AutoIAS: Automatic Integrated Architecture Searcher for Click-Trough Rate Prediction

Zhikun Wei[1], Xin Wang[1,2,*], Wenwu Zhu[1,2,*]
weizk19@mails.tsinghua.edu.cn,{xin_wang,wwzhu}@tsinghua.edu.cn
[1]Tsinghua University [2]Pengcheng Laboratory

ABSTRACT

Automating architecture design for recommendation tasks becomes a trending topic because expert efforts are saved, and better performance is expected. Neural Architecture Search (NAS) is introduced to discover powerful CTR prediction model architectures in recent works. CTR prediction model usually consists of three components: embedding layer, interaction layer, and deep neural network. However, existing automation works focus on searching single component and leaving other components hand-crafted. The isolated searching will cause incompatibility among components and lead to weak generalization ability. Moreover, there is not a unified framework for integrated CTR prediction model architecture searching. This paper presents Automatic Integrated Architecture Searcher (AutoIAS), a framework that provides a practical and general method to find optimal CTR prediction model architecture in an automatic manner. In AutoIAS, we unify existing interaction-based CTR prediction model architectures and propose an integrated search space for a complete CTR prediction model. We utilize a supernet to predict the performance of sub-architectures, and the supernet is trained with Knowledge Distillation(KD) to enhance consistency among sub-architectures. To efficiently explore the search space, we design an architecture generator network that explicitly models the architecture dependencies among components and generates conditioned architectures distribution for each component. Experiments on public datasets show the outstanding performance and generalization ability of AutoIAS. Ablation study shows the effectiveness of the KD-based supernet training method and the Architecture Generator Network.

CCS CONCEPTS

• **Information systems → Recommender systems**.

KEYWORDS

Click-Trough Rate Prediction; Neural Architecture Search; Unified Search Space; Architecture Generator Network

ACM Reference Format:
Zhikun Wei[1], Xin Wang[1,2,*], Wenwu Zhu[1,2,*]. 2021. AutoIAS: Automatic Integrated Architecture Searcher for Click-Trough Rate Prediction. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3459637.3482234

CIKM '21, November 1–5, 2021, Virtual Event, QLD, Australia

ACM ISBN 978-1-4503-8446-9/21/11…$15.00
https://doi.org/10.1145/3459637.3482234

1 INTRODUCTION

Recommender system plays an important role in many real-world scenarios[2, 22, 31]. These applications can be modeled as Click-Through Rate (CTR) prediction tasks. The objective is to estimate the probability of a user clicking a recommended item. The more accurate the CTR prediction is, the more profit can be made. It is in great need for the commodity or ads provider to design a prediction model that can maximize CTR.

Recent works show architecture design is crucial to the performance of CTR models [3, 10, 18]. Automated Machine Learning (AutoML) techniques are introduced to help design CTR prediction model architectures because hand-crafted architectures are heavily dependent on expert knowledge and it can not guarantee optimality [15, 19, 21, 32]. Neural Architecture Search (NAS) is the most used method, which aims at finding an optimal architecture from given search space automatically. Existing works apply NAS to optimize the components of CTR prediction model. For the embedding component, some works [7, 21, 41, 42] search embedding size. For the interaction component, previous works select important feature combination [19], model user behavior [40] or design interaction function [15] automatically. However, these works focus on searching only one isolated component and other components remain hand-crafted. This kind of isolated searching causes three problems: 1) The other hand-crafted components are less likely to construct an optimal framework, making the whole model defective even if the searched component perfectly fits in the given context. 2) The generalization ability is weak because they do not provide enough space for the algorithm to explore. 3) In practice, existing search space and search algorithms are designed for specific components, and they can not trivially extend to an integrated CTR prediction model search framework.

To address the above problems, we consider how to apply the NAS algorithm to an integrated CTR prediction model design effectively and efficiently. There are four critical challenges:
(1) **How to design an integrated search space for multiple components?** The unified model consists of different types of architectures, and each component has a different search space. Extra interfaces space should be designed to connect these components.
(2) **How to efficiently explore and optimize models in search space?** It is difficult to enumerate all the architectures and train them separately because of the large search space. We tackle this problem by adopting a parameter sharing supernet. Considering that the search spaces of most components are dimensions, we let the architectures with high dimensions share their parameters with

*Corresponding authors

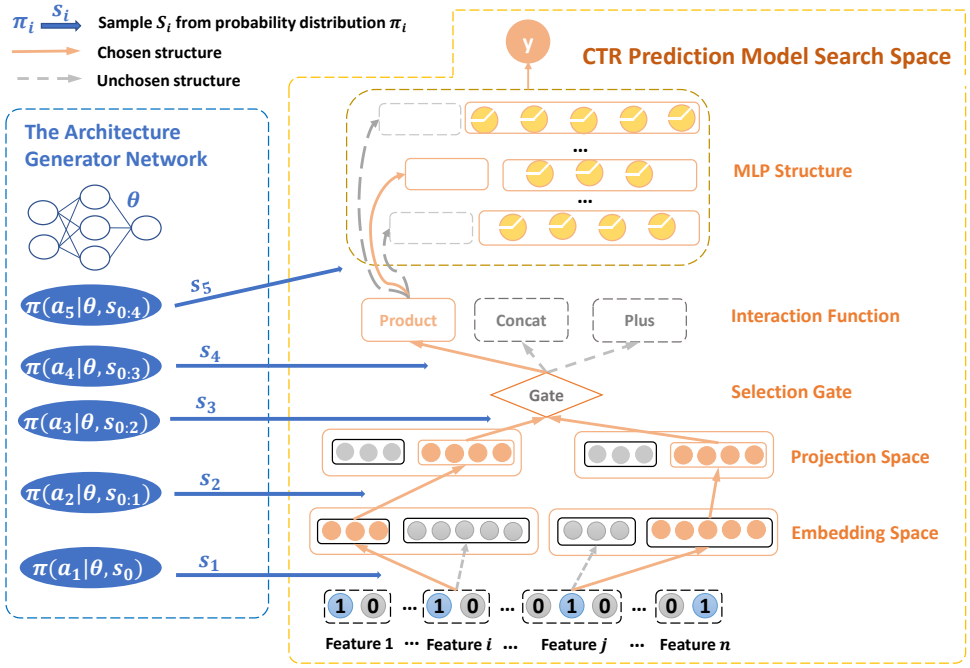

Figure 1: The framework of Automatic Integrated Architecture Searcher (AutoIAS). The left part is the architecture generator network. $\pi_i(a_i|\theta, s_{0:i-1})$ is the probability distribution over component architecture space a_i given the network parameter θ and all previous components architecture choice state $s_{0:i-1}$. The right part is the integrated search space where all components have their own architecture space, and the architecture choice is sampled from the probability distribution generated by the Architecture Generator Network.

architectures with low dimension. In this way, the parameters can be reused for architecture searching.

(3) **How to get stable and consistent performance prediction?** Stable and consistent performance predictions can guide the searching process and help to find better architectures. However, it is difficult to get a stable and consistent performance of sub-architectures from a supernet because updates of sub-architectures will influence each other. Previous works tried to solve it by fairly sampling[5], but the separated updating still may cause inconsistency between sub-architectures. To handle this problem, we utilize Knowledge Distillation(KD)[12] in the training process. We let the supernet digest incoming data to make a global parameter update and then distillate knowledge to its sub-architectures. In this way, all the sub-architectures will be updated in the same direction as the supernet, and the parameters of different sub-architectures will stay consistent.

(4) **How to model the dependency among different components?** CTR prediction model consists of multiple types of components which are correlated closely. The architecture choices of previous components can have a significant influence on later components. We design a generator network to explicitly model the relations. The generator network will generate an architecture probability distribution for a component based on the architectures of previous components.

Based on the above consideration, we integrate our solutions as a framework, Automatic Integrated Architecture Searcher (AutoIAS), to automatically search for an integrated CTR model architecture whose components are compatible with each other. An overview of AutoIAS is shown in Figure 1. The framework has two main parts. One is the architecture generator network. Given the previous components' architecture choice, it will generate probability distributions over the architecture choices for each component. The other is the CTR prediction model search space. The CTR prediction model architecture is divided into five components: the embedding component, the projection component, the interaction component, the selection component, and the MLP structure component. Each component contains an operation set according to its functionality. All the components compose a path for feature interaction. The search space includes not only traditional hand-crafted architectures such as DeepFM[10], Wide&Deep[3], but also the search space proposed by other NAS-based models such as AutoEmb[42], AutoFIS[19].

We conduct extensive experiments on public CTR prediction datasets which are used as benchmark for hand-crafted CTR prediction models[3, 10] and NAS-based methods[19, 42]. Experiment results show our framework can generate a CTR prediction architecture with better generalization ability than other baselines and is on a par with baselines in terms of performance. Ablation study shows the efficiency and effectiveness of the KD-based supernet training method and Architecture Generator Network.

In general, the main contributions of this paper are:

(1) We propose an Automatic architecture searching framework, **AutoIAS**, to search for the best integrated CTR prediction model architecture. To the best of our knowledge, it is the first searching framework that unifies all the components.

(2) We utilize Knowledge Distillation-based supernet training method to obtain stable and consistent architecture performance predictions for sub-architectures.
(3) We propose an effective searching strategy that utilizes an architecture generator network to explicitly model the architecture dependency between components.
(4) Substantial experiments conducted on public CTR datasets show that AutoIAS has better generalization ability than other CTR prediction model searchers. Ablation study shows that our proposed optimization methods are effective for the training.

The remaining parts of this paper are organized as follows. In Section 2, we review previous work about CTR prediction and NAS; In Section 3, we introduce the proposed framework AutoIAS in detail; In Section 4, we present the experiment settings and results; In Section 5, we concisely conclude this paper.

2 RELATED WORK

Our work involves two research topics: CTR prediction and NAS. Related work will be introduced briefly in this section.

2.1 CTR Prediction

Recent research shows that architecture design is crucial to the performance of CTR models[10, 26]. Due to the special data format (mostly sparse category data) in the recommendation tasks, CTR models usually consist of three components[15]: embedding layer, interaction layer, and deep neural networks. On the one hand, these components have unique architecture space, and they must be designed respectively. On the other hand, all the components must be considered simultaneously because they form an end-to-end pipeline to process data. There are plenty of works that design architecture in a hand-crafted manner. Linear models[22] shows decent performance. Factorization Machines (FM)[29] and its derivatives[11, 14, 30] achieve promising results by encoding sparse data into embedding vectors and designing explicit interaction functions to capture low-order feature interaction; Multi-Layer Perceptrons (MLP) or deep neural networks are used to implicitly model high-order feature interaction such as Wide&Deep[3], DLRM[23]; Both low- and high-order feature interactions are learned simultaneously by combining FM and deep neural networks in parallel (DeepFM[10]) or in a row (PNN[25], PIN[26]); DCN[34] and xDeepFM[18] further design architecture to handle explicit high-order feature interaction. However, the motivations of these architectures are heavily based on expert knowledge and experience. Besides, these architectures are usually hand-crafted, which requires high labor costs and causes uncertainty about the optimality.

2.2 NAS And Its Application in CTR Prediction Model.

AutoML techniques are introduced to help design CTR model architectures in recent works[15, 32]. The most used method is Neural Architecture Search (NAS), which automatically finds an optimal architecture from a given search space. There are usually three components in NAS: search space, search algorithm, and performance evaluation method. Search space defines what architectures can be searched and how the architectures are represented. Search algorithm defines how the search space is explored. The performance evaluation method defines how to evaluate the architectures and returns the optimality of the searched architectures. Since the proposal of NAS[43], extensive improvements are proposed to enhance utility and generalization[20, 24]. As a general solution to automatic model design, NAS has been widely used[35, 36]. Following the great success of NAS in Computer Vision, Natural Language Processing and Graph tasks[6, 9, 28, 37, 39], NAS is introduced to recommender system, especially CTR prediction task[13, 32]. When NAS automates the model's design process, the high labor costs are saved, and an optimal architecture is expected.

As aforementioned, there are three components in CTR prediction models, and they have different architecture spaces.

The embedding layer, also called the representation layer, learns to project the sparse category data into a continuous embedding space. One key hyper-parameter in the embedding layer is the embedding vector size for features (or users/items). Embedding vectors with proper size can represent the data correctly and boost the performance of the whole model[13], while improper embedding sizes may introduce noise or overfitting problems. NIS[13] first pays attention to the design of the input component of the recommender system and tries to find the best embedding and vocabulary sizes. DNIS[4] improves NIS using a differentiable method. AutoEmb[42] and [7] enable various embedding dimensions to represent users/items with different popularity. AutoDim[41] uses weight sharing[24] techniques to search embedding dimension. Embedding Size Adjustment Policy Network[21] is proposed to dynamically adjust embedding size in a streaming setting. However, the above methods have to project different dimension embeddings into a uniform dimension because subsequent components only accept uniform dimension vectors as inputs. No work notices that the interface between components also forms a search space. The projection dimension should be considered an important architecture parameter because it decides the representation ability for the original interacted feature information.

The interaction layer is regarded as the most important component in CTR models[38]. Explicit feature interaction methods can significantly improve the performance of CTR models[10]. Factorization machine (FM) based works[10, 14, 29] define the interaction function in a hand-crafted manner and select feature combinations by guessing or enumerating all possible situations. NAS is introduced to optimize the interaction layer. Yao et al.[38] finds expressive simple neural interaction functions. AutoFeature[15] searches proper sub-network structures to model essential feature interactions. AutoFIS[19] makes feature interaction selection differentiable. AMER[40] designs three stage search space to model user behavior automatically. However, aforementioned works only focus on interaction layer and leave other components hand-crafted.

MLP approximates the high-order feature interactions and serves as a subsidiary component in CTR models. The necessity of MLP lies in the fact that those easily understood feature interactions can be designed by experts, but there are many complex or irregular interactions hidden in data[10]. Although researchers attend to combine low- and high-order interaction in both explicit and implicit way[18, 34], little attention is paid to the unified search of explicit and implicit feature interaction.

3 AUTOIAS

This section will introduce the proposed framework AutoIAS in detail. In the following, we will show the problem formulation in Section 3.1, introduce how the integrated search space of each component is designed in Section 3.2, explain how to get performance prediction by supernet and its optimization method in Section 3.3, describe the search strategy which utilizes an architecture generator network and its optimization method in Section 3.4, and the overall optimization algorithm is presented in Algorithm 1.

3.1 Problem Definition

3.1.1 Native NAS Formulation. The goal of NAS is to find the architecture with the best performance in a given search space. It is a two-level objective: 1) find the best architecture; 2) find the best parameters for each architecture. Compared with previous work, the search space consists of multiple components in this paper. It can be formulated as a bi-level optimization problem[20]:

$$\begin{aligned} a^*_{1:T} &= \underset{a_i \in A_i, i=1,.,T}{\arg\min} \mathcal{L}_{val}(a_{1:T}, \omega^*(a_{1:T})), \\ s.t. \quad \omega^*(a_{1:T}) &= \underset{\omega \in \Omega(a_{1:T})}{\arg\min} \mathcal{L}_{train}(a_{1:T}, \omega), \end{aligned} \tag{1}$$

where T is the number of components, $a^*_{1:T}$ is the best architecture set for all components, $A_{1:T}$ is the search space for all the components, $\omega^*(a_{1:T})$ is the best model parameter for architecture $a_{1:T}$, and $\Omega(a_{1:T})$ is the parameter space given architecture $a_{1:T}$.

3.1.2 One-shot Formulation. It is almost impossible to solve Equation (1) directly. There are two critical problems: 1) we can not exhaustively search all the possible architectures in the search space; 2) To evaluate an architecture, we need to train it from scratch, which will cost an unaffordable amount of resources. To handle the above problems, we design an efficient search method inspired by previous work [1, 8]. Equation (1) is divided into two independent optimization problems by decoupling the dependency between architecture and parameters. The proposed algorithm is formulated in Equation (2) and (3). We omit subscribe of $a_{1:T}$ for convenience. The one-shot formulation is:

$$a^* = \underset{a \in A}{\arg\min} \mathcal{L}_{val}(a, \omega^*), \tag{2}$$

$$s.t. \qquad \omega^* = \underset{\omega \in \Omega}{\arg\min} \mathbb{E}_{a \sim \Gamma(A)} \mathcal{L}_{train}(a, \omega), \tag{3}$$

where $\Gamma(A)$ is a prior architecture distribution of A, Ω is the parameter space of supernet that contains all architectures in the search space, and ω^* is the best supernet parameter, a is the sub-architecture of the supernet, and all the sub-architectures share the same supernet parameters. In this way, to obtain the result in Equation (3), we no longer need to train each architecture from scratch, which significantly reduces computation costs.

3.2 Integrated Search Space

Our search space includes all components for a CTR prediction model. It is worth noting that these search spaces consist of different types and amounts of operations, and they must be represented separately. In addition, interfaces between existing components also form new search spaces. We conclude all the components as embedding layer, feature projection layer, interaction layer, feature interaction selection layer, MLP structure layer and MLP layer.

Embedding Layer. The main idea is to represent different features with mixed dimension embeddings. The candidate embedding dimensions form a dimensionality set $\mathcal{S}_1 = \{d_1, d_2, ..., d_n\}$. An input instance that has M features, i.e. $x = [x^1, x^2, ..., x^M] \in \mathbf{R}^M$. For each feature x^m, a dimension $d^m \in \mathcal{S}_1$ will be chosen. Thus for an input instance we have:

$$e = [e^1, e^2, ..., e^M], \tag{4}$$

where for $m \in \{1, 2, ..., M\}$, e^m is an embedding with size d^m.

Feature Projection Layer. After the embedding dimension is chosen, to let features be able to interact, we must unify the dimensions. Most previous works[21, 41, 42] project embeddings to a large and fixed dimension, but in this way the representation ability of new dimension may not suitable for the interaction. We introduce a new search space for the projection dimensions in the interface between embedding and interaction layer. The candidate projection dimension set is denoted as $\mathcal{S}_2 = \{d_1, d_2, ..., d_n\}$. For each feature interaction pair (e_i, e_j) with dimension d_i and d_j, if a projection dimension d_k is chosen, the embeddings of this pair will be both transformed to R^{d_k} space:

$$\begin{aligned} \hat{e}_i &= P_{i \to k}(e_i), \\ \hat{e}_j &= P_{j \to k}(e_j), \end{aligned} \tag{5}$$

where $P_{i \to k}(\cdot)$ and $P_{j \to k}(\cdot)$ are the linear transform functions from dimension d_i to d_k and d_j to d_k respectively.

Interaction Layer. Previous work [38] explores lots of interaction functions for feature interaction and indicates that different dimension interaction needs different function. In our method, the dimension of each interaction is different, which requires different interaction function for each interaction. The candidate interaction function set is $\mathcal{S}_3 = \{f_{product}, f_{concat}, f_{plus}, f_{max}\}$. For a pair of projected feature embeddings $(\hat{e}_i, \hat{e}_j)$:

$$\begin{aligned} f_{product} &= \hat{e}_i \circ \hat{e}_j, \\ f_{concat} &= Linear(concat[\hat{e}_i, \hat{e}_j]), \\ f_{plus} &= \hat{e}_i + \hat{e}_j, \\ f_{max} &= max(\hat{e}_i, \hat{e}_j), \end{aligned} \tag{6}$$

where $f_{product}$ is Hadamard product, f_{concat} concatenates two vectors as a new vector and transforms it to the dimension of original vectors. f_{plus} lets two vectors do bitwise addition. f_{max} takes bitwise maximum value from two vectors. All the interaction functions take two vectors as inputs and return a vector with the exact dimension of the input vectors, as shown in Equation (7).

$$v_{i,j} = f_*(\hat{e}_i, \hat{e}_j), \tag{7}$$

where $f_* \in \mathcal{S}_3$, $\hat{e}_i$ and $\hat{e}_j$ are projected feature of feature i and j respectively.

Selection of Feature Interactions. Suppose there are m features, then there will be m first-order features and C^2_m second-order feature interaction pairs. Putting all of them into a model is not an optimal solution because most of them are useless or even noisy. To select the optimal combination, we will face 2^n choices for the first-order and $2^{C^2_n}$ choices for second-order feature interaction. There will be $2^{n+C^2_n}$ possible combination for the interaction selection, which is difficult to handle. The search space is denoted as

$\mathcal{S}_4 = \{0,1\}^{(n+C_n^2)}$, where 1 stands for the interaction is selected and 0 otherwise. We will solve it by generating probability over each choice, and the method will be introduced later.

MLP Structure Layer. Existing works usually concatenate all vectors together and feed them to the MLP. However, low- and high-order implicit feature interactions are not considered simultaneously. In our search space, we design a novel space for mixed low- and high-order implicit feature interactions, as shown in Figure 2. All the layers in MLP are candidates for a feature interaction vector.

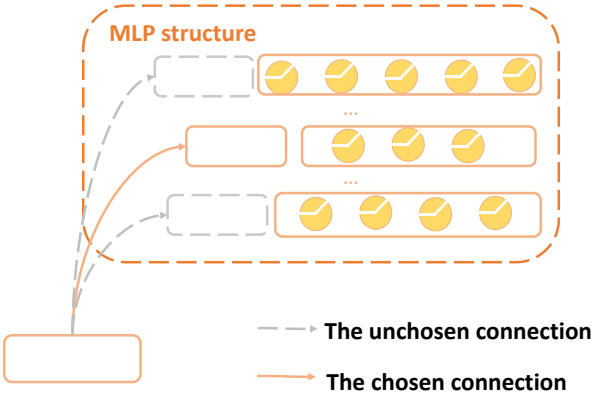

Figure 2: To make implicit interaction in the MLP component more effective, each explicit interaction results will be inserted into one of the layer.

The candidate set can be denoted as $\mathcal{S}_5 = \{1, 2, ..., L\}$, where L is the layer number of MLP.

When the interaction result $v_{i,j}$ for feature pair (i, j) is inserted into l-th layer, v_{ij} is concatenated with the output of $(l-1)$-th layer o_{l-1} to be the input of l-th layer, as shown in Equation (8).

$$o_l = f_l(concat[v_{ij}, o_{l-1}]), \tag{8}$$

MLP Layer. The search space for MLP is the basic unit number in each layer. Due to the insertion of interaction vectors, the structure space of MLP is dynamically changed. We define the search space as the basic dimension of MLP in each layer, i.e. $\mathcal{S}_6 = \{h_1, h_2, ..., h_n\}$. The final architecture of MLP will be decided by both the basic dimension and the insertion of interaction vectors.

3.3 Performance Prediction

We construct a supernet and predict the performance of the sub-architectures by inheriting parameters from the supernet.

3.3.1 Supernet with Ordinal Parameter sharing. We construct a supernet with ordinal parameter sharing. In the search space, the difference among sub-architectures is the dimension of embeddings or feature maps. It will be efficient for the supernet to use ordinal parameter sharing because it can reuse most of the parameters, and all the sub-architectures can conveniently share the supernet parameters.

The ordinal parameter sharing can be formulated as follow. Given a dimension search space $S = \{d_1, d_2, ..., d_n\}$ where $d_i < d_j$ for $i < j$, the supernet is constructed with the maximum dimension d_n. In the supernet, the feature map is kept as $v \in R^{d_n}$. To obtain a sub-architecture with dimension d^*, use

$$v^* = v \cdot \begin{bmatrix} \mathbf{I}^{d^*} \\ 0 \end{bmatrix}^{[d_n \times d^*]}, \tag{9}$$

where $\mathbf{I}^{d^*}$ is a square identity matrix with size d^*. The transform matrix is with size $d_n \times d^*$.

We define the operation with the most parameters or calculation as the supernet architecture for those layers whose choice set is not dimension. In detail, in the Interaction layer, the concatenation operation has the most parameters. In the Selection of Feature Interactions layer, we let all vectors be concatenated to the bottom layer in the supernet.

3.3.2 Knowledge Distillation Based Supernet Training. As described in Section 3.3.1, the supernet is the architecture that has the most parameters. To solve Equation (3), i.e., to find the best supernet parameters that maximize the reward expectation of all sub-architectures, we need a proper training method for the supernet that can utilize the shared parameters and get the accurate performance of sub-architecture.

Knowledge Distillation(KD) [12] was proposed to transfer knowledge from teacher network to student network. KD can be applied to help NAS training[17]. Inspired by previous works, we design a KD-based supernet training method.

We define the architecture with the most parameters as teacher network and any generated sub-architecture as student network.

Training Procedure. Firstly we declare some notations in one data batch: Denote the training data as $\{x, y\}$. Denote the supernet as $f(\cdot)$ and the supernet parameter as ω. Denote the architecture with most parameters as teacher network a_t. Denote any other sub-architecture as student network a_i $(i = 1, 2, ..., N)$ where N is the student network number. $f(x; \omega, a)$ is the performance prediction for x given supernet parameter ω and architecture a.

There are two steps in one data batch for supernet updating.

Step One: calculate the predicted value for teacher network $y_t = f(x; \omega, a_t)$. The objective is to minimize the cross-entropy of predicted values and the labels:

$$\mathcal{L}(y, y_t) = -y \log y_t - (1-y) \log(1-y_t), \tag{10}$$

where $y \in \{0, 1\}$ is the label and $y_t \in (0, 1)$ is the predicted probability of $y = 1$.

And supernet parameter ω will be updated by

$$\omega_t = \omega - \alpha \nabla_\omega \mathcal{L}(y, \hat{y}), \tag{11}$$

where α is the learning rate.

Step Two: Compute the prediction of teacher network

$$\hat{y_t} = f(x; \omega_t, a_t), \tag{12}$$

Generating N sub-architectures as student networks. For each student network a_i, compute prediction of student network:

$$y_i = f(x; \omega_t, a_i), \tag{13}$$

Use the prediction of teacher network as soft label for the sub-architectures to compute soft loss:

$$\mathcal{L}_{soft} = \mathcal{L}(\hat{y_t}, y_i), \tag{14}$$

Use the real label to compute hard loss:

$$\mathcal{L}_{hard} = \mathcal{L}(y, y_i), \tag{15}$$

After computing all losses of the student networks, the supernet parameters will be updated by

$$\omega_s = \omega_t - \alpha \frac{1}{N} \sum_{i=1}^{N} \nabla_{\omega i} [\beta \mathcal{L}_{soft} + (1-\beta)\mathcal{L}_{hard}], \tag{16}$$

where $\omega_i \subset \omega_t$ is the parameters of sub-architecture a_i. β is a hyper-parameter which determines the weight of soft and hard loss.

3.4 Search Strategy

We propose an architecture generator network to search through the search space. The architecture generator network will be trained by policy gradient.

3.4.1 Architecture Generator Network. To solve Equation (2), we need to find the best architecture from the search space. However, the search space introduced in Section 3.2 is large, and we need an efficient searching method. As introduced before, there are six components and six corresponding search spaces. Previous works usually utilize an RNN as the controller to generate architectures[43], and it models the relations between operations in an implicit manner. However, in our scenario, the components are not in the same structure space, and we must design a proper generator network to model the dependencies explicitly. We propose architecture generator network that can utilize the dependencies among components and output probability on each choice of the components. For each component, the input is the chosen state of previous components. The Architecture Generator Network is a neural network that takes in multiple states of previous components and returns the probability distribution P over the candidate set. The architecture of the corresponding component is sampled from P. The architecture generator networks are formulated as:

$$\begin{aligned} P_i &= \pi_i(\theta_i, a_{i-1}, ..., a_1), \\ a_i &\sim P_i, \end{aligned} \tag{17}$$

where $\pi_i(\cdot)$ represents the architecture generator network for component i, θ_i is the corresponding parameters, a_j $(j = 0, 1, .., i-1)$ is the choice of previous component. The architecture a_i is sampled from probability distribution P_i.

3.4.2 Optimization of the Architecture Generator Network. As described in Section 3.4.1, the generator is formulated as a policy network. To train the generator network, we adopt the policy gradient method. The parameters are denoted as θ. The expected reward of the policy network will be maximized. The expected reward is represented by $J(\theta)$:

$$J(\theta) = \mathbb{E}_{P(a;\theta)} R, \tag{18}$$

Considering that the reward R is non-differential, the policy gradient method is used to iteratively update θ, which is a common practice in previous NAS works[24, 43]. Thus,

$$\nabla_\theta J(\theta) = \mathbb{E}_{P(a;\theta)} \nabla_\theta \log P(a;\theta) R, \tag{19}$$

Algorithm 1: Overall Optimization Algorithm

Input: Training dataset D_{train}, Validation dataset D_{val}
Output: An architecture a^* with best performance

```
Initialize Supernet parameter ω randomly;
Initialize Architecture Generator Network θ randomly;
for Epoch =1,2,3,.. do
    Sample training batch {x_tra, y_tra} from D_train.
    Update supernet parameters ω by Equation (10)(11).
    Get teacher network prediction y_t by Equation (12).
    Generate architecture distribution P(a; θ).
    for i =1,2,..,N do
        Sample student network architecture a_i from P(a; θ).
        Compute prediction y_i by Equation (13).
        Compute soft and hard loss by Equation (14)(15).
    Update all student networks by Equation (16).
    for architecture batch i =1,2,..,M do
        Generate architecture distribution P(a; θ).
        Sample sub-architecture a_i from P(a; θ).
        Make prediction ŷ_i = f(x_tra; ω, a_i).
        Get reward R_i by Equation (22).
    Compute policy gradient by Equation (20).
    Update Generator θ by Equation (21).
    Sample an architecture a* from updated distribution.
    Validate performance of a* on D_val.
    if Validation performance is not improving then
        Break;
Output best architecture a*.
```

In practice, Equation (19) can be approximated as

$$\nabla_\theta J(\theta) \approx \frac{1}{M} \sum_{i=1}^{M} \nabla_\theta \log P(a_i;\theta) R_i, \tag{20}$$

where M is the number of different architectures sampled by the generator in one batch. R_i is the reward of the i-th architecture, as defined in Equation (22).

With the gradient $\nabla_\theta J(\theta)$, the parameters of policy network is updated with:

$$\theta = \theta + \alpha_\theta \nabla_\theta J(\theta), \tag{21}$$

where α_θ is the learning rate for the policy network.

Reward. The reward should reflect the performance of the generated architecture, and the policy network should be optimized to maximize the reward. We use Area Under roc* Curve (AUC) to measure the performance of CTR prediction model following previous works[19, 21]. When an architecture is generated, a corresponding CTR prediction model is created to be trained. Afterward, the model makes a prediction on the validation dataset and returns an AUC value τ. In our work, the performance prediction is made by the supernet, which brings a challenge: the AUC value will increase through the supernet training process, and the absolute AUC value can not give a correct signal. To ensure an adaptive and correct reward signal, we keep track of the history AUC average value $\bar{\tau}$ and use the difference between current AUC value and history

*roc: receiver operating characteristic.

average AUC value as the reward value:

$$R = \tau - \bar{\tau}, \tag{22}$$

and the history average AUC value is updated by:

$$\bar{\tau} = \alpha_\tau \bar{\tau} + (1 - \alpha_\tau)\tau, \tag{23}$$

where α_τ is a hyper-parameter that controls the weight of history AUC values.

By designing the above reward, when the policy network generates an architecture that achieves better performance than the historical average, the policy network is optimized along the direction to produce better architectures. On the contrary, if the performance of generated architecture is worse than the historical average, the policy network will be optimized to be away from current position.

4 EXPERIMENT

This section will introduce extensive experiments conducted on two public datasets to demonstrate the effectiveness of AutoIAG. The datasets are introduced in Section 4.1. The implementation details of our framework are introduced in Section 4.2. We present the experiment results and analysis of the performance of AutoIAG in Section 4.3. We conduct additional experiments to verify the effectiveness of the training methods in Section 4.4.

4.1 Dataset

To make a fair comparison with previous works, we conduct experiments on two public datasets. **Avazu** [†] was released for a mobile ad click prediction contest on Kaggle. Following the setting of previous works [15, 19], 80% data is sampled randomly for training and validation, the other 20% is used for testing. Categories appearing less than 20 times are regarded as a same feature value. **Criteo**[‡] was a click log dataset for the Criteo display advertisement challenge.

The statistics of aforementioned dataset are listed in Table 2.

4.2 Implementation Detail

The implementation details of our framework are introduced in this section.

4.2.1 Search Space. The search space of all components in our experiments is shown in Table 3, where n is the feature number in the dataset. The implementation of our framework is limited to second-order interactions, but it is easy to extend to higher-order interactions. In each component of the searched CTR prediction model, the choice for each feature or interaction is independent. Thus we have n or C_n^2 decisions to make. There are six components in our search space, including the feature embedding dimension, projection dimension, interaction function, selection of interaction, insertion of interactions, and hidden units in MLP. Theoretically, the feature embedding dimension and projection dimension can be a continuous number, but it will increase the difficulty for the generator network to produce a probability distribution over these candidates. To simplify the framework without losing representation capacity, we choose exponents of two as candidate dimensions. The candidate set for embedding and projection dimensions keeps identical because the same level of information capacity is expected. For the interaction function, we set four often used interaction functions as the candidates. The selection of interaction is to activate an interaction or not. The insertion of interactions decides each interaction will be inserted to which layer. The hidden units in MLP construct the basic structure of MLP, and we set a search space for it as well.

4.2.2 Supernet. To improve the training efficiency of the supernet, we implement the supernet using ordinal parameter sharing. We use the max dimension choice to construct the supernet for the components whose search space is dimension. To obtain its sub-architectures, we activate the proper dimension of the supernet. For example, if the search space of the embedding dimension is $\{4, 8, 16\}$, the embedding dimension in supernet will be 16. To obtain a sub-architecture with eight dimension embedding, we will take the first half of the supernet embedding as the parameter of the sub-architecture. We enumerate each choice for the components whose search space is not dimension choice, i.e., the interaction layer and selection of feature interactions. Because these components have no parameters, enumeration will not hurt efficiency. For the insertion of interactions, the parameters of each interaction are shared between sub-architectures. The supernet constructs a super MLP that includes all interactions to all layers. When an insertion instance is sampled, the corresponding connection will be activated. Considering the interactions have dimension choice, the super MLP component will also adopt ordinal parameter sharing.

4.2.3 Architecture Generator Network. In practice, we represent the choice state of each component in an embedding space. The embeddings will be updated along with the network. For each component, to generate the architecture probability distribution $\pi_i(a_i|\theta, s_{0:i-1})$, we feed state s_j $(j < i)$ from the embedding space to a three-layer MLP θ. The output of the MLP goes through a softmax layer to produce a probability distribution π_i. The final architecture choice is sampled from π_i and acts as the state of this component. For example, suppose the search space for the feature projection layer is $\{4, 8, 16\}$ and the corresponding embedding space will be three embeddings. The corresponding embedding will serve as the state when one candidate is chosen and the embedding will be fed to the generator network to generate architecture probability distribution for other components.

4.2.4 Optimization. The supernet is trained by the Adam[16] optimizer with an initial learning rate 0.001. The student number in Equation (16) is set to 9. The weight β for soft and hard loss is set to 0.5. The Adam optimizer trains the generator network with an initial learning rate of 0.001. The architecture sampled number in Equation (20) is set to 4.

4.3 Experiment Result and Analysis

4.3.1 Results. We compare the result with classical CTR prediction models and automation works in Table 1. We collect performance of representative hand-crafted CTR prediction models including FM[29], DeepFM[10] and PIN[27]. For automated CTR prediction model, we collect AutoInt[33], AutoCTR[32], AutoFIS[19], and AutoFeature[15] for comparison. For fair comparison, in AutoFIS and AutoFeature, search space for interaction is limited to

[†] http://www.kaggle.com/c/avazu-ctr-prediction
[‡] https://labs.criteo.com/2013/12/download-terabyte-click-logs/

Table 1: Experiment results on public datasets.

	Avazu			Criteo			Overall Rank
Models	AUC	Log Loss	Rank	AUC	Log Loss	Rank	(The smaller, the better)
FM[29]	0.7793	0.3805	6	0.7909	0.5500	8	14
DeepFM[10]	0.7836	0.3776	5	0.7991	0.5423	7	12
PIN[27]	0.7872	0.3755	3	0.8021	0.5390	4	7
AutoInt+[33]	0.7774	0.3811	8	0.8083	0.4434	2	10
AutoCTR[32]	0.7791	0.3800	7	0.8104	0.4413	1	8
AutoFIS(2nd)[19]	0.7852	0.3765	4	0.8009	0.5404	6	10
AutoFeature(2)[15]	0.7874	0.3753	1	0.8020	0.5395	5	6
AutoAIS(ours)	0.7873	0.3756	2	0.8059	0.5356	3	**5**

Table 2: Statistics of datasets

Dataset	Instances	Features	Category
Avazu	$\sim 4 * 10^7$	23	$\sim 6 * 10^5$
Criteo	$\sim 1 * 10^8$	39	$\sim 1 * 10^6$

Table 3: Search space for all components.

Component	Candidate Set
Embedding dimension	$\mathcal{S}_1 = \{4, 8, 16, 32, 64, 128\}^n$
Projection dimension	$\mathcal{S}_2 = \{4, 8, 16, 32, 64, 128\}^{C_n^2}$
Interaction function	$\mathcal{S}_3 = \{product, concat, plus, max\}^{C_n^2}$
Selection of interaction	$\mathcal{S}_4 = \{0, 1\}^{n+C_n^2}$
Insertion of interaction	$\mathcal{S}_5 = \{0, 1, 2, 3, 4, \}^{n+C_n^2}$
Hidden units in MLP	$\mathcal{S}_6 = \{64, 128, 256\}^5$

second order. Following previous work[15, 19, 32], we use the offline evaluation metrics of AUC(Area Under ROC) and log loss(cross entropy) which are commonly used as the surrogates for the actual CTR. Generalization ability is also an important metric for CTR prediction, especially for AutoML-based methods. We rank the testing AUC values for these works on two datasets respectively and use sum of the ranks to show model's overall generalization ability.

4.3.2 Performance Analysis. We first note that most baselines have inconsistent performances on two datasets. PIN uses a fixed subnetwork as the feature interaction function, and AutoFeature searches the sub-network for interaction. They both achieve remarkable performance on Avazu, but they do not perform well on Criteo. AutoCTR and AutoInt+ achieve the best performance on Criteo but perform poorly on Avazu. It can be observed that AutoML-based models outperform most of the hand-crafted baseline models. The searched architectures can better model the feature interactions and extract information from data.

However, for an AutoML-based method, the more critical metric should be the automation and generalization ability across different data. AutoAIS(ours) does not achieve the best performance in a single dataset compared to all the baselines, but AutoAIS always ensures stable and near-the-best performance across different datasets. For the overall rank on datasets, AutoAIS has the highest rank, which implies its generalization ability. The reason is that AutoAIS searches all the components, and it gives a larger space for finding the proper architectures to handle various kinds of data, while other automation works only focus on a single component leading to weaker generalization ability.

In conclusion, as an AutoML-based method, AutoAIS outperforms other automation works in terms of generalization ability and keeps competitive for each dataset.

4.4 Ablation Study

In this section, we conduct experiments to verify the effectiveness of the training methods in our framework. In each group of experiments, we only change the target condition while remaining other hyper-parameters as the baseline setting for convenience. Some settings are shared between experiment groups.

4.4.1 Influence of Knowledge Distillation. We verify the effectiveness of knowledge distillation by comparing it with directly training sub-architecture with hard loss (i.e., beta=0). We also explore the influence of weight between soft loss and hard loss. The result is shown in Figure 3. We can observe that Knowledge distillation can fasten the training process and achieve lower loss. With proper weight for soft and hard loss, we can train the supernet efficiently.

4.4.2 Influence of Student Number in KD-based Training. Student number refers to the sub-architecture number in each batch. It determines how many sub-architectures will be updated according to the knowledge of supernet. We study the influence of student number. The result is shown in Figure 4. It can be observed that when the student number is small, the training process is unstable. However, when the student number exceeds nine, the improvement is marginal. Considering that a large student number will cost more resources, we set the student number as nine to ensure a proper trade-off between the marginal improvement and efficiency.

4.4.3 Effectiveness of the Architecture Generator Network. Our search strategy is to train a generator network to learn the architecture distribution, guiding the sub-architecture sampling. To verify the effectiveness of the generator network, we replace the generator network with a random search in our framework. The comparison of the training process is shown in Figure 5. In this experiment, we choose the simplest setting: knowledge distillation is disabled, and

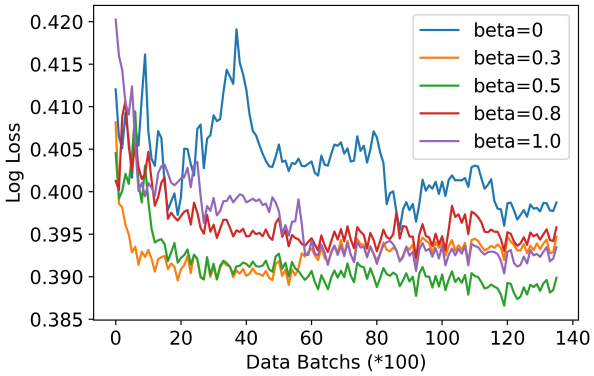

Figure 3: The influence of Knowledge Distillation. Beta is defined in Equation (16).

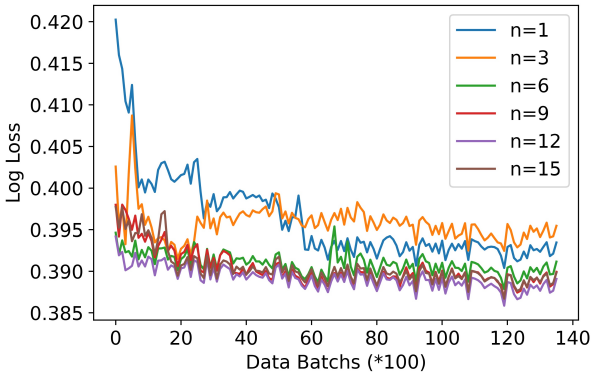

Figure 4: The influence of student sub-architecture number.

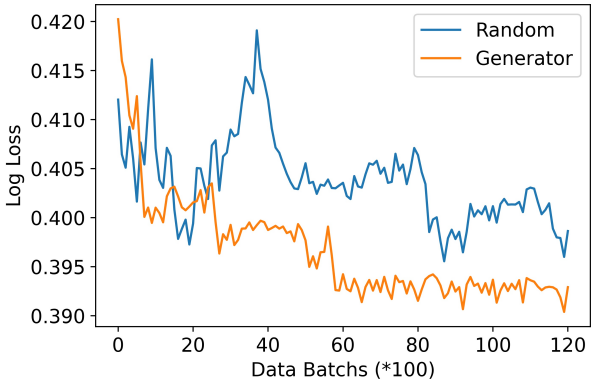

Figure 5: The influence of architecture generator network.

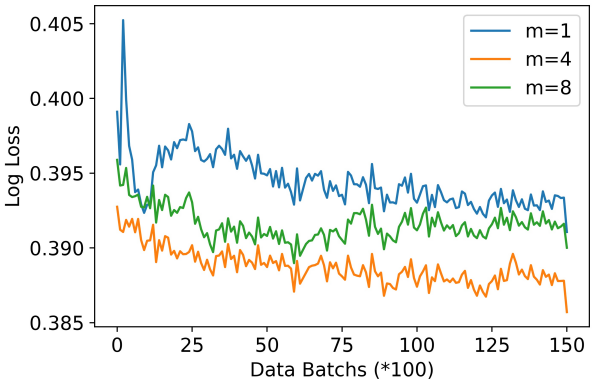

Figure 6: The influence of sub-architecture number to train the generator in one batch.

the sub-architecture number is one in each batch. The figure shows that the loss of generator-guided training is more stable and lower than that of random search.

4.4.4 Influence of Sub-architecture Number to Train the Generator. To optimize the generator network, we sample m architectures in one batch. The generator network will be updated according to the rewards of the sampled architectures. In practice, the number of sampled architecture will affect the training for the generator network, and experiments are conducted to study the influence. The result is shown in Figure 6. According to the experiment results, $m = 4$ is the best choice. If m=1, i.e., there is only one sub-architecture to generate a reward for the generator network, the generator network will not get enough guidance. If m is too large, all the sub-architectures may distract the generator and slow the improvement. A proper amount of sub-architectures will help training the generator to generate better architectures.

5 CONCLUSION

In this paper, we propose a NAS-based framework, Automatic Integrated Architecture Searcher (AutoIAS), to automatically search for an integrated interaction-based CTR prediction model architecture whose components are compatible with each other. In AutoIAS, the proposed search space covers all the existing interaction-based CTR prediction model components and introduces two novel interface search spaces: Projection Dimension and MLP Structure. Moreover, an architecture generator network explicitly models dependencies among components and generates conditioned architecture probability distribution for each component, and the architecture generator network is optimized by policy gradient. The performance of the generated architectures is obtained by a supernet. We utilize a knowledge-distillation-based supernet training method to train a supernet to ensure that the performance prediction is stable and consistent. Through substantial experiments, we show that AutoIAS outperforms previous automation works for the CTR prediction model in terms of stability and generalization ability. Ablation study shows the effectiveness of the proposed training methods in AutoIAS.

ACKNOWLEDGMENTS

This work was supported by the National Key Research and Development Program of China under Grant No.2020AAA0106301, National Natural Science Foundation of China No.62050110 and Tsinghua GuoQiang Research Center Grant 2020GQG1014.

REFERENCES

[1] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. 2018. Understanding and simplifying one-shot architecture search. In *Proceedings of the International Conference on Machine Learning*. 550–559.

[2] Olivier Chapelle, Eren Manavoglu, and Romer Rosales. 2014. Simple and scalable response prediction for display advertising. *ACM Transactions on Intelligent Systems and Technology (TIST)* 5, 4 (2014), 1–34.

[3] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*. 7–10.

[4] Weiyu Cheng, Yanyan Shen, and Linpeng Huang. 2020. Differentiable Neural Input Search for Recommender Systems. *CoRR* abs/2006.04466 (2020).

[5] Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Jixiang Li. 2019. FairNAS: Rethinking Evaluation Fairness of Weight Sharing Neural Architecture Search. *CoRR* abs/1907.01845 (2019).

[6] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural Architecture Search: A Survey. *J. Mach. Learn. Res.* 20 (2019), 55:1–55:21.

[7] Antonio Ginart, Maxim Naumov, Dheevatsa Mudigere, Jiyan Yang, and James Zou. 2019. Mixed Dimension Embeddings with Application to Memory-Efficient Recommendation Systems. *CoRR* abs/1909.11810 (2019).

[8] Chaoyu Guan, Yijian Qin, Zhikun Wei, Zeyang Zhang, Zizhao Zhang, Xin Wang, and Wenwu Zhu. [n.d.]. One-Shot Neural Channel Search: What Works and What's Next. ([n. d.]).

[9] Chaoyu Guan, Xin Wang, and Wenwu Zhu. 2021. Autoattend: Automated attention representation search. In *International Conference on Machine Learning*. PMLR, 3864–3874.

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

[11] Xiangnan He and Tat-Seng Chua. 2017. Neural factorization machines for sparse predictive analytics. In *Proceedings of the 40th ACM International Conference on Research and Development in Information Retrieval*. 355–364.

[12] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the Knowledge in a Neural Network. *CoRR* abs/1503.02531 (2015).

[13] Manas R. Joglekar, Cong Li, Mei Chen, Taibai Xu, Xiaoming Wang, Jay K. Adams, Pranav Khaitan, Jiahui Liu, and Quoc V. Le. 2020. Neural Input Search for Large Scale Recommendation Models. In *Proceedings of the 26th ACM Conference on Knowledge Discovery and Data Mining*. 2387–2397.

[14] Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. 2016. Field-aware factorization machines for CTR prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems*. 43–50.

[15] Farhan Khawar, Xu Hang, Ruiming Tang, Bin Liu, Zhenguo Li, and Xiuqiang He. 2020. AutoFeature: Searching for Feature Interactions and Their Architectures for Click-through Rate Prediction. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 625–634.

[16] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *Proceedings of the International Conference on Learning Representations*.

[17] Changlin Li, Jiefeng Peng, Liuchun Yuan, Guangrun Wang, Xiaodan Liang, Liang Lin, and Xiaojun Chang. 2020. Block-wisely supervised neural architecture search with knowledge distillation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1989–1998.

[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 *Proceedings of the 24th ACM International Conference on Knowledge Discovery and Data Mining*. 1754–1763.

[19] Bin Liu, Chenxu Zhu, Guilin Li, Weinan Zhang, Jincai Lai, Ruiming Tang, Xiuqiang He, Zhenguo Li, and Yong Yu. 2020. Autofis: Automatic feature interaction selection in factorization models for click-through rate prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2636–2645.

[20] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019. DARTS: Differentiable Architecture Search. In *Proceedings of the 7th International Conference on Learning Representations*.

[21] Haochen Liu, Xiangyu Zhao, Chong Wang, Xiaobing Liu, and Jiliang Tang. [n.d.]. Automated Embedding Size Search in Deep Recommender Systems. In *Proceedings of the 43rd ACM International conference on research and development in Information Retrieval*. 2307–2316.

[22] H Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, et al. 2013. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM International Conference on Knowledge Discovery and Data Mining*. 1222–1230.

[23] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. 2019. Deep learning recommendation model for personalization and recommendation systems. *CoRR* abs/1906.00091 (2019).

[24] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. 2018. Efficient Neural Architecture Search via Parameter Sharing. In *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80. 4092–4101.

[25] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. 2016. Product-based neural networks for user response prediction. In *Proceedings of the 16th IEEE International Conference on Data Mining*. 1149–1154.

[26] Yanru Qu, Bohui Fang, Weinan Zhang, Ruiming Tang, Minzhe Niu, Huifeng Guo, Yong Yu, and Xiuqiang He. 2018. Product-based neural networks for user response prediction over multi-field categorical data. *ACM Transactions on Information Systems (TOIS)* 37 (2018), 1–35.

[27] Yanru Qu, Bohui Fang, Weinan Zhang, Ruiming Tang, Minzhe Niu, Huifeng Guo, Yong Yu, and Xiuqiang He. 2018. Product-based neural networks for user response prediction over multi-field categorical data. *ACM Transactions on Information Systems (TOIS)* 37, 1 (2018), 1–35.

[28] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. 2020. A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions. *CoRR* abs/2006.02903 (2020).

[29] Steffen Rendle. 2010. Factorization machines. In *2010 IEEE International Conference on Data Mining*. 995–1000.

[30] Steffen Rendle. 2012. Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology (TIST)* 3, 3 (2012), 1–22.

[31] Matthew Richardson, Ewa Dominowska, and Robert Ragno. 2007. Predicting clicks: estimating the click-through rate for new ads. In *Proceedings of the 16th international conference on World Wide Web*. 521–530.

[32] Qingquan Song, Dehua Cheng, Hanning Zhou, Jiyan Yang, Yuandong Tian, and Xia Hu. 2020. Towards Automated Neural Interaction Discovery for Click-Through Rate Prediction. In *Proceedings of the 26th ACM International Conference on Knowledge Discovery & Data Mining*. 945–955.

[33] Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. 2019. Autoint: Automatic feature interaction learning via self-attentive neural networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 1161–1170.

[34] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & Cross Network for Ad Click Predictions. In *Proceedings of the ADKDD'17*. 12:1–12:7.

[35] Xin Wang, Shuyi Fan, Kun Kuang, and Wenwu Zhu. 2021. Explainable automated graph representation learning with hyperparameter importance. In *International Conference on Machine Learning*. PMLR, 10727–10737.

[36] Xin Wang and Wenwu Zhu. 2021. Automated Machine Learning on Graph. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 4082–4083.

[37] Lanning Wei, Huan Zhao, Quanming Yao, and Zhiqiang He. 2021. Pooling Architecture Search for Graph Classification. In *International Conference on Information and Knowledge Management*.

[38] Quanming Yao, Xiangning Chen, James T Kwok, Yong Li, and Cho-Jui Hsieh. 2020. Efficient neural interaction function search for collaborative filtering. In *Proceedings of The Web Conference 2020*. 1660–1670.

[39] Ziwei Zhang, Xin Wang, and Wenwu Zhu. 2021. Automated Machine Learning on Graphs: A Survey. In *International Joint Conference on Artificial Intelligence*. 4704–4712.

[40] Pengyu Zhao, Kecheng Xiao, Yuanxing Zhang, Kaigui Bian, and Wei Yan. 2020. AMER: Automatic Behavior Modeling and Interaction Exploration in Recommender System. *CoRR* abs/2006.05933 (2020).

[41] Xiangyu Zhao, Haochen Liu, Hui Liu, Jiliang Tang, Weiwei Guo, Jun Shi, Sida Wang, Huiji Gao, and Bo Long. 2020. Memory-efficient Embedding for Recommendations. *CoRR* abs/2006.14827 (2020).

[42] Xiangyu Zhao, Chong Wang, Ming Chen, Xudong Zheng, Xiaobing Liu, and Jiliang Tang. 2020. AutoEmb: Automated Embedding Dimensionality Search in Streaming Recommendations. *CoRR* abs/2002.11252 (2020).

[43] Barret Zoph and Quoc V. Le. 2017. Neural Architecture Search with Reinforcement Learning. In *Proceedings of the 5th International Conference on Learning Representations*.