《OptEmbed: Learning Optimal Embedding Table for Click-through Rate Prediction》复现报告

原论文作者：Fuyuan Lyu　Xing Tang　Hong Zhu　Huifeng Guo
Yingxue Zhang　Ruiming Tang　Xue Liu

摘要

在通用 CTR 预测模型架构中，嵌入表是最基本的组件，嵌入表的学习决定着 CTR 模型的性能和内存消耗。为了学习一个高效的嵌入表，最近的工作都是要么减少特征的嵌入，要么为特征分配不同的嵌入维数，没有将特征选择和嵌入维数选择两者统一起来，因此得不到一个最佳嵌入表。基于这个问题，OptEmbed 模型为了获得一个最佳嵌入表，提出了可学习的剪枝阈值，其根据相应特征的重要性来裁剪冗余嵌入，以达到特征选择的效果。此外，又提出了一种基于超网的进化搜索方法来寻找每个特征域的最优嵌入维数。OptEmbed 模型考虑到硬件友好的问题，将剪枝阈值和网络参数一起训练得到一个超网，接着基于超网使用进化算法搜索嵌入维数，以实现特征选择和嵌入维数选择的解耦。我基于开源代码对 OptEmbed 进行了复现，实验表明了 OptEmbed 可以学习到一个紧凑的嵌入表，从而提高模型性能。然而，OptEmbed 只考虑了单域场景的特征选择问题，直接把 OptEmbed 应用在多域场景中做特征选择无疑会低效且无法很好的区分一些域专属的特征。受 OptEmbed 模型的启发，我针对多域场景的特征选择问题，设计了一个双层特征选择学习框架 dualMaskFS，并做了一些 case study 实验验证了 dualMaskFS 存在的必要性和有效性。

关键词：CTR 预测；神经网络结构搜索；特征选择

1　引言

嵌入表本质上是一个二维张量，它的轴分别表示特征值的数量和嵌入维数，由于特征值的数量远远大于嵌入维数，因此特征值的数量决定着嵌入表占用内存的大小。如果把所有可能的特征值都输入到模型中时，这会导致产生大量的嵌入，造成训练和推理中的主要内存瓶颈，而且冗余嵌入不仅需要额外的内存成本，还对模型性能有害。因此，优化嵌入表的第一个方法是进行特征选择，即识别出冗余的特征嵌入然后过滤掉，确保仅有益的特征嵌入输入到交互层。此外，主流的 CTR 模型中的嵌入维数几乎是固定的，但先前的一些研究工作就已经指出，信息量小的特征用较大的嵌入维数来学习表示会导致过度学习，而信息量大的特征需要更大的维数来学习丰富的信息。因此，优化嵌入表的第二个方法是给特征灵活分配嵌入维数。

为了学习一个高效的嵌入表，最近的工作都是要么减少特征的嵌入，要么为特征分配不同的嵌入维数，没有将特征选择和嵌入维数选择两者统一起来。基于这个问题，OptEmbed 模型为了获得一个最佳嵌入表，提出了可学习的剪枝阈值，其根据相应特征的重要性来裁剪冗余嵌入，以达到特征选择的效果。此外，又提出了一种基于超网的进化搜索方法来寻找每个特征域的最优嵌入维数。考虑到硬件友好的问题，OptEmbed 模型先将剪枝阈值和网络参数一起训练得到一个超网，接着基于超网通过进化算法搜索嵌入维数，以实现特征选择和嵌入维数选择的解耦。虽然 OptEmbed 能获得一个紧凑的

嵌入表来提升模型性能，但针对 OptEmbed 只考虑了单域场景的特征选择问题，OptEmbed 应用在多域场景中做特征选择可能由于无法很好的区分一些域专属的特征而造成模型性能的损失。因此，受 OptEmbed 模型启发，我针对多域场景的特征选择问题，设计一个双层特征选择学习框架 dualMaskFS。其中，第一层为域共享的特征选择层，第二层为域专属的特征选择层。域共享的特征选择层可以通过学习来选择对所有域都有价值的特征，而在这一层中一些对某个域有价值的特征很可能会被过滤掉，因此设计域专属的特征选择层的目的在于可以将域共享的特征选择层过滤掉但对专属域有价值的特征重新找回。

2 相关工作

2.1 CTR 模型

主流的 CTR 预测研究以深度学习模型为主。其中，Wide&Deep、DNN、和 FNN 在嵌入表的基础上引入 MLP 层来学习输入特征的高级表示，而 DeepFM、DCN 和 IPNN 则依靠各种特征交互层来提高性能。目前，在 CTR 模型应用 AutoML 技术的研究越来越多，上述提到的 CTR 模型常常充当骨架网络的作用。在 OptEmbed 模型中，使用了 DeepFM、DCN、FNN 和 IPNN 模型作为骨架网络进行实验，而在我提出的 dualMaskFS 模型中，则使用 DNN、DeepFM 和 DCN 作为骨架网络进行实验。

2.2 嵌入表优化

CTR 模型中嵌入表优化研究主要可以分为特征选择和搜索嵌入维数。特征选择常通过学习特征权重来决定选择重要的特征或学习一个二值化特征掩码来过滤冗余特征。AdaFS 设计一个 MLP 的控制器网络来实现特征权重的学习，接着再根据设定的阈值选择权值大的特征，而 LPFS[1]则学习一个二值化特征掩码来过滤冗余特征。为了搜索嵌入维数，NAS 可以在基于明确定义的搜索空间中自动搜索，DARTS[2]可以预先设置若干的候选值，然后学习候选值的权重来选择。AutoEmb[3]和 AutoDim[4]都是应用了 DARTS 技术来实现嵌入维数的搜索。此外，通过剪枝的方法也可以达到搜索嵌入维数的效果。PEP 设计了一种软阈值方法，以预定的稀疏率过滤掉嵌入表中的低于阈值的值。

3 本文方法

3.1 问题定义

3.1.1 CTR 预测

在 CTR 预测任务中，原始的输入表示为原始的特征向量，这些特征向量由 n 个特征域 $\mathbf{x}=[\mathbf{x}_{(1)},\mathbf{x}_{(2)},...,\mathbf{x}_{(n)}]$ 拼接而成。通常，$\mathrm{x}_{(i)}$ 是一个非常稀疏且高维的独热向量表示。在将原始的特征向量输入到特征交叉层之前，通常需要通过嵌入表将原始特征向量映射为低维且稠密的实数向量。这可以表述为 $\mathbf{e}_{(i)}=\mathbf{E}\times\mathbf{x}_{(i)}$，$1\leqslant i\leqslant n$，其中 $\mathbf{E}\in\mathbb{R}^{|f|\times D}$ 是嵌入表，$|f|$ 是特征的数量，D 是嵌入维数。接着这些嵌入被堆叠在一起作为嵌入向量 $\mathbf{e}=[\mathbf{e}_{(1)},\mathbf{e}_{(2)},...,\mathbf{e}_{(n)}]$。

学习到嵌入表之后，主流 CTR 模型会基于 **e** 来训练特征交互层。基于嵌入的特征交叉可以定义为：

$$\mathbf{v}^{p}=o^{(p-1)}(o^{(p-2)}(\cdots(o^{(1)}(\mathbf{e}))\cdots)), \tag{1}$$

其中，o 可以是单层感知器或交叉层。接着，这些特征交叉可以聚合在一起：

$$\hat{y} = \sigma\left(\mathbf{w}^T\left(\mathbf{v}^{(1)} \oplus \mathbf{v}^{(2)} \oplus \cdots \oplus \mathbf{v}^{(n)}\right) + b\right) = \mathcal{F}(\mathbf{E} \times \mathbf{x} \mid \mathbf{W}), \tag{2}$$

其中，符合 $\oplus$ 表示连接操作，$\mathbf{v}^{(k)}$ 是特征交叉的输出，$\mathbf{W}$ 是除了嵌入表以外的网络参数。模型训练时，采用的是交叉熵损失函数：

$$\mathrm{CE}(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}). \tag{3}$$

最终 CTR 预测问题总结如下：

$$\min_{\mathbf{E},\mathbf{W}} \mathcal{L}_{\mathrm{CE}}(\mathcal{D} \mid \{\mathbf{E}, \mathbf{W}\}) = -\frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},y)\in\mathcal{D}} \mathrm{CE}(y, \mathcal{F}(\mathbf{E} \times \mathbf{x} \mid \mathbf{W})). \tag{4}$$

其中，y 是真实的用户点击，$\mathcal{D}$ 是训练集。

3.1.2 最优嵌入表

原始的嵌入表要么效果不好，要么效率不高。最佳嵌入表需要满足以下的要求来压缩模型大小并提高性能：（1）没有冗余的嵌入。（2）嵌入维数是灵活的。（3）对硬件友好。

为了满足以上三个要求，本文将原始单一的嵌入表 $\mathbf{E}$ 分解为一系列的 field-wise 嵌入表 $\mathbf{E}^* = \left[\mathbf{E}_{(1)}, \mathbf{E}_{(2)}, \cdots, \mathbf{E}_{(n)}\right]$，其中 $\mathbf{E}_{(i)} \in \mathbb{R}^{|f_{(i)}| \times D_{(i)}}$。一个最优嵌入表可以再定义为：

$$\begin{aligned} &\min_{\mathrm{E}^*,\mathbf{W}} \mathcal{L}_{\mathrm{CE}}\left(\mathcal{D} \mid \{\mathbf{E}^*, \mathbf{W}\}\right), \mathbf{E}^* = \left[\mathbf{E}_{(1)}, \mathbf{E}_{(2)}, \cdots, \mathbf{E}_{(n)}\right], \\ &\text{s.t. } \mathbf{E}_{(i)} \in \mathbb{R}^{|f_{(i)}| \times D_{(i)}}, \sum_{i=1}^{n} \left|f_{(i)}\right| \leqslant |f|, D_{(i)} \leqslant D, \forall i \leqslant n. \end{aligned} \tag{5}$$

3.2 OPTEMBED

本文提出了一个名为 OptEmbed 的框架来学习上一小节定义的最优嵌入表 $\mathbf{E}^*$，接着引入两个掩码来改写公式 (5) 为：

$$\min_{\mathbf{m}_e,\mathbf{m}_d,\mathbf{E},\mathbf{W}} \mathcal{L}_{\mathrm{CE}}\left(\mathcal{D} \mid \{\mathbf{E}^*, \mathbf{W}\}\right), \mathbf{E}^* = \mathbf{E} \odot \mathbf{m}_e \odot \mathbf{m}_d. \tag{6}$$

其中，$\mathbf{m}_d \in \{0,1\}^{D \times n}$ 表示 field-wise 维数掩码，$\mathbf{m}_e \in \{0,1\}^{|f|}$ 表示嵌入掩码，$|f|$ 和 n 分别表示特征数量和特征域数量，$\odot$ 表示可以广播的元素乘积。OptEmbed 框架的概述如图 1所示：

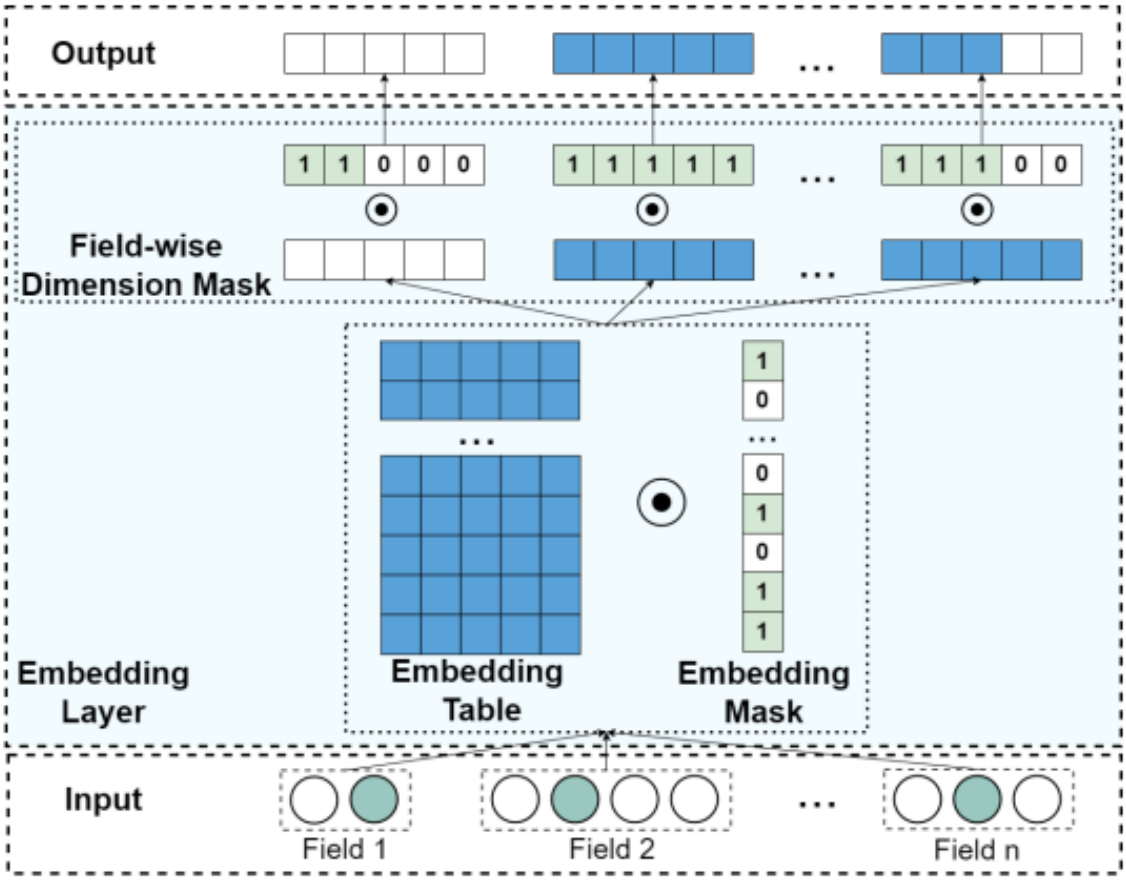

图 1: OptEmbed 模型概述【来源：原论文】

3.2.1 冗余嵌入修剪

冗余嵌入修剪组件决定嵌入表的哪些行是有信息作用的，并且应该被保留到最终的预测任务中。受网络修剪的启发，本文直接优化嵌入表 $\mathbf{E}$ 和自适应的通过对比 field-wise 阈值来修剪嵌入，而且这个 field-wise 阈值是可以通过梯度来更新学习的。$\mathbf{m}_e$ 的重参数化可以表述如下：

$$\mathbf{m}_e = S\left(L_\beta(\mathbf{E}) - \mathbf{t}\right), \tag{7}$$

其中，$\mathbf{t}$ 是 field-wise 阈值向量，L_β 是每个域嵌入的 β 范数，$S(\cdot)$ 是激活函数，所以 $\mathbf{m}_e$ 是一个可学习的动态掩码。

为了生成二值的掩码，本文引入了一个单位阶跃函数 $S(x)$ 作为激活函数。对给定的 field-wise 阈值向量 $\mathbf{t}$ 和单位阶跃函数 $S(x)$，可以生成嵌入的掩码 $\mathbf{m}_e$。对于特征 j 对应的嵌入 $\mathbf{e}^j$，它的嵌入掩码 $\mathbf{m}_e^j$ 可以按照如下计算：

$$\mathbf{m}_e^j = S\left(L_\beta\left(\mathbf{e}^j\right) - \mathbf{t}^{k_j}\right) = \begin{cases} 1, L_\beta\left(\mathbf{e}^j\right) - \mathbf{t}^{k_j} > 0 \\ 0, \text{ otherwise} \end{cases}, \tag{8}$$

其中，$L_\beta(\cdot)$ 是 L_β 归一化函数，k_j 表示映射特征 j 到相应的域中。通过单位阶跃函数 $S(x)$，可以容易生成二值嵌入掩码 $\mathbf{m}_e$。因此，嵌入表可以公式化为：

$$\hat{\mathbf{E}} = \mathbf{E} \odot \mathbf{m}_e = \mathbf{E} \odot S\left(L_\beta(\mathbf{E}) - \mathbf{t}\right). \tag{9}$$

然而，由于单位阶跃函数的导数是一个脉冲函数，所以公式 (9) 不能直接用来优化。为了保持梯度并使模型可训练，本文采用长尾求导估计器来代替单位阶跃函数的梯度 $dS(x)/dx$。长尾求导估计

器可以公式化为：

$$\frac{d}{dx}S(x) \approx H(x) = \begin{cases} 2-4|x|, & |x| \leqslant 0.4 \\ 0.4, & 0.4 < |x| \leqslant 1 \\ 0, & \text{otherwise} \end{cases}. \tag{10}$$

本文采用导数长尾求导估计器来优化 field-wise 阈值向量 $\mathbf{t}$，以便当 L_β 范数和阈值彼此接近时，梯度较大，而当 L_β 范数和阈值之间的差距足够大时，梯度为 0。本文将实际嵌入 $\hat{\mathbf{E}}$ 的梯度表示为 $d\hat{\mathbf{E}}$，更新 $\mathbf{E}$ 的嵌入梯度为：

$$d\mathbf{E} = d\hat{\mathbf{E}} \odot \mathbf{m}_e + d\hat{\mathbf{E}} \odot \mathbf{E} \odot H\left(L_\beta(\mathbf{E}) - \mathbf{t}\right) \odot dL_\beta(\mathbf{E}). \tag{11}$$

嵌入的梯度由两部分组成，第一部分 $d\hat{\mathbf{E}} \odot \mathbf{m}_e$ 是提高性能的性能梯度，第二部分 $d\hat{\mathbf{E}} \odot \mathbf{E} \odot H\left(L_\beta(\mathbf{E}) - \mathbf{t}\right) \odot dL_\beta(\mathbf{E})$ 是用来移除冗余嵌入的结构梯度。

为了移除更多的冗余嵌入，可以学习更大的阈值。因此，本文将指数正则化项 L_s 添加到 logloss 中，以惩罚低阈值。对于 field-wise 阈值 $\mathbf{t} \in \mathbb{R}^n$，指数正则化项为：

$$\mathcal{L}_s = \sum_{i=1}^{n} \exp\left(-t_i\right). \tag{12}$$

注意，正则化项随着 t_i 增大而逐渐减小为 0。因此，这一阶段的最终目标成为：

$$\min_{\mathbf{m}_e, \mathrm{E}, \mathbf{W}} \mathcal{L}_{\mathrm{CE}}(\mathcal{D} \mid \{\hat{\mathbf{E}}, \mathbf{W}\}) + \alpha \mathcal{L}_s, \hat{\mathbf{E}} = \mathbf{E} \odot \mathbf{m}_e. \tag{13}$$

其中，α 是稀疏正则化项的缩放系数，控制修剪多少嵌入。α 越大，$\mathcal{L}_s$ 往往会增大阈值 $\mathbf{t}$，从而更容易实现对冗余嵌入的修剪。

3.2.2 嵌入维数搜索

嵌入维数搜索组件旨在为所有域分配各种最佳维数。通过将一组 field-wise 维数掩码视为一个神经网络体系结构，然后设计了一种有效的神经体系结构搜索方法来有效地搜索最佳维数掩码。

因为最优嵌入表需要满足灵活的嵌入维数和对硬件友好，此方法将所有候选嵌入维数形成的维数集可以形式化为 $\mathcal{S}_e = \{1, \cdots, D-1, D\}$。因此，为了有效地搜索最佳嵌入维数掩码，将维数搜索重新定义为 one-shot NAS 问题：

$$\begin{aligned} \mathbf{m}_d^* &= \arg\min_{\mathbf{m}_d \in \mathcal{S}_e} \mathcal{L}_{\mathrm{CE}}\left(\mathcal{D}_{\mathrm{val}} \mid \left\{\hat{\mathbf{E}}_s \odot \mathbf{m}_d, \hat{\mathbf{W}}_s\right\}\right), \\ \text{s.t. } \left\{\hat{\mathbf{E}}_s, \hat{\mathbf{W}}_s\right\} &= \arg\min_{\{\mathbf{E}_s, \mathbf{W}_s\} \in \Omega} \mathbb{E}_{\mathbf{m}_d \sim \Gamma(\mathcal{S}_e)} \mathcal{L}_{\mathrm{CE}}\left(\mathcal{D} \mid \{\mathbf{E}_s \odot \mathbf{m}_d, \mathbf{W}_s\}\right), \end{aligned} \tag{14}$$

其中，$\mathcal{S}_e$ 表示搜索空间，$\Gamma\left(\mathcal{S}_e\right)$ 是搜索空间的先验分布，$\left\{\hat{\mathbf{E}}_s, \hat{\mathbf{W}}_s\right\}$ 是最优的超网参数，Ω 则表示超网的参数空间。通过解耦训练嵌入和维数搜索之间的依赖关系，不再需要从头开始训练子体系结构，从而大大降低了计算成本。

为了训练适应 $\mathbf{m}_e$ 的超网并进一步减少总训练时间，通过引入 $\mathbf{E}_s = \mathbf{E} \odot \mathbf{m}_e$ 以统一的方式进行超网训练和冗余嵌入修剪。最后，可以把超网训练表述为：

$$\begin{aligned} &\min_{\mathbf{m}_e, \mathrm{E}, \mathbf{W}} \mathbb{E}_{\mathbf{m}_d \sim \text{ Uniform }\left(\mathcal{S}_e\right)} \mathcal{L}_{\mathrm{CE}}(\mathcal{D} \mid \{\hat{\mathbf{E}}, \mathbf{W}\}) + \alpha \mathcal{L}_s, \\ &\hat{\mathbf{E}} = \mathbf{E}_s \odot \mathbf{m}_d = \mathbf{E} \odot \mathbf{m}_e \odot \mathbf{m}_d. \end{aligned} \tag{15}$$

在通过公式 (15) 训练超网 $\{\mathbf{E}_s^*, \mathbf{W}_s^*\}$ 后，本文提出了对最佳维数掩码 $\mathbf{m}_d^*$ 的进化搜索. 初始时，所

有候选都是随机生成的。在每次迭代中，每个候选维数掩码 $\mathbf{m}_d$ 通过继承超网的参数在验证集 $\mathcal{D}_{val}$ 上进行评估。这部分相对来说是有效的，因为不涉及训练。评估后，保留 Top-K 个候选者以进行交叉和突变操作，以生成下一次迭代的候选者。对于交叉，通过选择父母的交换部分的随机点，将两个随机选择的候选者交叉以产生新的候选者。在 T 次迭代之后，将性能最佳的候选维数掩码 $\mathbf{m}_d^*$ 输出为最佳维数掩码。

3.2.3 参数再训练

在超网的训练过程中，已经将所有原始特征映射到嵌入表中了。因此，为了消除这些嵌入的影响，需要一个仅使用最优的嵌入数和嵌入维数的“再训练阶段”来训练模型。通过公式 (15) 获得的嵌入掩码 $\mathbf{m}_e^*$ 和 field-wise 维数掩码 $\mathbf{m}_d^*$，在再训练阶段，优化目标可以表述为：

$$\operatorname{argmin}_{\mathrm{E},\mathbf{W}} \mathcal{L}_{\mathrm{CE}}\left(\mathcal{D} \mid \{\mathbf{E} \odot \mathbf{m}_e^* \odot \mathbf{m}_d^*, \mathbf{W}\}\right). \tag{16}$$

4 复现与改进

4.1 复现工作

本次复现是参考了论文作者开源的代码基础上完成的。在复现的过程中，自己依旧投入很多精力来完成论文复现的实验，主要做了两个工作。第一个工作是直接对原论文直接进行复现，而第二个工作是受原论文的启发，对原论文进行创新的尝试。

在这小节中，先介绍我第一个工作的内容。虽然是对原论文直接进行复现，但也不是很简单就实现。其中，我需要准备实验所需的环境、阅读代码来理解算法和对论文实验所需的数据集进行预处理等等，最后在实验室的服务器上进行实验并记录实验结果。在实验过程中，发现了使用开源代码跑出实验结果与论文中的实验结果有明显的差异，因此发现作者开源的代码是有问题的。后续和论文作者交流讨论了这个问题，得到了作者的肯定，作者对他开源代码做了一些修改。在使用修改后的代码进行实验时，得到的实验结果虽然还是没有和论文中的实验结果一致，不过已经可以验证论文中提出的 OptEmbed 框架的有效性。

其中，OptEmbed 算法的伪代码如下：

Procedure 1 The OptEmbed Algorithm

Input: training dataset $\mathcal{D}$, validation dataset $\mathcal{D}_{val}$
Output: optimal embedding table table $\mathbf{E}^*$ and model parameters $\mathbf{W}^*$
Supernet Training and Embedding Pruning
while *not converge* **do**
  Sample a mini-batch from the training dataset
  $\left\{\hat{\mathbf{E}}_s, \hat{\mathbf{W}}_s\right\}$, $\mathbf{m}_e$ = SupernetTrain($\mathcal{D}$)
end
$\mathbf{m}_e^*$ = GetBestPerform($\mathbf{m}_e$)
Dimension Mask Searching
$\tau = 0$; P_τ = Initialize_population($n_m + n_c$); Topk = 0
while $\tau < T$ **do**
  AUC$_\tau$ = Inference($\hat{\mathbf{E}}_s, \hat{\mathbf{W}}_s, \mathcal{D}_{val}, P_\tau$);
  Topk = Update_Topk(Topk,P_τ, AUC$_\tau$);
  P_τ^c = Crossover(Topk, n_c);
  P_τ^m = Mutation(Topk, $n_m, prob$);
  $P_{\tau+1} = P_\tau^m \cup P_\tau^c$;
  $\tau = \tau + 1$;
end
$\mathbf{m}_d^*$ = GetBestCand(P_τ)
Re-training
Retrain $\{\mathbf{E}_s^*, \mathbf{W}_s^*\}$ given $\mathbf{m}_e^*$ and $\mathbf{m}_d^*$

4.2 实验环境

表 1: 实验环境

名称	参数
操作系统	Linux 3.10.0
CPU	Intel(R) Xeon(R) Gold 6226R
GPU	NVIDIA TESLA V100S
CUDA	11.0
python	3.7.13
Pytorch	1.7.0
Tensorflow	2.4.1

4.3 创新工作

4.3.1 创新点

原始的 OptEmbed 只考虑了单域的场景来进行特征选择，直接把 OptEmbed 应用在多域场景中做特征选择无疑会低效且无法很好的区分一些域专属的特征。由于目前缺少针对多域场景的特征选择的 CTR 模型研究，但受 OptEmbed 模型启发，我针对多域场景的特征选择问题，设计一个双层特征选择模块。其中，第一层为域共享的特征选择层，第二层为域专属的特征选择层。域共享的特征选择层可以通过学习来选择对所有域都有价值的特征，而在这一层中一些对某个域有价值的特征很可能会被过滤掉，因此设计域专属的特征选择层的目的在于可以将域共享的特征选择层过滤掉但对专属域有价值的特征重新找回。模型框架的概述如图 2所示：

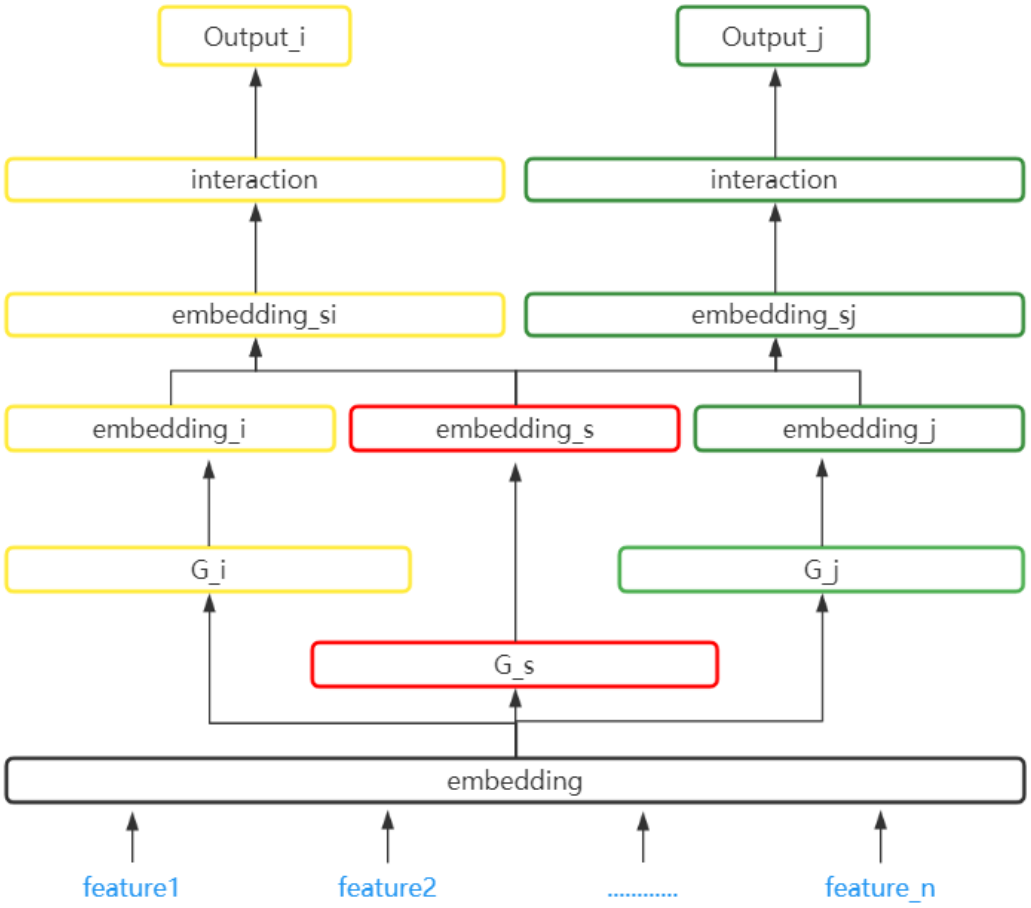

图 2: dualMaskFS 模型概述

直观上，图 2的 dualMaskFS 模型一开始是学习一个嵌入表和三个特征选择掩码。三个掩码的功能各不相同，G_s 掩码是域共享的，用于选择两个域共享的特征，G_i 是 i 域专属的掩码，用于选择 i 域专属的特征，G_j 是 j 域专属的掩码，用于选择 j 域专属的特征。初始的嵌入表通过三个掩码过滤得到三组特征嵌入，最终模型将不同域专属的特征嵌入和域共享的特征嵌入拼接得到不同的特征子集嵌入，这些嵌入即可用于后续的特征交叉和输出预测。

4.3.2 特征选择门控掩码

需要说明一下，这部分实验使用到的特征选择掩码的逻辑实现并非是 OptEmbed 模型中的特征选择模块，而是参考 LPFS 模型[1]实现的特征选择门控掩码模块 MaskFS，它们两者的原理是一致，因此并不失一般性。因为 MaskFS 实现起来更简单并且网络容易训练，所以更方便用于进行我们的实验。

MaskFS 学习到特征门控掩码是特征级粒度的，而不是特征域级粒度的。特征选择表述为每个特征嵌入 $\mathbf{e}_{k_i}$ 分配一个二值门控掩码 $\mathbf{g}_{k_i} \in \{0,1\}$。在选择后，特征嵌入可以表述如下：

$$\mathbf{e}_{k_i}^{\mathbf{g}} = \mathbf{g}_{k_i} \odot \mathbf{e}_{k_i} = \mathbf{g}_{k_i} \odot (\mathbf{E} \times \mathbf{x}_{k_i}) . \tag{17}$$

其中，当 $\mathbf{g}_{k_i} = 1$ 表示特征 $\mathbf{e}_{k_i}$ 被选择了。

为了有效地优化具有特征级粒度的的特征集，MaskFS 引入了连续门 $\mathbf{g}_c \in \mathbb{R}^m$。在门控掩码学习阶段，又引入了一个指数增加的温度值 τ，以近似 L_0 归一化。具体地说，实际的门控掩码 $\mathbf{g}$ 的计算公

式为:

$$\mathbf{g} = \frac{\sigma\left(\mathbf{g}_c \times \tau\right)}{\sigma\left(\mathbf{g}_c^{(0)}\right)}, \tau = \gamma^{t/T}. \tag{18}$$

其中，$\mathbf{g}_c^{(0)}$ 是连续门 $\mathbf{g}_c$ 初始值，σ 是 sigmoid 函数 $\sigma(x) = \frac{1}{1+e^{-x}}$，$t$ 为当前训练迭代次数，T 为总训练迭代次数，γ 为迭代训练 T 次后 τ 的终值。

4.3.3 Case Study 的数据集

由于改进工作针对的是多域场景，OptEmbed 复现实验中的数据集是单域场景的数据集，因此需要多域场景的数据集。最后，Case Study 实验采用的是阿里云天池中公共的速卖通（AliExpress）搜索系统数据集，这是从全球速卖通搜索系统的真实世界流量日志中收集的数据集。作为全球最大的电子商务平台之一，全球速卖通为 200 多个国家提供商品搜索服务。在速卖通电子商务平台中的一个搜索会话中，用户首先从搜索结果页面点击一个产品，然后决定是否购买该产品。因此该数据集可以用于学习排序 (LTR) 问题的研究。这个数据集是从 5 个国家收集的: 俄罗斯 (ru)、西班牙 (es)、法国 (fr)、荷兰 (nl) 和美国 (us)，这可以看作是 5 个场景。这是第一个针对多场景学习排序问题的大规模真实数据集，而在本次的 Case Study 中，我采用的是其中荷兰（nl）和法国 (fr) 的数据集。

4.3.4 Case Study

（1）单一特征门控掩码不能很好适用于多域场景。（2）多域场景中单一特征门控掩码选择的特征绝大多数是域共享的特征。（3）双层特征选择学习域共享门和域专属门的有效性。

5 实验结果分析

5.1 OptEmbed 复现结果

表 2: OptEmbed 在两个数据集上复现的结果

Dateset			DeepFM			DCN			FNN			IPNN		
			AUC	Logloss	Sparsity	AUC	Logloss	Sparsity	AUC	Logloss	Sparsity	AUC	Logloss	Sparsity
Avazu	Original	论文结果	0.7884	0.3751	-	0.7894	0.3748	-	0.7896	0.3748	-	0.7898	0.3745	-
		复现结果	**0.7860**	**0.3755**	-	**0.7886**	**0.3755**	-	**0.7889**	**0.3752**	-	**0.7891**	**0.3749**	-
	OptEmbed	论文结果	0.7888	0.3750	0.3927	0.7901	0.3740	0.6840	0.7902	0.3744	0.5563	0.7902	0.3740	0.4693
		复现结果	**0.7883**	**0.3751**	**0.4593**	**0.7907**	**0.3735**	**0.7439**	**0.7914**	**0.3730**	**0.6853**	**0.7906**	**0.3735**	**0.5569**
KDD12	Original	论文结果	0.7962	0.1532	-	0.8010	0.1522	-	0.8008	0.1522	-	0.8007	0.1522	-
		复现结果	**0.7970**	**0.1529**	-	**0.8010**	**0.1521**	-	**0.8009**	**0.1521**	-	**0.8009**	**0.1521**	-
	OptEmbed	论文结果	0.7971	0.1530	0.6183	0.8021	0.1519	0.4715	0.8027	0.1522	0.5105	0.8028	0.1521	0.4154
		复现结果	**0.7972**	**0.1528**	**0.5365**	**0.8031**	**0.1516**	**0.2953**	**0.8018**	**0.1519**	**0.5510**	**0.8020**	**0.1519**	**0.4069**

1: 加粗字体表示是复现的结果。

观察表 2中的复现结果可以发现，复现的实验结果和论文的实验结果并没有完全一致，不过大多数结果在三个模型评估指标（AUC、Logloss 和 Sparsity）上差距很小，此外复现中 OptEmbed 在两个数据集四个骨架模型的三个评估指标中结果都要优于 Original，这可以说明 OptEmbed 模型的有效性，也验证了此次复现的结果是可信的。至于为什么复现结果和论文结果不一致的问题，很可能是复现实验时的数据预处理和论文作者的数据预处理是有细微差异的，因此造成了最后实验结果的差异。

5.2 改进工作中 Case Study 实验结果

5.2.1 单一特征门控掩码不能很好适用于多域场景

表 3: 单一特征门控掩码在三个数据集上实验的结果

	lr	l_2	Original			MaskFS		
			AUC					
			nl	fr	nlfr	nl	fr	nlfr
DeepFM	0.0001	3.00E-05	0.7288	0.7199	0.7197	0.7311	0.7231	0.7225
	0.0001	1.00E-05	0.7344	0.7266	0.7261	0.7332	0.7287	**0.7278**
	0.0001	3.00E-06	0.7373	0.7303	0.7321	0.7293	**0.7293**	0.7265
	0.0001	1.00E-06	0.7381	0.7317	0.7314	0.7246	0.7221	0.7225
	3.00E-05	3.00E-05	0.7191	0.7186	0.7165	**0.7348**	0.7278	0.7239
DCN	0.0001	3.00E-05	0.7275	0.7219	0.7198	0.7299	0.7208	0.7229
	0.0001	1.00E-05	0.7340	0.7180	0.7243	0.7318	**0.7284**	**0.7276**
	0.0001	3.00E-06	0.7372	0.7299	0.7311	0.7283	0.7271	0.7244
	0.0001	1.00E-06	0.7381	0.7315	0.7306	0.7231	0.7234	0.7230
	3.00E-05	3.00E-05	0.7134	0.7151	0.7177	**0.7348**	0.7278	0.7233
DNN	0.0001	3.00E-05	0.7282	0.7206	0.7197	0.7305	0.7219	0.7206
	0.0001	1.00E-05	0.7340	0.7259	0.7259	0.7316	0.7289	**0.7264**
	0.0001	3.00E-06	0.7371	0.7301	0.7303	0.7288	**0.7286**	0.7257
	0.0001	1.00E-06	0.7381	0.7316	0.7299	0.7231	0.7245	0.7250
	3.00E-05	3.00E-05	0.7168	0.7178	0.7168	**0.7343**	0.7274	0.7230

1：lr 是模型的学习率，l_2 是模型的 L2 正则项系数。
2：nl 表示速卖通中荷兰的数据集，fr 表示法国的数据集，nlfr 则表示把 nl 和 fr 合起来的数据集。
3：加粗表示 MaskFS 在同一数据集同一骨架模型中 AUC 的结果最优。

认真观察表 3中的实验结果可以得到两个结论。第一，有特征选择功能的 MaskFS 在三个数据集和三个骨架模型的 AUC 多数好于不做特征选择（Original）的 AUC，特别是加粗的实验结果更是明显好于 Original 的，说明三个数据集很多的冗余特征以及 MaskFS 的特征门控掩码可以有效的做端到端的特征选择。第二，MaskFS 在单域数据集（nl、fr）的实验结果模型明显要优于多域数据集（nlfr）的实验结果，即使 nlfr 数据集中加粗的实验结果都低于同一骨架模型同一 lr 和 l_2 在单域上的实验结果，这说明单一特征门控掩码不能很好适用于在多域场景做特征选择。

5.2.2 多域场景中单一特征门控掩码选择的特征绝大多数是域共享的特征

表 4: 单一特征门控掩码在三个数据集上的特征选择

	数据集	原始特征数	剩余特征数	nl&fr	nl&fr&nlfr
DeepFM	nl	13446993	8878412	5852405	4465063
	fr	13952446	6769356		
	nlfr	14838184	5706477		
DCN	nl	13446993	9018798	5728597	2888544
	fr	13952446	6589705		
	nlfr	14838184	3611155		
DNN	nl	13446993	8842532	5604743	2988542
	fr	13952446	6418535		
	nlfr	14838184	3727142		

1：实验结果是 MaskFS 在 lr = 3E-05，l_2 = 3E-05 特征选择的结果。
2：nl&fr 表示对 nl 和 fr 被选择的特征取交集。
3：nl&fr&nlfr 表示对 nl、fr 和 nlfr 被选择的特征取交集。

观察表 4中的实验结果可以发现，单一特征门控掩码选择的特征大部分是域共享的特征，共享的特征数为 80% 左右，而剩下的部分特征应该属于域专属的。因此多域数据集 nlfr 使用单一特征门控掩码做特征选择得到的特征是将域共享特征和域专属特征混在一起的结果，这也是解释了为什么 nlfr 在 MaskFS 上的 AUC 不如 nl 和 fr 的 AUC。

5.2.3 双层特征选择学习域共享门和域专属门的有效性

基于前两个实验的发现的单一特征门控掩码并不适合应用在多域场景的问题，我们结合多域场景的特性和特征门控掩码的特点，设计了如图 2的 dualMaskFS 模型。对于 dualMaskFS 模型，我们期望的效果：第一，G_s 能学习到选择域共享的特征，G_i、G_j 能学习到选择域专属的特征，然后三个门彼此之间选择的特征重叠很小。第二，模型在每个域的性能表现比仅采用单一特征门控掩码 MaskFS 的性能表现要好。针对 dualMaskFS 模型想要达到的两个效果，我们在 nlfr 数据集、骨架模型 DNN、lr = 3E-05、l_2 = 3E-05 的设置下做了一些实验。遗憾的是，由于时间问题目前只完成了第一个效果的实验。

表 5: dualMaskFS 模型实验采用的一些学习策略

学习策略	reg1_s	reg1_i	reg1_j	reg2_si	reg2_sj	reg2_ij	G 的初始值	G_s 与 G_i 或 G_j 拼接方式
策略 1	√	√	√	√	√	×	1	直接相加
策略 2	√	√	√	×	×	√	1	直接相加
策略 3	√	√	√	×	×	√	0.5	阈值掩码相加

1：reg1_* 表示稀疏正则，例如 reg1_s 表示 G_s 的稀疏正则。
2：reg2_* 表示正交正则，例如 reg2_si 表示 G_s 和 G_i 的正交正则。
3：G 的初始值表示 G_s、G_i 和 G_j 三个门的初始值。
4：阈值掩码相加表示引入 G_s $\geqslant$ 0.5 的二值掩码 S，然后 G_s * S + G_i * (1 - S)，G_s * S + G_j * (1 - S)。

表 6: dualMaskFS 采用学习策略 1 的一些实验结果

lambda1	lambda2	original	G_s	G_i	G_j	G_s&G_i	G_s&G_j	G_i&G_j
1.00E-08	1.00E-07	14838184	929209	2064505	1644215	382033	610900	85252
5.00E-09	1.00E-07	14838184	1021077	2492326	2071138	425522	661773	104970
1.00E-09	1.00E-04	14838184	3	102	26	2	1	0
1.00E-09	1.00E-05	14838184	48	7194	2716	30	23	6
1.00E-09	5.00E-07	14838184	73277	426095	529379	27632	28456	5505
1.00E-09	1.00E-06	14838184	11290	167196	196718	4523	4615	1752

1：lambda1 是稀疏正则系数，lambda2 是正交正则系数。
2：original 表示原始特征数，G_s 表示 G_s 门选择的特征数，其他两个同理。
3：G_s&G_i 表示两个两个门选择的特征取交集，其他两个同理。

表 7: dualMaskFS 采用学习策略 2 的一些实验结果

lambda1_s	lambda1_i	lambda1_j	lambda2	original	G_s	G_i	G_j	G_s&G_i	G_s&G_j	G_i&G_j
1.00E-09	1.00E-09	1.00E-09	1.00E-08	14838184	14838180	3237848	4033325	3237848	4033325	509083
1.00E-09	1.00E-09	1.00E-09	1.00E-07	14838184	14838183	715793	1108831	715793	1108831	2279
1.00E-08	1.00E-08	1.00E-08	1.00E-07	14838184	4160000	471632	801938	471632	801938	1972
5.00E-08	1.00E-08	1.00E-08	1.00E-07	14838184	796516	574986	891862	218747	566648	4749
1.00E-08	1.00E-09	1.00E-09	1.00E-07	14838184	4102932	720400	1105768	720237	1105629	2437
1.00E-08	1.00E-09	1.00E-09	1.00E-08	14838184	3711510	3206947	3978790	1841204	2369109	518384
3.00E-08	1.00E-09	1.00E-09	1.00E-08	14838184	1172499	3375248	4015774	664112	898303	587296
5.00E-08	1.00E-09	1.00E-09	1.00E-08	14838184	607446	3504284	4098631	371842	520808	655971
1.00E-07	1.00E-09	1.00E-09	1.00E-08	14838184	210288	3662451	4193574	165174	182686	756948
5.00E-07	1.00E-09	1.00E-09	1.00E-08	14838184	7140	3892866	4460619	5665	6140	934615

1：lambda1_s、lambda1_i 和 lambda1_j 分别是 G_s、G_i 和 G_j 的稀疏正则系数。

表 8: dualMaskFS 采用学习策略 3 的一些实验结果

lambda1_s	lambda1_i	lambda1_j	lambda2	original	G_s	G_i	G_j	G_s&G_i	G_s&G_j	G_i&G_j	AUC_i	AUC_j
1.00E-08	1.00E-09	1.00E-09	1.00E-08	14838184	2319772	1400310	1222787	**61**	**22**	**8122**	0.7252	0.7137
5.00E-09	1.00E-09	1.00E-09	1.00E-08	14838184	6256490	61	22	61	22	0	0.7255	0.7137
1.00E-09	1.00E-09	1.00E-09	1.00E-08	14838184	14838182	55	19	55	19	0	0.7255	0.7139
5.00E-09	5.00E-10	5.00E-10	1.00E-08	14838184	6258883	147	82	147	82	0	0.7255	0.7136
1.00E-08	5.00E-09	5.00E-09	1.00E-08	14838184	2298577	591154	231231	**1**	**1**	**123**	0.7260	0.7112

1：加粗表示达到模型想要的效果。
2：AUC_i 表示 i 域的 AUC，AUC_j 表示 j 域的 AUC。

从表 6和表 7的实验结果可以发现，学习策略 1 和学习策略 2 都不能将域共享的特征和域专属的特征区分开，说明学习策略 1 和学习策略 2 是错误的。但一个有趣的现象是学习策略 2 因为设置有 G_i 和 G_j 的正交正则项 reg2_ij，所以能将两个专属域的特征分开是合理的，而学习策略 1 并没有 reg2_ij，但是也能将两个专属域的特征分开，这是很奇怪的。经过分析，导致这种奇怪的现象的原因是，学习策略 1 的 G_s 与 G_i 或 G_j 拼接方式是直接相加的且初始值为 1, 让模型在训练时 G_s、G_i、G_j 三个门控掩码的学习是耦合的，很难优化。因此，我们需要对 G_s、G_i、G_j 三个门控掩码的学习进行解耦，让 G_s 只学域共享的特征门控掩码，G_i、G_j 只学域专属的的特征门控掩码，而这就是最后采用的学习策略 3。

从表 8的实验结果来看，学习策略 3 已经可以满足 dualMaskFS 模型期望的第一个效果：G_s 能学习到选择域共享的特征，G_i、G_j 能学习到选择域专属的特征，然后三个门彼此之间选择的特征重叠很小。不过 AUC_i 和 AUC_j(i 域，j 域对应 nlfr 数据集中 nl 和 fr) 的值还不太理想，因此还需要进一步的调整一些参数来优化模型。

6 总结与展望

在本次论文复现中，我主要做了两个工作。工作一，我在 OptEmbed 的开源代码基础上，对 OptEmbed 模型在两个公共数据集上进行了复现，复现实验的结果真实可靠。工作二，我对 OptEmbed 模型的特征选择功能进行了扩展，考虑了多域场景的特征选择问题，提出了一个 dualMaskFS 模型，并做了一些 case study 来验证 dualMaskFS 模型存在的必要性和有效性。遗憾的是，dualMaskFS 模型的有效性暂时不能充分验证，因此在未来的工作中需要继续改进。此外，对 OptEmbed 模型的嵌入维数选择功能，也可以在未来扩展到多域场景中。

参考文献

[1] GUO Y, LIU Z, TAN J, et al. LPFS: Learnable Polarizing Feature Selection for Click-Through Rate Prediction[J]., 2022.

[2] LIU H, SIMONYAN K, YANG Y. DARTS: Differentiable Architecture Search[J]., 2018.

[3] ZHAO X, WANG C, CHEN M, et al. AutoEmb: Automated Embedding Dimensionality Search in Streaming Recommendations[Z]. 2020.

[4] ZHAO X, LIU H, LIU H, et al. Memory-efficient Embedding for Recommendations[J]., 2020.