Efficient Neural Matrix Factorization without Sampling for Recommendation

CHONG CHEN and MIN ZHANG, Tsinghua University, China
YONGFENG ZHANG, Rutgers University, United States
YIQUN LIU and SHAOPING MA, Tsinghua University, China

Recommendation systems play a vital role to keep users engaged with personalized contents in modern online platforms. Recently, deep learning has revolutionized many research fields and there is a surge of interest in applying it for recommendation. However, existing studies have largely focused on exploring complex deep-learning architectures for recommendation task, while typically applying the negative sampling strategy for model learning. Despite effectiveness, we argue that these methods suffer from two important limitations: (1) the methods with complex network structures have a substantial number of parameters, and require expensive computations even with a sampling-based learning strategy; (2) the negative sampling strategy is not robust, making sampling-based methods difficult to achieve the optimal performance in practical applications.

In this work, we propose to learn neural recommendation models from the whole training data without sampling. However, such a non-sampling strategy poses strong challenges to learning efficiency. To address this, we derive three new optimization methods through rigorous mathematical reasoning, which can efficiently learn model parameters from the whole data (including all missing data) with a rather low time complexity. Moreover, based on a simple Neural Matrix Factorization architecture, we present a general framework named ENMF, short for *Efficient Neural Matrix Factorization*. Extensive experiments on three real-world public datasets indicate that the proposed ENMF framework consistently and significantly outperforms the state-of-the-art methods on the Top-K recommendation task. Remarkably, ENMF also shows significant advantages in training efficiency, which makes it more applicable to real-world large-scale systems.

CCS Concepts: • **Information systems** → **Recommender systems**; • **Computing methodologies** → **Neural networks**;

Additional Key Words and Phrases: Matrix factorization, neural networks, implicit feedback, efficient learning, recommendation system

This article is an extension of Chen et al. [4]. Compared with the previous conference version, it introduces two new forms of the previously proposed efficient optimization method and a new efficient neural matrix factorization framework. It also includes extensive experimental assessments of the new methods and compares the efficiency and performance with the state-of-the-art recommendation models. This work is supported by Natural Science Foundation of China (Grants No. 61672311 and No. 61532011) and the National Key Research and Development Program of China (Grant no. 2018YFC0831900). The work is also partially supported by National Science Foundation (Grant No. IIS-1910154).

Authors' addresses: C. Chen, M. Zhang, Y. Liu, and S. Ma, Tsinghua University, Beijing, 100084, China; emails: cc17@mails.tsinghua.edu.cn, z-m@tsinghua.edu.cn, yiqunliu@tsinghua.edu.cn, msp@tsinghua.edu.cn; Y. Zhang, Rutgers University, New Brunswick, New Jersey, 08901, United States; email: yongfeng.zhang@rutgers.edu.

1046-8188/2020/01-ART14 $15.00
https://doi.org/10.1145/3373807

ACM Reference format:
Chong Chen, Min Zhang, Yongfeng Zhang, Yiqun Liu, and Shaoping Ma. 2020. Efficient Neural Matrix Factorization without Sampling for Recommendation. *ACM Trans. Inf. Syst.* 38, 2, Article 14 (January 2020), 28 pages.
https://doi.org/10.1145/3373807

1 INTRODUCTION

Recommender systems provide essential web services on the Internet to alleviate the information overload problem. An effective recommender system not only can facilitate the information seeking process of users but also can help providers display products accurately to the target population. With such an important role, recommendation has become a hot research topic and attracted increasing attention in information retrieval and data-mining communities [3, 23, 58].

The key to personalized recommender systems is in modelling users' preference on items based on their past interactions (e.g., ratings and clicks), known as collaborative filtering. Among various collaborative filtering methods, Matrix Factorization (MF) [26, 29] is the most popular one and has been widely adopted in many real-world applications [32, 46]. MF maps users and items to a shared latent factor space, so that user-item relationships can be captured by their latent factors' dot product. Early work on MF algorithms [27, 28] mainly focused on explicit feedback, where users' ratings that directly reflect their preference on items are utilized. However, later researchers found that this way of modelling only the observed positive feedbacks leads to poor performance of the real top-N recommendation system [8]. Moreover, explicit ratings are not always available in many applications. More commonly, users interact with items through implicit feedback, e.g., users' viewing records and purchase history.

Compared to explicit ratings, implicit feedback is easier to collect but more challenging to utilize, since it is binary and only has positive examples. To solve the problem of lacking negative feedback [37], two learning strategies have been proposed: (1) negative sampling strategy that randomly samples negative instances from the missing data [3, 23, 41]; (2) whole-data-based strategy that treats all the missing data as negative [4, 26, 30]. Both solutions have pros and cons: Negative sampling has controllable efficiency, but its effectiveness may suffer from the low quality of negative examples and slow convergence [40, 41], while modeling all missing data is costly, it can be more effective [4, 25].

Recently, deep learning has made massive strides in many research areas and achieved great performance [18]. The successful integration of deep-learning methods in recommendation systems has demonstrated the advantages of complex network structures over traditional models. However, existing studies have largely focused on exploring newly proposed deep-learning architectures for recommendation task, such as attention mechanisms [2, 5, 58], memory networks [3, 14], Convolutional Neural Network (CNN) [20], Generative Adversarial Networks (GAN) [21, 52], Graph Neural Networks (GNN) [16, 55], and so on. While for model learning, these works typically rely on the negative sampling strategy for efficient training. Despite effectiveness, we argue that existing deep-learning-based recommendation methods suffer from two important limitations: First, the methods with complex network structures have a substantial number of parameters, and require expensive computations even with a sampling-based learning strategy; second, as shown in previous work [23, 53], the performance of negative sampling is not robust as it is highly sensitive to the sampling distribution and the number of negative samples. Essentially, sampling is biased, making it difficult to converge to the same loss with all training examples, regardless of how many update steps have been taken [57].

To address the limitation caused by negative sampling, we propose to perform whole-data-based learning for neural recommendation models. In contrast to sampling, whole-data-based learning does not involve any sampling procedure and computes the gradient over all training data (including all missing data). As such, it can easily converge to a better optimum in a more stable way [26, 60]. Unfortunately, the difficulty in applying whole-data-based learning lies in the expensive computational cost for large-scale data, which makes the straightforward method less applicable to neural models.

Motivated by the above observations, in this work, we enhance (1) the effectiveness of neural recommendation models by performing whole-data-based learning strategy, and (2) the practicability of learning from the whole training data by developing three efficient optimization algorithms. The two significant enhancements of our methods make it easy to address large-scale scenarios with a more expressive modeling on implicit data. To ensure training efficiency, we accelerate the optimization method by reformulating a commonly used square loss function with rigorous mathematical reasoning. Specifically, we perform the optimization on each element of user and item latent vectors, rather than the traditional vector-wise manner [23, 26]. By leveraging the sparsity of implicit data, we successfully update each parameter in a manageable time complexity without sampling. Moreover, based on a simple Neural Matrix Factorization architecture, we present a general framework named ENMF (short for *Efficient Neural Matrix Factorization*), and propose three instantiations—ENMF-U, ENMF-I, and ENMF-A based on the newly derived optimization approaches.

To evaluate the recommendation performance and training efficiency of our proposed methods, we apply ENMF on three real-world datasets with extensive experiments. The results indicate that our ENMF methods consistently and significantly outperform the state-of-the-art methods on Top-K personalized recommendation task, while maintaining the favorable properties of not having compositional parameters. Furthermore, ENMF also shows significant advantages in training efficiency, which makes it more practical in practical E-commerce scenarios. Our main contributions are outlined as follows.

- We propose to learn neural recommendation models without sampling, which is more effective and stable due to the consideration of all samples in each parameter update. Three efficient optimization methods are derived: user-based, item-based, and alternating-based, which solve the challenging problem of learning neural models from the whole data with a controllable time complexity.
- A generic Efficient Neural Matrix Factorization framework (ENMF) is proposed based on the derived learning methods. It complements the mainstream sampling-based neural models for recommendation, providing a new approach to improve recommendation models.
- Extensive experiments are performed on three real-world datasets. The results show that ENMF significantly outperforms the state-of-the-art methods by more than 5.90%, 4.08%, 6.30%, on the three datasets, respectively, while maintaining the favorable properties of not having compositional parameters. Furthermore, ENMF also shows significant advantages in training efficiency, which makes it more applicable to real-world large-scale systems. Codes have been released to facilitate further developments on efficient whole-data-based neural methods.[1]

This work is a revised and extended edition of research that appeared at SIGIR 2019 [4], which is significantly different from its preliminary conference version in the methodology. Specifically, this work approaches a generic problem setting where any scenario with implicit feedback can

[1] https://github.com/chenchongthu/ENMF.

be applied, while the previous work [4] is designed to focus on social-aware recommendations. Moreover, this version extends the designed efficient learning method by adding item-based and alternating-based forms, as well as the corresponding analysis of results, while only the user-based form is discussed in the conference version.

The rest of this article is organized as follows. In the next section, we review related work. Then, we provide some preliminaries of our methods in Section 3. After that, we elaborate our proposed whole-data-based learning methods and Efficient Neural Matrix Factorization framework in Section 4. In Section 5, we report experimental results and corresponding analysis. In Section 6, we discuss the limitation and some extensions of our methods. Finally, we conclude the article and highlight some future directions in Section 7.

2 RELATED WORK

In this article, we study efficient whole-data-based learning for neural recommendation models. Thus, we review the related work of traditional recommendation models, neural recommendation models, and model learning in recommendation.

2.1 Traditional Recommendation Models

Among the various traditional recommendation methods, Matrix Factorization (MF) is the most popular one, and is also the basis of many effective models [42, 48]. Popularized by the Netflix Challenge, early MF methods [29] were designed to model users' explicit feedback by mapping users and items to a latent factor space, such that user-item relationships (ratings) can be obtained by their latent factors' dot product. After that, many research efforts have been devoted to enhancing MF, such as integrating it with neighbor-based models (SVD++) [27] and extending it to Factorization Machines (FM) [39] for a generic modeling of features.

Later on, some researchers found that a well-designed MF model in rating prediction may not perform well in Top-K recommendation, and called on recommendation research to focus more on the ranking task [8]. In this case, Hu et al. [26] proposed a whole-data-based method (WMF), which assumes that all unobserved items are negative samples and are equally weighted. Then several efforts [25, 30] focused on the weighting scheme by considering whether the unobserved items are indeed negative ones. On another line of research, Rendle et al. [41] proposed a pair-wise learning method BPR, which is a sampling-based method that optimizes the model based on the relative preference of a user over pairs of items. Then, the pairwise learning strategy has been widely used to optimize recommender models [3, 5, 23, 50] and become a dominant technique in recommendation.

2.2 Neural Recommendation Models

In the past few years, there is a large literature exploiting different neural networks for improving the performance of recommendation systems. Salakhutdinov et al. [43] proposed a Restricted Boltzmann Machines to predict explicit ratings, which was the first to apply neural network to recommender system. Among the early studies, autoencoder was a popular choice of deep-learning architecture. The autoencoder acts as a nonlinear decomposition of the rating matrix replacing the traditional linear inner product. In previous work, Sedhain et al. [45] designed AutoRec that using an autoencoder followed by reconstruction to directly predict ratings. Wu et al. [56] proposed Collaborative Denoising AutoEncoders (CDAE) that integrating a user-specific bias into an autoencoder. CDAE can be seen as a generalization of many existing collaborative filtering methods. Zhang et al. [61] designed AutoSVD++ that extending the original SVD++ model with a contrastive autoencoder to capture auxiliary item information.

Neural Collaborative Filtering (NCF) [23] addressed implicit feedback by jointly learning a matrix factorization and a feedforward neural network. The outputs are then concatenated before the final output to produce an interaction between the latent factors and the nonlinear factors. The NCF framework has been extended to adapt to different recommendation scenarios [5, 17, 54]. For example, Wang et al. [54] applied NCF to model user-item interaction in both information domain and social domain, Chen et al. [5] combined NCF with attention mechanism to recommend videos and images. Gao et al. [17] extended the architecture of NCF to a multi-task learning framework, which aims to solve the problem of learning recommender systems from multi-behavior data.

Other than for pure implicit data setting, many studies utilized neural network to extract the auxiliary information and features in recommender system, such as textual [2, 64], visual [33, 59], audio [51], and video [7]. Zhang et al. [62] proposed a Joint Representation Learning (JRL) framework to model heterogeneous information sources for recommendation. Recently, it has become a trend to explore the application of newly proposed deep-learning architectures in recommendation. Such as attention mechanisms [2, 5, 58], memory networks [3, 6, 14], Convolutional Neural Network (CNN) [20, 64], Recurrent Neural Network (RNN) [36], Generative Adversarial Networks (GAN) [21, 52], Graph Neural Networks (GNN) [16, 55], and so on.

Although existing neural recommendation models have achieved great success, they either focus on rating prediction task, or just typically apply the negative sampling strategy for model learning. As we have discussed in Section 1, complex deep models have a substantial number of parameters so that require expensive computations, and the negative sampling strategy is not robust in real applications. Compared with these methods, we specially design an efficient neural matrix factorization framework to address the above issues.

2.3 Model Learning in Recommendation

In many real-world applications, the data matrices can be highly sparse. To optimize a recommendation model with implicit feedback, two strategies have been widely used in previous studies, which are: (1) negative sampling strategy [3, 23, 41] and (2) whole-data-based strategy [11, 26, 30, 31].

Negative sampling strategy [3, 23, 41] samples negative instances from missing data. For example, BPR [41] is a sampling-based method that randomly samples negative instances from missing entries, maximizing the margin between the model prediction of observed entries and that of sampled negatives. By negative sampling, the number of negative instances is greatly reduced, therefore the overall time complexity is controllable [23]. However, the downside is that sampling-based methods usually have a slower convergence rate and the performance is highly dependent of the design of the sampler [25, 57]. Whole-data-based strategy [11, 26, 30, 31] sees all the missing data as negative. For example, the WMF method [26] models all missing entries as negative instances with a label of 0, assigning them with a lower weight in point-wise regression learning. The whole-data-based methods model negative instances with a higher coverage, but the downside is that the learning algorithm could be much slower. Existing neural recommendation methods [3, 23, 49, 59] typically rely on negative sampling for efficient optimization. To retain the model's fidelity, we persist in whole-data-based learning in this article, and we develop fast optimization methods to address the inefficiency issue.

Some efforts have been devoted to resolving the inefficiency issue of traditional whole-data-based methods. Most of them are based on Alternating Least Squares (ALS) [26]. For example, Pilaszy et al. [38] described an approximate solution of ALS. He et al. [25] proposed an efficient element-wise ALS with non-uniform missing data. Unfortunately, ALS-based methods are not applicable to neural models, which use Gradient Descent (GD) for optimization. Recently, some

Table 1. A Summary of Key Notations in This Work

Symbol	Description
$\mathbf{U}$	set of users
$\mathbf{B}$	batch of users or items
$\mathbf{V}$	set of items
$\mathbf{R}$	user-item interactions
$\mathcal{R}$	the set of user-item pairs whose values are non-zero
$\mathbf{p}_u$	latent factor vector of user u
$\mathbf{q}_v$	latent factor vector of item v
$\mathbf{h}$	prediction layer
c_{uv}	the weight of entry R_{uv}
d	latent factor number
Θ	set of neural parameters

researchers [57, 60] studied fast Batch Gradient Descent (BGD) methods to learn from all training examples. However, they also only focus on optimizing traditional non-neural models.

Distinct from previous studies, we devise three new flexible whole-data-based learning approaches in this work. To the best of our knowledge, our work is the first study tailored for learning neural recommendation models without sampling.

3 PRELIMINARIES

We first introduce the key notations used in this work, the whole-data-based MF methods, and the weighting strategies for missing data.

3.1 Notations

Table 1 depicts the notations and key concepts used in this article. Suppose we have M users and N items in the dataset, and we use the index u to denote a user, and v to denote an item. The user-item data matrix is denoted as $\mathbf{R} = [R_{uv}]_{M\times N} \in \{0, 1\}$, indicating whether u has purchased or clicked on item v. We use $\mathcal{R}$ to denote the set of observed entries in $\mathbf{R}$, i.e., for which the values are non-zero. Vector $\mathbf{p}_u$ denotes the latent vector of u, and $\mathbf{q}_v$ denotes the latent vector of v. c_{uv} denotes the weight of entry R_{uv}. More details are introduced in Section 4.

3.2 MF Method for Implicit Data

In implicit data, the user-item interactions $\mathbf{R}$ is defined as

$$R_{uv} = \begin{cases} 1, & \text{if interaction (user } u, \text{ item } v\text{) is observed,} \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

For R_{uv}, a value of 1 indicates that there is an interaction between user u and item v, which is taken as a positive instance that u likes v. However, a value of 0 does not necessarily mean u does not like v, it can be consider as the user is not aware of the item. This poses challenges in learning from implicit data. While observed entries reflect users' interest on items, the unobserved entries can be just missing data and there is a natural scarcity of negative feedback.

Matrix Factorization (MF) maps both users and items into a joint latent feature space of d dimension such that interactions are modeled as inner products in that space. Mathematically, each entry R_{uv} of $\mathbf{R}$ is estimated as

$$\hat{R}_{uv} =< \mathbf{p}_u, \mathbf{q}_v >= \mathbf{p}_u^T \mathbf{q}_v. \tag{2}$$

The item recommendation problem is formulated as estimating the scoring function R_{uv}, which is used to rank items.

For implicit data, the observed interactions are rather limited, and non-observed examples are of a much larger scale. To learn model parameters, Hu et al. [26] introduced a weighted regression function, which associates a confidence to each prediction in the implicit feedback matrix **R**:

$$\mathcal{L}(\Theta) = \sum_{u \in \mathbf{U}} \sum_{v \in \mathbf{V}} c_{uv} (R_{uv} - \hat{R}_{uv})^2, \tag{3}$$

where c_{uv} denotes the weight of entry R_{uv}. Note that in implicit feedback learning, missing entries are usually assigned a zero R_{uv} value but non-zero c_{uv} weight.

As can be seen, the time complexity of computing the loss in Equation (3) is $O(|\mathbf{U}||\mathbf{V}|d)$. Clearly, the straightforward way to calculate gradients is generally infeasible, because $|\mathbf{U}||\mathbf{V}|$ can easily reach billion level or even higher in real life.

3.3 Weighting Strategies for Missing Data

There have been many studies on how to assign proper weights for missing data. Here, we discuss four most common strategies:

(1) Zero weight on missing entries. This strategy applies a zero weight on missing entries, i.e., $c_{uv} = 0$ if $R_{uv} = 0$. It is a typical setting for the task of rating prediction [2, 64], which aims to predict the values of missing entries in user-item rating matrix. However, some researchers found that a well-designed model in rating prediction may not perform well in Top-K recommendation [8]. The reason is that the missing entries contain valuable signal about negative instances, and ignoring them will lead to suboptimal performance.

(2) Uniform weight on all entries. This strategy applies a uniform weight of 1 on all data entries. When the number of missing entries are of the same scale as the number of observed entries, such a setting may yield good performance. However, in real-word scenarios, the implicit data is usually very sparse, the observed entries are rather limited, and non-observed examples are of a much larger scale. For such highly imbalanced learning scenarios, a uniform weighting strategy will make the parameter estimation process dominated by the missing entries, resulting in suboptimal performance [24].

(3) Uniform weight on missing entries. This strategy assigns all missing entries with the same weight c_0, which can be different as the weight for observed entries [26]:

$$c_{uv} = \begin{cases} c_1 & \text{if } R_{uv} = 1, \\ c_0 & \text{if } R_{uv} = 0. \end{cases} \tag{4}$$

When dealing with sparse data, c_0 can be set as a smaller number than c_1 to alleviate the imbalanced learning issue. For example, in Reference [26], c_1 is set to a value of 1, while c_0 is a smaller number. Previous studies [12, 26, 53] have demonstrated that this strategy yields better performance than a uniform weight on all entries in recommendation task.

(4) Frequency-based weight on missing entries. This strategy assigns missing entries with non-uniform weights, while the weights dependent on item popularity [25, 30] or user activity [1]. The uniform weight strategy assumes all missing entries provide the same level of negative signal, which severely limits the fidelity for modeling real-world scenarios. For example, a popular item is more likely to be seen by users, thus it should be assigned a higher weight as negative if not clicked [25, 30]. Another reasonable intuition is that the missing entries of active users (who have clicked many items) are more likely to be true negatives [1]. A commonly used frequency-based weighting strategy [25, 57, 60] is as follows:

$$c_{uv} = \begin{cases} c_1 & \text{if } R_{uv} = 1, \\ c_v^- & \text{if } R_{uv} = 0, \end{cases} \tag{5}$$

where c_v^- is defined as

$$c_v^- = c_0 \frac{m_v^x}{\sum_{j=1}^{|\mathbf{V}|} m_j^x};\ m_v = \frac{|\mathcal{R}_v|}{\sum_{j=1}^{|\mathbf{V}|} |\mathcal{R}_j|}, \tag{6}$$

where m_v denotes the frequency of item v in $\mathbf{R}$; $\mathcal{R}_v$ denotes the positive interactions of v; c_0 determines the overall weight of missing data, and x controls the significance level of popular items over unpopular ones.

4 EFFICIENT NEURAL MATRIX FACTORIZATION

This section elaborates our proposed methods. Specifically, in Section 4.1, we first present a general framework of neural matrix factorization with the whole-data-based learning strategy. Then, we introduce three efficient learning methods under the framework, which are user-based, item-based, and alternating-based. In Section 4.5, we discuss the computational complexity and the application scenarios of our efficient optimization approaches. Last, in Section 4.6, we describe the training details of the methods.

4.1 General Framework

Figure 1 illustrates our proposed framework of Efficient Neural Matrix Factorization (ENMF) without sampling. The overall neural network architecture follows the design of Neural Collaborative Filtering (NCF) [23] with two major differences. First, in the input layer, distinct from NCF that input a user-item pair (u, v), we use a user and all his/her item interactions (user-based) or an item with all its user interactions (item-based) as inputs. This setting allows the neural network to learn from the whole training data. Second, instead of applying a sampling-based strategy to optimize the interaction between user and item, we adopt our proposed efficient optimization methods to learn the model from the whole training data without sampling.

In the designed framework, users and items are first converted to dense vector representations through embeddings. Then for each user-item instance (u, v), a mapping function is applied as

$$\phi_1(\mathbf{p}_u, \mathbf{q}_v) = \mathbf{p}_u \odot \mathbf{q}_v, \tag{7}$$

where $\mathbf{p}_u \in \mathbb{R}^d$ and $\mathbf{q}_v \in \mathbb{R}^d$ are latent vectors of user u and item v, d is the number of latent factors, and $\odot$ denotes the element-wise product of vectors. We then project the vector to the prediction layer:

$$\begin{aligned} \hat{R}_{uv} &= \mathbf{h}^T(\mathbf{p}_u \odot \mathbf{q}_v) \\ &= \sum_{i=1}^{d} h_i p_{u,i} q_{v,i}, \end{aligned} \tag{8}$$

where h is the prediction layer.

The general framework is simple, but with the optimization of our proposed efficient whole-data-based learning approaches, it significantly outperforms existing complex and state-of-the-art recommendation methods. Moreover, ENMF also shows significant advantages in training efficiency, which makes it more applicable to real-world large-scale systems. The details are shown in Section 5.

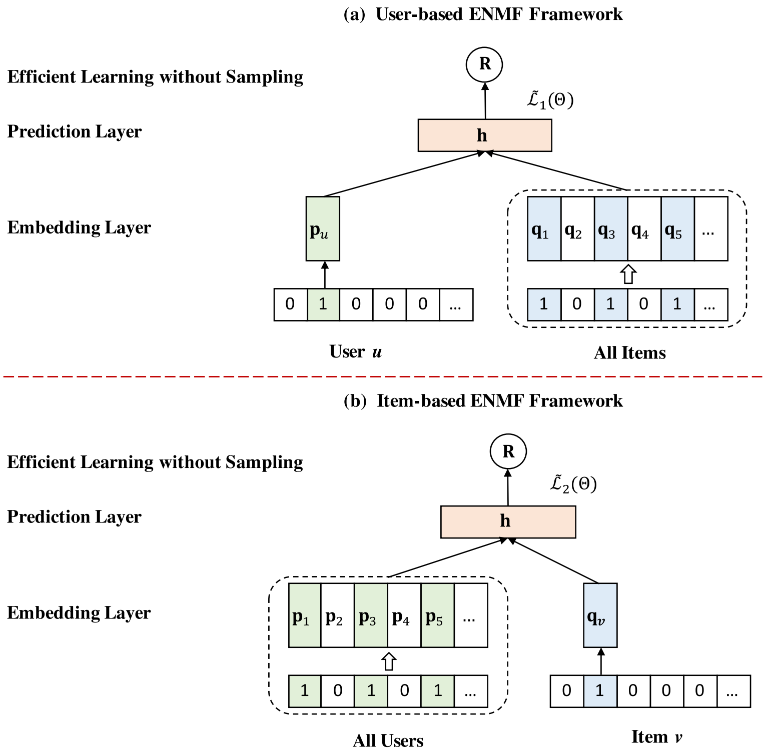

(a) User-based ENMF Framework
Efficient Learning without Sampling
Prediction Layer
Embedding Layer
R
$\tilde{\mathcal{L}}_1(\Theta)$
h
$\mathbf{p}_u$
0 1 0 0 0 ...
User u
$\mathbf{q}_1$ $\mathbf{q}_2$ $\mathbf{q}_3$ $\mathbf{q}_4$ $\mathbf{q}_5$ ...
1 0 1 0 1 ...
All Items
(b) Item-based ENMF Framework
Efficient Learning without Sampling
Prediction Layer
Embedding Layer
R
$\tilde{\mathcal{L}}_2(\Theta)$
h
$\mathbf{p}_1$ $\mathbf{p}_2$ $\mathbf{p}_3$ $\mathbf{p}_4$ $\mathbf{p}_5$ ...
1 0 1 0 1 ...
All Users
$\mathbf{q}_v$
0 1 0 0 0 ...
Item v

Fig. 1. Overview of our proposed Efficient Neural Matrix Factorization (ENMF) framework, which contains two forms: (a) User-based and (b) Item-based. The major difference between the two forms lies in whether the training batch is generated based on users or items.

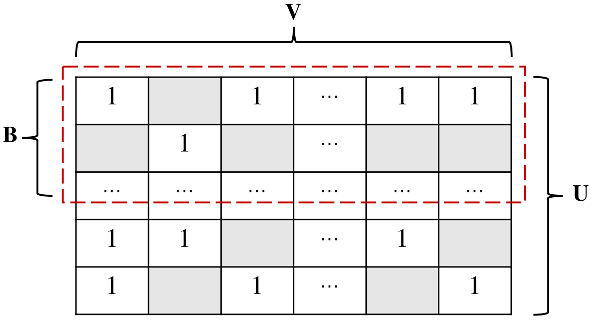
4.2 User-based Efficient Learning

Our designed efficient whole-data-based optimization strategy has three forms, which makes it more flexible in different application scenarios. We first introduce the user-based method in this section.

For implicit data, the observed interactions are rather limited, and non-observed examples are of a much larger scale. To learn model parameters, Hu et al. [26] introduced a weighted regression loss, which associates a confidence to each prediction in the implicit data matrix (Equation (3)). To make the loss suitable for learning neural models, we first adjust it to a mini-batch form, where the training batches are generated based on users. Figure 2 shows an example of the input. Following this setting, we have the loss for a batch of users as follows:

$$\begin{aligned}\mathcal{L}_1(\Theta) &= \sum_{u\in\mathbf{B}}\sum_{v\in\mathbf{V}} c_{uv}(R_{uv}-\hat{R}_{uv})^2\\ &= \sum_{u\in\mathbf{B}}\sum_{v\in\mathbf{V}} c_{uv}(R_{uv}^2-2R_{uv}\hat{R}_{uv}+\hat{R}_{uv}^2),\end{aligned} \tag{9}$$

where $\mathbf{B}$ denotes a batch of users, $\mathbf{V}$ denotes all the items in the data set, and c_{uv} denotes the weight of entry R_{uv}. As can be seen, the time complexity of computing this loss is $O(|\mathbf{B}||\mathbf{V}|d)$, which means the straightforward way to calculate gradients is generally unaffordable in practice.



1		1	···	1	1
	1		···		
···	···	···	···	···	···
1	1		···	1	
1		1	···		1

Fig. 2. The input batch of our user-based optimization method, which is generated based on users. The grey cells denote non-observed (u, v) examples, which are assigned a zero value with lower weights.

In implicit data, since $R_{uv} \in \{0, 1\}$ indicates whether u has purchased or clicked on item v, it can be replaced by a constant to simplify the equation:

$$\mathcal{L}_1(\Theta) = const - 2\sum_{u\in\mathbf{B}}\sum_{v\in\mathbf{V}^+} c^+_{uv}\hat{R}_{uv} + \sum_{u\in\mathbf{B}}\sum_{v\in\mathbf{V}} c_{uv}\hat{R}^2_{uv}, \tag{10}$$

where *const* denotes a Θ-invariant constant value that can be eliminated. Then, the loss of missing data can be expressed by the residual between the loss of all data and that of positive data. This is a key design of our method, it serves as the prerequisite for the efficient computation. The detailed derivation is as follows:

$$\begin{aligned}\mathcal{L}_1(\Theta) &= const - 2\sum_{u\in\mathbf{B}}\sum_{v\in\mathbf{V}^+} c^+_{uv}\hat{R}_{uv} + \sum_{u\in\mathbf{B}}\sum_{v\in\mathbf{V}^+} c^+_{uv}\hat{R}^2_{uv} + \sum_{u\in\mathbf{B}}\sum_{v\in\mathbf{V}^-} c^-_{uv}\hat{R}^2_{uv}\\ &= const - 2\sum_{u\in\mathbf{B}}\sum_{v\in\mathbf{V}^+} c^+_{uv}\hat{R}_{uv} + \sum_{u\in\mathbf{B}}\sum_{v\in\mathbf{V}^+} c^+_{uv}\hat{R}^2_{uv} + \sum_{u\in\mathbf{B}}\sum_{v\in\mathbf{V}} c^-_{uv}\hat{R}^2_{uv} - \sum_{u\in\mathbf{B}}\sum_{v\in\mathbf{V}^+} c^-_{uv}\hat{R}^2_{uv}\\ &= const + \overbrace{\sum_{u\in\mathbf{B}}\sum_{v\in\mathbf{V}^+}\left((c^+_{uv} - c^-_{uv})\hat{R}^2_{uv} - 2c^+_{uv}\hat{R}_{uv}\right)}^{\mathcal{L}^P_1(\Theta)} + \overbrace{\sum_{u\in\mathbf{B}}\sum_{v\in\mathbf{V}} c^-_{uv}\hat{R}^2_{uv}}^{\mathcal{L}^A_1(\Theta)},\end{aligned} \tag{11}$$

where $\mathcal{L}^P_1(\Theta)$ denotes the loss of positive data and $\mathcal{L}^A_1(\Theta)$ denotes the loss of all data. Thus, $\mathcal{L}_1(\Theta)$ can be seen as a combination of the loss of positive data and the loss of all data. And the loss of missing data has been eliminated. As can be seen, the first term focuses on the observed data only and leads to a low complexity in optimization. The major computational bottleneck lies in $\mathcal{L}^A_1(\Theta)$ now. In the following, we show how to reduce the huge volume of computation by a simple mathematical decouple.

Recall the prediction of $\hat{R}_{uv}$ (Equation (8)), based on a decouple manipulation for the inner product operation, the summation operator and elements in $\mathbf{p}_u$ and $\mathbf{q}_v$ can be rearranged:

$$\begin{aligned}\hat{R}^2_{uv} &= \sum_{i=1}^{d} h_i p_{u,i} q_{v,i} \sum_{j=1}^{d} h_j p_{u,j} q_{v,j}\\ &= \sum_{i=1}^{d}\sum_{j=1}^{d}\left(h_i h_j\right)\left(p_{u,i} p_{u,j}\right)\left(q_{v,i} q_{v,j}\right).\end{aligned} \tag{12}$$

By substituting Equation (12) in $\mathcal{L}^A_1(\Theta)$, there emerges a nice structure: If setting c^-_{uv} to a uniform [26] (Equation (4)) or an item-dependent parameter c^-_v [25, 30] (Equation (5)) as previous

ALGORITHM 1: User-based efficient learning without sampling

```
Require: Training data $\{\mathbf{R}, \mathbf{U}, \mathbf{V}\}$; weights of entries $c$; learning rate $\eta$; embedding size $d$
Ensure: Neural parameters $\Theta$
Randomly initialize neural parameters $\Theta$
while Stopping criteria is not met do
    Randomly draw a training batch based on user $\{\mathbf{R_B}, \mathbf{B}, \mathbf{V}\}$
    Compute the loss $\tilde{\mathcal{L}}_1(\Theta)$ (Equation (14))
    Update model parameters
end while
return $\Theta$
```

work, then the interaction between $p_{u,i}$ and $q_{v,i}$ can be properly separated. Thus, the optimization of $\sum_{v\in\mathbf{V}} c_v^- q_{v,i} q_{v,j}$ and $\sum_{u\in\mathbf{B}} p_{u,i} p_{u,j}$ are independent of each other, which means we could achieve a significant speed-up by precomputing the two terms:

$$\mathcal{L}_1^A(\Theta) = \sum_{i=1}^{d}\sum_{j=1}^{d}\left(\left(h_i h_j\right)\left(\sum_{u\in\mathbf{B}} p_{u,i} p_{u,j}\right)\left(\sum_{v\in\mathbf{V}} c_v^- q_{v,i} q_{v,j}\right)\right). \tag{13}$$

The rearrangement of nested sums in Equation (13) is the key transformation that allows the fast optimization. The computing complexity of $\mathcal{L}_1^A(\Theta)$ has been reduced from $O(|\mathbf{B}||\mathbf{V}|d)$ to $O((|\mathbf{B}| + |\mathbf{V}|)d^2)$.

By substituting Equation (13) in Equation (11) and removing the *const* part, we get the final user-based efficient loss as follows:

$$\tilde{\mathcal{L}}_1(\Theta) = \sum_{u\in\mathbf{B}}\sum_{v\in\mathbf{V}^+}\left((c_v^+ - c_v^-)\hat{R}_{uv}^2 - 2c_v^+\hat{R}_{uv}\right) + \sum_{i=1}^{d}\sum_{j=1}^{d}\left(\left(h_i h_j\right)\left(\sum_{u\in\mathbf{B}} p_{u,i} p_{u,j}\right)\left(\sum_{v\in\mathbf{V}} c_v^- q_{v,i} q_{v,j}\right)\right), \tag{14}$$

where c_{uv} is simplified to c_v as discussed before. We give the training process of user-based method in Algorithm 1. It is worth noting that our efficient optimization method is strictly equal to the original loss function, as no approximation is introduced during the derivation process.

4.3 Item-based Efficient Learning

In this section, we introduce our item-based efficient optimization method. In Figure 3, we show an example of the input, which contains a batch of items and all the users in the data set. Similarly, for a input batch, the original regression loss is

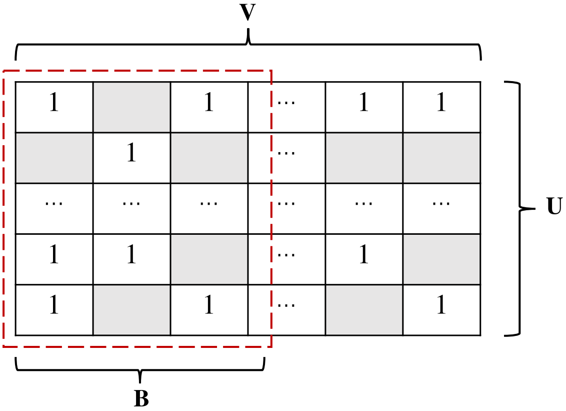
$$\mathcal{L}_2(\Theta) = \sum_{u\in\mathbf{U}}\sum_{v\in\mathbf{B}} c_{uv}(R_{uv} - \hat{R}_{uv})^2, \tag{15}$$

where $\mathbf{B}$ denotes a batch of items, $\mathbf{U}$ denotes the user set, and c_{uv} denotes the weight of entry R_{uv}.

The main difference between user-based method and item-based method lies in whether the batches are generated based on user or item. The derivation processes of the two forms are similar. To avoid repetition, we leave out the detailed derivation of the item-based method and only show the final result as follows:

$$\tilde{\mathcal{L}}_2(\Theta) = \sum_{u\in\mathbf{U}^+}\sum_{v\in\mathbf{B}}\left((c_v^+ - c_v^-)\hat{R}_{uv}^2 - 2c_v^+\hat{R}_{uv}\right) + \sum_{i=1}^{d}\sum_{j=1}^{d}\left(\left(h_i h_j\right)\left(\sum_{u\in\mathbf{U}} p_{u,i} p_{u,j}\right)\left(\sum_{v\in\mathbf{B}} c_v^- q_{v,i} q_{v,j}\right)\right). \tag{16}$$

The training process of item-based method is shown in Algorithm 2.



1	▨	1	···	1	1
▨	1	▨	···	▨	▨
···	···	···	···	···	···
1	1	▨	···	1	▨
1	▨	1	···	▨	1

Fig. 3. The input batch of our item-based optimization method, which is generated based on items. The grey cells denote non-observed (u, v) examples, which are assigned a zero value with lower weights.

ALGORITHM 2: Item-based efficient learning without sampling

```
Require: Training data $\{\mathbf{R}, \mathbf{U}, \mathbf{V}\}$; weights of entries $c$; learning rate $\eta$; embedding size $d$
Ensure: Neural parameters $\Theta$
Randomly initialize neural parameters $\Theta$
while Stopping criteria is not met do
    Randomly draw a training batch based on item $\{\mathbf{R_B}, \mathbf{B}, \mathbf{U}\}$
    Compute the loss $\tilde{\mathcal{L}}_2(\Theta)$ (Equation (16))
    Update model parameters
end while
return $\Theta$
```

4.4 Alternating-based Efficient Learning

So far, we have derived two forms of efficient whole-data-based optimization method—user-based and item-based. Although each individual method can achieve good results in practice, there is still a potential problem that they may encounter: the imbalance of learning speed between users and items. Take the user-based method as an example, each user is associated with all the items as inputs, which will result in more gradient steps on items than users. As a result, items may be over-trained locally at the expense of users, which would then be under-trained. Although some tricks like adaptive learning rates like Adagrad [13] have been proposed to alleviate this problem, we still want to address it fundamentally.

Since we have user-based method and item-based method, a natural question then arises: How can we fuse the two forms, so that they can mutually reinforce each other to further address the above problem and better learn the complex user-item interactions?

In traditional collaborative filtering methods, Alternating-Least-Square (ALS) is widely used to solve the whole-based MF [24, 26, 30]. It alternates between re-computing user-factors and item-factors, and each step is guaranteed to lower the value of the loss function. Motivated by ALS, we propose an alternating-based efficient learning method that conducts a two-step training procedure. During one learning epoch, the first step uses user-based loss and the second step uses item-based loss. Each step is guaranteed to lower the value of the loss function. The overall alternating-based learning method is summarised in Algprithm 3. During the training stage, the

ALGORITHM 3: Alternating-based efficient learning without sampling

```
Require: Training data {R, U, V}; weights of entries $c$; learning rate $\eta$; embedding size $d$
Ensure: Neural parameters $\Theta$
Randomly initialize neural parameters $\Theta$
while Stopping criteria is not met do
    for user-based step do
        Randomly draw a training batch based on user $\{\mathbf{R_B}, \mathbf{B}, \mathbf{V}\}$
        Compute the loss $\tilde{\mathcal{L}}_1(\Theta)$ (Equation (14))
        Update model parameters
    end for
    for item-based step do
        Randomly draw a training batch based on item $\{\mathbf{R_B}, \mathbf{B}, \mathbf{U}\}$
        Compute the loss $\tilde{\mathcal{L}}_2(\Theta)$ (Equation (16))
        Update model parameters
    end for
end while
return $\Theta$
```

model is trained using user-based loss and item-based loss alternatively via Equation (14) and Equation (16).

4.5 Discussion on Complexity

Now that the basic description of our technique is completed, we would like to further discuss the complexity of our methods in this section.

For user-based efficient loss (Equation (14)), updating a batch of users takes $O((|\mathbf{B}| + |\mathbf{V}|)d^2 + |\mathcal{R}_\mathbf{B}|d)$ time, where $\mathcal{R}_\mathbf{B}$ denotes positive user-item interactions of this batch. Then updating an epoch takes $O((|\mathbf{U}| + \frac{|\mathbf{U}||\mathbf{V}|}{|\mathbf{B}|})d^2 + |\mathcal{R}|d)$. Similarly, for item-based efficient loss (Equation (16)), one epoch takes $O((|\mathbf{V}| + \frac{|\mathbf{U}||\mathbf{V}|}{|\mathbf{B}|})d^2 + |\mathcal{R}|d)$ time. The complexity of alternating-based method is the sum of user-based and item-based. However, it generally needs fewer epochs to achieve optimal performance due to the balanced training of users and items, which reduces the overall time cost in practice. For the original regression loss, one epoch takes $O(|\mathbf{U}||\mathbf{V}|d)$. Since $|\mathcal{R}| \ll |\mathbf{U}||\mathbf{V}|$ and $d \ll |\mathbf{B}|$ in practice, the computational complexity of training a model without sampling is reduced by several magnitudes. This makes it possible to apply whole-data-based optimization strategy for neural models. Moreover, since no approximation is introduced during the derivation process, the optimization results are exactly the same with the original whole-data-based regression loss.

4.6 Training

Modern computing units such as CPU and GPU usually provide speedups for matrix-wise float operations. Thus, our mini-batch-based optimization methods can be naturally implemented in modern machine learning tools like Tensorflow and PyTorch. The model parameters can be calculated with standard back-propagation, which is omitted here, since the learner is not the main concern of this article. To optimize the objective function, we adopt mini-batch Adagrad [13] as the optimizer. Its main advantage is that the learning rate can be self-adapted during the training phase, which eases the pain of choosing a proper learning rate and alleviates the imbalanced training of users and items.

Overfitting is a perpetual problem in optimizing a machine learning model. Many works have mentioned that deep-learning models are even more likely to suffer from overfitting [18, 19]. To alleviate this issue, we consider dropout [47]—a widely used method in deep-learning models, in our work. The idea of dropout is randomly drop some neurons (along with their connections) during the training process [47]. When updating parameters, only part of them will be updated. Trough this process, it can prevent complex co-adaptations of neurons on training data. Moreover, as dropout is disabled during testing and the whole network is used for prediction, dropout has another side effect of performing model averaging with smaller neural networks, which may potentially improve the performance [19]. Specifically, in our efficient neural matrix factorization framework, we randomly drop ρ percent of latent factors after the element-wise production in Equation (7) to improve the model's generalization ability, where ρ is termed as the dropout ratio.

5 EXPERIMENTS

In this section, we perform experiments to verify the correctness, efficiency, and effectiveness of our efficient learning methods and ENMF framework. All experiments are conducted on three real-world datasets, which are commonly used in recommendation systems. We aim to anwser the following research questions:

- How are the effectiveness and efficiency of our proposed efficient learning methods?
- How does the proposed ENMF framework performs compared with state-of-the-art item recommendation methods?
- How do the key hyper-parameter settings impose influence on the performance of our methods?

In what follows, we first introduce the experimental settings, followed by answering the above three research questions.

5.1 Experimental Settings

5.1.1 Datasets. We experimented with three public accessible datasets: ***Ciao***,[2] ***Epinion***,[3] and ***Movielens***.[4] We briefly introduce the three datasets:

- ***Ciao***: This dataset contains users' ratings to the items they have purchased. Since we focus on the implicit feedback, the detailed rating is transformed into a value of 0 or 1, indicating whether a user has rated an item.
- ***Epinions***: *Epinions* is a widely used dataset for recommendation. It is collected from a who-trust-whom directed online social network that provides product rating and review service. The corresponding rating is also assigned to a value of 1 (as implicit feedback) in our experiments.
- ***Movielens***: *MovieLens* is a dataset of movie ratings that have been leveraged extensively to investigate the performance of CF algorithms. In our experiments, we choose the version including one million ratings where there are 20 ratings per user at least. It is the largest dataset in our experiments.

All the datasets were preprocessed to make sure that all items have at least five interactions to make them easier to evaluate CF algorithms. The statistical details of these datasets are summarized in Table 2.

5.1.2 Baselines. To evaluate the performance of our proposed methods, we compare with the following approaches:

[2] http://www.jiliang.xyz/trust.html.

[3] https://alchemy.cs.washington.edu/data/epinions/.

[4] https://grouplens.org/datasets/movielens/1m/.

Table 2. Statistical Details of the Evaluation Datasets

Dataset	#User	#Item	#Interaction	Density
Ciao	7,267	11,211	157,995	0.19%
Epinion	20,608	23,585	454,002	0.09%
Movielens	6,940	3,706	1,000,209	4.47%

Table 3. Comparison of the Methods

Characteristics	MP	ItemKNN	BPR	WMF	ExpoMF	GMF	NCF	ConvNCF	ENMF
Top-*N* Recommendation	√	√	√	√	√	√	√	√	√
Neural Model	\	\	\	\	\	√	√	√	√
Whole-data-based	\	\	\	√	√	\	\	\	√

- **MostPopular (MP)**: It is non-personalized, since it ranks items according to their popularity, which is measured by the number of interactions.
- **ItemKNN** [44]: This is a standard item-based method that measures the similarity among items for implicit feedback.
- **Bayesian Personalized Ranking (BPR)** [41]: This method optimizes MF with the Bayesian Personalized Ranking objective function to learn from implicit feedback data. It is a often opted baseline for item recommendation.
- **Weighted MF (WMF)** [26]: This is a whole-data-based method for item recommendation. It treats all missing interactions as negative instances and weighting them uniformly.
- **Exposure MF (ExpoMF)** [30]: This is a whole-data-based method for item recommendation. It treats all missing interactions as negative instances and weighting them with the corresponding item's popularity.
- **Generalized Matrix Factorization (GMF)** [23]: It is one of the state-of-the-art neural network-based recommendation method. Note that the network structure of GMF is the same as our ENMF except for adopting negative sampling for model learning.
- **Neural Collaborative Filtering (NCF)** [23]: This is the state-of-the-art deep-learning method that uses users' historical feedback for item ranking. It combines MF with a multi-layer perceptron (MLP) model and utilizes negative sampling for model learning.
- **Convolutional Neural Collaborative Filtering (ConvNCF)** [20]: This is a recently proposed deep-learning method that supercharges NCF modeling with an outer product operation and convolutional neural network (CNN). It also utilizes negative sampling for model learning.

The comparison of our ENMF and the baseline methods are listed in Table 3.

5.1.3 Evaluation Metrics. After a model is trained, we generate the personalized ranking list for a user by ranking all items that are not interacted by the user in the training set. The leave-one-out evaluation protocol [17, 23] is employed here to study the performance of item recommendation. Specifically, we sort the user-item interactions by the timestamps for each user at first; then the last records of users were used as test data, the second last records were used as validation data, and the remaining records were used for training.

We evaluate the ranking list using Hit Ratio (HR) and Normalized Discounted Cumulative Gain (NDCG). HR is a recall-based metric, measuring whether the testing item is in the top-K list, while NDCG is position-sensitive, which assigns higher score to hits at higher positions. We define the generated recommendation list for user u as $\mathbf{Re}_u = \{re_u^1, re_u^2, \ldots, re_u^N\}$, where N is the number of recommended items, re_u^i is ranked at the ith position in $\mathbf{Re}_u$ according to the predicted score. The set of u's interacted items in the test data is defined as $\mathbf{T}_u$. These metrics are computed as follows:

$$\begin{aligned} HR@N &= \frac{1}{|\mathbf{U}|}\sum_u I\left(|\mathbf{Re}_u \cap \mathbf{T}_u|\right), \\ NDCG@N &= \frac{1}{Z}DCG@N = \frac{1}{Z}\frac{1}{|\mathbf{U}|}\sum_u\sum_{i=1}^{N}\frac{2^{I\left(\left|\{re_u^i\}\cap \mathbf{T}_u\right|\right)}-1}{log2(i+1)}, \end{aligned} \tag{17}$$

where $I(x)$ is an indicator function whose value is 1 when $x > 0$, and 0 otherwise, $\mathbf{U}$ is the user set, and Z is a normalization constant, which is the maximum possible value of $DCG@N$.

For both metrics, larger values indicate better performance. Note that for a user, our evaluation protocol ranks all unobserved items in the training set. Through this way, the obtained results are more persuasive than ranking a random subset of negative times only [17]. Though this all-ranking protocol is very time-consuming for neural models NCF and ConvNCF, our ENMF can still rank all unobserved items quickly by leveraging matrix operation. For each method, we randomly initialize the model and run it five times. After that, we report the average results.

5.1.4 Parameter Settings. Our ENMF approaches are implemented with TensorFlow,[5] a well-known open-source software library for deep learning. All hyperparameters are tuned according to the validation set. The parameters for all baseline methods were initialized as in the corresponding papers, and were then carefully tuned to achieve optimal performances. The learning rate for all models were tuned amongst [0.005, 0.01, 0.02, 0.05]. To prevent overfitting, we tuned the dropout ratio in [0.1, 0.3, 0.5, 0.7, 0.9, 1]. The batch size was tested in [128, 256, 512, 1024], the dimension of the latent factor number d were tested in [8, 16, 32, 64]. In addition, the number of negative samples is tuned amongst [1, 2, 3, 4, 5, 6] for sampling-based methods. Note that we uniformly set the weight of missing data as c_0, as the effectiveness of popularity-biased weighting strategy is beyond the scope of this article. c_0 was tuned amongst [0.005, 0.01, 0.05, 0.1, 0.2, 0.5, 1]. After the tuning process, the batch size was set to 512, the size of the latent factor dimension d was set to 64, and the learning rate was set to 0.05. The dropout ratio ρ was set to 0.3 for Ciao, 0.5 for Epinion, and 0.7 for Movielens. c_0 was set to 0.05 for Ciao and Epinion, and 0.5 for Movielens. The effects of these hyperparameters are further explored in Section 5.4.

5.2 Performance Comparison

We first made a comparison between our proposed models and other recommendation approaches. We investigated the top-N performance with N setting to [50, 100, 200]. Note that for a user, all unobserved items in the training set were ranked to provide the recommendation list. In this case, small values of N will make the results have a large variance and unstable [17]. As such, we report results of a relatively large N. For the purpose of a fair comparison, the embedding size is set to 64 for all embedding-based approaches (BPR, WMF, ExpoMF, GMF, NCF, ConvNCF, and ENMF). In Section 5.4, we will alter the embedding size of these embedding-based approaches to observe embedding size performance trends. The results of the comparison of different methods on three datasets are shown in Table 4. Note that we show the results of our ENMF with three different learning methods: user-based (ENMF-U), item-based (ENMF-I), and alternating-based (ENMF-A). From the results, the following observations can be made:

(1) The popularity-based method MP is particularly ineffective here, indicating the necessity of modeling users' personalized preferences, rather than just recommending popular items

[5] https://www.tensorflow.org/.

Table 4. Performance of Different Models on Three Datasets

Ciao	**HR@50**	**HR@100**	**HR@200**	**NDCG@50**	**NDCG@100**	**NDCG@200**	**RI**
MP	0.1047	0.1384	0.1776	0.0396	0.0452	0.0506	+67.96%
ItemKNN	0.1453	0.1884	0.2468	0.0497	0.0581	0.0668	+26.14%
BPR	0.1531	0.1930	0.2558	0.0517	0.0598	0.0685	+21.91%
WMF	0.1587	0.2011	0.2608	0.0562	0.0631	0.0714	+16.40%
ExpoMF	0.1602	0.1994	0.2613	0.0569	0.0626	0.0709	+16.41%
GMF	0.1668	0.2103	0.2674	0.0633	0.0687	0.0752	+9.36%
NCF	0.1651	0.2108	0.2712	0.0629	0.0695	0.0764	+8.84%
ConvNCF	0.1682	0.2237	0.2741	0.0641	0.0714	0.0787	+5.90%
ENMF-U	**0.1750****	**0.2296****	**0.2945****	**0.0651****	**0.0741****	**0.0830****	–
ENMF-I	**0.1749****	**0.2311****	**0.2946****	**0.0643***	**0.0734****	**0.0823****	–
ENMF-A	**0.1757****	**0.2331****	**0.3015****	**0.0662****	**0.0753****	**0.0850****	–
Epinion	**HR@50**	**HR@100**	**HR@200**	**NDCG@50**	**NDCG@100**	**NDCG@200**	**RI**
MP	0.0661	0.1068	0.1659	0.0234	0.0299	0.0382	+153.96%
ItemKNN	0.1312	0.2082	0.2929	0.0455	0.0563	0.0682	+34.41%
BPR	0.1708	0.2338	0.3007	0.0548	0.0646	0.0747	+17.04%
WMF	0.1765	0.2384	0.3158	0.0605	0.0685	0.0789	+11.07%
ExpoMF	0.1784	0.2368	0.3064	0.0602	0.0691	0.0781	+11.70%
GMF	0.1811	0.2513	0.3388	0.0613	0.0739	0.0845	+5.52%
NCF	0.1816	0.2534	0.3442	0.0621	0.0750	0.0869	+4.08%
ConvNCF	0.1833	0.2510	0.3418	0.0617	0.0742	0.0851	+4.87%
ENMF-U	**0.1893****	**0.2647****	**0.3523****	**0.0639****	**0.0761****	**0.0883****	–
ENMF-I	**0.1888****	**0.2667****	**0.3534****	**0.0634****	**0.0759****	**0.0884****	–
ENMF-A	**0.1911****	**0.2688****	**0.3546****	**0.0648****	**0.0773****	**0.0893****	–
Movielens	**HR@50**	**HR@100**	**HR@200**	**NDCG@50**	**NDCG@100**	**NDCG@200**	**RI**
MP	0.1842	0.2099	0.3382	0.0441	0.0481	0.0659	+109.01%
ItemKNN	0.2101	0.2889	0.3918	0.0598	0.0724	0.0867	+59.18%
BPR	0.2637	0.4048	0.5710	0.0757	0.0986	0.1217	+17.59%
WMF	0.2924	0.4378	0.6040	0.0909	0.1073	0.1324	+6.47%
ExpoMF	0.2904	0.4368	0.5927	0.0865	0.1100	0.1346	+7.11%
GMF	0.2847	0.4226	0.5847	0.0821	0.1086	0.1289	+10.30%
NCF	0.2902	0.4316	0.6023	0.0837	0.1097	0.1324	+8.02%
ConvNCF	0.2943	0.4403	0.6017	0.0872	0.1112	0.1333	+6.30%
ENMF-U	**0.3117****	**0.4574****	**0.6092****	**0.0962****	**0.1198****	**0.1410****	–
ENMF-I	**0.3105****	**0.4576****	**0.6107****	**0.0956****	**0.1194****	**0.1398****	–
ENMF-A	**0.3124****	**0.4581****	**0.6139****	**0.0968****	**0.1202****	**0.1419****	–

* and ** denote the statistical significance for $p < 0.05$ and $p < 0.01$, respectively, compared to the best baseline. "RI" indicate the average relative improvements of our ENMF-A over the corresponding baseline.

to users. The learning-based methods also achieves better performance than the heuristic-based method ItemKNN.

(2) We can see that methods using whole-data-based learning strategies generally perform better than sampling-based methods. For example, in Table 4, the performances of WMF and ExpoMF are better than BPR; and our ENMF outperforms BPR, GMF, NCF and ConvNCF. This is consistent with previous work [4, 57, 60], which indicates that regardless of what sampler is utilized or how many updates are taken, sampling is still a biased approach.

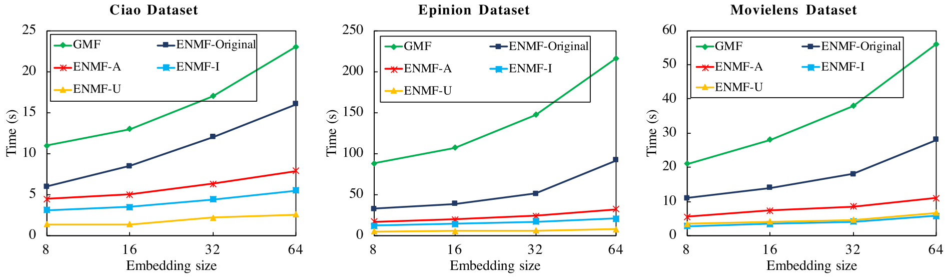


Fig. 4. Comparison on the per iteration training time of GMF, ENMF-Original, ENMF-U, ENMF-I, and ENMF-A with different embedding size d.

(3) The results show that neural methods generally performs better than traditional collaborative filtering methods, which verifies the advantages of neural networks over traditional models in representation learning. However, in our experiments, we find that the neural methods NCF and ConvNCF needs to be carefully tuned to avoid model collapse, while our ENMF is more stable and the parameters can be tuned very easily to obtain the optimal performance.

(4) Our proposed ENMF achieves the best performance (the highest HR and NDCG scores) on the three datasets, significantly outperforming all the state-of-the-art baseline methods. Furthermore, we conduct one-sample t-tests to justify that all of enhancements are statistically significant with p-Value < 0.01. Specifically, compared to ConvNCF—a recently proposed and very expressive deep-learning model, our ENMF-A exhibits average improvements of 5.90%, 4.87%, and 6.30% on the three datasets. This is very remarkable, since ENMF uses the shallow NMF framework that has much fewer parameters. Moreover, as compared to GMF, which has the same network structure but using negative sampling for model learning, ENMF-A outperforms it by 9.36%, 5.52%, and 10.30% on the three datasets. This further verifies the effectiveness of our methods and also implies the potential of improving conventional shallow methods with a better learning algorithm.

(5) There is no absolute superior between our user-based learning method and item-based method, since the results show that ENMF-U performs roughly the same as ENMF-I on the three datasets. The reason we think is that that the two methods are both under the whole-data-based learning framework and thus the results have no obvious differences. Moreover, using alternating-based learning methods (ENMF-A) generally achieves better performance, since it alleviates the imbalanced training issue caused by the use of user-based or item-based methods only.

5.3 Efficiency Analyses

In this section, we conducted experiments to explore the training efficiencies of our methods and three state-of-the-art neural recommendation methods: GMF, NCF, and ConvNCF. All experiments in this section are run on the same machine (Intel Xeon 8-Core CPU of 2.4 GHz and single NVIDIA GeForce GTX TITAN X GPU) for fair comparison on the efficiency.

We first investigated the actual speedup brought by our design of the efficient learning methods. The results of ENMF-Original are also added to show the efficiency of our proposed optimization methods, where ENMF-Original represents the method using the original regression loss (Equation (9)). Figure 4 shows the training time of GMF, ENMF-Original, ENMF-U, ENMF-I and ENMF-A with different embedding size d. In the figure, the x-axis denotes the setting of d, the y-axis

Table 5. Comparisons of Runtime (second/minute/hour [s/m/h])

Model	*Ciao*			*Epinion*			*Movielens*		
	S	I	T	S	I	T	S	I	T
GMF	23s	300	115m	216s	500	30h	56s	500	7h
NCF	34s	300	170m	305s	500	42h	91s	500	12h
ConvNCF	88s	300	440m	510s	500	70h	246s	500	34h
ENMF-Original	16s	300	80m	65s	200	216m	28s	300	140m
ENMF-U	2.6s	300	13m	8s	200	27m	6.7s	300	34m
ENMF-I	5.5s	300	28m	21s	200	70m	5.8s	300	29m
ENMF-A	8s	150	20m	32s	100	53m	11s	50	9m

"S," "I," and "T" represents the training time for a single iteration, the number of iterations to converge, and the total training time, respectively.

denotes the training time per iteration. Note that these comparison models all have the same network structure, but different learning methods.

From the figure, we can obviously observe that the training time costs of ENMF-U, ENMF-I and ENMF-A are much faster than GMF and ENMF-Original with different embedding size d. For example, on Epinion dataset, GMF and ENMF-Original require 216 and 92 seconds to train the model with d=64, respectively, while our methods ENMF-U, ENMF-I, and ENMF-A only need 8, 21, and 32 seconds, respectively. Since the five methods have the same neural network structure, we can attribute the acceleration to our designed efficient whole-data-based learning methods, which are more efficiently compared to the original regression loss and negative sampling strategy.

Furthermore, we compared the overall training time of our methods and three state-of-the-art neural recommendation methods: GMF, NCF, and ConvNCF. The embedding size is set to 64 for all the methods and the training time results are shown in Table 5. We have the following key observations:

(1) Our proposed methods are several magnitudes faster than the baseline models GMF, NCF, and ConvNCF. For example, on Epinion dataset, the baselines takes over 30 hours to train a model, while our methods only need 27, 70, and 53 minutes to achieve the optimal performance, respectively. This acceleration is over 30 times. In real E-commerce scenarios, the cost of training time is also an important factor to be considered. Our proposed ENMF methods show significant advantages in training efficiency, which makes them more practical in real life.

(2) Analytically, the time complexity of ENMF-U is $O((|\mathbf{U}| + \frac{|\mathbf{U}||\mathbf{V}|}{|\mathbf{B}|})d^2 + |\mathcal{R}|d)$, while for ENMF-I it is $O((|\mathbf{V}| + \frac{|\mathbf{U}||\mathbf{V}|}{|\mathbf{B}|})d^2 + |\mathcal{R}|d)$. The difference in time complexity between the two methods is the number of users and items. Specifically, on Ciao dataset, which has 7,267 users and 11,211 items, the total training time of ENMF-U is 13 minutes, while for ENMF-I it is 28 minutes.

(3) Although our ENMF-A takes more time to train a single iteration than ENMF-U and ENMF-I, the overall training time is generally less than them as it requires less iterations to achieve the optimal performance.

We also investigated the training process of the neural models GMF, ConvNCF and our ENMF methods. Figure 5 demonstrates the state of each method at embedding size 64 on three datasets. Note that we only show the results on HR@100 and NDCG@100 metrics. For other metrics, the observations are similar. From the figure, we can obviously find the effectiveness of our proposed methods. In particular, our methods ENMF-A, ENMF-U, ENMF-I converge much faster than GMF

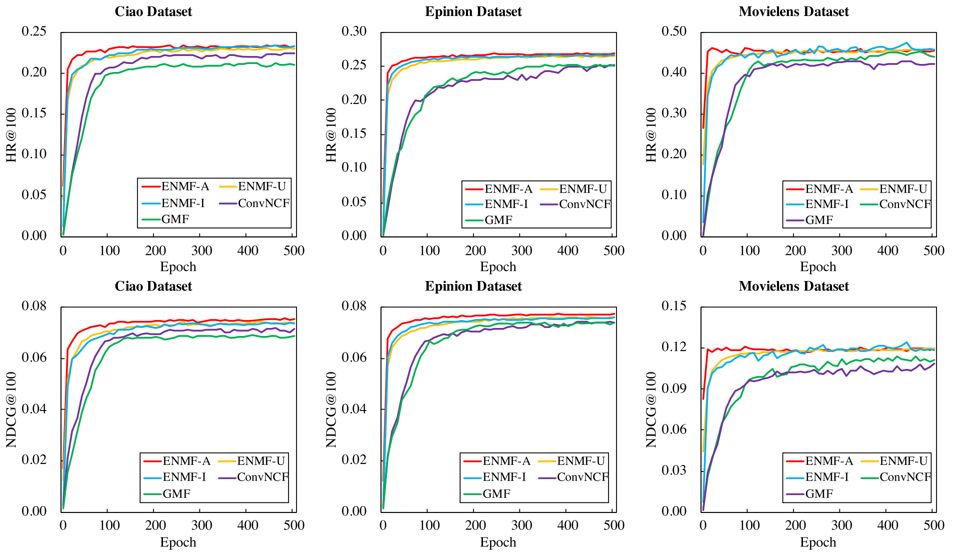


Fig. 5. Performance curves of GMF, ConvNCF, ENMF-U, ENMF-I, and ENMF-A on the three datasets.

and ConvNCF, and consistently achieves better performance. The reason is that our ENMF is optimized with newly derived whole-data-based methods, while GMF and ConvNCF are based on negative sampling, which generally requires more iterations and can be sub-optimal. Moreover, the results show that ENMF-A converges even more faster than ENMF-U and ENMF-I. Specifically, on Movielens dataset, ENMF-A only requires 50 epochs to achieve the optimal performance, while ENMF-U and ENMF-I require 300 epochs. This further verifies that by adopting alternating-based learning, ENMF-A can alleviate the under-trained issue caused by ENMF-U and ENMF-I, thus leading to a faster training process.

5.4 Hyper-parameter Analyses

In this section, we conducted experiments to investigate the impact of different values of the dropout hyper-parameter ρ and different negative weight w_0 on our ENMF methods. In addition, as embedding-based models, the embedding size is a critical hyper-parameter as well. Thus, we also compared the different embedding sizes on the performance trends. It is worth noting that our ENMF does not introduce any additional specific hyper-parameters, which makes it much easier to be tuned compared with other complex deep-learning models in practice.

5.4.1 Impact of Embedding Size. We conducted experiments to test the impact of embedding size d in this subsection. Figure 6 shows the performance of HR@100 and NDCG@100 with respect to the embedding size. For other metrics, the observations are similar. As can be seen from this figure, our ENMF methods outperform all the other models with different values of d for the two ranking metrics. Notably, ENMF with an embedding size of 32 even performs better than the best baseline ConvNCF with a larger embedding size of 64. This further verifies the positive effect of whole-data-based learning in our ENMF methods.

Moreover, as the latent dimension size increases, the performance of all models increase. This indicates that a larger dimension could capture more hidden factors of users and items, which is beneficial to Top-K recommendation due to the increased modeling capability. This observation is similar to previous work [3, 21, 23]. However, for most deep-learning methods, a larger dimension

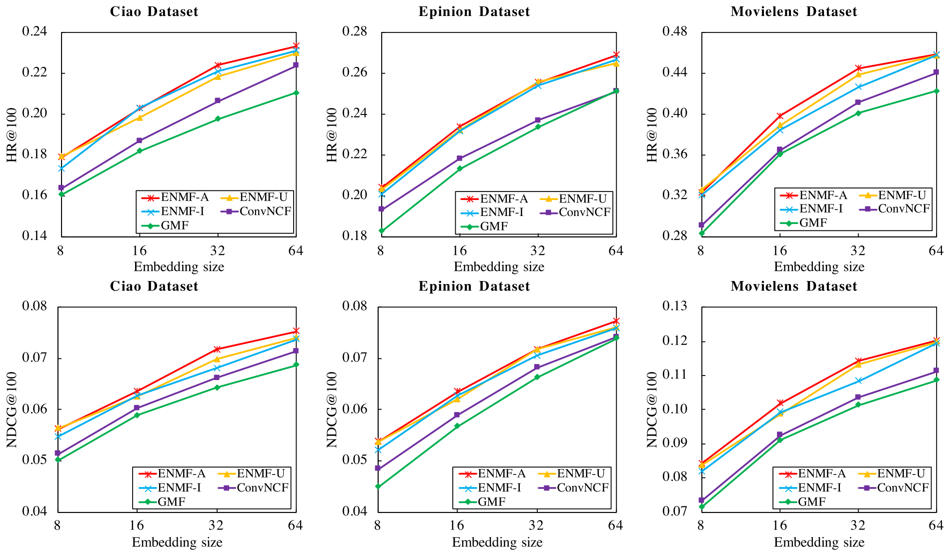


Fig. 6. Performance of GMF, ConvNCF, ENMF-U, ENMF-I, and ENMF-A w.r.t. the embedding size on the three datasets.

also requires more time for training. Thus, it is crucial to increase a model's efficiency by learning with efficient optimization methods.

The recommendation task typically involves a large space of input features (e.g., user ID, item ID, and other attributed and contextual variables). Given such a large feature space, even a shallow embedding model like Matrix Factorization [26] will have a large number of parameters, not to mention the more expressive deep neural networks [20]. This work introduces efficient whole-data-based learning for the ranking task, providing a new means to increase the training efficiency of large models and having the potential to improve a wide range of models.

5.4.2 Impact of Negative Weight. To illustrate the impact for whole-data-based methods WMF, ENMF-U, ENMF-I, and ENMF-A with regarding to the negative weight, we demonstrate the experimental results with different weights of negative instances in Figure 7. Note that in our experiments, we first uniformly set the weight of missing data as c_0. From the figure, we can make the following observations.

First, for Ciao and Epinion, the peak performance is achieved when c_0 is around 0.05, while for Movielens, the optimal c_0 is around 0.5. When c_0 becomes smaller or too larger, the performance of WMF and our ENMF both degrade. This highlights the necessity of accounting for the missing data when modeling implicit feedback for item recommendation. Second, the performances of our ENMF-U, ENMF-I and ENMF-A with respect to the negative weight on the three datasets are similar. Comparing to WMF, our ENMF methods are more robust to the weight of missing data. Specifically, on Ciao and Epinion datasets, ENMF with c_0 between 0.005 to 0.2 consistently perform better than WMF with an optimal c_0. The reasons are twofold. First, WMF optimizes with the vanilla ALS, while our ENMF methods are optimized with Adagrad, which adapts the learning rate for each parameter based on its frequency (i.e., smaller updates for frequent and larger updates for infrequent parameters). Second, WMF prevents overfitting via L_2 regularization, while we employ dropout, which can be more effective due to the model averaging effect. Third, considering the performance on each dataset, we find that the optimal weight of missing instance depends on the density of the dataset. The Movielens dataset is relatively dense in terms of user–item interactions

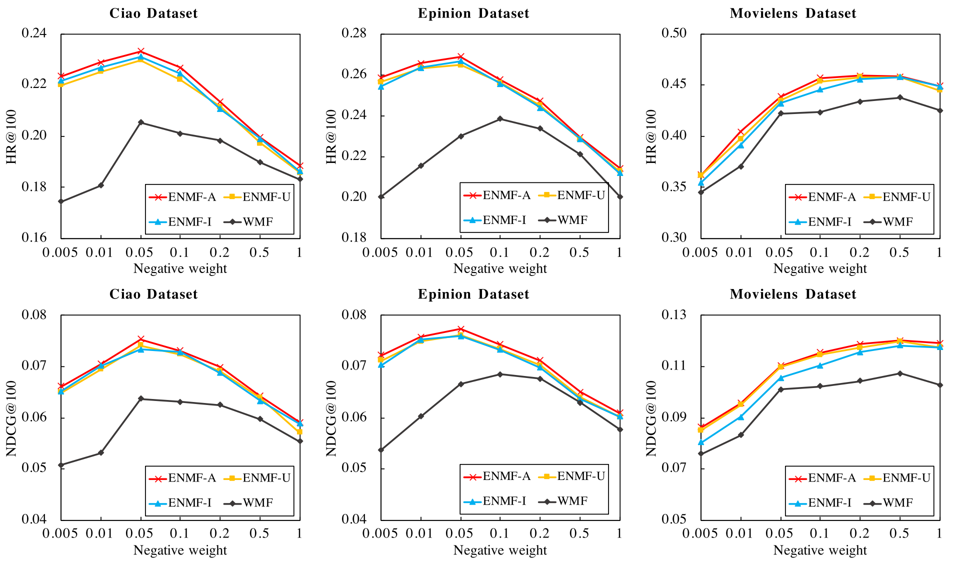


Fig. 7. Performance of WMF, ENMF-U, ENMF-I, and ENMF-A w.r.t. the negative weight on the three datasets.

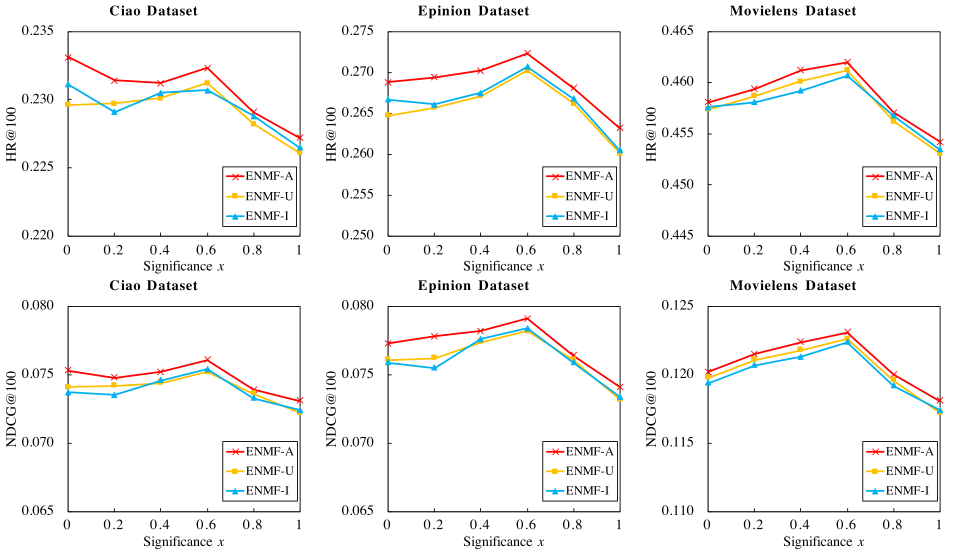


Fig. 8. Performance of ENMF-U, ENMF-I, and ENMF-A w.r.t. different significant value of item popularity on the three datasets.

(the density is 4.47%, compared with 0.19% and 0.09% for Ciao and Epinions, respectively). As shown in previous work [25, 30], popular items are more likely to be known by users, thus it is reasonable to assign a larger weight to a missing popular item as it is more probable to be a truly negative instance.

Then, we vary the significant value of item popularity x with the optimal c_0 to check the performance change. As shown in Figure 8, the optimal x is around 0.6 on the three datasets. Below 0.6, with the increase of x, the performances of our ENMF methods are gradually improved. But when

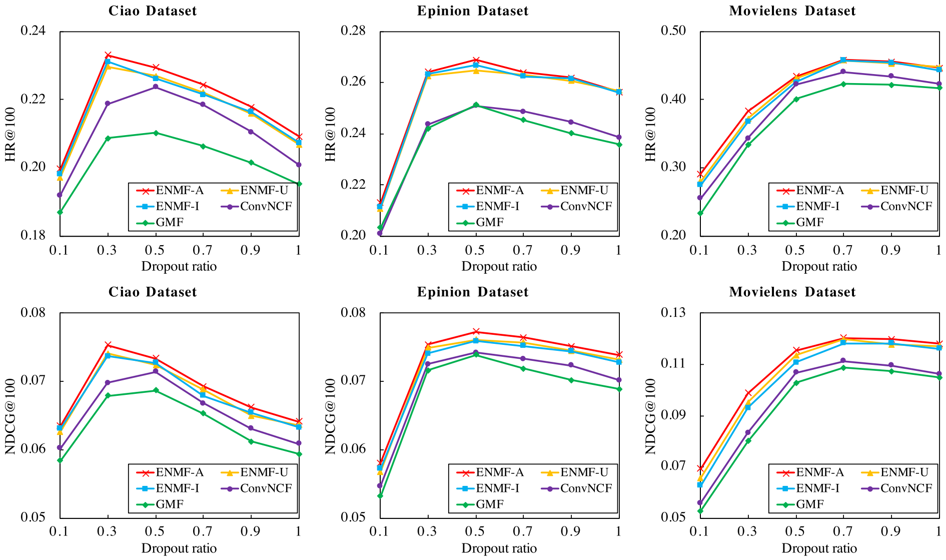


Fig. 9. Performance of GMF, ConvNCF, ENMF-U, ENMF-I, and ENMF-A w.r.t. to different dropout ratios on the three datasets.

x increases above 0.5, the performance of our methods become worse. This observation is similar to previous work [24, 25], which reveals that weighting missing data according item popularity is important for recommendation.

5.4.3 Impact of Dropout. We then studied the impact of dropout on deep-learning-based methods. Figure 9 shows the validation performances of GMF, ConvNCF, and our ENMF approaches with respect to different dropout ratios.

From the results, we can first see that by setting the dropout ratio to a proper value, all methods can be significantly improved. Specifically, for ENMF, the optimal dropout ratio on Ciao, Epinion, and MovieLens is 0.3, 0.5, and 0.7, respectively. This demonstrates the ability of dropout in preventing overfitting and as such, better generalization can be achieved.

Second, our ENMF methods consistently perform better than the state-of-the-art neural methods GMF and ConvNCF with respect to different dropout ratios. The reason we think is that the parameters in our methods are optimized using the whole data, while sample-based methods (GMF and ConvNCF) only use a fraction of sampled data and may ignore important negative example.

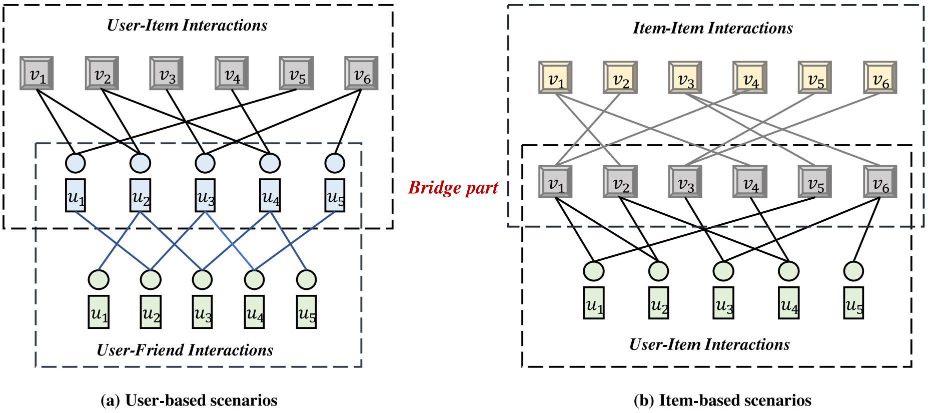
Moreover, considering the performance on each dataset, we find that the optimal dropout ratio also depends on the density and size of the dataset. For example, the results of Ciao are more sensitive to the dropout ratios than the results of Epinion and Movielens. This is intuitive, since neural methods are more likely to be overfitting on a small and sparse dataset.

6 LIMITATION AND EXTENSIONS

In this section, we discuss the limitation and some extensions of our efficient learning methods.

6.1 Limitation

As fast whole-data-based learning is a challenging problem, our current efficient learning methods are still preliminary and have a limitation. They are now limited to learn models with linear prediction layer, because the rearrange operation in Equation (13) requires the prediction of $\hat{R}_{uv}$ to be linear. We leave the extension as future work.


User-Item Interactions
v_1 v_2 v_3 v_4 v_5 v_6
u_1 u_2 u_3 u_4 u_5
u_1 u_2 u_3 u_4 u_5
User-Friend Interactions
Bridge part
Item-Item Interactions
v_1 v_2 v_3 v_4 v_5 v_6
v_1 v_2 v_3 v_4 v_5 v_6
u_1 u_2 u_3 u_4 u_5
User-Item Interactions

(a) User-based scenarios

(b) Item-based scenarios

Fig. 10. Examples of the specific scenarios for our user-based method and item-based method, respectively.

Nevertheless, our efficient learning methods still have a wide range of application scenarios. Since our methods use a generic feature representation for inputs, the user and item factors can be easily customized to utilize content features or support complex neural structures such as attention mechanisms [4, 22].

6.2 Leveraging Heterogeneous Information

Since we focus on the pure collaborative filtering setting in this work, we apply our efficient whole-data-based leaning methods to a general Neural Matrix Factorization framework. It is worth mention that our methods are not limited to the pure collaborative filtering setting. They can be customized to support a wide range of modelling of users and items, such as social-aware [4], content-based [19], neighbor-based [22], and multi-behavior-based [17].

Generally, our three efficient learning methods can be replaced by each other, the specific method can also be selected according to the actual situation. As shown in Figure 10, in social-aware scenarios where users act as the bridge of the social domain and the item domain, our user-based method will be a good choice to learn users' preference on both items and their social friends simultaneously [4], we provide a simple social-aware example as follows:

For item domain, we have

$$\tilde{\mathcal{L}_I}(\Theta) = \sum_{u\in\mathbf{B}}\sum_{v\in\mathbf{V}^+}\left((c_v^{I+} - c_v^{I-})\hat{R}_{uv}^2 - 2c_v^{I+}\hat{R}_{uv}\right) + \sum_{i=1}^{d}\sum_{j=1}^{d}\left(\left(h_i^I h_j^I\right)\left(\sum_{u\in\mathbf{B}} p_{u,i}p_{u,j}\right)\left(\sum_{v\in\mathbf{V}} c_v^{I-} q_{v,i}q_{v,j}\right)\right). \quad (18)$$

Similarly, we can derive the loss function for social domain:

$$\tilde{\mathcal{L}_S}(\Theta) = \sum_{u\in\mathbf{B}}\sum_{t\in\mathbf{U}^+}\left((c_t^{S+} - c_t^{S-})\hat{X}_{ut}^2 - 2c_t^{S+}\hat{X}_{ut}\right) + \sum_{i=1}^{d}\sum_{j=1}^{d}\left(\left(h_i^S h_j^S\right)\left(\sum_{u\in\mathbf{B}} p_{u,i}p_{u,j}\right)\left(\sum_{t\in\mathbf{U}} c_t^{S-} g_{t,i}g_{t,j}\right)\right), \quad (19)$$

where $\mathbf{X}$ denotes user-user social connections and g_t denotes the latent vector of user t as a friend.

After that, we can integrate both the subtasks of item domain and social domain into a unified multi-task learning framework whose objective function is

$$\mathcal{L}(\Theta) = \tilde{\mathcal{L}_I}(\Theta) + \mu\tilde{\mathcal{L}_S}(\Theta), \quad (20)$$

where μ is the parameter to adjust the weight proportion of each term. The whole framework can be efficiently trained using existing optimizers in an end-to-end manner. More details about the application on social-aware scenarios can be found at our previous conference version [4].

Similarly, in neighbor-based scenarios where items are interacted with both users and other items (Figure 10(b)), the item-based method will be a more suitable choice.

We leave the above extensions of our method as future work.

6.3 Scale Up Model Parameter

Generally, the methods of increasing the number of parameters include: (1) Increasing the embedding size; (2) Increasing the number of deep layers. In our experiments, we have shown that as the latent dimension size increases, the performance of all models increase. This indicates that a larger dimension could capture more hidden factors of users and items, which is beneficial to Top-K recommendation due to the increased modeling capability. This observation is similar to previous work. However, for most deep-learning methods, a larger dimension also requires more time for training. Thus, it is crucial to increase a model's efficiency by learning with efficient optimization methods. For the second method, previous work [9] has shown that deeper models do not necessarily lead to better performance, since they are more easily to be overfitting. As such, neural recommendation models like NCF [23], NFM [19], and so on, usually adopt a two-layer framework in their model designs.

Therefore, to scale up the parameters of our ENMF, we recommend to increase the embedding size or extend the inputs part [4] to introduce complex network architectures like attention mechanisms or memory network.

6.4 Application in Realistic Scenarios

Most recommendation systems [63] contain two stages: candidate generation and ranking. To the best of our knowledge, most academic recommendation studies focus on ranking because of the limitation of dataset and computing resources. For example, previous neural models [20, 23, 55] mainly focus on generating a ranked list from a few hundred candidates due to the efficiency issue. Different from them, our ENMF framework is more scalable for both candidate generation and ranking because of the efficient learning process. For example, on Epinion dataset that contains 20,608 users and 23,585 items, our ENMF only needs 27 minutes to achieve the optimal performance, which is over 30 times faster than the state-of-the-art neural models NCF and ConvNCF.

For vary large systems that contain billions of items and users, a pre-processing process of candidate generation can also be applied before our ENMF methods. For example, Reference [32] used co-occurrences of items to generate candidates, Reference [10] adopted a collaborative filtering-based method, [15] applied a random walk on (co-occurrence) graph, and Reference [7] described a hybrid approach using mixture of features. After the candidate generation stage, our ENMF can be applied to provide a ranked list so that items with highest utility to users will be shown at the top.

7 CONCLUSIONS AND FUTURE WORK

In this work, we propose to learn neural recommendation models from the whole training data without sampling. By reformulating a commonly used square loss function with rigorous mathematical reasoning, we succeed in updating each parameter in a manageable time complexity without sampling. We also devise an Efficient Neural Matrix Factorization (ENMF) framework and propose three instantiations—ENMF-U, ENMF-I, and ENMF-A—based on the newly derived optimization approaches. Extensive experiments have been made on three real-word datasets. The proposed ENMF approaches consistently and significantly outperform the state-of-the-art recommendation models in terms of both recommendation performance and training efficiency.

This work complements the mainstream sampling-based neural models for recommendation with implicit feedback, opening up a new avenue of research possibilities for neural recommendation models. Our efficient whole-data-based learning methods are not limited to the task presented in this article, they have the potential to benefit many other tasks where only positive data is observed. In the future, we are interested in exploring our efficient whole-data-based learning method in other related tasks like content-based recommendation [7], network embedding [34], and multi-domain classification [35]. Also, we will try to extend our optimization method to make it suitable for learning deep models with non-linear prediction layer.

REFERENCES

[1] Chong Chen, Min Zhang, Yiqun Liu, and Shaoping Ma. 2018. Missing data modeling with user activity and item popularity in recommendation. In *Proceedings of the Asia Information Retrieval Symposium*. Springer, 113–125.

[2] Chong Chen, Min Zhang, Yiqun Liu, and Shaoping Ma. 2018. Neural attentional rating regression with review-level explanations. In *Proceedings of the International Conference on World Wide Web (WWW'18)*. 1583–1592.

[3] Chong Chen, Min Zhang, Yiqun Liu, and Shaoping Ma. 2019. Social attentional memory network: Modeling aspect- and friend-level differences in recommendation. In *Proceedings of the ACM International Conference on Web Search and Data Mining (WSDM'19)*. 177–185.

[4] Chong Chen, Min Zhang, Chenyang Wang, Weizhi Ma, Minming Li, Yiqun Liu, and Shaoping Ma. 2019. An efficient adaptive transfer neural network for social-aware recommendation. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*. 225–234.

[5] Jingyuan Chen, Hanwang Zhang, Xiangnan He, Liqiang Nie, Wei Liu, and Tat-Seng Chua. 2017. Attentive collaborative filtering: Multimedia recommendation with item-and component-level attention. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*. 335–344.

[6] Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiaxi Tang, Yixin Cao, Zheng Qin, and Hongyuan Zha. 2018. Sequential recommendation with user memory networks. In *Proceedings of the 11th ACM International Conference on Web Search and Data Mining*. ACM, 108–116.

[7] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for YouTube recommendations. In *Proceedings of the ACM Conference on Recommender Systems (Recsys'16)*. 191–198.

[8] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. 2010. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the 4th ACM Conference on Recommender Systems (Recsys'10)*. ACM, 39–46.

[9] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. 2019. Are we really making much progress? A worrying analysis of recent neural recommendation approaches. *arXiv preprint arXiv:1907.06902*.

[10] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, et al. 2010. The YouTube video recommendation system. In *Proceedings of the Fourth ACM Conference on Recommender Systems (Recsys'10)*. ACM, 293–296.

[11] Robin Devooght, Nicolas Kourtellis, and Amin Mantrach. 2015. Dynamic matrix factorization with priors on unknown values. In *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 189–198.

[12] Jingtao Ding, Guanghui Yu, Xiangnan He, Yuhan Quan, Yong Li, Tat-Seng Chua, Depeng Jin, and Jiajie Yu. 2018. Improving implicit recommender systems with view data. In *Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI'18)*. 3343–3349.

[13] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.* 12 (July 2011), 2121–2159.

[14] Travis Ebesu, Bin Shen, and Yi Fang. 2018. Collaborative memory network for recommendation systems. In *Proceedings of the 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. ACM, 515–524.

[15] Chantat Eksombatchai, Pranav Jindal, Jerry Zitao Liu, Yuchen Liu, Rahul Sharma, Charles Sugnet, Mark Ulrich, and Jure Leskovec. 2018. Pixie: A system for recommending 3+ billion items to 200+ million users in real-time. In *Proceedings of the World Wide Web Conference (WWW'18)*. International World Wide Web Conferences Steering Committee, 1775–1784.

[16] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph neural networks for social recommendation. *arXiv preprint arXiv:1902.07243*.

[17] Chen Gao, Xiangnan He, Dahua Gan, Xiangning Chen, Fuli Feng, Yong Li, Tat-Seng Chua, and Depeng Jin. 2019. Neural multi-task recommendation from multi-behavior data. In *Proceedings of the International Conference on Data Engineering (ICDE'19)*.

[18] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep Learning*. Vol. 1. MIT Press Cambridge.

[19] Xiangnan He and Tat-Seng Chua. 2017. Neural factorization machines for sparse predictive analytics. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*. 355–364.

[20] Xiangnan He, Xiaoyu Du, Xiang Wang, Feng Tian, Jinhui Tang, and Tatseng Chua. 2018. Outer product-based neural collaborative filtering. In *Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI'18)*. 2227–2233.

[21] Xiangnan He, Zhankui He, Xiaoyu Du, and Tatseng Chua. 2018. Adversarial personalized ranking for recommendation. *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval* (2018), 355–364.

[22] Xiangnan He, Zhankui He, Jingkuan Song, Zhenguang Liu, Yugang Jiang, and Tatseng Chua. 2018. NAIS: Neural attentive item similarity model for recommendation. *IEEE Trans. Knowl. Data Eng.* 30, 12 (2018), 2354–2366.

[23] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the World Wide Web Conference (WWW'17)*. 173–182.

[24] Xiangnan He, Jinhui Tang, Xiaoyu Du, Richang Hong, Tongwei Ren, and Tat-Seng Chua. 2019. Fast matrix factorization with nonuniform weights on missing data. *IEEE Trans. Neural Netw. Learn. Syst.* DOI: 10.1109/TNNLS.2018.2890117

[25] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. 2016. Fast matrix factorization for online recommendation with implicit feedback. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*. 549–558.

[26] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *Proceedings of the International Conference on Data Mining (ICDM'08)*. 263–272.

[27] Yehuda Koren. 2008. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 426–434.

[28] Yehuda Koren. 2009. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 447–456.

[29] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer*8 (2009), 30–37.

[30] Dawen Liang, Laurent Charlin, James McInerney, and David M. Blei. 2016. Modeling user exposure in recommendation. In *Proceedings of the World Wide Web Conference (WWW'16)*. 951–961.

[31] Xiao Lin, Zhang Min, Zhang Yongfeng, Liu Yiqun, and Ma Shaoping. 2017. Learning and transferring social and item visibilities for personalized recommendation. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM'17)*. 337–346.

[32] David C. Liu, Stephanie Rogers, Raymond Shiau, Dmitry Kislyuk, Kevin C. Ma, Zhigang Zhong, Jenny Liu, and Yushi Jing. 2017. Related pins at pinterest: The evolution of a real-world recommender system. In *Proceedings of the 26th International Conference on World Wide Web Companion*. International World Wide Web Conferences Steering Committee, 583–592.

[33] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. 2015. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 43–52.

[34] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of the International Conference on Neural Information Processing Systems (NIPS'13)*. 3111–3119.

[35] Hyeonseob Nam and Bohyung Han. 2016. Learning multi-domain convolutional neural networks for visual tracking. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR'16)*. 4293–4302.

[36] Shumpei Okura, Yukihiro Tagami, Shingo Ono, and Akira Tajima. 2017. Embedding-based news recommendation for millions of users. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1933–1942.

[37] Rong Pan, Yunhong Zhou, Bin Cao, Nathan N. Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. 2008. One-class collaborative filtering. In *Proceedings of the 8th IEEE International Conference on Data Mining*. IEEE, 502–511.

[38] István Pilászy, Dávid Zibriczky, and Domonkos Tikk. 2010. Fast ALS-based matrix factorization for explicit and implicit feedback datasets. In *Proceedings of the ACM Recommender Systems Conference (Recsys'10)*. 71–78.

[39] Steffen Rendle. 2010. Factorization machines. In *Proceedings of the International Conference on Data Mining (ICDM'10)*. 995–1000.

[40] Steffen Rendle and Christoph Freudenthaler. 2014. Improving pairwise learning for item recommendation from implicit feedback. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*. ACM, 273–282.

[41] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI'09)*. 452–461.

[42] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. Introduction to recommender systems handbook. In *Recommender Systems Handbook*. Springer, 1–35.

[43] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. 2007. Restricted Boltzmann machines for collaborative filtering. In *Proceedings of the 24th International Conference on Machine Learning*. ACM, 791–798.

[44] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the International Conference on World Wide Web (WWW'01)*. 285–295.

[45] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. 2015. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th International Conference on World Wide Web*. ACM, 111–112.

[46] Brent Smith and Greg Linden. 2017. Two decades of recommender systems at Amazon. com. *IEEE Internet Comput.* 21, 3 (2017), 12–18.

[47] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting.*J. Mach. Learn. Res.* 15, 1 (2014), 1929–1958.

[48] Xiaoyuan Su and Taghi M. Khoshgoftaar. 2009. A survey of collaborative filtering techniques. *Adv. Artific. Intell.* 2009, Article 421425 (2009), 19 pages. DOI: http://dx.doi.org/10.1155/2009/421425

[49] Peijie Sun, Le Wu, and Meng Wang. 2018. Attentive Recurrent Social Recommendation. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*. 185–194.

[50] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. 2018. Latent relational metric learning via memory-based attention for collaborative ranking. In *Proceedings of the 24th International Conference on World Wide Web (WWW'18)*. 729–739.

[51] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. 2013. Deep content-based music recommendation. In *Advances in Neural Information Processing Systems*. MIT Press, 2643–2651.

[52] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. 2017. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 515–524.

[53] Menghan Wang, Mingming Gong, Xiaolin Zheng, and Kun Zhang. 2018. Modeling dynamic missingness of implicit feedback for recommendation. In *Proceedings of the Neural Information Processing Systems Conference (NeuIPS'18)*. 6669–6678.

[54] Xiang Wang, Xiangnan He, Liqiang Nie, and Tat-Seng Chua. 2017. Item silk road: Recommending items from information domains to social users. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*. 185–194.

[55] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*.

[56] Yao Wu, Christopher DuBois, Alice X. Zheng, and Martin Ester. 2016. Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the ACM International Conference on Web Search and Data Mining (WSDM'16)*. ACM, 153–162.

[57] Xin Xin, Fajie Yuan, Xiangnan He, and Joemon M. Jose. 2018. Batch IS NOT heavy: LearningWord representations from all samples. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 1853–1862.

[58] Feng Xue, Xiangnan He, Xiang Wang, Jiandong Xu, Kai Liu, and Richang Hong. 2019. Deep item-based collaborative filtering for top-N recommendation. *ACM Trans. Info. Syst.* 37, 3 (2019), 33.

[59] Wenhui Yu, Huidi Zhang, Xiangnan He, Xu Chen, Li Xiong, and Zheng Qin. 2018. Aesthetic-based clothing recommendation. In *Proceedings of the International Conference on World Wide Web (WWW'18)*. 649–658.

[60] Fajie Yuan, Xin Xin, Xiangnan He, Guibing Guo, Weinan Zhang, Chua Tat-Seng, and Joemon M. Jose. 2018. fbgd: Learning embeddings from positive unlabeled data with BGD. In *The Conference on Uncertainty in Artificial Intelligence*. 2018.

[61] Shuai Zhang, Lina Yao, and Xiwei Xu. 2017. Autosvd++: An efficient hybrid collaborative filtering model via contractive auto-encoders. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 957–960.

[62] Yongfeng Zhang, Qingyao Ai, Xu Chen, and W. Bruce Croft. 2017. Joint representation learning for top-n recommendation with heterogeneous information sources. In *Proceedings of the ACM Conference on Information and Knowledge Management*. ACM, 1449–1458.

[63] Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumthekar, Maheswaran Sathiamoorthy, Xinyang Yi, and Ed Chi. 2019. Recommending what video to watch next: A multitask ranking system. In *Proceedings of the 13th ACM Conference on Recommender Systems*. ACM, 43–51.

[64] Lei Zheng, Vahid Noroozi, and Philip S. Yu. 2017. Joint deep modeling of users and items using reviews for recommendation. In *Proceedings of the 10th ACM International Conference on Web Search and Data Mining*. ACM, 425–434.

Received July 2019; revised October 2019; accepted November 2019