deeprec—CTR模型系列：DIN和Autoint

原创 kylekzhang 算法工程师养成之路 2019-07-01

导引

我们在前面相继介绍了deeprec库，FM、FFM、AFM、DCN、Deepfm、CCPM和FGCNN相关模型基础和如何在deeprec中应用这些模型。在本节中我们介绍另外两种模型，即DIN和Autoint，相对于前几个模型，这两个模型的特点是侧重在处理序列模型。

特别提醒

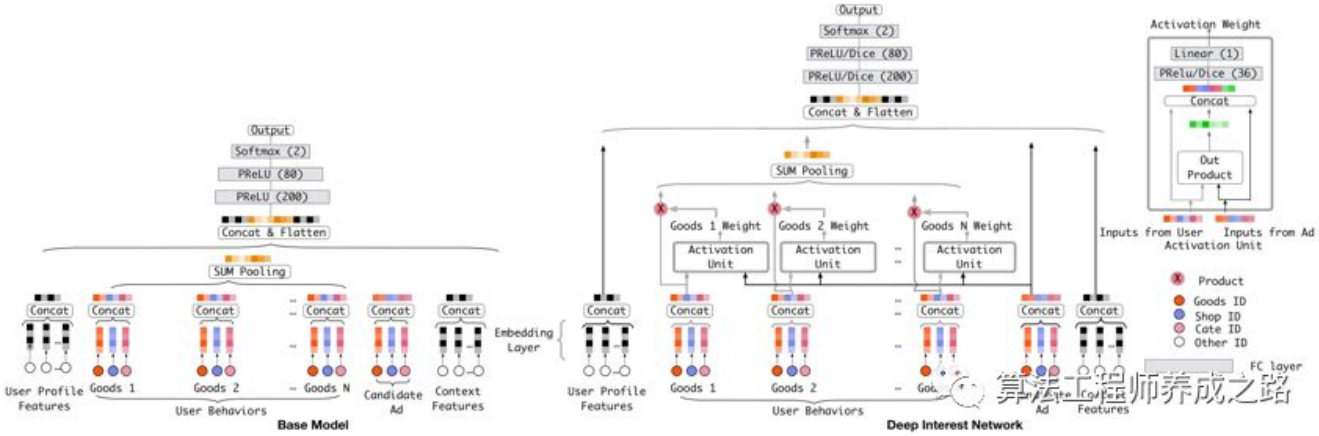
本文所有代码均可以参考：
https://github.com/End-the-cold-night/deeprec/tree/master/deeprec/ranking/ctr

deeprec是一个通用的推荐和广告算法库，目前支持常见的CTR模型，后续会陆续补充CVR，Graph embedding和kg相关的算法。欢迎大家star和contribute。

DIN 01

DIN的模型输入比较简单，主要是多了序列特征。以往模型在处理序列特征的时候是直接将序列特征分别做embedding后平均得到一个表示，这个显然是可以改进的，就像通常而言加权平均会比普通的平均要好，DIN也是类似的操作，只是这里的“权重”是通过和target item 的attention单元获取。

我们首先看系统结构图：

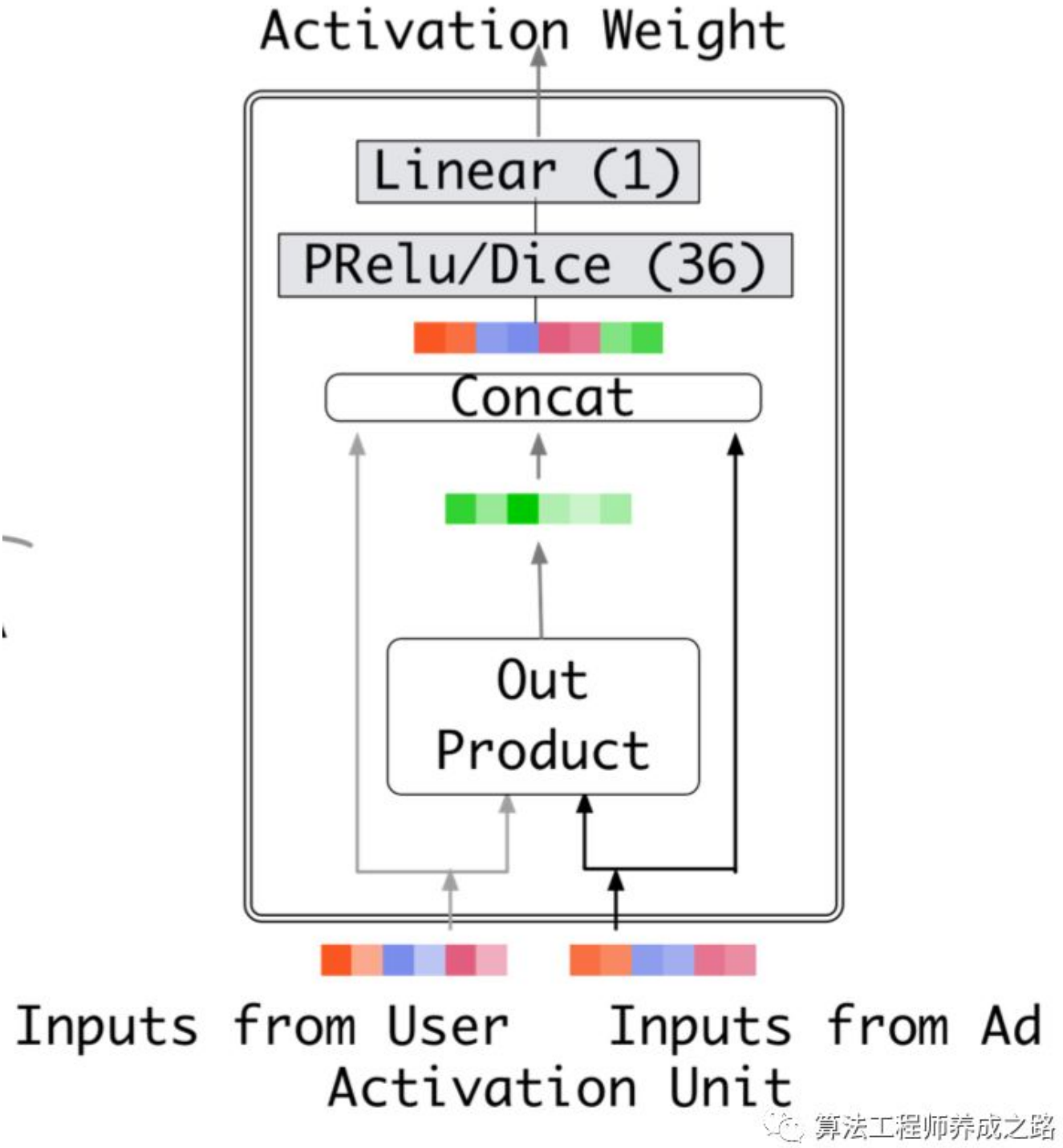


不难发现，从网络结构上，实际上只是修改了序列数据的处理方式。

具体的，线性部分处理方式和之前模型一致

```
# Linear part
sprase_feature, self.sprase_data_linear_embedding = \
    get_linear_embedding(self.feature_config_dict,self.sprase_data, self.number_of_sprase_feature)
```


核心的DIN的部分主要是图中的这个单元，即：



转换为tensorflow代码即：

```
din_all = tf.concat([queries, keys, queries - keys, queries * keys], axis=-1)  # bs * length * es

# dnn
d_layer_1_all = tf.layers.dense(din_all, 80, activation=dice, name='f1_att_{}'.format(name))
d_layer_2_all = tf.layers.dense(d_layer_1_all, 40, activation=dice, name='f2_att_{}'.format(name))
d_layer_3_all = tf.layers.dense(d_layer_2_all, 1, activation=None, name='f3_att_{}'.format(name))  # [ba

outputs = tf.reshape(d_layer_3_all, [-1, 1, tf.shape(keys)[1]])  # [bs, 1, length]
```


由于输入序列长度不等，所以改成这个attention方式后需要处理mask的信息，因此，完整版DIN需要额外定义mask输入，对于keras如果在Embedding层选择了支持masking，则不需要额外的定义。

此外，DIN的另一个contribution在于提出了DICE激活函数，

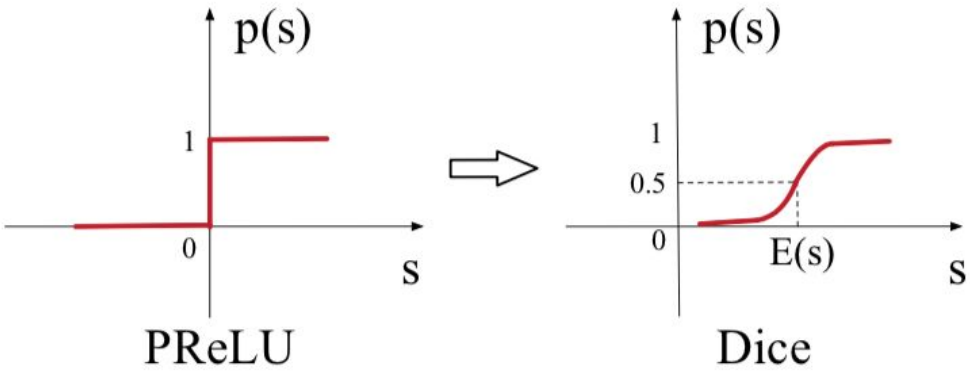


Figure 3: Control function of PReLU and Dice

实现：

```
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())  # [batch_size, flied_size * embedding_
    reduction_axes = list(range(len(input_shape)))  # [0,1]

    del reduction_axes[axis]  # [0]

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

    # case: train mode (uses stats of the current batch)
    mean = tf.reduce_mean(_x, axis=reduction_axes)  # [1 * hidden_unit_size]
    brodcast_mean = tf.reshape(mean, broadcast_shape)
    std = tf.reduce_mean(tf.square(_x - brodcast_mean) + epsilon, axis=reducti
    std = tf.sqrt(std)
    brodcast_std = tf.reshape(std, broadcast_shape)  # [1 * hidden_unit_size]
    x_normed = (_x - brodcast_mean) / (brodcast_std + epsilon)
    # x_normed = tf.layers.batch_normalization(_x, center=False, scale=False)
    x_p = tf.sigmoid(x_normed)

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

调包使用：

```
from deeprec.ranking.ctr import Din
```

完整的例子详见github

02 Autoint

Transfomer在NLP中被证明是有效的，且通常是优于RNN的，Autoint模型则是采用了该结构，用于处理数据。导引提到Autoint处理序列数据，这不准确，应该是可以处理，真正用到Transfomer处理序列数据的是DSIN，后续会介绍。同样的，我们先看网络结构图：

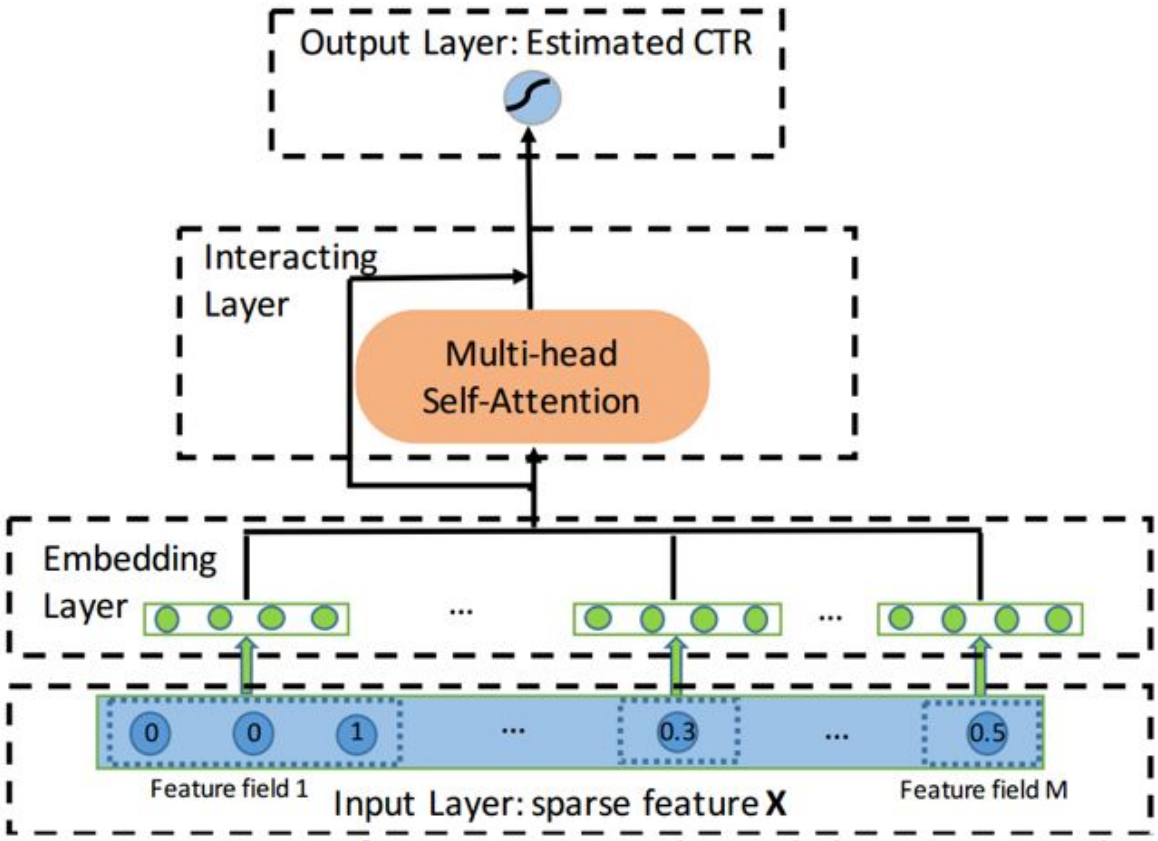


Figure 1: Overview of our proposed model AutoInt. The details of embedding layer and interacting layer are illustrated in Figure 2 and Figure 3 respectively.

Autoint的网络结构相当清晰，把输入通过embedding层映射到低维dense向量后，直接通过muti-head self-attention即可。这里我们先了解下muti-head self-attention，结构如下

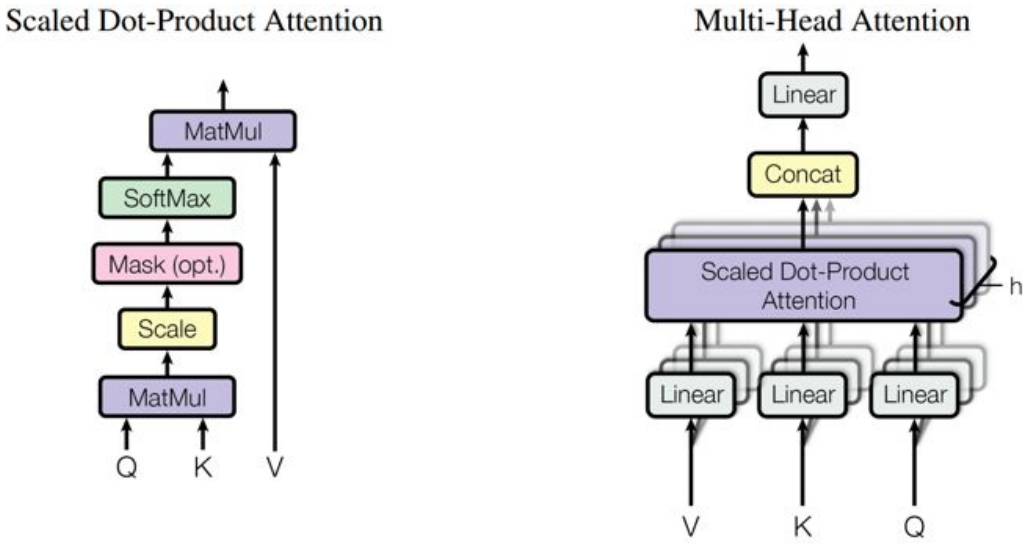


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

muti-head self-attention是单个self-attention的叠加，除了后续归一化会受到数量影响，彼此可以认为是独立的。类似CNN的多个channel。因此我们可以只看上面左边的图，Q ,K和V我感觉只是一个说法，并没有实际上qurey, key本身的物理意义。

对着上面图可以实现autoint的attention层：

```
querys = tf.tensordot(embeddings, W_Query, axes=(-1, 0))  # bs * fs * (as * head_num)
keys = tf.tensordot(embeddings, W_key, axes=(-1, 0))  # bs * fs * (as * head_num)
values = tf.tensordot(embeddings, W_Value, axes=(-1, 0))  # bs * fs * (as * head_num)

querys = tf.stack(tf.split(querys, head_num, axis=2))  # head_num * bs  * fs * as
keys = tf.stack(tf.split(keys, head_num, axis=2))  # head_num * bs  * fs * as
values = tf.stack(tf.split(values, head_num, axis=2))  # head_num * bs  * fs * as

inner_product = tf.matmul(querys, keys, transpose_b=True)  # head_num * bs * fs * fs
normalized_att_scores = tf.nn.softmax(inner_product)  # head_num * bs * fs * fs

result = tf.matmul(normalized_att_scores, values)  # head_num * bs * fs * as

result = tf.concat(tf.split(result, head_num, ), axis=-1)  # 1 *
result = tf.squeeze(result, axis=0)  # bs * fs * (as * head_num)
```


调包使用：

```
from deeprec.ranking.ctr import Autoint
```

完整的例子详见github