计算广告CTR预估系列(二)--DeepFM实践

原创 李宁宁 机器学习荐货情报局 2018-05-09

计算广告CTR预估系列(二)--DeepFM实践

> 本文是计算广告CTR预估系列的第二篇文章，上一篇计算广告CTR预估系列(一)—DeepFM理论侧重理论，本文侧重实践。两者一起食用，疗效最好！
> 认证阅读完这两篇DeepFM的文章，对该模型应该有比较深入的理解了。
> 预告：计算广告CTR预估系列(三)—FFM理论与实践。敬请期待...

0. 变量说明

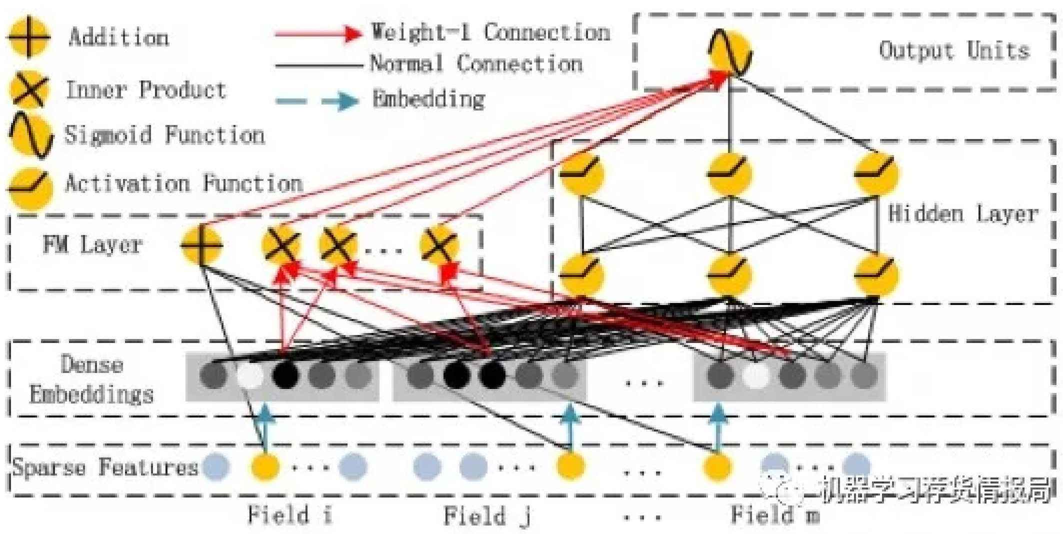
> 为了方便后面的阅读，我们先说明一下各个名称的含义：
>
> 1. field_size: 输入X在进行one-hot之前的特征维度

2. feature_size: 输入X在one-hot之后的特征维度， 又记作 n
3. embedding_size: one-hot后的输入，进行嵌入后的维度，又记作 k
4. 代码中tf中维度None表示任意维度，我们用来表示输入样本的数量这一维度。

1. 架构图与公式

1.1 架构图

让我们来先回顾一下架构图和公式：
架构图包含两部分：**FM Component**和**Deep Component**。
其中FM部分用于对1维特征和2维组合特征进行建模；Deep部分用于对高维组合特征进行建模。



1.2 公式

1.2.1 公式参考

DeepFM最后输出公式为：

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

其中FM部分贡献为：

$$y_{FM} = \langle w, x \rangle + \sum_{j_1=1}^{d} \sum_{j_2=j_1+1}^{d} \langle V_i, V_j \rangle x_{j_1} \cdot x_{j_2}, \quad (2)$$

需要注意的点：

1. 这里省去了原始FM中的常数项，为了方便。
2. 这里的x可以认为是一个样本，d是one-hot之后的总维度，也就是feature_size. 在原始FM论文中对应n参数。

Deep部分贡献为：

这个很好理解，就是神经网络的输出而已，就不再给公式了。

> FM论文中原始公式，方便对比参考：
>
> $$\hat{y}(\mathbf{x}) := w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j$$
>
>

二阶项的化简结果：

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

1.2.2 FM Component维度问题

> 先说结论：
> FM-1维：field_size

> FM-2维组合特征： embedding_size
> 这个和上面的公式有点不一样，但是实际代码实现是这样实现的。

这里比较**重要**：因为关系到代码里面怎么写，下面分别解释：

> 先看FM-1次项：

首先，我们看到一次项是**求和**了的。但是在实际代码中，**我们并没有对一次项求和。**而是输出一个维度为**k的向量**，这里的k就是embedding_size。

为什么要这么做那？我个人的感觉是：**提高最后模型的学习效果。**
最后神经网络的输出其实可以看做是**logistic regression**。它的输入由三部分组成：**FM-1维，FM-二维，Deep部分**。 自然而然，我们不希望FM-1维只是一个标量，一个数吧。这显然不利于LR模型的学习，那么如果用原始的维度那：**公式里是Wi * Xi**，Xi的维度是n，n是one-hot之后的维度，这个维度太大了以至于我们才想出了各种办法来解决这个问题，用n显然不行。所以，**就用field_size**。从逻辑上来说，也是对one-hot之前每个特征维度的一种建模。

> 再看FM-2维组合特征部分：

在上一小节中，我们看到了FM二阶项的化简结果。**最外层是在Embeeding_size上的求和。不做这个求和，而是得到一个embedding_size维度的向量**，就是送到最后输出单元的FM的二阶部分。
原因我想跟上面FM-1维的是一样的，求和之后就变成了一个标量，显然不利于后面的学习。

这两部分理解了之后，让我们来看下代码吧！

2. 核心代码拆解

2.1 输入

```
 feat_index = tf.placeholder(dtype=tf.int32, shape=[None, config.field_size], 1
feat_value = tf.placeholder(dtype=tf.float32, shape=[None, None], name='feat_va
label = tf.placeholder(dtype=tf.float16, shape=[None,1], name='label')
```

注意下各个输入变量的维度大小就可以了，没什么特别需要说明的。None在tf里面表示任意维度，此处表示样本数量维度。

2.2 Embedding

```
 # Sparse Features -> Dense Embedding
embeddings_origin = tf.nn.embedding_lookup(weights['feature_embedding'], ids=fe
```

重点来了，这里是完成 `Sparse Features` 到 `Dense Embedding` 的转换。

2.3 FM Component - 1维特征

1维特征本来是求和得到一个标量的，为了提高学习效果，我们改为**不求和，得到一个field_size维度的向量。相当于是进行了一次k=1的一次embedding。**

```
 y_first_order = tf.nn.embedding_lookup(weights['feature_bias'], ids=feat_index
```

得到W之后，和输入X相乘，并缩减维度，得到最终FM-1维的输出：

```
 w_mul_x = tf.multiply(y_first_order, feat_value_reshape) # [None, field_size,
y_first_order = tf.reduce_sum(input_tensor=w_mul_x, axis=2) # [None, field_size
```

2.4 FM Component - 2维组合特征

在第一节中，我们给出了FM-2维组合特征的化简公式。发现计算的两部分都需要计算 **vi * xi**。所以我们先把这一部分计算出来：

```
 feat_value_reshape = tf.reshape(tensor=feat_value, shape=[-1, config.field_siz
embeddings = tf.multiply(embeddings_origin, feat_value_reshape) # [None, field_
```

然后分别计算这两部分：

$$\left(\sum_{i=1}^{n} v_{i,f}\, x_i\right)^2$$

```
 // sum_square part 先sum，再square
summed_features_emb = tf.reduce_sum(input_tensor=embeddings, axis=1) # [None, e
summed_features_emb_square = tf.square(summed_features_emb)
```

$$\sum_{i=1}^{n} v_{i,f}^2 x_i^2$$

```
// square_sum part
squared_features_emb = tf.square(embeddings)
squared_features_emb_summed = tf.reduce_sum(input_tensor=squared_features_emb,
```

最后得到最终FM-2维组合特征输出结果，维度为embedding_size:

```
// second order
y_second_order = 0.5 * tf.subtract(summed_features_emb_square, squared_features_
```

2.5 Deep Component

网络部分比较简单，只要一层一层的前向传递就可以了。只有一个问题需要说明：
网络部分，第一个隐藏层的输入是什么？
我的感觉应该是原始输入嵌入后的结果：

```
// Deep Component
y_deep = tf.reshape(embeddings_origin, shape=[-1, config.field_size * config.em

for i in range(0, len(deep_layers)):
    y_deep = tf.add(tf.matmul(y_deep, weights['layer_%d' % i]), weights['bias_
    y_deep = config.deep_layers_activation(y_deep)
```

2.6 输出

把前面FM Component和Deep Component的两部分结合起来，就得到了最后输出单元的输入，经过sigmoid函数激活就可以得到最终结果了。

```
// output
concat_input = tf.concat([y_first_order, y_second_order, y_deep], axis=1)
out = tf.add(tf.matmul(concat_input, weights['concat_projection']), weights['co
out = tf.nn.sigmoid(out)
```

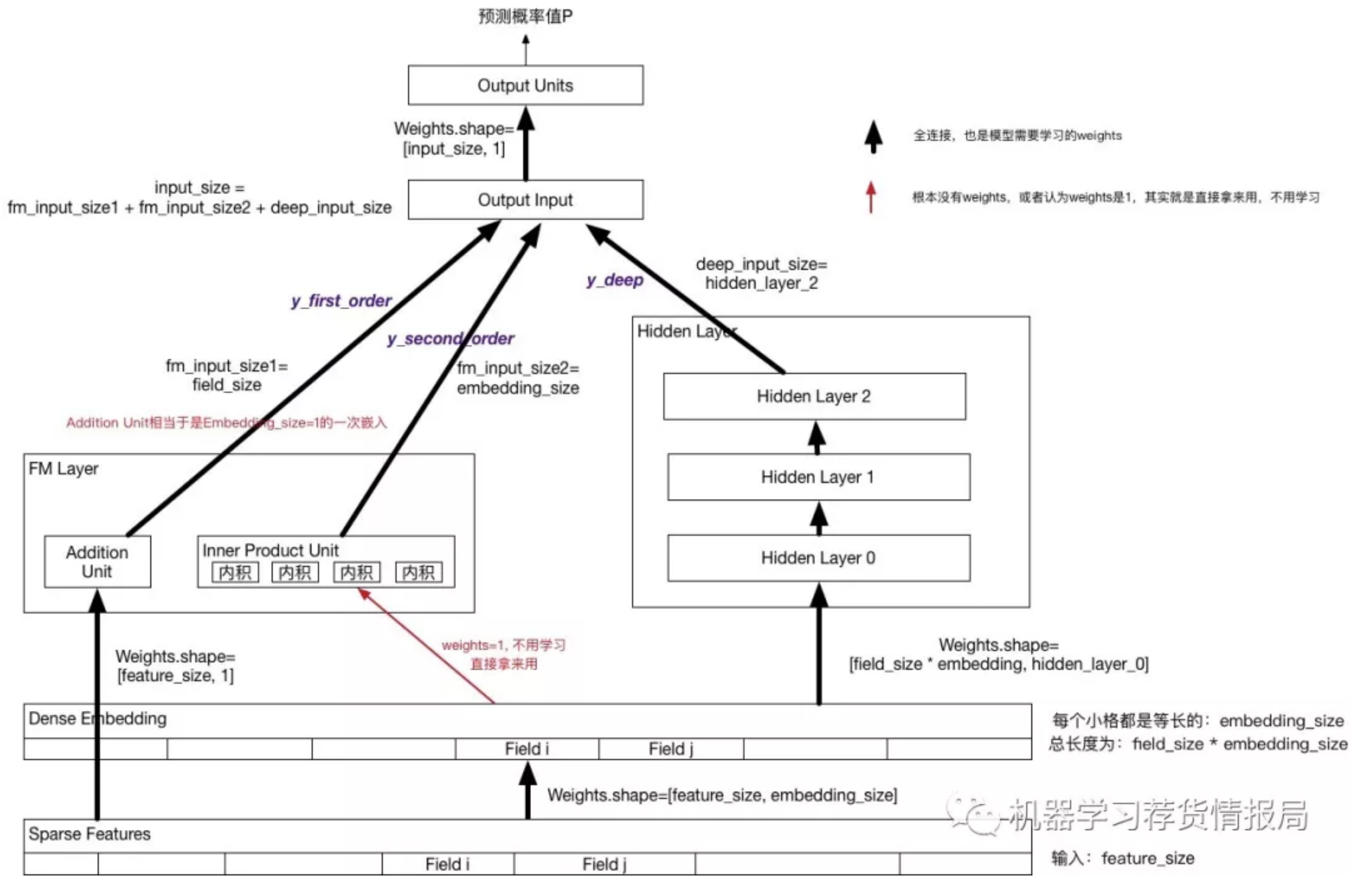
3. 完整代码：

> 这份代码最主要的目的是学习，直接应用于工程的话还需要做一些优化。比如batch normalization、stack train、batch train以及代码重构等。
> 本代码可以帮助你快速实验，实现DeepFM，掌握其原理。
>
> 另外，可直接运行。

本来是想给出完整代码的，但是贴出来发现太占篇幅，而且手机阅读起来也不是很方便。所以，给出地址好了：
github地址—欢迎follow/star/contribute

另外，附上一份整理的 DeepFM架构图-实现篇 的图。主要是帮助大家理解**各个参数的维度、权重weights的维度、需要学习的维度、FM Deep两部分输出的维度**。 要想实现DeepFM，**只要把每一部分的维度搞清楚，网络的架构搞清楚就问题不大了。**

图片看不清的话，去上面github的地址里面取吧。



Reference

1. 计算广告CTR预估系列(一)—DeepFM理论
2. DeepFM—个比较好的实现

获取更多机器学习干货、荐货，欢迎关注**机器学习荐货情报局**，加入荐货大家庭！