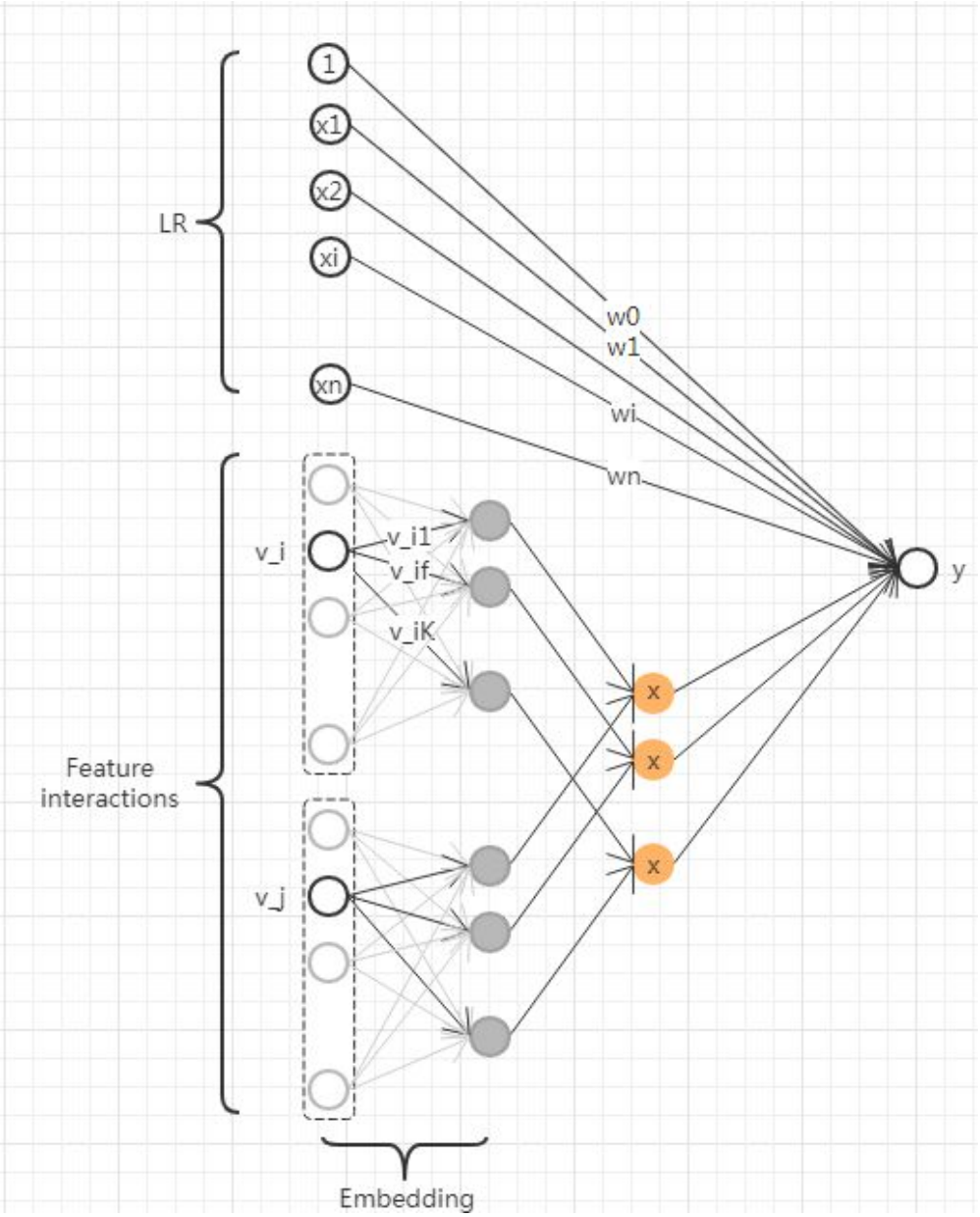
推荐系统遇上深度学习（*） - FNN&PNN

Notebook:	recommender		
Created:	10/6/2019 10:57 PM	**Updated:**	10/7/2019 2:14 AM
Author:	elvirasun28@outlook.com		
Tags:	rec		
URL:	https://zhuanlan.zhihu.com/p/33177517		

- FM 的神经网路形式
  - 其实FM 和FFM 都是可以从神经网路的角度理解

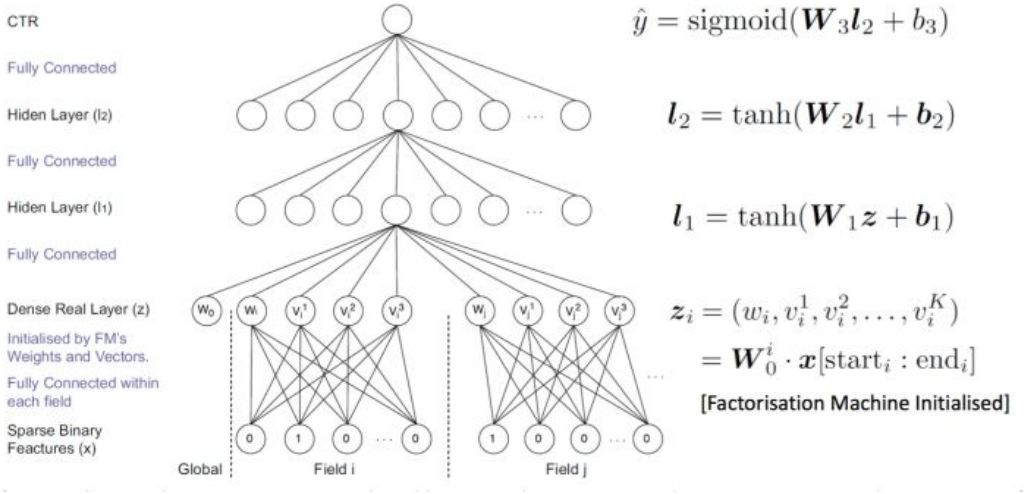


$$f(x) = logistics(linear(X) + \sum_{i=1}^{n} \sum_{j=i+1}^{n} < v_i, v_j > x_i x_j)$$

可以看成是三个神经网络。后面的二项式，可以看成是神经网路embedding后的每两个vector inner product

---

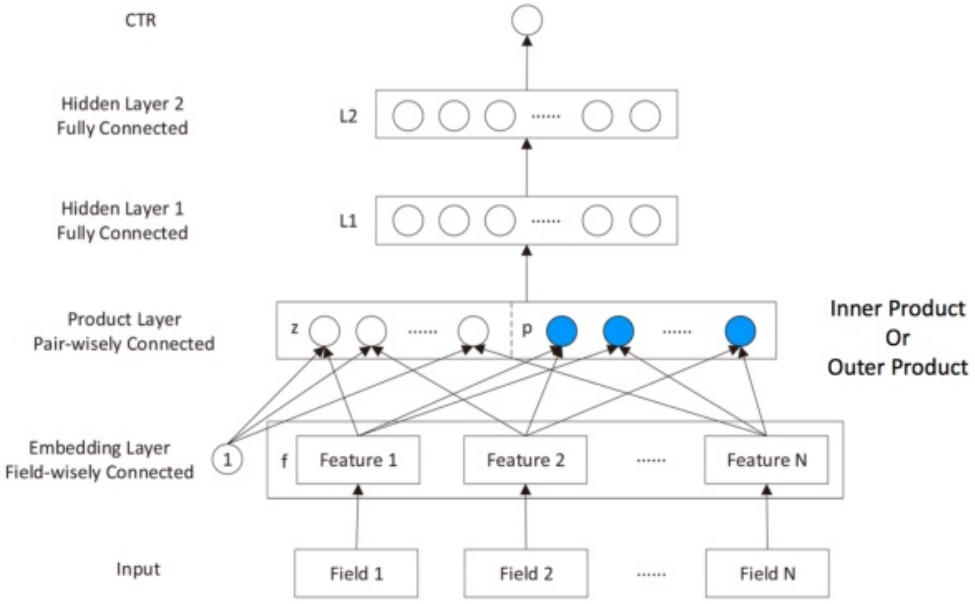
- FNN 的神经网路结构



  - 其实就是一个简单的embedding + FCs，只是对embedding 做了特殊的处理，利用了预训练好的FM一次项和二次项作为初始化
  - FNN only includes Deep part， no shallow part (lr/fm), and feature interactions are concatenation, and use several fcs to go deep layers.
  - x 是输入特征 (大规模离散稀疏)，分为N 个field，每个field 中， 只有一个值为1， 其他都是0 （one hot）。 $Field_i$ 可以表示称 $x[start_i : end_i]$， W_0^i 为Field i 的 embedding matrix. z_i 为 embedding 后的matrix. 它是由 一次项 w_i 二次项 $v_i = (v_i^1, v_i^2 ..., v_i^K)$， 其中K 是FM 二次项的向量dimension，而l1, l2 为FCs

---

- PNN 的介绍
  - Product-based Neural Networks，是一种基于乘法运算来提现特征交叉的dnn模型。其网络结构如下图所示：



- 从整体结构来看，PNN与FNN的区别在于多了一层**Product Layer**。PNN的结构为embeddding+product layer + fcs。而PNN使用product的方式做特征交叉的想法是认为在ctr场景中，特征的交叉更加提现在一种“且”的关系下，而add的操作，是一种“或”的关系，所以product的形式更加合适，会有更好的效果

- Product Layer 的设计

  $$A \odot B = \sum A_{i,j} B_{i,j}$$

  - 定义矩阵计算方法：

    product layer 可以分为两个部分，一部分是线性部分 l_z，一部分是非线性部分 l_p l_z, l_p 都是同样的维度。具体形式如下：

    $l_z = (l_z^1, l_z^2 \dots l_z^n \dots l_z^{D1})$ $l_z^n = W_z^n \odot z$

    $l_p = (l_p^1, l_p^2 \dots l_p^n \dots l_p^{D1})$ $l_p^i = W_p^n \odot p$

    其中 z ， p 为信号向量， z 为线性信号向量， p 为二次信号向量， W_z^i ， W_p^i 为权重矩阵。

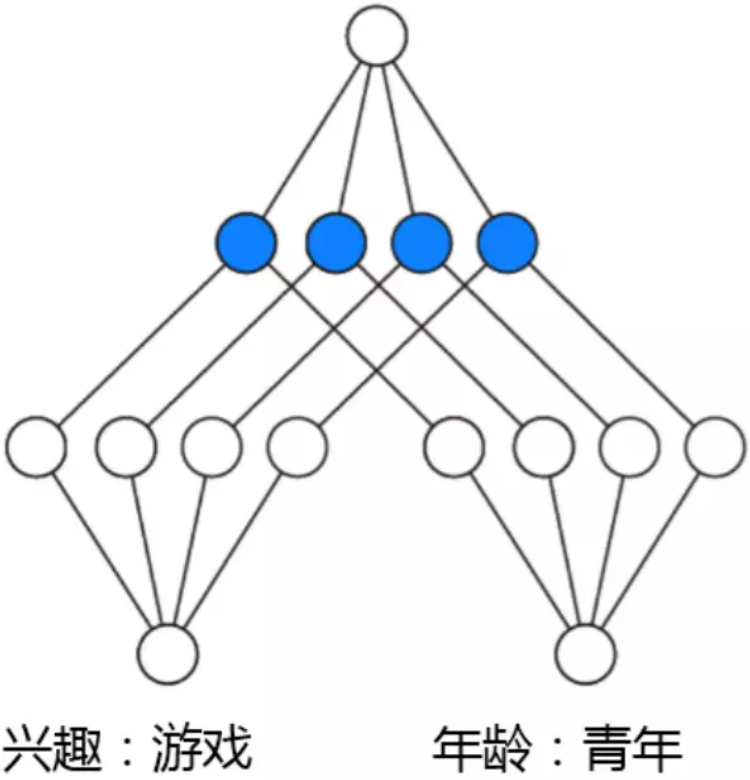
    其具体形式为： $z = (z_1, z_2 \dots z_N) = (f_1, f_2 \dots f_N)$

    $p = \{p_{i,j}\}, i = 1 \dots N, j = 1 \dots N$

    $p_{i,j} = g(f_i, f_j)$

    f_i 代表第i个特征(field)的编码后的向量。函数 $g(f_i, f_j)$ 为 f_i, f_j 的交叉函数，表示特征i，特征j的交叉。在该论文中，该函数的形式有两种：

1. inner product_based



$g(f_i, f_j) =< f_i, f_j >$ 内积表示特征交差

所以，pij其实是一个数，得到一个pij的时间复杂度为M，p的大小为N*N，因此计算得到p的时间复杂度为*NNM。而再由p得到lp的时间复杂度是NND1。因此 对于IPNN来说，总的时间复杂度为NN(D1+M)。文章对这一结构进行了优化，可以看到，我们的p是一个对称矩阵，因此我们的权重也可以是一个对称矩阵，对称矩阵就可以进行如下的分解：

$$W_p^n = \theta^n \theta^n T$$

So,

$$l_p^n = W_p^n dotp = \sum_{i=1}^{N} \sum_{j=1}^{N} \theta_i^n \theta_j^n < f_i, f_j > =< \sum_{i=1}^{N} \alpha_i^n, \sum_{i=1}^{N} \alpha_i^n >$$

$$\alpha_i^n = \theta_i^n f_i$$

Thus,

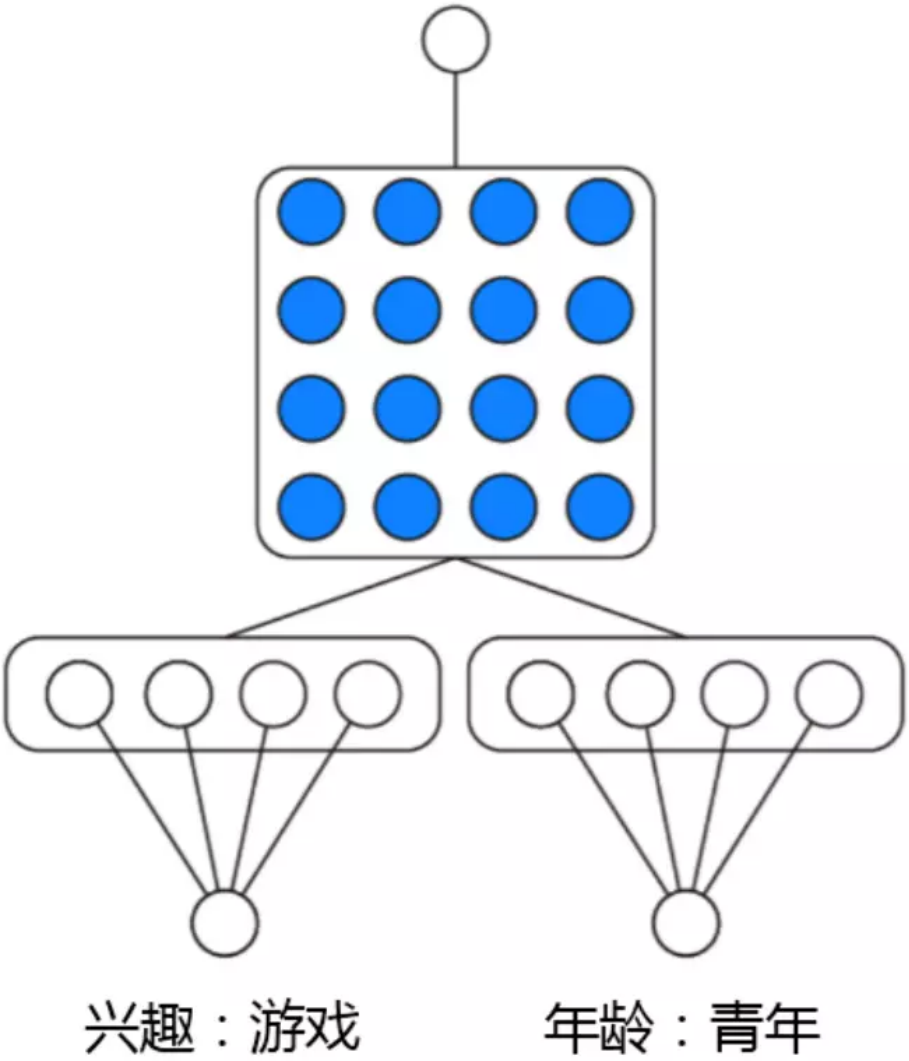
$$\alpha^n = (\alpha_1^n, \alpha_2^n ....., \alpha_i^n, ..., \alpha_N^n) \in R^{NxM}$$

Finally we get,

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

我们的权重只需要D1 * N就可以了，时间复杂度也变为了D1*M*N。

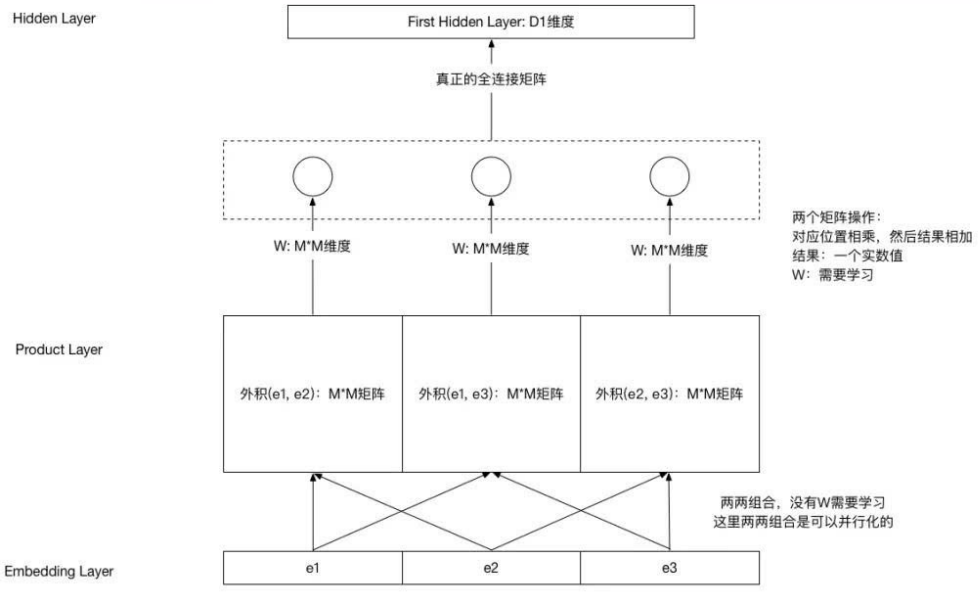
2. outer product_based



$g(f_i, f_j) = f_i f_j^T$ 矩阵乘法表示特征交叉

此时pij为M*M的矩阵，计算一个pij的时间复杂度为*M*M，而p是N*N*M*M的矩阵，因此计算p的事件复杂度为*N*N*M*M。从而计算lp的时间复杂度变为D1 * N*N*M*M。这个显然代价很高的。为了减少负责度，论文使用了叠加的思想，它重新定义了p矩阵

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



外积 p 部分架构逻辑说明

- 总结：
  1. Product Layer中z中每个圈都是一个向量，向量大小为Embedding Vector大小，向量个数 = Field个数 = Embedding向量的个数。Product Layer中如果是内积，p中每个圈都是一个值；如果是外积，p中每个圆圈都是一个二维矩阵。
  2. 在数据流中，假设Field个数为N，那么经过Embedding后的Field得到了N个Feature，或者说是N个嵌入向量。这N个嵌入向量直接拿过来并排放到z中就是z。这部分代表的是对低维度特征，或者说原始特征的建模。作者的说法是，不加入这一部分训练会非常不稳定，加入之后稳定了很多。然后，针对这N个嵌入向量，两两组合进行Product operation，把结果放到一起就得到了p。首先，N个向量两两组合，会产生对组合，这就是p中圆圈的个数，或者说神经元的个数。如果是内积运算，那么每个神经元就是一个实数值；如果进行外积运算，那么每个神经元就是一个二维矩阵
- product layer 的output: $l_1 = relu(l_z + l_p + b_1) $, 其中 b_1 dimension is D1

- Product Layer 的优化

  (1) inner product-based的优化

  考虑公式 $l_p^n = W_p^n \odot p$ ，此时p的维度为N*N，假设 f_i 的维度为M，则计算p的时间复杂度为N*N*M，而 l_p 的计算代价为N*N*D1，所以总的代价为N*N(D1+M)。而矩阵 W_p^n 是对称矩阵，受到FM的启发，可以将矩阵 W_p^n 进行分解： $W_p^n = \theta^n \theta^{nT}$ ，如果令 θ^i 的维度也为M的话，则：

  $$l_p^n = W_p^n \odot p = \sum_{i=1}^{N}\sum_{j=1}^{N} \theta_i^n \theta_j^n < f_i, f_j > = < \sum_{i=1}^{N} \delta_i^n, \sum_{i=1}^{N} \delta_i^n > ,$$

  其中， $\delta_i^n = \theta_i^n f_i$ 。

  令 $\delta^n = (\delta_1^n, \delta_2^n, \ldots, \delta_N^n)$,则其维度为N*M。而且 $l_p^n = ||\sum_i \delta_i^n||$ ，此时计算 l_p 的总的复杂度变成了D1*N*M。

  (2) outer product-based的优化

  考虑 $p_{i,j} = g(f_i, f_j) = f_i f_j^T$ ，可知此时 $p_{i,j}$ 为M*M的矩阵，计算 l_p 的时间复杂度为D1*M*M*N*N,这个显然代价很高的。为了减少负责度，论文使用了叠加的思想，它重新定义了p矩阵：

  $$p = \sum_{i=1}^{N}\sum_{j=1}^{N} f_i f_j^T = f_\sum (f_\sum)^T, f_\sum = \sum_{i=1}^{N} f_i$$

  则此时可以将复杂度降低为D1*M*(M+N)。