Recurrent Recommendation with Local Coherence

Jianling Wang and James Caverlee
Department of Computer Science and Engineering, Texas A&M University
{jlwang,caverlee}@tamu.edu

ABSTRACT

We propose a new time-dependent predictive model of user-item ratings centered around local coherence – that is, while both users and items are constantly in flux, within a short-term sequence, the neighborhood of a particular user or item is likely to be coherent. Three unique characteristics of the framework are: (i) it incorporates both implicit and explicit feedbacks by extracting the local coherence hidden in the feedback sequences; (ii) it uses parallel recurrent neural networks to capture the evolution of users and items, resulting in a dual factor recommendation model; and (iii) it combines both coherence-enhanced consistent latent factors and dynamic latent factors to balance short-term changes with long-term trends for improved recommendation. Through experiments on Goodreads and Amazon, we find that the proposed model can outperform state-of-the-art models in predicting users' preferences.

CCS CONCEPTS

• **Information systems** → **Recommender systems**;

KEYWORDS

Recommendation; Neural Networks; Local Coherence

ACM Reference Format:
Jianling Wang and James Caverlee. 2019. Recurrent Recommendation with Local Coherence. In *The Twelfth ACM International Conference on Web Search and Data Mining (WSDM '19), February 11–15, 2019, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3289600.3291024

1 INTRODUCTION

A key challenge for recommender systems is in *predicting* how users will rate items in the future. While many conventional approaches have shown good success in learning static models of users (or items) – e.g., [11, 16, 18, 24, 26, 28, 39] – these models are typically not well-suited to domains where users and items display dynamics in their evolution over time, leading to poor predictive power.

In contrast, many recent efforts have recognized the importance of tracking the dynamics in recommenders, to better capture the evolution of both users and items [21]. For example, a fantasy reader may progress from the young adult *Harry Potter* novels to the grittier *Game of Thrones*. Similarly, items themselves evolve in how they are perceived; for example, movie ratings tend to change with age and after major prizes like the Oscars [15, 34]. In these dynamic recommendation settings, recurrent neural networks (RNNs) have shown great success [6, 25, 34, 37], where the RNN's chain-like neural model can learn the evolution of users and items from their sequences of rating events.

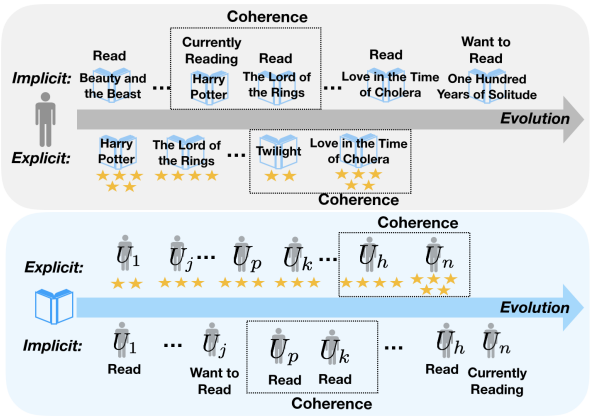

Figure 1: Users (top) and items (bottom) evolve over time, while also displaying local coherence.

While RNNs are well-suited to model the dynamics of user-item preferences, the short-term sequences of user (or item) activities often demonstrate consistency similar to what is modeled by traditional static latent factors (as we explore in Section 2). To illustrate this *local coherence*, consider the Goodreads user in Figure 1 who has provided explicit ratings on similarly themed books *Twilight* and *Love in the Time of Cholera* in a short (locally coherent) time period. This same user has also provided implicit activity traces of "reading" the locally coherent books *Harry Potter* and then *The Lord of the Rings*. Items themselves may also be locally coherent as a burst of interest in a book may lead to similar ratings by a group of similar users in a short-term window. Hence, our goal in this paper is to balance this local coherence with long-term evolution to build better predictive models of user-item ratings.

Considering both viewpoints – that users and items are always evolving, but that they demonstrate local coherence – we propose a new **R**ecurrent **R**ecommendation with **L**ocal **C**oherence (RRLC) framework with the following unique features:

First, we design a sequence-based embedding approach to capture the local coherence for users, items and rating events from both explicit and implicit feedback sequences. Second, we propose to capture the evolution of users and items with parallel recurrent networks, resulting in a dual factor recommendation model. Finally, we show how to integrate coherence-enhanced consistent latent factors and dynamic latent factors to balance short-term changes with long-term trends, resulting in improved predictive capabilities.

Related Work. Many approaches have been developed to capture the temporal dynamics of users and items. As one of the pioneer

WSDM '19, February 11–15, 2019, Melbourne, VIC, Australia

ACM ISBN 978-1-4503-5940-5/19/02…$15.00
https://doi.org/10.1145/3289600.3291024

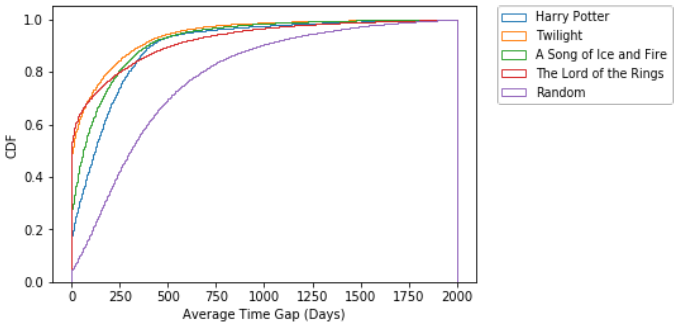

Figure 2: *Local Coherence*: CDF of the Time Gap between consecutive activities on Goodreads for book series vs. random pairs of books. Users are likely to read books from the same series or similar categories in a short time period.

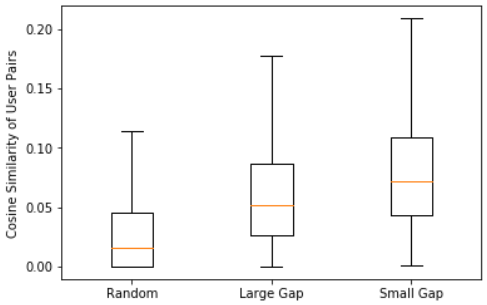

Figure 3: *Local Coherence*: Cosine Similarity for Different User Pairs. Users who rate or read the same book in a shorter time period are more similar.

works, Ding and Li [5] assign different weights for items in Collaborative Filtering to highlight the impact of items rated recently. Baltrunas and Amatriain [1] split a user's profile into several sub-profiles to represent the user in different time periods. Based on the assumption that the taste of users and the perception of items are drifting over time, Time SVD++ [15] extends SVD++ by introducing time-dependent latent factors and bias for both users and items. More recently, in [33], an opportunity model is used to predict the follow-up purchase for a product at a particular time. With the pre-trained latent embeddings of items, Guàrdia-Sebaoun et al. [8] model the movements of users among them. Translation-based recommendation models [10] focus on next-item recommendation and model the transition of users among sequences of implicit feedback events. While these next-item prediction models focus mainly on capturing the continuous long-term evolution, our RRLC model aims to incorporate local coherence as well.

Recently, there is growing interest in applying neural models to recommenders [11, 27, 28]. By introducing nonlinearity, neural-based methods can outperform conventional collaborative filtering methods, plus RNNs can capture the temporal dynamics of time-ordered user activities. For example, there are works done on session-based item recommendation with RNN [12, 25, 29, 31], in which no user-profiles are available and the system needs to infer users' preference with sessions of users' behavior. In addition, Latent Cross [2] has been proposed to incorporate contextual features with RNNs for implicit recommendation over YouTube. RIB [38] models and captures the effect of sequential micro behaviors with RNNs. These works make use of RNNs to capture the changing patterns for users while keeping items static. To uncover the dynamics of both items and users, Recurrent Recommender Network [34] predicts future ratings with separate LSTM models for users and items, in which the ratings are aggregated by time granularity before being fed into the networks. A key question is how to represent users and items at each time step in an RNN as we explore here.

2 PROBLEM SETTING AND CHALLENGES

We consider a recommendation system in which there are a set of N users $\boldsymbol{U} = \{u_1, u_2, ..., u_N\}$ and a set of M items $\boldsymbol{C} = \{i_1, i_2, ..., i_M\}$. Let $r_{u,i}$ denote the rating user u gave to i and $t_{u,i}$ represent the time this rating is placed. Our goal is to predict how user u will rate item i at time t. In this work, we focus on predictive (real-world) scenarios, where models are trained on data on or before a certain timestamp, and then are tested on future data after that timestamp.

To tackle this user-item rating prediction problem, we could rely on matrix factorization (to capture consistency) and recurrent neural networks (to capture dynamics). Here, we first show evidence of local coherence, and then highlight the challenges in these existing models that motivate our proposed approach.

Evidence of Local Coherence. While many models have been proposed to handle long-term dynamics, local coherence is an important factor for recommendation. Here we illustrate with evidence from a sample of 300k users of Goodreads, a long-lasting book reading community with more than 20 billion actions like reviews, ratings, and tags [30]. First, we examine the local coherence in user feedback sequences. We assume that books in the same series are highly similar to each other. In Figure 2, we compare the average time gap of a user leaving implicit feedback for books in the same series and for books in a random sample. We find that the time gap for books in the same series is shorter, meaning that *users are likely to read similar books in a short time period or even sequentially (demonstrating local coherence).* Second, to examine the local coherence of books' feedback sequences, we represent books' status with users who have implicit feedback on them. Thus we compare the similarity of users pairs who have implicit feedback on the same book under different time gaps. The three groups are: (1) "Small Gap" consisting of user pairs in which the users have left implicit feedback to the same item within 30 days. (2) In "Large Gap", each pair consists of the user leaving the first feedback and the user leaving the last feedback on the same item. (3) In "Random", we randomly select users to construct the pairs. We represent each user with a vector listing all of the user's ratings. From Figure 3, we find that users who have read the same book are closer to each other than the random user pairs. Further, *users who rate or read the book in a shorter time period are more similar.* Thus we can conclude that nearby books in a user's feedback sequence or nearby users in a book's feedback sequence are locally coherent.

Matrix Factorization: Consistency. A natural approach to incorporate this local coherence is through Matrix Factorization (MF), which usually identifies consistent latent factors that can be used to represent unchanging user preferences and item characteristics. In neural-based MF models, the latent factors are often derived from

multiplication between a weight matrix and one-hot representation of the element [11, 35]. We denote this embedding operation as $F(\cdot)$. Let $\mathbf{h}_u$ and $\mathbf{h}_i$ denote the one-hot representations of user u and item i, the standard latent factors $\mathbf{F}_u$ and $\mathbf{F}_i$ are determined by:

$$\mathbf{F}_u = F(u) = \mathbf{W}_{User}\mathbf{h}_u \quad \mathbf{F}_i = F(i) = \mathbf{W}_{Item}\mathbf{h}_i \tag{1}$$

in which $\mathbf{W}_{User}$ and $\mathbf{W}_{Item}$ are the embedding weight matrices for users and items correspondingly. With the standard latent factors, in a neural-based General Matrix Factorization (GMF) model [11], the rating $\hat{r}_{iu}$ between u and i is calculated as:

$$\hat{r}_{iu} = a(\mathbf{v}^T(\mathbf{F}_i \circ \mathbf{F}_u)) \tag{2}$$

in which the linear activation function $a(\cdot)$ and the edge vector $\mathbf{v}$ transfer the element-wise product $\mathbf{F}_i \circ \mathbf{F}_u$ to a 1-dimension output.

Challenge. In practice, user-item interactions are often sparse, so many previous works aim to expand traditional MF-based methods to make use of neighboring users or items to overcome this sparsity. For example, TrustSVD [9] expands to similar users in the social network. SVD++ [14] treats items rated by the same user as neighbors. *We propose to make use of the local-coherence hidden in implicit feedback sequences to effectively uncover these* ***neighboring*** *relationships for overcoming sparsity.*

RNN: Dynamics. In contrast to MF, RNNs have shown success in capturing the dynamics of users and items. Each unit in an RNN has a hidden state, which can be passed through the network to persist the information, thus tracking the development of preference in a recommendation system. There are several variants of RNN. In this work, we use Gated Recurrent Unit (GRU) [4], which performs similar to Long Short-Term Memory (LSTM) but is more efficient.

GRU introduces the update gate $\mathbf{z}_t$ and reset gate $\mathbf{r}_t$ to control the long short-term dependencies. Let $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{b}$ denote the corresponding weights and bias of different gate layers. And $g(\cdot)$ denotes the activation function such as sigmoid function. The calculation for each layer of a single unit is shown as below. Let $\mathbf{x}_t$ denote the input at time step t. Firstly, with the similar equation, the update gate $\mathbf{z}_t$ and reset gate $\mathbf{r}_t$ will aggregate the information $\mathbf{h}_{t-1}$ from time $t-1$ and the input at t:

$$\mathbf{z}_t = g_z(\mathbf{W}_z\mathbf{x}_t + \mathbf{U}_z\mathbf{h}_{t-1} + \mathbf{b}_z)$$

$$\mathbf{r}_t = g_r(\mathbf{W}_r\mathbf{x}_t + \mathbf{U}_r\mathbf{h}_{t-1} + \mathbf{b}_r)$$

However, they function differently. The update gate $\mathbf{z}_t$ decides how much information from previous time steps is going to be retained, while the reset gate $\mathbf{r}_t$ determines how much of the previous information is to be forgotten with another hidden state $\tilde{\mathbf{h}}_t$:

$$\tilde{\mathbf{h}}_t = g_h(\mathbf{W}_h\mathbf{x}_t + \mathbf{U}_h(\mathbf{h}_{t-1} \circ \mathbf{r}_t) + \mathbf{b}_h)$$

Lastly, the output $\mathbf{h}_t$ of the GRU at step t is the weighted sum of the current and the last hidden state.

$$\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \circ \tilde{\mathbf{h}}_t \tag{3}$$

To capture the dynamics of both users and items, we can follow the idea in Recurrent Recommender Networks (RRN) [34] of using two separate RNNs. While modeling the dynamics of a user or an item, ratings by the user or ratings for the item can be fed into the RNN to indicate their status at each time step (see Figure 4).

Challenge. We need to efficiently represent the rating events. In RRN, for a dataset with M items, at time step t, it uses a rating vector $\mathbf{x}_t \in R^M$ as input for the given user, where x_{tj} equals to rating for item j at time t or equal to zero for no rating. Input to represent items' status is constructed in a similar way. Since the one-hot representation is in high-dimension but extremely sparse, this method is inefficient and weakly models users (and items). *We propose to make use of the local-coherence within explicit rating sequences to identify effective* ***representations of rating events****.*

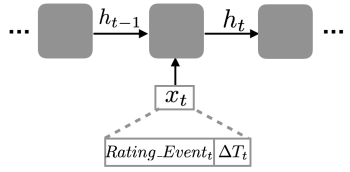

Figure 4: A single unit for RNN. An effective representation of the rating event is needed for the input at each time step.

3 RECURRENT RECOMMENDATION WITH LOCAL COHERENCE

With these challenges faced by existing models, we turn in this section to the design of our **R**ecurrent **R**ecommender with **L**ocal **C**oherence (RRLC). We propose to integrate both implicit and explicit feedback together to improve the user-item rating prediction through careful modeling of local coherence in the following steps:

- First, with local-coherence in implicit feedback sequences, we can uncover neighboring users and items to enrich the consistent latent factors.
- Second, after extracting the local-coherence from explicit feedback sequences, the resulting embeddings can be fed into the parallel RNN models to capture the evolution of users and items.
- Finally, we propose a joint model combining both coherence-enhanced consistent latent factors and dynamic latent factors for improved predictive quality.

3.1 Consistent Latent Factors

First, to identify consistent latent factors, we exploit local coherence to identify neighbors (of users and items).

Coherence-based Neighbors. Based on local coherence, the books a user reads in a short time period or users who rate the same book in a short time period are usually coherent with each other. Hence, we can follow a *word2vec*-style approach [3, 7, 22, 23, 32, 36], where we treat the ordered sequence of implicit feedback of each user as a "sentence" and aim to find the embedding for each item (or "word") in the system, e.g., book, movie. Similarly, we can train on a sequence of implicit feedback for each item to find representations for users. With the vector representations, we will be able to find neighboring items and neighboring users to enhance the general MF model discussed in Section 2.

We follow the skip-gram model as shown in Figure 5. Given a time-ordered sequence of items $\mathbf{I}_u$=$\{i_1, i_2, ...,i_k, ...i_{|\mathbf{I}_u|}\}$ on which user u has left implicit feedbacks, if we set the window size to be j, then the neighboring "words" of i_k will be (i_{k-j},...i_{k-1},i_{k+1}, ..., i_{k+j}). We construct positive pairs between each feedback and their

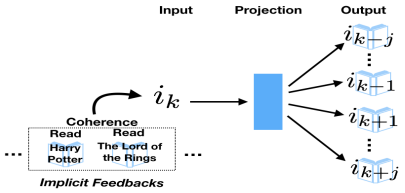

Figure 5: The skip-gram framework for item embedding.

neighbors. And we aim to optimize the log probability for them:

$$l = \sum_{-j \le w \le j, w \ne 0} \log P(i_{k+w}|i_k) \tag{4}$$

in which $P(i_{k+w}|i_k)$ represents the probability that item i_{k+w} and item i_k receive implicit feedback from the same user within a short time window. To help in speeding up the training process, we also construct some negative pairs. Then, for each positive neighboring items pair (i_x, i_y), the log probability is calculated as:

$$P(i_y|i_x) = \log \sigma(\mathbf{e}_{i_y}^T \mathbf{e}_{i_x}) + \sum_{i_{\bar{y}} \in \mathbf{n}} \log \sigma(-\mathbf{e}_{i_{\bar{y}}}^T \mathbf{e}_{i_x}) \tag{5}$$

Here σ represents the Sigmoid function and $\mathbf{n}$ denotes the set of negative items with i_x, which are sampled based on their probability distribution. $\mathbf{e}_{i_x}$, $\mathbf{e}_{i_y}$ and $\mathbf{e}_{i_{\bar{y}}}$ represent their embeddings from the model. We iterate over all the items and "sentences". We can get the representation for each user by training on all the implicit feedback sequences of all items, while getting the representation for each item by training on implicit feedback sequences of all users.

Neighbor-enhanced Consistent Latent Factors. With the embeddings learned by optimizing Equation 5, we will be able to define neighboring users and neighboring items based on cosine similarity. To counter the sparsity in recommendation, we can take the characteristics of the similar users and items into consideration. We denote the set of K_i nearest coherence-based neighbors of item i as $N(i)$ and K_u nearest coherence-based neighbors of user u as $N(u)$. To aggregate the features of the neighbors, we feed the standard latent factors defined with Equation 1 of all items in $N(i)$ and all users in $N(u)$ to the average-pooling layers.

$$\mathbf{F}_i^{neighbor} = \frac{1}{K_i} \sum_{\bar{i} \in N(i)} \mathbf{F}_{\bar{i}} \qquad \mathbf{F}_u^{neighbor} = \frac{1}{K_u} \sum_{\bar{u} \in N(u)} \mathbf{F}_{\bar{u}} \tag{6}$$

The standard latent factors of i and u can be expanded with the aggregated neighbors latent factors $\mathbf{F}_i^{neighbor}$ and $\mathbf{F}_u^{neighbor}$.

$$\tilde{\mathbf{L}_i} = [\mathbf{F}_i \quad \mathbf{F}_i^{neighbor}]^T \qquad \tilde{\mathbf{L}_u} = [\mathbf{F}_u \quad \mathbf{F}_u^{neighbor}]^T \tag{7}$$

Finally, we multiply the concatenations $\tilde{\mathbf{L}_u}$ and $\tilde{\mathbf{L}_i}$ with the affine matrices $\mathbf{A}_{user}^c$ and $\mathbf{A}_{item}^c$ correspondingly to get the neighbor-enhanced consistent latent representations of user u and item j.

$$\mathbf{L}_u^c = \mathbf{A}_{user}^c \tilde{\mathbf{L}_u} \qquad \mathbf{L}_i^c = \mathbf{A}_{item}^c \tilde{\mathbf{L}_i} \tag{8}$$

3.2 Dynamic Latent Factors

To identify dynamic latent factors, we first define a rating embedding based on local coherence.

Rating Embedding with Local Coherence. Different ratings for the same item reveal different levels of preference. And the ratings from different users may have different meanings. Some rate

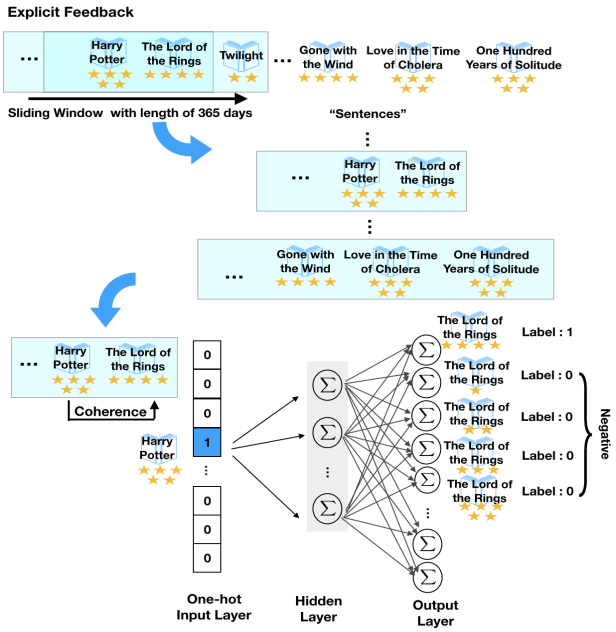

Figure 6: Model of Generating Embeddings for *book:rating*.

Twilight 1 star because they are dissatisfied with the story ending, while others give it 1 star because they dislike vampires. To define the representations for these rating events based on local coherence, similar to Section 3.1, we adopt the idea from *word2vec*, but treat different $book : rating$ or $user : rating$ events as "words", and the explicit feedback sequences as "sentences" (see Figure 6). We make the following modifications on the original *word2vec* algorithm for our rating embeddings.

To maintain the short-term coherence, we truncate the rating history of each user with a sliding window of 365 days. Iterating over all the users, we can get lots of short-term rating sequences, which are treated as "sentences" during training. The resulting embeddings will be representations for different $book : rating$ events.

A neighboring ratings pair reveals both positive and negative information. Given a pair of neighboring ratings, like *Harry Potter : 5 star* and *The Lord of the Rings : 4 star*, we get a positive signal for this pair and also a negative signal between *Harry Potter : 5 star* and *The Lord of the Rings : 1 star*. Thus we add additional negative samples to Equation 5. Then the objective function for the ground truth neighboring ratings pair $(i_x : r_x, i_y : r_y)$ becomes:

$$\log \sigma(\mathbf{e}_{i_y:r_y}^T \mathbf{e}_{i_x:r_x}) + \sum_{\bar{n} \in \mathbf{n}} \log \sigma(-\mathbf{e}_{\bar{n}}^T \mathbf{e}_{i_x:r_x}) + \sum_{\bar{r} \in \mathbf{R}/r_y} \log \sigma(-\mathbf{e}_{i_y:\bar{r}}^T \mathbf{e}_{i_x:r_x}) \tag{9}$$

where $\mathbf{R}$ represents the set of all possible rating levels and $\mathbf{n}$ represents the set of negative $book : rating$ events sampled based on their overall probability distribution. After iterating over all the rating sequences, we can use the resulted embeddings $\mathbf{e}$ from the trained model to represent different $book : rating$ events. Generating embeddings for $user : rating$ events is symmetric and similar.

Dynamic Latent Factors with RNN. By optimizing Equation 9, we will be able to learn the dense representations of rating events

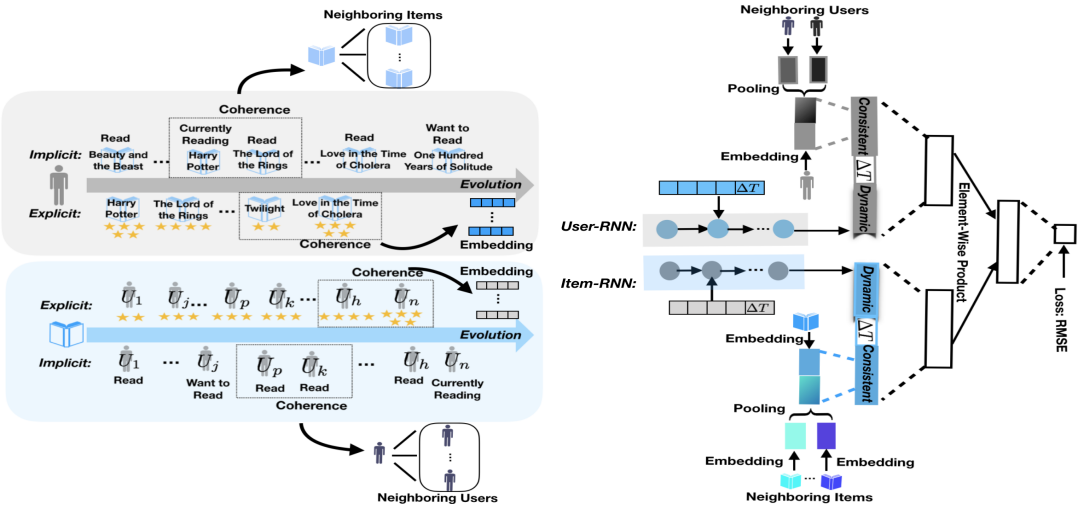

Figure 7: Structure of the Proposed *RRLC* Model. Separate RNNs are used to capture the dynamics of users and items. Local coherence extracted from both the implicit and explicit feedbacks can enrich the neighbor-enhanced consistent latent factors and dynamic latent factors (from RNN). They are combined with the time gap to predict the future ratings.

for the RNN model (in Figure 4). Our task is to predict the rating for item i by user u at time t. Before time t, user u already left a sequence of ratings $\{r_{u,1}, r_{u,2}, ..., r_{u,k}\}$ on items $\{i_1, i_2, ..., i_k\}$ at corresponding timestamps $\{t_{u,1}, t_{u,2}, ..., t_{u,k}\}$. Because the time gap between two ratings can be a signal on how much the user changes, we concatenate the time changing information with coherence-based rating event embedding to construct the input for each GRU unit. The input at time step $t_{u,j}$ will be represented as:

$$\mathbf{x}_{u,j,t} = [\mathbf{e}_{i_j:r_{u,j}} \quad \Delta\mathbf{t}_{u,j}]^T \qquad \Delta t_{u,j} = t_{u,j} - t_{u,j-1}$$

where $\mathbf{e}_{i_j:r_{u,j}}$ is coherence-based embedding for the rating event and $\Delta\mathbf{t}_{u,j}$ is the vector representation for time gap generated by the embedding operation as Equation 1. Output $\mathbf{L}^D_{u,t}$ from user-RNN on the sequence of rating events of user u before t will be treated as the user's most recent dynamic preference by time t.

Then, for the item i, we will feed the rating history on this item before time t to the item-RNN. The output $\mathbf{L}^D_{i,t}$ will be the dynamic characteristics for it by time t. We feed in the rating history for user u and item i in parallel to capture their dynamics separately.

3.3 RRLC

In the previous section, we explained how local-coherence can help us overcome the challenges faced by the MF model and RNN recommendation model. We can combine the coherence-enhanced dynamics latent factors and the consistent latent factors for an improved recommendation system (Figure 7).

Our task is to predict how u will rate i at time t. Based on all rating events before time t, we get $\mathbf{L}^D_{u,t}$ and $\mathbf{L}^D_{i,t}$ representing the most recent dynamic characteristic of user u and item i. We use $\Delta\mathbf{t}_{u,p}$ to represent the gap between t and the last rating of u, which can hint at how much the user has changed from their most recent status. Similarly, $\Delta\mathbf{t}_{i,p'}$ denotes the gap between time t and the last rating on item i to indicate the possible shifting of i. Combining the dynamic latent factors, the neighbor-enhanced consistent latent factors and also ΔT together, the ultimate joint factor vectors of user u and item i are as below:

$$\mathbf{L}_{u,t} = [\mathbf{L}^D_{u,t} \quad \Delta\mathbf{t}_{u,p} \quad \mathbf{L}^c_u]^T \qquad \mathbf{L}_{i,t} = [\mathbf{L}^D_{i,t} \quad \Delta\mathbf{t}_{i,p'} \quad \mathbf{L}^c_i]^T \tag{10}$$

With the latent representations of both the user and the item, we conduct element-wise multiplication to simulate their interaction:

$$\mathbf{M}_{i,u,t} = \mathbf{L}_{i,t} \circ \mathbf{L}_{u,t}$$

Finally, we apply an edge vector $\mathbf{v}$ and the linear activation function $a(\cdot)$ to convert the result to 1-dimension. The predicted rating $\hat{r}_{iu}$ from our **R**ecurrent **R**ecommender with **L**ocal **C**oherence (RRLC) at time t will be calculated as:

$$\hat{r}_{iu} = a(\mathbf{v}^T(\mathbf{L}_{i,t} \circ \mathbf{L}_{u,t}))$$

We use Mean Square Error (MSE) between $\hat{r}_{iu}$ and ground truth rating r_{iu} to calculate the loss and optimize the model.

4 EXPERIMENTS

In this section, we evaluate the quality of the proposed recurrent recommender with local coherence.

4.1 Datasets

We evaluate our models on data (see Table 1) from a book reading platform (Goodreads) and an e-commerce platform (Amazon). To test the predictive power of the model, we must avoid the common random train-test split that can leak information. For example, the training set may contain information that a user rates *Harry Potter Book 7* 5 stars in 2017 and we need to predict how he rated *Harry Potter Book 3* in 2015. Instead, we train the model over training data before a "cutting edge" date. We use the first rating of each user after the edge as validation for model tuning. And then we test the models on all the other ratings after the edge date. The edge date is set to be Jan 1, 2017 for Goodreads and Jan 1, 2013 for Amazon.

Goodreads. We crawl rating and bookshelves data with the official APIs of Goodreads. Ratings in Goodreads range from 1 to 5, which are the explicit signals revealing the preference levels of users and

Dataset	*#Users*	*#Items*	*#Ratings*	*Sparsity*
Goodreads	86,409	179,506	9,696,327	0.0625%
Amazon (Total)	86,426	105,094	2,690,664	0.0296%

Table 1: Overview of Datasets

what we want to predict. The behavior that a user places books onto a bookshelf is similar to tagging the books with the name of the bookshelf. We treat this behavior as a user leaving implicit feedback. There are three default bookshelves of each user named "read", "want to read", and "currently reading". Users can also create new bookshelves and name them. The user IDs in Goodreads are consecutive integer numbers. We randomly select 1 million users and crawl all the feedback information they have left from the date registering to January 2018. We filter out users with fewer than 10 ratings and books rated by fewer than 10 users. We treat the "date updated" as the date they leave the feedback. We keep only users who are active for more than one year, that is the gap between the user's last rating and the first rating should be longer than 365 days.

Amazon. To test whether our model can also be generalized for recommendation settings in e-commerce, we test over an Amazon rating dataset [20] (http://snap.stanford.edu/data/web-Amazon.html), containing about 35 million product reviews from June 1995 to March 2013. For our experiments, we use the ratings (1 to 5) and corresponding timestamps. Following the same settings as in previous work [14, 15], the set of items a user has rated is treated as the set of items the user has left implicit feedbacks on. We delete users who are active for less than one year and left no ratings in 2013. We only keep users who rate more than 10 products and items rated by more than 10 users.

4.2 Setup

Our experiments were conducted on a desktop machine with Intel i7-4829K CPU and a 12 GB Nvidia GeForce Titan XP GPU.

Metrics. We adopt Root Mean Square Error (RMSE) to evaluate our recommender, which is widely used in related work for user-item rating prediction and recommendation [15, 16, 28, 34]. For item and user pairs (i, u) in test set $\boldsymbol{S}$, we denote the ground truth rating as r_{iu} and the predicted rating as $\hat{r_{iu}}$. Then RMSE is calculated as:

$$RMSE = \sqrt{\frac{1}{|\boldsymbol{S}|} \sum_{(i,u)\in \boldsymbol{S}} (r_{iu} - \hat{r_{iu}})^2}$$

Baselines. We compare our final model with the following neural-based MF models.

- *General Matrix Factorization (GMF).* In this basic neural MF model [11], the prediction is generated by Equation 2.
- *Neural Collaborative Filtering (NCF).* This method [11] consists of multi-layer perceptron (MLP) and GMF to model the interaction between users and items. It can take advantage of both deep learning and factorization models.

We also compare RRLC with some time-dependent or RNN-based recommendation models:

- *Time SVD++.* This method [15] extends SVD++ with temporal information by introducing time changing latent factors and bias for both users and items.
- *Session-Aware Recommendation (User-SAR / Item-SAR).* It trains an RNN on sequences of user activities (sessions) [31]. *User-SAR* trains the RNN on users' explicit feedback sequences but treats items as static. In *Item-SAR*, an RNN is trained on items' explicit feedback sequences while users are static. We use the coherence-based rating embeddings to train the RNN.
- *Recurrent Recommender Networks (RRN).* This model [34] trains RNNs for users and items in parallel but both local-coherence and implicit feedback are not considered.

With the simplified variants of RRLC below, we can test the coherence-enriched consistent latent factors and dynamic latent factors individually.

- *GMF with Implicit Coherence (GMF++).* This method enriches *GMF* with local coherence from implicit feedback. Only the neighbor-enhanced consistent latent factors $\mathbf{L}_u^c$ and $\mathbf{L}_i^c$ are considered in Equation 10 for RRLC.
- *Parallel Recurrent Neural Network (P-RNN).* In this model, the prediction is based only on the dynamic latent factors. That is we ignore $\mathbf{L}_u^c$ and $\mathbf{L}_i^c$ in Equation 10 for RRLC.

Parameter Settings. For TimeSVD++, we use the implementation provided in Graphchi [17]. We use grid search for regularization λ over $\{1, 10^{-1}, ...10^{-6}\}$ and latent factor size over $\{20, 50, 100, 150, 200\}$. Each time bin contains 30 days.

For all the *word2vec*-style embedding models (in Section 3.1 and 3.2), we set the window size to be 5 and negative sampling rate to be 5. The dimension for the resulted embedding vectors are all set to be 100 empirically considering the tradeoff between computational complexity and performance.

For neural-based models, we implement them in Python and Keras. We adopt Adam optimizer [13] and mini-batch approach of gradient descent. The batch size for Goodreads and Amazon data is 5120 and 2048. The dropout rate is set to be 0.3 and 0.4 separately. We fine-tuned the regularization parameters in the experiments, and then set it to be $\lambda = 10^{-5}$ for Goodreads data and $\lambda = 10^{-4}$ for Amazon data. For all the RNN-based models, we choose to use GRU. The input of each GRU unit is 50-dimension and the output is also 50-dimension. The sensitivity of timestamps is all set to be 30-days. For fair comparison between the dynamics of users and items, in models with parallel RNNs, we set the maximum length of users and items to be the same.

Impact of Neighbors. With Equation 6, we aggregate the coherence-based neighboring users or items to enhance the matrix factorization. In order to isolate other factors, we test on *GMF++* to check how the neighbors can influence the prediction. In Figure 8, we show the resulted testing RMSE while changing the values of K_u and K_i separately. At the starting points, both K_u and K_i are set to be 1. While increasing the number of neighboring items and users, the errors decrease first and then become relatively stable. In addition, the model is more sensitive to the change of neighboring items. In the following experiments, we take top-10 neighbors while generating consistent latent factors in Equation 7 for both stability and efficiency.

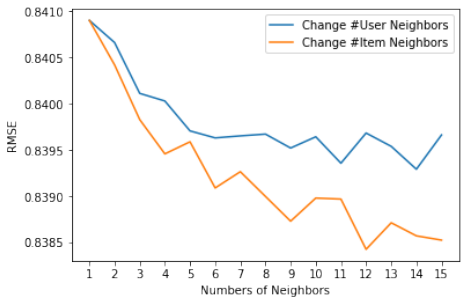

Figure 8: RMSE w.r.t number of neighbors in GMF++. While involving more neighboring items and users in Equation 6, the error decreases and then becomes relatively stable.

	Goodreads	Amazon
GMF	0.8638	1.0716
NCF	0.8461	1.0298
GMF++	0.8365	1.0286
RRN	0.8545	1.0657
TimeSVD++	0.8455	1.0471
User-SAR	0.8295	1.0505
Item-SAR	0.8547	1.0476
P-RNN	0.8209	1.0282
RRLC	**0.8039**	**1.0114**

Table 2: RMSE for different methods

4.3 Evaluating RRLC

We begin our experiments by comparing RRLC with all of the baselines for both Goodreads and Amazon in Table 2. We see that RRLC achieves the best (lowest) RMSE on both datasets. Compared with TimeSVD++, which also takes implicit feedback and temporal information into consideration, our model results in a 4.92% improvement on Goodreads and 3.41% improvement on Amazon. The combination of consistent latent factors (GMF++) and dynamic latent factors (P-RNN) in our model can improve GMF++ by 3.90% and 1.67%, and improve P-RNN by 2.07% and 1.63% for Goodreads and Amazon. This indicates the effectiveness of RRLC's careful integration of both implicit and explicit feedback.

Comparison with GMF, NCF and GMF++. GMF++ extends GMF with local coherence-defined neighbors. Comparing GMF++ with GMF, we can see an improvement of 3.16% and 4.01 % for Goodreads and Amazon. Since besides the particular user and item, their local coherence-defined neighbors are also updated at the same time during training, GMF++ converges faster with fewer iterations. We find that GMF++ can even outperform NCF, which makes use of the MLP component to take advantage of deep learning. In addition, we find that the improvement on the sparser dataset (Amazon) is larger than in Goodreads. Thus we can conclude that the local coherence based on implicit feedback can help to improve the prediction of user-item ratings and it can help overcome the sparsity of the traditional MF models.

Comparison with RRN. Implicit feedback information and local-coherence is not considered in the baseline RRN model. We can see that RRLC brings in a 5.92% improvement for Goodreads and a 5.10% improvement for Amazon comparing with RRN. This shows the importance of applying the rating embedding from local coherence to capture the dynamics of users and items.

Comparison with User-SAR, Item-SAR and P-RNN. All these methods use our local coherence-based embedding as input to RNN. With the promising results from all of them, we find that the rating embedding approach proposed in our RRLC framework can be effectively generalized to other RNN-based recommenders. We find that *User-SAR* works better than *Item-SAR* in Goodreads. But they perform almost the same in Amazon. The reason may be that the users change more frequently than items in Goodreads and thus capturing the dynamic of users is more important. In Amazon, users and items appear to change at a similar rate. P-RNN which treats both users and items as dynamic can improve *User-SAR* and *Item-SAR*. Thus we can conclude that in those communities, it is necessary to model the evolution for both users and items.

4.4 Visualization of Local Coherence

Complementing this evaluation, we visualize here the local coherence and rating embeddings that are important for RRLC.

Coherence-based Neighbors. With the local coherence in implicit feedbacks of users, we get the vector representations for books on Goodreads. The result of the top-100 (most popular in Goodreads) is shown in Figure 9. We can find the series of *Harry Potter* are close to each other on the top-left corner. Classic romantic books fall on the top-right corner.

Rating Embedding. To illustrate the coherence-based rating embedding, we calculate the cosine similarity between rating of 1 star and all the other ratings (2,3,4,5) for the top-100 books in Goodreads. The results are summarized in Figure 10. We find that on average, for a certain item, rating 1 star is close to 2 star but less similar to other ratings, especially 5 star, showing that the embedding (in Section 3.2) learns the difference between extreme ratings.

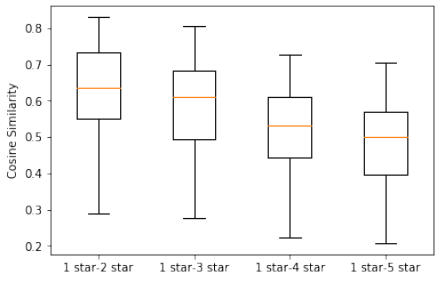

Figure 10: Similarity between different rating levels.

4.5 Time Effect and Dynamics

In this section, we investigate whether more rating events can provide better results or are the recent events enough for us to capture the dynamics of users and items with RNNs? And whether our model captures temporal dynamics well?

Length of Training Records. To examine how the length of history ratings will influence the model training, we change the maximum length of the rating sequence and evaluate the performance of the model. We show the result of our experiment on Goodreads in Figure 11. We find that when we train on longer rating sequences,

Figure 9: 2-D visualization (with t-SNE [19]) of books based on local coherence. Books from the same series or similar categories are close to each other (e.g. *Harry Potter* at the top-left corner, classic romantic books at the top-right corner).

because more information is retained, the RMSE is decreasing. However, if the length is set to be too large, the performance starts to get worse. Because the data is sparse, the majority of users and items of Goodreads have fewer than 100 ratings, so padding the training introduces noise, leading to worse performance. Similar patterns also appears in the Amazon data.

Length of Testing Records. Then we filter out the test data containing users with more than 100 ratings and items with more than 100 ratings in Goodreads. We test our model on this set by varying the length of rating records. From the result in Figure 11, even though both the users and items have more than 100 ratings, RMSE decreases sharply when we vary the length from 10 to 30 and tends to converge after 50. There is similar pattern with the Amazon dataset. It indicates that the dynamic status of users and items are more sensitive to recent ratings.

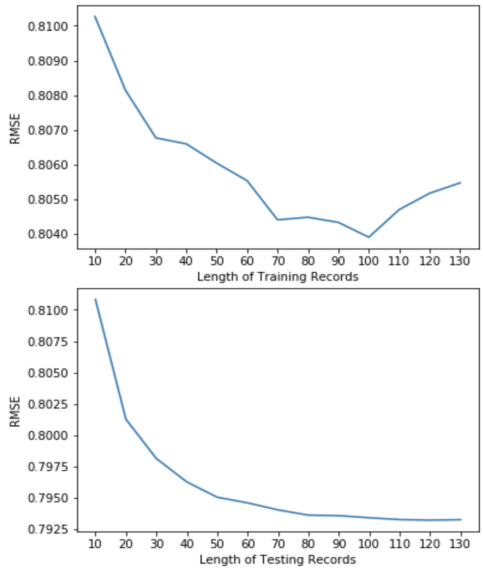

Figure 11: RMSE w.r.t Length of Training or Testing Records

Dynamics in Goodreads. To explore how our model captures the dynamics of users and items, we pick out three books which were adapted into a movie in 2016 and also randomly select 10% of the users. The users may or may not leave any feedback to the movies. We use our model to predict how those users would rate the movies from Jan 2016 to Aug 2018 and the result is summarized in Figure 12. We find that around the time when a book adoption movie was released, the rating for the particular book tends to increase. The reason may be that people were influenced by the movie and were more likely to give higher ratings. However, 2 to 3 months later, the ratings for those books fades as this enthusiasm begins to wane.

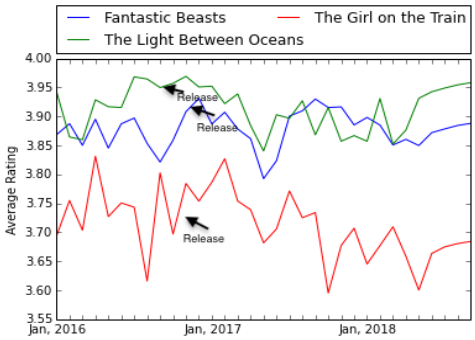

Figure 12: Change of ratings predicted by RRLC for books which are turned into movies. Around the time when the book-adapted movie was released, the rating for the particular book tends to increase. The rating fades 2 to 3 months later as the enthusiasm begins to wane.

4.6 Cold Start

The cold-start problem commonly happens for recommenders in which new users or new items have only limited feedback to train a model. To examine how our model performs in the cold-start scenario, we compare it with NCF and TimeSVD++, which perform

Goodreads	TimeSVD++	NCF	RRLC
New User	0.8683	0.8472	**0.8147**
New Item	0.8792	0.8679	**0.8367**
Amazon	TimeSVD++	NCF	RRLC
New User	1.0920	1.0525	**1.0310**
New Item	1.1244	1.0750	**1.0567**

Table 3: RMSE under Cold-Start Setting.

the best among all the baselines without local-coherence. We filter out users who have fewer than 20 ratings (new users) and items which have been rated by fewer than 20 users (new items) on Goodreads and Amazon. From Table 3, we can see that RRLC still outperforms other methods under the cold-start setting. For new users, our model improves TimeSVD++ by 6.17% and 5.59% on Goodreads and Amazon, which are larger than those under the warm-start setting. For new items, our model can gain a similar improvement. These results illustrate that local coherence can help to keep a robust improvement even in a sparser situation.

5 CONCLUSION

We have proposed and evaluated a new RNN-based recommendater enhanced with local coherence over both implicit and explicit feedback sequences. Based on the experiments on both Goodreads and Amazon, we find that our model outperforms state-of-the-art models in RMSE, and in the cold-start setting. In the future, we will explore social networks in those long-lasting communities to see how users may be influenced by their friends and followees. In addition, we would like to extend our model to support more interaction types and in analyzing review text to track the evolution and consistency of users and items.

ACKNOWLEDGMENTS

This work was supported in part by NSF grant IIS-1841138.

REFERENCES

[1] Linas Baltrunas and Xavier Amatriain. 2009. Towards time-dependant recommendation based on implicit feedback. In *Workshop on context-aware recommender systems (CARS'09)*. Citeseer.

[2] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H Chi. 2018. Latent Cross: Making Use of Context in Recurrent Recommender Systems. In *WSDM*. ACM.

[3] Zhiyong Cheng, Jialie Shen, Lei Zhu, Mohan Kankanhalli, and Liqiang Nie. 2017. Exploiting music play sequence for music recommendation. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. AAAI Press.

[4] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[5] Yi Ding and Xue Li. 2005. Time weight collaborative filtering. In *CIKM*. ACM.

[6] Nan Du, Yichen Wang, Niao He, Jimeng Sun, and Le Song. 2015. Time-sensitive recommendation from recurrent user activities. In *Advances in Neural Information Processing Systems*. 3492–3500.

[7] Mihajlo Grbovic, Vladan Radosavljevic, Nemanja Djuric, Narayan Bhamidipati, Jaikit Savla, Varun Bhagwan, and Doug Sharp. 2015. E-commerce in your inbox: Product recommendations at scale. In *SIGKDD*. ACM.

[8] Elie Guàrdia-Sebaoun, Vincent Guigue, and Patrick Gallinari. 2015. Latent trajectory modeling: A light and efficient way to introduce time in recommender systems. In *RecSys*. ACM.

[9] Guibing Guo, Jie Zhang, and Neil Yorke-Smith. 2015. TrustSVD: Collaborative Filtering with Both the Explicit and Implicit Influence of User Trust and of Item Ratings.. In *AAAI*, Vol. 15. 123–125.

[10] Ruining He, Wang-Cheng Kang, and Julian McAuley. 2017. Translation-based Recommendation. In *RecSys*. ACM.

[11] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 173–182.

[12] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939* (2015).

[13] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[14] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *SIGKDD*. ACM.

[15] Yehuda Koren. 2010. Collaborative filtering with temporal dynamics. *Commun. ACM* 53, 4 (2010), 89–97.

[16] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009).

[17] Aapo Kyrola, Guy E Blelloch, and Carlos Guestrin. 2012. Graphchi: Large-scale graph computation on just a pc. USENIX.

[18] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing* 7, 1 (2003), 76–80.

[19] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.

[20] Julian McAuley and Jure Leskovec. 2013. Hidden factors and hidden topics: understanding rating dimensions with review text. In *RecSys*. ACM.

[21] Julian John McAuley and Jure Leskovec. 2013. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. In *Proceedings of the 22nd international conference on World Wide Web*. ACM, 897–908.

[22] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *Computer Science* (2013).

[23] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[24] Andriy Mnih and Ruslan R Salakhutdinov. 2008. Probabilistic matrix factorization. In *Advances in neural information processing systems*. 1257–1264.

[25] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. 2017. Personalizing Session-based Recommendations with Hierarchical Recurrent Neural Networks. In *RecSys*. ACM.

[26] Steffen Rendle. 2012. Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology (TIST)* 3, 3 (2012), 57.

[27] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. 2007. Restricted Boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*. ACM, 791–798.

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

[29] Yong Kiam Tan, Xinxing Xu, and Yong Liu. 2016. Improved recurrent neural networks for session-based recommendations. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. ACM, 17–22.

[30] Mike Thelwall and Kayvan Kousha. 2017. Goodreads: A social network site for book readers. *Journal of the Association for Information Science and Technology* 68, 4 (2017), 972–983.

[31] Bartłomiej Twardowski. 2016. Modelling contextual information in session-aware recommender systems with neural networks. In *RecSys*. ACM.

[32] Flavian Vasile, Elena Smirnova, and Alexis Conneau. 2016. Meta-Prod2Vec: Product embeddings using side-information for recommendation. In *RecSys*. ACM.

[33] Jian Wang and Yi Zhang. 2013. Opportunity model for e-commerce recommendation: right product; right time. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*. ACM, 303–312.

[34] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. 2017. Recurrent recommender networks. In *WSDM*. ACM.

[35] Yao Wu, Christopher DuBois, Alice X Zheng, and Martin Ester. 2016. Collaborative denoising auto-encoders for top-n recommender systems. In *WSDM*. ACM.

[36] Longqi Yang, Chen Fang, Hailin Jin, Matthew D Hoffman, and Deborah Estrin. 2017. Personalizing Software and Web Services by Integrating Unstructured Application Usage Traces. In *Proceedings of the 26th International Conference on World Wide Web Companion*. International World Wide Web Conferences Steering Committee, 485–493.

[37] Feng Yu, Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan. 2016. A dynamic recurrent model for next basket recommendation. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 729–732.

[38] Meizi Zhou, Zhuoye Ding, Jiliang Tang, and Dawei Yin. 2018. Micro behaviors: A new perspective in e-commerce recommender systems. In *WSDM*. ACM.

[39] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. 2008. Large-scale parallel collaborative filtering for the netflix prize. In *International Conference on Algorithmic Applications in Management*. Springer, 337–348.