深度兴趣网络DIN(Deep Interest Network)浅析和实现

原创 小六 机器学习与数据挖掘实践 4月19日

收录于话题
#推荐系统学习

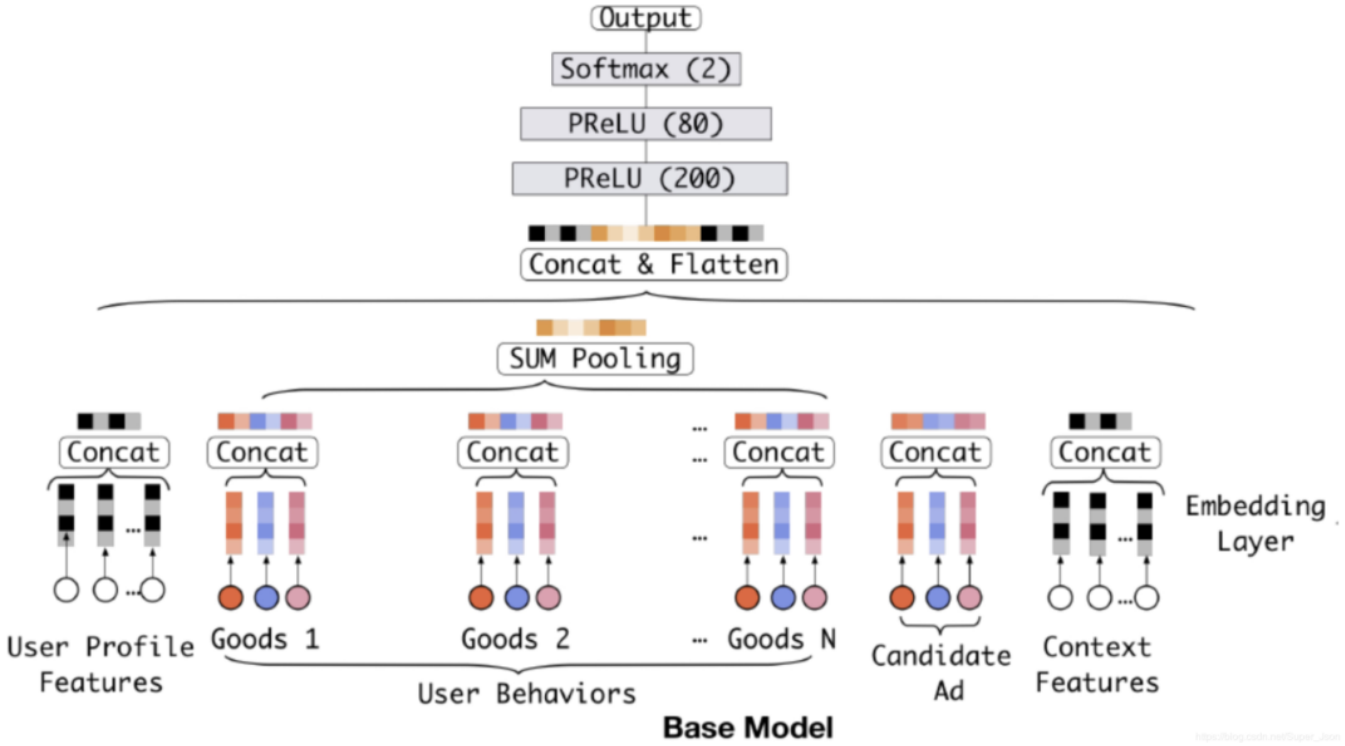
7个

> 本文收录在推荐系统专栏，专栏系统化的整理推荐系统相关的算法和框架，并记录了相关实践经验，所有代码都已整理至推荐算法实战集合(hub-recsys)。

目录

一. 论文浅析

常见的深度学习网络应用于推荐系统或者CTR预估时，都具备如下的基本模式：Sparse Features -> Embedding Vector -> MLPs -> Sigmoid -> Output，如下图所示。这种方法主要通过DNN网络抽取特征的高阶特征，减少人工特征组合。对用户历史行为数据进行处理时，需要把它们编码成一个固定长的向量，但是每个用户的历史点击个数是不相等的，通常的做法是对每个item embedding后，进入pooling层（求和或最大值)。DIN认为这样操作损失了大量的信息，故此引入attention机制，并提出了 Dice 激活函数，自适应正则，显著提升了模型性能与收敛速度



1.1 注意力机制-attention

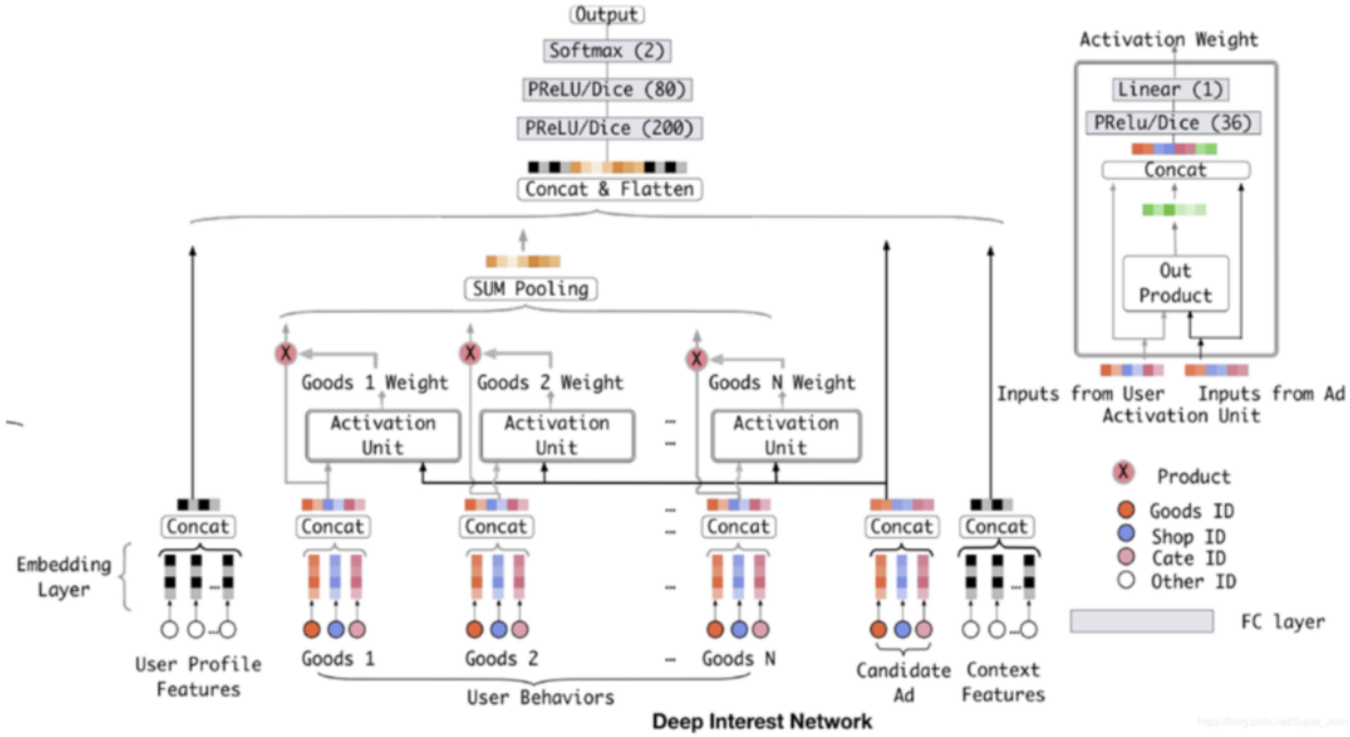
对于用户的兴趣而言，存在以下两点特性：

- **Diversity**：用户在浏览电商网站的兴趣多样性。
- **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。

为了充分挖掘用户历史行为的这两点特性，区别与一般的深度模型，引入attention，在模型预测时赋予不同的历史行为不同的权重，实现局部激活，“相关”的行为历史看重一些，“不相关”的历史甚至可以忽略。

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

上式中， Vu是用户的embedding向量，Va是候选广告商品的embedding向量， Vi是用户u的第i次行为的embedding向量，因为这里用户的行为就是浏览商品或店铺，所以行为的embedding的向量就是那次浏览的商品或店铺的embedding向量。因为加入了注意力机制， Vu从Vi的加和变成对Vi的加权和，Vi的权重wi就由Vi和Va共同刻画，即g(Vi,Va)。



一般attention，可以直接利用向量点击。DIN的activation unit层即g(Vi,Va)，**首先是把u和v以及u v的element wise差值向量合并起来作为输入，然后喂给全连接层，最后得出权重，**这样的方法显然损失的信息更少。同时引入field的概念，每个ad会有good_id, shop_id 两层属性，shop_id只跟用户历史中的shop_id序列发生作用，good_id只跟用户的good_id序列发生作用，这样做的原因也是显而易见的。

1.2 激活函数-Dice

从ReLU到PReLU

在介绍Dice函数之前，我们回顾下ReLU函数和PReLU函数。ReLU函数其实是分段线性函数，把所有的负值都变为0，而正值不变，具备①单侧抑制 ②相对宽阔的兴奋边界 ③稀疏激活性特性，利用单侧抑制，使得神经网络中的神经元也具有了稀疏激活性。因此通过ReLU实现稀疏后的模型能够更好地挖掘相关特征，拟合训练数据。

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。因此又了PRelu，也叫Leaky Relu，形式如下：

$$PReLU(s) = \begin{cases} s, & s>0 \\ \alpha s, & s \leq 0 \end{cases} = p(s) * s + (1 - p(s)) * \alpha s$$

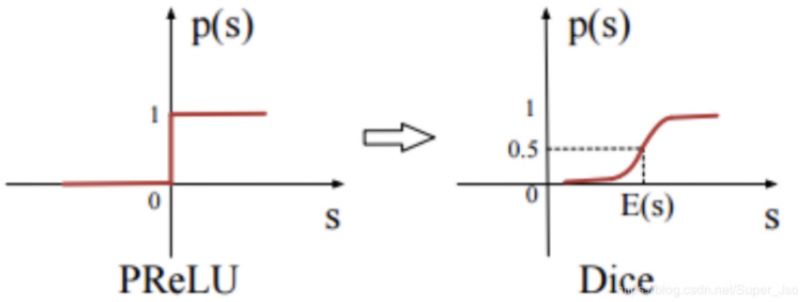
这样，及时值小于0，网络的参数也得以更新，加快了收敛速度。

从PReLU到Dice

尽管对Relu进行了修正得到了PRelu，但是无论 ReLU 还是 PReLU 突变点都在 0，论文里认为，对于所有输入不应该都选择 0 点为突变点而是应该依赖于数据的。于是提出了一种 **data dependent** 的方法：**Dice 激活函数。**

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

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



可以看出，pi 是一个概率值，这个概率值决定着输出是取 yi 或者是 alpha_i * yi，pi 也起到了一个整流器的作用。pi 的计算分为两步：

1. 首先，对 x 进行均值归一化处理，这使得整流点是在数据的均值处，实现了 data dependent 的想法；
2. 其次，经过一个 sigmoid 函数的计算，得到了一个 0 到 1 的概率值。巧合的是最近 google 提出的 Swish 函数形式为x*sigmoid(x) 在多个实验上证明了比 ReLU 函数x*Max(x,0)表现更优。

另外，期望和方差使用每次训练的 mini batch data 直接计算，并类似于 Momentum 使用了指数加权平均：

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

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

此处对计算复杂度和性能提升有一些疑问。

1.3 评价指标-GAUC

用户级别的AUC计算：AUC 表示正样本得分比负样本得分高的概率。在 CTR 实际应用场景中，CTR 预测常被用于对每个用户候选广告的排序。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。实现了用户级别的 AUC 计算。

用户加权AUC计算：上述评估适用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间的展示次数或者点击数，有些用户天生就是点击率高。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

$$\text{GAUC} = \frac{\sum_{i=1}^{n} w_i * \text{AUC}_i}{\sum_{i=1}^{n} w_i} = \frac{\sum_{i=1}^{n} impression_i * \text{AUC}_i}{\sum_{i=1}^{n} impression_i}$$

1.4 自适应正则-Adaptive Regularization

由于商品id维度符合长尾定律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。这类特征对应的embedding矩阵表是巨大的，模型参数太多，如果不加正则化则模型很快过拟合。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。

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

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

DIN提出了新的正则化方式，只对batch中参与了前向计算的embedding向量进行更新。

二. 代码解读

本文参考网上相关实现代码，复现了DIN的实现，并且使用亚马逊数据集进行简单的实践，数据集主要包括品评论和产品原始数据。

2.1 数据处理

我们将数据进行整理和切分，用户的所有行为都是（b1，b2，…，bk，…，bn），我们构造预测任务为利用前k个评论商品来预测第（k + 1）个评论的商品，任务是通过 训练数据集是用每个用户的k = 1,2，…，n-2生成的。在测试集中，我们预测最后一个给出第一个n - 1评论商品。

2.2 attention机制

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

多的，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) #
    # 三层全链接
    d_layer_1_all = tf.layers.dense(din_all, 80, activation=tf.nn.sigmoid, na
    d_layer_2_all = tf.layers.dense(d_layer_1_all, 40, activation=tf.nn.sigmo
    d_layer_3_all = tf.layers.dense(d_layer_2_all, 1, activation=None, name='

    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
    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
```

```
30      return outputs
```

完整的实现代码：https://github.com/hxyue/hub-recsys

推荐阅读：

我做算法工程师的第1年

24个终极数据科学项目(含数据集，免费获取)

十道海量数据处理面试题

机器学习 | 数据挖掘 | NLP | 爬虫

长 按 二 维 码 关 注 我 们

阅读原文