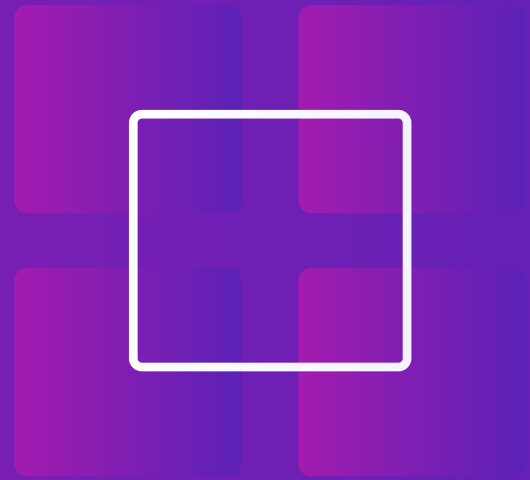
Feature Generation by Convolutional Neural Network for Click-Through Rate Prediction

WWW'19,华为诺亚方舟实验室

乔梁 2022.11.11



一、摘要

CTR

`背景`：在许多真实世界的广告和排名系统中，点击率 (CTR) 预测起着核心作用。

`之前的方法`：该领域中提出了许多模型，例如LR，基于树的模型，FM和基于深度学习的CTR模型。但是，当前的许多方法都以简单的方式计算特征交互，例如Hadamard乘积和内积，并且不太关心特征的重要性。

`本文方法`:本文提出了一种名为FiBiNET的新模型，用于动态学习特征重要性和细粒度特征交互。一方面，FiBiNET可以通过挤压激励网络 (SENET) 机制动态学习特征的重要性; 另一方面，它能够通过双线性函数有效地学习特征相互作用。

我们在两个真实世界数据集上进行了广泛的实验，结果表明，我们的浅层模型优于其他浅层模型，例如因式分解机 (FM) 和模糊感知因式分解机 (FFM)。为了进一步提高性能，我们将经典的深度神经网络 (DNN) 组件与浅层模型相结合，成为深度模型。deep FiBiNET始终优于其他最先进的deep模型，例如DeepFM和xdeepFM。

二、引言

本文贡献

文献综述略。

本文提出了一种名为FiBiNET的新模型，用于动态学习特征重要性和细粒度特征交互。众所周知，不同的特征对目标任务有不同的重要性。例如，当我们预测一个人的收入时，特征职业比特征爱好更重要。考虑到这一点，我们引入了挤压激励网络 (SENET) 来动态学习特征的权重。此外，特征交互是CTR预测领域的一个关键挑战，许多相关工作以简单的方式计算特征交互，如哈达玛积和内积。本文提出了一种新的细粒度方法来计算与双线性函数的特征相互作用。主要贡献如下:

- 受SENET在计算机视觉领域的成功启发，使用SENET机制来动态学习特征的权重。
- 引入了三种类型的双线性交互层，以细粒度的方式学习特征交互。
- 将SENET机制与双线性特征交互相结合，我们的浅层模型在诸如Criteo和Avazu数据集上的FFM之类的浅层模型中实现了最优结果。
- 为了进一步提高性能，将DNN组件与浅层模型结合在一起成为深度模型。deep FiBiNET始终优于Criteo和Avazu数据集上的其他最先进的deep模型。

三、相关工作

FM及其变体

略

基于深度学习的CTR模型

略

SENET模块

挤压和激励网络 (SENET)，通过在各种图像分类任务中明确建模卷积特征通道之间的相互依赖性来提高网络的表示能力。事实证明，SENET在图像分类任务中是成功的，并在ILSVRC 2017分类任务中获得了第一名。

四、本文模型

图1：本文提出的方法框架

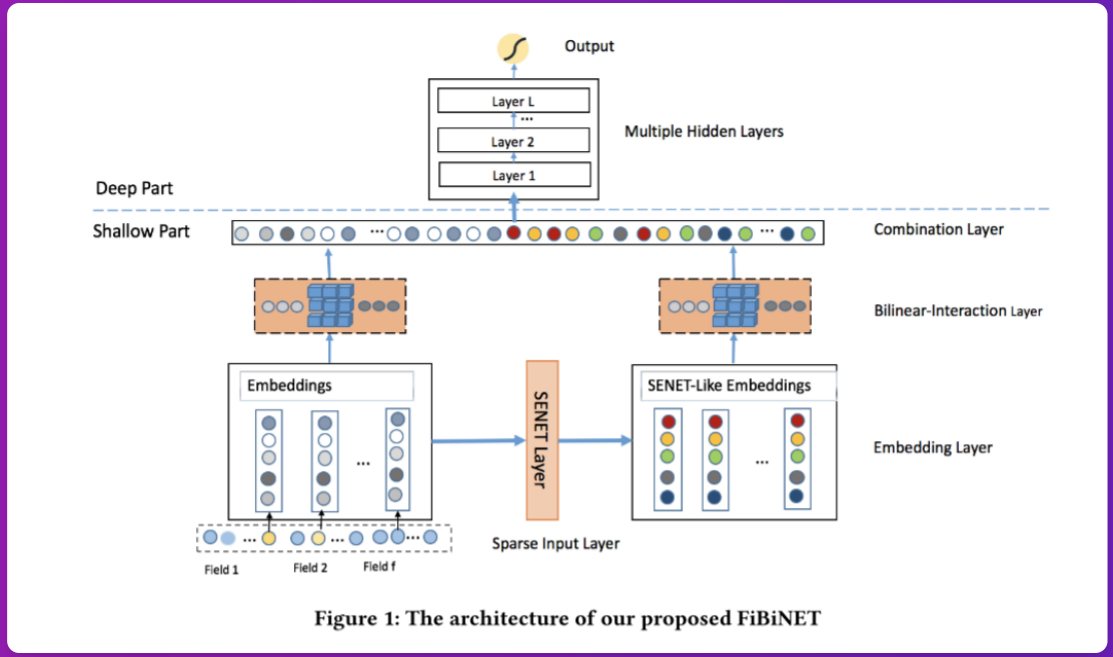


Figure 1: The architecture of our proposed FiBiNET

图2: SENET层

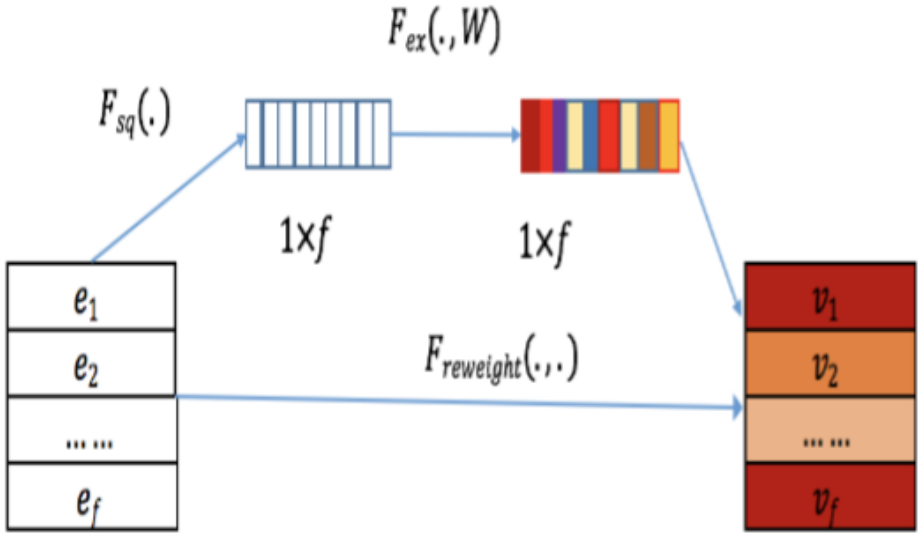


Figure 2: The SENET Layer

图3: 不同的特征交互计算方法

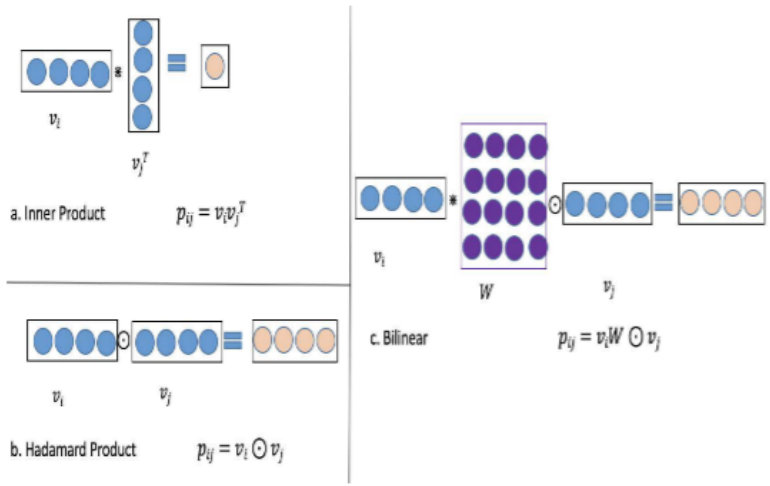


Figure 3: The different methods to calculate the feature interactions. (a): Inner product. (b): Hadamard product. (c): our proposed bilinear interaction. Here p_{ij} in inner product method is a scalar while it is a vector in Hadamard product and our proposed bilinear function.

稀疏输入和嵌入层

稀疏输入层对原始输入特征采用稀疏表示。嵌入层能够将稀疏特征嵌入到低维、密集的实值向量中。嵌入层的输出是 $E = [e_1, e_2, \cdots, e_i, \cdots, e_f]$，其中$f$表示域的数量，$e_i \in \mathbb{R}^k$表示第$i$个域的嵌入，$k$是嵌入层的维度。

SENET层

不同的特征对目标任务有不同的重要性。例如，当我们预测一个人的收入时，职业比爱好更重要。受SENET在计算机视觉领域的成功启发，我们引入了SENET机制，以使模型更加关注特征重要性。对于特定的CTR预测任务，我们可以通过SENET机制动态增加重要特征的权重并降低无信息特征的权重。

使用特征嵌入作为输入，SENET为嵌入生成权重向量$A = \{a_1, \cdots, a_i, \cdots, a_f\}$，然后将原始嵌入$E$与向量$A$重新缩放以获得新的嵌入 (类似SENET的嵌入) $V = [v_1, \cdots, v_i, \cdots, v_f]$，其中$a_i \in R$是表示第$i$个嵌入$v_i$的权重的标量，$v_i \in \mathbb{R}^k$表示第$i$个域的类似SENET的嵌入，$i \in [1, 2, \cdots, f]$，$V \in \mathbb{R}^{f \times k}$，$k$是嵌入大小，$f$是域的数量。

如图2所示，SENET由三个步骤组成: 挤压、激励和重加权。

SENET层

`挤压`。此步骤用于计算每个嵌入的“汇总统计”。具体来说，我们使用一些池化方法，例如max或mean，将原始嵌入$E = [e_1, e_2, \cdots, e_f]$挤压成统计向量$Z = [z_1, z_2, \cdots, z_i, \cdots, z_f]$，其中$i \in [1, 2, \cdots, f]$，$z_i$是一个标量值，它表示关于第$i$个特征表示的全局信息。$z_i$可以通过以下公式计算:

$$z_i = F_{sq}(e_i)$$

函数F_{sq}可以通过以下方法实现:

$$z_i = F_{sq}\left(e_i\right) = \frac{1}{k}\sum_{t=1}^{k} e_i^{(t)}$$
$$z_i = F_{sq}\left(e_i\right) = \sum_{t=1}^{k} e_i^{(t)}$$
$$z_i = F_{sq}\left(e_i\right) = \max_{1 \le t \le k}\left(e_i^{(t)}\right)$$

原始SENET论文中的挤压函数是最大池化。但是，我们的实验结果表明，平均池化的性能优于最大池化。

SENET层

`激励`。该步骤可用于基于统计向量Z来学习每个嵌入的权重。我们使用两个全连接 (FC) 层来学习权重。第一个FC层是具有参数W_1的降维层，其降低率r是超参数。第二个FC层随参数W_2增加维数。嵌入的权重可以计算如下:

$$A = F_{ex}(Z) = \sigma_2\left(W_2\sigma_1\left(W_1 Z\right)\right)$$

其中$A \in \mathbb{R}^f$是向量，σ_1和σ_2是激活函数，学习参数为$W_1 \in \mathbb{R}^{f\times\frac{f}{r}}$，$W_2 \in \mathbb{R}^{\frac{f}{r}\times f}$，$r$是降低率。

SENET层

`重加权`。SENET中的最后一步是重新加权步骤，在原始论文中称为重新缩放。它在原始嵌入E和权重向量A之间进行乘法，并输出新的嵌入 (类似SENET的嵌入) $V=[v_1，\ \cdots，\ v_i，\ \cdots，\ v_f]$。类似SENET的嵌入V可以计算如下:

$$V=F_{\text{ReWeight}}\left(A,E\right)=[a_1\cdot e_1,\cdots,a_f\cdot e_f]=[v_1,\cdots,v_f]$$

其中，$a_i\in\mathbb{R}$，$e_i\in\mathbb{R}^k$，$v_i\in\mathbb{R}^k$。

SENET使用两个FCs来动态学习功能的重要性。对于特定任务，它会增加重要特征的权重，并降低无信息特征的权重。

核心代码

```
class SqueezeExcitationLayer(nn.Module):
    def __init__(self, num_fields, reduction_ratio=3):
        super(SqueezeExcitationLayer, self).__init__()
        reduced_size = max(1, int(num_fields / reduction_ratio))
        self.excitation = nn.Sequential(nn.Linear(num_fields, reduced_size, bias=False),
                                        nn.ReLU(),
                                        nn.Linear(reduced_size, num_fields, bias=False),
                                        nn.ReLU())

    def forward(self, feature_emb):
        Z = torch.mean(feature_emb, dim=-1, out=None)
        A = self.excitation(Z)
        V = feature_emb * A.unsqueeze(-1)
        return V
```

双线性相互作用层

交互层是计算二阶特征交互的层。交互层中特征交互的经典方法是内积和哈达玛积。内积广泛用于FM和FFM等浅层模型，而Hadamard积通常用于AFM和NFM等深层模型。内积和Hadamard积的形式分别表示为 $\{(v_i \cdot v_j)\, x_i x_j\}_{(i,j)\in R_x}$ 和 $\{(v_i \odot v_j)\, x_i x_j\}_{(i,j)\in R_x}$，其中$R_x = \{(i,j)\}_{i\in\{1,\cdots,f\},j\in\{1,\cdots,f\},j>i}$，$v_i$是第$i$个嵌入向量，·表示内积，$\odot $表示哈达玛积。

交互层中的内积和哈达玛积过于简单，无法对稀疏数据集中的特征交互进行有效建模。因此，我们提出了一种更细粒度的方法，该方法结合了内积和哈达玛积来学习具有额外参数的特征交互。

如图3.C所示，在矩阵W和向量v_i之间使用内积，在矩阵W和向量v_j之间使用哈达玛积。具体地，我们在此层中提出了三种类型的双线性函数，并将此层称为双线性交互层。

双线性相互作用层

以第i个嵌入v_i和第j个嵌入v_j为例，特征交互作用的结果p_{ij}可以计算如下:

$$p_{ij} = v_i \cdot W \odot v_j$$

其中$W \in \mathbb{R}^{k \times k}$，$v_i$，$v_j \in \mathbb{R}^k$是第$i$个和第$j$个嵌入，$1 \leq i \leq f, i \leq j \leq f$。这里$W$在所有 (v_i, v_j) 交互对之间共享，并且双线性交互层中有$k \times k$个参数，因此在这里我们将这种类型称为 “Field-All Type”。

双线性相互作用层

$$p_{ij} = v_i \cdot W_i \odot v_j$$

其中$W_i \in \mathbb{R}^{k \times k}$，$v_i$，$v_j \in \mathbb{R}^k$是第$i$个和第$j$个嵌入，$1 \leq i \leq f, i \leq j \leq f$。这里$W_i$是第$i$个域的相应参数矩阵，并且在双线性相互作用层中有$f \times k \times k$参数，因为我们有$f$个不同的域，因此在这里我们将这种类型称为 “Field-Each'”。

双线性相互作用层

$$p_{ij} = v_i \cdot W_{ij} \odot v_j$$

其中$W_{ij} \in \mathbb{R}^{k \times k}$，$v_i$，$v_j \in \mathbb{R}^k$是第$i$个和第$j$个嵌入之间相互作用的对应参数矩阵，$1 \leq i \leq f, i \leq j \leq f$。该层中学习参数的总数为$n \times k \times k$，$n$是相互作用的数量，等于$\frac{f(f-1)}{2}$。在这里，我们将这种类型称为 “Field-Interaction Type”。

双线性相互作用层

如图1所示，我们有两个嵌入 (原始嵌入和类似SENET的嵌入)，并且我们可以采用双线性函数或Hadamard乘积作为任何嵌入的特征交互操作。所以我们在这一层有几个特征交互的组合。

双线性相互作用层可以从原始嵌入E输出一个相互作用向量$p = [p_1, \ldots, p_i, \ldots, p_n]$，从类似SENET的嵌入$V$输出一个类似SENET的相互作用向量$q = [q_1, \ldots, q_i, \ldots, q_n]$，其中$p_i$， $q_i \in \mathbb{R}^k$是向量。

核心代码

```
class BilinearInteractionLayer(nn.Module):
    def __init__(self, num_fields, embedding_dim, bilinear_type="field_interaction"):
        super(BilinearInteractionLayer, self).__init__()
        self.bilinear_type = bilinear_type
        if self.bilinear_type == "field_all":
            self.bilinear_layer = nn.Linear(embedding_dim, embedding_dim, bias=False)
        elif self.bilinear_type == "field_each":
            self.bilinear_layer = nn.ModuleList([nn.Linear(embedding_dim, embedding_dim, bias=False)
                                                 for i in range(num_fields)])
        elif self.bilinear_type == "field_interaction":
            self.bilinear_layer = nn.ModuleList([nn.Linear(embedding_dim, embedding_dim, bias=False)
                                                 for i, j in combinations(range(num_fields), 2)])
        else:
            raise NotImplementedError()
```

核心代码

```
    def forward(self, feature_emb):
        feature_emb_list = torch.split(feature_emb, 1, dim=1)
        if self.bilinear_type == "field_all":
            bilinear_list = [self.bilinear_layer(v_i) * v_j
                             for v_i, v_j in combinations(feature_emb_list, 2)]
        elif self.bilinear_type == "field_each":
            bilinear_list = [self.bilinear_layer[i](feature_emb_list[i]) * feature_emb_list[j]
                             for i, j in combinations(range(len(feature_emb_list)), 2)]
        elif self.bilinear_type == "field_interaction":
            bilinear_list = [self.bilinear_layer[i](v[0]) * v[1]
                             for i, v in enumerate(combinations(feature_emb_list, 2))]
        return torch.cat(bilinear_list, dim=1)
```

组合层

组合层将相互作用向量p和q连接起来，并将连接的向量馈送到FiBiNET中的以下层中，该层是标准神经网络层。可以表示为以下形式:

$$c = F_{\text{concat}}(p, q) = [p_1, \cdots, p_n, q_1, \cdots, q_n] = [c_1, \cdots, c_{2n}]$$

如果我们将向量c中的每个元素求和，然后使用sigmoid函数输出预测值，则我们有一个浅层CTR模型。为了进一步提高性能，我们将浅层组件和经典的深度神经网络 (DNN) 结合到一个统一模型中，以形成深度网络结构，该统一模型在本文中称为深度模型。

深度网络

深层网络由几个全连接层组成，这些层隐式捕获高阶特征交互。如图1所示，深度网络的输入是组合层的输出。令$a^{(0)} = [c_1, c_2, \cdots, c_{2n}]$表示组合层的输出，其中$c_i \in \mathbb{R}^k$，$n$是域相互作用的数量。然后，将$a^{(0)}$送入深度神经网络，前馈过程为:

$$a^{(l)} = \sigma\left(W^{(l)} a^{(l-1)} + b^{(l)}\right)$$

其中l是深度，σ是激活函数。$W^{(l)}$，$b^{(l)}$，$a^{(l)}$是第l层的模型权重，偏差和输出。之后，生成密集的实值特征向量，该向量被馈入sigmoid函数以进行CTR预测: $y_d = \sigma\left(W^{|L|+1} a^{|L|} + b^{|L|+1}\right)$，其中$|L|$是DNN的深度。

输出层

我们给出了我们提出的模型输出的总体公式为:

$$\hat{y} = \sigma\left(w_0 + \sum_{i=0}^{m} w_i x_i + y_d\right)$$

其中$y \in (0,1)$ 为CTR的预测值，σ 为sigmoid函数，m为特征大小，x为输入，w_i为线性部分的第i个权重。我们模型的参数为 $\theta = \left\{w_0, \{w_i\}_{i=1}^{m}, \{e_i\}_{i=1}^{m}, \{W_i\}_{i=1}^{2}, \left\{W^{(i)}\right\}_{i=1}^{|L|}\right\}$。学习过程旨在最小化以下目标函数 (交叉熵):

$$\text{loss} = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log\left(\hat{y}_i\right) + \left(1 - y_i\right) * \log\left(1 - \hat{y}_i\right)\right)$$

五、结论

结论

由于最先进模型的缺点，我们提出了一种名为FiBiNET的新模型，旨在动态学习特征重要性和细粒度特征交互。我们提出的FiBiNET在以下几个方面为提高性能做出了贡献:

1. 对于CTR任务，SENET模块可以动态学习特征的重要性。它提高了重要特征的权重，抑制了不重要特征的权重。
2. 我们引入三种类型的双线性交互层来学习特征交互，而不是计算与Hadamard乘积或内积的特征交互。
3. 在我们的浅模型中将SENET机制与双线性特征交互相结合，优于其他浅模型，例如FM和FFM。
4. 为了进一步提高性能，我们将经典的深度神经网络 (DNN) 组件与浅模型相结合，成为深度模型。deep FiBiNET始终优于DeepFM和XdeepFM等其他最先进的deep模型。

六、学习收获

学习收获

- 如何进行细粒度的特征交互；
- 通过引入新的网络结构进行创新。

Thank you