IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received July 1, 2019, accepted August 22, 2019, date of publication September 12, 2019, date of current version September 25, 2019.

*Digital Object Identifier 10.1109/ACCESS.2019.2941083*

# Recurrent Translation-Based Network for Top-N Sparse Sequential Recommendation

**NUTTAPONG CHAIRATANAKUL[1], TSUYOSHI MURATA[1], AND XIN LIU[2]**
[1]Department of Computer Science, School of Computing, Tokyo Institute of Technology, Tokyo 152-8550, Japan
[2]National Institute of Advanced Industrial Science and Technology, Tokyo 135-0064, Japan

Corresponding author: Nuttapong Chairatanakul (nuttapong.c@net.c.titech.ac.jp)

This work was supported in part by the Japan Science and Technology Agency (JST) Core Research for Evolutional Science and Technology (CREST) under Grant JPMJCR1687, in part by the Japan Society for the Promotion of Science (JSPS) Grant-in-Aid for Scientific Research (B) under Grant 17H01785, in part by the JSPS Grant-in-Aid for Early-Career Scientists under Grant 19K20352, and in part by the New Energy and Industrial Technology Development Organization (NEDO).

**ABSTRACT** Fulfilling users' needs and increasing the retention rate of recommendation systems are challenging. Most users have consumed a few items in most systems. Translation-based model performs well on sparse datasets. However, a user and only single previous item are considered for the user suggestion of next items. Alternatively, recurrent neural network utilizes sequential dependency but performs poorly on sparse datasets. We unify both and propose Recurrent Translation-based Network (RTN). RTN utilizes sequences of users' consumed items without limiting interactions between items to the most recent one. The results of conducting experiments on real-world datasets show that RTN outperforms other state-of-the-art approaches on sparse datasets.

**INDEX TERMS** Recommender system, collaborative filtering, recurrent neural network.

## I. INTRODUCTION

Nowadays, since a massive amount of data is available on the Internet, retailers and providers highly compete with each other to deliver the best contents to fit each user's preference. Recommendation systems [1] are built for that purpose. The idea is to build a model from past interactions between users and items and use it for predicting future interactions. Top-N recommendation restricts suggesting only to their top matched items. It is seen almost everywhere in online shopping websites, online video streaming applications, or even blogging sites. Better at recommending items for users will increase their satisfaction and encourage them to use the system more. In many real-world cases, most users have a few purchased items, watched movies, or read articles (interactions) in a system. Consequently, building a model for sparse datasets is important.

With long sequential user behavior data in dense datasets, recurrent neural networks (RNN)-based models [2] significantly outperform traditional Matrix Factorization (MF)-based models [3], [4]. RNN has its advantage over MF for modeling and capturing sequential patterns. It has been applied for other recommendation tasks, e.g., explicit rating prediction [5], session-based recommendation [6], [7] and next basket recommendation [8]. However, it is difficult to train and sometimes do not generalize well when there is an only small amount of data. Therefore, MF-based models are preferable for sparse datasets.

On sparse datasets, a traditional approach is to build a model based on two-way interactions embedded in an inner product space. Matrix Factorization models an interaction between a user and an item. First-order Markov chain aims to represent transitions between pairs of adjacent items (a *previous item* and a *next item*) in a sequence. *Factorized Personalized Markov Chains* (FPMC) [9] incorporates both of them by the summation of the pairwise interactions from each of them. Recently, *Factorized Sequential Prediction with Item Similarity Models* (Fossil) [10] extends FPMC to capture long-term dependencies between a user and all previous items in the user's sequence by combining similarity models [11] with high-order Markov chain. Achieving better results on sparse datasets of Fossil indicates that *higher-order* interactions need further investigation and are essential for predicting the future behavior of a user.

Inspired by knowledge graph embedding [12], Translation-based Recommendation (TransRec) [13] represents users as *translation vectors* moving their previous items' vectors to be close to their next items' vectors in the same embedding space. The model exploits the distance as its scoring scheme bringing those vectors to be closer. For this reason, TransRec

The associate editor coordinating the review of this manuscript and approving it for publication was Madhusanka Liyanage.

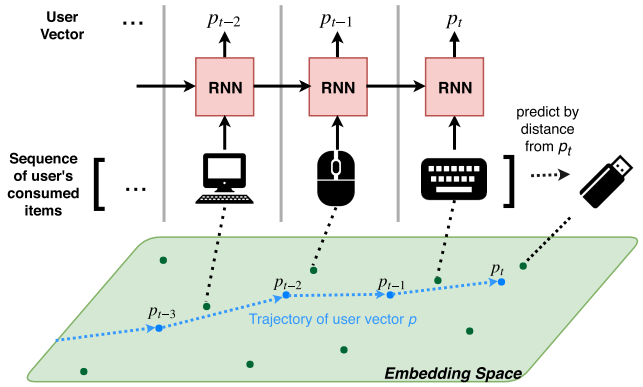



**FIGURE 1.** User short-term preference modeling of RTN. A user's embedding vector is created by the sequence of the user's consumed items. An update of the vector happens whenever the user interacts with an item. A recurrent unit is used for the update. The next item is predicted based on the distance in the embedding space.

performs better than MF-based model on sparse datasets. However, TransRec cannot handle higher-order interactions and is limited to only one previous item and a user as prior.

Integrating higher-order interactions between items is challenging for sparse datasets. Algorithms need to be carefully designed to tackle sparsity issues. It also needs to balance between *long-term* and *short-term* preferences of a user for which the signal of sequential patterns are weak but vital for prediction of next items.

In this work, we tackle those problems and present a new algorithm named *Recurrent Translation-based Network* (RTN).[1] RTN aims to model both short-term and long-term of a user's preference. For modeling the short-term, we propose an innovative way to fuse the translation-based model with RNN, which enables RTN to incorporate higher-order interactions between items for representing a user's short-term preference. Figure 1 illustrates how RTN updates a user's embedding vector based on the previously interacted items. In specific, we prefer the model to have the following property: when a user interacted with an item, and this item is close to another item, then the model will recommend that item to the user. This means that information on historical items can directly infer to the next one. However, using existing recurrent units may not have this property we desire. Then, we propose our recurrent unit that utilizes the advantage of the distance as the scoring scheme.

In summary, our contributions are listed as follows:

- We propose a new algorithm RTN, using higher-order interactions between items in a user's sequence for profiling short-term preference of the user. RTN is a framework that incorporates previous methods like TransRec [13] and PRME [14] as special cases.
- We build RTN's recurrent unit by exploiting distance in the embedding space to satisfy the above property and improve the performance on sparse datasets.
- We show that RTN outperforms other algorithms, including the state-of-the-art methods for sparse datasets on several publicly available datasets.

[1]The source code is available online at https://github.com/nutcrtnk/RTN.

## II. RELATED WORK

Our work is related to the works in multiple areas. We first introduce and discuss general and sequential recommendations. Then we describe recurrent neural networks in recommender systems. Last, we briefly state about translation-based models.

### A. GENERAL RECOMMENDATION

Recommender Systems aim to predict future behaviors of users based on their historical feedbacks. User feedbacks can be *explicit* (e.g., ratings) [15], [16] and be *implicit* (e.g., purchases, views, likes, check-ins). Utilizing implicit feedbacks is difficult due to heavily imbalance between the number of observed and unobserved data. To alleviate this problem, [3] and [17] use different weights for samples' losses in point-wise manner. Similarly, negative sampling techniques are also widely adopted. Alternatively, [18] proposes pair-wise ranking using common sense that a user prefers items in the observed than items in the unobserved.

Since users' feedbacks can be represented as a matrix, matrix factorization decomposes the matrix as the dot product between low-rank user matrix and item matrix. Those matrices can be interpreted as latent factors of users' preferences and items' features, respectively. With higher expressiveness and their success for solving similar problems, neural networks have become a popular approach to recognize such latent factors and modeling complex fine-grained interactions. For example, [19] and [20] use factorization machine [21] to approximate the compatibility between users' embeddings and items' embeddings. Reference [22] uses multi-layer perceptrons. Reference [23] adopts denoising auto-encoders.

### B. SEQUENTIAL RECOMMENDATION

The main difference between sequential and general recommendation is in the form of input data. In sequential recommendation, input data must include sequential interactions of each user in chronological order. Sequential recommender systems are essentially required to utilize such data. Many works seek to model transitions between previous items to the next items. For example, FPMC [9] incorporates first-order Markov Chains to recognize these transitions with matrix factorization. Because a user's last interaction is often a key to predict the next interaction, the model performs well on sparse datasets. Fossil [10] extends FPMC by fusing higher-order Markov Chains with similarity models.

Higher order interactions can also be integrated by adopting common techniques from other areas of machine learning tasks (e.g., Natural Language Processing (NLP)). For instance, [24] and [25] adopt the concept of convolution neural networks. Convolution layers are used to extract features of $L$ previous items which are interpreted as *images* or

*words*. Reference [26] and [27] propose attention mechanism designed to identify key items for sequential recommendation. With success of Transformer [28] in NLP, self-attention has been explored by [29]. A recurrent neural network is an alternative approach to leverage a user's sequential of interactions.

### C. RECURRENT NEURAL NETWORK

Recurrent neural network (RNN) is a kind of neural networks which specializes in processing sequential data. Information from the past is memorized by its internal cells as cell states. For each time step, its internal cells are updated according to a function given the current cell states and the input of that time step. RNN is optimized to approximate such function. RNN has its advantage over MF for modeling and capturing sequential patterns. In the following, we illustrate the areas in the recommendation tasks in which RNN shows its advantages and achieves state-of-the-art over traditional methods.

*Explicit rating prediction [5]:* uses Long-Short Term Memory (LSTM) [30], a popular recurrent neural network architecture. By slicing timestamps into time windows, the model uses explicit ratings of each time window in order to evolve a user's current state and an item's current state to their next states. Reference [31] proposes two layers of LSTMs for learning textual content and temporal dynamics.

*Session-Based Recommendation:* While a user is browsing, the session-based recommender system needs to suggest suitable items right away based on the user's behavior in the current session. A notable work on this topic is [6] that proposes a recurrent neural network using gated recurrent units (GRU) [32] for predicting next items in a session. [33] extends the work by propagating a user's information across the user's sessions. Reference [7] improves the work by including an attention mechanism.

In addition, RNN have been used for next basket recommendation [8], Point-of-Interest (POI) recommendation [34], scientific paper recommendation [35] and citation recommendation [36].

### D. TRANSLATION-BASED MODEL

The first translation-based model (TransE) [12] was proposed for knowledge graph embedding. In knowledge graph embedding, the goal is to predict missing *relationships* between a pair of *entities*. It is common to represent the data as triples of (*head*, *relation*, *tail*). For example, (''A'', ''parent of'', ''B'') means ''A'' is a parent of ''B''. In TransE, each entities and relations is embedded in the same low-dimensional space. Then it is optimized to make summation of a head and a relation to be close to the tail measured by the euclidean distance in that space. In spite of its simplicity, the model can handle *anti-symmetric*, *inversion* and *composition* patterns of relations [37]. Many works [38]–[40] improved TransE by projecting the embedding vectors of a head and a tail to a target *relational space* before calculating the distance. TransRec [13] reinterprets a triple as (previous item, user, next item) for sequential recommendation. Reference [41] extended TransRec by fusing it with factorization machine. Furthermore, translation-based models have been used for movie recommendation [42] and top-N recommendation [43].

## III. PROBLEM FORMULATION

For each user $u \in \mathcal{U}$, we have a sequence of items which the user has interacted with in chronological order: $Seq_u = (Q_u^1, Q_u^2, \ldots, Q_u^n)$. Our aim is to utilize the above information and predict the next item which will be interacted by each user. In this paper, *time step t* or index *t* in $Q_u^t$ refers to the *t*-th order number, not the real timestamps used in [16] and [44]. Notations which are often used in this paper are indicated in Table 1. Bold characters indicate vectors or matrices.

**TABLE 1. Notation.**

| Notation | Explanation |
|---|---|
| $\mathcal{U}$ | set of users |
| $\mathcal{I}$ | set of items |
| $k$ | the number of embedding dimensions |
| $Q_u^t$ | the item that user $u$ interact with at time $t$ |
| $\mathbf{P}_u^t$ | short-term embedding vector of user $u$ at time $t$ |
| $\mathbf{Q}_u^t$ | short-term embedding vector of item $Q_u^t$ |
| $\mathbf{Q}_i$ | short-term embedding vector of item $i$ |
| $\mathbf{P}_u^{\mathrm{L}}$ | long-term embedding vector of user $u$ |
| $\mathbf{Q}_i^{\mathrm{L}}$ | long-term embedding vector of item $i$ |
| $b_i$ | bias of item $i$ |
| $S_{u,j}^t$ | score of user $u$ to item $j$ at time $t$ |
| $\mathrm{d}(x, y)$ | distance between x and y (square $\mathrm{L}_2$) |
| $\|\cdot\|_F^2$ | Frobenius norm |

## IV. RECURRENT TRANSLATION-BASED NETWORK

Our proposed model consists of the following two parts.

1) User short-term preference modeling: It aims to recognize the temporal dynamics of a user's preference with higher-order interactions between items in the user's sequence.
2) User long-term preference modeling: It captures static factors of a user's preference.

### A. USER SHORT-TERM PREFERENCE MODELING

Since we extend concepts of the translation-based model to this model, we restate Translation-based Recommendation (TransRec) [13]. TransRec embeds users and items into the same embedding space. It uses three main components which are a user's embedding vector, a previous item's embedding vector, and a next item's embedding vector to compute the recommendation score. A recommendation score $S_u^{i \to j}$ of user $u$ from previous item $i$ to item $j$ is defined as:

$$S_u^{i \to j} = b_j - \mathrm{d}(\mathbf{P}_u + \mathbf{Q}_i, \mathbf{Q}_j), \tag{1}$$

where $\mathbf{P}_u$ is the embedding vector of user $u$, $\mathbf{Q}_i$ is the embedding vector of previous item $i$, and $\mathbf{Q}_j$ and $b_j$ are the embedding vector and the bias of item $j$, respectively.

In TransRec, a user's embedding vector is interpreted as a translation vector which enables a transition from a previous

item to the next item. Conversely, the model can also be interpreted such that each previous item as a translation vector enables a transition from a user to the next item. However, the model cannot deal with sequentially related items when the gap of their time steps are more than one. The reason is that a user's embedding vector cannot keep the information on past items that are farther than one time step from the current. For example, after user $u$ interacts with $j$, the information of $i$ ($\mathbf{Q}_i$) is no longer included in eq. (1) and thus will not affect $\mathbf{P}_u$ any more.

To incorporate higher-order interactions between items, a simple way to extend TransRec is to directly apply recurrent units, e.g., LSTM [30] and GRU [32], for updating $\mathbf{P}^t_u$ and making $\mathbf{P}^t_u \approx \mathbf{Q}^{t+1}_u$, where $\mathbf{P}^t_u$ is the embedding vector of user $u$ at time $t$, and $\mathbf{Q}^{t+1}_u$ is the embedding vector of $Q^{t+1}_u$.

However, we also want to exploit distance in the embedding space for next item recommendation. Specifically, if user $u$ interacted with item $i$ at time $t$ (namely $i = Q^t_u$) and item $i$ and item $j$ are close in the embedding space, then the model should recommend item $j$ to user $u$ at time $t+1$. This helps the model dealing with sparsity issues by grouping sequentially related items together.

By triangular inequality, we have:

$$\mathrm{d}(\mathbf{P}^t_u, \mathbf{Q}_j) \leq \mathrm{d}(\mathbf{P}^t_u, \mathbf{Q}_i) + \mathrm{d}(\mathbf{Q}_i, \mathbf{Q}_j) \tag{2}$$

Note that $\mathbf{Q}_i$ and $\mathbf{Q}_j$ are close, and $\mathbf{P}^{t-1}_u$ and $\mathbf{Q}_i$ are also close (since we have recommended $Q_i$ to $u$ at time $t$). That is,

$$\mathrm{d}(\mathbf{Q}_i, \mathbf{Q}_j) \leq \epsilon, \tag{3}$$
$$\mathrm{d}(\mathbf{P}^{t-1}_u, \mathbf{Q}_i) \leq \epsilon, \tag{4}$$

where $\epsilon$ is a small number. If we can have

$$\mathrm{d}(\mathbf{P}^t_u, \mathbf{Q}_i) \leq \mathrm{d}(\mathbf{P}^{t-1}_u, \mathbf{Q}_i), \tag{5}$$

we will arrive at

$$\mathrm{d}(\mathbf{P}^t_u, \mathbf{Q}_j) \leq 2\epsilon. \tag{6}$$

That is, $\mathbf{P}^t_u$ and $\mathbf{Q}_j$ are close and hence we will recommend item $j$ to user $u$ at time $t+1$ if ineq. (5) is satisfied.

Since the model utilizes euclidean distance, a way to guarantee ineq. (5) is to make

$$\forall r : |(\mathbf{P}^t_u)_r - (\mathbf{Q}_i)_r| \leq |(\mathbf{P}^{t-1}_u)_r - (\mathbf{Q}_i)_r|, \tag{7}$$

where $1 \leq r \leq k$, and the subscript $r$ indicates the $r$-th element of respective embedding vectors. By solving the ineq. (7) to find $(\mathbf{P}^t_u)_r$, we have:

$$\begin{aligned} &2(\mathbf{Q}_i)_r - (\mathbf{P}^{t-1}_u)_r \leq (\mathbf{P}^t_u)_r \leq (\mathbf{P}^{t-1}_u)_r \quad &&:\text{if } (\mathbf{Q}_i)_r \leq (\mathbf{P}^t_u)_r, \\ &2(\mathbf{Q}_i)_r - (\mathbf{P}^{t-1}_u)_r \geq (\mathbf{P}^t_u)_r \geq (\mathbf{P}^{t-1}_u)_r \quad &&:\text{otherwise.} \end{aligned}$$

We can see that $(\mathbf{P}^t_u)_r$ is between $2(\mathbf{Q}_i)_r - (\mathbf{P}^{t-1}_u)_r$ and $(\mathbf{P}^{t-1}_u)_r$. Therefore, we can write $\mathbf{P}^t_u$ as:

$$\mathbf{P}^t_u = (1 - \mathbf{z}_t) \circ \mathbf{P}^{t-1}_u + \mathbf{z}_t \circ (2\mathbf{Q}_i - \mathbf{P}^{t-1}_u), \tag{8}$$

or

$$\mathbf{P}^t_u = (1 - \mathbf{z}_t) \circ \mathbf{P}^{t-1}_u + \mathbf{z}_t \circ (2\mathbf{Q}^t_u - \mathbf{P}^{t-1}_u), \tag{9}$$

where $\mathbf{z}_t \in [0, 1]^k$, $\circ$ denotes Hadamard product, and we have used $\mathbf{Q}_i = \mathbf{Q}^t_u$. $\mathbf{z}$ controls the amount of information to be updated in same manner as the update gate of GRU. In specific, $\mathbf{z}_t$ can be obtained by:

$$\mathbf{z}_t = \sigma(\mathbf{W}_z\mathbf{Q}^t_u + \mathbf{U}_z\mathbf{P}^{t-1}_u + \mathbf{b}_z), \tag{10}$$

where $\mathbf{W}_z, \mathbf{U}_z \in \mathbb{R}^{k\times k}$ denote weights, $\mathbf{b} \in \mathbb{R}^k$ denotes bias of the gate, and $\sigma$ denotes the sigmoid function. Moreover, we found that limiting the range of $(\mathbf{P}^t_u)_r$ to $(\mathbf{Q}^t_u)_r$ instead of $2(\mathbf{Q}^t_u)_r - (\mathbf{P}^{t-1}_u)_r$ still satisfies the property and empirically achieves higher performance for sparse datasets (Please see Sec. V-D for details). This is because it reduces the influence of previous item (from $\mathbf{z}_t \circ 2\mathbf{Q}^t_u$ to $\mathbf{z}_t \circ \mathbf{Q}^t_u$). In summary, the update of the user's short-term embedding vector is:

$$\mathbf{z}_t = \sigma(\mathbf{W}_z\mathbf{Q}^t_u + \mathbf{U}_z\mathbf{P}^{t-1}_u + \mathbf{b}_z), \tag{11}$$
$$\mathbf{P}^t_u = (1 - \mathbf{z}_t) \circ \mathbf{P}^{t-1}_u + \mathbf{z}_t \circ \mathbf{Q}^t_u. \tag{12}$$

### B. USER LONG-TERM PREFERENCE MODELING

User Short-term Preference modeling groups sequentially related items together. However, a user may consider items based on his/her long-term preference, not sequentially related to recent items. For example, a user may consider smartphone accessories if he/she recently bought a smartphone, but his/her long-term preference may be others and purchases them shortly after the smartphone. We assume that long-term preferences of users do not change over time. Using latent factor approach [45], [46], we assign each user $u \in \mathcal{U}$ and each item $i \in \mathcal{I}$ a latent factor (embedding) $\mathbf{P}^{\mathrm{L}}_u \in \mathbb{R}^K$ and $\mathbf{Q}^{\mathrm{L}}_i \in \mathbb{R}^K$, respectively. We adopt distance as the scoring scheme for long-term preference of user $u$ to item $i$: $-\mathrm{d}(\mathbf{P}^{\mathrm{L}}_u, \mathbf{Q}^{\mathrm{L}}_i)$.

Note that we differentiate items' long-term and short-term embeddings because they can be related to different items. For example, a movie is *long-term* related to movies in the same category, but can be *short-term* related to movies released in the same year.

Observed by [45], due to the triangular inequality, the relationships propagated using distance as a scoring scheme are not only explicit between user-item pairs, but also implicit between item-item pairs and between user-user pairs. Thus, user long-term preference modeling will bring items interacted by the same user to be close regardless of their positions in the user's sequence.

### C. COMBINING SHORT-TERM AND LONG-TERM PREFERENCE MODELING

By combining both user short-term and long term preference modeling, we propose our RTN model. Specifically, RTN computes a recommendation score between user $u$ and item $j$ at time step $t$ using the following equation:

$$S^t_{u,j} = -\alpha\,\mathrm{d}(\mathbf{P}^{\mathrm{L}}_u, \mathbf{Q}^{\mathrm{L}}_j) - (1 - \alpha)\,\mathrm{d}(\mathbf{P}^t_u, \mathbf{Q}_j) + b_j, \tag{13}$$

where $\alpha$ is a hyper-parameter for balancing weights of both of the following terms. The first term indicates the recommendation score of long-term preference, while the second term

denotes the recommendation score of short-term preference. Recommending top $N$ items to user $u$ is done by selecting items with the top $N$ highest value of the recommendation scores.

To train RTN, we use the following objective function:

$$L(\Theta) = \sum_{u \in \mathcal{U}} \sum_{t=2}^{|Seq_u|} \sum_{\substack{i \in \mathcal{I} \\ i \neq Q_u^t}} -\ln \sigma(S_{u,Q_u^t}^t - S_{u,i}^t) + \lambda_\Theta \left\| \Theta \right\|_F^2, \quad (14)$$

where $\lambda_\Theta$ is a regularization hyper-parameter for all of the model parameters $\Theta$. Since the objective of the model is to rank positives over negatives, we obtain the term $-\ln \sigma(S_{u,Q_u^t}^t - S_{u,i}^t)$ from adjusting *Bayesian Personalized Ranking* [9] by including time steps. Back-Propagation Through Time (BPTT) [47] algorithm can be used for learning the parameters. Because the number of negative items of each user is huge for sparse datasets, we instead use random uniform sampling to sample 9 negatives for each positive. Regularization term is also applied to only related items.

### D. CONNECTIONS TO EXISTING MODELS

By restricting some of its components, we can transform RTN to the following existing models.

*Personalized Ranking Metric Embedding (PRME) [14]:* In PRME, a recommendation score $S_u^{i \to j}$ of user $u$ from previous item $i$ to item $j$ can be written as:

$$S_u^{i \to j} = -\alpha \, \mathrm{d}(\mathbf{P}_u^{\mathrm{L}}, \mathbf{Q}_j^{\mathrm{L}}) - (1 - \alpha) \, \mathrm{d}(\mathbf{Q}_i, \mathbf{Q}_j). \quad (15)$$

Suppose the previous item $i = Q_u^t$. By comparing eq. (15) and eq. (13), we can find that we can transform RTN to PRME by strictly restricting $\mathbf{z}_t$ to 1 and remove the bias term. In this case, user short-term preference modeling of RTN will use only a single previous item of a user and cannot hold information of the past interactions.

*Translation-Based Recommendation (TransRec) [13]:* From eq. (1), we can also consider $\mathbf{P}_u$ as the long-term embedding of user $u$, namely $\mathbf{P}_u = \mathbf{P}_u^{\mathrm{L}}$. With slight modification, we can combine short-term and long-term embeddings of the user's preference to the same space in the same manner as TransRec:

$$S_u^{i \to j} = -\mathrm{d}(\mathbf{P}_u^{\mathrm{L}} + \mathbf{P}_u^t, \mathbf{Q}_j) + b_j. \quad (16)$$

In this way, the number of parameters is reduced. However, the model will lose the property for recommending next items based on closeness in the embedding space to a previous item. Additionally, long-term and short-term relations between items cannot be separated.

## V. EXPERIMENTS

We evaluate our model on a wide range of real-world datasets to answer the following questions:

*Q1:* Does the proposed model achieve state-of-the-art performance on sparse datasets?

*Q2:* What is the influence of each component of RTN on the performance?

**TABLE 2. Dataset statistics.**

| Dataset | $\lvert\mathcal{U}\rvert$ | $\lvert\mathcal{I}\rvert$ | #acts | #acts/$\lvert\mathcal{U}\rvert$ |
|---|---|---|---|---|
| Amz-App | 22700 | 20888 | 368398 | 16.23 |
| Amz-Elec | 63168 | 120105 | 1000572 | 15.84 |
| Amz-Game | 8056 | 20382 | 149081 | 18.51 |
| Amz-Home | 25238 | 70142 | 366784 | 14.53 |
| Amz-Music | 6086 | 16966 | 76560 | 12.58 |
| Amz-Office | 3710 | 13052 | 58209 | 15.69 |
| Amz-Video | 1368 | 5322 | 19828 | 14.49 |
| Foursquare | 7390 | 6884 | 97133 | 13.14 |
| Ml-1M | 6040 | 3708 | 1000156 | 165.59 |

*Q3:* Can RTN deal with various lengths of users' sequences or the effect of data sparsity?

### A. EXPERIMENTAL SETUP

#### 1) DATASETS

We conduct experiments on widely used public datasets: *Amazon*, *Foursquare*, and *MovieLens*. *Amz* and *Ml* are the abbreviations of *Amazon* and *MovieLens* respectively.

***Amazon***.[2] A large group of datasets collected by [48] from May 1996 to July 2014 from *Amazon.com*.

***Foursquare***.[3] A dataset commonly used for evaluating a model performance on POI recommendation. The data is collected by [49] from December 2011 to April 2012 from *Foursquare.com*. We use only check-in information of users.

***MovieLens***.[4] A very popular dataset which is widely used for many areas of recommendation tasks. We use the MovieLens 1M where each user has rated at least 20 items. Based on the number of actions per user, the dataset is not sparse, but we include this for evaluating the performance of our model on a dataset with long-term dependency.

We utilize only the information of user ids, item ids (or location ids in Foursquare) and timestamps. We filter out users and items with less than 10 and 5 interactions, respectively.

#### 2) EVALUATION PROTOCOL

For each dataset, we split it into three sets for training, validation, and testing. Following [10], [13], the last interaction of each user is held for testing while the second last is held for validation. The remaining data is used for training purpose. We remove items which do not appear in the training set from the validation set and the test set. We train models on the training set while tuning their hyper-parameters on the validation set. The model achieving the highest score on the validation set is then evaluated on the test set, and finally, we report the performance. The statistics of datasets are indicated in Table 2.

[2] http://jmcauley.ucsd.edu/data/amazon/
[3] https://archive.org/details/201309_foursquare_dataset_umn
[4] https://grouplens.org/datasets/movielens/

#### 3) EVALUATION METRICS

Following [22], [50], we evaluate each methodology using following metrics:

***Hit Ratio @K*** (HR@$K$): Given a list of top $K$ predicted items for each user, HR@$K$ measures whether the test item is in the list.

***Normalized Discounted Cumulative Gain @k*** (NDCG@$K$): It is similar to HR@$K$ with consideration of the position of the test item in the list.

We set $K$ to 50 as same as [13]. We report the average scores over users for both metrics.

### B. COMPARISON METHODS

To evaluate the performance of our proposed algorithm, we compare our model to several popular methods and standard baselines. We also compare with recent state-of-the-art methods in sparse datasets. We list methods evaluated in this work in the following.

***Popularity*** **(Pop):** It is a naive baseline method. The method always ranks items and make recommendations based on their popularity only.

***Matrix Factorization with Bayesian Personalized Ranking*** (MF) [18]: The method uses the matrix factorization technique with the inner product between a user and an item as its scoring scheme.

***Factorized Personalized Markov Chains*** (FPMC) [9]: The method uses combination of Matrix Factorization and first-order Markov chains.

***Personalized Ranking Metric Embedding*** (PRME) [14]: Instead of using the inner product as scoring scheme like FPMC, the model uses Euclidean distance and has an additional hyper-parameter for weighting between two of them.

***Factorized Sequential Prediction with Item Similarity Models*** (Fossil) [10]: Instead of learning user embedding vectors directly, Fossil uses item similarity model to represent long-term preference of a user, while high-order Markov Chains handles short-term preference.

***GRU4Rec*** [6]: This was proposed for session-based recommendation. The model uses GRU to capture sequential dependencies and make a prediction.

***Translation-based Recommendation*** (TransRec) [13]: The model is described in eq. (1).

***Self-Attentive Sequential Recommendation*** (SAS) [29]: SAS uses self-attention mechanism to capture higher-order interactions between items in a user's sequence.

***Recurrent Translation-based Network*** (RTN): Our proposed method.

Hyper-parameters are tuned with grid search with the validation set. The number of embedding dimensions $k$ is tuned among {16, 32, 64}. Regularization terms are tuned among {0, 0.1, 0.01, 0.001}. $\alpha$ and $L$ of Fossil are set to 0.2 and 3 respectively. $\alpha$ of PRME and RTN is search from {0.2, 0.5, 0.8}. Dropout of GRU4Rec is tuned among {0, 0.25, 0.5}. BPRMF, FMPC, Fossil, and TransRec are trained with Stochastic Gradient Ascent with learning rate 0.05. We use the code provided by [10], [13] for training them. We train SAS with the code provided by the authors [29] with its default setting, which also used for evaluating the model in their paper (except the number of embedding dimensions). We have implemented GRU4Rec and RTN using PyTorch[5] library. GRU4Rec and RTN are trained with mini-batch gradient descent and optimized by Adam [51]. We set the batch size of both to 100 and learning rate to 0.001.

### C. PERFORMANCE COMPARISON

The results are collected and illustrated in Table 3. Our proposed model, RTN, clearly achieves the best performance on all datasets. Performing on par with GRU on the dense dataset indicates that RTN is flexible in dealing with several lengths of users' sequences. Achieving state-of-the-art performance beyond all baselines on sparse datasets answers to **Q1**.

Furthermore, we make several observations on the comparison between our baselines. Unsurprisingly, Pop model performs the worst on all datasets and both metrics. This indicates the importance of each user's personal preference. Achieving higher performance of FPMC over MF shows that the latest item is also crucial. Fossil constructs user preference vector by aggregating the user's history instead of learning the user preference vector directly like FPMC. This seems to be a weakness of the model when applying on the dense dataset, *MovieLens-1M*. GRU4Rec performs inconsistently due to its complexity. Performing worse than best baselines in most cases of GRU4Rec implicitly informs the difference in data from session-based recommendations. In session-based recommendation, items in the same session highly correlate to each other and GRU4Rec exploits those relations. We can see the importance of modeling higher-order interactions of RTN from its superior performance over PRME and TransRec. SAS performs on the top among the baselines for several cases. This validates that the attention mechanism is an alternative way to incorporate higher-order interactions effectively.

In Figure 2, we analyze the effect of a key hyper-parameter, the number of embedding dimensions $k$ from 8 to 64. We can observe that all models are benefited from a large number of embedding dimensions. RTN performs well compared to other methods in low dimensional space. This supposedly dues to the separated components, long-term and short-term modules, of RTN. Similarly, FPMC and TransRec also perform well as RTN in low dimensional space.

### D. ABLATION STUDY

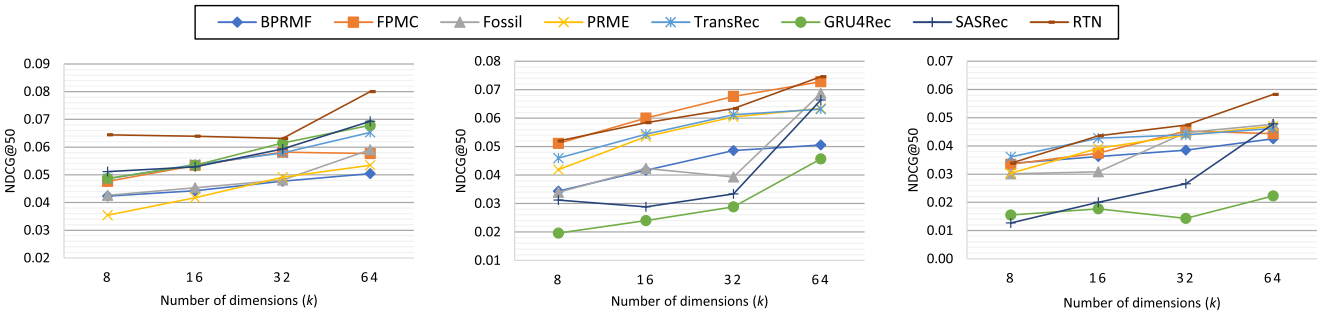
To answer **Q2**, we conduct further experiments to analyze the model components. Table 4 shows the performance of RTN with the default configuration and its eight variants on three categories of Amazon dataset (with $k = 64$). We analyze their effect, respectively:

(1) w/o Short-term (without Short-term preference modeling): Unsurprisingly, the performance is tremendously worse

[5]https://pytorch.org

**TABLE 3.** Top-N recommendation results on datasets (higher is better). Bold texts indicate the best performance in each case. Underline texts indicate the best among baselines. GRU refers to GRU4Rec, and TR refers to TransRec. The last two columns (%Improve) show the improvement of RTN over other methods and TransRec.

| Dataset | Metric | Model | | | | | | | | | %Improve | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Pop | BPRMF | FPMC | Fossil | PRME | TR | GRU | SAS | RTN | vs all | vs TR |
| Amz-App | HR | .121 | .165 | .182 | .188 | .163 | .206 | .193 | .214 | **.235** | 9.67% | 14.21% |
| | NDCG | .041 | .050 | .058 | .059 | .053 | .065 | .068 | .069 | **.080** | 15.48% | 22.54% |
| Amz-Elec | HR | .055 | .063 | **.072** | .051 | .049 | .067 | .058 | .049 | .071 | -0.56% | 6.12% |
| | NDCG | .019 | .019 | .023 | .015 | .017 | .021 | .020 | .014 | **.024** | 7.33% | 16.62% |
| Amz-Game | HR | .056 | .159 | .213 | .211 | .187 | .197 | .141 | .210 | **.215** | 1.05% | 9.59% |
| | NDCG | .018 | .051 | .073 | .069 | .063 | .063 | .046 | .066 | **.075** | 2.41% | 18.11% |
| Amz-Home | HR | .030 | .038 | .046 | .042 | .031 | .046 | .027 | .041 | **.049** | 4.96% | 7.03% |
| | NDCG | .008 | .011 | .014 | .012 | .011 | .014 | .009 | .012 | **.016** | 12.61% | 18.29% |
| Amz-Music | HR | .032 | .140 | .130 | .145 | .131 | .148 | .074 | .152 | **.167** | 9.70% | 12.56% |
| | NDCG | .010 | .042 | .044 | .048 | .047 | .046 | .022 | .048 | **.058** | 21.91% | 26.23% |
| Amz-Office | HR | .012 | .046 | .148 | .148 | .180 | .091 | .156 | **.221** | .193 | -12.85% | 111.70% |
| | NDCG | .003 | .014 | .043 | .045 | .056 | .031 | .047 | .063 | **.064** | 2.20% | 104.67% |
| Amz-Video | HR | .129 | .248 | .219 | .248 | .198 | .241 | .122 | .246 | **.261** | 4.96% | 8.03% |
| | NDCG | .035 | .085 | .079 | .083 | .078 | .088 | .040 | .088 | **.097** | 9.30% | 10.13% |
| Foursquare | HR | .545 | .764 | .765 | .688 | .682 | .747 | .703 | .763 | **.777** | 1.46% | 4.02% |
| | NDCG | .200 | .443 | .419 | .255 | .375 | .363 | .333 | .452 | **.466** | 3.07% | 28.44% |
| Ml-1M | HR | .138 | .263 | .436 | .143 | .376 | .340 | **.528** | .527 | .521 | -1.26% | 53.44% |
| | NDCG | .041 | .080 | .157 | .040 | .134 | .108 | .198 | .194 | **.199** | 0.36% | 83.55% |




(a) Amz-App (b) Amz-Game (c) Amz-Music

**FIGURE 2.** Performance (NDCG@50) of all models with varying numbers of embedding dimensions $k$.

**TABLE 4.** Ablation analysis (HR@50) on three categories of Amazon dataset. Performance better than the default version is boldfaced. ↓ indicates a significant drop in performance (more than 10%).

| Architecture | App | Game | Office |
|---|---|---|---|
| (0) Default | .234 | .215 | .193 |
| (1) w/o Short-term | .164↓ | .159↓ | .054↓ |
| (2) w/o Long-term | .193↓ | .188↓ | .183 |
| (3) w/o Bias | .228 | **.220** | .185 |
| (4) w/o Limit | .191↓ | .159↓ | .169↓ |
| (5) Tied weights | .215 | **.217** | .159↓ |
| (6) RNN | .176↓ | .123↓ | .139↓ |
| (7) LSTM | .191↓ | .162↓ | .173↓ |
| (8) GRU | .201↓ | .162↓ | .169↓ |

without short-term preference modeling because the model cannot capture and utilize sequential information without it.

(2) w/o Long-term (without Long-term preference modeling): Without Long-term preference modeling, the performance drops significantly but not as much as without short-term preference modeling. Because the short-term preference modeling can hold the information of multiple recent items, therefore, it partially captures the long-term preference of a user.

(3) w/o Bias (without bias term): The performance slightly changes without the bias term.

(4) w/o Limit (without limiting the update to $\mathbf{Q}_u^t$, namely updating $\mathbf{P}_u^t$ by (9) instead of (12)): In this setting, the performance is much inferior. We investigate the influence of previous items on the update of user's short-term vectors by monitoring the average value of $\mathbf{z}_t$ in the equation (9) and (12) across all embedding dimensions as the influence. We find that the influence of previous items is much stronger without the limit. For example, in the App category, the average influence is 1.024 ($2\mathbf{z}_t$) without the limit and 0.639 with the limit. This affects in less reliance on past user's preference vectors of the model and hinders the performance.

(5) Tied weights: By tying $\mathbf{Q}_i$ to $\mathbf{Q}_i^{\mathrm{L}}$, the performance is increased in the Game category. This may be due to a weaker signal of sequential patterns in the category compared to the others (as can be observed from the difference in the performance between GRU4Rec and MF). However, we can see the decline of performance when the signal is stronger in App and Office categories.

(6)-(8) Recurrent units: We change our modified recurrent unit to RNN (simple with the hyperbolic tangent activation), LSTM and GRU respectively. As can be seen, the performance drops significantly. By comparing the performance of the model with GRU variance to GRU4Rec (in Table 3), it achieves higher performance and can be benefited from using distance as a scoring scheme.

### E. EFFECT OF DATA SPARSITY

To evaluate the effect of data sparsity for answering **Q3**, we follow the test used in [10]. We perform experiments on MovieLens-1M dataset. For converting the dataset to be sparse, we truncate each user's sequence by taking only the latest $N$ interactions and dropping the rest. We conduct experiments with the decreasing of $N$ from 200 to 5, leading the data to be more sparsity and inspect the change of performance of each model. The experiment results are collected in Table 5.

**TABLE 5. Performance (HR@50) of five models on Ml-1M dataset with varying $N$.**

| Dataset | Fossil | TR | GRU | SAS | RTN |
|---|---|---|---|---|---|
| Ml-1M_200 | .210 | .323 | .474 | .479 | **.500** |
| Ml-1M_100 | .267 | .308 | .442 | .463 | **.481** |
| Ml-1M_50 | .315 | .324 | .409 | **.463** | .451 |
| Ml-1M_20 | .344 | .370 | .346 | .409 | **.411** |
| Ml-1M_10 | .340 | **.376** | .253 | .374 | .374 |
| Ml-1M_5 | .282 | **.303** | .127 | .260 | .269 |

As can be seen from the table, the performance of RTN, SAS, and GRU4Rec which utilize high-order sequential information drops as decreasing of $N$. This is due to the length of sequences which the models can exploit is lower. With long sequences, $N = 200$, GRU4Rec performs on par with RTN. However, the drop in performance of GRU4Rec is significantly larger than RTN. With $N = 10$, RTN and TransRec accomplish similar results which are enormously higher than GRU4Rec. With $N = 5$, RTN performs slightly worse than TransRec and Fossil. It cannot benefit from sequential information in this case. In contrast to RTN, with higher $N$, the performance of Fossil and TransRec are not improved and are declined instead. This shows that both cannot handle long sequences of users' items properly. Achieving top performance of RTN on various lengths of users' sequences of items answers to **Q3**. SAS performs nearly as well as RTN in multiple $N$. The self-attention mechanism is worth considering for tackling sparsity issues.

### F. EMBEDDING VISUALIZATION

For understanding the model, we visualize the item embedding space ($\mathbf{Q}_*$) in a two-dimensional space using t-SNE [52]. It is shown in Figure 3 and 4. It is intuitively known that users prefer watching movies from their favorite genres. We can see that movies are clustered according to their genres. Note that this information has not been presented to the model. Therefore, the model implicitly learned that from users' sequences of movies in order to predict their next watched movies, since the model groups sequentially related items via its recurrent unit and distance as its scoring scheme. In addition, from the difference between Figure 3 and Figure 4, the model is better at categorizing those when it is fed with longer the sequences. This indicates that the model can handle long sequences of users' items as well.

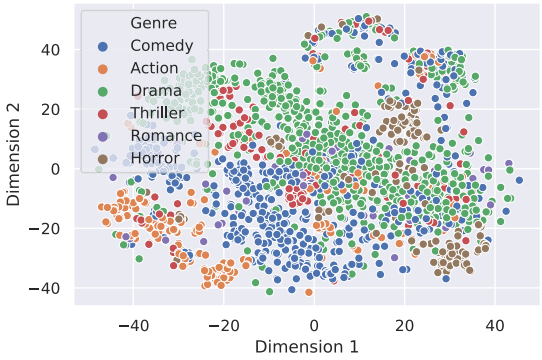



**FIGURE 3. Visualization of item embedding space on Ml-1M_20 dataset.**

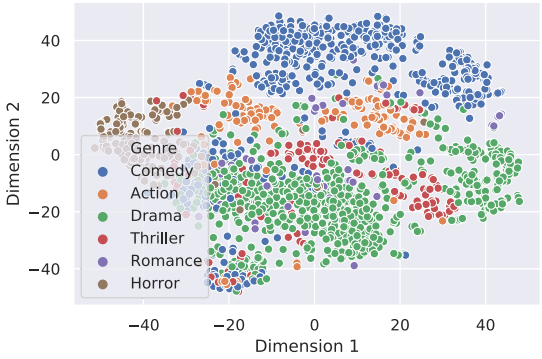



**FIGURE 4. Visualization of item embedding space on Ml-1M_100 dataset.**

## VI. CONCLUSION

In this paper, we proposed RTN, a novel higher-order interactions approach for sparse sequential recommendation. RTN models both temporal dynamics factors and static factors of user preference. RTN combines the idea of the translation-based model, making user transition from one item to another, with recurrent neural network architecture. For utilizing distance in an embedding space, we use our recurrent unit for modeling user's short-term preference. It makes sequentially related items to be close in the embedding space and helps the model tackle sparsity issues. Our thorough experiments on many sparse datasets demonstrate that RTN outperforms the state-of-the-art methods for sparse sequential recommendation.

## REFERENCES


[1] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, *Recommender Systems Handbook*, 1st ed. New York, NY, USA: Springer-Verlag, 2010.

[2] T. Donkers, B. Loepp, and J. Ziegler, "Sequential user-based recurrent neural network recommendations," in *Proc. 11th ACM Conf. Recommender Syst.*, 2017, pp. 152–160.

[3] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," in *Proc. 8th IEEE Int. Conf. Data Mining*, Dec. 2008, pp. 263–272.

[4] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *IEEE Comput.*, vol. 42, no. 8, pp. 30–37, Aug. 2009.

[5] C.-Y. Wu, A. Ahmed, A. Beutel, A. J. Smola, and H. Jing, "Recurrent recommender networks," in *Proc. 10th ACM Int. Conf. Web Search Data Mining*, 2017, pp. 495–503.

[6] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks," in *Proc. ICLR*, 2016, pp. 1–10.

[7] J. Li, P. Ren, Z. Chen, Z. Ren, T. Lian, and J. Ma, "Neural attentive session-based recommendation," in *Proc. ACM Conf. Inf. Knowl. Manage.*, 2017, pp. 1419–1428.

[8] F. Yu, Q. Liu, S. Wu, L. Wang, and T. Tan, "A dynamic recurrent model for next basket recommendation," in *Proc. 39th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, 2016, pp. 729–732.

[9] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme, "Factorizing personalized Markov chains for next-basket recommendation," in *Proc. 19th Int. Conf. World Wide Web*, 2010, pp. 811–820.

[10] R. He and J. McAuley, "Fusing similarity models with Markov chains for sparse sequential recommendation," in *Proc. IEEE 16th Int. Conf. Data Mining (ICDM)*, Dec. 2016, pp. 191–200.

[11] S. Kabbur, X. Ning, and G. Karypis, "FISM: Factored item similarity models for top-N recommender systems," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2013, pp. 659–667.

[12] A. Bordes, N. Usunier, A. Garcia-Durán, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *Proc. 26th Int. Conf. Neural Inf. Process. Syst.*, vol. 2, 2013, pp. 2787–2795.

[13] R. He, W.-C. Kang, and J. McAuley, "Translation-based recommendation," in *Proc. 11th ACM Conf. Recommender Syst.*, 2017, pp. 161–169.

[14] S. Feng, X. Li, Y. Zeng, G. Cong, Y. M. Chee, and Q. Yuan, "Personalized ranking metric embedding for next new POI recommendation," in *Proc. 24th Int. Conf. Artif. Intell.*, 2015, pp. 2069–2075.

[15] R. Salakhutdinov and A. Mnih, "Probabilistic matrix factorization," in *Proc. 20th Int. Conf. Neural Inf. Process. Syst.*, 2007, pp. 1257–1264.

[16] Y. Koren, "Collaborative filtering with temporal dynamics," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2009, pp. 447–456.

[17] X. He, H. Zhang, M.-Y. Kan, and T.-S. Chua, "Fast matrix factorization for online recommendation with implicit feedback," in *Proc. 39th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, 2016, pp. 549–558.

[18] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "BPR: Bayesian personalized ranking from implicit feedback," in *Proc. 25th Conf. Uncertainty Artif. Intell.*, 2012, pp. 452–461.

[19] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He, "DeepFM: A factorization-machine based neural network for CTR prediction," in *Proc. 26th Int. Joint Conf. Artif. Intell.*, 2017, pp. 1725–1731.

[20] J. Lian, X. Zhou, F. Zhang, Z. Chen, X. Xie, and G. Sun, "xDeepFM: Combining explicit and implicit feature interactions for recommender systems," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2018, pp. 1754–1763.

[21] S. Rendle, "Factorization machines," in *Proc. IEEE Int. Conf. Data Mining*, Dec. 2010, pp. 995–1000.

[22] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proc. 26th Int. Conf. World Wide Web*, 2017, pp. 173–182.

[23] Y. Wu, C. DuBois, A. X. Zheng, and M. Ester, "Collaborative denoising auto-encoders for Top-N recommender systems," in *Proc. 9th ACM Int. Conf. Web Search Data Mining*, 2016, pp. 153–162.

[24] J. Tang and K. Wang, "Personalized Top-N sequential recommendation via convolutional sequence embedding," in *Proc. 11th ACM Int. Conf. Web Search Data Mining*, 2018, pp. 565–573.

[25] F. Yuan, A. Karatzoglou, I. Arapakis, J. M. Jose, and X. He, "A simple convolutional generative network for next item recommendation," in *Proc. 12th ACM Int. Conf. Web Search Data Mining*, 2019, pp. 582–590.

[26] H. Ying, F. Zhuang, F. Zhang, Y. Liu, G. Xu, X. Xie, H. Xiong, and J. Wu, "Sequential recommender system based on hierarchical attention networks," in *Proc. 27th Int. Joint Conf. Artif. Intell.*, 2018, pp. 1–7.

[27] C. Zhou, J. Bai, J. Song, X. Liu, Z. Zhao, X. Chen, and J. Gao, "ATRank: An attention-based user behavior modeling framework for recommendation," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 1–8.

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.

[29] W.-C. Kang and J. McAuley, "Self-attentive sequential recommendation," in *Proc. IEEE Int. Conf. Data Mining*, Nov. 2018, pp. 197–206.

[30] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[31] J.-D. Zhang and C.-Y. Chow, "SEMA: Deeply learning semantic meanings and temporal dynamics for recommendations," *IEEE Access*, vol. 6, pp. 54106–54116, 2018.

[32] K. Cho, B. van Merrienboer, C. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2014, pp. 1724–1734.

[33] M. Quadrana, A. Karatzoglou, B. Hidasi, and P. Cremonesi, "Personalizing session-based recommendations with hierarchical recurrent neural networks," in *Proc. 11th ACM Conf. Recommender Syst.*, 2017, pp. 130–137.

[34] Q. Liu, S. Wu, L. Wang, and T. Tan, "Predicting the next location: A recurrent model with spatial and temporal contexts," in *Proc. 13th AAAI Conf. Artif. Intell.*, 2016, pp. 194–200.

[35] T. Bansal, D. Belanger, and A. McCallum, "Ask the GRU: Multi-task learning for deep text recommendations," in *Proc. 10th ACM Conf. Recommender Syst.*, 2016, pp. 107–114.

[36] L. Yang, Y. Zheng, X. Cai, H. Dai, D. Mu, L. Guo, and T. Dai, "A LSTM based model for personalized context-aware citation recommendation," *IEEE Access*, vol. 6, pp. 59618–59627, 2018.

[37] Z. Sun, Z.-H. Deng, J.-Y. Nie, and J. Tang, "RotatE: Knowledge graph embedding by relational rotation in complex space," in *Proc. ICLR*, 2019, pp. 1–18.

[38] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes," in *Proc. 28th AAAI Conf. Artif. Intell.*, 2014, pp. 1112–1119.

[39] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, "Learning entity and relation embeddings for knowledge graph completion," in *Proc. 29th AAAI Conf. Artif. Intell.*, 2015, pp. 2181–2187.

[40] G. Ji, S. He, L. Xu, K. Liu, and J. Zhao, "Knowledge graph embedding via dynamic mapping matrix," in *Proc. 53rd Annu. Meeting Assoc. Comput. Linguistics 7th Int. Joint Conf. Natural Lang. Process.*, vol. 1, 2015, pp. 687–696.

[41] R. Pasricha and J. McAuley, "Translation-based factorization machines for sequential recommendation," in *Proc. 12th ACM Conf. Recommender Syst.*, 2018, pp. 63–71.

[42] M. He, B. Wang, and X. Du, "HI2Rec: Exploring knowledge in heterogeneous information for movie recommendation," *IEEE Access*, vol. 7, pp. 30276–30284, 2019.

[43] H. Wang, M. Zhao, X. Xie, W. Li, and M. Guo, "Knowledge graph convolutional networks for recommender systems," in *Proc. World Wide Web Conf.*, 2019, pp. 3307–3313.

[44] Y. Zhu, H. Li, Y. Liao, B. Wang, Z. Guan, H. Liu, and D. Cai, "What to do next: Modeling user behaviors by time-LSTM," in *Proc. 26th Int. Joint Conf. Artif. Intell.*, 2017, pp. 3602–3608.

[45] C.-K. Hsieh, L. Yang, Y. Cui, T.-Y. Lin, S. Belongie, and D. Estrin, "Collaborative metric learning," in *Proc. 26th Int. Conf. World Wide Web*, 2017, pp. 193–201.

[46] J. Yu, M. Gao, W. Rong, Y. Song, and Q. Xiong, "A social recommender based on factorization and distance metric learning," *IEEE Access*, vol. 5, pp. 21557–21566, 2017.

[47] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proc. IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct. 1990.

[48] J. McAuley, C. Targett, Q. Shi, and A. van den Hengel, "Image-based recommendations on styles and substitutes," in *Proc. 38th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, 2015, pp. 43–52.

[49] J. J. Levandoski, M. Sarwat, A. Eldawy, and M. F. Mokbel, "LARS: A location-aware recommender system," in *Proc. IEEE 28th Int. Conf. Data Eng.*, Apr. 2012, pp. 450–461.

[50] X. He, T. Chen, M.-Y. Kan, and X. Chen, "TriRank: Review-aware explainable recommendation by modeling aspects," in *Proc. 24th ACM Int. Conf. Inf. Knowl. Manage.*, 2015, pp. 1661–1670.

[51] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. ICLR*, 2014, pp. 1–15.
[52] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, pp. 2579–2605, Nov. 2008.

**NUTTAPONG CHAIRATANAKUL** is currently pursuing the Ph.D. degree with the Tokyo Institute of Technology. His research interests include recommender systems, machine learning, and graph embeddings.

**TSUYOSHI MURATA** is currently an Associate Professor with the Department of Computer Science, Tokyo Institute of Technology. His current research interests include artificial intelligence, especially complex networks, machine learning, and data mining.

**XIN LIU** received the master's degree in computer science from Wuhan University and the Ph.D. degree in computer science from the Tokyo Institute of Technology. He is currently a Research Scientist with the Artificial Intelligence Research Center (AIRC), National Institute of Advanced Industrial Science and Technology (AIST), Japan. His current research interests include graph mining, representation learning, and neural networks.