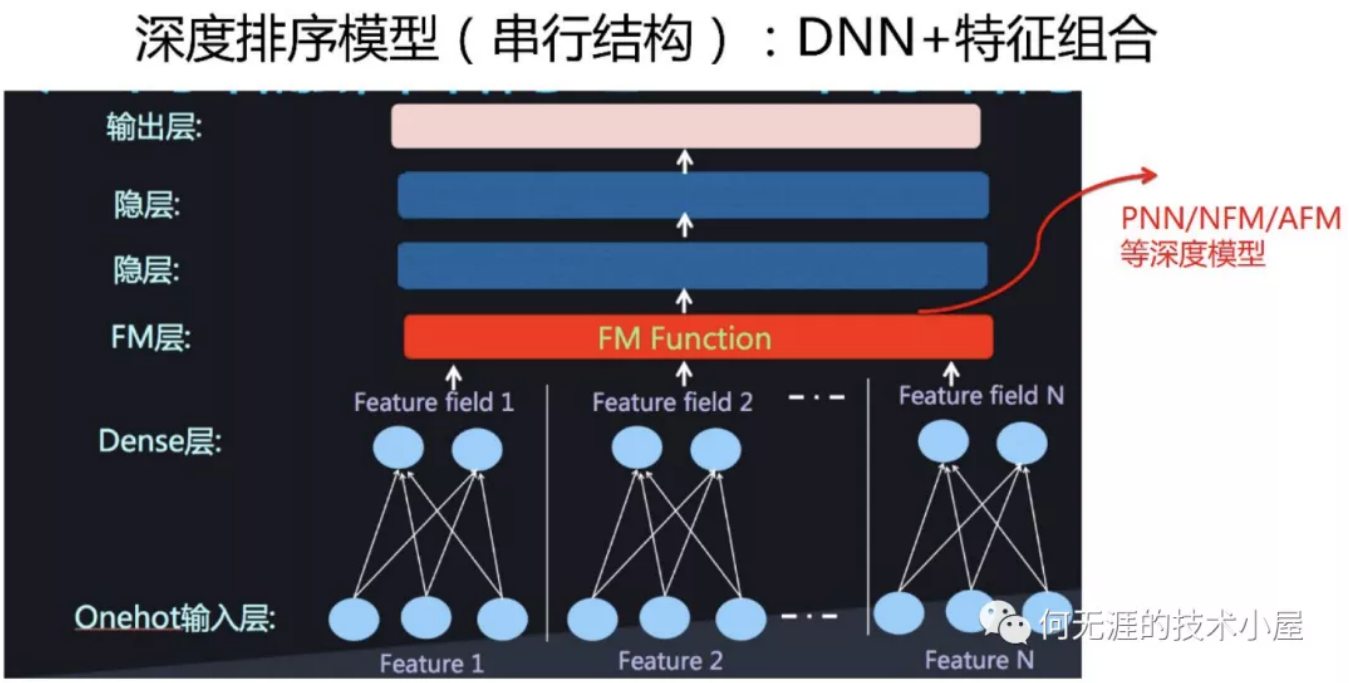


有关FM和深度神经网络的结合有两种主流的方法：并行结构和串型结构。

并行结构中，FM部分和DNN部分分开计算（如下图），只在输出层进行一次融合得到结果，这种结构常见的模型有DeepFM、DCN、Wide&Deep模型。



串行结构中，将FM的一次项和二次项结果（或其中之一）作为DNN的输入，经DNN得到最终结果（如下图），这种结构常见的模型有AFM、NFM、PNN模型。



这一小节将介绍几种经典的串行结构模型。

二、串型结构之NFM模型

1. NFM的基本思想

NFM原始论文：Neural Factorization Machines for Sparse Predictive Analytics：https://arxiv.org/pdf/1708.05027

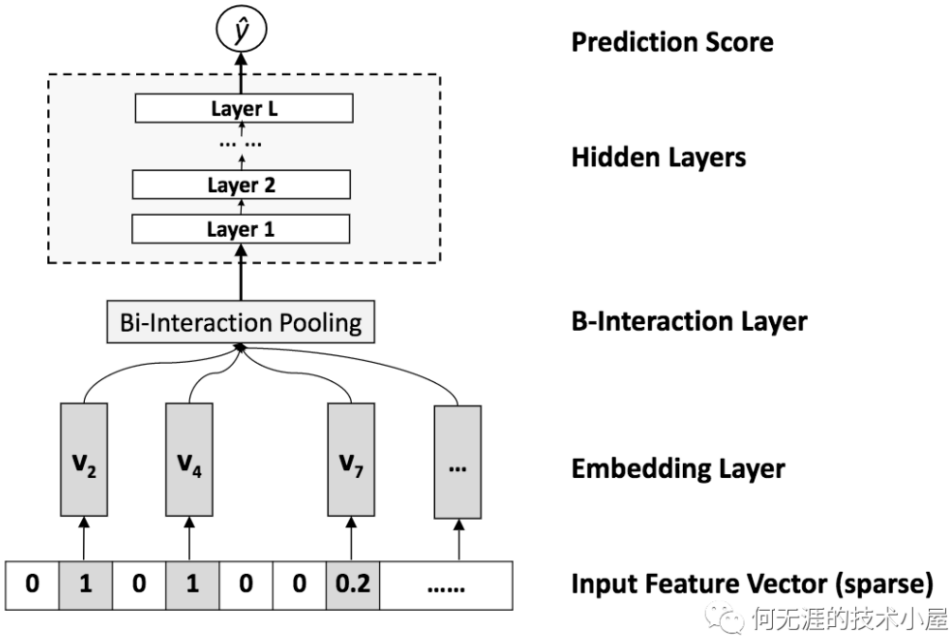
FM模型能够建模二阶特征交互，而深度神经网络可以建模高阶特征交互，那么很简单的想法，能不能将二者结合呢？NFM就是这么干的，而且以串行的方式将FM的输出直接接深度神经网络，非常简单。

2. NFM的结构

类似于FM模型，NFM模型目标值的预测公式为：

$$\hat{y}_{NFM}(\mathbf{x}) = w_0 + \sum_{i=1}^{n} w_i x_i + f(\mathbf{x}),$$

可以看到前面两项和FM模型基本相同，不同的是后面的f(x)，f(x)是用来建模特征之间交互关系的多层前馈神经网络模块，NFM模型的基本结构如下图所示，



Embedding Layer和FM模型是一样的，Bi-Interaction Layer是计算FM中的二次项的过程，因此得到向量的维度就是Embedding层的向量的维度，最终的结果是：

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

Hidden Layers就是DNN部分，将Bi-interaction Layer得到的结果接入多层的神经网络进行训练，从而捕获到特征之间的高阶的特征交互，最后得到预测的输出。

整体的用公式表示是：

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

3. NFM模型的实现

NFM模型PyTorch实现的代码如下：

```
class NeuralFactorizationMachineModel(torch.nn.Module):
    """
    A pytorch implementation of Neural Factorization Machine.
```

```
    Reference:
        X He and TS Chua, Neural Factorization Machines for Sparse Predictive
    """

    def __init__(self, field_dims, embed_dim, mlp_dims, dropouts):
        super().__init__()
        self.embedding = FeaturesEmbedding(field_dims, embed_dim)
        self.linear = FeaturesLinear(field_dims)
        self.fm = torch.nn.Sequential(
            FactorizationMachine(reduce_sum=False),
            torch.nn.BatchNorm1d(embed_dim),
            torch.nn.Dropout(dropouts[0])
        )
        self.mlp = MultilayerPerception(embed_dim, mlp_dims, dropouts[1])

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

三、串型结构之AFM模型

1. AFM的基本思想

AFM原始论文：Attentional Factorization Machines: Learning the Weight of Feature Interactions via Attention Networks：https://arxiv.org/pdf/1708.04617

这里先简单回顾一下FM模型。FM模型为了学习到特征之间的交互关系，为每一个特征学习了一个向量，并将两两组合特征的向量的内积作为组合特征的权重。用公式表示如下：

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

为了计算方便，化简过程如下：

$$\sum_{i=1}^{n}\sum_{j=i+1}^{n}\langle \mathbf{v}_i,\mathbf{v}_j\rangle x_i x_j$$

$$=\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\langle \mathbf{v}_i,\mathbf{v}_j\rangle x_i x_j-\frac{1}{2}\sum_{i=1}^{n}\langle \mathbf{v}_i,\mathbf{v}_i\rangle x_i x_i$$

$$=\frac{1}{2}\left(\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{f=1}^{k} v_{i,f}\,v_{j,f}\,x_i x_j-\sum_{i=1}^{n}\sum_{f=1}^{k} v_{i,f}\,v_{i,f}\,x_i x_i\right)$$

$$=\frac{1}{2}\sum_{f=1}^{k}\left(\left(\sum_{i=1}^{n} v_{i,f}\,x_i\right)\left(\sum_{j=1}^{n} v_{j,f}\,x_j\right)-\sum_{i=1}^{n} v_{i,f}^2\,x_i^2\right)$$

$$=\frac{1}{2}\sum_{f=1}^{k}\left(\left(\sum_{i=1}^{n} v_{i,f}\,x_i\right)^2-\sum_{i=1}^{n} v_{i,f}^2\,x_i^2\right)$$

可以看到，如果不考虑最外层求和的话，我们得到了一个K维的向量。所以，不难发现，FM模型其实是给每个特征学习一个特定的向量，当这个特征与其他特征进行交叉时，都是用同样的向量进行计算。这是很不合理的，因为不容特征之间的交叉，重要程度是不一样的，如何体现这种重要程度？之前介绍的FFM模型就是一种解决方案。那么还有没有其他办法呢？有，Attention机制，因为Attention机制相当于一种加权平均，attention的值就是其中权重，用来判断不同特征之间交互的重要性。

2. AFM的结构

AFM说白了就是加入了Attention机制的FM，刚才也提到过，Attention相当于加权的过程，因此使用公式表示为：

$$y'_{AFM}=w_0+\sum_{i=1}^{n} w_i x_i+p^T\sum_{i=1}^{n}\sum_{j=i+1}^{n} a_{ij}(v_i\odot v_j)x_i x_j$$

其中有个点的符号代表哈达马乘积，注意到还有一个p向量，因为后面那个式子求和之后得到的是一个K维的向量，乘以p向量得到的才是一个数值。

AFM模型的前两部分和FM是相同的，后面一项经由如下的网络得到，

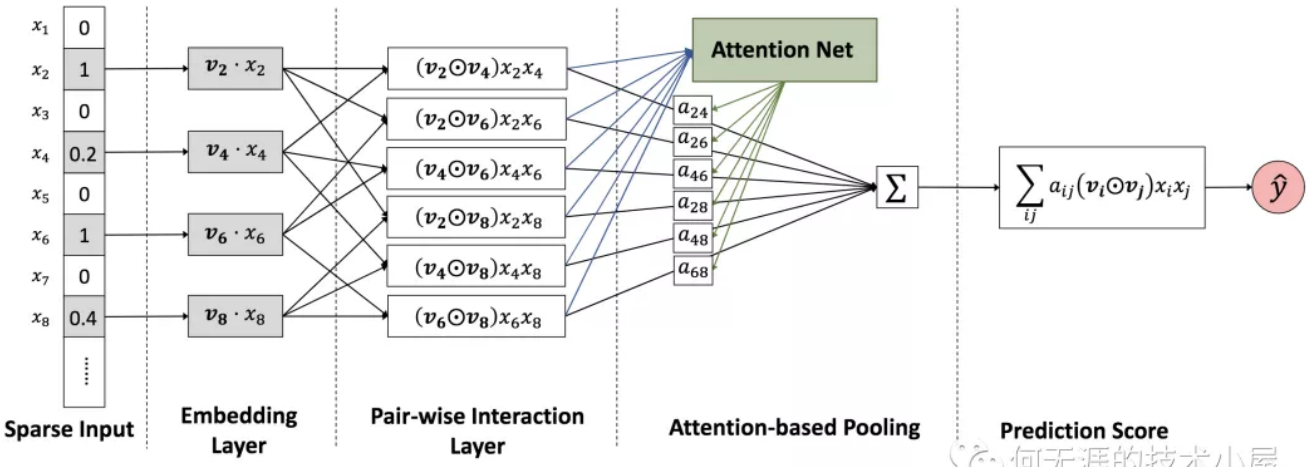


Figure 1: The neural network architecture of our proposed Attentional Factorization Machine model.

注意到图中的前三部分：sparse input、embedding layer、pair-wise interaction layer都是和FM一样的。而后面的两部分则是AFM的创新所在，也就是我们的Attention Net，Attention的计算公式如下：

$$a'_{ij} = \mathbf{h}^T ReLU(\mathbf{W}(\mathbf{v}_i \odot \mathbf{v}_j)x_i x_j + \mathbf{b}),$$

$$a_{ij} = \frac{\exp(a'_{ij})}{\sum_{(i,j)\in\mathcal{R}_x} \exp(a'_{ij})},$$

计算完Attention之后，对交叉特征进行加权求和，得到最后的预测结果。可以看到，AFM只是在FM的基础之上添加了Attention机制，但是实际上，由于最后的加权累加，二次项并没有进行更深度的网络去学习非线性交叉特征，所以AFM并没有发挥出 DNN的优势，也许结合DNN能够达到更好的效果。

3. AFM的实现

AFM的PyTorch代码实现如下：

```
class AttentionalFactorizationMachine(torch.nn.Module):
    """Attentiona Factorization Machine"""
    def __init__(self, embed_dim, attn_size, dropouts):
        super().__init__()
        self.attention = nn.Linear(embed_dim, attn_size)
        self.projection = nn.Linear(attn_size, 1)
        self.fc = nn.Linear(embed_dim, 1)
        self.dropouts = dropouts

    def forward(self, x):
        """
        :param x: Float tensor of size ``(batch_size, num_fields, embed_dim)``
        """
        num_fields = x.shape[1]
```

```
        row, col = list(), list()
        for i in range(num_fields - 1):
            for j in range(i + 1, num_fields):
                row.append(i), col.append(j)
        p, q = x[:, row], x[:, col]
        inner_product = p * q
        attn_scores = F.relu(self.attention(inner_product))
        attn_scores = F.softmax(self.projection(attn_scores), dim=1)
        attn_scores = F.dropout(attn_scores, p=self.dropouts[0])
        attn_output = torch.sum(attn_scores * inner_product, dim=1)
        attn_output = F.dropout(attn_output, p=self.dropouts[1])
        return self.fc(attn_output)
```

四、串型结构之PNN模型

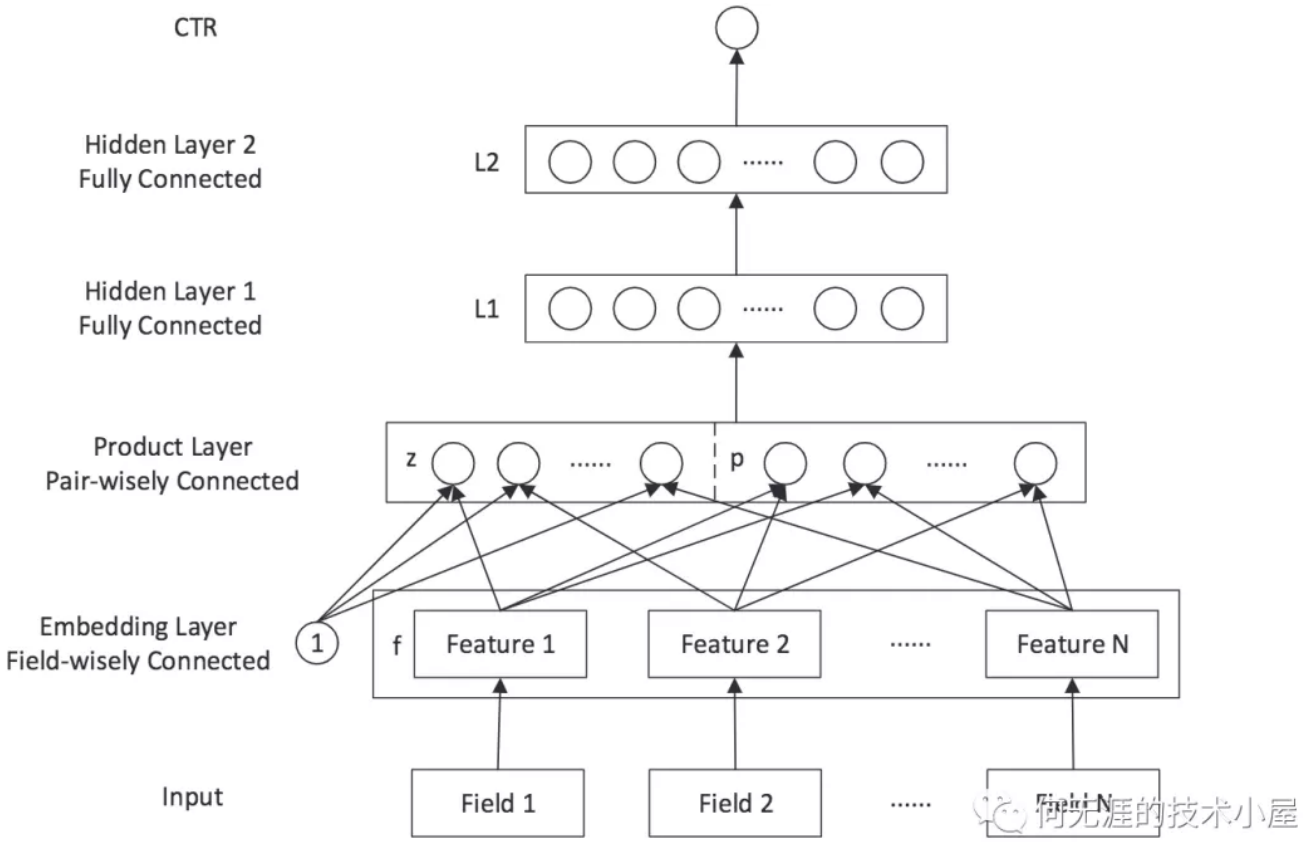
1. PNN的基本思想

PNN原始论文：Product-based Neural Networks for User Response Prediction：https://arxiv.org/pdf/1611.00144

PNN，全称Product-based Neural Network，认为巨大的特征空间导致的高纬输入并不能直接输入到DNN等深度神经网络中来捕获高阶特征，因此PNN提出一个product layer来捕获类别特征之间的交互，然后再接深度神经网络来捕获高阶特征交互。

2. PNN的结构

PNN的基本结构如下图所示，



Input和Embedding layer这个和之前的模型没啥区别，product layer负责捕获类别特征之间的交互，最后再过深度神经网络DNN捕获高阶特征交互，PNN的重点的就是是这里的product layer。

另外注意到，全连接层L1的输入是Product Layer的输出，公式如下：

$$\boldsymbol{l}_1 = \text{relu}(\boldsymbol{l}_z + \boldsymbol{l}_p + \boldsymbol{b}_1),$$

product layer思想来源于，在CTR预估中，认为特征之间的关系更多的是一种“且(and)”的关系，而非“加(add)”的关系。例如，性别为男且喜欢游戏的人群，性别男和喜欢游戏的人群，前者的组合比后者更能体现特征交叉的意义。

product layer可以分成两个部分，一部分是线性部分lz，一部分是非线性部分lp。二者的计算方式如下：

$$\boldsymbol{l}_z = \left(l_z^1, l_z^2, \ldots, l_z^n, \ldots, l_z^{D_1}\right), \qquad l_z^n = \boldsymbol{W}_z^n \odot \boldsymbol{z}$$
$$\boldsymbol{l}_p = \left(l_p^1, l_p^2, \ldots, l_p^n, \ldots, l_p^{D_1}\right), \qquad l_p^n = \boldsymbol{W}_p^n \odot \boldsymbol{p}$$

注意上面的公式，z和p都是通过Embedding层得到的，其中z是线性信号向量，直接可通过embedding层得到：

$$\boldsymbol{z} = \left(\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_N\right) \triangleq \left(\boldsymbol{f}_1, \boldsymbol{f}_2, \ldots, \boldsymbol{f}_N\right),$$

可以认为这里直接是embedding层的复制。

而对于p来说，这里需要一个公式进行映射，

$$\boldsymbol{p} = \{\boldsymbol{p}_{i,j}\}, i = 1...N, j = 1...N,$$

$$\boldsymbol{p}_{i,j} = g(\boldsymbol{f}_i, \boldsymbol{f}_j)$$

那么这个映射g是什么呢？不同的g的选择使得有两种PNN的计算方法，一种叫做Inner PNN，简称IPNN，一种叫做Outer PNN，简称OPNN。接下来将详细介绍这两种形式的PNN模型，为了方便，定义Embedding的大小为M，filed的大小为N，而lz和lp的长度为D1。

第一种是IPNN，g使用内积运算。因为pij是一个数，得到一个pij的时间复杂度是M，p的大小是N*N，所以得到p的时间复杂度是N*N*M。而再由p得到lp的时间复杂度是N*N*D1，所以总的时间复杂度为N*N*(D1+M)。

受益于FM模型的思想，论文对IPNN的计算进行了简化，可以看到p是一个对称矩阵，所以权重矩阵W也是一个对称矩阵，而对称矩阵就可以进行如下分解：

$$\boldsymbol{W}_p^n = \boldsymbol{\theta}^n \boldsymbol{\theta}^{nT},$$

因此，

$$\boldsymbol{W}_p^n \odot \boldsymbol{p} = \sum_{i=1}^{N}\sum_{j=1}^{N} \theta_i^n \theta_j^n \langle \boldsymbol{f}_i, \boldsymbol{f}_j \rangle = \langle \sum_{i=1}^{N} \boldsymbol{\delta}_i^n, \sum_{i=1}^{N} \boldsymbol{\delta}_i^n \rangle$$

其中：

$$\boldsymbol{\delta}_i^n = \theta_i^n \boldsymbol{f}_i$$

因此，

$$\boldsymbol{\delta}^n = \left(\boldsymbol{\delta}_1^n, \boldsymbol{\delta}_2^n, \ldots, \boldsymbol{\delta}_i^n, \ldots, \boldsymbol{\delta}_N^n\right) \in \mathbb{R}^{N \times M}.$$

从而得到：

$$\boldsymbol{l}_p = \left(\| \sum_i \boldsymbol{\delta}_i^1 \|, \ldots, \| \sum_i \boldsymbol{\delta}_i^n \|, \ldots, \| \sum_i \boldsymbol{\delta}_i^{D_1} \|\right).$$

所以总的时间复杂度是D1*M*N。

第二种是OPNN，g的计算方式是外积，计算公式如下：

$$g(\boldsymbol{f}_i, \boldsymbol{f}_j) = \boldsymbol{f}_i \boldsymbol{f}_j^T$$

此时pij为M*M的矩阵，就算一个pij的时间复杂度为M*M，而p是N*N的矩阵，因此计算**p**的时间复杂度为N*N*M*M，从而计算lp的时间复杂度为D1*N*N*M*M。这个显然复杂度很高，为了减少复杂度，论文使用了叠加的思想，它重新定义了**p**矩阵，公式如下：

$$\boldsymbol{p} = \sum_{i=1}^{N} \sum_{j=1}^{N} \boldsymbol{f}_i \boldsymbol{f}_j^T = \boldsymbol{f}_\Sigma (\boldsymbol{f}_\Sigma)^T, \quad \boldsymbol{f}_\Sigma = \sum_{i=1}^{N} \boldsymbol{f}_i,$$

此时的**p**是M*M的矩阵，所以权重矩阵Wp也是M*M矩阵，这样总的时间复杂度就变为D1*M*(M+N)。（有个小疑问，这个复杂度咋算的）

3. PNN的实现

PNN的Pytorch实现代码如下：

```
class ProductNeuralNetworkModel(torch.nn.Module):
    """
    A pytorch implementation of inner/outer Product Neural Network.
    Reference:
        Y Qu, et al. Product-based Neural Networks for User Response Predictio
    """

    def __init__(self, field_dims, embed_dim, mlp_dims, dropout, method='inner
        super().__init__()
        num_fields = len(field_dims)
        if method == 'inner':
            self.pn = InnerProductNetwork()
        elif method == 'outer':
            self.pn = OuterProductNetwork(num_fields, embed_dim)
        else:
            raise ValueError('unknown product type: ' + method)
        self.embedding = FeaturesEmbedding(field_dims, embed_dim)
        self.linear = FeaturesLinear(field_dims, embed_dim)
        self.embed_output_dim = num_fields * embed_dim
        self.mlp = MultilayerPerception(num_fields * (num_fields - 1) // 2 + s

    def forward(self, x):
        """
```

```
        :param x: Long tensor of size ``(batch_size, num_fields)``
        """
        embed_x = self.embedding(x)
        cross_term = self.pn(embed_x)
        x = torch.cat([embed_x.view(-1, self.embed_output_dim), cross_term], d
        x = self.mlp(x)
        return torch.sigmoid(x.squeeze(1))
```

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

参考文章:

【1】FFM及DeepFFM模型在推荐系统中的探索 https://zhuanlan.zhihu.com/p/67795161

【2】推荐系统遇上深度学习(六)--PNN模型理论和实践：https://mp.weixin.qq.com/s?__biz=MzI1MzY0MzE4Mg==&mid=2247483907&idx=1&sn=5cce40cd2e96f2403d1d60aa2cc3e1e7&chksm=e9d012c2dea79bd46012fe78b9603204103f4139fa5c2c7577d7bf4231f2fa4c7bc762ee6a7c&scene=21#wechat_redirect