基于Tensorflow实现FFM

slade_sal LeadAI OpenLab 2018-08-14

正文共3084张图，16张图，预计阅读时间15分钟。
github：https://github.com/sladesha/deep_learning

没错，这次登场的是FFM。各大比赛中的“种子”算法，中国台湾大学Yu-Chin Juan荣誉出品，美团技术团队背书，Michael Jahrer的论文的field概念灵魂升华，土豪公司鉴别神器。通过引入field的概念，FFM把相同性质的特征归于同一个field，相当于把FM中已经细分的feature再次拆分，可不可怕，厉不厉害？好，让我们来看看怎么一个厉害法。

FMM理论

特征交互

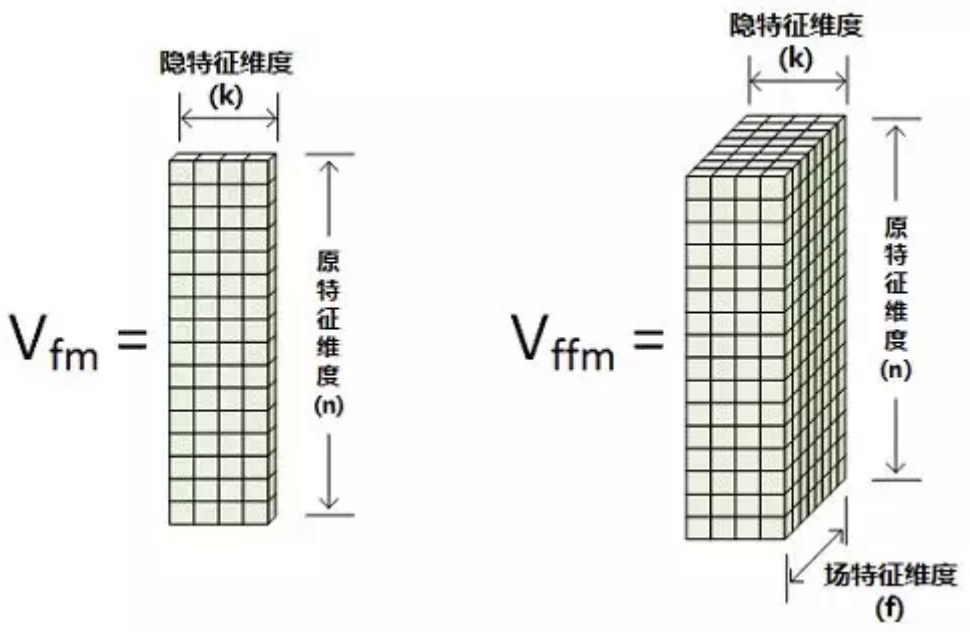
网上已经说烂了的美团技术团队给出的那张图：

Clicked?	Country	Day	Ad_type
1	USA	26/11/15	Movie
0	China	1/7/14	Game
1	China	19/2/15	Game

针对Country这个变量，

FM的做法是one-hot-encoding，生成country_USA，country_China两个稀疏的变量，再进行embedding向量化。

FFM的做法是cross-one-hot-encoding，生成country_USA_Day_26/11/15，country_USA_Ad_type_Movie...M个变量，再进行embedding向量化。



就和上图一样，fm做出来的latent factor是二维的，就是给每个特征一个embedding结果；而暴力的ffm做出的latent factor是三维的，出来给特征embedding还考虑不同维度特征给不同的embedding结果，也是FFM中“field-aware”的由来。

$$y(\mathbf{x}) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_{i,f_j}, \mathbf{v}_{j,f_i} \rangle x_i x_j$$

同时从公式中看，对于xi这个特征为什么embedding的latent factor向量的V是Vifj，其实就是因为xi乘以的是xj，所以latent factor向量的信息提取才是field j，也就是fj。多说一句，网上很多给出的现成的代码，这边都是写错了的，都写着写着变成了vifi，可能是写的顺手。

都说到这里了，我再多说一句，为什么说ffm是土豪公司鉴别神器呢？我们看下仅仅是二次项，ffm需要计算的参数有 nfk 个，远多于FM模型的 nk个，而且由于每次计算都依赖于乘以的xj的field，所以，无法用fm的那个计算技巧(ab = 1/2(a+b)2-a2-b^2)，所以计算复杂度是 O(kn^2)。这种情况下，没有GPU就不要想了，有GPU的特征多于50个，而且又很离散的，没有个三位数的GPU也算了。之前我看美团说他们在用，我想再去看看他们的实际用的过程的时候，发现文章被删了，真的是可惜，我其实一直也想知道如何减轻这个变态的计算量的方法。

给个实例给大家看下以上的这些的应用：

依旧来自美团技术研发团队中给出的案例，有用户数据如下：

User	Movie	Genre	Price
YuChin	3Idiots	Comedy, Drama	$9.99

这条记录可以编码成5个特征，其中“Genre=Comedy”和“Genre=Drama”属于同一个field，“Price”是数值型，不用One-Hot编码转换。为了方便说明FFM的样本格式，我们将所有的特征和对应的field映射成整数编号。

Field name	Field index	Feature name	Feature index
User	1	User=YuChin	1
Movie	2	Movie=3Idiots	2
Genre	3	Genre=Comedy	3
Price	4	Genre=Drama	4
		Price	5

红色部分对应的是field，来自于原始特征的个数；蓝色部分对应的是feature，来自于原始特征onehot之后的个数。

对于特征Feature:User=YuChin而言，有Movie=3Idiots、Genre=Comedy、Genre=Drama、Price四项要交互：

$$\langle \mathbf{v}_{1,2}, \mathbf{v}_{2,1} \rangle \cdot 1 \cdot 1 + \langle \mathbf{v}_{1,3}, \mathbf{v}_{3,1} \rangle \cdot 1 \cdot 1 + \langle \mathbf{v}_{1,3}, \mathbf{v}_{4,1} \rangle \cdot 1 \cdot 1 + \langle \mathbf{v}_{1,4}, \mathbf{v}_{5,1} \rangle \cdot 1 \cdot 9.99$$

User=YuChin与Movie=3Idiots交互是<V1,2,V2,1>·1·1，也就是第一项，为什么是V1,2呢？因为User=YuChin是Featureindex=1，而交互的Movie=3Idiots是Fieldindex=2，同理V2,1也是这样的，以此类推，那么，FFM的组合特征有10项，如下图所示：

$$\begin{aligned}\langle \mathbf{v}_{1,2}, \mathbf{v}_{2,1} \rangle \cdot 1 \cdot 1 + \langle \mathbf{v}_{1,3}, \mathbf{v}_{3,1} \rangle \cdot 1 \cdot 1 + \langle \mathbf{v}_{1,3}, \mathbf{v}_{4,1} \rangle \cdot 1 \cdot 1 + \langle \mathbf{v}_{1,4}, \mathbf{v}_{5,1} \rangle \cdot 1 \cdot 9.99 \\ + \langle \mathbf{v}_{2,3}, \mathbf{v}_{3,2} \rangle \cdot 1 \cdot 1 + \langle \mathbf{v}_{2,3}, \mathbf{v}_{4,2} \rangle \cdot 1 \cdot 1 + \langle \mathbf{v}_{2,4}, \mathbf{v}_{5,2} \rangle \cdot 1 \cdot 9.99 \\ + \langle \mathbf{v}_{3,3}, \mathbf{v}_{4,3} \rangle \cdot 1 \cdot 1 + \langle \mathbf{v}_{3,4}, \mathbf{v}_{5,3} \rangle \cdot 1 \cdot 9.99 \\ + \langle \mathbf{v}_{4,4}, \mathbf{v}_{5,3} \rangle \cdot 1 \cdot 9.99\end{aligned}$$

这就是一个案例的实际操作过程。

特征处理

为什么要把这个单拎出来说呢？我看了网上不少的对于特征的处理过程，版本实在是太多了，而且差异化也蛮大，这边就和大家一起梳理一下：

1.feature index * feature value
这个就是上面我这个实际案例的方式，对于分类变量采取onehot，对于连续变量之间进行值的点积，不做处理。优点是快速简单，不需要预处理，但是缺点也很明显，离群点影响，值的波动大等。

2.连续值离散化
这个方法借鉴了Cart里面对连续值的处理方式，就是把所有的连续值都当成一个分类变量处理。举例，现在有一个年龄age的连续变量[10,19,20,22,22,34]，这种方法就生成了age_10,age_19,age_20,age_22,age_34这些变量，如果连续值一多，这个方法带来的计算量就直线上升。

3.分箱下的连续值离散化
这种方法优化了第二种方法，举例解释，现在有一个年龄age的连续变量[10,19,20,22,22,34]，我们先建立一个map，[0,10):0,[10,20):1,[20,30):2,[30,100):3。原始的数据就变成了[1,1,2,2,2,3]，再进行2的连续值离散化方法，生成了age_1,age_2,age_3这几个变量，优化了计算量，而且使得结果更具有解释性。

损失函数

logistic loss

这个是官方指定的方法，是-1/1做二分类的时候常用的loss计算方法：

$$\min_{\mathbf{w}} \sum_{i=1}^{L} \log\left(1 + \exp\{-y_i \phi(\mathbf{w}, \mathbf{x}_i)\}\right) + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

这边需要注意的是，在做的时候，需要把label拆分成-1/1而不是0/1，当我们预测正确的时候，predlabel>0且越大正确的程度越高，相应的log项是越小的，整体loss越小；相反，如果我们预测的越离谱，predlabel<0且越小离谱的程度越高，相应的log项是越大的，整体loss越大。

交互熵

我看到很多人的实现依旧用了tf.nn.softmax_cross_entropy_with_logits，其实就是多分类中的损失函数，和大家平时的图像分类、商品推荐召回一模一样：

$$H_{y'}(y) = -\sum_i y'_i \log(y_i)$$

这边需要注意的是，在做的时候，需要把label拆分成[1,0]和[0,1]进行计算。不得不说，大家真的是为了省事很机智(丧心病狂)啊！

代码实现

我这边只给一些关键地方的代码，更多的去GitHub里面看吧。

embedding part

```
self.v = tf.get_variable('v', shape=[self.p, self.f, self.k], dtype='float32',initi
```

看到了，这边生成的v就是上面Vffm的形式。

inference part

```
for i in range(self.p):
        # 寻找没有match过的特征，也就是论文中的j = i+1开始
        for j in range(i + 1, self.p):
            print('i:%s,j:%s' % (i, j))
            # vifj
            vifj = self.v[i, self.feature2field[j]]
            # vjfi
            vjfi = self.v[j, self.feature2field[I]]
            # vi · vj
            vivj = tf.reduce_sum(tf.multiply(vifj, vjfi))
            # xi · xj
            xixj = tf.multiply(self.X[:, i], self.X[:, j])
            self.field_cross_interaction += tf.multiply(vivj, xixj)
```

我这边强行拆开了写，这样看起来更清晰一点，注意这边的vifj和vjfi的生成，这边也可以看到找对于的field的方法是用了filed这个字典，这就是为什么不能用fm的点击技巧。

loss part

```
# -1/1情况下的logistic loss
self.loss = tf.reduce_mean(tf.log(1 + tf.exp(-self.y * self.y_out)))
```

这边记得论文中的负号，如果有batch的情况下记得求个平均再进行bp过程。

论文结论

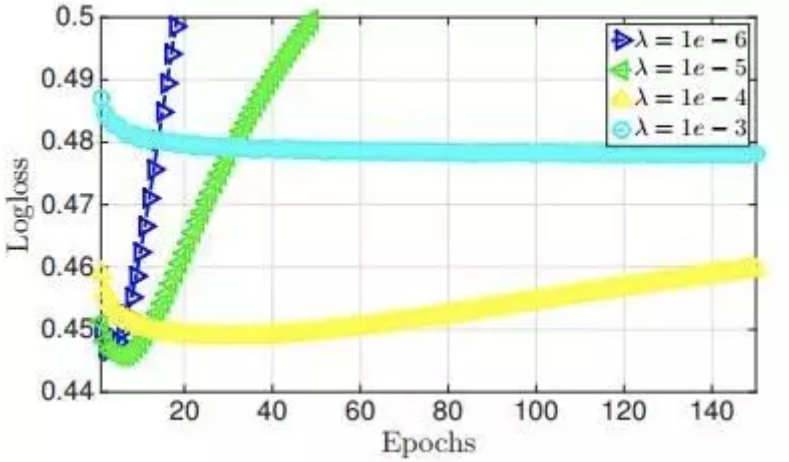
原始的ffm论文中给出了一些结论，我们在实际使用中值得参考：

- k值不用太大，没啥提升

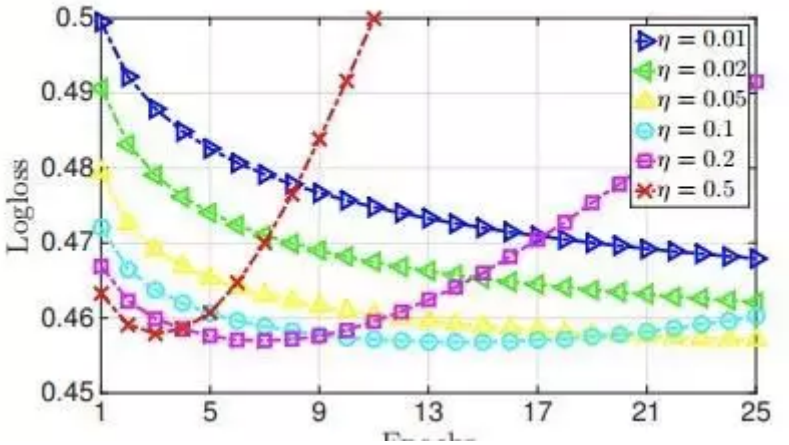
k	time	logloss
1	27.236	0.45773
2	26.384	0.45715
4	27.875	0.45696
8	40.331	0.45690
16	70.164	0.45725

(a) The average running time (in seconds) per epoch and the best logloss with different values of k. Because we use SSE instructions, the running time of $k = 1, 2, 4$ is roughly the same.

- 正则项lambda和学习率alpha需要着重调参



(b) The impact of λ



- epoch别太大，既会拖慢速度，而且造成过拟合；在原论文中甚至要考虑用early-stopping避免过拟合，所以epoch=1，常规的来讲就可以了，论文中提到的early-stopping操作：

```
1. Split the data set into a training set and a validation set.
2. At the end of each epoch, use the validation set to calcu-
late the loss.
3. If the loss goes up, record the number of epochs. Stop or
go to step 4.
4. If needed, use the full data set to re-train a model with
the number of epochs obtained in step 3.
```

总结

FFM是一个细化隐向量非常好的方法，虽然很简单，但还是有很多细节之处值得考虑，比如如何线上应用，如何可解释，如何求稀疏解等等。在部署实现FFM之前，我还是建议大家先上线FM，当效果真的走投无路的时候再考虑FFM，FFM在工业界的影响着实不如学术界那么强大，偷偷说一句，太慢了，真的是太慢了，慢死了，我宁可去用deepfm。

最后，给出代码实现的Github地址FFM，这边是我自己写的，理解理解算法可以，但是实际用的时候建议参考FFM的实现比较好的项目比如libffm，最近比较火的xlearn。