DIN

早期的做法：

- Embedding+MLP：不考虑顺序，只进行简单相加。虽然"看上去"简单相加好像是损失了很多信息，譬如说想象一个二维空间，如果把四个向量位置加权平均，那么最后的点会是这四个点的中点，是个"四不像"。但实际上，在高维空间是我们无法想象的，可以理解为每个点都是一个"小尖尖"上的点，那么平均之后不是"四不像"，而是"四都像"。其实，concat之后+MLP和平均+MLP的表征能力是相似的。见知乎问题：https://www.zhihu.com/question/483946894

1. DIN

之前的CTR预估模型大多使用 Embedding+MLP的方法。这样，不管candidate ads是什么，**一个用户只用一个定长的、不变的向量来表示**，这样模型不能充分把握来源于用户**丰富历史行为**的**多样兴趣(diverse interest)**，所以这个定长的向量就会称为表征的bottleneck。为了增强这样一个定长向量的**表征能力**，需要大大增加这个向量的**维度**。但不幸的是，这样会大大增加参数量、从而导致过拟合。

在这篇文章中，作者提出Deep Interest Network(DIN)：

1. 使用Attention机制来实现Local Activation unit，能够**针对不同的target ads来计算应该分配给用户历史行为的权重**，从而得到不同的user embedding，这样能够打破定长向量这个bottleneck，增强模型的表征能力。
2. 针对模型训练，提出了**Dice**激活函数，**mini-batch aware regularization**，显著提升了模型性能与收敛速度。

1.1 简介

计算广告的一些知识：

- eCPM(effective cost per mille)指的就是**每一千次展示可以获得的广告收入**, eCPM = bid price * CTR

因此，广告中的CTR预估十分重要。

- QPS (query per second)

1.2 系统总览

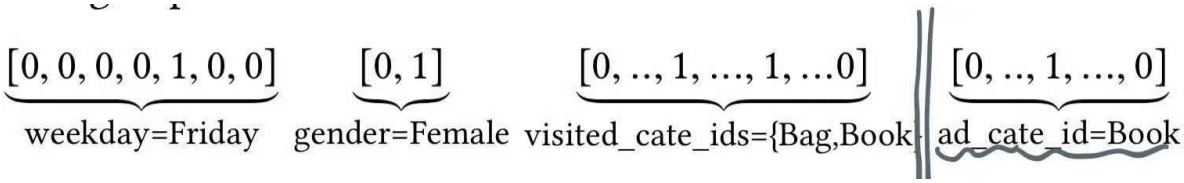
在工业界的搜索、广告、推荐架构中，通常包括召回、排序（粗排、精排等）两大模块，无论是对于召回模块还是排序模块来说，对用户历史行为建模都是十分重要的。同时需要指出的是在对用户历史行为进行建模的时候需要了解到用户的行为兴趣是diverse的而且可能是动态变化的，所以如何捕捉用户的多个兴趣点是在用户历史行为建模中非常重要的一点。

1.3 Base模型：Embedding+MLP

Base模型就是现在比较常见的多层神经网络，即先对特征进行Embedding操作，得到一系列Embedding向量之后，将不同group的特征concate起来之后得到一个固定长度的向量，然后将此向量喂给后续的全连接网络。这样的方法比起之前的logistic regression，不用手动构造交叉特征，MLP完成隐式的特征交叉。

1.3.1 Feature Representation

在CTR预估中的输入特征一般是高维、稀疏的。比如：

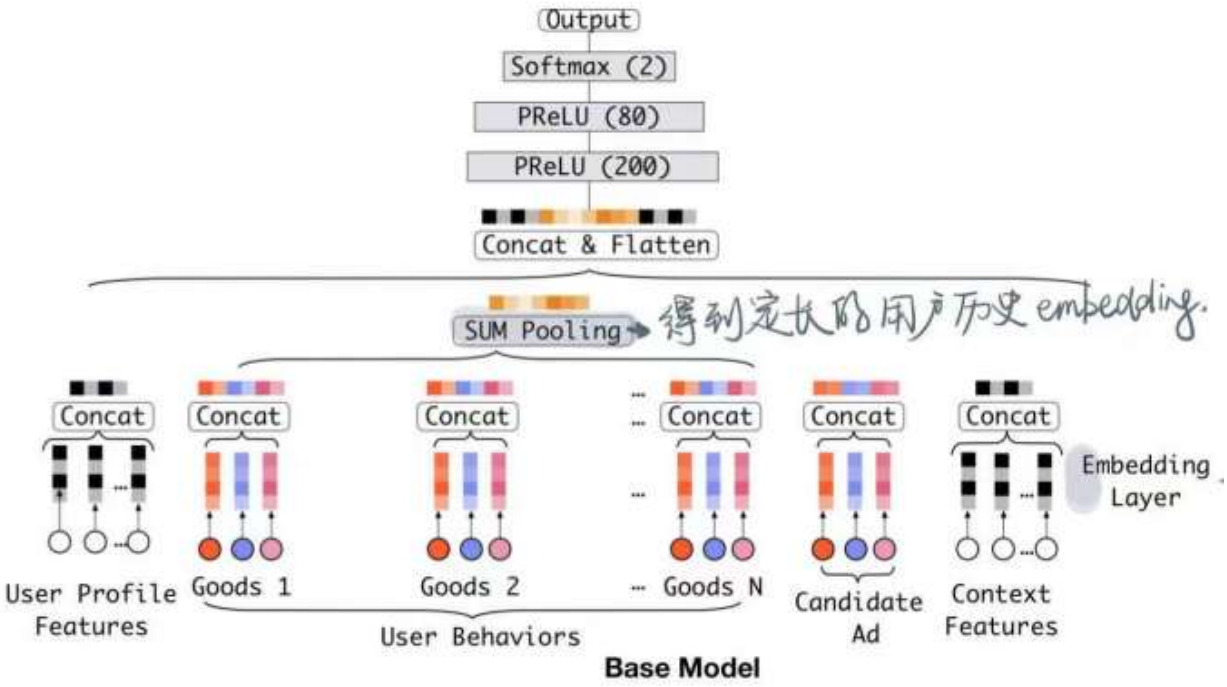


其中，visited-cate-ids为用户的历史行为信息，是multi-hot，其他都是one-hot。

论文中作者把特征分为四大类，并没有进行特征交叉，而是**通过DNN去学习特征间的交互信息**(隐式交互)。特征如下：

Category	Feature Group	Dimemsionality	Type	#Nonzero Ids per Instance
User Profile Features	gender	2	one-hot	1
	age_level	~ 10	one-hot	1
	...	...	...	...
User Behavior Features	visited_goods_ids	~ 10^9	multi-hot	~ 10^3
	visited_shop_ids	~ 10^7	multi-hot	~ 10^3
	visited_cate_ids	~ 10^4	multi-hot	~ 10^2
Ad Features	goods_id	~ 10^7	one-hot	1
	shop_id	~ 10^5	one-hot	1
	cate_id	~ 10^4	one-hot	1
	...	...	...	...
Context Features	pid	~ 10	one-hot	1
	time	~ 10	one-hot	1
	...	...	...	...

1.3.2 Base Model



Embedding：就是去查look-up table。对于one-hot编码，得到一个embedding向量；对于multi-hot编码，得到多个embedding向量，然后做sum-pooling/max-pooling使其降到固定的向量长度。

Concat：把所有的向量（user features, 用户历史行为的pooling,Target ad, Context features）全部concat起来，作为一个综合的表征，送入MLP中。

损失函数为负log损失：

$$L = -\frac{1}{N} \sum_{(x,y)\in S} (y \log p(x) + (1-y) \log(1-p(x)))$$

1.4. DIN结构

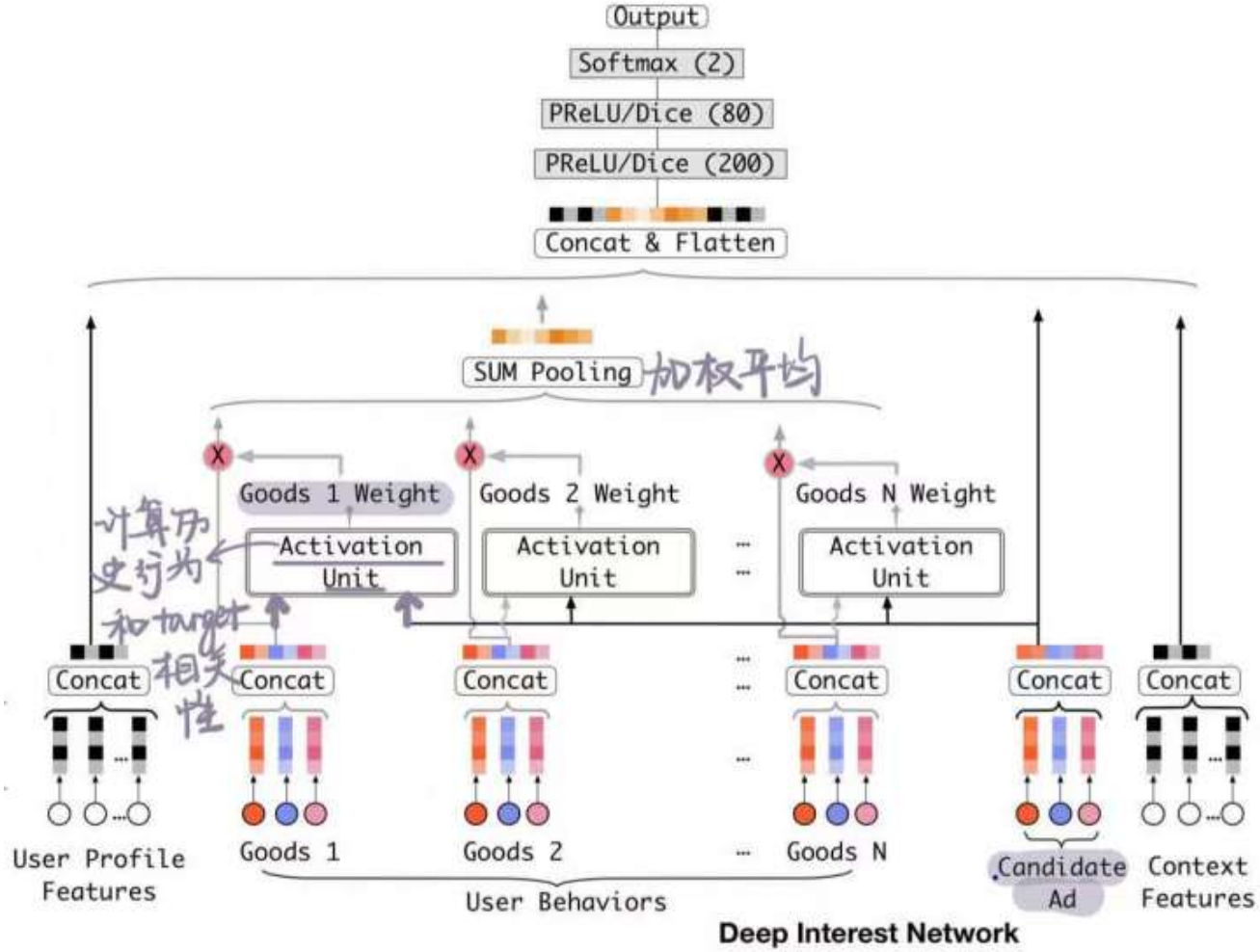
上文介绍了一些特征，包括user features, user behaviors, Target ad features, Context features.其中，用户历史行为信息(user behaviors)尤其重要。Base模型中，不管candidate ads是什么，一个用户只用一个定长的、不变的向量来表示，不能体现用户【多样的兴趣】。

我们模型的目标：基于用户历史行为，充分挖掘用户历史行为和候选广告(target ad)之间的关系。用户是否点击某个广告往往是基于他之前的**部分兴趣**，这是应用Attention机制的基础，所以我们应该对历史行为进行soft search，只去关注那些和target ads有关的行为，给他们以高的权重，最后的行为向量表示就是一个加权求和。例如，一位年轻的母亲购买过一些包包、水杯、童装、鞋子、羽绒服，她的兴趣是多样的。这时，我们要考虑她点击一个羽绒服的概率。其实我们只需要关注她之前买的羽绒服就好了，这个概率和她之前买的包包、水杯等物品关系不大：

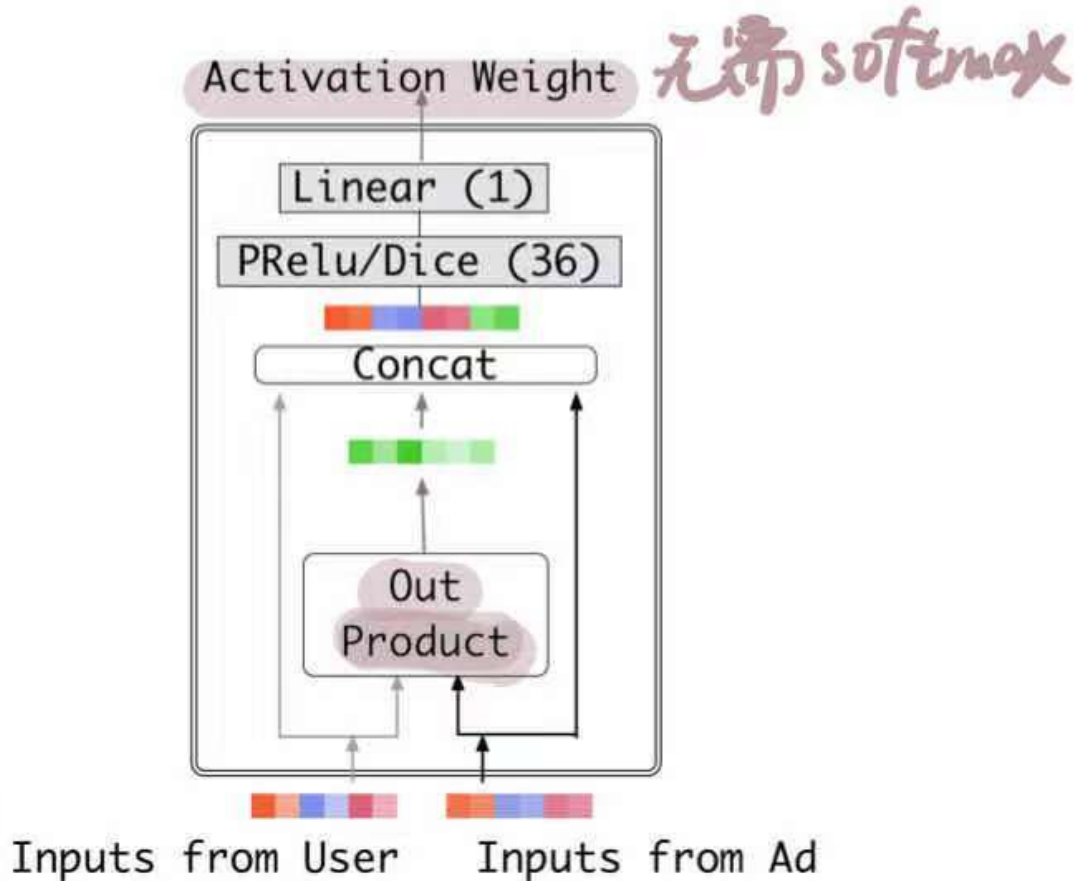
> Behaviors related to displayed ad greatly contribute to the click action.

Figure 5: Illustration of adaptive activation in DIN. Behaviors with high relevance to candidate ad get high activation weight.

DIN结构：



对历史行为中的每个商品，都增加一个activation unit，计算一下该商品和candidate(target) ad的相关性。
activation unit的结构如下：




```
def din_attention(self, query, facts, mask, sum = True):

        queries = tf.tile(query, [1, tf.shape(facts)[1]]) ##(?,640)
        queries = tf.reshape(queries, tf.shape(facts)) ##(128,20,32)
        din_all = tf.concat([queries, facts, queries-facts, queries*facts], axis=-1)#
(128,20,128)
        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,
name='f2_att')
        d_layer_3_all = tf.layers.dense(d_layer_2_all, 1, activation=None,
name='f3_att') #(128,20,1)
        d_layer_3_all = tf.reshape(d_layer_3_all, [-1, 1, tf.shape(facts)[1]]) #
(128,1,20)

        scores = d_layer_3_all

        if mask is not None:
            mask = tf.equal(mask, tf.ones_like(mask))
            key_masks = tf.expand_dims(mask, 1) # [Batchsize, 1, max_len]
            paddings = tf.ones_like(scores) * (-2 ** 32 + 1) # so that will be 0 after
softmax
```

```
            scores = tf.where(key_masks, scores, paddings)  # if key_masks then scores
else paddings

        scores = tf.nn.softmax(scores)  # [B, 1, T]
        if sum:
            output = tf.matmul(scores, facts)  # [B, 1, H]
            output = tf.squeeze(output)
        else:
            scores = tf.reshape(scores, [-1, tf.shape(facts)[1]])
            output = facts * tf.expand_dims(scores, -1)
            output = tf.reshape(output, tf.shape(facts))
            scores = tf.expand_dims(scores,-1)
        return output, scores
```

注意力得分可以由一个MLP来完成；当然了，如果使用现在流行的Transformer结构，我们可以把target ad当成query，把用户历史行为当成Key和Value，来进行用户历史行为的加权求和。

综合以上，一个用户历史行为的embedding为：

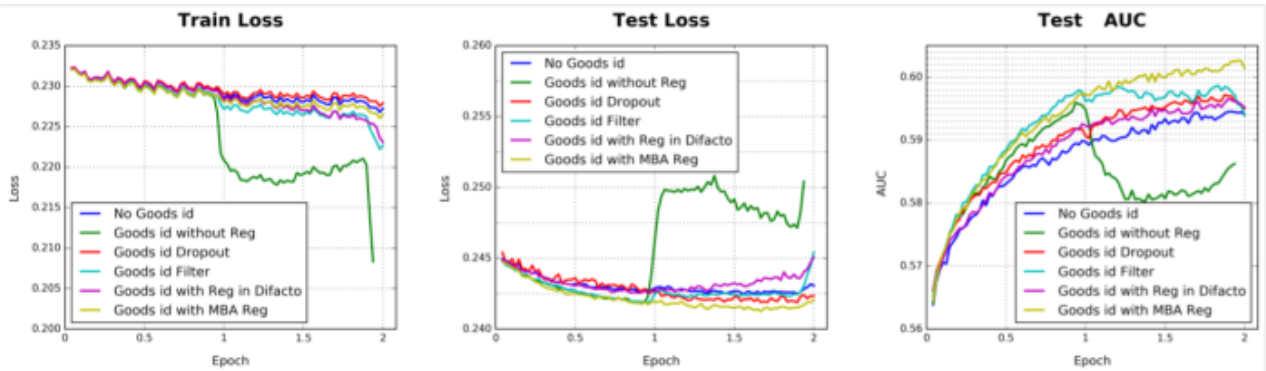
$$\boldsymbol{v}_U(A) = f(\boldsymbol{v}_A, \boldsymbol{e}_1, \boldsymbol{e}_2, .., \boldsymbol{e}_H) = \sum_{j=1}^{H} a(\boldsymbol{e}_j, \boldsymbol{v}_A)\boldsymbol{e}_j = \sum_{j=1}^{H} w_j \boldsymbol{e}_j, \quad (3)$$

文中作者尝试用LSTM来建模用户行为序列，但没有提升。这是因为用户的兴趣有很多跳跃，带来了噪声。所以，DIN模型**是没有建模序列信息的**（这也正是DIEN的优化点）。

2. 训练的Tricks

2.1 Mini-batch Aware Regularization

在大规模稀疏场景下，一些**id类特征的维度很高**（即：有非常多的取值），例如在实验中goods_id有6亿维。如果不做正则化，会导致严重的过拟合 (**因为Embedding table的参数量巨大**)，如图中深绿色线所示：



但是，由于Embedding table**参数量巨大**，所以直接使用 L1或者L2 正则化也是不现实的。

> Only parameters of **non-zero** sparse features appearing in each mini-batch needs to be updated in the scenario of SGD based optimization methods without regularization. However, when adding $\ell 2$ regularization it needs to calculate L2-norm over the **whole** parameters for each mini-batch, which leads to extremely heavy computations and is unacceptable with parameters scaling up to hundreds of millions.

Mini-Batch Aware regularization 主要解决的就是在大规模**稀疏**场景下，采用SGD对引入L2正则的loss进行更新时计算开销过大的问题。因为引入正则化以后，不管特征是不是0，都需要计算梯度，对大规模的稀疏特征，参数规模也非常庞大，增加的计算量就非常大。而MBA方法只对每一个mini-batch中**不为0的特征**对应的embedding table行进行梯度更新。

“only update those parameters of sparse features **appearing** in each mini-batch”.

实际上，主要就是Embedding look-up table导致的参数量巨大。Embedding table记作 $W \in \mathbb{R}^{D*K}$,其中K是特征取值的数量，D是embedding vector的维度。$w_j \in \mathbb{R}^D$ 是W中第 j 个特征对应的embedding。原始的L2正则化项为：

$$L_2(W) = ||W||_2^2 = \sum_{j=1}^{K} ||w_j||_2^2$$

即对所有的特征取值的embedding（整个embedding table）都做正则化，而这样的计算量太大了。

推导得到修改过的L2正则项为：

$$L_2(\mathbf{W}) \approx \sum_{j=1}^{K} \sum_{m=1}^{B} \frac{\alpha_{mj}}{n_j} \| \boldsymbol{w}_j \|_2^2 .$$

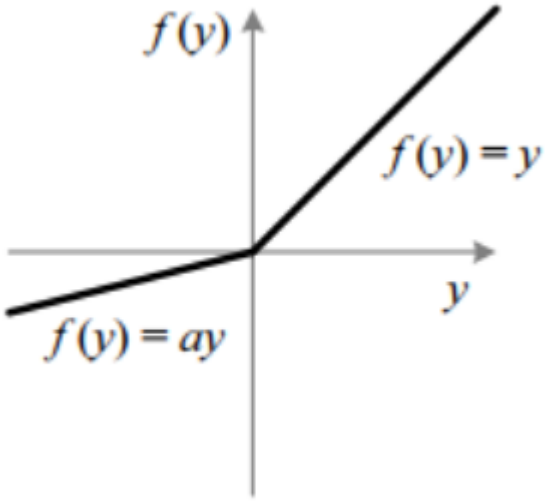
n_j 表示特征 j 在所有样本中出现的次数， $\alpha_{mj} = 1$表示当前batch中特征 j 至少有一个样本非0。这样，**只有在batch m中出现过的特征对应的Embedding table行会计算在L2正则项中**。而且，**正则的强度与特征的频率有关**，频率越高，正则越小，频率越低，正则越大。

求导后得到梯度：

$$\boldsymbol{w}_j \leftarrow \boldsymbol{w}_j - \eta \left[\frac{1}{|\mathcal{B}_m|} \sum_{(\boldsymbol{x}, y) \in \mathcal{B}_m} \frac{\partial L(p(\boldsymbol{x}), y)}{\partial \boldsymbol{w}_j} + \lambda \frac{\alpha_{mj}}{n_j} \boldsymbol{w}_j \right],$$

4.2 自适应激活函数Dice

文章认为采用PRelu激活函数时，它的**rectified point固定为0**，这在每一层的输入分布发生变化时是不适用的。



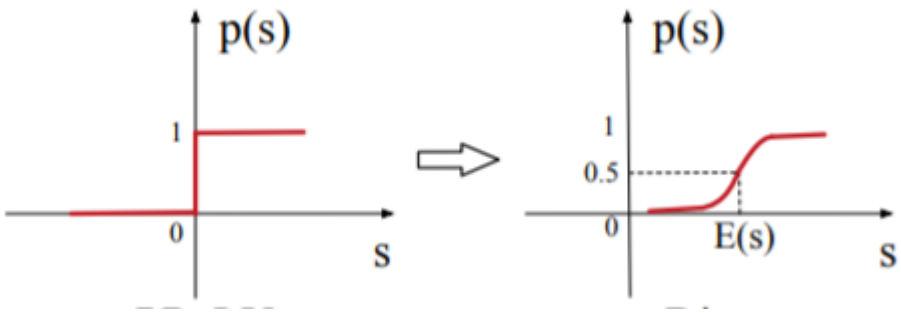
(PRelu)

所以文章对该激活函数进行了改进：**平滑**了rectified point附近曲线的同时，激活函数会**根据每层输入数据的分布来自适应调整rectified point的位置**，具体形式如下：

"adaptively adjust the rectified point w.r.t. distribution of inputs"

$$f(s) = p(s) \cdot s + (1 - p(s)) \cdot \alpha s, \quad p(s) = \frac{1}{1 + e^{-\frac{s - E[s]}{\sqrt{Var[s] + \epsilon}}}}$$

其中，



左侧是PRelu的p(s)曲线，右侧是DICE的p(s)曲线

DIN在线上表现：10.0%的CTR提升和3.8%的RPM(Revenue Per Mille) 提升