从FM推演各深度学习CTR预估模型（上）｜推荐系统

七月在线实验室 1周前

文 | 七月在线
编 | 小七

解析：

点击率(click-through rate, CTR)是互联网公司进行流量分配的核心依据之一。比如互联网广告平台，为了精细化权衡和保障用户、广告、平台三方的利益，准确的CTR预估是不可或缺的。CTR预估技术从传统的逻辑回归，到近两年大火的深度学习，新的算法层出不穷：DeepFM, NFM, DIN, AFM, DCN……

然而，相关的综述文章不少，但碎片罗列的居多，模型之间内在的联系和演化思路如何揭示？怎样才能迅速get到新模型的创新点和适用场景，快速提高新论文速度，节约理解、复现模型的成本？这些都是亟待解决的问题。

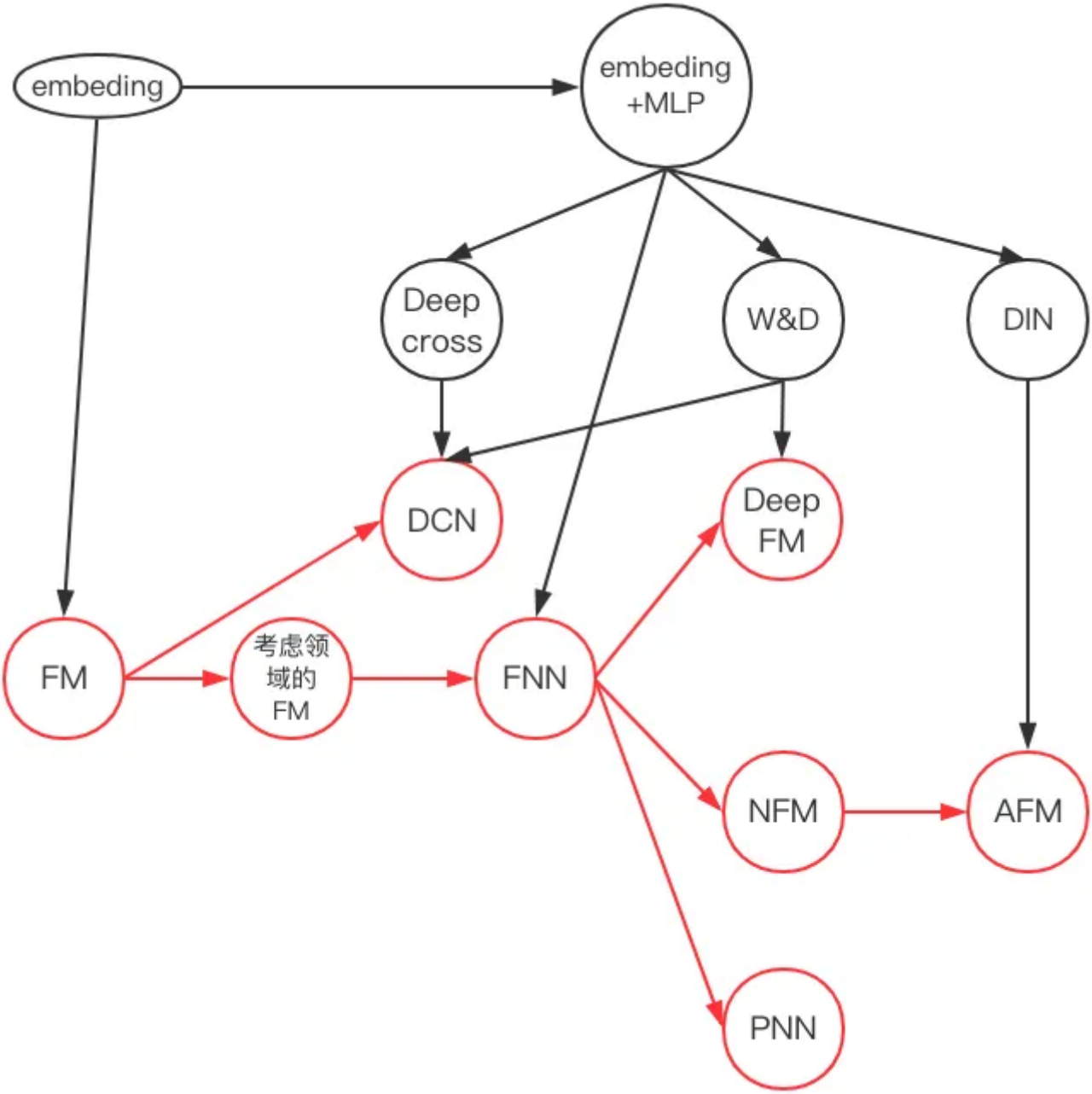
我们认为，从FM及其与神经网络的结合出发，能够迅速贯穿很多深度学习CTR预估网络的思路，从而更好地理解和应用模型。

本文的思路与方法

我们试图从原理上进行推导、理解各个深度CTR预估模型之间的相互关系，知其然也知其所以然。（以下的分析与拆解角度，是一种我们尝试的理解视角，并不是唯一的理解方式）

推演的核心思路：“通过设计网络结构进行组合特征的挖掘”。

具体来说有两条：**其一是从FM开始推演其在深度学习上的各种推广（对应下图的红线），另一条是从embedding+MLP自身的演进特点结合CTR预估本身的业务场景进行推演（对应下图黑线部分）。**



为了便于理解，我们简化了数据案例——只考虑离散特征数据的建模，以分析不同神经网络在处理相同业务问题时的不同思路。

同时，我们将各典型论文不同风格的神经网络结构图统一按照计算图来绘制，以便于对比不同模型。

二、FM：降维版本的特征二阶组合

CTR预估本质是一个二分类问题，以移动端展示广告推荐为例，依据日志中的用户侧的信息（比如年龄，性别，国籍，手机上安装的app列表）、广告侧的信息（广告id，广告类别，广告标题等）、上下文侧信息（渠道id等），去建模预测用户是否会点击该广告。

FM出现之前的传统的处理方法是人工特征工程加上线性模型（如逻辑回归Logistic Regression）。为了提高模型效果，关键技术是找到到用户点击行为背后隐含的特征组合。如男性、大学生用户往往会点击游戏类广告，因此"男性且是大学生且是游戏类"的特征组合就是一个关键特征。但这本质仍是线性模型，其假设函数表示成内积形式一般为：

$$y_{linear} = \sigma\left(\langle \vec{w}, \vec{x} \rangle\right)$$

其中$\vec{x}$为特征向量，$\vec{w}$为权重向量，σ()为sigmoid函数。

但是人工进行特征组合通常会存在诸多困难，如特征爆炸、特征难以被识别、组合特征难以设计等。

为了让模型自动地考虑特征之间的二阶组合信息，线性模型推广为二阶多项式（2d−Polynomial 2d-Polynomial2d−Polynomial）模型：

$$y_{poly} = \sigma\left(\langle \vec{w}, \vec{x} \rangle + \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} \cdot x_i \cdot x_j\right)$$

其实就是对特征两两相乘（组合）构成新特征(离散化之后其实就是"且"操作)，并对每个新特征分配独立的权重，通过机器学习来自动得到这些权重。将其写成矩阵形式为：

$$y_{poly} = \sigma\left(\vec{w}^T \cdot \vec{x} + \vec{x}^T \cdot W^{(2)} \cdot \vec{x}\right)$$

其中$W^{(2)}$为二阶特征组合的权重矩阵，是对称矩阵。而这个矩阵参数非常多，为$O(n^2)$。为了降低该矩阵的维度，可以将其因子分解（Factorization FactorizationFactorization）为两个低维

（比如n∗k n*kn∗k）矩阵的相乘。则此时W WW矩阵的参数就大幅降低，为O(nk)。公式如下：

$$W^{(2)} = W^T \cdot W$$

这就是Rendle RendleRendle等在2010年提出因子分解机（Factorization Machines，FM）的名字的由来。FM的矩阵形式公式如下：

$$y_{FM} = \sigma\left(\vec{w}^T \cdot \vec{x} + \vec{x}^T \cdot W^T \cdot W \cdot \vec{x}\right)$$

将其写成内积的形式：

$$y_{FM} = \sigma(\langle \vec{w}, \vec{x} \rangle + \langle W \cdot \vec{x}, W \cdot \vec{x} \rangle)$$

利用

$$\langle \sum_{i=1}^{n} \vec{a_i}, \sum_{i=1}^{n} \vec{a_i} \rangle = \sum_{i=1}^{n} \sum_{j=1}^{n} \langle \vec{a_i}, \vec{a_j} \rangle$$

，可以将上式进一步改写成求和式的形式：

$$y_{FM} = \sigma\left(\langle \vec{w}, \vec{x} \rangle + \sum_{i=1}^{n} \sum_{j=1}^{n} \langle x_i \cdot \vec{v}_i, x_j \cdot \vec{v}_j \rangle\right)$$

其中 $\overrightarrow{vi}$ 向量是矩阵W WW的第i ii列。为了去除重复项与特征平方项，上式可以进一步改写成更为常见的FM公式：

$$y_{FM} = \sigma\left(\langle \vec{w}, \vec{x} \rangle + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \vec{v}_i, \vec{v}_j \rangle x_i \cdot x_j\right)$$

对比二阶多项式模型，FM模型中特征两两相乘（组合）的权重是相互不独立的，它是一种参数较少但表达力强的模型。

在给出FM的TensorFlow代码实现之前，值得一提的是FM通过内积进行无重复项与特征平方项的特征组合过程使用了一个小trick，就是：

$$\sum_{i=1}^{n}\sum_{j=i+1}^{n} x_i x_j = 1/2 \times [(\sum_{i=1}^{n} x_i)^2 - \sum_{i=1}^{n} x_i^2]$$

```
class FM(Model):
    def __init__(self, input_dim=None, output_dim=1, factor_order=10, init_path=None, opt_algo='gd', learning_rate=1e-2,
                 l2_w=0, l2_v=0, random_seed=None):
        Model.__init__(self)
        # 一次、二次交叉、偏置项
        init_vars = [('w', [input_dim, output_dim], 'xavier', dtype),
                     ('v', [input_dim, factor_order], 'xavier', dtype),
                     ('b', [output_dim], 'zero', dtype)]
        self.graph = tf.Graph()
        with self.graph.as_default():
            if random_seed is not None:
                tf.set_random_seed(random_seed)
            self.X = tf.sparse_placeholder(dtype)
            self.y = tf.placeholder(dtype)
            self.vars = init_var_map(init_vars, init_path)
```

```
w = self.vars['w']
v = self.vars['v']
b = self.vars['b']

# [(x1+x2+x3)^2 - (x1^2+x2^2+x3^2)]/2
# 先计算所有的交叉项，再减去平方项(自己和自己相乘)
X_square = tf.SparseTensor(self.X.indices, tf.square(self.X.values), tf.to_int64(tf.shape(self.X)))
xv = tf.square(tf.sparse_tensor_dense_matmul(self.X, v))
p = 0.5 * tf.reshape(
    tf.reduce_sum(xv - tf.sparse_tensor_dense_matmul(X_square, tf.square(v)), 1),
    [-1, output_dim])
xw = tf.sparse_tensor_dense_matmul(self.X, w)
logits = tf.reshape(xw + b + p, [-1])
self.y_prob = tf.sigmoid(logits)

self.loss = tf.reduce_mean(
    tf.nn.sigmoid_cross_entropy_with_logits(logits=logits, labels=self.y)) + \
        l2_w * tf.nn.l2_loss(xw) + \
        l2_v * tf.nn.l2_loss(xv)
self.optimizer = get_optimizer(opt_algo, learning_rate, self.loss)

#GPU设定
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
self.sess = tf.Session(config=config)
# 图中所有variable初始化
tf.global_variables_initializer().run(session=self.sess)
```

三、用神经网络的视角看FM：嵌入后再进行内积

我们观察FM公式的矩阵内积形式：

$$y_{FM} = \sigma(\langle \vec{w}, \vec{x} \rangle + \langle W \cdot \vec{x}, W \cdot \vec{x} \rangle)$$

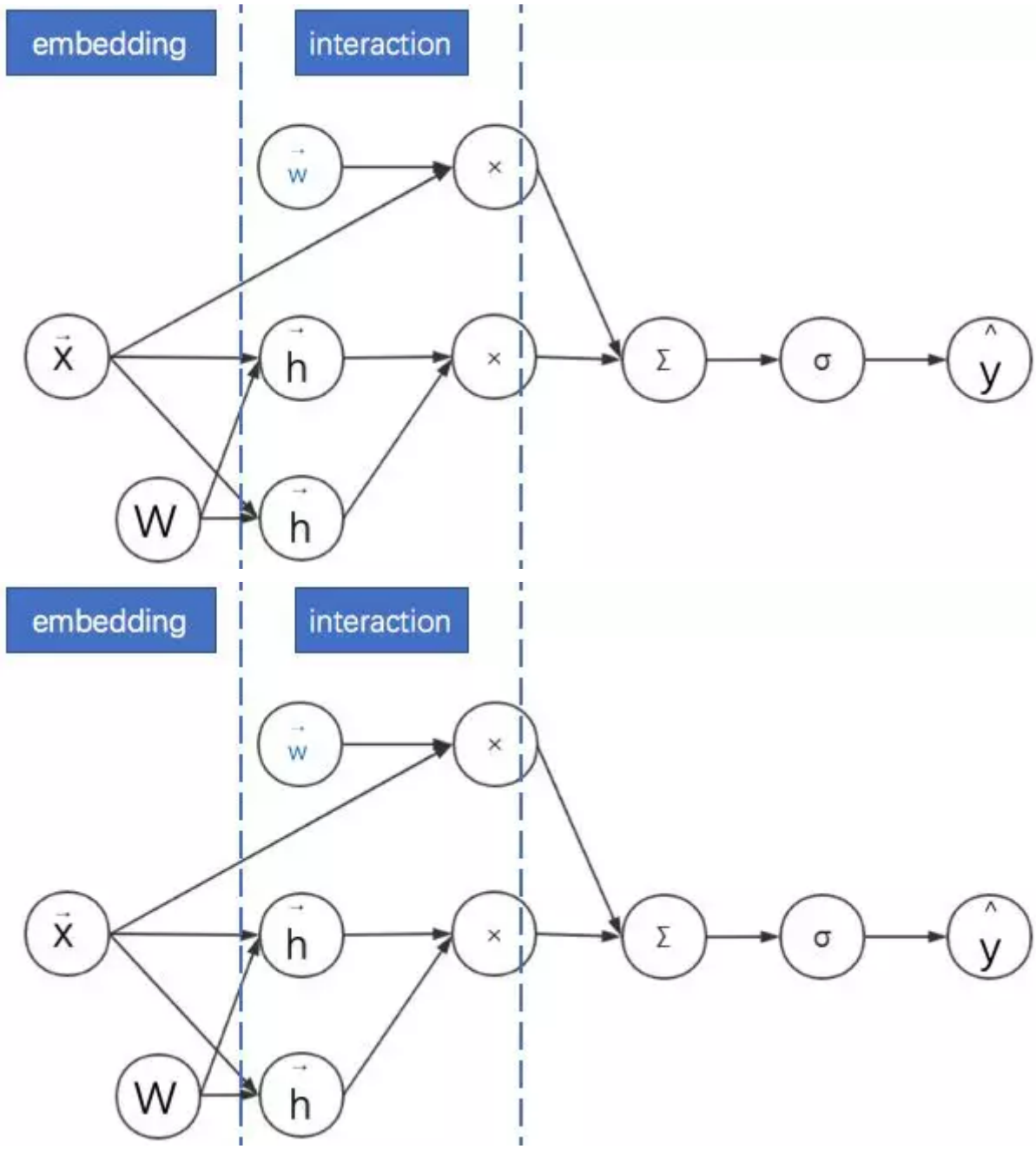
发现 $W \cdot \vec{x}$ 部分就是将离散系数特征通过矩阵乘法降维成一个低维稠密向量。这个过程对神经网络来说就叫做嵌入（embedding）。所以用神经网络视角来看：

a) FM首先是对离散特征进行嵌入。

b) 之后通过对嵌入后的稠密向量进行内积来进行二阶特征组合。

c) 最后再与线性模型的结果求和进而得到预估点击率。

其示意图如下。为了表述清晰，我们绘制的是神经网络计算图而不是网络结构图——在网络结构图中增加了权重W位置。

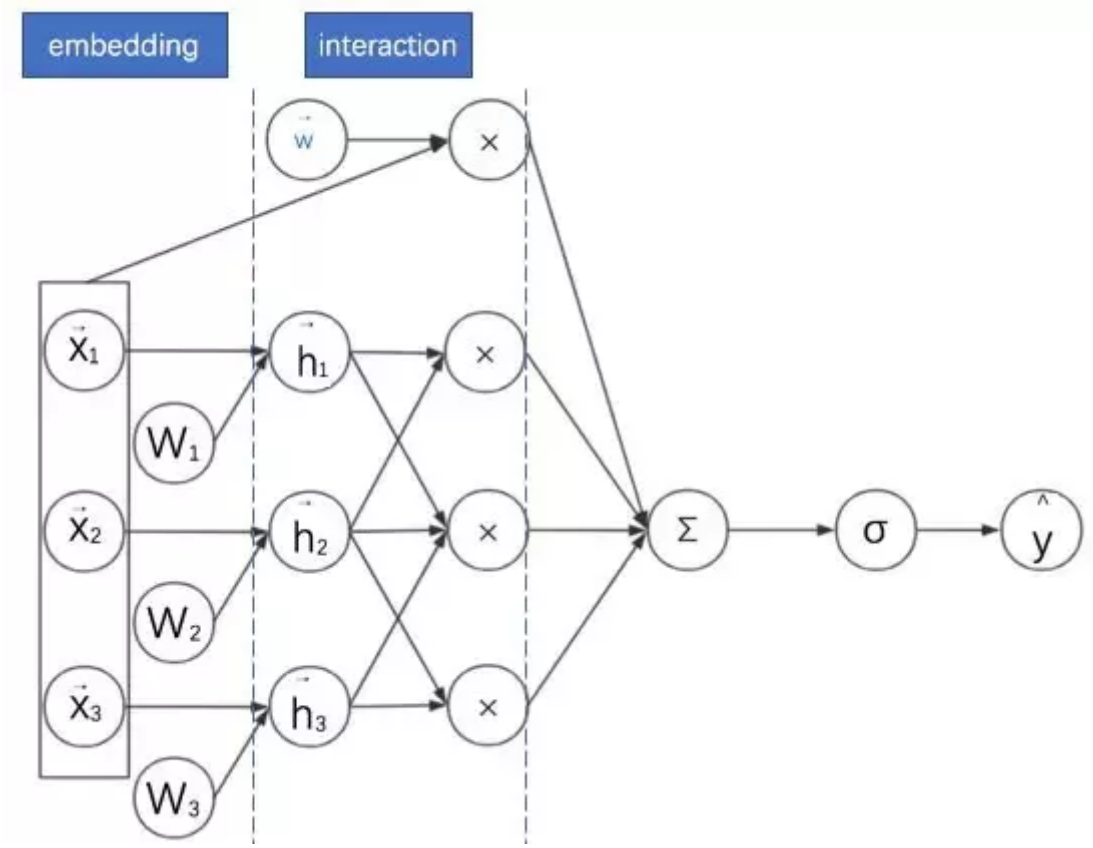


四、FM的实际应用：考虑领域信息

广告点击率预估模型中的特征以分领域的离散特征为主，如：广告类别、用户职业、手机APP列表等。由于连续特征比较好处理，为了简化起见，本文只考虑同时存在不同领域的离散特征的情形。处理离散特征的常见方法是通过独热（one-hot）编码转换为一系列二值特征向量。然后将这些高维稀疏特征通过嵌入（embedding）转换为低维连续特征。前面已经说明FM中间的一个核心步骤就是嵌入，但这个嵌入过

程没有考虑领域信息。这使得同领域内的特征也被当做不同领域特征进行两两组合了。

其实可以将特征具有领域关系的特点作为先验知识加入到神经网络的设计中去：同领域的特征嵌入后直接求和作为一个整体嵌入向量，进而与其他领域的整体嵌入向量进行两两组合。而这个先嵌入后求和的过程，就是一个单领域的小离散特征向量乘以矩阵的过程。此时FM的过程变为：对不同领域的离散特征分别进行嵌入，之后再进行二阶特征的向量内积。其计算图图如下所示：



这样考虑其实是给FM增加了一个正则：考虑了领域内的信息的相似性。而且还有一个附加的好处，这些嵌入后的同领域特征可以拼接起来作为更深的神经网络的输入，达到降维的目的。接下来我们将反复看到这种处理方式。

此处需要注意，这与“基于领域的因子分解机”（Field-aware Factorization Machines，FFM）有区别。FFM也是FM的另一种变体，也考虑了领域信息。但其不同点是同一个特征与不同领域进行特征组合时，其对应的嵌入向量是不同的。本文不考虑FFM的作用机制。

经过这些改进的FM终究还是浅层网络，它的表现力仍然有限。为了增加模型的表现力(model capacity)，一个自然的想法就是将该浅层网络不断“深化”。

五、embedding+MLP：深度学习CTR预估的通用框架

embedding+MLP是对于分领域离散特征进行深度学习CTR预估的通用框架。深度学习在特征组合挖掘

(特征学习)方面具有很大的优势。比如以CNN为代表的深度网络主要用于图像、语音等稠密特征上的学习，以W2V、RNN为代表的深度网络主要用于文本的同质化、序列化高维稀疏特征的学习。CTR预估的主要场景是对离散且有具体领域的特征进行学习，所以其深度网络结构也不同于CNN与RNN。

具体来说， embedding+MLP的过程如下：

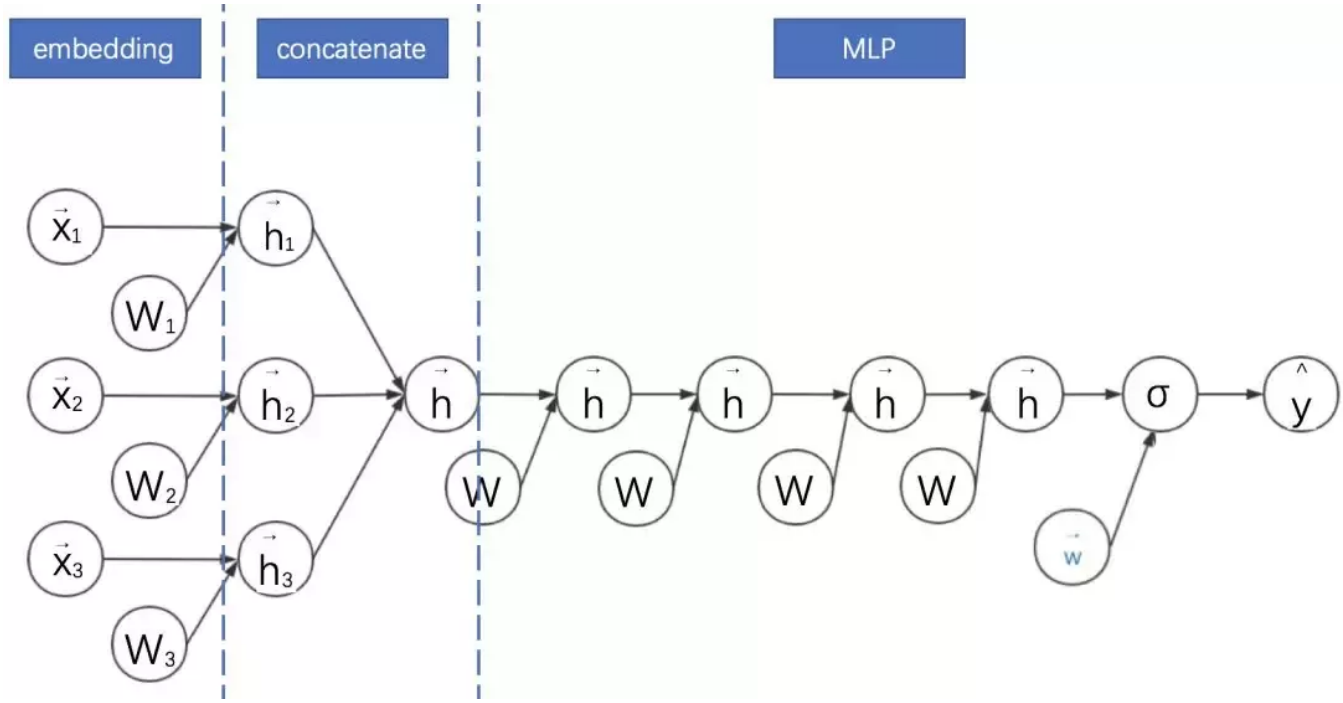
①对不同领域的one-hot特征进行嵌入（embedding），使其降维成低维度稠密特征。

②然后将这些特征向量拼接（concatenate）成一个隐含层。

③之后再不断堆叠全连接层，也就是多层感知机(Multilayer Perceptron, MLP，有时也叫作前馈神经网络)。

④最终输出预测的点击率。

其示意图如下：



embedding+MLP的缺点是只学习高阶特征组合，对于低阶或者手动的特征组合不够兼容，而且参数较多，学习较困难。

六、FNN:FM与MLP的串联结合

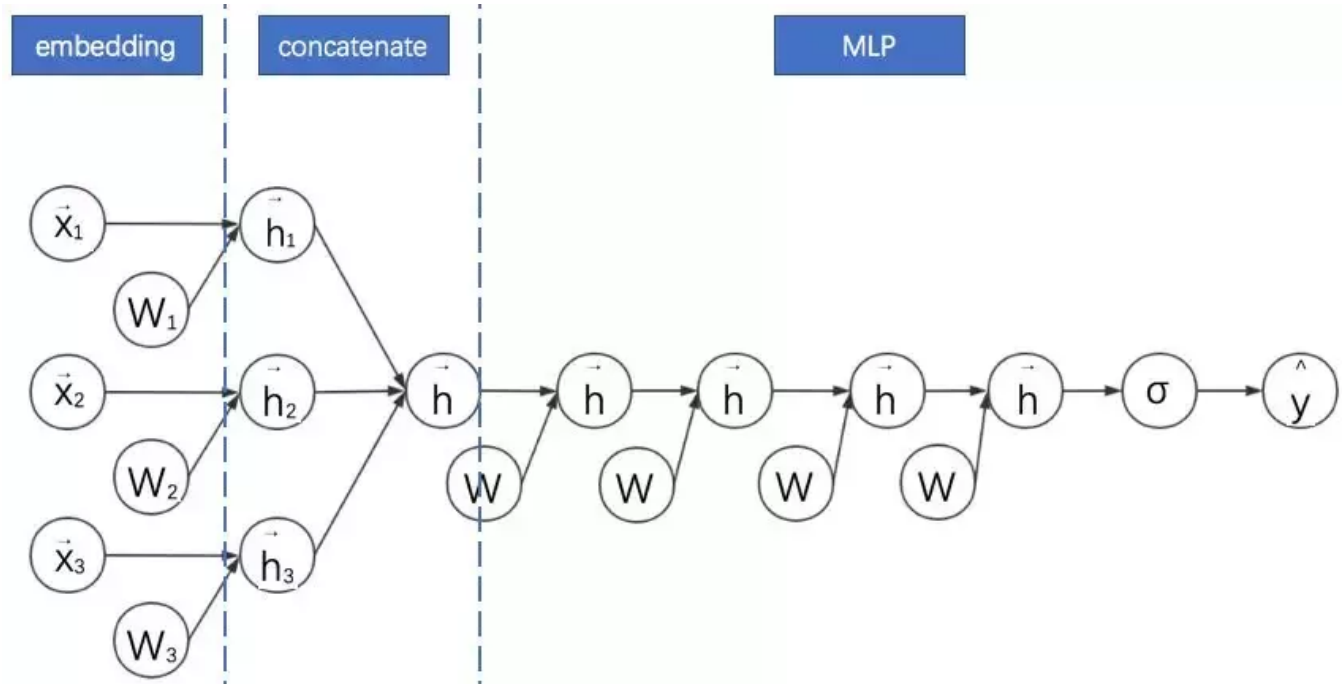
Weinan Zhang等在2016年提出的因子分解机神经网络(Factorisation Machine supported Neural Network，FNN)将考FM与MLP进行了结合。它有着十分显著的特点：

采用FM预训练得到的隐含层及其权重作为神经网络的第一层的初始值，之后再不断堆叠全连接层，最终输出预测的点击率。

可以将FNN理解成一种特殊的embedding+MLP，其要求第一层嵌入后的各领域特征维度一致，并且嵌入权重的初始化是FM预训练好的。

这不是一个端到端的训练过程，有贪心训练的思路。而且如果不考虑预训练过程，模型网络结构也没有考虑低阶特征组合。

其计算图如下所示：



通过观察FFN的计算图可以看出其与embedding+MLP确实非常像。**不过此处省略了FNN的FM部分的线性模块。**这种省略为了更好地进行两个模型的对比。接下来的计算图我们都会省略线性模块。

此处附上FNN的代码实现：

```
class FNN(Model):
    def __init__(self, field_sizes=None, embed_size=10, layer_sizes=None, layer_acts=None, drop_out=None,
                 embed_l2=None, layer_l2=None, init_path=None, opt_algo='gd', learning_rate=1e-2, random_seed=None):
        Model.__init__(self)
        init_vars = []
        num_inputs = len(field_sizes)
        for i in range(num_inputs):
            init_vars.append(('embed_%d' % i, [field_sizes[i], embed_size], 'xavier', dtype))
        node_in = num_inputs * embed_size
        for i in range(len(layer_sizes)):
            init_vars.append(('w%d' % i, [node_in, layer_sizes[i]], 'xavier', dtype))
            init_vars.append(('b%d' % i, [layer_sizes[i]], 'zero', dtype))
            node_in = layer_sizes[i]
        self.graph = tf.Graph()
        with self.graph.as_default():
            if random_seed is not None:
                tf.set_random_seed(random_seed)
            self.X = [tf.sparse_placeholder(dtype) for i in range(num_inputs)]
            self.y = tf.placeholder(dtype)
            self.keep_prob_train = 1 - np.array(drop_out)
            self.keep_prob_test = np.ones_like(drop_out)
            self.layer_keeps = tf.placeholder(dtype)
            self.vars = init_var_map(init_vars, init_path)
            w0 = [self.vars['embed_%d' % i] for i in range(num_inputs)]
            xw = tf.concat([tf.sparse_tensor_dense_matmul(self.X[i], w0[i]) for i in range(num_inputs)], 1)
            l = xw
```

```
        #全连接部分
for i in range(len(layer_sizes)):
    wi = self.vars['w%d' % i]
    bi = self.vars['b%d' % i]
    print(l.shape, wi.shape, bi.shape)
    l = tf.nn.dropout(
        activate(
            tf.matmul(l, wi) + bi,
            layer_acts[i]),
        self.layer_keeps[i])

l = tf.squeeze(l)
self.y_prob = tf.sigmoid(l)

self.loss = tf.reduce_mean(
    tf.nn.sigmoid_cross_entropy_with_logits(logits=l, labels=self.y))
if layer_l2 is not None:
    self.loss += embed_l2 * tf.nn.l2_loss(xw)
    for i in range(len(layer_sizes)):
        wi = self.vars['w%d' % i]
        self.loss += layer_l2[i] * tf.nn.l2_loss(wi)
self.optimizer = get_optimizer(opt_algo, learning_rate, self.loss)

config = tf.ConfigProto()
config.gpu_options.allow_growth = True
self.sess = tf.Session(config=config)
tf.global_variables_initializer().run(session=self.sess)
```

七、DeepFM: FM与MLP的并联结合

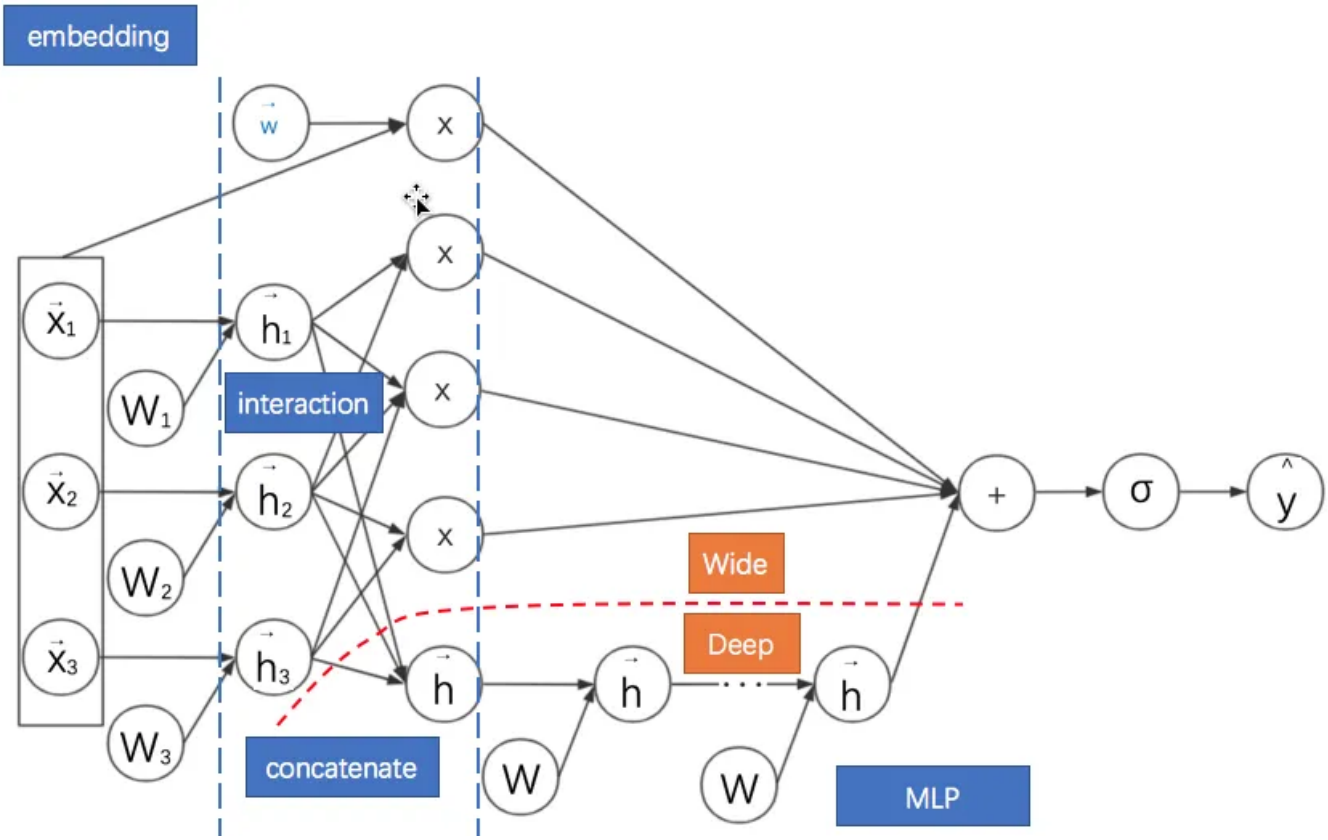
针对FNN需要预训练的问题，Huifeng Guo等提出了深度因子分解机模型（Deep Factorisation Machine, DeepFM, 2017）。**该模型的特点是：**

i) 不需要预训练。

ii)将考虑领域信息的FM部分与MLP部分并联起来（借用初中电路的术语），其实就是对两个模型进行联合训练。

iii)考虑领域信息的FM部分的嵌入向量拼接起来作为MLP部分的输入特征，也就是是两个模型共享嵌入后的特征。

其计算图如下所示：



通过观察DeepFM的计算图可以看出红色虚线以上部分其实就是FM部分，虚线以下就是MLP部分。

本文素材来源于七月在线面试题，关注公号，获取更多面试资料。

本期特训课程