推荐系统系列（五）：Deep Crossing理论与实践

原创 默存 SOTA Lab 2019-11-12

背景

特征工程是绕不开的话题，巧妙的特征组合也许能够为模型带来质的提升。但同时，特征工程耗费的资源也是相当可观的，对于后期模型特征的维护、模型线上部署不太友好。2016年，微软提出Deep Crossing模型，旨在解决特征工程中特征组合的难题，降低人力特征组合的时间开销，通过模型自动学习特征的组合方式，也能达到不错的效果，且在各种任务中表现出较好的稳定性。

与之前介绍的FNN、PNN不同的是，Deep Crossing并没有采用显式交叉特征的方式，而是利用残差网络结构挖掘特征间的关系。本文将对DeepCrossing从原理到实现细节进行详细分析。

分析

1. DeepCrossing模型结构

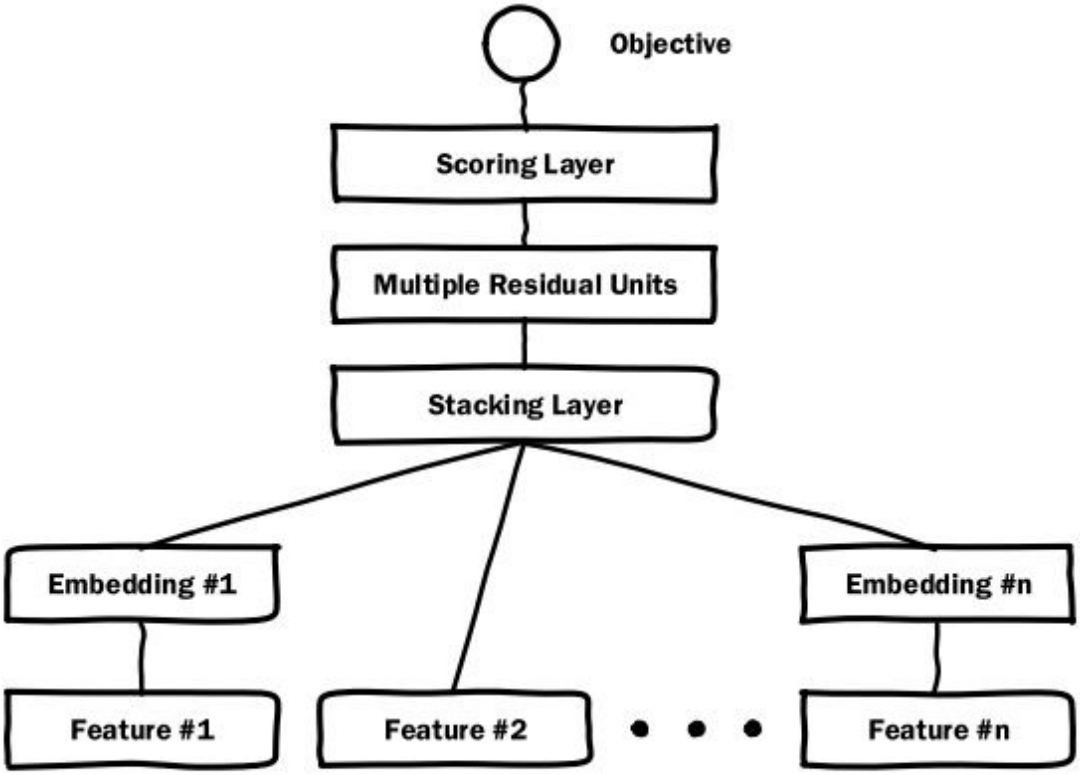


Figure 1: Deep Crossing Model Architecture

整个模型包含四种结构：Embedding，Stacking，Residual Unit，Scoring Layer。

论文中使用的目标函数为

$logloss$：$logloss = -\frac{1}{N}\sum_{i=1}^{N}(y_i log(p_i) + (1-y_i)log(1-p_i))$，在实际应用中，可以灵活替换为其他目标函数。

下面对各层结构进行分析：

1.1 Embedding & Stacking

Embedding的主要目的是将高维稀疏特征转化为低维稠密特征，其公式化定义为：$X_j^O = max(0, W_j X_j^I + b_j)$，其中 X_j^I 代表输入的第 j 个特征Field，并且已经过one-hot编码表示，Wb 分别表示对应的模型参数。与前几篇paper介绍的Embedding过程不同的是，DeepCrossing加上了偏置项 b。公式中的 max 操作等价于使用 $relu$ 激活函数。

尽管可以通过分Field的操作，减少Embedding层的参数量，但是由于某些 **高基数特征** 的存在，如paper中提到的CampaignID，其对应的 W_j 仍然十分庞大。为此作者提出，针对这些高基数特征构造衍生特征，具体操作如下。根据CampaignID的历史点击率从高到低选择Top1000个，编号从0到999，将剩余的ID统一编号为1000。同时构建其衍生特征，将所有ID对应的历史点击率组合成1001维的稠密矩阵，各个元素分别为对应ID的历史CTR，最后一个元素为剩余ID的平均CTR。通过降维引入衍生特征的方式，可以有效的减少高基数特征带来的参数量剧增问题。

经过Embedding之后，直接对所有的 X_j^O 进行拼接Stacking，$X^O = [X_0^O, X_1^O, \cdots, X_K^O]$ 。作者将特征embedding为256维，但是对于本身维度低于256的特征Field，无需进行Embedding，直接送入Stacking层，如上图中的 $Feature\#2$ 所示。

1.2 Residual Unit

残差的网络结构如下：

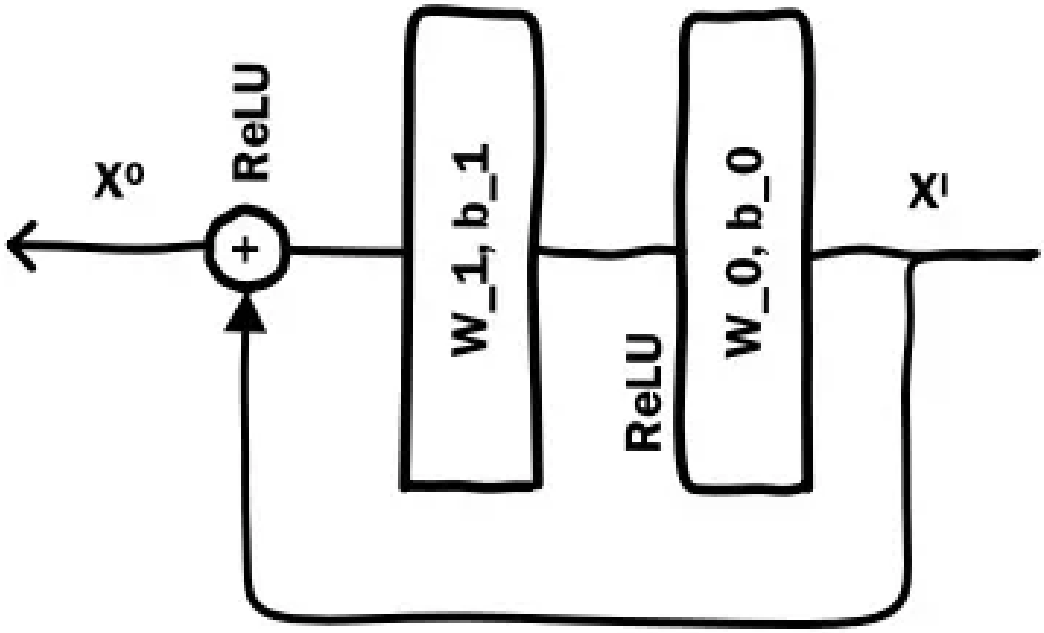


Figure 2: A Residual Unit

公式定义为：$X^O = F(X^I, \{W_0, W_1\}, \{b_0, b_1\}) + X^I$

将 X^I 移项到等式左侧，可以看出 F 函数拟合的是输入与输出之间的残差。对输入进行全连接变换之后，经过 $relu$ 激活函数送入第二个全连接层，将输出结果与原始输入进行 element-wise add 操作，再经过 $relu$ 激活输出。有分析说明，残差结构能更敏感的捕获输入输出之间的信息差 [2]。

作者通过各种类型各种大小的实验发现，DeepCrossing具有很好的鲁棒性，推测可能是因为残差结构能起到类似于正则的效果，但是具体原因是如何的并未明确指出，如果有同学了解具体原因，欢迎交流。

1.3 Scoring Layer

使用 $logloss$ 作为目标函数，可以灵活改用其他函数表示。

2. Early Crossing vs. Late Crossing

在paper中，作者针对特征交叉的时间点先后的问题进行试验对比。在DeepCrossing中，特征是在Embedding之后就开始进行交叉，但是有一些模型如DSSM，是在各类特征单独处理完成之后再进行交叉计算，这类模型的结构如下所示：

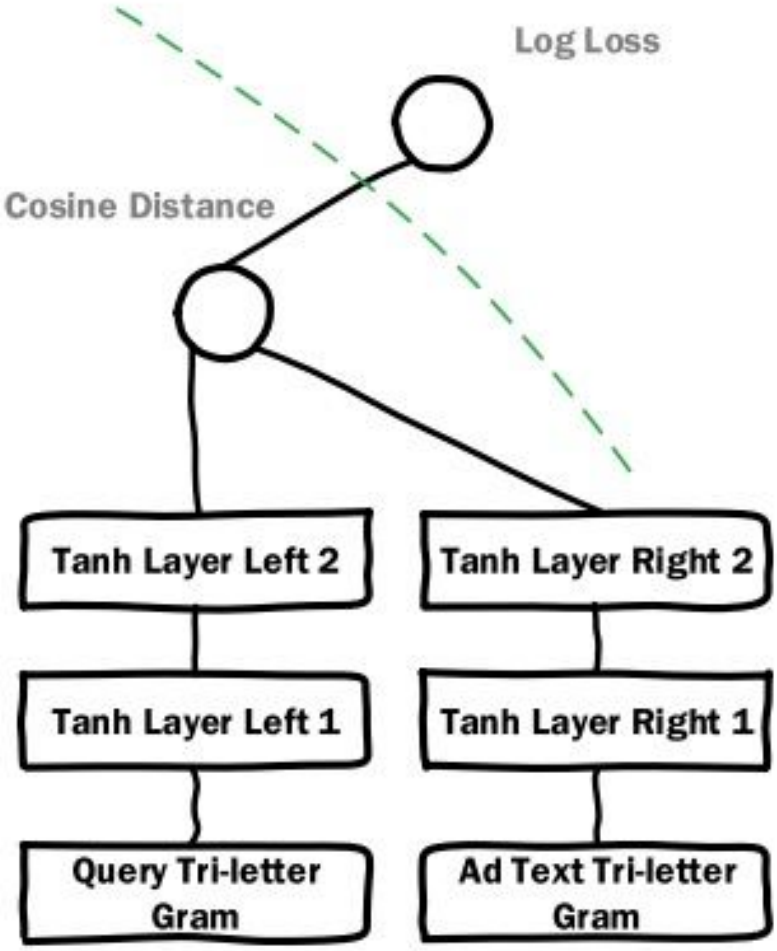


Figure 3: A DSSM Model with Log Loss

文中提到，DSSM更擅长文本处理，设计文本处理相关实验，DeepCrossing比DSSM表现更优异。作者认为，DeepCrossing表现优异主要来源于：1）残差结构；2）及早的特征交叉处理；

3. 性能分析

3.1 文本输入

输入特征相同，以DSSM作为baseline，根据Table 3可以看出DeepCrossing相对AUC更高。

Training Data	DSSM	Deep Crossing
text_cp1_tn_s	100	100.46
text_cp1_tn_b	100	101.02

Table 3: Click prediction results for task CP1 with a pair of text inputs where performance is measured by relative AUC using DSSM as the baseline

将生产环境的已有模型Production作为baseline进行对比，虽然DeepCrossing比DSSM表现更好，但稍逊Production。这是因为Production的训练数据集不同，且有更为丰富的特征。

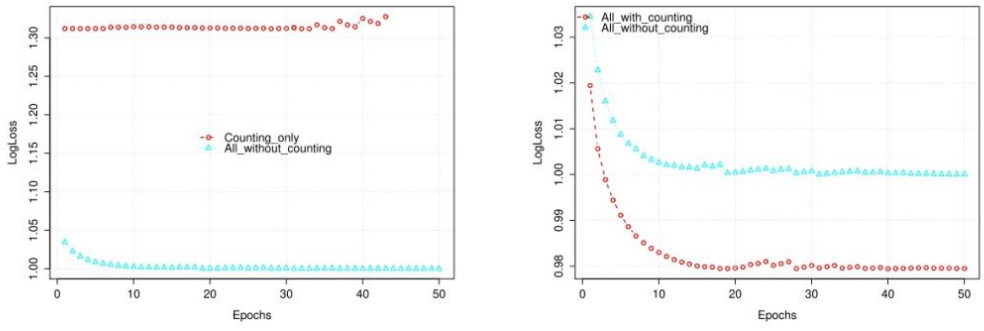
Test Data	DSSM	Deep Crossing	Production
text_cp2_tt	98.68	98.99	100

Table 4: Click prediction results for task CP2 with a pair of text inputs where performance is measured by relative AUC using the production model as the baseline

3.2 其他对比

1）衍生特征Counting Feature的重要性比较

在1.1节中讨论过，为了对高基数特征进行降维处理，引入了统计类衍生特征（称之为Counting Feature）。对比此类特征对于模型的影响，从实验结果可以看出衍生特征能够带来较大提升。



(a) A model (in red) using the selected counting feature only is weak

(b) Adding the selected counting feature reduced log loss significantly

Figure 6: The effect of adding the selected counting feature on task CP1 in Deep Crossing where performance is measured by relative log loss on the validation set over training epochs (relative log loss is defined in Sec.7.2)

2）与生产环境模型Production进行比较

使用Production训练特征的子集，使用22亿条数据进行训练。最终DeepCrossing表现超过了Production。

Test Data	Deep Crossing	Production
all_cp1_tt	101.02	100

Table 5: Click prediction model compared with the production model on task CP1 where performance is measured by relative AUC using production model as the baseline

实验

使用 $MovieLens100K dataset$ ，核心代码如下。

```
class DeepCrossing(object):
    def __init__(self, vec_dim=None, field_lens=None, lr=None, residual_unit_num=None, residual_
        self.vec_dim = vec_dim
        self.field_lens = field_lens
        self.field_num = len(field_lens)
        self.lr = lr
        self.residual_unit_num = residual_unit_num
        self.residual_w_dim = residual_w_dim
        self.dropout_rate = dropout_rate
        self.lamda = float(lamda)

        self.l2_reg = tf.contrib.layers.l2_regularizer(self.lamda)

        self._build_graph()

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

    def add_input(self):
        self.x = [tf.placeholder(tf.float32, name='input_x_%d'%i) for i in range(self.field_num)
        self.y = tf.placeholder(tf.float32, shape=[None], name='input_y')
        self.is_train = tf.placeholder(tf.bool)

    def _residual_unit(self, input, i):
        x = input
        in_node = self.field_num*self.vec_dim
        out_node = self.residual_w_dim
        w0 = tf.get_variable(name='residual_w0_%d'%i, shape=[in_node, out_node], dtype=tf.float3
        b0 = tf.get_variable(name='residual_b0_%d'%i, shape=[out_node], dtype=tf.float32)
        residual = tf.nn.relu(tf.matmul(input, w0) + b0)
        w1 = tf.get_variable(name='residual_w1_%d'%i, shape=[out_node, in_node], dtype=tf.float3
        b1 = tf.get_variable(name='residual_b1_%d'%i, shape=[in_node], dtype=tf.float32)
```

```
            residual = tf.matmul(residual, w1) + b1
            out = tf.nn.relu(residual+x)
            return out

    def inference(self):
        with tf.variable_scope('emb_part'):
            emb = [tf.get_variable(name='emb_%d'%i, shape=[self.field_lens[i], self.vec_dim], dt
            emb_layer = tf.concat([tf.matmul(self.x[i], emb[i]) for i in range(self.field_num)],
        x = emb_layer
        with tf.variable_scope('residual_part'):
            for i in range(self.residual_unit_num):
                x = self._residual_unit(x, i)
                x = tf.layers.dropout(x, rate=self.dropout_rate, training=self.is_train)
        w = tf.get_variable(name='w', shape=[self.field_num*self.vec_dim, 1], dtype=tf.float32,
        b = tf.get_variable(name='b', shape=[1], dtype=tf.float32)

        self.y_logits = tf.matmul(x, w) + b
        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_
        reg_variables = tf.get_collection(tf.GraphKeys.REGULARIZATION_LOSSES)
        if len(reg_variables) > 0:
            self.loss += tf.add_n(reg_variables)
        self.train_op = tf.train.AdamOptimizer(self.lr).minimize(self.loss)
```

reference

[1] 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.

[2] https://zhuanlan.zhihu.com/p/72679537

[3] https://www.zhihu.com/question/20830906/answer/681688041

专注知识分享，欢迎关注 SOTA Lab~