AutoFIS: Automatic Feature Interaction Selection in Factorization Models for Click-Through Rate Prediction*

Bin Liu[†*], Chenxu Zhu[‡*], Guilin Li[†*], Weinan Zhang[‡]
Jincai Lai[†], Ruiming Tang[†], Xiuqiang He[†], Zhenguo Li[†], Yong Yu[‡]

[†]Huawei Noah's Ark Lab [‡]Shanghai Jiao Tong University
liubinbh@126.com; {zhuchenxv,wnzhang}@sjtu.edu.cn; {liguilin2,tangruiming}@huawei.com

ABSTRACT

Learning feature interactions is crucial for click-through rate (CTR) prediction in recommender systems. In most existing deep learning models, feature interactions are either manually designed or simply enumerated. However, enumerating all feature interactions brings large memory and computation cost. Even worse, useless interactions may introduce noise and complicate the training process. In this work, we propose a two-stage algorithm called Automatic Feature Interaction Selection (AutoFIS). AutoFIS can automatically identify important feature interactions for factorization models with computational cost just equivalent to training the target model to convergence. In the *search stage*, instead of searching over a discrete set of candidate feature interactions, we relax the choices to be continuous by introducing the architecture parameters. By implementing a regularized optimizer over the architecture parameters, the model can automatically identify and remove the redundant feature interactions during the training process of the model. In the *re-train stage*, we keep the architecture parameters serving as an attention unit to further boost the performance. Offline experiments on three large-scale datasets (two public benchmarks, one private) demonstrate that AutoFIS can significantly improve various FM based models. AutoFIS has been deployed onto the training platform of Huawei App Store recommendation service, where a 10-day online A/B test demonstrated that AutoFIS improved the DeepFM model by 20.3% and 20.1% in terms of CTR and CVR respectively.

CCS CONCEPTS

• **Information systems → Recommender systems**;

KEYWORDS

Recommendation; Factorization Machine; Feature Selection; Neural Architecture Search

*Co-first authors with equal contributions. Bin Liu is now affiliated with ByteDance. Weinan Zhang and Ruiming Tang are the corresponding authors. This work is sponsored by Huawei Innovation Research Program and NSFC (61702327,61772333).

KDD '20, August 23–27, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7998-4/20/08…$15.00
https://doi.org/10.1145/3394486.3403314

ACM Reference Format:
Biu Liu, Chenxu Zhu, Guilin Li, Weinan Zhang, Jincai Lai, Ruiming Tang, Xiuqiang He, Zhenguo Li, Yong Yu. 2020. AutoFIS: Automatic Feature Interaction Selection in Factorization Models for Click-Through Rate Prediction. In *26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'20), August 23–27, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3394486.3403314

1 INTRODUCTION

Click-through rate (CTR) prediction is crucial in recommender systems, where the task is to predict the probability of the user clicking on the recommended items (e.g., movie, advertisement) [6, 32]. Many recommendation decisions can then be made based on the predicted CTR. The core of these recommender systems is to extract significant low-order and high-order feature interactions.

Explicit feature interactions can significantly improve the performance of CTR models [2, 8, 11, 22]. Early collaborative filtering recommendation algorithms, such as matrix factorization (MF) [16] and factorization machine (FM) [27], extract second-order information with a bi-linear learning model.

However, not all interactions are conducive to performance. Some tree-based methods have been proposed to find useful intersections automatically. Gradient boosting decision tree (GBDT) [5, 11] tries to find the interactions with higher gradients of the loss function. AutoCross [21] searches effective interactions in a tree-structured space. But tree models can only explore a small fraction of all possible feature interactions in recommender systems with multi-field categorical data [25], so their exploration ability is restricted.

In the meantime, Deep Neural Network (DNN) models [7, 31] are proposed. Their representational ability is stronger and they could explore most of the feature interactions according to the universal approximation property [12]. However, there is no guarantee that a DNN naturally converges to any expected functions using gradient-based optimization. A recent work proves the insensitive gradient issue of DNN when the target is a large collection of uncorrelated functions [25, 28]. Simple DNN models may not find the proper feature interactions. Therefore, various complicated architectures have been proposed, such as Deep Interest Network (DIN) [32], Deep Factorization Machine (DeepFM) [8], Product-based Neural Network (PNN) [24, 25], and Wide & Deep [6]. **Factorization Models** (specified in Definition 1), such as FM, DeepFM, PNN, Attention Factorization Machine (AFM) [30], Neural Factorization Machine (NFM) [10], have been proposed to adopt a feature extractor to explore explicit feature interactions.

However, all these models are simply either enumerating all feature interactions or requiring human efforts to identify important feature interactions. The former always brings large memory and computation cost to the model and is difficult to be extended into high-order interactions. Besides, useless interactions may bring unnecessary noise and complicate the training process [28]. The latter, such as identifying important interactions manually in Wide & Deep [6], is of high labor cost and risks missing some counter-intuitive (but important) interactions.

If useful feature interactions can be identified beforehand in these factorization models, the models can focus on learning over them without having to deal with useless feature interactions. Through removing the useless or even harmful interactions, we would expect the model to perform better with reduced computation cost.

To automatically learn which feature interactions are essential, 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 is either specified beforehand by expert knowledge [6] or set as all open [8, 18]. From a data-driven point of view, whether open or closed a gate should depend on the contribution of each feature interaction to the final prediction. Apparently, those contributing little should be closed to prevent introducing extra noise to model learning. However, it is an NP-Hard problem to find the optimal set of open gates for model performance, as we face an incredibly huge ($2^{C_m^2}$, with m the number of feature fields, if we only consider 2^{nd}-order feature interactions) space to search.

Inspired by the recent work DARTS [9, 17, 20] for neural architecture search, we propose a two-stage method AutoFIS for automatic selecting low-order and high-order feature interactions in *factorization models*. In the *search stage*, instead of searching over a discrete set of candidate feature interactions, we relax the choices to be continuous by introducing a set of architecture parameters (one for each feature interaction) so that the relative importance of each feature interaction can be learned by gradient descent. The architecture parameters are jointly optimized with neural network weights by GRDA optimizer [3] (an optimizer which is easy to produce a sparse solution) so that the training process can automatically abandon unimportant feature interactions (with zero values as the architecture parameters) and keep those important ones. After that, in the *re-train stage*, we select the feature interactions with non-zero values of the architecture parameters and re-train the model with the selected interactions while keeping the architecture parameters as attention units instead of indicators of interaction importance.

Extensive experiments are conducted on three large-scale datasets (two are public benchmarks, and the other is private). Experimental results demonstrate that AutoFIS can significantly improve the CTR prediction performance of factorization models on all datasets. As AutoFIS can remove about 50%-80% 2^{nd}-order feature interactions, original models can always achieve improvement on efficiency. We also apply AutoFIS for 3^{rd}-order interaction selection by learning the importance of each 3^{rd}-order feature interaction. Experimental results show that with about 1%–10% of the 3^{rd}-order interactions selected, the AUC of factorization models can be improved by 0.1%–0.2% without introducing much computation cost. The results show a promising direction of using AutoFIS for automatic high-order feature interaction selection. Experiments also demonstrate that important 2^{nd}- and 3^{rd}-order feature interactions, identified by AutoFIS in factorization machine, can also greatly boost the performance of current state-of-the-art models, which means we can use a simple model for interaction selection and apply the selection results to other models. Besides, we analyze the effectiveness of feature interactions selected by our model on real data and synthetic data. Furthermore, a ten-day online A/B test is performed in a Huawei App Store recommendation service, where AutoFIS yielding recommendation model achieves improvement of CTR by 20.3% and CVR by 20.1% over DeepFM, which contributes a significant business revenue growth.

To summarize, the main contributions of this paper can be highlighted as follows:

(1) We empirically verify that removing the redundant feature interactions is beneficial when training factorization models.
(2) We propose a two-stage algorithm **AutoFIS** to automatically select important low-order and high-order feature interactions in factorization models. In the *search stage*, AutoFIS can learn the relative importance of each feature interaction via architecture parameters within one full training process. In the *re-train stage*, with the unimportant interactions removed, we re-train the resulting neural network while keeping architecture parameters as attention units to help the learning of the model.
(3) Offline experiments on three large-scale datasets demonstrate the superior performance of AutoFIS in factorization models. Moreover, AutoFIS can also find a set of important high-order feature interactions to boost the performance of existing models without much computation cost introduced. A ten-day online A/B test shows that AutoFIS improves DeepFM model by approximately 20% on average in terms of CTR and CVR.

2 RELATED WORK

CTR prediction is generally formulated as a binary classification problem [19]. In this section we briefly review factorization models for CTR prediction and AutoML models for recommender systems.

Factorization machine (FM) [27] projects each feature into a low-dimensional vector and models feature interactions by inner product, which works well for sparse data. Field-aware factorization machine (FFM) [14] further enables each feature to have multiple vector representations to interact with features from other fields.

Recently, deep learning models have achieved state-of-the-art performance on some public benchmarks [19, 29]. Several models use MLP to improve FM, such as Attention FM [30], Neural FM [10]. Wide & Deep [6] jointly trains a wide model for artificial features and a deep model for raw features. DeepFM [8] uses an FM layer to replace the wide component in Wide & Deep. PNN [24] uses MLP to model the interaction of FM layer and feature embeddings while PIN [25] introduces a network-in-network architecture to model pairwise feature interactions with sub-networks rather than simple inner product operations in PNN and DeepFM. Note that all existing factorization models simply enumerate all 2^{nd}-order feature interactions which contain many useless and noisy interactions.

Gradient boosting decision tree (GBDT) [5] is a method to do feature engineering and search interactions by decision tree algorithm. Then the transformed feature interactions can be fed into

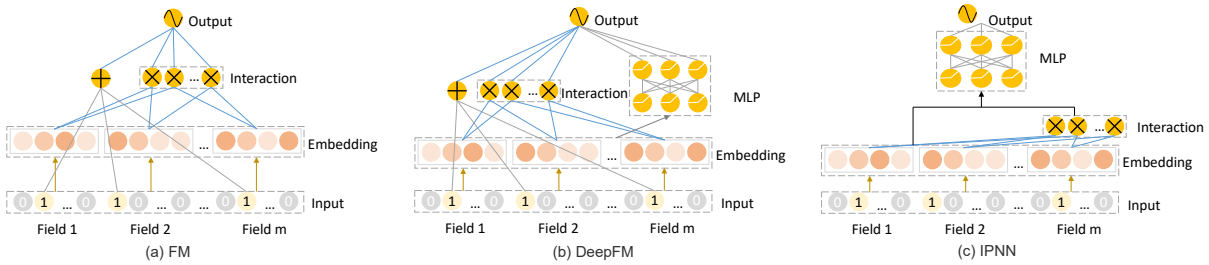

Figure 1: Architectures of FM, DeepFM and IPNN

to logistic regression [11] or FFM [15]. In practice, tree models are more suitable for continuous data but not for high-dimensional categorical data in recommender system because of the low usage rate of categorical features [25].

In the meantime, there exist some works using AutoML techniques to deal with the problems in recommender system. AutoCross [21] is proposed to search over many subsets of candidate features to identify effective interactions. This requires training the whole model to evaluate the selected feature interactions, but the candidate sets are incredibly many: i.e., there are $2^{C_m^2}$ candidate sets for a dataset with m fields for just 2^{nd}-order feature interactions. Thus AutoCross accelerates by two aspects of approximation: (i) it greedily constructs local-optimal feature sets via beam search in a tree structure, and (ii) it evaluates the newly generated feature sets via field-aware logistic regression. Due to such two approximations, the high-order feature interactions extracted from AutoCross may not be useful for deep models. Compared with AutoCross, our proposed AutoFIS only needs to perform the search stage once to evaluate the importance of all feature interactions, which is much more efficient. Moreover, the learned useful interactions will improve the deep model as they are learned and evaluated in this deep model directly.

Recently, one-shot architecture search methods, such as DARTS [20], have become the most popular neural architecture search (NAS) algorithms to efficiently search network architectures [1]. In recommender systems, such methods are utilized to search proper interaction functions for collaborative filtering models [26]. The model in [26] focuses on identifying proper interaction functions for feature interactions while our model focuses on searching and keeping important feature interactions. Inspired by the recent work DARTS for neural architecture search, we formulate the problem of searching the effective feature interactions as a continuous searching problem by incorporating architecture parameters. Different from DARTS using two-level optimization to optimize the architecture parameters and the model weights alternatively and iteratively with the training set and the validation set, we use one-level optimization to train these two types of parameters jointly with all data as the training set. We analyze their difference theoretically in Section 3.2, and compare their performance in the Experiments Section.

3 METHODOLOGY

In this section, we describe the proposed AutoFIS, an algorithm to select important feature interactions in factorization models automatically.

3.1 Factorization Model (Base Model)

First, we define factorization models:

Definition 3.1. **Factorization models** are the models where the interaction of several embeddings from different features is modeled into a real number by some operation such as inner product or neural network.

We take FM, DeepFM, and IPNN as instances to formulate our algorithm and explore the performance on various datasets. Figure 1 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 model 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 brief the feature embedding layer and feature interaction layer in FM. To work with DeepFM and IPNN model, the MLP layer and output layer are also formalized. Then the detail of how our proposed AutoFIS working on the feature interaction layers is elaborated, i.e., selecting important feature interactions based on architecture parameters.

Feature Embedding Layer. In most CTR prediction tasks, data is collected in a multi-field categorical form[1]. A typical data preprocess 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. A data instance can be represented as

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

where m is the number of fields and $\boldsymbol{x}_i$ is the one-hot or multi-hot encoding vector of the i-th field. A feature embedding layer is used to transform an encoding vector into a low-dimensional vector as

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

where $V_i \in R^{d \times n_i}$ is the a matrix, n_i is the number of feature values in the i-th field and d is the dimension of low-dimensional vectors.

- If $\boldsymbol{x}_i$ is a one-hot vector with j-th element $\boldsymbol{x}_i[j] = 1$, then the representation of $\boldsymbol{x}_i$ is V_i^j.
- If $\boldsymbol{x}_i$ is a multi-hot vector with $\boldsymbol{x}_i[j] = 1$ for $j = i_1, i_2, \cdots, i_k$ and the embeddings of these elements are $\{V_i^{i1}, V_i^{i2}, \cdots, V_i^{ik}\}$, then the representation of $\boldsymbol{x}_i$ is the sum or average of these embeddings [7].

The output of the feature embedding layer is then the concatenation of multiple embedding vectors as

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

[1]Features in numerical form are usually transformed into categorical form by bucketing.

Feature Interaction Layer. After transforming the features to low-dimensional space, the feature interactions can be modeled in such a space with the feature interaction layer. First, the inner product of the pairwise feature interactions is calculated:

$$[\langle \boldsymbol{e}_1, \boldsymbol{e}_2 \rangle, \langle \boldsymbol{e}_1, \boldsymbol{e}_3 \rangle, ..., \langle \boldsymbol{e}_{m-1}, \boldsymbol{e}_m \rangle], \tag{2}$$

where $\boldsymbol{e}_i$ is the feature embedding of i-th field, $\langle \cdot, \cdot \rangle$ is the inner product of two vectors. The number of pair-wise feature interactions in this layer is C_m^2.

In FM and DeepFM models, the output of the feature interaction layer is:

$$l_{fm} = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \sum_{i=1}^{m} \sum_{j>i}^{m} \langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle. \tag{3}$$

Here, all the feature interactions are passed to the next layer with equal contribution. As pointed in Section 1 and will be verified in Section 4, not all the feature interactions are equally predictive and useless interactions may even degrade the performance. Therefore, we propose the AutoFIS algorithm to select important feature interactions efficiently.

To study whether our methods can be used to identify important high-order interactions, we define the feature interaction layer with 3^{rd}-order interactions (i.e., combination of three fields) as:

$$l_{fm}^{3rd} = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \sum_{i=1}^{m} \sum_{j>i}^{m} \langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle + \sum_{i=1}^{m} \sum_{j>i}^{m} \sum_{t>j}^{m} \langle \boldsymbol{e}_i, \boldsymbol{e}_j, \boldsymbol{e}_t \rangle. \tag{4}$$

MLP Layer. MLP Layer consists of several fully connected layers with activation functions, which learns the relationship and combination of features. The output of one such layer is

$$\boldsymbol{a}^{(l+1)} = \text{relu}(W^{(l)} \boldsymbol{a}^{(l)} + \boldsymbol{b}^{(l)}), \tag{5}$$

where $\boldsymbol{a}^{(l)}, W^{(l)}, \boldsymbol{b}^{(l)}$ are the input, model weight, and bias of the l-th layer. Activation $\text{relu}(z) = \max(0, z)$. $\boldsymbol{a}^{(0)}$ is the input layer and $\texttt{MLP}(\boldsymbol{a}^{(0)}) = \boldsymbol{a}^{(h)}$, where h is the depth of MLP layer $\texttt{MLP}$.

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

$$\hat{y}_{\texttt{FM}} = \text{sigmoid}(l_{fm}) = \frac{1}{1 + \exp(-l_{fm})}, \tag{6}$$

where $\hat{y}_{\texttt{FM}}$ is the predicted CTR.

DeepFM combines feature interaction layer and MLP layers in parallel as

$$\hat{y}_{\texttt{DeepFM}} = \text{sigmoid}(l_{fm} + \texttt{MLP}(\boldsymbol{E})). \tag{7}$$

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

$$\hat{y}_{\texttt{IPNN}} = \text{sigmoid}(\texttt{MLP}([\boldsymbol{E}, l_{fm}])). \tag{8}$$

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. We would show in the following subsections and Section 4 that how the proposed method AutoFIS could improve IPNN.

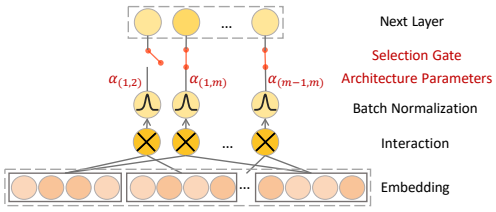

Figure 2: Overview of AutoFIS

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

$$\mathcal{L}(y, \hat{y}_{\texttt{M}}) = -y \log \hat{y}_{\texttt{M}} - (1 - y) \log(1 - \hat{y}_{\texttt{M}}), \tag{9}$$

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

3.2 AutoFIS

AutoFIS automatically selects useful feature interactions, which can be applied to the feature interaction layer of any factorization model. In this section, we elaborate on how it works. AutoFIS can be split into two stages: *search stage* and *re-train stage*. In the search stage, AutoFIS detects useful feature interactions; while in the re-train stage, the model with selected feature interactions is re-trained.

Search Stage. To facilitate the presentation of the algorithm, we introduce the *gate* operation to control whether to select a feature interaction: an open gate corresponds to selecting a feature interaction, while a closed gate results in a dropped interaction. The total number of gates corresponding to all the 2^{nd}-order feature interactions is C_m^2. It is very challenging to find the optimal set of open gates in a brute-force way, as we face an incredibly huge $(2^{C_m^2})$ 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. The overview of the proposed AutoFIS is illustrated in Figure 2.

This architecture selection scheme by gradient learning is inspired by DARTS [20], 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 3) as

$$l_{\texttt{AutoFIS}} = \langle w, x \rangle + \sum_{i=1}^{m} \sum_{j>i}^{m} \alpha_{(i,j)} \langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle, \tag{10}$$

where $\boldsymbol{\alpha} = \{\alpha_{(1,2)}, \cdots, \alpha_{(m-1,m)}\}$ are the architecture parameters. In the *search stage* of AutoFIS, $\alpha_{(i,j)}$ values are learned in such a way that $\alpha_{(i,j)}$ 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.

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 10). 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) [13] 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,

$$\hat{\boldsymbol{z}} = \frac{\boldsymbol{z}_{in} - \boldsymbol{\mu}_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \quad \text{and} \quad \boldsymbol{z}_{out} = \theta \cdot \hat{\boldsymbol{z}} + \boldsymbol{\beta}, \tag{11}$$

where $\boldsymbol{z}_{in}$, $\hat{\boldsymbol{z}}$ and $\boldsymbol{z}_{out}$ are input, normalized and output values of BN; $\boldsymbol{\mu}_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}$ are the mean and standard deviation values of $\boldsymbol{z}_{in}$ over a mini-batch $\mathcal{B}$; θ and $\boldsymbol{\beta}$ 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

$$\langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle_{BN} = \frac{\langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle - \mu_{\mathcal{B}}(\langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle)}{\sqrt{\sigma^2_{\mathcal{B}}(\langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle) + \epsilon}}, \tag{12}$$

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 [3, 4] 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:

$$\alpha_{t+1} = \arg\min_{\alpha} \{ \alpha^T (-\alpha_0 + \gamma \sum_{i=0}^{t} \nabla L(\alpha_t; Z_{i+1}) + g(t, \gamma) \|\alpha\|_1 + 1/2 \|\alpha\|_2^2 \} \tag{13}$$

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.

In the search stage, we use GRDA optimizer to learn the architecture parameters $\boldsymbol{\alpha}$ and get a sparse solution. Those unimportant feature interactions (i.e., with zero $\alpha_{(i,j)}$ values) will be thrown away automatically. Other parameters are learned by Adam optimizer, as normal.

One-level Optimization. To learn the architecture parameters $\alpha_{(i,j)}$ in the *search stage* of AutoFIS, we propose to optimize $\boldsymbol{\alpha}$ jointly with all the other network weights $\boldsymbol{v}$ (such as $\boldsymbol{w}$ in Equation 3 and $W^{(l)}, \boldsymbol{b}^{(l)}$ in Equation 5). This is different from DARTS. DARTS treats $\boldsymbol{\alpha}$ as higher-level decision variables and the network weights as lower-level variables, then optimizes them with a bi-level optimization algorithm. In DARTS, it is assumed that the model can select the operation only when the network weights are properly learned so that $\boldsymbol{\alpha}$ can "make its proper decision". In the context of AutoFIS formulation, this means that we can decide whether a gate should be open or closed after the network weights are properly trained, which leads us back to the problem of fully training $2^{C_m^2}$ models to make the decision. To avoid this issue, DARTS proposes to approximate the optimal value of the network weights with only one gradient descent step and train $\boldsymbol{\alpha}$ and $\boldsymbol{v}$ iteratively.

We argue that the inaccuracy of this approximation might downgrade the performance. Therefore, instead of using bi-level optimization, we propose to optimize $\boldsymbol{\alpha}$ and $\boldsymbol{v}$ jointly with one-level optimization. Specifically, the parameters $\boldsymbol{\alpha}$ and $\boldsymbol{v}$ are updated together with gradient descent using the training set by descending on $\boldsymbol{\alpha}$ and $\boldsymbol{v}$ based on

$$\partial_v \mathcal{L}_{train}(v_{t-1}, \alpha_{t-1}) \quad \text{and} \quad \partial_\alpha \mathcal{L}_{train}(v_{t-1}, \alpha_{t-1}). \tag{14}$$

In this setting, $\boldsymbol{\alpha}$ and $\boldsymbol{v}$ can explore their design space freely until convergence, and $\boldsymbol{\alpha}$ is learned to serve as the contribution of individual feature interactions. In Section 4, we would show the superiority of one-level optimization over two-level optimization.

Re-train Stage. After the training of the *search stage*, some unimportant interactions are thrown away automatically according to the architecture parameters $\boldsymbol{\alpha}^*$ in search stage. We use $\mathcal{G}_{(i,j)}$ to represent the gate status of feature interaction $\langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle$ and set $\mathcal{G}_{(i,j)}$ as 0 when $\alpha^*_{(i,j)} = 0$; otherwise, we set $\mathcal{G}_{(i,j)}$ as 1. In the *re-train stage*, the gate status of these unimportant feature interactions are fixed to be closed permanently.

After removing these unimportant interactions, we re-train the new model with $\boldsymbol{\alpha}$ kept in the model. Specifically, we replace the feature interaction layer in Equation 3 with

$$l^{re}_{fm} = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \sum_{i=1}^{m} \sum_{j>i}^{m} \alpha_{(i,j)} \mathcal{G}_{(i,j)} \langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle. \tag{15}$$

Note here $\alpha_{(i,j)}$ no longer serves as an indicator for deciding whether an interaction should be included in the model (as in *search stage*). Instead, it serves as an attention unit for the architecture to learn the relative importance of the kept feature interaction. In this stage, we do not need to select the feature interactions. Therefore, all parameters are learned by Adam optimizer.

4 EXPERIMENTS

In this section, we conduct extensive offline experiments[2] on two benchmark public datasets and a private dataset, as well as online A/B test, to answer the following questions:

- **RQ1**: Could we boost the performance of factorization models with the selected interactions by AutoFIS?
- **RQ2**: Could interactions selected from simple models be transferred to the state-of-the-art models to reduce their inference time and improve prediction accuracy?
- **RQ3**: Are the interactions selected by AutoFIS really important and useful?
- **RQ4**: Can AutoFIS improve the performance of existing models in a live recommender system?
- **RQ5**: How do different components of our AutoFIS (e.g., BN) contribute to the performance?

4.1 Datasets

Experiments are conducted for the following two public datasets (Avazu and Criteo) and one private dataset:

Avazu[3]: Avazu was released in the CTR prediction contest on Kaggle. 80% of randomly shuffled data is allotted to training and

[2] Repeatable experiment code: https://github.com/zhuchenxv/AutoFIS

[3] http://www.kaggle.com/c/avazu-ctr-prediction

Table 1: Benchmark performance: "time" is the inference time for 2 million samples. "top" represents the percentage of feature interactions kept for 2^{nd} / 3^{rd} order interaction. "cost" contain the GPU time of the search and re-train stage. "Rel. Impr." is the relative AUC improvement over FM model. Note: FFM has a lower time and cost due to its smaller embedding size limited by GPU memory constraint.

Model	Avazu						Criteo					
	AUC	log loss	top	time (s)	search + re-train cost (min)	Rel. Impr.	AUC	log loss	top	time (s)	search + re-train cost (min)	Rel. Impr.
FM	0.7793	0.3805	100%	0.51	0 + 3	0	0.7909	0.5500	100%	0.74	0 + 11	0
FwFM	0.7822	0.3784	100%	0.52	0 + 4	0.37%	0.7948	0.5475	100%	0.76	0 + 12	0.49%
AFM	0.7806	0.3794	100%	1.92	0 + 14	0.17%	0.7913	0.5517	100%	1.43	0 + 20	0.05%
FFM	0.7831	0.3781	100%	0.24	0 + 6	0.49%	0.7980	0.5438	100%	0.49	0 + 39	0.90%
DeepFM	0.7836	0.3776	100%	0.76	0 + 6	0.55%	0.7991	0.5423	100%	1.17	0 + 16	1.04%
GBDT+LR	0.7721	0.3841	100%	0.45	8 + 3	-0.92%	0.7871	0.5556	100%	0.62	40 + 10	-0.48%
GBDT+FFM	0.7835	0.3777	100%	2.66	6 + 21	0.54%	0.7988	0.5430	100%	1.68	9 + 57	1.00%
AutoFM(2nd)	0.7831*	0.3778*	29%	**0.23**	4 + 2	0.49%	0.7974*	0.5446*	51%	**0.48**	14 + 9	0.82%
AutoDeepFM(2nd)	**0.7852***	**0.3765***	**24%**	0.48	7 + 4	0.76%	**0.8009***	**0.5404***	**28%**	0.69	22 + 11	1.26%
FM(3rd)	0.7843	0.3772	100%	5.70	0 + 21	0.64%	0.7965	0.5457	100%	8.21	0 + 72	0.71%
DeepFM(3rd)	0.7854	0.3765	100%	5.97	0 + 23	0.78%	0.7999	0.5418	100%	13.07	0 + 125	1.14%
AutoFM(3rd)	0.7860*	0.3762*	**25% / 2%**	**0.33**	22 + 5	0.86%	0.7983*	0.5436*	35% / 1%	**0.63**	75 + 15	0.94%
AutoDeepFM(3rd)	**0.7870***	**0.3756***	21% / 10%	0.94	24 + 10	0.99%	**0.8010***	**0.5404***	**13% / 2%**	0.86	128 + 17	1.28%

∗ denotes statistically significant improvement (measured by t-test with p-value<0.005) over baselines with same order. AutoFM compares with FM and AutoDeepFM compares with all baselines.

validation with 20% for testing. Categories with less than 20 times of appearance are removed for dimensionality reduction.

Criteo[4]: Criteo contains one month of click logs with billions of data samples. We select "data 6-12" as training and validation set while selecting "day-13" for evaluation. To counter label imbalance, negative down-sampling is applied to keep the positive ratio roughly at 50%. 13 numerical fields are converted into one-hot features through bucketing, where the features in a certain field appearing less than 20 times are set as a dummy feature "other".

Private: Private dataset is collected from a game recommendation scenario in Huawei App Store. The dataset contains app features (e.g., ID, category), user features (e.g., user's behavior history) and context features. Statistics of all the datasets are summarized in Table 2.

Table 2: Dataset Statistics

Dataset	#instances	#dimension	#fields	pos ratio
Avazu	4×10^7	6×10^5	24	0.17
Criteo	1×10^8	1×10^6	39	0.50
Private	2×10^8	3×10^5	29	0.02

4.2 Experimental Settings

4.2.1 Baselines and Evaluation Metrics. We apply AutoFIS to FM [27] and DeepFM [8] models to show its effectiveness (denoted as **AutoFM** and **AutoDeepFM**, respectively). We compare it with GBDT-based methods (GBDT+LR [11], GBDT+FFM [15]) and Factorization Machine models (AFM [30], FwFM [23], FFM [14], IPNN [25]). Due to its huge computational costs and the unavailability of the source code, we do not compare our models with AutoCross [21].

The common evaluation metrics for CTR prediction are **AUC** (Area Under ROC) and **Log loss** (cross-entropy).

4.2.2 Parameter Settings. To enable any one to reproduce the experimental results, we have attached all the hyper-parameters for each model in the supplementary material.

4.2.3 Implementation Details. Selecting 2^{nd}-order feature interactions for AutoFM and AutoDeepFM, in the search stage, we first train the model with $\boldsymbol{\alpha}$ and $\boldsymbol{v}$ jointly on all the training data. Then we remove those useless interactions and re-train our model.

To implement AutoFM and AutoDeepFM for 3^{rd}-order feature interaction selection, we reuse the selected 2^{nd}-order interactions in Equation 15 and enumerate all the 3^{rd}-order feature interactions in the search stage to learn their importance. Finally, we re-train our model with the selected 2^{nd}- and 3^{rd}-order interactions.

Note that in the search stage, the architecture parameters $\boldsymbol{\alpha}$ are optimized by GRDA optimizer and other parameters $\boldsymbol{v}$ are optimized by Adam optimizer. In the re-train stage, all parameters are optimized by Adam optimizer.

Table 3: Performance in Private Dataset. "Rel. Impr." is the relative AUC improvement over FM model.

Model	AUC	log loss	top	Rel. Impr
FM	0.8880	0.08881	100%	0
FwFM	0.8897	0.08826	100%	0.19%
AFM	0.8915	0.08772	100%	0.39%
FFM	0.8921	0.08816	100%	0.46%
DeepFM	0.8948	0.08735	100%	0.77%
AutoFM(2nd)	0.8944*	0.08665*	37%	0.72%
AutoDeepFM(2nd)	**0.8979***	**0.08560***	**15%**	1.11%

∗ denotes statistically significant improvement (measured by t-test with p-value<0.005). AutoFM compares with FM and AutoDeepFM compares with all baselines.

4.3 Feature Interaction Selection by AutoFIS (RQ1)

Table 1 summarizes the performance of AutoFM and AutoDeepFM by automatically selecting 2^{nd}- and 3^{rd}-order important interactions on Avazu and Criteo datasets and Table 3 reports their performance on Private dataset. We can observe:

(1) For Avazu dataset, 71% of the 2^{nd}-order interactions can be removed for FM and 76% for DeepFM. Removing those useless interactions can not only make the model faster at inference time: the inference time of AutoFM(2nd) and AutoDeepFM(2nd) is apparently less than FM and DeepFM; but also significantly increase the prediction accuracy: the relative performance improvement of AutoFM(2nd) over FM is 0.49% and that of AutoDeepFM(2nd) over DeepFM is 0.20% in terms of AUC. Similar improvement can also be drawn from the other datasets.

(2) For high-order feature interaction selection, only 2% – 10% of all the 3^{rd}-order feature interactions need to be included in the model. The inference time of AutoFM(3rd) and AutoDeepFM(3rd) is much less than that of FM(3rd) and DeepFM(3rd) (which is comparable to FM and DeepFM). Meanwhile, the accuracy is significantly improved by removing unimportant 3^{rd}-order feature interactions, i.e., the relative performance improvement of AutoFM(3rd) over FM(3rd) is 0.22% and that of AutoDeepFM(3rd) over DeepFM(3rd) is 0.20% in terms of AUC on Avazu. Observations on Criteo are similar.

[4] http://labs.criteo.com/downloads/download-terabyte-click-logs/

(3) All such performance boost could be achieved with marginal time cost (for example, it takes 24 minutes and 128 minutes for AutoDeepFM(3rd) to search important 2^{nd}- and 3^{rd}-order feature interactions in Avazu and Criteo with a single GPU card). The same result might take the human engineers many hours or days to achieve by identifying such important feature interactions manually.

Note that directly enumerating the 3^{rd}-order feature interactions in FM and DeepFM enlarges the inference time about 7 to 12 times, which is unacceptable in industrial applications.

4.4 Transferability of the Selected Feature Interactions (RQ2)

Table 4: Performance of transferring interactions selected by AutoFM to IPNN. AutoIPNN(2nd) indicates IPNN with 2^{nd}-order interactions selected by AutoFM(2nd) and AutoIPNN(3rd) indicates IPNN with 2^{nd}- and 3^{rd}-order interactions selected by AutoFM(3rd).

Model	Avazu			Criteo		
	AUC	log loss	time(s)	AUC	log loss	time(s)
IPNN	0.7868	0.3756	0.91	0.8013	0.5401	1.26
AutoIPNN(2nd)	0.7869	0.3755	0.58	0.8015	0.5399	0.76
AutoIPNN(3rd)	**0.7885***	**0.3746***	0.71	**0.8019***	**0.5392***	0.86

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

In this subsection, we investigate whether the feature interactions learned by AutoFM (which is a simple model) could be transferred to the state-of-the-art models such as IPNN to boost their performance. As shown in Table 4, using 2^{nd}-order feature interactions selected by AutoFM (namely AutoIPNN(2nd)) achieves comparable performance to IPNN, with around 30% and 50% of all the interactions in Avazu and Criteo. Moreover, the performance is significantly improved by using both 2^{nd}- and 3^{rd}-order feature interactions (namely AutoIPNN(3rd)) selected by AutoFM. Both evidences verify the transferability of the selected feature interactions in AutoFM.

4.5 The Effectiveness of Feature Interaction Selected by AutoFIS (RQ3)

In this subsection, we will discuss the effectiveness of feature interaction selected by AutoFIS. We conduct experiments on real data and synthetic data to analyze it.

4.5.1 The Effectiveness of selected feature interaction on Real Data. 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 statistics_AUC with respect to this given feature interaction. Higher statistics_AUC indicates a more important role of this feature interaction in prediction. Then we visualize the relationship between statistics_AUC and α value.

As shown in Figure 3, we can find that most of the feature interactions selected by our model (with high absolute α 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 also exist in other interactions which are selected by our model.

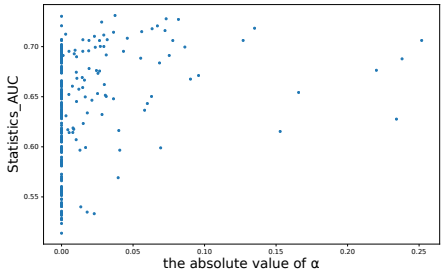

Figure 3: Relationship between statistics_AUC and α value for each second-order interaction

Table 5: Performance comparison between the model with interactions selected by our model and by statistics_AUC on Avazu Dataset

Model	AUC	log loss
Selected by statistics_AUC	0.7804	0.3794
Selected by AutoFM	0.7831	0.3778

To evaluate the effectiveness of the selected interactions by our model, we also select the top-N (N is the number of second-order feature interactions selected by our model) interactions based on statistics_AUC and re-train the model with these interactions. As shown in Table 5, the performance of our model is much better than the model with selected interactions by statistics_AUC with same computational cost.

4.5.2 The Effectiveness of selected feature interaction on Synthetic Data. In this section, we conduct a synthetic experiment to validate the effectiveness of selected feature interaction.

This synthetic dataset is generated from an incomplete poly-2 function, where the bi-linear terms are analogous to interactions between categories. Based on this dataset, we investigate (i) whether our model could find the important interactions (ii) the performance of our model compared with other factorization machine models.

The input x of this dataset is randomly sampled from N categories of m fields. The output y is binary labeled depending on the sum of linear terms and parts of bi-linear terms.

$$y = \delta(\sum_{i=1}^{m} w_i x_i + \sum_{i,j \in C} v_{i,j} x_i x_j + b + \epsilon) \tag{16}$$

$$\delta(z) = \begin{cases} 1, & if\ z \geq threshold \\ 0, & otherwise \end{cases} \tag{17}$$

The data distribution $p(x)$, selected bi-linear term sets C and $\boldsymbol{w}, \boldsymbol{v}, b$ are randomly sampled and fixed. The data pairs are $i.i.d.$ sampled to build the training and test datasets. We also add a small random noise ϵ to the sampled data. We use FM and our model to fit the synthetic data. We use AUC to evaluate these models on the test dataset.

We choose $m = 6, N = 60$ to test the effectiveness of our model. Selected bi-linear term sets C is randomly initialized as $C = \{(x_0, x_1), (x_2, x_5), (x_3, x_4)\}$. Figure 4 presents the performance comparison between our model and FM, which demonstrates the superiority of our model. As shown in Figure 5, our model could extract the important interactions precisely. The interactions in C have the highest α and some unimportant interactions (with α value 0) have been removed.

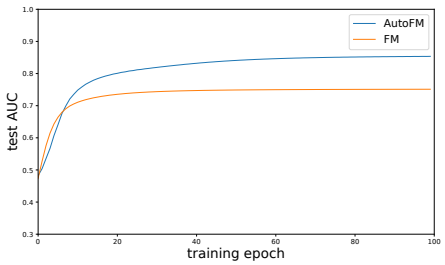

Figure 4: Training results of the synthetic experiments to verify AutoFM as a better model.

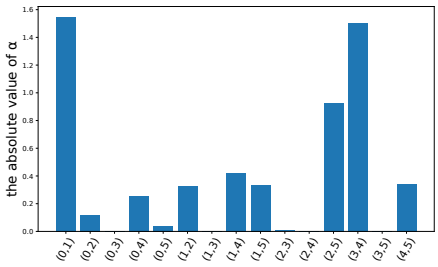

Figure 5: Training results of the synthetic experiments to verify AutoFM could find more important interactions.

4.6 Deployment & Online Experiments (RQ4)

Online experiments were conducted in the recommender system of Huawei App Store to verify the superior performance of AutoDeepFM. Huawei App Store has hundreds of millions of daily active users which generates hundreds of billions of user log events everyday in the form of implicit feedback such as browsing, clicking and downloading apps. In online serving system, hundreds of candidate apps that are most likely to be downloaded by the users are selected by a model from the universal app pool. These candidate apps are then ranked by a fine-tuned ranking model (such as DeepFM, AutoDeepFM) 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. To deploy AutoDeepFM, we utilize a three-node 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 ten-day AB test is conducted in a game recommendation scenario in the App Store. Our baseline in online experiments is DeepFM, which is a strong baseline due to its extraordinary accuracy and high efficiency which has been deployed in the commercial system for a long time.

For the control group, 5% of users are randomly selected and presented with recommendation generated by DeepFM. DeepFM is chosen as a strong baseline due to its extraordinary accuracy and high efficiency, which has been deployed in our commercial system for a long time. For the experimental group, 5% of users are presented with recommendation generated by AutoDeepFM.

Figure 6 and Figure 7 show the improvement of the experimental group over the control group with CTR (*#downloads/#impressions*) and CVR (*#downloads/#users*) respectively. We can see that the system is rather stable where both CTR and CVR fluctuated within 8% during the A/A testing. Our AutoDeepFM model is launched to the live system on Day 8. From Day 8, we observe a significant

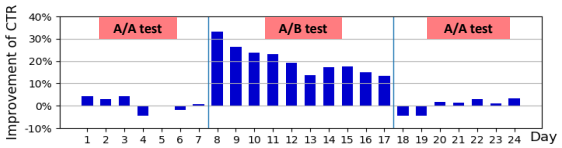

Figure 6: Online experimental results of CTR.

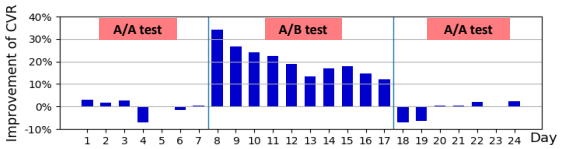

Figure 7: Online experimental results of CVR.

improvement over the baseline model with respect to both CTR and CVR. The average improvement of CTR is **20.3%** and the average improvement of CVR is **20.1%** over the ten days of A/B test. These results demonstrate the magnificent effectiveness of our proposed model. From Day 18, we conduct again A/A test to replace our AutoDeepFM model with the baseline model in the experimental group. We observe a sharp drop in the performance of the experimental group, which once more verifies that the improvement of online performance in the experimental group is indeed introduced by our proposed model.

4.7 Ablation Study (RQ5)

4.7.1 Stability of $\boldsymbol{\alpha}$ estimation across different seeds. In this part, we conduct experiments to check whether the trained value of $\boldsymbol{\alpha}$ is stable across different random initializations. A stable estimation of $\boldsymbol{\alpha}$ means that the model's decision on which interaction is important is not affected by the random seed. We run the *search stage* of AutoFM with different seeds on Avazu. The Pearson correlation of α estimated from different seeds is around 0.86, this validates that the estimation of $\boldsymbol{\alpha}$ is stable. Without the use of BN for the feature interaction (which is essentially FwFM model), this Pearson correlation drop to around 0.65.

Table 6: Different Variants.

Variants	*search stage*		*re-train stage*	
	AutoFIS	Random	BN	α
AutoFM	√	×	√	√
AutoFM-BN	√	×	×	√
AutoFM-BN-α	√	×	×	×
Random+FM	×	√	×	×

Table 7: Performance comparison of different feature interaction selection strategies. *: with fewer interactions, FM may have better performance.

Model	Avazu		Criteo	
	AUC	log loss	AUC	log loss
FM	0.7793	0.3805	0.7909	0.5500
AutoFM	**0.7831**	**0.3778**	**0.7974**	**0.5446**
AutoFM-BN	0.7824	0.3783	0.7971	0.5450
AutoFM-BN-α	0.7811	0.3793	0.7946	0.5481
Random+FM	0.7781	0.3809	0.7940*	0.5486

4.7.2 Effectiveness of components in AutoFIS. To validate the effectiveness of individual components in AutoFIS, we propose several variants, which are enumerated in Table 6. Recall that AutoFIS has two stages: *search stage* and *re-train stage*. To verify the effectiveness of the *search stage* of AutoFIS, we compare it with "Random"

strategy, which selects feature interactions randomly. Similarly, in the *re-train stage*, we validate the advantages of BN and α. The relationship between different components in the two stages is presented in Table 6. The performance of such variants presented in Table 7. Note that for "Random" strategy, we choose the same number of interactions with AutoFM, and we try ten different "Random" strategies and average the results. We can get several conclusions:

(1) Comparing AutoFM-BN-α with Random+FM, we can see that selection by AutoFIS can always achieve better performance than Random selection with same number of interactions. It demonstrates that important interactions are identified by AutoFIS in the *search stage*.
(2) The performance gap between Random+FM and FM in Criteo dataset indicates that random selection on feature interactions may outperform the model keeping all the feature interactions under some circumstances, which supports our statement: removing some useless feature interactions could improve the performance.
(3) The comparison between AutoFM and AutoFM-BN validates the effectiveness of BN in the *re-train stage*, where the reason is stated in "AutoFIS" section.
(4) The performance gap between AutoFM-BN and AutoFM-BN-α shows that $\boldsymbol{\alpha}$ improve the performance, as it differentiates the contribution of different feature interactions in the *re-train stage*.

Table 8: Comparison of one-level and bi-level optimization

Model	Avazu		Criteo	
	AUC	log loss	AUC	log loss
AutoFM	**0.7831**	**0.3778**	**0.7974**	**0.5446**
Bi-AutoFM	0.7816	0.3787	0.7957	0.5464
AutoDeepFM	**0.7852**	**0.3765**	**0.8009**	**0.5404**
Bi-AutoDeepFM	0.7843	0.3771	0.8002	0.5412

4.7.3 One-level V.S. bi-level optimization. In this section, we compare the one-level and bi-level optimization on AutoFM and the results are presented in Table 8. The performance gap between AutoFM and Bi-AutoFM (and between AutoDeepFM and Bi-AutoDeepFM) demonstrates the superiority of one-level optimization over bi-level, with the reason stated in "One-level Optimization" section.

5 CONCLUSION

In this work, we proposed AutoFIS to automatically select important 2^{nd}- and 3^{rd}-order feature interactions. The proposed methods are generally applicable to all the *factorization models* and the selected important interactions can be transferred to other deep learning models for CTR prediction. The proposed AutoFIS is easy to implement with marginal search costs, and the performance improvement is significant in two benchmark datasets and one private dataset. The proposed methods have been deployed onto the training platform of Huawei App Store recommendation service, with significant economic profit demonstrated.

REFERENCES

[1] Gabriel Bender. 2019. Understanding and simplifying one-shot architecture search. In *CVPR*.
[2] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H. Chi. 2018. Latent Cross: Making Use of Context in Recurrent Recommender Systems. In *WSDM*. 46–54.
[3] Shih-Kang Chao and Guang Cheng. 2019. A generalization of regularized dual averaging and its dynamics. In *CoRR*. abs/1909.10072 (2019).
[4] Shih-Kang Chao Chao, Zhanyu Wang, Yue Xing, and Guang Cheng. 2020. Directional Pruning of Deep Neural Networks. *arXiv preprint arXiv:2006.09358* (2020).
[5] Tianqi Chen and Carlos Guestrin. 2016. XGBoost:A Scalable Tree Boosting System. In *SIGKDD*. 785–794.
[6] Hengtze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Deepak Chandra, Hrishi Aradhye, Glen Anderson, Greg S Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & Deep Learning for Recommender Systems. In *DLRS@RecSys*.
[7] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *RecSys*. 191–198.
[8] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. In *IJCAI*. 1725–1731.
[9] Chaoyang He, Haishan Ye, Li Shen, and Tong Zhang. 2020. MiLeNAS: Efficient Neural Architecture Search via Mixed-Level Reformulation. In *CVPR*.
[10] Xiangnan He and Tat-Seng Chua. 2017. Neural Factorization Machines for Sparse Predictive Analytics. In *SIGIR*. 355–364.
[11] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, and Stuart Bowers. 2014. Practical Lessons from Predicting Clicks on Ads at Facebook. In *ADKDD@KDD*. 5:1–5:9.
[12] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. In *Neural Networks*.
[13] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *ICML*. 448–456.
[14] Yu-Chin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. 2016. Field-aware Factorization Machines for CTR Prediction. In *RecSys*.
[15] YuChin Juan, Yong Zhuang, and Wei-Sheng Chin. 2014. 3 IdiotsâĂŹ Approach for Display Advertising Challenge. https://www.csie.ntu.edu.tw/ r01922136/kaggle-2014-criteo.pdf.
[16] Yehuda Koren, Robert M. Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *IEEE Computer* 42, 8 (2009), 30–37.
[17] Guilin Li, Xing Zhang, Zitong Wang, Zhenguo Li, and Tong Zhang. 2019. StacNAS: Towards stable and consistent optimization for differentiable Neural Architecture Search. *arXiv preprint arXiv:1909.11926* (2019).
[18] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. xDeepFM: Combining Explicit and Implicit Feature Interactions for Recommender Systems. In *KDD*.
[19] Bin Liu, Ruiming Tang, Yingzhi Chen, Jinkai Yu, Huifeng Guo, and Yuzhou Zhang. 2019. Feature Generation by Convolutional Neural Network for Click-Through Rate Prediction. In *WWW*. ACM, 1119–1129.
[20] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019. DARTS: Differentiable Architecture Search. In *ICLR*.
[21] 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 *KDD*. 1936–1945.
[22] H. Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg, Arnar Mar Hrafnkelsson, Tom Boulos, and Jeremy Kubica. 2013. Ad click prediction: a view from the trenches. In *KDD*.
[23] Junwei Pan, Jian Xu, Alfonso Lobos Ruiz, Wenliang Zhao, Shengjun Pan, Yu Sun, and Quan Lu. 2018. Field-weighted factorization machines for click-through rate prediction in display advertising. In *WWW*. 1349–1357.
[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. (2016), 1149–1154.
[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] James Kwok Yong Li Cho-Jui Hsieh Quanming Yao, Xiangning Chen. 2020. Efficient Neural Interaction Function Search for Collaborative Filtering. In *WWW*.
[27] Steffen Rendle. 2010. Factorization Machines. In *ICDM*. 995–1000.
[28] Shai Shalev-Shwartz, Ohad Shamir, and Shaked Shammah. 2017. Failures of Gradient-Based Deep Learning. In *ICML*. 3067–3075.
[29] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & cross network for ad click predictions. In *ADKDD@KDD*. 12.
[30] Jun Xiao, Hao Ye, Xiangnan He, Hanwang Zhang, Fei Wu, and Tat-Seng Chua. 2017. Attentional Factorization Machines: Learning the Weight of Feature Interactions via Attention Networks. In *IJCAI*. 3119–3125.
[31] Weinan Zhang, Tianming Du, and Jun Wang. 2016. Deep learning over multi-field categorical data. In *European conference on information retrieval*. Springer, 45–57.
[32] Guorui Zhou, Xiaoqiang Zhu, Chengru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep Interest Network for Click-Through Rate Prediction. In *KDD*. 1059–1068.

A PARAMETER SETTINGS

For Avazu and Criteo datasets, the parameters of baseline models are set following [25]. For AutoFM and AutoDeepFM we use the same hyper-parameters as the base models (i.e., FM and DeepFM accordingly) except for extra ones in AutoFM and AutoDeepFM.

Table 9: Parameter Settings

Model	Avazu	Criteo
General	bs=2000 opt=Adam lr=1e-3	bs=2000 opt=Adam lr=1e-3
GBDT+LR	#tree=50 #child=2048	#tree=80 #child=1024
GBDT+FFM	#tree=50 #child=1024	#tree=20 #child=512
FM	k=40	k=20
FwFM	k=40 wt_init =0.7 wt_l1 = 1e-8 wt_l2=1e-7	k=20 wt_init =0.7 wt_l1 = 0 wt_l2=1e-7
FFM	k=4	k=4
AFM	k=40 t=1 h=256 l2_a =0	k=20 t=0.01 h=32 l2_a=0.1
DeepFM	k=40 net=[700× 5, 1] l2=0 drop=1 BN=True	k=20 net=[700× 5, 1] l2=0 drop=1 BN=True
AutoDeepFM	c=0.0005 mu=0.8	c=0.0005 mu=0.8
AutoFM	c=0.005 mu=0.6	c=0.0005 mu=0.8

* Note: bs=batch size, opt=optimizer, lr=learning rate, k=embedding size, wt_init = initial value for α, wt_l1 = l_1 regularization on α, wt_l2 = l_2 regularization on α, t=Softmax Temperature, l2_a= L2 Regularization on Attention Network, net=MLP structure, LN=layer normalisation, BN=batch normaliation, c and mu are parameters in GRDA Optimizer.