# Field-wise Embedding Size Search via Structural Hard Auxiliary Mask Pruning for Click-Through Rate Prediction

Tesi Xiao[1], Xia Xiao[2], Ming Chen[2] and Youlong Cheng[2,*]

[1]University of California, Davis, USA
[2]ByteDance, Mountain View, USA

Abstract

Feature embeddings are one of the most important steps when training deep learning based Click-Through Rate prediction models, which map high-dimensional sparse features to dense embedding vectors. Classic human-crafted embedding size selection methods are shown to be "sub-optimal" in terms of the trade-off between memory usage and model capacity. The trending methods in Neural Architecture Search (NAS) have demonstrated their efficiency to search for embedding sizes. However, most existing NAS-based approaches suffer from expensive computational costs, the curse of dimensionality of the search space, and the discrepancy between continuous search space and discrete candidate space. Other methods that prune embeddings in an unstructured manner fail to explicitly reduce the computational costs. In this paper, to address those limitations, we propose a novel strategy that searches for the optimal mixed-dimension embedding scheme by structurally pruning a super-net via Hard Auxiliary Mask. Our method aims to directly search candidate models in the discrete space using a simple and efficient gradient-based method. Furthermore, we introduce orthogonal regularity on embedding tables to reduce correlations within embedding columns and enhance representation capacity. Extensive experiments demonstrate it can effectively remove redundant embedding dimensions without great performance loss.

**Keywords**

CTR Prediction, Embedding Size, Network Pruning, Neural Architecture Search

## 1. Introduction

Deep learning based recommender systems (DLRS) have demonstrated superior performance over more traditional recommendation techniques [3]. The success of DLRS is mainly attributed to their ability to learn meaningful representations with categorical features, that subsequently help with modeling the non-linear user-item relationships efficiently. Indeed, real-world recommendation tasks usually involve a large number of categorical feature fields with high *cardinality* (i.e. the number of unique values or vocabulary size) [4]. *One-Hot Encoding* is a standard way to represent such categorical features. To reduce the memory cost of *One-Hot Encoding*, DLRS first maps the high-dimensional one-hot vectors into real-valued dense vectors via the *embedding* layer. Such embedded vectors are subsequently used in predictive models for obtaining the required recommendations.

In this pipeline, the choice of the dimension of the embedding vectors, also known as *embedding dimension*, plays a crucial role in the overall performance of the DLRS. Most existing models assign a fixed and uniform embedding dimension for all features, either due to the prerequisites of the model input or simply for the sake of convenience. If the embedding dimensions are uniformly high, it leads to increased memory usage and computational cost, as it fails to handle the heterogeneity among different features. As a concrete example, encoding features with few unique values like *gender* with large embedding vectors leads to over-parametrization. Conversely, the selected embedding size may be insufficient for highly-predictive features with large cardinalities, such as the user's *last search query*. Therefore, finding appropriate embedding dimensions for different feature fields is essential.

The existing works towards automating embedding size search can be categorized into two groups: (i) *field-wise* search [5, 2, 6]; (ii) *vocabulary-wise* search [7, 8, 9, 10]. The former group aims to assign different embedding sizes to different feature fields, while embeddings in the same feature field share the same dimension. The latter further attempts to find different embedding sizes to different feature values within the same feature field, which is generally based on the frequencies of feature values. Although it has been shown that the latter group can significantly reduce the model size without a great performance drop, the second group of works suffers from several challenges and drawbacks (we refer readers to Section 3.4 in [2] for details): (a) a large number of unique values in each feature field leads to a huge search space in which the optimal solution is difficult to find; (b) the feature values frequencies are time-varying and

*DL4SR'22: Workshop on Deep Learning for Search and Recommendation, co-located with the 31st ACM International Conference on Information and Knowledge Management (CIKM), October 17-21, 2022, Atlanta, USA*

*Corresponding author.
texiao@ucdavis.edu (T. Xiao); x.xiaxiao@bytedance.com (X. Xiao); ming.chen@bytedance.com (M. Chen); youlong.chen@bytedance.com (Y. Cheng)

© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

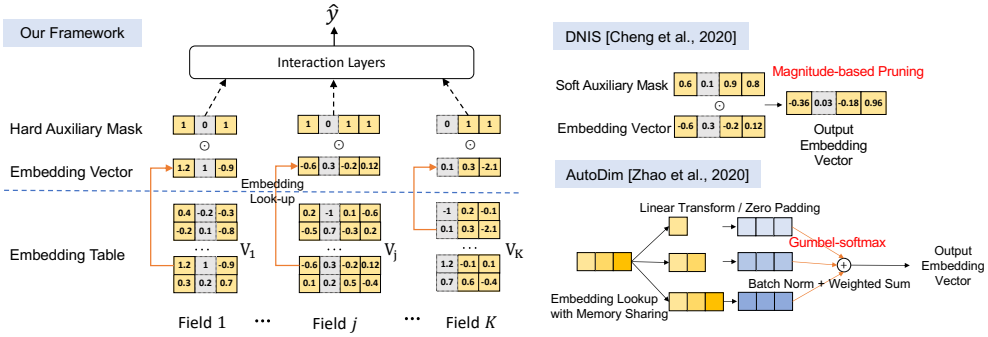


**Figure 1:** (*left*) The framework of our method. The operator $\odot$ denotes the element-wise product. The gray embedding components are masked and thus can be removed directly. (*right*) The basic ideas of DNIS [1] and AutoDim [2].

not pre-known in real-time recommender system; (c) it is difficult to handle embeddings with different dimensions for the same feature field in the state of art DLRS due to the feature crossing mechanism. As a result, we fix our attention to the *field-wise* embedding size search in this work.

The majority of recent works towards automating embedding size search are based on the trending ideas in Neural Architecture Search (NAS). NAS has been an active research area recently, which is composed of three elements: (i) a search space $\mathcal{A}$ of all candidate models; (ii) a search strategy that goes through models in $\mathcal{A}$; (iii) a performance estimation strategy that evaluates the performance of the selected model. To leverage the NAS techniques to search embedding sizes for each feature field, Joglekar et al. [7] formulate the search space by discretizing an embedding matrix into several sub-matrices and take a Reinforcement Learning (RL) approach with the *Trainer-Controller* framework for search and performance estimation, in which the controller samples different embedding blocks to maximize the reward observed from the trainer that trains the model with the controller's choices. Liu et al. [6] also adopt the RL-based search which starts with small embedding sizes and takes actions to keep or enlarge the size based on the prediction error. Their proposed method requires sequentially training DRLS and the policy network repeatedly until convergence, which is computationally expensive.

Given the rise of *one-shot NAS* aiming to search and evaluate candidate models together in an over-parametrized supernet, Zhao et al. [5, 2] follow the idea of Differential Neural Architecture Search (DARTS)[11] and solve a continuous bilevel optimization by the gradient-based algorithm. To be specific, they define a search space that includes all possible embedding sizes of each feature field; for instance in [2], 5 candidate sizes $\{2, 8, 16, 24, 32\}$ are selected for each feature field. These embedding vectors in different sizes are then lifted into the same dimension with batch normalization so that they can be aggregated with architecture weights of candidate sizes and fed into the subsequent layers to obtain the prediction. Cheng et al. [1] also leverage the idea of DARTS but with the architecture weights being the weights of sub-blocks of embedding matrices.

Despite the obtained promising results, the existing methods have certain limitations which arise from the discrepancy between the large discrete candidate space and its continuous relaxation. On the one hand, the DARTS-based methods in [5, 2, 1] are algorithmically efficient but search in an augmented continuous space; it has been shown that the discrepancy may lead to finding unsatisfactory candidates because the magnitude of architecture weights does not necessarily indicate how much the operation contributes to the overall performance [12]. On the other hand, while the RL-based methods in [7, 6] and the hard selection variant in [5] directly search and evaluate models in the original discrete candidate space, they are computationally heavy and thus not favored by the large-scale DLRS. Therefore, a natural question follows:

*Is it possible to come up with an efficient method that searches straight through the discrete candidate space of all possible embedding sizes?*

In this work, we provide a **positive** answer by proposing an *end-to-end field-aware* embedding size search method leveraging the Straight Through Estimator (STE) of gradients for non-differentiable functions. Motivated by one-shot NAS and network pruning, our method seeks to adaptively search for a good sub-network in a pre-trained supernet by masking redundant dimensions. The mask, named as **Hard Auxiliary Mask** (HAM), is realized by the indicator function of auxiliary weights. Furthermore, to reduce the information redundancy and boost the search performance, we impose the orthogonal

regularity on embedding tables and train models with regularized loss functions. **Contributions and novelty** of our work are: (i) we propose to prune embedding tables column-wisely via hard auxiliary masks for the CTR prediction models, which can effectively compress the model; (ii) we offer a gradient-based pruning method and delve into its dynamics with mathematical insights; (iii) we introduce orthogonal regularity for training recommendation models and experimentally show that orthogonal regularity can help to achieve significant improvement; (iv) we obtain state-of-the-art results on various modern models and our method is scalable on large datasets.

## 2. Related Work

**Embedding Size Search for Recommender Systems.** To deal with the heterogeneity among different features and reduce the memory cost of sizable embedding tables, several recent works introduce a new paradigm of mixed-dimension embedding tables. In particular, Ginart et al. [10] propose to substitute the uniform-dimension embeddings $\mathbf{V} \in \mathbb{R}^{C\times d}$ with $\overline{\mathbf{V}} = \mathbf{EP}$, where $\mathbf{E} \in \mathbb{R}^{C\times s}$ is the smaller embeddings with $s \ll d$ and $\mathbf{P} \in \mathbb{R}^{s\times d}$ is the projection that lift the embeddings into the base dimension $d$ for feature crossing purposes. A popularity-based rule is further designed to find the dimension $s$ for each feature field which is too rough and cannot handle important features with low frequency. In addition, plenty of works seek to provide an end-to-end framework leveraging the advances in NAS, including RL approaches [7, 6] and differentiable NAS methods [5, 13, 2, 1]. The most relevant work to us is [1], which employs a soft auxiliary mask to prune the embeddings structurally. However, it requires magnitude-based pruning to derive fine-grained mixed embedding, in which the discrepancy between the continuous relaxed search space and discrete candidate space could lead to a relatively great loss in performance. The most recent work [14] proposes a Plug-in Embedding Pruning (PEP) approach to prune embedding tables into sparse matrices to reduce storage costs, Albeit significantly reducing the number of non-zero embedding parameters, this type of unstructured pruning method fails to explicitly reduce the embedding size.

**Neural Network Pruning via Auxiliary Masks.** To deploy deep learning models on resource-limited devices, there is no lack of work in neural network pruning; see [15] and the reference therein. One popular approach towards solving this problem is to learn a pruning mask through auxiliary parameters, which considers pruning as an optimization problem that tends to minimize the supervised loss of the masked neural network with a certain sparsity constraint. As learning the optimal mask is indeed a discrete optimization problem for binary inputs, existing works attempt to cast it into a differentiable problem and provide gradient-based search algorithms. A straightforward method is to directly replace binary values by smooth functions of auxiliary parameters during the forward pass, such as sigmoid [16], Gumbel-sigmoid [17], softmax [11], Gumbel-softmax [18], and piece-wise linear [19]. However, all these methods, which we name Soft Auxiliary Mask (SAM), suffer from the discrepancy caused by continuous relaxation. Other methods, which we refer to Hard Auxiliary Mask (HAM), preserve binary values in the forward pass with indicator functions [20] or Bernoulli random variables [21] and optimize parameters using the straight-through estimator (STE) [22, 23]. A detailed comparison of four representative auxiliary masking methods for approximating binary values is provided in Table 1.

## 3. Methodology

### 3.1. Preliminaries

Here we briefly introduce the mechanism of DLRS and define the terms used throughout the paper.

**Model Architecture.** Consider the data input involves $K$ features fields from users, items, their interactions, and contextual information. We denote these raw features by multiple one-hot vectors $\mathbf{x}_1 \in \mathbb{R}^{C_1}, \ldots, \mathbf{x}_K \in \mathbb{R}^{C_K}$, where field dimensions $C_1, \ldots C_K$ are the cardinalities of feature fields[1]. Architectures of DLRS often consist of three key components: (i) *embedding* layers with tables $\mathbf{V}_1 \in \mathbb{R}^{C_1\times d_1}, \ldots, \mathbf{V}_K \in \mathbb{R}^{C_K\times d_K}$ that map sparse one-hot vectors to dense vectors in a low dimensional embedding space by $\mathbf{v}_i = \mathbf{V}_i^\top \mathbf{x}_i$; (ii) *feature interaction* layers that model complex feature crossing using embedding vectors; (iii) *output* layers that make final predictions for specific recommendation tasks.

The feature crossing techniques of existing models fall into two types - *vector-wise* and *bit-wise*. Models with *vector-wise* crossing explicitly introduce interactions by the inner product, such as Factorization Machine (FM) [26], DeepFM [27] and AutoInt [28]. The *bit-wise* crossing, in contrast, implicitly adds interaction terms by element-wise operations, such as the outer product in Deep Cross Network (DCN) [29], and the Hadamard product in NFM [30] and DCN-V2 [31]. In this work, we deploy our framework to the Click-Through Rate (CTR) prediction problem with four base models: FM, DeepFM, AutoInt, and DCN-V2.

**Notations.** Throughout this paper, we use $\odot$ for the Hadamard (element-wise) product of two vectors. The indicator function of a scalar-valued $\alpha$ is defined as $\mathbf{1}_{\alpha>0} = 1$ if $\alpha > 0$; $\mathbf{1}_{\alpha>0} = 0$ if $\alpha \leq 0$. The

[1] Numeric features are converted into categorical data by binning.

**Table 1**
A List of Representative Auxiliary Masking Methods

| Mask | | Forward | Backward | $\text{Model}_{\text{eval}} = \text{Model}_{\text{sel}}$ |
|---|---|---|---|---|
| Soft | Deterministic | $\alpha \in [0,1]$[1]; $\text{Sigmoid}(\alpha), \alpha \in \mathbb{R}$[16] | autograd | ✗ |
| | Stochastic | $\text{Sigmoid}(\log\alpha + \log(\frac{u}{1-u}))$ [17], $\alpha \in \mathbb{R}, u \sim \text{Uniform}(0,1)$ | autograd | ✗ |
| Hard | Stochastic | Bernoulli(p) | STE [24], Gumbel-STE [21] | ✗ |
| | Deterministic | $\mathbf{1}_{\alpha>0}$ | STE [20, 25] | (Our Approach) ✓ |

**Remark.** $\text{Model}_{\text{eval}}$ stands for the evaluated masked model; $\text{Model}_{\text{sel}}$ is the final model selected by the algorithm.

indicator function of a vector $\boldsymbol{\alpha} \in \mathbb{R}^d$ is defined as $\mathbb{1}_{\boldsymbol{\alpha}>0} = [\mathbf{1}_{\alpha_1>0}, \ldots, \mathbf{1}_{\alpha_d>0}]^\top$. The identity matrix is written as $\mathbf{I}$ and the function $\text{diag}(\cdot)$ returns a diagonal matrix with its diagonal entries as the input. We use $\|\cdot\|_1, \|\cdot\|_F$ for the $\ell_1$ norm of vectors and the Frobenius norm of matrices respectively.

## 3.2. Background

In general, the CTR prediction models takes the concatenation of all feature fields from a user-item pair, denoted by $\mathbf{x} = [\mathbf{x}_1; \mathbf{x}_2; \ldots; \mathbf{x}_K]$ as the input vector. Given the embedding layer $\mathbb{V} = \{\mathbf{V}_1, \mathbf{V}_2, \ldots, \mathbf{V}_K\}$, the feature interaction layers take the embedding vectors $\mathbf{v}$ and feed the learned hidden states into the output layer to obtain the prediction. The embedding vectors $\mathbf{v}$ are the dense encoding of the one-hot input $\mathbf{x}$ that can be defined as follows:

$$\begin{aligned}\mathbf{v} &= [\mathbf{v}_1; \mathbf{v}_2; \ldots; \mathbf{v}_K] \\ &= \left[\mathbf{V}_1^\top \mathbf{x}_1; \mathbf{V}_2^\top \mathbf{x}_2; \ldots; \mathbf{V}_K^\top \mathbf{x}_K\right] := \mathcal{V}\mathbf{x},\end{aligned}$$

where $\mathcal{V}$ is the embedding lookup operator. The prediction score $\hat{y}$ is then calculated with models' other parameters $\boldsymbol{\Theta}$ in the feature interaction layers and output layer by $\hat{y} = \psi(\mathbf{v}|\boldsymbol{\Theta}) = \psi(\mathcal{V}\mathbf{x}|\boldsymbol{\Theta}) = \phi(\mathbf{x}|\mathbb{V}, \boldsymbol{\Theta})$, where $\hat{y} \in [0,1]$ is the predicted probability, $\psi(\cdot|\Theta)$ is the prediction function of embedding vectors, and $\phi = \psi \circ \mathcal{V}$ is the prediction function of raw inputs. To learn the model parameters $\mathbb{V}, \boldsymbol{\Theta}$, we aim to minimize the Log-loss on the training data, i.e.,

$$\begin{aligned}&\min_{\mathbb{V},\boldsymbol{\Theta}} \mathcal{L}_{\text{train}}(\mathbb{V}, \boldsymbol{\Theta}) \\ &\quad := -\frac{1}{N}\sum_{j=1}^{N}\left[y_j \log(\hat{y}_j) + (1-y_j)\log(1-\hat{y}_j)\right]\end{aligned}$$

where $N$ is the total number of training samples.

**Hard Auxiliary Mask** As illustrated in Figure 1, we add auxiliary masks for each embedding dimension slot. Specifically, the auxiliary masks are indicator functions of auxiliary parameters $\boldsymbol{\alpha} = [\boldsymbol{\alpha}_1; \boldsymbol{\alpha}_2; \ldots; \boldsymbol{\alpha}_K]$, where $\boldsymbol{\alpha}_i \in \mathbb{R}^{d_i}$ is in the same size of the corresponding embedding vector $\mathbf{v}_i$. Provided with the masks, the predicted probability score $\tilde{y}$ is given as follows:

$$\tilde{y} = \psi(\mathbf{v} \odot \mathbb{1}_{\boldsymbol{\alpha}>0}|\boldsymbol{\Theta}) = \phi(\mathbf{x}|\tilde{\mathbb{V}}_{\boldsymbol{\alpha}}, \boldsymbol{\Theta}) = \phi(\mathbf{x}|\boldsymbol{\alpha}, \mathbb{V}, \boldsymbol{\Theta}),$$

where $\tilde{\mathbb{V}}_{\boldsymbol{\alpha}} = \{\tilde{\mathbf{V}}_1^{\boldsymbol{\alpha}_1}, \ldots, \tilde{\mathbf{V}}_K^{\boldsymbol{\alpha}_K}\}$ is the pruned embedding layer with

$$\tilde{\mathbf{V}}_i^{\boldsymbol{\alpha}_i} = \mathbf{V}_i \,\text{diag}(\mathbb{1}_{\boldsymbol{\alpha}_i>0}), \quad i = 1, \ldots K.$$

We emphasize that embedding tables $\tilde{\mathbf{V}}_i^{\boldsymbol{\alpha}_i}$ are pruned column-wisely in a structural manner unlike PEP [14] that prunes entry-wisely.

**Orthogonal Regularity** Given the embedding table $\mathbf{V}_j \in \mathbb{R}^{C_j \times d_j}$ for feature field $j$, its column vectors, denoted by $\mathbf{V}_{j,1}, \ldots, \mathbf{V}_{j,d_j} \in \mathbb{R}^{C_j}$, can be regarded as $d_j$ different representations of feature $j$ in the embedding space. The auxiliary masks above aim to mask relatively uninformative column vectors so as to reduce the model size. Nonetheless, the presence of correlation between these vectors may complicate the selection procedure. Specifically, presuming that the most predictive one $\mathbf{V}_{j,p}$ has been selected, it would be problematic if we greedily select the next column $\mathbf{V}_{j,q}$ that brings the largest loss drop when included in. For instance, if $\mathbf{V}_{j,q} \not\perp \mathbf{V}_{j,p}$, we have the following decomposition: $\mathbf{V}_{j,q} = \mathbf{p} + \mathbf{p}^\perp$, where $\mathbf{p} = c\mathbf{V}_{j,p} \parallel \mathbf{V}_{j,p}$ and $\mathbf{p}^\perp \perp \mathbf{p}$. Therefore, it would be difficult to determine whether the increments are attributed to the existing direction $\mathbf{p}$ or the new factor $\mathbf{p}^\perp$. To address this issue, we follow [32] to train embedding parameters $\mathbb{V}$ with Soft Orthogonal (SO) regularizations:

$$\mathcal{R}(\mathbb{V}) = \sum_{j=1}^{K} \|\mathbf{V}_j^\top \mathbf{V}_j - \mathbf{I}\|_F^2 / d_j^2, \tag{1}$$

where divisors $d_j^2$ are introduced to handle heterogeneous dimensionality of embedding tables. We also adopt a relaxed SO regularization in which $\mathbf{V}_j$ replaced by the normalized matrix $\overline{\mathbf{V}}_j$ with unit column vectors, which corresponds to the pair-wise cosine similarities within embedding columns [33].

## 3.3. Framework

Our proposed framework is motivated by one-shot NAS, which consists of three stages: *pretrain*, *search*, and *retrain*.

**Pretrain.** As shown in [34], pre-training architecture representations improve the downstream architecture search efficiency. Therefore, in the pretraining stage, we train the base model with a large embedding layer $\mathbb{V}$. The base dimension $d_j$ for each feature field is determined by prior knowledge. In addition, the embedding dimension $d_j$ should not exceed the field dimension $C_j$ to avoid column-rank-deficiency. The SO regularization term (1) is added to the mini-batch training loss for the optimizer to learn near-orthogonal embeddings. The learned model parameters, denoted by $\mathbb{V}_{\text{init}}, \boldsymbol{\Theta}_{\text{init}}$, are passed to the search stage as initialization.

**Search.** Provided with the pre-trained model, the goal of the search stage is to find the column-wise sparse embedding layer that preserves model accuracy, which can be formulated as:

$$\min_{\boldsymbol{\alpha}} \min_{\mathbb{V},\boldsymbol{\Theta}} \quad \mathcal{L}_{\text{train}}\left(\tilde{\mathbb{V}}_{\boldsymbol{\alpha}}, \boldsymbol{\Theta}\right) + \mu\left|\|\mathbb{1}_{\boldsymbol{\alpha}>0}\|_1 - s\right|,$$

where $\|\mathbb{1}_{\boldsymbol{\alpha}>0}\|_1$ counts the number of non-zero embedding columns and $s$ is the target number of non-zero columns. Note that instead of direct regularization on $\|\mathbb{1}_{\boldsymbol{\alpha}>0}\|_1$ as [16, 20, 25] do, we include the target number $s$ to reduce instability from batched training and the choice of hyperparameter $\mu$. However, given that the objective function above is non-differentiable when $\alpha = 0$ and has a zero gradient anywhere else, traditional gradient descent methods are not applicable. To that end, we use the straight-through estimator (STE) [22], which replaces the ill-defined gradient in the chain rule with a fake gradient. Despite various smooth alternatives used in the literature, such as sigmoid [16] and piecewise polynomials [35], we adopt the simplest identity function for backpropagation, i.e.,

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\alpha}} = \frac{\partial \mathcal{L}}{\partial \mathbb{1}_{\boldsymbol{\alpha}>0}} \frac{\partial \mathbb{1}_{\boldsymbol{\alpha}>0}}{\partial \boldsymbol{\alpha}} \approx \frac{\partial \mathcal{L}}{\partial \mathbb{1}_{\boldsymbol{\alpha}>0}} \frac{\partial \boldsymbol{\alpha}}{\partial \boldsymbol{\alpha}} = \frac{\partial \mathcal{L}}{\partial \mathbb{1}_{\boldsymbol{\alpha}>0}}. \quad (2)$$

Then, the search stage starts with the unmasked model that $\boldsymbol{\alpha}_0 = \epsilon \cdot \vec{1}$ for some small $\epsilon > 0$. The gradient update rules for $\boldsymbol{\alpha}$ at iteration $t$ are given by:

$$\boldsymbol{\alpha}_{t+1} = \boldsymbol{\alpha}_t - \eta \cdot \nabla_{(\mathbb{1}_{\boldsymbol{\alpha}_t>0})} \mathcal{L}_{\text{batch}} - \mu \cdot \text{sign}(\|\mathbb{1}_{\boldsymbol{\alpha}_t>0}\|_1 - s)\vec{1}, \quad (3)$$

where $\eta$ is the learning rate. We will illustrate in Section 3.4 that the above updates enjoy certain benefits and the last term plays an important role by pushing the optimizer iteratively to evaluate candidate models with a hard auxiliary mask. Furthermore, to enhance the stability and performance, we implement a multi-step training through iteratively training auxiliary parameters on validation data and retraining the original weights, which attempts to solve the following bi-level optimization problem:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \mathcal{L}_{\text{val}}\left(\tilde{\mathbb{V}}^{\star}_{\boldsymbol{\alpha}}, \boldsymbol{\Theta}^{\star}\right) + \mu\left|\|\mathbb{1}_{\boldsymbol{\alpha}>0}\|_1 - s\right| \\ \text{s.t.} \quad & \tilde{\mathbb{V}}^{\star}_{\boldsymbol{\alpha}}, \boldsymbol{\Theta}^{\star} = \arg\min_{\tilde{\mathbb{V}}_{\boldsymbol{\alpha}}, \boldsymbol{\Theta}} \quad \mathcal{L}_{\text{train}}\left(\tilde{\mathbb{V}}_{\boldsymbol{\alpha}}, \boldsymbol{\Theta}\right). \end{aligned}$$

**Early Stopper for the Search Stage.** We can determine to stop the search stage if the number of positive auxiliary weights is close to the target size with no significant gain in validation AUC since the sign of auxiliary weights exactly indicates whether the corresponding embedding components are pruned or not. On the contrary, it is hard to deploy the early stopper for the search stage using other auxiliary mask pruning methods because the value of validation AUC during the search epochs is unable to represent the performance of the model selected in the retraining stage.

**Retrain.** After training $\{\boldsymbol{\alpha}, \mathbb{V}, \boldsymbol{\Theta}\}$ for several epochs in the search step, we obtain a binary mask based on the sign of $\boldsymbol{\alpha}$ and continue optimizing $\{\mathbb{V}, \boldsymbol{\Theta}\}$ till convergence. The early stopper is deployed for both pretraining and retraining steps, which terminates training if model performance on validation data cannot be improved within a few epochs. The overall framework is described in Algorithm 1.

**Algorithm 1:** Structural HAM Pruning

**Input:** data $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}, \mathcal{D}_{\text{test}}$,
base dimensions $d_j (\leq C_j)$ for each field $j$,
target embedding size $s$, hyperparameters
$\mu, \epsilon > 0$;
▷ **Pretrain**:
**while** *stopping criteria not met* **do**
  obtain a batch of samples from $\mathcal{D}_{\text{train}}$ and update $\mathbb{V}, \boldsymbol{\Theta}$ regularized with SO (1) by certain optimizer;
**end**
▷ **Search**: initialize $\boldsymbol{\alpha} = \epsilon\vec{1}$;
**while** *stopping criteria not met* **do**
  obtain a batch of samples from $\mathcal{D}_{\text{val}}$ and update $\boldsymbol{\alpha}$ by Eq. (3);
  obtain a batch of samples from $\mathcal{D}_{\text{train}}$ and update $\mathbb{V}, \boldsymbol{\Theta}$ by certain optimizer;
**end**
▷**Retrain**: mask the dimensions with 0 where $\boldsymbol{\alpha} < 0$ and retrain the model until the stopping criteria are met.

## 3.4. On the Gradient Dynamics

As finding the optimal mask is essentially a combinatorial optimization problem over the set of $2^S$ possible status of on-off switches ($S$ is the number of switches), the proposed gradient-based search rule given in (3) provides an alternative approach to look through the discrete candidate sets in a continuous space. We elaborate below that the STE-based gradient in (2) is related to the Taylor approximation for the Leave-One-Out error and the penalty term drifts auxiliary variables to find a model of

the target size.

For illustration purpose, we start with the dynamics of the scalar-valued parameter $\alpha_{i,j}$, the $j$-th element of $\boldsymbol{\alpha}_i$, that controls the mask for the $j$-th column vector, $\mathbf{V}_{i,j}$, of the embedding table $\mathbf{V}_i$. Define the function $\ell(c) := \mathcal{L}_{[c\cdot\mathbf{V}_{i,j}]}$ as the value of loss function when replacing $\mathbf{V}_{i,j}$ by $c \cdot \mathbf{V}_{i,j}$ ($0 \leq c \leq 1$). By the first-order Taylor approximation, we have $\ell(1) - \ell(0) \approx \ell'(0)$ and $\ell(0) - \ell(1) \approx -\ell'(1)$. Moreover, it is worth noting that the STE-based gradient in (2) with regards to $\alpha_{i,j}$ is exactly $\ell'(\mathbf{1}_{\alpha_{i,j}>0})$, i.e.,

$$\frac{\partial \mathcal{L}_{\text{batch}}}{\partial \mathbf{1}_{\alpha_{i,j}>0}} = \ell'(\mathbf{1}_{\alpha_{i,j}>0}) \approx \ell(1) - \ell(0) = \mathcal{L}_{[\mathbf{V}_{i,j}]} - \mathcal{L}_{[0\cdot\mathbf{V}_{i,j}]}. \tag{4}$$

In other words, the proposed gradient calculates the importance (with batches of data) of the embedding component $\mathbf{V}_{i,j}$ using the first-order Taylor approximation. The above heuristics are also mentioned in [36, 37, 25].

We now consider the dynamics of all the auxiliary parameters $\alpha_{i,j}$'s provided by (3). As illustrated above, the term $\nabla_{(\mathbb{1}_{\boldsymbol{\alpha}_t>0})}\mathcal{L}_{\text{data}}$ measures the importance of each embedding component by approximating the difference between the loss value with un-masked embeddings and the loss value with a mask on. Furthermore, the sign of the penalty term $\mu$ in (3) is determined by the comparison between the number of un-masked components and the target size $s$. As an analogue, the dynamics of auxiliary parameters can be viewed as a particle system in a neighborhood of 0 on real line, in which the particles are initialized at the same point $\epsilon > 0$ (i.e., the model starts with all embeddings un-masked) and the velocity of each particle is roughly $\eta \cdot (\mathcal{L}_{[0\cdot\mathbf{V}_{i,j}]} - \mathcal{L}_{[\mathbf{V}_{i,j}]})$ by the approximation in (4). In addition, an external force is introduced by the penalty term, $\mu\big|\|\mathbb{1}_{\boldsymbol{\alpha}>0}\|_1 - s\big|$, which can be interpreted as the wind flow with its velocity being $-\mu < 0$ when the number of positive particles is larger than $s$ and being $\mu > 0$ otherwise. As a result, when the number of positive particles exceeds $s$, those particles with smaller $\mathcal{L}_{[0\cdot\mathbf{V}_{i,j}]} - \mathcal{L}_{[\mathbf{V}_{i,j}]}$ tend to be negative, and vice versa; see Figure 2 for further illustration of the proposed algorithm.

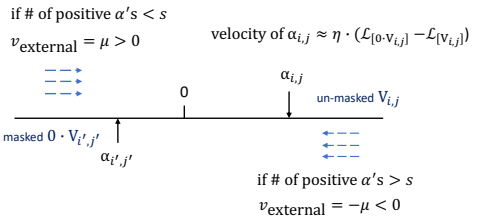


**Figure 2:** An intuitive viewpoint of the gradient dynamics.

# 4. Experiments

To validate the performance of our proposed framework, we conduct extensive experiments on three real-world recommendation datasets. Through the experiments, we seek to answer the following three questions: (i) How does our proposed method, HAM, perform compared with other auxiliary mask pruning (AMP) methods? (ii) How does our proposed framework perform compared with the existing field-wise embedding size search methods in the literature in terms of prediction ability, and memory consumption? (iii) How does the soft orthogonality regularity boost the performance of our proposed framework? In the following, we will first introduce the experimental setups including datasets, baselines, and implementation details, and then present results as well as discussions.

## 4.1. Datasets and Data Preparation.

We use three benchmark datasets in our experiments: (i) **MovieLens-1M**: This dataset contains users' ratings (1-5) on movies. We treat samples with ratings greater than 3 as positive samples and samples with ratings below 3 as negative samples. Neutral samples with a rating equal to 3 are dropped; (ii) **Criteo**: This dataset has 45 million users' clicking records on displayed ads. It contains 26 categorical feature fields and 13 numerical feature fields; (iii) **Avazu**: This dataset has 40 million clicking records on displayed mobile ads. It has 22 feature fields spanning from user/device features to ad attributes. We follow the common approach to remove the infrequent features and discretize numerical values. First, we remove the infrequent feature values and treat them as a single value "*unknown*", where the threshold is set to $\{10, 4\}$ for Criteo, and Avazu respectively. Second, we convert all numerical values in into categorical values by transforming a value $z$ to $\text{int}(\log(z)^2)$ if $\text{int}(z) > 2$ and to $\text{int}(z) - 2$ otherwise, which is proposed by the winner of Criteo Competition [2]. Third, we randomly split all training samples into 80% for training, 10% for validation, and 10% for testing.

## 4.2. Baselines

We compare our proposed pruning method HAM with several baseline methods below.

- **Uniform**: The method assigns a uniform embedding size for all feature fields;
- **Auxiliary Mask Pruning**: We compare our proposed method with other common approaches that apply an

[2] https://www.csie.ntu.edu.tw/~r01922136/kaggle-2014-criteo.pdf

auxiliary mask to the embeddings. To be specific, we select three representative types of the auxiliary mask listed in Table 1 and apply the same one-shot NAS framework as described in Algorithm 1:

(i) SAM: the embeddings are masked directly by auxiliary weights $0 \leq \alpha \leq 1$, which is equivalent to DNIS [1].

(ii) SAM-GS: the auxiliary mask is obtained by the Gumbel-sigmoid function of auxiliary parameters $\alpha \in (0, 1)$ and $u \sim$ Uniform(0,1) be;ow

$$\text{Sigmoid}[(\log(\frac{\alpha}{1-\alpha}) + \log(\frac{u}{1-u}))/\lambda].$$

(iii) HAM-p: the mask is generated by Bernoulli random variables with parameters $p$'s when forwarding the network. The gradient for $p$'s is calculated by STE.

In the retraining stage, we keep the embedding components with top-$s$ auxiliary weights for fair comparisons.

- **AutoDim** [2]: the state-of-the-art NAS-based method in the literature, which outperforms a list of search methods [8, 10, 5, 7]; see Section 3.4 in [2].

## 4.3. Implementation Details

**Base Model architecture.** We adopt four representative CTR-prediction models in the literature: FM [26], DeepFM [27], AutoInt [28], DCN-V2 [31], as our base models to compare their performance. For FM, DeepFM, and AutoInt, we add an extra embedding vector layer between the feature interaction layer and the embedding layer as proposed in [10]. Linear transformations (without bias) are applied to mixed-dimension embeddings so that the transformed embedding vectors are of the same size, which is set as 16 in our experiments. This is not required for DCN-V2 due to the bit-wise crossing mechanism. In the pretraining stage, the base dimension $d_j$ for each feature field $j$ is chosen as $\min(16, C_j)$ where $C_j$ is the field dimension.

**Optimization.** We follow the common setup in the literature to employ Adam optimizer with the learning rate of $10^{-3}$ to optimize model parameters in all three stages and use SGD optimizer to optimize auxiliary weights in the search stage. To fairly compare different AMP methods, the number of search epochs is fixed as 10, and the learning rates of auxiliary parameters are chosen from the search grid $\{1, 10^{-1}, 10^{-2}, 10^{-3}\}$, and the temperature of the sigmoid function is chosen from the search grid $\{1, 10^{-1}, 10^{-2}\}$. To obtain the best results of each method, we pick $10^{-2}$ as the learning rate of SGD for SAM, SAM-p, and HAM-p, $10^{-3}$ for our method HAM. In HAM, the initial value of auxiliary weights $\epsilon = 0.01$, and the hyperparameter $\mu = 5 \times 10^{-5}$. For the orthogonal regularity, we employ $\lambda\mathcal{R}(\mathbb{V})$ with $\lambda = 10^{-3}$ for training models on MovieLens-1M, and use the pair-wise cosine similarities $\lambda\mathcal{R}(\bar{\mathbb{V}})$ with $\lambda = 10^{-6}$ on Avazu. We use PyTorch to implement our method and train it with mini-batch size 2048 on a single 16G-Memory Nvidia Tesla V100.

## 4.4. Performance Comparison

We next present detailed comparisons with other methods. We adopt AUC (Area Under the ROC Curve) on test datasets to measure the performance of models and the number of embedding parameters to measure memory usage.

**Comparison with other AMP Methods.** We first compare our proposed AMP methods to other common AMP methods listed in Table 1 on MovieLens-1M. To fairly compare the performance, we not only consider comparing the test AUC under the same target total embedding size $s$ but also take the number of model parameters into account. In Figure 3, we report the average values of test AUC under the same total embedding size and the best results among 10 independent runs in terms of AUC and plot *Test AUC - # Parameter curve* on MovieLens-1M. For each method, three measurements are provided from left to right with the target total embedding size $s = 14, 28, 42$ respectively. We drop the *Test AUC - # Parameter curve* of the method with the worst performance at the bottom. We observe that: **(a)** fixing the target total embedding size, we can tell that our method HAM outperforms all other methods on four base models with regard to the value of test AUC. As illustrated in *Test AUC - # Parameter curves*, with the same budget of total embedding size, HAM tends to assign larger sizes to informative feature fields with larger vocabulary sizes compared with other methods. This should be attributed to the gradient dynamics described in Section 3.4 which helps to identify the important embedding components. As a result, HAM obtains models with higher AUC under the same total embedding size; **(b)** it is interesting to observe that SAM also assigns larger embedding sizes to relatively un-informative features compared to other methods with the same total embedding size, which verify the findings in [12] that the magnitudes of auxiliary weights may not indicate the importance; **(c)** HAM exhibits its superiority, not only on the vector-wise feature crossing models (FM, DeepFM, AutoInt), but also on the bit-wise feature crossing model (DCN-V2) where Uniform serves as a strong baseline. The performance of HAM should be credited to the hard selection caused by indicator functions. With the deterministic binary mask, the masked model evaluated in the search stage is equivalent to the selected model in the retraining stage, as illustrated in Table 1. As a result, we conclude that HAM exhibits stable performance on all four base models

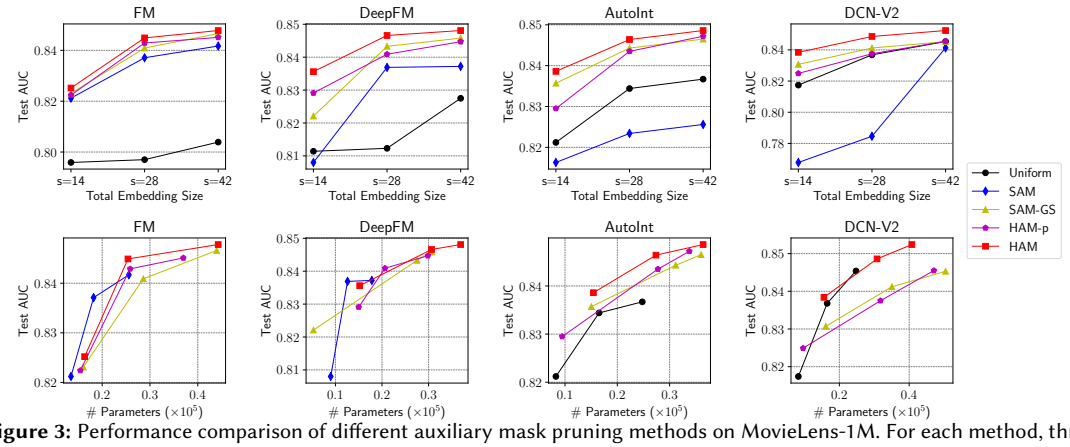


**Figure 3:** Performance comparison of different auxiliary mask pruning methods on MovieLens-1M. For each method, three measurements from left to right are reported with the target total embedding size $s = 14, 28, 42$ respectively.

while SAM, SAM-GS, and HAM-p suffer from instability and suboptimality in some circumstances.

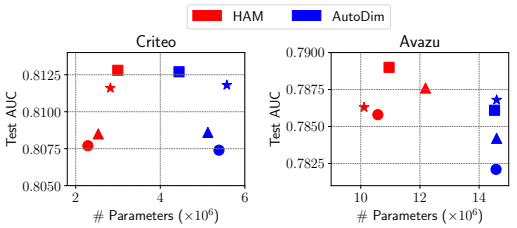


**Figure 4:** Performance comparison between HAM and AutoDim. ○ - FM; △ - DeepFM; ⋆ - AutoInt; □ - DCN-V2.

**Comparison with AutoDim.** We also compare our method with state-of-art NAS-based method AutoDim with $\{2, 4, 8\}$ as the candidate embedding size on Criteo and Avazu datasets in Figure 4. In our method, the total embedding size $s$ is set to be $90, 50$ for Criteo and Avazu respectively. Our method outperforms AutoDim by finding models with comparably higher AUC and smaller sizes. In addition, from computational perspectives, our approach HAM has several advantages over AutoDim: **(a)** HAM has a larger search space due to the structural masking mechanism. For each feature, its candidate embedding size range from 0 to its base dimension $d_j$. On the contrary, AutoDim requires manually prespecifying a set of candidate embedding sizes for each feature; **(b)** HAM is more computationally efficient. We emphasize that AutoDim introduces extra space and time complexity by the operations of lifting, batch normalization, and aggregation for each candidate size while HAM only requires extra element-wise product between the binary mask and embedding vectors. Moreover, HAM can output multiple models of different total embedding sizes given the same pre-trained model, whereas AutoDim requires pretraining the model once more if changing the candidate embedding size.

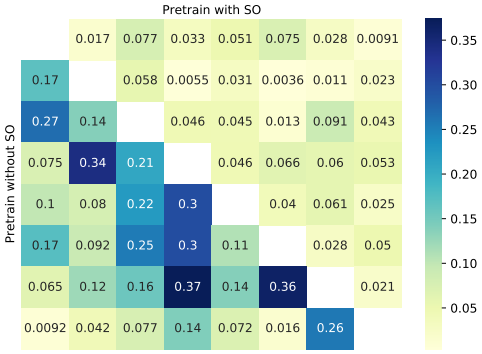


**Figure 5:** Cosine similarities between column vectors from the embedding table of *genre* when pretraining DCN-V2 on MovieLens-1M with and without SO.

**Table 2**
The test AUC gain by Algorithm 1 with SO under different target embedding sizes on MovieLens-1M and Avazu

| Data | Size $s$ | Test AUC | | | |
|---|---|---|---|---|---|
| | | FM | DeepFM | AutoInt | DCN-V2 |
| ML-1M | 14 | +.0006 | **+.0029** | +.0013 | +.0012 |
| | 28 | **+.0018** | **+.0034** | **+.0018** | **+.0027** |
| | 42 | **+.0036** | **+.0033** | +.0009 | **+.0037** |
| Avazu | 44 | **+.0034** | **+.0039** | **+.0026** | **+.0031** |
| | 66 | **+.0039** | **+.0044** | **+.0020** | **+.0033** |

*The results are obtained by averaging 10 runs and the bold font indicates statistically significant under two-sided t-tests ($p < 0.05$).*

**On the Orthogonal Regularity.** We further analyze the effect of the orthogonal regularity proposed in our framework. In Figure 5, we visualize the cosine similarities between column vectors from the embedding of *genre* when pretraining DCN-V2 on `MovieLens-1M` with and without `SO` where the embedding size is set as 8. It is clear to see that `SO` helps to reduce the correlations within those embedding column vectors. Moreover, we compare the performance of models selected by our proposed Algorithm 1 with the one without `SO` in Table 2. Statistical significant gains in test AUC can be observed for `MovieLens-1M` and `Avazu` on all base models while the training time per epoch with `SO` only increases by about 1.6 times compared to the time without `SO`.

### 4.5. Discussions and Future Work

**Multi-stage Search.** As the initial base dimension for each feature fields it tunable, we observe a gain, $\sim$0.3%, in test AUC from preliminary experiments on `MovieLens-1M` by searching via `HAM` with a smaller initial size $= 8$. This observation confirms the claim made in recent works [38] that enlarging search space is unbeneficial or even detrimental to existing NAS methods. To address this issue, our method can also be adapted to design a multi-stage search strategy with a sufficiently larger initial size and stage-wisely prune embeddings with decreasing target embedding size $s$.

**Comparison with PEP.** To justify the advantages of our proposed framework, we also discuss the most recent Plug-in Embedding Pruning (PEP) [14] method. As introduced before, PEP is an unstructured pruning method that retains a large number of zero parameters and requires the technique of sparse matrix storage. On the contrary, our method can not only prune the embedding columns explicitly but also reduce the parameters in the dense layers while PEP cannot. For example, `HAM` ($s = 28$) can obtain a lighter DCN-V2 model on `MovienLens-1M` with only $\sim$8% number of parameters (3,281/40,241) in the dense layers comparing to the model with a uniform embedding size 16.

**Comparison with SSEDS.** During the revisions of this paper, we noticed that a concurrent work [39] also proposes to prune embeddings column-wisely as well as row-wisely using indicator functions. Our work is different from theirs in two aspects: (i) we focus on field-wise search by pruning nearly orthogonal embedding column vectors; (ii) we use a gradient-descent-based method in the search stage to solve the bilevel problem on the training and validation set; the gradient dynamics enable us to re-activate (un-mask) masked embeddings while SSEDS directly mask all components with small gradient magnitudes. Comparing our method with SSEDS and incorporating our method into SSEDS is our future work.

## 5. Conclusions

In this paper, we propose a general auxiliary masking pruning framework to search the embedding sizes for different feature fields adaptively in today's recommender systems. The proposed method leverages the indicator function to search candidate models exactly, which is efficient and can be easily applied to various models. Extensive experiments demonstrate it can effectively remove redundant embedding dimensions without great performance loss.

## Acknowledgments

The authors are grateful to anonymous reviewers for their constructive comments that greatly improved the presentation of this paper. T. Xiao thanks his Ph.D. advisor Dr. Krishnakumar Balasubramanian for supports and helpful discussions during the internship at ByteDance.

## References

[1] W. Cheng, Y. Shen, L. Huang, Differentiable neural input search for recommender systems, arXiv preprint arXiv:2006.04466 (2020).

[2] X. Zhao, H. Liu, H. Liu, J. Tang, W. Guo, J. Shi, S. Wang, H. Gao, B. Long, Memory-efficient embedding for recommendations, arXiv preprint arXiv:2006.14827 (2020).

[3] S. Zhang, L. Yao, A. Sun, Y. Tay, Deep learning based recommender system: A survey and new perspectives, ACM Computing Surveys (CSUR) 52 (2019) 1–38.

[4] P. Covington, J. Adams, E. Sargin, Deep neural networks for youtube recommendations, in: Proceedings of the 10th ACM conference on recommender systems, 2016, pp. 191–198.

[5] X. Zhao, C. Wang, M. Chen, X. Zheng, X. Liu, J. Tang, Autoemb: Automated embedding dimensionality search in streaming recommendations, arXiv preprint arXiv:2002.11252 (2020).

[6] H. Liu, X. Zhao, C. Wang, X. Liu, J. Tang, Automated embedding size search in deep recommender systems, in: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, 2020, pp. 2307–2316.

[7] M. R. Joglekar, C. Li, M. Chen, T. Xu, X. Wang, J. K. Adams, P. Khaitan, J. Liu, Q. V. Le, Neural input search for large scale recommendation models, in: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2020, pp. 2387–2397.

[8] T. Chen, L. Li, Y. Sun, Differentiable product quantization for end-to-end embedding compression,

in: International Conference on Machine Learning, PMLR, 2020, pp. 1617–1626.

[9] W.-C. Kang, D. Z. Cheng, T. Chen, X. Yi, D. Lin, L. Hong, E. H. Chi, Learning multi-granular quantized embeddings for large-vocab categorical features in recommender systems, in: Companion Proceedings of the Web Conference 2020, 2020, pp. 562–566.

[10] A. A. Ginart, M. Naumov, D. Mudigere, J. Yang, J. Zou, Mixed dimension embeddings with application to memory-efficient recommendation systems, in: 2021 IEEE International Symposium on Information Theory (ISIT), IEEE, 2021, pp. 2786–2791.

[11] H. Liu, K. Simonyan, Y. Yang, Darts: Differentiable architecture search, in: International Conference on Learning Representations, 2018.

[12] R. Wang, M. Cheng, X. Chen, X. Tang, C.-J. Hsieh, Rethinking architecture selection in differentiable nas, arXiv preprint arXiv:2108.04392 (2021).

[13] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, Q. Le, Understanding and simplifying one-shot architecture search, in: International Conference on Machine Learning, PMLR, 2018, pp. 550–559.

[14] S. Liu, C. Gao, Y. Chen, D. Jin, Y. Li, Learnable embedding sizes for recommender systems, in: International Conference on Learning Representations, 2020.

[15] D. Blalock, J. J. G. Ortiz, J. Frankle, J. Guttag, What is the state of neural network pruning?, arXiv preprint arXiv:2003.03033 (2020).

[16] P. Savarese, H. Silva, M. Maire, Winning the lottery with continuous sparsification, in: Advances in Neural Information Processing Systems, volume 33, Curran Associates, Inc., 2020, pp. 11380–11390.

[17] C. Louizos, M. Welling, D. P. Kingma, Learning sparse neural networks through l_0 regularization, in: International Conference on Learning Representations, 2018.

[18] S. Xie, H. Zheng, C. Liu, L. Lin, Snas: stochastic neural architecture search, in: International Conference on Learning Representations, 2018.

[19] Y. Guo, A. Yao, Y. Chen, Dynamic network surgery for efficient dnns, in: Proceedings of the 30th International Conference on Neural Information Processing Systems, 2016, pp. 1387–1395.

[20] X. Xiao, Z. Wang, Autoprune: Automatic network pruning by regularizing auxiliary parameters, Advances in Neural Information Processing Systems 32 (NeurIPS 2019) 32 (2019).

[21] X. Zhou, W. Zhang, H. Xu, T. Zhang, Effective sparsification of neural networks with global sparsity constraint, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 3599–3608.

[22] Y. Bengio, N. Léonard, A. Courville, Estimating or propagating gradients through stochastic neurons for conditional computation, arXiv preprint arXiv:1308.3432 (2013).

[23] E. Jang, S. Gu, B. Poole, Categorical reparameterization with gumbel-softmax, in: International Conference on Learning Representations, 2016.

[24] S. Srinivas, A. Subramanya, R. Venkatesh Babu, Training sparse neural networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition workshops, 2017, pp. 138–145.

[25] M. Ye, D. Choudhary, J. Yu, E. Wen, Z. Chen, J. Yang, J. Park, Q. Liu, A. Kejariwal, Adaptive dense-to-sparse paradigm for pruning online recommendation system with non-stationary data, arXiv preprint arXiv:2010.08655 (2020).

[26] S. Rendle, Factorization machines, in: 2010 IEEE International conference on data mining, IEEE, 2010, pp. 995–1000.

[27] H. Guo, R. Tang, Y. Ye, Z. Li, X. He, Deepfm: a factorization-machine based neural network for ctr prediction, in: Proceedings of the 26th International Joint Conference on Artificial Intelligence, 2017, pp. 1725–1731.

[28] W. Song, C. Shi, Z. Xiao, Z. Duan, Y. Xu, M. Zhang, J. Tang, Autoint: Automatic feature interaction learning via self-attentive neural networks, in: Proceedings of the 28th ACM International Conference on Information and Knowledge Management, 2019, pp. 1161–1170.

[29] R. Wang, B. Fu, G. Fu, M. Wang, Deep & cross network for ad click predictions, in: Proceedings of the ADKDD'17, 2017, pp. 1–7.

[30] X. He, T.-S. Chua, Neural factorization machines for sparse predictive analytics, in: Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval, 2017, pp. 355–364.

[31] R. Wang, R. Shivanna, D. Cheng, S. Jain, D. Lin, L. Hong, E. Chi, Dcn v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems, in: Proceedings of the Web Conference 2021, 2021, pp. 1785–1797.

[32] N. Bansal, X. Chen, Z. Wang, Can we gain more from orthogonality regularizations in training deep networks?, Advances in Neural Information Processing Systems 31 (2018) 4261–4271.

[33] P. Rodríguez, J. Gonzalez, G. Cucurull, J. M. Gonfaus, X. Roca, Regularizing cnns with locally constrained decorrelations, arXiv preprint arXiv:1611.01967 (2016).

[34] S. Yan, Y. Zheng, W. Ao, X. Zeng, M. Zhang, Does unsupervised architecture representation learning help neural architecture search?, Advances in Neural Information Processing Systems 33 (2020) 12486–12498.

[35] H. Hazimeh, Z. Zhao, A. Chowdhery, M. Sathiamoorthy, Y. Chen, R. Mazumder, L. Hong, E. H. Chi, Dselect-k: Differentiable selection in the mixture of experts with applications to multi-task learning, arXiv preprint arXiv:2106.03760 (2021).

[36] P. Molchanov, S. Tyree, T. Karras, T. Aila, J. Kautz, Pruning convolutional neural networks for resource efficient inference, arXiv preprint arXiv:1611.06440 (2016).

[37] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, J. Kautz, Importance estimation for neural network pruning, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 11264–11272.

[38] Y. Ci, C. Lin, M. Sun, B. Chen, H. Zhang, W. Ouyang, Evolving search space for neural architecture search, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 6659–6669.

[39] L. Qu, Y. Ye, N. Tang, L. Zhang, Y. Shi, H. Yin, Single-shot embedding dimension search in recommender system, arXiv preprint arXiv:2204.03281 (2022).