ABSTRACT

Click-through rate (CTR) prediction model usually consists of three components: embedding table, feature interaction layer, and classifier. Learning embedding table plays a fundamental role in CTR prediction from the view of the model performance and memory usage. The embedding table is a two-dimensional tensor, with its axes indicating the number of feature values and the embedding dimension, respectively. To learn an efficient and effective embedding table, recent works either assign various embedding dimensions for feature fields and reduce the number of embeddings respectively or mask the embedding table parameters. However, all these existing works cannot get an optimal embedding table. On the one hand, various embedding dimensions still require a large amount of memory due to the vast number of features in the dataset. On the other hand, decreasing the number of embeddings usually suffers from performance degradation, which is intolerable in CTR prediction. Finally, pruning embedding parameters will lead to a sparse embedding table, which is hard to be deployed. To this end, we propose an optimal embedding table learning framework OptEmbed, which provides a practical and general method to find an optimal embedding table for various base CTR models. Specifically, we propose pruning the redundant embeddings regarding corresponding features’ importance by learnable pruning thresholds. Furthermore, we consider assigning various embedding dimensions as one single candidate architecture. To efficiently search the optimal embedding dimensions, we design a uniform embedding dimension sampling scheme to equally train all candidate architectures, meaning architecture-related parameters and learnable thresholds are trained simultaneously in one supernet. We then propose an evolution search method based on the supernet to find the optimal embedding dimensions for each field. Experiments on public datasets show that OptEmbed can learn a compact embedding table which can further improve the model performance.

CCS CONCEPTS
• Information systems → Recommender systems; Online advertising.

KEYWORDS
CTR Prediction, Neural Architecture Search, Recommendation

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:
Fuyuan Lyu, Xing Tang, Hong Zhu, Huifeng Guo, Yingxue Zhang, Ruiming Tang, and Xue Liu. 2022. OptEmbed: Learning Optimal Embedding Table for Click-through Rate Prediction. In Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM ’22), October 17–21, 2022, Atlanta, GA, USA. ACM, Atlanta, GA, USA, 11 pages. https://doi.org/10.1145/3511808.3557411

1 INTRODUCTION

Click-through rate (CTR) prediction has been a critical task in real-world commercial recommender systems and online advertising systems [3, 26]. It aims to predict the probability of a certain user clicking a recommended item (e.g. movie, advertisement) [12, 29, 33, 34]. General CTR prediction model architecture consists of embedding table, interaction layer, and classifier as illustrated in Fig. 1 [12, 33, 34, 36]. The typical inputs of CTR models consist of many categorical features. We term the values of these categorical features as feature values, which are organized as feature fields. For example, a feature field gender contains three feature values, male, female and unknown. These predictive models use the embedding table to map the categorical feature values into real-valued dense vectors. Then these embeddings are fed into the feature interaction layer, such as factorization machine [29], cross network [34], self-attention layer [33]. The final classifier aggregates the representation vector to make the prediction.

In the general CTR prediction model architecture, the embedding table dominates the number of parameters and plays a fundamental

Fuyuan Lyu†
McGill University
Montreal, Canada
fuyuan.lyu@mail.mcgill.ca

Xing Tang†
xing.tang@huawei.com
Huawei Noah’s Ark Lab
Shenzhen, China

Hong Zhu
Huifeng Guo
zhuhong8@huawei.com
huifeng.guo@huawei.com
Huawei Noah’s Ark Lab
Shenzhen, China

Yingxue Zhang
Huawei Noah’s Ark Lab
Montreal, Canada
yingxue.zhang@huawei.com

Ruiming Tang†
Huawei Noah’s Ark Lab
Shenzhen, China
tangruiming@huawei.com

Xue Liu
McGill University
Montreal, Canada
xueliu@cs.mcgill.ca

† Both authors contributed equally to this research.
Corresponding authors

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:
Fuyuan Lyu, Xing Tang, Hong Zhu, Huifeng Guo, Yingxue Zhang, Ruiming Tang, and Xue Liu. 2022. OptEmbed: Learning Optimal Embedding Table for Click-through Rate Prediction. In Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM ’22), October 17–21, 2022, Atlanta, GA, USA. ACM, Atlanta, GA, USA, 11 pages. https://doi.org/10.1145/3511808.3557411

1 INTRODUCTION

Click-through rate (CTR) prediction has been a critical task in real-world commercial recommender systems and online advertising systems [3, 26]. It aims to predict the probability of a certain user clicking a recommended item (e.g. movie, advertisement) [12, 29, 33, 34]. General CTR prediction model architecture consists of embedding table, interaction layer, and classifier as illustrated in Fig. 1 [12, 33, 34, 36]. The typical inputs of CTR models consist of many categorical features. We term the values of these categorical features as feature values, which are organized as feature fields. For example, a feature field gender contains three feature values, male, female and unknown. These predictive models use the embedding table to map the categorical feature values into real-valued dense vectors. Then these embeddings are fed into the feature interaction layer, such as factorization machine [29], cross network [34], self-attention layer [33]. The final classifier aggregates the representation vector to make the prediction.

In the general CTR prediction model architecture, the embedding table dominates the number of parameters and plays a fundamental

Fuyuan Lyu†
McGill University
Montreal, Canada
fuyuan.lyu@mail.mcgill.ca

Xing Tang†
xing.tang@huawei.com
Huawei Noah’s Ark Lab
Shenzhen, China

Hong Zhu
Huifeng Guo
zhuhong8@huawei.com
huifeng.guo@huawei.com
Huawei Noah’s Ark Lab
Shenzhen, China

Yingxue Zhang
Huawei Noah’s Ark Lab
Montreal, Canada
yingxue.zhang@huawei.com

Ruiming Tang†
Huawei Noah’s Ark Lab
Shenzhen, China
tangruiming@huawei.com

Xue Liu
McGill University
Montreal, Canada
xueliu@cs.mcgill.ca

† Both authors contributed equally to this research.
Corresponding authors

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:
Fuyuan Lyu, Xing Tang, Hong Zhu, Huifeng Guo, Yingxue Zhang, Ruiming Tang, and Xue Liu. 2022. OptEmbed: Learning Optimal Embedding Table for Click-through Rate Prediction. In Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM ’22), October 17–21, 2022, Atlanta, GA, USA. ACM, Atlanta, GA, USA, 11 pages. https://doi.org/10.1145/3511808.3557411

1 INTRODUCTION

Click-through rate (CTR) prediction has been a critical task in real-world commercial recommender systems and online advertising systems [3, 26]. It aims to predict the probability of a certain user clicking a recommended item (e.g. movie, advertisement) [12, 29, 33, 34]. General CTR prediction model architecture consists of embedding table, interaction layer, and classifier as illustrated in Fig. 1 [12, 33, 34, 36]. The typical inputs of CTR models consist of many categorical features. We term the values of these categorical features as feature values, which are organized as feature fields. For example, a feature field gender contains three feature values, male, female and unknown. These predictive models use the embedding table to map the categorical feature values into real-valued dense vectors. Then these embeddings are fed into the feature interaction layer, such as factorization machine [29], cross network [34], self-attention layer [33]. The final classifier aggregates the representation vector to make the prediction.

In the general CTR prediction model architecture, the embedding table dominates the number of parameters and plays a fundamental
role in prediction performance. Therefore, it is critical to obtain optimal embedding tables that reduce the model size and improve performance [18, 32]. The embedding table is a two-dimensional tensor \( E \in \mathbb{R}^{F \times D} \), which maps each feature value to an indexed row. The first dimension size \( |F| \) thus equals the total number of feature values, and the second one \( D \) is the embedding dimension. The memory cost of the embedding table is \( O(|F|D) \), where \( f \) mainly decides the memory usage given \( |f| \gg D \). When all possible feature values are fed into the model [12, 15], \( |f| \) becomes a vast number (up to millions on web-scale applications). This leads to a large number of embeddings, which contributes to the primary memory bottleneck within both training and inference [11]. However, the redundant embedding not only necessitates additional memory cost but is also detrimental to the model performance [35]. Therefore, the first requirement for an optimal embedding table is to distinguish the redundant embeddings and zero them out before they are fed into the following layers, as shown in Fig. 1. The embedding dimension \( D \) is mostly fixed across all the feature values. Previous works [18, 43] point out that the over-parameterizing features with smaller feature cardinality may induce overfitting, and features with larger cardinality need larger dimensions to convey fruitful information. Hence, the second requirement for an optimal embedding table is to assign various embedding dimensions to feature values as flexible embedding shown in Fig. 1. An alternative to optimize the embedding table is to mask the embedding parameters directly [23, 27]. It makes embedding dimension \( D \geq 0 \) with discontinuous parameters, illustrated as sparse embedding in Fig. 1. Nevertheless, the sparse embedding table requires storing extra structural information and additional computation cost in the inference stage, which is not suitable for hardware in practical [6]. The final requirement is to optimize the embedding table without storing additional structural information on hardware requirements. Given the above requirements, it is highly desired to prune redundant embeddings and search embedding dimensions in a unifying way.

Previous work on optimizing the embedding table either treats embedding reduction and dimensions search separately or generates sparse embedding. To reduce the number of embeddings, one natural approach is to design a hash function, which maps the categorical features to the embedding table index [37, 38, 41]. Since the embedding table index size is far less than the number of feature values, this approach optimizes memory usage. However, blindly mapping different feature values into the same embeddings without distinguishing the redundant embeddings may lead to performance degradation, which is intolerable in CTR prediction. On the other hand, AutoField [35] utilizes the differential architecture search [21] method to prune redundant feature fields. But this method may prune some informative feature values while preserving some redundant ones since the feature field is not fine-grained enough to generate an optimal embedding. To search for flexible embedding dimension, AutoDim [43] also utilize the differential architecture search [21] method. However, this method cannot get an optimal embedding table, as it does not remove redundant features. Moreover, some research optimizes the embedding table in a unifying way based on embedding pruning [23, 27, 31]. These methods identify and mask redundant values in embeddings, where embeddings are pruned when their dimensions are equal to zero. However, these methods result in a sparse embedding table, which poses challenges when fitting into modern computation units.

In this paper, we propose a framework to address two main challenges and learn an optimal embedding table (OptEmbed) that satisfy all three requirements. First, for the problem of how to prune the redundant embeddings and search feature fields embedding dimensions in a unifying way, we transform it into the problem of identifying the importance of each feature value. Besides, searching field-wise embedding dimensions can be formulated as an architecture search problem. To this end, inspired by structural pruning and network architecture search [2, 8], we introduce learnable pruning thresholds to distinguish informative embeddings and to allocate the dimensions to those embeddings in an automated and data-driven manner. Specifically, we mask redundant embeddings adaptively with thresholds. Meanwhile, we design a uniform embedding dimension sampling scheme to train a supernet with the learnable thresholds. Then, we conduct an evolutionary search based on the supernet with informative embeddings to assign optimal field-wise dimensions. To address the second challenge of how to optimize the embedding table efficiently, we reparameterize the problem with a threshold vector, which makes the original problem differentiable and only needs a few preallocate memory [7]. Moreover, the search space of embedding dimensions is also too huge to explore [43]. Therefore, we design a one-shot embedding dimension search method to save search time by decoupling parameter training and dimension search based on the supernet mentioned above. The experimental results on three public datasets demonstrate the efficiency and effectiveness of our proposed framework OptEmbed. We summarize our major contributions as below:

- This paper firstly proposes the requirements for an optimal embedding table: no redundant embedding, embedding dimension flexible and hardware friendly. We propose a novel optimization method called OptEmbed, which improves model performance and reduces memory usage based on these requirements.
- The proposed OptEmbed optimizes the embedding table in a unifying way. It can efficiently train a supernet with informative feature values and the embedding parameters simultaneously. Moreover, we design a one-shot embedding dimension search method based on the supernet, which produces the optimal embedding table without sparse embedding.
- The extensive experiments are conducted on three public datasets. The experimental results demonstrate the effectiveness and efficiency of the proposed framework.
the experiments. In Section 5, we briefly introduce related works. Finally, we conclude this work in Section 6.

2 PROBLEM DEFINITION

In this section, we formulate how the CTR prediction model output the prediction result with the concatenation of multiple features and define the requirements for an optimal embedding table.

2.1 CTR Prediction

We represent the raw inputs as the raw feature vector that concatenates \( n \) feature fields \( \mathbf{x} = [x_{1(1)}, x_{2(2)}, \cdots, x_{n(n)}] \). Usually, \( x_{i(j)} \) is a one-hot representation, which is very sparse and high-dimensional. For example, the feature field gender has three unique feature values, male, female, and unknown, then they can be represented by three one-hot vectors \([1, 0, 0],[0, 1, 0] \) and \([0, 0, 1] \), respectively. Before raw feature vectors are fed into the feature interaction layer, we usually employ embedding table to convert them into low dimensional and dense real-value vectors. This can be formulated as \( e_{i(j)} = \mathbf{x} \times x_{i(j)}, 1 \leq i \leq n \), where \( \mathbf{E} \in \mathbb{R}^{ DF \times \mathcal{D}} \) is the embedding table, \( |\mathcal{D}| \) is the number of feature values and \( \mathcal{D} \) is the size of embedding. Then embeddings are stacked together as an embedding vector \( e = [e_{1(1)}, e_{2(2)}, \cdots, e_{n(n)}] \).

Following learnable embedding table, the feature interaction layer will be performed based on \( e \) in mainstream CTR models. There are several types of feature interaction in previous study, e.g. inner product [12]. As discussed in previous work [25], feature interaction can be defined as based on embeddings:

\[
v^p = o^p(o^{p-1}(\cdots(o^{(1)}(e)\cdots)))
\]

where \( o \) can be a single layer perceptron or cross layer[34]. The feature interaction can be aggregated together:

\[
y = \sigma(w^T(x^{(1)} \oplus x^{(2)} \oplus \cdots \oplus x^{(n)}) + b) = \mathcal{F}(\mathbf{x} \times \mathbf{W}),
\]

where symbol \( \oplus \) denotes the concatenation operation, \( v^{(k)} \) is the output of feature interaction, and \( \mathbf{W} \) is network parameters except for embedding table. The cross entropy loss (i.e. log-loss) is adopted for training the model:

\[
CE(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}).
\]

We summarize the final CTR prediction problem as follows:

\[
\min_{\mathbf{E},\mathbf{W}} \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}} CE(y, \mathcal{F}(\mathbf{x} \times \mathbf{W})).
\]

where \( y \) is the ground truth of user clicks, \( \mathcal{D} \) is the training dataset.

2.2 Optimal Embedding Table

The original embedding table \( \mathbf{E} \in \mathbb{R}^{ DF \times \mathcal{D}} \) is neither effective nor efficient [23, 35, 36, 43]. An optimal embedding table that satisfies the following requirements can greatly reduce the model size and improve performance:

Req. 1. No Redundant Embeddings: The optimal embedding table should only map informative feature values to embeddings. Feature value \( x_i \) is considered informative if the performance of the model is degraded when masking its corresponding embedding \( e_i \). Otherwise, it is deemed to be redundant.

Req. 2. Embedding Dimension Flexible: The optimal embedding table should assign various embedding dimensions, improving the performance of the predictive model the most.

Req. 3. Hardware Friendly: The optimal embedding table should be compatible with the modern parallel-processing hardware (e.g. GPU) – requiring no additional resources when training and inference.

To fulfill the three requirements above, we decompose the original single embedding table \( \mathbf{E} \) into a series of field-wise embedding tables \( \mathbf{E}^i = [e_{1(i)}, e_{2(i)}, \cdots, e_{n(i)}] \), where \( e_{i(j)} \in \mathbb{R}^{D(i) \times \mathcal{D}(i)} \). To satisfy requirement (i), we prune some embeddings related to redundant feature values, which can be formulated as \( \sum_{i=1}^n |f(i)| \leq |f| \). As to requirements (ii) and (iii), different embedding sizes for each field-wise embedding tables are allocated. In summary, an optimal embedding table can be further defined as:

\[
\min_{\mathbf{E}^i, \mathbf{W}} \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}} CE(y, \mathcal{F}(\mathbf{x} \times \mathbf{W})), \quad \mathbf{E}^i = \mathbf{E} \odot \mathbf{m}_e \odot \mathbf{m}_d.
\]

Here \( \mathbf{m}_e \odot \mathbf{m}_d \in \{0, 1\}^{DF \times \mathcal{D}} \) denotes the field-wise dimension mask, \( \mathbf{m}_e \in \{0, 1\}^{DF} \) denotes the embedding mask. \( |f| \) and \( n \) denote the feature number and field number. \( \odot \) denotes element-wise product with broadcasting. By doing so, we decompose the task of learning the optimal embedding table \( \mathbf{E}^i \) into three parts: (i) train embedding mask \( \mathbf{m}_e^* \) to preserve informative embeddings; (ii) search for field-wise dimension mask \( \mathbf{m}_d^* \) to assign various embedding size to each field; (iii) re-train the optimal embedding table \( \mathbf{E}^i \) under the constraints of (i) and (ii). The overview of OptEmbed framework is shown in Fig. 2. Both the embedding mask and dimension mask are applied to the embedding table. Some of the embeddings are removed, while others become sparse.

Below, we first illustrate how to determine embedding mask \( \mathbf{m}_e \) and field-wise dimension mask \( \mathbf{m}_d \), respectively. We then introduce the re-training stage and discuss how OptEmbed compares with other methods that optimize the architecture of embedding tables from various aspects.

3 OPTEMBED

In this section, we propose a framework called OptEmbed to learn the optimal embedding table \( \mathbf{E}^i \) defined in Section 2. We rewrite Eq. 5 into the following by introducing two masks:

\[
\min_{\mathbf{m}_e, \mathbf{m}_d, \mathbf{E}, \mathbf{W}} \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}} CE(y, \mathcal{F}(\mathbf{x} \times \mathbf{W})), \quad \mathbf{E}^i = \mathbf{E} \odot \mathbf{m}_e \odot \mathbf{m}_d.
\]
which can be updated by gradient descent. The reparameterization of the $\mathbf{m}_e$ is formulated as follows:

$$
\mathbf{m}_e = S(L_\beta(\mathbf{E}) - t),
$$

where $t$ is the field-wise threshold vector, $L_\beta$ is the $\beta$ norm of embedding in each field, $S(\cdot)$ is the activation function, which works as trainable dynamic mask. We will illustrate three parts in the following sections in details.

3.1.1 Field-wise Threshold Vector. We introduce a trainable field-wise vector $t \in \mathbb{R}^{|f|}$ to serve as pruning thresholds for embeddings in every field. We do not adopt a global threshold because corresponding features from different fields are likely to have different properties. For instance, the average frequency for the gender field is expected to be much higher than that for the ID field. Assessing the importance of features from different fields would lead to a non-robust and hard-to-train network. Meanwhile, we do not adopt a feature-wise threshold vector considering it would significantly increase the number of total parameters and make it more likely to over-fit.

3.1.2 $L_\beta$ norm. It is commonly believed that features with higher frequency tend to be more important and informative in CTR prediction [39, 41]. To measure the importance of features precisely, we empirically train a prediction model and investigate the relation between the frequency of each feature and $L_\beta$ norm of the corresponding embedding in each field. Three fields in Avazu dataset1 are randomly selected as an example; here we set the base model as FNN [42], $L_\beta$ norm as $L_1$ norm. The results are illustrated in Fig. 3. Each green dot represents one embedding in the embedding table, with its x-axis denoting the $L_1$ norm of the embedding and y-axis denoting the log frequency of the corresponding feature value. A fitting curve is also shown in blue to summarize the relationships between these two variables. As we can observe, with the increment of feature frequency, the $L_1$ norm of the corresponding embedding also grows linearly. As a result, we adopt $L_\beta$ norm of corresponding embeddings as a measurement in our framework [14, 22].

3.1.3 Unit Step Function. We introduce a unit step function $S(x)$ as the activation function to generate a binary mask. Given the field-wise threshold vector $t$ and unit step function $S(x)$, we can formally generate the embedding mask $\mathbf{m}_e$. For feature $j$ corresponding embedding $\mathbf{e}^j$, its embedding mask $\mathbf{m}_e^j$ is given by:

$$
\mathbf{m}_e^j = S(L_\beta(\mathbf{e}^j) - t^j) = \begin{cases} 
1, & L_\beta(\mathbf{e}^j) - t^j > 0 \\
0, & \text{otherwise}
\end{cases},
$$

where $L_\beta(\cdot)$ indicates the $L_\beta$ normalization function and $k_j$ maps feature $j$ to the corresponding field. With the unit step function $S(x)$, we can easily generate binary embedding mask $\mathbf{m}_e$. Then the embedding table can be formulated as

$$
\hat{\mathbf{E}} = \mathbf{E} \odot \mathbf{m}_e = \mathbf{E} \odot S(L_\beta(\mathbf{E}) - t).
$$

The prediction score will be calculated with $\hat{\mathbf{E}}$. However, because the derivative of step unit function is an impulse function, Eq.9 cannot be directly optimized. To preserve gradients and make the model trainable, we adopt the long-tail derivation estimator [22] to replace the gradient $\langle \frac{dS(x)}{dx} \rangle$ of the step unit function. The long-tail derivation estimator can be formulated as

$$
\frac{d}{dx} S(x) \approx H(x) = \begin{cases} 
2 - 4|x|, & |x| \leq 0.4 \\
0.4, & 0.4 < |x| \leq 1 \\
0, & \text{otherwise}
\end{cases}.
$$

We adopt this derivative long-tail estimator to optimize the field-wise threshold vector $t$, so that the gradient would be large when the $L_\beta$ norm and threshold value are close to each other and would be 0 when the gap between $L_\beta$ norm and threshold value are large enough. We denote the gradient of actual embedding $\hat{\mathbf{E}}$ as $\hat{\mathbf{E}}$. Notes that the gradient of embedding for updating $\mathbf{E}$ is

$$
\frac{d\mathbf{E}}{d\mathbf{E}} = \hat{\mathbf{E}} \odot \mathbf{m}_e + \hat{\mathbf{E}} \odot H(L_\beta(\mathbf{E}) - t) \odot dL_\beta(\mathbf{E}).
$$

The gradient of embedding is composed of two parts. The first part $d\hat{\mathbf{E}} \odot \mathbf{m}_e$ is the performance gradient that improves the performance. The second part $d\hat{\mathbf{E}} \odot H(L_\beta(\mathbf{E}) - t) \odot dL_\beta(\mathbf{E})$ is the structure gradient that removes redundant embeddings. As the final embedding is jointly influenced by both the performance and structure, OptEmbed can recover some embeddings. Specifically, once an embedding is accidentally removed, the performance gradient becomes zero because the embedding is zeroed-out. However, the embedding still receives the structure gradient. So the pruned embedding may be recovered again if the gap between $L_\beta$ norm and threshold are not too large (i.e. smaller than one in this case).

It is worth mentioning that the $\beta$ in $L_\beta$ norm needs to be carefully selected. With $L_\beta(\mathbf{e}) = \left( \sum |\mathbf{e}^j|^\beta \right)^{1/\beta}$ for embedding $\mathbf{e}$, we can get derivation for particular element:

$$
\frac{dL_\beta}{d\mathbf{e}} = \left( \sum |\mathbf{e}^j|^\beta \right)^{1/\beta-1} \cdot |\mathbf{e}|^\beta \cdot \mathbf{e}^{\beta-1}.
$$

\footnote{http://www.kaggle.com/c/avazu-ctr-prediction}

\hspace{1cm}

\begin{center}
Figure 2: The Overview of OptEmbed.
\end{center}

\begin{center}
Figure 3: Relationship between $L_1$-norm and frequency in selected fields of Avazu dataset.
\end{center}
Due to the first term \((\sum e_i^t)^{1/\beta - 1}\), the gradient of \(e_i\) will be influenced by all elements from embedding \(e\) unless \(\beta = 1\). Therefore we select \(\beta = 1\) for OptEmbed hereafter to get rid of the influence.

### 3.1.4 Sparse Regularization Term

To remove more redundant embeddings, higher thresholds are encouraged. To achieve this, we explicitly add an exponential regularization term \(L_s\) to the logloss that penalizes low threshold values. For the field-wise threshold \(t \in \mathbb{R}^n\), the exponential regularization term is

\[
L_s = \sum_{i=1}^{n} \exp(-t_i). \tag{13}
\]

Notice that the regularization term gradually decreases to zero as \(x\) increases. Hence, the final objective in this stage becomes

\[
\min_{m_e, \hat{E}, \hat{W}} \mathcal{L}_{CE}(D | \hat{E}, \hat{W}) + \alpha L_s, \qquad \hat{E} = E \otimes m_e. \tag{14}
\]

Here \(\alpha\) is the scaling coefficient for the sparse regularization term, controlling how many embeddings are pruned. With higher \(\alpha\), \(L_s\) tends to increase the threshold \(t\), making it easier to prune redundant embeddings. However, once \(\alpha\) becomes too large, it may accidentally remove some informative embeddings, leading to the increase of the cross-entropy loss \(\mathcal{L}_{CE}\). Therefore, our method can dynamically remove redundant embeddings, leading to a proper balance between model performance and size.

### 3.2 Embedding Dimension Search

The embedding dimension search component aims to assign various optimal dimensions for all fields. By viewing a group of field-wise dimension masks as one neural network architecture, we design an efficient neural architecture search method to search for optimal dimension masks efficiently in this section.

#### 3.2.1 One-shot NAS Problem

Because the optimal embedding table should satisfy Req. 2 and 3, the dimensionality set in our method, formed by all candidate embedding dimensions, can be formulated as \(S_e = \{1, \cdots, D - 1, D\}\). Notice that the complexity of this search space is \(O(D^n)\), which is impossible to search all the possible architectures in the search space exhaustively. On the other hand, to evaluate architecture, we need to train \(m_e\) and network parameters again, which costs a lot of computation resources. To efficiently search for the optimal embedding dimension mask, we hence reformulate the dimension search as a one-shot NAS problem [1, 13]:

\[
\mathbf{m}_d^* = \arg\min_{m_d \in S_d} \mathcal{L}_{CE}(D_{real} | (\hat{E}_d \odot m_d, \hat{W}_s)),
\]

\[
s.t. \{\hat{E}_d, \hat{W}_s\} = \arg\min_{(E_d, W_s) \in \Omega} \mathbb{E}_{m_d \sim \Gamma(S_d)} \mathcal{L}_{CE}(D | (E_d \odot m_d, W_s)), \tag{15}
\]

where \(S_e\) denotes the search space, \(\Gamma(S_d)\) is the prior distribution of the search space, \(\{E_d, W_s\}\) is the best supernet parameter and \(\Omega\) denotes the parameter space of the supernet. By decoupling the dependency between training embedding and dimension search, we no longer need to train a sub-architecture from scratch, which reduces computation cost significantly.

#### 3.2.2 Supernet Training

Following Eq. 15, we construct the supernet embedding table \(E_s\) with ordinal parameter sharing [36], which is efficient to reuse most of parameters. In our method with search space \(S_e = \{1, \cdots, D - 1, D\}\), the supernet is constructed with maximum dimension \(D\). With no prior knowledge of \(\Gamma\), we then assume \(\Gamma\) as a uniform distribution. Such an assumption proves to be empirically good enough and efficient to apply [13]. Specially, given \(d \sim \text{Uniform}(1, D)\), the first \(d\) elements of \(m_d\) are ones and the rest are zeros. Different from other methods [23, 27], \(m_d\) induce flexible embedding, which is hardware-friendly, instead of sparse embedding, which requires a lot of structure information.

Moreover, \(\hat{E}\) retains various embeddings during training both \(m_e\) and embedding parameters, which affects the supernet directly. To train the supernet adapting to \(m_e\) and reduce the total training time further, we conduct the supernet training and redundant embedding pruning in a unifying way by introducing \(E_s = E \odot m_e\). Finally, we can formulate supernet training as:

\[
\min_{m_e, \hat{E}, \hat{W}} \mathbb{E}_{m_d \sim \text{Uniform}(S_e)} \mathcal{L}_{CE}(D | (\hat{E}, \hat{W})) + \alpha L_s,
\]

\[
\hat{E} = E \odot m_d = E \odot m_e \odot m_d. \tag{16}
\]

#### 3.2.3 Search Strategy

After training the supernet \(\{E_s^*, W_s^*\}\) from Eq. 16, we present an evolutionary search for the optimal dimension mask \(m_d^*\). In the beginning, all candidates are randomly generated. At each epoch, each candidate dimension mask \(m_d\) is evaluated on the validation set \(D_{val}\) by inheriting parameters from the supernet. This part is relatively efficient as no training is involved. After the evaluation, the Top-k candidates are preserved for crossover and mutation operation to generate the candidates for the next epoch. For crossover, two randomly selected candidates are crossed to produce a new one by selecting a random point where the parent’s parts exchange happens. Fig. 4(b) details an example where the blue parts of the two candidates are crossed. For mutation, a randomly selected candidate mutates its choice at each position with the given mutation probability \(prob\).

An example is illustrated in Fig. 4(a) where the blue point is a random mutation. Crossover and mutation are repeated to generate enough new candidates given the corresponding number \(n_c\) and \(n_m\). After \(T\) epoch, we output the best-performed candidate dimension mask as the optimal dimension mask \(m_d^*\). This process is shown in Algorithm 1 in line 7-17.

### 3.3 Parameter Re-training

During the supernet training, we map all raw features into embeddings. Thus to eliminate the influence of these embeddings, a re-training stage is desired to train the model with only optimal number of embeddings and embedding dimensions. The embedding mask \(m_e^*\) and the field-wise dimension mask \(m_d^*\) are obtained followed Eq. 16. In the re-training stage, the objective becomes

\[
\arg\min_{E, W} \mathcal{L}_{CE}(D | (E \odot m_e^* \odot m_d^*, W)). \tag{17}
\]

In summary, the overall process of OptEmbed can be summarized as Algorithm 1.
Algorithm 1: The OptEmbed Algorithm

Require: training dataset $\mathcal{D}$, validation dataset $\mathcal{D}_{val}$
Ensure: optimal embedding table $E^*$ and model parameters $W^*$
1: ## SuperNet Training and Embedding Pruning ##
2: while not converge do
3: Sample a mini-batch from the training dataset
4: $(E^*, W^*_t), m_e = \text{SupernetTrain}(\mathcal{D})$ 
5: end while
6: $m_e^*_t = \text{GetBestPerform}(m_e)$
7: ## Dimension Mask Searching ##
8: $\tau = 0; P_t = \text{Initialize population}(n_g + n_c); \text{Topk} = \emptyset$
9: while $\tau < T$ do
10: $\text{AUC}_t = \text{Inference}(E^*_t, W^*_t, \mathcal{D}_{val}, P_t)$
11: $\text{Topk} = \text{Update}\_\text{Topk}(\text{Topk}, P_t, \text{AUC}_t)$
12: $P^c_t = \text{Crossover}(\text{Topk}, P^e_t, P^d_t)$
13: $P^c_t = \text{Mutation}(\text{Topk}, P^e_t; \text{prob})$
14: $P_{t+1} = P^c_t \cup P^d_t$
15: $\tau = \tau + 1$
16: end while
17: $m_e^* = \text{GetBestCand}(P_T)$
18: ## Re-training ##
19: Retrain $(E^*, W^*)$ given $m_e^*$ and $m_d^*$

3.4 Method Discussion

The combination of pruning redundant embeddings and embedding dimension search makes our OptEmbed approach efficient and effective. Table 1 performs a comprehensive comparison of our approach with others that optimize embedding table on whether they satisfied the Req. 1, 2 and 3. Some methods [5, 9, 18, 36, 43] search optimal dimension for embedding table from different granularity: feature-wise (usually grouped by feature value frequency) or field-wise. Another method [32] uses hashing technique to reduce the number of embeddings in the embedding table. The other methods [23, 31] utilize pruning techniques [8, 22] to learn the sparse embedding table directly, which is hard to compatible with the hardware. OptEmb method is the only method that satisfies Req. 1, 2 and 3. The rest of the method tends to violate one or two requirements.

Table 1: Comparison of embedding learning approaches.

| Approach | R1: N.R.F. | R2: E.D.F. | R3: H.F. |
|----------|------------|------------|----------|
| MDE [9]  | ×          | ✓          | ✓        |
| DNIS [5] | ×          | ✓          | ✓        |
| AutoDim [43] | ✓      | ✓          | ✓        |
| AutoField [35] | ✓     | ✓          | ✓        |
| QR [32]  | ✓          | ×          | ✓        |
| PEP [23] | ✓          | ✓          | ✓        |
| OptEmbed | ✓          | ✓          | ✓        |

N.R.F, E.D.F, and H.F. are abbreviations for No Redundant Feature, Embedding Dimension Flexible and Hardwar Friendly.

4 EXPERIMENT

We design experiments to answer the following research questions:

- RQ1: Could OptEmbed achieve superior performance compared with mainstream CTR prediction models and other algorithms that optimize the embedding table?
- RQ2: How does each component of OptEmbed contribute to the final result?
- RQ3: What is the impact of the re-training stage in OptEmbed on the final result?
- RQ4: How efficient is OptEmbed compared with SOTA handcrafted models and other algorithms that optimize the embedding table?
- RQ5: Does OptEmbed output the optimal embedding table?

4.1 Experiment Setup

4.1.1 Datasets. We conduct our experiments on three public datasets. In all following dataset, we randomly split them into 8:1:1 as the training set, validation set, and test set respectively.

- Criteo dataset consists of ad click data over a week. It consists of 26 categorical feature fields and 13 numerical feature fields. We follow the winner solution of the Criteo contest to discretize each numeric value $x$ to $\lfloor \log^2(x) \rfloor$, if $x > 2$; $x = 1$ otherwise. Following the best practice [44], we replace infrequent categorical features with a default "OOV" (i.e. out-of-vocabulary) token, with min_count=2.

- Avazu dataset contains 10 days of click logs. It has 24 fields with categorical features, including instance id, app id, device id, etc. Following the best practice [44], we remove the instance id field and transform the timestamp field into three new fields: hour, weekday and is_weekend. We replace infrequent categorical features with the "OOV" token, with min_count=2.

- KDD12 dataset contains training instances derived from session logs. It has 11 categorical fields, and the click field is the number of times the user clicks the ad. We replace infrequent features with an "OOV" token, with min_count=10.

4.1.2 Metrics. Following the previous works [12, 28, 42], we adopt the commonly used evaluation metric in CTR prediction community: AUC (Area Under ROC) and Log loss (cross-entropy). Notes that in CTR prediction task, 0.1% AUC improvement is considered significant [35, 43]. Besides, we also record the sparsity ratio of the embedding table, the inference time per batch and the training time of models to measure efficiency. The sparsity ratio is calculated as follows:

$$\text{Sparsity} = 1 - \frac{\# \text{Remaining Params}}{|f| \times D}. \quad (18)$$

4.1.3 Baseline Models. We compare the proposed method OptEmbed with the following embedding architecture search methods:

1. AutoDim [43]: This baseline utilizes neural architecture search techniques [21] to select feasible embedding dimensions from a set of pre-defined search space.
2. AutoField [35]: This baseline utilizes neural architecture search techniques [21] to select the essential feature fields.
3. QR [32]: This baseline utilizes the Quotient-Remainder hashing trick to reduce the number of features explicitly.
4. PEP [23]: This baseline adopts trainable thresholds to remove...
redundant elements in the embedding table. We apply the above baselines and OptEmbed method over the following well-known models: DeepFM [12], DCN [34], FNN [42], and IPNN [28].

4.1.4 Implementation Details. In this section, we provide the implementation details. For OptEmbed, (i) General hyper-params: We set the embedding dimension as 64 and batch size as 2048. For the MLP layer, we use three fully-connected layers of size 1024, 512, 256. Following previous work [28], Adam optimizer, Batch Normalization [16] and Xavier initialization [10] are adopted. We select the optimal learning ratio from \{1e-3, 3e-4, 1e-4, 3e-5, 1e-5\} and $l_2$ regularization from \{1e-3, 3e-4, 1e-4, 3e-5, 1e-5, 1e-6\}. (ii) feature mask hyper-params: we select the optimal threshold learning ratio $l_1$ from \{1e-2, 1e-3, 1e-4\} and threshold regularization $a$ from \{1e-4, 3e-5, 1e-5, 3e-6, 1e-6\}. (iii) embedding mask hyper-params: we adopt the same hyper-parameters from previous work [13]. For Criteo, we adopt the same hyper-parameters from previous work [28]. For KDD12, its inconsistency highlights the necessity of our embedding table. Autodim on KDD12. Its inconsistency highlights the necessity of our embedding table. Finally, other baselines tend to behave differently under different cases. Without considering the effect of redundant features, AutoField often results in low sparsity as its granularity is too large. The performance degrade brought by PEP is usually higher than other baselines. However, it might be surpassed by AutoField and AutoDim on KDD12. Due to the lack of available implementation for the AutoDim[43] and AutoField[35] method, we re-implement them based on the details provided by the authors.

4.2 Overall Performance (RQ1)

The overall performance of our OptEmbed and other baselines on four different models using three datasets are reported in Table 2. We summarize our observations below:

First, our OptEmbed is effective and efficient compared with the original model and other baselines. OptEmbed can achieve higher AUC with fewer parameters. However, the benefit brought by OptEmbed differs on various datasets. On Criteo, the benefit tends to be memory reduction. OptEmbed is able to reduce 67% ~ 97% parameters with improvement not considered significant statistically. On KDD12 and Avazu datasets, the benefit tends to be both performance boosting and memory reduction. OptEmbed can significantly increase the AUC by up to 0.15% compared with the original model while saving roughly ~50% of the parameters.

Secondly, among all baselines, PEP is the most similar to OptEmbed. It also tends to be the best-performed baseline. AutoField and AutoDim on KDD12. Its inconsistency highlights the necessity of our embedding table. Moreover, the searching phase of PEP will only stop once the embedding table reaches a predetermined sparsity ratio, completely neglecting the model performance. Stopping criteria may result in a sub-optimal embedding table.

Further, other baselines tend to behave differently under different cases. Without considering the effect of redundant features, AutoField often results in low sparsity as its granularity is too large. The performance degrade brought by PEP is usually higher than other baselines. This might be related to its hashing trick, as it blindly forces different features to merge into one without considering the performance of its embedding.

4.3 Ablation on Different Components (RQ2)

In this section, we discuss the influence of different components of OptEmbed. Here we adopt two variants of OptEmbed: OptEmbed-E for only using the embedding pruning component and OptEmbed-D for only using the dimension search component. The results are shown in Table 3. As we can observe, embedding pruning and dimension search components behave differently given various
datasets. On the Criteo dataset, both the components reduce the embedding parameters. On DCN model, OptEmbed-E and OptEmbed-D can slightly improve model performance. OptEmbed combines these two components to obtain an optimal embedding table with fewer parameters and higher model performance. On the Avazu dataset, OptEmbed-E makes no significant difference compared with original model. This may be due to the overwhelming majority of feature values in the Avazu dataset being ID features, which tend to be informative in prediction. Hence, the optimal embedding table obtained by OptEmbed and OptEmbed-D usually is similar to each other. In all, these two components should be utilized in a unifying way to obtain the optimal embedding table considering differences between datasets.

### 4.4 Ablation on Re-training(RQ3)

We investigate the necessity of Section 3.3 upon the result of the DCN model over both Criteo and Avazu datasets. Results are shown in Table 4. We compare the performance of OptEmbed under different settings with and without re-training. It can be observed that re-training can generally improve the performance. Without re-training, the neural network will inherit the sub-optimal model parameters from the supernet, which is trained for predicting the performance of all possible field-wise dimension masks. Re-training makes the model parameter optimal under the constraint of the embedding mask and field-wise dimension mask.

| Basic Model | Metrics     | Metrics     |
|-------------|-------------|-------------|
|              | AUC | Logloss | Sparsity |
| DeepFM       | Original 0.8104 | 0.4409 | -       |
|              | OptEmbed-E 0.8103 | 0.4410 | 0.6267    |
|              | OptEmbed-D 0.8105 | 0.4409 | 0.9684    |
|              | OptEmbed 0.8105 | 0.4409 | -       |
| DCN          | Original 0.8104 | 0.4408 | -       |
|              | OptEmbed-E 0.8110 | 0.4404 | 0.6111    |
|              | OptEmbed-D 0.8110 | 0.4403 | 0.7192    |
|              | OptEmbed 0.8113 | 0.4402 | 0.8534    |
| DeepFM       | Original 0.7884 | 0.3751 | -       |
|              | OptEmbed-E 0.7884 | 0.3752 | 0.0000    |
|              | OptEmbed-D 0.7888 | 0.3750 | 0.3927    |
|              | OptEmbed 0.7888 | 0.3750 | 0.3927    |
| DCN          | Original 0.7894 | 0.3748 | -       |
|              | OptEmbed-E 0.7895 | 0.3746 | 0.0024    |
|              | OptEmbed-D 0.7900 | 0.3740 | 0.3044    |
|              | OptEmbed 0.7900 | 0.3743 | 0.6840    |

Table 4: Ablation About Re-training Stage.

Retrain w. w.o. Avazu KDD12

| Dataset | Criteo | Avazu | KDD12 |
|---------|--------|-------|-------|
|        | AUC    | Logloss |       |
| Retrain w. w.o. | 0.8113 | 0.4402 | 0.3743 | 0.1523 | 0.1526 |
| Logloss | 0.8110 | 0.4404 | 0.3749 | 0.1523 | 0.1526 |

w. stands for with re-training, w.o. stands for without re-training.

### 4.5 Efficiency Analysis(RQ4)

In addition to the model effectiveness, training and inference efficiency are also vital when deploying the CTR prediction model into reality. In this section, we investigate the efficiency of OptEmbed from both the time and space aspects.

#### 4.5.1 Time Complexity

We illustrate the total training and inference time of DeepFM model trained on all three datasets in Fig. 5. Here we define the total training time as the sum of mask searching time(time required to obtain the embedding and/or dimension mask given different methods) and re-training time(time for re-training the parameters under the constraint of the embedding and/or dimension mask).

For the total training time in Fig. 5(a), we can observe that QR and AutoField tend to have faster speeds than other methods. For QR, no network architecture search is involved. It also has a smaller embedding table, leading to a faster training speed per epoch. AutoField has a smaller search space than other baselines since it only contains the feature field. Surprisingly, the total training time of original model is not always the fastest. This is because original model may take more epochs to converge. OptEmbed is faster than PEP and AutoDim because they have respectively slower convergence speeds during the mask searching and re-training phase.

The inference time is crucial when deploying the model in reality. As shown in Fig. 5(b), OptEmbed achieves the least inference time. This is because the final embedding table obtained by OptEmbed has the least parameters. PEP requires the longest inference time, even longer than original model, because its embedding table tends to be sparse and hardware-unfriendly. Note that it is inevitable to cost additional time to search masks for OptEmbed. However, the cost is worth considering the performance increase and inference time saving, which are more important in practice.

Figure 5: A case study about the candidate setting.

#### 4.5.2 Space Complexity

We plot the parameter-AUC curve of the DeepFM model on both Criteo and KDD12 datasets in Fig. 6, which reflects the relationship between the space complexity of the embedding table and model performance. There are multiple PEP points as it requires predetermined sparsity ratios as stopping criteria. So we can easily control the final sparsity ratio. AutoDim, AutoField and OptEmbed primarily aim to improve model performance. However, there is no guarantee of the final sparsity ratio. Hence we only plot one point for each method. From Fig. 6 we can make the following observations: (i) OptEmbed outperforms other baselines with the highest AUC score and the smallest embedding size. (ii) Model performance of PEP tends to degrade with the decrease of embedding parameters. (iii) AutoDim and AutoField only optimize the embedding table along one axis. Hence they do not have stable performance among datasets. They perform well on the Criteo dataset. However, they are surpassed by PEP on the KDD12 dataset.
4.6 Case Study (RQ5)

This section uses a case study to investigate the optimal embedding table obtained from OptEmbed. We select the embedding table of FNN model trained on Avazu dataset as an example and exclude all anonymous feature fields. Criteo and KDD12 datasets are not selected because all fields are anonymous. In Fig. 7, we plot total parameters, embedding numbers, dimensions and normalize them with corresponding values from original model, respectively. We can make the following observations. First, each field’s optimal dimensions and remaining feature values vary from one to another. This highlights the necessity for OptEmbed. Second, id-like features (like `site_id`, `app_id`, `device_id`) tend to have higher values than others. Such an observation is consistent with human intuition as the id-like features are the core of collaborative filtering-based recommender system. Third, it is surprising to find out that no embedding is assigned for the `weekday` and `is_weekday` fields. These two fields are manually created following the best practice [44]. However, their contained information is more likely to be covered by the `hour` field. Such an observation shows the limit of human-defined feature selection methods.

Figure 7: Case Study of OptEmbed output on Avazu

5 RELATED WORK

We discuss how our work is situated in two research topics: CTR prediction and embedding table optimization. Many machine learning models have been developed for CTR prediction [3, 29, 30]. Due to the powerful learning ability, the mainstream CTR prediction research is dominated by deep learning models [26, 44]. Wide&Deep [4] and FNN [42] introduce an embedding table to transform the raw inputs and an MLP layer to model high-level representations. DeepFM [12], DCN [34] and IPNN [28] rely on various feature interaction layers to improve performance. Increasing research focus on how to model complex feature interaction [19, 20, 24]. With AutoML technique, AutoFIS [20] and AutoFeature [19] search for feature interaction instead of modeling with implicit layer. OptEmbed is perpendicular to all these researches by providing an optimal embedding table for various base models.

Studies on optimizing embedding tables can mainly be categorized into reducing memory usage, searching embedding dimensions, and pruning redundant values. Hash embedding [37] designs a hash method to reduce the embedding table size. Double hash [41] adopts the double hash method for one feature value to reduce the collision in hashing method. Q-R trick [32] is also introduced to conquer the collision problem. Some quantized techniques [17, 18] are also borrowed for compressing the embedding table. To search field-wise embedding dimension, NAS has been utilized automatically based on well-defined search space [5, 9, 43]. NAS has also been utilized to search informative feature field [35] automatically. AutoAS [36] introduces a one-shot search for both embedding dimension and architecture. Pruning redundant values in embedding tables attracted more attention recently. PEP [23] designs a soft threshold method to filter out low magnitude values in the embedding table with a predetermined sparsity ratio. DeepLight [6] prune embedding table and other components on a pre-train network. The single-shot [27] pruning method is also used to prune the embedding table carefully. This paper proposes three requirements for an optimal embedding table in Req. 1, 2 and 3, and make a thorough comparison with these works. To the best of our knowledge, we are the first to design an optimal embedding framework satisfying three requirements for CTR prediction.

6 CONCLUSION

This paper first proposes the requirements for an optimal embedding table. Based on these requirements, a novel, model-agnostic framework OptEmbed is proposed. OptEmbed optimizes the embedding table in a unifying way. It is capable of combining the supernet parameter training with redundant embedding pruning. A one-shot embedding search method is proposed based on the supernet to efficiently find optimal dimensions for different fields and obtain the optimal embedding table. Extensive experiments on three large-scale datasets demonstrate the superiority of OptEmbed in terms of both model performance and model size reduction. Several ablation studies demonstrate that basic models require optimal embedding tables on various datasets. Moreover, we also interpret the obtained result on feature fields, highlighting that our method learns the optimal embedding table.

ACKNOWLEDGMENTS

We specifically want to thanks Dr. Jin Guo for her helpful suggestions regarding the paper writing.
Association for Computing Machinery, New York, NY, USA, 3563–3567.

[39] Bencheng Yan, Pengjie Wang, Kai Zhang, Wei Lin, Kuang-Chih Lee, Jian Xu, and Bo Zheng. 2021. Learning Effective and Efficient Embedding via an Adaptively-Masked Twins-based Layer. In CIKM ’21: The 30th ACM International Conference on Information and Knowledge Management. ACM, Australia, 3568–3572.

[40] Xin Yuan, Pedro Henrique Pamplona Savarrese, and Michael Maire. 2021. Growing Efficient Deep Networks by Structured Continuous Sparsification. In 9th International Conference on Learning Representations, ICLR 2021. OpenReview.net, Austria. https://openreview.net/forum?id=wb3wxCObbRT

[41] Caojin Zhang, Yicun Liu, Yuanpu Xie, Sofia Ira Ktena, Akshay Gupta, Pranay Kumar Myana, Deepak Dilipkumar, Suvadip Paul, Ikuhiro Ihara, Prasang Upadhyaya, Ferenc Huszar, and Wenzhe Shi. 2020. Model Size Reduction Using Frequency-Based Double Hashing for Recommender Systems. Association for Computing Machinery, New York, NY, USA, 521–526.

[42] Weinan Zhang, Tianming Du, and Jun Wang. 2016. Deep Learning over Multi-field Categorical Data - - A Case Study on User Response Prediction. In Advances in Information Retrieval - 38th European Conference on IR Research, ECIR 2016, Vol. 9626. Springer, Italy, 45–57. https://doi.org/10.1007/978-3-319-30671-1_4

[43] Xiangyu Zhao, Haochen Liu, Hui Liu, Jiliang Tang, Werset Guo, Jun Shi, Sida Wang, Huji Gao, and Bo Long. 2021. AutoDim: Field-aware Embedding Dimension Searchin Recommender Systems. In WWW ’21: The Web Conference 2021. ACM / IW3C2, Slovenia, 3015–3022.

[44] Jieming Zhu, Jinyang Liu, Shuai Yang, Qi Zhang, and Xiuqiang He. 2021. Open Benchmarking for Click-Through Rate Prediction. In Proceedings of the 30th ACM International Conference on Information & Knowledge Management. Association for Computing Machinery, Australia, 2759–2769.