Sequential User-based Recurrent Neural Network Recommendations

Tim Donkers
University of Duisburg-Essen
Duisburg, Germany
tim.donkers@uni-due.de

Benedikt Loepp
University of Duisburg-Essen
Duisburg, Germany
benedikt.loepp@uni-due.de

Jürgen Ziegler
University of Duisburg-Essen
Duisburg, Germany
juergen.ziegler@uni-due.de

ABSTRACT

Recurrent Neural Networks are powerful tools for modeling sequences. They are flexibly extensible and can incorporate various kinds of information including temporal order. These properties make them well suited for generating sequential recommendations. In this paper, we extend Recurrent Neural Networks by considering unique characteristics of the Recommender Systems domain. One of these characteristics is the explicit notion of the user recommendations are specifically generated for. We show how individual users can be represented in addition to sequences of consumed items in a new type of Gated Recurrent Unit to effectively produce personalized next item recommendations. Offline experiments on two real-world datasets indicate that our extensions clearly improve objective performance when compared to state-of-the-art recommender algorithms and to a conventional Recurrent Neural Network.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; • **Computing methodologies** → **Artificial intelligence**;

KEYWORDS

Recommender Systems, Deep Learning, Neural Networks, Recurrent Neural Networks, Sequential Recommendations

ACM Reference format:
Tim Donkers, Benedikt Loepp, and Jürgen Ziegler. 2017. Sequential User-based Recurrent Neural Network Recommendations. In *Proceedings of RecSys '17, Como, Italy, August 27-31, 2017,* 9 pages.
https://doi.org/10.1145/3109859.3109877

1 INTRODUCTION

The majority of today's *Recommender Systems* (RS) [41] relies on algorithms that are designed under the assumption that user preferences are static patterns [29, 50]. *Collaborative Filtering* (CF) [28] approaches, both neighborhood- and model-based, have proved immensely useful without any consideration of time. However, assuming consumption events to be independent from each other precludes taking advantage of the temporal dynamics that naturally exist in user behavior [50]. Thus, despite the practical success of time-agnostic models, it seems promising to investigate whether the inclusion of temporal aspects can improve effectiveness. For instance, many commercial goods are only bought during a specific season. Music songs are played in succession according to, among others, the user's current mood or the desire for diversity, and are often arranged in playlists. Fictional series are typically consumed one episode after another, before any other content is considered. In contrast to the conception prevailing in many RS, individual data points used in temporal or sequential recommendations thus cannot be assumed independent and uniformly distributed, as they typically form correlated sequences. Several ways exist to approach this issue, e.g. by extending conventional time-independent algorithms [e.g. 7, 27], integrating time as a contextual factor [e.g. 22, 24], or by applying sequence-based methods primarily used in other domains [e.g. 42]. Still, many approaches ignore that generating recommendations is an inherently time-dependent task and focus only on achieving increasingly accurate predictions.

Compared to other areas, sequence modeling has overall been less explored in RS research due to the long-lasting focus on time-agnostic models. This gap may also be explained by the unique nature of the recommendation problem. Although heavily influenced by Machine Learning [41], many modern techniques successfully applied in other domains cannot easily be transfered to RS. This is particularly true for techniques from a research field currently attracting much attention, *Deep Learning* (DL). DL allows to solve problems of representation learning, i.e. automatically discovering adequate representations of data without manual feature engineering [10]. These representations are expressed in terms of stacked, hierarchically ordered simpler representations. While DL is widely used with considerable success in areas such as Natural Language Processing [6, 11, 12, 33, 36] or Image Processing [16, 30, 47], it has only rarely been applied to the recommendation problem. The few exceptions focus on purposes other than sequential recommending, e.g. using DL as a preprocessing step to conventional techniques [49, 52]. Even if they consider consumption sequences [13, 18, 19, 48, 53, 54], they often have other limitations.

RecSys '17, August 27-31, 2017, Como, Italy

Recurrent Neural Networks (RNN) represent a specific form of DL models which possess several properties that make them attractive for sequence modeling [10]. In particular, they are capable of incorporating input from past consumption events, allowing to derive a wide range of sequence-to-sequence mappings. In order to consider time as a first class factor for modeling user preferences in RS, RNNs thus constitute a promising family of techniques. One shortcoming of simple variants, however, is the limited number of input variables that can effectively be handled [2]. Gated architectures [6] are designed to overcome this limitation by including gating units trained to control information flow through the network, thereby learning to keep information over a long period of time. Both *Long Short-Term Memory* (LSTM) [9, 20] and *Gated Recurrent Unit* (GRU) [5, 6] networks have shown advantages in real-world applications. Gated RNNs, however, have not been designed with the recommendation domain in mind. In particular, they are not optimized for taking interaction between user and system into account. As mentioned above, first attempts have been made to use DL for sequential recommending [13, 18, 19, 48, 53, 54]. Yet, they tend to reuse standard networks without modifications. As a result, the models usually have no notion about the specification of a target sequence, i.e. that respective RS have no mechanism to consider the fact that a sequence of consumed items is unique to an individual user. The networks treat every single sequence equivalently, thus learning global consumption behavior, but are unable to (deeply) integrate user-specific information.

In this paper, we propose a novel DL approach to address the sequential recommendation problem. First, relying on the benefits of gated RNNs, we model the temporal dynamics of consumption sequences. Second, through a novel gated architecture with additional input layers, we explicitly represent the individual user in such a network. These user-based GRUs are uniquely designed and optimized for the purpose of generating personalized next item recommendations. We theoretically and empirically show that not only out-of-the-box RNNs, but especially our approach using a novel gated architecture, may outperform state-of-the-art recommender algorithms as well as their time-dependent counterparts with respect to objective performance when predicting sequences. Offline experiments conducted on two real-world datasets indicate significant improvements when comparing our approach to common baseline RS algorithms and to a conventional RNN.

The remainder of this paper is organized as follows: First, we discuss relevant work related to sequential recommending and to applying DL in RS research. Second, we present our approach of user-based RNNs for RS. Next, we describe how we evaluated this approach in offline experiments. Finally, we conclude the paper and discuss potential future work.

2 RELATED WORK

User representation in RS is often designed as a statistically stationary process where every expressed preference is assumed to be fixed over time [29, 50]. However, relational data in real-world scenarios are often evolving and exhibit strong temporal patterns [55]. Time may therefore be considered an important contextual dimension also in RS [4]. Generating recommendations in a time-independent manner may in contrast result in estimations of preferences based on user-item relations that are no longer valid. This lack of adaptability with respect to one of the most natural properties of user behavior has motivated several approaches that integrate temporal dynamics[1]. In the following, we discuss the most important ones, and especially those most closely related to ours that also exploit DL techniques as a means to generate sequential recommendations.

2.1 Towards Sequential Recommendations

To adequately reflect user interests, functions that rank items according to some notion of utility are today considered most suitable for generating personalized recommendations [14]. Advances in RS research suggest that implicit feedback thereby often provides more comprehensive and extensive insights into user behavior than explicit ratings [39]. Fitting a static model to data naturally depending on dynamic processes approximates an improper data-generating function [42]. For this reason, several methods exist to incorporate temporal information into RS at some stage, which can be differentiated as follows [50]:

- *Time-aware RS* consider time as a contextual feature during the training phase. Timestamps serve as an additional source of information by which the model is enriched. The rationale behind is that user behavior underlies certain habits and regularities that repeat in regular time intervals, consequently allowing a more accurate prediction of similar patterns in the future.
- *Time-dependent RS* consider user preference data as chronologically ordered sequences, assuming that the most intrinsic property is that time establishes an order of the events. Input is required in chronological form, while exact time spans do not need to be taken into account. The algorithms consequently do not aim at modeling time as being cyclic, but rather at adapting to changes.

Existing time-aware methods include pre- and post-filtering [1] as well as Tensor Factorization [24]. In general, statistical models explicitly designed to predict sequences as used, for example, in automated playlist generation [3], may also be applied. However, they typically do not aim at modeling higher-order long-term dependencies. Eventually, time-dependent RS are conceptually closest to the definition of time used throughout this paper. Therefore, in the following, we focus on attempts that utilize chronological input data. By incrementally training models based on recency, the respective

[1] See one of the extensive surveys for an overview [e.g. 4, 50].

systems may capture, among others, continuous temporal dynamics such as concept drifts [26], changes in item popularity, or virtually any effect observable in sequential data.

Many approaches extend conventional CF techniques, for instance, by translating k-NN methods. In [18], it has been parenthetically shown that an unpersonalized variant of item-based k-NN achieves reasonably good results in sequence prediction tasks. Under an assumption similar to the Markov property, each prediction only depends on the item the user has previously consumed. In [7], a fully personalized time-discounted version has been proposed by extending k-NN with an exponential decaying component that penalizes items consumed a long time ago. Other approaches use time as a means for determining similarities, e.g. when consumption events are close together [17] or lie within a sliding window employed over the most recent sessions [35].

Matrix Factorization (MF) [29] algorithms have also been adapted. For example, baseline predictors can be added to the original framework to account for temporal dynamics [27]. Similar to their time-aware counterparts, sequential MF models have been extended to Tensor Factorization [55]. The dimensions contain time intervals so that the tensor can be seen as a periodic collection of ratings over time. Others have turned the MF problem into a graph-based model [38], integrated a Markov model for predicting the next shopping basket [40], or generalized existing temporal MF approaches to latent time series models [57]. Overall, the variant proposed in [23] comes closest to the definition we use: The authors model temporal order by means of a time window that includes feature vectors of previously consumed items. Thus, both collaborative interactions and time series aspects of the collaborative data can be taken into account.

2.2 Deep Learning in RS Research

The approaches discussed before constitute extensions of established time-agnostic recommendation methods. DL techniques such as RNNs can also be considered useful for sequential recommending as they are specifically designed for modeling temporal aspects. Despite the advantages of such techniques, only few neural network approaches have yet been generally proposed in RS research—most of them not focused on sequential recommendations.

One of the first attempts that only uses a shallow network structure is presented in [43]. Here, two-layered undirected graphical models called Restricted Boltzmann Machines are used to model user ratings for the prediction task. Deep neural networks have in contrast primarily been applied in preprocessing steps to conventional RS techniques. In [49], for instance, features are extracted from unstructured content, here raw music data. These features then serve as input to conventional CF in order to improve its performance. In [52], DL is performed on generic content-based information. The resulting deep feature representation is used as an enhancement to address the sparsity problem occurring in CF. More specific data, e.g. tag-based user and item profiles, have been embedded into a deep feature space in order to approach the hardly controllable dynamics of underlying tag corpora [56]. For that, the authors model similarities between users and potential target items by inferring a deep semantic model. Finally, some approaches actually exploit user interaction behavior. For example, for the purpose of generating adaptive user interfaces, i.e. recommending next actions based on similar interaction patterns found in the user base, GRUs have been used to embed learned interaction patterns together with users and interaction elements [46].

Only few works have yet used DL to explicitly generate sequential recommendations in a self-contained manner. The proposed approaches thereby often only focus on modeling consumption sequences without explicitly representing individual users [13, 18, 19, 48]. For instance, the applicability of the approach presented in [18] is restricted to sessions only. While user representation remains uncovered, the authors introduce a variant of a GRU model that utilizes pair-wise loss functions. In [48], this approach is extended by means of a priori data augmentation in form of sequence preprocessing. In addition, to reduce influence of outdated properties, essentially two models are trained: The first one on the complete dataset, which is then used to initialize the second one subsequently trained only on a subset of newer samples. Only recently, the RNN from [18] has been merged with feature-rich content information [19]: Item one-hot vectors and vectors of extracted features (e.g. corresponding to images of the respective items) are simultaneously treated as input to a GRU layer.

Few exceptions also consider user-related aspects [53, 54], but lack a deep integration of user vectors into the gating process. In [53], the RNN framework is extended by solving two recommendation problems and then merging the outputs: First, information is recurrently extracted from consumed item sequences. Second, specific user concepts are distinguished in a feed-forward manner. As a result, user characteristics are only considered independently from sequence properties. In [54], the authors train individual RNNs to model user and item evolution separately. The outputs from both networks are subsequently coupled with further auxiliary parameters capturing stationary concepts in order to predict user ratings. This architecture requires learning two RNNs such that user and item properties can yet again only loosely be intertwined.

3 USER-BASED RECURRENT NEURAL NETWORKS

Generating sequential recommendations relies on the assumption that individual data points form correlated sequences. Vectorized abstractions of user preferences or behavioral patterns are valuable information sources that can help to improve recommendation quality. In the previous section, we have discussed several existing methods for formalizing a sequential problem in the context of time-dependent recommendations. In the following, we elaborate on how we consider temporal dynamics in RS using DL techniques, and present a novel gated architecture for RNNs using specialized

GRUs that allows us to seamlessly integrate user-related information into the model.

The explicit notion of user information distinguishes our work from other DL approaches proposed for RS that usually focus on consumption sessions without explicitly representing the user. Compared to the few exceptions that already integrate user-related information (see Section 2), our approach is the first to deeply integrate user vectors into the gating process.

3.1 Recurrent Neural Networks for RS

First, for generating sequential recommendations, we need to concretize the generalized, domain-independent formulation of RNNs and transfer it to RS. RNNs, especially when augmented with gating layers, are powerful tools for modeling sequences of any kind[2]. We rely on a variant of RNNs that produces an output $\mathbf{o}^t \in \mathbb{R}^{|I|}$ at each time step via an affine transformation of the current hidden state $\mathbf{h}^t \in \mathbb{R}^n$:

$$\mathbf{o}^t = T_{n,|I|}\mathbf{h}^t, \tag{1}$$

where $T_{k,l} : \mathbb{R}^k \to \mathbb{R}^l$ is an affine transformation of the form $\mathbf{W}\mathbf{x} + \mathbf{b}$. In conventional RNNs, the computation of a hidden state is defined as a function of the previous hidden state and an input vector. In the context of RNNs for RS, we define this input vector $\mathbf{i}^t \in \{0,1\}^{|I|}$ as a one-hot vector where the only index different from zero corresponds to the index of a particular item.

3.2 User-based Gated Recurrent Units

Now, in order to integrate user characteristics into the model, we first define a one-hot user variable $\boldsymbol{\upsilon}^t \in \{0,1\}^{|\Upsilon|}$ with Υ being the set of users. We assume the data to be comprised of user-item tuples such that $\boldsymbol{\upsilon}$ can be interpreted as an indicator of the user consuming item $\mathbf{i}$ at a certain time step t in the sequence. Based on this, we extend the original definition (see [e.g. 6]) of the hidden state:

$$\mathbf{h}^t = f\mathbf{h}^{t-1}, \mathbf{i}^t, \boldsymbol{\upsilon}^t; \boldsymbol{\theta} \tag{2}$$

The network's predictive power may benefit from explicitly memorizing concepts about the user inside the recurrent cells. In the following, we discuss several realizations of (2) that architecturally modify the original recurrent unit[3]. By incorporating a user variable $\boldsymbol{\upsilon}^t$, we can thus deeply integrate user-related information into the network.

3.2.1 Linear User Integration. Linear user integration is comparable to strategies of considering context for gated units. For instance, in [34], word-related topic vectors are incorporated into a neural network language model to improve performance. Similarly, $\boldsymbol{\upsilon}^t$ can be viewed as an additional input layer that is connected to the gated units. Thereby, it can influence decisions about forgetting certain pieces of information or updating hidden states:

$$\begin{bmatrix}\mathbf{u}\\ \mathbf{r}\end{bmatrix} = \begin{bmatrix}\sigma\\ \sigma\end{bmatrix} T_{3n,2n} \begin{bmatrix}\mathbf{h}^{t-1}\\ \mathbf{E}_i\mathbf{i}^t\\ \mathbf{E}_\upsilon\boldsymbol{\upsilon}^t\end{bmatrix}, \tag{3}$$

where σ is the logistic sigmoid. $\mathbf{E}_i \in \mathbb{R}^{|I|\times n}$ and $\mathbf{E}_\upsilon \in \mathbb{R}^{|\Upsilon|\times n}$ are embedding matrices that map one-hot vectors into a densified information space. This kind of densification via projection is widely applied in DL research (see [e.g. 32, 33]).

While the gating processes inside the hidden units deal with temporal aspects of the data, non-temporal structures can be exploited by including embedding. In the embedded space, items that appear in similar contexts are also spatially close. For notational simplicity, we assume that the embedding dimension is equal to the hidden dimension n although this does not necessarily has to be the case.

Besides influencing update and reset gates, the user vector can also be included in the calculation of hidden states. For this, we first calculate a state update vector $\mathbf{k} \in \mathbb{R}^n$:

$$\begin{aligned}\mathbf{k} = \tanh T_{n,n}\, \mathbf{r} \odot \mathbf{h}^{t-1}\\ + T_{n,n}\mathbf{E}_i\mathbf{i}^t\\ + T_{n,n}\mathbf{E}_\upsilon\boldsymbol{\upsilon}^t\,,\end{aligned} \tag{4}$$

where tanh is the hyperbolic tangent and $\odot$ represents element-wise multiplication.

Subsequently, we can leakily integrate $\mathbf{k}$ into the hidden state update mechanism subject to $\mathbf{u}$:

$$\mathbf{h}^t = 1_n - \mathbf{u} \odot \mathbf{h}^{t-1} + \mathbf{u} \odot \mathbf{k}, \tag{5}$$

where 1_n is a vector of ones.

While $\mathbf{i}^t$ changes with every time step, consumption sequences are unique to only a single user, i.e. $\boldsymbol{\upsilon}^t$ remains constant. Generalized properties of each item in the sequence relate to the consuming user by feeding back succeeding losses to parameters conditioned by this particular $\boldsymbol{\upsilon}^t$. As a result, trained user-related parameters are representing superordinate concepts. This means, parameters corresponding to $\boldsymbol{\upsilon}^t$ express the user's general preference structure. Figure 1 depicts such a linear user GRU cell.

3.2.2 Rectified Linear User Integration. The linear integration of $\boldsymbol{\upsilon}^t$ allows the recurrent cell to condition the activation based on user characteristics. However, repeatedly applying the same user vector increases redundancy that might lead to undesired effects like higher sensitivity to underfitting and non-zero predictions for all user-item pairs since the particular network parameters are trained with respect to unchanging input over a long series of steps.

Furthermore, user-related parameters might not be equally important at every step in time. Some recommendations might be sufficiently derived only based on item vectors without considering a user component. For example, one can assume that users will consume all parts of a movie trilogy in a row. Thus, cells should learn to selectively set focus on different parts of the user representation. We use a leaky

[2]For a general introduction to RNNs, please refer to [e.g. 31].

[3]We use GRUs due to notational simplicity, although all principles can easily be transferred to LSTM in analogous form.

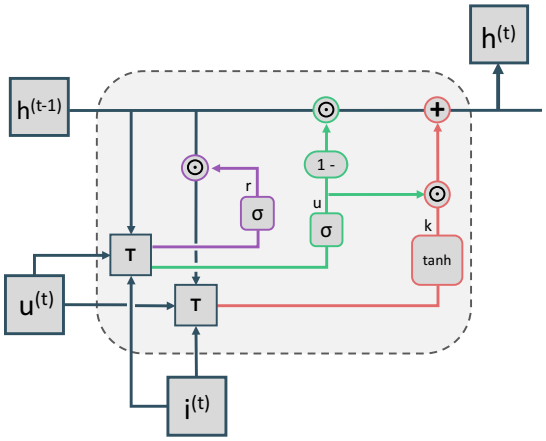

Figure 1: Linear user-based GRU cell: In addition to the item input vector i, the user representation υ is included.

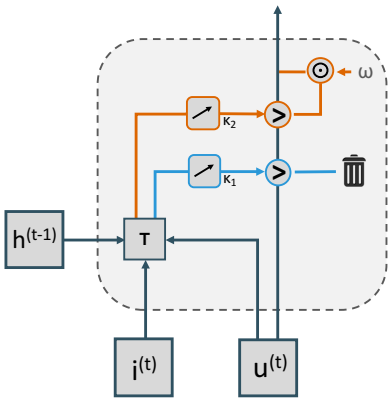

Figure 2: The rectified linear gating process detached from the complete cell: Item and user vectors as well as previous hidden state are concatenated and mapped linearly twice in order to calculate $\boldsymbol{\kappa}_1$ and $\boldsymbol{\kappa}_2$. Components of the user vector that are element-wise smaller than the corresponding values in $\boldsymbol{\kappa}_1$ are discarded. Components smaller than $\boldsymbol{\kappa}_2$ are diminished by element-wise multiplication with ω.

integrator inspired by rectified linear units [15] to attach weight to the transformed components of $\boldsymbol{\upsilon}^t$:

$$\mathbf{E}_\upsilon \boldsymbol{\upsilon}^t{}_l \leftarrow \begin{cases} 0, & \text{if } \mathbf{E}_\upsilon \boldsymbol{\upsilon}^t{}_l < \kappa_{1,l} \\ \omega \mathbf{E}_\upsilon \boldsymbol{\upsilon}^t{}_l, & \text{if } \kappa_{1,l} < \mathbf{E}_\upsilon \boldsymbol{\upsilon}^t{}_l < \kappa_{2,l} \\ \mathbf{E}_\upsilon \boldsymbol{\upsilon}^t{}_l, & \text{else} \end{cases}, \tag{6}$$

where l refers to a particular entry of vector $\mathbf{E}_\upsilon \boldsymbol{\upsilon}^t$. $\boldsymbol{\kappa}_1 \in \mathbb{R}^n$ and $\boldsymbol{\kappa}_2 \in \mathbb{R}^n$ are threshold parameter vectors conditioned on previous hidden state as well as item and user vector:

$$\begin{bmatrix} \boldsymbol{\kappa}_1 \\ \boldsymbol{\kappa}_2 \end{bmatrix} = T_{3n,2n} \begin{bmatrix} \mathbf{h}^{t-1} \\ \mathbf{E}_i \mathbf{i}^t \\ \mathbf{E}_\upsilon \boldsymbol{\upsilon}^t \end{bmatrix} \tag{7}$$

The thresholds dynamically ensure that only relevant parts of the user representation are exploited to produce network output. User-specific concepts can thus be shut off if there is reason to assume that the current input should be handled independently. Note that very important concepts, i.e. high values in $\mathbf{E}_\upsilon \boldsymbol{\upsilon}^t$, are only discarded if there is a strong indication of orthogonality, i.e. high values for $\boldsymbol{\kappa}_1$.

In cases where a component is only important to a certain degree, i.e. $\kappa_{1,l} < \mathbf{E}_\upsilon \boldsymbol{\upsilon}^t{}_l < \kappa_{2,l}$, it might be advantageous not to turn it off completely, but only to limit its throughput. For this purpose, we also integrate a leaky variant with $\omega \in 0, 1$. Update and reset cells are then calculated according to (3) with the filtered user component. The leaky integrator is included in the cell shown in Figure 2.

3.2.3 Attentional User Integration. As already derived in Section 3.2.1, the user component represents stable concepts as opposed to the current input vector $\mathbf{i}^t$ that rather expresses short-term aspects. The rectified linear user integration is only designed to impede the user component when necessary. As an alternative, the network could adaptively shift focus between user and item aspects. For instance, the user component's influence should be low at $t=0$ when no information about the user is present, and progressively increase afterwards as the sequence is further propagated. Moreover, at times, a consumed item might be out of the ordinary and would decrease succeeding estimation quality. In such cases, resorting to a stable user representation and excluding the outlier could attenuate or even annul its impact.

We therefore propose an adaptive approach that includes a new kind of gated cell that regulates the gating process. Let $\boldsymbol{\xi} \in \mathbb{R}^n$ be the attentional regulation gate:

$$\boldsymbol{\xi} = \sigma T_{n,n} \mathbf{h}^{t-1} + T_{n,n} \mathbf{E}_i \mathbf{i}^t + T_{n,n} \mathbf{E}_\upsilon \boldsymbol{\upsilon}^t \tag{8}$$

We use logistic sigmoid as a squashing function to leakily regulate the proportion of item and user focus. In contrast to a linear user-based GRU cell, item and user vectors are now weighted by $\boldsymbol{\xi}$:

$$\begin{bmatrix} \mathbf{u} \\ \mathbf{r} \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \end{bmatrix} T_{3n,2n} \begin{bmatrix} \mathbf{h}^{t-1} \\ 1_n - \boldsymbol{\xi} \odot \mathbf{E}_i \mathbf{i}^t \\ \boldsymbol{\xi} \odot \mathbf{E}_\upsilon \boldsymbol{\upsilon}^t \end{bmatrix} \tag{9}$$

Furthermore, the hidden state's update mechanism is not only dependent on $\mathbf{i}^t$ and $\boldsymbol{\upsilon}^t$, but also on $\boldsymbol{\xi}$:

$$\begin{aligned} \mathbf{k} = \tanh & T_{n,n} \, \mathbf{r} \odot \mathbf{h}^{t-1} \\ & + T_{n,n} \, 1_n - \boldsymbol{\xi} \odot \mathbf{E}_i \mathbf{i}^t \\ & + T_{n,n} \, \boldsymbol{\xi} \odot \mathbf{E}_\upsilon \boldsymbol{\upsilon}^t \end{aligned} \tag{10}$$

The state update vector $\mathbf{k}$ is then integrated analogously to (5). In contrast to (4), the attentional gate $\boldsymbol{\xi}$ now acts as a leaky integrator that can choose to completely ignore the user aspect (extremely low sigmoids) or simply copy it (extremely high sigmoids), see Figure 3. Informally speaking, the gate regulates the extent to which users are considered as opposed to items. Note that parameters for the attention gate should be initialized with values close to zero such that the network

behaves similar to a standard GRU at the beginning of the training phase, and then may gradually shift focus. Thus, the network will not explicitly focus on the user component until it has learned to do so.

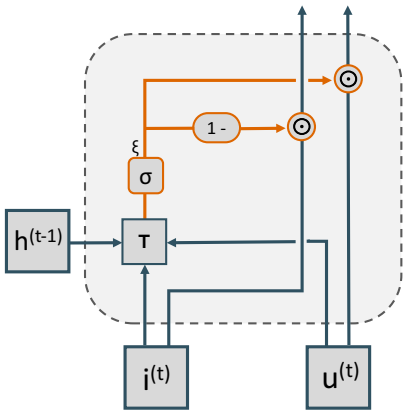

Figure 3: The attentional layer ξ detached from the complete cell: Item and user vectors as well as previous hidden state are concatenated and mapped with sigmoid squashing to form a gate that controls information flow between item and user component. Low sigmoids increase the throughput of item-related components, high sigmoids support the user side.

4 EVALUATION

To objectively justify our theoretical models derived in the previous section, we conducted several offline experiments. In particular, we aimed at answering the question how our novel user-based RNN approach with its different variants to deeply integrate user information performs on the sequential recommendation task when compared to a conventional RNN as well as state-of-the-art recommendation techniques.

Hence, we compared our three user-based networks (i.e. using linear, rectified linear, and attentional user integration) with a standard GRU network without any of our proposed extensions[4]. We also considered the following baselines: We trained time-independent item-based k-NN as well as its time-dependent exponentially decaying counterpart [7]. Moreover, using the BPR learning criterion [39], we trained a standard MF [29] as well as a sequential MF [23] variant.

4.1 Methodology

In the following, we describe the evaluation metrics and datasets we used, as well as the algorithmic setup.

[4]We implemented the different variants using *TensorFlow* (https://www.tensorflow.org/), an open source software framework especially designed for building deep neural networks as data flow graphs. For all experiments, we used ECP P2 instances provided by *Amazon Web Services* (https://aws.amazon.com/), which are powerful scalable instances designed for GPU-based operations using CUDA.

4.1.1 Metrics. Based on temporally ordered lists of consumed items, our objective is to correctly predict the next item a target user will likely consume. The ground truth at a particular time step is therefore represented by a single user-item tuple. To present the user with adequate recommendations, the target item should be among the first few recommended items. In accordance with recent RS research, we thus use the following evaluation metrics:

- MRR@k (*Mean Reciprocal Rank*) is defined as the average of the reciprocal ranks of the desired items [51]. The rank is set to zero if it is above k.
- *Recall@k* is defined as the fraction of cases where the item actually consumed in the next event is among the top k items recommended [37].

We set $k = 20$, as it appears desirable from a user's perspective to expect the target among the first 20 items [18].

4.1.2 Datasets. We ran our approach as well as all the baselines on the following two real-world datasets:

- *MovieLens 10M:* The MovieLens 10M dataset[5] consists of 10 000 054 ratings assigned to 10 681 movies by 71 567 users. In order to mimic implicit data, we binarized all ratings independent of their value, considering them as positive feedback as it has been done in [e.g. 39]. Using the timestamps provided, we thus got an ordered sequence of consumption events for each user. The dataset contains only sequences with a minimum length of 20, making further preprocessing of the training set unnecessary. The average sequence length is 115. We aimed at predicting the next movie to watch.
- *LastFM 1K Users:* The LastFM 1K Users dataset[6] contains user-timestamp-artist-song tuples collected via the LastFM API. The dataset has a total of 19 150 868 data points for 992 users. Due to computational reasons we performed our evaluation on a 10 % subsample. The resulting average sequence length is 1 738. We aimed at predicting the next song title.

We split each dataset into three parts: First, the major fraction of every sequence serves as training data. Second, we use a validation set during training to measure performance on unseen data. These validation results determine, for instance, early stopping used in the learning phase to avoid overfitting [21]. Third, we use a distinct test set to measure actual performance after learning is completed. The training set consists of the first 90 % of every user consumption sequence. The remaining 10 % of each sequence are split evenly into validation and test set while maintaining the order. As it is common practice in RS research, items not seen during training are filtered out from validation and test set.

4.1.3 Hyperparameter and Network Setup. Table 1 shows all hyperparameters we set for our experiments based on extensive pretesting with grid search.

For the RNN variants, we propagate mini-batch based learning with batch size of 1000. We use shallow networks

[5]https://grouplens.org/datasets/movielens/
[6]http://www.dtic.upf.edu/~ocelma/MusicRecommendationDataset/

Table 1: Hyperparameter values for all algorithms used in our experiments on the two datasets.

Parameter	Value
Item-based k-NN	
Nearest Neighbors	100
Exp. Dec. Item-based k-NN	
Decaying Constant	1
Matrix Factorization	
Latent Factors	20
Iterations	30
Learning Rate	0.01
Seq. Matrix Factorization	
Window Size	2

Parameter	Value
General GRU	
Batch size	1000
Layers	1
Hidden Units	1000
Unfold Dimension	20
Dropout	0.8
Gradient Cap	5.0
Iterations	10
Learning Rate	0.001
User-based GRU	
User Dropout	0.5

Table 2: MRR@20 and Recall@20 for all baselines as well as our proposed user-based RNN variants.

	MovieLens		LastFM	
	MRR@20	Recall@20	MRR@20	Recall@20
Baselines				
Item-based k-NN	0.036 39	0.121 42	0.044 38	0.101 26
Exp. Dec. Item-based k-NN	0.042 31	0.128 53	0.055 94	0.167 24
Matrix Factorization	0.011 92	0.077 44	0.032 89	0.127 23
Seq. Matrix Factorization	0.015 50	0.107 30	0.027 69	0.112 24
Standard GRU	0.047 30	0.157 73	0.143 74	0.202 69
User-based RNN				
Linear User-based GRU	0.052 51	0.160 28	0.182 52	0.249 20
Rectified Linear User-based GRU	0.059 01	0.186 97	**0.191 54**	0.254 09
Attentional User-based GRU	**0.062 55**	**0.205 40**	0.187 31	**0.254 85**

of depth 1 with hidden dimensionality 1000 that is unfolded for 20 time steps. Stacking multiple hidden recurrent layers on top of each other to create depth [8, 12, 44] did not improve overall performance in our pretests. According to [36], depth in RNNs introduces capabilities to represent different timescales. When trained on the datasets we used, the networks however did not seem to benefit from this property.

For regularization, we apply Dropout [58] with a keep probability of 0.8 for item and 0.5 for user vectors. We use a gradient cap of 5.0 to clip large gradients that might otherwise lead to skip over minima. Furthermore, we introduce a L^2 parameter norm penalty on user side in order to control for redundancy. Its contribution weight is set to 0.01. Regarding the matrices embedding the one-hot encoded input vectors, we set the dimension equal to the hidden dimension. Weight matrices and biases are initialized randomly with values drawn from $-0.1, 0.1$. We run training for 10 iterations.

Linear and attentional user integration do not require setting any further parameters. In case of rectified linear user integration, grid search indicated to set $\omega = 0.2$, i.e. diminished user-related concepts are only considered by 20 % of their original magnitude.

Pretests also indicated that the RNNs are trained best with Adam optimizer [25] using a learning rate of 0.001 against Cross Entropy loss. For this, we normalize output vectors $\mathbf{o}^t$ via softmax function in order to produce a valid probability distribution.

4.2 Results

Table 2 shows the results received after training the baselines as well as our DL models on both datasets.

The RNNs yield results superior to the baselines in terms of both metrics. On the MovieLens dataset, there is a substantial gain in objective performance compared to the next best baseline. Namely, there is an improvement of 12 % in MRR@20 and 23 % in Recall@20, respectively, when comparing exponentially decaying item-based k-NN to the standard GRU. On the LastFM dataset, the differences in ranking quality become even more apparent (at least 157 % improvement in terms of MRR@20, 21 % in Recall@20).

Integrating user information significantly improves statistical power in GRU networks. All user-based RNN variants outperform the standard GRU network that does not consider user information. In particular, the integration variant that performs best in terms of MRR@20, i.e. attentional on the MovieLens and rectified linear on the LastMF dataset, leads to an improvement of 32 % or 33 %, respectively. For Recall@20, there is similar gain with results 30 % better on the MovieLens, and 28 % better on the LastFM dataset.

Taking a closer look at the results for the user-based RNNs, attentional user integration achieves the overall best results. Only on the LastFM dataset, MRR@20 is slightly lower than for the rectified linear variant. In all other cases, the attentional realization outperforms the others.

Finally, concerning the baselines, item-based k-NN is better at modeling sequences than MF. Even sequential MF achieves clearly inferior results than the original item-based k-NN without any temporal extensions. As a side note, while the temporal item-based k-NN generally outperforms its time-independent counterpart, sequential MF seems not beneficial compared to standard MF on the LastFM dataset.

4.3 Discussion

Overall, the results indicate that RNNs clearly outperform other recommending approaches when it comes to generating sequential recommendations. As earlier experiments by others suggested [e.g. 18, 48], even out-of-the-box RNNs achieve superior results with respect to widely used evaluation metrics when compared to state-of-the-art item-based k-NN or MF approaches. This is particularly true for their derivatives specifically extended to consider temporal effects.

By deeply integrating user information, we were able to personalize the sequence prediction task. The experiments suggest that our networks are able to learn the implicit relations between events more effectively when passing in externally encoded information about users. Nonetheless, the advantages of exploiting temporal dynamics in combination with user information seem to depend on the background data. For instance, the relative performance gain is much higher on the LastFM than on the MovieLens dataset. Considering the nature of the datasets, this is however not surprising: During one session, users generally listen to more than one song in a row, e.g. they consume playlists or whole albums at once, and factors such as the user's mood determine which

title is likely to be consumed next. Thus, similar to the underlying grammar in Natural Language Processing, there exist inherent dependencies between succeeding events that can be exploited. Such relations are much less obvious when recommending movies as people in this case generally do not consume multiple items in immediate succession. Furthermore, sequences in the MovieLens dataset can be considered more artificially, as they do not describe actual consumption events at certain points in time, but reproduce the process of actively providing feedback in form of ratings. Still, there seem to be some implicit connections between the events that can be captured by the networks: Even with such a dataset, RNNs achieve superior results for predicting the next item compared to baseline techniques.

Regarding the different variants of user integration, rectified linear and attentional realization yield generally better results than the linear one. This indicates that regulating redundancy of user input in fact appears to be beneficial. The mechanism seems to support cells in controlling information flow more accurately. In both of the advanced variants, influence of the user variable can be diminished when necessary, thus restricting its impact and preventing overgeneralization. Thereby, it is important to note that integrating user information does not come along with considerably higher runtime requirements. In particular, the user-based RNNs can be trained in almost same amount of time as a standard GRU network. For instance, on the MovieLens dataset the number of processed input tuples per second was on average 24 917 for the Standard GRU, 22 887 for the Linear User-based GRU, 22 840 for the Rectified Linear User-based GRU, and 22 822 for the Attentional User-based GRU. In general, as earlier work on RNNs in RS research has shown [e.g. 18], training the models with sufficient computational power can be done in reasonable time.

5 CONCLUSIONS AND OUTLOOK

With the rise of DL in the past decade, RNNs have become practical and powerful tools for large-scale supervised learning of sequences. This progress has become most apparent in Natural Language Processing where they have set several new benchmarks outperforming techniques that have been considered state-of-the-art for a long time. Similar to other works, the promising results from our experiments show that RNNs allow to take a novel perspective on applications such as RS that were originally designed in a time-agnostic manner. Our novel approach proposed in this paper allows generating personalized suggestions while time is considered as a first class factor, thereby significantly improving recommendation quality. The way we formulate RNNs enables us to model user behavior and to capture dependencies between consumption events more adequately than with established recommendation techniques that often fail at appropriately representing temporal dynamics in user interests.

Gated recurrent networks have set records in accuracy on many tasks in recent years. Noteworthy, these advances result from novel or extended architectures rather than from fundamentally novel algorithms [31, 45]. This also applies to our user-based RNNs: We essentially adopted the original architecture to take the unique characteristics of the recommendation domain into account. Our specific extensions have then shown to increase statistical expressiveness: Deeply integrating a user representation leads to significant improvements when predicting user behavior such as watching movies or listening to music. Including a user-specific layer, the networks seem capable of learning concepts that are unique to a certain user. Since changing inputs are related to a user vector that remains constant over the course of a particular sequence, the networks learn to represent global user concepts explicitly, rather than implicitly as it would be the case in conventional gated units. Hence, the networks can distinguish between user preferences more accurately.

We have presented a DL-based framework that is particularly designed to generate personalized next item recommendations, thereby independent of techniques used in contemporary RS research. In future work, we plan to additionally integrate state-of-the-art recommendation techniques such as MF into the design space of RNNs, e.g. in form of contextual variables. By this means, we would be able to combine the best of both worlds while also extending the scope of applicability. Also, this might be helpful for better supporting situations where the user's consumption sequence is rather short, e.g. at cold-start. Moreover, we aim at taking varying time intervals into account. RNNs are designed for modeling sequential data with no notion of time spans between succeeding events. Especially for recommendation tasks, however, time deltas can be extremely valuable information. For instance, if two consumption events are distant from each other, the first item might not be a good predictor because user preference likely has changed over time. In this context, a comparison with statistical models particularly designed for predicting sequences would also be of interest. Finally, as interactive approaches are more and more discussed in RS research, it seems promising to examine ways of increasing control and transparency also in DL-based RS.

REFERENCES

[1] L. Baltrunas and X. Amatriain. 2009. Towards time-dependant recommendation based on implicit feedback. In *CARS '09*.

[2] Y. Bengio, P. Frasconi, and P. Simard. 1993. The problem of learning long-term dependencies in recurrent networks. In *ICNN '93*. IEEE, 1183–1188.

[3] G. Bonnin and D. Jannach. 2015. Automated generation of music playlists: Survey and experiments. *ACM Comput Surv* 47, 2 (2015), 26:1–26:35.

[4] P. G. Campos, F. Díez, and I. Cantador. 2014. Time-aware recommender systems: A comprehensive survey and analysis of existing evaluation protocols. *User Model User-Adap* 24, 1-2 (2014), 67–119.

[5] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. In *SSST '14*.

[6] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *Deep Learning Workshop at NIPS '14*.

[7] Y. Ding and X. Li. 2005. Time weight collaborative filtering. In *CIKM '05*. ACM, 485–492.

[8] S. El Hihi and Y. Bengio. 1995. Hierarchical recurrent neural networks for long-term dependencies. In *NIPS '95*. 493–499.

[9] F. Gers. 2001. *Long short-term memory in recurrent neural networks*. Ph.D. Dissertation. University of Hannover.

[10] I. Goodfellow, Y. Bengio, and A. Courville. 2016. *Deep learning*. MIT Press.

[11] A. Graves. 2013. Generating sequences with recurrent neural networks. (2013).

[12] A. Graves, A. Mohamed, and G. Hinton. 2013. Speech recognition with deep recurrent neural networks. In *ICASSP '13*. IEEE, 6645–6649.

[13] A. Greenstein-Messica, L. Rokach, and M. Friedman. 2017. Session-based recommendations using item embedding. In *IUI '17*. ACM, 629–633.

[14] A. Gunawardana and G. Shani. 2009. A survey of accuracy evaluation metrics of recommendation tasks. *J Mach Learn Res* 10 (2009), 2935–2962.

[15] K. He, X. Zhang, S. Ren, and J. Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV '15*. 1026–1034.

[16] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep residual learning for image recognition. In *CVPR '16*. IEEE, 770–778.

[17] C. Hermann. 2010. Time-based recommendations for lecture materials. In *EdMedia '10*. 1028–1033.

[18] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk. 2015. Session-based recommendations with recurrent neural networks. In *ICLR '16*.

[19] B. Hidasi, M. Quadrana, A. Karatzoglou, and D. Tikk. 2016. Parallel recurrent neural network architectures for feature-rich session-based recommendations. In *RecSys '16*. ACM, 241–248.

[20] S. Hochreiter and J. Schmidhuber. 1997. Long short-term memory. *Neural Comput* 9, 8 (1997), 1735–1780.

[21] T. Hofmann. 2001. Unsupervised learning by probabilistic latent semantic analysis. *Mach Learn* 42, 1-2 (2001), 177–196.

[22] T. Hussein, T. Linder, W. Gaulke, and J. Ziegler. 2014. Hybreed: A software framework for developing context-aware hybrid recommender systems. *User Model User-Adap* 24, 1-2 (2014), 121–174.

[23] A. Karatzoglou. 2011. Collaborative temporal order modeling. In *RecSys '11*. ACM, 313–316.

[24] A. Karatzoglou, X. Amatriain, L. Baltrunas, and N. Oliver. 2010. Multiverse recommendation: N-dimensional tensor factorization for context-aware collaborative filtering. In *RecSys '10*. ACM, 79–86.

[25] D. Kingma and J. Ba. 2015. Adam: A method for stochastic optimization. In *ICLR '15*.

[26] R. Klinkenberg and T. Joachims. 2000. Detecting concept drift with support vector machines. In *ICML '00*. Morgan Kaufmann, 487–494.

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

[28] Y. Koren and R. Bell. 2015. *Recommender systems handbook*. Springer, Chapter Advances in collaborative filtering, 77–118.

[29] Y. Koren, R. Bell, and C. Volinsky. 2009. Matrix factorization techniques for recommender systems. *IEEE Computer* 42, 8 (2009), 30–37.

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS '12*. 1097–1105.

[31] Z. C. Lipton, J. Berkowitz, and C. Elkan. 2015. A critical review of recurrent neural networks for sequence learning. (2015).

[32] T. Mikolov, K. Chen, G. Corrado, and J. Dean. 2013. Efficient estimation of word representations in vector space. *Workshop at ICLR '13*.

[33] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS '13*. 3111–3119.

[34] T. Mikolov and G. Zweig. 2012. Context dependent recurrent neural network language model. In *SLT '12*. 234–239.

[35] O. Nasraoui, J. Cerwinske, C. Rojas, and F. Gonzalez. 2007. Performance of recommendation systems in dynamic streaming environments. In *SDM '07*. SIAM, 569–574.

[36] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio. 2014. How to construct deep recurrent neural networks. In *ICLR '14*.

[37] D. M. Powers. 2011. Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation. *J Mach Learn Tech* 2, 1 (2011), 37–63.

[38] S. Rallapalli, L. Qiu, Y. Zhang, and Y.-C. Chen. 2010. Exploiting temporal stability and low-rank structure for localization in mobile networks. In *MobiCom '10*. ACM, 161–172.

[39] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *UAI '09*. AUAI Press, 452–461.

[40] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. 2010. Factorizing personalized Markov chains for next-basket recommendation. In *WWW '10*. ACM, 811–820.

[41] F. Ricci, L. Rokach, and B. Shapira. 2015. *Recommender systems handbook* (2nd ed.). Springer.

[42] N. Sahoo, P. V. Singh, and T. Mukhopadhyay. 2012. A hidden Markov model for collaborative filtering. *MIS Quarterly* 36, 4 (2012), 1329–1356.

[43] R. Salakhutdinov, A. Mnih, and G. E. Hinton. 2007. Restricted Boltzmann machines for collaborative filtering. In *ICML '07*. ACM, 791–798.

[44] J. Schmidhuber. 1992. Learning complex, extended sequences using the principle of history compression. *Neural Comput* 4, 2 (1992), 234–242.

[45] J. Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural Networks* 61 (2015), 85–117.

[46] H. Soh, S. Sanner, M. White, and G. Jamieson. 2017. Deep sequential recommendation for personalized adaptive user interfaces. In *IUI '17*. ACM, 589–593.

[47] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. 2015. Going deeper with convolutions. In *CVPR '15*. IEEE, 1–9.

[48] Y. K. Tan, X. Xu, and Y. Liu. 2016. Improved recurrent neural networks for session-based recommendations. In *DLRS '16*. ACM, 17–22.

[49] A. van den Oord, S. Dieleman, and B. Schrauwen. 2013. Deep content-based music recommendation. In *NIPS '13*. 2643–2651.

[50] J. Vinagre, A. M. Jorge, and J. Gama. 2015. An overview on the exploitation of time in collaborative filtering. *Data Min Knowl Disc* 5, 5 (2015), 195–215.

[51] E. M. Voorhees. 1999. The TREC-8 question answering track report. In *TREC '99*. 77–82.

[52] H. Wang, N. Wang, and D.-Y. Yeung. 2015. Collaborative deep learning for recommender systems. In *KDD '15*. ACM, 1235–1244.

[53] C. Wu, J. Wang, J. Liu, and W. Liu. 2016. Recurrent neural network based recommendation for time heterogenous feedback. *Knowl-Based Syst* 109 (2016), 90–103.

[54] C.-Y. Wu, A. Ahmed, A. Beutel, A. J. Smola, and H. Jing. 2017. Recurrent recommender networks. In *WSDM '17*. ACM, 495–503.

[55] L. Xiong, X. Chen, T.-K. Huang, J. Schneider, and J. Carbonell. 2010. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *SDM '10*. SIAM, 211–222.

[56] Z. Xu, C. Chen, T. Lukasiewicz, Y. Miao, and X. Meng. 2016. Tag-aware personalized recommendation using a deep-semantic similarity model with negative sampling. In *CIKM '16*. ACM, 1921–1924.

[57] H.-F. Yu, N. Rao, and I. S. Dhillon. 2015. High-dimensional time series prediction with missing values. (2015).

[58] W. Zaremba, I. Sutskever, and O. Vinyals. 2014. Recurrent neural network regularization. (2014).