推荐系统系列（九）：DeepFM理论与实践

原创 默存 SOTA Lab 2019-12-02

前言

当FM [2]与DNN相遇之后，相关的模型结构层出不穷，今天要介绍的DeepFM [1]便是其中的佼佼者。DeepFM是2017年由哈工大与华为诺亚方舟实验室联合推出的深度推荐模型，该模型在工业界应用广泛，其主要创新在于，将FM与DNN以并行结构组合在一起，FM侧与DNN侧共享特征嵌入层（Embedding Layer），通过联合训练的方式使模型达到最优。

DeepFM模型结构与Wide&Deep [3]很相似，二者最大的不同点在于，DeepFM两个部分共享底层输入特征，无需进行特征工程，而Wide&Deep模型两侧保持输入独立，其他异同点将在下文进行详细分析。话不多说，下面开始对模型进行拆解分析。

分析

1. DeepFM 定义

模型结构如下所示：

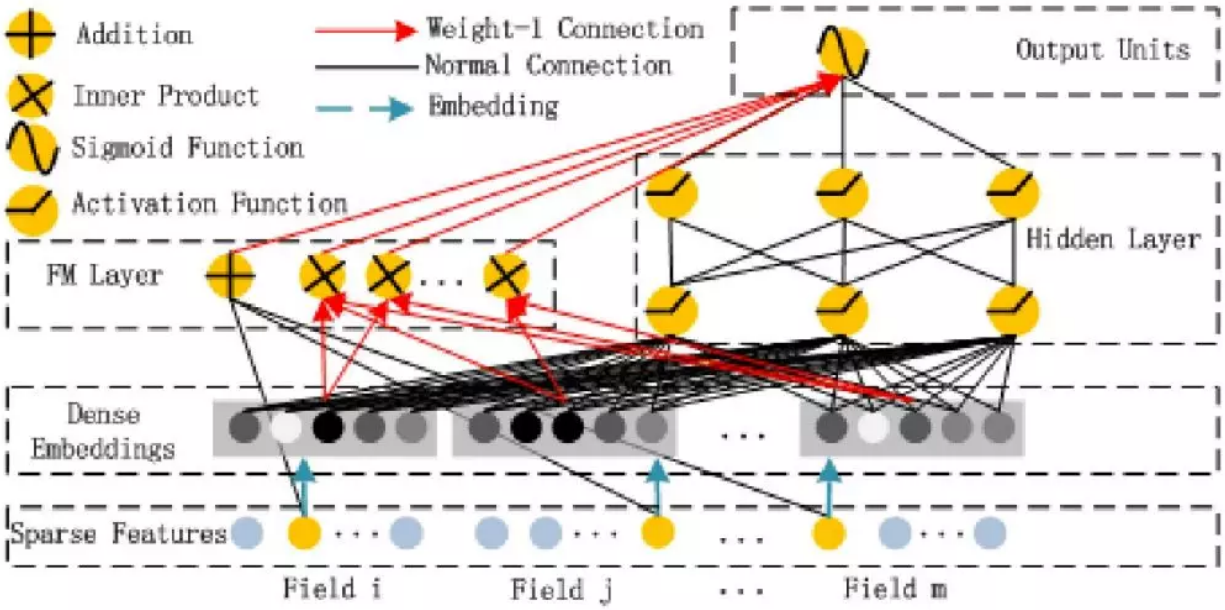


Figure 1: Wide & deep architecture of DeepFM. The wide and deep component share the same input raw feature vector, which enables DeepFM to learn low- and high-order feature interactions simultaneously from the input raw features.

SOTA Lab

整个模型分为FM与DNN两个部分，两部分共享输入层，并在输出层进行简单求和后通过激活函数输出。其数学化表示为：

$$\hat{y} = sigmoid(y_{FM} + y_{DNN}) \qquad (1)$$

1.1 FM 部分

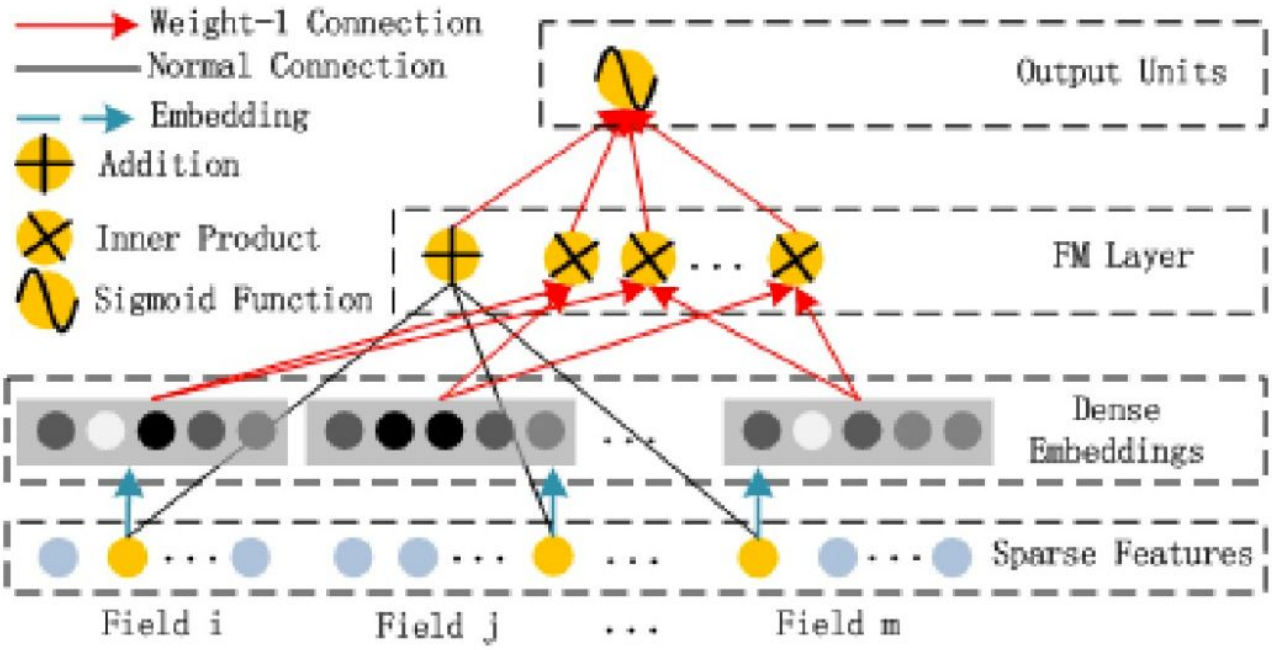


Figure 2: The architecture of FM.

SOTA Lab

该部分与FM完全等价，仅仅是通过神经网络来模拟FM的整个计算过程。回顾FM模型的数学定义为：

$$\hat{y}_{FM}= b+\sum_{i=1}^{m} w_i x_i+\sum_{i=1}^{n}\sum_{j=i+1}^{n}\langle v_i, v_j\rangle x_i x_j \qquad (2)$$

DeepFM为了简单起见，去除了FM原始定义中的偏置项 b，仅保留了公式（2）中的一阶项与二阶交叉项。有：

$$y_{FM}=\sum_{i=1}^{m} w_i x_i+\sum_{i=1}^{n}\sum_{j=i+1}^{n}\langle v_i, v_j\rangle x_i x_j \qquad (3)$$

在实践部分，FM模块最终是将一阶项与二阶项进行简单concat [6]。

上图中的Field为特征组，例如性别属性可以看做是一个Field，其有两个特征分别为“男”、“女”。通常来说一个Field中往往只有一个非零特征，但也有可能为多值Field，需要根据实际输入进行适配。

上图的图例中展示了三种颜色的线条，其中绿色的箭头表示为特征的Embedding过程，即得到特征对应的Embedding vector，通常使用 $v_i x_i$ 来表示，而其中的隐向量 v_i 则是通过模型学习得到的参数。红色箭头表示权重为1的连接，也就是说红色箭头并不是需要学习的参数。而黑色连线则表示为正常的，需要模型学习的参数 w_i 。

1.2 DNN 部分

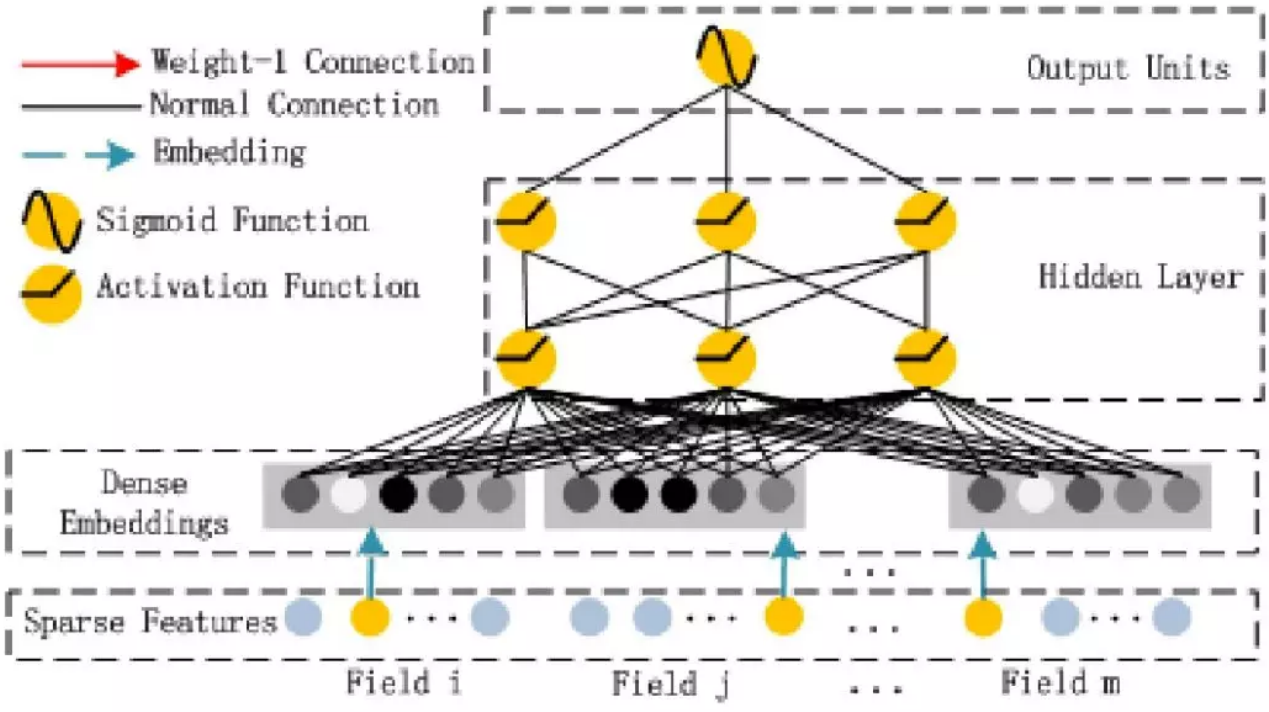


Figure 3: The architecture of DNN.

这个部分是简单的全连接网络，但是其中使用的输入：Dense Embeddings，与上述FM部分的是同一层。DNN与FM的Dense Embedding共享，这也是DeepFM与Wide&Deep模型最大的不同之处。

在FM部分中，Dense Embedding是为了计算二阶交叉特征信息，但在DNN部分，Dense Embedding是为更高阶的特征交叉信息提供输入（通常认为DNN能够学习高阶特征组合信息）。所以，Dense Embedding同时编码了低阶组合特征与高阶组合特征所需要的信息，作者认为这种方式更有利于模型性能的提升。并且这种共享参数的方式，不需要额外的特征工程，节省了模型构建时间。

数学定义如下：

$$a^{(0)} = [e_1, e_2, \cdots, e_m] \qquad (4)$$

其中 $a^{(0)}$ 表示Dense Embedding的输出， e_i 表示第 i 个Field对应的特征向量。

$$a^{(l+1)} = \sigma(W^{(l)}a^{(l)} + b^{(l)}) \qquad (5)$$

其中 $a^{(l)}, W^{(l)}, b^{(l)}$ 分别为第 l 层的输出、参数以及偏置项，σ 为激活函数。则，DNN部分的输出可以表示为：

$$y_{DNN} = \sigma(W^{(H)}a^{(H)} + b^{(H)}) \qquad (6)$$

H 表示隐藏层层数。

1.3 小结

至此，我们已经将两个模块分析完毕（真的很简单，大道至简），将公式（3）与公式（6）简单合并，得到最初的公式（1）定义：

$$\hat{y} = sigmoid(y_{FM} + y_{DNN}) \qquad (1)$$

在实践部分，最终是将FM模块与DNN模块进行简单concat，通过投影层进行融合[6]。

从上述分析来看，本质上DeepFM是显式的针对特征各种组合建模：一阶特征与二阶交叉特征（FM部分）、高阶特征（DNN部分），最终将低阶到高阶的所有特征以并行的方式连接到一起。之前的模型或多或少都没有这么完备，三者至少缺其一。

列举几种模型进行完备性对比，结果如下所示。FNN模型与PNN模型将重心放在提取高阶特征信息，PNN中Product Layer精心构建低阶交叉特征信息（小于等于2阶），但是仅作为后续DNN的输入，并未将低阶特征与高阶特征并行连接。并且FNN需要进行参数预训练，模型构建时间开销较多。Wide&Deep模型将低阶与高阶特征同时建模，但是在Wide侧通常需要更多的特征工程工作。所以，整体对比下来DeepFM的完备性更高。

Table 1: Comparison of deep models for CTR prediction

	No Pre-training	High-order Features	Low-order Features	No Feature Engineering
FNN	×	√	×	√
PNN	√	√	×	√
Wide & Deep	√	√	√	×
DeepFM	√	√	√	√

SOTA Lab

2. 实验对比

作者使用了两份数据集进行实验对比，分别为 *Criteo* 数据集与华为内部数据集。实验对比模型分别为：LR、FM、FNN [4]、PNN（三个变种模型）[5]、Wide&Deep（LR+DNN与FM+DNN两种模型）、DeepFM。

2.1 计算效率

为了对比各个模型的计算效率，作者选择 *Criteo* 数据集，以LR模型训练时间作为baseline，通过以下公式量化其他模型的训练时间，对比结果显示DeepFM是几个模型中训练效率最高的。

$$\frac{|training \quad time \quad of \quad deep \quad CTR \quad model|}{|training \quad time \quad of \quad LR|} \tag{7}$$

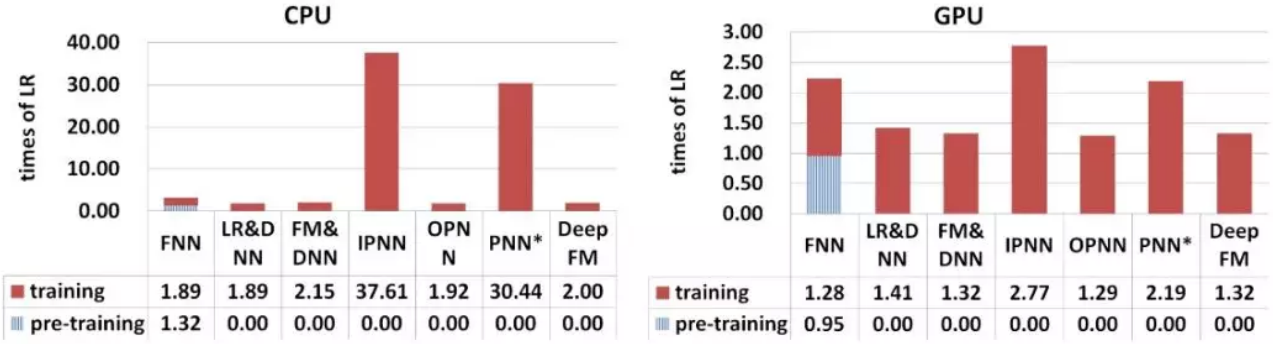


Figure 6: Time comparison.

SOTA Lab

2.2 性能对比

各模型对比结果如下，DeepFM效果最佳。对比模型中，FNN与PNN变种模型都属于仅对高阶特征进行建模，缺少低阶特征部分，而DeepFM与其对比之后仍能获胜，说明同时建模低阶与高阶交叉特征的必要性。而通过对比LR&DNN、FM&DNN与DeepFM模型，则说明DeepFM共享输入层这种做法的优越性。

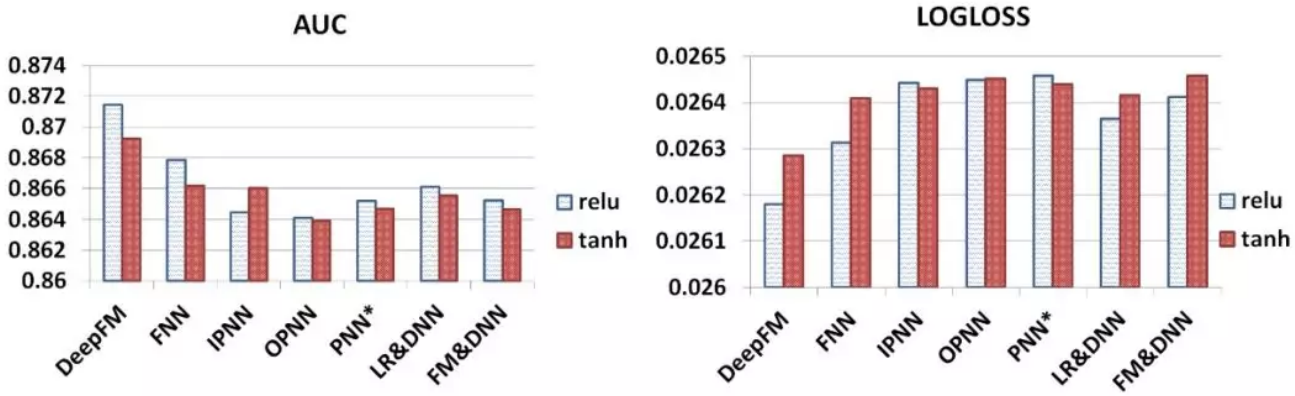
Table 2: Performance on CTR prediction.

	Company*		Criteo	
	AUC	LogLoss	AUC	LogLoss
LR	0.8640	0.02648	0.7686	0.47762
FM	0.8678	0.02633	0.7892	0.46077
FNN	0.8683	0.02629	0.7963	0.45738
IPNN	0.8664	0.02637	0.7972	0.45323
OPNN	0.8658	0.02641	0.7982	0.45256
PNN*	0.8672	0.02636	0.7987	0.45214
LR & DNN	0.8673	0.02634	0.7981	0.46772
FM & DNN	0.8661	0.02640	0.7850	0.45382
DeepFM	**0.8715**	**0.02618**	**0.8007**	**0.45083**

SOTA Lab

2.3 超参数研究

1）激活函数影响



SOTA Lab

Figure 7: AUC and Logloss comparison of activation functions.

对比结果说明，对于大多数深度模型而言relu激活函数更合适。

2）dropout比例影响

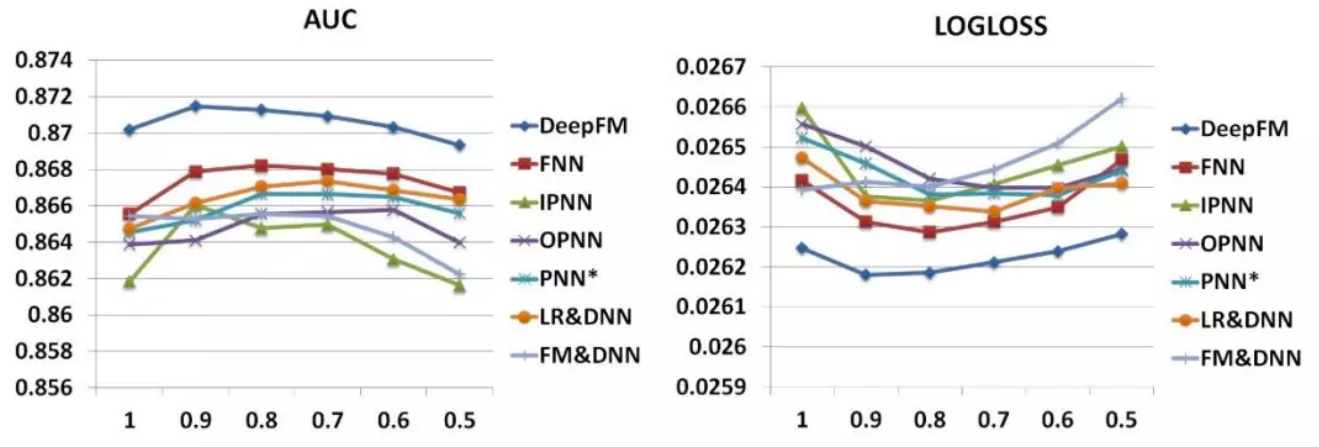


Figure 8: AUC and Logloss comparison of dropout.

3）隐藏层节点数

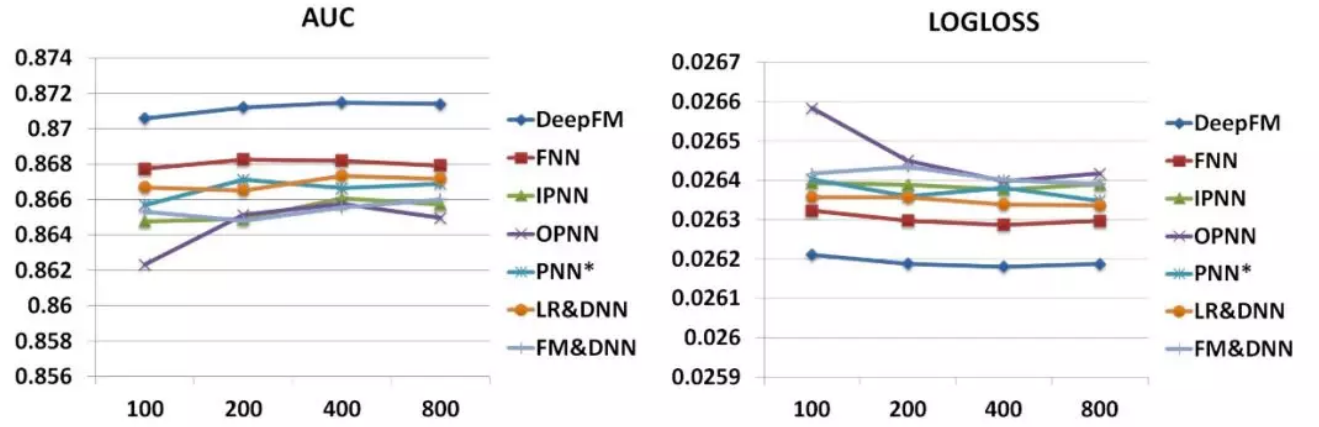


Figure 9: AUC and Logloss comparison of number of neurons.

调节每个隐藏层的节点数会对模型表现产生影响。

4）隐藏层层数

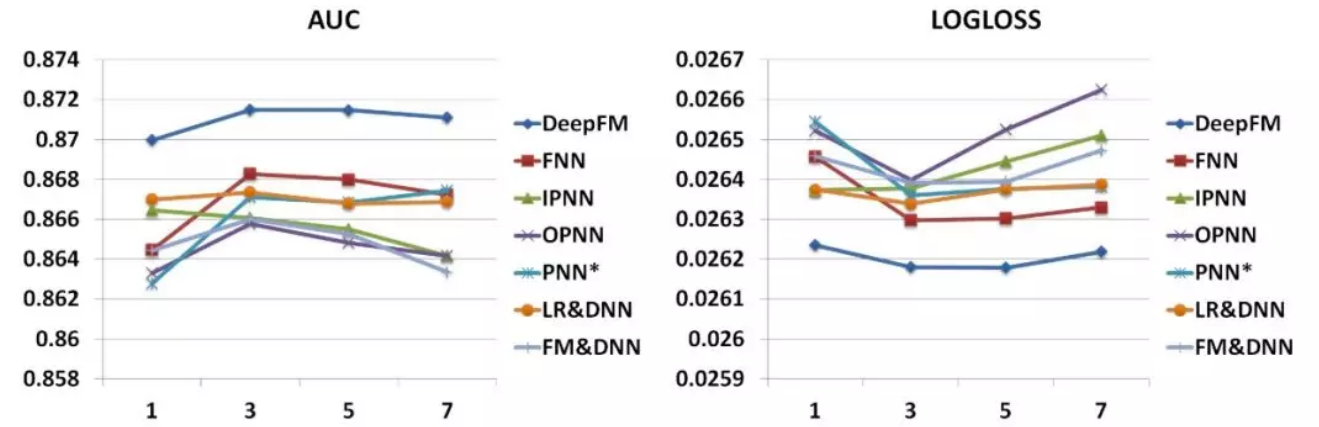


Figure 10: AUC and Logloss comparison of number of layers.

实验表明，隐藏层层数为3时，各模型效果表现最佳。

5）DNN结构

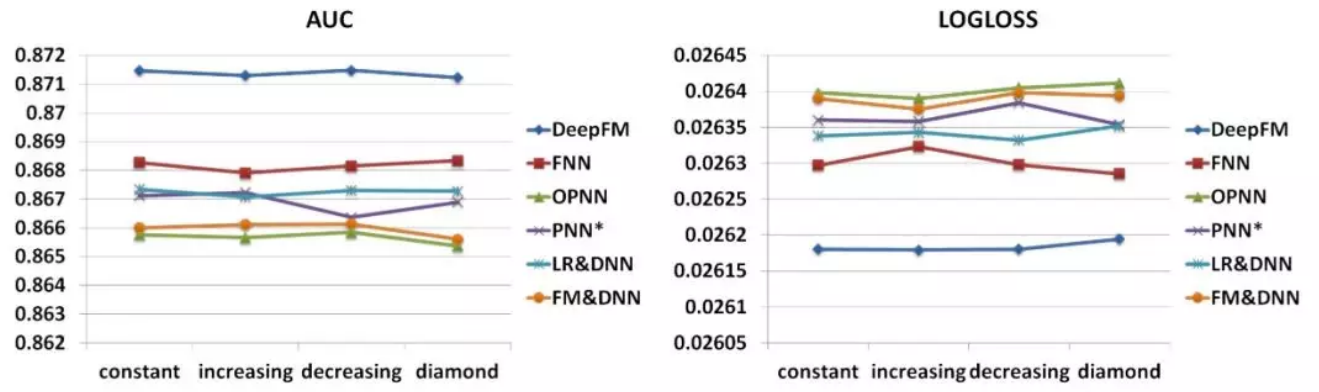


Figure 11: AUC and Logloss comparison of network shape.

几种结构对比，constant结构表现最佳。

实践

仍然使用 $MovieLens100K$ 作为数据集，核心代码如下。为了方便模型构建，模型输入与以往有所不同，具体请参阅代码逻辑。

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

`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 DeepFM(object):
    def __init__(self, vec_dim=None, field_lens=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.feat_num = np.sum(field_lens)
        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') # (b
        self.x = tf.placeholder(tf.float32, shape=[None, self.field_num], name='feat_value') # (bat
        self.y = tf.placeholder(tf.float32, shape=[None], name='input_y')
        self.is_train = tf.placeholder(tf.bool)

    def inference(self):
        with tf.variable_scope('first_order_part'):
            first_ord_w = tf.get_variable(name='first_ord_w', shape=[self.feat_num, 1], dtype=tf.fl
            first_order = tf.nn.embedding_lookup(first_ord_w, self.index) # (batch, F, 1)
            first_order = tf.reduce_sum(tf.multiply(first_order, tf.expand_dims(self.x, axis=2)), a
        with tf.variable_scope('emb_part'):
            embed_matrix = tf.get_variable(name='second_ord_v', shape=[self.feat_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)
        with tf.variable_scope('second_order_part'):
            sum_emb_square = tf.square(tf.reduce_sum(embed_x, axis=1)) # (batch, K)
            square_emb_sum = tf.reduce_sum(tf.square(embed_x), axis=1) # (batch, K)
            second_order = 0.5 * (sum_emb_square - square_emb_sum)
            fm = tf.concat([first_order, second_order], axis=1) # (batch, F+K)
        with tf.variable_scope('dnn_part'):
            embed_x = tf.layers.dropout(embed_x, rate=self.dropout_rate, training=self.is_train) #
            in_num = self.field_num * self.vec_dim
            dnn = tf.reshape(embed_x, shape=(-1, in_num)) # (batch, in_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 += self.field_num + self.vec_dim
            output = tf.concat([fm, dnn], axis=1)
            proj_w = tf.get_variable(name='proj_w', shape=[in_num, 1], dtype=tf.float32)
            proj_b = tf.get_variable(name='proj_b', shape=[1], dtype=tf.float32)
            self.y_logits = tf.matmul(output, proj_w) + proj_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_hat
```

```
self.train_op = tf.train.AdamOptimizer(self.lr).minimize(self.loss)
```

reference

[1] Guo, Huifeng, et al. "DeepFM: a factorization-machine based neural network for CTR prediction." *arXiv preprint arXiv:1703.04247* (2017).

[2] Rendle, S. (2010, December). Factorization machines. In 2010 IEEE International Conference on Data Mining (pp. 995-1000). IEEE.

[3] Cheng, Heng-Tze, et al. "Wide & deep learning for recommender systems." *Proceedings of the 1st workshop on deep learning for recommender systems*. ACM, 2016.

[4] Zhang, Weinan, Tianming Du, and Jun Wang. "Deep learning over multi-field categorical data." *European conference on information retrieval*. Springer, Cham, 2016.

[5] Qu, Yanru, et al. "Product-based neural networks for user response prediction." *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 2016.

[6]https://github.com/ChenglongChen/tensorflow-DeepFM/blob/master/DeepFM.py

专注知识分享，欢迎关注 SOTA Lab~

如果这篇文章对你有帮助，可以点击下方“在看”，推荐给更多小伙伴 ✌