Knowledge-Based Systems 172 (2019) 64–75

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Learning binary codes with neural collaborative filtering for efficient recommendation systems

Yang Li [a], Suhang Wang [c], Quan Pan [a], Haiyun Peng [b], Tao Yang [a], Erik Cambria [b,*]

[a] School of Automation, Northwestern Polytechnical University, China
[b] School of Computer Science and Engineering, Nanyang Technological University, Singapore
[c] College of Information Sciences and Technology, Pennsylvania State University, USA

ARTICLE INFO

Article history:
Received 14 September 2018
Received in revised form 4 February 2019
Accepted 11 February 2019
Available online 15 February 2019

Keywords:
Recommendation systems
Binary code learning
Neural networks
Neural collaborative hashing

ABSTRACT

The fast-growing e-commerce scenario brings new challenges to traditional collaborative filtering because the huge amount of users and items requires large storage and efficient recommendation systems. Hence, hashing for collaborative filtering has attracted increasing attention as binary codes can significantly reduce the storage requirement and make similarity calculations efficient. In this paper, we investigate the novel problem of deep collaborative hashing codes on user–item ratings. We propose a new deep learning framework for it, which adopts neural networks to better learn both user and item representations and make these close to binary codes such that the quantization loss is minimized. In addition, we extend the proposed framework for out-of-sample cases, i.e., dealing with new users, new items, and new ratings. Extensive experiments on real-world datasets demonstrate the effectiveness of the proposed framework.

1. Introduction

Recommender systems, which aim at recommending potential products that user may be interested in, have recently attracted huge attention [1–6]. Among various recommender systems, collaborative filtering has achieved great success due to its superior performance. Despite the great success of traditional collaborative filtering, the fast growth of online shopping platforms brings new challenges to recommender systems due to the huge amount of users and products. For example, it is reported that Amazon has in total 300 million users[1] and 398 million products as on January, 2017.[2] There were approximately 226.6 million active customers and fulfilled 1.6 billion orders in 2016 from JD.[3] Traditional collaborative filtering, which learns continuous vector representations of users and items and makes recommendation based on the similarity of the representations, suffers from severe issues as: (i) the massive user and item continuous vector representation requires huge storage (e.g., it needs 256 bytes for a single item if its length is 32, note that data are stored in double format); and (2) it is time consuming to make recommendation to each user because we need to calculate the similarity between a user and the huge number of items (e.g., it takes more than ten seconds to process 1 million calculation in a recommendation). Therefore, efficient representations of users and items that can reduce storage requirement and search time cost are needed.

Hash collaborative filtering, which learns the binary representation of users and items, has become a popular efficient recommendation technique [7–10]. By learning binary codes, the storage requirement can be significantly reduced as storing each binary code only require 4 bytes if the code length is 32. Also, the time complexity of calculating the hamming distance between two binary codes is very efficient especially when the binary code length is short. The majority of existing hash collaborative filtering algorithms exploits a two-stage approach, which first learns continuous vector representation, then quantizes the continuous vector representation to binary codes. However, the quantization will introduce large information loss, which significantly degrades the recommendation performance. Therefore, to alleviate the information loss, Zhang et al. [10] seek to learn the binary codes in a one-stage process by solving a discrete optimization problem. However, discrete optimization is NP-hard and is inefficient.

Another direction to compensate the quantization loss is to (i) learn better continuous vector representations which contain the semantic information of users and items that are better at the recommendation, and (ii) make the continuous vector representation be close to binary vector representation. Recently, deep learning for collaborative filtering has significantly improved the recommendation performance [4,11] because of its great representation learning ability. Therefore, it is very promising to exploit deep

* Corresponding author.
E-mail address: cambria@ntu.edu.sg (E. Cambria).

1 http://expandedramblings.com/index.php/amazon-statistics.

2 http://scrapehero.com/how-many-products-are-sold-on-amazon-com-January-2017-report.

3 http://ir.jd.com/phoenix.zhtml?c=253315&p=irol-homeprofile.

https://doi.org/10.1016/j.knosys.2019.02.012
0950-7051/

collaborative filtering for learning binary codes. However, the work on investigating deep collaborative filtering binary codes on user–item ratings is rather limited.

Therefore, in this paper, we study the novel problem of investigating deep learning to learn binary codes for collaborative filtering. In essence, we need to solve two problems: (i) how to design a deep collaborative filtering architecture that is good at learning user and item representations; and (ii) how to make the learned representation close to binary codes such that the quantization loss is minimized when doing the hashing? In an attempt to solve these two questions, we propose a novel deep learning framework for deep collaborative hashing (DCH). The model is composed of two neural networks for learning user and item representations, respectively, and the two neural networks are trained to reconstruct ratings. The activation of the neural networks at the last layer is specially designed such that the outputs are close to binary code. Furthermore, the out-of-sample cases are also explored. The contributions of this work are listed as follows:

- We investigate a new problem of deep hashing for collaborative filtering;
- We propose a novel neural collaborative filtering framework for learning binary codes and extend the framework for out-of-sample extension;
- We conducts extensive experiments on three real-world datasets to demonstrate the effectiveness of the proposed framework for both the normal case and the out-of-sample cases;

The remainder of this paper is organized as follows: Section 2 describes related works, including hashing for efficient recommendation, hashing by deep learning, and deep learning for recommender systems; Section 3 introduces the details of the proposed framework; Section 4 provides an optimization framework for training DCH; Section 5 extends the DCH for dealing with new users, new items, and new ratings; Section 6 describes experiments; finally, Section 7 discusses conclusion and future work.

2. Related works

In this paper, we investigate binary codes with neural collaborative filtering for an efficient recommendation. The work is related to hashing for the efficient recommendation, deep learning based hashing and recommendation.

2.1. Hashing for efficient recommendation

Hashing is a popular method for efficient approximate nearest neighbor search on massive dataset [12,13]. It aims at learning a low-dimensional binary vector representation of the data points, which is called binary codes. The binary vector representation can (i) significantly reduce the storage requirement for large-scale datasets as each element of the binary code, i.e., 1/-1, only takes up 1 bit in the storage and; (ii) significantly reduce the cost of querying as the similarity calculation of binary codes is much efficient. Therefore, learning to hash is widely used for efficient similarity search of text, image and video data in the search engine [14–16].

Collaborative filtering is essentially a similarity search problem, where even linear time complexity is prohibitive for large-scale recommendation tasks. Thus, hashing for collaborative filtering has attracted increasing attention [7–9]. For example, Karatzoglou et al. [7] and Liu et al. [9] learned the continuous representation with traditional CF, then the technique of rounding or rotation is applied to get the binary codes. Zhou et al. [8] built their model by relaxing the hashing with real value at first, then the rounding was applied to get the binary codes. Apparently, all of these works contain two stages, i.e., first learn the continuous vector representation of users and items, and then quantize the continuous vector representation to binary codes. Such a two-stage scheme will generate large quantization loss. Therefore, Zhang et al. [10] proposed the discrete collaborative filtering, which is a unified framework that integrates the hash learning process into matrix factorization by solving a discrete optimization problem. However, discrete optimization is NP-hard and inefficient. Therefore, in this paper, we try to compensate the quantization loss by achieve two goals (i) learn better continuous vector representations of users and items that for the better recommendation; and (ii) learn continuous representation that is close to binary codes to reduce quantization loss.

As we have discussed, the majority of existing hash collaborative filtering is based on matrix factorization, while deep learning algorithms have been demonstrated to be effective in learning user and item latent features from user–item rating history [4,17]. Therefore, in this paper, we investigate the novel problem of exploiting deep learning to learn binary codes for collaborative filtering, which is to achieve the two goals for compensating quantization loss.

2.2. Hashing by deep learning

Deep learning has become one of the most successful ways in hashing learning, especially in the searching system. Based on different data, lots of models are proposed. For the image data, models like deep hashing (DH) [18], supervised deep hashing (SDH) [18], deep quantization network (DQN) [19] and deep hashing network (DHN) [20], etc., extract the features from the image and learn the hash code with neural network models. Most of them are end-to-end methods which are composed of feature representation learning over the image and hashing code learning, and belongs to the two-stage schema mentioned before. Furthermore, the supervised information is adopted to guide the feature representation learning [18]. Same as that in the image data, deep learning for hashing models can be proposed for the text data. Suthee et al. [21] build the variational deep semantic hashing (VDSH) based on the variational auto-encoder [22]. In the meantime, there are works over the multi-modal data. Cao et al. [19] build the deep visual-semantic hashing (DVSH) with fusing visual embedding from the image and semantic embedding from the text before the hash function, which also is a two-stage model. Hu et al. [23] propose the deep binary reconstruction (DBRC) model for the multi-modal hash code learning without the similarity information. Then Wu et al. [24] proposed a unified framework self-supervised deep multi-model hashing (SSDMH) by integrating deep learning and the latent representation regularization.

Unlike previous works, the hash learning in this paper is over the users and items only. As far as we know, this is the first work to learn to hash with deep learning over this type of data.

2.3. Deep learning for recommender systems

Recently, along with the successful application of deep learning in natural language processing (NLP) research [25–30], deep networks have attracted a lot of attention in the recommendation community, and many deep learning based recommender systems have shown promising results [4,17,31].

Compared with the traditional latent feature learning methods, deep learning based models are more effective in learning representations that can capture relations between users and items. Methods like community embeddings [32], User2Vec [33], and Item2Vec [34] learn user and item vectors by applying embedding techniques [35,36]. Works like neural collaborative filtering

(NCF) [4] applied neural networks to perform collaborative filtering and has achieved promising results. Pei et al. [37] proposed the interacting attention-gated recurrent network (IARN) which learned the feature from the user and item at the same time. Models like collaborative variational auto-encoder (CVAE) [11], collaborative denoising auto-encoder (CDAE) [38] applied the auto-encoder frameworks to get the latent representation of the user and items. For the CVAE, it is the inference process for the latent information learning, and as to the CDAE, it can be considered as the SVD++ if the activate function is identity.

However, the works above aim at learning continuous vector representation of users and items for recommendation. The work on exploring deep learning to learn hash for collaborative filtering is rather limited. One reason is that learning binary codes brings challenges to training neural networks: the binary codes are the discrete value which limits the back-propagation in the neural networks. Therefore, in this paper, we propose a novel framework that utilizes deep learning for hash collaborative filtering by overcoming the challenges.

3. Neural collaborative filtering for hashing

Before introducing the details of the proposed framework, we first introduce the notations used in this paper. Throughout the paper, matrices are written in boldface capital letters and vectors are denoted as boldface lowercase letters. For an arbitrary matrix $\mathbf{M}$, M_{ij} denotes the (i,j)th entry of $\mathbf{M}$ and $\|\mathbf{M}\|_F^2$ is the Frobenius norm. Capital letters in calligraphic math font such as $\mathcal{N}$ are used to denote sets and $|\mathcal{N}|$ is the cardinality. Let $\mathcal{U} = \{u_1, u_2, \ldots, u_n\}$ be the set of n-users and $\mathcal{V} = \{v_1, v_2, \ldots, v_m\}$ be the set of m items. We use $\mathbf{S} \in \mathbf{R}^{n\times m}$ to denote user–item rating matrix where S_{ij} is the rating score from u_i to v_j if u_i rates v_j, otherwise, $S_{ij} = 0$. We assume that $\mathcal{U}$, $\mathcal{V}$ and $\mathbf{S}$ are fixed. Function sign is represented by *sgn*, and softsign is abbreviated to *softsgn*.

3.1. Collaborative filtering with MLP

Matrix factorization is one of the most popular models for collaborative filtering and has been proven to be effective [1]. The essential idea of matrix factorization is to decompose the rating matrix $\mathbf{S}$ into two low-rank matrices $\mathbf{B}$ and $\mathbf{D}$ which are good at reconstructing the rating

$$\min_{\mathbf{U},\mathbf{V}} \sum_{ij:S_{ij}>0} (S_{ij} - \mathbf{u}_i^T \mathbf{v}_j)^2 + \lambda \Omega(\mathbf{U}, \mathbf{V}) \tag{1}$$

where $\mathbf{U} \in \mathbf{R}^{d\times n}$ is the user latent feature matrix and $\mathbf{V} \in \mathbf{R}^{d\times m}$ is the item latent feature matrix. $\Omega(\mathbf{U}, \mathbf{V})$ is the regularizer on $\mathbf{U}$ and $\mathbf{V}$ to alleviate overfitting. With $\mathbf{U}$ and $\mathbf{V}$, we can learn binary user and item representations that are close to $\mathbf{U}$ and $\mathbf{V}$ or we can simply quantize $\mathbf{U}$ and $\mathbf{V}$ to binary codes.

It is obvious that using binary codes to represent user preferences and item properties can introduce information loss. Therefore, we need a powerful model for learning user and item latent features to compensate for the loss. Deep learning methods are very good at learning features from raw input data. Recently, it has been proven to increase the performance of recommender systems [4] significantly. Therefore, we exploit deep learning algorithms for learning user and item latent features.

The architecture of the deep learning model proposed for learning user and item representation is shown in Fig. 1. In the proposed framework, we assume that the simple neural network can work well over the feature extraction from the users and items, and this has been validated in NCF [4]. Also, it can be a universal approximator when there are only two fully connected layers network, and this has been proven in [39] if there are sufficient hidden nodes. To make the framework simplicity and efficiency, the two-layer MLP has been applied in the work. Thus, the framework is composed of two columns of deep neural networks, MLP_u and MLP_v, which are used to learn user preferences and item latent features, respectively. To reduce the learned continues values are close to the binary codes, the *tanh* but not the *sigmoid* activation function is adopted to make the non-linear transformation. Also, it can be a more complex neural network structure for these two columns neural networks, such as CNN, LSTM, etc.

Because of the sparsity of the data, it is hard for the deep learning to learn good feature from the user and item. Inspired by the method of word embedding in the text data [36,40], each word can be represented by the low dimension vector. Hence, in the bottom layers, the look up method is applied to get the user embedding and item embedding, where each user or item is depicted as $\mathbf{e}(u_{ik}) \in \mathbf{R}^{l_u\times 1}$ or $\mathbf{e}(v_{jk}) \in \mathbf{R}^{l_v\times 1}$, and l_u and l_v are the user embedding dimension and item embedding dimension, respectively. For user u_i, its input to MLP_u is the sequence of items bought by u_i, i.e., $\mathcal{N}_{ui} = \langle v_{i1}, v_{i2}, \ldots, v_{iN_{ui}}\rangle$, where N_{ui} is the total items number that bought by u_i.

Then latest n items and latest m users are selected from $\mathcal{N}_{ui}$ and $\mathcal{N}_{vj}$ respectively as the input. If the user or item does not have enough history information, then the empty values will be replaced with zeros. Then the embedding layer will project the user/item to embeddings as $\mathbf{H}_{u_i} = \langle \mathbf{e}(v_{i1}), \mathbf{e}(v_{i2}), \ldots, \mathbf{e}(v_{in})\rangle$, item array change to $\mathbf{H}_{v_j} = \langle \mathbf{e}(u_{j1}), \mathbf{e}(u_{j2}), \ldots, \mathbf{e}(u_{jn})\rangle$.

The embeddings then go through standard MLP layers to learn latent features. The output of MLP_u and MLP_v are used to predict the rating as

$$\begin{aligned} &\min_{\theta_u,\theta_v} \sum_{ij:S_{ij}>0} (S_{ij} - \mathbf{u}_i^T \mathbf{v}_j)^2 + \lambda_1 \|\theta_u\|_F^2 + \lambda_2 \|\theta_v\|_F^2 \\ &s.t.\ \mathbf{u}_i = MLP_u(\mathcal{N}_{ui}), \quad \mathbf{v}_j = MLP_v(\mathcal{N}_{vj}) \end{aligned} \tag{2}$$

where θ_u and θ_v are the parameters of MLP_u and MLP_v, respectively and λ_1 and λ_2 are the two scalars to control the contribution of the regularization.

3.2. Binary representation learning

With the powerful neural collaborative filtering described in last section, we are going to introduce how to exploit it for learning binary codes. One simple approach is to use the two-stage approach as first learning $\mathbf{U}$ and $\mathbf{V}$ with Eq. (2), then simply getting the binary codes as $\mathbf{b}_i = sgn(\mathbf{u}_i)$ and $\mathbf{d}_j = sgn(\mathbf{v}_j)$. However, this will introduce large quantization loss. Another choice is to learn $\mathbf{b}_i$ and $\mathbf{d}_j$ in one-stage as

$$\begin{aligned} &\min_{\theta_u,\theta_v} \sum_{ij:S_{ij}>0} (S_{ij} - \mathbf{b}_i^T \mathbf{d}_j)^2 + \lambda_1 \|\theta_u\|_F^2 + \lambda_2 \|\theta_v\|_F^2 \\ &s.t.\ \mathbf{b}_i = sgn(MLP_u(\mathcal{N}_{ui})), \quad \mathbf{d}_j = sgn(MLP_v(\mathcal{N}_{vj})) \end{aligned} \tag{3}$$

However, the gradients of $(S_{ij} - \mathbf{b}_i^T \mathbf{d}_j)$ w.r.t. to the parameters of θ_u and θ_v are always zero due to the *sgn* operation, which makes the training of the neural networks intractable. In an attempt to alleviate the quantization loss and make the training tractable, we use *softsgn* to replace *sgn* as

$$\begin{aligned} &\min_{\theta_u,\theta_v} \sum_{ij:S_{ij}>0} (S_{ij} - \mathbf{b}_i^T \mathbf{d}_j)^2 + \lambda_1 \|\theta_u\|_F^2 + \lambda_2 \|\theta_v\|_F^2 \\ &s.t.\ \mathbf{b}_i = softsgn(MLP_u(\mathcal{N}_{ui})), \quad \mathbf{d}_j = softsgn(MLP_v(\mathcal{N}_{vj})) \end{aligned} \tag{4}$$

where *softsgn* is defined as

$$softsgn(x) = \frac{x}{1 + |x|} \tag{5}$$

It is easy to see that *softsgn* is an approximation of *sgn* especially when x is large. Unlike *tanh*, which converges exponentially and

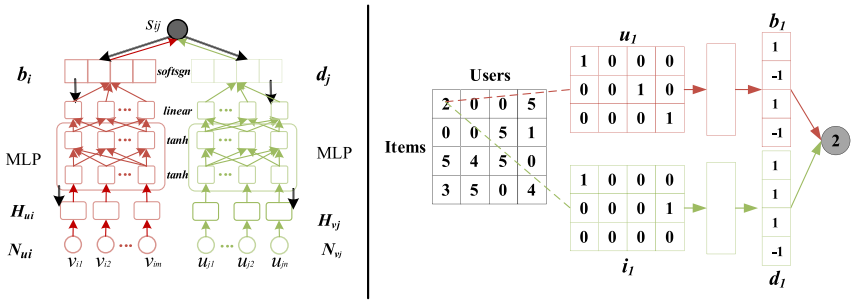


Fig. 1. The structure and the toy example of the DCH framework.

has an extreme activation distribution, *softsgn* converges polynomially, and it activation distribution is around the knee and has smoother asymptotes, which make it not easy to be saturated. In addition, *softsgn* is differentiable, which makes the training of neural networks using gradient descent possible. To make the output of *softsgn* close to $1/-1$ to minimize the quantization error, we want the output of $MLP_u(\mathcal{N}_{ui})$ to be large. Because *tanh* is applied in MLPs, which makes the output value within $[-1, 1]$. Therefore, a linear layer is added on the top of MLP_u and MLP_v which allows the output of MLP_u and MLP_v have a chance to be the large value.

3.3. The proposed framework

As we have described, matrix factorization is a popular way to do the collaborative filtering. In order to learn the user and item representation from the explicit information, the neural network is applied. To make the framework simple, the two-layer MLPs which are MLP_u and MLP_v are adopted to extract the latent features. The ideal representation about user and item should be expressive and diverse. To make the learned binary code uncorrelated, the constraint $\mathbf{B}^T\mathbf{B} = m\mathbf{I}$ is added. To make the binary code diverse, the constraint $\mathbf{b}_i^T\mathbf{1} = 0$ is added. Thus, the objective function of the DCH framework is given as

$$\begin{aligned} &\arg\min_{\theta_u,\theta_v} \sum_{ij:S_{ij}>0} (S_{ij} - \mathbf{b}_i^T\mathbf{d}_j)^2 + \lambda_1\|\theta_u\|_F^2 + \lambda_2\|\theta_v\|_F^2 \\ &s.t.\ \ \mathbf{B}^T\mathbf{1} = 0,\ \mathbf{D}^T\mathbf{1} = 0, \mathbf{B}^T\mathbf{B} = m\mathbf{I},\ \mathbf{D}^T\mathbf{D} = n\mathbf{I} \\ &\quad \mathbf{b}_i = softsgn(MLP_u(\mathcal{N}_{ui})),\ \mathbf{d}_j = softsgn(MLP_v(\mathcal{N}_{vj})) \end{aligned} \tag{6}$$

where $\mathbf{1}$ denotes all one vector and $\mathbf{I}$ is the identity matrix. $\mathbf{B}^T\mathbf{1} = 0$ and $\mathbf{B}^T\mathbf{B} = m\mathbf{I}$ are to make sure that the binary codes are expressive. Specifically, the constraint $\mathbf{b}_i^T\mathbf{1} = 0$ is to enforce each bit of $\mathbf{b}_i$ to be activated 50% of the time and the constraint $\mathbf{B}^T\mathbf{B} = m\mathbf{I}$ makes sure that the bits are uncorrelated. θ_u and θ_v are the neural network parameters that in MLP_u and MLP_v.

After learning $\mathbf{B}$ and $\mathbf{D}$, the binary codes can be obtained as $sgn(\mathbf{B})$ and $sgn(\mathbf{D})$ with little information loss, because the elements of the matrices $\mathbf{B}$ and $\mathbf{D}$ are already close to 1 or -1. In addition, the proposed framework exploits MLP_u MLP_v and takes into the consideration of the contextual information, which makes it promising in learning better binary codes for the recommendation.

4. An optimization framework

The objective function in Eq. (6) involves the discrete optimization on the constraints and the training of the neural networks, which is difficult to update the parameters jointly. Therefore, following previous work [41], we adopt Alternating Direction Method of Multiplier (ADMM) [42] to update the parameters alternatively. ADMM is a popular method for solving non-convex and constrained optimization problem. It can break the complicated problem into small sub-problems, each of which is then easier to solve. We next give details of using ADMM. Specifically, we introduce auxiliary variables $\mathbf{P}$ and $\mathbf{Q}$ with the constraint $\mathbf{P} = \mathbf{B}$ and $\mathbf{Q} = \mathbf{D}$. Then the objective function is rewritten as

$$\begin{aligned} &\arg\min_{\theta_u,\theta_v,\mathbf{P},\mathbf{Q}} \sum_{ij:S_{ij}>0} (S_{ij} - \mathbf{b}_i^T\mathbf{d}_j)^2 + \lambda_1\|\theta_u\|_F^2 + \lambda_2\|\theta_v\|_F^2 \\ &s.t.\ \ \mathbf{b}_i = softsgn(MLP_u(\mathcal{N}_{ui})),\quad \mathbf{d}_j = softsgn(MLP_v(\mathcal{N}_{vj})) \\ &\quad \mathbf{P}^T\mathbf{1} = 0,\ \mathbf{Q}^T\mathbf{1} = 0,\ \mathbf{P}^T\mathbf{P} = m\mathbf{I}, \mathbf{Q}^T\mathbf{Q} = n\mathbf{I} \\ &\quad \mathbf{P} = \mathbf{B},\quad \mathbf{Q} = \mathbf{D} \end{aligned} \tag{7}$$

With these two auxiliary variables, the ADMM objective function is given as

$$\begin{aligned} &\arg\min_{\theta_u,\theta_v,\mathbf{P},\mathbf{Q},\mathbf{E},\mathbf{F}} \sum_{ij:S_{ij}>0} (S_{ij} - \mathbf{b}_i^T\mathbf{d}_j)^2 + Tr(\mathbf{E}^T(\mathbf{P} - \mathbf{B})) + Tr(\mathbf{F}^T(\mathbf{Q} - \mathbf{D})) \\ &+ \frac{\mu}{2}(\|\mathbf{P} - \mathbf{B}\|_F^2 + \|\mathbf{Q} - \mathbf{D}\|_F^2) + \lambda_1\|\theta_u\|_F^2 + \lambda_2\|\theta_v\|_F^2 \\ &s.t.\ \ \mathbf{b}_i = softsgn(MLP_u(\mathcal{N}_{ui})),\quad \mathbf{d}_j = softsgn(MLP_v(\mathcal{N}_{vj})), \\ &\quad \mathbf{P}^T\mathbf{1} = 0, \mathbf{Q}^T\mathbf{1} = 0,\quad \mathbf{P}^T\mathbf{P} = m\mathbf{I}, \mathbf{Q}^T\mathbf{Q} = n\mathbf{I} \end{aligned} \tag{8}$$

where $\mathbf{E}$ and $\mathbf{F}$ are the Lagrangian multipliers and μ is a scalar to control the penalty for the violation of equality constraint $\mathbf{P} = \mathbf{B}$ and $\mathbf{Q} = \mathbf{D}$.

4.1. Updating weights of MLP_u and MLP_v

To update the weights of MLP_u and MLP_v, we fix the other terms except the weights of MLP_u and MLP_v and remove the irrelevant terms. Then Eq. (8) becomes:

$$\begin{aligned} &\arg\min_{\theta_u,\theta_v} \sum_{ij:S_{ij}>0} (S_{ij} - \mathbf{b}_i^T\mathbf{d}_j)^2 - Tr(\mathbf{E}^T\mathbf{B}) - Tr(\mathbf{F}^T\mathbf{D}) \\ &+ \frac{\mu}{2}(\|\mathbf{P} - \mathbf{B}\|_F^2 + \|\mathbf{Q} - \mathbf{D}\|_F^2) + \lambda_1\|\theta_u\|_F^2 + \lambda_2\|\theta_v\|_F^2 \\ &s.t. \mathbf{b}_i = softsgn(MLP_u(\mathcal{N}_{ui})),\ \mathbf{d}_j = softsgn(MLP_v(\mathcal{N}_{vj})) \end{aligned} \tag{9}$$

After merging the terms, the objective function can be written as :

$$\begin{aligned} J_{(\theta_u,\theta_v)} = &\sum_{ij:S_{ij}>0} (S_{ij} - \mathbf{b}_i^T\mathbf{d}_j)^2 + \frac{\mu}{2}\|\mathbf{B} - (\mathbf{P} + \frac{1}{\mu}\mathbf{E})\|_F^2 \\ &+ \frac{\mu}{2}\|\mathbf{D} - (\mathbf{Q} + \frac{1}{\mu}\mathbf{F})\|_F^2 + \lambda_1\|\theta_u\|_F^2 + \lambda_2\|\theta_v\|_F^2 \\ s.t.\ \ &\mathbf{b}_i = softsgn(MLP_u(\mathcal{N}_{ui})), \mathbf{d}_j = softsgn(MLP_v(\mathcal{N}_{vj})) \end{aligned} \tag{10}$$

Now we have reduced the problem to the standard training of neural networks with the cost function given in Eq. (10). We use ADAM [43] optimizer to train the neural networks. ADAM is a method for efficient stochastic optimization with adaptive learning rate that only requires first-order gradients with little memory requirement, which is widely used for training deep neural networks.

4.2. Update **P**

Similarly, to update **P**, we fix the other variables except **P** and remove the irrelevant terms, which result in:

$$\begin{aligned} &\arg\min_{\mathbf{P}} Tr(\mathbf{E}^T\mathbf{P}) + \frac{\mu}{2}\|\mathbf{P}-\mathbf{B}\|_F^2 \\ &s.t. \quad \mathbf{P}^T\mathbf{1}=0, \quad \mathbf{P}^T\mathbf{P}=m\mathbf{I} \end{aligned} \tag{11}$$

Note that $\mathbf{P}^T\mathbf{P}=m\mathbf{I}$, we have $\|\mathbf{P}\|_F^2=m^2$. Then the above equation can be written into a more compact form as

$$\arg\min_{\mathbf{P}} Tr(\mathbf{P}^T\mathbf{T}_1) \quad s.t. \quad \mathbf{P}^T\mathbf{1}=0, \quad \mathbf{P}^T\mathbf{P}=m\mathbf{I} \tag{12}$$

where $\mathbf{T}_1 = \mathbf{B} - \frac{1}{\mu}\mathbf{E}$. Eq. (12) has a closed form solution by the following lemma.

Lemma 1. *Given a non-negative matrix* $\mathbf{T}$*,*

$$\arg\min Tr(\mathbf{X}^\mathbf{T}\mathbf{T}) \quad s.t. \quad \mathbf{X}^T\mathbf{1}=0, \quad \mathbf{X}^T\mathbf{X}=m\mathbf{I} \tag{13}$$

it is a monotonous function, according to [44], it is a convergence process using Schmidt gram orthogonalization when solving the problem. The closed form solution is

$$\mathbf{X} = \sqrt{m}(sg([\mathbf{1};\mathbf{T}]))[2:m] \tag{14}$$

where sg denotes the Schmidt gram orthogonalization.

With Lemma 1, the optimal solution for **P** is given as:

$$\mathbf{P} = \sqrt{m}(sg([\mathbf{1};\mathbf{T}_1]))[2:m] \tag{15}$$

4.3. Update **Q**

By removing the terms that are irrelevant to **Q**, we arrive at:

$$\begin{aligned} &\arg\min_{\mathbf{Q}} Tr(\mathbf{F}^T\mathbf{Q}) + \frac{\mu}{2}\|\mathbf{Q}-\mathbf{D}\|_F^2 \\ &s.t. \quad \mathbf{Q}^T\mathbf{1}=0, \quad \mathbf{Q}^T\mathbf{Q}=n\mathbf{I} \end{aligned} \tag{16}$$

Similarly, the above equation can be rewritten as follows by completing the square quadratic equation:

$$\arg\min_{\mathbf{Q}} Tr(\mathbf{Q}^T\mathbf{T}_2) \quad s.t. \quad \mathbf{P}^T\mathbf{1}=0, \quad \mathbf{Q}^T\mathbf{Q}=n\mathbf{I} \tag{17}$$

where $\mathbf{T}_2 = \mathbf{D} - \frac{1}{\mu}\mathbf{F}$. With Lemma 1, we have the closed form update rule for **Q** as:

$$\mathbf{Q} = \sqrt{n}(sg([\mathbf{1};\mathbf{T}_2]))[2:n] \tag{18}$$

4.4. Update **E**, **F** and μ

After updating the variables, we now need to update the ADMM parameters, i.e., **E**, **F** and μ. According to Boyd et al. [42], they are updated as follows:

$$\begin{aligned} &\mathbf{E} = \mathbf{E} + \mu(\mathbf{P}-\mathbf{B}) \\ &\mathbf{F} = \mathbf{F} + \mu(\mathbf{Q}-\mathbf{D}) \\ &\mu = \min(\rho\mu, \mu_{max}) \end{aligned} \tag{19}$$

where $\rho > 1$ is a parameter to control the convergence speed and μ_{max} is to prevent μ become too large.

4.5. Training algorithm

With these updating rules, a training algorithm is summarized in Algorithm 1. In Line 1, we first randomly initialize the parameters **P**, **Q**, **E** and **F**. For the parameters of MLP_u and MLP_v, following the common way to initialize the parameters of neural networks, the output length of the first layer is set to 128, and the second layer is set to the binary code length which ranges from {8, 16, 32, 64} in different experiment cases. From Line 3 to Line 6, we update the parameters alternatively. The convergence of ADMM [45] guarantees the convergence of the algorithm.

Algorithm 1 Training algorithm of the DCH framework.

Input: $\{S_{ij}|i,j \in V\}$: the rating sequences of users and items, r: code length
Output: $\mathbf{B} \in \{\pm1\}^{r\times m}$: user codes, $\mathbf{D} \in \{\pm1\}^{r\times n}$: item codes.
1: Initialize the parameters θ_u, θ_v, **P**, **Q**, **E**, **F**
2: **repeat**
3: Using ADAM to update the parameters of the neural networks, i.e., θ_u, θ_v with the cost function given in Eq. (10)
4: Update **P** using Eq. (15).
5: Update **Q** using Eq. (18).
6: Update **E**, **F** and μ using Eq. (19).
7: **until** convergence
8: **return** **B**, **D**

5. Out-of-sample extension

When new users, items, and new ratings come in, it is impractical to retrain the DCH framework for obtaining hashing codes of these out-of-sample data. Instead, an economical way is to learn ad-hoc codes for new data online and then update for the whole data off-line when possible. In this section, we propose efficient ways to update existing representations or learn new user/item representations. Note that for out-of-sample cases, the model is already trained. The parameters of the neural networks are fixed. Generally, there are three cases in total, which are *new ratings given by existing user to existing items (EUEI)*, *new ratings given by existing user to new items (EUNI)* and *new ratings given by new users to existing item (NUEI)*.

Next, we will give details of how to efficiently deal with these situations, respectively.

5.1. EUEI

When an existing user gives ratings to existing items that are not in the historical data, then these new ratings can also be used to learn a user's preferences. Since both the user and items are in the historical data, the representation of the user and items have already learned. Generally, we can have two choices, one is to make recommendations based on current parameters, and to see how the generalization of the proposed model is, and this solution is named *EUEI1*.

However, *EUEI1* does not take into consideration new ratings, which may help learn better user/item latent features. The other solution, which is named *EUEI2*, is to make slight update over the new rating scores, and learn the new binary codes $\hat{\mathbf{b}}_i$ and $\hat{\mathbf{d}}_j$ for the existing user and existing item respectively. The problem in this case is if we retrain the model based on those new ratings, how to ensure the learned code not to change so much. We fix the parameters of the neural networks and only update the latent features of involved users and items. The essential idea is that the new latent features should be close to the original features and at the same time these new latent features should reflect the new ratings. Let $\mathcal{U}_i$ be a set of new ratings given by user i to existing items, and we have modeled it as

$$\begin{aligned} &\arg\min_{\hat{\mathbf{b}}_i} \sum_{j\in\mathcal{U}_i}(S_{ij}-\hat{\mathbf{b}}_i^T\mathbf{d}_j)^2 + \alpha\|\hat{\mathbf{b}}_i-\mathbf{b}_i\|_2^2 \\ &\quad s.t. \quad \hat{\mathbf{b}}_i \in \{\pm1\}^{r\times1}, \hat{\mathbf{b}}_i^T\mathbf{1}=0 \end{aligned} \tag{20}$$

where $\hat{\mathbf{b}}_i$ is the new representations we want to learn. α is a large scalar to control how close the new representation should be to the old representation. The above equation can be solved using the discrete optimization technique proposed in [10]. As solving the equation is not the focus of the paper, we omit the detail here.

5.2. EUNI & NUEI

In the case of *EUNI*, the binary codes of existing users are know. When a new item v_k is introduced, let $\mathcal{V}_k$ be the set of ratings given by existing users. We want to learn the binary code for v_k. There are two choices. The first choice is to keep all of the existing user representation fixed and only learn the new item binary code. We name this *EUNI1* and the objective function is given as.

$$\begin{aligned} &\arg\min_{\hat{\mathbf{d}}_k} \sum_{i\in\mathcal{V}_k} (S_{ik} - \mathbf{b}_i^T \hat{\mathbf{d}}_k)^2 \\ &s.t. \quad \hat{\mathbf{d}}_k \in \{\pm 1\}^{r\times 1}, \hat{\mathbf{d}}_k^T \mathbf{1} = 0 \end{aligned} \tag{21}$$

The other choice is to also update the users who bought this new item. We name it *EUNI2* and the objective function is given as.

$$\begin{aligned} \arg \min_{\hat{\mathbf{d}}_k, \hat{\mathbf{b}}_i, i\in\mathcal{V}_k} &\sum_{i\in\mathcal{V}_k} (S_{ik} - \hat{\mathbf{b}}_i^T \hat{\mathbf{d}}_k)^2 + \alpha \sum_{i\in\mathcal{V}_k} \|\hat{\mathbf{b}}_i - \mathbf{b}_i\|_2^2 \\ s.t. &\hat{\mathbf{d}}_k \in \{\pm 1\}^{r\times 1}, \hat{\mathbf{d}}_k^T \mathbf{1} = 0, \hat{\mathbf{b}}_i \in \{\pm 1\}^{r\times 1}, \hat{\mathbf{b}}_i^T \mathbf{1} = 0 \end{aligned} \tag{22}$$

where $\hat{\mathbf{d}}_k$ is the binary code for the new item v_k and $\hat{\mathbf{b}}_i$ are the new binary code for the involved users.

Similarly, there are also two choices for *NUEI*, where *NUEI1* only learn the binary representation of a new user while *NUEI2* learn the binary representation of a new user and update the representation of involved items. Let u_k be the new user and $\mathcal{U}_k$ be the set of items bought by the new user. Then the objective function of *NUEI1* is given as

$$\begin{aligned} &\arg\min_{\hat{\mathbf{b}}_k} \sum_{j\in\mathcal{U}_k} (S_{kj} - \hat{\mathbf{b}}_k^T \mathbf{d}_j)^2 \\ &s.t. \quad \hat{\mathbf{b}}_k \in \{\pm 1\}^{r\times 1}, \hat{\mathbf{b}}_k^T \mathbf{1} = 0 \end{aligned} \tag{23}$$

where $\hat{\mathbf{b}}_k$ is the binary code for the new user u_k. Similarly, the objective function for *NUEI2* is given as

$$\begin{aligned} \arg \min_{\hat{\mathbf{b}}_k, \hat{\mathbf{d}}_j, j\in\mathcal{U}_k} &\sum_{j\in\mathcal{U}_k} (S_{kj} - \hat{\mathbf{b}}_k^T \hat{\mathbf{d}}_j)^2 + \alpha \sum_{j\in\mathcal{U}_k} \|\hat{\mathbf{d}}_j - \mathbf{d}_j\|_2^2 \\ s.t. &\hat{\mathbf{b}}_k \in \{\pm 1\}^{r\times 1}, \hat{\mathbf{b}}_k^T \mathbf{1} = 0, \hat{\mathbf{d}}_j \in \{\pm 1\}^{r\times 1}, \hat{\mathbf{d}}_j^T \mathbf{1} = 0 \end{aligned} \tag{24}$$

where $\hat{\mathbf{b}}_k$ is the binary code for the new user u_k and $\hat{\mathbf{d}}_j$ are the new binary code for the involved items.

6. Experiments

In this section, we conduct experiments to evaluate the effectiveness of the proposed framework. Specifically, we aim to answer the following questions:

- How does DCH perform as compared to other state-of-the-art hashing for CF methods?
- Does DCH generalize well to new ratings?
- How much quantization loss in the hashing?
- Compared with the *tanh*, how well of the activation function *softsgn* works in DCH during the learning?
- How many MLP layers are needed for DCH in feature extraction from the users and items?

We will first introduce the datasets followed by the experimental settings; we then conduct experiments to answer the five research questions.

6.1. Datasets

We use three widely used publicly available datasets from real-world online websites, which includes MovieLens, Amazon, and Yelp. The details of the datasets are given as follows:

Table 1
Statistics of the datasets in evaluation.

Dataset	Users#	Items#	Interaction #	Sparsity
MovieLens	6,040	3,704	994,169	95.56%
Amazon	36,327	30,774	1,048,246	99.91%
Yelp	77,018	74,454	2,140,741	99.96%

- **MovieLens**: This is a classical movie rating dataset [46], which contains 6040 users and 3704 items with 994,169 ratings in total.
- **Amazon**: The amazon dataset that we used contains reviews from the category of Movies and TV,[4] which contains 123,960 users, 50,052 items and 1,697,533 ratings in total.
- **Yelp**: The Yelp dataset that we used is from the latest Yelp challenge,[5] which originally contains 1,183,361 users, 156,637 items and 4,736,897 ratings in total.

Note that for all the three datasets, the timestamps that the ratings are given are also available, which makes it possible to generate inputs for DCH. Due to the sparsity of Amazon and Yelp datasets, following the common way [10], we filter out users and items whose number of ratings are smaller than 10. The statistics of the datasets after preprocessing are shown in Table 1.

6.2. Compared methods

We compare with representative and state-of-the-art hashing methods for collaborative filtering, which are listed as follows:

- **CH:** Collaborative Hashing [9] is a popular two-stage hashing method, where the first stage applies the matrix-factorization to learn user and item feature, which are then quantized to binary codes at the second stage. It is a competitive method in binary code learning. It was originally proposed for visual features. Following [10], we implemented CH for collaborative filtering as: $\arg\min_{\mathbf{U},\mathbf{V}} \|\mathbf{S} - \mathbf{U}^T\mathbf{V}\|_F^2$, *s.t.*, $\mathbf{U}\mathbf{U}^T = n\mathbf{I}$, $\mathbf{V}\mathbf{V}^T = m\mathbf{I}$. The binary codes are then obtained as $sgn(\mathbf{U})$ and $sgn(\mathbf{V})$.
- **LCH:** Laplacian Co-Hashing [47] treats the targets (the pairwise value (e.g., user and item, term and document)) as the bipartite graph, then based on the graph Laplacian, the Laplacian Eigenmap [48] is applied during the binary code learning.
- **DCF:** Discrete Collaborative Filtering [10] is a standard method in binary code learning for the user and item, and it is a one stage-learning process based on the collaborative filtering, and their optimization is over the discrete value directly.

6.3. Top-K recommendation performance comparison

To answer the first question which is how DCH performs as compared to other state-of-the-art hashing for CF methods, we perform Top-K recommendation to evaluate the effectiveness of the proposed framework compared with the other methods. The *leave-one-out* evaluation is adopted [4,49]. The three datasets are prepared in the following way. For each user, we first sort the item the user rated according to the timestamp. In the training phase, we use all the rating history except the last one for training the model. In the evaluation process, the last item that bought by the user is selected as the target item. We also sample one hundred items from the set of items that are never bought by the user as negative samples for testing. Following the common way to measure the performance of Top-K recommendation [10], we adopt the widely used evaluation metrics, i.e., HIT@K and NDCG@K. Hit counts the percentage of the returned Top-K items actually

[4] http://jmcauley.ucsd.edu/data/amazon.
[5] http://yelp.com/dataset.

contains the target item. NDCG gives a higher score if the target item is ranked higher. The larger HIT and NDCG are, the better the performance is.

The parameters of the compared methods are set through the cross-validation on the training data. Specifically, for the proposed framework, the batch size is 256. From the empirical results, the embedding lengths for the user l_u and item l_v are 100 which leads a better recommendation besides the small training time which can be seen from Fig. 2(a)–(c). From Fig. 2(d)–(e), we find that there is a slight difference between minimum and maximum. When the length of the user sequence m is set to be 80 and the length of the item sequence n is set to be 50 will get the better prediction.

The hyper-parameter ρ is 1.1 initially, and μ_{max} is 1000. The initial learning rate is set to be 5e-4, which will be updated adaptively by ADAM. The model is implemented in TensorFlow [50], and all of the codes are running on the GPU cluster.[6] We vary the code length as {8, 16, 32, 64} to understand how the model performances under different bit lengths. We also vary the value of K (in Top-K) to {5, 10, 15, 20}, which is to give a comprehensive understanding of the performance. The performances in terms of HIT@K and NDCG@K on the three datasets are shown in Fig. 3. From those figures, we make the following observations:

1. As K increases, the performance in terms of HIT@K and NDCG@K also increase for all the methods tested, which is in line with the intuition that when K is bigger, the probability of the target in the top-K is bigger. In addition, the line slope of our model is bigger than that of the other models in most of the cases. Especially, in the case of the MovieLens that has 8 binary code bits, we can see that the increment of our model is the biggest among other methods, though the accuracy is no better than CH when the K is no bigger than 15. From this perspective, we can see that the places of the targets that predicted by DCH are more likely in the top range of the recommendation.
2. Generally, the proposed DCH framework performs best in all the three datasets when using HIT as the evaluation metric. Significantly, our model performs better when the code length is large, that is to say, our model is more distinguishable in long length code, especially when there is a large number of the data, like Yelp and Amazon. That indicates the high expressiveness of the binary codes learned from DCH which extracting the feature from the users and items with MLPs respectively.
3. Our model performs best in all of the cases when using NDCG as the evaluation metric, especially over the Amazon and Yelp datasets, the results almost have about 10% improvement. That is to say, the position in the ranked list of the target item predicted by our model usually is higher than that predicted by other three models, which implies the effectiveness of the proposed framework.
4. For better comparison of how the performance changes as code length become longer, the performance in terms of HIT@20 and NDCG@20 *w.r.t.* the binary code length is shown in Fig. 4. From these two figures, we can see that as binary code length increases, the performance of the proposed framework tends first to increase and then become stable or slightly decrease. This is because a short code length is not expressive while a too long code length may result in overfitting for our model. Another observation is that the proposed framework has better performance under different code lengths compared with other methods, generally.

6 Supported by Parallel & Distributed Systems Lab in Nanyang Technological University.

Table 2
Results of HIT@20 in the out of samples when the binary code length is 64.

	Cases	MovieLens	Amazon	Yelp
EUEI	*DCF*	37.74%	19.29%	30.26%
	CH	20.87%	23.43%	39.14%
	EUEI1	37.94%	49.05%	51.98%
	EUEI2	44.54%	49.19%	59.49%
EUNI	*DCF*	30.00%	2.17%	10.71%
	CH	50.00%	–	–
	EUNI1	50.00%	63.24%	47.40%
	EUNI2	60.00%	64.61%	52.67%
NUEI	*DCF*	14.91%	15.31%	15.38%
	CH	19.77%	–	–
	NUEI1	40.44%	21.63%	34.91%
	NUEI2	42.66%	29.63%	35.50%

Table 3
Results of NDCG@20 in the out of samples when the binary code length is 64.

	Cases	MovieLens	Amazon	Yelp
EUEI	*DCF*	10.08%	4.68%	2.83%
	CH	5.43%	5.95%	10.0%
	EUEI1	14.59%	22.01%	23.35%
	EUEI2	18.87%	19.54%	28.98%
EUNI	*DCF*	5.94%	0.5%	2.45%
	CH	11.47%	–	–
	EUNI1	12.02%	28.84%	22.64%
	EUNI2	20.47%	29.93%	20.40%
NUEI	*DCF*	3.60%	3.80%	3.82%
	CH	4.98%	–	–
	NUEI1	14.84%	7.63%	13.10%
	NUEI2	14.78%	10.48%	14.02%

To sum up, generally, DCH has better performance under different experiment settings, which shows that by exploiting neural network, contextual information of the users and items can be considered, which makes DCH learn more effective binary codes.

6.4. Out-of-sample performance comparison

To answer the second question about the DCH generalization to new ratings, we investigate the model generalization ability in processing the out of scope data. First, we introduce how the data are prepared for training and testing as follows

- In case of *EUEI*, the out-of-sample data is built by selecting the half items that bought by the user and making sure all of the selected items have appeared in the remaining part which will be the training data. The last item purchased by the user from the out-of-sample data is chosen as the target point in the test data. Then a hundred negative items which are never bought by the user are sampled to build the test data.
- In case of *EUNI*, each dataset is divided into two parts in average according to the user, then we select one part as the training data, and the other part is the out-of-sample data. Same as the previous step, the last item purchased by the user from the out-of-sample data is selected as the target point in the test data. Then a hundred negative items which are never bought by the user are sampled to build the test data.
- And the data are prepared symmetrically in the case of *NUEI* compare that in *EUNI*.

From the results of the previous section, when the binary code length is set to 64, all of the models achieve their best results. Hence, in the next two groups of experiments, the binary code length is set to 64.

Firstly, all of the out-of-sample cases of our proposed model are compared, and the results are listed in Tables 2 and 3 when Top-K is set to 20. From those two tables, we can see that the results are

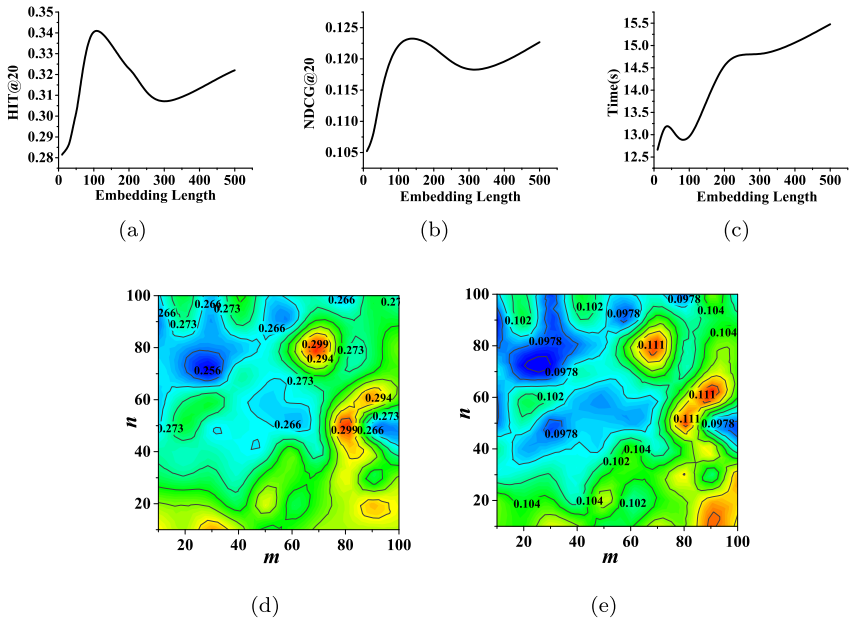


Fig. 2. (a)–(c) are the HIT@20, NDCG@20 and the training time with different embedding length in MovieLens. And the time is for one epoch. (d)–(e) are the skeleton map of the HIT@20 and NDCG@20 respectively *w.r.t.* m and n ranging from 10 to 100 in MovieLens.

different in different cases. First, the proposed models are better than the state-of-the-art. That means the neural networks in our models can capture the feature of the users and items from their profiles in any out of sample cases. In the *EUEI*, *EUNI* and *NUEI*, the second solutions are better than the first one in most of the cases, especially in the evaluation of HIT in Table 2, *EUEI2*, *EUNI2*, and *NUEI2* are higher than *EUEI1*, *EUNI1* and *NUEI1* in all of the datasets. And these are almost the same when the evaluation is NDCG in Table 3. From those points, we can see that the second solution which adds the slight different constraints help the model change with the new item and user.

Then we pick up one out-of-sample case *EUNI2* when Top-K varies from $\{5, 10, 15, 20\}$ to make full comparisons. And the results are illustrated in Fig. 5. From those results, we can see that the proposed model in out-of-sample case *EUNI2* still can achieve the best results in most of the cases, especially when the dataset is large which helps the neural network learn more features from the user and item, and in the measurement of NDCG, our model *EUNI2* achieves almost 5% improvement. While in the case of the HIT over the MovieLens, CH is competitive with our model, and this also reflects from Fig. 3. That mainly because CH applied the projection-based linear hash functions when learning the binary codes which seems like one neural networks layers in our model.

Because in the cases of *EUNI, NUEI*, and *EUNI*, some items or users have no data during the training, and the matrix in that cases cannot be used in decomposition, and that caused the empty blank in Tables 2 and 3 and no lines about LCH Fig. 5.

6.5. Binary codes v.s. real number codes

To answer the third question about how much quantization loss in the hashing, we validate how much improvement of the performance will be if making the recommendation with real number codes by deleting the hashing procedure in DCH, the experiments with these two different codes are conducted. The comparison results can be concluded from the red lines with the asterisk and black lines with square scatter in Fig. 6. From those lines, we can see that there is a smaller gap between the binary code and real number code that means there is little information leaking during the binary code hashing in our model. Furthermore, from the dataset of MovieLens, using the real number codes making the recommendation is no better than that using the binary codes. And this also happens in the other two datasets when evaluating the HIT at Top-20. From the dataset description in Table 1, we know that MovieLens has the smallest number of user and item, that is to say when there is a small number of user and item, the denoising process of our model is highlighted, while when the number of users and items are large, the real number codes will contain more information than the binary codes, and will have little quantization loss during the codes hashing.

6.6. Activation function discussion

To answer the fourth question about how well of the activation function *softsgn* works in DCH during the learning, we replace the *softsgn* in the last layer with *tanh*, whose range is also $[-1, -1]$, and conduct the comparison experiments. The results can be seen from the blue lines with triangle scatter and black lines with square scatter in Fig. 6. From those lines, we can see that all of the blue lines with triangle scatter are under the black lines with square scatter. That is to say that the activation function *softsgn* is more suitable for the code hashing in the binary code learning.

6.7. The layers of the MLPs

To answer the fifth question about that how many layers of MLP are needed for DCH in feature extraction from the users and items, the experiments with different layers of MLP are conducted. The HIT@20, NDCG@20 and the parameters size of the framework are listed in Table 4. MLP-0 means that there are no MLPs in DCH, and the embedding of the users and items are fed into the final layer directly, MLP-1 implies that there is one MLP layer in DCH, etc. Params denotes the parameter size in different cases. As we can see that, if there is no MLP layer, the results are the worst on all of the datasets, which means the deep learning is necessary for DCH during the feature extraction. As along with the increase of the MLP layers, the parameters sizes increase linearly, which means more parameters needed to be trained. More parameters result in the more accurate recommendation, which is line with the intuition. And this also is validated in [4]. From the results, we can see that

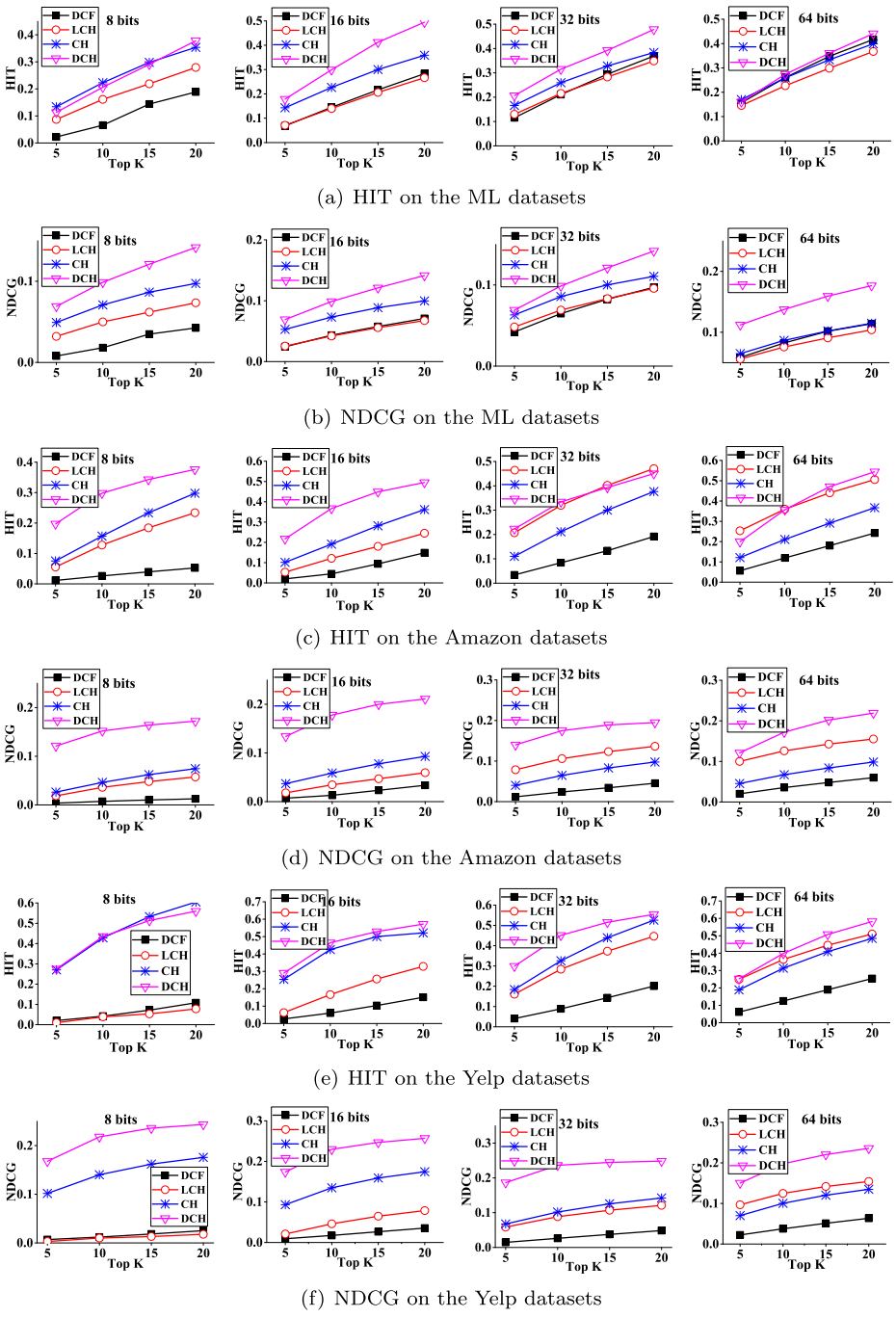


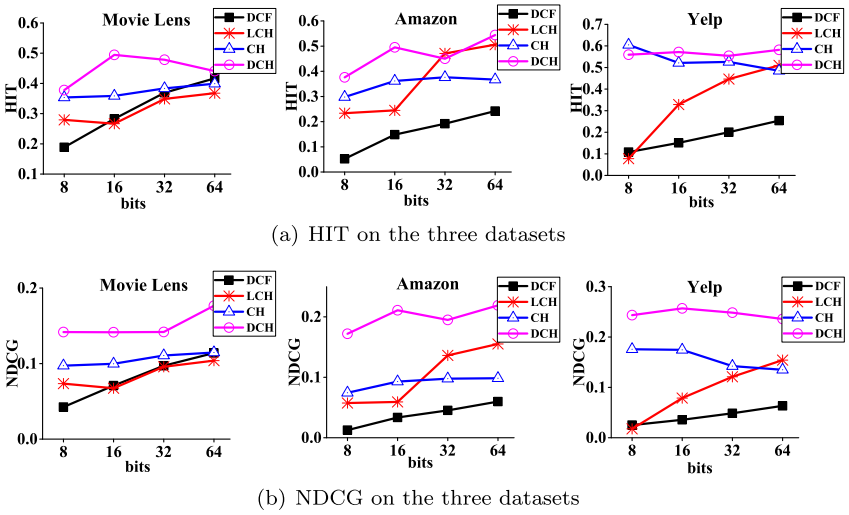
Fig. 3. The results of the Top-K recommendation accuracy with HIT and NDCG where K ranges from {5, 10, 15, 20} on the three datasets.

HIT@20 and NDCG@20 in MLP-2 are far better than that in MLP-1 and MLP-0. Although the results in MLP-2 are not as good as that in MLP-3 and MLP-4, they are close. To balance the parameter sizes and the recommendation efficiency, also for the framework simply, the two-layer MLPs are adopted in DCH. Furthermore, different number layers MLP can be adopted as needed.

6.8. Time complexity and convergence

To compare the time complexity, we show the time cost for the hashing 6000 new users in the dataset of Yelp in Table 5. To make the comparison fairly, the time about the SVD and Non-negative matrix factorization (NMF) is learning the 6000 new users' latent representation whose length range also is {8, 16, 32, 64}. All of the codes are run over the supported cluster. We can see that CH takes the shortest time which mainly counts on its simple design. Then comes to LCH which needs two times of SVD during the matrix factorization, and NMF takes the longest time. Because DCF getting the binary code bit by bit, it is slower than CH and LCH. The proposed model DCH takes the shorter time than DCF with the similar objective function. Although DCH is slower than LCH and CH, we believe it is still acceptable as the proposed model provides the better performance.

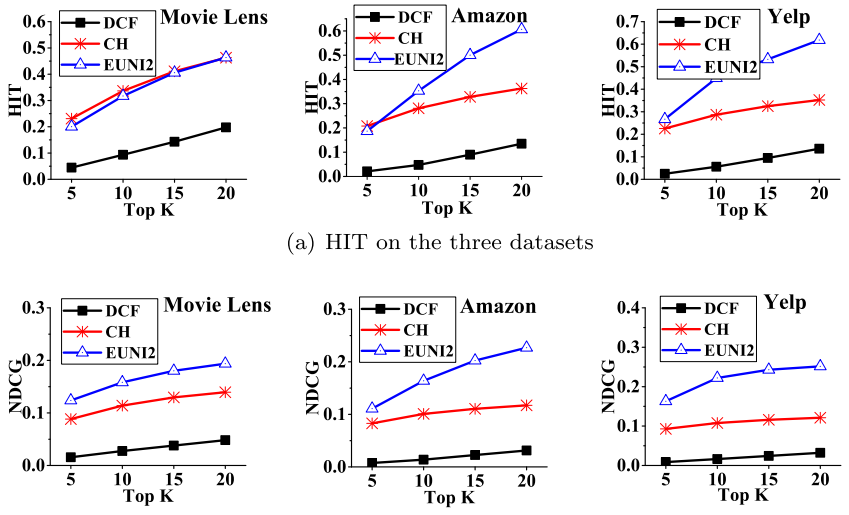
The evidence of the model convergence is shown in Fig. 7. From the figure, we can see that the proposed model is converging empirically.



(a) HIT on the three datasets

(b) NDCG on the three datasets

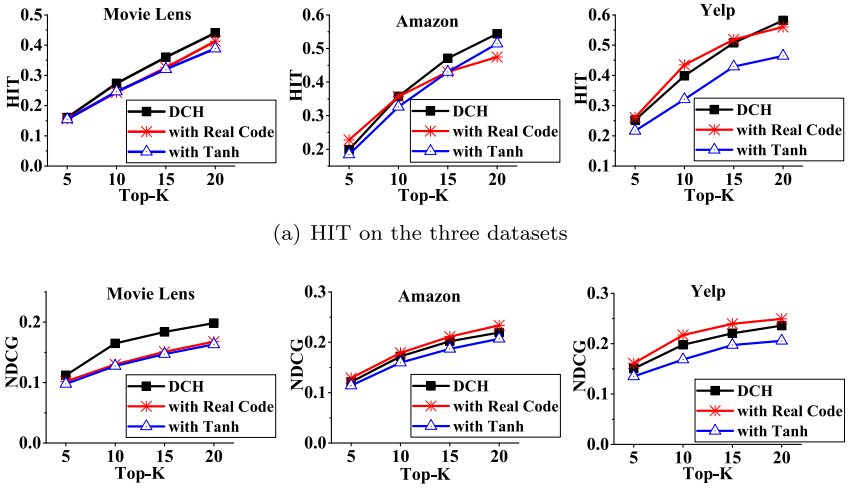
Fig. 4. The results of the HIT@20 and NDCG@20 *w.r.t.* the length of the binary code bits on the three datasets.



(a) HIT on the three datasets

(b) NDCG on the three datasets

Fig. 5. The results of the EUNI2 Top-*K* recommendation accuracy with HIT and NDCG where *K* ranges from {5, 10, 15, 20} on the three datasets, and the binary code length is 64.



(a) HIT on the three datasets

(b) NDCG on the three datasets

Fig. 6. The results of Top-*K* recommendation accuracy with HIT and NDCG where *K* ranges from {5, 10, 15, 20} on the three datasets, and the binary code length is 64.

Table 4
The HIT@20, NDCG@20 and the parameters size with different number MLP layers when binary code length is 64.

Layers	MovieLens			Amazon			Yelp		
	HIT@20	NDCG@20	Params	HIT@20	NDCG@20	Params	HIT@20	NDCG@20	Params
MLP-0	0.2684	0.0996	2.21 MB	0.2568	0.0956	8.63 MB	0.2768	0.1047	18.29MB
MLP-1	0.2749	0.1019	3.36 MB	0.2841	0.1149	9.78 MB	0.3357	0.1407	19.43MB
MLP-2	0.2776	0.1031	4.50 MB	0.3771	0.1436	10.15 MB	0.4620	0.1886	19.81MB
MLP-3	0.2760	0.1023	4.11 MB	0.4115	0.1645	10.53 MB	0.4904	0.2178	20.19MB
MLP-4	0.2978	0.1110	4.49 MB	0.4071	0.1670	10.91 MB	0.4964	0.2197	20.57MB

Table 5
Time cost (s) of the hashing new users over Yelp.

Model/length	8	16	32	64
LCH	0.35	0.38	0.43	0.47
CH	0.04	0.14	0.31	0.61
DCF	0.94	0.78	0.99	1.05
DCH	0.39	0.40	0.45	0.48
SVD	0.94	1.05	1.13	1.02
NMF	1.18	1.22	1.23	1.32

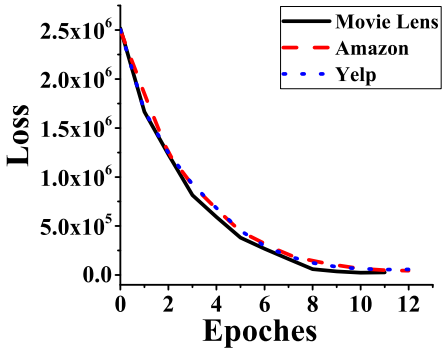


Fig. 7. The evidence convergence of the proposed model.

7. Conclusion

In this paper, we proposed a novel model of DCH, which exploits neural networks over both user and item for learning binary codes. The proposed framework also takes into consideration the orders in which users rate items. In addition, we also provided recommendations on how to deal with out-of-sample cases. Extensive experiments on three real-world datasets demonstrated the effectiveness of the proposed framework for learning efficient binary codes and for dealing with new users, new items, and new ratings. Furthermore, the quantized loss in DCH is acceptable during the hash, and the activation function *softsgn* is fitful for the binary code learning.

There are several interesting directions for future work. Firstly, in this paper, we designed neural networks for learning user and item latent features. We would like to investigate more deep learning algorithms such as LSTM for learning binary codes. Secondly, in this paper, we mainly focused on explicit ratings. We would like to extend the proposed framework for dealing with implicit feedbacks by adopting evaluation methods such as RMSE and MAE. Thirdly, the size of the dataset in this paper is rather limited. We plan to test our findings on a much bigger dataset in the near future.

References

[1] Y. Koren, R. Bell, C. Volinsky, Matrix factorization techniques for recommender systems, Computer 42 (8) (2009) 42–49.
[2] S. Wang, J. Tang, Y. Wang, H. Liu, Exploring implicit hierarchical structures for recommender systems, in: IJCAI, 2015, pp. 1813–1819.
[3] S. Wang, J. Tang, H. Liu, Toward dual roles of users in recommender systems, in: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, ACM, 2015, pp. 1651–1660.
[4] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, T.-S. Chua, Neural collaborative filtering, in: Proceedings of the 26th International Conference on World Wide Web, International World Wide Web Conferences Steering Committee, 2017, pp. 173–182.
[5] S. Wang, J. Tang, Y. Wang, H. Liu, Exploring hierarchical structures for recommender systems, IEEE Trans. Knowl. Data Eng. 30 (6) (2018) 1022–1035.
[6] X. Meng, S. Wang, K. Shu, J. Li, B. Chen, H. Liu, Y. Zhang, Towards privacy preserving social recommendation under personalized privacy settings, World Wide Web (2018) 1–29.
[7] A. Karatzoglou, A. Smola, M. Weimer, Collaborative filtering on a budget, in: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, 2010, pp. 389–396.
[8] K. Zhou, H. Zha, Learning binary codes for collaborative filtering, in: Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2012, pp. 498–506.
[9] X. Liu, J. He, C. Deng, B. Lang, Collaborative hashing, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 2139–2146.
[10] H. Zhang, F. Shen, W. Liu, X. He, H. Luan, T.-S. Chua, Discrete collaborative filtering, in: Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, in: SIGIR '16, ACM, 2016, pp. 325–334.
[11] X. Li, J. She, Collaborative variational autoencoder for recommender systems, in: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2017, pp. 305–314.
[12] A. Gionis, P. Indyk, R. Motwani, et al., Similarity search in high dimensions via hashing, in: Proceedings of the 25th VLDB Conference, 1999, pp. 518–529.
[13] B. Kulis, T. Darrell, Learning to hash with binary reconstructive embeddings, in: Advances in neural information processing systems, 2009, pp. 1042–1050.
[14] J. Song, Y. Yang, Z. Huang, H.T. Shen, R. Hong, Multiple feature hashing for real-time large scale near-duplicate video retrieval, in: Proceedings of the 19th ACM international conference on Multimedia, ACM, 2011, pp. 423–432.
[15] P. Zhang, W. Zhang, W.-J. Li, M. Guo, Supervised hashing with latent factor models, in: Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval, ACM, 2014, pp. 173–182.
[16] F. Shen, C. Shen, W. Liu, H. Tao Shen, Supervised discrete hashing, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 37–45.
[17] C.-Y. Wu, A. Ahmed, A. Beutel, A.J. Smola, H. Jing, Recurrent recommender networks, in: Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, ACM, 2017, pp. 495–503.
[18] V. Erin Liong, J. Lu, G. Wang, P. Moulin, J. Zhou, Deep hashing for compact binary codes learning, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 2475–2483.
[19] Y. Cao, M. Long, J. Wang, Q. Yang, P.S. Yu, Deep visual-semantic hashing for cross-modal retrieval, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2016, pp. 1445–1454.
[20] H. Zhu, M. Long, J. Wang, Y. Cao, Deep hashing network for efficient similarity retrieval, in: Proceedings of the Association for the Advancement of Artificial Intelligence, 2016, pp. 2415–2421.
[21] S. Chaidaroon, Y. Fang, Variational deep semantic hashing for text documents, in: Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 2017, pp. 75–84.
[22] Y. Li, Q. Pan, S. Wang, H. Peng, T. Yang, E. Cambria, Disentangled variational auto-encoder for semi-supervised learning, Inform. Sci. 482 (2019) 73–85.
[23] D. Hu, F. Nie, X. Li, Deep binary reconstruction for cross-modal hashing, IEEE Trans. Multimed. 14(8) (2018) 1–12.
[24] G. Wu, J. Han, Z. Lin, G. Ding, B. Zhang, Q. Ni, Joint image-text hashing for fast large-scale cross-media retrieval using self-supervised deep learning, IEEE Trans. Ind. Electron. (2018) 1–9.
[25] T. Young, D. Hazarika, S. Poria, E. Cambria, Recent trends in deep learning based natural language processing, IEEE Comput. Intell. Mag. 13 (3) (2018) 55–75.

[26] Y. Li, Q. Pan, S. Wang, T. Yang, E. Cambria, A generative model for category text generation, Inform. Sci. 450 (2018) 301–315.
[27] S. Poria, E. Cambria, D. Hazarika, P. Vij, A deeper look into sarcastic tweets using deep convolutional neural networks, in: COLING, 2016, pp. 1601–1612.
[28] I. Chaturvedi, Y.-S. Ong, I. Tsang, R. Welsch, E. Cambria, Learning word dependencies in text by means of a deep recurrent belief network, Knowl.-Based Syst. 108 (2016) 144–154.
[29] Y. Ma, H. Peng, E. Cambria, Targeted aspect-based sentiment analysis via embedding commonsense knowledge into an attentive LSTM, in: AAAI, 2018, pp. 5876–5883.
[30] H. Peng, Y. Ma, Y. Li, E. Cambria, Learning multi-grained aspect target sequence for chinese sentiment analysis, Knowl.-Based Syst. 148 (2018) 167–176.
[31] S. Zhang, L. Yao, A. Sun, Deep learning based recommender system: A survey and new perspectives, arXiv preprint arXiv:1707.07435.
[32] S. Cavallari, V. Zheng, H. Cai, K. Chang, E. Cambria, Learning community embedding with community detection and node embedding on graphs, in: CIKM, 2017, pp. 377–386.
[33] M. Grbovic, V. Radosavljevic, N. Djuric, N. Bhamidipati, J. Savla, V. Bhagwan, D. Sharp, E-commerce in your inbox: product recommendations at scale, in: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2015, pp. 1809–1818.
[34] O. Barkan, N. Koenigstein, Item2vec: neural item embedding for collaborative filtering, in: Machine Learning for Signal Processing (MLSP), 2016 IEEE 26th International Workshop on, IEEE, 2016, pp. 1–6.
[35] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, arXiv preprint arXiv:1301.3781.
[36] Y. Li, Q. Pan, T. Yang, S. Wang, J. Tang, E. Cambria, Learning word representations for sentiment analysis, Cogn. Comput. 9 (6) (2017) 843–851.
[37] W. Pei, J. Yang, Z. Sun, J. Zhang, A. Bozzon, D.M. Tax, Interacting Attention-gated Recurrent Networks for Recommendation, arXiv preprint arXiv:1709.01532.
[38] Y. Wu, C. DuBois, A.X. Zheng, M. Ester, Collaborative denoising auto-encoders for top-n recommender systems, in: Proceedings of the Ninth ACM International Conference on Web Search and Data Mining, ACM, 2016, pp. 153–162.
[39] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2(5) (1989) 359–366.
[40] Y. Li, T. Yang, Word embedding for understanding natural language: a survey, in: Guide to Big Data Applications, Springer, 2018, pp. 83–104.
[41] S. Wang, J. Tang, H. Liu, Embedded unsupervised feature selection., in: Proceedings of the Association for the Advancement of Artificial Intelligence, 2015, pp. 470–476.
[42] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, Distributed optimization and statistical learning via the alternating direction method of multipliers, Found. Trends Mach. Learn. 3 (1) (2011) 1–122.
[43] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980.
[44] A. Ruhe, Numerical aspects of gram-schmidt orthogonalization of vectors, Linear Algebra Appl. 52 (1983) 591–601.
[45] Y. Wang, W. Yin, J. Zeng, Global convergence of ADMM in nonconvex nonsmooth optimization, arXiv preprint arXiv:1511.06324.
[46] F.M. Harper, J.A. Konstan, The movielens datasets: history and context, ACM Trans. Inter. Intell. Syst. (TiiS) 5 (19) (2015) 1–19.
[47] D. Zhang, J. Wang, D. Cai, J. Lu, Laplacian co-hashing of terms and documents, in: ECIR , Vol. 5993, Springer, 2010, pp. 577–580.
[48] M. Belkin, P. Niyogi, Laplacian eigenmaps for dimensionality reduction and data representation, Neural Comput. 15 (6) (2003) 1373–1396.
[49] X. He, H. Zhang, M.-Y. Kan, T.-S. Chua, Fast matrix factorization for online recommendation with implicit feedback, in: Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 2016, pp. 549–558.
[50] M. Abadi, A. Agarwal, P. Barham, et al., TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015.