xLightFM: Extremely Memory-Efficient Factorization Machine

Gangwei Jiang
University of Science and Technology of China
gwjiang@mail.ustc.edu.cn

Hao Wang
University of Science and Technology of China
wanghao3@mail.ustc.edu.cn

Jin Chen
University of Electronic Science and Technology of China
chenjin@std.uestc.edu.cn

Haoyu Wang
SUNY Buffalo
hwang79@buffalo.edu

Defu Lian*
University of Science and Technology of China
liandefu@ustc.edu.cn

Enhong Chen
University of Science and Technology of China
cheneh@ustc.edu.cn

ABSTRACT

The factorization-based models have achieved great success in online advertisements and recommender systems due to the capability of efficiently modeling combinational features. These models encode feature interactions by the vector product between feature embedding. Despite the improvement of generalization, the memory consumption of these models grows significantly, because they usually take hundreds to thousands of large categorical features as input. Several existing works try to reduce the memory footprint by hashing, randomized embedding composition, and dimensionality search, but they suffer from either substantial performance degradation or limited memory compression. To this end, in this paper, we propose an extremely memory-efficient Factorization Machine (xLightFM), where each category embedding is composited with latent vectors selected from codebooks. Based on the characteristics of each categorical feature, we further propose to adapt the codebook size with the neural architecture search techniques for compositing the embedding of each categorical feature. This further pushes the limits of memory compression while incurring negligible degradation or even some improvements in prediction performance. We extensively evaluate the proposed algorithm with two real-world datasets. The results demonstrate that xLightFM can outperform the state-of-the-art lightweight factorization-based methods in terms of both prediction quality and memory footprint, and achieve more than 18x and 27x memory compression compared to the vanilla FM on these two datasets, respectively.

CCS CONCEPTS

• **Information systems → Collaborative filtering**.

KEYWORDS

AutoML, Quantization, FM, Lightweight, Memory-Efficient

*Corresponding author

SIGIR '21, July 11–15, 2021, Virtual Event, Canada

ACM ISBN 978-1-4503-8037-9/21/07…$15.00
https://doi.org/10.1145/3404835.3462941

ACM Reference Format:
Gangwei Jiang, Hao Wang, Jin Chen, Haoyu Wang, Defu Lian, and Enhong Chen. 2021. xLightFM: Extremely Memory-Efficient Factorization Machine. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '21), July 11–15, 2021, Virtual Event, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3404835.3462941

1 INTRODUCTION

Many modern predictive systems, such as online advertisements and recommender systems, suffer from data or feature sparsity. For data sparsity, some works [18, 21, 28] suggest using cross-domain data or negative samples to solve the problem. For the latter, modeling the interactions between features is essential for these systems. Traditional feature engineering methods like feature crossing would make engineered features more sparse, greatly increasing the complexity of learning models and easily suffering from the over-fitting problem. Factorization Machine (FM) [32] uses pairwise factorized interactions between variables to address this issue, leading to high prediction quality and achieving great success in online advertisements and recommender systems. Since FM only encodes the second-order feature interactions, FM has been extended to modeling high-order interactions explicitly [2, 23, 37] or implicitly [8, 9]. Since some interactions may not be useful for prediction, these interactions could not be selected into the models with the attention techniques [38] or architecture search techniques [17, 24].

Despite the improvement of generalization, the memory consumption of these FM models grows significantly, because the FM models usually take hundreds to thousands of large categorical features as input. For example, when the standard FM is trained for real-world recommender systems with billions of items and users, FM requires hundreds of Gigabytes memory footprint in the absence of other side features, so that it is challenging for them to be deployed in real-world online serving systems. Moreover, the FM models have become an important machine learning method for sparse features, so they have a wide range of application scenarios. When these applications are tailored for mobile devices, lightweight FM models are urgently needed at the inference stage[1]. Therefore, it is meaningful and necessary to design the lightweight FM models for enjoying economic memory consumption of inference.

To reduce the memory footprint of FM, Discrete FM [25] has been proposed to binarize latent factors (i.e., embedding) of each feature.

[1]The learning of the FM models is usually finished in the computing servers, so the compression is not our goal at the training stage

However, this method suffers from substantial performance degradation since binarization leads to much information loss. To preserve more information while still achieving low memory footprint, product quantization [13] has been adapted to matrix factorization [20] by compositing feature embedding from codebooks, each of which is a fixed-size set of latent factors (i.e., codewords). Other existing work focuses on reducing the memory consumption of recommender systems by randomized embedding composition [16, 35], AutoML-based dimensionality search [47], or dimension scaling along with feature popularity [7]. However, some of them still suffer from comparatively large performance degradation due to data-independent hashing. Another of them lead to a much smaller magnitude of memory compression than binarization and quantization since they still use real-valued embedding. Therefore, it is worth adapting product quantization to the FMs, but this is not straightforward, because each categorical feature[2] has distinct characteristics, so that the codebook size (i.e., the number of codewords in the codebook) or the number of codebooks should vary from feature to feature.

To this end, in this paper, we propose an extremely memory-efficient Factorization Machine (xLightFM for short), with the aim of minimizing memory footprint at small or even no price of accuracy degradation. Particularly, xLightFM composites each category embedding with multiple codebooks, from each of which only one codeword could be selected. The codeword selection is guided by the loss of FMs and the Euclidean distance between composited representation and each category's original embedding. To further push the limits of memory compression, we propose to leverage the Neural Architecture Search (NAS) techniques [10, 26, 39] for adapting the codebook size to the categorical feature's characteristic, including the number of categories and the category frequency.

The contributions delivered in this paper are summarized as follows:

- To the best of our knowledge, we propose an end-to-end quantization-based factorization machine for the first time, to greatly reduce the memory footprint of the model while lowering the quantization loss as much as possible.
- Based on each categorical feature's characteristic, we propose to adapt the codebook size for compositing category embedding based on the AutoML techniques. This pushes the limits of memory compression of the FM models.
- We evaluate the proposed algorithm (xLightFM for short) on two real-world datasets for the CTR prediction. The results demonstrate that xLightFM only has 18x and 27x less memory footprint than the vanilla FM on the two datasets, respectively and outperforms the state-of-the-art baselines in terms of both prediction quality and memory footprint.

2 RELATED WORK

In this section, we will review the recent advance of memory-efficient recommendation algorithms and factorization-based recommendation models.

[2]For concise, we assume FMs take categorical features as input. Each categorical feature contains many categories, and categorical feature is also called field in FM

2.1 Memory-Efficient Recommendation

Usually the embedding sizes are the same for all items and features in recommender systems. However, this representation can be a waste of memory footprint, because certain items or features can be encoded by low-dimensional vectors. To solve this problem, recent methods [7, 14, 47] suggest assigning varying dimensions to different items or categorical features. However, the memory compression is limited since they still use real-valued embedding. The more memory-efficient recommendation can be organized into the following two categories.

The first taxonomy is hashing-based recommendation, to transform continuous user and item embeddings into binary codes. The binary representation benefits memory efficiency a lot, since binary codes can be encoded into integers. The learning can be data-independent [5, 11, 16, 29, 35, 36, 40], or from pre-trained continuous representation [27, 46, 48], or from data directly [19, 22, 41, 44, 45]. However, the hashing-based methods usually suffer from poor accuracy of approximate recommendation, because the binary representation is more difficult to optimize and less expressive than the continuous representation.

The second taxonomy is quantization-based recommendation, representing each item embedding via a semi-structured vector. They usually use a small number of representative vectors to approximate the continuous representation. The representative quantization methods include PQ [13], OPQ [6], CQ [42], AQ [1], etc. However, they suffer from two significant flaws: 1) they are data-independent quantization methods, i.e. they quantize the pre-trained item representation, which can not be trained jointly with recommendation model in an end-to-end way, and 2) their objective functions are reconstruction errors based on Euclidean distance, which is incompatible with user-item's relevance. One recent work most close to ours is LightRec [20], which is an end-to-end quantization model recommendation. Unfortunately, it is devised for matrix factorization-based models, which can not handle enormous numbers of categorical features.

2.2 Factorization-based Recommendation

How to learn patterns from categorical feature interactions is a significant challenge in recommendation. Researchers apply factorization machine (FM) [31] and its variant FFM [15] to design second-order cross features automatically. However, the importance of each feature interaction may be different, and therefore, attentional FM (AFM)[38] is proposed. Nonetheless, these models all simply develop second-order feature interaction, suffering from the same expressiveness issue for modeling complex real-world data accurately. With the benefit of DNNs, researchers adopt deep models to design high-order interactions. FNN [43], Deep Crossing [34], Wide&Deep [4], PNN [30], DeepFM [8], and NFM [9] are representative models to build up implicit high-order interactions. CrossNet [37] and xDeepFM [23] are representative models to build up explicit high-order interactions. Unfortunately, the aforementioned models simply enumerate all feature interactions and ignore the difference of complexity among features. Thus, AutoFIS [24] and AutoFeature [17] are proposed to find crucial feature interactions automatically. However, whatever FM or deep factorization-based

models, all incur high memory consumption and computation complexity when there are an enormous number of categorical features. There is no work proposing models to alleviate the memory efficiency problem of FM based on quantization. The only one lightweight model for FM is based on hashing [25] while hashing-based methods may suffer from low accuracy as we discuss in Section 2.1.

3 METHOD

In this section, we will first briefly overview the entire model pipeline, then introduce the preliminaries and specific analysis of memory footprint, and finally, elaborate the technical details of two proposed models LightFM and xLightFM. xLightFM differs from LightFM in the adaptive size of codebooks in each field.

3.1 Overview

Before introducing the proposed models, we will first illustrate the typical online recommendation model – Factorization Machine (FM), which is the backbone of LightFM and xLightFM, and analyze its memory footprint in the inference stage. Considering extensive memory overhead in the vanilla FM model, we propose a novel memory-efficient model (LightFM for short), which composites each category embedding with multiple codebooks and from each of codebooks only one codeword is selected. Since only the indexes of the selected codewords in codebooks are stored for each item while the size of codebooks is irrelevant to data, the FM model could be substantially compressed. Then, the composited embedding of each category, after continuous relaxation, is forwarded into the FM model for end-to-end training.

Meanwhile, it's worth noting that the cardinality and frequency distribution of categorical features are different from each other. This indicates that the number of codewords, an indicator of representation ability, should be adaptive to characteristics of categorical features. However, LightFM directly uses the same number of codewords in each codebook for categorical features, lacking the ability of dynamically allocating the appropriate number of codewords for each categorical feature. To this end, we further propose a variant – xLightFM, which adopts the Neural Architecture Search (NAS) techniques to adapt the codebook size to each feature's characteristics. This further reduces the memory footprint of each category embedding, and the overall architecture of the xlightFM is shown in Fig. 1. Specifically, the codebook size of each categorical feature is used to define the search space of NAS, and then the parameters and architecture of the xLightFM models were iteratively optimized under the memory constraints on the training set and the validation set, respectively, by a differentiable architecture search algorithm. Next we will further elaborate on the technical details of each part.

3.2 Preliminary and Analysis

In the real-world scenario of recommendation or advertisements, training samples are often composed of hundreds to thousands of categorical features. Feature interactions are important for high prediction quality in these systems. To address the sparsity problem resulting from feature crossing, Factorization Machine (FM) is proposed for efficiently modeling the second-order feature interactions by pairwise factorized interactions and has achieved great success in these predictive systems. FM has been further extended for modeling high-order interactions explicitly or implicitly, which improves the prediction quality. Without loss of generality, we only consider FM as the backbone of LightFM and xLightFM, but it is straightforward to extend our methods to the FM variants. Formally, in the FM model, the output can be represented as follows:

$$\hat{y}(\boldsymbol{x}) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left\langle \boldsymbol{v}_i, \boldsymbol{v}_j \right\rangle x_i x_j, \tag{1}$$

where $\boldsymbol{x}$ is a sparse feature vector of length n, and clearly x_i indicate the i-th element of the vector $\boldsymbol{x}$ while $\boldsymbol{v}_i$ being the corresponding embedding of feature i. The output $\hat{y}$ of FM consists of two parts. The linear weight w_i in the former part represents the importance of the first-order feature x_i, and inner product $\langle v_i, v_j \rangle$ in the latter part measures the importance of the second-order features $x_i \cdot x_j$. Though the length n is usually very large, the sparsity of the vector $\boldsymbol{x}$ make many inner product being zero. Each occurrence of a feature pair will update the embedding of these two features, so FM can address the sparsity to improve the generalization ability. Here we stack the embedding of features by rows, and obtain the embedding matrix $\boldsymbol{V} \in \mathbb{R}^{n \times l}$, where n denotes the number of features and l indicates the embedding size. As aforementioned, we consider to use categorical features for illustration, n equals the total number of categories in all categorical features. For example, in collaborative filtering, there is only user id and item id, then n equals the number of users plus the number of items. Then the embedding matrix $\boldsymbol{V}$ is obtained by stacking user embedding matrix with item embedding matrix by row.

It's no doubt that FM model has proved its effectiveness in various scenarios including large-scale industrial recommendation system. However, since the feature embeddings $\boldsymbol{V}$ may be a large size matrix (over million rows), the memory consumption is an essential bottleneck for FM in practice. For example, if 1 billion products are for sale on the E-commerce website, which serves 100 millions of users each day on average, FM (l=128) occupies more than 524 GB of memory. In this case, such substantial memory cost may not be afforded by real-world online serving systems, limiting the application of FM model. Therefore, it's necessary to design a memory-efficient FM so that the FM model can be applied in many real-world scenarios.

3.3 LightFM: Memory-Efficient Factorization Machine

For alleviating the memory overhead problem, researchers have proposed different types of memory-efficient methods, like hashing-based, index-based, and quantization-based methods. Although hashing-based methods can significantly compress the memory overhead, they also sacrifice predictive capacity. For quantization-based methods, they can not be directly applied here due to numerous categorical features. Motivated by the much stronger representation of multi-codebook composition [13], it's valuable to apply it to FM for reducing memory. Therefore, we propose LightFM based on multi-codebook composition.

3.3.1 Product Quantization. Product quantization(PQ) [13] is a multi-codebook quantization technique, which first decomposes the embedding space into the Cartesian product of subspaces and

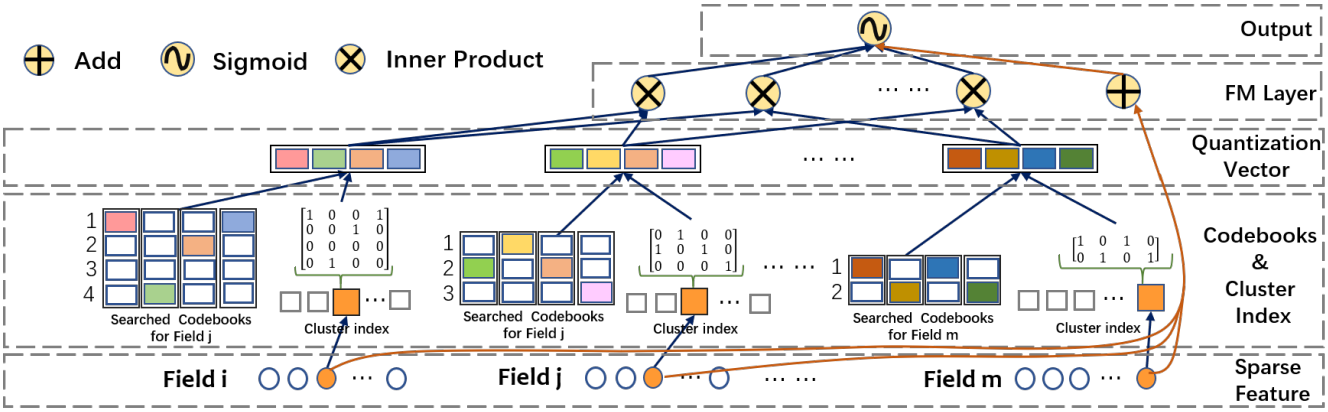

Figure 1: Framework of xLightFM.

conducts the k-means clustering in each subspace for obtaining center vectors. Then each embedding vector can be represented by a set of indexes, each of which indicates the cluster indicator in each subspace. Since the center vector (cluster vector for unification) of each subspace is independent to each other, both Euclidean distance and inner product between feature vectors can be efficiently computed based on the precomputed lookup tables.

The definitions and notations of PQ when applied to FM are presented as below. For the FM model, each feature i belongs to a unique field $F(i)$, and the cardinality and distribution between these fields are different, so we need to use a separate quantizer to compress the embedding for each field. That is, in our task, M quantizers should be applied for quantifying the embedded matrix $\boldsymbol{V}$, where M is the number of fields. And then in the framework of PQ, each $\boldsymbol{v}_i$ can be represented as the concatenation of D subvectors of length $l' = l/D$, i.e. , $\boldsymbol{v}_i = [\boldsymbol{v}_i^1, ..., \boldsymbol{v}_i^d, ..., \boldsymbol{v}_i^D]$, and the quantizer for the field $F(i)$ will have D distinct codebooks, $C_{F(i)}^d, 1 \le d \le D$. Specifically, $C_{F(i)}^d \equiv \{\boldsymbol{c}_{F(i),k}^d | k = 1, ..., K\} \in \mathbb{R}^{l' \times K}$ is composed of a fixed number K of codewords, each of which represents a clustering center in the d-th subspace of embedding in the field $F(i)$.

Below, for simplicity, we abbreviate $C_{F(i)}^d$ to C_i^d and $\boldsymbol{c}_{F(i),k}^d$ to $\boldsymbol{c}_{i,k}^d$. Assuming the indexes of embedding $\boldsymbol{v}_i$ in D subspaces are denoted by $k_i^1, \cdots, k_i^D$, the concatenation of all selected codewords $\boldsymbol{q}_i \equiv [\boldsymbol{c}_{i,k_i^1}^1, \cdots, \boldsymbol{c}_{i,k_i^D}^D]$ forms the approximate vector. The objective function of PQ for FM model is formulated as follows:

$$\min_{C,\boldsymbol{b}} \sum_{i \in F} \sum_{d=1}^{D} \left\| \boldsymbol{v}_i^d - C_i^d \boldsymbol{b}_i^d \right\|^2, \tag{2}$$

where F is a field and $\boldsymbol{b}_i^d$ is an one-hot vector to represent cluster indicator k_i^d of $\boldsymbol{v}_i$ in the d-th subspace. Hence, we compress the embedding matrix $\boldsymbol{V}$ of M fields into K-way D-dim discrete codes.

Suppose there are 20 fields, each field is represented by codebooks with $K = 256, D = 4, l = 128$, the compressed FM model only cost $20 * (256 * \lceil log256 \rceil / 8 * 4 + 256 * 128 * 4)/1024/1024 = 2.52$MB memory for storage. In our problem, indexes can be stored numerically, where K indexes, only cost $K\lceil logK \rceil$ bits.

3.3.2 Objective Function. It's intuitive to first obtain the feature embeddings learned from the vanilla FM and directly utilize the PQ algorithm to obtain the compressed codewords. However, the inner product based similarity between features in FM is incompatible with Euclidean distance used for vector quantization. Moreover, the learned feature embedding may not be the most suitable for direct vector quantization. Therefore, it's necessary to simultaneously learn codebooks and assign codewords for category embeddings of FM in an end-to-end manner. Along this line, we propose the LightFM to integrate the learning of feature embeddings into quantization, as illustrated in Fig. (2). Specifically, based on Eq. (2), which aims to minimize distortion error between feature embedding and composited codewords, we can regard the $C_i^d \boldsymbol{b}_i^d$ as the quantized representation of $\boldsymbol{v}_i$ from another perspective. Then, we utilize these quantized vectors to estimate the predicted score:

$$\hat{y}(\mathbf{x}) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{1 \le i < j \le n} \sum_{d} \left\langle C_i^d \boldsymbol{b}_i^d, C_j^d \boldsymbol{b}_j^d \right\rangle x_i x_j. \tag{3}$$

Furthermore, we adopt the widely-used differentiable Binary Cross Entropy (BCE) loss as the objective function, which is formulated as follows:

$$\mathcal{L}_{bce}(\hat{y}, y) = \sum_{(\boldsymbol{x}, y) \in \mathcal{D}} -y \log \sigma(\hat{y}(\boldsymbol{x})) - (1 - y) \log(1 - \sigma(\hat{y}(\boldsymbol{x}))), \tag{4}$$

where $\mathcal{D}$ represents the whole training set and $\sigma(x) = 1/(1 + e^{-x})$. During the training process, the only gradient information for each category embedding vector is from the $\boldsymbol{b}_i^d$, which is insufficient for guiding their learning. To this end, we add the regularizer term into the original loss function, for the sake of penalizing the large difference between embedding vector and composite representation. The overall loss function is then formulated as follows:

$$\begin{aligned} \mathcal{L}(\hat{y}, y) &= \mathcal{L}_{bce} + \lambda \mathcal{L}_{distance}, \\ \mathcal{L}_{distance} &\equiv \sum_{i=0}^{n} w_i * \mathrm{s}(v_i, q_i), \end{aligned} \tag{5}$$

where $w_i = \log f_i / \sum_{k=1}^{N} \log f_k$ is calculated based on the popularity of each feature i, and f_i is the frequency that the feature i appears. v_i and q_i is the embedding vector and composite representation. Note that the distance loss is weighted with the popularity. The idea of this design is that we believe that the higher the popularity of the category, the more accurate the results will be in the pre-training process, and the greater the impact on the prediction

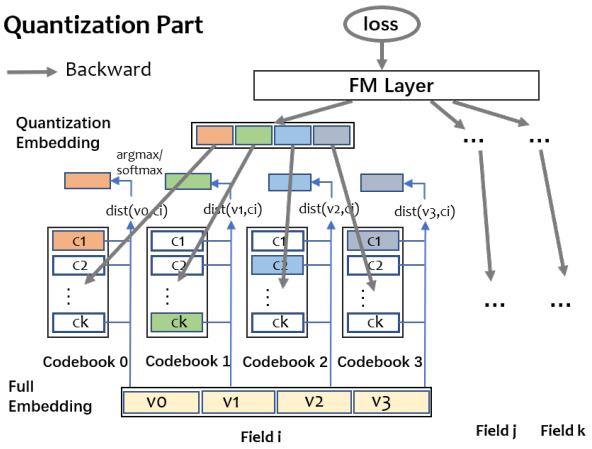

Figure 2: The optimization process of LightFm.

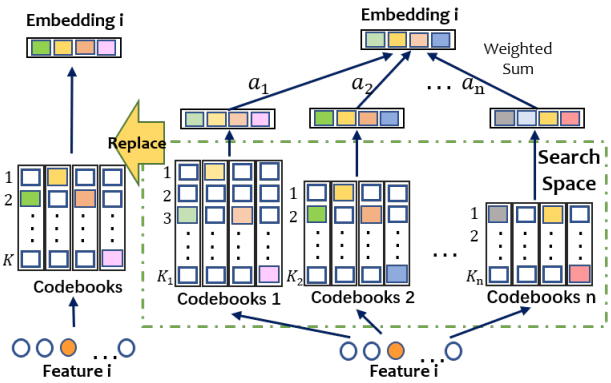

Figure 3: The quatization module in xLightFM.

quality. And later in the experiment part of the paper, we also verify the correctness of this idea.

3.3.3 Optimization. However, this objective function Eq. 4 cannot directly be optimized in an end-to-end manner, since the maximum selection operator in codeword assignment process is non-differentiable. In order to address this problem, we utilize the continuous relaxation to the one-hot index $\boldsymbol{b}$ of the corresponding codeword. Concretely, for a specific codeword assignment $\boldsymbol{b}_i$, the selection operation is defined as follows:

$$\boldsymbol{b}_i = \text{one-hot}(\arg\max_k \text{s}(\boldsymbol{v}_i, \boldsymbol{c}_{i,k})), \tag{6}$$

where $\boldsymbol{v}_i$ is the embedding of the category i, and $\boldsymbol{c}_{i,k}$ is the k-th codeword. $\boldsymbol{b}_i$ is the one-hot representation of codeword index of feature i, and $\text{s}(\boldsymbol{v}_i, \boldsymbol{c}_k)$ measures the similarity between the embedding vector $\boldsymbol{v}_i$ and the codeword $\boldsymbol{c}_k$. Then, the most similar codeword could be computed as $C_i\boldsymbol{b}_i$. For addressing this non-differentiable maximum operation, we adopt tempered softmax to relaxed it, which can be regarded as a soft variant of the max function. For a specific k-th element of b_i, the definition is illustrated as follows:

$$b_i[k] \approx \widetilde{b}_i[k] = \frac{\exp\left(\text{s}\left(\boldsymbol{v}_i, \boldsymbol{c}_{i,k}\right)/T\right)}{\sum_k \exp\left(\text{s}\left(\boldsymbol{v}_i, \boldsymbol{c}_{i,k}\right)/T\right)}, \tag{7}$$

when the temperature $T \to 0$, the formula Eq. 7 is a complete approximation to discrete index $b_i[k]$. Furthermore, the maximization selection of the most similar codeword could be related by $C_i\widetilde{b}_i$. It's worth noting that we adopt the continuous relaxation $\widetilde{b}_i$ for the back-propagation of the gradient. However, we directly utilize the original b_i in the forward process to make the consistent with the online recommendation. To close the difference between forward pass and backward pass, we follow a similar idea to Straight-Through Estimator [3, 20] and rewrite the codeword index b_i as follows:

$$\widehat{b}_i = \widetilde{b}_i + \text{stop\_gradient}(b_i - \widetilde{b}_i), \tag{8}$$

where the stop_gradient operator will prevent the gradient from back-propagating through it. In the forward pass, stop_gradient does not work and the most similar codeword is directly utilized for inference: $\widehat{b}_i = b_i$. Besides, in the backward pass, the stop_gradient operator will take effect to make $\nabla_{\widehat{b}_i}\mathcal{L} = \nabla_{\widetilde{b}_i}\mathcal{L}$ by taking a larger T to back-propagate gradient through it. Therefore, with the combination of tempered softmax and stop_gradient trick, LightFM can optimize the compressed representation of feature embeddings with loss function of original FM in an end-to-end training manner.

3.4 xLightFM: Extremely Memory-Efficient Factorization Machine

Although LightFM has quantized the feature embeddings to efficiently reduce the memory overhead, it compresses the embeddings with a fixed number of codewords in the codebooks for different fields, which leads to the failure to achieve maximum performance under memory constraint. As observed, the cardinality and distribution of each categorical feature are not consistent. For example, the values of feature fields Gender are always binary and perform a Bernoulli distribution, while the values of feature fields Age are often represented in multi-valued intervals satisfied with the Gaussian distribution. It's intuitive that different types of fields contain different amounts of information, which encourages us to propose an adaptive method to dynamically assign the suitable numbers to codewords, so as to further reduce memory. However, it's difficult to manually allocate an appropriate number of codewords for a large number of fields. Therefore, we aim to dynamically search for the number of codewords via AutoML techniques, and formulate it as a one-shot search problem. The xLightFM is then proposed to search for the optimal number of codewords for different feature fields under memory constraints.

3.4.1 Codebook Search Space. In this work, the search space is the different number of codewords for different feature fields and we transfer the problem as the selection from the codebooks with different numbers of codewords. Specifically, G candidate codebook spaces with different codeword numbers are set for each codebook C in Eq. 3. Each codebook is assigned with a real-valued weight α_g, indicating the probability of being chosen. Thus, the continuous codebook search space is defined. At each time, we predict the weights corresponding to the codebooks and choose the codebook with the maximum probability. After the introduction of the search space, the representation of embedding Cb is transformed into the following equation:

$$\boldsymbol{c}(\boldsymbol{\alpha}, \widetilde{C}, \widetilde{b}) = \sum_{j=1}^{G} \alpha_j C_j b_j, \text{ where } \boldsymbol{\alpha} \in \{\boldsymbol{\alpha} | \, \|\boldsymbol{\alpha}\|_2 = 1\}, \tag{9}$$

Algorithm 1: Memory-efficient Codebooks Search

Input: Training dataset $\mathcal{D}$, Validation dataset $\overline{\mathcal{D}}$

while *not converged* **do**
  Sample a mini-batch from validation dataset;
  Update $\boldsymbol{\alpha}$ by descending $\nabla_{\boldsymbol{\alpha}} \mathcal{H}(T, \boldsymbol{\alpha})$ according to Eq(12);
  if $\boldsymbol{\alpha}$ *does not meet memory constraints* **then**
    Update $\boldsymbol{\alpha}$ according to Eq(14);
  end
  Sample a mini-batch from trainning dataset;
  Update model parameters T by descending $\nabla_T \mathcal{F}_{\boldsymbol{\alpha}}(T)$ according to Eq(12);
end
return *Searched codebooks described by* $\boldsymbol{\alpha}, T$.

where C_j, b_j is the j-th codebook matrix and index in search space, and $\widetilde{\cdot}$ denotes the set of candidate search spaces for specified parameter, for example $\widetilde{C} \equiv \{C_1, ..., C_j, ..., C_G\}$. $\boldsymbol{\alpha}$ denotes the continuous vector of choosing codebooks. The constraint $\|\boldsymbol{\alpha}\|_2 = 1$ ensures that only one codebook is selected each time. A more detailed explanation is shown in Fig. 3 .

After obtaining the relaxation space representation $\boldsymbol{c}(\boldsymbol{\alpha}, \widetilde{C}, \widetilde{b})$, we further integrate it with the former LightFM framework, and obtain the prediction score of xLightFM as:

$$\hat{y}(\mathbf{x}) = \sum_{1 \le i < j \le n} \sum_{d} \left\langle \boldsymbol{c}(\boldsymbol{\alpha}_i^d, \widetilde{T}_i^d), \boldsymbol{c}(\boldsymbol{\alpha}_j^d, \widetilde{T}_j^d) \right\rangle x_i x_j, \tag{10}$$

where $T = \{C, b\}$ denotes the model parameters involving the codebook matrix and the discrete indices, and $\boldsymbol{\alpha}_i^d$ terms architecture parameter for the d-th codebook of feature i. Then $\boldsymbol{c}(\boldsymbol{\alpha}_i^d, \widetilde{T}_i^d) = \boldsymbol{c}(\boldsymbol{\alpha}_i^d, \widetilde{C}_i^d, \widetilde{b}_i^d)$. For simplicity, only the second order part of the prediction score is given.

3.4.2 Differentiable Searching Strategy. Note that the objective function of xLightFM in Eq. 10 contains both discrete (architecture parameters $\boldsymbol{\alpha}$) and continuous (model parameters T) variables. Therefore, similar to the optimization of discrete variable (index) in Section 3.3.3, the architecture parameters $\boldsymbol{\alpha}$ should be relaxed to a continuous vector to make training differentiable, and here we adopted the Gumbel-Softmax [12] operation to Eq. 9:

$$\boldsymbol{c}(\boldsymbol{\alpha}, \widetilde{C}, \widetilde{b}) = \sum_{j=1}^{G} \frac{\exp\left(\left(\log\left(\alpha_j\right) + g_j\right)/\tau\right)}{\sum_{i=1}^{N} \exp\left(\left(\log\left(\alpha_i\right) + g_i\right)/\tau\right)} C_j b_j, \tag{11}$$

where g_i is gumbel noise which sampled i.i.d from Gumbel(0,1) and τ is the temperature which controls the smoothness of the distribution of sampling results. As the temperature approaches 0, the output of Gumbel-Softmax is similar to the one-hot vector.

With the continuous representation of the space, the objective function can be formalized as a one-shot search problem, which jointly optimizes the model parameters T and architecture parameters $\boldsymbol{\alpha}$. The objective function is as follows:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \mathcal{H}(T, \boldsymbol{\alpha}) \equiv \textstyle\sum_{(x,y) \in \overline{\mathcal{D}}} \ell(\hat{y}(x, \boldsymbol{\alpha}, T^*), y) \\ \text{s.t.} \quad & T^* \equiv \arg\min_T \mathcal{F}_{\alpha}(T) \approx T - \xi \nabla_T \mathcal{F}_{\boldsymbol{\alpha}}(T), \end{aligned} \tag{12}$$

where $\mathcal{F}_{\boldsymbol{\alpha}}$ is the training objective, derivated from the Eq. 5.

$$\mathcal{F}_{\boldsymbol{\alpha}}(T) \equiv \sum_{(x,y) \in \mathcal{D}} \ell(\hat{y}(x, \boldsymbol{\alpha}, T), y) + \lambda \mathcal{L}_{distance} \tag{13}$$

Based on this formulation, xLightFM alternately updates the model parameters T on the training dataset $\mathcal{D}$ and optimizes the architectural parameters $\boldsymbol{\alpha}$ on the validation dataset $\overline{\mathcal{D}}$. However, the global optimization of T^* makes the update of the architecture gradient extremely expensive and even impracticable. To alleviate this calculation problem, we conduct the same approximation scheme as the differentiable architecture search (DARTS) [26] to calculate T^* in Eq. 12, and ξ is the learning rate. In particular, ξ is usually set to 0 to perform the first-order approximation. In this case, the T^* is assumed to be the same as current T for simpl, which could reduce the training time for xLightFM.

Different from other tasks, memory is always a tight resource here, and the tradeoff between memory and performance is one that needs to be considered. While traditional NAS only considers whether the current architecture is conducive to the optimization of the objective function, we add a memory-efficient constraint in the search algorithm. When the step results of $\boldsymbol{\alpha}$ does not meet the constraint of memory, for example, the selected structure consumes more than 100MB of memory. We update $\boldsymbol{\alpha}$ as the follow formulation, and α_i is the i-th element of vector $\boldsymbol{\alpha}$:

$$\alpha_i = \min(\alpha_i, \alpha_{i,last}) \quad \text{if} \quad i \ge S_{last}, \tag{14}$$

where S_{last} denotes the subscript of the codebook selected previous, and we assume that the larger the codebook is, the larger the corresponding subscript is. $\alpha_{i,last}$ represents the value of α_i before updating. Under such updating rules, the algorithm will further search within a smaller range of memory consumption, so that the final result can meet the memory constraint.

The detailed optimization procedure for codebook search is illustrated in Algorithm. 1. In this way, we can simultaneously quantize the feature embeddings and search for the optimal codebook spaces in a joint framework.

3.4.3 Re-train and Inference. When the learned architectural parameters corresponding to codebook spaces are obtained, we abandon the Gumbel-Softmax trick and directly select the optimal codebook by the maximum operation on weight $\boldsymbol{\alpha_g}$, to derive the architecture which contains the most suitable size of the codebook for each field. $\overline{C}, \overline{b}$ denotes the selected codebook and cluster index. Besides, based on these selected codebook, we further re-train xLightFM based on former objective function of LightFM Eq. 4 to finetune the codebook matrices and indexes for better performance. Finally, the formal definition is illustrated as follows:

$$\hat{y}(\mathbf{x}) = \sum_{1 \le i < j \le n} \sum_{d} \left\langle \overline{C}_i^d \overline{b}_i^d, \overline{C}_j^d \overline{b}_j^d \right\rangle x_i x_j \tag{15}$$

In the inference stage of xLightFM, only the codebooks and indexes need to be stored, and the similarity between feature embeddings can be computed rapidly by the pre-defined lookup table. The former greatly reduces the memory footprint of xlightFM while the latter further reduces the inference time. The final framework for xLightFM is shown in Fig. 1 .

4 EXPERIMENT

In this section, we conduct experiments to verify the efficiency and effectiveness of our proposed **LightFM** and **xLightFM**.

4.1 Dataset

Two different datasets are utilized to validate the performances of proposed models. Table 1 details the statistics of these datasets. The **Criteo** dataset[3], an industry dataset available on Kaggle, is a collection of user clicks during one week for predicting ad click-through rate (CTR). We follow the data preprocessing version described in [47]. And the numerical features are normalized and converted into categorical features via bucketing. The second benchmark dataset is **Avazu**[4], which is also provided on Kaggle for CTR prediction. It contains user clicks over 11 consecutive days, indicating whether the user clicked the displayed mobile advertisement.

During our experiments, we split all the interaction data into three parts. The 80% part of interactions are sampled as the training set and the rest are equally divided into the validation and test set.

Dataset	#Interactions	#Feature Fields	#Features
Criteo	45,840,617	39	1,086,810
Avazu	40,428,968	22	2,018,012

Table 1: The summary of Datasets

4.2 Baselines

We compare our proposed method with the following competing lightweight factorization machine based approaches. The parameters are fine-tuned according to the AUC metric.

- **FM** [33], factorization machine, decodes the feature embeddings with same size for each feature field.
- **DFM** [25], discrete factorization machine, binarizes the real-valued embeddings into the binary codes, so as to store it efficiently and speed up the calculation of prediction.
- **QR** [35], Quotient-Remainder, utilizes the quotient and the remainder function to calculate two different embeddings and combines these two embeddings as the final embedding. This approach allows the embedding size to decrease from $O(|S|D)$ to $O(\sqrt{|S|}D)$, where $|S|$ denotes the number of the features and D denotes the embedding size. We set collection to 8.
- **MD** [7], mixed dimension embedding, is a memory-efficient algorithm for recommendation systems, which allocates different embedding sizes for different feature fields depending on the statistical frequency. The popular features embedding have a larger size depending on its heuristic search method.We set α to 0.2.
- **DHE** [16], deep hashing embedding, embeds the non-one-hot encodings into the dense and real-valued vectors via the multiple hashing. The final embeddings are then transformed by the deep neural networks. The dense encoding size is set to 256.

4.3 Evaluation Metrics

We test the algorithms on the three popular metrics, AUC, Logloss and Params. The AUC (Area Under the ROC Curve) awards the higher rank of the positive items (i.e. the samples with label 1) than the negative one. A higher AUC means a better performance of recommendation. The Logloss measures the likelihood of the user clicks depending on the prediction and the ground-truth labels, which consider the certainty. The Logloss is consistent with the objective function and a smaller Logloss is better. For the CTR prediction tasks, it is worth noticing that even a slight improvement is considered a significant promotion as it may lead to a remarkable increase in revenue. Apart from the common metrics for CTR prediction, we also pay attention to the memory cost of parameters, shorted as Params here, since the memory efficiency is the focus of our work. .Specifically, we define a float parameter to require 32 bits, a bool parameter to require 1 bit, and a codebook index to require $\log W$ bits, where W is the size of the codebook.And the cost of the linear part of factorization machine model is neglected here as they just constitute a small part.

4.4 Experimental Settings

The experiments are implemented in the Pytorch framework with NVIDIA 2080ti GPUs. Although the larger number of dimensions may lead to better performance, we set the dimension to 32 within the memory constraints. For the proposed **LightFM**, the number of codebooks for each feature field is set to 4 and there are 1,024 codewords for each codebook. And for the **xLightFM** ,the candidate codebook sizes are {64,128,256,512,1024,2048}. Those fields with fewer features than the number of codewords will be replaced by the original embedding. Finally, the codebook size ranges from 64 to 2048 for inference.

The batch size is set to 2,048. We utilize the Adam optimizer with the learning rate tuning with {0.0001, 0.0005, 0.0003, 0.001}. Finally, we set the learning rate 0.0003, 0.0005 for LightFM, xLightFM respectively. As for the architecture weight α, the temperature $\tau = max(0.001, 1 - 0.00003 * batch)$ is used for the Gumbel-softmax, where $batch$ is the number of training steps.

Regarding the **DFM**, the embeddings are first pre-trained via the basic FM models. And the discrete binary embeddings are learned from the real-value embedding through a two layers perceptions for each feature field and are utilized for inference.

4.5 Comparisons with Baselines

Experiments are conducted on the two datasets with all the competing baselines and we report the overall results in Table 2. From them, we have the following findings.

Finding 1: The compression of embeddings alleviate the massive burden of parameters and benefit improving the performance of recommendation. The number of parameters in vanilla FM is the greatest among all the compared approaches but almost all compression algorithms are better than it in AUC. The reason may lie in that the full embeddings result in the overfitting problem, where unpopular features are encoded into large embeddings. This brings the noise for training and harm the optimization. The superior performance of MDE among the baselines also demonstrates that it has great benefits for training to represent different fields with different amounts of information. The MDE heuristically allocates the appropriate dimensions for diverse feature fields, where the popular features

[3]https://www.kaggle.com/c/criteo-display-ad-challenge/
[4]https://www.kaggle.com/c/avazu-ctr-prediction/

		FM	DFM	QR	DHE	MD	LightFM	xLightFM
Avazu	AUC	0.7768	0.74005	0.7743	0.7709	0.7790	**0.7824**	0.7823
	LogLoss	0.3816	7.218	0.3832	0.3847	0.3802	**0.3788**	0.3792
	Params(MB)	246.3	7.70	30.86	23.72	22.74	10.66	**8.9125**
Criteo	AUC	0.7985	0.7692	0.7990	0.8040	0.8023	0.8055	**0.8057**
	LogLoss	0.4532	14.17	0.4530	0.4472	0.4484	0.4464	**0.4460**
	Params(MB)	132.7	4.14	16.74	40.34	17.95	7.38	**7.29**

Table 2: Comparisons with baselines

may be embedded with larger sizes for more efficient representation. Meanwhile, the unfrequent features are encoded into smaller sizes which reduces the memory consumption while preventing the overfitting problem. Again, xLightFM does not change the size of embeddings but instead tunes the number of codewords to achieve the same effect. As the number of codewords increases, the average distance between the centroid and the embedding becomes smaller. That is, more information is preserved with a larger codebook.

Finding 2: Our product quantization based algorithm performs better than the hashing based algorithm. Compared with the hashing based efficient methods, our proposed method LightFM and xLightFM lead to relative 3.60% and 3.59% performances respectively on average. And obviously, DFM achieves the best utilization of storage with the binary code embedding, but performs poorly due to the weak fitting ability of the binary codes. Although the hashing based models try their best to enhance the expressiveness from DFM to DHE, a lot of information is still discarded due to the limited representation of binary codes. On the contrary, our methods utilizing quantizers to reconstruct the vectors of embeddings with closer real-value centroids to preserve more information, as the distance between the original embedding and the reconstructed vectors are much smaller than binary codes. Moreover, the product quantization based algorithms are also memory efficient methods, since only the codebooks and the indices would be stored.

Finding 3: LightFM shows superior performances over all the algorithms with the effort of compressing the embeddings of each feature field. LightFM shows at least a 0.7% improvement over the other efficient methods, QR or MD, while the parameters only cost around 45% of memory space. LightFM associates feature with each other through multiple codebook interactions.In this way, features with low popularity are aggregated to train the clustering center to which they belong, thus ensuring higher accuracy and stronger generalization ability of the centroid. For example, the number of features in a field can reach hundreds of thousands, and multiple Codebooks connect these large numbers of features through clustering, so that they can conduct joint training on specific parameters, which has never been added in previous works.

Finding 4: xLightFM further reduces the memory consumption and shows comparable performance with LightFM. Although the LightFM shows superior performance, there is still room for a reduction in memory cost. Further, the xlightFM is proposed to search the size of codebooks for each feature field. Compared with the fixed size of codebooks in LightFM, xLightFM not only trains parameters for each embedding, but also intelligently searches for the size of codebooks within a specific memory constraint. xlightFM uses less than

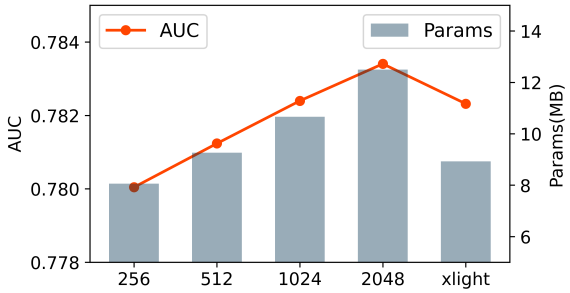

Figure 4: Comparisons between LightFM and xLightFM. The x-coordinate denotes the number of codewords in LightFM for the first 4 histograms.

20x less memory than vanilla FM, while the best performance of other baselines cost 10x less memory. With less parameter memory consumption, xlightFM still shows competitive performance.

4.6 Effectiveness & Efficiency of Automatic Codebooks Search

To test the efficiency of xLightFM, we design experiments with different numbers of codewords for the LightFM and compare them with xLightFM. The experiments are conducted on the Avazu dataset and we report the results of AUC and Params in Fig. 4.

Finding 1: When the number of the codewords increases, LightFM model performs better but results in a sharp increase in memory consumption. With the increasing number of codewords, the reconstructed vectors are closer to the original embedding, thus capturing more interactions, but large codebooks also increase storage consumption. As shown in the figure, when the number of codewords is doubled, the AUC only has an improvement of 0.001. This can be explained by the heuristic algorithm's inability to find the extraction requirements for every feature field, some of which are now assigned redundant sizes of codebooks. These redundant codewords are figured to make noise in training process, which leads to worse performances.

Finding 2: The xLightFM achieves the same level of AUC within remarkably fewer parameters. xLightFM only needs about 8.9MB parameters to achieve the AUC of 0.783 while LightFM achieves similar performance with 10.9MB of parameters, 1.2 times the cost of xLightFM. The memory cost in xlightFM is strictly limited and can be loosened by parameter, so the model learns to search for the most appropriate number of codewords for each feature field under the constraints. So xlightFM can get better performance under the same memory cost.

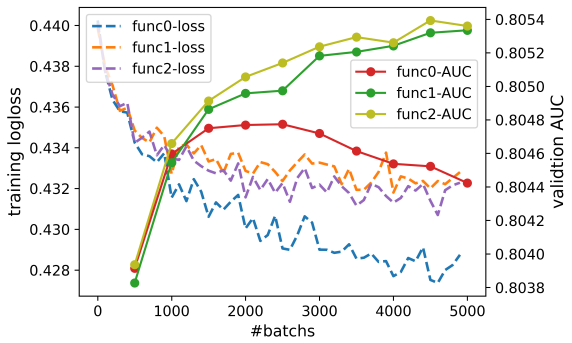

Figure 5: Curves of logloss and AUC w.r.t. loss functions

4.7 Scalability of the proposed approaches.

We exploit LightFM and xLightFM into the generic factorization machine based models to figure out the good scalability of our methods. The AUC results are detailed in Table 3. The experiments are conducted on the Avazu dataset. The number of codewords is set to 1,024 and the learning rate is set to 0.0003.

	DeepFM	NFM
FM(emb)	0.78202	0.77865
LightFM(emb)	0.78498	0.78308

Table 3: Scalability of the algorithm.

Finding: Regarding the famous factorization machine based models, our proposed memory-efficient methods are easy to extend into the models and perform well. LightFM performs well when applied in both models, achieving better performance than the normal model while costing less memory. This further illustrates that the full embeddings would result in the occurrence of noises and make it difficult for training.

4.8 Effects of different loss functions

To verify the effect of the designed loss function claimed in Section 5, we capture the curves of the logloss on the training set and AUC on the validation set during training, as shown in Figure 5. The func0 here indicates the original binary cross-entropy loss. The func1 refers to the version with the average distance while the func2 represents the distance loss which is weighted by the popularity.

Finding: The loss functions, introducing a popularity-weighted distance, help prevent overfitting and increase the performance of AUC. At the beginning of training, all the logloss decreases remarkably and the AUC improves quickly. However, the AUC performance on validation of the func0 drops a lot when more batches of data are fed into the model while the logloss on training set still decreases. It shows the binary cross-entropy loss only focuses on maximizing the likelihood function between the prediction and the ground truth but ignoring the quantization module which leads to serious overfitting. Introducing the distance loss penalizes the huge distances between the raw embeddings and the estimated vector and prevents the parameters from fluctuating greatly with the current batch data distribution during the training process. And the func2 performs better than the func1, demonstrating the rationality of the addition of popularities. This is because the popularity is correlated with the overall data distribution or feature distributions, and it can be concluded that better performances can be obtained if the distance loss contains more information about data distributions.

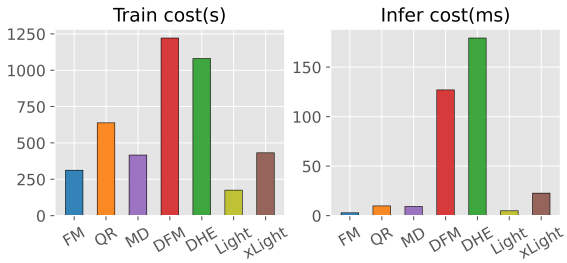

Figure 6: Times of Training and Inference

4.9 Time Efficiency

In this part, we further study the training and inference efficiency, which are another important indicator. The training time is obtained in the model training, while the inference time is the forward time of a batch with 2048 pieces of data. Each method is run 100 times, with the average time declared in Figure 6.

Finding: The proposed LightFM shows the best performance in time efficiency. The hashing based algorithms, DFM and DHE take the most time in both training and inference as they construct the deep neural networks. The QR method needs to calculate the quotient of the huge embeddings which takes additional time compared with FM and LightFM. As for xLightFM, it has to conduct computation of the structure parameters on the validation, so it takes more time than the LightFM. But xLightFM is still competitive compared with other baselines. In the aspect of inference, all the methods that do not involve deep network show sufficient competitiveness, and LightFM also shows its certain advantages because of the least number of parameters.

5 CONCLUSION

In this paper, we proposed an end-to-end quantization-based factorization machine named xLightFM, to extremely reduce the memory footprint of vanilla FM and maintain the lower quantization loss. Specifically, xLightFM decomposed each feature field into multiple codebooks, and directly learned the corresponding codeword assignments with loss of FMs and the Euclidean distances between compressed representation and original feature embedding. Furthermore, xLightFM adopted the Neural Architecture Search(NAS) techniques to adaptively allocate the most suitable number of codewords in codebooks of different fields. Extensive experiments on two datasets demonstrated the effectiveness and efficiency of xLightFM by comparing with state-of-the-art baselines, and achieve more than 18x and 27x memory compression compared to the vanilla FM. In the future work, we will explore more efficient NAS methods to further reduce the memory requirements and running time, so as to apply xLightFM to more general application scenarios.

ACKNOWLEDGMENTS

The work was supported by grants from the National Key R&D Program of China (No. 2020AAA0103800), National Natural Science Foundation of China (No. 61976198 and 62022077), JD AI Research and the Fundamental Research Funds for the Central Universities.

REFERENCES

[1] Artem Babenko and Victor Lempitsky. 2014. Additive quantization for extreme vector compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 931–938.

[2] Mathieu Blondel, Akinori Fujino, Naonori Ueda, and Masakazu Ishihata. 2016. Higher-order factorization machines. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*. 3359–3367.

[3] Ting Chen, Martin Renqiang Min, and Yizhou Sun. 2018. Learning K-way D-dimensional Discrete Codes for Compact Embedding Representations. In *International Conference on Machine Learning*. 853–862.

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

[5] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. 2007. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*. 271–280.

[6] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized product quantization for approximate nearest neighbor search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2946–2953.

[7] Antonio Ginart, Maxim Naumov, Dheevatsa Mudigere, Jiyan Yang, and James Zou. 2019. Mixed dimension embeddings with application to memory-efficient recommendation systems. *arXiv preprint arXiv:1909.11810* (2019).

[8] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. In *Proceedings of IJCAI'17*. AAAI Press, 1725–1731.

[9] Xiangnan He and Tat-Seng Chua. 2017. Neural factorization machines for sparse predictive analytics. In *Proceedings of SIGIR'17*. ACM, 355–364.

[10] Shoukang Hu, Sirui Xie, Hehui Zheng, Chunxiao Liu, Jianping Shi, Xunying Liu, and Dahua Lin. 2020. DSNAS: Direct Neural Architecture Search without Parameter Retraining. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12084–12092.

[11] Qiang Huang, Guihong Ma, Jianlin Feng, Qiong Fang, and Anthony KH Tung. 2018. Accurate and fast asymmetric locality-sensitive hashing scheme for maximum inner product search. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1561–1570.

[12] Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144* (2016).

[13] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2010), 117–128.

[14] Manas R Joglekar, Cong Li, Mei Chen, Taibai Xu, Xiaoming Wang, Jay K Adams, Pranav Khaitan, Jiahui Liu, and Quoc V Le. 2020. Neural input search for large scale recommendation models. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2387–2397.

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

[16] Wang-Cheng Kang, Derek Zhiyuan Cheng, Tiansheng Yao, Xinyang Yi, Ting Chen, Lichan Hong, and Ed H Chi. 2020. Deep Hash Embedding for Large-Vocab Categorical Feature Representations. *arXiv preprint arXiv:2010.10784* (2020).

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

[18] Defu Lian, Qi Liu, and Enhong Chen. 2020. Personalized ranking with importance sampling. In *Proceedings of The Web Conference 2020*. 1093–1103.

[19] Defu Lian, Rui Liu, Yong Ge, Kai Zheng, Xing Xie, and Longbing Cao. 2017. Discrete content-aware matrix factorization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 325–334.

[20] Defu Lian, Haoyu Wang, Zheng Liu, Jianxun Lian, Enhong Chen, and Xing Xie. 2020. Lightrec: A memory and search-efficient recommender system. In *Proceedings of The Web Conference 2020*. 695–705.

[21] Defu Lian, Yongji Wu, Yong Ge, Xing Xie, and Enhong Chen. 2020. Geography-aware sequential location recommendation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2009–2019.

[22] Defu Lian, Xing Xie, and Enhong Chen. 2019. Discrete Matrix Factorization and Extension for Fast Item Recommendation. *IEEE Transactions on Knowledge and Data Engineering* (2019). https://doi.org/10.1109/TKDE.2019.2951386

[23] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1754–1763.

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

[25] Han Liu, Xiangnan He, Fuli Feng, Liqiang Nie, Rui Liu, and Hanwang Zhang. 2018. Discrete Factorization Machines for Fast Feature-based Recommendation. In *Proceedings of IJCAI'18*. International Joint Conferences on Artificial Intelligence Organization, 3449–3455.

[26] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).

[27] Xianglong Liu, Junfeng He, Cheng Deng, and Bo Lang. 2014. Collaborative hashing. In *Proceedings of CVPR'14*. 2139–2146.

[28] Xiao Ma, Liqin Zhao, Guan Huang, Zhi Wang, Zelin Hu, Xiaoqiang Zhu, and Kun Gai. 2018. Entire space multi-task model: An effective approach for estimating post-click conversion rate. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. 1137–1140.

[29] Behnam Neyshabur and Nathan Srebro. 2015. On symmetric and asymmetric lshs for inner product search. In *International Conference on Machine Learning*. PMLR, 1926–1934.

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

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

[32] Steffen Rendle. 2012. Factorization machines with libFM. *ACM Trans. Intell. Syst. Tech.* 3, 3 (2012), 57.

[33] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of WWW'10*. ACM, 811–820.

[34] Ying Shan, T Ryan Hoens, Jian Jiao, Haijing Wang, Dong Yu, and JC Mao. 2016. Deep crossing: Web-scale modeling without manually crafted combinatorial features. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 255–262.

[35] Hao-Jun Michael Shi, Dheevatsa Mudigere, Maxim Naumov, and Jiyan Yang. 2020. Compositional embeddings using complementary partitions for memory-efficient recommendation systems. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 165–175.

[36] Anshumali Shrivastava and Ping Li. 2014. Improved asymmetric locality sensitive hashing (ALSH) for maximum inner product search (MIPS). *arXiv preprint arXiv:1410.5410* (2014).

[37] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*. 1–7.

[38] Jun Xiao, Hao Ye, Xiangnan He, Hanwang Zhang, Fei Wu, and Tat-Seng Chua. 2017. Attentional factorization machines: Learning the weight of feature interactions via attention networks. *arXiv preprint arXiv:1708.04617* (2017).

[39] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. 2018. SNAS: stochastic neural architecture search. In *International Conference on Learning Representations*.

[40] Xiao Yan, Jinfeng Li, Xinyan Dai, Hongzhi Chen, and James Cheng. 2018. Norm-ranging lsh for maximum inner product search. In *Advances in Neural Information Processing Systems*. 2952–2961.

[41] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative knowledge base embedding for recommender systems. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 353–362.

[42] Ting Zhang, Chao Du, and Jingdong Wang. 2014. Composite quantization for approximate nearest neighbor search. In *International Conference on Machine Learning*. PMLR, 838–846.

[43] Weinan Zhang, Tianming Du, and Jun Wang. 2016. Deep learning over multi-field categorical data. In *European conference on information retrieval*. Springer, 45–57.

[44] Yan Zhang, Defu Lian, and Guowu Yang. 2017. Discrete personalized ranking for fast collaborative filtering from implicit feedback. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 31.

[45] Yan Zhang, Haoyu Wang, Defu Lian, Ivor W Tsang, Hongzhi Yin, and Guowu Yang. 2018. Discrete ranking-based matrix factorization with self-paced learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2758–2767.

[46] Zhiwei Zhang, Qifan Wang, Lingyun Ruan, and Luo Si. 2014. Preference preserving hashing for efficient recommendation. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*. 183–192.

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

[48] Ke Zhou and Hongyuan Zha. 2012. Learning binary codes for collaborative filtering. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. 498–506.