

推荐系统系列（十）：DCN理论与实践

原创 默存 SOTA Lab 2019-12-08

前言

在Wide&Deep [2]之后，2017年Stanford与Google联合推出了Deep&Cross Network（DCN）[1]。该模型主要特点在于提出Cross network，用于特征的自动化交叉编码。传统DNN对于高阶特征的提取效率并不高，Cross Network通过调整结构层数能够构造出有限阶（bounded-degree）交叉特征，对特征进行显式交叉编码，在精简模型参数的同时有效的提高了模型的表征能力。

从模型结构上来看，DCN是将Wide&Deep中的Wide侧替换为Cross Network，利用该部分自动交叉特征的能力，模型无需进行额外的特征工程工作。同时，DCN参考了Deep Crossing[3]模型引入了残差结构的思想，使得模型能够更深。熟悉DeepFM [4]的同学可能觉得DCN与DeepFM也有几分相似，并且二者都采用了底层特征共享的模式。但二者其实是同时期提出的工作，所以并无参考之说。言归正传，接下来让我们一起来对模型进行分析学习。

分析

1. DCN模型结构

模型结构如下，共分为4个部分，分别为 Embedding and Stacking Layer（特征预处理输入）、Cross network（自动化特征显式交叉）、Deep network（特征隐式交叉）和Combination output Layer（输出）。

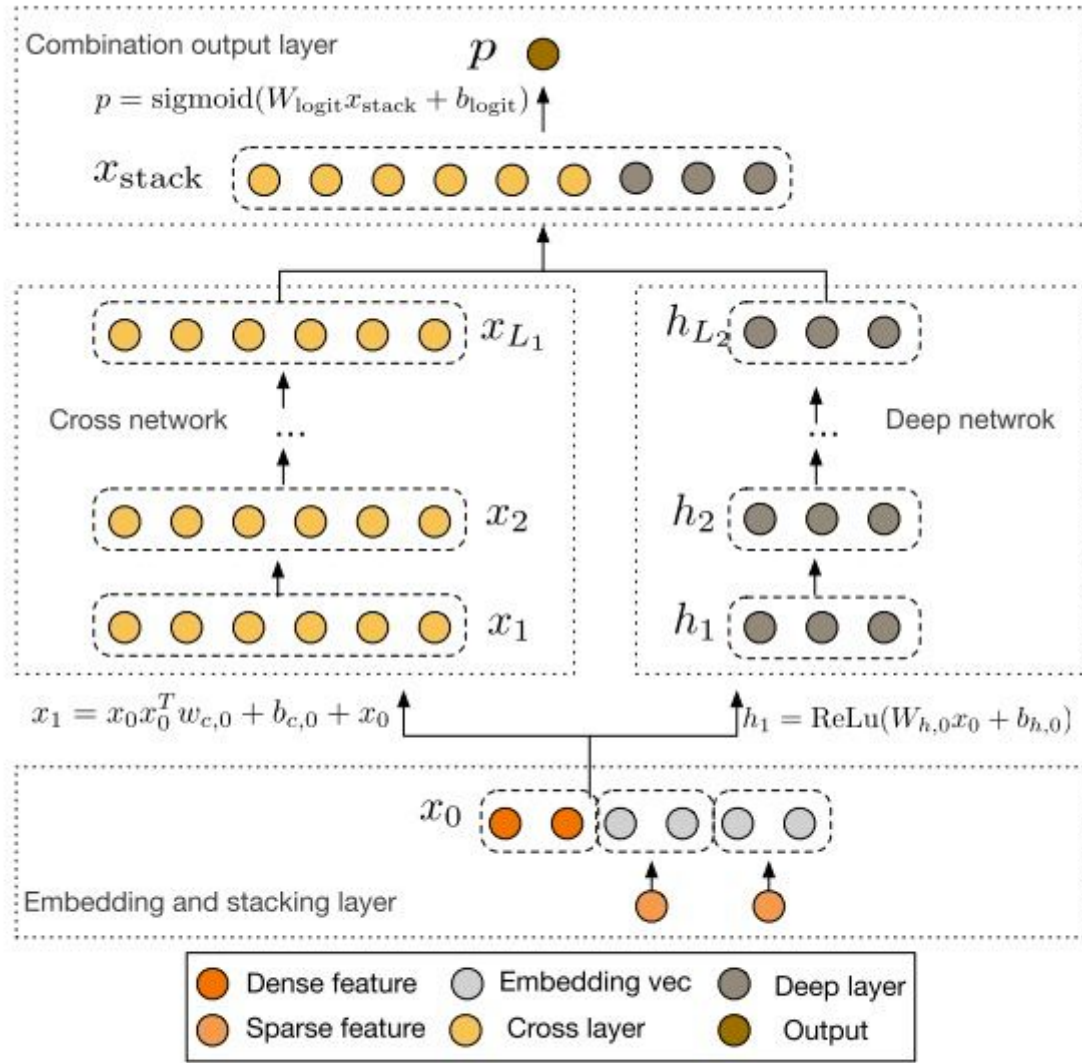


Figure 1: The Deep & Cross Network SOTA Lab

1.1 Embedding and Stacking Layer

常规操作，首先针对原始特征进行预处理，其中类别特征（Sparse feature）可以通过二值化处理，然后进行特征Embedding，将高维稀疏特征转化为低维稠密的实值向量（Embedding vec），再拼接其他连续特征（Dense feature）作为模型的输入。

$$X_0 = [X_{\text{embed},1}^T, \dots, X_{\text{embed},d}^T, X_{\text{dense}}^T] \quad (1)$$

其中 $X_{\text{embed},i}^T$ 代表特征Embedding vec， X_{dense}^T 代表连续实值特征向量。为了方便与下面的介绍保持统一，假设 $X_0 \in \mathbb{R}^d$ ， $X_0 = [x_1, x_2, \dots, x_d]$ 。

1.2 Cross Network

这个部分是模型的核心，数学表达式如下：

$$X_{l+1} = X_0 X_l^T W_l + b_l + X_l = f(X_l, W_l, b_l) + X_l \quad (2)$$

其中 $X_l, X_{l+1} \in \mathbb{R}^d$ 分别代表Cross Network的第 $l, l+1$ 层的输出， $W_l, b_l \in \mathbb{R}^d$ 分别为该层的参数与偏置项。因为 $f(X_l, W_l, b_l) = X_{l+1} - X_l$ ，所以函数 $f: \mathbb{R}^d \mapsto \mathbb{R}^d$ 是拟合 X_{l+1} 与 X_l 的残差，这个思想与Deep Crossing一致。

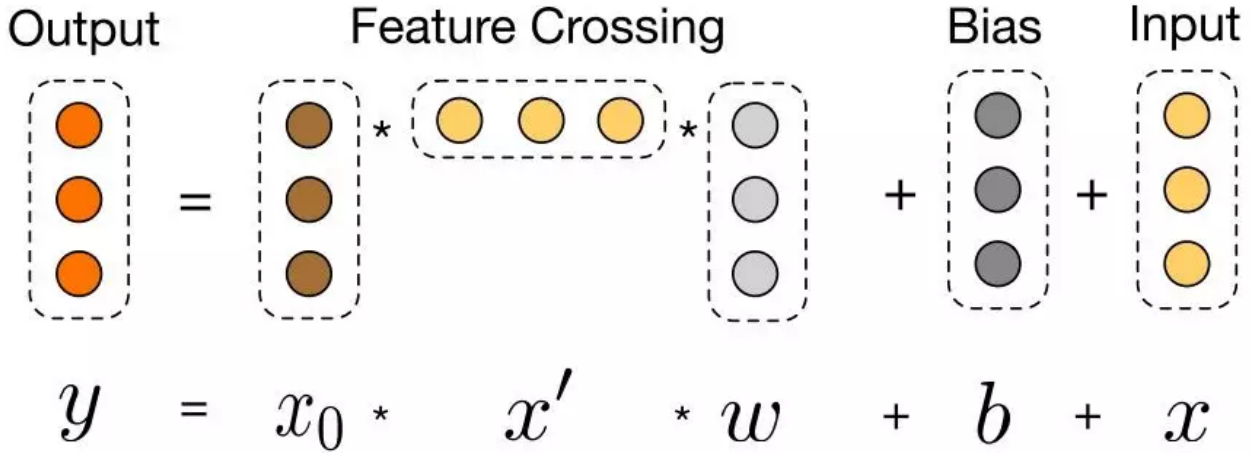



Figure 2: Visualization of a cross layer  SOTA Lab

公式（2）对应的图形化表示如图Fig 2，图中所有的向量维度都是一样的。利用 X_0 与 X' 做向量外积得到所有的元素交叉组合，层层叠加之后便可得到任意有界阶组合特征，当cross layer叠加 l 层时，交叉最高阶可以达到 $l+1$ 阶，参考[5]可以举例说明：

为了方便起见，首先将 b 设置为零向量，令 $X_0 = \begin{bmatrix} x_{0,1} \\ x_{0,2} \end{bmatrix}$ ，那么

$$\begin{aligned}
 X_1 &= X_0 X_0' W_0 + X_0 \\
 &= \begin{bmatrix} x_{0,1} \\ x_{0,2} \end{bmatrix} [x_{0,1} \ x_{0,2}] \begin{bmatrix} w_{0,1} \\ w_{0,2} \end{bmatrix} + \begin{bmatrix} x_{0,1} \\ x_{0,2} \end{bmatrix} \\
 &= \begin{bmatrix} x_{0,1}^2 & x_{0,1}x_{0,2} \\ x_{0,2}x_{0,1} & x_{0,2}^2 \end{bmatrix} \begin{bmatrix} w_{0,1} \\ w_{0,2} \end{bmatrix} + \begin{bmatrix} x_{0,1} \\ x_{0,2} \end{bmatrix} \\
 &= \begin{bmatrix} w_{0,1}x_{0,1}^2 + w_{0,2}x_{0,1}x_{0,2} \\ w_{0,1}x_{0,2}x_{0,1} + w_{0,2}x_{0,2}^2 \end{bmatrix} + \begin{bmatrix} x_{0,1} \\ x_{0,2} \end{bmatrix} \\
 &= \begin{bmatrix} w_{0,1}x_{0,1}^2 + w_{0,2}x_{0,1}x_{0,2} + x_{0,1} \\ w_{0,1}x_{0,2}x_{0,1} + w_{0,2}x_{0,2}^2 + x_{0,2} \end{bmatrix}
 \end{aligned} \tag{3}$$

继续计算 X_2 ，有：

$$\begin{aligned}
X_2 &= X_0 X_1' W_1 + X_1 \\
&= \begin{bmatrix} x_{0,1} \\ x_{0,2} \end{bmatrix} \begin{bmatrix} w_{0,1}x_{0,1}^2 + w_{0,2}x_{0,1}x_{0,2} + x_{0,1} & w_{0,1}x_{0,2}x_{0,1} + w_{0,2}x_{0,2}^2 + x_{0,2} \end{bmatrix} \begin{bmatrix} w_{1,1} \\ w_{1,2} \end{bmatrix} \\
&\quad + \begin{bmatrix} w_{0,1}x_{0,1}^2 + w_{0,2}x_{0,1}x_{0,2} + x_{0,1} \\ w_{0,1}x_{0,2}x_{0,1} + w_{0,2}x_{0,2}^2 + x_{0,2} \end{bmatrix} \\
&= \begin{bmatrix} w_{0,1}x_{0,1}^3 + w_{0,2}x_{0,1}^2x_{0,2} + x_{0,1}^2 & w_{0,1}x_{0,2}x_{0,1}^2 + w_{0,2}x_{0,2}^2x_{0,1} + x_{0,2}x_{0,1} \\ w_{0,1}x_{0,1}^2x_{0,2} + w_{0,2}x_{0,1}x_{0,2}^2 + x_{0,1}x_{0,2} & w_{0,1}x_{0,2}^2x_{0,1} + w_{0,2}x_{0,2}^3 + x_{0,2}^2 \end{bmatrix} \begin{bmatrix} w_{1,1} \\ w_{1,2} \end{bmatrix} \\
&\quad + \begin{bmatrix} w_{0,1}x_{0,1}^2 + w_{0,2}x_{0,1}x_{0,2} + x_{0,1} \\ w_{0,1}x_{0,2}x_{0,1} + w_{0,2}x_{0,2}^2 + x_{0,2} \end{bmatrix} \\
&= \begin{bmatrix} w_{0,1}w_{1,1}x_{0,1}^3 + w_{0,2}w_{1,1}x_{0,1}^2x_{0,2} + w_{1,1}x_{0,1}^2 + w_{0,1}w_{1,2}x_{0,2}x_{0,1}^2 + w_{0,2}w_{1,2}x_{0,2}^2x_{0,1} + w_{1,2}x_{0,2}x_{0,1} \\ w_{0,1}w_{1,1}x_{0,1}^2x_{0,2} + w_{0,2}w_{1,1}x_{0,1}x_{0,2}^2 + w_{1,1}x_{0,1}x_{0,2} + w_{0,1}w_{1,2}x_{0,2}^2x_{0,1} + w_{0,2}w_{1,2}x_{0,2}^3 + w_{1,2}x_{0,2}^2 \\ w_{0,1}x_{0,1}^2 + w_{0,2}x_{0,1}x_{0,2} + x_{0,1} \\ w_{0,1}x_{0,2}x_{0,1} + w_{0,2}x_{0,2}^2 + x_{0,2} \end{bmatrix}
\end{aligned} \quad ($$

从公式（3）（4）的标红处可以看出，当cross layer叠加 l 层时，交叉最高阶可以达到 $l+1$ 阶，并且包含了所有的交叉组合，这是DCN的精妙之处。

复杂性分析：

一般来说，要对特征进行高阶显式交叉，一定会加大模型的参数量。在DCN中，假设Cross Layer有 L_c 层，最底层输入 X_0 为 d 维。因为每一层仅有 W, b 参数，所以模型参数量会额外增加 $d \times L_c \times 2$ 个。

因为 $X_0 X_l' W_l = X_0 (X_l' W_l)$ ，先计算 $X_l' W_l$ 得到一个标量，然后再与 X_0 相乘，在时间与空间上计算效率都得到提升，最终 L_c 层 Cross Layer 的时空复杂度均为 $O(dL_c)$ ，也就是说时空复杂度随着输入与层数线性增长，这是非常好的性质。

让我们更进一步分析一下Cross Layer的设计理念，通常来说当两个向量的外积之后，往往想到的是再利用一个矩阵来对结果进行压缩变换，假设向量外积之后为 $d \times n$ 维矩阵，那么为了将结果变换为 d 维向量，需要使用的参数矩阵为 $n \times d$ 维，不仅参数量变多了，而且矩阵相乘的运算复杂度高达三次方。

参数量过多很容易造成过拟合，参数量的适当精简反而可以提高模型的泛化能力与鲁棒性。如Cross Layer中使用向量而不是矩阵来对结果进行变换，本质上是通过参数共享的模式减少参数量。共享参数能够对样本外数据仍有较好的泛化能力，且能够对噪声数据带来的参数变化进行纠正。

1.3 Deep Network

与之前的Wide&Deep一样，这个部分是简单的DNN结构。可以表示为：

$$h_{l+1} = f(W_l h_l + b_l) \quad (5)$$

其中 h_l, h_{l+1} 分别为第 $l, l+1$ 层的输出, W_l 与 b_l 为参数与偏置项, f 为 relu 激活函数。假设 DNN 中的隐层节点为 m 个, 共有 L_d 层, 且最底层输入为 d 维, 那么 DNN 部分参数数量为 $d \times m + m + (m^2 + m) \times (L_d - 1)$ 。

1.4 Combination Output Layer

将 Cross Network 与 Deep Network 部分的输出进行简单拼接, 通过激活函数作为最后的输出。

$$p = \sigma([X_{L_1}^T, h_{L_2}^T]W_{logits}) \quad (6)$$

其中 $X_{L_1}^T \in \mathbb{R}^d$, $h_{L_2}^T \in \mathbb{R}^m$ 分别为 Cross Network 与 Deep Network 的输出, $W_{logits} \in \mathbb{R}^{d+m}$ 是变换矩阵, σ 为 sigmoid 激活函数。

模型使用的 Loss 函数为 logloss , 并且加入了正则项:

$$\text{loss} = -\frac{1}{N} \sum_{i=1}^N y_i \log(p_i) + (1 - y_i) \log(1 - p_i) + \lambda \sum \|W\|^2 \quad (7)$$

2. 性能对比

作者使用 *Criteo* 数据集进行实验对比, 对比模型有 LR、FM、DNN、Deep Crossing。至于为什么没有 Wide&Deep 模型进行对比, 在原文中提到, 因为 Wide 部分需要手工构造合适的交叉特征, 这需要领域知识来对特征进行选择, 不方便作为对比试验。

W&D. Different than DCN, its wide component takes as input raw sparse features, and relies on exhaustive searching and domain knowledge to select predictive cross features. We skipped the comparison as no good method is known to select cross features.

几个模型的对比结果如下, 结果来看 DCN 的确表现更优。

Table 1: Best test logloss from different models. “DC” is deep crossing, “DNN” is DCN with no cross layer, “FM” is Factorization Machine based model, “LR” is logistic regression.

Model	DCN	DC	DNN	FM	LR
Logloss	0.4419	0.4425	0.4428	0.4464	0.4474

SOTA Lab

令人不能理解的是，排除了Wide&Deep模型之后，后续的对比实验都是DCN与DNN模型的对比....作者重点对比的是二者的效率问题，DCN使用更少的参数量能够达到更优的效果（感觉没什么信服力）。

Table 2: #parameters needed to achieve a desired logloss.

Logloss	0.4430	0.4460	0.4470	0.4480
DNN	3.2×10^6	1.5×10^5	1.5×10^5	7.8×10^4
DCN	7.9×10^5	7.3×10^4	3.7×10^4	3.7×10^4

Table 3: Best logloss achieved with various memory budgets.

#Params	5×10^4	1×10^5	4×10^5	1.1×10^6	2.5×10^6
DNN	0.4480	0.4471	0.4439	0.4433	0.4431
DCN	0.4465	0.4453	0.4432	0.4426	0.4423

最后，作者设计实验观察不同Cross Layer层数与隐层节点数对结果带来的影响。

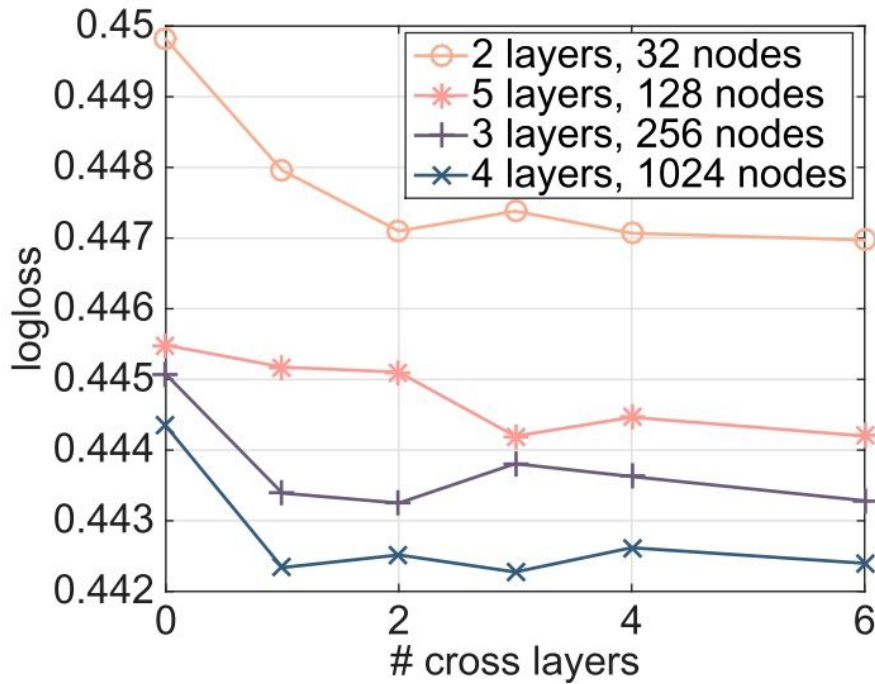


Figure 3: Improvement in the validation logloss with the growth of cross layer depth. The case with 0 cross layers is equivalent to a single DNN model. In the legend, “layers” is hidden layers, “nodes” is hidden nodes. Different symbols represent different hyperparameters for the deep network.

3. 实践

仍然使用 *MovieLens100K* 作为数据集，核心代码如下。

参数含义：

`vec_dim` : 代表embedding vector维度

`field_lens` : list结构，其中每个元素代表对应Field有多少取值。例如gender有两个取值，那么其对应的元素为2

`cross_layer_num` : cross network 层数

`dnn_layers` : list结构，其中每个元素对应DNN部分节点数目

`lr` : 学习率

```

class DCN(object):

    def __init__(self, vec_dim=None, field_lens=None, cross_layer_num=None, dnn_layers=None, lr=None, dropout_rate=None):
        self.vec_dim = vec_dim
        self.field_lens = field_lens
        self.field_num = len(field_lens)
        self.feats_num = np.sum(field_lens)
        self.cross_layer_num = cross_layer_num
        self.dnn_layers = dnn_layers
        self.lr = lr
        self.dropout_rate = dropout_rate

        self._build_graph()

    def _build_graph(self):
        self.add_input()
        self.inference()

    def add_input(self):
        self.index = tf.placeholder(tf.int32, shape=[None, self.field_num], name='feat_index') # (batch, field_num)
        self.x = tf.placeholder(tf.float32, shape=[None, self.field_num], name='feat_value') # (batch, field_num)
        self.y = tf.placeholder(tf.float32, shape=[None], name='input_y')
        self.is_train = tf.placeholder(tf.bool)

    def cross_layer(self, x0, x1, name):
        with tf.variable_scope(name):
            node_num = self.field_num * self.vec_dim
            w = tf.get_variable(name='w', shape=[node_num], dtype=tf.float32)
            b = tf.get_variable(name='b', shape=[node_num], dtype=tf.float32)
            x1_w = tf.tensordot(x1, w, axes=[1,0]) # (batch, )
            x0_x1_w = tf.multiply(x0, tf.expand_dims(x1_w, -1)) # (batch, node_num)
            x = tf.add(x0_x1_w, b) # (batch, node_num)
            x = x+x1 # (batch, node_num)

        return x

    def inference(self):
        with tf.variable_scope('emb_part'):
            embed_matrix = tf.get_variable(name='second_ord_v', shape=[self.feats_num, self.vec_dim])
            embed_v = tf.nn.embedding_lookup(embed_matrix, self.index) # (batch, F, K)
            embed_x = tf.multiply(tf.expand_dims(self.x, axis=2), embed_v) # (batch, F, K)
            embed_x = tf.layers.dropout(embed_x, rate=self.dropout_rate, training=self.is_train) # (batch, F, K)
            node_num = self.field_num * self.vec_dim
            embed_x = tf.reshape(embed_x, shape=(-1, node_num)) # (batch, node_num)

        with tf.variable_scope('cross_part'):
            cross_vec = embed_x

```



```

    for i in range(self.cross_layer_num):
        cross_vec = self.cross_layer(embed_x, cross_vec, 'cross_layer_%d'%i) # (batch, node

with tf.variable_scope('dnn_part'):
    dnn = embed_x
    in_num = node_num
    for i in range(len(self.dnn_layers)):
        out_num = self.dnn_layers[i]
        w = tf.get_variable(name='w_%d'%i, shape=[in_num, out_num], dtype=tf.float32)
        b = tf.get_variable(name='b_%d'%i, shape=[out_num], dtype=tf.float32)
        dnn = tf.matmul(dnn, w) + b
        dnn = tf.layers.dropout(tf.nn.relu(dnn), rate=self.dropout_rate, training=self.is_t
        in_num = out_num

with tf.variable_scope('output_part'):
    in_num += node_num
    output = tf.concat([cross_vec, dnn], axis=1)
    proj_w = tf.get_variable(name='proj_w', shape=[in_num, 1], dtype=tf.float32)
    self.y_logits = tf.matmul(output, proj_w)

self.y_hat = tf.nn.sigmoid(self.y_logits)
self.pred_label = tf.cast(self.y_hat > 0.5, tf.int32)
self.loss = -tf.reduce_mean(self.y*tf.log(self.y_hat+1e-8) + (1-self.y)*tf.log(1-self.y_hat
self.train_op = tf.train.AdamOptimizer(self.lr).minimize(self.loss)

```

reference

- [1] Wang, R., Fu, B., Fu, G., & Wang, M. (2017, August). Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17* (p. 12). ACM.
- [2] Cheng, Heng-Tze, et al. "Wide & deep learning for recommender systems." *Proceedings of the 1st workshop on deep learning for recommender systems*. ACM, 2016.
- [3] Shan, Ying, et al. "Deep crossing: Web-scale modeling without manually crafted combinatorial features." *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, 2016.
- [4] Guo, Huifeng, et al. "DeepFM: a factorization-machine based neural network for CTR prediction." *arXiv preprint arXiv:1703.04247* (2017).
- [5] <https://zhuanlan.zhihu.com/p/55234968>