LightRec: A Memory and Search-Efficient Recommender System

Defu Lian[1], Haoyu Wang[3], Zheng Liu[2], Jianxun Lian[2], Enhong Chen[1], Xing Xie[2]
[1]Anhui Province Key Laboratory of Big Data Analysis and Application, School of Computer Science and Technology & School of Data Science, University of Science and Technology of China
[2]Microsoft Research Asia
[3]University at Buffalo
{liandefu,cheneh}@ustc.edu.cn,{zheng.liu,jianxun.lian,xingx}@microsoft.com,hwang79@buffalo.edu

ABSTRACT

Deep recommender systems have achieved remarkable improvements in recent years. Despite its superior ranking precision, the running efficiency and memory consumption turn out to be severe bottlenecks in reality. To overcome both limitations, we propose LightRec, a lightweight recommender system which enjoys fast online inference and economic memory consumption. The backbone of LightRec is a total of B codebooks, each of which is composed of W latent vectors, known as codewords. On top of such a structure, LightRec will have an item represented as additive composition of B codewords, which are optimally selected from each of the codebooks. To effectively learn the codebooks from data, we devise an end-to-end learning workflow, where challenges on the inherent differentiability and diversity are conquered by the proposed techniques. In addition, to further improve the representation quality, several distillation strategies are employed, which better preserves user-item relevance scores and relative ranking orders. LightRec is extensively evaluated with four real-world datasets, which gives rise to two empirical findings: 1) compared with those the state-of-the-art lightweight baselines, LightRec achieves over 11% relative improvements in terms of recall performance; 2) compared to conventional recommendation algorithms, LightRec merely incurs negligible accuracy degradation while leads to more than 27x speedup in top-k recommendation.

KEYWORDS

Composite Encoding, Quantization, Online Recommendation, Recommender System

ACM Reference Format:
Defu Lian[1], Haoyu Wang[3], Zheng Liu[2], Jianxun Lian[2], Enhong Chen[1], Xing Xie[2]. 2020. LightRec: A Memory and Search-Efficient Recommender System. In *Proceedings of The Web Conference 2020 (WWW '20), April 20–24, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3366423.3380151

1 INTRODUCTION

Personalized recommendation is an important way to address information overload, by finding a short list of items with largest relevance scores. With the development of deep learning techniques, significant leap-forward has been made recently, particularly with those techniques on modeling high-order features [9, 12, 16, 19, 28, 32, 39], sequential behavior [40, 41, 50] and joint incorporation of text [20, 36], images [7, 14], and knowledge graph [38, 45].

However, one of the notable challenges for today's recommender system is running efficiency, as its scale is becoming far more magnificent than ever before. This challenge remains for the models ranging from the simplest matrix factorization to DSSM [20] and other matching networks [18, 25], which estimate relevance scores via inner product, Euclidean distance or cosine similarity between user representation and item representation. The representation is learned from their features, which can include unique id of users or items. Given D-dimensional representation of N items, the recommendation of the top-k results will incur a time complexity of $O(ND + k \log k)$. With approximate search approaches, such as hashing [43], PCATree [3], hierarchical metric tree [24], or product quantization [23], acceleration can be achieved for the recommendation calculation. However, the search index is usually constructed independently of the learning of recommendation system; and sometimes, user-item's relevance may be expressed in different ways from the distance metric used by the search index. Because of such an incompatibility, the recommendation accuracy will probably get impaired. For example, if user-item's relevance is expressed by inner product but the distance metric is based on Euclidean distance, two items with similar relevance scores may locate in different clusters since inner product is not a valid metric [18]. Apart from the running efficiency, the memory consumption is another unneglectable challenge, given that the size of representation parameters grows linearly with the ever-increasing scale of items. For example, if items are represented by 256-dim double-precision float vectors, 10 million items' representations will take over 9.5 GB of storage space, which is hard to be deployed into general devices with limited memory, such as GPU.

To address these challenges, we propose LightRec – a lightweight recommender system, which enjoys both fast online inference and economic memory consumption. The backbone of LightRec is a set of B codebooks, each of which is composed of W D-dimensional vectors, known as codewords. These codebooks serve as a basis of the latent space, with which items can be represented in a highly compact way. In particular, instead of representing an item with a D-dimensional float vector, we compose the most similar codeword within each codebook for representation. Since the binary indicator of whether a specific codeword is included can be compactly encoded by $\log W$ bits, the item is now merely represented with $B * \log W$ bits. As illustrated in the upper part of Fig. 1, where there are 4 codewords in each of 4 codebooks, each item is encoded with 2 bits in each codebook, and then fully represented with an 8-bit unsigned integer. By assuming vector length equals bit length, i.e.,

WWW '20, April 20–24, 2020, Taipei, Taiwan

ACM ISBN 978-1-4503-7023-3/20/04.
https://doi.org/10.1145/3366423.3380151

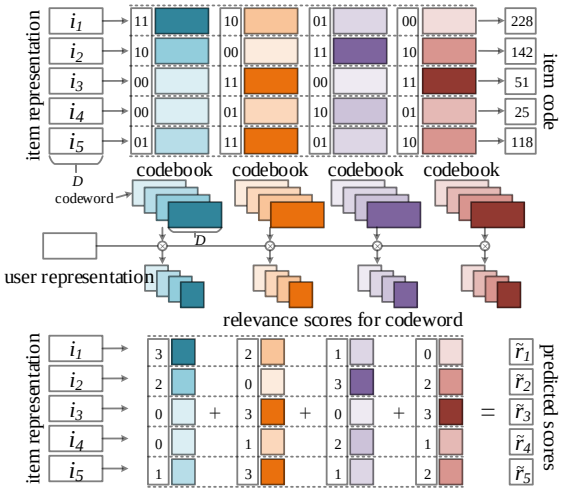


Figure 1: Illustration of Memory and Search Efficiency.

$D = B \log W$, the overall memory consumption of N items turns out to be $\frac{1}{8}ND + 4D^2W/\log W$ bytes. Back to the aforementioned case of 10 million items: by setting $W = 256$ the memory cost will drop to around 337 MB, which means almost 96.5% reduction. LightRec also greatly benefits the running efficiency. By storing each user's relevance scores for BW codewords in a table ($BW \ll N$), as shown in the lower part of Fig. 1, her relevance scores for items can be calculated via efficient table look-up, instead of float multiplication. Moreover, such an encoding scheme can be coordinated with inverted file system [23] to avoid exhaustive search.

However, it is very challenging to end-to-end train LightRec since codeword maximum selection is not differentiable and gradient can not be back-propagated. Noting that softmax is considered as a soft variant of the max function, we follow [6, 22, 29] to use tempered softmax to approximate maximum selection of codewords. To make similarity for selecting codewords compatible with the relevance function, we propose to parameterize the similarity, so that LightRec is capable of learning the similarity function from data. Another difficulty is how to ensure differences among B codebooks, so that composite representation can be as informative as possible. One may consider to impose diversity regularization [42] on these codebooks, but computational cost may be comparatively high, since diversity is required to compute between any two codebooks. Instead, LightRec proposes a recurrent mechanism to learn diversified codebooks. Concretely, the first codebook is used to approximate item representation, and the b-th codebook is used to approximate item residual representation, which subtracts composition of the first $b-1$ codewords from item representation. To further improve quality of composite representation, we employ several distillation strategies, which preserve either relevance scores or relative ranking order as much as possible. The contributions are summarized as follows:

- We propose LightRec – a lightweight recommender system, which enjoys both fast online inference and economic memory consumption. LightRec reduces memory cost of item representation by more than 96% and achieves over 27x speedup in top-k recommendation while leads to negligible accuracy degradation.
- We propose to parameterize similarity for codeword selection in LightRec, so that the similarity function is able to automatically learn from data. This makes the similarity function more compatible with user-item's relevance.
- We develop a recurrent relaxed optimization algorithm to end-to-end learn codeword assignments and diversified codebooks in LightRec, and employ several strategies of knowledge distillation to improve quality of composite representation.
- We extensively evaluate the proposed framework with four real-world datasets. The experimental results show that LightRec outperforms the state-of-the-art lightweight baselines by more than 11% in terms of recall performance. The experimental results also reveal the effectiveness of recurrent mechanism, knowledge distillation and side information.

2 RELATED WORK

2.1 Search-Efficient Recommendation

Search-efficient recommendation is organized into three categories.

The first taxonomy is hashing based recommendation, to learn binary representation for users and items. Learning can be data-independent [10, 21, 30, 35, 43], or from real-valued representation [18, 49, 51] or directly from data [26, 27, 46, 48]. Memory cost is very low since binary vector can be compactly stored into integers. User-item relevance based on both inner product and Euclidean distance can be quickly evaluated by bit-wise CPU operations (such as xor and popcount) over integers. Top-k recommendation can be approximately achieved in sublinear time via inverted multi-index [2, 31]. However, hashing based recommendation usually suffers from low capacity of representation and low accuracy of approximate recommendation.

The second taxonomy is index-based recommendation, which builds search index from real-valued representation. For example, Koenigstein et al. built a metric tree, a binary space-partitioning tree, from item representation [24, 33]. Instead of using beam search for sublinear-time recommendation, they established a new bound for maximum inner product search and used a simple branch-and-bound algorithm to efficiently obtain exact top-k recommendation. Metric tree can be hierarchically built by approximately picking a pair of pivot points farthest apart from each other, and assigning points to the closest pivot. Data points are recursively split until the number of points in the node exceeds a given threshold. Multiple metric trees were implemented in ANNOY[1] – a open-source library for similarity search, to improve accuracy of approximate search. Balanced binary tree can be also used for approximate search [52], by recursively applying balanced 2-means for data splitting until each node only contains one data point. Instead of defining node as the mean of data points, this work learned node representation from behavior data whose items are the descendant of this node. However, search index is built independently of representation learning and can not learned jointly with recommendation models in an end-to-end way. Moreover, the metric used for tree construction is based on Euclidean distance, being incompatible with user-item's relevance.

The last taxonomy is quantization-based recommendation, where each item is represented by a semi-structured vector. For example,

[1]https://github.com/spotify/annoy

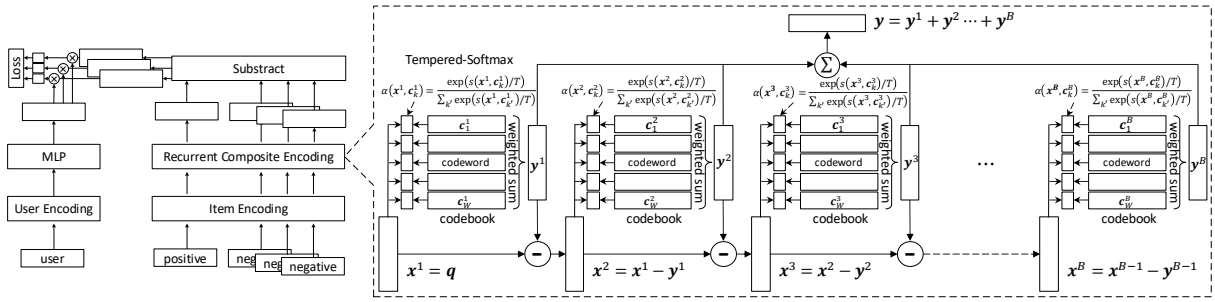


Figure 2: The framework of LightRec.

product quantization [23] equally divided representation space into several subspaces, and applied K-means for clustering items in each subspace, yielding a codebook of each subspace. Each item is then represented by concatenating center representation in all subspaces. Optimized product quantization [11] jointly learned space decomposition and subspace clustering. Composite Quantization [47] and Additive Quantization [1] did not decompose space but directly learned multiple codebooks in an iterative way. Residual vector quantization [8] learned codebooks from residual representation, which subtracts composition of learned codebooks from item representation. The work [13] extended PQ from Euclidean space to inner product space. However, they are based on either Euclidean distance or inner product, and leaned independently of representation learning. Therefore, these works suffers from similar issues to index-based recommendation. LightRec belongs to this taxonomy.

2.2 Deep Learning-based Recommendation

We organize the literature into two categories, but we don't consider sequence-based or session-based models. The first is to model side information including textual data [20, 36], visual data [7, 14], social networks [37], and knowledge graph [38, 45] via deep learning techniques. In spite of this, behavior data is still shallowly modeled based on inner product or Euclidean distance between user representation and item representation. Deep modeling is easily achieved by replacing them with a neural architecture. Such neural networks can be generalized to asymmetric siamese network and regularized by the gravity regularizer [25] for improving matching accuracy. The advantage of such models is easy to use the state-of-the art methods for approximate search. The second is to model feature interaction, since categorical features are more prevalence in recommender system while they suffer from sparsity. Representative work for explicit or implicit interaction modeling include Deep&Wide [9], product-based neural network [32], neural factorization machine [15], DeepFM [12], XDeepFM [28], Deep&Cross [39]. We focus on the first taxonomy among these work, since computing relevance scores can be more efficient. In the second taxonomy, memory is also straightforwardly reduced if we replace composite encoding for conventional embedding.

3 MEMORY AND SEARCH-EFFICIENT RECOMMENDER SYSTEMS

In this section, we will get started with the preliminaries and framework of LightRec; then elaborate the design of LightRec.

3.1 Preliminaries and Analysis

Let M denote the number of users, N denote the number of items. Reserve u to index user and i to index item. We will use the DSSM-like network for recommendation, due to the power of modeling side information, as illustrated in the left part of Fig. 2. In the DSSM-like network, relevance scores are based on user representation and item representation. Concretely, users are fed into user encoding module for obtaining user representation and items are fed into item encoding module for item representation. Both encoding modules can be a simple embedding module or the state-of-the-art module, such as CNN and LSTM, for modeling side information.

Let B denote the number of codebooks, W denote the number of codewords in each codebook, and $\boldsymbol{c}_w^b \in \mathbb{R}^D$ denote the w-th codeword in the b-th codebook. LightRec encodes items by composing the most similar codeword of each codebook. Concretely, an item's representation $\boldsymbol{q}_i \in \mathbb{R}^D$ is approximately encoded as follows:

$$\boldsymbol{q}_i \approx \sum_{b=1}^{B} \boldsymbol{c}^b_{w_i^b}, \text{ s.t. } w_i^b = \arg\max_w \mathrm{s}(\boldsymbol{q}_i, \boldsymbol{c}_w^b), \tag{1}$$

where $\mathrm{s}(\boldsymbol{q}_i, \boldsymbol{c}_w^b)$ is a similarity function to measuring the similarity between codeword and item representation. This is totally different from [6], where codeword assignments are derived only from item representation $\boldsymbol{q}_i$. The proposed method is thus more effective for codeword assignment, due to explicit relationship modeling between item representation and codeword. Moreover, the proposed method is better connected with Gaussian Mixture model (or K-means) according to Proposition 2.

In the DSSM-like network, we will use inner product rather than cosine similarity to estimate relevance scores in the model, since the computation can be dramatically accelerated in this case and gradient explosion due to reciprocal of norm can be easily prevented. Concretely, denoting $\boldsymbol{p}_u \in \mathbb{R}^D$ user representation outputted by MLP, the relevance score is defined as

$$\hat{r}_{ui} = \langle \boldsymbol{p}_u, \boldsymbol{q}_i \rangle \approx \sum_{b=1}^{B} \langle \boldsymbol{p}_u, \boldsymbol{c}^b_{w_i^b} \rangle. \tag{2}$$

In the testing stage, $\boldsymbol{q}_i$ is discarded and only codeword indices in each codebook, i.e. $[w_i^1, \cdots, w_i^B] \in \{1, \cdots, W\}^B$, are stored with the codebooks. Therefore, memory cost of N items is reduced from $4ND$ bytes to $\frac{1}{8}NB\log W + 4DBW$ bytes, where codeword index in each codebook is compactly encoded by $\log W$ bits. In practice, we assume code length of composite encoding is equal to the dimension of representation, i.e., $B \log W = D$, and set the number

of codewords $W = 256$. The memory cost of N items becomes $\frac{1}{8}ND + 128D^2$ bytes. For 10 million items, memory cost of item representation is reduced from over 9.5 GB to around 337 MB. Consider scoring all items for a user, time complexity is reduced from $O(ND)$ to $O(N+DBW)$, since the relevance scores are computed by looking up a table. The table stores user relevance scores for BW codewords, as illustrated in Fig 1. In practice, exhaustive search can be avoided via inverted file system [23], so only a part of items are required to score. Therefore, composite encoder is usually coordinated with the use of inverted file system.

Below we establish theoretical connection between composite encoding and generalized low-rank factorization of item presentation matrix. Note that the result is different from [6], whose approximation is based on Euclidean distance instead of inner product.

Proposition 1. *Composite encoding is equivalent to a generalized binary low-rank factorization of item representation matrix with respect to user representation matrix.*

The proof is provided in the appendix.

It is straightforward to first learn item representation in the network, and then feed them into vector quantization methods to learn codebooks and codeword assignments, but inner product is incompatible with Euclidean distance used for vector quantization. Moreover, user representation and item representation may be not best suitable for vector quantization. Therefore, it is especially important to end-to-end learn codebooks and codeword assignments.

3.2 Continuous Relaxation

Given the estimation of relevance scores $\tilde{r}_{ui} = \sum_{b=1}^{B} \langle \boldsymbol{p}_u, \boldsymbol{c}^b_{w_i^b} \rangle$, we can use them to formulate the loss function of recommendation. We can use any differentiable loss function. Here we adopt the commonly-used BPR loss in recommender system [34], which is defined as follows:

$$\mathcal{L} = \sum_{(u,i)} \sum_{j \notin I_u} -\log \sigma(\tilde{r}_{ui} - \tilde{r}_{uj}) + \lambda \|\Theta\|^2, \tag{3}$$

where $\sigma(x) = 1/(1+e^{-x})$. $\lambda\|\Theta\|^2$ is a regularization term to prevent overfitting. However, this objective function is non-differentiable due to the maximum selection operator in codeword assignment. Codeword selection in each codebook can be expressed by multiplication, as long as codeword index is transferred into an one-hot vector. Below we focus on discussion of the b-th codebook and drop the superscript b for concise representation. Assume $\boldsymbol{e}_i = \text{one-hot}(w_i)$ is one-hot representation of codeword index of item i in the b-th codebook, where $w_i = \arg\max_w \text{s}(\boldsymbol{q}_i, \boldsymbol{c}_w)$. Then the most similar codeword $\boldsymbol{c}_{w_i} = \boldsymbol{C}\boldsymbol{e}_i$, where $\boldsymbol{C} \in \mathbb{R}^{D \times W}$ is a codeword matrix in the b-th codebook. The maximum can be relaxed by tempered softmax, since softmax is a soft variant of the max function. Concretely, the w-th element of $\boldsymbol{e}_i$

$$e_i[w] \approx \tilde{e}_i[w] = \frac{\exp(\text{s}(\boldsymbol{q}_i, \boldsymbol{c}_w)/T)}{\sum_{w'} \exp(\text{s}(\boldsymbol{q}_i, \boldsymbol{c}_{w'})/T)}. \tag{4}$$

When $T \to 0$, the approximation becomes exact. Finally, the most similar codeword $\boldsymbol{c}_{w_i} \approx \boldsymbol{C}\tilde{\boldsymbol{e}}_i$. However, it is important to control temperature T, which should be neither too large nor too small.

To make the similarity function for codeword selection compatible with relevance function based on inner product, we learn the similarity function from the data, which is parameterized in a very general manner,

$$\text{s}(\boldsymbol{q}_i, \boldsymbol{c}_w) = \boldsymbol{q}_i^T \boldsymbol{W} \boldsymbol{c}_w + \langle \boldsymbol{w}_1, \boldsymbol{q}_i \rangle + \langle \boldsymbol{w}_2, \boldsymbol{c}_w \rangle, \tag{5}$$

which can be connected with Euclidean distance (when $\boldsymbol{W} = -2\boldsymbol{I}$, $\boldsymbol{w}_1 = \boldsymbol{q}_i$, $\boldsymbol{w}_2 = \boldsymbol{c}_w$), Mahalanobis Distance (when $\boldsymbol{W} = -2\boldsymbol{Z}$, $\boldsymbol{w}_1 = \boldsymbol{Z}\boldsymbol{q}_i$, $\boldsymbol{w}_2 = \boldsymbol{Z}\boldsymbol{c}_w$), scaled inner product (when $\boldsymbol{W} = 1/\sqrt{D}\boldsymbol{I}$, $\boldsymbol{w}_1 = 0$, $\boldsymbol{w}_2 = 0$) and inner product. Given the similarity function, the next proposition provides a good interpretation of Eq (4).

Proposition 2. *$\tilde{e}_i[w]$ can be considered as the posterior probability that item representation $\boldsymbol{q}_i$ belongs to cluster w, when $s(\boldsymbol{q}_i, \boldsymbol{c}_w) = -\|\boldsymbol{q}_i - \boldsymbol{c}_w\|^2$.*

The proof is provided in the appendix. In this case, if reconstruction error $\|\boldsymbol{q}_i - \boldsymbol{C}\tilde{\boldsymbol{e}}_i\|$ is considered as the objective function, this is equivalent to learning Gaussian mixture model.

Such a general similarity function leads to the following useful properties, which help us to derive a concise equation for recurrent composite encoding in next subsection.

Proposition 3. *Both $\boldsymbol{e}_i$ and $\tilde{\boldsymbol{e}}_i$ are invariant to codebook scaling.*

The proof is provided in the appendix. In other words, multiplying codebook with a positive number does not alter $\boldsymbol{e}_i$ and $\tilde{\boldsymbol{e}}_i$.

Till now, the continuous relaxation $\tilde{\boldsymbol{e}}_i$ is used to approximate $\boldsymbol{e}_i$ for back-propagation. When estimating relevance scores, we should directly use $\boldsymbol{e}_i$ instead of $\tilde{\boldsymbol{e}}_i$, to be consistent with online recommendation. To close the gap between forward pass and backward pass, we follow a similar idea to Straight-Through Estimator [4], to rewrite $\mathbf{e}_i$ as follows

$$\hat{\mathbf{e}}_i = \tilde{\boldsymbol{e}}_i + \text{stop\_gradient}(\mathbf{e}_i - \tilde{\boldsymbol{e}}_i), \tag{6}$$

where the stop_gradient operator will prevent the gradient from back-propagating through it. In the foward pass, stop_gradient does not take affect, $\hat{\mathbf{e}}_i = \mathbf{e}_i = \text{one-hot}(\arg\max_w \text{s}(\boldsymbol{q}_i, \boldsymbol{c}_w))$. One one hand, this indicates only the most similar codeword in each codebook is always used for inference, i.e., estimating relevance scores and loss values. On the other hand, the temperature $T \to 0$ is set in the forward pass. In the backward pass, stop_gradient takes affect, so $\nabla_{\hat{\boldsymbol{e}}_i} \mathcal{L} = \nabla_{\tilde{\boldsymbol{e}}_i} \mathcal{L}$. Since a larger T is usually used, gradient can back-propagate through it. Therefore, based on tempered softmax and stop_gradient tricks, this framework is capable of end-to-end learning composite representation.

3.3 Recurrent Composite Encoding

After continuous relaxation, the network can be end-to-end trained by any available solvers like SGD and ADAM. Since composite encoding directly sums the most similar codeword in each of B codebooks, codebooks should be distinguished from each other. If codebooks are exactly the same, only one of B codebooks is useful. Random initialization to centers may take some effect, just as K-means algorithm can produce different clustering results when selecting different center points. A more appealing way is to design some strategies to ensure their differences. Note that composite encoding can be considered as cluster ensemble, a straightforward way is to impose diversity regularizer (e.g. squared Frobenius norm, von Neumann divergence, or log-determinant divergence [42]) over

codebooks, since diversity of base learners in ensemble learning improves generalization error [44]. However, since regularizer should be imposed between any two codebooks, there are total $\frac{B(B-1)}{2}$ terms in the regularizer. The time cost may be comparatively high. Moreover, these regularizers are usually vulnerable to permutation of codewords. In other words, if one codebook is composed of codewords in another codebook with permutation, the diversity regularization is non-zero. The larger the regularization is, the more diverse codebooks are.

In this paper, we propose Recurrent Composite Encoding to learn diversified codebooks. Concretely, as illustrated in Fig. 2, the codebooks are sequentially learned, where the b-th codebook is learned from item residual representation, which subtracts composition of the first $b-1$ codewords from item representation. Let $\boldsymbol{y}^b$ be the most similar codeword in the b-th codebook, i.e., $\boldsymbol{y}^b = \boldsymbol{c}^b_{w^b}$, and $Q(\boldsymbol{x}|\boldsymbol{C}^b) : \mathbb{R}^D \rightarrow \mathbb{R}^D$ be a function to quantize $\boldsymbol{x}^b = \boldsymbol{x}^{b-1} - \boldsymbol{y}^{b-1}$ with codebook $\boldsymbol{C}^b$, such that $\boldsymbol{y}^b = Q(\boldsymbol{x}^b|\boldsymbol{C}^b)$. The recursive equation can be unfolded as follows

$$\boldsymbol{y}^b = Q(\boldsymbol{x}^1 - \sum_{t=1}^{b-1} \boldsymbol{y}^t | \boldsymbol{C}^b), \tag{7}$$

where $\boldsymbol{x}^1 = \boldsymbol{q}$ is item representation, as input of the recurrent composite encoding module. Recurrent composite embedding can be well interpreted by the following proposition.

Proposition 4. *Recurrent composite encoding learns the encoding function by following gradient descent of reconstruction error with respect to the encoding function.*

The proof is provided in appendix. Note that weighting each codebook is not necessary any more due to the absorption of weights into codebooks according to Proposition 3.

Based on this result, recurrent composite encoding is connected with gradient boosting, so it can provides diversified codebooks for quantization functions. Significantly different from gradient boosting, recurrent composite encoding is trained in an end-to-end manner, and much easier to extend. For example, since input to the b-th quantizer is $\boldsymbol{x}^b = \boldsymbol{x}^{b-1} - \boldsymbol{y}^{b-1}$, it can be generalized to $\boldsymbol{x}^b = \phi(\boldsymbol{x}^{b-1}, \boldsymbol{y}^{b-1})$. Therefore, this can be intuitively implemented by RNN, where $\boldsymbol{y}^{b-1}$ is input and $\boldsymbol{x}^{b-1}$ is state. This is also the reason that this method is called recurrent composite encoding. We will pay special attention to the following two cases. The one is $\boldsymbol{x}^b = \boldsymbol{x}^{b-1} - \boldsymbol{W}\boldsymbol{y}^{b-1}$ and the other one is $\boldsymbol{x}^b = \boldsymbol{W}_1\boldsymbol{x}^{b-1} - \boldsymbol{W}_2\boldsymbol{y}^{b-1}$. These two cases may better align codewords with item residual representation. They are especially useful when codeword and item representation are not of the same dimension.

3.4 Knowledge Distillation

Though LightRec can be end-to-end trained by adjusting parameters according to noisy gradient from the end task, it fails to well learn item representation and representation network (when side information is available). The main reason is that LightRec does not understand the goodness of item representation but only the goodness of item's composite representation. It is possible to add a shortcut before and after recurrent composite encoding to allow the end task to guide the learning of item representation [6], however, the quality of representation is still not as good as expected. In this part, we focus on how to distill knowledge from pre-trained representation, where the basic idea is to enforce composite representation to mimic pre-trained item representation. Note that representation can be simultaneously learned well with composite encoder as long as item representation and item's composite encoding are respectively fed into different loss functions for multi-task learning. However, this is independent of the design of knowledge distillation, and thus left for future work.

As illustrated in Fig. 2, let $\boldsymbol{q}_i$ and $\boldsymbol{y}_i$ denote input and output of recurrent composite encoding, respectively. The first strategy is to minimize reconstruction error, i.e., the ℓ_2 distance between them

$$\mathcal{L}_{E-KD} = \|\boldsymbol{q}_i - \boldsymbol{y}_i\|^2. \tag{8}$$

During training, $\mathcal{L}_{E-KD}$ is added into the loss $\mathcal{L}$ to train jointly.

Since relevance score is estimated based on inner product, it is more appealing to preserve inner product, that is

$$\mathcal{L}_{IP-KD} = \|\boldsymbol{P}_1\boldsymbol{q}_i - \boldsymbol{P}_2\boldsymbol{y}_i\|^2 \tag{9}$$

where $\boldsymbol{P}_1^T$ and $\boldsymbol{P}_2^T$ are user representation of all users. The reason to use two distinct user representations lies in significant differences between real-valued vector representation and composite representation and its superiority to a single user representation according to our empirical observation. However, to reduce the number of parameters, we can derive $\boldsymbol{P}_2$ from $\boldsymbol{P}_1$ via MLP, as illustrated in Fig. 2. To reduce the time complexity, which is now in linear proportion to the number of users, we rewrite the loss:

$$\mathcal{L}_{IP-KD} = \begin{bmatrix} \boldsymbol{q}_i^T & \boldsymbol{y}_i^T \end{bmatrix} \begin{bmatrix} \boldsymbol{P}_1^T\boldsymbol{P}_1 & -\boldsymbol{P}_1^T\boldsymbol{P}_2 \\ -\boldsymbol{P}_2^T\boldsymbol{P}_1 & \boldsymbol{P}_2^T\boldsymbol{P}_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{q}_i \\ \boldsymbol{y}_i \end{bmatrix}. \tag{10}$$

As long as we cache $\boldsymbol{P}_1^T\boldsymbol{P}_1$, $\boldsymbol{P}_1^T\boldsymbol{P}_2$ and $\boldsymbol{P}_2^T\boldsymbol{P}_2$, and dynamically update them when $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ are updated, the computational cost can be reduced to $O(D^2)$. The techniques for caching and updating these matrices can refer to [25] and are not elaborated any more.

The last strategies is to preserve ranking order of items with respect to a specific user. However, this task is very challenging, one one hand, because the order is discrete and not differentiable, on the other hand because there are so many items available. Preserving relative order of top-k preferred items via pairwise/listwise ranking losses seems a good and efficient method. However, we observe that such losses are very difficult to reduce. Note that we use SGD for optimizing LightRec, where for each positive item of a user, we randomly sample a fixed number of items as negative. Therefore, we propose to preserve the relative order of the negative samples. Denoting these L negative samples by $j_1, \cdots j_L$, ranking preservation loss is formulated by following a list-wise ranking loss [5]:

$$\mathcal{L}_{R-KD} = -\sum_l \frac{\exp(\langle \boldsymbol{p}_{1u}, \boldsymbol{q}_{j_l} \rangle / T)}{\sum_{l'} \exp(\langle \boldsymbol{p}_{1u}, \boldsymbol{q}_{j_{l'}} \rangle / T)} \log \frac{\exp(\langle \boldsymbol{p}_{2u}, \boldsymbol{y}_{j_l} \rangle / T)}{\sum_{l'} \exp(\langle \boldsymbol{p}_{2u}, \boldsymbol{y}_{j_{l'}} \rangle / T)} \tag{11}$$

where $\frac{\exp(\langle \boldsymbol{p}_{1u}, \boldsymbol{q}_{j_l} \rangle / T)}{\sum_{l'} \exp(\langle \boldsymbol{p}_{1u}, \boldsymbol{q}_{j_{l'}} \rangle / T)}$ is considered the top one probability of the item j_l, being ranking on the top, and cross entropy is used to measure difference between two probability distributions. And $\boldsymbol{p}_{1u}$ and $\boldsymbol{p}_{2u}$ are the u-th row of matrices $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ respectively. The objective function is usually used together with $\mathcal{L}_{IP-KD}$, and both of them are added into the task specific loss $\mathcal{L}$ for training.

4 EXPERIMENTS

In this part, we firstly compare the proposed LightRec with baselines, including state-of-the-art lightweight algorithms, and then study the effect of modeling side information and post-ranking. Following that, we conduct ablation study for investigating effectiveness of each component in LightRec. Finally, we show efficiency improvement and memory reduction of LightRec due to composite representation.

4.1 Experiment Settings

4.1.1 Dataset. We use four public real-world datasets, including *CiteULike*, *Gowalla*, *Amazon* and *MovieLens*, to evaluate the proposed algorithm. Because Amazon and MovieLens are explicit feedback data, we set ratings higher than 3 as positive samples, to convert them into implicit feedback data. In addition, due to the extreme sparsity of the datasets, we filter users who have less than 3 ratings. Table 1 summarizes the filtered datasets. For each user, we randomly sample 80% positive samples as training and the remaining as test. 10% of training data are held-out for validation. We repeat five random splits and report the averaged results.

Table 1: Statistics of datasets

Dataset	#User	#Item	#Rating	Density
CiteULike	7,947	25,975	107,154	0.052%
Gowalla	29,858	40,988	822.358	0.067%
Amazon	157,465	128,939	3,031,673	0.015%
MovieLens	69,662	8,939	4,702,571	0.755%

4.1.2 Comparison Methods. We compare the proposed LightRec and a variant LightRec+, which incorporates textual information, with the following baselines:

- **BPR** [34], as the upper bound of LightRec, is a matrix factorization method based on the Bayesian Personalized Ranking loss. It has been one of the most widely-used methods for implicit feedback data. It is implemented in the tensorflow framework and optimized by ADAM. For each positive item of a user, we randomly sample a fixed number of unobserved items as negative.
- **KDE** [6], K-way D-dimensional Encoding, can be considered the state-of-the-art composite encoding to end-to-end train, but it is not specially designed for recommender system. Its codeword selector is only based on item representation, and no constrains are imposed on codebooks.
- **RVQ** [8], Residual Vector Quantization, quantizes residual item representation each time via K-means clustering by minimizing reconstruction error. This algorithm is not end-to-end trained but a two-stage solution.
- **OPQ** [11], Optimized Product Quantization, jointly optimizes space decomposition and reconstruction error. It is an improved version of Product Quantization [23], and a two-stage solution.
- **DCMF** [26], the state-of-the-art hashing-based collaborative filtering with side information. Binary codes are directly learned from the end task, similar to LightRec, but representation space is smaller than LightRec.
- **DSSM** [20], as the upper bound of LightRec+, is a deep structured semantic model for CTR prediction and recommendation with textual information. DSSM maps bag-of-words representation with word hashing to latent semantic vector via multilayer perceptron. For fair comparison, we also leverage inner product to estimate relevance scores and optimize the BPR loss.

Remarks: Here we only include BPR and DSSM as the baselines for conventional recommendation models because BPR and DSSM bounds LightRec and LightRec+ from the above in terms of model capacity, and the margin between them exactly tells information loss due to composite representation. Other advanced algorithms can not be fairly compared, so they are not included.

4.1.3 Evaluation Metric. Since a practical recommender system is to perform top-k recommendation, we retrieve the top-k preferred items for each user based on relevance scores from all unobserved items and evaluate the top-k ranking list against positive items in the test set. We use two common metrics in recommender system – NDCG and Recall. Recall@k, recall at a cutoff k, is the proportion of positive items in the top-k ranking list over the total number of positive items in the test set. NDCG@k, NDCG at a cutoff k, rewards methods that ranks positive items at the top of the ranking. The positive items at bottom positions of the ranking list contribute less to the final score than that at the top positions. NDCG (i.e., without a cutoff) indicates an average of NDCG@N_u, where N_u denotes the number of positive items of user u in the test set. Recall does better in assessing item recalling while NDCG is good at measuring how well "ground-truth" items are ranked. We will report NDCG, Recall@100 and NDCG@k, where k ranges from 10 to 100.

4.1.4 Parameter Settings. The dimension of representation space is set to $D = 32$ by default. The number of codewords in each codebook is set to $K = 256$, such that codeword indexes are compactly encoded by 1 byte. The number of codebooks is set to 4, such that binary code length just equals to 32 bits. For DCMF, we tune α, λ on the validation set by grid search over {1e-4,1e-3,1e-2,1e-1,1e0,1e1,1e2} and β on the validation set by grid search over {1e-6,1e-5,1e-4,1e-3,1e-2,1e-1,1e0}. For OPQ, each subspace is of dimension 8, and clustered into 256 groups. For BPR, KDE, LightRec and the variant, we set batch size 1024, learning rate 0.001 and regularization coefficient 0.001 on all datasets. For every positive item of a user, we randomly sample 5 negative samples from unobserved items. Though temperature schedule is very important for approximating the max function with tempered softmax, we find that the temperature lower than 1 performs well. Therefore, we set the temperature in Eq (4) to 0.9. The temperature in Eq (11) is set 0.5 by following [17]. The coefficient of knowledge distillation losses is tuned on the validation set by grid search over {1e-10,1e-9,1e-8,1e-7,1e-6,1e-5,1e-4}.

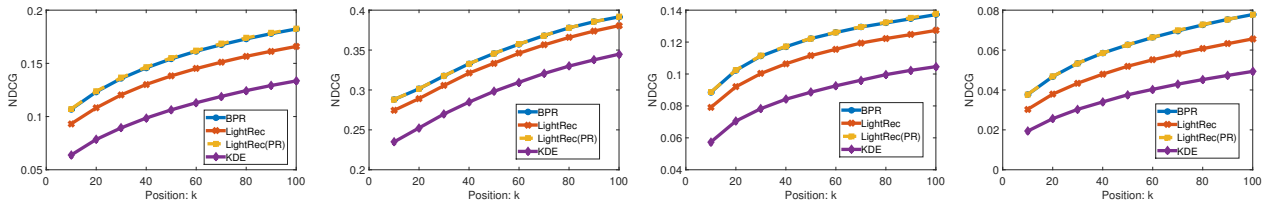
4.2 Comparison with Baselines

Table 2 and Fig. 3 show the results of comparison with baselines. From them, we have following important findings.

- *LightRec dramatically outperforms the state-of-the-art lightweight baselines in terms of recall performance.* On average, LightRec has more than 11.56% improvements with respect to Recall@100 and over 6.59% improvements with respect to NDCG. Note that Recall is a better metric of the item recalling task, so the improvements are remarkable. In more details, the relative improvements to the

Table 2: Comparison with baselines in the four datasets.

	CiteULike		Gowalla		Amazon		MovieLens10M		
	Recall@100	IMP(%)	Recall@100	IMP(%)	Recall@100	IMP(%)	Recall@100	IMP(%)	AVG_IMP(%)
BPR	0.3288	-4.47	0.3653	-6.30	0.1983	-13.36	0.6194	-2.34	-6.62
KDE	0.2846	+10.37	0.3024	+13.19	0.1371	+25.30	0.5711	+5.92	+13.70
RVQ	0.2845	+10.40	0.3198	+7.04	0.1437	+19.55	0.5537	+9.25	+11.56
OPQ	0.2799	+12.22	0.3050	+12.23	0.1444	+18.98	0.5610	+7.83	+12.82
DCMF	0.2181	+44.02	0.2507	+36.54	0.1213	+41.63	0.4510	+34.12	+39.08
LightRec	0.3141	-	0.3423	-	0.1718	-	0.6049	-	-
	CiteULike		Gowalla		Amazon		MovieLens10M		
	NDCG	IMP(%)	NDCG	IMP(%)	NDCG	IMP(%)	NDCG	IMP(%)	AVG_IMP(%)
BPR	0.2237	-3.44	0.3065	-4.01	0.1958	-4.90	0.5061	-1.52	-3.46
KDE	0.1965	+9.92	0.2678	+9.85	0.1722	+8.13	0.4714	+5.73	+8.41
RVQ	0.2014	+7.25	0.2505	+5.45	0.1748	+6.52	0.4652	+7.14	+6.59
OPQ	0.1995	+8.27	0.2653	+10.89	0.1748	+6.52	0.4542	+9.73	+8.85
DCMF	0.1745	+23.78	0.2509	+17.26	0.1661	+12.10	0.3902	+27.73	+20.22
LightRec	0.2160	-	0.2942	-	0.1862	-	0.4984	-	-



(a) Gowalla (b) MovieLens10M (c) CiteULike (d) Amazon

Figure 3: The performance of LightRec with Post Ranking, i.e. LightRec(PR).

state-of-the-art end-to-end composite encoding are up to 13.70% on average with respect to Recall@100 while the relative improvements to the state-of-the-art hashing-based recommendation are up to 39.08%.

- *End-to-end codebook learning leads to superior performance of item recalling.* This can be observed by comparing LightRec with RVQ, since RVQ also quantizes item residual representation. The main difference between them lies in the end-to-end framework, which enables to plug in a general similarity function so as to better align with the relevance function based on inner product.
- *Recurrent mechanisms can remarkably promote the quality of codebooks.* This is based on the superiority of LightRec to KDE. We know that both of them are end-to-end trained, but LightRec uses the recurrent mechanism to learn multiple codebooks while KDE purely depends on random initialization. Though they are also different in modeling codeword selection, the recurrent mechanism is the main difference. This can also be observed in later ablation study.
- *Hashing-based recommendation can not show comparable performance to quantization-based recommendation.* This is indicated by the worst performance of DCMF, though it is the state-of-the-art algorithm to learn binary codes from the end task. This is mainly because representation capacity of binary Hamming space is much smaller than semi-structured vector space used in LightRec. Therefore, when mapping item presentation into binary Hamming space, much more information will be lost, and recommendation performance will be degraded more.
- *Without post ranking, LightRec shows comparatively worse performance than BPR, and the gap between them is very small.* How to narrow the performance gap between recommendation algorithm and the approximation is a key research problem. This is achieved by increasing representation capacity by replacing Hamming space with semi-structured vector space, and improving the quality of approximate representation by the use of recurrent mechanism, a general yet compatible similarity function and knowledge distillation. Finally, without post reranking, the performance degradation of LightRec is only 6.62% with respect to Recall@100 on average and 3.46% with respect to NDCG.
- *Recommendation models are more precisely approximated in the denser datasets.* This is based on the observation that LightRec is best trained in the densest MovieLens dataset, leading to the smallest margin from BPR. However, in the sparsest Amazon dataset, the margin between LightRec and BPR is the largest. The possible reason is that there are more useful interactions between user and item in the denser dataset, from which both codebook and representation can be better learned.

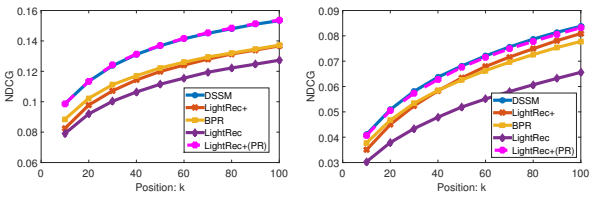
Table 3: The effect of features for LightRec.

	Recall@100		NDCG	
	CiteULike	Amazon	CiteULike	Amazon
BPR	0.3288	0.1983	0.2237	0.1958
DSSM	0.3726	0.2097	0.2364	0.2017
LightRec	0.3141	0.1718	0.2160	0.1862
LightRec+	0.3490	0.2198	0.2227	0.2048

4.3 Effectiveness of Side Textual Information

4.3.1 Settings. Following the comparison with baselines, we study the power of modeling textual information in LightRec, to understand the effect of side information for learning composite representation. This variant, denoted by LightRec+, is evaluated with the CiteULike and Amazon datasets, both of which is provided with abundant reviews. We follow DSSM to process review text. The generated semantic vector representation is concatenated with item embedding for final item representation. Table 3 and Fig. 4 show the results of comparison.

4.3.2 Findings – Side information leads to improvements of both item recalling and item ranking. This is indicated by the superiority of DSSM to BPR and the superiority of LightRec+ to LightRec. In more details, the improvements for item ranking are 13.32% with respect to Recall@100 and 5.68% with respect to NDCG in the CiteUlike dataset, while item recalling is improved by 11.11% with respect to Recall@100 and 3.10% with respect to NDCG. The improvements in item recalling are comparatively smaller than item ranking. In the Amazon dataset, LightRec is improved by 27.94% with respect to Recall@100 and 9.99% with respect to NDCG, while DSSM only outperforms BPR by 5.75% with respect to Recall@100 and 3.01% with respect to NDCG. The improvements of item recalling are surprisingly much larger than item ranking. Noting that the Amazon dataset is much sparser, this result can be explained by the role of regularization which composite encoding plays, since DSSM is more likely to overfit in the sparse dataset. It is worth mentioning review texts are only processed simply, so these results well illustrate the great flexibility of LightRec.



(a) CiteULike (b) Amazon

Figure 4: The effect of features for LightRec with Post Ranking, i.e., LightRec+(PR).

4.4 Study of Post-Ranking

4.4.1 Settings. LightRec learns composite encoding for items, with the aims of fast recalling potentially positive items. The ranking of these candidate items can be improved by post-ranking according to exact recommendation models. In this part, we investigate the effect of post ranking. Concretely, we use LightRec (LightRec+) to firstly recall top-500 items and leverage BPR (DSSM) to post-rank these items. To better understand how well these items are ranked, particularly in the first few positions, we only show NDCG@k in Fig. 3 and Fig. 4, where k ranges from 10 to 100.

4.4.2 Findings – Post-ranking dramatically improves ranking of candidate items recalled by LightRec/LightRec+; With post-ranking, LightRec/LightRec+ only achieves negligible performance degradation. The first finding is evidenced by the superiority of LightRec(PR) to LightRec and KDE, and the superiority of LightRec+(PR) to LightRec+. The second finding is indicated by observations that the yellow (purple) dashed line are almost coincident with the blue line in Fig. 3 (Fig. 4). Both findings imply that candidate items recalled by LightRec include most positive items returned by BPR, but not in a completely accurate order. More advanced algorithms should further improve recommendation accuracy, since their performance bounds that of LightRec from the above. This is better implied in Fig. 4, where LightRec+(PR) does not surpass DSSM while LightRec+ recalls more positive items than DSSM according to Table 3. Another interesting observation is that without side information, LightRec benefits more from post-ranking in sparser dataset like Amazon. This is evidenced in Fig. 3 by the largest improvements in the Amazon dataset due to post-ranking. In other words, LightRec easily suffers from sparsity issue. When side information is available, LightRec+ alleviates the sparsity issue and even prevents it to overfit, as shown in right figure of Fig. 4 and Table 3.

4.5 Effectiveness of Recurrent Mechanisms

One key idea of LightRec is to exploit recurrent mechanisms to ensure codebook divergence. Therefore, in this part, we study the effect of recurrent mechanisms in LightRec, by considering the following three recurrent modeling methods: 1) $\boldsymbol{x}^b = \boldsymbol{x}^{b-1} - \boldsymbol{y}^{b-1}$; 2) $\boldsymbol{x}^b = \boldsymbol{x}^{b-1} - \boldsymbol{W}\boldsymbol{y}^{b-1}$, where $\boldsymbol{W}$ is a learnable matrix to align codeword with residual representation; 3) $\boldsymbol{x}^b = \boldsymbol{W}_1\boldsymbol{x}^{b-1} - \boldsymbol{W}_2\boldsymbol{y}^{b-1}$, where $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ are learnable matrices. Note that if $\boldsymbol{x}^b = \boldsymbol{x}^{b-1}$, then $\boldsymbol{x}^b = \boldsymbol{x}^1 = \boldsymbol{q}$. This is a non-recurrence based method in the state-of-the-art end-to-end composite encoding [6]. The results of study in the CiteULike and Gowalla datasets are shown in Table 4, and the findings are summarized as follows.

- *Recurrent mechanism remarkably improves the performance of LightRec in recalling candidate items.* This can be observed by comparing the first two rows in the table. The improvements are more than 15% with respect to Recall@10 and more than 10% with respect to NDCG in both datasets. This implies the significant effect of ensuring the diversity of codebooks. Therefore, the proposed recurrent mechanism is not only theoretically guaranteed, but also empirically well-performing, playing an important role in learning LightRec.
- *Three methods for modeling recurrence are not significantly different from each other.* The main reason lies in the additive composition of codewords, so that only the first method is best suitable for recurrence modeling. However, other two methods are useful when codewords and representation are not of the same dimension or even not in the same type of space. For example,

codewords are in Hamming space while representation are in real space. When composition is based on more complex functions, such as LSTM, they should also be useful, but may require nonlinear transformation.

Table 4: Effectiveness study of recurrent mechanisms in the CiteULike and Gowalla datasets.

$\boldsymbol{x}^b = \phi(\boldsymbol{x}^{b-1}, \boldsymbol{y}^{b-1})$	Recall@100		NDCG	
	CiteULike	Gowalla	CiteULike	Gowalla
$\boldsymbol{x}^{b-1}$	0.2718	0.2956	0.1961	0.2616
$\boldsymbol{x}^{b-1} - \boldsymbol{y}^{b-1}$	0.3141	0.3423	0.2160	0.2942
$\boldsymbol{x}^{b-1} - \boldsymbol{W}\boldsymbol{y}^{b-1}$	0.2911	0.3426	0.2067	0.2938
$\boldsymbol{W}_1\boldsymbol{x}^{b-1} - \boldsymbol{W}_2\boldsymbol{y}^{b-1}$	0.2934	0.3375	0.2066	0.2910

4.6 Effectiveness of the Similarity Functions

The other key idea of LightRec is the general similarity function, to be compatible with the relevance function. Since the proposed similarity function is a variant of bilinear, it is denoted by Bilinear in this part. To investigate its effect, we compare it with representation-only score $s(\boldsymbol{q}, \boldsymbol{c}) = f(\boldsymbol{q})$; dot product based similarity: $s(\boldsymbol{q}, \boldsymbol{c}) = \langle \boldsymbol{q}, \boldsymbol{c} \rangle$; scaled inner product based similarity: $s(\boldsymbol{q}, \boldsymbol{c}) = 1/\sqrt{D}\langle \boldsymbol{q}, \boldsymbol{c} \rangle$, where D is the dimension of $\boldsymbol{q}$; and Euclidean distance based similarity: $s(\boldsymbol{q}, \boldsymbol{c}) = -||\boldsymbol{q} - \boldsymbol{c}||_2^2$. Table 5 reports the results, from which we have the following findings.

- *Codeword selector based on similarity selects better codewords than that purely based on item representation.* This is evidenced by the inferiority of representation-only score to most of other similarity. In the best case, the improvements are more than 10.4% with respect to Recall@100 in the CiteULike dataset and more than 6.4% in the Amazon dataset. This reveals the importance of modeling relationship between item representation and codewords for selecting better codewords.
- *Bilinear based similarity is best for codeword selector,* since Bilinear performs best among the four similarity functions. Notice that Bilinear is the only parameterized similarity, whose parameters can be learned from data, so the similarity function is more compatible with the relevance function based on inner product. This addresses performance degradation due to the inconsistency from Euclidean distance based similarity. Moreover, inner product and scaled inner product are special cases of Bilinear, whose superiority and flexibility become apparent.

Table 5: Effectiveness of the similarity functions in the CiteULike and Gowalla datasets.

$s(\boldsymbol{q}, \boldsymbol{c})$	Recall@100		NDCG	
	CiteULike	Gowalla	CiteULike	Gowalla
$f(\boldsymbol{q})$	0.2846	0.3216	0.2009	0.2851
$\langle \boldsymbol{q}, \boldsymbol{c} \rangle$	0.2851	0.3332	0.2011	0.2888
$1/\sqrt{D}\langle \boldsymbol{q}, \boldsymbol{c} \rangle$	0.3036	0.3370	0.2111	0.2895
Bilinear	0.3141	0.3423	0.2160	0.2942
$-\|\|\boldsymbol{q} - \boldsymbol{c}\|\|_2^2$	0.2913	0.3357	0.2061	0.2916

4.7 Effectiveness of Knowledge Distillation

We then study effect of knowledge distillation by comparing Euclidean based Knowledge Distillation (E-KD), Inner Product based Knowledge Distillation (IP-KD), and Ranking and Inner Product based Knowledge Distillation (RIP-KD). LightRec without guidance, denoted by "No KD", is also included for comparison, as shown Table 6. The observations are summarized as follows.

- *Knowledge distillation improves the quality of composite representation,* based on the superiority of E-KD, IP-KD and RIP-KD to "No KD". The main reason is that the end task can not guarantee the quality of representation due to the sparsity issue and optimization challenge. The auxiliary distillation loss can help to guide learning better composite representation for items. This also indicates the difficulty of learning good composite representation in an end-to-end framework.
- *Preserving inner product may be more important for recommendation,* according to the superiority of IP-KD to E-KD. This is because inner product is used to estimate relevance scores. *Ranking based knowledge is useful but difficult to preserve,* since RIP-KD is slightly better than IP-KD. It is well-known that top-k recommendation is the ultimate goal of recommender system, preserving relative order of top-k preferred items will be carefully investigated in future work within the current framework.

Table 6: Effectiveness of knowledge distillation in the CiteULike and Gowalla datasets.

	Recall@100		NDCG	
	CiteULike	Gowalla	CiteULike	Gowalla
No KD	0.3044	0.3309	0.2087	0.2900
E-KD	0.3187	0.3389	0.2134	0.2919
IP-KD	0.3133	0.3423	0.2151	0.2942
RIP-KD	0.3141	0.3439	0.2160	0.2939

4.8 Study of Memory and Search Efficiency

4.8.1 Settings. We vary the number of candidate items recalled by LightRec, from 100 to 500, and evaluate DCMF, LightRec and BPR in the the Amazon dataset, in which there is the largest number of items. We then report the recommendation performance in Fig 5 and speedup ratio of DCMF and LightRec to BPR in Table 7. Note that the recalled items are post-ranked by BPR and the time cost of post-ranking is included into online inference. The compress ratio of item representation is also reported in Table 7

4.8.2 Findings – LightRec incurs significant speedup of top-k recommendation while leads to negligible accuracy degradation, and dramatically reduces memory consumption. In LightRec, item representation can be compressed by 25x, and top-k recommendation can be accelerated by 27x when the degradation of recommendation performance in terms of both Recall and NDCG can be negligible. In DCMF, compress ratio of item representation is larger, due to excluding codebooks, and speedup ratio of top-k recommendation is slightly higher than LightRec, due to saving time for computing relevance scores for codewords. However, the performance of

recommendation degrades much more. In order to reduce performance degradation, more candidate items are required to recall. As discussed before, this lies in much smaller representation capacity of binary Hamming space. This finding confirms that LightRec strikes a better balance between accuracy and efficiency of online recommendation.

Table 7: Speedup ratio of DCMF and LightRec to BPR in the Amazon dataset, and compress ratio of item representation.

	Speedup ratio (k=100–500)					Compress ratio
	100	200	300	400	500	
LightRec	31.10	28.95	27.23	26.78	25.27	25.51
DCMF	32.86	30.42	28.26	26.30	25.40	31.99

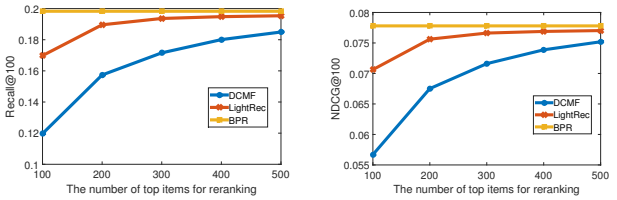


Figure 5: Performance comparison of DCMF and LightRec with BPR in the Amazon dataset.

5 CONCLUSION AND FUTURE WORKS

In this paper, we propose LightRec – a lightweight recommender system. LightRec can reduce memory consumption of item representation by more than 96%, making it possible to load a tremendous pool of items into main memory once a time. The core of LightRec is recurrent composite encoding, which can be implemented by a custom layer and easily plugged into deep recommendation models. The extensive experiments with four real-world datasets show that LightRec outperforms the state-of-the-art lightweight baselines by more than 11% in terms of recall performance. Compared to conventional recommendation algorithms, LightRec only leads to negligible accuracy degradation while achieves more than 27x speedup in top-k recommendation. Future works can include designing new efficient methods for learning codebooks and investigating more effective strategies of ranking preserved distillation.

ACKNOWLEDGMENTS

The work was supported by grants from the National Natural Science Foundation of China (Grant No. 61976198, 61727809 and 61832017).

A PROOFS OF PROPOSITIONS

Proof of Proposition 1

Proof. Codedword indices of each item are represented as concatenation of B one-hot vectors, $\boldsymbol{e}_i = [\boldsymbol{e}_i^1, \cdots, \boldsymbol{e}_i^B]$, where $\boldsymbol{e}_i^b$ is a one-hot vector of length W. Codebooks are correspondingly organized as a matrix $\boldsymbol{C} \in \mathbb{R}^{D\times BW}$. We have $\sum_{b=1}^{B} \boldsymbol{c}^b_{w_i^b} = \boldsymbol{C}\boldsymbol{e}_i$. Recalling we approximate inner product, i.e., $\langle \boldsymbol{p}_u, \boldsymbol{q}_i\rangle \approx \langle \boldsymbol{p}_u, \boldsymbol{C}\boldsymbol{e}_i\rangle$. The approximation is achieved by minimization the objective function $1/M \sum_u \|\boldsymbol{Q}\boldsymbol{p}_u - \boldsymbol{E}\boldsymbol{C}^T\boldsymbol{p}_u\|^2 = 1/M\|\boldsymbol{Q}\boldsymbol{P}^T - \boldsymbol{E}\boldsymbol{C}^T\boldsymbol{P}^T\|_F^2 = 1/M \text{ tr}[(\boldsymbol{Q} - \boldsymbol{E}\boldsymbol{C}^T)\boldsymbol{P}^T\boldsymbol{P}(\boldsymbol{Q} - \boldsymbol{E}\boldsymbol{C}^T)^T]$. Moreover, each row of matrix $\boldsymbol{E}$ is a sparse binary vector, since only 1/W of entries are non-zero. Following the definition of generalized SVD, this can be considered as generalized binarized low-rank factorization of item representation matrix. □

Proof of Proposition 2

Proof. In Gaussian mixture model, the posterior probability $\gamma(z_w) = \frac{\pi_w \mathcal{N}(\boldsymbol{q}_i|\boldsymbol{c}_w, \Sigma_w)}{\sum_{w'} \pi_{w'} \mathcal{N}(\boldsymbol{q}_i|\boldsymbol{c}_{w'}, \Sigma_{w'})}$ if cordwords are considered cluster centers. Let $\pi_w = 1/W$ and $\Sigma_w = T\boldsymbol{I}/2$, $\gamma(z_w) = \frac{\exp(-\|\boldsymbol{q}_i - \boldsymbol{c}_w\|^2/T)}{\sum_{w'} \exp(-\|\boldsymbol{q}_i - \boldsymbol{c}_{w'}\|^2/T)}$. Therefore, $\gamma(z_w) = \tilde{e}_i[w]$. □

Proof of Proposition 3

Proof. $\mathbf{e}_i = \text{one-hot}(\arg\max_w \text{s}(\boldsymbol{q}_i, \boldsymbol{c}_w))$. Given a scaling factor $\alpha > 0$ for a codebook,

$$\begin{aligned}
&\arg\max_w \text{s}(\boldsymbol{q}_i, \alpha\boldsymbol{c}_w) \\
=&\arg\max_w \boldsymbol{q}_i^T\boldsymbol{W}\alpha\boldsymbol{c}_w + \langle \boldsymbol{w}_1, \boldsymbol{q}_i\rangle + \langle \boldsymbol{w}_2, \alpha\boldsymbol{c}_w\rangle \\
=&\arg\max_w \alpha(\boldsymbol{q}_i^T\boldsymbol{W}\boldsymbol{c}_w + \langle \boldsymbol{w}_2, \boldsymbol{c}_w\rangle) \\
=&\arg\max_w \boldsymbol{q}_i^T\boldsymbol{W}\boldsymbol{c}_w + \langle \boldsymbol{w}_2, \boldsymbol{c}_w\rangle \\
=&\arg\max_w \text{s}(\boldsymbol{q}_i, \boldsymbol{c}_w)
\end{aligned}$$

Thus, $\mathbf{e}_i$ is invariant to codebook scaling. $\tilde{e}_i[w] = \frac{\exp(\text{s}(\boldsymbol{q}_i, \boldsymbol{c}_w)/T)}{\sum_{w'} \exp(\text{s}(\boldsymbol{q}_i, \boldsymbol{c}_{w'})/T)}$

$$\begin{aligned}
&\frac{\exp(\text{s}(\boldsymbol{q}_i, \alpha\boldsymbol{c}_w)/T)}{\sum_{w'} \exp(\text{s}(\boldsymbol{q}_i, \alpha\boldsymbol{c}_{w'})/T)} \\
=&\frac{\exp\left((\boldsymbol{q}_i^T\boldsymbol{W}\alpha\boldsymbol{c}_w + \langle \boldsymbol{w}_1, \boldsymbol{q}_i\rangle + \langle \boldsymbol{w}_2, \alpha\boldsymbol{c}_w\rangle)/T\right)}{\sum_{w'} \exp\left((\boldsymbol{q}_i^T\boldsymbol{W}\alpha\boldsymbol{c}_{w'} + \langle \boldsymbol{w}_1, \boldsymbol{q}_i\rangle + \langle \boldsymbol{w}_2, \alpha\boldsymbol{c}_{w'}\rangle)/T\right)} \\
=&\frac{\exp\left((\boldsymbol{q}_i^T\boldsymbol{W}\boldsymbol{c}_w + \langle \boldsymbol{w}_2, \boldsymbol{c}_w\rangle)\alpha/T\right)}{\sum_{w'} \exp\left((\boldsymbol{q}_i^T\boldsymbol{W}\boldsymbol{c}_{w'} + \langle \boldsymbol{w}_2, \boldsymbol{c}_{w'}\rangle)\alpha/T\right)} \\
=&\frac{\exp(\text{s}(\boldsymbol{q}_i, \boldsymbol{c}_w)\alpha/T)}{\sum_{w'} \exp(\text{s}(\boldsymbol{q}_i, \boldsymbol{c}_{w'})\alpha/T)}
\end{aligned}$$

Thus, codebook scaling is equivalent to inversely scaling temperature T. Since T is hyper-parameter to tune, $\tilde{\boldsymbol{e}}_i$ can be considered invariant to codebook scaling. □

Proof of Proposition 4

Proof. Recurrent composite encoding is assumed to learn a encoding function $Q(\boldsymbol{x})$. Given reconstruction error $\mathcal{L} = \frac{1}{2}\|\boldsymbol{x} - Q(\boldsymbol{x})\|^2$, by treating $Q(\boldsymbol{x})$ as a parameter, the gradient $\nabla_{Q(\boldsymbol{x})}\mathcal{L} = -(\boldsymbol{x} - Q(\boldsymbol{x}))$. Based on gradient descent, in the b-th iteration, the function can be updated by $Q_b(\boldsymbol{x}) = Q_{b-1}(\boldsymbol{x}) + \alpha(\boldsymbol{x} - Q_{b-1}(\boldsymbol{x}))$. Applying $Q(\boldsymbol{x}|\boldsymbol{C}^b)$ for approximating $\boldsymbol{x} - Q_{b-1}(\boldsymbol{x})$, we have $Q_b(\boldsymbol{x}) = Q_{b-1}(\boldsymbol{x}) + Q(\boldsymbol{x} - Q_{b-1}(\boldsymbol{x})|\alpha\boldsymbol{C}^b)$, where $\alpha Q(\boldsymbol{x}|\boldsymbol{C}^b) = Q(\boldsymbol{x}|\alpha\boldsymbol{C}^b)$ can be deduced from Proposition 3. Let $Q_0(\boldsymbol{x}) = 0$, then $Q_b(\boldsymbol{x}) = \sum_{t=1}^{b} Q(\boldsymbol{x} - Q_{t-1}(\boldsymbol{x})|\boldsymbol{C}^b)$. □

REFERENCES

[1] Artem Babenko and Victor Lempitsky. 2014. Additive quantization for extreme vector compression. In *Proceedings of CVPR'14*. 931–938.

[2] Artem Babenko and Victor Lempitsky. 2014. The inverted multi-index. *IEEE transactions on pattern analysis and machine intelligence* 37, 6 (2014), 1247–1260.

[3] Yoram Bachrach, Yehuda Finkelstein, Ran Gilad-Bachrach, Liran Katzir, Noam Koenigstein, Nir Nice, and Ulrich Paquet. 2014. Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces. In *Proceedings of RecSys'14*. ACM, 257–264.

[4] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432* (2013).

[5] Z. Cao, T. Qin, T.Y. Liu, M.F. Tsai, and H. Li. 2007. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of ICML'07*. ACM, 129–136.

[6] 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.

[7] Xu Chen, Hanxiong Chen, Hongteng Xu, Yongfeng Zhang, Yixin Cao, Zheng Qin, and Hongyuan Zha. 2019. Personalized Fashion Recommendation with Visual Explanations based on Multimodal Attention Network: Towards Visually Explainable Recommendation. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 765–774.

[8] Yongjian Chen, Tao Guan, and Cheng Wang. 2010. Approximate nearest neighbor search by residual vector quantization. *Sensors* 10, 12 (2010), 11259–11273.

[9] 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.

[10] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. 2007. Google news personalization: scalable online collaborative filtering. In *Proceedings of WWW'07*. ACM, 271–280.

[11] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2014. Optimized product quantization. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36, 4 (2014), 744–755.

[12] 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.

[13] Ruiqi Guo, Sanjiv Kumar, Krzysztof Choromanski, and David Simcha. 2016. Quantization based fast inner product search. In *Artificial Intelligence and Statistics*. 482–490.

[14] Ruining He and Julian McAuley. 2016. VBPR: visual Bayesian Personalized Ranking from implicit feedback. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI Press, 144–150.

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

[16] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of WWW'17*. 173–182.

[17] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[18] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge Belongie, and Deborah Estrin. 2017. Collaborative metric learning. In *Proceedings of WWW'17*. International World Wide Web Conferences Steering Committee, 193–201.

[19] Binbin Hu, Chuan Shi, Wayne Xin Zhao, and Philip S Yu. 2018. Leveraging meta-path based context for top-n recommendation with a neural co-attention model. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1531–1540.

[20] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of CIKM'13*. ACM, 2333–2338.

[21] 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 KDD'18*. ACM, 1561–1570.

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

[23] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2011. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 1 (2011), 117–128.

[24] Noam Koenigstein, Parikshit Ram, and Yuval Shavitt. 2012. Efficient retrieval of recommendations in a matrix factorization framework. In *Proceedings of CIKM'12*. ACM, 535–544.

[25] Walid Krichene, Nicolas Mayoraz, Steffen Rendle, Li Zhang, Xinyang Yi, Lichan Hong, Ed Chi, and John Anderson. 2018. Efficient training on very large corpora via gramian estimation. *arXiv preprint arXiv:1807.07187* (2018).

[26] Defu Lian, Rui Liu, Yong Ge, Kai Zheng, Xing Xie, and Longbing Cao. 2017. Discrete Content-aware Matrix Factorization. In *Proceedings of KDD'17*. 325–334.

[27] 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

[28] 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 KDD'18*. ACM, 1754–1763.

[29] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. 2016. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712* (2016).

[30] Behnam Neyshabur and Nathan Srebro. 2015. On Symmetric and Asymmetric LSHs for Inner Product Search. In *Proceedings of ICML'15*. 1926–1934.

[31] Mohammad Norouzi, Ali Punjani, and David J Fleet. 2012. Fast search in hamming space with multi-index hashing. In *Proceedings of CVPR'12*. IEEE, 3108–3115.

[32] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. 2016. Product-based neural networks for user response prediction. In *Proceedings of ICDM'16*. IEEE, 1149–1154.

[33] Parikshit Ram and Alexander G Gray. 2012. Maximum inner-product search using cone trees. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 931–939.

[34] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of UAI'09*. AUAI Press, 452–461.

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

[36] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. 2015. Collaborative Deep Learning for Recommender Systems. In *Proceedings of KDD'15*. ACM, 1235–1244.

[37] Hao Wang, Tong Xu, Qi Liu, Defu Lian, Enhong Chen, Dongfang Du, Han Wu, and Wen Su. 2019. MCNE: An End-to-End Framework for Learning Multiple Conditional Network Representations of Social Network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1064–1072.

[38] Hongwei Wang, Fuzheng Zhang, Jialin Wang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. 2018. Ripplenet: Propagating user preferences on the knowledge graph for recommender systems. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, 417–426.

[39] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*. ACM, 12.

[40] Chuhan Wu, Fangzhao Wu, Mingxiao An, Jianqiang Huang, Yongfeng Huang, and Xing Xie. 2019. Npa: Neural news recommendation with personalized attention. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2576–2584.

[41] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. 2017. Recurrent recommender networks. In *Proceedings of the tenth ACM international conference on web search and data mining*. ACM, 495–503.

[42] Pengtao Xie, Wei Wu, Yichen Zhu, and Eric P Xing. 2018. Orthogonality-Promoting Distance Metric Learning: Convex Relaxation and Theoretical Analysis. In *International Conference on Machine Learning*. 5399–5408.

[43] 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.

[44] Yang Yu, Yu-Feng Li, and Zhi-Hua Zhou. 2011. Diversity regularized machine. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

[45] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative knowledge base embedding for recommender systems. In *Proceedings of KDD'16*. ACM, 353–362.

[46] Hanwang Zhang, Fumin Shen, Wei Liu, Xiangnan He, Huanbo Luan, and Tat-Seng Chua. 2016. Discrete collaborative filtering. In *Proceedings of SIGIR'16*. ACM, 325–334.

[47] Ting Zhang, Chao Du, and Jingdong Wang. 2014. Composite Quantization for Approximate Nearest Neighbor Search. In *Proceedings of ICML'14*. 838–846.

[48] Yan Zhang, Defu Lian, and Guowu Yang. 2017. Discrete Personalized Ranking for Fast Collaborative Filtering from Implicit Feedback. In *Proceedings of AAAI'17*. 1669–1675.

[49] Zhiwei Zhang, Qifan Wang, Lingyun Ruan, and Luo Si. 2014. Preference preserving hashing for efficient recommendation. In *Proceedings of SIGIR'14*. ACM, 183–192.

[50] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1059–1068.

[51] Ke Zhou and Hongyuan Zha. 2012. Learning binary codes for collaborative filtering. In *Proceedings of KDD'12*. ACM, 498–506.

[52] Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, and Kun Gai. 2018. Learning Tree-based Deep Model for Recommender Systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1079–1088.