AIM: Automatic Interaction Machine for Click-Through Rate Prediction

Chenxu Zhu, Bo Chen, Weinan Zhang, Jincai Lai, Ruiming Tang, Xiuqiang He, Zhenguo Li and Yong Yu

Abstract—Feature embedding learning and feature interaction modeling are two crucial components of deep models for Click-Through Rate (CTR) prediction in recommender systems. Most existing deep CTR models suffer from the following three problems. First, feature interactions are either manually designed or simply enumerated. However, not all the feature interactions are useful for the prediction task and useless feature interactions may introduce noisy signals thus causing overfitting. Second, all the feature interactions are modeled with an identical interaction function, whereas different interaction functions introduce different inductive biases to better capture various feature interaction patterns. Third, in most existing models, different features share the same embedding size. However, model size can be further optimized without sacrificing performance by differentiating embedding sizes for individual features, as the amount of information contained in each feature varies much. To address the three issues mentioned above, we propose **Automatic Interaction Machine (AIM)** with three core components, namely, Feature Interaction Search (FIS), Interaction Function Search (IFS) and Embedding Dimension Search (EDS), respectively. To tackle the first problem, FIS component automatically identifies different orders of essential feature interactions with useless ones pruned. Taking care of the second problem, IFS component selects appropriate interaction functions for each individual feature interaction in a learnable way. Moreover, to avoid learning conflict among different interaction functions, IFS proposes function-wise embeddings via performing multiple embeddings for each feature, where each feature embedding corresponds to one possible interaction function. However, utilizing multiple embeddings for each feature may make the model size affordably large if we keep the same embedding size as utilizing shared embedding (i.e., each feature shares the same embedding for different interaction functions). To solve this third problem, EDS automatically selects proper embedding size for each feature. Such a flexible embedding size adaptation is able to reduce the large amount of embedding parameters introduced by function-wise embeddings. Offline experiments on three large-scale datasets (two public benchmarks, one private dataset) validate that AIM can significantly improve various FM-based models. AIM has been deployed in the recommendation service of a mainstream app market, where a three-week online A/B test demonstrated the superiority of AIM, improving DeepFM model by 4.4% in terms of CTR.

Index Terms—CTR Prediction, Feature Interaction, Neural Architecture Search, Factorization Machine, Recommender Systems

1 Introduction

Click-Through Rate (CTR) prediction, which aims to predict the probability of the user clicking on the recommended items (e.g., music, advertisement), plays a core role in recommender systems. The estimated CTR may influence the subsequent recommendation decision-makings such as item ranking and ad placement. Accurate CTR prediction not only improves the user experience but also boosts the profit for the service providers. Since 2016, deep learning has been introduced to CTR prediction due to its high capacity of modeling high-order patterns and end-to-end learning manner [2]. So far, various deep CTR models have been deployed in the recommendation services of industrial companies, such as Wide & Deep [3] in Google Play, DeepFM [4] in Huawei AppGallery and DIN [5] in Taobao.

Most of the existing deep CTR prediction models consist of two key components: feature embedding learning and feature interaction modeling. Feature embeddings are learned via mapping categorical features into low-dimensional embedding vectors (short for embeddings). Feature interactions are learned by utilizing some functions to model the relationship among a set of feature embeddings. Many research works focus on designing interaction functions (or more generally, network architectures) to better capture feature interactions.

At the early stage, Deep Neural Network models [3, 6, 7] are proposed to model feature interactions implicitly with the multi-layer perceptron (MLP) built on the feature embeddings. In theory, a DNN could explore arbitrary feature interactions according to its universal approximation property [8]. However, there is no guarantee that a DNN naturally converges to any expected function using gradient-based optimization. Recent works prove the insensitive gradient issue of DNN when the target is a large collection of uncorrelated functions [9, 10]. Simple DNN models may not find the proper feature interactions, including the simple inner product [11]. Therefore, various complicated architectures have been proposed, such as DIN [5], DeepFM [4] and PNN [12]. **Factorization Models** (specified in Definition 1), such as FM [13], DeepFM, PNN, AFM [14], NFM [15], have been proposed to adopt an embedding layer and an inner-product feature extractor to respectively learn feature embedding and feature interactions.

However, there are three significant downsides in most of these existing models. First, all these models simply enu-

- *This paper is an extended version of AutoFIS [1] which has been accepted for the presentation at 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.*
- *Chenxu Zhu, Weinan Zhang and Yong Yu are with the Shanghai Jiao Tong University. E-mail: {zhuchenxv, wnzhang, yyu}@sjtu.edu.cn.*
- *Bo Chen, Jincai Lai, Ruiming Tang and Xiuqiang He are with the Huawei Noah's Ark Lab. E-mail: {chenbo116, laijincai, tangruiming, hexiuqiang1, Li.Zhenguo}@huawei.com.*
- *Corresponding author: Weinan Zhang.*

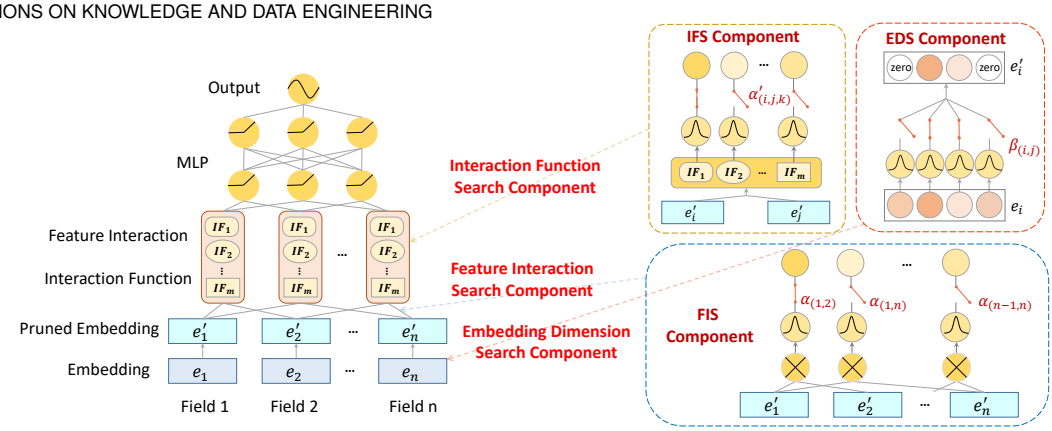

Fig. 1. Architectures of Automatic Interaction Machine (AIM).

merate all feature interactions or require human efforts to identify important feature interactions. The former always brings large memory and computation cost to the model and is difficult to extend into high-order interactions. Besides, useless interactions may bring unnecessary noise to cause overfitting and complicate the training process [10]. The latter, such as identifying important interactions manually in Wide & Deep [3], is of high labor cost and risks missing some counter-intuitive (but important) interactions.

Second, all the feature interactions with different complexity are modeled with the same specific interaction function (IF), such as inner product [4, 12, 14] or a predefined sub-network [9]. Nevertheless, using different interaction functions is conducive to better modeling various feature interactions under different mapping sub-spaces, especially for higher-order feature interactions as pointed out in AutoFeature [16].

Third, different features share the same embedding size to represent their embeddings. As pointed out in some remarkable works [17, 18, 19, 20], such a strategy may lead to memory inefficiency. More specifically, allocating the same embedding size to all features may lose the information of high predictive features while waste memory on non-predictive features. Therefore, model size can be further optimized without sacrificing performance by differentiating embedding sizes for individual features, i.e., assigning large embedding size to high predictive features, while assigning small embedding size to low predictive ones.

Some recent works improve the existing models by solving part of the above three problems. AutoFIS [1] and AutoGroup [21] identify important feature interactions while performing identical IF for all the selected feature interactions with uniform embedding size (namely, solving the first problem only). AutoFeature [16] solves the first and second problems by automatically searching proper network architectures to model different feature interactions via an evolutionary algorithm with the Naive Bayes tree. However, high complexity hinders the application of this method to large-scale industrial scenarios. To solve the third problem, a bunch of previous works [17, 18, 19, 20] utilize various techniques to find proper embedding sizes for each feature automatically, but the first two issues about feature interactions are totally ignored in these works. In other words, none of these advanced work solves all these three important problems in a unified framework.

To fill this gap, in this paper, we propose a unified framework called **Automatic Interaction Machine (AIM)** with three core components, i.e., *Feature Interaction Search (FIS)*, *Interaction Function Search (IFS)* and *Embedding Dimension Search (EDS)*, which are elaborated in Figure 1.

First, FIS component automatically learns which feature interactions are essential. Specifically, we introduce a *gate* (in open or closed status) for each feature interaction to control whether its output should be passed to the next layer. In previous works, the status of the gates are either specified beforehand by expert knowledge [3] or set as all open [4, 22]. From a data-driven point of view, whether to open or close a gate should depend on the contribution to the final prediction. Apparently, those contributing little should be closed to prevent the learning procedure from introducing extra noise. However, it is an NP-Hard problem to find the optimal set of open gates for model performance, as we face an incredibly huge space to search ($2^{\mathcal{C}_n^2}$ with n fields, even if we only consider 2^{nd}-order feature interactions). To make the search efficient in such a huge space, FIS component relaxes the choices of gates to be continuous such that gradient-based optimizations can be performed. Furthermore, FIS component is also able to select high-order feature interactions. To make the search process efficient, we exploit the selected low-order interaction to first restrict high-order interactions into a small candidate pool heuristically and then search from this pool rigorously, achieving $O(n^2)$ complexity.

As the second component of AIM, IFS can select appropriate IF for each essential feature interaction. To achieve this goal, a *gate* to each interaction-IF pair is needed. It can be observed that the search space of IFS is larger than that of FIS, which reaches to $2^{m\mathcal{C}_n^2}$ (with n fields and m IFs), if we only consider 2^{nd}-order feature interactions. To avoid learning conflict among different IFs, IFS proposes to utilize function-wise embeddings, i.e., multiple embeddings learned for each feature, where each of such embeddings corresponds to one possible IF[1].

As function-wise embeddings are used in IFS, the model size grows significantly (about m times than shared embedding). The third component, EDS, is proposed to optimize the model size without sacrificing performance by assigning small embedding sizes to low predictive features, which is achieved by pruning redundant embedding dimensions for each feature. Similar to FIS and IFS components, *gates* are

1. As the opposite, we refer *shared embedding* as the case that each feature shares the same embedding for different IFs.

also introduced, where a *gate* is annotated to each dimension of a feature embedding. In the end, the dimensions with open gates are recognized as important dimensions while the ones with closed gates are pruned.

Inspired by the recent work DARTS [23, 24, 25] for neural architecture search, instead of searching over a discrete set of candidate gates (i.e., feature interactions in FIS, interaction-IF pairs in IFS and embedding dimensions in EDS), we relax the choices to be continuous by introducing a set of architecture parameters (one for each gate) so that the relative importance can be learned by gradient descent. The architecture parameters are jointly optimized with neural network weights by GRDA optimizer [26], an optimizer that is easy to produce a sparse solution, so that the training process can automatically abandon unimportant feature interactions, useless interaction-IF pairs and unnecessary embedding dimensions with zero values as the architecture parameters and keep those important ones. Finally, we re-train the model with the selected feature interactions, interaction-IF pairs as well as embedding dimensions.

Extensive experiments are conducted on three large-scale datasets and the experimental results demonstrate that AIM can significantly improve the CTR prediction performance in terms of AUC and log loss. Besides, as AIM can remove about 80% of 2^{nd}-order interactions, our model can consistently achieve improvement on inference efficiency. AIM is able to model high-order feature interactions in a novel way with quadratic complexity. Experimental results show that with about 1%–5% of 3^{rd}-order feature interactions and about 0.03%–0.6% of 4^{th}-order feature interactions selected, the AUC of factorization models can be improved by 0.1%–0.4% without introducing much inference cost. Furthermore, AIM has been deployed in the recommendation service of a mainstream app market. From a three-week online A/B test, AIM achieves 4.4% CTR improvement over the production model DeepFM, which contributes a significant business revenue growth. To summarize, the main contributions of this paper can be highlighted as follows:

- We propose **AIM** with three core components (namely, FIS, IFS and EDS components) to select significant feature interactions, appropriate IFs and necessary embedding dimensions automatically in a unified framework.
- *Gates* are introduced in FIS, IFS and EDS components, with open (closed) status representing important (unimportant) candidates. We relax discrete selection of open gates to be continuous by introducing a set of architecture parameters that can be jointly optimized with neural network weights by GRDA optimizer.
- Offline experiments on three large-scale datasets demonstrate the superior performance of AIM. A three-week online A/B test in the recommendation service of a mainstream app market shows that AIM improves DeepFM model by 4.4% on average in terms of CTR.

2 Related Work

2.1 CTR Models

One core of CTR models is to extract effective low-order and high-order feature interactions, which is also one of the optimization targets of our work. Therefore, in this section, we discuss the CTR models that focus on learning feature interactions effectively.

FM [13] projects each feature into a low-dimensional vector and models pair-wise feature interactions by inner product, which works well for sparse data. FFM [27] extends FM, by further assigning each feature with multiple vectors to interact with features from other fields. Despite the significant improvement over FM, FFM introduces much more parameters and suffers from overfitting issues. HOFM [28] introduces the ANOVA kernel to approximate high-order feature interactions. However, it is shown in [14] that HOFM achieves only marginal improvement over FM whereas using much more parameters.

Recently, deep learning models have achieved state-of-the-art performance on some public CTR prediction benchmarks [29, 30]. As a powerful approach to learning feature representation, deep learning models have the potential to learn sophisticated feature interactions. Wide & Deep [3] jointly trains a wide model for artificial features and a deep model for raw features. Several models use MLP to improve FM, such as AFM [14], NFM [15] and DeepFM [4]. DeepFM uses an FM layer to replace the wide component in Wide & Deep. PNN [12] uses MLP to model high-order implicit information over the feature embeddings and interaction of FM layer while PIN [9] introduces a network-in-network architecture to model pairwise feature interactions with subnetworks rather than simple inner product operations in PNN and DeepFM. Due to the curse of dimensionality, DeepFM, PNN and PIN cannot explicitly model high-order feature interactions, which limits the further improvement of the model performance. xDeepFM [22] uses a CIN structure to enumerate and compress all feature interactions for modeling explicit interactions. However, it uses so many parameters that great challenges are posed to identify important feature interactions in the huge combination space.

As stated earlier, all the above mentioned CTR models suffer from three limitations: (1) enumerating all the feature interactions or requiring human efforts; (2) utilizing the same IF to model all the feature interactions; (3) assigning the same embedding size to all the features. Our proposed AIM improves these models by solving such three downsides with AutoML techniques.

2.2 AutoML for CTR Models

AutoML for CTR models has been an active research area and there exist some works using AutoML techniques to devise automated methods for architecture design including embedding dimension search and feature interaction search.

Existing works leverage the AutoML to optimize the embeddings by searching embedding sizes for different features adaptively and automatically. NIS [20] and ESAPN [19] perform a hard selection strategy and use reinforcement learning (RL) to search for mixed feature embedding sizes automatically. On the contrary, soft selection strategies based on differentiable search algorithms (e.g., DARTS [23]) are proposed in AutoEmb [18] and AutoDim [17] by summing over the embeddings of the candidate sizes with learnable weights. The former leverages a controller network with Softmax layer to search the weights of different embedding sizes while the latter uses the Gumbel-softmax operation [31].

As stated earlier, for CTR models, feature interaction modeling is an important component, where the main tasks are to find which feature interactions are useful and decide how the interaction should be modeled. Some research works are proposed to explore these two questions. AutoFIS [1] automatically identifies and then selects important feature interactions for factorization models with a set of learnable architecture parameters. Besides, AutoGroup [21] proposes to generate some groups of features, such that their interactions of a given order are effective, which enables high-order feature interaction modeling. To determine suitable IFs, SIF [32] exploits the DARTS method to search proper interaction functions for matrix factorization. AutoFeature [16] applies an evolutionary algorithm with the Naive Bayes tree that recursively reduces the search space of IFs for feature interactions. However, AutoFeature trains excessive models for different structures to select the most appropriate structure, which requires a long period to search and a much high computational resource.

However, the above mentioned works solve one or two limitations (the three limitations are mentioned in Section 1 and also in Section 2.1) of the existing CTR models. None of them is able to solve all the three important issues in a unified framework. Our proposed AIM designs a gate-based unified framework and formulates the problem of searching proper embedding size, effective feature interactions as well as appropriate IFs as a continuous searching problem by incorporating architecture parameters that can be jointly optimized with neural network weights.

3 METHODOLOGY

In this section, we first formally define a family of popular CTR models (Factorization Model), with which our proposed AIM is able to work collaboratively to improve their performance. Then Automatic Interaction Machine (AIM) with three core components, namely, Feature Interaction Search (FIS), Interaction Function Search (IFS) and Embedding Dimension Search (EDS) is proposed to select significant feature interactions, appropriate IFs and necessary embedding dimensions automatically in a unified framework, as shown in Figure 1. Finally we discuss some concrete training details.

3.1 Factorization Model (Base Model)

First, we define factorization models:

Definition 1. **Factorization models** are the models where the interaction of several embeddings from different features is modeled into a real number by some functions such as inner product or neural network.

FM [13], DeepFM [4] and IPNN [12] (IPNN is one kind of PNN when the IF is inner product) are popular CTR models, which are in the family of factorization model. Therefore, we take FM, DeepFM, and IPNN as instances to formulate our algorithm and explore the performance on various datasets. Figure 2 presents the architectures of FM, DeepFM and IPNN models. FM consists of a *feature embedding layer* and a *feature interaction layer*. Besides these two layers, DeepFM and IPNN include an extra layer: *MLP layer*. The difference between DeepFM and IPNN is that feature interaction layer and MLP layer work in parallel in DeepFM while ordered in sequence in IPNN.

In the subsequent subsections, we will first elaborate on these layers. Then the details of how our proposed AIM working on the feature embedding layer and feature interaction layer are elaborated, i.e., selecting crucial feature interaction with proper interaction functions (IFs) and appropriate embedding sizes based on architecture parameters.

Feature Embedding Layer. In most CTR prediction tasks, data is collected in a multi-field categorical form[2]. A typical data pre-process is to transform each data instance into a high-dimensional sparse vector via one-hot or multi-hot encoding. A field is represented as a multi-hot encoding vector only when it is multivariate. The data instance can be represented as

$$\boldsymbol{x} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n],$$

where n is the number of fields and $\boldsymbol{x}_i$ is the one-hot or multi-hot encoding vector of the i^{th} field. Then, a feature embedding layer is used to transform the encoding vector into a low-dimensional vector via

$$\boldsymbol{e}_i = V_i \boldsymbol{x}_i, \tag{1}$$

where $V_i \in R^{d \times h_i}$ is the embedding matrix for the i^{th} field, h_i is vocabulary size of the i^{th} field and d is embedding size.

Then, the output of the feature embedding layer is represented as the concatenation of multiple embedding vectors:

$$\boldsymbol{E} = [\boldsymbol{e}_1, \boldsymbol{e}_2, ..., \boldsymbol{e}_n].$$

In the traditional paradigm, all feature embeddings in various fields have identical embedding sizes. However, not all features in distinct fields are equally predictive, i.e., some fields are informative while the other fields are not. Therefore, allocating the same embedding size to all features may lose the information of high predictive features while wasting memory on non-predictive features. To tackle this issue, we will later present EDS component to search different embedding sizes (by pruning useless embedding dimensions) for various features in the different fields.

Feature Interaction Layer. After transforming the features to a low-dimensional space, the feature interactions can be modeled in such a space with the feature interaction layer. Various interaction functions (IFs) can be used to model distinctive interactive signals, such as inner product, outer product and kernel product:

$$\begin{cases} f_{\texttt{INNER}}\ (\boldsymbol{e}_i, \boldsymbol{e}_j) = \langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle \\ f_{\texttt{OUTER}}\ (\boldsymbol{e}_i, \boldsymbol{e}_j) = \boldsymbol{e}_i \otimes \boldsymbol{e}_j \\ f_{\texttt{KERNEL}}\ (\boldsymbol{e}_i, \boldsymbol{e}_j) = \langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle_\phi \end{cases}, \tag{2}$$

where $\langle \cdot, \cdot \rangle$ is the inner product, $\otimes$ is the outer product and $\langle \cdot, \cdot \rangle_\phi$ is the kernel product (with the kernel ϕ). Specially, we further divide the kernel product into matrix kernel product, vector kernel product, scalar kernel product according to the shape of kernel ϕ.

2. Features in the numerical form are usually transformed into categorical form by bucketing.

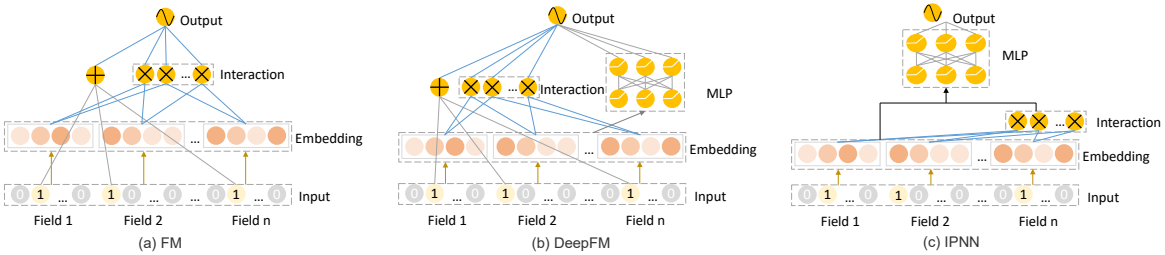

Fig. 2. Architectures of FM, DeepFM and IPNN.

Take the inner product as an example, we introduce how the feature interaction layer works. The inner product of all the pair-wise feature interactions is calculated:

$$[\langle \boldsymbol{e}_1, \boldsymbol{e}_2 \rangle, \langle \boldsymbol{e}_1, \boldsymbol{e}_3 \rangle, ..., \langle \boldsymbol{e}_{n-1}, \boldsymbol{e}_n \rangle], \tag{3}$$

In FM, the output of the feature interaction layer is:

$$l_{\text{FM}} = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \sum_{i=1}^{n} \sum_{j>i}^{n} \langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle. \tag{4}$$

However, the inner product operation may not be able to learn the interaction information of all the pairwise features. That is to say, various interaction functions (IFs) are needed when modeling different feature interactions, as also stated in [32]. Taking this concern into consideration, we extend FM model with multiple IFs as

$$l_{\text{FM\_EXTEND}} = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \sum_{i=1}^{n} \sum_{j>i}^{n} \sum_{k=1}^{m} f_k \left(\boldsymbol{e}_i, \boldsymbol{e}_j \right), \tag{5}$$

where f_k is a predefined IF (such as inner product, outer product, kernel product) and m is the number of IFs.

In Equation 5, all the feature interactions are assumed to be equally important and therefore are designed to contribute equally to the prediction. Whereas not all the feature interactions are equally predictive and useless interactions may even degrade the performance. Therefore, we propose the FIS component to select important feature interactions automatically and efficiently.

In addition, Equation 5 assumes k different IFs are all needed and contribute equally to model each feature interaction. As stated earlier, different IFs are needed when modeling individual feature interactions. To select suitable IFs for each feature interaction, we propose the IFS component, which will also be elaborated later.

Besides pairwise feature interactions, we also target at identifying effective high-order feature interactions. Formally, we define p^{th}-order feature interaction (i.e., the combination of p fields) as:

$$\sum_{q \in \psi_p} f_k \left(\boldsymbol{e}_{q_1}, \boldsymbol{e}_{q_2}, \cdots, \boldsymbol{e}_{q_p} \right), \tag{6}$$

where ψ_p contains all p^{th}-order interactions ($|\psi_p| = \mathcal{C}_n^p$) and q is one such p^{th}-order interaction. $f_k(\cdot)$ is the k^{th} high-order IF (high-order IFs are expanded by 2^{nd}-order IFs and the specific expansions are described in Section 4.2.3).

The complexity of feature interaction layer with p^{th}-order interaction is $O(n^p)$, exponential to the number of fields n, which makes searching high-order feature interactions in a brute-force manner unaffordable in practice. To tackle the efficiency issue, we propose a method to select high-order feature interactions with quadratic complexity, which will be illustrated in Section 3.2.

MLP Layer. MLP layer consists of several fully connected layers with activation functions, which learn the implicit non-linear relationship among features. The output of one such layer is

$$\boldsymbol{a}^{(l+1)} = \text{relu}(W^{(l)} \boldsymbol{a}^{(l)} + \boldsymbol{b}^{(l)}), \tag{7}$$

where $\boldsymbol{a}^{(l)}, W^{(l)}, \boldsymbol{b}^{(l)}$ are the input, weight and bias of the l^{th} layer, respectively. The activation function is $\text{relu}(z) = \max(0, z)$. $\boldsymbol{a}^{(0)}$ is the input to MLP layers and $\boldsymbol{a}^{(L)} = \texttt{MLP}(\boldsymbol{a}^{(0)})$, where L is the depth of MLP layer.

Output Layer. FM model has no MLP layer and connects the feature interaction layer with prediction layer directly:

$$\hat{y}_{\text{FM}} = \text{sigmoid}(l_{\text{FM}}) = \frac{1}{1 + \exp(-l_{\text{FM}})}, \tag{8}$$

where $\hat{y}_{\text{FM}}$ is the predicted CTR. DeepFM combines feature interaction layer and MLP layers in parallel as

$$\hat{y}_{\text{DeepFM}} = \text{sigmoid}(l_{\text{FM}} + \texttt{MLP}(\boldsymbol{E})). \tag{9}$$

While in IPNN, MLP layer is subsequent to feature interaction layer as

$$\hat{y}_{\text{IPNN}} = \text{sigmoid}(\texttt{MLP}([\boldsymbol{E}, l_{\text{FM}}])). \tag{10}$$

Note that the MLP layer of IPNN can also serve as a re-weighting of the different feature interactions to capture their relative importance. This is also the reason that IPNN has a higher capacity than FM and DeepFM. However, with the IPNN formulation, one cannot retrieve the exact value corresponding to the relative contribution of each feature interaction. Therefore, the useless feature interactions in IPNN can be neither identified nor dropped, which brings extra noise and computation cost to the model.

Objective Function. FM, DeepFM, and IPNN share the same objective function, i.e., to minimize the cross-entropy of predicted values $\hat{y}$ and the labels y as

$$\mathcal{L}(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}), \tag{11}$$

where $y \in \{0, 1\}$ is the label and $\hat{y} \in [0, 1]$ is the predicted probability of $y = 1$.

3.2 Feature Interaction Search (FIS)

In this section, we present one of the three core components, namely FIS component, which is performed on the feature interaction layer of any factorization model to automatically identify different orders of essential feature interactions.

The overview of the FIS component is illustrated in Figure 3. To ease the presentation of FIS component, we introduce the *gate* operation to control whether to select

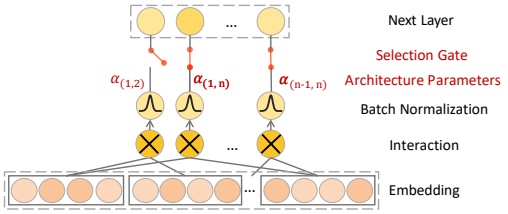

Fig. 3. Overview of FIS component.

a feature interaction, i.e., an open gate corresponds to selecting this feature interaction, while a closed gate results in dropping this interaction. The total number of gates corresponding to all the 2^{nd}-order feature interactions is $\mathcal{C}_n^2$. It is very challenging to find the optimal set of open gates in a brute-force way, as we face an incredibly huge $(2^{\mathcal{C}_n^2})$ space to search. In this work, we approach the problem from a different viewpoint. Instead of searching over a discrete set of open gates, we relax the choices to be continuous by introducing architecture parameters $\boldsymbol{\alpha}$, so that the relative importance of each feature interaction can be learned by gradient descent.

This architecture selection scheme based on gradient learning is inspired by DARTS [23], where the objective is to select one operation from a set of candidate operations in convolutional neural network (CNN) architecture. To be specific, we reformulate the interaction layer in factorization models (shown in Equation 4) as

$$l_{\text{FIS}} = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \sum_{i=1}^{n} \sum_{j>i}^{n} \alpha_{(i,j)} \langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle, \tag{12}$$

where $\boldsymbol{\alpha} = \{\alpha_{(1,2)}, \cdots, \alpha_{(n-1,n)}\}$ are the architecture parameters and their value $\alpha_{(i,j)}$ can represent the relative contribution of each feature interaction to the final prediction. Then, we can decide the gate status of each feature interaction by setting those unimportant ones (i.e., with zero $\alpha_{(i,j)}$ values) closed.

After the search, some unimportant interactions are thrown away automatically according to the architecture parameters $\boldsymbol{\alpha}$ and the new model with remained feature interactions can be re-trained.

High-order Feature Interaction Search. Note that, Equation 12 only formulates the case of selecting important 2^{nd}-order feature interactions. Besides the 2^{nd}-order, FIS component also aims to search high-order feature interactions. The number of all p^{th}-order feature interaction is $\mathcal{C}_n^p$, which is exponential to the number of fields n. Therefore, it is not practical to identify effective high-order feature interactions in a similar way as for 2^{nd}-order feature interactions (by enumerating all the feature interactions first).

To search effective high-order feature interactions efficient, we propose a two-step process to select p^{th}-order feature interactions from the selected $(p-1)^{th}$-order interactions, as follows.

- Step 1: Exploit $(p-1)^{th}$-order and first-order feature interaction set to generate a preliminary candidate pool $\mathcal{M}_p^{tmp}$ for potential effective p^{th}-order interactions.
- Step 2: Perform FIS component with architecture parameters to select effective p^{th}-order feature interactions $\mathcal{M}_p$, from $\mathcal{M}_p^{tmp}$ generated by Step 1.

This two-step search process is summarized in Algorithm 1. We elucidate the core part of this algorithm, i.e., *line 4* to *line 7*. For *line 4*, we combine top important $(p-1)^{th}$-order feature interaction set $\mathcal{M}_{p-1}^{top}$ and first-order feature interaction set $\mathcal{M}_1$ (that is each feature) in a Cartesian product manner to get the candidate pool $\mathcal{M}_p^{tmp}$ for p^{th}-order. For *line 5*, we leverage the FIS component to learn the relative importance of each p^{th}-order interaction by the architecture parameters. More specifically, we abandon the feature interactions with zero α value in $\mathcal{M}_p^{tmp}$ and retain the rest to get the final selected p^{th}-order interaction set $\mathcal{M}_p$ with their corresponding $\boldsymbol{\alpha_p}$. For *line 6-7*, preparing for the next round, we employ $\boldsymbol{\alpha_p}$ to sort $\mathcal{M}_p$ and get the top-k $(k \leq n)$ important p^{th}-order feature interaction to generate $(p+1)^{th}$-order feature interaction set.

Algorithm 1 High-order Feature Interaction Search

Require: field number n, the highest order P, top number k for pool selection
Ensure: the selected interactions $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \cdots, \mathcal{M}_P$
1: $\mathcal{M}_1 \leftarrow \{(\boldsymbol{e}_1), (\boldsymbol{e}_2), \cdots, (\boldsymbol{e}_n)\}$
2: $\mathcal{M}_1^{top} \leftarrow \mathcal{M}_1$
3: **for** $p \leftarrow 2$ to P **do**
4: $\quad \mathcal{M}_p^{tmp} \leftarrow \text{Combine}(\mathcal{M}_{p-1}^{top}, \mathcal{M}_1)$
5: $\quad \mathcal{M}_p, \boldsymbol{\alpha_p} \leftarrow \text{FIS}(\mathcal{M}_p^{tmp})$
6: $\quad$ Sort $\mathcal{M}_p$ by $\boldsymbol{\alpha_p}$
7: $\quad \mathcal{M}_p^{top} \leftarrow Top_k(\mathcal{M}_p)$
8: **end for**
9: **return** $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \cdots, \mathcal{M}_P$

Complexity Analysis. To select effective p^{th}-order feature interactions, there exists at most n feature interactions in $\mathcal{M}_{p-1}^{top}$, therefore $\mathcal{M}_p^{tmp}$ (which combining $\mathcal{M}_{p-1}^{top}$ and $\mathcal{M}_1$) has at most $O(n^2)$ p^{th}-order feature interactions. Both time and space complexity of feature interaction selection for each order by our method is $O(n^2)$, instead of $O(n^p)$ complexity by simple enumeration.

3.3 Interaction Function Search (IFS)

FIS component selects important feature interactions which are modeled with the same specific interaction function (IF). However, as stated earlier, not all the feature interactions can be modeled with the same IF. That is to say, individual feature interactions may need different IFs. Due to this reason, the second core component, IFS, is proposed to select appropriate IFs for each important feature interaction.

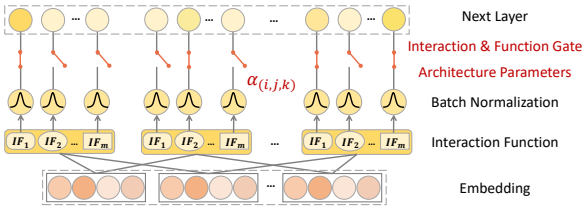

Fig. 4. The architecture of IFS component.

Similar to FIS, IFS component also leverages the *gate* operation to control the IF selection. For each feature interaction, one or more IFs from m candidate IFs are selected to learn the relationship of the features. As a special case, a feature interaction is abandoned if none of m IFs is selected. The architecture parameters $\boldsymbol{\alpha}'$ are introduced to get the relative importance of each interaction-IF pair, which can be

learned by gradient descent (in a similar way as in FIS). The overview of IFS component is presented in Figure 4.

We reformulate the interaction layer in the extend factorization models (shown in Equation 5) as

$$l_{\text{IFS}} = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \sum_{i=1}^{n} \sum_{j>i}^{n} \sum_{k=1}^{m} \alpha'_{(i,j,k)} f_k(\boldsymbol{e}_i, \boldsymbol{e}_j), \qquad (13)$$

where $\boldsymbol{\alpha'} = \{\alpha'_{(i,j,k)}\}$ are the architecture parameters. $\alpha'_{(i,j,k)}$ corresponding to the k^{th} IF for feature interaction (i, j). Those unimportant interaction-IF pairs (i.e., with zero $\alpha'_{(i,j,k)}$ values) will be thrown away automatically.

Function-wise Embeddings. In Equation 13, feature interaction modeling leverages shared embedding (i.e., each feature shares the same embedding for different IFs) to learn the latent effect over different IFs. However, the learning of shared feature embedding becomes difficult as one such embedding receives different gradient signals from individual learning spaces, where each learning space corresponds to an IF. Such different gradient signals cause learning conflict and lead to sub-optimal performance, as also observed in [33, 34].

To avoid such learning conflict among different IFs, IFS proposes to utilize the function-wise embeddings (FWEs), i.e., multiple embeddings are learned for each feature, where each of such embedding corresponds to one IF. With a specific IF, the corresponding embedding vectors are used to model the interactive correlations. Equipped with FWE, Equation 13 is updated as:

$$l^{\text{FWE}}_{\text{IFS}} = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \sum_{i=1}^{n} \sum_{j>i}^{n} \sum_{k=1}^{m} \alpha'_{(i,j,k)} f_k(\boldsymbol{e}_{ik}, \boldsymbol{e}_{jk}), \qquad (14)$$

where $\boldsymbol{e}_{ik}$ represents the embedding of the i^{th} feature for the k^{th} IF. In this way, each embedding only needs to learn the latent effect in one mapping space of a specific IF, such that learning conflict is avoided. In Section 4.4, we will show the superiority of function-wise embeddings over shared embedding. However, function-wise embeddings introduce more parameters, which leads to much more space cost. To reduce the space complexity, the third core component, EDS is proposed, which will be presented in Section 3.4.

After the architecture parameters $\boldsymbol{\alpha'}$ are learned, appropriate IFs are chosen for each feature interaction automatically. More specifically, zero value of $\alpha'_{(i,j,k)}$ indicates the interaction (i, j) with k^{th} IF is not effective and thus can be dropped. The other interaction-IF pairs with non-zero $\alpha'_{(i,j,k)}$ values are retained.

3.4 Embedding Dimension Search (EDS)

Although function-wise embeddings (FWEs) improve the performance, the model size is significantly larger than the case of allocating the same embedding size (as shared embedding) to all features. To deal with the excessive parameters, the third component, EDS, is proposed to optimize the model size without sacrificing performance by assigning low embedding size to non-predictive features, which is achieved by pruning redundant embedding dimensions for each feature.

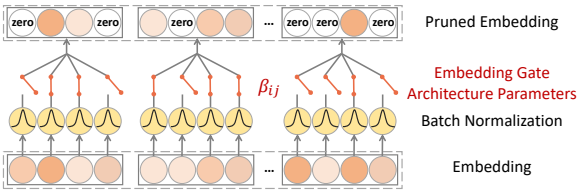

Fig. 5. The architecture of EDS component.

Similarly, a *gate* operation is introduced to determine whether the corresponding embedding dimension is selected for a field. The total number of the gates is $n \times d$, where n is the number of the fields and d is the largest embedding size. Here we make use of architecture parameters $\boldsymbol{\beta} \in \mathbb{R}^{n \times d}$ to turn the embedding dimension search problem into a continuous process which can be solved by gradient descent. The overview of EDS component is shown in Figure 5. Note that FWEs in the same field share the same set of architecture parameters.

During the embedding dimension search, each embedding dimension multiplies to the embedding architecture parameter $\boldsymbol{\beta}$ of the corresponding field, as

$$\boldsymbol{e}'_i = \boldsymbol{\beta_i} \cdot \boldsymbol{e}_i, \qquad (15)$$

where $\boldsymbol{\beta_i}$ is the embedding architecture parameter of the i^{th} field and it represents the relative importance among different dimensions in the i^{th} field.

The feature embedding layer concatenates the embedding vector $\boldsymbol{e}'_i$ of each feature, as

$$\boldsymbol{E}' = [\boldsymbol{e}'_1, \boldsymbol{e}'_2, ..., \boldsymbol{e}'_n].$$

Then $\boldsymbol{E}'$ feeds to the feature interaction layer for capturing interactive signals.

After the search, the embedding size of each feature field has been determined automatically according to $\boldsymbol{\beta}^*$. More specifically, EDS component prunes redundant dimensions (i.e., with zero β_{ij} value) and reserves necessary dimensions with their positions (since the positions also reflect the effect on various feature interactions). The embedding size for each field is

$$d_i = \sum_{j=1}^{d} \mathbb{1}\left[\beta_{ij} \neq 0\right]. \qquad (16)$$

Besides, the position set of the retained embedding dimensions for the i^{th} field is

$$\phi_i = \{j \mid \beta_{ij} \neq 0\ ,\ j \in [1, \cdots, d]\}. \qquad (17)$$

To derive new embeddings of adaptive sizes, we design a function $Map_i(\cdot)$ for the i^{th} field that maps from the position set of retained dimension ϕ_i to the new embedding position set $\{1, 2, \cdots, d_i\}$ in order as

$$Map_i : \phi_i \longrightarrow \{1, 2, \cdots, d_i\}, \qquad (18)$$

where d_i is the new embedding size of the i^{th} fields. Then we can re-train the model with length-adaptive embedding sizes. For each feature e_i in the i^{th} field, instead of initializing it with embedding size d, we initialize it with a shorter embedding size d_i.

As some IFs require embedding sizes of features to be the same (such as inner product), in such cases, we can trivially utilize Map_i to transform embeddings with size

d_i to ones with size d, by setting the dimensions that are not in the domain of Map_i with zero values. Note that such transformation with Map_i introduces no extra parameters therefore does not increase the space complexity.

3.5 Automatic Interaction Machine (AIM)

To summarize, FIS component searches for feature interactions; IFS component searches for appropriate IF for each feature interaction; and EDS component searches for proper embedding size for each feature. To fully exploit the advantages of these three components, we build the AIM framework by combining them, to select significant feature interactions, appropriate IFs and necessary embedding dimensions automatically in a unified framework.

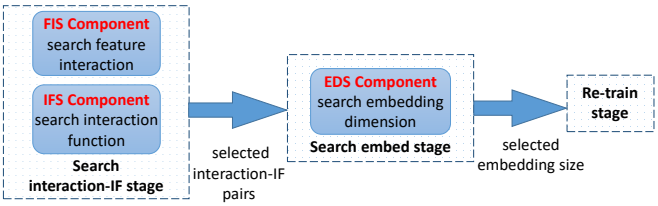

Fig. 6. The flow chart of AIM.

Specifically, AIM has the following three stages as shown in Figure 6:

- **Search interaction-IF stage.** Train the model and then use FIS component and IFS component to select useful interactions and appropriate IFs.
- **Search embed stage.** Train the model with selected interaction-IF pairs in the first stage, and then use EDS component to select a suitable embedding size for each field, by discarding useless embedding dimensions.
- **Re-train stage.** Re-train the model with selected interaction-IF pairs and learned embedding sizes in the first two stages.

Note that FIS component and IFS component can be trained together, however, it is inappropriate to combine the training process of them with EDS component due to the coupling problem between architecture parameters. Therefore, we have to divide the search process into Search interaction-IF stage and Search embed stage.

Transferability. The chosen Interaction-IF pairs by *gate* architecture learned from a simple model could be transferred to the state-of-the-art models to boost their performance. We will verify this transferability in Section 4.6.

3.6 Training Details

In this subsection, we present some training details in AIM, i.e., batch normalization and GRDA Optimizer. We take FIS component as an illustration example and similar training procedures are performed in IFS and EDS.

Batch Normalization. From the viewpoint of the overall neural network, the contribution of a feature interaction is measured by $\alpha_{(i,j)} \cdot \langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle$ (in Equation 12). Exactly the same contribution can be achieved by re-scaling this term as $(\frac{\alpha_{(i,j)}}{\eta}) \cdot (\eta \cdot \langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle)$, where η is a real number.

Since the value of $\langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle$ is jointly learned with $\alpha_{(i,j)}$, the coupling of their scale would lead to unstable estimation of $\alpha_{(i,j)}$, such that $\alpha_{(i,j)}$ can no longer represent the relative importance of $\langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle$. To solve this problem, we apply Batch Normalization (BN) [35] on $\langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle$ to eliminate its scale issue. BN has been adopted by training deep neural networks as a standard approach to achieve fast convergence and better performance. The way that BN normalizes values gives an efficient yet effective way to solve the coupling problem of $\alpha_{(i,j)}$ and $\langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle$.

The original BN normalizes the activated output with statistics information of a mini-batch. Specifically,

$$\hat{\boldsymbol{z}} = \frac{\boldsymbol{z}_{\text{IN}} - \boldsymbol{\mu}_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad \text{and} \quad \boldsymbol{z}_{\text{OUT}} = \theta \cdot \hat{\boldsymbol{z}} + \boldsymbol{\delta}, \tag{19}$$

where $\boldsymbol{z}_{\text{IN}}$, $\hat{\boldsymbol{z}}$ and $\boldsymbol{z}_{\text{OUT}}$ are input, normalized and output values of BN; $\boldsymbol{\mu}_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}$ are the mean and standard deviation values of $\boldsymbol{z}_{\text{IN}}$ over a mini-batch $\mathcal{B}$; θ and $\boldsymbol{\delta}$ are trainable *scale* and *shift* parameters of BN; ϵ is a constant for numerical stability.

To get stable estimation of $\alpha_{(i,j)}$, we set the *scale* and *shift* parameters to be 1 and 0 respectively. The BN operation on each feature interaction $\langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle$ is calculated as

$$\langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle_{\text{BN}} = \frac{\langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle - \mu_{\mathcal{B}}(\langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle)}{\sqrt{\sigma_{\mathcal{B}}^2(\langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle) + \epsilon}}, \tag{20}$$

where $\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}$ are the mean and standard deviation of $\langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle$ in mini-batch $\mathcal{B}$.

GRDA Optimizer. Generalized regularized dual averaging (GRDA) optimizer [26] is aimed to get a sparse deep neural network. To update α at each gradient step t with data Z_t we use the following equation:

$$\alpha_{t+1} = \arg\min_{\alpha} \{\alpha^T(-\alpha_0 + \gamma \sum_{i=0}^{t} \nabla L(\alpha_t; Z_{i+1})) + g(t,\gamma)\|\alpha\|_1 + \frac{1}{2}\|\alpha\|_2^2\} \tag{21}$$

where $g(t,\gamma) = c\gamma^{1/2}(t\gamma)^u$, and γ is the learning rate, c and μ are adjustable hyper-parameters to trade-off between accuracy and sparsity.

With the GRDA optimizer, the important feature interactions (or functions, embedding dimensions) will be retained, and the unnecessary ones will be abandoned according to the architecture parameters (the unimportant ones will be learned to zero). In this way, GRDA optimizer adaptively determines the remained ones, avoiding artificially setting hyper-parameters for deciding how many feature interactions (or functions, embedding dimensions) to be remained.

4 Experiments

In this section, we conduct extensive offline experiments[3] on two public benchmark datasets and a private dataset, as well as an online A/B test in the recommendation service of a mainstream app market, to evaluate the effectiveness of Automatic Interaction Machine (AIM). In particular, we answer the following research questions:

- **RQ1**: Can AIM outperform the state-of-the-art models with the selected interaction-IF pairs?
- **RQ2**: How do different components of AIM (e.g., FIS, IFS, EDS) contribute to the performance? Are the interactions, IFs and embedding dimensions selected by these components really important and valuable?

3. Repeatable code of all the experimental study and all hyper-parameters are available at https://github.com/zhuchenxv/AIM

- **RQ3**: How about the space and time complexity of AIM compared with other models?
- **RQ4**: Can the interaction-IF pairs selected from AIM be transferred to other models for boosting their performance?
- **RQ5**: Can AIM improve the performance of an existing model in a live recommender system?

4.1 Datasets

Experiments are conducted on the following two public datasets (Avazu and Criteo) and one private dataset (Huawei), whose statistics are summarized in Table 1. We follow the existing works [1, 9, 12, 16, 21] to process Avazu and Criteo datasets.

Avazu[4]: Avazu was released in the CTR prediction contest on Kaggle. 80% of randomly shuffled data is allotted to training and validation with 20% for testing.

Criteo[5]: Criteo contains one month of click logs with billions of data samples. We select "data 6-12" as training and validation set while select "day-13" for evaluation.

Huawei: The industrial dataset is sampled and collected from an app recommendation scenario of Huawei AppGallery for a week. The dataset contains 10 feature fields, including user behavior (user click list, *etc.*), app information (id, category, *etc.*), and context information (time, *etc.*).

TABLE 1
Dataset Statistics

Dataset	#instances	#features	#fields	pos ratio
Avazu	4×10^7	6×10^5	24	0.17
Criteo	1×10^8	1×10^6	39	0.50
Huawei	3×10^8	1×10^5	10	0.07

4.2 Experimental Settings

4.2.1 Baselines and Evaluation Metrics

We compare our propoed **AIM** with representative factorization models (i.e., FM [13], FwFM [36], AFM [14], FFM [27], DeepFM [4], IPNN [9]), AutoML-based model (AutoFeature [16], AutoGroup [21] and AutoFIS [1]) and some other interaction model (xDeepFM [22], FiGNN [37], AutoInt [38]). Note that AutoFIS use DeepFM as base model in our experiment.

We use the common evaluation metrics for CTR prediction: **AUC** and **Log loss**. All the experiments are repeated five times by changing the random seeds. The two-tailed unpaired t-test is performed to detect significant differences between our model and the best baseline.

4.2.2 Parameter Settings

In AIM model, we set the embedding size $d = 40$ for Avazu dataset and $d = 20$ for Criteo dataset. The MLP structure in two datasets are both $[700 \times 5, 1]$. With regard to GRDA parameters c and mu, we set $c = 0.05, mu = 0.6$ for Avazu dataset and $c = 0.005, mu = 0.9$ for Criteo dataset. The more detailed parameters for AIM model and the hyperparameters for other models can be shown in our code link[3].

4. http://www.kaggle.com/c/avazu-ctr-prediction
5. http://labs.criteo.com/downloads/download-terabyte-click-logs/

4.2.3 Implementation Details

In the *search interaction-IF stage*, we first train the model with $\boldsymbol{\alpha}'$ to seek important interaction-IF pairs. Then in the *search embed stage*, we train the model with $\boldsymbol{\beta}$ to search embedding dimensions. Finally, we re-train the model with the selected components. We consider 4 different common-used IFs, containing inner product, outer product, vector kernel product and scalar kernel product. The output of outer product is a $d \times d$ vector (d is the embedding size), and we use a linear layer to transform it into a single output. To reduce the complexity of outer product, we utilize $(\boldsymbol{w}_1^T \boldsymbol{e}_i)(\boldsymbol{w}_2^T \boldsymbol{e}_j) = \boldsymbol{w}_1^T \boldsymbol{e}_i \boldsymbol{e}_j^T \boldsymbol{w}_2$ to approximate outer product. Here $\boldsymbol{e}_i$, $\boldsymbol{e}_j$ are the embeddings and $\boldsymbol{w}_1$, $\boldsymbol{w}_2$ are learnable parameters.

For high-order interaction implementation such as p^{th}-order interaction, inner product, vector kernel product and scalar kernel product are easily to be extended by element-wise multiplication with p vectors. To expand the outer product into p^{th}-order, we employ $(\boldsymbol{w}_1^T \boldsymbol{e}_{q_1})(\boldsymbol{w}_2^T \boldsymbol{e}_{q_2}) \cdots (\boldsymbol{w}_p^T \boldsymbol{e}_{q_p})$ to calculate it ($q \in \psi_p$ where ψ_p contains all p^{th}-order interactions).

To implement high-order interaction selection, we manipulate top k (set $k = \lfloor n/2 \rfloor$) low-order interactions to construct a candidate pool (with at most $n^2/2$ interactions) first and the high-order interactions are chosen from this pool. This method is applied in both AutoFIS and AIM.

In the first two stages, the selected architecture parameters are optimized by GRDA optimizer and the other parameters are optimized by Adam optimizer. In the re-train stage, all the parameters are optimized by Adam optimizer.

Inspired by the transferability mentioned before, we sequentially introduce an MLP layer to learn more information and promote the performance in the re-train stage of AIM while this MLP layer does not exist in the search stage.

4.3 Overall Performance (RQ1)

Table 2 shows the overall performance of AIM and baselines on three datasets. Table 3 summarizes the performance and FI ratio (i.e., percentage of selected feature interaction) with different maximum interaction order in AutoFIS and AIM. From these two tables, we have the following observations.

1) Compared with the **Factorization Models** and **Other Interaction Models**, AIM has achieved significant improvement. Among these base models, IPNN leverages an MLP to model high-order implicit information over the feature embeddings and interaction, mostly achieving better performance than the others. However, AIM further improves IPNN in terms of AUC by 0.66% in Avazu and 0.21% in Criteo via performing automatic search of IFs and embedding dimensions.
2) In comparison with the existing representative **AutoML**-based models, AIM achieves the best performance. AutoFIS and AutoGroup only consider selecting useful feature interactions, ignoring the effectiveness of different IFs. Besides, none of these AutoML-based models take the embedding dimension search into consideration. In contrast, AIM proposes three components (i.e., FIS, IFS and EDS) to select significant feature interactions, appropriate IFs and necessary embedding dimensions automatically in a unified framework.

TABLE 2
Benchmark performance."Rel. Impr" is the relative AUC improvement over FM model.

Category	Model	Avazu			Criteo			Huawei		
		AUC	log loss	Rel. Impr.	AUC	log loss	Rel. Impr.	AUC	log loss	Rel. Impr.
Factorization Model	FM	0.7793	0.3805	0	0.7909	0.5500	0	0.8168	0.1939	0
	FwFM	0.7822	0.3784	0.37%	0.7948	0.5475	0.49%	0.8268	0.1907	1.22%
	AFM	0.7806	0.3794	0.17%	0.7913	0.5517	0.05%	0.8195	0.1934	0.33%
	FFM	0.7831	0.3781	0.49%	0.7980	0.5438	0.90%	0.8279	0.1900	1.36%
	DeepFM	0.7836	0.3776	0.55%	0.7991	0.5423	1.04%	0.8338	0.1878	2.08%
	IPNN	0.7868	0.3756	0.96%	0.8013	0.5401	1.31%	0.8363	0.1867	2.39%
Other Interaction Model	Fi-GNN	0.7853	0.3767	0.77%	0.8003	0.5410	1.19%	0.8376	0.1863	2.55%
	AutoInt	0.7847	0.3770	0.69%	0.8001	0.5413	1.16%	0.8371	0.1865	2.49%
	xDeepFM	0.7855	0.3766	0.80%	0.8006	0.5408	1.23%	0.8368	0.1865	2.45%
AutoML-based Model	AutoFeature	0.7904	0.3737	1.42%	0.8023	0.5390	1.44%	0.8365	0.1861	2.41%
	AutoGroup	0.7909	0.3732	1.49%	0.8026	0.5386	1.48%	0.8360	0.1860	2.35%
	AutoFIS	0.7883	0.3748	1.15%	0.8012	0.5402	1.30%	0.8373	0.1864	2.51%
AIM		**0.7920***	**0.3727***	1.63%	**0.8030***	**0.5379***	1.53%	**0.8385***	**0.1858***	2.66%

$*$ denotes statistically significant improvement (measured by t-test with p-value<0.005) over baselines with same order.

TABLE 3
Performance with different maximum interaction orders.
"FI ratio" is the remained ratio of feature interaction (FI).

Model	Avazu			Criteo		
	AUC	log loss	FI ratio	AUC	log loss	FI ratio
AutoFIS(2^{nd})	0.7852	0.3765	18%	0.8007	0.5406	9%
AIM(2^{nd})	0.7891	0.3742	16%	0.8022	0.5388	22%
AutoFIS(3^{rd})	0.7870	0.3755	5%	0.8010	0.5404	2%
AIM(3^{rd})	0.7912	0.3730	2%	0.8029	0.5381	1%
AutoFIS(4^{th})	0.7883	0.3748	0.6%	0.8012	0.5402	0.03%
AIM(4^{th})	0.7920	0.3727	0.3%	0.8030	0.5379	0.09%

3) From Table 3, we can observe that about 80% of the 2^{nd}-order interactions can be removed. As for high-order feature interaction selection, only about 2% of all the 3^{rd}-order feature interactions and about 0.3% of all the 4^{th}-order feature interactions are selected with significant performance improvement. With a small amount of high-order interactions integrated, the relative performance improvement of AIM (3^{rd}-order) over AIM (2^{nd}-order) is 0.27% and that of AIM (4^{th}-order) over AIM (2^{nd}-order) is 0.37% in terms of AUC.

4.4 Ablation Study (RQ2)

In this subsection, we will first present the effectiveness of different components (namely, FIS, IFS and EDS) in AIM. Then we conduct experiments on real data to analyze why these three components are valid.

4.4.1 Effectiveness of different components in AIM

To validate the effectiveness of individual components in AIM, we propose several variants. Recall that AIM has three core components: FIS, IFS, EDS, and we apply experiments on these variants to verify the effectiveness of these components. The relationship among different variants and their performance is presented in Table 4, from which we can get the following conclusions.

TABLE 4
Performance comparison of different variants.

Model	Component			Avazu		Criteo	
	FIS	IFS	EDS	AUC	log loss	AUC	log loss
DeepFM	-	-	-	0.7836	0.3776	0.7991	0.5423
AIM-IFS-EDS	√	×	×	0.7869	0.3755	0.8015	0.5399
AIM-EDS	√	√	×	0.7894	0.3740	0.8024	0.5387
AIM	√	√	√	0.7891	0.3742	0.8022	0.5388

1) Comparing AIM-IFS-EDS with DeepFM, we can observe that removing those useless interactions can not only simplified the model but also significantly boost the prediction accuracy. The relative performance improvement of AIM-IFS-EDS over DeepFM is 0.42% and 0.30% for Avazu and Criteo dataset respectively in terms of AUC, which demonstrates the effectiveness of FIS component.
2) Besides, the performance gap between AIM-EDS and AIM-IFS-EDS indicates that selecting proper IFs for each feature interaction conduces to better performance, which is the functionality of IFS component.
3) Comparing AIM-EDS and AIM, we can find that introducing EDS component in AIM for pruning redundant dimensions will not sacrifice the performance distinctly. However, EDS reduces model parameters significantly, by $\times$2.79 and $\times$3.16 in Avazu and Criteo dataset respectively, which will be shown in Section 4.4.4.

4.4.2 The Effectiveness of Selected Feature Interactions by FIS component

To see how well $\boldsymbol{\alpha}$ values are learned in FIS component, we analyze the relationship between $\boldsymbol{\alpha}$ values and **statistics_AUC**.

We define **statistics_AUC** to represent the importance of a feature interaction to the final prediction. For a given interaction, we construct a predictor only considering this interaction where the prediction of a test instance is the statistical CTR (`#downloads/#impressions`) of specified feature interaction in the training set. Then the AUC of this predictor is defined as statistics_AUC with respect to this given feature interaction. Higher statistics_AUC indicates a more important role of this feature interaction in prediction.

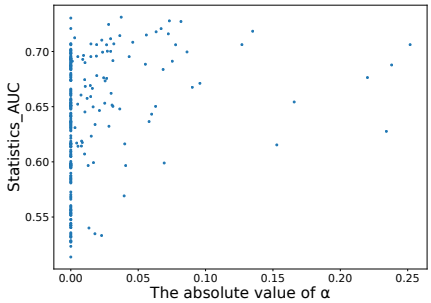

Fig. 7. Relationship between statistics_AUC and α value for each 2^{nd}-order interaction

As shown in Figure 7, we can find that most of the feature interactions selected by FIS component (with high absolute $\boldsymbol{\alpha}$ value) have high statistics_AUC, but not all feature interactions with high statistics_AUC are selected. That is because the information in these interactions may

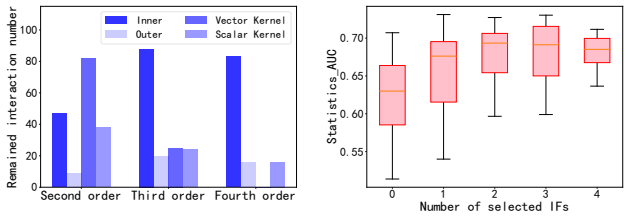

Fig. 8. The Effectiveness of Selected Interaction Functions by IFS. The left sub-graph is the selection frequency of each IF while the right sub-graph is the relationship between statistics_AUC and number of selected IFs for each 2^{nd}-order interaction.

already exist in other interactions which are selected by our model.

To evaluate the effectiveness of the selected interactions by FIS component, we select the top N 2^{nd}-order feature interactions by FIS component and by statistics_AUC separately. We re-train two models with these two sets of interactions and compare their performance. The experimental result shows that, compared with the model by statistics_AUC with the same computational cost, the model with selected interactions by FIS component has improved AUC from 0.7804 to 0.7831 (log loss from 0.3794 to 0.3778), which demonstrates the superiority of FIS component.

4.4.3 The Effectiveness of Selected Interaction Functions by IFS component

To demonstrate that IFS component is able to find suitable IFs for individual interactions, we depict the number of selected IFs with respect to different orders of interactions on Avazu dataset, presented in the left sub-graph of Figure 8. The following observations can be concluded. (1) Each IF occupies a certain proportion, which verifies the need of different IFs for individual interactions. The performance superiority of IFS component validates that suitable IFs are identified for different interactions. (2) Inner product takes the majority for high-order interactions and ranks at the second place for 2^{nd}-order interaction. (3) Vector kernel product is the most popular IF for 2^{nd}-order interaction but is unpopular for high-order interaction. (4) Scalar kernel product and outer product take only a small part of the overall selection, but still hold an indispensable part.

Furthermore, we explore the relationship between the number of selected IFs and statistics_AUC for each feature interaction, which is presented in the right sub-graph of Figure 8. The figure indicates that more important feature interactions are likely to retain more IFs. Moreover, as the number of selected IFs increases, the improvement of statistics_AUC is gradually disappeared. This is reasonable, because it is more likely to be overfitting when more IFs are performed for a feature interaction. In other words, it is not the best to keep all the IFs. Instead, selecting suitable IFs for individual interactions is more appropriate.

4.4.4 The Effectiveness of Selected Embedding Dimensions by EDS component

In this section, we analyze why the reserved embedding size of a given feature determined by EDS component is reasonable. Before that, we first show the effectiveness of EDS.

As mentioned in Section 3.4, function-wise embeddings (FWE) improve the performance compared with shared embedding (SE), therefore we compare the performance of AIM-EDS (SE) and AIM-EDS (FWE). As the model size is significantly larger if the embedding size of FWE keeps the same as that of SE, which is the motivation of proposing EDS component. We further compare AIM (FWE) and AIM-EDS (FWE) to validate the effectiveness of EDS.

TABLE 5
Performance and parameters comparison. "params" does not include the pruned parameters and it is count as 10^7.

Model	Avazu			Criteo		
	AUC	log loss	params	AUC	log loss	params
AIM-EDS (SE)	0.7870	0.3754	2.7	0.8015	0.5399	2.8
AIM-EDS (FWE)	0.7894	0.3740	10.6	0.8024	0.5387	9.8
AIM (FWE)	0.7891	0.3742	3.8	0.8022	0.5388	3.1

The above mentioned experimental comparisons are presented in Table 5, from which We can get the following conclusions.

1) The comparison between AIM-EDS (SE) and AIM-EDS (FWE) verifies that multiple embeddings for different IFs can boost the performance but lead to much more parameters.
2) Comparing AIM (FWE) with AIM-EDS (FWE), equipping EDS component achieves almost the same high performance as AIM-EDS (FWE) but needs only one third of the parameters required by AIM-EDS (FWE). The parameters of AIM-EDS (FWE) are slightly more than that of AIM-EDS (SE) in a comparable amount.

To see the reserved embedding size of each field, we summarize these numbers in Table 6, together with the vocabulary size of each field (i.e., the number of features in a field) on Avazu dataset. The correlation between vocabulary size and reserved embedding size presented in Table 6 indicates that the fields with larger vocabulary size tend to achieve larger embedding size. That is because these fields may be more informative and predictive to the task. However, as can also be observed, vocabulary size is not the only factor to determine the embedding size. Other unknown factors need to be learned by machine learning models, such as EDS component in AIM.

TABLE 6
Correlation between vocabulary size and reserved embedding size

vocabulary	embedding	vocabulary	embedding	vocabulary	embedding
4	1	10	25	426	54
4	11	24	6	2417	33
5	1	24	35	3135	55
7	1	28	5	3487	56
7	8	60	23	4002	55
7	16	67	6	5925	81
8	1	166	32	101449	22
9	1	252	15	523672	98

4.5 Space and Time Complexity (RQ3)

In this section, we discuss the space and time complexity of AIM with 2^{nd}-order interactions and high-order interactions, respectively. In order to facilitate calculation, we choose Avazu (with 645,195 features) to analyze the memory usage, training and inference efficiency of AIM in comparison with other models. To compare the training and inference efficiency, we train these models for the whole training set in one epoch and test these models with 200,000 instances (batch size=2,000) on an NVIDIA 1080Ti GPU.

4.5.1 Complexity Analysis for 2^{nd}-order interaction

First, we analyze the complexity of all the models with 2^{nd}-order interaction modeling and present the results in Table 7. Note that we only consider the parameters and training time in the re-train stage of AutoFIS and AIM model. Because the useful feature interactions, effective interaction-IF pairs and reserved embedding dimensions can be searched once for all, while the model training over new data only needs to perform the re-train stage based on the searched results in the first two search stages. To complete the complexity analysis of AIM, the complexity analysis of the search stages is presented in Section 4.5.2.

TABLE 7
Space and time complexity comparison of models with 2^{nd}-order interaction. "train" is the training time for the whole training set in one epoch and "inference" is the inference time for 200,000 samples.

Model	params (10^7)	train (min)	inference (s)
FM	2.6	3.3	0.53
FFM	6.0	3.7	0.25
DeepFM	2.9	5.0	0.88
IPNN	2.9	5.5	0.96
AutoFIS	2.9	4.1	0.59
AIM-EDS	10.6	8.7	0.81
AIM	3.8	9.0	0.83

As for the space complexity, the number of parameters of AutoFIS and its base model (DeepFM) are very similar. AIM-EDS consumes the largest amount of memory as it utilizes multiple embeddings for each feature. By integrating EDS component, AIM reduces the amount of parameters to a comparable level as other models without sacrificing model performance (as already discussed in Section 4.4.4).

In terms of training and inference efficiency, we find that AutoFIS is apparently faster than the base model (DeepFM) both in training and inference time, because removing those useless interactions contributes to accelerating the model training and inference, with higher performance. Although the training time of AIM is acceptably larger than the other models, the inference of AIM is very efficient (i.e., even more efficient than DeepFM). Such time complexity makes it possible to deploy AIM in an industrial system.

4.5.2 Complexity Analysis for high-order interaction

In this section, we analyze the complexity of AIM with high-order interactions. We consider the training and inference time as time complexity. The training time contains three parts: *search interaction-IF stage*, *search embed stage* and *re-train stage*. Also, we take the parameters into account to compare space complexity.

TABLE 8
Space and time complexity for AIM with high-order interactions. "train" is the training time for the whole training set in one epoch and "inference" is the inference time for 200,000 samples.

Model	params (10^7)	train (min)	inference(s)
AIM(2^{nd})	3.8	9.0	0.83
AIM(3^{rd})	7.1	14.8	1.35
AIM(4^{th})	9.9	19.5	1.72

From Table 8 and Table 9 we can find that, thanks to the selection algorithm for high-order feature interaction proposed in Section 3.2 which reduces the time complexity of exploring p^{th}-order interactions from $O(n^p)$ to $O(n^2)$, the overhead of introducing high-order interactions in terms of training time and space complexity is acceptable. It takes about 155 minutes in total for AIM to search important 2^{nd}-, 3^{rd}- and 4^{th}-order feature interactions with appropriate IFs and proper embedding sizes for each field with *a single GPU*. The same decisions will take the human engineers dozens of days or weeks to make by analysis and experiments.

TABLE 9
Training time of AIM for different stages. It is the training time for the whole training set in one epoch and counted by min.

Model	search interaction-IF	search embed	re-train
AIM(2^{nd})	13.0	9.2	9.0
AIM(3^{rd})	20.5	17.1	14.8
AIM(4^{th})	29.4	22.2	19.5

4.6 Transferability of the Selected Interaction-IF pairs by AIM (RQ4)

TABLE 10
Performance of transferring interaction-IF pairs selected by AIM to other models.

Model	Avazu		Criteo	
	AUC	log loss	AUC	log loss
DeepFM	0.7836	0.3776	0.7991	0.5423
DeepFM+IF(2^{nd})	0.7858	0.3762	0.7999	0.5415
DeepFM+IF(3^{rd})	0.7868	0.3758	0.8004	0.5411
DeepFM+IF(4^{th})	**0.7872**	**0.3755**	**0.8005**	**0.5410**
PNN	0.7868	0.3756	0.8013	0.5401
PNN+IF(2^{nd})	0.7893	0.3741	0.8024	0.5387
PNN+IF(3^{rd})	0.7915	0.3728	0.8028	0.5382
PNN+IF(4^{th})	**0.7921**	**0.3725**	**0.8029**	**0.5381**

In this subsection, we investigate whether the important feature interactions with appropriate IFs learned by AIM could be transferred to other models for boosting their performance. We apply the searched interaction-IF pairs to two benchmark methods, i.e., DeepFM [4] and PNN [12], to explore its transferability.

As shown in Table 10, using 2^{nd}-order feature interactions and IFs selected by our model achieves much better performance in both DeepFM and IPNN, with around 16% of all the 2^{nd}-order interactions in Avazu dataset and around 20% in Criteo dataset. Moreover, the promotion is more significant by using high-order interactions with IFs. More specifically, with 4^{th}-order interactions and corresponding IFs, the performance is improved by 0.46% for DeepFM and 0.67% for PNN, respectively. Both evidences verify the transferability of the selected feature interactions with chosen IFs in AIM.

4.7 Deployment & Online Experiments (RQ5)

Online experiments are conducted in the recommender system of a mainstream app market to verify the superior performance of our model AIM, where hundreds of millions of daily active users generate hundreds of billions of user log events every day in the form of implicit feedback such as browsing, clicking and downloading apps. In the online serving system, hundreds of candidate apps that are most likely to be downloaded by the users are retrieved from the universal app pool. Then these candidate apps are ranked by a fine-tuned ranking model (such as DeepFM, AIM) before presenting to users. To guarantee user experience, the overall latency of the above-mentioned candidate selection and ranking is required to be within a few milliseconds.

Because of the longer training time and more complicated functions of AIM, we simplify it and put a simplified version to deploy on our recommender system for the trade-off between efficiency and performance. The commercial model is deployed in a cluster, where each node is with 48 core Intel Xeon CPU E5-2670 (2.30GHZ), 400GB RAM and as well as 2 NVIDIA TESLA V100 GPU cards.

Specifically, a three-week A/B test is conducted in a major list of an app recommendation scenario. Our baseline in online experiments is DeepFM, which is a strong baseline due to its extraordinary accuracy and high efficiency. For the control group, users are randomly selected and presented with recommendation results generated by DeepFM. For the experimental group (7% users), users are presented with recommendation apps generated by our AIM model.

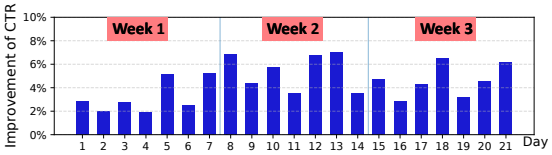

Fig. 9. Online experimental results of CTR.

Figure 9 shows the improvement of the experimental group over the control group in terms of CTR (`#downloads/#impressions`). We can observe that the average improvement of CTR is **4.4%** (statistically significant), which brings enormous commercial profits. These results demonstrate the magnificent effectiveness of our proposed model in industrial applications.

5 Conclusion

In this work, we propose Automatic Interaction Machine (AIM) with three core components, namely, Feature Interaction Search (FIS), Interaction Function Search (IFS) and Embedding Dimension Search (EDS) to automatically select significant feature interactions, appropriate IFs and necessary embedding dimensions in a unified framework. The proposed AIM is easy to implement with marginal search costs, and the performance improvement is significant in two benchmark datasets and one private dataset. Our model has been deployed onto the training platform of a mainstream app market recommendation service, with significant economic profit demonstrated.

Acknowledgements

The SJTU team is supported by "New Generation of AI 2030" Major Project (2018AAA0100900), Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102) and National Natural Science Foundation of China (62076161, 62177033). The work is also sponsored by Huawei Innovation Research Program.

References

[1] B. Liu, C. Zhu, G. Li, W. Zhang, J. Lai, R. Tang, X. He, Z. Li, and Y. Yu, "Autofis: Automatic feature interaction selection in factorization models for click-through rate prediction," in *KDD*, 2020.

[2] W. Zhang, J. Qin, W. Guo, R. Tang, and X. He, "Deep learning for click-through rate estimation," *IJCAI Survey Track*, 2021.

[3] H. Cheng, L. Koc, J. Harmsen, T. Shaked, T. D. Chandra, H. Aradhye, G. Anderson, G. S. Corrado, W. Chai, M. Ispir *et al.*, "Wide & deep learning for recommender systems," in *DLRS@RecSys*, 2016.

[4] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He, "Deepfm: A factorization-machine based neural network for CTR prediction," in *IJCAI*, 2017.

[5] G. Zhou, X. Zhu, C. Song, Y. Fan, H. Zhu, X. Ma, Y. Yan, J. Jin, H. Li, and K. Gai, "Deep interest network for click-through rate prediction," in *KDD*, 2018.

[6] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *RecSys*, 2016.

[7] W. Zhang, T. Du, and J. Wang, "Deep learning over multi-field categorical data: A case study on user response prediction," *ECIR*, 2016.

[8] K. Hornik, M. B. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," in *Neural Networks*, 1989.

[9] Y. Qu, B. Fang, W. Zhang, R. Tang, M. Niu, H. Guo, Y. Yu, and X. He, "Product-based neural networks for user response prediction over multi-field categorical data," *TOIS*, 2018.

[10] S. Shalev-Shwartz, O. Shamir, and S. Shammah, "Failures of gradient-based deep learning," in *ICML*, 2017.

[11] S. Rendle, W. Krichene, L. Zhang, and J. Anderson, "Neural collaborative filtering vs. matrix factorization revisited," in *RecSys*, 2020.

[12] Y. Qu, H. Cai, K. Ren, W. Zhang, Y. Yu, Y. Wen, and J. Wang, "Product-based neural networks for user response prediction," in *ICDM*, 2016.

[13] S. Rendle, "Factorization machines," in *ICDM*, 2010.

[14] J. Xiao, H. Ye, X. He, H. Zhang, F. Wu, and T. Chua, "Attentional factorization machines: Learning the weight of feature interactions via attention networks," in *IJCAI*, 2017.

[15] X. He and T. Chua, "Neural factorization machines for sparse predictive analytics," in *SIGIR*, 2017.

[16] F. Khawar, X. Hang, R. Tang, B. Liu, Z. Li, and X. He, "Autofeature: Searching for feature interactions and their architectures for click-through rate prediction," in *CIKM*, 2020.

[17] X. Zhao, H. Liu, H. Liu, J. Tang, W. Guo, J. Shi, S. Wang, H. Gao, and B. Long, "Memory-efficient embedding for recommendations," *CoRR*, 2020.

[18] X. Zhao, C. Wang, M. Chen, X. Zheng, X. Liu, and J. Tang, "Autoemb: Automated embedding dimensionality search in streaming recommendations," *CoRR*, 2020.

[19] H. Liu, X. Zhao, C. Wang, X. Liu, and J. Tang, "Automated embedding size search in deep recommender systems," in *SIGIR*, 2020.

[20] M. R. Joglekar, C. Li, M. Chen, T. Xu, X. Wang, J. K. Adams, P. Khaitan, J. Liu, and Q. V. Le, "Neural input search for large scale recommendation models," in *KDD*, 2020.

[21] B. Liu, N. Xue, H. Guo, R. Tang, S. Zafeiriou, X. He, and Z. Li, "Autogroup: Automatic feature grouping for modelling explicit high-order feature interactions in ctr prediction," in *SIGIR*, 2020.

[22] J. Lian, X. Zhou, F. Zhang, Z. Chen, X. Xie, and G. Sun,

"xDeepFM: Combining explicit and implicit feature interactions for recommender systems," in *KDD*, 2018.
[23] H. Liu, K. Simonyan, and Y. Yang, "DARTS: differentiable architecture search," in *ICLR*, 2019.
[24] C. He, H. Ye, L. Shen, and T. Zhang, "Milenas: Efficient neural architecture search via mixed-level reformulation," in *CVPR*, 2020.
[25] G. Li, X. Zhang, Z. Wang, Z. Li, and T. Zhang, "Stacnas: Towards stable and consistent optimization for differentiable neural architecture search," *arXiv*, 2019.
[26] S.-K. Chao and G. Cheng, "A generalization of regularized dual averaging and its dynamics," in *CoRR*, 2019.
[27] Y. Juan, Y. Zhuang, W. Chin, and C. Lin, "Field-aware factorization machines for CTR prediction," in *RecSys*, 2016.
[28] M. Blondel, A. Fujino, N. Ueda, and M. Ishihata, "Higher-order factorization machines," in *NIPS*, 2016.
[29] B. Liu, R. Tang, Y. Chen, J. Yu, H. Guo, and Y. Zhang, "Feature generation by convolutional neural network for click-through rate prediction," in *WWW*, 2019.
[30] R. Wang, B. Fu, G. Fu, and M. Wang, "Deep & cross network for ad click predictions," in *ADKDD@KDD*, 2017.
[31] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *ICLR*, 2017.
[32] Q. Yao, X. Chen, J. Kwok, Y. Li, and C.-J. Hsieh, "Efficient neural interaction function search for collaborative filtering," in *WWW*, 2020.
[33] W. Guo, R. Tang, H. Guo, J. Han, W. Yang, and Y. Zhang, "Order-aware embedding neural network for CTR prediction," in *SIGIR*, 2019.
[34] G. Zhou, W. Bian, K. Wu, L. Ren, Q. Pi, Y. Zhang, C. Xiao, X. Sheng, N. Mou, X. Luo, C. Zhang, X. Qiao, S. Xiang, K. Gai, X. Zhu, and J. Xu, "CAN: revisiting feature co-action for click-through rate prediction," *CoRR*, 2020.
[35] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *ICML*, 2015.
[36] J. Pan, J. Xu, A. L. Ruiz, W. Zhao, S. Pan, Y. Sun, and Q. Lu, "Field-weighted factorization machines for click-through rate prediction in display advertising," in *WWW*, 2018.
[37] Z. Li, Z. Cui, S. Wu, X. Zhang, and L. Wang, "Fi-gnn: Modeling feature interactions via graph neural networks for ctr prediction," in *CIKM*, 2019.
[38] W. Song, C. Shi, Z. Xiao, Z. Duan, Y. Xu, M. Zhang, and J. Tang, "Autoint: Automatic feature interaction learning via self-attentive neural networks," in *CIKM*, 2019.