Combining User-Based and Session-Based Recommendations with Recurrent Neural Networks: 25th International Conference, ICONIP 2018, Siem Reap, Cambodia, December 13-16, 2018, Procee...

Chapter · January 2018

DOI: 10.1007/978-3-030-04167-0_44

CITATIONS
0

READS
178

3 authors, including:

Tu Minh Phuong
Posts and Telecommunications Institute of Technology
61 PUBLICATIONS **440** CITATIONS
SEE PROFILE

Ngo Xuan Bach
Posts and Telecommunications Institute of Technology
33 PUBLICATIONS **160** CITATIONS
SEE PROFILE

Combining User-based and Session-based Recommendations with Recurrent Neural Networks

Tu Minh Phuong[1], Tran Cong Thanh[1], and Ngo Xuan Bach[1,2]

[1] Posts and Telecommunications Institute of Technology, Hanoi, Vietnam
[2] FPT Software, Hanoi, Vietnam
phuongtm@ptit.edu.vn,thanh.ptit.96@gmail.com,bachnx@ptit.edu.vn

Abstract. Recommender systems generate recommendations based on user profiles, which consist of past interactions of users with items. When user profiles are not available, session-based recommendation can be used instead to make predictions based on sequences of user clicks within short sessions. Although each approach can be used separately, it is desired to utilize both user profiles and session information, and other information such as context, when those are available. In this paper, we propose a Recurrent Neural Networks (RNNs) based method that combines different types of information to generate recommendations. Specifically, we learn user and item representations from user-item interaction data and explore a new type of RNN cells to combine global user embeddings with sequential behavior within each session to generate next item recommendations. The proposed model uses an attention mechanism to adaptively regulate the contributions of different input components based on specific situations. The model can be extended to incorporate other input, such as contextual information. Experimental results on two real-world datasets show that our method outperforms state-of-the-art baselines that use only user or session information.

Keywords: Recommender systems, Session-based recommendation, Recurrent neural networks, Next item recommendation.

1 Introduction

Traditional recommendation algorithms rely on user historic data to generate personalized recommendations about products or items that a user is likely interested in [2]. Collaborative filtering (CF) [8], the most widely used recommendation approach, uses past user-item interactions to create a low rank representation for each user which is then matched against item representations to calculate recommendation scores (the model-based CF [11]) or to find users with similar preferences (the neighborhood-based CF [16]). The CF based recommender systems have proved useful as they can provide accurate and personalized recommendations in many domains.

An important condition for CF algorithms to work is the availability of informative user profiles, which implies that each user has enough interactions recorded and the user identity is visible for each interaction event. In many online systems such as e-

commerce websites or music online service, this condition is not satisfied, either because users are not required to authenticate and/or they are newcomers. To make predictions in such settings session-based recommendation has recently been proposed [10]. A session is a sequence of user interactions (such as clicks) that occur within a given time frame and are separated from other sessions by time intervals of sufficient length. Examples of sessions are a sequence of items clicked by a user on an e-commerce website within a day or a list of songs a user listens to in an evening. Using sequential patterns of session events, session-based systems can produce relatively accurate recommendations, outperforming other approaches such as item-based methods in many domains [7][10]. However, because users are assumed to be anonymous, session-based recommendations cannot take into account user-specific preferences and therefore are not personalized.

In practice, there are many cases where the systems have access to both session information and user identifiers, for example via explicit authentication or other identifiers such as mobile device IDs. In these settings, using user-only or session-only methods does not allow utilizing all available information, which may be valuable to predict user intents. Specifically, session-based methods cannot model global patterns of user preferences, which are preserved from session to session, while CF methods cannot take into account the user intent specific for each session and the sequential nature of consuming certain types of items such as movie series. Simple modifications, for example, by concatenating sessions belonging to the same user, still ignore part of information and do not give the best results [14].

In this paper, we propose a method to combine user-based CF with session-based recommendation based on RNNs. The method learns representations for users, items, and other context information (if available). These representations are passed to RNNs that model sequential events within sessions. To account for importance of different information sources that vary from situation to situation we explore a new type of RNN cells with an attention mechanism. This makes the method easily extendable to incorporate additional context information, and able to deal with situations when the user is new and does not have enough past interactions. We describe a procedure to prepare training data, which is specially designed to take more training signals from session data.

We evaluate the proposed method on two real-world datasets: one is a public dataset in the music domain, and one proprietary dataset from a video-on-demand domain. The experimental results show clear superiority of our method over plain session-based algorithms, CF algorithms, and a model using concatenated sessions.

2 Related Work

Collaborative filtering. Collaborative filtering (CF) methods such as matrix factorization [11] and neighborhood-based methods [5][8] are widely used for building recommender systems based on user-item interaction data. Matrix factorization decomposes the sparse matrix of user-item interactions to represent users and items by latent factor vectors. The recommendation problem is then considered as a matrix comple-

tion task, where a missing value is the inner product of the respective user and item vectors. Neighborhood based methods utilize user-item interactions to compute the similarities between users or between items. A neighborhood of an active user (in used-based recommendations) or item neighborhoods (in item-based recommendations) are then exploited to generate recommendations. Content features can also be included to deal with the new user problem [13], or different methods can be combined to reduce the negative effect of data sparseness [1].

Recommendations for sequential data. Conventional CF methods can not model the temporal dynamics that naturally exist in user behavior. For instance, series movies are typically consumed one episode after another. Therefore, several methods have been proposed to explicitly incorporate temporal information when generating recommendations, which adapt sequence-based methods from other domains. Among methods of this kind, Markov models [7][15] and RNNs [6] based recommendations are the most successful. By learning a transition graph over actions of users, Markov chains [7] can model sequential behavior and predict the next action based on the last ones. Donkers et al. [6] present a RNN architecture to model user-item interactions arranged in sequences. They introduce a new type of Gated Recurrent Unit cells that combine user and item vectors for updating RNN hidden states. Our method is similar to [6] with two main differences: 1) we propose a new type of attentive gate which requires less parameters but be easily extended to incorporate more input components; and 2) we adopt a training procedure specially designed to work with session data.

Session-based recommendations. Emerged recently, this kind of systems has attracted a great research interest because session-based settings are common in practice. The pioneer work by Hidasi et al. [10] is the first introduction to session-based recommendation which presents a nice application of RNNs for modeling short sessions of user interactions. Since then, RNNs remain the most popular architectures for systems of this kind but with a number of improvements and modifications. Tan et al. [17] present an improved procedure for training data augmentation and selection. Hidasi et al. [9] extend RNNs to incorporate content features of session events. Li et al. [12] propose a mechanism to take into account the user's purpose within a session. Quadrana et al. [14] propose a hierarchical architecture to incorporate user identifiers when those are available. A deviation from the mainstream is the work by Tuan et al. [18], who propose a Convolutional Neural Networks (CNNs) architecture to model both item content and session events. The use of CNNs has two advantages: CNNs are very powerful tool to represent content features while are faster and easier to train than RNNs. Our method in this work is similar to [15] in that it takes into account user identifiers but out model can shift the focus between user and item component depending on a specific session event.

3 Methods

In this section we describe the proposed RNNs based model for combining user-related and session information. We also explain the training procedure and how training data are prepared.

In a typical session-based setting, the system takes as input current session events in forms of clicked items' IDs up to the current time point and outputs, for each available item, a score indicating how likely the item will be clicked next. The top-scored items are then recommended to the user. We consider the settings in which each session is also associated with a user's ID, and possibly the timestamp of each event. Formally, let $[e_1, e_2, ..., e_L] (L \geq 2)$ denote a session of L clicks by user u where $e_i \in P$ $(1 \leq i \leq L)$ is the index of the clicked item (we omit index u from e_i to simplify the notation). Here P denotes the set of all items. For any prefix of the session $[e_1, e_2, \ldots, e_l]$ $(1 \leq l < L)$, the recommendation task is to predict the next click by estimating, for each item $p \in P$, a recommendation score y_i indicating the likelihood of being clicked next. In many cases, we are also given context times of the clicks, which are denoted by $[c_1, c_2, \ldots, c_L]$. We seek to find a model able to take into account all this information when making recommendations.

Fig. 1 shows the general architecture of the proposed method, which consists of two main components: an embedding layer for the explicit representation of users, items, and time context, and an RNN to model click sequences. We propose a new type of RNN cells: Gated Recurrent Units (GRUs) with an attention mechanism, which we call *Attentive GRU* (AGRU), to adaptively combine different types of input signals. We describe the proposed model in more detail in the subsequent sections.

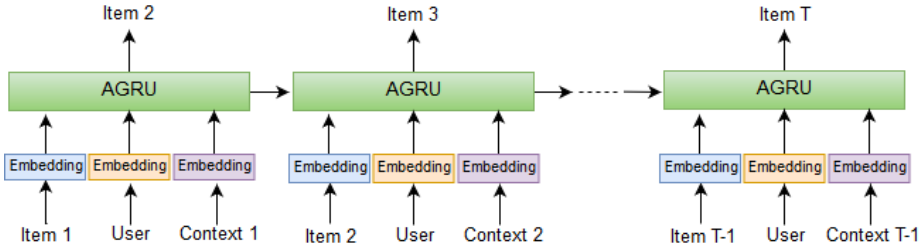

Fig. 1. General architecture of the proposed method. AGRU is GRU with additional attentive gate to regulate the contributions of input components.

3.1 User and Item Embedding

Existing session-based recommendation RNN architectures mainly use one-hot vector to represent input to RNNs [10][14]. For the recommender system domain, however, it has been proven useful to embed users and items in spaces of lower dimensions so that users with similar tastes are close in the embedded space [11][19]. Similarly, items appeared in the same context or consumed by the same set of users are also spatially close in the embedded space. For a user performing multiple sessions, the embedding provides a convenient way to represent global preference structures across sessions.

Let $\mathbf{u}_O \in \Re^{|U|}$, $\mathbf{p}_O \in \Re^{|P|}$, and $\mathbf{c}_O \in \Re^{|C|}$ be the one-hot vector representations of user u, item p and context c respectively, where U, P, and C denote the sets of users, items and contexts. We consider embedding matrices $\mathbf{E}_U \in \Re^{n_U \times |U|}$, $\mathbf{E}_P \in \Re^{n_P \times |P|}$, and $\mathbf{E}_C \in \Re^{n_C \times |C|}$ that map corresponding one-hot representations into embedded

spaces of lower dimensions. With the introduced matrices, the user vector in the embedded space is calculated as $\mathbf{u} = \mathbf{E}_U \mathbf{u}_O$. Similarly, the item and the context are represented as embedded vectors $\mathbf{p} = \mathbf{E}_P \mathbf{p}_O$ and $\mathbf{c} = \mathbf{E}_C \mathbf{c}_O$. The embedding matrices are randomly initialized and jointly learned during the training process.

3.2 RNNs with Modified Gated Recurrent Units

RNNs are a class of deep neural networks specially designed to model sequences of any kind. An RNN operates on a sequence of variable length $\mathbf{x} = (x_1, x_2, \ldots, x_L)$ by updating a hidden state **h** and optionally produces output **y** over **x**. Here, we consider the variant of RNNs which outputs the next item at each time step *t*:

$$\mathbf{y}_t = \sigma_y(\mathbf{W}_y \mathbf{h}_t + \mathbf{b}_y), \tag{1}$$

where $\mathbf{W}_y$ and $\mathbf{b}_y$ by are parameter matrix and vector which are learned, σ_y is the softmax function. The hidden state is updated by a function of the previous hidden state and the current input vector $\mathbf{x}_t$ as follows:

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t), \tag{2}$$

where *f* is a non-linear activation function. Here, as function *f* we consider Gated Recurrent Units (GRUs) [4], which updates the hidden state as follows:

$$\mathbf{z}_t = \sigma_g(\mathbf{W}_z \mathbf{x}_t + \mathbf{V}_z \mathbf{h}_{t-1} + \mathbf{b}_z) \tag{3}$$

$$\mathbf{r}_t = \sigma_g(\mathbf{W}_r \mathbf{x}_t + \mathbf{V}_r \mathbf{h}_{t-1} + \mathbf{b}_r) \tag{4}$$

$$\mathbf{h}_t = \mathbf{z}_t \circ \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \circ \sigma_h(\mathbf{W}_h \mathbf{x}_t + \mathbf{V}_h(\mathbf{r}_t \circ \mathbf{h}_{t-1}) + \mathbf{b}_h) \tag{5}$$

where $\circ$ denotes element-wise multiplication, σ_g and σ_h are sigmoid and tangent functions respectively, **W**, **V**, and **b** are parameter matrices and vectors.

In the pure session-based settings where only item ID is considered the input vector $\mathbf{x}_t$ is the item vector $\mathbf{p}_t$ at step *t* ($\mathbf{x}_t = \mathbf{p}_t$). With user and context vectors $\mathbf{u}_t$ and $\mathbf{c}_t$ available, we now extend the gated unit to incorporate this information when updating the hidden state. In this work, we consider two ways of integrating user and context into the model.

Simple Integration. The most simple and straightforward way is to concatenate user, item, and context vectors and use the resulting vector as the input to the recurrent unit:

$$\mathbf{x}_t = \begin{bmatrix} \mathbf{u}_t \\ \mathbf{p}_t \\ \mathbf{c}_t \end{bmatrix} \tag{6}$$

Since user and context now become parts of the input, they can influence the computation of the hidden state by regulating the amount of information to be forgotten or

updated. Once weight matrices are learned, the importance of each input component on the output becomes fixed and remains the same regardless of changing sessions, which is not desired in many cases and makes this integration method not flexible.

Attentive Integration. Depending on concrete situations, factors influencing the user's decision may vary. For example, movie series are often watched in their order regardless of user taste, or Christmas songs are more frequently listened in the context of Christmas season. Thus, a desired characteristic of the system is the ability to adaptively change the focus between the input components depending on specific situations. To achieve this, we propose a modified version of gated cells, in which an attentive gate is introduced to regulate the contributions of the user, item, and context factors. Specifically, at each time step, the attentive gate computes the following weights:

$$a_i = \sigma_a(\mathbf{v}_i^T \mathbf{d}_i + b_i), \quad i = 1,2,3 \tag{7}$$

$$\alpha_i = \frac{\exp(a_i)}{\sum_i \exp(a_i)}, \quad i = 1,2,3 \tag{8}$$

where $\mathbf{d}_i$ $(i = 1,2,3)$ denote $\mathbf{u}_t, \mathbf{p}_t,$ and $\mathbf{c}_t$ respectively, α_i $(i = 1,2,3)$ are their importance weights; $\mathbf{v}_i$ and b_i are parameters to be learned; σ_a is the sigmoid function. Here, we measure the importance of each component (user, item, context) as the similarity between its embeddings and a vector $\mathbf{v}_i$. We use the sigmoid as a squashing function and get normalized attention weights through a softmax. Note that the user component is fixed for the whole session so we need to compute a_1 only once. Finally, we form the input vector concatenating user, item, and context vectors multiplied by corresponding weights:

$$\mathbf{x}_t = \begin{bmatrix} \alpha_1 \mathbf{u}_t \\ \alpha_2 \mathbf{p}_t \\ \alpha_3 \mathbf{c}_t \end{bmatrix} \tag{9}$$

In this way, the attentive gate controls the contribution of each input component to the computation of the current state and output. Fig. 2 shows graphical depiction of the proposed gated units. The attentive gate not only allows regulating the extents to which each component influences the decision but also provides a nice way to deal with the new user problem. A new user is simply represented by a randomized vector with elements close to zeros and the gate will ignore the user component by giving it small weight, thus focusing on session and context information. It is also easy to add different types of context information by considering them as additional input components and apply the same integration method.

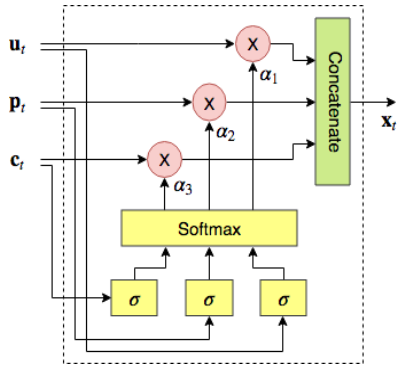

Fig. 2. The attentive gate detached from the complete unit.

3.3 Training

We train our model using standard stochastic gradient descent with Back-Propagation-Through-Time (BPTT) to minimize a chosen loss function. Here we use cross entropy loss between the predicted vector and the one-hot representation of the next clicked item. Note that user embeddings are updated with each click during training process while they are kept fixed throughout the session at prediction time.

To make the input and output of the model similar at the training and prediction times, we prepare training data following the preprocessing procedures described in [17][18] but with a modification to include user identifiers. Specifically, for a given training session $[e_1, e_2, \ldots, e_T]$ we generate input sequences and corresponding labels $([e_1], e_2)$, $([e_1, e_2], e_3)$,..., $([e_1, e_2, \ldots, e_{T-1}], e_T)$ as training samples. We order training sequences by length and user identifiers so that each mini-batch consists of training sequences of the same size but originating from different users. Grouping equal-size sequences together has been widely used in practice as this speeds up training, while distributing users across mini-batches introduces more diversity within a single batch.

4 Experiments

In this section we describe the experiments we performed to evaluate the proposed method and provide an analysis of the results.

4.1 Datasets

We evaluated the proposed method and baselines on the following datasets:

- *Last.fm 1K Users*[1]. This dataset contains music listening history data for 992 users collected by Celma [3] using Last.fm API. The dataset has a total of 19150868 tuples of the form user-timestamp-artist-song. For computational

[1] http://www.dtic.upf.edu/~ocelma/MusicRecommendationDataset/

reasons, we used 20% subsample of data in our experiments, which resembles the setting in other works [6]. From a timestamp we extracted two values to use as the context: day-of-week with values ranging from 1 to 7, and half-month-of-year with values ranging from 1 to 24.

- *VOD*. This is a proprietary dataset collected from a video-on-demand service over a period of two months. The dataset contains 7945218 video-viewing events from 749K users. Events lasting shorter than a fixed threshold (normalized by video length) were not included. We did not use timestamp for this dataset because the time period is too short and thus is not informative.

We performed the following preprocessing on both datasets. First, based on timestamps provided we partitioned the data into sessions with a one-hour idle threshold. Sessions with more than 20 interaction events are further partitioned into session of 20 events with 5-event overlapping. Then, we removed sessions with only one event since those are too short and not informative, and removed users having less than five sessions to provide enough user-related information across sessions. We also removed items appearing less then 10 times as such cannot be modeled reliably. The statistics of two datasets are summarized in Table 1.

For each user, we used the last session to build the training set and the second last session to build the evaluation set that we used to tune model parameters. The remaining sessions form the training set.

Table 1. Statistics of the datasets

	Last.fm	VOD
Users	767	18,775
Items	107,138	61,335
Sessions	203,869	348,172
Events	4,258,811	4,209,416
Average session length	20,89	12,09
Sessions per user	265,8	18,54

4.2 Experimental Setup

Metrics. We consider a typical setting, in which the system returns top K items as recommendations and used Recall@K and Mean Reciprocal Rank (MRR)@K with $K = 5, 20$ as the evaluation metrics.

Parameter Tuning. We varied the embedding dimensions between 50 and 150 but kept them equal for users and items for simplicity (although they can be different in general). The embedding dimensions of time context were fixed at 5. We evaluated our models with 100 and 200-dimensional GRUs. The models were optimized by using standard SGD, with mini-batch size of 64, for 20 epochs with the training rate

of 0.001. We implemented and trained our models in TensorFlow. For training sequences longer than 2, we used item dropout with probability of 25% as described in [17].

Baselines. We compared our method with a general recommendation method (Item-based kNN), a for sequential data TribeFlow), and two RNN session-based methods.

- *Item-based kNN*. The first, and very competitive baseline is Item-based kNN. This is a simple, yet effective method, which is often considered industry standard. Item-based kNN measures the similarity between two items as the number of times they co-occur in sessions divided by a normalization factor.
- *TribeFlow*: This method by [7] is designed to handle sequential data by applying a semi-Markov random walk model on short fragments generated from sequences of user activities. TribeFlow was chosen as a baseline based on its reported good performance compared with other sequential recommendation methods.
- *Ses RNN*. We refer to the model proposed in [10] as Ses RNN. This model uses RNNs with standard GRUs to model clicks; no user information is used. We used the implementation provided by the method's authors. We kept all default parameter values with the only exception of GRU size set to 100 for a fair comparison with other RNN models.
- *Improved Ses RNN*. This is an improved version of Ses RNN by [17], in which a number of modifications including data augmentation and adaptation for shifts over time have been applied to improve the prediction accuracy. We used the provided implementation with all parameters set to the default values.
- *User-Ses GRU*: This is our GRU model with simple integration of user and item vectors.
- *User-Ses AGRU*: This is our GRU model that uses attentive gate to combine user, item, and context vectors.

4.3 Results

In the first experiment we compared the performance of our models with embedding and GRU size set to 100, which are default GRU size for other RNN-based baselines. The Recall@K and MRR@K values for each model on Last.fm and VOD datasets are summarized in Tables 2 and 3, respectively. Surprisingly, TribeFlow consistently performed the worst over different evaluation metrics and datasets, but was still able to correctly place nearly 27% of the next items on the top 20 for both datasets. Simple Item-based kNN achieved substantially higher Recall@K and MRR@K values than TribeFlow, but performed worse than most RNN based methods. The superiority of RNN methods over TribeFlow and Item-based KNN may come from the use of session information and modeling power of the RNN models. Our simple GRU model (User-Ses GRU) achieved better results than Ses RNN but much worse results than Improved Ses RNN, suggesting that simple integration of user factor may not give the best results. The results show that User-Ses AGRU consistently achieved strong performance, outperforming other methods on all metrics over both datasets. The only

exception is MRR@20 on VOD, where User-Ses AGRU achieved a slightly lower value than Improved RNN.

Table 2. Recall@K and MRR@K values of the baselines and our models on Last.fm dataset

	Recall@5	Recall@20	MRR@5	MRR@20
Item-based kNN	0.165	0.317	0.104	0.122
TribeFlow	0.067	0.269	0.023	0.1
Ses RNN	0.266	0.295	0.238	0.241
Improved Ses RNN	0.287	0.339	0.241	0.246
User-Ses GRU	0.271	0.303	0.232	0.238
User-Ses AGRU	**0.325**	**0.384**	**0.286**	**0.292**

Table 3. Recall@K and MRR@K values of the baselines and our models on VOD dataset

	Recall@5	Recall@20	MRR@5	MRR@20
Item-based kNN	0.277	0.497	0.184	0.209
TribeFlow	0.057	0.268	0.019	0.099
Ses RNN	0.25	0.512	0.168	0.197
Improved Ses RNN	0.339	0.555	0.209	**0.244**
User-Ses GRU	0.298	0.508	0.180	0.204
User-Ses AGRU	**0.359**	**0.585**	**0.221**	0.242

Table 4. Performance of User-Ses AGRU for different GRU and embedding sizes

GRU size	Embedding size	Last.fm		VOD	
		Recall@20	MRR@20	Recall@20	MRR@20
100	50	0.371	0.276	0.564	0.232
100	100	0.384	0.292	0.585	0.242
100	150	0.391	0.301	0.589	0.248
200	50	0.374	0.289	0.545	0.227
200	100	0.388	0.294	0.565	0.242
200	150	0.393	0.306	0.570	0.243

Two important parameters that influence the performance of RNN based models are the embedding and GRU sizes. We varied embedding size from 50 to 150 (embeddings of higher dimensions were not experimented due to high memory requirements for Last.fm dataset), in combination with GRU size of 100 and 200. Results are summarized in Table 4. As can be seen, increasing GRU size from 100 to 200 does not always yield better performance: GRU models with 200 dimensions just slightly improved Recall on Last.fm while achieved lower Recall on VOD. This result is con-

sistent with those reported in [17]. On the other side, higher embedding sizes resulted in better results. This effect is more significant when moving from 50 to 100. Further increase to 200 resulted in small improvements, suggesting that embedding size of 100 is a good tradeoff between accuracy and computational efficiency.

5 Conclusion

We have presented a method for incorporating user and context factor into session-based recommendation. By extending the GRU architecture with an attentive gate that operated on the embeddings of users and items, the proposed method allows adaptively combining inputs of different types by shifting focus between user, item, and context factors. The model is easy extended to incorporate new type of context conditions. Experimental results show that our method achieved better performance in terms of Recall and MRR metrics on two real-world datasets.

Acknowledgement. We would like to thank FPT for financial support, which made this work possible

References

1. Bach, N.X., Hai, N.D., Phuong, T.M.: Personalized recommendation of stories for commenting in forum-based social media. Information Sciences, 352-353, pp. 48–60 (2016)
2. Bobadilla, J., Ortega, F., Hernando, A., Gutiérrez, A.: Recommender systems survey. Knowledge-Based Systems, 46, pp. 109–132 (2013)
3. Celma, O.: Music Recommendation and Discovery in the Long Tail. Springer (2010)
4. Cho, K., Merrienboer, B.V., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning Phrase Representations using RNN Encoder--Decoder for Statistical Machine Translation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1724–1734 (2014)
5. Deshpande, M., Karypis, G.: Item-based top-N recommendation algorithms. ACM Transactions on Information Systems (TOIS), 22(1), pp. 143–177 (2004)
6. Donkers, T., Loepp, B., Ziegler, J.: Sequential User-based Recurrent Neural Network Recommendations. In: Proceedings of the Eleventh ACM Conference on Recommender Systems (RecSys), pp. 152–160 (2017)
7. Figueiredo, F., Ribeiro, B., Almeida, J.M., Faloutsos, C.: TribeFlow: Mining & Predicting User Trajectories. In: Proceedings of the 25th International Conference on World Wide Web (WWW), pp. 695–706 (2016)
8. Goldberg, D., Nichols, D., Oki, B.M., Terry, D.: Using Collaborative Filtering to Weave an Information Tapestry. Communications of the ACM, 35(12), pp. 61–70 (1992)
9. Hidasi, B. et al.: Parallel Recurrent Neural Network Architectures for Feature-rich Session-based Recommendations. In: Proceedings of the 10th ACM Conference on Recommender Systems (RecSys), pp. 241–248 (2016).

10. Hidasi, B., Karatzoglou, A., Baltrunas, L., Tikk, D.: Session-based recommendations with recurrent neural networks. In: Proceedings of the International Conference on Learning Representations (ICLR), (2016)
11. Koren, Y., Bell, R., Volinsky, C.: Matrix Factorization Techniques for Recommender Systems. Computer, 42(8), pp. 30–37 (2009)
12. Li, J., Ren, P., Chen, Z., Ren, Z., Lian, T., Ma, J.: Neural Attentive Session-based Recommendation. In: Proceedings of the 2017 ACM Conference on Information and Knowledge Management (CIKM), pp. 1419–1428 (2017)
13. Phuong, N.D., Thang, L.Q., Phuong, T.M.: A Graph-Based Method for Combining Collaborative and Content-Based Filtering. In: Proceedings of the Tenth Pacific Rim International Conference on Artificial Intelligence (PRICAI), pp. 859–869 (2008)
14. Quadrana, M., Karatzoglou, A., Hidasi, B., Cremonesi, P.: Personalizing Session-based Recommendations with Hierarchical Recurrent Neural Networks. In: Proceedings of the Eleventh ACM Conference on Recommender Systems (RecSys), pp. 130–137 (2017)
15. Rendle, S., Freudenthaler, C., Schmidt-Thieme, L.: Factorizing Personalized Markov Chains for Next-basket Recommendation. In: Proceedings of the 19th International Conference on World Wide Web (WWW), pp. 811–820 (2010)
16. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based Collaborative Filtering Recommendation Algorithms. In: Proceedings of the 10th International Conference on World Wide Web (WWW), pp. 285–295 (2001)
17. Tan, Y.K., Xu, X., Liu, Y.: Improved Recurrent Neural Networks for Session-based Recommendations. In: Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, pp. 17–22 (2016)
18. Tuan, T.X., Phuong, T.M.: 3D Convolutional Networks for Session-based Recommendation with Content Features. In: Proceedings of the Eleventh ACM Conference on Recommender Systems (RecSys), pp. 138–146 (2017)
19. Vasile, F., Smirnova, E., Conneau, A.: Meta-Prod2Vec: Product Embeddings Using Side-Information for Recommendation. In: Proceedings of the 10th ACM Conference on Recommender Systems (RecSys), pp. 225–232 (2016)