AutoDim: Field-aware Embedding Dimension Search in Recommender Systems

Xiangyu Zhao[1,2], Haochen Liu[1], Hui Liu[1], Jiliang Tang[1]
Weiwei Guo[3], Jun Shi[3], Sida Wang[3], Huiji Gao[3], Bo Long[4]
[1]Michigan State University, [2]City University of Hong Kong, [3]Linkedin Corporation, [4]JD.com
{zhaoxi35,liuhaoc1,liuhui7,tangjili}@msu.edu
{wguo,jshi,sidwang,hgao}@linkedin.com,bo.long@gmail.com

ABSTRACT

Practical large-scale recommender systems usually contain thousands of feature fields from users, items, contextual information, and their interactions. Most of them empirically allocate a unified dimension to all feature fields, which is memory inefficient. Thus it is highly desired to assign various embedding dimensions to different feature fields according to their importance and predictability. Due to the large amounts of feature fields and the nuanced relationship between embedding dimensions with feature distributions and neural network architectures, manually allocating embedding dimensions in practical recommender systems can be challenging. To this end, we propose an AutoML-based framework (AutoDim) in this paper, which can automatically select dimensions for different feature fields in a data-driven fashion. Specifically, we first proposed an end-to-end differentiable framework that can calculate the weights over various dimensions in a soft and continuous manner for feature fields, and an AutoML-based optimization algorithm; then, we derive a hard and discrete embedding component architecture according to the maximal weights and retrain the whole recommender framework. We conduct extensive experiments on benchmark datasets to validate the effectiveness of AutoDim.

KEYWORDS

Embedding, Recommender System, AutoML

ACM Reference Format:
Xiangyu Zhao[1,2], Haochen Liu[1], Hui Liu[1], Jiliang Tang[1] and Weiwei Guo[3], Jun Shi[3], Sida Wang[3], Huiji Gao[3], Bo Long[4]. 2021. AutoDim: Field-aware Embedding Dimension Search in Recommender Systems. In *Proceedings of the Web Conference 2021 (WWW '21), April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3442381.3450124

1 INTRODUCTION

Real-world deep learning based recommender systems (DLRSs) typically involve a massive amount of categorical feature fields from users (e.g., occupation and userID), items (e.g., category and itemID), contextual information (e.g., time and location), and their interactions (e.g., user's purchase history of items). DLRSs first map these categorical features into real-valued dense vectors via an *embedding-component* [20, 39], i.e., the embedding-lookup process, which leads to huge amounts of embedding parameters. For instance, the YouTube recommender system consists of 1 million unique videoIDs, and assign each videoID with a specific 256-dimensional embedding vector; in other words, the videoID feature field alone occupies 256 million parameters [5]. Then, the DLRSs nonlinearly transform the input embeddings from all feature fields and generate the outputs (predictions) via the *MLP-component* (Multi-Layer Perceptron), which usually involves only several fully-connected layers in practice. Therefore, compared to the MLP-component, the embedding-component dominates the number of parameters in practical DLRSs, which naturally plays a tremendously impactful role in the recommendations.

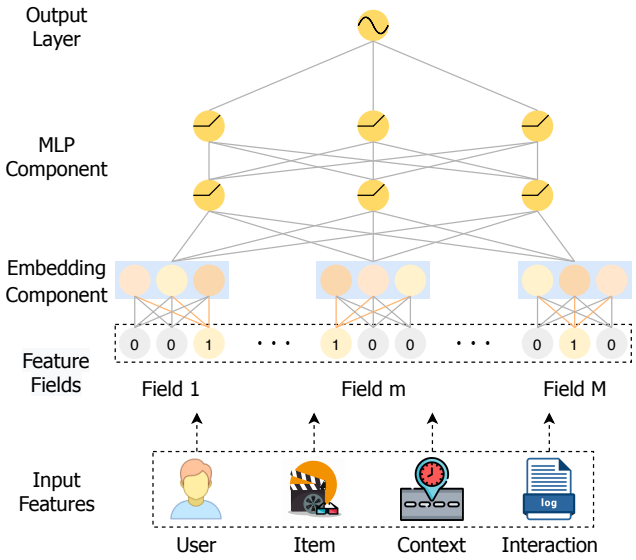

Figure 1: The Typically DLRS architecture.

The majority of existing recommender systems assign fixed and unified embedding dimension for all feature fields, such as the famous Wide&Deep model [4], which may lead to memory inefficiency. First, the embedding dimension often determines the capacity to encode information. Thus, allocating the same dimension to all feature fields may lose the information of highly predictive features while wasting memory on non-predictive features. Therefore, we should assign a large dimension to the highly informative and predictive features, for instance, the "location" feature in location-based recommender systems [1]. Second, different feature fields have different cardinality (i.e., the number of unique values). For

WWW '21, April 19–23, 2021, Ljubljana, Slovenia

ACM ISBN 978-1-4503-8312-7/21/04.
https://doi.org/10.1145/3442381.3450124

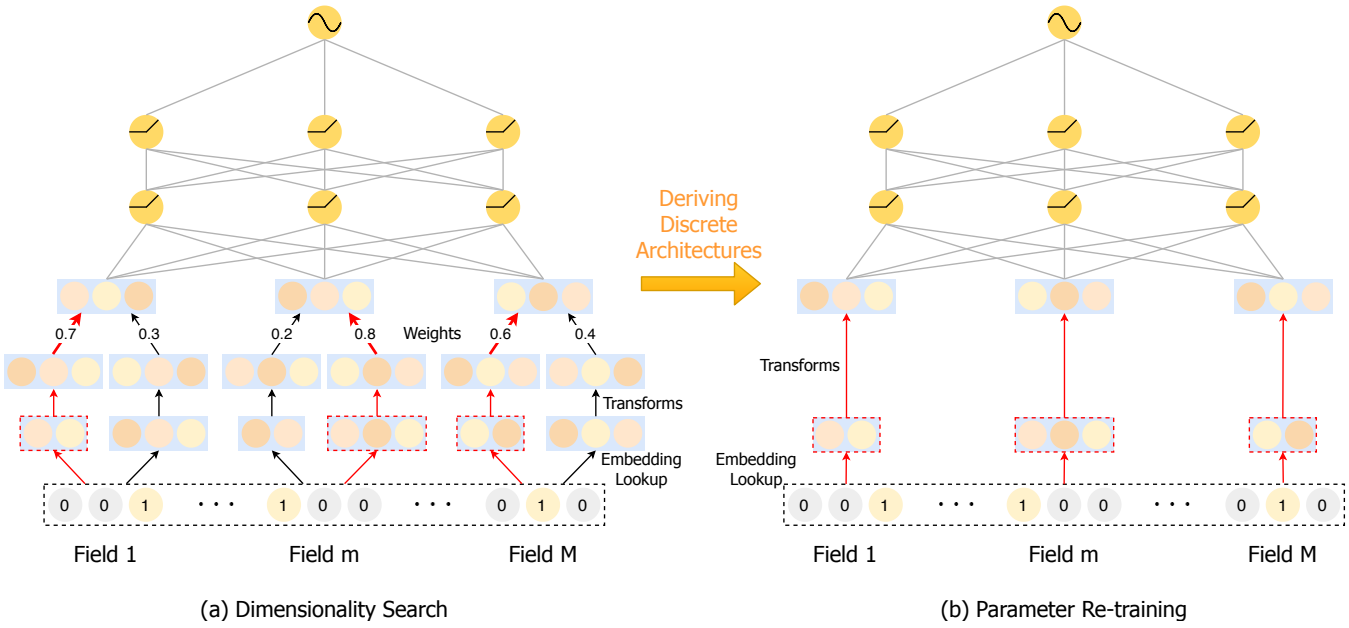

Figure 2: Overview of the proposed AutoDim framework.

example, the gender feature has only two (i.e., male and female), while the itemID feature usually involves millions of unique values. Intuitively, we should allocate larger dimensions to the feature fields with more unique feature values to encode their complex relationships with other features, and assign smaller dimensions to feature fields with smaller cardinality to avoid the overfitting problem due to the over-parameterization [8, 15]. According to the above reasons, it is highly desired to assign different embedding dimensions to different feature fields in a memory-efficient manner.

In this paper, we aim to enable different embedding dimensions for different feature fields for recommendations. We face several tremendous challenges. First, the relationship among embedding dimensions, feature distributions and neural network architectures is highly intricate, which makes it hard to manually assign embedding dimensions to each feature field [8]. Second, real-world recommender systems often involve hundreds and thousands of feature fields. It is difficult, if possible, to artificially select different dimensions for all feature fields, due to the expensive computation cost from the incredibly huge (N^M, with N the number of candidate dimensions for each feature field, and M the number of feature fields) search space. Our attempt to address these challenges results in an end-to-end differentiable AutoML-based framework (**AutoDim**), which can efficiently allocate embedding dimensions to different feature fields in an automated and data-driven manner. Our experiments on benchmark datasets demonstrate the effectiveness of the proposed framework. We summarize our major contributions as: (i) we identify the phenomenon that assigning various embedding dimensions to different feature fields can enhance recommendation performance; (ii) we propose an end-to-end AutoML-based framework AutoDim, which can automatically select various embedding dimensions to different feature fields; and (iii) we demonstrate the effectiveness of the proposed framework on real-world benchmark datasets.

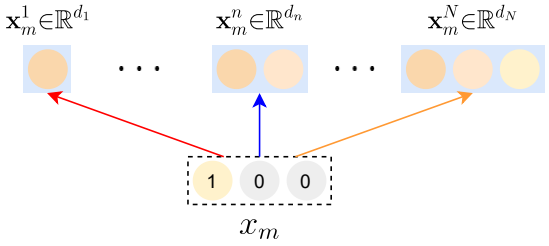

Figure 3: Embedding lookup method.

2 FRAMEWORK

In this section, we propose an AutoML-based framework, which effectively achieve the automated allocation of varying embedding dimensions to different feature fields. We illustrate the overall framework in Figure 2, which contains *dimensionality search stage* and *parameter re-training stage*.

2.1 Dimensionality Search

As the aforementioned challenges in Section 1, it is difficult to manually select embedding dimensions via conventional dimension reduction methods. An intuitive solution to tackle this challenge is to assign several embedding spaces with various dimensions to feature fields, and then the DLRS automatically selects the optimal embedding dimension for each feature field.

2.1.1 ***Embedding Lookup.*** Suppose for each user-item interaction instance, we have M input features $(x_1, \cdots, x_M)$, and each feature x_m belongs to a specific feature field, such as gender and age, etc. For the m^{th} feature field, we assign N candidate embedding spaces $\{\mathbf{X}_m^1, \cdots, \mathbf{X}_m^N\}$. The dimension of an embedding in each space is $d_1, \cdots, d_N$, where $d_1 < \cdots < d_N$; and the cardinality of these embedding spaces are the number of unique feature values

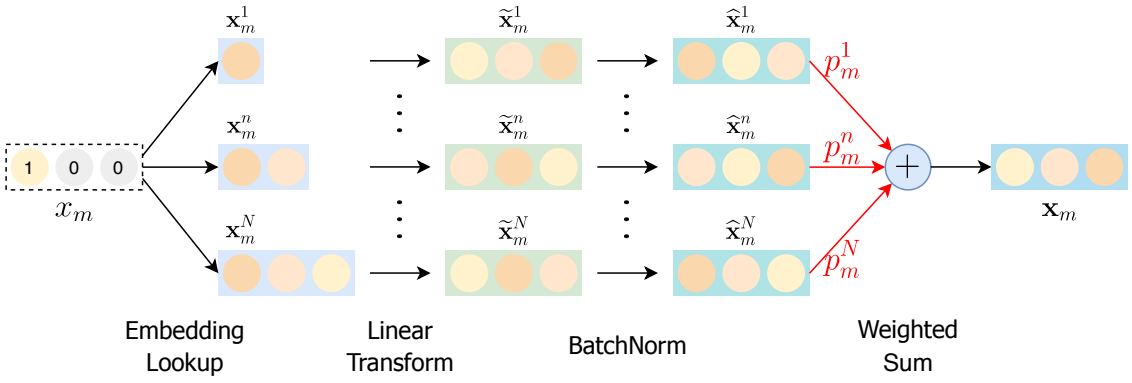

Figure 4: Linear transformation method to unify various dimensions.

in this feature field. Correspondingly, we define $\{\mathbf{x}_m^1, \cdots, \mathbf{x}_m^N\}$ as the set of candidate embeddings for a given feature x_m from all embedding spaces, as shown in Figure 3. Therefore, the total space assigned to the feature x_m is $\sum_{n=1}^{N} d_n$. Note that we assign the same candidate dimensions to all feature fields for simplicity, but it is straightforward to introduce different candidate sets.

2.1.2 ***Unifying Various Dimensions.*** Since the input dimension of the first MLP layer in existing DLRSs is often fixed, it is difficult for them to handle various candidate dimensions. Thus we need to unify the embeddings $\{\mathbf{x}_m^1, \cdots, \mathbf{x}_m^N\}$ into same dimension.

Figure 4 illustrates the linear transformation method to handle the various embedding dimensions. We introduce N fully-connected layers, which transform embedding vectors $\{\mathbf{x}_m^1, \cdots, \mathbf{x}_m^N\}$ into the same dimension d_N:

$$\widetilde{\mathbf{x}}_m^n \leftarrow \mathbf{W}_n^\top \mathbf{x}_m^n + \mathbf{b}_n \quad \forall n \in [1, N] \tag{1}$$

where $\mathbf{W}_n \in \mathbb{R}^{d_n \times d_N}$ is weight matrice and $\mathbf{b}_n \in \mathbb{R}^{d_N}$ is bias vector. For each field, all candidate embeddings with the same dimension share the same weight matrice and bias vector, which can reduce the amount of model parameters. With the linear transformations, we map the original embedding vectors $\{\mathbf{x}_m^1, \cdots, \mathbf{x}_m^N\}$ into the same dimensional space, i.e., $\{\widetilde{\mathbf{x}}_m^1, \cdots, \widetilde{\mathbf{x}}_m^N\} \in \mathbb{R}^{d_N}$. In practice, we can observe that the magnitude of the transformed embeddings $\{\widetilde{\mathbf{x}}_m^1, \cdots, \widetilde{\mathbf{x}}_m^N\}$ varies significantly, which makes them become incomparable. To tackle this challenge, we conduct BatchNorm [12] on the transformed embeddings $\{\widetilde{\mathbf{x}}_m^1, \cdots, \widetilde{\mathbf{x}}_m^N\}$ as:

$$\widehat{\mathbf{x}}_m^n \leftarrow \frac{\widetilde{\mathbf{x}}_m^n - \mu_{\mathcal{B}}^n}{\sqrt{(\sigma_{\mathcal{B}}^n)^2 + \epsilon}} \quad \forall n \in [1, N] \tag{2}$$

where $\mu_{\mathcal{B}}^n$ is the mini-batch mean and $(\sigma_{\mathcal{B}}^n)^2$ is the mini-batch variance for $\forall n \in [1, N]$. ϵ is a small constant added to the mini-batch variance for numerical stability when $(\sigma_{\mathcal{B}}^n)^2$ is very small. After BatchNorm, the linearly transformed embeddings $\{\widetilde{\mathbf{x}}_m^1, \cdots, \widetilde{\mathbf{x}}_m^N\}$ become to magnitude-comparable embedding vectors $\{\widehat{\mathbf{x}}_m^1, \cdots, \widehat{\mathbf{x}}_m^N\}$ with the same dimension d_N. Next, we will introduce embedding dimension selection process.

2.1.3 ***Dimension Selection.*** This paper aims to select the optimal embedding dimension for each feature field in an automated and data-driven manner. This is a hard (categorical) selection on the candidate embedding spaces, which will make the whole framework not end-to-end differentiable. To tackle this challenge, in this work, we approximate the hard selection over different dimensions via introducing the Gumbel-softmax operation [13], which simulates the non-differentiable sampling from a categorical distribution by a differentiable sampling from the Gumbel-softmax distribution.

To be specific, suppose weights $\{\alpha_m^1, \cdots, \alpha_m^N\}$ are the class probabilities over different dimensions. Then a hard selection z can be drawn via the the gumbel-max trick [9] as:

$$\begin{aligned} z &= \text{one\_hot}\left(\arg\max_{n \in [1,N]} \left[\log \alpha_m^n + g_n\right]\right) \\ where\ g_n &= -\log(-\log(u_n)) \\ u_n &\sim Uniform(0, 1) \end{aligned} \tag{3}$$

The *gumbel noises* $\{g_i, \cdots, g_N\}$ are i.i.d samples, which perturb $\{\log \alpha_m^n\}$ terms and make the arg max operation that is equivalent to drawing a sample by $\{\alpha_m^1, \cdots, \alpha_m^N\}$ weights. However, this trick is non-differentiable due to the arg max operation. To deal with this problem, we use the softmax function as a continuous, differentiable approximation to arg max operation, i.e., straight-through gumbel-softmax [13]:

$$p_m^n = \frac{\exp\left(\frac{\log(\alpha_m^n) + g_n}{\tau}\right)}{\sum_{i=1}^{N} \exp\left(\frac{\log(\alpha_m^i) + g_i}{\tau}\right)} \tag{4}$$

where τ is the temperature parameter, which controls the smoothness of the output of gumbel-softmax operation. When τ approaches zero, the output of the gumbel-softmax becomes closer to a one-hot vector. Then p_m^n is the probability of selecting the n^{th} candidate embedding dimension for the feature x_m, and its embedding $\mathbf{x}_m$ can be formulated as the weighted sum of $\{\widehat{\mathbf{x}}_m^1, \cdots, \widehat{\mathbf{x}}_m^N\}$:

$$\mathbf{x}_m = \sum_{n=1}^{N} p_m^n \cdot \widehat{\mathbf{x}}_m^n \quad \forall m \in [1, M] \tag{5}$$

We illustrate the weighted sum operations in Figure 4. With gumbel-softmax operation, the hard-like dimensionality search process is end-to-end differentiable. The discrete embedding dimension selection conducted based on the weights $\{\alpha_m^n\}$ will be detailed in the following subsections.

Then, we concatenate the embeddings $\mathbf{h}_0 = [\mathbf{x}_1, \cdots, \mathbf{x}_M]$ and feed $\mathbf{h}_0$ input into L multilayer perceptron layers:

$$\mathbf{h}_l = \sigma\left(\mathbf{W}_l^\top \mathbf{h}_{l-1} + \mathbf{b}_l\right) \quad \forall l \in [1, L] \tag{6}$$

where $\mathbf{W}_l$ and $\mathbf{b}_l$ are the weight matrix and the bias vector for the l^{th} MLP layer. $\sigma(\cdot)$ is the activation function such as *ReLU* and *Tanh*. Finally, the output layer that is subsequent to the last MLP layer, produces the prediction of the current user-item interaction instance as:

$$\hat{y} = \sigma\left(\mathbf{W}_o^\top \mathbf{h}_L + \mathbf{b}_o\right) \tag{7}$$

where $\mathbf{W}_o$ and $\mathbf{b}_o$ are the weight matrix and bias vector for the output layer. Activation function $\sigma(\cdot)$ is selected based on different recommendation tasks, such as Sigmoid for regression [4], and Softmax for multi-class classification [24]. Correspondingly, the objective function $\mathcal{L}(\hat{y}, y)$ between prediction $\hat{y}$ and ground truth label y also varies. In this work, we leverage negative log-likelihood function:

$$\mathcal{L}(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}) \tag{8}$$

where y is the ground truth (1 for like or click, 0 for dislike or non-click). By minimizing the objective function $\mathcal{L}(\hat{y}, y)$, the dimensionality search framework updates the parameters of all embeddings, hidden layers, and weights $\{\alpha_m^n\}$ through back-propagation. The high-level idea of the dimensionality search is illustrated in Figure 2 (a), where we omit some details of embedding-lookup, transformations and gumbel-softmax for the sake of simplicity.

2.2 Optimization

In this subsection, we will detail the optimization method of the proposed AutoDim framework. In AutoDim, we formulate the selection over different embedding dimensions as an architectural optimization problem and make it end-to-end differentiable by leveraging the Gumbel-softmax technique. The parameters to be optimized in AutoDim are two-fold, i.e., (i) $\mathbf{W}$: the parameters of the DLRS, including the embedding-component and the MLP-component; (ii) $\boldsymbol{\alpha}$: the architectural weights $\{\alpha_m^n\}$ on different embedding spaces ($\{p_m^n\}$ are calculated based on $\{\alpha_m^n\}$ as in Eq. (4)). DLRS parameters $\mathbf{W}$ and architectural weights $\boldsymbol{\alpha}$ can not be optimized simultaneously on the training dataset as with the conventional supervised attention mechanism since the optimization of them are highly dependent on each other. In other words, optimization on the training dataset simultaneously may result in the model overfitting on the examples from the training dataset.

Inspired by the differentiable architecture search (DARTS) techniques [17], $\mathbf{W}$ and $\boldsymbol{\alpha}$ are alternately optimized through gradient descent. Specifically, we alternately update $\mathbf{W}$ by optimizing the loss $\mathcal{L}_{train}$ on the training data and update $\boldsymbol{\alpha}$ by optimizing the loss $\mathcal{L}_{val}$ on the validation data:

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \ \mathcal{L}_{val}\big(\mathbf{W}^*(\boldsymbol{\alpha}), \boldsymbol{\alpha}\big) \\ &s.t. \ \mathbf{W}^*(\boldsymbol{\alpha}) = \arg\min_{\mathbf{W}} \mathcal{L}_{train}(\mathbf{W}, \boldsymbol{\alpha}^*) \end{aligned} \tag{9}$$

this optimization forms a bilevel optimization problem [19], where architectural weights $\boldsymbol{\alpha}$ and DLRS parameters $\mathbf{W}$ are identified as the upper-level variable and lower-level variable. Since the inner optimization of $\mathbf{W}$ is computationally expensive, directly optimizing $\boldsymbol{\alpha}$ via Eq.(9) is intractable. To address this challenge, we take advantage of the approximation scheme of DARTS:

$$\arg\min_{\mathbf{W}} \mathcal{L}_{train}(\mathbf{W}, \boldsymbol{\alpha}^*) \approx \mathbf{W} - \xi \nabla_{\mathbf{W}} \mathcal{L}_{train}(\mathbf{W}, \boldsymbol{\alpha}) \tag{10}$$

where ξ is the learning rate. In the approximation scheme, when updating $\boldsymbol{\alpha}$ via Eq.(10), we estimate $\mathbf{W}^*(\boldsymbol{\alpha})$ by descending the gradient $\nabla_{\mathbf{W}} \mathcal{L}_{train}(\mathbf{W}, \boldsymbol{\alpha})$ for only one step, rather than to optimize $\mathbf{W}(\boldsymbol{\alpha})$ thoroughly to obtain $\mathbf{W}^*(\boldsymbol{\alpha}) = \arg\min_{\mathbf{W}} \mathcal{L}_{train}(\mathbf{W}, \boldsymbol{\alpha}^*)$.

The DARTS based optimization algorithm for AutoDim is detailed in Algorithm 1. Specifically, in each iteration, we first sample a batch of user-item interaction data from the validation set (line 2); next, we update the architectural weights $\boldsymbol{\alpha}$ upon it (line 3); afterward, the DLRS make the predictions $\hat{y}$ on the batch of training data with current DLRS parameters $\mathbf{W}$ and architectural weights $\boldsymbol{\alpha}$ (line 4-5); eventually, we update the DLRS parameters $\mathbf{W}$ by descending $\nabla_{\mathbf{W}} \mathcal{L}_{train}(\mathbf{W}, \boldsymbol{\alpha})$ (line 6).

Algorithm 1 DARTS based Optimization for AutoDim.

Input: the features $(x_1, \cdots, x_M)$ of user-item interactions and the corresponding ground-truth labels y
Output: the well-learned DLRS parameters $\mathbf{W}^*$; the well-learned weights on various embedding spaces $\boldsymbol{\alpha}^*$

1: **while** not converged **do**
2: Sample a mini-batch of user-item interactions from validation data
3: Update $\boldsymbol{\alpha}$ by descending $\nabla_{\boldsymbol{\alpha}} \mathcal{L}_{val}\big(\mathbf{W}^*(\boldsymbol{\alpha}), \boldsymbol{\alpha}\big)$ with the approximation in Eq.(10)
4: Collect a mini-batch of training data
5: Generate predictions $\hat{y}$ via DLRS with current $\mathbf{W}$ and architectural weights $\boldsymbol{\alpha}$
6: Update $\mathbf{W}$ by descending $\nabla_{\mathbf{W}} \mathcal{L}_{train}(\mathbf{W}, \boldsymbol{\alpha})$
7: **end while**

2.3 Parameter Re-Training

Since the suboptimal embedding dimensions in the dimensionality search stage also influence the model training, a retraining stage is desired to train the model with only optimal dimensions, eliminating these suboptimal influences. This subsection will introduce how to select the optimal embedding dimension for each feature field and the details of retraining the recommender system with the selected embedding dimensions.

2.3.1 ***Deriving Discrete Dimensions.*** During re-training, the gumbel-softmax operation is no longer used, which means that the optimal embedding space (dimension) are selected for each feature field as the one corresponding to the largest weight, based on the well-learned $\boldsymbol{\alpha}$. It is formally defined as:

$$\mathbf{X}_m = \mathbf{X}_m^k, \quad where \ \ k = \arg\max_{n \in [1,N]} \alpha_m^n \quad \forall m \in [1, M] \tag{11}$$

Figure 2 (a) illustrates the architecture of AutoDim framework with a toy example about the optimal dimension selections based on two candidate dimensions, where the largest weights corresponding to the 1^{st}, m^{th} and M^{th} feature fields are 0.7, 0.8 and 0.6, then the embedding space $\mathbf{X}_1^1$, $\mathbf{X}_m^2$ and $\mathbf{X}_M^1$ are selected for these feature fields. The dimension of an embedding vector in these embedding spaces is d_1, d_2 and d_1, respectively.

Algorithm 2 The Optimization of DLRS Re-training Process.

Input: the features $(x_1, \cdots, x_M)$ of user-item interactions and the corresponding ground-truth labels y
Output: the well-learned DLRS parameters $\mathbf{W}^*$

1: **while** not converged **do**
2: Sample a mini-batch of training data
3: Generate predictions $\hat{y}$ via DLRS with current $\mathbf{W}$
4: Update $\mathbf{W}$ by descending $\nabla_{\mathbf{W}} \mathcal{L}_{train}(\mathbf{W})$
5: **end while**

2.3.2 ***Model Re-training.*** As shown in Figure 2 (b), given the selected embedding spaces, we can obtain unique embedding vectors $(\mathbf{x}_1, \cdots, \mathbf{x}_M)$ for features $(x_1, \cdots, x_M)$. Then we concatenate these embeddings and feeds them into hidden layers. Next, the prediction $\hat{y}$ is generated by the output layer. Finally, all the parameters of the DLRS, including embeddings and MLPs, will be updated via minimizing the supervised loss function $\mathcal{L}(\hat{y}, y)$ through back-propagation. The model retraining algorithm is detailed in Algorithm 2. The retraining process is based on the same training data as Algorithm 1.

Note that the majority of existing deep recommender algorithms (such as FM [21], DeepFM [10], xDeepFM [16]) capture the interactions between feature fields via interaction operations, such as inner product. These interaction operations require the embedding vectors from all fields to have the same dimensions. Therefore, the embeddings selected in Section 2.3.1 are still mapped into the same dimension as in Section 2.1.2. In the retraining stage, the BatchNorm operation is no longer in use, since there are no competitions between candidate embeddings in each field.

3 EXPERIMENTS

In this section, we first introduce experimental settings. Then we conduct extensive experiments to evaluate the effectiveness of the proposed AutoDim framework.

3.1 Dataset

We evaluate our model on benchmark Criteo dataset[1]: This is a benchmark industry dataset to evaluate ad click-through rate prediction models. It consists of 45 million users' click records on displayed ads over one month. For each data example, it contains 13 numerical feature fields and 26 categorical feature fields. We normalize numerical features by transforming a value $v \rightarrow \lfloor \log(v)^2 \rfloor$ if $v > 2$ as proposed by the Criteo Competition winner[2], and then convert it into categorical features through bucketing. All $M = 39$ feature fields are anonymous. We use 90% user-item interactions as the training/validation set (8:1), and the rest 10% as the test set.

3.2 Implement Details

Next, we detail the AutoDim architectures. For the DLRS, (i) embedding component: we set the maximal embedding dimension as 32 within our GPU memory constraints. For each feature field, we select from $N = 5$ candidate embedding dimensions $\{2, 8, 16, 24, 32\}$. (ii) MLP component: we have two hidden layers with the size $|h_0| \times 128$ and 128×128, where $|h_0|$ is the input size of first hidden layer, $|h_0| = 32 \times M$ with $M = 39$ the number of feature fields for Criteo dataset, and we use batch normalization, dropout ($rate = 0.2$) and ReLU activation for both hidden layers. The output layer is 128×1 with Sigmoid activation.

For architectural weights $\boldsymbol{\alpha}$: $\alpha_m^1, \cdots, \alpha_m^N$ of the m^{th} feature field are produced by a Softmax activation upon a trainable vector of length N. We use an annealing temperature $\tau = \max(0.01, 1 - 0.00005 \cdot t)$ for Gumbel-softmax, where t is the training step.

The learning rate for updating DLRS and weights are 0.001 and 0.001, and the batch-size is set as 2000. Our model can be applied to **any deep recommender systems with embedding layers**. In this paper, we show the performances of applying AutoDim on the well-known FM [21], W&D [4] and DeepFM [10].

3.3 Evaluation Metrics

The performance is evaluated by AUC, Logloss and Params, where a higher AUC or a lower Logloss indicates a better recommendation performance. A lower Params means fewer embedding parameters. A slightly higher AUC or lower Logloss at 0.001-level is regarded as significant for the CTR prediction task [4, 10]. For an embedding dimension search model, the "Params" metric is the optimal number of embedding parameters selected by this model for the recommender system. We omit the number of MLP parameters, which only occupy a small part of the total model parameters, e.g., $\sim 0.5\%$ in W&D and DeepFM on Criteo dataset. FM model has no MLP component.

3.4 Overall Performance

We compare the proposed framework with following embedding dimension search methods: (i) **FDE** (Full Dimension Embedding): In this baseline, we assign the maximal candidate dimension to all feature fields, i.e., 32. (ii) **MDE** (Mixed Dimension Embedding) [8]. (iii) **DPQ** (Differentiable Product Quantization) [3]. (iv) **NIS** (Neural Input Search) [14]. (v) **MGQE** (Multi-granular quantized embeddings) [15]. (vi) **AEmb** (Automated Embedding Dimensionality Search) [31]. (vii) **RaS** (Random Search): Random search is strong baseline in neural network search [17]. We apply the same candidate embedding dimensions, randomly allocate dimensions to feature fields in each experiment time, and report the best performance. (viii) **AD-s**: This baseline shares the same architecture with AutoDim, while we update the DLRS parameters and architectural weights simultaneously on the same training batch in an end-to-end backpropagation fashion.

The overall results are shown in Table 1. We can observe: ***(1)*** FDE achieves the worst recommendation performance and largest Params, where FDE is assigned the maximal embedding dimension 32 to all feature fields. This result demonstrates that allocating the same dimension to all feature fields is not only memory inefficient, but introduces numerous noises into the model. ***(2)*** RaS, AD-s, AutoDim performs better than MDE, DPQ, NIS, MGQE, AEmb. The major differences between these two groups of methods are: (i) the first group aims to assign different embedding dimensions to different feature fields, while embeddings in the same feature field share the same dimension; (ii) the second group attempts to assign different embedding sizes to different feature values within the

[1] https://www.kaggle.com/c/criteo-display-ad-challenge/
[2] https://www.csie.ntu.edu.tw/ r01922136/kaggle-2014-criteo.pdf

Table 1: Performance comparison of different embedding search methods

Dataset	Model	Metrics	Search Methods								
			FDE	MDE	DPQ	NIS	MGQE	AEmb	RaS	AD-s	AutoDim
Criteo	FM	AUC	0.8020	0.8027	0.8035	0.8042	0.8046	0.8049	0.8056	0.8063	**0.8078***
		Logloss	0.4487	0.4481	0.4472	0.4467	0.4462	0.4460	0.4457	0.4452	**0.4438***
		Params (M)	34.778	15.520	20.078	13.636	12.564	13.399	16.236	31.039	**11.632***
Criteo	W&D	AUC	0.8045	0.8051	0.8058	0.8067	0.8070	0.8072	0.8076	0.8081	**0.8098***
		Logloss	0.4468	0.4464	0.4457	0.4452	0.4446	0.4445	0.4443	0.4439	**0.4419***
		Params (M)	34.778	18.562	22.628	14.728	15.741	15.987	18.233	30.330	**12.455***
Criteo	DeepFM	AUC	0.8056	0.8060	0.8067	0.8076	0.8080	0.8082	0.8085	0.8089	**0.8101***
		Logloss	0.4457	0.4456	0.4449	0.4442	0.4439	0.4438	0.4436	0.4432	**0.4416***
		Params (M)	34.778	17.272	25.737	12.955	13.059	13.437	17.816	31.770	**11.457***

"*" indicates the statistically significant improvements (i.e., two-sided t-test with $p < 0.05$) over the best baseline. (M=Million)

same feature fields, which are based on the frequencies of feature values. The second group of methods surfer from several challenges: (ii-a) there are numerous unique values in each feature field, e.g., 2.7×10^4 values for each feature field on average in the Criteo dataset. This leads to a huge search space (even after bucketing) in each feature field, which makes it difficult to find the optimal solution, while the search space for each feature field is $N = 5$ in AutoDim; (ii-b) allocating dimensions solely based on feature frequencies (i.e., how many times a feature value appears in the training set) may lose other important characteristics of the feature; and (ii-c) the feature values frequencies are usually dynamic and not pre-known in real-time recommender systems, e.g., the cold-start users/items. ***(3)*** AutoDim outperforms RaS and AD-s, where AutoDim updates the architectural weights $\boldsymbol{\alpha}$ on the validation batch, which can enhance the generalization; AD-s updates the $\boldsymbol{\alpha}$ with DLRS on the same training batch simultaneously, which may lead to overfitting; RaS randomly search the dimensions, which has a large search space. AD-s has much larger Params than AutoDim, which indicates that larger dimensions are more efficient in minimizing training loss.

To sum up, compared with the representative baselines, AutoDim achieves significantly better recommendation performance, and saves 70% ∼ 80% embedding parameters. These results prove the effectiveness of the AutoDim framework.

3.5 Efficiency Analysis

In addition to model effectiveness, training and inference efficiency are also essential metrics for deploying a recommendation model into commercial recommender systems. This section investigates the efficiency of applying search methods to DeepFM on the Criteo dataset (on one Tesla K80 GPU). We illustrate the results in Figure 5.

For the training time in Figure 5 (a), we can observe that AutoDim and AD-s have fast training speed. As discussed in Section 3.4, the reason is that they have a smaller search space than other baselines. FDE's training is fastest since we directly set its embedding dimension as 32, i.e., no searching stage, while its recommendation performance is worst among all methods in Section 3.4. For the inference time, which is more crucial when deploying a model in commercial recommender systems, AutoDim achieves the least inference time, as shown in Figure 5 (b). This is because the final

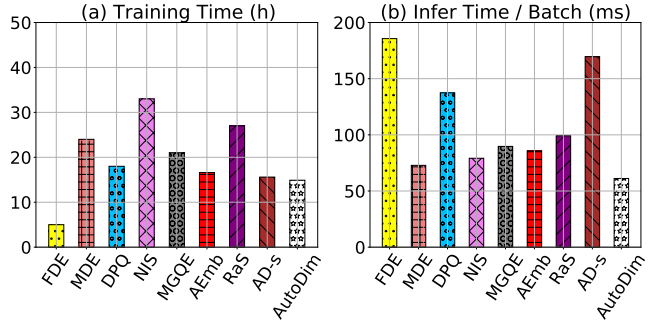

Figure 5: Efficiency analysis of DeepFM on Criteo dataset.

recommendation model selected by AutoDim has the least embedding parameters, i.e., the Params metric. To summarize, AutoDim can efficiently achieve better performance, making it easier to be launched in real-world recommender systems.

3.6 Parameter Analysis

In this section, we investigate how the essential hyper-parameters influence model performance. Besides common hyper-parameters of deep recommender systems such as the number of hidden layers (we omit them due to limited space), our model has one particular hyper-parameter, i.e., the frequency to update architectural weights $\boldsymbol{\alpha}$, referred to as f. In Algorithm 1, we alternately update DLRS's parameters on the training data and update $\boldsymbol{\alpha}$ on the validation data. In practice, we find that updating $\boldsymbol{\alpha}$ can be less frequently than updating DLRS's parameters, which apparently reduces lots of computations, and also enhances the performance.

To study the impact of f, we investigate how DeepFM with AutoDim performs on Criteo dataset with the changes of f, while fixing other parameters. Figure 6 shows the parameter sensitivity results, where in x-axis, $f = i$ means updating $\boldsymbol{\alpha}$ once, then updating DLRS's parameters i times. We can observe that the AutoDim achieves the optimal AUC/Logloss when $f = 10$. In other words, updating $\boldsymbol{\alpha}$ too frequently/infrequently results in suboptimal performance. Figure 6 (d) shows that setting $f = 10$ can reduce ∼ 50% training time compared with setting $f = 1$.

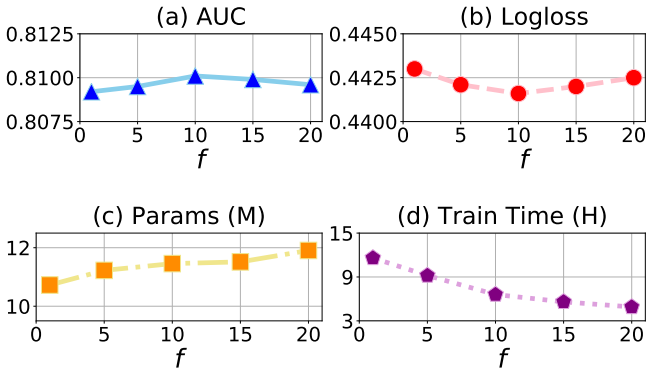

Figure 6: Parameter analysis of DeepFM on Criteo dataset.

Figure 6 (c) shows that lower f leads to lower Params, vice versa. The reason is that AutoDim updates $\boldsymbol{\alpha}$ by minimizing validation loss, which improves the generalizability of model [17, 19]. When updating $\boldsymbol{\alpha}$ frequently (e.g., $f = 1$), AutoDim tends to select a smaller embedding size that has better generalization, while may has an under-fitting problem; while when updating $\boldsymbol{\alpha}$ infrequently (e.g., $f = 20$), AutoDim prefers larger embedding sizes that perform better on the training set, but may lead to an over-fitting problem. $f = 10$ is a good trade-off between model performance on training and validation sets.

3.7 Case Study

In this section, we investigate whether AutoDim can assign larger embedding dimensions to more important features. Since feature fields are anonymous in Criteo, we apply W&D with AutoDim on MovieLens-1m dataset[3]. There are $M = 8$ categorical feature fields: movieId, year, genres, userId, gender, age, occupation, zip. Since MovieLens-1m is much smaller than Criteo, we set the candidate embedding dimensions as $\{2, 4, 8, 16\}$.

To measure the contribution of a feature field to the final prediction, we build a W&D model with only this field, train this model and evaluate it on the test set. A higher AUC and a lower Logloss means this feature field is more predictive for the final prediction. Then, we build a comprehensive W&D model incorporating all feature fields, and apply AutoDim to select the dimensions. The results are shown in Table 2. It can be observed that: ***(1)*** No feature fields are assigned 16-dimensional embedding space, which means candidate embedding dimensions $\{2, 4, 8, 16\}$ are sufficient to cover all possible choices. ***(2)*** Compared to the AUC/Logloss of W&D with each feature field, we can find that AutoDim assigns larger embedding dimensions to important (highly predictive) feature fields, such as movieId and userId, vice versa. ***(3)*** We build a full dimension embedding (FDE) version of W&D, where all feature fields are assigned as the maximal dimension 16. Its performances are AUC=0.8077, Logloss=0.5383, while the performances of W&D with AutoDim are AUC=0.8113, Logloss=0.5242, and it saves 57% embedding parameters.

In short, above observations validates that AutoDim can assign larger embedding dimensions to more predictive feature fields.

[3]https://grouplens.org/datasets/movielens/1m/

Table 2: Embedding dimensions for Movielens-1m

feature field	W&D (one field)		AutoDim
	AUC	Logloss	Dimension
movieId	0.7321	0.5947	8
year	0.5763	0.6705	2
genres	0.6312	0.6536	4
userId	0.6857	0.6272	8
gender	0.5079	0.6812	2
age	0.5245	0.6805	2
occupation	0.5264	0.6805	2
zip	0.6524	0.6443	4

4 RELATED WORK

In this section, we will discuss the related works. We summarize the works related to our research from two perspectives: deep recommender systems and AutoML for neural architecture search.

Deep recommender systems have drawn increasing attention from both the academia and the industry thanks to their great advantages over traditional methods [28]. Various types of deep learning approaches in recommendation are developed. Sedhain et al. [22] present an AutoEncoder based model named AutoRec. Hidasi et al. [11] introduce an RNN based recommender system named GRU4Rec. Cheng et al. [4] introduce a Wide&Deep framework for both regression and classification tasks. Guo et al. [10] propose the DeepFM model. It combines the factorization machine (FM) and MLP. Wang et al. [26] utilizes CNN to extract visual features to help POI (Point-of-Interest) recommendations. Wang et al. [25] propose a generative adversarial network (IRGAN) based information retrieval model. Zhao et al. [6, 7, 29, 30, 32–38, 42] propose a series of deep reinforcement learning based recommendation models.

The research of AutoML for neural architecture search can be traced back to NAS [40], which first utilizes an RNN based controller to design neural networks and proposes a reinforcement learning algorithm to optimize the framework. After that, many endeavors are conducted to reduce the high training cost of NAS. Pham et al. [19] propose ENAS, where the controller learns to search a subgraph from a large computational graph to form an optimal neural network architecture. Brock et al. [2] introduce a framework named SMASH, in which a hyper-network is developed to generate weights for sampled networks. DARTS [17] and SNAS [27] formulate the problem of network architecture search in a differentiable manner and solve it using gradient descent. Luo et al. [18] investigate representing network architectures as embeddings. Some works raise another way of thinking, which is to limit the search space. Zoph et al. [41] propose a transfer learning framework called NASNet, which train convolution cells on smaller datasets and apply them on larger datasets. Tan et al. [23] introduce MNAS. They propose to search hierarchical convolution cell blocks independently, so that a deep network can be built based on them.

5 CONCLUSION

In this paper, we propose a novel framework, AutoDim, which automatically assigns various embedding dimensions to different feature fields in a data-driven manner. To be specific, we first provide an

end-to-end differentiable model, which computes the weights over different dimensions for different feature fields simultaneously in a soft and continuous form, and we propose an AutoML-based optimization algorithm; then, according to the maximal weights, we derive a discrete embedding architecture, and re-train the DLRS parameters. We evaluate the AutoDim framework with extensive experiments based on widely used benchmark datasets. The results show that our framework can achieve better recommendation performance with much fewer embedding space demands.

ACKNOWLEDGMENTS

This work is supported by National Science Foundation (NSF) under grant numbers IIS1907704, IIS1928278, IIS1714741, IIS1715940, IIS1845081 and CNS1815636.

REFERENCES

[1] Jie Bao, Yu Zheng, David Wilkie, and Mohamed Mokbel. 2015. Recommendations in location-based social networks: a survey. *Geoinformatica* 19, 3 (2015), 525–565.
[2] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. 2017. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344* (2017).
[3] Ting Chen, Lala Li, and Yizhou Sun. 2019. Differentiable product quantization for end-to-end embedding compression. *arXiv preprint arXiv:1908.09756* (2019).
[4] 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*. ACM, 7–10.
[5] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. 191–198.
[6] Wenqi Fan, Tyler Derr, Xiangyu Zhao, Yao Ma, Hui Liu, Jianping Wang, Jiliang Tang, and Qing Li. 2020. Attacking Black-box Recommendations via Copying Cross-domain User Profiles. *arXiv preprint arXiv:2005.08147* (2020).
[7] Yingqiang Ge, Shuchang Liu, Ruoyuan Gao, Yikun Xian, Yunqi Li, Xiangyu Zhao, Changhua Pei, Fei Sun, Junfeng Ge, Wenwu Ou, et al. 2021. Towards Long-term Fairness in Recommendation. *arXiv preprint arXiv:2101.03584* (2021).
[8] Antonio Ginart, Maxim Naumov, Dheevatsa Mudigere, Jiyan Yang, and James Zou. 2019. Mixed Dimension Embeddings with Application to Memory-Efficient Recommendation Systems. *arXiv preprint arXiv:1909.11810* (2019).
[9] Emil Julius Gumbel. 1948. *Statistical theory of extreme values and some practical applications: a series of lectures*. Vol. 33. US Government Printing Office.
[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] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939* (2015).
[12] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*. 448–456.
[13] Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144* (2016).
[14] 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 SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2387–2397.
[15] Wang-Cheng Kang, Derek Zhiyuan Cheng, Ting Chen, Xinyang Yi, Dong Lin, Lichan Hong, and Ed H Chi. 2020. Learning Multi-granular Quantized Embeddings for Large-Vocab Categorical Features in Recommender Systems. *arXiv preprint arXiv:2002.08530* (2020).
[16] 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 SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1754–1763.
[17] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).
[18] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. 2018. Neural architecture optimization. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 7827–7838.
[19] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. 2018. Efficient Neural Architecture Search via Parameters Sharing. In *International Conference on Machine Learning*. 4095–4104.
[20] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. 2016. Product-based neural networks for user response prediction. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 1149–1154.
[21] Steffen Rendle. 2010. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE, 995–1000.
[22] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. 2015. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th international conference on World Wide Web*. 111–112.
[23] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. 2019. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2820–2828.
[24] Yong Kiam Tan, Xinxing Xu, and Yong Liu. 2016. Improved recurrent neural networks for session-based recommendations. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. 17–22.
[25] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. 2017. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. 515–524.
[26] Suhang Wang, Yilin Wang, Jiliang Tang, Kai Shu, Suhas Ranganath, and Huan Liu. 2017. What your images reveal: Exploiting visual contents for point-of-interest recommendation. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 391–400.
[27] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. 2018. SNAS: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926* (2018).
[28] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)* 52, 1 (2019), 1–38.
[29] Weinan Zhang, Xiangyu Zhao, Li Zhao, Dawei Yin, Grace Hui Yang, and Alex Beutel. 2020. Deep Reinforcement Learning for Information Retrieval: Fundamentals and Advances. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2468–2471.
[30] Xiangyu Zhao, Changsheng Gu, Haoshenglun Zhang, Xiaobing Liu, Xiwang Yang, and Jiliang Tang. 2019. Deep Reinforcement Learning for Online Advertising in Recommender Systems. *arXiv preprint arXiv:1909.03602* (2019).
[31] Xiangyu Zhao, Chong Wang, Ming Chen, Xudong Zheng, Xiaobing Liu, and Jiliang Tang. 2020. AutoEmb: Automated Embedding Dimensionality Search in Streaming Recommendations. *arXiv preprint arXiv:2002.11252* (2020).
[32] Xiangyu Zhao, Long Xia, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2019. Toward Simulating Environments in Reinforcement Learning Based Recommendations. *arXiv preprint arXiv:1906.11462* (2019).
[33] Xiangyu Zhao, Long Xia, Jiliang Tang, and Dawei Yin. 2019. Deep reinforcement learning for search, recommendation, and online advertising: a survey by Xiangyu Zhao, Long Xia, Jiliang Tang, and Dawei Yin with Martin Vesely as coordinator. *ACM SIGWEB Newsletter* Spring (2019), 4.
[34] Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2018. Deep Reinforcement Learning for Page-wise Recommendations. In *Proceedings of the 12th ACM Recommender Systems Conference*. ACM, 95–103.
[35] Xiangyu Zhao, Long Xia, Lixin Zou, Hui Liu, Dawei Yin, and Jiliang Tang. 2020. Whole-Chain Recommendations. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1883–1891.
[36] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Long Xia, Jiliang Tang, and Dawei Yin. 2018. Recommendations with Negative Feedback via Pairwise Deep Reinforcement Learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1040–1048.
[37] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Dawei Yin, Yihong Zhao, and Jiliang Tang. 2017. Deep Reinforcement Learning for List-wise Recommendations. *arXiv preprint arXiv:1801.00209* (2017).
[38] Xiangyu Zhao, Xudong Zheng, Xiwang Yang, Xiaobing Liu, and Jiliang Tang. 2020. Jointly learning to recommend and advertise. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3319–3327.
[39] Guorui Zhou, Xiaoqiang Zhu, Chenru 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 *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1059–1068.
[40] Barret Zoph and Quoc V Le. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* (2016).
[41] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 8697–8710.
[42] Lixin Zou, Long Xia, Yulong Gu, Xiangyu Zhao, Weidong Liu, Jimmy Xiangji Huang, and Dawei Yin. 2020. Neural Interactive Collaborative Filtering. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 749–758.