基于深度学习的推荐(六)：CTR预估经典模型NFM

原创 如雨星空 推荐算法工程师 2019-10-25

前言

早期做特征工程的时候,采用人工或决策树等来选择特征,然而这些方法无法学习到训练集中没有出现的特征组合.而近几年出现的基于embedding的方法,可以学习到训练集中没有出现的组合,**作者将embedding方法归为两类,一类是FM这种线性模型,之前介绍过的FNN就是利用FM作为初始化的embedding;另一类是基于神经网络的非线性模型**.

NFM(Neural Factorization Machine)则是将两种embedding结合起来.NFM是发表在SIGIR 2017上的文章,**出现在深度学习与推荐系统结合的初期,模型相对较为简单,可以拿来练习tensorflow**.

论文地址:https://arxiv.org/pdf/1708.05027.pdf

NFM模型

首先来回顾下FM模型:

$$\hat{y}(\mathbf{x}) := w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j$$

设embedding向量维度为k,其中的**二阶交叉项可以进行优化**

$$
\begin{aligned}
&\sum_{i=1}^{n}\sum_{j=i+1}^{n}\langle \mathbf{v}_i, \mathbf{v}_j\rangle\, x_i\, x_j \\
=&\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\langle \mathbf{v}_i, \mathbf{v}_j\rangle\, x_i\, x_j - \frac{1}{2}\sum_{i=1}^{n}\langle \mathbf{v}_i, \mathbf{v}_i\rangle\, x_i\, x_i \\
=&\frac{1}{2}\left(\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{f=1}^{k} v_{i,f}\, v_{j,f}\, x_i\, x_j - \sum_{i=1}^{n}\sum_{f=1}^{k} v_{i,f}\, v_{i,f}\, x_i\, x_i\right) \\
=&\frac{1}{2}\sum_{f=1}^{k}\left(\left(\sum_{i=1}^{n} v_{i,f}\, x_i\right)\left(\sum_{j=1}^{n} v_{j,f}\, x_j\right) - \sum_{i=1}^{n} v_{i,f}^2\, x_i^2\right) \\
=&\frac{1}{2}\sum_{f=1}^{k}\left(\left(\sum_{i=1}^{n} v_{i,f}\, x_i\right)^2 - \sum_{i=1}^{n} v_{i,f}^2\, x_i^2\right)
\end{aligned}
$$

交叉项得到的是**一个值**,如果去掉最外面那层求和,得到**一个k维的向量**.这个k维的向量就是所谓的"Bi-Interaction Layer"的结果:

$$
f_{BI}(\mathcal{V}_x) = \frac{1}{2}\left[(\sum_{i=1}^{n} x_i\mathbf{v}_i)^2 - \sum_{i=1}^{n}(x_i\mathbf{v}_i)^2\right],
$$

将这个向量输入全连接层,得到预测结果**f(x)**,而最终的预估公式就是:

$$
\hat{y}_{NFM}(\mathbf{x}) = w_0 + \sum_{i=1}^{n} w_i x_i + f(\mathbf{x}),
$$

此时再看模型一目了然:

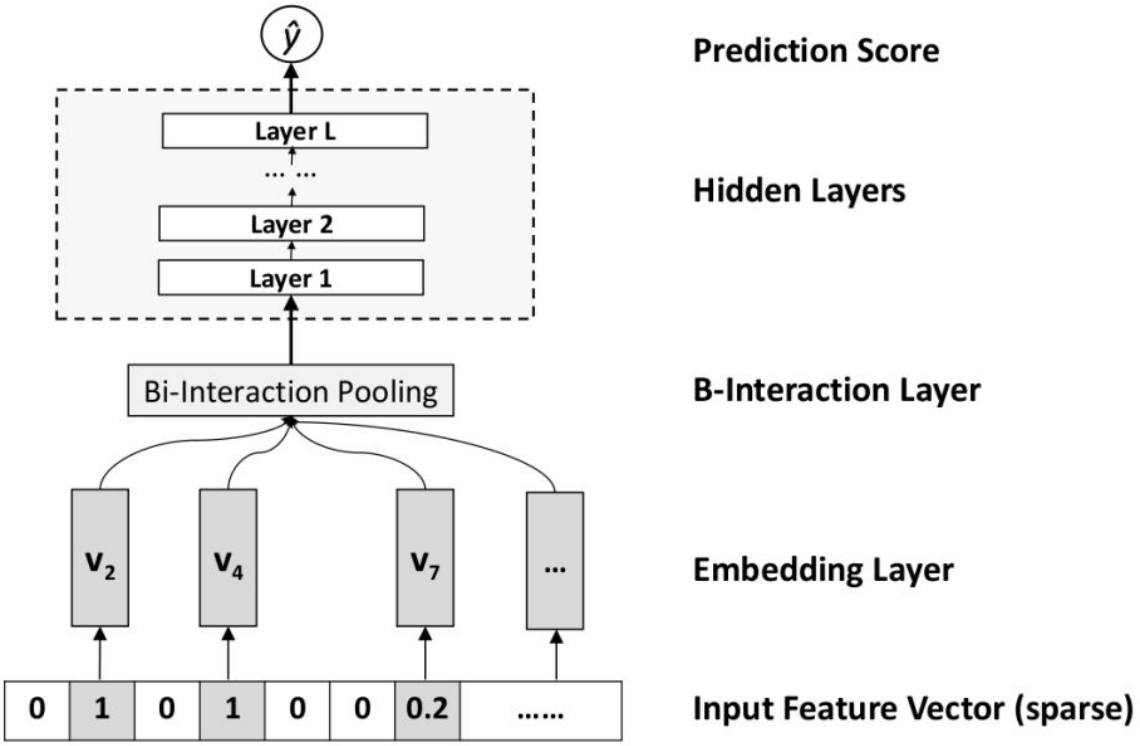

Figure 2: Neural Factorization Machines model (the first-order linear regression part is not shown for clarity).

代码实战

这部分代码改自之前AFM的代码,有兴趣可以自己改改试一试,挺简单的.其中interaction layer的实现提供了优化前和优化后两种写法,可以运行下比较比较时间,差距蛮大.简单看看几个关键点的实现,首先是embedding layer:

```
with tf.name_scope('Embedding_Layer'):
    self.embeddings = tf.nn.embedding_lookup(self.weights['feature_embeddings'], self.fe
    feat_value = tf.reshape(self.feature_value, shape=[-1, self.field_size, 1])  # [None
    self.embeddings = tf.multiply(self.embeddings, feat_value)  # [None, field_size, emb
```

然后是预测公式的线性部分:

```
with tf.name_scope('linear_part'):
    self.linear_part = tf.nn.embedding_lookup(self.weights['linear_w'],self.feature_inde
    self.linear_part = tf.reduce_sum(tf.multiply(self.linear_part, feat_value), axis=2)
    self.linear_part = tf.nn.dropout(self.linear_part, self.dropout_keep_fm[0])  # [None
    self.linear_out = tf.reduce_sum(self.linear_part, axis=1, keep_dims=True) # [None, 1
    self.w0 = tf.multiply(self.biases['w0'], tf.ones_like(self.linear_out)) # [None, 1]
```

B-Interaction Layer这一层的实现比较简单:

```
with tf.variable_scope('interaction_layer'):
    self.sum_square_emb = tf.square(tf.reduce_sum(self.embeddings, axis=1)) # [None, emb
    self.square_sum_emb = tf.reduce_sum(tf.square(self.embeddings), axis=1) # [None, emb
    self.fully_out = 0.5 * tf.subtract(self.sum_square_emb, self.square_sum_emb) # [None
```

然后就是后面的全连接层和最后的预测结果:

```
with tf.name_scope('fully_layer'):
    for i in range(len(self.deep_layers)):
        self.fully_out = tf.add(tf.matmul(self.fully_out, self.weights[i]), self.biases[
        self.fully_out = self.deep_layers_activation(self.fully_out)
        if(self.batch_norm):
            self.fully_out = self.batch_norm_layer(self.fully_out, self.train_phase)
        self.fully_out = tf.nn.dropout(self.fully_out, keep_prob=self.dropout_fm[i])

with tf.name_scope('out'):
    self.out = tf.add_n([self.w0, self.linear_out, self.fully_out]) # # yAFM = w0 + wx +
```

参考代码:

https://github.com/wyl6/Recommender-Systems-Samples/tree/master/RecSys%20And%20Deep%20Learning/Attention/AFM

https://github.com/faychu/nfm/blob/master/NeuralFM.py

完整数据和代码:

https://github.com/wyl6/Recommender-Systems-Samples/tree/master/RecSys%20And%20Deep%20Learning/DNN/nfm

参考

https://arxiv.org/pdf/1708.05027.pdf