

推荐系统遇上深度学习(十八)--探秘阿里之深度兴趣网络(DIN)浅析及实现

原创 石晓文 小小挖掘机 2018-06-26

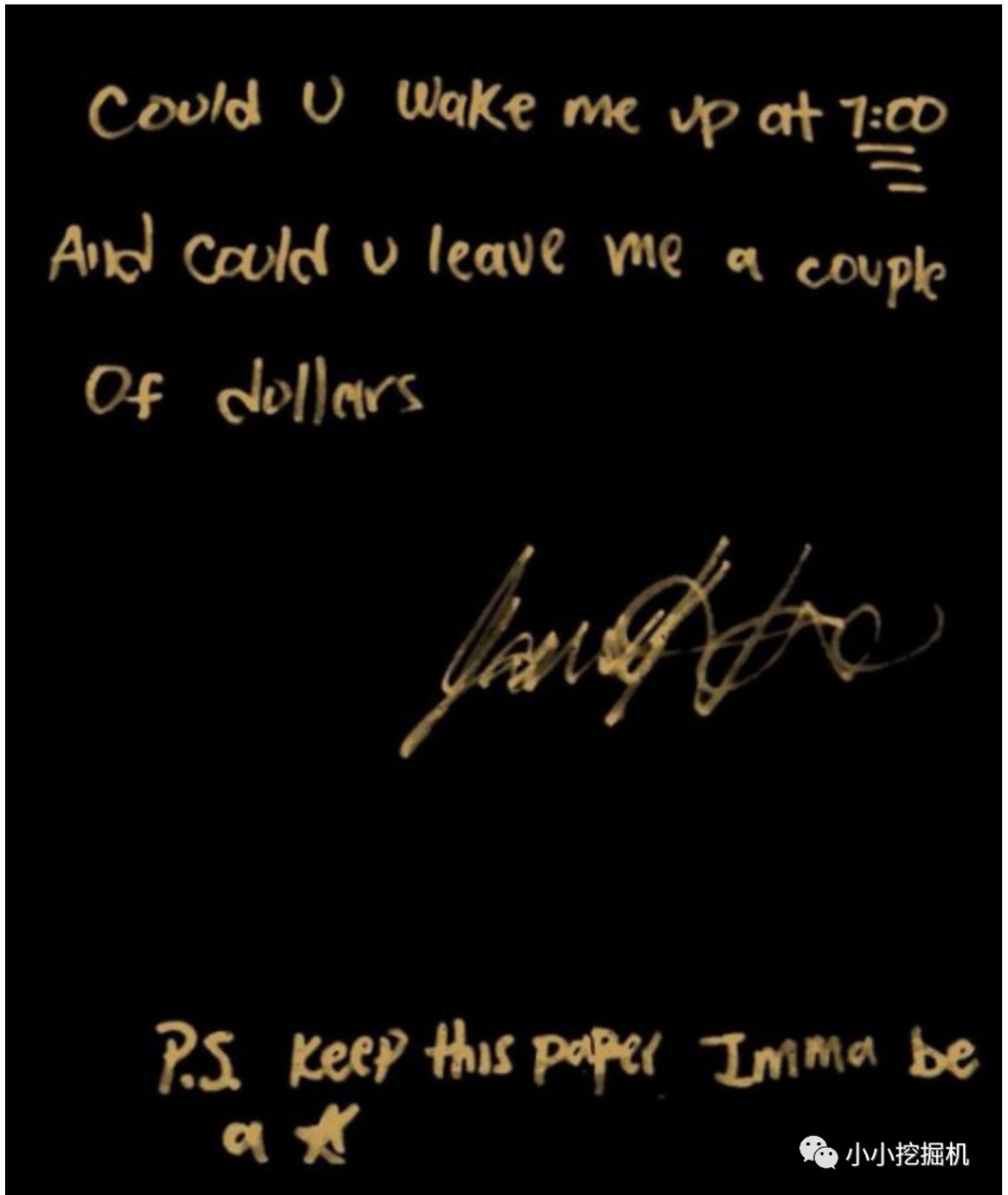
收录于话题

#推荐系统遇上深度学习

95个

今天的开头，首先恭喜登哥加冕mvp！实至名归！

如果有梦，那就去追！ --- James Harden



好了，回归正题！

阿里近几年公开的推荐领域算法有许多，既有传统领域的探索如MLR算法，还有深度学习领域的探索如entire -space multi-task model，Deep Interest Network等，同时跟清华大学合作展开了强化学习领域的探索，提出了MARDPG算法。

上一篇，我们介绍了MLR算法，通过分而治之的思想改进了传统的LR算法，使其能够拟合更复杂的线性关系。这一篇，我们来简单理解和实现一下阿里在去年提出的另一个重要的推荐系统模型-深度兴趣网络(DIN, Deep Interest Network). 该方法由盖坤大神领导的阿里妈妈的精准定向检索及基础算法团队提出，充分利用/挖掘用户历史行为数据中的信息来提高CTR预估的性能。

1、背景

深度学习在CTR预估领域已经有了广泛的应用，常见的算法比如Wide&Deep, DeepFM等。这些方法一般的思路是：通过Embedding层，将高维离散特征转换为固定长度的连续特征，然后通过多个全联接层，最后通过一个sigmoid函数转化为0-1值，代表点击的概率。即 **Sparse Features -> Embedding Vector -> MLPs -> Sigmoid -> Output**.

这种方法的优点在于：通过神经网络可以拟合高阶的非线性关系，同时减少了人工特征的工作量。

不过，阿里的研究者们通过观察收集到的线上数据，发现了用户行为数据中有两个很重要的特性：


Diversity：用户在浏览电商网站的过程中显示出的兴趣是十分多样性的。

Local activation：由于用户兴趣的多样性，只有部分历史数据会影响到当次推荐的物品是否被点击，而不是所有的历史记录。

这两种特性是密不可分的。

举个简单的例子，观察下面的表格：

Table 1: Examples of user behavior history from online product.

User	Behavior History	Candidate Ad
Young Mother	woolen coat, T-shirts, earrings, children's coat leather handbag, miniskirt, sports underwear	long sleeved jacket
Swimmer	bathing suit, kickboard, swimming cap, travel book tent, potato chips, nuts, potato chips, ice cream	 小小挖掘机

Diversity体现在年轻的母亲的历史记录中体现的兴趣十分广泛，涵盖羊毛衫、手提袋、耳环、童装、运动装等等。而爱好游泳的人同样兴趣广泛，历史记录涉及浴装、旅游手册、跳水板、马铃薯、冰激凌、坚果等等。

Local activation体现在，当我们给爱好游泳的人推荐goggle(护目镜)时，跟他之前是否购买过薯片、书籍、冰激凌的关系就不大了，而跟他游泳相关的历史记录如游泳帽的关系就比较密切。

针对上面提到的用户行为中存在的两种特性，阿里将其运用于自身的推荐系统中，推出了深度兴趣网络DIN，接下来，我们就一起来看一下模型的一些实现细节，然后我们会给出一个简化版的tensorflow实现。

2、模型设计

整体框架

我们先来看一下推荐系统的整体框架：



Figure 1: Overview of display advertising system. User behavior data plays a key role.

整个流程可以描述为：

- 1.检查用户历史行为数据
- 2.使用matching module产生候选ads。
- 3.通过ranking module做point-wise的排序，即得到每个候选ads的点击概率，并根据概率排序得到推荐列表。
- 4.记录下用户在当前展示广告下的反应(点击与否)，作为label。

特征设计

本文将所涉及到的特征分为四个部分：**用户特征、用户行为特征、广告特征、上下文特征**，具体如下：

Table 2: Feature Representations and Statistics in our display advertising system.

Feature Category	Feature Name	Dimemision	Type	#Nonzero Ids/Sample
User Profile Features	gender	2	one-hot	1
	age_level	~ 10	one-hot	1

User Behavior Features	visited good_ids	$\sim 10^9$	multi-hot	$\sim 10^3$
	visited shop_ids	$\sim 10^7$	multi-hot	$\sim 10^3$
	visited cate_ids	$\sim 10^4$	multi-hot	$\sim 10^2$

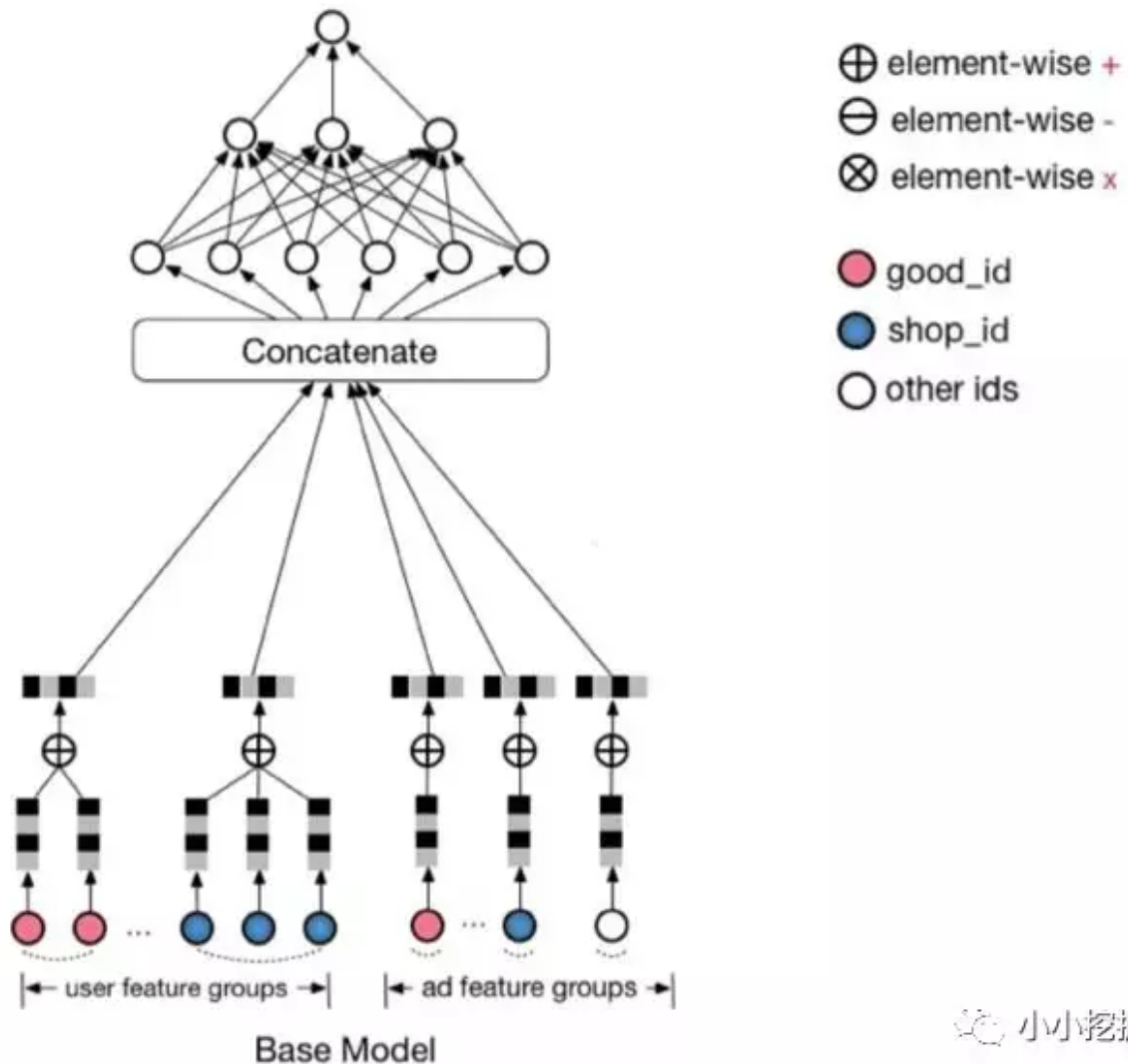
Ad Features	good_id	$\sim 10^7$	one-hot	1
	shop_id	$\sim 10^5$	one-hot	1
	cate_id	$\sim 10^4$	one-hot	1

Scene Features	pid	~ 10	one-hot	1
	time	~ 10	one-hot	1

其中，用户行为特征是multi-hot的，即多值离散特征。针对这种特征，由于每个涉及到的非0值个数是不一样的，常见的做法就是将id转换成embedding之后，加一层pooling层，比如average-pooling, sum-pooling, max-pooling。DIN中使用的是weighted-sum，其实就是加权的sum-pooling，权重经过一个activation unit计算得到。这里我们后面还会再介绍到。

BaseModel

在介绍DIN之前，我们先来看一下一个基准模型，结构如下：



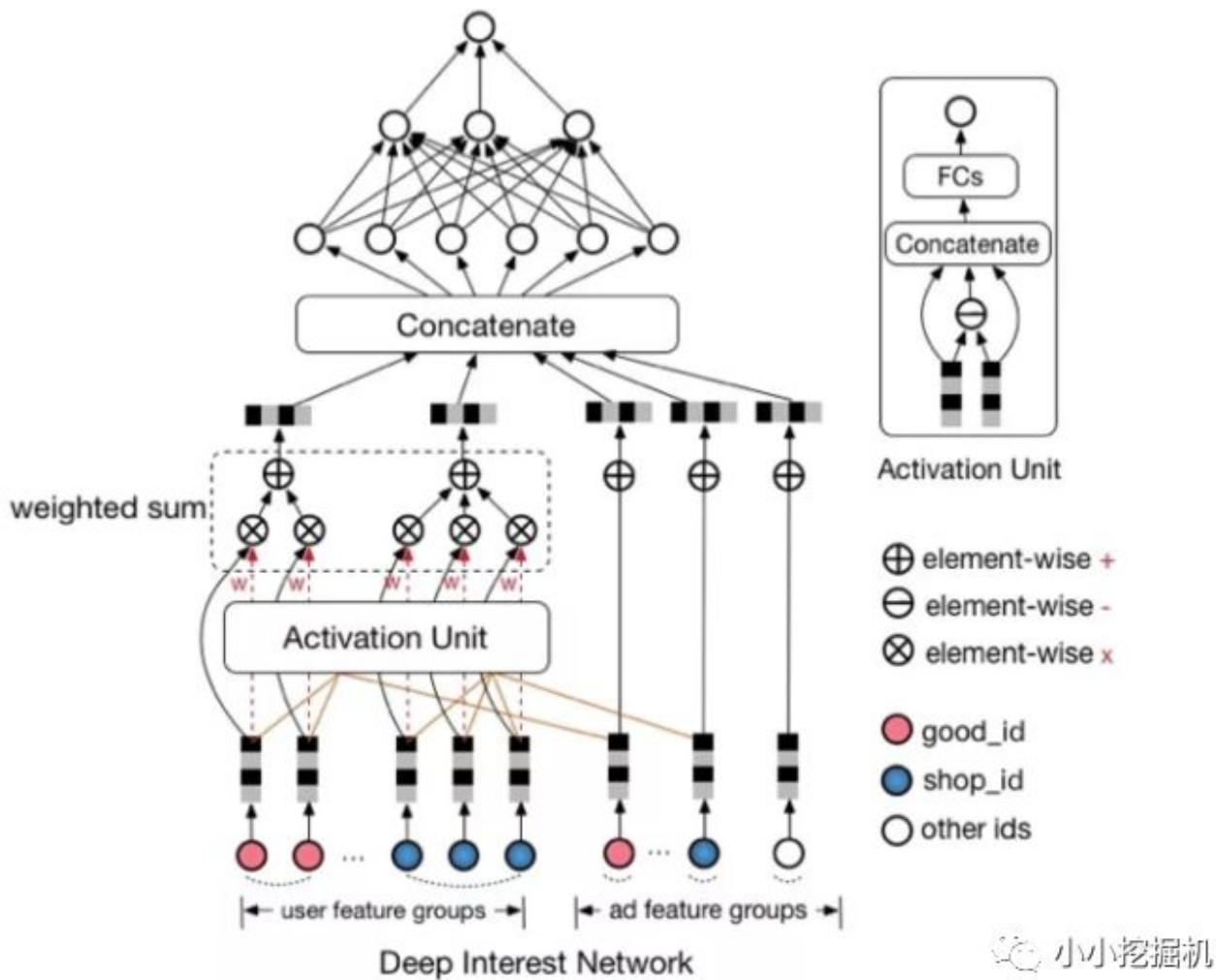
小小挖掘机

这里element-wise的意思其实就是元素级别的加减，同时，可不要忽略广播的存在哟。一个元素和一个向量相乘，也可以看作element-wise的，因为这个元素会广播成和向量一样的长度嘛，嘻嘻。

可以看到，Base Model首先吧one-hot或multi-hot特征转换为特定长度的embedding，作为模型的输入，然后经过一个DNN的part，得到最终的预估值。特别地，针对multi-hot的特征，做了一次element-wise+的操作，这里其实就是sum-pooling，这样，不管特征中有多少个非0值，经过转换之后的长度都是一样的！

Deep Interest Network

Base Model有一个很大的问题，它对用户的历史行为是同等对待的，没有做任何处理，这显然是不合理的。一个很显然的例子，离现在越近的行为，越能反映你当前的兴趣。因此，对用户历史行为基于Attention机制进行一个加权，阿里提出了深度兴趣网络（Deep Interest Network），先来看一下模型结构：



Attention机制简单的理解就是，针对不同的广告，用户历史行为与该广告的权重是不同的。假设用户有ABC三个历史行为，对于广告D，那么ABC的权重可能是0.8、0.1、0.1；对于广告E，那么ABC的权重可能是0.3、0.6、0.1。这里的权重，就是Attention机制即上图中的Activation Unit所需要学习的。

为什么要引入这一个机制呢？难道仅仅是通过观察历史数据拍脑袋决定的么？当然不是，如果不用Local activation的话，将会出现下面的情况：假设用户的兴趣的Embedding是 V_u ，候选广告的Embedding是 V_a ，用户兴趣和候选的广告的相关性可以写作 $F(U,A) = V_a * V_u$ 。如果没有Local activation机制的话，那么同一个用户对于不同的广告， V_u 都是相同的。举例来说，如果有两个广告A和B，用户兴趣和A，B的相似性都很高，那么在 V_a 和 V_b 连线上的广告都会有很高的相似性。这样的限制使得模型非常难学习到有效的用户和广告的embedding表示。

在加入Activation Unit之后，用户的兴趣表示计算如下：

$$V_u = f(V_a) = \sum_{i=1}^N w_i * V_i = \sum_{i=1}^N g(V_i, V_a) * V_i$$

小小挖掘机

其中， V_i 表示behavior id i的嵌入向量，比如good_id,shop_id等。 V_u 是所有behavior ids的加权和，表示的是用户兴趣； V_a 是候选广告的嵌入向量； w_i 是候选广告影响着每个behavior

id的权重，也就是Local Activation。wi通过Activation Unit计算得出，这一块用函数去拟合，表示为g(Vi,Va)。

3、模型细节

3.1 评价指标GAUC

模型使用的评价指标是GAUC，我们先来看一下GAUC的计算公式：

$$GAUC = \frac{\sum_{i=1}^n w_i * AUC_i}{\sum_{i=1}^n w_i} = \frac{\sum_{i=1}^n impression_i * AUC_i}{\sum_{i=1}^n impression_i}$$

我们首先要肯定的是，AUC是要分用户看的，我们的模型的预测结果，只要能够保证对每个用户来说，他想要的结果排在前面就好了。

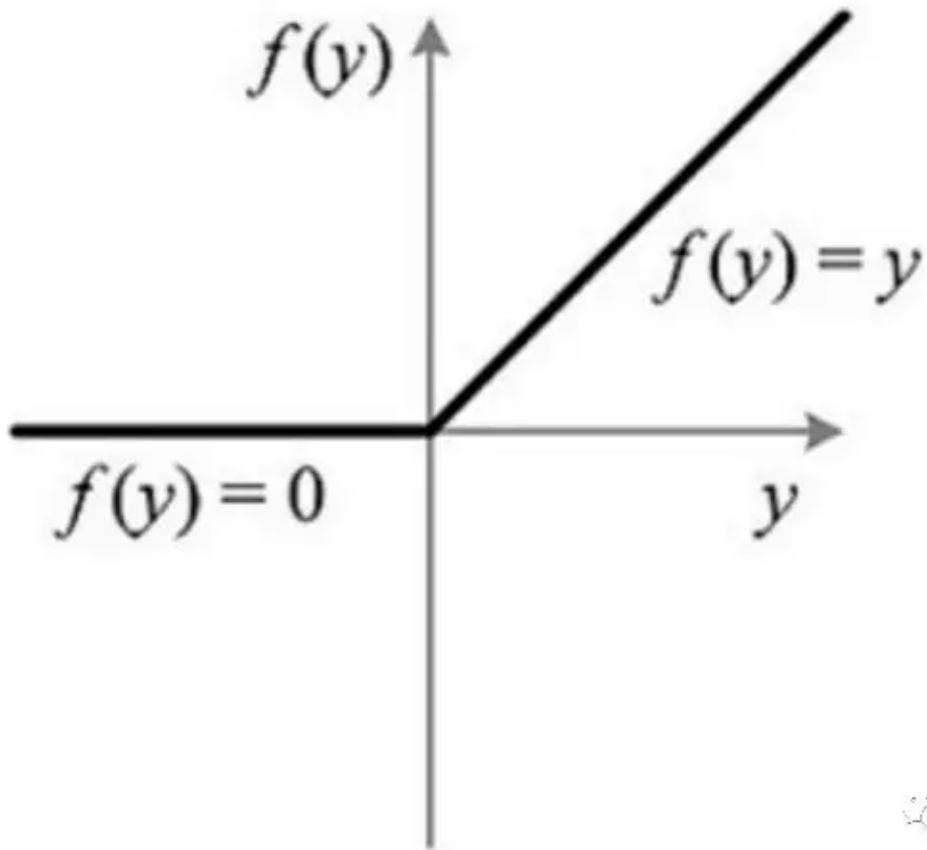
假设有两个用户A和B，每个用户都有10个商品，10个商品中有5个是正样本，我们分别用TA，TB，FA，FB来表示两个用户的正样本和负样本。也就是说，20个商品中有10个是正样本。假设模型预测的结果大小排序依次为TA，FA，TB，FB。如果把两个用户的结果混起来看，AUC并不是很高，因为有5个正样本排在了后面，但是分开看的话，每个用户的正样本都排在了负样本之前，AUC应该是1。显然，分开看更容易体现模型的效果，这样消除了用户本身的差异。

但是上文中所说的差异是在用户点击数即样本数相同的情况下说的。还有一种差异是用户的展示次数或者点击数，如果一个用户有1个正样本，10个负样本，另一个用户有5个正样本，50个负样本，这种差异同样需要消除。那么GAUC的计算，不仅将每个用户的AUC分开计算，同时根据用户的展示数或者点击数来对每个用户的AUC进行加权处理。进一步消除了用户偏差对模型的影响。通过实验证明，GAUC确实是一个更加合理的评价指标。

3.2 Dice激活函数

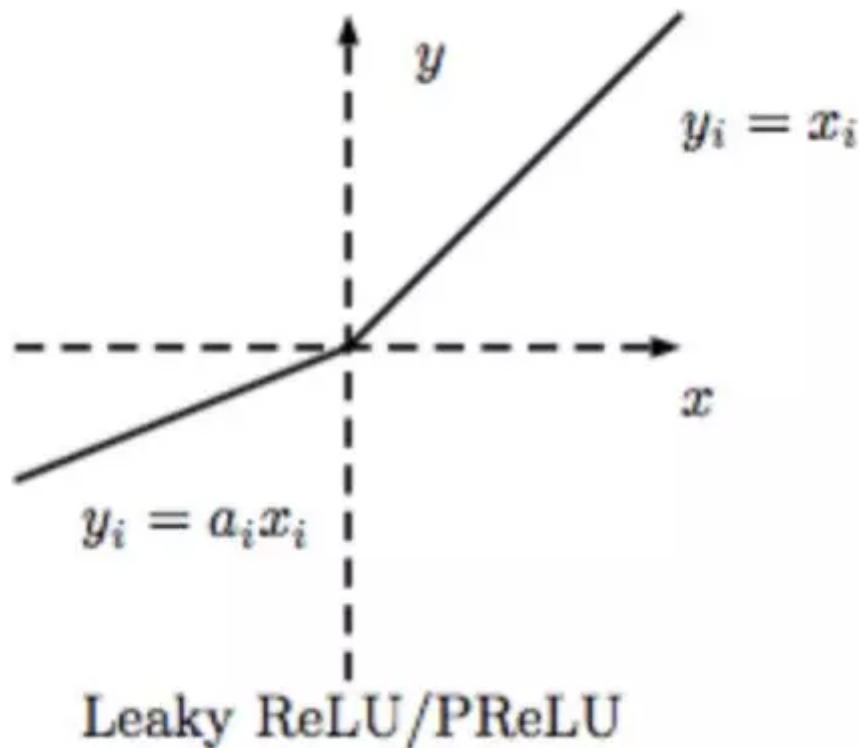
从Relu到PRelu

Relu激活函数形式如下：



小小挖掘机

Relu激活函数在值大于0时原样输出，小于0时输出为0。这样的话导致了许多网络节点的更新缓慢。因此又有了PRelu，也叫Leaky Relu，形式如下：



小小挖掘机

这样，及时值小于0，网络的参数也得以更新，加快了收敛速度。

从PReLU到Dice

尽管对Relu进行了修正得到了PRelu，但是仍然有一个问题，即我们认为分割点都是0，但实际上，分割点应该由数据决定，因此文中提出了Dice激活函数。

Dice激活函数的全称是Data Dependent Activation Function，形式如下：

$$y_i = a_i(1 - p_i)y_i + p_i y_i$$

$$p_i = \frac{1}{1 + e^{-\frac{y_i - E[y_i]}{\sqrt{\text{Var}[y_i]} + \epsilon}}}$$

小小挖掘机

其中，期望和方差的计算如下：

$$E[y_i]_{t+1}' = E[y_i]_t' + \alpha E[y_i]_{t+1}$$

$$Var[y_i]_{t+1}' = Var[y_i]_t' + \alpha Var[y_i]_{t+1}$$

可也看到，每一个 y_i 对应了一个概率值 p_i 。 p_i 的计算主要分为两步：将 y_i 进行标准化和进行sigmoid变换。

3.3 自适应正则 Adaptive Regularization

CTR中输入稀疏而且维度高，通常的做法是加入L1、L2、Dropout等防止过拟合。但是论文中尝试后效果都不是很好。用户数据符合长尾定律long-tail law，也就是说很多的feature id只出现了几次，而一小部分feature id出现很多次。这在训练过程中增加了很多噪声，并且加重了过拟合。

对于这个问题一个简单的处理办法就是：直接去掉出现次数比较少的feature id。但是这样就人为的丢掉了一些信息，导致模型更加容易过拟合，同时阈值的设定作为一个新的超参数，也是需要大量的实验来选择的。

因此，阿里提出了**自适应正则**的做法，即：

- 1.针对feature id出现的频率，来自适应的调整他们正则化的强度；
- 2.对于出现频率高的，给与较小的正则化强度；
- 3.对于出现频率低的，给予较大的正则化强度。

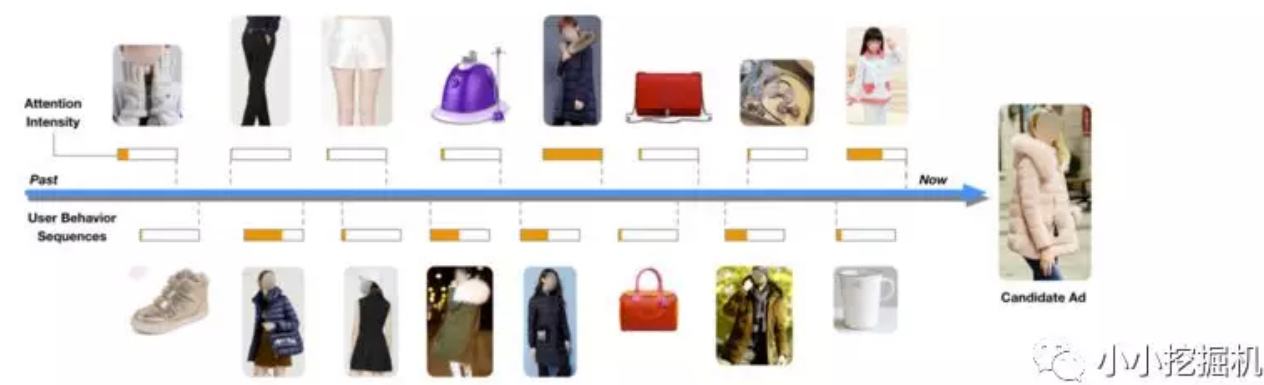
计算公式如下：

$$I_i = \begin{cases} 1, & \exists (x_j, y_j) \in B, s.t. [x_j]_i \neq 0 \\ 0, & \text{other wises} \end{cases}$$

$$w_i \leftarrow w_i - \eta \left[\frac{1}{b} \sum_{(x_j, y_j) \in B} \frac{\partial L(f(x_j), y_j)}{\partial w_i} + \lambda \frac{1}{n_i} w_i I_i \right]$$

4、效果展示

下图是对Local Activation效果的一个展示，可以看到，对于候选的广告是一件衣服的时候，用户历史行为中跟衣服相关的权重较高，而非衣服的部分，权重较低。



下图是对使用不同正则项的结果进行的展示，可以发现，使用自适应正则的情况下，模型的验证集误差和验证集GAUC均是最好的。

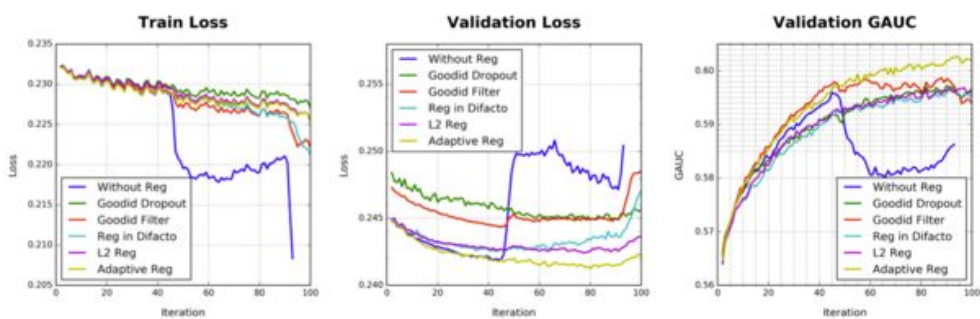


Figure 6: Performance of reduction of overfitting with different regularizations.

下图对比了Base Model和DIN的实验结果，可以看到，DIN模型在加入Dice激活函数以及自适应正则之后，模型的效果有了一定的提升：

Table 3: Comparison of model performance.		
	GAUC	GAUC gain on Base
Base Model	59.59%	0.0%
Base Model with Drop out	59.70%	0.11%
Base Model with adaptive_reg	60.31%	0.72%
DIN Model with adaptive_reg	60.60%	1.01%
DIN Model with adaptive_reg and Dice	60.83%	1.24%

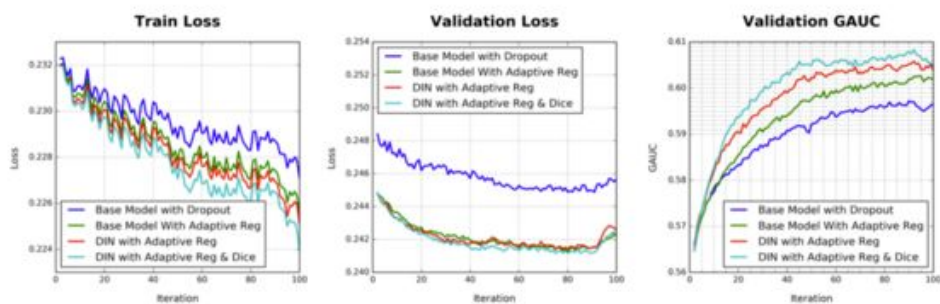


Figure 7: Performance of DIN and Base Model.

5、实战DIN

本文的github地址为：

https://github.com/princewen/tensorflow_practice/tree/master/recommendation/Basic-DIN-Demo

参考的github地址为：

<https://github.com/zhougr1993/DeepInterestNetwork/tree/master/din>

这里我们只给出一些模型细节的实现，具体的数据处理以及其他方面的内容大家可以根据上面两个地址进行学习：

数据准备

按照下面的方法下载数据：

Download dataset and preprocess

- Step 1: Download the amazon product dataset of electronics category, which has 498,196 products and 7,824,482 records, and extract it to `raw_data/` folder.

```
mkdir raw_data/;
cd utils;
bash 0_download_raw.sh;
```

- Step 2: Convert raw data to pandas dataframe, and remap categorical id.

```
python 1_convert_pd.py;
python 2_remap_id.py
```



Dice激活函数

这里实现的Dice激活函数没有根据上一步的均值方差来计算这一步的均值方差，而是直接计算了这个batch的均值方差。我们可以根据计算出的均值方差对x进行标准化(代码中被注释掉了)，也可以直接调用batch_normalization来对输入进行标准化。

注意的一点是，alpha也是需要训练的一个参数。

```
import tensorflow as tf

def dice(_x,axis=-1,epsilon=0.0000001,name=''):

    alphas = tf.get_variable('alpha'+name,_x.get_shape()[-1],
                              initializer = tf.constant_initializer(0.0),
                              dtype=tf.float32)

    input_shape = list(_x.get_shape())
```

```

reduction_axes = list(range(len(input_shape)))

del reduction_axes[axis] # [0]

broadcast_shape = [1] * len(input_shape) #[1,1]
broadcast_shape[axis] = input_shape[axis] # [1 * hidden_unit_size]

# case: train mode (uses stats of the current batch)
mean = tf.reduce_mean(_x, axis=reduction_axes) # [1 * hidden_unit_size]
broadcast_mean = tf.reshape(mean, broadcast_shape)
std = tf.reduce_mean(tf.square(_x - broadcast_mean) + epsilon, axis=reduction_
s)
std = tf.sqrt(std)
broadcast_std = tf.reshape(std, broadcast_shape) #[1 * hidden_unit_size]
# x_normed = (_x - broadcast_mean) / (broadcast_std + epsilon)
x_normed = tf.layers.batch_normalization(_x, center=False, scale=False) # a sim
ple way to use BN to calculate x_p
x_p = tf.sigmoid(x_normed)

return alphas * (1.0 - x_p) * _x + x_p * _x

```

Activation Unit

这里的输入有三个，候选广告queries，用户历史行为keys，以及Batch中每个行为的长度。这里为什么要输入一个keys_length呢，因为每个用户发生过的历史行为是不一样的，但是输入的keys维度是固定的(都是历史行为最大的长度)，因此我们需要这个长度来计算一个mask，告诉模型哪些行为是没用的，哪些是用来计算用户兴趣分布的。

经过以下几个步骤得到用户的兴趣分布：

1. 将queries变为和keys同样的形状B * T * H(B指batch的大小，T指用户历史行为的最大长度，H指embedding的长度)
2. 通过三层神经网络得到queries和keys中每个key的权重，并经过softmax进行标准化
3. 通过weighted sum得到最终用户的历史行为分布

```

def attention(queries,keys,keys_length):
    ...

    queries:      [B, H]
    keys:         [B, T, H]
    keys_length:  [B]
    ...

    queries_hidden_units = queries.get_shape().as_list()[-1]
    queries = tf.tile(queries,[1,tf.shape(keys)[1]])
    queries = tf.reshape(queries,[-1,tf.shape(keys)[1],queries_hidden_units])

    din_all = tf.concat([queries,keys,queries-keys,queries * keys],axis=-1) # B*T*4H

```



```

# 三层全链接
d_layer_1_all = tf.layers.dense(din_all, 80, activation=tf.nn.sigmoid, name='f1_
att')
d_layer_2_all = tf.layers.dense(d_layer_1_all, 40, activation=tf.nn.sigmoid, nam
e='f2_att')
d_layer_3_all = tf.layers.dense(d_layer_2_all, 1, activation=None, name='f3_att'
) #B*T*1

outputs = tf.reshape(d_layer_3_all, [-1,1,tf.shape(keys)[1]]) #B*1*T
# Mask
key_masks = tf.sequence_mask(keys_length,tf.shape(keys)[1])
key_masks = tf.expand_dims(key_masks,1) # B*1*T
paddings = tf.ones_like(outputs) * (-2 ** 32 + 1) # 在补足的地方附上一个很小的值，
而不是0
outputs = tf.where(key_masks,outputs,paddings) # B * 1 * T
# Scale
outputs = outputs / (keys.get_shape().as_list()[-1] ** 0.5)
# Activation
outputs = tf.nn.softmax(outputs) # B * 1 * T
# Weighted Sum
outputs = tf.matmul(outputs,keys) # B * 1 * H 三维矩阵相乘，相乘发生在后两维，即 B
* (( 1 * T ) * ( T * H ))
return outputs

```

参考文献

- 1、盖坤演讲视频：<http://www.itdks.com/dakalive/detail/3166>
- 2、论文：Deep Interest Network for Click-Through Rate Prediction
- 3、github：<https://github.com/zhougr1993/DeepInterestNetwork>

推荐系统遇上深度学习系列：

推荐系统遇上深度学习(十四)--强化学习与推荐系统的强强联合！

推荐系统遇上深度学习(十五)--强化学习在京东推荐中的探索

推荐系统遇上深度学习(十六)--详解推荐系统中的常用评测指标

推荐系统遇上深度学习(十七)--探秘阿里之MLR算法浅析及实现

有关作者：

石晓文，中国人民大学信息学院在读研究生，美团外卖算法实习生

简书ID：石晓文的学习日记(<https://www.jianshu.com/u/c5df9e229a67>)

天善社区：<https://www.hellobi.com/u/58654/articles>

腾讯云：<https://cloud.tencent.com/developer/user/1622140>

开发者头条：<https://toutiao.io/u/470599>