白话推荐系统(二)：一文看懂DCN(DeepCrossNet)

陈壮实的搬砖日记
熟悉推荐系统，爱好AIGC、OCR，软工硕士

DCN（Deep & Cross Network，深度交叉网络✦）是一种用于点击率预估（CTR）的深度学习模型，由Google和斯坦福大学于2017年提出。它结合了显式特征交叉和深度神经网络的优势，能够高效地学习低阶和高阶特征交互，而无需手动特征工程。

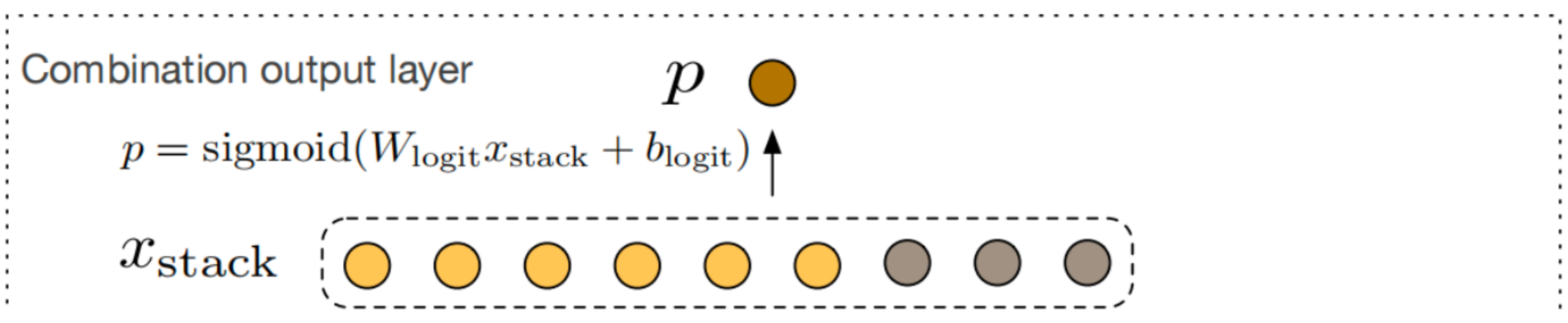
> 论文：Deep & Cross Networkfor Ad Click Predictions, KDD'2017, Google
> 这篇论文并不长，建议有时间读下原文

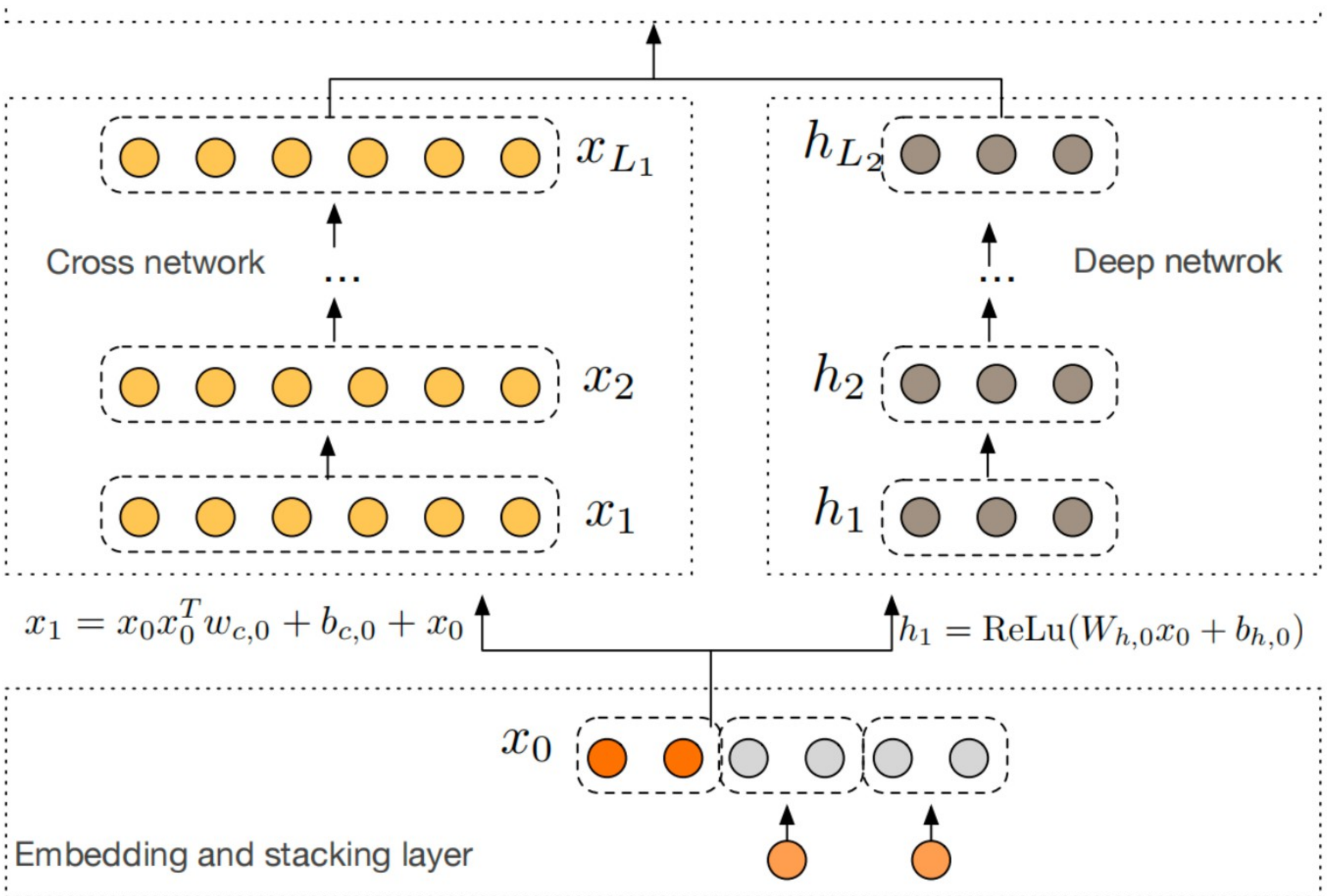
1. 论文提出背景

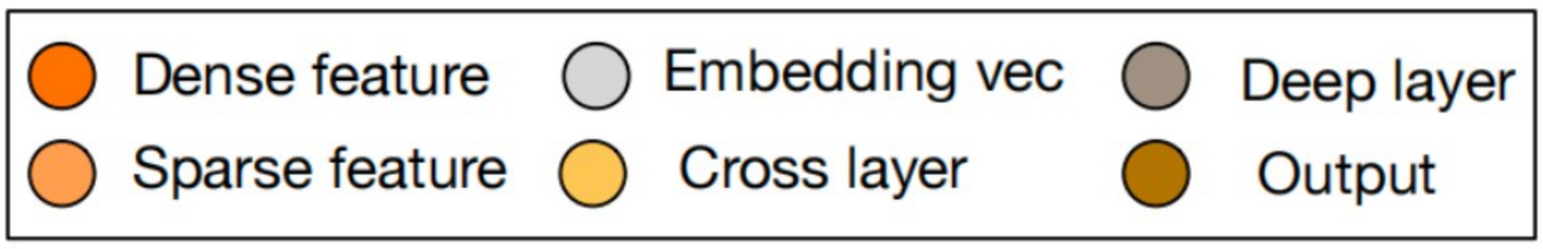
> 这里直接饮用原文，原文分析简单易懂

- **ABSTRACT（摘要）**：手工制作的组合特征在许多成功模型中作用显著，但在网络规模应用场景下，其创建、维护和部署成本过高。论文提出 DeepCross 模型，这是一种深度神经网络，能自动组合特征。它以单独的稠密或稀疏特征作为输入，通过嵌入层✦、堆叠层和一系列残差单元✦构成的网络隐式发现重要的交叉特征。该模型借助计算网络工具包✦（CNTK）实现，并在多 GPU 平台支持下，为大型付费搜索引擎构建网络规模模型，仅用部分特征就取得更好效果，显示出作为通用建模范式的潜力，有助于改进现有产品、加速新模型开发，减少特征工程投入和领域知识获取需求。
- **INTRODUCTION（引言）**：在网络规模的机器学习应用中，手工制作组合特征面临诸多挑战。一方面，特征组合数量会随着特征维度增加呈指数级增长，人工难以穷举和筛选；另一方面，新特征的引入可能带来过拟合风险，且难以判断新特征的有效性，维护成本高。为应对这些问题，研究人员尝试使用决策树、因子分解机等方法自动学习特征交互，但这些方法在处理高维稀疏数据时存在局限性。而深度学习模型在图像、语音等领域成果丰硕，因此作者提出 DeepCross 模型，期望利用深度学习自动学习特征组合的能力，在网络规模建模中摆脱对手工组合特征的依赖，实现高效、低成本的模型构建 。

2. 模型架构解析：

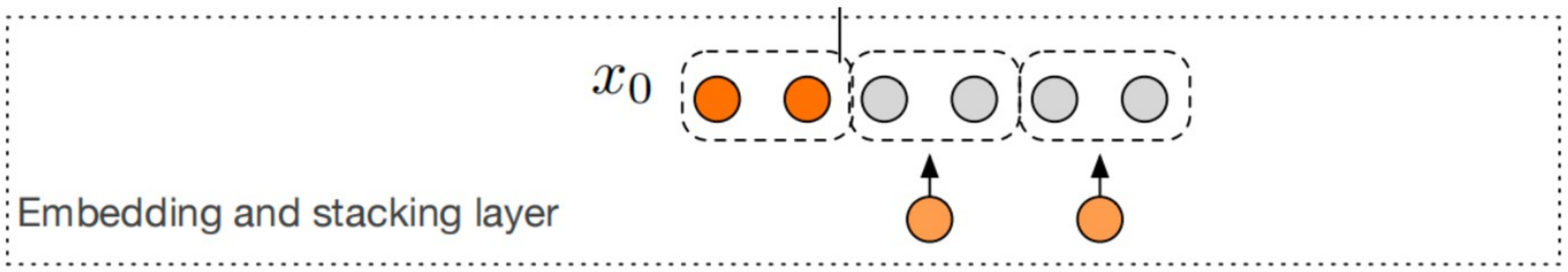






如上图所示，DCN主要是由四部分组成：Embedding Layer, 交叉网络（Cross network）、深层网络（Deep network）和Combination output layer四部分组成，下面将分别介绍这四个部分：

2.1 Embedding Layer



一般来说，都是对离散特征（如：性别、地区，产品类别等）进行Embedding，因为他们不是数字，而模型的输入只能是数字。**Embedding很大的一个作用就是可以将这些非数字特征转化为数字**。有时，一些连续特征也可以使用Embedding，如：年龄，年龄是一个连续特征，但当我们想把年龄分成年龄段时，如：0～6岁为幼年；7～18青少年；18～30为青年等等，此时年龄段其实也是离散特征。

Embedding过程就是对离散特征进行稀疏化处理后，再映射到低维的向量空间。这里不对稀疏处理做详细介绍，以“性别"举例说明即可：

Step1: 分类

将性别女，标记为0；性别男，标记为1。

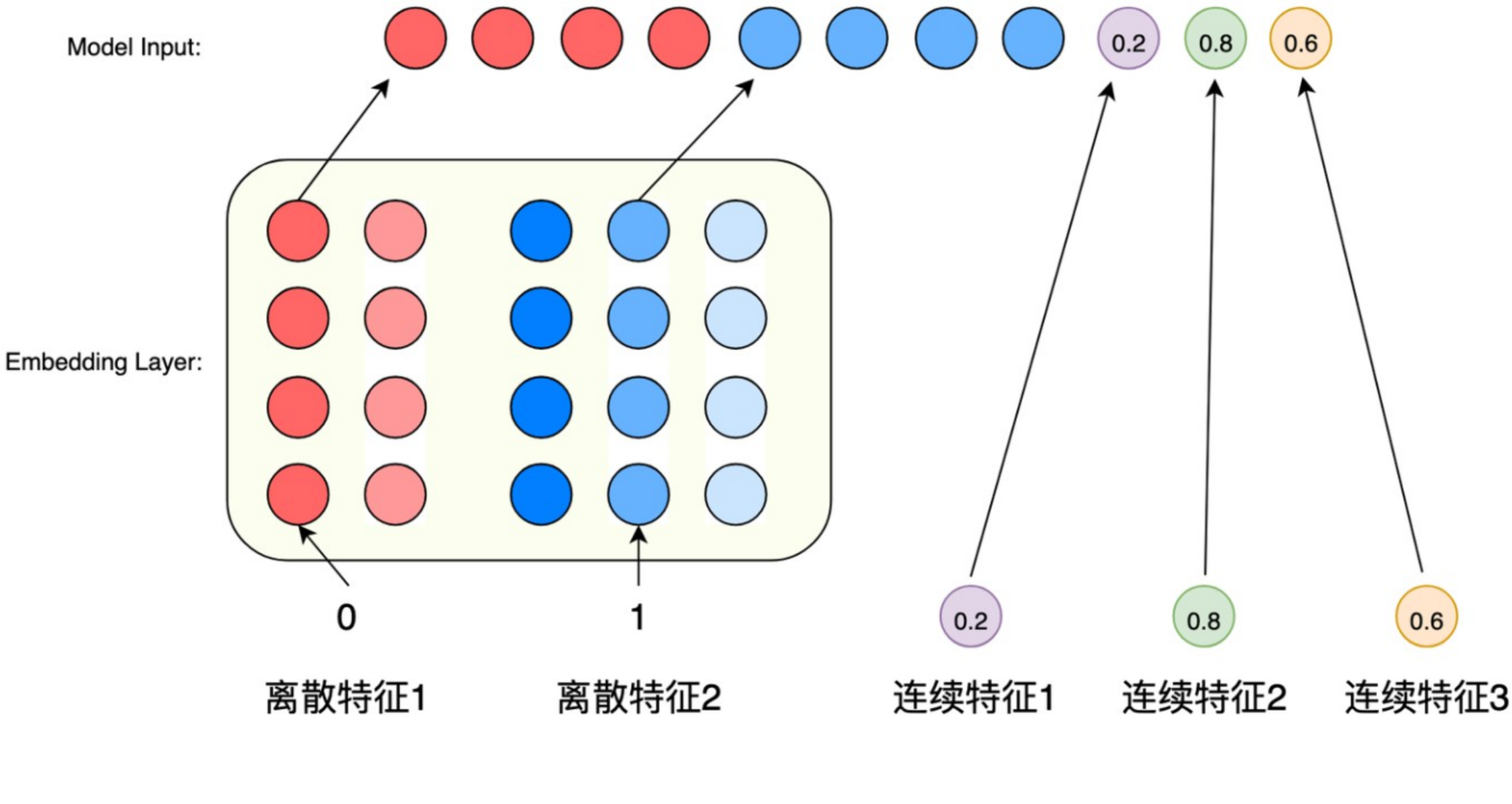
Step2: 生成一个 $len*dim$ 的向量 Q

这里的 dim 可以根据情况进行设置，这里设为4，但 len 只能为2，因为只有男女这2中类别。

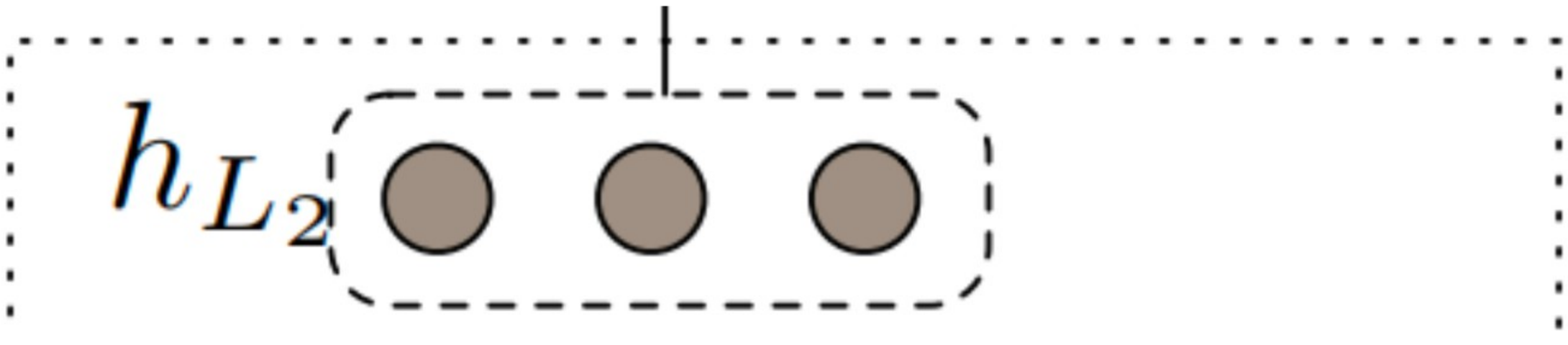
Step3: 向量映射

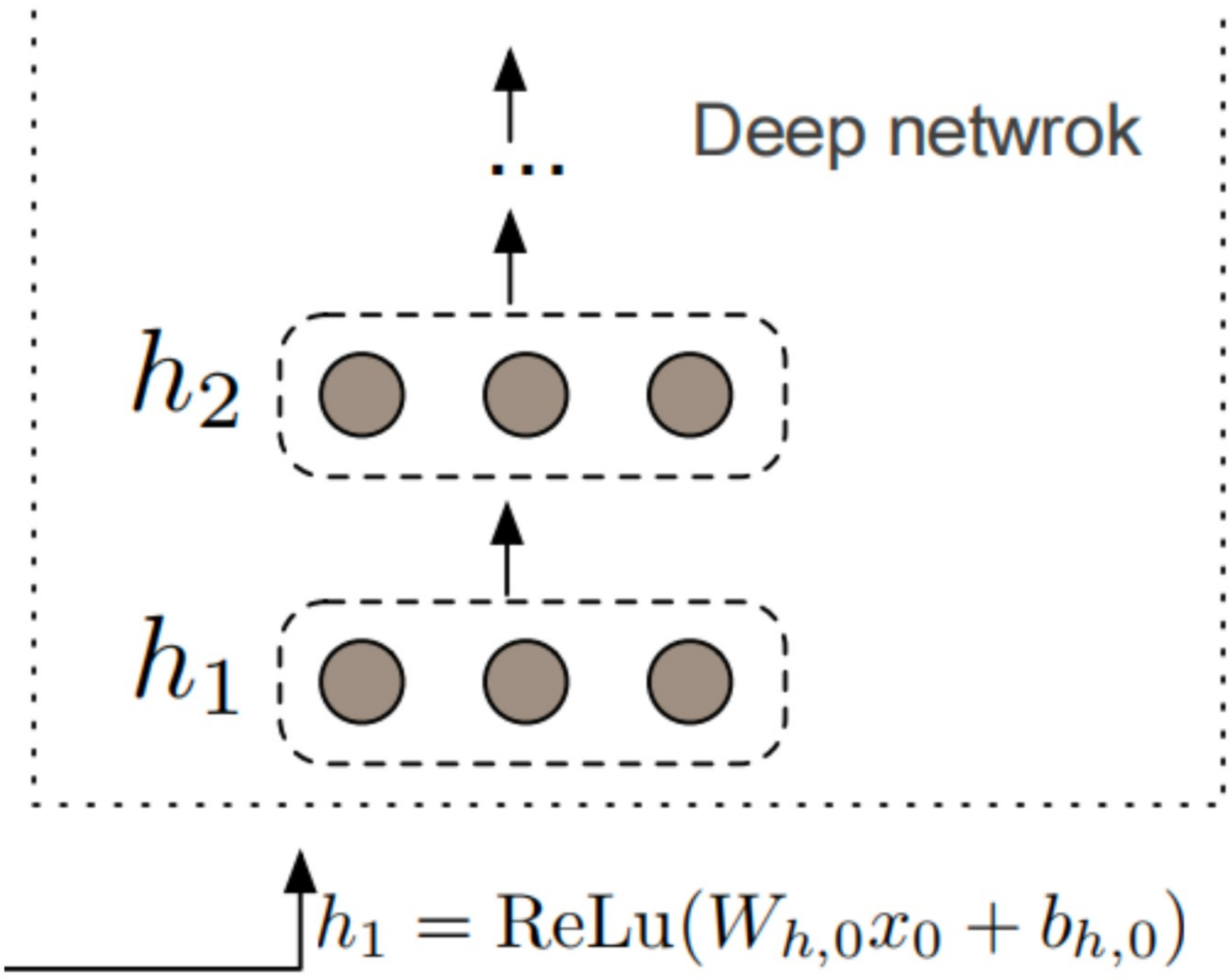
将0映射为 $Q[0]$, 将1映射为$Q[1]$ 。这样就讲男女这种离散特征转化为了4维的向量特征了。

将所有的离散特征转化为向量后，就将他们展开拼接起来，同时，也需要将连续特征拼接起来，但往往连续特征也需要进行一些处理（如：正则化），就可以作为模型的输入了，如下图所示：



2.2 Deep network



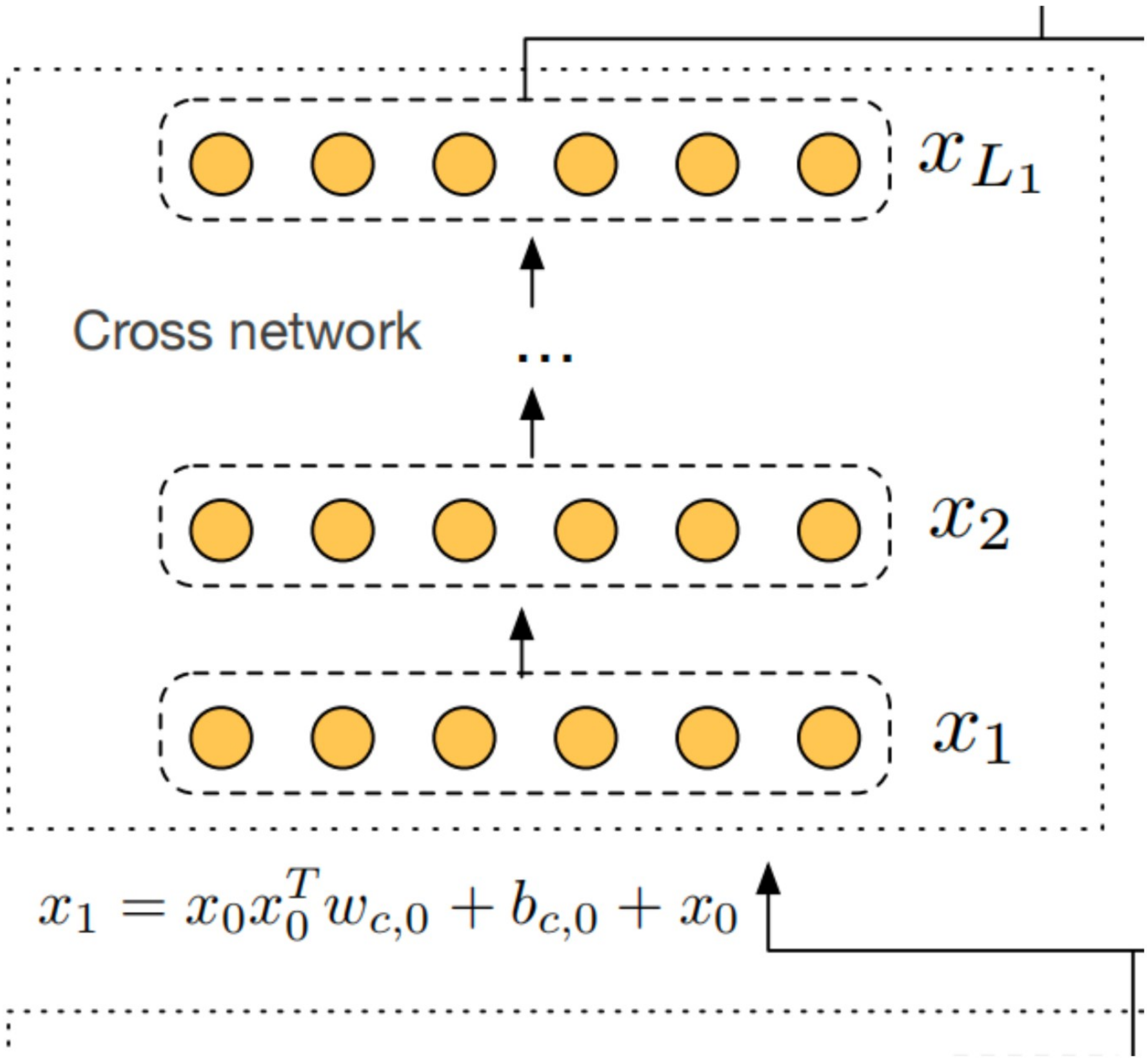


右侧是Deep netWork，就是常见的深度网络✦。代码如下：

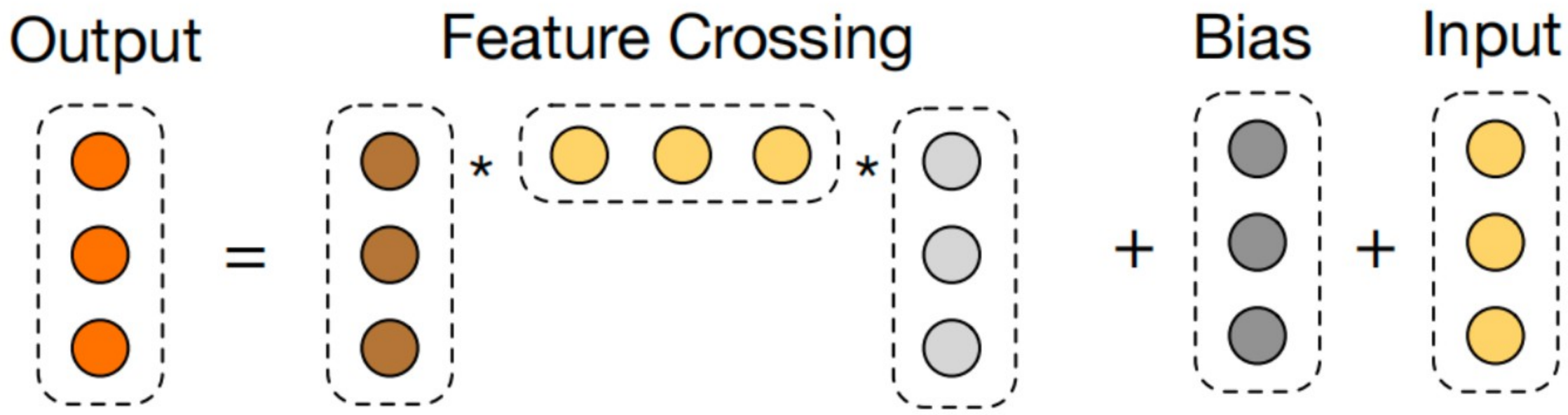
```
class Deep(nn.Module):
    def __init__(self, hidden_layers, dropout_p=0.0):
        """
        Deep: 深层网络
        Args:
            hidden_layers: deep网络的隐层维度, 如[128, 64, 32]
            dropout_p: dropout_p值. Defaults to 0.0.
        """
        super(Deep, self).__init__()
        self.dnn = nn.ModuleList()
        for layer in list(zip(hidden_layers[:-1], hidden_layers[1:])):
            linear = nn.Linear(in_features=layer[0], out_features=layer[1])
            self.dnn.append(linear)
        self.dropout = nn.Dropout(p=dropout_p)
```

```
    def forward(self, X):
        """
        Args:
            X: 输入数据, shape=(batch_size, input_dim)
        Returns:
            res: deep网络的输出, shape=(batch_size, hidden_layers[-1])
        """
        for linear in self.dnn:
            X = linear(X)
        res = self.dropout(X)
        return res
```

2.3 Cross network



左侧部分是交叉网络（Cross Network），也是DCN的精髓所在，它的每一层可如下图所示：



$$y = x_0 * x' * w + b + x$$

图中的公式没有体现出前后层的递进关系，结合下面公式可以更清晰的理解前后层之间的递进关系：

$$X_{i+1} = X_0 X_i^T W_i + X_i + b_i$$

从公式中可以看出， X_{i+1} 是由 X_i 和 X_0 决定的，这就是Cross network的特征交叉能力强的根本所在。

它的特征交叉能力为何而强呢？（重要!!）

详见如下推导：

为方便展示，推导过程中我们舍去公式（1）中 b_i （它是一个常数项，不会影响特征交叉的效果证明），设输入 X_0 有两个特征 x_a, x_b 为:

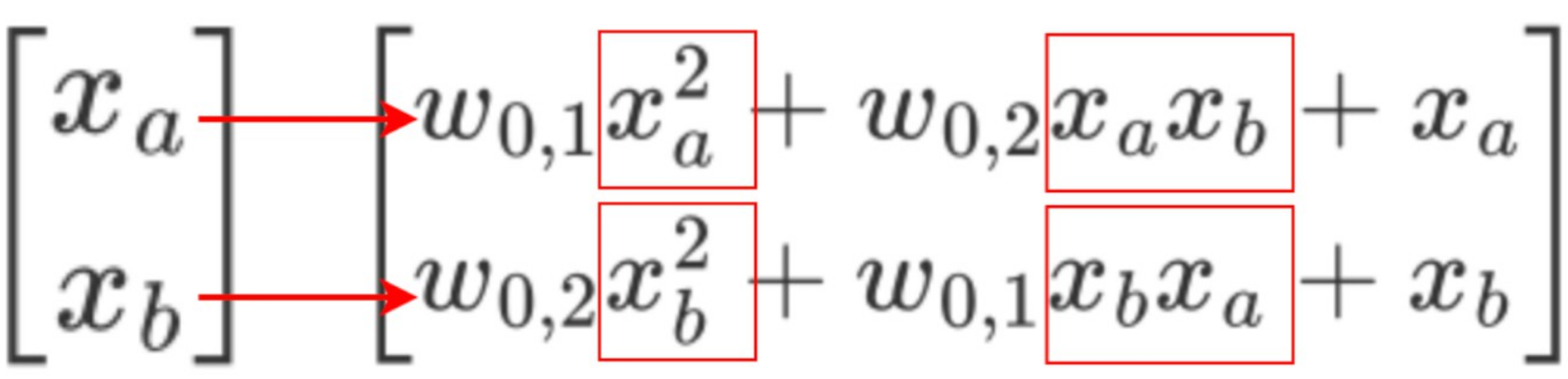
$$X_0 = \begin{bmatrix} x_a \\ x_b \end{bmatrix}$$

则进行一次特征交叉后有：

$$\begin{aligned} X_1 &= X_0 X_0^T W_0 + X_0 \\ &= \begin{bmatrix} x_a \\ x_b \end{bmatrix} [x_a \ x_b] \begin{bmatrix} w_{0,1} \\ w_{0,2} \end{bmatrix} + \begin{bmatrix} x_a \\ x_b \end{bmatrix} \\ &= \begin{bmatrix} x_a^2 & x_a x_b \\ x_b x_a & x_b^2 \end{bmatrix} \begin{bmatrix} w_{0,1} \\ w_{0,2} \end{bmatrix} + \begin{bmatrix} x_a \\ x_b \end{bmatrix} \end{aligned}$$

$$= \begin{bmatrix} w_{0,1}x_a^2 + w_{0,2}x_a x_b + x_a \\ w_{0,2}x_b^2 + w_{0,1}x_b x_a + x_b \end{bmatrix}$$

仔细看，从 X_0 中的两个特征项到 X_1 的两个特征项由哪些变化？变化如下图所示：

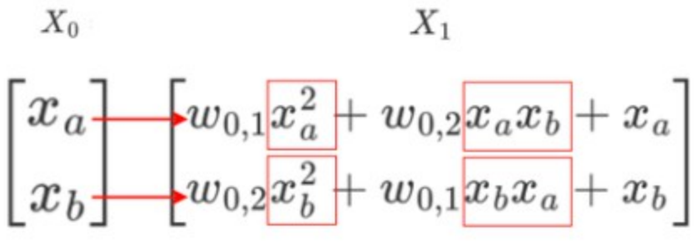


观察出来了吗?

从 X_1 中即可看出 x_a, x_b 两个特征既有自己和自己的交叉（如： x_a^2 ），又有两个特征间的交叉（如： $x_a x_b$)。如果没有观察出来的话，那我们就再计算一个 X_2 进一步观察吧：

$$\begin{aligned} X_2 &= X_0 X_1^T W_1 + X_1 \\ &= \begin{bmatrix} x_a \\ x_b \end{bmatrix} \begin{bmatrix} w_{0,1}x_a^2 + w_{0,2}x_a x_b + x_a & w_{0,1}x_b x_a + w_{0,2}x_b^2 + x_b \end{bmatrix} \begin{bmatrix} w_{1,1} \\ w_{1,2} \end{bmatrix} \\ &+ \begin{bmatrix} w_{0,1}x_a^2 + w_{0,2}x_a x_b + x_a \\ w_{0,1}x_b x_a + w_{0,2}x_b^2 + x_b \end{bmatrix} \\ &= \begin{bmatrix} w_{0,1}w_{1,1}x_a^3 + w_{0,2}w_{1,1}x_a^2 x_b + w_{1,1}x_a^2 + w_{0,1}w_{1,2}x_b x_a^2 + w_{0,2}w_{1,2}x_b^2 x_a + w_{1,2} \\ w_{0,1}w_{1,1}x_a^2 x_b + w_{0,2}w_{1,1}x_a x_b^2 + w_{1,1}x_a x_b + w_{0,1}w_{1,2}x_b^2 x_a + w_{0,2}w_{1,2}x_b^3 + w_1 \end{bmatrix} \\ &+ \begin{bmatrix} w_{0,1}x_a^2 + w_{0,2}x_a x_b + x_a \\ w_{0,1}x_b x_a + w_{0,2}x_b^2 + x_b \end{bmatrix} \\ &= \begin{bmatrix} w_{0,1}w_{1,1}x_a^3 + (w_{0,1} + w_{1,1})x_a^2 + x_a + w_{0,2}w_{1,1}x_a^2 x_b + w_{0,2}w_{1,2}x_b^2 x_a + w_{0,1}w_1 \\ w_{0,2}w_{1,2}x_b^3 + (w_{1,2} + w_{0,2})x_b^2 + x_b + w_{0,1}w_{1,2}x_b^2 x_a + w_{0,1}w_{1,1}x_a^2 x_b + w_{0,2}w_1 \end{bmatrix} \end{aligned}$$

现在将 X_0, X_1, X_2 合起来看看吧：



$$X_0 \begin{bmatrix} x_a \\ x_b \end{bmatrix}$$

$$X_1 \begin{bmatrix} w_{0,1}x_a^2 + w_{0,2}x_a x_b + x_a \\ w_{0,2}x_b^2 + w_{0,1}x_b x_a + x_b \end{bmatrix}$$

$$X_2 \begin{bmatrix} w_{0,1}w_{1,1}x_a^3 + (w_{0,1}+w_{1,1})x_a^2 + x_a + w_{0,2}w_{1,1}x_a^2 x_b + w_{0,2}w_{1,2}x_b^2 x_a + w_{0,1}w_{1,2}x_b x_a^2 + w_{0,2}x_a x_b + w_{1,2}x_b x_a \\ w_{0,2}w_{1,2}x_b^3 + (w_{1,2}+w_{0,2})x_b^2 + x_b + w_{0,1}w_{1,2}x_b^2 x_a + w_{0,1}w_{1,1}x_a^2 x_b + w_{0,2}w_{1,1}x_a x_b^2 + w_{1,1}x_a x_b + w_{0,1}x_b x_a \end{bmatrix}$$

从上面可以看出：

X_0：进行0次特征交叉，交叉项的次数最高为1，即：各特征独立，$x_a; x_b$ 。

X_1：进行1次特征交叉，交叉项的次数最高为2，且会出现次数为2的所有交叉情况，即：

$x_a -> x_a^2, x_a x_b;$
$x_b -> x_b^2, x_b x_a$。

X_2：进行2次特征交叉，交叉项的次数最高为3，且会出现次数为3的所有交叉情况，即：

$x_a -> x_a^2, x_a x_b -> x_a^3 \ , x_a^2 x_b, x_a x_b^2$
$x_b -> x_b^2, x_b x_a -> x_b^3 \ , x_b^2 x_a, x_b x_a^2$

由上不难推理出：

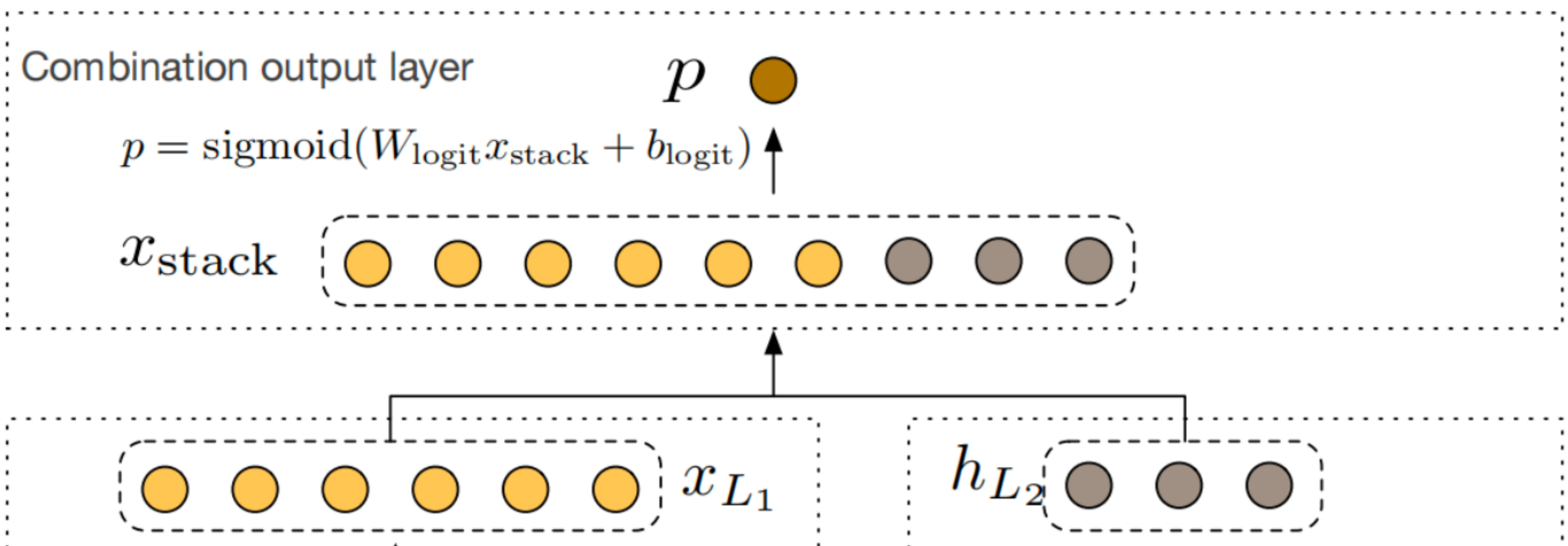
X_1 中最多可以达到两项特征交叉;

X_2 中最多可以达到三项特征交叉；

.....

一般来说，交叉项越多，越能挖掘特征组合间的有效性，这就是DCN特征交叉能力强的主要原因。

2.4 Combination output layer



最后一层就相对简单，将DeepNet和CorssNet的输出拼接（concat）起来，然后和一个矩阵做乘法，再加上一个偏置，最后进过一个sigmoid激活函数，得到一个0～1的值。这里如上图所示，较为简单，就不过多介绍了。

3. DCN特点分析

结合网上资料及个人理解，我觉得DCN模型，主要是其中的CrossNet的特点：

（1）自动特征交叉

CrossNet无需人工进行特征工程，会自行交叉每个特征，极大地解放人力；

（2）特征交叉能力强

由1.3节分析可知，每增加一层，特征交叉的次数就加一次，层数越多，特征交叉的深度就越深，越能挖掘出特征间的潜在联系。

（3）巧夺天工的公式设计

我们回过头来仔细观察一下CrossNet的公式：

$$X_{i+1} = X_0 X_i^T W_i + X_i + b_i$$

大家好好看看： X_0 是输入，可以看作是常数， W_i 可以看作是函数 F ，那么我们是不是可以把它换成另外一种写法呢：

$$X_{i+1} = F(X_i) + X_i + b_i$$

这下看起来是不是非常熟悉了，它不就类似于残差连接的公式吗？不信的话，我把残差连接公式写出来：

$$y = f(x) + x$$

现在，对比一下，是不是越看越像了 。

所以残差连接有的特点，CrossNet也有，主要特点就是：

避免梯度消失，无惧深层网络 !!!

以上就是DCN的特点，当然它也有其他特点，如：网络参数随层数线性变化以及论文中提到的贡献，但我个人理解最有特点的就是上面三种了。

欢迎评论区交流讨论！！！

4. 代码

pytorch代码如下：

```
import torch
from torch import nn
from DeepRecommand.pytorch.FeatureEmbedding.criteo_feature_embedding_v1 import


class Deep(nn.Module):
    def __init__(self, hidden_layers, dropout_p=0.0):
        """
        Deep: 深度网络
        Args:
            hidden_layers: deep网络的隐层维度, 如[128, 64, 32]
            dropout_p: dropout_p值. Defaults to 0.0.
        """
        super(Deep, self).__init__()
        self.dnn = nn.ModuleList()
        for layer in list(zip(hidden_layers[:-1], hidden_layers[1:])):
            linear = nn.Linear(in_features=layer[0], out_features=layer[1])
            self.dnn.append(linear)
        self.dropout = nn.Dropout(p=dropout_p)
```

```
    def forward(self, X):
        """
        Args:
            X: 输入数据, shape=(batch_size, input_dim)
        Returns:
            res: deep网络的输出, shape=(batch_size, hidden_layers[-1])
        """
        for linear in self.dnn:
            X = linear(X)
        res = self.dropout(X)
        return res


class CrossInteraction(nn.Module):
    def __init__(self, input_dim):
        """
        CrossInteraction: 交叉网络的单层
        Args:
            input_dim: 输入维度
        """
        super(CrossInteraction, self).__init__()
        self.w = nn.Linear(input_dim, 1, bias=False)
        self.b = nn.Parameter(torch.rand(input_dim))

    def forward(self, X_i, X_0):
        """
        Args:
            X_i: 本层的输入, shape=(batch_size, input_dim)
            X_0: 第0层的输入, shape=(batch_size, input_dim)
        Returns:
            out: 本层的输出, shape=(batch_size, input_dim)
        """
        out = self.w(X_i) * X_0 + self.b
        return out


class CrossNet_v1(nn.Module):
    def __init__(self, input_dim, num_layers):
        """
        CrossNet_v1: 交叉网络
        Args:
            input_dim: 输入维度
            num_layers: cors网络的层数
        """
```

```
        super(CrossNet_v1, self).__init__()
        self.num_layers = num_layers
        self.corss_layers = nn.ModuleList(
            CrossInteraction(input_dim) for _ in range(num_layers)
        )

    def forward(self, X_0):
        """
        Args:
            X_0: 网络输入, shape=(batch_size, input_dim)
        Returns:
            X_i: 网络输出, shape=(batch_size, input_dim)
        """
        X_i = X_0
        for i in range(self.num_layers):
            X_i = X_i + self.corss_layers[i](X_i, X_0)  # 注意：这个地方不要写成X_
        return X_i

class DCN(nn.Module):
    def __init__(self, feature_map, model_config):
        """
        DCN_v1: 即常说的DCN(Deep Cross Net✦work)
        Args:
            feature_map: 特征map
            model_config: 模型配置
        """
        super(DCN, self).__init__()
        self.input_dim = feature_map["sample_len"]
        if model_config["hidden_layers"][0] != self.input_dim:
            model_config["hidden_layers"].insert(0, self.input_dim)    # 因为第一
        if model_config['hidden_layers'][-1] != 1:
            model_config['hidden_layers'].append(1)
        self.hidden_layers = model_config['hidden_layers']
        self.num_cross_layers = model_config['num_cross_layers']
        self.dropout_p = model_config['dropout_p']

        self.embedding_layer = CriteoFeatureEmbedding(feature_map=feature_map)

        self.cross = CrossNet_v1(self.input_dim, self.num_cross_layers)
        self.deep = Deep(self.hidden_layers, self.dropout_p)
        self.fc = nn.Linear(self.hidden_layers[-1] + self.input_dim, 1)
        self.sigmoid = nn.Sigmoid()
```

```
    def forward(self, X):
        """
        Args:
            X: 输入数据, shape=(batch_size, input_dim)
        Returns:
            y_pred: 预测值, shape=(batch_size, 1)
        """
        X = self.embedding_layer(X) # 先进行特征嵌入，映射成为稠密向量
        cross_out = self.cross(X)
        deep_out = self.deep(X)
        concat_out = torch.cat([cross_out, deep_out], dim=-1)
        y_pred = self.fc(concat_out)
        y_pred = self.sigmoid(y_pred)
        return y_pred


if __name__ == "__main__":
    X = torch.randn(4, 128)
    # model = DCN_v1(128, [128, 64, 32])
    # y_pred = model(X)
    # print(y_pred.shape)
```

更多代码见：EasyDeepRecommand

5. EasDeepRecommand个人推荐系统开源项目介绍

链接如下：

EasyDeepRecommand
github.com/lamctb/EasyDeepRecommand

一个通俗易懂的开源推荐系统（A user-friendly open-source project for recommendation systems）.

本项目将结合：**代码、数据流转图、博客、模型发展史** 等多个方面通俗易懂地讲解经典推荐模型，让读者通过一个项目了解推荐系统概况！

持续更新中..., 欢迎star , 第一时间获取更新，感谢!!!

编辑于 2025-03-03 14:15 · IP 属地北京