Optimizing Feature Set for Click-Through Rate Prediction

Fuyuan Lyu*
McGill University
Montreal, Canada
fuyuan.lyu@mail.mcgill.ca

Xing Tang*
FiT, Tencent
Shenzhen, China
shawntang@tencent.com

Dugang Liu†‡
Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ)
Shenzhen, China
dugang.ldg@gmail.com

Liang Chen
FiT, Tencent
Shenzhen, China
leocchen@tencent.com

Xiuqiang He‡
FiT, Tencent
Shenzhen, China
xiuqianghe@tencent.com

Xue Liu
McGill University
Montreal, Canada
xueliu@cs.mcgill.ca

ABSTRACT

Click-through prediction (CTR) models transform features into latent vectors and enumerate possible feature interactions to improve performance based on the input feature set. Therefore, when selecting an optimal feature set, we should consider the influence of both features and their interaction. However, most previous works focus on either feature field selection or only select feature interaction based on the fixed feature set to produce the feature set. The former restricts search space to the feature field, which is too coarse to determine subtle features. They also do not filter useless feature interactions, leading to higher computation costs and degraded model performance. The latter identifies useful feature interaction from all available features, resulting in many redundant features in the feature set. In this paper, we propose a novel method named OptFS to address these problems. To unify the selection of features and their interaction, we decompose the selection of each feature interaction into the selection of two correlated features. Such a decomposition makes the model end-to-end trainable given various feature interaction operations. By adopting feature-level search space, we set a learnable gate to determine whether each feature should be within the feature set. Because of the large-scale search space, we develop a learning-by-continuation training scheme to learn such gates. Hence, OptFS generates the feature set containing features that improve the final prediction results. Experimentally, we evaluate OptFS on three public datasets, demonstrating OptFS can optimize feature sets which enhance the model performance and further reduce both the storage and computational cost.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; *Online advertising*.

*Both authors contributed equally to this research.
†This work was done when working at FiT, Tencent.
‡Corresponding authors

KEYWORDS

CTR Prediction, Feature Set, Feature Interaction

ACM Reference Format:
Fuyuan Lyu, Xing Tang, Dugang Liu, Liang Chen, Xiuqiang He, and Xue Liu. 2023. Optimizing Feature Set for Click-Through Rate Prediction. In *Proceedings of the ACM Web Conference 2023 (WWW '23), April 30–May 04, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3543507.3583545

1 INTRODUCTION

Click-through rate prediction has been a crucial task in real-world commercial recommender systems and online advertising systems. It aims to predict the probability of a certain user clicking a recommended item (e.g. movie, advertisement) [2, 27]. The standard input for CTR prediction consists mainly of a large set of categorical features organized as feature fields. For example, every sample contains a feature field *gender* in CTR prediction, and the field *gender* may include three feature values, *male*, *female* and *unknown*. To avoid ambiguity, we term feature values as features hereafter. General CTR prediction models first map each feature in the feature set into a unique real-valued dense vector through embedding table [21]. Then these vectors are fed into the feature interaction layer to improve the prediction by explicitly modelling low-order feature interaction by enumerating feature set [22]. The final prediction of the classifier is made upon the feature embedding and feature interactions, which are both heavily influenced by the input feature set. The general framework is shown in Figure 1. Hence, the input feature set plays an important role in CTR prediction.

Blindly inputting all available features into the feature set is neither effective nor efficient. From the view of effectiveness, certain features can be detrimental to model performance. Firstly, these features themselves may only introduce extra learnable parameters, making the prediction model prone to overfitting [1, 9]. Secondly, certain useless interactions introduced by these features also bring unnecessary noise and complicate the training process [14], which degrades the final prediction. Notice that these two factors are closely related when selecting the feature set. If one feature $\mathbf{x}_i$ is filtered out from the set, all its related interactions $\langle \mathbf{x}_i, \cdot \rangle$ should be excluded in the model as well. Correspondingly, informative interactions $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ is a strong indicator to keep $\mathbf{x}_i$ in the feature set [19]. From the view of efficiency, introducing redundant features into a feature set can be inefficient in both storage space and computation cost. As the embedding table dominates the number of parameters in CTR models [6], a feature set without redundant features will greatly decrease the size of the models. Moreover, a feature set with useful features can zero out the computation of many useless feature interaction, which greatly reduce the computation cost in practice. An optimal feature set should keep features considering both effectiveness and efficiency.

Efforts have been made to search for an optimal feature set from two aspects. Firstly, Several methods produce the feature set based on feature selection. Because of the large-scale CTR dataset, some methods [8, 30, 32] focus on the field level, which results in hundreds of fields instead of millions of features. However, the field level is too coarse to find an optimal feature set. For instance, the feature field *ID* contains user/item feature *id* in real datasets. The *id* of certain cold users/items might be excluded from the feature set due to the sparsity problem [29], which is difficult to handle at the field level. Besides, these methods [8, 13] fail to leverage the influence of feature interaction, which is commonly considered an enhancement for the model performance [20, 38]. Secondly, there is also some weakness concerning feature interaction methods, which implicitly produce the feature set. On the one hand, some feature interaction selection methods [12, 14, 20], inspired by the ideas of neural architecture search [15, 18], tend to work on a fixed subset of input feature set, which commonly includes the redundant features. On the other hand, some method [19] constructs a locally optimal feature set to generate feature interaction in separated stages, which requires many handcraft rules to guide the search scheme. Given that many operations of feature interactions are proposed [7, 24, 31], searching an optimal feature set with these operations in a unified way can reduce useless feature interaction. As discussed, optimizing the feature set incorporated with the selection of both feature and feature interaction is required.

In this paper, we propose a method, **Opt**imizing **F**eature **S**et (OptFS), to address the problem of searching the optimal feature set. There are two main challenges for our OptFS. The first challenge is how to select the feature and its interaction jointly, given various feature interaction operations. As discussed above, an optimal feature set should exclude features that introduce useless interaction in models. We tackle this challenge by decomposing the selection of each feature interaction into the selection of two correlated features. Therefore, OptFS reduces the search space of feature interaction and trains the model end-to-end, given various feature interaction operations. The second challenge is the number of features in large-scale datasets. Notice that the possible number of features considered in our research could be 10^6, which is incredibly larger than 100 feature fields in previous works [8, 32]. To navigate in the large search space, we introduce a learnable gate for each feature and adopt the learning-by-continuation [16, 28, 36] training scheme. We summarize our major contributions as follows:

- This paper first distinguishes the optimal feature set problem, which focuses on the feature level and considers the effectiveness of both feature and feature interaction, improving the model performance and computation efficiency.
- We propose a novel method named OptFS that optimizes the feature set. Developing an efficient learning-by-continuation training scheme, OptFS leverages feature interaction operations trained together with the prediction model in an end-to-end manner.
- Extensive experiments are conducted on three large-scale public datasets. The experimental results demonstrate the effectiveness and efficiency of the proposed method.

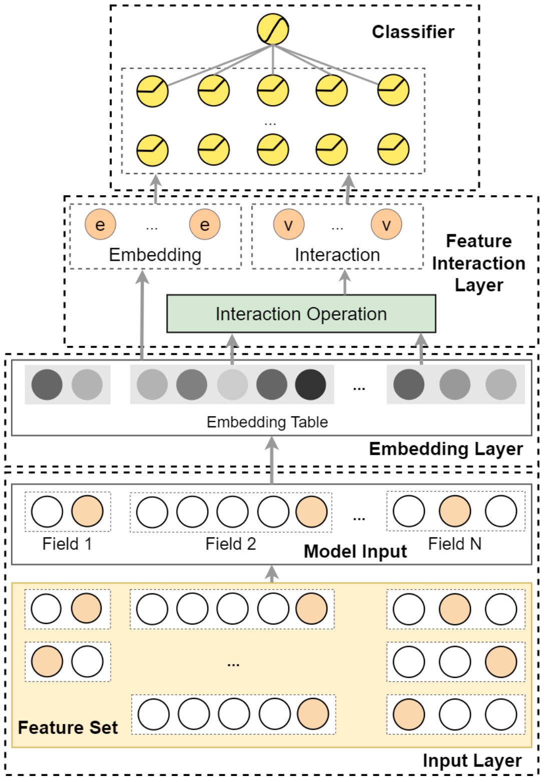


Figure 1: Overview of the general CTR framework.

We organize the rest of the paper as follows. In Section 2, we formulate the CTR prediction and feature selection problem and propose a simple but effective method OptFS. Section 3 details the experiments. In Section 4, we briefly introduce related works. Finally, we conclude this paper in Section 5.

2 OPTFS

In this section, we will first distinguish the feature set optimization problem in Section 2.1 and detail how OptFS conduct feature selection in Section 2.2. Then, we will illustrate how OptFS influences feature interaction selection in Section 2.3. Finally, we will illustrate the learning-by-continuation method in Section 2.4.

2.1 Problem Formulation

In this subsection, we provide a formulation of the feature set optimization problem. Usually, features that benefit the accurate prediction are considered useful in CTR models. In our setting, we represent all possible features as $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m\}$. $\mathbf{x}_i$ is a one-hot representation, which is very sparse and high-dimensional. As previously discussed, the feature set optimization problem aims to determine the useful features among all possible ones, which can be defined as finding an optimal feature set $\mathbf{X}^g \subset \mathbf{X}$. This can be formulated as follows:

$$\min_{\mathbf{W}} \mathcal{L}(\mathcal{D}|\mathbf{W}),\ \mathcal{D} = \{\mathbf{X^g}, \mathbf{Y}\},$$
$$s.t. \forall \mathbf{x} \in \mathbf{X^g}, \mathcal{L}(\mathbf{X^g}) > \mathcal{L}(\mathbf{X^g} - \{\mathbf{x}\}), \tag{1}$$
$$\forall \mathbf{x} \notin \mathbf{X^g}, \mathcal{L}(\mathbf{X^g}) \geq \mathcal{L}(\mathbf{X^g} + \{\mathbf{x}\}),$$

where $\mathcal{L}$ denotes the loss function, $\mathbf{W}$ denotes the model parameters, and $\mathbf{Y}$ denotes the corresponding labels.

2.2 Feature Selection

Each field $\mathbf{z}_i$ contains a proportion of all possible features, denoted as:

$$\mathbf{z}_i = \{\mathbf{x}_{k_i}\},\ 1 \leq k_i \leq m, \tag{2}$$

which indicates that the relationship between field and feature is a one-to-many mapping. In practice, the number of field n is much smaller than that of feature m. For instance, online advertisement systems usually have $n \leq 100$ and $m \approx 10^6$. So the input of CTR models can be rewritten as follows from both feature and field perspectives:

$$\mathbf{z} = [\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_n] = [\mathbf{x}_{k_1}, \mathbf{x}_{k_2}, \cdots, \mathbf{x}_{k_n}], \tag{3}$$

where the second equal sign means that for input $\mathbf{z}$, the corresponding feature for field $\mathbf{z}_i$ is $\mathbf{x}_{k_i}$ as shown in Equation 2.

We usually employ embedding tables to convert $\mathbf{z}_i$s into low-dimensional and dense real-value vectors. This can be formulated as $\mathbf{e}_i = \mathbf{E} \times \mathbf{z}_i = \mathbf{E} \times \mathbf{x}_{k_i}, 1 \leq i \leq n, 1 \leq k_i \leq m$, where $\mathbf{E} \in \mathbb{R}^{m \times D}$ is the embedding table, m is the number of feature values and D is the size of embedding. Then embeddings are stacked together as a embedding vector $\mathbf{e} = [\mathbf{e}_1, \mathbf{e}_2, \cdots, \mathbf{e}_n]$.

In our work, we propose feature-level selection. Instead of doing field-level selection, we formulate selection as assigning a binary gate $\mathbf{g}_{k_i} \in \{0, 1\}$ for each feature embedding $\mathbf{e}_{k_i}$. After selection, the feature embeddings can be formulated as follows:

$$\mathbf{e}^{\mathbf{g}}_{k_i} = \mathbf{g}_{k_i} \odot \mathbf{e}_{k_i} = \mathbf{g}_{k_i} \odot (\mathbf{E} \times \mathbf{x}_{k_i}). \tag{4}$$

When $\mathbf{g}_{k_i} = 1$, feature $\mathbf{x}_{k_i}$ is in the optimal feature set $\mathbf{X^g}$ and vice versa. Notice that previous work [8, 30, 32] assigns field-level feature selection. This means that $\mathbf{g}_{k_i} \equiv \mathbf{g}_i \in \{0, 1\}$ for each field z_i, indicating the keep or drop of all possible features $\{\mathbf{x}_{k_i}\}$ in corresponding field.

Then, these embeddings are stacked together as a feature-selected embedding vector $\mathbf{e^g} = [\mathbf{e}^{\mathbf{g}}_{k_1}, \mathbf{e}^{\mathbf{g}}_{k_2}, \cdots, \mathbf{e}^{\mathbf{g}}_{k_n}]$. The final prediction can be formulated as follows:

$$\hat{y} = \mathcal{F}(\mathbf{g} \odot \mathbf{E} \times \mathbf{x}|\mathbf{W}) = \mathcal{F}(\mathbf{E^g} \times \mathbf{x}|\mathbf{W}), \tag{5}$$

where $\mathbf{g} \in \{0, 1\}^m$ refers to gating vectors indicating whether certain feature is selected or not, $\mathbf{E^g} = \mathbf{g} \odot \mathbf{E}$ indicates the feature-selected embedding tables. The $\mathbf{E^g}$ can also be viewed as the feature set $\mathbf{X^g}$ after transformation from the embedding table, denoted as $\mathbf{E^g} = \mathbf{E} \times \mathbf{X^g}$.

2.3 Feature Interaction Selection

The feature interaction selection aims to select beneficial feature interaction for explicitly modelling [14, 20]. The feature interaction layer will be performed based on $\mathbf{e}$ in mainstream CTR models. There are several types of feature interaction in previous study [12], e.g. inner product [7]. The interaction between two features $\mathbf{e}_i$ and $\mathbf{e}_j$ can be generally represented as:

$$\mathbf{v}_{(i,j)} = \mathcal{O}(\mathbf{e}_i, \mathbf{e}_j), \tag{6}$$

where $\mathcal{O}$, as the interaction operation, can vary from a single layer perceptron to cross layer[31]. The feature interaction selection can be formulated as assigning $\mathbf{g}'_{(i,j)}$ for each feature interaction. All feature interactions can be aggregated together for final prediction:

$$\hat{y} = \mathcal{H}((\mathbf{g}' \odot \mathbf{v}) \oplus \mathcal{G}(\mathbf{e^g})) = \mathcal{H}(\mathbf{v}^{\mathbf{g}'} \oplus \mathcal{G}(\mathbf{e^g})), \tag{7}$$

where symbol $\oplus$ denotes the concatenation operation, $\mathcal{G}(\cdot)$ denotes the transformation function from embedding space to feature interaction space, such as MLP [7, 31] or null function [24]. $\mathcal{H}(\cdot)$ represents the prediction function. The combinations of $\mathcal{G}(\cdot)$, $\mathcal{O}(\cdot)$ and $\mathcal{H}(\cdot)$ in mainstream models are summarized in Table 1.

Table 1: Summary of $\mathcal{G}(\cdot)$, $\mathcal{O}(\cdot)$ and $\mathcal{H}(\cdot)$ in various models

Model	$\mathcal{G}(\cdot)$	$\mathcal{O}(\cdot)$	$\mathcal{H}(\cdot)$
FM [26]	null	inner product	null
DeepFM [7]	MLP	inner product	average
DCN [31]	MLP	cross network	average
IPNN [24]	null	inner product	MLP
OPNN [24]	null	outer product	MLP
PIN [25]	null	MLP	MLP

In reality, a direct way to explore all possible feature interaction is introducing a feature interaction matrix $\{\mathbf{g}'_{(k_i,k_j)}\}$ for 2nd-order feature interaction $\{\mathbf{x}_{k_i}, \mathbf{x}_{k_j}\}$. But it is impossible as we would have $C^2_m \approx 10^{12}$ gate variables. To efficiently narrow down such a large space, previous works [12, 14, 20] restrict the search space to feature field interaction, reducing the number of variables to $C^2_n \approx 1000$. This can be formulated as $\mathbf{g}'_{(i,j)} \equiv \mathbf{g}'_{(k_i,k_j)}$. However, such relaxation may not be able to distinguish the difference between useful and useless feature interaction within the same field. As it has been proven that informative interaction between features tends to come from the informative lower-order ones [34], we decompose the feature interaction as follows:

$$\mathbf{g}'_{(k_i,k_j)} = \mathbf{g}_{k_i} \times \mathbf{g}_{k_j}, \tag{8}$$

which indicates that the feature interaction is only deemed useful when both features are useful. An illustration of the decomposition is shown in Figure 2. Hence, the final prediction can be written as:

$$\hat{y} = \mathcal{H}((\mathbf{g} \times \mathbf{g} \odot \mathbf{v}) \oplus \mathcal{G}(\mathbf{g} \odot \mathbf{e})), \tag{9}$$

which means that the gating vector $\mathbf{g}$ that selects features can also select the feature interaction given $\mathcal{O}(\cdot)$. Such a design can reduce the search space and obtain the optimal feature set in an end-to-end manner.

2.4 Learning by Continuation

Even though the search space has been narrowed down from $C^2_m + m$ to m in Section 2.3, we still need to determine whether to keep or drop each feature in the feature set. This can be formulated as a ℓ_0 normalization problem. However, binary gate vector m is hard to compute valid gradient. Moreover, ℓ_0 optimization is

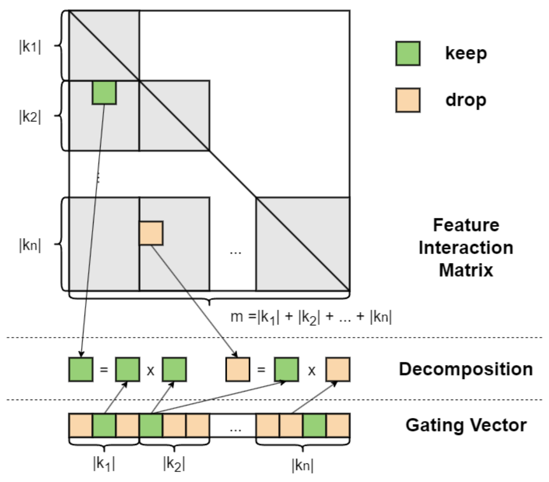


Figure 2: The Overview of OptFS.

known as a NP-hard problem [23]. To efficiently train the entire model, we introduce a learning-by-continuation training scheme. Such a training scheme has proven to be an efficient method for approximating ℓ_0 normalization [28], which correlates with our goal.

The learning-by-continuation training scheme consists of two parts: the searching stage that determines the gating vector $\mathbf{g}$ and the rewinding stage that determines the embedding table $\mathbf{e}$ and other parameters $\mathbf{W}$. We will introduce them separately in the following sections.

2.4.1 Searching. To efficiently optimize the feature set with feature-level granularity, we introduce a continual gate $\mathbf{g}_c \in \mathbb{R}^m$. During the searching stage, we introduce an exponentially-increased temperature value τ to approximate L_0 normalization. Specifically, the actual gate $\mathbf{g}$ is computed as:

$$\mathbf{g} = \frac{\sigma(\mathbf{g}_c \times \tau)}{\sigma(\mathbf{g}_c^{(0)})}, \ \tau = \gamma^{t/T} \tag{10}$$

where $\mathbf{g}_c^{(0)}$ is the initial value of the continual gate $\mathbf{g}_c$, σ is the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ applied element-wise, t is the current training epoch number, T is the total training epoch and γ is the final value of τ after training for T epochs. This would allow the continuous gating vector $\mathbf{g}_c$ to receive valid gradients in early stages yet increasingly approximate binary gate as the epoch number t grows. An illustration of Equation 10 is shown in Figure 3(a).

The final prediction is calculated based on Equation 9. The cross-entropy loss (i.e. log-loss) is adopted for each sample:

$$\text{CE}(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}), \tag{11}$$

where y is the ground truth of user clicks. We summarize the final accuracy loss as follows:

$$\mathcal{L}_{\text{CE}}(\mathcal{D}|\{\mathbf{E}, \mathbf{W}\}) = -\frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}} \text{CE}(y, \mathcal{F}(\mathbf{E} \times \mathbf{x}|\mathbf{W})), \tag{12}$$

where $\mathcal{D}$ is the training dataset and $\mathbf{W}$ is network parameters except the embedding table $\mathbf{E}$. Hence, the final training objective becomes:

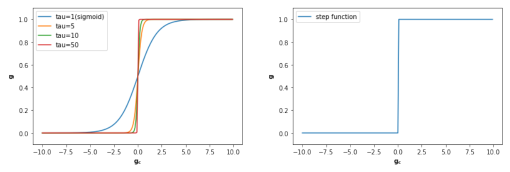
$$\min_{\mathbf{g}_c, \mathbf{E}, \mathbf{W}} \mathcal{L}_{\text{CE}}(\mathcal{D}|\{\mathbf{g}_c \odot \mathbf{E}, \mathbf{W}\}) + \lambda \|\mathbf{g}\|_1, \tag{13}$$

where λ is the regularization penalty, $\|\cdot\|_1$ indicates the ℓ_1 norm to encourage sparsity. Here we restate ℓ_0 norm to ℓ_1 norm given the fact that $\|\mathbf{g}\|_0 = \|\mathbf{g}\|_1$ for binary $\mathbf{g}$.

After training T epochs, the final gating vector $\mathbf{g}$ is calculated through a unit-step function as follows:

$$\mathbf{g} = \begin{cases} 0, & \mathbf{g}_c \leq 0 \\ 1, & \text{otherwise} \end{cases}. \tag{14}$$

Such a unit step function is also visualized in Figure 3(b).



(a) Searching Stage (b) Re-training Stage

Figure 3: Visualization of gating vector g during searching and retraining stages.

2.4.2 Retraining. In the searching stage, all possible features are fed into the model to explore the optimal feature set $\mathbf{X}^{\mathbf{g}}$. Thus, the useless features might hurt the model's performance. To address this problem, we need to retrain the model after obtaining the optimal feature set $\mathbf{X}^{\mathbf{g}}$.

After determining the gating vector $\mathbf{g}$, we retrain the model parameters $\mathbf{E}$ and $\mathbf{W}$ as the corresponding values at T_c epoch, which is carefully tuned in our setting. This is because most CTR models early stopped in several epochs, making them more sensitive towards initialization and prone to overfitting [37]. The final parameters $\mathbf{E}$ and $\mathbf{W}$ are trained as follows:

$$\min_{\mathbf{E}, \mathbf{W}} \mathcal{L}_{\text{CE}}(\mathcal{D}|\{\mathbf{g} \odot \mathbf{E}, \mathbf{W}\}). \tag{15}$$

The overall process of our model is summarized in Algorithm 1.

Algorithm 1 The OptFS Algorithm

```
Require: training dataset $\mathcal{D}$, initialization epoch $T_c$, total epoch $T$
Ensure: gating vector $\mathbf{g}$, model parameters $\{\mathbf{E}, \mathbf{W}\}$
 1: ## Searching ##
 2: t=0
 3: while t < T do
 4:     t = t + 1
 5:     while $\mathcal{D}$ is not fully iterated do
 6:         Sample a mini-batch from the training dataset
 7:         $\{\mathbf{E}_t, \mathbf{W}_t\}$, $\mathbf{g}$ = Searching($\mathcal{D}$)        ▷ Equation 13
 8:     end while
 9:     if $t == T_c$ then
10:         $\{\hat{\mathbf{E}}, \hat{\mathbf{W}}\} \Leftarrow \{\mathbf{E}_t, \mathbf{W}_t\}$
11:     end if
12: end while
13: $\mathbf{g}$ = Discretization($\{\mathbf{g}_c\}$)        ▷ Equation 14
14: ## Retraining##
15: Retrain $\{\mathbf{E}, \mathbf{W}\}$ given $\mathbf{g}$ with $\{\hat{\mathbf{E}}, \hat{\mathbf{W}}\}$ as initialization        ▷ Equation 15
```

3 EXPERIMENT

In this section, to comprehensively evaluate our proposed method, we design experiments to answer the following research questions:

- **RQ1**: Could OptFS achieve superior performance compared with mainstream feature (interaction) selection methods?
- **RQ2**: How does the end-to-end training scheme influence the model performance?
- **RQ3**: How does the re-training stage influence the performance?
- **RQ4**: How efficient is OptFS compared to other feature (interaction) selection methods?
- **RQ5**: Does OptFS select the optimal features?

3.1 Experiment Setup

3.1.1 Datasets. We conduct our experiments on three public real-world datasets. We describe all datasets and the pre-processing steps below.

Criteo[1] dataset consists of ad click data over a week. It consists of 26 categorical feature fields and 13 numerical feature fields. Following the best practice [38], we discretize each numeric value x to $\lfloor \log^2(x) \rfloor$, if $x > 2$; $x = 1$ otherwise. We replace infrequent categorical features with a default "OOV" (i.e. out-of-vocabulary) token, with *min_count*=10.

Avazu[2] dataset contains 10 days of click logs. It has 24 fields with categorical features. Following the best practice [38], we remove the *instance_id* field and transform the *timestamp* field into three new fields: *hour*, *weekday* and *is_weekend*. We replace infrequent categorical features with the "OOV" token, with *min_count*=10.

KDD12[3] dataset contains training instances derived from search session logs. It has 11 categorical fields, and the click field is the number of times the user clicks the ad. We replace infrequent features with an "OOV" token, with *min_count*=10.

3.1.2 Metrics. Following the previous works [7, 26], we use the common evaluation metrics for CTR prediction: **AUC** (Area Under ROC) and **Log loss** (cross-entropy). Note that **0.1**% improvement in AUC is considered significant [7, 24]. To measure the size of the feature set, we normalize it based on the following equation:

$$\text{Ratio} = \#\text{Remaining Features}/m. \tag{16}$$

3.1.3 Baseline Methods and Backbone Models. We compare the proposed method OptFS with the following feature selection methods: (i) AutoField [32]: This baseline utilizes neural architecture search techniques [15] to select the informative features on a field level; (ii) LPFS [8]: This baseline designs a customized, smoothed-ℓ_0-liked function to select informative fields on a field level; (iii) AdaFS [13]: This baseline that selects the most relevant features for each sample via a novel controller network. We apply the above baselines over the following mainstream backbone models: FM [26], DeepFM [7], DCN [31] and IPNN [24].

We also compare the proposed method OptFS with a feature interaction selection method: AutoFIS [14]. This baseline utilizes GRDA optimizer to abandon unimportant feature interaction in a field-level manner. We apply AutoFIS over the following backbone models: FM [26], DeepFM [7]. We only compare with AutoFIS on FM and DeepFM backbone models because the original paper only provides the optimal hyper-parameter settings and releases source code under these settings.

3.1.4 Implementation Details. In this section, we provide the implementation details. For OptFS, (i) General hyper-params: We set the embedding dimension as 16 and batch size as 4096. For the MLP layer, we use three fully-connected layers of size [1024, 512, 256]. Following previous work [24], Adam optimizer, Batch Normalization [10] and Xavier initialization [5] are adopted. We select the optimal learning ratio from {1e-3, 3e-4, 1e-4, 3e-5, 1e-5} and l_2 regularization from {1e-3, 3e-4, 1e-4, 3e-5, 1e-5, 3e-6, 1e-6}. (ii) OptFS hyper-params: we select the optimal regularization penalty λ from {1e-8, 5e-9, 2e-9, 1e-9}, training epoch T from {5, 10, 15}, final value γ from {2e+2, 5e+2, 1e+3, 2e+3, 5e+3, 1e+4}. During the re-training phase, we reuse the optimal learning ratio and l_2 regularization and choose the rewinding epoch T_c from $\{1, 2, \cdots, T-1\}$. For AutoField and AdaFS, we select the optimal hyper-parameter from the same hyper-parameter domain of OptFS, given the original paper does not provide the hyper-parameter settings. For LPFS and AutoFIS, we reuse the optimal hyper-parameter mentioned in original papers.

Our implementation[4] is based on a public Pytorch library for CTR prediction[5]. For other baseline methods, we reuse the official implementation for the AutoFIS[6] [14] method. Due to the lack of available implementations for the LPFS [8], AdaFS[13] and AutoField[32] methods, we re-implement them based on the details provided by the authors and open-source them to benefit future researchers[7].

3.2 Overall Performance(RQ1)

In this section, we conduct two studies to separately compare feature selection methods and feature interaction selection methods in Section 3.2.1 and 3.2.2. Notes that both these methods can be viewed as a solution to the feature set optimization problem.

3.2.1 Feature Selection. The overall performance of our OptFS and other feature selection baseline methods on four different backbone models using three benchmark datasets are reported in Table 2. We summarize our observation below.

Firstly, our OptFS is effective and efficient compared with other baseline methods. OptFS can achieve higher AUC with a lower feature ratio. However, the benefit brought by OptFS differs on various datasets. On Criteo, OptFS tends to reduce the size of the feature set. OptFS can reduce 86% to 96% features with improvement not considered significant statistically. On the Avazu and KDD12 datasets, the benefit tends to be both performance boosting and feature reduction. OptFS can significantly increase the AUC by 0.01% to 0.45% compared with the backbone model while using roughly 10% of the features. Note that the improved performance is because OptFS considers feature interaction's influence during selection. Meanwhile, other feature selection baselines tend to bring performance degradation. This is likely because they adopt the feature field selection. Such a design will inevitably drop useful features or keep useless ones.

[1]https://www.kaggle.com/c/criteo-display-ad-challenge
[2]http://www.kaggle.com/c/avazu-ctr-prediction
[3]http://www.kddcup2012.org/c/kddcup2012-track2/data

[4]https://github.com/fuyuanlyu/OptFS
[5]https://github.com/rixwew/pytorch-fm
[6]https://github.com/zhuchenxv/AutoFIS
[7]https://github.com/fuyuanlyu/AutoFS-in-CTR

Table 2: Performance Comparison Between OptFS and Feature Selection Methods.

	Method	FM			DeepFM			DCN			IPNN		
		AUC↑	Logloss↓	Ratio↓	AUC↑	Logloss↓	Ratio↓	AUC↑	Logloss↓	Ratio↓	AUC↑	Logloss↓	Ratio↓
Criteo	Backbone	0.8055	0.4457	1.0000	0.8089	0.4426	1.0000	0.8107	0.4410	1.0000	0.8110	0.4407	1.0000
	LPFS	0.7888	0.4604	0.0157	0.7915	0.4579	0.2415	0.7802	0.4743	0.1177	0.7789	0.4705	0.3457
	AutoField	0.7932	0.4567	**0.0008**	0.8072	0.4439	0.3811	**0.8113**	**0.4402**	0.5900	0.8115	**0.4401**	0.9997
	AdaFS	0.7897	0.4597	1.0000	0.8005	0.4501	1.0000	0.8053	0.4472	1.0000	0.8065	0.4448	1.0000
	OptFS	**0.8060**	**0.4454**	0.1387	**0.8100***	**0.4415***	**0.0422**	0.8111	0.4405	**0.0802**	**0.8116**	**0.4401**	**0.0719**
Avazu	Backbone	0.7838	0.3788	1.0000	0.7901	0.3757	1.0000	0.7899	0.3755	1.0000	0.7913	0.3744	1.0000
	LPFS	0.7408	0.4029	0.7735	0.7635	0.3942	0.9975	0.7675	0.3889	0.9967	0.7685	0.3883	0.9967
	AutoField	0.7680	0.3862	**0.0061**	0.7870	0.3773	1.0000	0.7836	0.3782	0.9992	0.7865	0.3770	0.9992
	AdaFS	0.7596	0.3913	1.0000	0.7797	0.3837	1.0000	0.7693	0.3954	1.0000	0.7818	0.3833	1.0000
	OptFS	**0.7839**	**0.3784**	0.8096	**0.7946***	**0.3712***	**0.8686**	**0.7932***	**0.3718***	**0.8665**	**0.7950***	**0.3709***	**0.9118**
KDD12	Backbone	0.7783	0.1566	1.0000	0.7967	0.1531	1.0000	0.7974	0.1531	1.0000	0.7966	0.1532	1.0000
	LPFS	0.7725	0.1578	1.0000	0.7964	0.1532	1.0000	0.7970	**0.1530**	1.0000	0.7967	0.1532	1.0000
	AutoField	0.7411	0.1634	**0.0040**	0.7919	0.1542	0.9962	0.7943	0.1536	**0.8249**	0.7926	0.1541	**0.8761**
	AdaFS	0.7418	0.1644	1.0000	0.7917	0.1543	1.0000	0.7939	0.1538	1.0000	0.7936	0.1539	1.0000
	OptFS	**0.7811***	**0.1560***	0.5773	**0.7988***	**0.1527***	**0.9046**	**0.7987***	**0.1527**	0.8945	**0.7976**	**0.1530**	0.8762

Here $*$ denotes statistically significant improvement (measured by a two-sided t-test with p-value < 0.05) over the best baseline. **Bold** font indicates the best-performed method.

Secondly, different datasets behave differently regarding the redundancy of features. For example, on the Criteo dataset, all methods produce low feature ratios, indicating that this dataset contains many redundant features. On the other hand, on the Avazu and KDD12 datasets, all methods produce high feature ratios, suggesting that these two datasets have lower redundancy. OptFS can better balance the trade-off between model performance and efficiency compared with other baselines in all datasets.

Finally, field-level feature selection methods achieve different results on various backbone models. Compared to other deep models, FM solely relies on the explicit interaction, i.e. inner product. If one field $\mathbf{z}_i$ is zeroed out during the process, all its related interactions will be zero. The other fields are also lured into zero, as their interaction with field $\mathbf{z}_i$ does not bring any information into the final prediction. Therefore, it can be observed that LPFS has a low feature ratio on Criteo and high feature ratios on Avazu and KDD12 datasets. On the other hand, AutoField generates low feature ratios (~0%) on all three datasets. These observations further highlight the necessity of introducing feature-level granularity into the feature set optimization problem as OptFS does.

3.2.2 Feature Interaction Selection. In this subsection, we aim to study the influence of the OptFS method on feature interaction selection. The overall performance of our OptFS and AutoFIS on DeepFM and FM backbone models are reported in Table 3. We summarize our observation below.

Firstly, compared with backbone models that do not perform any feature interaction selection, AutoFIS and OptFS achieve higher performance. Such an observation points out the existence of useless feature interaction on both datasets.

Secondly, the performance of OptFS and AutoFIS differs on different models. With fewer features in the feature set, OptFS achieves nearly the same performance as AutoFIS on FM while performing significantly better on DeepFM. This is because OptFS focuses on feature-level interactions, which are more fine-grained than the field-level interactions adopted by AutoFIS.

Table 3: Performance Comparison Between OptFS and Feature Interaction Selection Method.

	Model	Method	Metrics		
			AUC↑	Logloss↓	Ratio↓
Criteo	FM	Backbone	0.8055	0.4457	1.0000
		AutoFIS	**0.8063**	**0.4449**	1.0000
		OptFS	0.8060	0.4454	**0.1387**
	DeepFM	Backbone	0.8089	0.4426	1.0000
		AutoFIS	0.8097	0.4418	1.0000
		OptFS	**0.8100**	**0.4415**	**0.0422**
Avazu	FM	Backbone	0.7838	0.3788	1.0000
		AutoFIS	**0.7843**	0.3785	1.0000
		OptFS	0.7839	**0.3784**	**0.8096**
	DeepFM	Backbone	0.7901	0.3757	1.0000
		AutoFIS	0.7928	0.3721	1.0000
		OptFS	**0.7946***	**0.3712***	**0.8686**

Here $*$ denotes statistically significant improvement (measured by a two-sided t-test with p-value < 0.05) over the best baseline. **Bold** font indicates the best-performed method.

Finally, it is also worth mentioning that OptFS can reduce 13% to 96% of features while AutoFIS is conducted on all possible features without any reduction.

3.3 Transferability Study(RQ2)

In this subsection, we investigate the transferability of OptFS's result. The experimental settings are listed as follows. First, we search the gating vector $\mathbf{g}$ from one model, which we named the source. Then, we re-train another backbone model given the obtained gating vector, which we call the target. We study the transferability between DeepFM, DCN and IPNN backbone models over both Criteo and Avazu datasets. Based on the results shown in Table 4, we can easily observe that all transformation leads to performance degradation. Such degradation is even considered significant over the Avazu dataset. Therefore, feature interaction operations require different feature sets to achieve high performance. We can conclude that the selection of the feature set needs to incorporate the interaction operation, which further highlights the importance of selecting both features and their interactions in a unified, end-to-end trainable way.

Table 4: Transferability Analysis on Criteo and Avazu.

	Target	Source	Metrics		
			AUC↑	Logloss↓	Ratio↓
Criteo	DeepFM	DeepFM	**0.8100**	**0.4415**	**0.0422**
		DCN	0.8097	0.4419	0.0802
		IPNN	0.8097	0.4418	0.0719
	DCN	DCN	**0.8111**	**0.4405**	0.0802
		DeepFM	0.8106	0.4410	**0.0422**
		IPNN	0.8107	0.4410	0.0719
	IPNN	IPNN	**0.8116**	**0.4401**	0.0719
		DCN	0.8113	0.4404	0.0802
		DeepFM	0.8114	0.4403	**0.0422**
Avazu	DeepFM	DeepFM	**0.7946***	**0.3712***	0.8686
		DCN	0.7873	0.3754	**0.8665**
		IPNN	0.7872	0.3755	0.9118
	DCN	DCN	**0.7932***	**0.3718***	**0.8665**
		DeepFM	0.7879	0.3784	0.8686
		IPNN	0.7860	0.3762	0.9118
	IPNN	IPNN	**0.7950***	**0.3709***	0.9118
		DCN	0.7907	0.3747	**0.8665**
		DeepFM	0.7908	0.3748	0.8686

Here $*$ denotes statistically significant improvement (measured by a two-sided t-test with p-value < 0.05) over the best baseline. **Bold** font indicates the best-performed method.

3.4 Ablation Study(RQ3)

In this subsection, we conduct the ablation study over the influence of the re-training stage, which is detailedly illustrated in Section 2.4.2. In Section 2.4.2, we propose a **c**ustomized **i**nitialization method, namely *c.i.*, during the re-training stage. Here we compare it with the other three methods of obtaining model parameters: (i) *w.o.*, which is the abbreviation for **with**out re-training, directly inherit the model parameters from the searching stage; (ii) *r.i.* **r**andomly **i**nitialize the model parameters; (iii) *l.t.h.*, which stands for **l**ottery **t**icket **h**ypothesis, is a common method for re-training sparse network [4]. Specifically, it initializes the model parameters with the same seed from the searching stage. The experiment is conducted over three backbone models, DeepFM, DCN and IPNN, over Criteo and Avazu benchmarks. We can make the following observations based on the result shown in Table 5.

Firstly, we can easily observe that re-training can improve performance regardless of its setting. Without re-training, the neural network will inherit the sub-optimal model parameters from the searching stage, which is influenced by the non-binary element in the gating vector. Re-training improves the model performance under the constraint of the gating vector.

Secondly, *c.i.* constantly outperforms the other two re-training methods. Such performance gaps are considered significant on all three backbone models over the Avazu dataset. This is likely because, on the Avazu dataset, the backbone models are usually trained for only one epoch before they get early-stopped for over-fitting. Hence, it further increases the importance of initialization during the re-training stage. This observation validates the necessity of introducing customized initialization in CTR prediction.

Table 5: Ablation Study Regarding the Re-training Stage.

	Model	Metrics	Methods			
			w.o.	r.i.	l.t.h.	c.i.
Criteo	DeepFM	AUC↑	0.8012	**0.8100**	**0.8100**	**0.8100**
		Logloss↓	0.4686	0.4416	**0.4415**	**0.4415**
	DCN	AUC↑	0.8077	0.8109	0.8108	**0.8111**
		Logloss↓	0.4522	0.4407	0.4408	**0.4405**
	IPNN	AUC↑	0.7757	0.8113	0.8114	**0.8116**
		Logloss↓	0.4998	0.4404	0.4403	**0.4401**
Avazu	DeepFM	AUC↑	0.6972	0.7873	0.7883	**0.7946***
		Logloss↓	0.5017	0.3754	0.3790	**0.3712***
	DCN	AUC↑	0.7122	0.7870	0.7858	**0.7932***
		Logloss↓	0.4736	0.3801	0.3764	**0.3718***
	IPNN	AUC↑	0.7560	0.7912	0.7910	**0.7950***
		Logloss↓	0.4411	0.3745	0.3745	**0.3709***

Here $*$ denotes statistically significant improvement (measured by a two-sided t-test with p-value < 0.05) over the best baseline. **Bold** font indicates the best-performed method. Here *w.o.* stands for **with**out re-training, *r.i.* stands for re-training with **r**andom **i**nitialization, *l.t.h.* stands for initialization using **l**ottery **t**icket **h**ypothesis [4], *c.i.* stands for re-training with **c**ustomized **i**nitialization, as previously discussed in Section 2.4.

3.5 Efficiency Analysis(RQ4)

In addition to model performance, efficiency is vital when deploying the CTR prediction model in reality. In this section, we investigate the time and space complexity of OptFS.

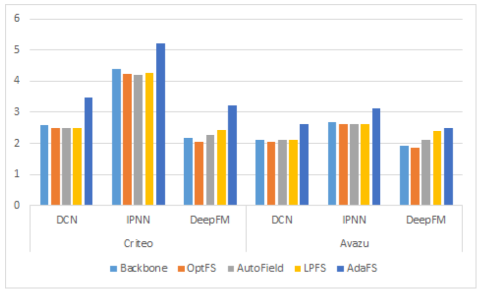
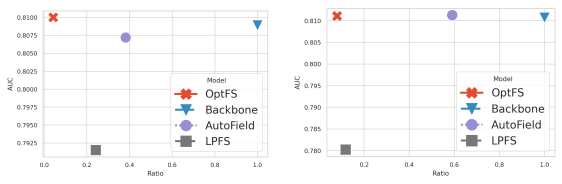


Figure 4: Inference Time on Criteo and Avazu Dataset. The Y-axis represents the influence time, measured by ms

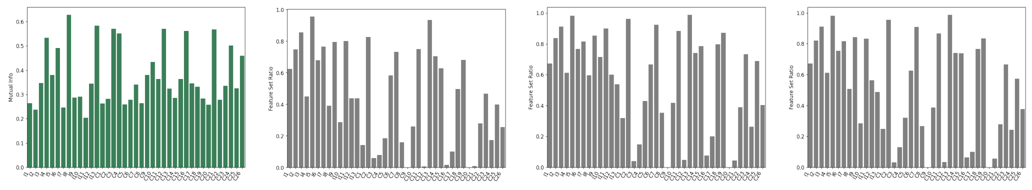


(a) DeepFM (b) DCN

Figure 5: Visualization of efficiency-effectiveness trade-off on Criteo datasets. The closer to the top-left the better.

3.5.1 Time Complexity. The inference time is crucial when deploying the model into online web systems. We define inference time as the time for inferencing one batch. The result is obtained by averaging the inference time over all batches on the validation set.

As shown in Figure 4, OptFS achieves the least inference time. This is because the feature set obtained by OptFS usually has the least features. Meanwhile, AdaFS requires the longest inference



(a) Mutual Info (b) DeepFM (c) DCN (d) IPNN

Figure 6: A Case Study of OptFS output on Criteo. In all subfigures, the X-axis indicates the field identifiers. Subfigure (a) plots the mutual information scores, while subfigures (b), (c) and (d) plot the feature set ratio of OptFS on DeepFM, DCN and IPNN.

time, even longer than the backbone model. This is because it needs to determine whether keep or drop each feature dynamically during run-time.

3.5.2 Space Complexity. We plot the Feature Ratio-AUC curve of the DeepFM, DCN and IPNN model on the Criteo datasets in Figure 5, which reflects the relationship between the space complexity of the feature set and model performance. Notes that LPFS, AutoField and OptFS are methods that primarily aim to improve model performance. These methods have no guarantee over the final feature ratios. Hence we only plot one point for each method in the figure.

From Figure 5 we can make the following observations: (i) OptFS outperforms all other baselines with the highest AUC score and the least number of features. (ii) The model performance of AutoField is comparable with OptFS and Backbone. However, given it only selects the feature set on field-level, its feature ratio tends to be higher than OptFS. (iii) The performance of LPFS is much lower than other methods.

3.6 Case Study(RQ5)

This subsection uses a case study to investigate the optimal feature set obtained from OptFS. In Figure 6, we plot the mutual information with the feature ratio on each field. For field $\mathbf{z}_i = \{\mathbf{x}_{k_i}\}$ and ground truth labels $\mathbf{y}$ ($y \in \mathbf{y}$), the mutual information between them is defined as:

$$\mathbf{MI}(\mathbf{x}_{k_i}, \mathbf{y}) = -\sum \mathbf{P}(y) \log \mathbf{P}(y) + \sum \mathbf{P}(\mathbf{x}_{k_i}, y) \log \mathbf{P}(y|\mathbf{x}_{k_i}), \quad (17)$$

where the first term is the marginal entropy and the second term is the conditional entropy of ground truth labels $\mathbf{y}$ given field $\mathbf{z}_i = \{\mathbf{x}_{k_i}\}$. Note that fields with high mutual information scores are more informative (hence more important) to the prediction.

As a case study, we investigate the feature ratio for each field, shown in Figure 6. We select the result from DeepFM, DCN and IPNN on the Criteo dataset. Figure 6(a) shows the mutual information scores of each field, which represents how informative each field is in predicting the label. Figure 6(b), 6(c) and 6(d) shows the feature ratio given each fields. As can be seen, fields with higher mutual information scores are likely to keep more features in the feature set, which indicates that OptFS obtains the optimal feature set from the field perspective.

4 RELATED WORK

In this section, we review the related work. Optimizing feature set is related two topics, feature selection and feature interaction selection. The training scheme of proposed OptFS is related to learning by continuation. Thus we summarize the related work in following two subsection.

4.1 Feature and Feature Interaction Selection

Feature selection is a key component for prediction task [3]. Several methods have been proposed [8, 13, 17, 30, 32] to conduct feature selection for CTR models. Traditional methods [17, 30] exploit the statistical metrics of different feature fields and conduct feature field selection. Inspired by neural architecture search (NAS) [15, 18] and smoothed-ℓ_0 optimization respectively, AutoField [32] and LPFS [8] determine the selection of feature fields automatically. AdaFS [13] proposes a novel controller network to decide feature fields for each sample, which fits the dynamic recommendation. Feature interaction selection is often employed to enhance the prediction. Some methods [12, 14] model the problem as NAS to exploit the field-level interaction space. OptInter [20] investigates the way to do feature interaction. AutoCross [19] targets on tabular data and iterative finds feature interaction based on locally optimized feature set. We first highlight the feature set optimization problem in CTR prediction, and OptFS is different from previous methods by solving both problems in a unified manner.

4.2 Learning by Continuation

Continuation methods are commonly used to approximate intractable optimization problems by gradually increasing the difficulty of the underlying objective. By adopting gradual relaxations to binary problems, gumbel-softmax [11] is used to back-propagate errors during the architecture search [33] and spatial feature sparsification [35]. Other methods [16, 28, 36] introduce continuous sparsification framework to speed up neural network pruning and ticket search. OptFS adopts the learning-by-continuation scheme to effectively explore the huge feature-level search space.

5 CONCLUSION

This paper first distinguishes the feature set optimization problem. Such a problem unifies two mutually influencing questions: the selection of features and feature interactions. To our knowledge, no previous work considers these two questions uniformly. Besides, we also upgrade the granularity of the problem from field-level to feature-level. To solve such the feature set optimization problem efficiently, we propose a novel method named OptFS, which assigns a gating value to each feature for its usefulness and adopt a learning-by-continuation approach for efficient optimization. Extensive experiments on three large-scale datasets demonstrate the superiority of OptFS in model performance and feature reduction. Several ablation studies also illustrate the necessity of our design. Moreover, we also interpret the obtained result on feature fields and their interactions, highlighting that our method properly solves the feature set optimization problem.

ACKNOWLEDGMENTS

We explicitly want to thank Chaohua Yang for his help regarding the experiment.

REFERENCES

[1] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. 2013. Representation Learning: A Review and New Perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.* 35, 8 (2013), 1798–1828. https://doi.org/10.1109/TPAMI.2013.50

[2] Olivier Chapelle, Eren Manavoglu, and Romer Rosales. 2015. Simple and Scalable Response Prediction for Display Advertising. *ACM Trans. Intell. Syst. Technol.* 5, 4 (dec 2015), 61.

[3] Jérémie Donà and Patrick Gallinari. 2021. Differentiable Feature Selection, A Reparameterization Approach. In *Machine Learning and Knowledge Discovery in Databases. Research Track - European Conference, ECML PKDD 2021 (Lecture Notes in Computer Science, Vol. 12977)*. Springer, Bilbao, Spain, 414–429. https://doi.org/10.1007/978-3-030-86523-8_25

[4] Jonathan Frankle and Michael Carbin. 2019. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In *7th International Conference on Learning Representations, ICLR 2019*. OpenReview.net, New Orleans, LA, USA.

[5] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *13th International Conference on Artificial Intelligence and Statistics, AISTATS 2010 (JMLR Proceedings, Vol. 9)*. JMLR.org, Italy, 249–256.

[6] Huifeng Guo, Wei Guo, Yong Gao, Ruiming Tang, Xiuqiang He, and Wenzhi Liu. 2021. ScaleFreeCTR: MixCache-based Distributed Training System for CTR Models with Huge Embedding Table. In *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, Virtual Event, Canada, 1269–1278. https://doi.org/10.1145/3404835.3462976

[7] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. In *26th International Joint Conference on Artificial Intelligence, IJCAI 2017*. ijcai.org, Melbourne, Australia, 1725–1731.

[8] Yi Guo, Zhaocheng Liu, Jianchao Tan, Chao Liao, Daqing Chang, Qiang Liu, Sen Yang, Ji Liu, Dongying Kong, Zhi Chen, and Chengru Song. 2022. LPFS: Learnable Polarizing Feature Selection for Click-Through Rate Prediction. *CoRR* abs/2206.00267 (2022). https://doi.org/10.48550/arXiv.2206.00267 arXiv:2206.00267

[9] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition*. Springer, Berlin, Germany. https://doi.org/10.1007/978-0-387-84858-7

[10] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *32nd International Conference on Machine Learning, ICML 2015 (JMLR Workshop and Conference Proceedings, Vol. 37)*. JMLR.org, France, 448–456.

[11] Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical Reparameterization with Gumbel-Softmax. In *5th International Conference on Learning Representations, ICLR 2017*. OpenReview.net, Toulon, France.

[12] Farhan Khawar, Xu Hang, Ruiming Tang, Bin Liu, Zhenguo Li, and Xiuqiang He. 2020. AutoFeature: Searching for Feature Interactions and Their Architectures for Click-through Rate Prediction. In *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management*. ACM, Virtual Event, Ireland, 625–634. https://doi.org/10.1145/3340531.3411912

[13] Weilin Lin, Xiangyu Zhao, Yejing Wang, Tong Xu, and Xian Wu. 2022. AdaFS: Adaptive Feature Selection in Deep Recommender System. In *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, Washington, DC, USA, 3309–3317. https://doi.org/10.1145/3534678.3539204

[14] Bin Liu, Chenxu Zhu, Guilin Li, Weinan Zhang, Jincai Lai, Ruiming Tang, Xiuqiang He, Zhenguo Li, and Yong Yu. 2020. AutoFIS: Automatic Feature Interaction Selection in Factorization Models for Click-Through Rate Prediction. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, Virtual Event, CA, USA, 2636–2645. https://doi.org/10.1145/3394486.3403314

[15] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019. DARTS: Differentiable Architecture Search. In *7th International Conference on Learning Representations, ICLR 2019*. OpenReview.net, USA.

[16] Junjie Liu, Zhe Xu, Runbin Shi, Ray C. C. Cheung, and Hayden Kwok-Hay So. 2020. Dynamic Sparse Training: Find Efficient Sparse Network From Scratch With Trainable Masked Layers. In *8th International Conference on Learning Representations, ICLR 2020*. OpenReview.net, Addis Ababa, Ethiopia.

[17] Qiang Liu, Zhaocheng Liu, Haoli Zhang, Yuntian Chen, and Jun Zhu. 2021. Mining Cross Features for Financial Credit Risk Assessment. In *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management*. ACM, Virtual Event, Queensland, Australia, 1069–1078. https://doi.org/10.1145/3459637.3482371

[18] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. 2018. Neural Architecture Optimization. In *31st Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018*. Curran Associates, Montréal, Canada, 7827–7838.

[19] Yuanfei Luo, Mengshuo Wang, Hao Zhou, Quanming Yao, Wei-Wei Tu, Yuqiang Chen, Wenyuan Dai, and Qiang Yang. 2019. AutoCross: Automatic Feature Crossing for Tabular Data in Real-World Applications. In *25th ACM International Conference on Knowledge Discovery & Data Mining, KDD 2019*. ACM, Anchorage, AK, USA, 1936–1945. https://doi.org/10.1145/3292500.3330679

[20] Fuyuan Lyu, Xing Tang, Huifeng Guo, Ruiming Tang, Xiuqiang He, Rui Zhang, and Xue Liu. 2022. Memorize, Factorize, or be Naive: Learning Optimal Feature Interaction Methods for CTR Prediction. In *38th IEEE International Conference on Data Engineering, ICDE 2022*. IEEE, Kuala Lumpur, Malaysia, 1450–1462. https://doi.org/10.1109/ICDE53745.2022.00113

[21] Fuyuan Lyu, Xing Tang, Hong Zhu, Huifeng Guo, Yingxue Zhang, Ruiming Tang, and Xue Liu. 2022. OptEmbed: Learning Optimal Embedding Table for Click-through Rate Prediction. *CoRR* abs/2208.04482 (2022).

[22] Ze Meng, Jinnian Zhang, Yumeng Li, Jiancheng Li, Tanchao Zhu, and Lifeng Sun. 2021. A General Method For Automatic Discovery of Powerful Interactions In Click-Through Rate Prediction. In *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, Canada, 1298–1307.

[23] B. K. Natarajan. 1995. Sparse Approximate Solutions to Linear Systems. *SIAM J. Comput.* 24, 2 (1995), 227–234. https://doi.org/10.1137/S0097539792240406

[24] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. 2016. Product-Based Neural Networks for User Response Prediction. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, Barcelona, Spain, 1149–1154. https://doi.org/10.1109/ICDM.2016.0151

[25] Yanru Qu, Bohui Fang, Weinan Zhang, Ruiming Tang, Minzhe Niu, Huifeng Guo, Yong Yu, and Xiuqiang He. 2019. Product-Based Neural Networks for User Response Prediction over Multi-Field Categorical Data. *ACM Trans. Inf. Syst.* 37, 1 (2019), 5:1–5:35.

[26] Steffen Rendle. 2010. Factorization Machines. In *ICDM 2010, The 10th IEEE International Conference on Data Mining*. IEEE Computer Society, Sydney, Australia, 995–1000.

[27] Matthew Richardson, Ewa Dominowska, and Robert Ragno. 2007. Predicting clicks: estimating the click-through rate for new ads. In *16th International Conference on World Wide Web, WWW 2007*. ACM, Banff, Alberta, Canada, 521–530.

[28] Pedro Savarese, Hugo Silva, and Michael Maire. 2020. Winning the Lottery with Continuous Sparsification. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*. Curran Associates, virtual.

[29] Andrew I. Schein, Alexandrin Popescul, Lyle H. Ungar, and David M. Pennock. 2002. Methods and metrics for cold-start recommendations. In *SIGIR 2002: the 25th Annual International Conference on Research and Development in Information Retrieval*. ACM, Tampere, Finland, 253–260. https://doi.org/10.1145/564376.564421

[30] Robert Tibshirani. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)* 58, 1 (1996), 267–288.

[31] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & Cross Network for Ad Click Predictions. In *ADKDD'17 (ADKDD'17)*. Association for Computing Machinery, Canada, Article 12, 7 pages.

[32] Yejing Wang, Xiangyu Zhao, Tong Xu, and Xian Wu. 2022. AutoField: Automating Feature Selection in Deep Recommender Systems. In *WWW '22: The ACM Web Conference 2022*. ACM, Virtual Event, Lyon, France, 1977–1986. https://doi.org/10.1145/3485447.3512071

[33] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. 2019. FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019*. Computer Vision Foundation / IEEE, Long Beach, CA, USA, 10734–10742 pages. https://doi.org/10.1109/CVPR.2019.01099

[34] Yuexiang Xie, Zhen Wang, Yaliang Li, Bolin Ding, Nezihe Merve Gürel, Ce Zhang, Minlie Huang, Wei Lin, and Jingren Zhou. 2021. FIVES: Feature Interaction Via Edge Search for Large-Scale Tabular Data. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, Virtual Event, Singapore, 3795–3805. https://doi.org/10.1145/3447548.3467066

[35] Zhenda Xie, Zheng Zhang, Xizhou Zhu, Gao Huang, and Stephen Lin. 2020. Spatially Adaptive Inference with Stochastic Feature Sampling and Interpolation. In *Computer Vision - ECCV 2020 - 16th European Conference (Lecture Notes in Computer Science, Vol. 12346)*. Springer, Glasgow, UK, 531–548. https://doi.org/10.1007/978-3-030-58452-8_31

[36] Xin Yuan, Pedro Henrique Pamplona Savarese, and Michael Maire. 2021. Growing Efficient Deep Networks by Structured Continuous Sparsification. In *9th International Conference on Learning Representations, ICLR 2021*. OpenReview.net, Virtual Event, Austria.

[37] Zhao-Yu Zhang, Xiang-Rong Sheng, Yujing Zhang, Biye Jiang, Shuguang Han, Hongbo Deng, and Bo Zheng. 2022. Towards Understanding the Overfitting Phenomenon of Deep Click-Through Rate Prediction Models. *CoRR* abs/2209.06053 (2022). https://doi.org/10.48550/arXiv.2209.06053 arXiv:2209.06053

[38] Jieming Zhu, Jinyang Liu, Shuai Yang, Qi Zhang, and Xiuqiang He. 2021. Open Benchmarking for Click-Through Rate Prediction. In *30th ACM International Conference on Information & Knowledge Management*. Association for Computing Machinery, Australia, 2759–2769.