Information Sciences 470 (2019) 121–140

Contents lists available at ScienceDirect

Information Sciences

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

DeepRec: A deep neural network approach to recommendation with item embedding and weighted loss function

Wen Zhang[a,b,*], Yuhang Du[b], Taketoshi Yoshida[c], Ye Yang[d]

[a] School of Economics and Management, Beijing University of Technology, Beijing 100124, PR China
[b] Research Center on Big Data Sciences, Beijing University of Chemical Technology, Beijing 100029, PR China
[c] School of Knowledge Science, Japan Advanced Institute of Science and Technology, 1-1 Ashahidai, Nomi, Ishikawa 923-1292, Japan
[d] School of Systems and Enterprises, Stevens Institute of Technology, Hoboken, NJ 07030, USA

ARTICLE INFO

Article history:
Received 30 October 2017
Revised 10 August 2018
Accepted 12 August 2018
Available online 23 August 2018

Keywords:
DeepRec
Deep neural network
Recommender system
Item embedding
Weighted loss function

ABSTRACT

Traditional collaborative filtering techniques suffer from the data sparsity problem in practice. That is, only a small proportion of all items in the recommender system occur in a user's rated item list. However, in order to retrieve items meeting a user's interest, all possible candidate items should be investigated. To address this problem, this paper proposes a recommendation approach called DeepRec, based on feedforward deep neural network learning with item embedding and weighted loss function. Specifically, item embedding learns numerical vectors for item representation, and weighted loss function balances popularity and novelty of recommended items. Moreover, it introduces two strategies, i.e. sampling by random (Ran-Strategy) and sampling by distribution (Pro-Strategy), to leave one item as output and the remaining as input from each user's historically rated item list. Max-pooling and average-pooling are employed to combine individual item vectors to derive users' input vectors for feedforward deep neural network learning. Experiments on the App dataset and the Last.fm dataset demonstrate that the proposed DeepRec approach is superior to state-of-the-art techniques in recommending Apps and songs in terms of accuracy and diversity as well as complexity.

1. Introduction

With the development of online E-commerce, information overload about products and services is pervasive across the Internet. For instance, in online shopping platforms such as Taobao and Amazon, numerous products and services are constantly available to meet potential diverse demands. Meanwhile, for online shoppers, retrieving satisfactory products and services from those hundreds of thousands of options is very time consuming burden. For instance, thousands of Apps are available in online App stores for users to download and use. In order to better promote products and alleviate users' burden, recommender system has become an essential component of online platforms and markets [2,17].

For simplicity, we can treat the recommendation problem as relatedness link prediction between users and items in a bipartite network, which consists of two types of nodes as users and items [35]. If a user u has interest on an item i,

* Corresponding author.
E-mail addresses: zhangwen@bjut.edu.cn (W. Zhang), 2016200852@mail.buct.edu.cn (Y. Du), yoshida@jaist.ac.jp (T. Yoshida), ye.yang@stevens.edu (Y. Yang).

https://doi.org/10.1016/j.ins.2018.08.039
0020-0255/

then there is a link to relate the user u and the item i. Otherwise, there is no link relating the user u and the item i. We can further define the relatedness as weighted link and unweighted link for the scenarios of rating with more than two scales and rating with binary scale. For example, in the former case, a user rates a movie in the MovieLens dataset[1] from 1 to 5 to express her likeness on a movie; in the latter case, a user expresses her attitude as "like" or "dislike" on the movie or the recommender system captures that the user has "watched" or "not watched" the movie by implicit feedback. Recently, the attention on recommendation for the former case is diminishing as intensive labor needed in collecting rating data, inaccurate rating by users and the goal of recommendation as relevant item locating rather than rating prediction [32]. However, there is an increasing trend to study the recommendation problem in the latter case as it is easy for a recommender system to collect the implicit feedbacks such as songs listened to, items bought, Apps installed, etc. To put it in a formal way, we call the recommendation problem in the latter case as recommendation for binary, positive-only data [32].

Generally, we can use the historical data that records the users' rating on the items for recommendation. However, a series of techniques are present to make use of more than historical rating data for recommendation such as content-based techniques [25], knowledge-based techniques [7,21] and social-network based techniques [3,18]. Content-based techniques make use of textual descriptions of items to define users' preferences and further use textual similarity between items and users' preferences to recommend new items to users. Knowledge-based techniques make use of domain knowledge of items and users to produce recommendations. Domain knowledge is frequently expressed via ontology, and relations among items and users are derived through formal inference on concepts from the ontology structure. Social network based techniques capture social interactions among users, such as online friending, commenting, tagging, etc. in order to characterize the so-called "Trust" relationship for social recommendation [4,36]. In this paper, we use historical rating data from users' implicit feedbacks for recommendation, without requiring any other additional information.

One of the challenging problems in recommendation is the issue of data sparsity [34]. That is, most users only have ratings for a very small percentage of items and a small number of users have ratings for a very large percentage of items. If the items' rating frequencies by theirs ranks were plot, we would find an obvious long tail phenomenon, i.e. most items have a very small number of ratings [26]. Data sparsity introduces difficulty in recommendation due to three factors. The first factor is the inability of traditional collaborative filtering algorithms to accurate recommendation. If a user-item matrix was derived from historical rating data, most of the elements in the matrix will be zeros, which usually has a sparsity ranging from 0.98 to 0.99. Consequently, 99% of all links between users and items will need to be predicted by using merely 1% of all links instances in the user-item bipartite network in recommendation [32]. The second factor lies in the huge computation involved in parameter updating for machine learning algorithms. If a $1 \times |U|$ vector (U denotes the user set in historical rating data) is used to represent an item for statistical machine learning, then most of the elements in the vector are zeros however these zero elements cannot be ignored in pattern learning. The processing will require tremendous computation in updating parameters in learning model for recommendation. The third factor is trivial recommendation of items. If a recommender system merely recommends popular items for all users, then its performance measured by accuracy would be encouraging because popular items occur most frequently in users' rating lists. However, the recommendation result will be trivial for users because there would never be new items to be recommended and this will make users bored with the recommender [10].

This paper proposes a recommendation approach called DeepRec based on deep neural network learning with item embedding [5] and weighted loss function. To improve the recommendation accuracy, DeepRec employs a deep neural network with 3 ReLU layers for output vector regression. To address the huge computation problem, DeepRec employs item embedding to represent each item as a dense numeric vector. To ensure novelty of recommended item, DeepRec employs weighted loss function for the deep neural network based on popularity of items in historical rating dataset. The experiments on the App dataset and the Last.fm dataset demonstrate that DeepRec outperforms state-of-the-art techniques on both accuracy and diversity.

The remainder of the paper is organized as follows. Section 2 describes the research problem. Section 3 summarizes the related work. Section 3 proposes the DeepRec approach, including item embedding and weighted loss function. Section 4 presents results from experiment conducted on two evaluation dataset. Section 5 concludes the paper.

2. Problem statement

This paper considers a typical recommendation problem in its simplest formation as follows. For a recommender system, it has $|U|$ users as $U = \{u_1, ..., u_{|U|}\}$ and $|I|$ items as $I = \{i_1, ..., i_{|I|}\}$ and a historical rating data described as a matrix as $R_{|U| \times |I|}$ where each element $r_{k,j}$ is either one or zero denoting whether or not a user u_k has installed an App i_j or whether nor not a user u_k has watched a movie i_j. We can treat a row vector $r_{k,.}$ as the historical rated item list for user u_k and a column vector $r_{.,j}$ as the user list who rated item i_j in history. Thus, the problem addressed in this paper is quite straightforward. That is, given a user u_k and the historical rating matrix $R_{|U| \times |I|}$, we need to predict which items in item set I but not in $r_{k,.}$ will be rated in the near future by the user u_k and thus recommend these items to user u_k. For instance, in an App market, we need to predict which Apps a user will install in her/his smart phone according to her/his installation history. In a Movie

[1] MovieLens, online: https://grouplens.org/datasets/movielens/.

website, we need to predict which movie the user would also like to watch according to her/his movie watching history. From the above analysis, the problem we study in this paper is to predict the user-item rating likelihood by making use of the information in the historical rating matrix $R_{|U|\times|I|}$.

3. Related work

The related work of the paper includes two aspects: collaborative filtering for recommendation and deep learning in recommendation. Matrix factorization and neighborhood models are often than not adopted methods in collaborative filtering. RBM (Restricted Boltzmann Machine) and NADE (Neural Autoregressive Distribution Estimator) are widely accepted deep learning methods recently.

3.1. Collaborative filtering for recommendation

Collaborative filtering (CF) is a class of methods for predicting a user's rating of an item, based on his/her previous ratings and the ratings made by similar users [17]. When using matrix factorization for collaborative filtering, an element $r_{k,j}$ in the historical rating matrix $R_{|U|\times|I|}$ is estimated as $\hat{r}_{k,j}$ in Eq. (1), where the vector p_k is used to characterize the user u_k's preferences, and the vector q_j is used to characterize the item i_j's features in the recommendation system. To learn the representation vectors p_k and q_j, we usually use matrix factorization methods such as pLSA [11] and least square error minimization [30] by using the observed data.

$$\hat{r}_{k,j} = q_j^T p_k. \tag{1}$$

The neighborhood methods for collaborative filtering generally include user-based method [12] and item-based method [9]. The basic idea of user-based method is based on the assumption that two users u_l and u_q having similar interest in history would also rate similar items in the future. User-based method [12] models the ratings on items of user u_k as $r_{k,.}$ in the historical rating matrix $R_{|U|\times|I|}$. The rating similarity on items between two users u_l and u_q is measured by the Jaccard coefficient [19] between two vectors $r_{l,.}$ and $r_{q,.}$. Thus, the items rated frequently by the similar users of user u_k are recommended to him/her. Similarly, the basic idea of item-based method is based on the assumption that since a user has rated an item i_j in the history, he or she would also rate similar items with item i_j in the future. Item-based method [9] represents an item i_j as $r_{.,j}$ in the historical rating matrix $R_{|U|\times|I|}$. The similarity between item i_l and item i_q is measured by Jaccard coefficient between two vectors $r_{.,l}$ and $r_{.,q}$ [12]. Thus, the items which are similar to the items rated by user u_k in history are recommended to him/her.

3.2. Deep learning for recommendation

Due to the recent success of deep learning in computer vision, natural language processing, speech recognition, etc., researchers also attempt to use deep learning in recommendation systems [20]. For instance, researchers at Google propose to use deep neural network for YouTube video recommendation [8]. They split video recommendation into two distinct problems as candidate generation and ranking and, use different deep neural network architectures to deal with them. They also point out that in real practice of using deep learning for video recommendation, there is more art than science in feature engineering, candidate selection and ranking because human beings co-watching behaviors are very difficult to characterize.

Salakhutdinov et. al. [27] propose to use RBM for collaborative filtering to model user-item interaction and perform recommendations. Their proposed deep neural network is shown in Fig. 1 where V is a $K\times|I|$ observed binary indicator matrix with $v_j^l = 1$ if the user rated item i_j as l. In the context of the paper in considering the binary, positive-only data, i.e. the elements $r_{k,j}$ in the historical rating matrix $R_{|U|\times|I|}$ belong to {0, 1}, the parameter K is equal to 2 and l is either zero or one. The hidden nodes h_f ($f = 1, \ldots, |H|$) in the RBM network have binary values of hidden (latent) variables that can be treated as representing stochastic binary features having different values for different users. The connections between visible units V and hidden nodes H are associated with a matrix of weights $W = (w_{i,j})_{(2\times|I|)\times|H|}$ where each item i_j is represented by a length-2 vector. $|V|$ is the number of visible units and $|H|$ is the number of hidden units.

RBM is trained to find an optimal weight matrix $W = (w_{i,j})_{(2\times|I|)\times F}$ to maximize the product of probabilities assigned to the training set of inputs V. Usually, the contrastive divergence proposed by Hinton et al. [15] is used to train the RBM by using Gibbs sampling [13] and gradient descent procedure [28]. After the RBM model is trained to local minima, we can use the derived weight matrix W and the bias vectors b for prediction by using Eqs. 2 and 3. Here, σ is the sigmoid activation function, W_{jf}^l is the weight of the connection between v_j^l and h_f, b_j^l is the bias on node v_j^l and b_f is the bias on node h_f.

$$\hat{p}_f = p(h_f = 1|V) = \sigma\left(b_f + \sum_{j=1}^{|I|}\sum_{l=1}^{K} v_j^l W_{jf}^l\right) \tag{2}$$

$$p\left(v_j^l = 1|\hat{\mathrm{P}}\right) = \frac{\exp\left(b_j^l + \sum_{f=1}^{F}\hat{p}_f W_{jf}^l\right)}{\sum_{s=1}^{K}\exp\left(b_j^s + \sum_{f=1}^{F}\hat{p}_f W_{jf}^s\right)} \tag{3}$$

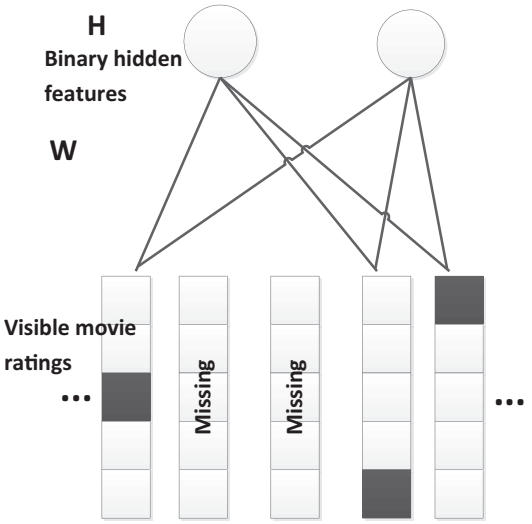

Fig. 1. The deep neural network of RBM used for collaborative filtering.

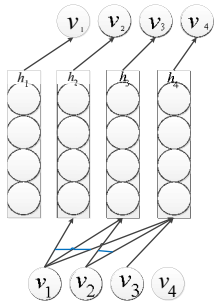

Fig. 2. The deep neural network of NADE used for collaborative filtering. The blue lines indicate shared or tied parameters.

Inspired by the RBM model, Zheng et al. [37] propose to use neural autoregressive distribution estimator (NADE) for recommendation shown in Fig. 2. The basic idea of NADE is that since there too many parameters such as W and b involved in the RBM model resulting in huge updating in computation, NADE simply estimate the probability $p(v_j = 1|v_{<j})$ by sharing parameters between visible nodes V and hidden nodes H. Here, $v_{<j}$ refers to those nodes whose indices are smaller than j in the visible node sequence. We can see from Fig. 2 that $\hat{v}_2$, i.e. $p(\hat{v}_2 = 1|v_{<2})$ is estimated given v_1 and $\hat{v}_3$ is estimated given v_1 and v_2. Moreover, the weights $W_{1,2}$ and $W_{1,3}$ are equivalent to each other as the shared parameter from the visible node v_1 to make the computation to be convergent more rapidly. The estimates of visible nodes $p(\hat{v}_j = 1|v_{<j})$ and hidden variables h_f are updated using Eqs. (4) and (5), iteratively. Thus, training under NADE is done by minimizing the average negative log-likelihood of the parameters given the training set.

$$p\left(\hat{v}_j = 1|v_{<j}\right) = \sigma\left(b_j + \left(W^T\right)_{j,\cdot} h_j\right) \tag{4}$$

$$h_j = \sigma\left(c + W_{\cdot,<i} v_{<i}\right) \tag{5}$$

4. The proposed DeepRec approach

This section proposes the DeepRec approach to use feedforward deep neural network for recommendation. We describe the details of the DeepRec approach as follows.

4.1. The architecture of DeepRec

The basic idea of the DeepRec approach is to treat the recommendation problem as a regression problem for prediction, by making use of a user's historical item rating list to regress his/her preferred item in the future. Fig. 3 shows the architecture of the DeepRec approach for recommendation. The input of DeepRec is a user's historical rating item list and the output is the user's preferred item. We can see that actually, DeepRec is a feedforward deep neural network that comprises

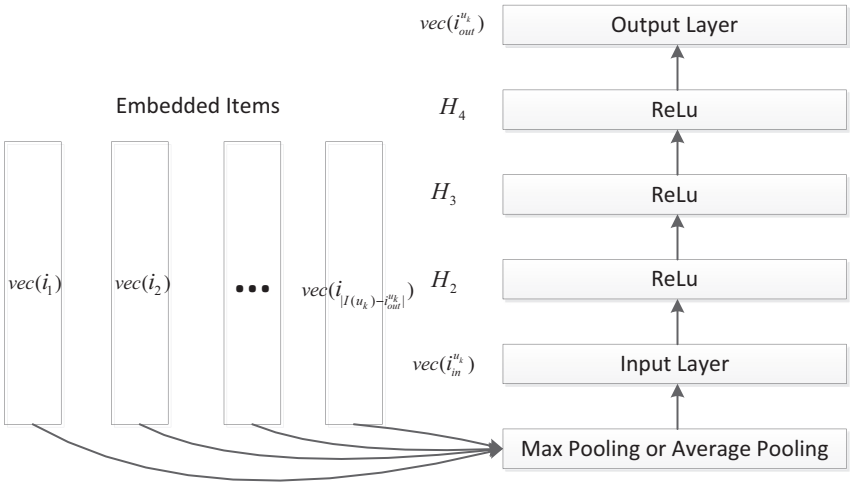

Fig. 3. The architecture of the DeepRec approach for recommendation.

five layers. The first layer is the input layer H_{in} to feed input vectors for the network. The second to fourth layers are the 3 ReLU layers (H_2 to H_4) to learn the hidden representations from input vectors to output vectors. The fifth layer is the output layer H_{out} to predict the preferred item using regression. The reason to use ReLu layers as hidden layers is that rectified linear unit (ReLU), i.e. $\max(0, x)$, is a highly useful activation function that doesn't saturate on shallow gradients as the sigmoid activation function does [24].We also attempted other layer options to construct the deep neural network, and found that five layers is an appropriate choice for recommendation. This architecture is also validated by Covington et al. [8] in recommending YouTube videos.

4.2. Item embedding

Item embedding is proposed by Barken and Koenigstein [5] which is actually an application of word embedding [22] in the field of recommendation. This basic idea is to treat an item is a user's item list as a word in a text. Thus, a static environment is assumed in recommendation where all items are rated by users in the same context without considering the order and times of the items rated by users like session-based recommendation [14]. Although in this way the spatial and time information of items is lost, it can still produce better performance than traditional methods like $1 \times |U|$ representation as $r_{.,j}$ in Section 2 for item representation [5].

In DeepRec, we assume that items were rated by users at an arbitrary order and sort the items in users' item list in ascending order alphabetically to align the items for embedding. Otherwise, there would be no unique embedding for items if without sorting them. By item embedding, the sparse vector $r_{.,j}$ with length $|U|$ can be avoided to represent item i_j and a dense numeric vector $vec(i_j)$ is used for item representation in computation. Usually, the length of vector $vec(i_j)$ is much smaller than $|U|$ resulting in that a great computation can be saved.

In this paper, we adopt the Skip-gram model for item embedding as its simplicity and computation efficiency. We sort the items of each user alphabetically and use the item i_j to predict its 21 maximum distance items (i_{j-10},...,i_{j-1}, i_{j+1},..., i_{j+10}). Actually, we also experimented different parameters as the maximum distance and concluded that an item with its 21 surrounding items has produced the best performance in the paper. Note that i_{j-10},...,i_{j-1},i_{j+1},...,i_{j+10} are all continuous numerical vectors and the projection can be treated as a neural network with one layer of hidden units. For brevity, the Skip-gram model used for item embedding can be depicted in Fig. 4.

The goal of the Skip-gram model training is to tune the item vectors that can be used to predict the surrounding items in a user's rating history. That is, given a sequence of training items i_1, i_2, ..., i_T, the Skip-gram needs to maximize the average log probability

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\le j\le c}\log p(i_{t+j}|i_t). \tag{6}$$

Here, c is the maximum distance from the item i_t. The larger value c is, the more training examples are involved in the model leading to a higher accuracy of the model but with more computation complexity. The basic Skip-gram formulation defines $p(i_{t+j}|i_t)$ using the softmax function

$$p\left(i_{t+j}|i_t\right) = \frac{\exp\left(vec'\left(i_{t+j}\right)^T vec(i_t)\right)}{\sum_{w=1}^{|I|}\exp\left(vec'(i_w)^T vec(i_t)\right)}. \tag{7}$$

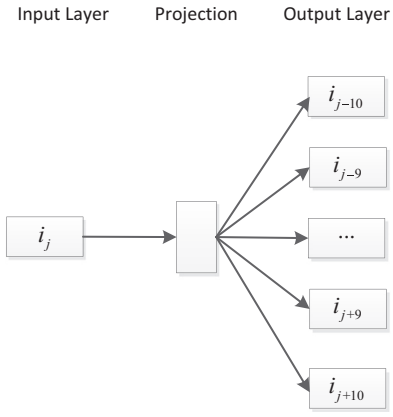

Fig. 4. The Skip-gram model.

$v'(i_w)$ is the output word vector for the item i_w and $|I|$ is the size of the item set. In practice, it is very difficult to maximize Eq. (6) if $|I|$ is huge because, at each run of gradient descent, it will involve updating $O(e|I|)$ parameters where e is the length of the item embedding vector $vec(i_w)$. Thus, negative sampling [5] is employed to solve this problem to compute $p(i_{t+j}|i_t)$ within a scope of k negative samples. That is to say, for each observation (i_{t+j}, i_t), we draw k negative samples by using the empirical unigram distribution $p(i_w) = \frac{\#(i_w)}{|I|}$ and the Eq. (7) is revised to Eq. (8) where i_w is one of the k negative samples and $\#(i_w)$ is the number of occurrences of item i_w in users' rating lists. Usually, k is in the range 5–20 are useful for small training datasets, while for large datasets the k can be as small as 2–5.

$$p(i_{t+j}|i_t) = \frac{\exp(vec'(i_{t+j})^T vec(i_t))}{\sum_{w=1}^{k} \exp(vec'(i_w)^T vec(i_t))}. \tag{8}$$

One can refer to Barken and Koenigstein [5] for more details of item embedding and Mikolov et al. [22] for more details on solution of the Skip-gram model. After the processing of item embedding, each item is represented by a length-e dense numeric vector where the parameter e is the number of features used to represent an item in the Skip-gram model.

We also evaluated max pooling and average pooling respectively to produce the representation vectors of users' historical preferences on items. The max pooling is to extract the maximum components of features from all historical items' embedding vectors in $I(u_k) - i_{out}^{u_k}$ to produce the representation vector of input, i.e. $vec(i_{in}^{u_k}) = \{\max \bigcup_{j=1}^{|I(u_k)-i_{out}^{u_k}|} vec(i_j)_1, ..., \max \bigcup_{j=1}^{|I(u_k)-i_{out}^{u_k}|} vec(i_j)_l, ..., \max \bigcup_{j=1}^{|I(u_k)-i_{out}^{u_k}|} vec(i_j)_e\}$. The average pooling is used to average all the components of features of all historical items' embedding vectors in $I(u_k) - i_{out}^{u_k}$ to produce the representation vector of input, i.e. $vec(i_{in}^{u_k}) = \sum_{j=1}^{|I(u_k)-i_{out}^{u_k}|} vec(i_j)$.

4.3. Network training and recommendation

In order to train DeepRec, we leave one item out from a user's historical rating item list as output item and the remaining items are used as input items. The rationale is that we treat the input items as historical rating list and the output item as the most preferred item of the user in near future to build the predictive model. That is, for a user u_k, we retrain one item $i_{out}^{u_k}$ ($i_{out}^{u_k} \in I(u_k)$) as the output item from her rating history $I(u_k) = \{i_1^{u_k}, ..., i_{|I(u_k)|}^{u_k}\}$ and all the items in $I(u_k)$-$i_{out}^{u_k}$ are used as the input items. In fact, $I(u_k)$ contain the items whose values in vector $r_{k,.}$ in Section 2 are equal to 1. In other words, $vec(i_{out}^{u_k})$ is assigned as the output vector and the vectors $\bigcup_{i_j \in I(u_k)-i_{out}^{u_k}} vec(i_j)$ are used to derive the representation vector of input $vec(i_{in}^{u_k})$. Further, we develop two strategies to retain the output item $i_{out}^{u_k}$ as random sampling strategy (Ran-Strategy) and probability based sampling strategy (Pro-Strategy). Under the Ran-Strategy, we select the output item $i_{out}^{u_k}$ randomly at equal probability from $I(u_k)$. Under the Pro-Strategy, we select the output item $i_{out}^{u_k}$ based on its popularity $p(i_{out}^{u_k})$ in the history. That is to say, the probability of an item being assigned in the output item $i_{out}^{u_k}$ is equal to the probability being rated by users in history.

The training of DeepRec is quite straightforward. Firstly, the input vector of the first hidden layer H_2 is $vec(i_{in}^{u_k})$ and the output with ReLU activation function is $y_k^{(2)}$ as described in Eq. (9). Here, W_2^T is weight matrix and b_2 is the bias vector in layer H_2 to map input vector $vec(i_{in}^{u_k})$ to output vector $y_k^{(2)}$.

$$y_k^{(2)} = \max\left(0, W_2^T vec(i_{in}^{u_k}) + b_2\right) \tag{9}$$

By analogy, the input vector of the second hidden layer H_3 is $y_k^{(2)}$ and its output with ReLU activation function is $y_k^{(3)}$ as described in Eq. (10) and the same computation is conducted in the third hidden layer H_4 as described in Eq. (11). The parameters W_3^T weight matrix and b_3 bias vector belong to the second layer and the parameters W_4^T weight matrix and b_4 bias vector belong to the third layer.

$$y_k^{(3)} = \max\left(0, W_3^T y_k^{(2)} + b_3\right) \tag{10}$$

$$y_k^{(4)} = \max\left(0, W_4^T y_k^{(3)} + b_4\right) \tag{11}$$

Finally, the input vector of the output layer H_{out} with liner units is $y_k^{(4)}$ and the output vector is $y_k^{(5)}$ as described in Eq. (12) with weight matrix as W_5^T and bias vector b_5.

$$y_k^{(5)} = W_5^T y_k^{(4)} + b_5 \tag{12}$$

Thus, from the above training procedure of the proposed DeepRec approach, the parameters we need to decide for the deep neural network are $\theta = \{W_2, b_2, W_3, b_3, W_4, b_4, W_5, b_5\}$ where $W_2 \in R^{e\times H_2}$, $b_2 \in R^{H_2}$,$W_3 \in R^{H_2\times H_3}$, $b_3 \in R^{H_3}$, $W_4 \in R^{H_3\times H_4}$, $b_4 \in R^{H_4}$, $W_5 \in R^{H_4\times e}$, $b_5 \in R^e$. The DeepRec approach is trained in traditional method by gradient descent procedure. That is, for the loss as $Loss(u_k) = Loss(y_k^{(5)}, vec(i_{out}^{u_k}))$ of u_k, where $y_k^{(5)}$ is the predicted vector and $vec(i_{out}^{u_k})$ is the actual vector, the overall loss on all users is $Loss^{sum} = \sum_{k=1}^{|U|} Loss(u_k)$. Thus, the weight matrices and bias vectors are updated iteratively as $W_i := W_i + \varepsilon \frac{\partial Loss^{sum}}{\partial W_i}$ and $b_i := b_i + \varepsilon \frac{\partial Loss^{sum}}{\partial b_i}$ $(i = 2, 3, 4, 5)$. Usually, the chain rule is employed to implement the updating process of parameters of feedforward deep neural network effectively [31].

When using the trained DeepRec for recommendation, we first aggregate a user's rating items in history to derive its embedding vectors $I(u_l)$. Second, the average pooling or the max pooling is used to transfer the embedding vectors into an input vector $vec(i_{in}^{u_l})$ to represent the user's history preference. Third, the vector $vec(i_{in}^{u_l})$ is fed into the feedforward deep neural network to predict the output vector $y_l^{(5)}$. Finally, the top item I $(i_{u_l}^*)$ which is most similar to the out vector $y_l^{(5)}$ by cosine similarity measure is treated as the user's most preferred item for recommendation.

4.4. Weighted loss function

The DeepRec approach adopts a weighted loss function $Loss^{sum}(\theta)$ as $Loss^{sum}$ for item recommendation to users where θ represents the parameters for the feedforward deep neural network, i.e. $\{W_2, b_2, W_3, b_3, W_4, b_4, W_5, b_5\}$. The basic idea is that in fact, the items under recommendation are rated by users with different popularities. It is not difficult to understand this outcome that some items are rated by a large number of users and other items are rated by a small number of users. We call the former items as "hot" items and the latter items as "cold" items. Usually, if the "hot" items were recommended, the accuracy of recommendation will be improved but the diversity of recommendation will be hurt. The opposite case would occur if the "cold" items were recommended. Following this line of thought, we devise a weighted loss function as described in Eq. (13) to consider accuracy and diversity of recommendation comprehensively where a linear interpolation function as described in Eq. (14) is used to weight the items used for the actual output of the DeepRec approach.

$$Loss^{sum}(\theta) = \frac{1}{|U|}\sum_{k=1}^{|U|} w\left(i_{out}^{u_k}\right)||y_k^{(5)} - vec\left(i_{out}^{u_k}\right)||^2 \tag{13}$$

$$w(i_{out}^{u_k}) = \frac{\left(\ln p\left(i_{out}^{uk}\right) - \ln p_{\min}\right)}{\left(\ln p_{\max} - \ln p_{\min}\right)} \times (0.6 - 0.5) + 0.5 \tag{14}$$

We use liner interpolation to weight each item in the range [0.5, 0.6]. $p(i_{out}^{u_k})$ denotes the probability of item $i_{out}^{u_k}$ in users' rating history. $p_{\max}$ and $p_{\min}$ denote maximum probability and minimum probability in users' rating history, respectively. Essentially, the loss function of DeepRec $L(\theta)$ is a weighted mean square error between the predicted item vector $y_k^{(5)}$ and the actual item vector $vec(i_{out}^{u_k})$. The reasons for us to use sigmoid function in weighting lie in that on the one hand, it is an increasing function making "hot" items having larger weights than "cold" items. On the other hand, the sigmoid function $w(i_{out}^{u_k})$ squashes the probabilities $p(i_{out}^{u_k})$ into the range [0.5, 0.65] resulting in that there would be only a small difference in weights between "hot" and "cold" items. We admit that the adopted linear interpolation function $w(i_{out}^{u_k})$ is a little arbitrary for item weighting in the loss function $L(\theta)$. However, to the best of our knowledge, there is no better method to set the weights than trial and error. The similar idea is adopted by Aiolli [1] to assign the popular items with less importance in measuring the similarity of two items.

With the devised loss function in Eq. (10), the gradient of $y_k^{(5)}$ can be computed directly in Eq. (15).

$$\frac{\partial Loss^{sum}(\theta)}{\partial y_k^{(5)}} = \frac{2}{|U|}\sum_{k=1}^{|U|} w\left(i_{out}^{u_k}\right)\left(y_k^{(5)} - vec\left(i_{out}^{u_k}\right)\right) \tag{15}$$

By the chain rule, the gradients of $y_k^{(4)}$, W_5 and b_5 can be derived by using Eqs. (16)–(18) accordingly.

$$\frac{\partial Loss^{sum}(\theta)}{\partial y_k^{(4)}} = \frac{\partial Loss^{sum}(\theta)}{\partial y_k^{(5)}} \frac{\partial y_k^{(5)}}{\partial y_k^{(4)}} = W_5^T \frac{\partial Loss^{sum}(\theta)}{\partial y_k^{(5)}} \tag{16}$$

$$\nabla W_5 = \frac{\partial Loss^{sum}(\theta)}{\partial W_5} = \frac{\partial Loss^{sum}(\theta)}{\partial y_k^{(5)}} \frac{\partial y_k^{(5)}}{\partial W_5} = \frac{\partial Loss^{sum}(\theta)}{\partial y_k^{(5)}} \cdot (y_k^{(4)})^T \tag{17}$$

$$\nabla b_5 = \frac{\partial Loss^{sum}(\theta)}{\partial b_5} = \frac{\partial Loss^{sum}(\theta)}{\partial y_k^{(5)}} \frac{\partial y_k^{(5)}}{\partial b_5} = \frac{\partial Loss^{sum}(\theta)}{\partial y_k^{(5)}} \tag{18}$$

For a ReLU layer, its gradient descent is a little bit different from a linear continuous function for its expression as a piecewise function. Here, we take the H_4 ReLU layer as an example. For $y_k^{(3)}$, W_4 and b_4, their gradients are derived in Eqs. (19)–(21), respectively.

$$\begin{aligned}
&\left(\frac{\partial Loss^{sum}(\theta)}{\partial y_k^{(3)}}\right)_j = 0, \text{ for } \left(y_k^{(4)}\right)_j = 0;\\
&\left(\frac{\partial Loss^{sum}(\theta)}{\partial y_k^{(3)}}\right)_j = \left(W_4^T \frac{\partial Loss^{sum}(\theta)}{\partial y_k^{(4)}}\right)_j, \text{ for } \left(y_k^{(4)}\right)_j \neq 0;\\
&(\nabla W_4)_{(:,j)} = 0, \text{ for } \left(y_k^{(4)}\right)_j = 0;
\end{aligned} \tag{19}$$

$$(\nabla W_4)_{(:,j)} = \left(\frac{\partial Loss^{sum}(\theta)}{\partial y_k^{(4)}} \frac{\partial y_k^{(4)}}{\partial W_4}\right)_{(:,j)} = \left(\frac{\partial Loss^{sum}(\theta)}{\partial y_k^{(4)}} \cdot \left(y^{(3)}\right)^T\right)_{(:,j)}, \text{for} \left(y_k^{(4)}\right)_j \neq 0 \tag{20}$$

$$\begin{aligned}
&(\nabla b_4)_j = 0, \text{ for } \left(y_k^{(4)}\right)_j = 0;\\
&(\nabla b_4)_j = \left(\frac{\partial Loss^{sum}(\theta)}{\partial y_k^{(4)}} \frac{\partial y_k^{(4)}}{\partial b_4}\right)_j = \left(\frac{\partial Loss^{sum}(\theta)}{\partial y_k^{(4)}}\right)_j, \text{for } \left(y_k^{(4)}\right)_j \neq 0
\end{aligned} \tag{21}$$

By analogy, we can deduce the gradient descents of ∇W_3, ∇W_2, ∇b_3, ∇b_2 one by one according to the chain rule.

With the above network architecture and the gradient descents of all layers, we train the neural network of the DeepRec approach as shown in Fig. 5. We use the "Xavier" method [33] to initialize the weight matrix for each W_l $(2 \le l \le 5)$. That is, a uniform distribution as $W_l \sim U(-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}})$ is used to initialize W_l to ensure $Var(W_l) = \frac{2}{n_{in}+n_{out}}$, where n_{in} is the number of "fan-in" with respect to W_l and n_{out} is the number of "fan-out" with respect to W_l [6]. Each element in bias vectors b_l is initialized as a constant as 0.001. In the implementation, we use the method of stochastic gradient descent (SGD) [6] on the randomly sampled dataset $U^{(PE)}$ in order to reduce the computation complexity. At each iteration, we only update a small part of W_l and b_l rather than the full weight matrices and bias vectors. The parameter *maxIter* should be set to ensure that the loss function is convergent to a stable level (i.e., $Loss^{sum(i+1)} - Loss^{sum(i)} < \varepsilon$ and ε is small number near to zero). In our case, we set the parameter *maxIter* as 1000.

5. Experiments

This section presents the experiments and results to evaluate the performance of the proposed DeepRec approach in recommending Apps and music on benchmarking datasets. State-of-the-art techniques are also implemented as baselines to compare with the proposed DeepRec approach.

5.1. The datasets

Two datasets are used to evaluate the performance of the proposed DeepRec approach in recommendation. One dataset is a mobile App installation dataset provided by TalkingData which available in the Kaggle website.[2] We abbreviate it as App dataset hereafter. Originally, the App dataset contains 31,794,715 records about 202 users on rating 1802 mobile Apps within a week. To clarify, we refer "rating" an App here as that a user installs and continuously uses the App within the investigated week. Based our statistics on the dataset, 803 Apps were rated by only one user and 1,581 Apps were rated by

[2] TalkingData Mobile User Demographics. Online: https://www.kaggle.com/c/talkingdata-mobile-user-demographics.

Input: $\theta^{(0)}=\{W_2^{(0)},\ b_2^{(0)},\ W_3^{(0)},\ b_3^{(0)},W_4^{(0)},\ b_4^{(0)},W_5^{(0)},\ b_5^{(0)}\}$;

α=0.001 , the learning rate of in training process;

PE, the percentage of random sampling users from the whole dataset;

Q , the number of recommended items;

$maxIter$, the maximum number of iterations in the training process;

Output: $\theta^{(*)}=\{W_2^{(*)},\ b_2^{(*)},\ W_3^{(*)},\ b_3^{(*)},\ W_4^{(*)},\ b_4^{(*)},\ W_5^{(*)},\ b_5^{(*)}\}$;

Procedure:

For i from 1 to $maxIter$

Randomly sample PE percentage of users $U^{(PE)}$ from the whole dataset and Q items from each user's ranking history $I(u_k)$ by uniform distributions;

Use each of the Q items and item embedding to compose in-out pair ($vec(i_{in}^{u_k})$! $vec(i_{out}^{u_k})$);

Forward procedure:

1) Use Equations 6-9 to compute $y_k^{(5)}$, i.e. the predicted item within the interest of user u_k;
2) For all users, use Equation 10 to compute the loss function $Loss^{sum}$ at current iteration;

Backward procedure:

1) Using the chain rule as described in Equations 12-18 to compute the gradients of $\nabla\theta^{(i)}=\{\nabla W_2^{(i)},\ \nabla b_2^{(i)},\ \nabla W_3^{(i)},\ \nabla b_3^{(i)},\nabla W_4^{(i)},\ \nabla b_4^{(i)},\ \nabla W_5^{(i)},\ \nabla b_5^{(i)}\}$;
2) Update $\theta^{(i+1)}=\theta^{(i)}+\alpha\nabla\theta^{(i)}$

End for

$\theta^{(*)}=\theta^{(maxIter)}$

Fig. 5. The algorithm to train the proposed DeepRec network.

Table 1
The distribution of the 177 users on the 225 Apps preprocessed from the App dataset.

Range of App #	# of users
51 < #ofApps ≤ 70	10
41 < #ofApps ≤ 50	18
31 < #ofApps ≤ 40	20
21 < #ofApps ≤ 30	57
10 ≤ #ofApps ≤ 20	72

less than 10 users (i.e. 5 percentages of 202 users). Thus, we remove those Apps being rated less than 10 users and remain 221 Apps in the experiments. Further, we remove 4 users who rated more than 130 Apps (i.e. more than 57 percentages of all Apps in consideration) because it is unusual for a user to install more than 130 Apps in her or his mobile device. Also, we remove 21 users who have installed less than 10 Apps, i.e. 5 percentages of all Apps.

After the above preprocessing, we obtain 221 Apps with 177 users from the dataset for App recommendation experiments. The rating distribution of the 177 users on the 221 Apps is shown in Table 1. That is to say, the historical rating matrix $R_{|U|\times|I|}$ has the number of users $|U|$ as 177 and the number of items $|I|$ as 221. We can see that most users often use 21 to 30 Apps in their mobile devices from time to time. Fig. 6(a) shows the distribution of rated times of Apps. We can see that there is an obvious popularity difference of Apps. The most popular App is installed by 79 users while the least popular App is installed by only 10 users in the tailored App dataset. Fig. 6(b) shows the weights assigned to the Apps

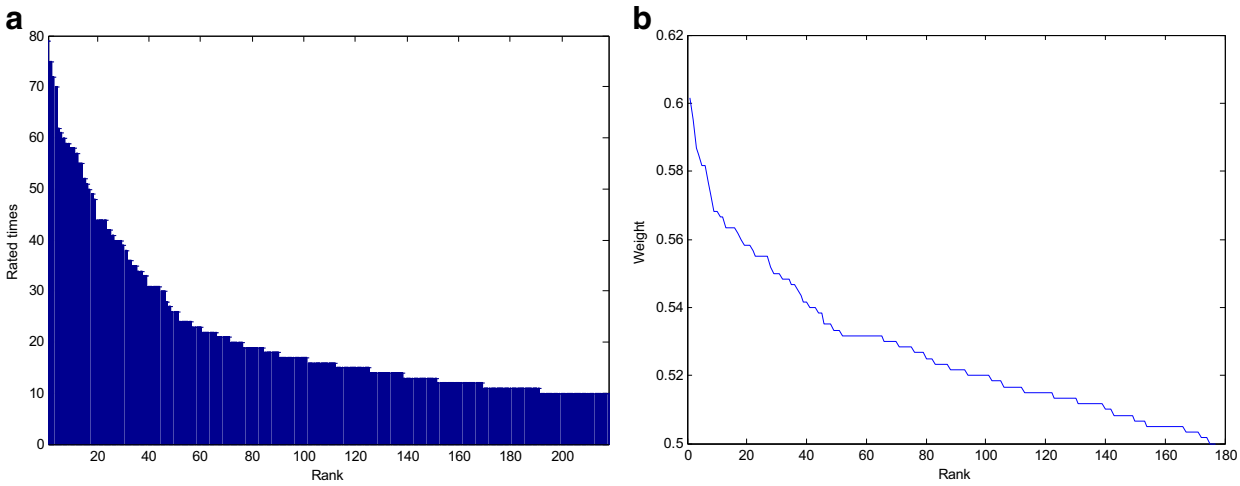

Fig. 6. The distribution of rated times on items (a) and the weights of items correspondingly (b) on the App dataset.

Table 2
The distribution of the 987 users on the 148,725 songs preprocessed from the Last.fm-1K dataset.

Range of Song #	# of users
$10,000 < \#ofSongs \leq 30,000$	70
$5000 < \#ofSongs \leq 10,000$	176
$1000 < \#ofSongs \leq 5000$	535
$100 \leq \#ofSongs \leq 1000$	206

correspondingly by using the linear interpolation function in Eq. (11). We can see that all the weights are squashed in the range from 0.5 to 0.6.

The other one is the Last.fm-1K dataset[3] which contains 19,150,868 records of 1000 users listening to 611,063 songs of 71,828 artists in the period from 7th January, 2008 to 4th May, 2009. We find that 17 users listened to less than 100 songs (on average 0.238 songs each day) thus we remove the 17 unusual users from the experiments. In addition, we also find that 462,336 songs have less than 10 users (i.e. less than 1 percentage of all users) thus we remove these songs from candidates. As a result, we obtain 983 users and 148,725 songs for music recommendation as listed in Table 2. That is to say, the historical rating matrix $R_{|U|\times|I|}$ in the Last.fm dataset has the number of users $|U|$ as 983 and the number of items $|I|$ as 148,725. Fig. 7(a) shows the distribution of rated times of Songs. We can see that there is also an obvious popularity differences among different songs. The most popular Song is listened to by 2,441 users while the least popular App is listened to by only 10 users in our tailored Last.fm dataset. Fig. 7(b) shows the weights assigned to the Songs correspondingly by using the linear interpolation function. We can see that all the weights are squashed in the range from 0.5 to 0.6.

5.2. Setting up

We use state-of-the-art recommendation techniques including three collaborative filtering methods and three deep learning methods as the baseline methods for comparison. The three collaborative filtering methods are the matrix factorization (MF) by pLSA method [16], the user-based collaborative filtering method (User-CF) [12], the item-based collaborative filtering method (Item-CF) [9]. The three deep learning methods for recommendation are RBM and NADE introduced in Section 3.2 and the deep belief network (DBN) [15]. Specifically, we construct the DBN by stacking RBMs on top of each other. That is to say, the hidden layer of the RBM at layer l becomes the input of the $l+1$ RBM layer. The first RBM layer has the input as the same as that of the DeepRec approach, i.e. $vec(i_{in}^{u_k})$. For recommendation, we treat the DBN network as a multi-layer perception and add a logistic regression layer on the top layer for multi-label classification problem. That is to say, the output of the DBN is also as the same as that of the DeepRec approach, i.e. $vec(i_{out}^{u_k})$. We adopt the DeepLearning4J open source platform[4] to implement the Skip-gram model for item embedding and build the DeepRec approach depicted in Fig. 3.

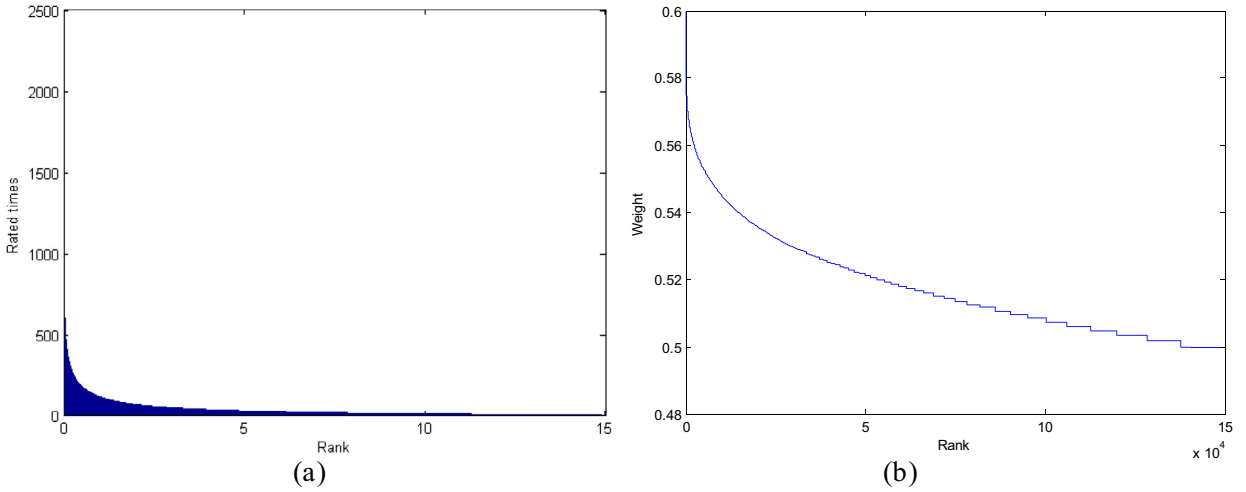

Fig. 7. The distribution of rated times on items (a) and the weights of items correspondingly (b) on the Last.fm dataset.

Table 3
The parameters of the compared methods in the experiments.

Method	Parameters and its description
MF by pLSA	c-# of hidden topics
User-CF	s_k^u-# of similar users of given user u_k
Item-CF	s_j^i-# of similar users of given item i_j
RBM	$\|H\|$-# of hidden units in the hidden layer
NADE	$\|H\|$-# of hidden units of each hidden layer
DBN	L-# of hidden layers
	$\|H_l\|$-# of hidden units of the lth hidden layer
DeepRec	e-# of features for item embedding in the input layer
	H_2-# of hidden units in the first hidden layer
	H_3-# of hidden units in the second hidden layer
	H_4-# of hidden units in the third hidden layer

The parameters of the methods mentioned in the paper are described in Table 3. For all the baseline methods, we tune the parameters by trial and error in the experiments. For MF by pLSA, we set the parameter as the number of hidden topics c as 100 for the App dataset and 300 for the Last.fm dataset. For User-CF, we set the parameter as the number of similar users of each user as 50 for the App dataset and 200 for the Last.fm dataset. For Item-CF, we set the parameter as the number of similar items of each item as 100 for the App dataset and 1,000 for the Last.fm dataset. For the RBM method, we set the number of hidden units in the hidden layer as 150 for the App dataset and 2000 for the Last.fm dataset. For the NADE method, we set the same parameters as that of the RBM method. For the DeepRec approach, we set H_2 as $4e$, H_3 as $3e$ and H_4 as $2e$ inspired by [8] in recommending Youtube videos and e is a parameter to be tuned in the experiments. We also find that this setting performs best among all the investigated architectures with different numbers of hidden layers and units. We evaluated different number of RBM layers for DBN training and found that when the number of RBM layers is set as 5 and, the number of hidden units in each layer is set as 150 for the App dataset and 2,000 for the Last.fm dataset, the adopted DBN achieves its best performance on recommendation. Actually, as pointed out by Srivastava et al. [29], it is not necessary the better to add more layers in a deep neural network for it will cause serious overfitting. This validates our observation that adding more layers and neuron units in both the proposed DeepRec approach and DBN cannot bring out an improvement in recommendation performance.

Actually, there are also other two parameters involved in the experiments for each recommendation method to examine its performance. The first parameter is the number of recommended items Q in consideration (see Section 2) and the second parameter is the percentage PE of all data samples used as the training data to train each of the mentioned recommendation algorithms.

Therefore, we set two scenarios in the experiments correspondingly. The first scenario is to set the percentage PE as 0.95. That is, we use 95 percentages of the dataset as training data and the remaining 5 percentages of the dataset as test data. For the App dataset, we vary the number of recommended items Q as 1, 2, 4, 6 and 8 one by one because the smallest

[3] Last.fm Dataset - 1K, online: http://www.dtic.upf.edu/~ocelma/MusicRecommendationDataset/lastfm-1K.html.
[4] Deeplearing4j: Open source, Distributed Deep Learning for the JVM: https://deeplearning4j.org/.

number of Apps of a user is 10 shown in Table 1. For the Last.fm-1 K dataset, we vary the number of recommended items Q as 10, 20, 50, 70 and 90 because the smallest number of movies of a user is 100 as shown in Table 2.

The second scenario is to predefine the number of recommended items Q and vary the training data percentage PE from 0.75 to 0.95 with 0.05 as the interval. For the App dataset, we set the recommended Q as 4 because the DeepRec approach has produced the best performance in the first scenario (see Section 5.3.2). For the Last.fm-1K dataset, we vary the number of recommended items Q as 20 for the same reason (see Section 5.3.3).

It should be noticed that under the Ran-Strategy in Section 4.3, in the training phase, each of the Q items of user u_k in the rating list of training data will be extracted randomly from his or her rating history to compose the output vector $vec(i_{out}^{u_k})$ and the remaining items will be used to compose the input vector $vec(i_{in}^{u_k})$. In the test phase, we firstly resample Q items randomly of user u_k as the test items. Then, we use each of the sampled Q items as the output item $i_{out}^{u_k}$ and the remaining items as $I(u_k)$-$i_{out}^{u_k}$ are used to compose the input vector. By analogy, under the Pro-Strategy in Section 4.3, the same steps are conducted in the experiments based on probabilities of items in $R_{|U|\times|I|}$.

For simplicity and fluency, hereafter in the paper, we use the acronym "Ran-Ave" to stands for the Ran-Strategy with average pooling method, the acronym "Ran-Max" to stands for the Ran-Strategy with max pooling method, the acronym "Pro-Ave" to stands for the Pro-Strategy with average pooling method and the acronym "Pro-Max" to stands for the Ran-Strategy with average pooling method.

We adopt MAP (Mean Average Precision) measure and use diversity measure to evaluate the performance of DeepRec and the baseline methods in recommendation. MAP provides a single-figure measure of information retrieval quality when a query may have multiple relevant objects [23]. In this paper, we treat items in the input vector of a user as a query and item in output vector of a user as the relevant object of the query. The $AvgP_k$ (average precision of a user u_k) is the average of the precision values obtained for the kth user, which is computed as Eq. (22), where j is the rank, Q is the number of recommended items, $pos(j)$ indicates whether or not the object of rank j is the correct item for u_k or not. $P(j)$ defined in Eq. (23) is the precision at the given cut-off rank j. Then the MAP value for a set of Q test data samples is the mean of the average precision for all samples in the test set. The higher the MAP value, the better the recommendation performance.

$$AvgP_k = \sum_{j=1}^{Q} \frac{P(j) \times pos(j)}{\#\,\text{of relevant items}} \tag{22}$$

$$P(j) = \frac{\#\text{ofrelevantitemsintop}\,j\text{positions}}{j} \tag{23}$$

If only accuracy was considered in recommendation, a recommendation algorithm would be prone to produce those popular items because they have higher probability to be rated by users than those unpopular items [10]. For this reason, we adopt the diversity measure to take into account the ability of a recommendation algorithm in recommending novel items as Eq. (24) where $sim(i, j)$ denotes the cosine similarity between item i and item j represented by $1 \times |U|$ vectors in the original user-item matrix $R_{|U|\times|I|}$. The higher the diversity value, the better the recommendation performance.

$$diversity_k = \frac{1}{Q(Q-1)} \sum_{i=1}^{Q} \sum_{i=1, i\neq j}^{Q} [1 - sim(i, j)] \tag{24}$$

5.3. Results

5.3.1. Tuning parameter e

The number of features e in item embedding of the proposed DeepRec approach is a decisive parameter to determine its performance in recommendation. If the parameter e is set small, it will cause items embedded in a condensed space resulting in great information loss in item representation in deep learning. Nevertheless, if the parameter e is set large, it will cause great sparseness of the learned feature space resulting in great computation without any performance increase. In this paper, we tune the parameter e by trial and error and the results are shown in Tables 4a and 4b on the App dataset and the Last.fm-1K dataset, respectively. For the App dataset, the parameter PE is set as 0.90 and the parameter Q is set as 4 for the App dataset. For the Last.fm-1K dataset, the parameter PE is set as 0.95 and the parameter Q is set for the Last.fm-1K dataset. The reason here is that we see from Sections 5.3.2 and 5.3.3 that under this these settings, the DeepRec approach have produced the best performance. We see that on the App dataset, the performance measured by MAP increase gradually until e is set as 100 and then keep stable. The performances measured by diversity decrease gradually until e is set as 100 and then also keep nearly stable. On the Last.fm-1K dataset, the performance measured by MAP increase gradually until e is set as 300 and then keep stable. The performances measured by diversity decrease gradually until e is set as 300 and then also keep nearly stable. For this observataion, in the following experiments, we set the number of features in item embedding e as 100 for the App dataset and 300 for the Last.fm-1K dataset.

5.3.2. Results of the first scenario

Table 5 shows the experimental results of the first scenario devised in Section 5.2. We can see that roughly, when PE is increasing, the accuracies measured by MAP of all the compared methods are also increasing steadily and the novelty

Table 4a
The performance of the proposed DeepRec approach on the App dataset measured by average MAP and diversity (in the first line) as well as its standard deviations (in the second line) when tuning the parameter e as the length of vectors of item embedding. The best performances are indicated in the bold type.

MAP (PE = 0.95, Q = 4)									
e	20	40	60	80	100	130	150	180	200
Ran-Ave	0.3034	0.3357	0.3764	0.4016	**0.4233**	0.4189	0.4204	0.4181	0.4105
	0.0873	0.0978	0.0980	0.0720	**0.0931**	0.0842	0.0750	0.0878	0.0949
Ran-Max	0.2717	0.2958	0.3464	0.3707	**0.3933**	0.3884	0.3932	0.3875	0.3785
	0.0975	0.0649	0.0976	0.1060	**0.1065**	0.1045	0.0803	0.1041	0.1042
Pro-Ave	0.3532	0.3827	0.4246	0.4497	**0.4823**	0.4695	0.4713	0.4650	0.4614
	0.1051	0.0722	0.0995	0.1057	**0.1037**	0.0991	0.0840	0.0878	0.0977
Pro-Max	0.3255	0.3524	0.4051	0.4311	**0.4506**	0.4469	0.4491	0.4445	0.4447
	0.0931	0.0822	0.0955	0.0812	**0.1030**	0.0918	0.0979	0.1042	0.0852
Diversity (PE = 0.95, Q = 4)									
e	20	40	60	80	100	130	150	180	200
Ran-Ave	**0.7005**	0.6936	0.6631	0.6517	0.6458	0.6339	0.6308	0.6328	0.6241
	0.1032	0.0948	0.0871	0.0902	0.1034	0.0838	0.0986	0.1050	0.1020
Ran-Max	**0.7605**	0.7539	0.7314	0.7125	0.6996	0.6851	0.6835	0.6857	0.6649
	0.0619	0.0988	0.0856	0.0863	0.0879	0.0947	0.1002	0.1047	0.1051
Pro-Ave	**0.6755**	0.6727	0.6528	0.6315	0.6163	0.6119	0.6011	0.6106	0.6034
	0.1026	0.1032	0.1044	0.1050	0.1044	0.0953	0.0882	0.0913	0.0903
Pro-Max	**0.7408**	0.7224	0.7043	0.6824	0.6629	0.6650	0.6419	0.6621	0.6516
	0.0906	0.0919	0.1040	0.1053	0.1042	0.0986	0.0893	0.0927	0.0818

Table 4b
The performance of the proposed DeepRec approach on the Last.fm-1K dataset measured by average MAP and diversity (in the first line) as well as its standard deviations (in the second line) when tuning the parameter e as the length of vectors of item embedding. The best performances are indicated in the bold type.

MAP (PE = 0.90, Q = 50)										
e	50	100	150	200	250	300	350	400	450	500
Ran-Ave	0.0634	0.0957	0.1264	0.1416	0.1466	**0.1533**	0.1489	0.1504	0.1481	0.1405
	0.0254	0.0403	0.0435	0.0429	0.0535	**0.0466**	0.0551	0.0560	0.0473	0.0517
Ran-Max	0.0517	0.0758	0.0964	0.1007	0.1147	**0.1213**	0.1214	0.1202	0.1175	0.1085
	0.0238	0.0363	0.0449	0.0417	0.0488	**0.0472**	0.0583	0.0543	0.0456	0.0515
Pro-Ave	0.0932	0.1127	0.1346	0.1597	0.1693	**0.1723**	0.1795	0.1713	0.1705	0.1714
	0.0453	0.0579	0.0440	0.0499	0.0588	**0.0664**	0.0691	0.0673	0.0662	0.0674
Pro-Max	0.0625	0.0924	0.1051	0.1111	0.1156	**0.1206**	0.1169	0.1181	0.1196	0.1103
	0.0331	0.0585	0.0465	0.0606	0.0630	**0.0554**	0.0568	0.0548	0.0645	0.0683
Diversity (PE = 0.90, Q = 50)										
e	50	100	150	200	250	300	350	400	450	500
Ran-Ave	**0.8305**	0.8136	0.7931	0.7617	0.7321	0.7458	0.7339	0.7308	0.7328	0.7241
	0.1432	0.1390	0.1256	0.1358	0.1483	0.1167	0.1487	0.1427	0.1576	0.1692
Ran-Max	**0.8505**	0.8439	0.8214	0.8125	0.8103	0.7896	0.7751	0.7735	0.7697	0.7649
	0.1464	0.1579	0.1428	0.1680	0.1842	0.1592	0.1428	0.1523	0.1521	0.1638
Pro-Ave	**0.7655**	0.7427	0.7228	0.7115	0.7013	0.7083	0.7019	0.7011	0.7006	0.7034
	0.1368	0.1552	0.1361	0.1496	0.1797	0.1349	0.1450	0.1435	0.1326	0.1352
Pro-Max	**0.7808**	0.7764	0.7643	0.7524	0.7429	0.7429	0.7165	0.7019	0.7121	0.7116
	0.1325	0.1418	0.1476	0.1639	0.1533	0.1484	0.1540	0.1417	0.1309	0.1459

measured by diversity of all the compared methods are decreasing gradually. For the App dataset, in Table 5a the Pro-Ave method outperforms other methods on accuracy and the methods derived from the DeepRec approach as Pro-Ave, Pro-Max, Ran-Ave and Ran-Max have produced better performance than state-of-the-art techniques. On novelty, the Ran-Max method outperforms other methods and the methods derived from the DeepRec approach have produced better performance than state-of-the-art methods. The similar experimental outcomes can be observed from the Last.fm-1K dataset in Table 5b. For both datasets, we see that with on both accuracy and novelty, the performance of all the methods become stable if PE goes with 0.90.

We explain the outcome that when more data are added to train the recommendation methods mentioned in the paper, their performance are improved because the relations between users, items and users to items are more elaborately characterized. For instance, the performance of the User-CF method depends on users' similarity measure in the historical data and the performance of the proposed DeepRec approach depends largely on items' relative positions in the embedding space. Nevertheless, the stable performance of all methods when PE is larger than 0.90 can be explained as that when enough data

Table 5a
The performance of the proposed DeepRec approach measured by average MAP and diversity (in the first line) as well as its standard deviations (in the second line) when tuning the training percentage PE in recommending movies in the App dataset. The best performances are indicated in the bold type.

MAP ($e = 100$, $Q = 4$)						
PE	0.70	0.75	0.80	0.85	0.90	0.95
User-CF	0.2002	0.2168	0.2241	0.2424	0.2342	0.2346
	0.0847	0.0589	0.0720	0.0493	0.0634	0.0759
Item-CF	0.2518	0.2582	0.2742	0.2901	0.2891	0.2785
	0.0785	0.0569	0.0755	0.0649	0.0576	0.0706
MF	0.2781	0.2898	0.2943	0.3001	0.2912	0.2941
	0.0572	0.0854	0.0830	0.0419	0.0428	0.0571
RBM	0.2848	0.2931	0.3012	0.3143	0.3320	0.3236
	0.0722	0.0595	0.0565	0.0357	0.0491	0.0439
NADE	0.2817	0.3098	0.3189	0.3281	0.3552	0.3348
	0.0749	0.0547	0.0672	0.0677	0.0655	0.0626
DBN	0.2832	0.3031	0.3112	0.3137	0.3425	0.3318
	0.0654	0.0431	0.0453	0.0632	0.0752	0.0813
Ran-Ave	0.3672	0.3813	0.3953	0.4198	0.4233	0.4181
	0.0678	0.0577	0.0473	0.0392	0.0655	0.0712
Ran-Max	0.3588	0.3610	0.3758	0.3829	0.3933	0.3851
	0.0706	0.0318	0.0629	0.0462	0.0719	0.0538
Pro-Ave	**0.4203**	**0.4289**	**0.4448**	**0.4632**	**0.4823**	**0.4624**
	0.0648	**0.0371**	**0.0529**	**0.0344**	**0.0438**	**0.0386**
Pro-Max	0.3852	0.4001	0.4087	0.4113	0.4406	0.4268
	0.0655	0.0527	0.0619	0.0488	0.0465	0.0634

Diversity ($e = 100$, $Q = 4$)						
PE	0.70	0.75	0.80	0.85	0.90	0.95
User-CF	0.5931	0.5885	0.5750	0.5717	0.5886	0.5857
	0.0910	0.0844	0.0597	0.0443	0.0853	0.0823
Item-CF	0.6054	0.5938	0.5868	0.5776	0.5854	0.5931
	0.0735	0.0552	0.0560	0.0619	0.0809	0.0593
MF	0.6207	0.6162	0.6094	0.5913	0.6069	0.6116
	0.0574	0.0638	0.0493	0.0575	0.0478	0.0543
RBM	0.6362	0.6263	0.6154	0.6019	0.5851	0.5948
	0.0843	0.0432	0.0392	0.0500	0.0669	0.0521
NADE	0.6408	0.6392	0.6283	0.6215	0.6035	0.6126
	00,610	0.0743	0.0517	0.0830	0.0853	0.0751
DBN	0.6523	0.6412	0.6154	0.6058	0.6036	0.6112
	0.1117	0.1058	0.1103	0.1002	0.0984	0.1034
Ran-Ave	0.6901	0.6874	0.6783	0.6619	0.6458	0.6548
	0.0727	0.0858	0.0572	0.0757	0.0804	0.0587
Ran-Max	**0.7423**	**0.7316**	**0.7235**	**0.7121**	**0.6896**	**0.7062**
	0.0759	**0.0540**	**0.0583**	**0.0729**	**0.0349**	**0.0594**
Pro-Ave	0.6682	0.6573	0.6419	0.6352	0.6163	0.6231
	0.0568	0.0469	0.0811	0.0713	0.0621	0.0734
Pro-Max	0.7228	0.7143	0.7032	0.6915	0.6829	0.6883
	0.0312	0.0528	0.0516	0.0602	0.0632	0.0541

are used to train the methods, more than enough data will cause overfitting to counteract the positive effects induced by adding more training data.

On the side of recommendation accuracy, it is interesting to see that at all PE values, the performance of the methods derived from the DeepRec approach are better than those of state-of-the-art techniques. We explain that the DeepRec approach can more accurately characterize a user's preference by deep learning the user's historically rated items. We may treat that traditional User-CF and Item-CF are used to learn the explicit relations of users and items. However, on the side of deep learning, it is used to learn the inherent relations among users and items although this kind of relation cannot be described explicitly by using mathematical formulas as that of User-CF and Item-CF. To be specific, the DeepRec approach adopts a 5-layer deep neural network to learn the inherent relations among items, i.e. the representation of the output vector $vec(i_{out}^{u_k})$ by neural combination of the input vectors $vec(i_{in}^{u_k})$. Moreover, we see that the Pro-Ave method performs better than the Pro-Max method and, the Ran-Ave method performs better than the Ran-Max method. This is also validated by Covington et al. [8] that the average pooling method is better than the max pooling method in producing input vectors for deep learning in recommendation.

On the side of recommendation novelty, we see that the recommended items by the DeepRec approach are more diverse than those by state-of-the-art techniques. It is not difficult to understand that the items produced by User-CF and Item-CF

Table 5b
The performance of the proposed DeepRec approach measured by average MAP and diversity (in the first line) as well as its standard deviations (in the second line) when tuning the training percentage PE in recommending songs in the Last.FM dataset. The best performances are indicated in the bold type.

MAP ($e = 300$, $Q = 20$)						
PE	0.70	0.75	0.80	0.85	0.90	0.95
User-CF	0.0191	0.0232	0.0301	0.0427	0.0404	0.0385
	0.0092	0.0875	0.0506	0.0186	0.0922	0.0139
Item-CF	0.0201	0.0267	0.0352	0.0421	0.0403	0.0383
	0.0124	0.0085	0.0083	0.0196	0.0078	0.0147
MF	0.0261	0.0351	0.0421	0.0531	0.0478	0.0476
	0.0107	0.0169	0.0146	0.0183	0.0173	0.0086
RBM	0.0361	0.0417	0.0532	0.0654	0.0801	0.0761
	0.0084	0.0098	0.0159	0.0181	0.0344	0.0196
NADE	0.0512	0.0591	0.0689	0.0761	0.0938	0.0842
	0.0188	0.0168	0.0245	0.0236	0.0398	0.0275
DBN	0.0352	0.0369	0.0436	0.0601	0.0738	0.0654
	0.0073	0.0086	0.0132	0.0201	0.0198	0.0151
Ran-Ave	0.1089	0.1132	0.1276	0.1325	0.1533	0.1462
	0.0499	0.0450	0.0503	0.0622	0.0351	0.0513
Ran-Max	0.0653	0.0724	0.0818	0.0967	0.1213	0.1138
	0.0184	0.0360	0.0299	0.0233	0.0391	0.0420
Pro-Ave	**0.1258**	**0.1317**	**0.1457**	**0.1523**	**0.1723**	**0.1612**
	0.0473	**0.0497**	**0.0297**	**0.0448**	**0.0490**	**0.0384**
Pro-Max	0.0731	0.0865	0.0984	0.1023	0.1206	0.1145
	0.0377	0.0209	0.0369	0.0411	0.0378	0.0398

Diversity ($e = 300$, $Q = 20$)						
PE	0.70	0.75	0.80	0.85	0.90	0.95
User-CF	0.7552	0.7451	0.7251	0.7127	0.6889	0.6914
	0.1417	0.1394	0.1569	0.1321	0.0421	0.0516
Item-CF	0.7489	0.7319	0.7278	0.7127	0.6918	0.7064
	0.0572	0.0598	0.0384	0.0353	0.0821	0.0514
MF	0.7368	0.7251	0.7163	0.7078	0.6881	0.6958
	0.0430	0.0609	0.0461	0.0713	0.0674	0.0459
RBM	0.71582	0.7132	0.7098	0.6923	0.6789	0.6801
	0.0752	0.0692	0.0447	0.0890	0.0686	0.0835
NADE	0.7425	0.7367	0.7241	0.7113	0.6989	0.7042
	0.0368	0.0526	0.0728	0.0811	0.0924	0.0757
DBN	0.7423	0.7368	0.7256	0.7111	0.6953	0.6986
	0.1356	0.1457	0.1368	0.1421	0.1542	0.1623
Ran-Ave	0.7843	0.7697	0.7542	0.7437	0.7389	0.7458
	0.0864	0.0549	0.0446	0.0633	0.0855	0.1085
Ran-Max	**0.8137**	**0.8032**	**0.7941**	**0.7913**	**0.7826**	**0.7896**
	0.0761	**0.0643**	**0.0763**	**0.0811**	**0.0583**	**0.0735**
Pro-Ave	0.7438	0.7325	0.7234	0.7171	0.7083	0.7084
	0.0930	0.0879	0.1552	0.0622	0.0587	0.0772
Pro-Max	0.7759	0.7689	0.7588	0.7468	0.7429	0.7489
	0.1230	0.0973	0.5032	0.0834	0.0849	0.0529

are similar to each other because the neighboring items for a given item are more prone to be similar to each other than non-neighboring items. Moreover, the popular items usually appears more often than those unpopular in neighboring item list for a given item as if they were similar to most items under consideration. For this reason, in the recommendation lists of state-of-the-art techniques, we find a large number of popular items which causes the recommended items with low novelty. However, in DeepRec approach, with the weighted loss function as described in Eq. (14), those unpopular items are also given approximately similar weights to those popular items. Although the weights of unpopular items are relatively smaller than those popular items as shown in Figs. 4 and 5, the number of unpopular items is much larger than the number of popular items and as a result, the sum of unpopular item weights is much larger than that of the popular item weights. Thus, those unpopular items have larger probabilities to be recommended in DeepRec approach than that in state-of-the-art techniques.

5.3.3. Results of the second scenario

Table 6 shows the experimental results of the second scenario devised in Section 5.2. We see from Tables 6a and 6b that on accuracy, all the methods perform better on the App dataset better than that on the Last.fm-1 K dataset while it is the opposite case on novelty. In Table 6a, we vary Q from 1 to 8 on the App dataset because most users have 10–20

Table 6a
The performance of the proposed DeepRec approach measured by average MAP and diversity (in the first line) as well as its standard deviations (in the second line) when tuning the number of recommended items Q in recommending movies in the App dataset. The best performances are indicated in the bold type.

MAP ($e = 100$, PE = 0.90)					
Q	2	3	4	6	8
User-CF	0.1998	0.2126	0.2346	0.2246	0.2184
	0.0707	0.0722	0.0535	0.1113	0.0932
Item-CF	0.2423	0.2651	0.2785	0.2669	0.2643
	0.0532	0.0848	0.0949	0.0797	0.0839
MF	0.2743	0.2808	0.2941	0.2813	0.2732
	0.0911	0.0852	0.0874	0.0949	0.0622
RBM	0.3117	0.3226	0.3325	0.3235	0.3171
	0.0628	0.0711	0.0721	0.0742	0.0967
NADE	0.3381	0.3409	0.3552	0.3412	0.3358
	0.0831	0.1029	0.0975	0.0855	0.0625
DBN	0.3231	0.3412	0.3437	0.3315	0.3243
	0.0654	0.0832	0.0913	0.0723	0.0817
Ran-Ave	0.4086	0.4137	0.4233	0.4102	0.4098
	0.0810	0.0922	0.0928	0.0733	0.0866
Ran-Max	0.3781	0.3892	0.3933	0.3887	0.3796
	0.0785	0.0723	0.0854	0.0639	0.0846
Pro-Ave	**0.4669**	**0.4781**	**0.4823**	**0.4689**	**0.4593**
	0.0521	**0.0613**	**0.0889**	**0.0624**	**0.0671**
Pro-Max	0.4288	0.4318	0.4406	0.4301	0.4275
	0.0955	0.0674	0.0773	0.0852	0.0913

Diversity ($e = 100$, PE = 0.90)					
Q	2	3	4	6	8
User-CF	0.5878	0.5712	0.5557	0.5836	0.5983
	0.0796	0.0987	0.0619	0.0534	0.0967
Item-CF	0.5943	0.5719	0.5631	0.5746	0.5862
	0.0631	0.0721	0.1068	0.0865	0.0492
MF	0.6013	0.5824	0.5716	0.5871	0.5979
	0.0971	0.0751	0.0907	0.0899	0.1024
RBM	0.6157	0.5948	0.5851	0.5989	0.6174
	0.0987	0.1078	0.0893	0.1047	0.1020
NADE	0.6221	0.6147	0.6035	0.6182	0.6228
	0.0799	0.0947	0.0969	0.1076	0.0859
DBN	0.6058	0.6013	0.5942	0.6018	0.6132
	0.0643	0.0798	0.0853	0.1123	0.1232
Ran-Ave	0.6721	0.6582	0.6458	0.6589	0.6793
	0.0855	0.0767	0.0829	0.1239	0.0886
Ran-Max	**0.7185**	**0.7049**	**0.6896**	**0.7126**	**0.7238**
	0.0872	**0.0948**	**0.1079**	**0.0978**	**0.1207**
Pro-Ave	0.6347	0.6272	0.6163	0.6257	0.6389
	0.1005	0.0956	0.0820	0.0724	0.1074
Pro-Max	0.7038	0.6983	0.6829	0.6912	0.7129
	0.0852	0.0967	0.0818	0.0781	0.0722

Apps installed in their smart phones shown in Table 1. In Table 6b, we vary Q from 10 to 90 on the Last.fm dataset because usually, a user cannot browse more than 100 songs at a time.

It is not difficult to understand that when the accuracy of recommendation is improved, the novelty of recommendation would be degraded because at this time, more similar items to the correct items of output vectors are recommended. Essentially, the similarities of items are defined based on their co-occurrences in users' rating lists. More accurate recommendation means more items in same lists are recommended and this would bring about decrease in novelty.

When the number of recommended items Q is set small, those correct items have a smaller probability to be ranked as top items as mentioned in Section 4.3. However, when the number of recommended items Q becomes large, those irrelevant and relevant items are recommended by all methods and this would bring about decrease on its accuracies. For this reason, we see from Table 6a that the accuracies of all methods attain up to its maxima when Q is set as 4 for the App dataset and from Table 6b that all the methods attain up to its maxima when Q is set as 20 for the Last.fm-1K dataset.

On accuracy, the Pro-Ave method has produced better performance than other method and on novelty, the Ran-Max method has produced better performance than other methods. On both accuracy and novelty, the methods derived from the DeepRec approach have produced better performance than state-of-the-art techniques. Moreover, the best diversity of

Table 6b
The performance of the proposed DeepRec approach measured by average MAP and diversity (in the first line) as well as its standard deviations (in the second line) when tuning the number of recommended items Q in recommending songs in the Last.FM dataset. The best performances are indicated in the bold type.

MAP ($e = 100$, PE = 0.90)						
Q	10	20	30	50	70	90
User-CF	0.0557	0.0685	0.0568	0.0561	0.0516	0.0498
	0.0144	0.0196	0.0156	0.0130	0.0149	0.0183
Item-CF	0.0557	0.0683	0.0512	0.0552	0.0446	0.0332
	0.0183	0.0204	0.0130	0.0183	0.0183	0.0132
MF	0.0687	0.0776	0.0702	0.0712	0.0696	0.0567
	0.0173	0.0239	0.0134	0.0308	0.0263	0.0132
RBM	0.0664	0.0801	0.0701	0.0671	0.0612	0.0623
	0.0293	0.0264	0.0219	0.0228	0.0173	0.0175
NADE	0.0846	0.0938	0.0736	0.0786	0.0712	0.0628
	0.0148	0.0271	0.0106	0.0270	0.0198	0.0153
DBN	0.0352	0.0369	0.0436	0.0601	0.0738	0.0654
	0.0073	0.0086	0.0132	0.0201	0.0198	0.0211
Ran-Ave	0.1402	0.1533	0.1432	0.1372	0.1214	0.1162
	0.0456	0.0502	0.0527	0.0379	0.0319	0.0371
Ran-Max	0.1125	0.1213	0.1097	0.1067	0.0891	0.0761
	0.0427	0.0513	0.0481	0.0282	0.0211	0.0139
Pro-Ave	**0.1621**	**0.1723**	**0.1611**	**0.1651**	**0.1602**	**0.1521**
	0.0364	**0.0579**	**0.0528**	**0.0465**	**0.0575**	**0.0649**
Pro-Max	0.1063	0.1206	0.1192	0.1182	0.1102	0.0893
	0.0276	0.0360	0.0275	0.0254	0.0308	0.0216

Diversity (e = 100, PE = 0.90)						
Q	10	20	30	50	70	90
User-CF	0.7182	0.7082	0.7071	0.6913	0.6923	0.6889
	0.1419	0.1824	0.2219	0.1557	0.1375	0.1498
Item-CF	0.7282	0.7132	0.7035	0.6978	0.6988	0.6918
	0.1493	0.1678	0.1577	0.1347	0.1239	0.1556
MF	0.7168	0.7123	0.7052	0.6954	0.6871	0.6861
	0.1412	0.1307	0.1380	0.1276	0.1462	0.1097
RBM	0.7145	0.7032	0.6992	0.6891	0.6889	0.6789
	0.1352	0.1468	0.1371	0.1402	0.1344	0.1387
NADE	0.7336	0.7278	0.7358	0.7331	0.7389	0.7319
	0.1368	0.1556	0.1529	0.1342	0.1348	0.1445
DBN	0.7223	0.7368	0.7256	0.7111	0.6953	0.6986
	0.1356	0.1457	0.1368	0.1421	0.1542	0.1623
Ran-Ave	0.7724	0.7558	0.7634	0.7746	0.7658	0.7668
	0.1463	0.1277	0.1323	0.1584	0.1352	0.1462
Ran-Max	**0.7965**	**0.7696**	**0.7734**	**0.7881**	**0.7896**	**0.7881**
	0.1407	**0.1321**	**0.1421**	**0.1632**	**0.1460**	**0.1541**
Pro-Ave	0.7342	0.7083	0.7124	0.7221	0.7283	0.7253
	0.1451	0.1435	0.1352	0.1724	0.1632	0.1623
Pro-Max	0.7561	0.7429	0.7548	0.7645	0.7529	0.7629
	0.1342	0.1421	0.1409	0.1652	0.1528	0.1452

the Ran-Max method has enlightened us that random sampling and max pooling can be used to ensure novelty in deep learning for recommendation. We use feedforward deep neural network to learn inherent relations among items to ensure that items co-occurring frequently can be "memorized" in the network to improve recommendation accuracy and, we use the weighted loss function to ensure that items which occur infrequently in users' rating history could be recommended to improve recommendation diversity.

5.3.4. Complexity of the deeprec approach

Most computation involved in the DeepRec approach is in network training to find the optimal parameter set $\theta* = \{W_2^*, b_2^*, W_3^*, b_3^*, W_4^*, b_4^*, W_5^*, b_5^*\}$. We can treat that the bias vectors as b_2,b_3,b_4 and b_5 are an added dimension in W_2,W_3, $W_4$and W_5 correspondingly. Thus, we can speculate from Fig. 5 that the computation complexity of the DeepRec approach is approximately as $O((e(H_2+1)+(H_2+1)(H_3+1)+(H_3+1)(H_4+1)+(H_4+1)e)\times T)$ where T is the number of epochs in training the deep neural network of the DeepRec approach. For the User-CF method, the computation is cost mostly on calculating the similarities of users and for each user, it is represented by an $1\times|I|$ vector. Thus, the computation of the User-CF method is approximately as $O(|I|^2|U|(|U|-1)/2)$. By analogy, the computation of the Item-CF method is approxi-

Table 7a
The time spent (in seconds) by each of the recommendation method investigated in the paper as well as its performance when tuning the training percentage PE. The first decimal in the parentheses indicates the average MAP and the second decimal in the parentheses indicates the average diversity.

Seconds @ different performance on the App dataset						
PE	0.70	0.75	0.80	0.85	0.90	0.95
User-CF	60	68	72	76	78	82
	(0.2002,0.5931)	(0.2168,0.5885)	(0.2241,0.5750)	(0.2424,0.5717)	(0.2342,0.5886)	(0.2346,0.5957)
Item-CF	52	54	57	59	62	65
	(0.2518,0.6054)	(0.2582,0.5938)	(0.2742,0.5868)	0.2901,0.5776)	(0.2891,0.5854)	(0.2785,0.5931)
MF	26	28	32	36	40	42
	(0.2781,0.6207)	(0.2898,0.6162)	(0.2943,0.6094)	(0.3001,0.5913)	(0.2912,0.6069)	(0.2941,0.6116)
RBM	36	41	45	48	53	56
	(0.2848,0.6362)	(0.2931,0.6263)	(0.3012,0.6154)	(0.3143,0.6019)	(0.3320,0.5851)	(0.3236,0.5948)
NADE	37	40	43	47	50	51
	(0.2817,0.6408)	(0.3098,0.6392)	(0.3189,0.6283)	(0.3281,0.6215)	(0.3552,0.6035)	0.3348,0.6126)
DBN	19	21	23	25	28	31
	(0.2832,0.6523)	(0.3013,0.6413)	(0.3031,0.6154)	(0.3137,0.6058)	(0.3425,0.0636)	(0.3618,0.6112)
Ran-Ave	**19**	**21**	**23**	**25**	**28**	**31**
	(0.3672,0.6901)	**(0.3813,0.6874)**	**(0.3953,0.6783)**	**(0.4198,0.6619)**	**(0.4233,0.6458)**	**(0.4181,0.6548)**
Seconds @ different performance on the Last.FM dataset						
PE	0.70	0.75	0.80	0.85	0.90	0.95
User-CF	182	189	201	210	230	234
	(0.0191,0.7552)	(0.0232,0.7451)	(0.0301,0.7251)	(0.0427,0.7127)	(0.0404,0.6889)	(0.0385,0.6914)
Item-CF	167	180	188	193	201	203
	(0.0201,0.7489)	(0.0267,0.7319)	(0.0352,0.7278)	(0.0421,0.7127)	(0.0403,0.6918)	(0.0383,0.7064)
MF	114	128	1410	153	147	163
	(0.0261,0.7328)	(0.0351,0.7251)	(0.0421,0.7163)	(0.0531,0.7078)	(0.0428,0.6881)	(0.4760,0.6958)
RBM	128	142	158	169	178	181
	(0.0361,0.7132)	(0.0417,0.7158)	(0.0532,0.7098)	(0.0654,0.6923)	(0.0801,0.6789)	(0.0761,0.6801)
NADE	137	145	166	175	184	188
	(0.0512,0.7425)	(0.0591,0.7367)	(0.0689,0.7241)	(0.0761,0.7113)	(0.0938,0.6989)	(0.0842,0.7042)
DBN	75	98	103	106	120	123
	(0.0352,0.7223)	(0.0369,0.7368)	(0.0436,0.7256)	(0.0601,0.7111)	(0.0738,0.6953)	(0.0654,0.6986)
Ran-Ave	**75**	**98**	**103**	**106**	**120**	**123**
	(0.1089,0.7843)	**(0.1132,0.7697)**	**(0.1276,0.7542)**	**(0.1325,0.7437)**	**(0.1533,0.7381)**	**(0.1462,0.7458)**

mately as $O(|U|^2|I|(|I|-1)/2)$. Usually, in a recommendation system, the condition $|U| \ll |I|$ holds and this is the reason that the Item-CF method has much less complexity than User-CF method [17]. For the MF method, its computation is cost on pLSA factorization of the rating matrix $R_{|U|\times|I|}$ thus its computation complexity is approximately as $O(k|U||I|)$ where k is the number of dimensions used to demote a user and an item [11].

For the RBM method, its computation is also cost on updating the weight matrix $W = (w_{i,j})_{(2\times|I|)\times|H|}$. Thus, its computation complexity is approximately as $O(K|U||I||H|T)$ where K is the number of scales for rating (i.e. K is 2 for binary data in the paper). For the NADE method, it is very similar to the RBM method essentially except weight sharing between input vectors. However, for each vector v_j, it has only one weight matrix $(W)_{j,.}$ shared to connect all the hidden nodes. We can treat its computation complexity as $O(2|U||I|T)$ in the paper. More of than not, we set the parameters as $e < |H| \ll |U| \ll |I|$ and the number of epochs T is of order of thousand to make learning to be convergent. For this reason, we draw that the computation complexity of the proposed DeepRec approach is much smaller than that of state-of-the-art methods. For the DBN method, we can speculate its computation complexity is the same as that of the DeepRec approach, i.e. $O((e(H_2+1)+(H_2+1)(H_3+1)+(H_3+1)(H_4+1)+(H_4+1)e)T)$.

Tables 7a and 7b show the time in seconds spent on each of the tasks of the first scenario and the second scenario, respectively. We conduct all the experiments on a PC with CPU speed as 3.60 GHZ and RAM size as 4 G. Notice that we only use the Ran-Ave method to investigate the time consumed by the DeepRec approach because the other derived methods consume approximately the same time as the Ran-Ave method. It can be seen that on all the recommendation tasks, the User-CF method takes the longest time to complete the computation and the proposed DeepRec approach needs the least time to complete the recommendation. The differences of the seconds spent by the RBM method and the NADE method are very small on all the devised tasks although the time spent by the NADE method is smaller than that spent by the RBM method. Among all the baseline methods, we see that the MF method by pLSA takes the shortest time to finish recommendation without considering the DBN method. In Section 5.2, we set k as 100 and 300 for the App dataset and the Last.fm-1K dataset respectively and as a result, its computation complexity is much smaller than other baseline methods. The DBN method cost the same time as that of the DeepRec approach as they have the same computation complexity.

Table 7b
The time spent (in seconds) by each of the recommendation method investigated in the paper as well as its performance tuning the number of recommended items Q. The first decimal in the parentheses indicates the average MAP and the second decimal in the parentheses indicates the average diversity.

Seconds @ different performance(on the App dataset)					
Q	2	3	4	6	8
User-CF	75 (0.1998,0.5878)	76 (0.2126,0.5712)	78 (0.2546,0.5557)	82 (0.2246,0.5836)	84 (0.2184,0.5983)
Item-CF	66 (0.2423,0.5943)	66 (0.2651,0.5719)	67 (0.2785,0.5631)	68 (0.2669,0.5746)	69 (0.2643,0.5862)
MF	38 (0.2743,0.6013)	39 (0.2808,0.5824)	40 (0.2941,0.5716)	41 (0.2813,0.5871)	43 (0.2732,0.5979)
RBM	51 (0.3117,0.6157)	52 (0.3226,0.5948)	53 (0.3325,0.5851)	55 (0.3235,0.5989)	57 (0.3171,0.6174)
NADE	47 (0.3381,0.6221)	48 (0.3409,0.6147)	50 (0.3552,0.6035)	53 (0.3412,0.6182)	54 (0.3358,0.6218)
DBN	29 (0.3213,0.6058)	30 (0.3412,0.6010)	31 (0.3431,0.5942)	33 (0.3315,0.6018)	34 (0.3243,0.6132)
Ran-Ave	**29 (0.4086,0.6058)**	**30 (0.4137,0.6582)**	**31 (0.4233,0.6458)**	**33 (0.4102,0.6589)**	**34 (0.4098,0.6693)**

Seconds @ different performance (on the Last.FM dataset)						
Q	10	20	30	50	70	90
User-CF	223 (0.0557,0.7182)	230 (0.0685,0.7082)	236 (0.0568,0.7071)	238 (0.0561,0.6913)	236 (0.516,0.6923)	248 (0.498,0.6889)
Item-CF	208 (0.0557,0.7282)	209 (0.0683,0.7132)	213 (0.0512,0.7035)	225 (0.0552,0.6978)	228 (0.0446,0.6988)	228 (0.0332,0.6918)
MF	143 (0.0687,0.7168)	147 (0.0776,0.7123)	158 (0.0702,0.7052)	168 (0.0712,0.6954)	158 (0.0696,0.6871)	174 (0.0567,0.6861)
RBM	188 (0.0664,0.7145)	188 (0.0861,0.7032)	191 (0.0701,0.6992)	195 (0.0671,0.6891)	196 (0.0612,0.6889)	198 (0.0628,0.6789)
NADE	169 (0.0846,0.7336)	184 (0.0938,0.7278)	183 (0.0736,0.7358)	180 (0.0786,0.7331)	181 (0.0712,0.7389)	184 (0.0628,0.7319)
DBN	113 (0.0751,0.7213)	120 (0.0812,0.7214)	128 (0.0712,0.7058)	133 (0.0685,0.7123)	142 (0.0652,0.7217)	147 (0.0613,0.7133)
Ran-Ave	**113 (0.1402,0.7724)**	**120 (0.1533,0.7558)**	**128 (0.1432,0.7634)**	**133 (0.1372,0.7746)**	**142 (0.1214,0.7658)**	**147 (0.1162,0.7668)**

6. Conclusion and future work

This paper proposed a novel approach called DeepRec to address the problem of data sparsity in collaborative filtering for recommendation on implicit feedback data. In DeepRec, each item is represented by a dense numeric vector and a feedforward deep neural network is proposed to predict users' preference on items. Moreover, a weighted loss function combined with linear interpolation function was employed to balance accuracy and novelty in recommendation. Average pooling and the max pooling are incorporated to aggregate a user's historical items to an input vector for the deep neural network. Ran-Strategy and Pro-Strategy are employed to sample items as the output vectors to train the deep neural network. Evaluation experimental were conducted using two dataset, i.e. the App and Last.fm datasets, and the results demonstrate that on accuracy, the Pro-Ave method derived from the DeepRec approach has produced the best performance and on novelty, the Ran-Max method derived from the DeepRec approach has produced the best performance. Moreover, the methods derived from the DeepRec approach have produced better performance than state-of-the-art techniques on both accuracy and novelty.

Future work consists of two directions. The first direction is to conduct more experiments with diverse datasets of implicit feedbacks to further validate the effectiveness of the DeepRec approach in recommendation. The second direction is to reformulate the problem in the paper as session-based recommendation and adopt recurrent neural network structure in the DeepRec approach to further improve its performance.

Acknowledgment

This research was supported in part by Natural Science Foundation of China under Grant Nos. 61379046, 71601023 and 61432001; the Innovation Fund Project of Xi'an Science and Technology Program (Special Series for Xi'an University No. 2016CXWL21).

References

[1] F. Aiolli, Convex auc optimization for top-n recommendation with implicit feedback, in: Proceedings of the 8th ACM Conference on Recommender Systems, 2014, pp. 293–296.
[2] J. Bobadilla, F. Ortega, A. Hernando, A. Gutiérrez, Recommender systems survey, Knowl. Based Syst. 46 (1) (2013) 109–132.

[3] R. Buettner, in: Predicting User Behavior in Electronic Markets Based On Personality-Mining in Large Online Social Networks: A Personality-Based Product Recommender Framework, Electronic Markets, Springer, Berlin Heidelberg, 2016, pp. 1–19.
[4] D. Ben-Shimon, A. Tsikinovsky, L. Rokach, A. Meisles, G. Shani, L. Naamani, in: Recommender System from Personal Social Networks, Advances in Intelligent Web Mastering, Springer, Berlin Heidelberg, 2007, pp. 47–55.
[5] O. Barkan, N. Koenigstein, Item2Vec: neural item embedding for collaborative filtering, in: Proceedings of 26th IEEE International Workshop on Machine Learning for Signal Processing, 2016, pp. 1–6.
[6] L. Bottou, Online Algorithms and Stochastic Approximations, Online Learning and Neural Networks, in: David Saad (Ed.), Cambridge University Press, Cambridge, UK, 1998.
[7] I. Cantador, A. Bellogín, P. Castells, A multilayer ontology-based hybrid recommendation model, AI Commun. 21 (2008) (2008) 203–210.
[8] P. Covington, J. Adams, E Sargin, Deep neural networks for YouTube recommendations, in: Proceedings of the 10th ACM Conference on Recommender Systems, 2016, pp. 191–198.
[9] M. Deshpande, G. Karypis, Item-based top-n recommendation algorithms, ACM Trans. Inf. Syst. 22 (1) (2004) 143–177.
[10] N. Hurley, M. Zhang, Novelty and diversity in top-n recommendation-analysis and evaluation, ACM Trans. Int. Technol. 10 (4) (2011) Article 14.
[11] T. Hofmann, Probabilistic latent semantic indexing, in: Proceedings of the 22nd Annual International ACM Conference on Research and Development in Information Retrieval, 1999, pp. 50–57.
[12] T. Ha, S. Lee, Item-network-based collaborative filtering: a personalized recommendation method based on a user's item network, Inf. Proces. Manag. 53 (5) (2017) 1171–1184.
[13] G. Hinton, et al., A practical guide to training restricted boltzmann machines, in: G. Montavon, et al. (Eds.), NN: Tricks of the Trade, 7700, 2nd ed., LNCS, 2012, pp. 599–619.
[14] B. Hidasi, A. Karatzoglou, L. Baltrunas, D. Tikk, Session-based recommendations with recurrent neural networks, in: Proceedings of the International Conference on Learning Representation, San Juan, Puerto Rico, 2016.
[15] G.E. Hinton, S. Osindero, Y.W. The, A fast learning algorithm for deep belief nets, Neural Comput. 18 (7) (2006) 1527–1554.
[16] Y. Koren, R. Bell, CH. Volinsky, Matrix factorization techniques for recommender systems, IEEE Comput. 42 (8) (2009) 42–49.
[17] J. Lu, D. Wu, M. Mao, W. Wang, G. Zhang, Recommender system application developments, Decis. Support Syst. 74 (C) (2015) 12–32.
[18] D.H. Lee, P. Brusilovsky, Improving personalized recommendations using community membership information, Inf. Proces. Manag. 53 (5) (2017) 1201–1214.
[19] H. Liu, Z. Hu, A. Mian, H. Tian, X Zhu, A new user similarity model to improve the accuracy of collaborative filtering, Knowl. Based Syst. 56 (3) (2014) 156–166.
[20] J. Liu, C. Wu, Deep learning based recommendation: a survey, in: K. Kim, N. Joukov (Eds.), Information Science and Applications 2017, Lecture Notes in Electrical Engineering, 424, 2017, pp. 451–458.
[21] S. Middleton, D. Roure, N. Shadbolt, Ontology-based recommender systems, in: S. Staab, R. Studer (Eds.), Handbook On Ontologies, Springer, Berlin Heidelberg, 2009, pp. 779–796.
[22] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, Proceedings of the International Conference on Learning Representations (ICLR 2013), May 2nd–May 4th Scottsdale, Arizona, USA, 2013.
[23] C.D. Manning, P. Raghavan, Introduction to Information Retrieval, in: H. Schütze (Ed.), Cambridge University Press, 2008.
[24] V. Nair, G.E. Hinton, Rectified linear units improve restricted boltzmann machines, in: Proceedings of the 27th International Conference on Machine Learning, 2010, pp. 807–814.
[25] M. Pazzani, D. Billsus, in: Content-based Recommendation Systems, The Adaptive Web, Springer, Berlin Heidelberg, 2007, pp. 325–341.
[26] Y. Park, A. Tuzhilin, The long tail of recommender systems and how to leverage it, in: Proceedings of the RecSys'08, 2008, pp. 11–18.
[27] R. Salakhutdinov, A. Mnih, G. Hinton, Restricted boltzmann machines for collaborative filtering, in: Proceedings of the 24th International Conference on Machine Learning, 2007, pp. 168–177.
[28] J.A. Snyman, Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms, Springer Publishing, 2009 February 22.
[29] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (2014) 1929–1958.
[30] H. Tong, Q. Wu, Learning performance of regularized moving least square regression, J. Computat. Appl. Math. 325 (1) (2017) 42–55.
[31] A. Tom, Mathematical Analysis, 2nd ed., Addison Wesley, 1974.
[32] K. Verstrepen, K. Bhaduri, B. Cule, B. Geothals, Collaborative filtering for binary, positiveonly data, ACM Sigkdd Explorat. Newslett. 19 (1) (2017) 1–21.
[33] G. Xavier, B. Yoshua, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS), 2010, pp. 249–256.
[34] H. Yin, B. Cui, J. Li, J. Yao, C. Chen, Challenging the long tail recommendation, Proc. VLDB Endow. 5 (9) (2012) 896–907.
[35] T. Zhou, J. Ren, M. Medo, Y. Zhang, Bipartite network projection and personal recommendation, Phys. Rev. E 76 (2) (2007) 046115.
[36] C.N. Ziegler, G. Lausen, in: Analyzing Correlation Between Trust and User Similarity in Online Communities, Trust Management, Springer, Berlin Heidelberg, 2004, pp. 251–265.
[37] Y. Zheng, B. Tang, W. Ding, H. Zhou, A neural autoregressive approach to collaborative filtering, in: Proceedings of the 33rd International Conference on Machine Learning, 2016, pp. 353–360.