A Top-N-Balanced Sequential Recommendation Based on Recurrent Network*

Zhenyu ZHAO††††, Ming ZHU†††, Nonmembers, Yiqiang SHENG††††††, Member, and Jinlin WANG†††,††††††, Nonmember

SUMMARY To solve the low accuracy problem of the recommender system for long term users, in this paper, we propose a top-N-balanced sequential recommendation based on recurrent neural network. We postulated and verified that the interactions between users and items is time-dependent in the long term, but in the short term, it is time-independent. We balance the top-N recommendation and sequential recommendation to generate a better recommender list by improving the loss function and generation method. The experimental results demonstrate the effectiveness of our method. Compared with a state-of-the-art recommender algorithm, our method clearly improves the performance of the recommendation on hit rate. Besides the improvement of the basic performance, our method can also handle the cold start problem and supply new users with the same quality of service as the old users.

key words: top-N recommendation, sequential recommendation, recurrent neural network, word embedding, item embedding, time dependent, cold start

1. Introduction

In the era of information overload, there are massive resources on the Internet and it is hard to find out something useful and valuable. Then the recommender system (RS) comes out and plays an important role in our life to solve the problem. The RS can learn what the user wants and recommend them to the user by describing the user’s preferences.

Robin distinguishes between six different classes of recommendation approaches [1]. The content-based recommendation and collaborative filtering (CF) are the most commonly used recommendation strategies. A content-based system learns to recommend items which are similar to the items that user liked in the past. This method is able to recommend to users with unique tastes and provide several explanations for the results. However, it requires content that can be encoded as meaningful features and users must be represented as a learnable function of content features, which brings lots of difficulty in data collection and data cleaning. A CF system considers the relationship between users, items and their historical interactions instead of explicit profiles. The major benefit of CF is that it is domain free. However, CF needs a reasonable amount of users and interactions to calculate the similarities between users and items. If there are few users and items in the system, it would be hard to find a match, and we called this as the cold start problem. Due to the computation complexity of nearest neighbor grows with both the number of users and items that the CF performs worse in large scale.

These days, with the widely used and great performed of the neural network in natural language processing, image processing and other fields, some researchers apply the neural network to recommender system. Most of the popular neural networks have been combined with the recommender system, such as restricted Boltzmann machine (RBM) [2], deep belief network (DBN) [3], convolutional neural network (CNN) [4], and so forth. These methods use the feature of users and items as the input of the neural network, then use the neural network to learn the interaction between users and items. There are mainly two types of the combination of neural network and recommender system. One is using neural network instead of CF to give recommendations directly. Another one is using the neural network to construct the data preprocessing module. In these systems, the neural network is used to extract user’s and item’s features, fulfill the user-item matrix [5], etc. Then the recommender system use CF to give recommendations.

The performance of these approaches are acceptable, but none of them consider the timing issues. User’s behavior might be different and affected by seasons or years. Many commercial goods are bought according to the season, for example, people hardly buy T-shirts in winter and down jacket in summer. User’s interests, preference, and personal needs might alter with the evolution of time. Most of the approaches ignore that generating recommendations is an inherently time-dependent task and still focus on achieving increasing accurate rating predictions [6]. Some researchers start to use RNN to address the sequential recommendation problem and session-based recommendation problem since 2015 [6]–[8]. By default, their recommendations are the se-
quency outputs of the network. However, their works are mostly based on sessions, which means that there is very little data in a single session and it stays for a short time. For long sequence output, the latest output is quite unreliable due to the cumulative error. In our daily life, most of the recommender systems are used to provide service for long-term users rather than anonymous session users. Users want to get better and better service while recommender systems have more and more information about user’s historical reactions.

Most of these approaches generate the recommendation list by scoring and ranking the items from the alternative item subset, which is produced by the leave-one-out method. This approach has been widely used in literature [9]. Since the original item set is very large in practical application, researchers usually select some available items from the original item set as an alternative subset. Usually, the alternative subset is generated by random sampling. Then they score and recommend items on the subset. Thus, the performance of recommender system is easily affected by the alternative subset.

To summarize the issues we mentioned above, we pay more attention to the recommendation for long-term users and find a better way to generate the recommender list. In this paper, we describe a top-N-balanced sequential recommender system based on recurrent neural network. We analysis the interactions between users and items and find out that the sequence relationship among the neighboring items is not obvious, but it has a high degree of similarity in user’s historical behavior. We input the item embedding vectors into the network and construct a loss function to balance the loss from the top-N output and sequence output for model training. We also improved the generation method of the recommender list according to the loss function.

The main contributions of this paper are summarized as follows:

1. We propose a recurrent neural network for top-N-balanced sequential recommendation. We modified the network and its loss function to balance the top-N recommendation and sequential recommendation for a better performance. The experiment results show that our approach outperform state-of-the-art recommender algorithms when predicting recommender list on several performance measurements, such as hit rate (HR), normalized discounted cumulative gain (HDCG), etc. Our approach also performs well on the cold start.

2. We postulated and verified that the interactions between users and items is time-dependent in the long term, but in the short term, it is time-independent. According to the conclusion, we also modified the generation method of the recommender list. By balancing the top-N output and sequence output, we generate a better recommender list following user’s short-term and long-term preference.

3. Each item and user in the system is abstract and anonymous. We only use the sequence information to train our model, other information are ignored which can protect the privacy of users and reduce the difficulty of data collection. One of the additional advantage of our system is the user can be represented by the state of network which can be stored in the local memory for privacy.

The rest of the paper is organized as follows: Section 2 reviews the related work. Section 3 describes the recommender system we proposed in detail. In Sect. 4 we evaluate our recommender system on several datasets. The experimental results show that our system achieves comparable performance to existing models. Finally, we conclude this paper in Sect. 5.

2. Related Work

He et al. summarize two critical issues on recommendation in [10]. The first one is that due to the large space of implicit feedback, most of the researchers use the uniform weight to modeling the missing data which is invalid in real-world settings. The second one is that most approaches are designed in an offline setting and fail to keep up with the dynamic nature of online data. In the paper, he proposed dedicated models to weight missing data based on item popularity.

The CF task with implicit feedback is usually viewed as an item recommendation problem. Compared with rating prediction, addressing the item recommendation problem is more practical but challenging [11]. And these methods need to predict the full user-item matrix with sparse original data, with a large amount of user and items the performance is unacceptable.

With the great performed of deep learning, some researchers attempt to use the neural network in a recommender system. Paulo Chiliguano et al. proposed a hybrid recommender system [12] based on convolutional deep neural networks which are used to represent an audio segment in an n-dimensional vector. In [3], Cui et al. combines the DBN and CF. They split the user-item matrix into 0-1 matrices and take them as the input of DBN model. Then they use the output as the user feature to calculate the similarities between users and make recommendations by using user-based CF. In [5], Wei et al. proposed a hybrid recommendation model to address the cold start problem. They explore the item content features learned from a deep learning neural network and applies them to the timeSVD++ CF model.

Only a few pure neural network approaches have yet been generally proposed in RS research, but most of them are focused on rating prediction and time irrelevant recommendations.

One of the first attempts is Hinton et al. [2]. He proposed a two-layer restricted Boltzmann machines (RBMs) to model user’s explicit ratings on items. He et al. [9] proposed to leverage a multi-layer perceptron to learn the user-item interaction function. They embed users and items into two feature vectors and combine them as the input of neural network to map the latent vectors to predict scores. In [13],
Oren Barkan et al. proposed item2vec for item-based CF to embed items in a latent space. But they discard the spatial and time information.

Methods that we mentioned above regard the action sequence as a set or basket of items and view all the historical items at the same level. However, user behavior might be different at different times and in different places. User preference might change and the same item have a different meaning to the user at different times. Some researchers have noticed that and do some research on sequential recommendation and session-based recommendation.

In [14], Yu et al. use the pooling operation on the items in a basket to get the representation of the basket, then input the representation to an RNN to generate the scores of this user towards all items. Donkers et al. in the paper [6] extend RNN approach to address the sequential recommendation problem by considering unique characteristics of the recommender system domain. Instead of using the constant input matrix and transition matrix in conventional RNN models, Context-Aware RNN (CARNN) [15] employs adaptive context specific input matrices and adaptive context specific transition matrices. The adaptive context specific input matrices capture external situations where user behaviors happen, given the behavioral history of the user input and transition contexts. Then predict the next selected item of the user.

There are also some other studies on session-based recom-}mendation and sequential recommendation [7], [8]. The paper [7] can be viewed as the basic attempts of using RNN in recommender system. They construct a basic RNN and use the item embedding vector as the inputs. Tan et al. in [8] proposed to use data augmentation and a method to account for shifts in the input data distribution based on RNN. Their models mostly based on the standard networks and the output of the network is one hot vector which is unacceptable in the real-world due to the large scale of users and items.

These works are based on sessions, which means that there is very little data in a single session and it stays for a short time. In our daily life, most of the recommender systems are used to provide service for long-term users. Users want to get better and better service while recommender systems have more and more information about user’s historical reactions. The RNN based approaches we mentioned above, generate the recommendations by sequence output, but the latest output is quite unreliable due to the cumulative error.

3. Proposed Recommender System

In this section, we introduce the recommender system we proposed.

3.1 Basic Architecture of the Recommender System

We construct our recommender system based on RNN as shown in Fig. 1. It is because we want to use the minimum information of items and users to give an recommendation, that we only use the ratings with item identity as the input of network.

From the Fig. 1, we can see that there is an embedding layer above the input layer. It is a fully connected layer that maps the one-hot encoding vectors $x_{ot} \in \mathbb{R}^N$ into the embedding vectors $x_k \in \mathbb{R}^n$. $N$ denotes the number of items and $n$ denotes the dimension of the embedding vector. In the encoding vector, the index of value ‘1’ corresponds to the index of item while others are ‘0’. Then we fed the embedding vector $x_k$ which describes the item $k$ into the network. The main body of the network is a recurrent neural network, and the output layer is a fully connected layer without activation function. When we get the output of the network, we can calculate the loss of the network and update the parameters by using gradient decent.

Then we formulate our recommender system as follows. Let $m$ denotes the dimension of hidden layers. The input of the network is $x_t \in \mathbb{R}^m$ which is the embedding vector of the input item at time step $t$. At time step $t$, the hidden state vector is $h_t \in \mathbb{R}^m$, $o_t$ and $y_i \in \mathbb{R}^n$ are the outputs of RNN and the system separately. We can compute the hidden state vectors and output as follows:

$$o_t = f(Wx_t + Uh_{t-1} + b_h)$$

(1)

$$y_i = V o_t + b_o$$

(2)

In the above equations, $W$, $U$, and $b_h$ are weights of the RNN, $V$ and $b_o$ are the weights of the output layer, $f$ is nonlinear activation function.

3.2 Loss Function

In a real recommender system, there usually has millions of items in the system which makes it difficult and inefficient to recommend items from the numerous items. Thus, in our system, we view the embedding vector of items as the target. The output of the network $y_i$ is the predicted embedding vector of the target item at time $t$. Then we use the output and target embedding vector to calculate the loss of the network.
There are two parts in the loss function, which are represented by the loss of top-N output and sequence output separately. For a given input \( x_t \), we can get several top-\( k \) outputs which are similar to the surrounding outputs in skip-gram.

First of all, we calculate the similarity between the output and the embedding vector as follows:

\[
Sim(y_t, y_i) = \cos_{\text{similarity}}(y_t, y_i) = \frac{y_t^T y_i}{\|y_t\|\|y_i\|}
\]  

In Eq. (3), \( y_i \) is the embedding vector of item \( i \). Then we can get the most similar items to the output \( y_t \) by comparing the similarities. The selected items are the top-\( k \) outputs \( \{y_1^t, \ldots, y_k^t\} \). Then we can define the probability of \( P(y_i|x_{t-1}) \) as follows:

\[
P(y_i|x_{t-1}) = \frac{Sim(\hat{y}_t, y_i) + 1}{\sum_{i\in S} (Sim(\hat{y}_t, y_i) + 1)}
\]  

In the above equation, \( S \) is the set of item, \( y_i \) is the embedding vector of item \( i \), \( \hat{y}_t \) is the embedding vector of target item at time step \( t \). It is because the value of \( Sim \) is from \(-1\) to \(1\), that we plus \( 1 \) on \( Sim \) when we calculate the probability \( P(y_i|x_{t-1}) \).

After we get the top-\( k \) outputs and sequence outputs, we can calculate the loss function. For input \( x_t \) and top-\( k \) outputs \( \{y_1^t, \ldots, y_k^t\} \), we aims to maximizing the following term:

\[
\sum_{j=0}^{k} \alpha_j P(y_j^t|x_t)
\]  

In the Eq. (5), \( k \) is the top-N window size, \( \alpha_j \) is the user defined scaling factor. This equation represents the prediction loss between top-N outputs and the surrounding items of the target.

Based on the RNN, we can also get the sequential outputs. We feed the output of last time step \( y_t \) into the network and get output \( y_{t+1} \). Feeding outputs into the network and we can get the sequence outputs \( \{y_{t+1}, y_{t+2}, \ldots, y_{t+c}\} \), in which \( c \) is the size of sequence window. For \( y_j^t \in \{y_1^t, \ldots, y_k^t\} \) and its sequence outputs \( \{y_{t+1}^j, y_{t+2}^j, \ldots, y_{t+c}^j\} \), we aims to maximizing the following term:

\[
\sum_{j=0}^{c} \beta_j P(y_{t+1}^j|y_{t+c}^j)
\]  

In the above Eq. (6), \( c \) is the sequence window size, \( \beta_j \) is the user defined scaling factor. This equation represents the prediction loss of the sequence output after \( y_t \).

Then, for a given sequence of items with length \( T \), the objective can be rewritten as

\[
\frac{1}{T} \sum_{t=1}^{T} \sum_{j=0}^{k} \alpha_j P(y_{t+j}|x_t) + \sum_{i=0}^{T} \frac{1}{k} \sum_{j=0}^{k} \beta_j P(y_{t+1+j}|y_{t+c+i})
\]  

3.3 Training Process

Figure 2 gives an example of the training process. For a given user sequence, for example \( I = \{I_1, \ldots, I_6\} \), we input the items into the network one by one. When we feed the item \( I_t \) into the network, we view the following items \( \{I_{t+1}, \ldots, I_{t+k}\} \) as the target and calculate the loss function as we mentioned above.

In training procedure, in order to improve the performance of the system in the test procedure, the input of the network is randomly chosen from the original input and the output of last time step, for example, the input of the network can be ‘O2’ or ‘I3’ at time step 3 in Fig. 2.

And then we can use backpropagation and gradient descent to update the parameters and train the model.

3.4 Generate the Recommender List

Figure 3 gives an example of how to generate a recommendation list for a user in our system. There are some historical records of the user which are painted in gray and marked with \( I \). We input these items into the network one by one to update the state of the user until the latest time step. In the latest time step, top-\( k \) outputs are chosen as the recommended items which are painted in orange and marked with \( R \).

In the next time step, these items are also fed into the system separately. We iterate the procedure till the number...
of items meets the size of the recommendation list. Usually, there would be more items than we need in the recommendation list. We sort these items by their probabilities and choose the top-N items into the recommendation list.

4. Experiment

4.1 Datasets and Data Preprocessing

In this paper, we concentrate on the long term recommendation, which means that the historical records span from weeks to months. We experimented our model on public datasets, MovieLens1. MovieLens has been widely used to evaluate the performance of recommender system. We use two different versions of MovieLens datasets, one is the latest small and the other is 1M version. Both versions of the dataset have at least 20 ratings for each user and most of the user ratings are span for months. The characteristics of the two datasets are summarized in Table 1.

In order to simulate the cold start, when we divide the dataset into training set and test set, we use two divide schemes to generate the subsets.

• Based on record. The records of each user are divided into two parts, one part saved as the training set and the other part saved as a test set. The training set has 80% of user’s data and the rest are in the test set by default.
• Based on the user. We view the records of each user as an integer and divide the dataset by users’ index. The training set has 80% of users and their rating records, the rest of users are in the test set. This division scheme is similar to the division scheme of session-based datasets.

In MovieLens dataset, the ratings are explicit feedback data. It is because that we want to use the minimum information, we transformed the rating records into implicit data. The implicit data is marked as 0 or 1 to indicate whether the user has rated the item or not. We ignore the feature information or genres about users and items, only use the rating records to train our model. We evaluate the performance of recommendation on the full item set instead of a subset generated by the leave-one-out evaluation. In order to research the sequence features, we also sort the ratings of each user in MovieLens dataset by timestamp.

4.2 Metrics

In our experiment, we do not use precision as the performance measurement, because the rating information is in the form of implicit feedback. Specifically, a zero entry may cause that the user is not interested in the item, or that the user is not aware of its existence. So in this situation, precision is not a suitable performance measurement. Like most of the recommender systems, we generate the top N items and recommend them to the target user.

Then we use the Hit Rate (HR) and Normalized Discounted Cumulative Gain (NDCG) to evaluate our system. The HR is used to reflect the proportion of users for which at least one item from the user’s test profile is recommended. And the definition of HR is:

\[
\text{HitRate} = \begin{cases} 
1, & \text{if hit in the top-N list,} \\
0, & \text{else}
\end{cases}
\]  

(8)

In our experiment results, we give the value of HR as an average through users. Besides considering whether the test item is present on the top-N list, the NDCG can reflect the position of the hit by assigning higher scores to hits at top ranks. The definition of DCG and NDCG are as follows:

\[
DCG_n = \sum_{j=1}^{n} 2^{r(j) - 1} \frac{1}{\log_2(1 + j)}
\]

(9)

\[
NDCG_n = \frac{DCG_n}{\text{IDCG}_n}
\]

(10)

In the Eq. (9) and Eq. (10), n is the size of recommender list, r(j) is the relevance rating of the item at position j. The NDCG is the normalization of Discounted Cumulative Gain (DCG) by dividing the maximum possible DCG through position, also called Ideal DCG (IDCG). The NDCG values for all queries can be averaged to obtain a measure of the average performance.

4.3 Baselines

We compared our proposed the performance of recommender system with the following methods:

• Item-based kNN and user-based kNN. In the experiment, we use a basic implementation of the approach. Item-based kNN is the standard item-based collaborative filtering method. We use this method as the baseline.
• NCF method[9]. It is the new neural architecture for collaborative filtering proposed in 2017 by He. In their paper, they compared the performance of NCF with ItemPop, BPR, and eALS. NCF performs better than others. It is a highly competitive baseline for item recommendation. So we did not reproduce another recommendation methods, and we use the best performance in their paper as our baseline.

4.4 Experiment

First of all, we did some calculation and summarize on different datasets to verify the assumption we proposed, that in
long term the item sequence is time dependent and in short
term it is time independent. In fact, it is hard to qualify the
sequence relationship directly. So, we use the similarity be-
tween items to qualify the relationship in sequence instead.

We calculate the item similarity on the MovieLens
dataset firstly. Thus, for any item $i$, we can get the sim-
ilarity between itself and others. Secondly, we divide the
rating records of user into slices by time interval. Then, we
can calculate the similarity between slices.

In order to verify wether the item sequence is time in-
dependent in short term, for each user, we calculate the sim-
ilarity between $I_t$ and $I_{t+r}$ as follows:

$$S_{\text{short}}^r = \text{Sim}(I_t, I_{t+r})$$ (11)

In the Eq. (11), $r$ is the size of the short term time inter-
val, $I_t$ is the item at time step $t$, $\text{Sim}(I_t, I_{t+r})$ is the cosine sim-
ilarity between item $I_t$ and $I_{t+r}$ as we mentioned above. In
this part, the item similarity is calculated from the user-item
matrix as we usually do in item-based kNN. $S_{\text{short}}^r$ can reflect
the relationship between $I_t$ and its following items, which
viewed as short term items. Figure 4 shows the changes of
$S_{\text{short}}^r$ over time with different size of the short term time
interval.

From the Fig. 4, we can see that the value of average simi-
arity $S_{\text{short}}^r$ with different $r$ is similar. Due to sparsity of
data, the curves on the right side of the graph are not exactly the same and fluctuate. This means that in short
term, for item at time step $t$, the relationship between it and
the subsequent items is approximately the same.

We also calculate the similarity between different slices
to research the relationship between items in long term. The similarity between the first slice $S_{\text{init}} \in \{I_0, \ldots, I^n\}$ and the following slices $S_j \in \{I_j, \ldots, I^n\}$ is calculated by the following
formula:

$$S_{\text{long}}^j = \frac{1}{a} \sum_{v \in S_{\text{init}}} \frac{1}{b} \sum_{v' \in S_j} \text{Sim}(I^v, I'^v)$$ (12)

In the Eq. (12), $a$ is the size of $S_{\text{init}}$ and $b$ is the size of
$S_j$. $S_{\text{long}}^j$ is the average similarity between $S_{\text{init}}$ and $S_j$. It
can reflect the relationship between items in long term with
the increase of time gap $j$. Figure 5 shows the changes of
$S_{\text{long}}^j$ over time.

From the Fig. 5, we can see that the value of $S_{\text{long}}^j$ is
inversely proportional to time gap, which means that in long
term, the similarity is affected by time and the relationship
between items is time-dependent.

After verified our assumption, we test the performance
of our recommender system and compare our method with
several recommendation methods. In this part, the experi-
ments tested on the record-based datasets by default. In or-
der to get the best performance of these methods, we use the
explicit ratings and record-based dataset to test these meth-
ods.

Figure 6 and Fig. 7 show the performance of top-N rec-
ommendation lists on the MovieLens-1M dataset where the
size of list ranges from 1 to 10. Figure 6 shows the qual-
ity of ranking on the MovieLens-1M dataset with di-
ferent methods. From the figure, we can see that our method out-
performs than NCF with different size of top-N. When $N$
is small, the NDCG of our method increase rapidly with $N$,
and then remain stable. At top-10 item recommendation, the
NDCG of our method is 15% higher than NCF which is one
of the state-of-art method.

And Fig. 7 shows the performance of hit rate on the
dataset with different methods. We can see that the perfor-
mance of our proposed method is better than others with
different size of recommendation list. Compared with the baseline methods, the hit rate of our method at top-10 outperforms NCF with about 6% improvement.

Item-based kNN performs better than user-based kNN, which indicates the necessity of modeling the user’s personalized preferences and the effectiveness of using recurrent neural network and state to model the recommender system.

We also did some test on cold start problem. Like Wang et al. used in [13], we select $P$ earliest records of each user to form the training set and use all the rest of the dataset as the test set. In order to simulate the cold start and experienced user, we set $P$ from 1 to 20 in our experiments.

Figure 8 shows the top-10 recommendation results of our recommender system using the record-based MovieLens-1M dataset under different sparsity settings. In the very sparse setting, there are few data for network training, thus, item-based kNN performs better than our method. This is because that we do not have enough data to learn the relationship between items. As we can see, with the increase of training records, our method is significantly better than item-based kNN by a margin of 20%. And with the increase in the number of training $P$, the performance of our method is going stable. This verified that our method can follow the user’s short-term preference and provide the personalized recommendations.

| Table 2 | Train and test on the user-based MovieLens-1M subset for cold start |
|---------|-------------------------------------------------------------|
|          | User-based dataset | Item-kNN | NCF |
| Input size of test users | |
| NDCG     | 10 | 10 | 10 | 10 |
| TOP-1 | 0.18 | 0.21 | 0.18 | 0.21 | 0.22 |
| TOP-5 | 0.36 | 0.33 | 0.35 | 0.34 | 0.41 |
| TOP-10 | 0.37 | 0.36 | 0.37 | 0.37 | 0.45 |
| Hit Rate | 10 | 10 | 10 | 10 |
| TOP-1 | 0.18 | 0.21 | 0.18 | 0.21 | 0.22 |
| TOP-5 | 0.49 | 0.45 | 0.46 | 0.47 | 0.56 |
| TOP-10 | 0.68 | 0.63 | 0.61 | 0.59 | 0.72 |

We also test our approach on user-based datasets to simulate the cold start problem. In this part, we do not compare the performance of our approach and CF, MF, etc. Because those methods could not handle the new user cold start problem. We compare the performance of our method on new user cold start with normal settings.

Table 2 shows the experiment results. We estimate our method on the new user with different number of historical items. With the increase of the number of historical items, the performance is steady, which means the new user of our system would get the same good service as old users.

We also test our method on the MovieLens-100K dataset. Figure 9 shows the performance of NDCG and HR with the size of recommender list ranges from 1 to 10. From the figure, we can know that our system has similar performance on different datasets.

5. Conclusion and Future Work

In this work, we have proposed a top-N-balanced sequence recommender system based on recurrent neural network. We analysis the interactions between users and items and find out that the interactions are time-dependent in the long term, and are time-independent in the short term. We balance the top-N recommendation and sequence recommendation to generate a better recommender list by improving the loss function and generation method. We also improve the performance of recommendation by predicting recommender list on full item set. We test our method on two ver-
sions of MovieLens dataset and the experiments show that our method outperforms the state-of-the-art recommender algorithm. For high sparsity data, our system can learn the preference of users quickly by a few records. By predicting the embedding vector of target items, the network can handle the cold start problem and provide new users with the same quality of service as old users.

In the future, we will study more on extending our method to a distributed network [16] and training personalized recommender system with small data. And we will further explore generative adversarial networks for the application of item recommendation.

References

[1] B. Robin, “Hybrid web recommender systems,” The Adaptive Web: Methods and Strategies of Web Personalization, Springer, vol. Lecture Notes in Computer Science, pp.377–408, 2007.
[2] R. Salakhutdinov, A. Mnih, and G. Hinton, “Restricted boltzmann machines for collaborative filtering,” Proc. 24th International Conference on Machine Learning, pp.791–798, 2007.
[3] H.-L. Cui and X.-N. Qin “The video recommendation system based on DBN,” 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing, pp.1016–1021, 2015.
[4] T.X. Tuan and T.M. Phuong, “3D convolutional networks for session-based recommendation with content features,” Proc. Eleventh ACM Conference on Recommender Systems - RecSys ’17, pp.138–146, 2017.
[5] J. Wei, J. He, K. Chen, Y. Zhou, and Z. Tang, “Collaborative filtering and deep learning based hybrid recommendation for cold start problem,” Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress, pp.874–877, 2016.
[6] T. Donkers, B. Loepp, and J. Ziegler, “Sequential user-based recurrent neural network recommendations,” Proc. Eleventh ACM Conference on Recommender Systems, pp.152–160, 2017.
[7] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, “Session-based recommendations with recurrent neural networks,” CoRR, vol.abs/1511.06939, 2015.
[8] Y.K. Tan, X. Xu, and Y. Liu, “Improved recurrent neural networks for session-based recommendations,” Proc. 1st Workshop on Deep Learning for Recommender Systems - DLRS 2016, pp.17–22, 2016.
[9] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, “Neural collaborative filtering,” Proc. 26th International Conference on World Wide Web - WWW ’17, pp.173–182, 2017.
[10] X. He, H. Zhang, M.-Y. Kan, and T.-S. Chua, “Fast matrix factorization for online recommendation with implicit feedback,” Proc. 39th International ACM SIGIR Conference on Research & Development in Information Retrieval, pp.549–558, 2016.
[11] F. Cremonesi, Y. Koren, and R. Turrin, “Performance of recommender algorithms on top-n recommendation tasks,” Proc. Fourth ACM Conference on Recommender Systems, pp.39–46, 2010.
[12] P. Chilimigiano and G. Fazeekas, “Hybrid music recommender using content-based and social information,” 2016 IEEE International Conference on Acoustics, Speech and Signal Processing, pp.2618–2622, 2016.
[13] O. Barkan and N. Koenigstein, “ITEM2VEC: Neural item embedding for collaborative filtering,” 2016 IEEE International Workshop on Machine Learning for Signal Processing, pp.1–6, 2016.
[14] F. Yu, Q. Liu, S. Wu, L. Wang, and T. Tan, “A dynamic recurrent model for next basket recommendation,” Proc. 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp.729–732, 2016.
[15] Q. Liu, S. Wu, D. Wang, Z. Li, and L. Wang, “Context-aware sequential recommendation,” 2016 IEEE 16th International Conference on Data Mining (ICDM), pp.1053–1058, 2016.
[16] Y.-Q. Sheng, Z.-Z. Zhao, and Y. Liao, “Coarse-grained distributed deep learning for personal data mining,” Journal of Network New Media, vol.5, no.6, pp.1–6, 2016.