Abstract

Modern technologies enable us to record sequences of online user activity at an unprecedented scale. Although such activity logs are abundantly available, most approaches to recommender systems are based on the rating-prediction paradigm, ignoring temporal and contextual aspects of user behavior revealed by temporal, recurrent patterns. In contrast to explicit ratings, such activity logs can be collected in a non-intrusive way and can offer richer insights into the dynamics of user preferences, which could potentially lead to more accurate user models.

In this work we advocate studying this ubiquitous form of data and, by combining ideas from latent factor models for collaborative filtering and language modeling, propose a novel, flexible and expressive collaborative sequence model based on recurrent neural networks. The model is designed to capture a user’s contextual state as a personalized hidden vector by summarizing cues from a data-driven, thus variable, number of past time steps, and represents items by a real-valued embedding. We found that, by exploiting the inherent structure in the data, our formulation leads to an efficient and practical method. Furthermore, we demonstrate the versatility of our model by applying it to two different tasks: music recommendation and mobility prediction, and we show empirically that our model consistently outperforms static and non-collaborative methods.

Keywords: Recurrent Neural Network, Recommender System, Neural Language Model, Collaborative Filtering

1. Introduction

As ever larger parts of the population routinely consume online an increasing amount of digital goods and services, and with the proliferation of affordable storage and computing resources, to gain valuable insight, content-, product- and service-providing organizations face the challenge and the opportunity of tracking at a large scale the activity of their users. Modeling the underlying mechanisms that govern a users’ choice for a particular item at a particular time is useful for, e.g., boosting user engagement or sales by assisting users in navigating overwhelmingly large product catalogs through recommendations, or by using profits from accurately targeted advertisement to keep services free of charge. Developing appropriate methodologies that use the available data effectively is therefore of great practical interest.
In recent years, the recommendation problem has often been cast into an explicit-rating-prediction problem, possibly facilitated by the availability of appropriate datasets (e.g., Bennett and Lanning, 2007; Miller et al., 2003). Incorporating a temporal aspect into the explicit-rating recommendation scenario is an active area of research (e.g., Rendle, 2010; Koren, 2010; Koenigstein et al., 2011; Chi and Kolda, 2012). However, concerns are increasingly being raised about the suitability of explicit rating prediction as an effective paradigm for user modeling. Featuring prominently among the criticism are concerns about the availability and reliability of explicit ratings, as well as the static nature of this paradigm (Yi et al., 2014; Du et al., 2015), in which the tastes and interests of users are assumed to be captured by a one-time rating, thus neglecting the immediate context of the rating at the instant it is issued by the user.

In contrast, the implicit-feedback scenario is advantageous due to abundantly available data that can be gathered unintrusively in the background (Rendle et al., 2009). Although methods for processing implicit feedback often only consider a static aggregate (e.g., Ko and Khan, 2014), the raw data typically consist of user activity logs, i.e., sequences of user interactions with a service, and is thus a far more suitable basis for dynamic user models. Data of this form is generated in many domains where user interactions can range from navigating a website by following links, to consuming multi-media content on a streaming site or to announcing the current physical locations on social media, thus making appropriate methods widely applicable. Moreover, a user’s behaviour is intimately linked to the context in which it is observed: factors such as mood, current activity, social setting, location, time-of-day etc. can temporarily cause a shift in a user’s item preferences. While the relevant contextual factors themselves are hard to observe or even to define, their effect might be visible in the form of particular temporal patterns in the activity logs, the exploitation of which would then lead to more fine-grained and accurate user models. The purpose of this work is therefore to develop and evaluate methods for analyzing activity sequences generated by a heterogeneous set of users.

In the remainder of this section, we will introduce the problem more formally, state the contributions in the context of related work and outline the rest of the paper.

1.1. Preliminaries

Let \( \mathcal{U} \) denote a set of users of cardinality \( |\mathcal{U}| = U \) and let \( \mathcal{I} \) denote a set of items of cardinality \( |\mathcal{I}| = I \). As usual, the sets of users and items are assumed to be fixed. We use indices \( u \) and \( i \) to refer to users and items, respectively. For every user \( u \in \mathcal{U} \) we observe a sequence \( x^{(u)} = [x^{(u)}_1, \ldots, x^{(u)}_{T_u}] \) with elements \( x^{(u)}_t \in \mathcal{I} \), i.e., each event in a user sequence is an exclusive choice over the set of items. We will refer to a user making such a choice as consuming the item. Similar to (Rendle et al., 2010), we focus on sequences without taking into account the absolute time at which the events occurred, because here, we are interested in exploiting temporal patterns expressed by the ordering of events\(^1\). We denote by \( x^{(u)}_{<t} = [x^{(u)}_1, \ldots, x^{(u)}_{t-1}] \) and \( x^{(u)}_{\geq t} = [x^{(u)}_t, \ldots, x^{(u)}_{T_u}] \) subsequences up to and from index \( t \), respectively.

---

\(^1\) Different ways of incorporating explicit timestamps are discussed in Section 4.
Collaborative RNNs

The basic query a sequence model should support is the quantification of the likelihood of a user \( u \) to consume items at \( t = k, k + 1, k + 2, \ldots \) given access to the history \( x^{(u)}_{<k} \) of events.

Our approach for such prediction tasks is to model the observed sequences probabilistically. Given the ordering imposed by time, it is reasonable to choose a factorization of the joint distribution over a sequence that respects that order:

\[
P_{\theta}(x^{(u)}) = \prod_{t=1}^{T_u} P_{\theta}(x_t^{(u)} | x_{<t}^{(u)}),
\]

where \( \theta \) denotes a set of model parameters.

As we model a discrete set of outcomes, the conditionals are multinomial distributions over \( I \), represented by probability vectors \( p^{(u)}(\cdot) \) of dimension \( I \), whose concrete parameterization and form is determined by the assumptions that underlie a particular model:

\[
P_{\theta}(x_t^{(u)} | x_{<t}^{(u)}) = p^{(u)}(\cdot | x_{<t}^{(u)}).
\]

The evaluation metric we report is the average negative log likelihood computed for a held-out part of the sequence: For every user, we split \( x^{(u)} \) into \( x_{<r_u}^{(u)} \) and \( x_{\geq r_u}^{(u)} \) at \( 1 < r_u \leq T_u \) and define the error as

\[
E(x_{\geq r_1}^{(1)}, \ldots, x_{\geq r_U}^{(U)} | \theta) = -\frac{1}{U} \sum_{u \in U} \frac{1}{T_u - r_u + 1} \log P_{\theta}(x_{\geq r_u}^{(u)}).
\]

The desired traits of a model for this kind of data are (1) flexibility, (2) the ability to model complex, potentially long-range dependencies in the sequences, and (3) collaboration between users.

**Flexibility.** Although we often have at our disposal datasets richer than the sequences described above, these sequences are a common denominator for many applications. Thus, we devise practical methods suitable for processing sequences, but with the flexibility for handling different tasks with minor modifications, ideally in an end-to-end fashion, and for tapping into existing sources of metadata to augment the sequences.

**Long-range Dependencies.** The model needs to be powerful enough to represent complex dependencies within long sequences that might be of different length, in contrast to methods that, in order to achieve scalability, have to make compromises regarding the number of past events taken into account (Rendle et al., 2010; Wang et al., 2015).

**Collaboration.** Conceivably, there are two extreme cases: each individual user’s behavior is modeled in isolation, or all users are described by a single prototypical behavioral profile. There are several drawbacks to both extremes: the former approach clearly relies on the sufficient availability of observations about each user, whereas realistically, the distribution of ratings over users is often heavily skewed in favor of a few very active users. Furthermore, users typically only access a subset of the items, thus making it difficult to recommend anything outside of this subset in a data-driven way. The latter approach, i.e. ignoring to the concept of individual users, lies on the other end of the spectrum and implies that all
sequences are pooled together. Although pooling mitigates the sparsity problem, it is a strong assumption that neglects potentially important differences in user behavior. Neither extreme satisfies the assumptions we stated previously. Instead, our model needs to be personalized yet collaborative to make efficient use of the available data by identifying parameters that can be learned across users.

In this work, we develop a model that fulfills these requirements based on recurrent neural networks (RNN, Rumelhart et al., 1988; Werbos, 1990).

1.2. Contributions

Using ideas from language modeling and the collaborative-filtering literature, we propose a novel collaborative sequence model that, based on RNNs, makes efficient use of the multi-user structure in the data. We study different architectural variants of our model and find that including the collaborative aspect can lead to an efficient, practical method. To test the viability of our approach, we conduct a broad empirical study on two very different real-world tasks. We find that the new model significantly outperforms various baseline methods.

Related Work. As previously mentioned, there has been much work dedicated to the rating-prediction problem. Some of these methods also take into account the effect of time (Rendle, 2010; Koren, 2010; Koenigstein et al., 2011; Chi and Kolda, 2012). The difference is that recurrent temporal patterns are not apparent in the underlying datasets, because ratings are one-time events. Du et al. (2015) propose a model for recurrent user activities, but use a fundamentally different approach: they use continuous-time stochastic processes to explicitly model time and tie the recommendation to a specific point in time and focus more on modeling when the next event might occur by using the self-excitation property of the Hawkes process. A problem closely related to our setup is the next-basket prediction for e-commerce applications (Rendle et al., 2010; Wang et al., 2015), which models sequences of sets of items that represent the contents of shopping carts bought by users. Wang et al. (2015) learn a hierarchical latent-factor representation reminiscent of a multi-layer perceptron arguing that the non-linearity in their model is a crucial component. Rendle et al. (2010) use a Markov chain per user and assume low-rank structure for the transition tensor and optimize a BPR-inspired objective (Rendle et al., 2009). Our work differs in mainly two ways: first, we model events at a finer granularity, i.e., per event, which is more appropriate for the applications we considered. Second, the use of RNNs enables us to model long-range dependencies, whereas their models use much stricter factorization assumptions for computational reasons. To the best of our knowledge, this is the first use of RNNs in this context.

Outline. This paper is organized as follows. In Section 2, we begin by motivating our approach and then presenting various baseline methods that comply with our framework. Next, we introduce RNN-based language models that serve as the basis for our collaborative RNN model. We describe the learning algorithm we use and architectural variants. We present the experimental evaluation in Section 3. In Section 4, we summarize and discuss our work and give an outlook on future extensions.
2. The Collaborative Recurrent Neural Network Model

We adopting a generative view on sequences and in the same spirit as latent factor models (Koren et al., 2009) and word embeddings (Mikolov et al., 2013), we postulate a latent feature embedding that characterizes the items and that can be considered static for the time-scale of interest. Then, it is plausible that a user’s subsequent choice is determined by the user’s internal state, describing her current valuation of the item features. This internal state is hidden and can only be inferred from recent activity and could be due to a mechanism too complicated to explicitly describe. Recurrent neural networks are powerful sequence models that operate in precisely this way: They maintain a hidden state that is updated by a complex, non-linear function learned from the data itself when the next element in the input sequence is presented to the network.

In recent years, practical advances in the area of recurrent neural networks and their variants (Hochreiter and Schmidhuber, 1997; Cho et al., 2014b) have led to enormously successful methods to tackle various interesting, yet extremely challenging, tasks that are inherently sequential in nature, most prominently in, but not restricted to, the area of Natural Language Processing (NLP). The success of RNN-based models in these applications demonstrates their versatility, flexibility and ability to capture complex patterns that require the model to retain information gathered at various previous time steps, possibly in the distant past. Therefore, in spite of shortcomings such as the lack of theoretical guarantees on the performance, we deemed RNNs to be a solid foundation on top of which we developed a model that fulfills the previously posed requirements.

Next, we briefly present the RNN Language Model (RNNLM, Mikolov et al., 2010) in Section 2.2, the main source of inspiration for this work, as we find their formulation to closely match our setup. In their setup, $I$ represents the vocabulary, and user sequences are sentences. The crucial difference to our setup is that the goal in language modeling is to learn a single underlying concept that ties together all sequences: a generative model for the language. Thus, it makes sense to not seek to model the dynamics per sequence, but to do so for the whole corpus. The necessary model-complexity for capturing the nuances of a language is achieved by adding more layers (Graves et al., 2013; Sutskever et al., 2014). This perspective is to an extent supported by the type of data itself: For language models, a corpus consists of a plethora of relatively short sequences from the same language, whereas in our case, sequences are very long and are expressions of relatively few distinctly individual preferences. This difference in perspective underlies our approach in Section 2.3.

2.1. Baseline Models

Many approaches to specify a model within the framework of Eq. 1 come to mind. Here, we present several straightforward models, that differ on whether or not they have a mechanism to represent dynamics and collaboration, and against which we will benchmark our method.

**Static Uniform.** As a trivial example, consider the uniform distribution over $I$.

$$P(x_{t}^{(u)} | x_{<t}^{(u)}) = P(x_{t}^{(u)}) = \frac{1}{I}. \quad (4)$$

**n-grams.** n-gram models, popular in NLP (Brown et al., 1992), are distributions over co-occurrences of $n$ consecutive items and thus have to maintain $O(I^n)$ parameters. Due
to data sparsity, we consider only $n = 1$ (unigram) and $n = 2$ (bigram). We define the following quantities:

$$c_i = \sum_{u \in U} \sum_{t=1}^{T_u} \mathbb{1}(x_t^{(u)} = i), \quad b_{ij} = \sum_{u \in U} \sum_{t=2}^{T_u} \mathbb{1}(x_t^{(u)} = i, x_{t-1}^{(u)} = j).$$  \hspace{1cm} (5)

Then, we define the unigram model as

$$P(x_t^{(u)} = i \mid x_{<t}^{(u)}) = P(x_t^{(u)} = i) = \frac{c_i + \epsilon}{\sum_{k \in \mathcal{I}} (c_k + \epsilon)}. \hspace{1cm} (6)$$

Similarly, the bigram model is defined as

$$P(x_t^{(u)} = i \mid x_{<t}^{(u)}) = P(x_t^{(u)} = i \mid x_{t-1}^{(u)} = j) = \frac{b_{ij} + \epsilon}{\sum_{k \in \mathcal{I}} (b_{kj} + \epsilon)}, \hspace{1cm} (7)$$

where we added Laplace-smoothing parameterized by $\epsilon$.

**Matrix Factorization.** Methods based on matrix factorization have become a widely used tool for collaborative filtering due to their early success for recommender systems (Koren et al., 2009). Although the low-rank representation of a rating matrix is helpful for sparse static rating data by transferring knowledge between similar users and items, they are less suitable for sequential data. Approaches, such as tensor factorization (Pragarauskas and Gross, 2010; Chi and Kolda, 2012) generalize matrix factorization to more than two dimensions, one of which can be used to represent time under the assumptions that observations per user are temporally aligned. Related to this issue is the a-priori unknown optimal size of time slices for grouping together related observations. These challenges could greatly increase the complexity of the model or of the processing pipeline. Nevertheless, a static method based on matrix factorization is an interesting baseline to compare against in order to study the effect of a collaborative component in isolation. From the sequential data, we construct a matrix of log-counts\(^2\), i.e., we construct a (sparse) matrix $M$ with entries $m_{iu} = \log \left( c_i^{(u)} + 1 \right)$. Then, we compute the low-rank decomposition of $M \approx V^T U$ with $V \in \mathbb{R}^{D \times I}$ and $U \in \mathbb{R}^{D \times U}$ by solving the weighted-$\lambda$-regularized formulation by Zhou et al. (2008):

$$\min_{U,V} \sum_{(i,u):m_{iu} > 0} \left( m_{iu} - v_i^T u_u \right)^2 + \lambda \left( \sum_{u \in \mathcal{U}} d_u \| u_u \|^2 + \sum_{i \in \mathcal{I}} d_i \| v_i \|^2 \right), $$  \hspace{1cm} (8)

where with a slight abuse of notation, we denote by $d_u = \sum_{i \in \mathcal{I}} \mathbb{1}(m_{iu} > 0)$ and $d_i = \sum_{u \in \mathcal{U}} \mathbb{1}(m_{iu} > 0)$. Next-step distributions can then be defined as

$$P(x_t^{(u)} \mid x_{<t}^{(u)}) = P(x_t^{(u)}) = g(z_u), \quad z_u = V^T u_u, \hspace{1cm} (9)$$

$$g(z) = \left\{ \begin{array}{ll} \exp(z_i) \\ \sum_j \exp(z_j) \end{array} \right\}_{\forall i}. \hspace{1cm} (10)$$

\(^2\) We use log counts to combat over-dispersion due to the heavy-tailed nature of the data. Note that as a side effect the output of the softmax (Eq. 9) can then be interpreted as a ratio of pseudo-counts.
This model is collaborative and makes the whole set $\mathcal{I}$ accessible for each user, but it neglects the sequential nature of the data.

**Hidden Markov Model (HMM).** Another natural candidate for comparing against is the HMM. We consider a standard formulation and refer to Rabiner and Juang (1986).

### 2.2. The RNN Language Model (RNNLM)

Generally, RNNs compute a mapping from the input sequence to a corresponding sequence of real-valued hidden state vectors of dimension $D$:

$$
\text{RNN} ([x_1, \ldots, x_T]) = [h_1, \ldots, h_T], \quad h_t \in \mathbb{R}^D.
$$

(11)

The hidden state is a very flexible representation that enables us to use the network for different tasks by defining an appropriate output layer on top. To obtain a multinomial distribution over the next item, we can simply apply a matrix $W_{out} \in \mathbb{R}^{I \times D}$ and pass the output vector $y_t \in \mathbb{R}^I$ through the softmax function $g(\cdot)$ from Eq. 10:

$$
y_t = W_{out} h_t,
$$

(12)

$$
P(x_t \mid x_{<t}) = g(y_t).
$$

(13)

The ability of RNNs to model sequences comes from explicitly representing dependencies between hidden states at different times. In graphical terms, they are a generalization of the directed, acyclic graph structure of feed-forward networks by allowing cycles to represent dependencies on the past state of the network. The RNNLM (Mikolov et al., 2010) uses a simple network architecture\(^3\) that goes back to Elman (1990) and expresses the past dependency through the following recursive definition of the hidden state vectors:

$$
a_t = W_h h_{t-1} + W_{in}\delta_{x_t},
$$

(14)

$$
h_t = f(a_t).
$$

(15)

The recurrence is initialized by a constant vector $h_0$ with small- or zero components. The matrices $W_h \in \mathbb{R}^{D \times D}$ and $W_{in} \in \mathbb{R}^{D \times I}$ are parameters of the RNN and $f(\cdot)$ is a non-linear function that is applied element-wise to its input, such as the logistic sigmoid, the hyperbolic tangent or, more recently, the rectified linear function. The input is presented to the network as one-hot encoded vectors denoted by $\delta_{x_t}$, in which case the corresponding matrix-vector product $W_{in}\delta_{x_t}$ reduces to projecting out the $x_t$-th column of $W_{in}$. Note, that the network can be trivially extended to accept arbitrary side information that characterizes the input at time $t$.

**Parameter Learning.** We focus on how RNNLM processes a single sequence as we will use this procedure as a building block later on. For the loss function Mikolov et al. (2010) use the log-probability of sequences (see Eq. 1), which is differentiable with respect to all parameter matrices. The network is trained using backpropagation through time (BPTT, Williams and Zipser, 1995). BPTT computes the gradient by unrolling the RNN in time and by treating it as a multi-layer feed-forward neural network with parameters tied across every

---

\(^3\) Similarly to Jozefowicz et al. (2015) we will refer to this network as a tanh RNN.
layer and error signals at every layer. For computational reasons, the sequence unrolling is truncated to a fixed size $B$. This is a popular approximation for processing longer sequences computationally more efficiently. This method is summarized in Algorithm 1. To train on a full corpus, Algorithm 1 is used on individual sentences in a stochastic fashion. The learning rule in the original RNNLM is a gradient descent step with a scalar step size. The training process is regularized by using early stopping and small amounts of Tikhonov regularization on the network weights.

Algorithm 1 processSequence()

Input: Sequence $x$, $\theta = \{W_{in}, W_h, W_{out}\}$, batch size $B$

Output: Updated parameters $\theta$

1: $B \leftarrow$ Split $x$ into sub-sequences of length $B$
2: $h_0 \leftarrow \epsilon 1$
3: for $b \in B$ do
4: $h_1, \ldots, h_B \leftarrow$ RNN $(b, h_0; \theta)$ (Forward pass using Eq. 15)
5: $\nabla_{\theta} \log E \leftarrow$ BPTT $(b, h_1, \ldots, h_B; \theta)$ (Backward pass, see Appendix A)
6: $\theta \leftarrow$ LearningRule $(\nabla_{\theta}, \theta)$
7: $h_0 = h_B$
8: end for

2.3. The Collaborative Recurrent Neural Network (C-RNN)

We explained previously that our model needs to represent hidden contextual state that can vary per user:

$$C\text{-RNN} \left( [x_1^{(u)}, \ldots, x_{T_u}^{(u)}] \right) = \begin{bmatrix} h_1^{(u)}, \ldots, h_{T_u}^{(u)} \end{bmatrix}, \quad h_t^{(u)} \in \mathbb{R}^D.$$

(16)

Using a RNNLM-type network per user would result in several problems: the limitations to individual item libraries and excessive, potentially redundant, parameterization, with potentially insufficient data to learn the parameters effectively. Concretely, we would have to handle $O(2DIU + UD^2)$ parameters, where the problem stems from the $IU$ term as in general $D \ll I$.

We propose the following compromise: the input and output parameter matrices $W_{in}$ and $W_{out}$ can be thought of as real-valued embeddings, akin to latent factors in matrix factorization models (Koren et al., 2009) or word embeddings (Mikolov et al., 2013). With this interpretation, we assume that such latent factors embody certain global traits of items, valid across users and static on the time-scale of interest, and we attribute the dynamics to users by maintaining per-user the part of the model responsible for capturing the dynamics, i.e., the relatively small matrix $W_h$. The parameterization is thus reduced to a more tolerable $O(2DI + UD^2)$, i.e., $\theta = \{W_{in}, W_{out}, W_h^{(1)}, \ldots, W_h^{(U)}\}$. Correspondingly, the forward equations become personalized:

4. There are more nuances to their algorithm, but for the purpose of our development this basic exposition is sufficient.
5. For comparison, a latent factor model would typically have $O(D(I + U))$ parameters.
\[ a_t^{(u)} = W_{h_t-1}^{(u)} + W_{in} x_t^{(u)}, \quad (17) \]
\[ h_t^{(u)} = f(a_t^{(u)}), \quad (18) \]
\[ y_t^{(u)} = W_{out} h_t^{(u)}, \quad (19) \]

\[ P(x_t^{(u)} | x_{<t}^{(u)}) = g(y_t^{(u)}). \quad (20) \]

**Regularization and Learning Algorithm.** We train the network by using Algorithm 2 that uses the BPTT procedure from Algorithm 1 as a building block. For the learning rule, we found it more effective to use RMSprop (Tieleman and Hinton, 2012). For the regularization, we use early stopping and dropout (Srivastava et al., 2014), following the insights about its application to RNNs presented by Zaremba et al. (2014). Similarly to Jozefowicz et al. (2015), we initialize the weights by randomly drawing each component i.i.d. from \( N(0, D^{-1}) \).

**Algorithm 2 Collaborative RNN Training**

1. Init \( \theta = \left\{ W_{in}, W_{out}, W_{h_{(1)}}, \ldots, W_{h_{(U)}} \right\} \)
2. repeat
3. \( \mathcal{R} = \text{randperm}(U) \) (process users in random order)
4. for \( u \in \mathcal{R} \) do
5. \( \theta_u \leftarrow \left\{ W_{in}, W_{out}, W_{h_{(u)}} \right\} \)
6. \( W_{in}, W_{out}, W_{h_{(u)}} \leftarrow \text{processSequence} (x^{(u)}, \theta_u, B) \) (Algorithm 1)
7. end for
8. until Early Stopping Criterion holds

**2.4. Variants**

Recently, since the RNNLM was first introduced, great strides have been made in the area of RNNs. Special attention has been paid to the fact that RNNs are known to be difficult to train due to vanishing and exploding gradients (Hochreiter et al., 2001).

Among the approaches to mitigating this issue (Pascanu et al., 2012; Mikolov et al., 2014), architectural modifications to the standard RNN stand out as being easily adoptable and effective in practice. The Long Short-Term Memory (LSTM) cell introduced by Hochreiter and Schmidhuber (1997) is the first and best-known variant, specifically designed to counteract the vanishing-gradient problem by including a gating mechanism to enable the network to retain information in a data-driven way. This method is especially appealing as it requires virtually no modifications to standard stochastic gradient descent-based learning techniques, but this comes at the cost of an increased parameterization.

In this work, we choose to use the Gated Recurrent Unit (GRU, Cho et al., 2014a), a simplified architecture that is similar in spirit to the LSTM cell and that works well in
practice (Jozefowicz et al., 2015). The forward equations for the Collaborative GRU are

\[ z_t^{(u)} = \sigma \left( W_{zh} h_{t-1}^{(u)} + W_z x_t \right), \]  

\[ r_t^{(u)} = \sigma \left( W_{rh} h_{t-1}^{(u)} + W_r x_t \right), \]  

\[ \tilde{h}_t^{(u)} = g(W_h(r_t^{(u)} \circ h_{t-1}^{(u)}) + W_i x_t), \]  

\[ h_t^{(u)} = (1 - z_t^{(u)}) \circ h_{t-1}^{(u)} + z_t^{(u)} \tilde{h}_t^{(u)}. \]

Therefore, the number of parameters is increased by a small factor to \( O(4DI + 3UD^2) \).

2.5. Implementation Details

We implement the learning algorithm by using Theano (Theano Development Team, 2016) and execute the code on NVIDIA Titan X GPUs on Intel Xeon E5-2680 servers with 256 GB of RAM. As we back-propagate on the fixed-sized sub-segments of the original sequence\(^6\), we can unroll the Theano scan operation, trading a minor increase in compilation time for significant speedups in execution.

3. Empirical Evaluation

In this part, we present our experimental findings. We begin by introducing the datasets that we use in Section 3.1 and then address the following points. In Section 3.2, we investigate the influence of the RNN architecture, as well as that of the collaborative aspect of the model in contrast to the user-agnostic version. In Section 3.3, we show the comparison against various baselines. Lastly, in Section 3.4 we shed some light on the difficulty of the problem by characterizing the dependence of the performance on properties of the user profile.

3.1. Datasets

We used two publicly available datasets for our evaluation: Brightkite\(^7\) (BK) and LastFM\(^8\) (LFM).

Brightkite, discontinued in 2011, used to be a location-based social network where users could actively announce (“check in”) their location and find their nearby friends. For our purposes, we focus on the check-in logs consisting of triplets of the form (user id, location id, check-in time).

The LastFM\(^9\) dataset consists of sequences of songs played by a user’s music player collected by using a tracking plug-in. In our evaluation, we consider the sequences of artists.

We apply the following preprocessing steps: As Rendle et al. (2010), we start with a 10-core subset\(^10\) and remove pathologically homogeneous user profiles (e.g., overall reporting

---

6. In our experiments we use a window of length 128.
10. We keep only users with at least 10 observations and items that were consumed at least 10 times.
only a single place but hundreds or thousands of times). We prepare test sets that consist of the last 2.5% of the data clipping their length to the interval [3, 500]. Additionally, in the case of LFM, we restrict the maximum sequence length to 5120, because we did not find qualitative differences beyond that length. The set of items $I$ is taken to be the union of the individual training sequences. The resulting datasets, along with the error of a uniform baseline predictor, are summarized in Table 1. We see that the total number of events of the two datasets differ by an order of magnitude, suggesting that results on the LFM dataset might be more meaningful.

$$
\begin{array}{cccc}
\text{Users} & \text{Items} & \text{Events} & E_{\text{Unif.}} \\
\hline
\text{BK} & 1679 & 23243 & 599618 & 10.05 \\
\text{LFM} & 954 & 48188 & 4320170 & 10.78
\end{array}
$$

Table 1: Summary of the datasets. $E_{\text{Unif.}}$ is the negative log probability assigned to a sequences by the uniform baseline given in Eq. 4.

### 3.2. Comparison of RNN Variants

The aforementioned difficulties in training RNNs has sparked numerous architectural variants of the basic tanh RNN. Understanding their effect on performance is an interesting aspect in its own right with major relevance for practitioners. Therefore, e.g., Jozefowicz et al. (2015) conducted large empirical studies comparing many thousands of variants, focusing on modifications of the original LSTM cell (Hochreiter and Schmidhuber, 1997) in comparison to the GRU (Cho et al., 2014b). Although they did not identify a single best model, they corroborated the observation that such advanced RNN architectures as the GRU and LSTM outperform the tanh RNN. We choose GRUs over LSTMs for their simpler formulation and parameterization and state-of-the-art performance for many tasks, and we compared it to the collaborative tanh RNN (C-tanh). We found that the advantage of the gated version over the basic RNN carries over to our setup.

Furthermore, we compared our collaborative model to the user-agnostic GRU model that we will refer to as pooled RNN (P-RNN). It is conceivable that a RNN with a large-enough number of hidden units can eventually capture even distinct, user-specific patterns. However, we found that for our problem the collaborative RNN, taylored to the structure of the problem, is far more efficient in terms of computation. This is due to the fact that, whilst the collaborative RNN maintains recurrent connection matrices per-user thus more parameters in absolute numbers, each of these matrices can be of smaller dimension. As the learning algorithms are otherwise virtually equivalent, the pooled model incurs a major performance hit. At this point, we do not rule out that the P-RNN can match the collaborative version. But with the large running times required, due to the large number of hidden units required to come close to the performance of the collaborative model, thorough model selection for even higher-dimensional models becomes extremely cumbersome. We summarize these findings in Figure 1.
In a side note, another advantage of the collaborative model, apart from the computational aspect, is that it meets the potential requirement of discovering user specific embeddings.

<table>
<thead>
<tr>
<th></th>
<th>P-GRU</th>
<th>C-tanh</th>
<th>C-GRU</th>
</tr>
</thead>
<tbody>
<tr>
<td>BK</td>
<td>6.45</td>
<td>6.08</td>
<td><strong>6.02</strong></td>
</tr>
<tr>
<td>LFM</td>
<td>4.58</td>
<td>4.90</td>
<td><strong>4.51</strong></td>
</tr>
</tbody>
</table>

Figure 1: **Left:** Errors of different RNN models on the two datasets. **Right:** Running time comparison of a single Epoch of different RNN models on the larger LFM dataset, demonstrating the difference in the number of hidden units for each model to perform well. We found that the P-RNN achieves a similar performance requiring an excessive amount of hidden units ($D = 768$). The negative impact on running time prevented us from exploring even higher-dimensional models.

### 3.3. Baseline Comparison

In this part, we present the comparison of our model against the various baselines that we introduced earlier. The static methods, i.e., those that do not take into account time (unigram, matrix factorization), do not perform very well, whereas the simple bigram model does (Table 2).

For HMM we used the GHMM\(^\text{11}\) library and for MF we used the parallel coordinate descent algorithm by Yu et al. (2012) implemented in the libpmf\(^\text{12}\) library.

<table>
<thead>
<tr>
<th></th>
<th>1-gram</th>
<th>MF</th>
<th>HMM</th>
<th>2-gram</th>
<th>C-GRU</th>
</tr>
</thead>
<tbody>
<tr>
<td>BK</td>
<td>9.53</td>
<td>9.40</td>
<td>8.81</td>
<td>6.73</td>
<td><strong>6.02</strong></td>
</tr>
<tr>
<td>LFM</td>
<td>8.60</td>
<td>8.86</td>
<td>7.66</td>
<td>5.87</td>
<td><strong>4.51</strong></td>
</tr>
</tbody>
</table>

Table 2: Comparison with baseline methods on different datasets.

### 3.4. Characterization of Error

To conclude this section, we examine the error incurred on individual users and relate it to a notion of difficulty. An intuitive way to quantify the difficulty of predicting the behavior of a user is to use the Shannon entropy of the empirical distribution of items. A lower value corresponds to a lower difficulty with a lower bound of 0, which corresponds to a user having consumed the same item at all times. We divided users into three equally sized bins, according to the entropies of their sequences; in Figure 2 we show the distribution of errors in each bin. Unsurprisingly, there is a distinct correlation between our proxy for difficulty and the median error. We see that the C-RNN not only outperforms the closest competing baseline on average, but also in terms of variability of the errors over all bins.

\(^11\) http://ghmm.org
\(^12\) http://www.cs.utexas.edu/~rofuyu/libpmf/
Figure 2: LFM: Distribution of errors per user for three different difficulty regimes. These regimes were found by dividing the users into three equally-sized groups according to the entropy of their sequences. Solid lines indicate the mean errors across all users. The dashed line is the error associated with the uniform baseline. Errors are correlated with difficulty. The RNN-based model appears to not only incur a lower error on average, but also to perform more consistently.

4. Conclusion

We presented a novel collaborative RNN-based model to analyze sequences of user activity useful for modern recommender systems based on specific assumptions and modeling goals. We empirically showed that these models consistently outperform straightforward baseline methods on two real-world datasets. The model is practical in that it can be scaled to large datasets by using techniques such as those of Chen et al. (2015) for the sub-linear evaluation of the output layer in case of large item sets, or those of Recht et al. (2011) to parallelize training.

Future Work. We would like to investigate ways to incorporate into the model the absolute time that, in the case of activity logs, can be assumed to be available. One way could be to augment the input by the time passed since the last event. Another way could be to introduce a special symbol (similar to an end-of-sequence or unknown-word symbol in language models) to indicate a gap.

Acknowledgments We thank Holly Cogliati-Bauereis for careful proofreading.

References


Appendix A. Backpropagation Through Time

We derive the BPTT equations for the gradient computations in Algorithm 1. Recall the forward equations:

\begin{align}
\alpha_t &= W_h h_{t-1} + W_{in} \delta x_t \\
h_t &= f(\alpha_t) \\
y_t &= W_{out} h_t
\end{align}

The purpose here is to find the gradients \( \nabla \alpha_t L, \forall t \). The forward equations reveal the dependencies on a particular \( \alpha_t \) as being two-fold: both \( \ell_t \) and \( \alpha_{t+1} \) contribute to the gradient. Therefore, the gradient \( \nabla \alpha_t L \) is given by (assuming row vectors)

\[
\nabla \alpha_t L = \nabla \alpha_t \ell_t + \sum_{k} \frac{\partial L}{\partial a_{t+1,k}} \frac{\partial a_{t+1,k}}{\partial \alpha_t} = \nabla \alpha_t \ell_t + (\nabla \alpha_{t+1} L)(\Delta a_{t} a_{t+1})
\]

The recursion is initialized by \( \nabla a_{B-1} L = \nabla a_{B-1} \ell_{B-1} \) since only the output \( \ell_{B-1} \) depends on the last activation. The gradient and Jacobian used above are given by:

\[
\nabla \alpha_t \ell_t = \sum_{i} \frac{\partial \ell_t}{\partial y_{t,i}} \frac{\partial y_{t,i}}{\partial \alpha_t} = (\delta x_{t+1} - P_t)^T W_{out} \text{diag}(f'(a_t))
\]

\[
\Delta a_{t} a_{t+1} = W_h \text{diag}(f'(a_t))
\]

Now, the parameter gradients can be easily obtained using \( \nabla W L = \sum_{t,k} \frac{\partial L}{\partial a_{t,k}} \frac{\partial a_{t,k}}{\partial W} \) and

\[
\frac{\partial a_{t,k}}{\partial W_h} = \frac{\partial}{\partial W_h} \delta_{t}^T (W_h h_{t-1} + W_{in} x_t) = \delta_t h_{t-1}^T
\]

\[
\frac{\partial a_{t,k}}{\partial W_{in}} = \frac{\partial}{\partial W_{in}} \delta_{t}^T (W_h h_{t-1} + W_{in} x_t) = \delta_t x_t^T
\]

Plugging in, we get

\[
\nabla W_h L = \sum_t (\nabla a_t L) h_{t-1}^T \quad \nabla W_{in} L = \sum_t (\nabla a_t L) x_t^T
\]

The output weights are not affected by the recurrent structure:

\[
\nabla W_{out} \ell_t = (\nabla y_t \ell_t) h_t^T
\]