# Neural Survival Recommender

How Jing *
LinkedIn, Inc.
Sunnyvale, CA
kublai.jing@gmail.com

Alexander J. Smola
Amazon Machine Learning
Palo Alto, CA
alex@smola.org

## ABSTRACT

The ability to predict future user activity is invaluable when it comes to content recommendation and personalization. For instance, knowing when users will return to an online music service and what they will listen to increases user satisfaction and therefore user retention.

We present a model based on *Long-Short Term Memory* to estimate when a user will return to a site and what their future listening behavior will be. In doing so, we aim to solve the problem of Just-In-Time recommendation, that is, to recommend the right items at the right time. We use tools from survival analysis for return time prediction and exponential families for future activity analysis. We show that the resulting multitask problem can be solved accurately, when applied to two real-world datasets.

## 1. INTRODUCTION

Satisfaction and engagement are keys in retaining happy users and making apps sticky. A prerequisite for this task is the ability to *predict* when a user will act, e.g. *when* he will open an app or visit a music recommendation service, and also *what* he will listen to once he revisits. In other words, the goal is twofold — to predict both the activity and the time of the activity.

This poses an interesting challenge to recommender systems. For instance, the same type of music may not always be equally desirable (Skrillex on a Monday morning vs. Friday night). The ability to estimate future content consumption and the time of such consumption allows us to plan advertising inventories with much higher accuracy and thus with a much better degree of monetization. Likewise, better content prediction allows one to target better content deals (play counts, time, recency) and it ultimately leads to happier and thus more loyal users.

Typically the problem of modeling user activity (*when*) and user interests (*what*) have been handled independently using different technologies (e.g. survival analysis and matrix factorization). Instead, we tackle both problems jointly with a single, deep neural network.

### 1.1 User Activity

Users come back to social network platforms such as Facebook, twitter, LinkedIn to connect to their friends and colleagues; to online shopping stores such as eBay, Alibaba, Amazon when they want to make a purchase, or just to look around for cool new things; to news apps like *New York Times*, *Bloomberg* or *CNN* to get the latest information that they want. Being able to infer *when* the user will come back is crucial for a variety of reasons,

**User Retention.** First and foremost, the goal of any service is to retain its users and to grow its network [25, 6]. Hence it makes sense to be proactive and to find measures how to engage the user more deeply. Information regarding a user's natural propensity to return to the service is valuable in this context.

**User Acquisition.** To grow a service we need to grow the base of engaged users. The ability to estimate the long-term value of an acquired user, i.e. whether he will use the app once or turn into a loyal long-term user, allows us to be much more selective in our targeting process.

**Supply Forecasting.** Mobile app developers who run their own ad network benefit from being able to anticipate future inventory of users. In particular, it is valuable to predict not only total numbers of viewers but the entire *distribution* of user demographics. The goal is to know at what time of day, which type of users from which location is likely to log on and use the app.

### 1.2 User Interests

We wish not only to predict *when* the user returns to our service, but also *how* they act once they return. To provide just-in-time recommendation and to rank the most preferred items at each session, the model will need to capture short-term, recurrent interests for the users. Such patterns usually do not exist in explicit feedback datasets (e.g. Netflix) where users rate the movie once and never come back to rate the same movie again. Yet for websites that provide online music service, for instance, it is very natural to observe a user stick to a small set of particularly indulging songs, every time he returns to the service. Implicit feedbacks in the forms of clicks/listening counts particularly expose recurrent nature of users' behavior on the web.

*Work done while at Carnegie Mellon University.



*WSDM 2017, February 06 - 10, 2017, Cambridge, United Kingdom*



ISBN 978-1-4503-4675-7/17/02…$15.00

DOI: http://dx.doi.org/10.1145/3018661.3018719

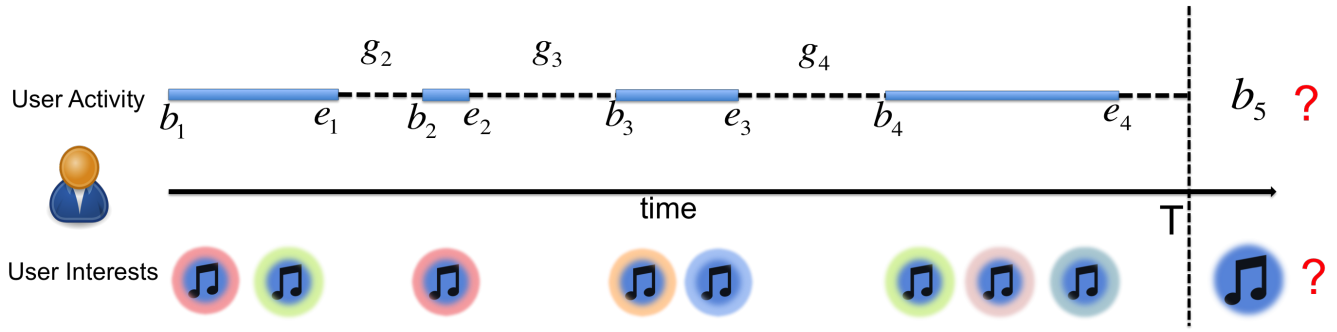


**Figure 1: Modeling return time and user interests. Solid blue lines denote that the user is in-session and dotted black lines mean the user is out-of-session. For each session we observe the *begin* time $b_i$, *end* time $e_i$, and the users' interests (listening behavior in our case). The goal is to predict the next return time $b_5$ (or equivalently, $g_5$) and user's future interests.**

Note that much existing data only provides implicit information about a user's interests. That is, we only have information about the songs played in a playlist and the changes relative to the playlist when a user was dissatisfied with the current songs. Nonetheless, listening information is quite sufficient to get a good estimate of a user's interests.

## 1.3 Background and Notation

Denote by $\mathcal{U} = \{1, 2, ..., m\}$ the set of users and let $T$ be the time point up to which we have the historical data. Let

$$\mathcal{S}_u = \{(b_1, e_1, a_1), (b_2, e_2, a_2), ...(b_{m_u}, e_{m_u}, a_{m_u})\} \quad (1)$$

be the associated session data for user $u$ having $m_u$ sessions in the dataset, where each session is defined by a 3-tuple $s_i = (b_i, e_i, a_i)$ indicating the *begin* time, *end* time, and the *action* taken for session $i$. In our case, $a_i$ is a (sparse) vector of clicks over all the items in session $i$ where each item is in the set $\mathcal{V} = \{1, 2, ..., I\}$.

Moreover, we let $g_i$ be the *gap* of time between the absence of session $i-1$ and return time of session $i$, that is, $g_i = b_i - e_{i-1}$. Since we are always given the previous absence time $e_{i-1}$ when we want to predict the future return time, estimating $g_i$ is equivalent to estimating $b_i$.

Throughout the paper we use $s_u$ to denote a session $s$ for user $u$, if not specified otherwise. We will use **bold** symbol to denote the corresponding embedding vector if the variable in context is of *id-type* (e.g. $u_i$ denotes the user *id* then $\mathbf{u}_i$ represents its low-dimensional embedding vector treated as parameters to be learned). Without loss of generality, we assume that every embedding vector we will consider to be in $\mathcal{R}^d$.

Given such a user's historical sequence $S_u$, the goal is to learn a function for **return time prediction** and **item recommendation** simultaneously: $f : \mathcal{S}_u \mapsto (b_{m_u+1}, a_{m_u+1})$. For a graphical illustration of the problem, see Fig. 1.

## 1.4 Survival Analysis

Survival analysis [26, 16] has its roots in clinical studies where one tries to estimate the probability of *survival* of a patient. More specifically, it studies the probability of surviving at least until time $T$, i.e. $S(T) = \Pr(t_{\text{survival}} \geq T)$, which is called the survival function. By design, once a patient dies it cannot be revived. Moreover, it makes sense to parametrize the *amount* of change relative to the current fraction of survivors. It thus makes sense to model the *rate* of change. In other words, the survival probability satisfies the differential equation

$$\frac{d}{dT} \Pr(t_{\text{survival}} \geq T) = -\lambda(T) \Pr(t_{\text{survival}} \geq T). \quad (2)$$

where $\lambda(t)$ is called the rate function which captures the instantaneous survival rate at time $t$. We assume that we start with a population that is entirely alive, that is, it satisfies $\Pr(t_{\text{survival}} \geq 0) = 1$ .

It is well known that the differential equation can be solved by partial integration, i.e.

$$\Pr(t_{\text{survival}} \geq T) = \exp\Big(-\int_0^T \lambda(t)dt\Big). \quad (3)$$

Hence, if the patient survives until time $T$ and we stop observing at that point, the likelihood is simply given by $\exp\left(-\int_0^T \lambda(T)dt\right)$. This is often referred to as right-censoring since we observe no further data to the "right" of time $T$. Moreover, if an event occurs at time $t^{'} \leq T$, the likelihood is

$$p(t^{'}) = \lambda(t^{'}) \exp\Big(-\int_0^{t^{'}} \lambda(t)dt\Big) \quad (4)$$

Note that there exists a second type of censoring, namely events occurring in the interval $[0, t_0]$, which denotes an initial interval during which we do not record times, commonly referred as left-censoring. While of some theoretical interest, this leads to nonconvex optimization problems and moreover, it does not apply to us, since we have full data available without an initial censoring period.

In our context the 'death' event amounts to the desirable event of a user returning to the service, and the time of survival amounts to the time passing until they use the service again. Immortality amounts to abandonment. And yes, a patient can not revive after (s)he dies, whereas a user surely can come back multiple times.

## 1.5 Contributions

In a nutshell, we make the following contributions in this paper,

- We show that the problem of survival analysis can be framed in a way to be solved by neural networks and

propose to capture the history of users' returning behavior via the hidden states of a recurrent neural network.

- We propose a architecture for learning the return time prediction and recommendations jointly.
- We show the efficacy of our model by comparing against state-of-the-art competitors for both return time prediction and recommendation problems using two real-world datasets.

We begin with a introduction of the related work in Section 2, followed by a detailed description to our proposed model in section 3. We show empirical results in Section 4 and conclude this paper with some remarks in Section 5.

## 2. RELATED WORK

Item recommendation has been a hot topic due to the booming demands for almost every Internet company that makes prediction based on users' data. Notably due to the Netflix prize [2], factorization-based models have been popular even after a decade of its first development, and a lot of the effort has been put into improving the performance under various circumstances [1, 23, 34, 35, 15, 21]. Among those works, temporal dynamics was one of the main themes since after all, users' behavior change over time and it is quite crucial that the model is able to capture such effects [22, 33]. However, most work focused on rather course-grained temporal dynamics where one tries to learn the gradually changing users' preference and/or items' popularity in the units of months or years, instead of short-term, inter-session dependency, which is the focus of this work.

One work that is closely related to ours was the one in [31] which proposed to solve the "session-based" recommendation using recurrent networks. Yet we highlight that the problem there was to solve *within*-session recommendation. That is, within the same session, to make recommendation for the next item that is of users' interests. On the other hand, we are mostly interested in modeling *inter*-session users' interests – given a set of historical session data, make good recommendation for the next session, when user comes back to the service. In fact, our problem scenario is more similar to so called *next-basket recommendation* where each basket corresponds to a session in our case. However, in next-basket recommendation usually only the order for which the events happen matter and the *time* factor was ignored and user returning behavior was not part of the problem that previous work tried to solve. Popular models on this line of research include FPMC [30] and HRM [32].

On the other side, survival analysis has been applied to many mining problems such as modeling the dynamics of cascade [10], document clustering in a streaming fashion when coupled with Dirichlet process [8], discovering disease relations over time [5]. However, to our knowledge it is only up until fairly recent years has the technology been used to predict users' return time to a service [18]. Following that up in [19], similar idea was used to model the dynamics of boredom in activity streams, which was one of the recent work that addressed the problem of "Just-In-Time" recommendation with a solution based on Semi-Hidden Markov Model (sHMM). In [9], the author proposed a more advanced Hawkes process coupled with low-rank penalty to solve similar problem. These two were similar work to ours in a sense that they also tried to predict the return time as well as recommending preferred items to users. What distinguishes our work is that they focused on the returning behavior of a single item, whereas we treat session as the basic working units and try to learn inter-session dynamics. In a nutshell, their models were proposed to capture when a user will return to a particular item, by modeling the temporal dynamics of $(u, v)$ along with the time axis $t$ in the data. Our scenario, on the other hand, tries to predict when a user returns to the service itself and recommends top-K items when they do. Finally, [7] proposed recurrent networks-based models to solve to model a sequence of events, but they did not attempt to solve the recommendation problems and return time prediction jointly.

## 3. NEURAL SURVIVAL RECOMMENDER

### 3.1 Approximation of The Survival Problem

Given a sequence $S_u = \{(b_1, e_1, a_1), , ...(b_{m_u}, e_{m_u}, a_{m_u})\}$ for user $u$, recall that the problem is to estimate the next return time $b_{m_u+1}$, or equivalently the *gap* to the next return $g_{m_u+1}$.

Using the terminology from survival analysis, the likelihood of such sequence can be written as,

$$p(\mathcal{S}_u|T, u) = \Big[\prod_{i=1}^{m_u} \lambda_u(b_i)\Big] \exp\Big(-\sum_{i-1}^{m_u}\int_{e_{i-1}}^{b_i} \lambda_u(t)dt - \int_{e_{m_u}}^{T} \lambda_u(t)dt\Big)$$

Taking the negative logarithm and summing up terms from all users, we get the (negative-loglikelihood) training objective of the entire dataset,

$$\begin{aligned} L_{sur}(\mathcal{S}, T) &= -\sum_u \log p(\mathcal{S}_u|T, u) \\ &= \sum_u \sum_{i=1}^{m_u} \Big(\int_{e_{i-1}}^{b_i} \lambda_u(t)dt - \log \lambda_u(b_i)\Big) + \int_{m_u}^{T} \lambda_u(t)dt \end{aligned}$$

We can then go ahead and minimize such objective using standard optimization method such as gradient descent, except that for non-linear function class $\lambda_u(.)$, such as a neural network, it's very hard to carry out the integration efficiently. We resolve this issue by approximating the integration with a finite sum. We assume that the survival rate $\lambda_u(.)$ is fixed within a interval $\delta t$, turning the integration into a summation. Or equivalently, we are approximating the continuous rate function $\lambda(t)$ by a piece-wise constant function,

$$\int_{e_{i-1}}^{b_i} \lambda_i(t)du \approx \sum_{t=e_{i-1}}^{t\le b_i; t+=\delta t} \lambda_u(t)\delta t \tag{5}$$

Consequently, the optimization problem becomes more tractable as follows,

$$\begin{aligned} L_{sur}(S, T) \approx \sum_u \sum_{i=1}^{m_u} \Big(\sum_{t=e_{i-1}}^{t\le b_i; t+=\delta t} \lambda_u(t)\delta t - \log \lambda_u(b_i)\Big) \\ + \sum_{t=e_{m_u}}^{t\le T; t+=\delta t} \lambda_u(t)\delta t \end{aligned} \tag{6}$$

We can control the trade-off between computational feasibility and the accuracy of approximation by varying $\delta t$. For a online service that aims to predict return time, it should

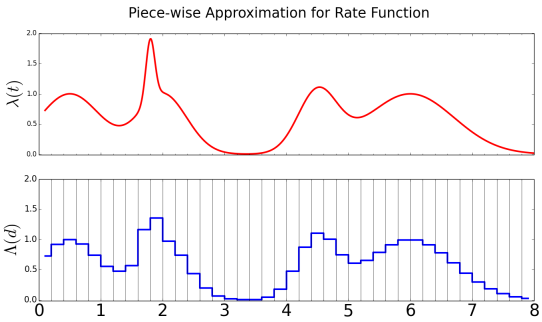


**Figure 2: Illustration of piecewise approximation for the rate function. Top: full continuous function. Bottom: prediction $\Lambda(d)$ where $d = \lceil \frac{t}{\delta t} \rceil$.**

be fine to set $\delta t$ to be something like 30 minutes or even an hour, provided that we do not lose much of the benefits in estimation.

Now let us define $\Lambda_i$ to be the vector which is composed of the concatenation of all the discretized rate output from time $e_{i-1}$ to $e_{i-1} + U_T$ for some cut-off threshold $U_T$. That is,

$$\Lambda_i = [\lambda_u(e_{i-1}), \lambda_u(e_{i-1} + \delta t), \lambda_u(e_{i-1} + 2\delta t)..., \lambda_u(e_{i-1} + U_T)]^T \tag{7}$$

With this representation, the objective in Eq. 6 corresponds to simple arithmetic operations in $\Lambda$. See Algorithm 1 for computing the gradient for this approximate objective w.r.t output vector $\Lambda$.

In our implementation we set $U_T$ to be 180 days which means that we can't predict answer larger than this point, but it is reasonable to say that we have "lost" this user after this point since that translates into 6 months without any session activity for that user. It should be noted that under such truncation, it only amounts to 4320 dimensional output when using a 1-hour interval $\delta t$, which is a pretty modest number compared to what's typically been tackled in language tasks, where the "vocabulary" size could easily exceed tens of thousands.

**Algorithm 1:** Gradient Computation for Eq. 6

**Input** : predicted survival rate $\Lambda$, true gap return time $g$, interval length $\delta t$
**Output**: gradient vector $\nabla\Lambda$

$grad = \mathbf{0}$ ;
d = $\lceil \frac{g}{\delta t} \rceil$ ;
**for** *i=0 : d - 1* **do**
  $grad[i] = \delta t$
**end**
$grad[d] = -\frac{1}{\Lambda[d]} + g - \delta t(d-1)$;
**return** *grad*

## 3.2 Recurrent Survival Networks

### 3.2.1 Long-Short Term Memory Networks

A key requirement in modeling time series is the ability to capture the inherent dynamics in the time series. This is usually accomplished by a latent variable model, such as a Hidden Markov Model [29] or a nonlinear Kalman Filter [17].

Rather than using an implicit dynamic defined by a graphical model we can also resort to a latent autoregressive model. In it, the latent state is updated via $h_{t+1} = f(h_t, o_t)$ and the observable state satisfies $o_{t+1} = g(h_{t+1}, o_t)$ for some data $o_t$. Here $f$ and $g$ are nonlinear functions that can be learned from data. In deep learning they are commonly known as recurrent neural networks.

A particularly useful variant is the *Long-Short Term Memory* [14] with specially designed units to avoid the problem of vanishing gradients. It proves to work well across many problems when the data is sequential, e.g. speech recognition, machine translation, or image captioning. See the reviews [13, 12] for further details and references. It thus serves as our building block to learn the recurrent patterns residing naturally in users' behavior. There are several variants of LSTM. In this paper we use the following equations for our hidden units,

$$\begin{aligned}
[i_t, f_t, o_t] &= \sigma(x_t W + h_{t-1} R + b) \\
z_t &= \tanh(x_t W_z + h_{t-1} R_z + b_z) \\
c_t &= i_t \odot z_t + f_t \odot c_{t-1} \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned} \tag{8}$$

where $x_t$ is the input data, $h_t$ being the hidden activations for time step $t$; $i_t, f_t, o_t$ serve as *input gate, forget gate, output gate* respectively. LSTMs have the advantage of being able to model long-term dependencies, which was not possible in traditional recurrent neural networks due to the problems of vanishing gradients.

### 3.2.2 Survival Analysis with LSTM

Our LSTM network is fed with all the $g_{j,j<i}$ and tries to predict $\Lambda_i$. To deal with the continuous nature of $g_j$, we again quantize them into a $\delta t$ interval. That is, each $g_j$ is mapped into an *id* by the following function $ind(g_j) = \lceil \frac{g_j}{\delta t} \rceil$ and we introduce one embedding parameter matrix $\mathbf{G} \in \mathcal{R}^{N_G \times d}$, such that each $ind(g_j)$ has its own embedding vector $\mathbf{g}_j$, and there are in total $N_G$ such vectors. We treat $\mathbf{g}_j$ as the input to our LSTM and we get the hidden states, denoted as $h_i$ after feeding through $\mathbf{g}_{j,j<i}$ and follow the LSTM equation in (8). We then predict $\Lambda_i$ using this representation,

$$\Lambda_i = softrelu(W_{h\lambda} h_i + b_\lambda) \tag{9}$$

where $softrelu(x) = \log(1 + \exp(x))$ is used to ensure that the rate is always positive and $W_{h\lambda}, b_\lambda$ are the weight matrix and bias vector respectively. This modeling choice is reminiscent of the Hawkes process which uses a kernel function to capture how past activities affect the future survival rate [28],

$$\lambda(t) = \mu + \alpha \sum_{t_i < t} \exp(-\beta(t - t_i)) \tag{10}$$

Our RNN is more powerful in capturing how past activities affect the future through hidden states.

### 3.2.3 hour2vec – Learning the Semantics of Time

Gap time captures the recurrent patterns residing in users' visit behavior. There is another important source of time information we brought into our network reflected by the observation that different time of day, and different day of

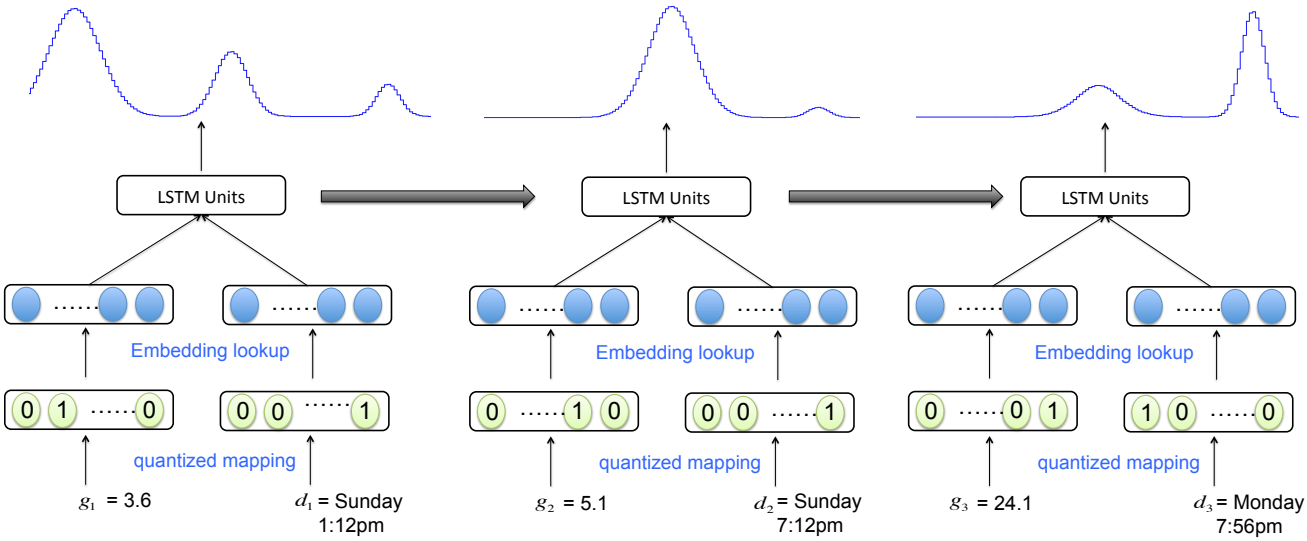


**Figure 3: Solving return time prediction problems using LSTM. At each session, previous gap time and the hour *id* are fed into the model. LSTM captures the recurrent patterns over the past and predicts the future survival rate for the next session.**

week carries different meaning. For example, the probability that the user will return within the next 6 hours is drastically different if her previous absence time was 1:00am compared to, say 8:00pm. It's highly likely that user will go to sleep after the 1:00am visit, whereas user may come back soon after the 8:00pm visit.

To capture such effect, we again quantize the time frame of a week by hour. There is 168 hours in a week from Monday 00:00am to Saturday 23:59pm. Each such hour is assigned an *id*. We then introduce an embedding matrix $\mathbf{D} \in \mathcal{R}^{168 \times d}$ where each hour-of-the-week has its corresponding embedding vector. We denote $d_i$ as the corresponding hour-of-week *id* for $e_i$. For instance, if the user left on Monday 2:03am, $d_i$ will be equal to 2 (assuming the index starts from 0).

## 3.3 Modeling Recurrent Interests

Learning recurrent users interests fits nicely into our recurrent networks framework. We simply feed in the past session's actions for a user and let LSTM figures out how to predict the actions for the next session. We use negative Poisson log-likelihood as training objective,

$$L_{rec}(S,T) = -\sum_{u}\sum_{i=1}^{m_u}\sum_{v=1}^{I} a_i[v]\log(\hat{a}_i[v]) - \log a_i[v]! - \hat{a}_i[v] \tag{11}$$

with slight abuse of notation that $a_i[v]$ denotes the $v$-th component in the action vector $a_i$. Again, *softrelu* is used to ensure that the predicted counts is never negative.

## 3.4 Personalization and Multi-Task Training

Deep networks are very flexible for us to mix and match different input signal and capture their correlations through hidden layers. Latent representation of users, often used in factor model, can be easily incorporated into our network just like other embedding representations do. We introduce an *user embedding* parameters matrix $\mathbf{U} \in \mathcal{R}^{|\mathcal{U}| \times d}$ where each user has his/her latent representation.

Finally putting things together we have our input vector formed via concatenation for all the input signals,

$$\mathbf{x}_i = [\mathbf{g}_{i-1}; \mathbf{d}_{i-1}; \mathbf{a}_{i-1}; \mathbf{u}] \tag{12}$$

with slight caveat that $\mathbf{a}_{i-1}$ here is not the original counts vector. Instead, we have it going through another hidden layer of transformation before concatenating with other data modalities. This was inspired by [27] where modalities with different properties (sparse v.s. dense in our case) do not work well by direct concatenation.

Having the two tasks at hand and the input from the two perspectives (i.e. recurrent visiting patterns and action), we consider solving them jointly by learning to predict the two output vectors $\hat{\Lambda}_i, \hat{a}_i$. The parameters in LSTM and all the embedding matrices $\mathbf{U}, \mathbf{G}, \mathbf{D}$ are then trained by minimizing the combined objective,

$$\min . \sum_{u}(1-\alpha)L_{sur}(S_u,T) + \alpha L_{rec}(S_u,T) \tag{13}$$

where $\alpha$ controls the relative importance. Both objectives are differentiable and the parameters can be learned using stochastic gradient descent.

### *3-Way Factored Units.*

In addition to the additive model, which was implemented via simple concatenation of input vectors, a 3-way factored model can capture additional multiplicative effects between variables [20]. By factoring a full tensor into three matrices, our 3-way factored model adds on this additive model through gating the user embedding $\mathbf{u}$ with the output of the hidden activations $h$. This is achieved via three additional

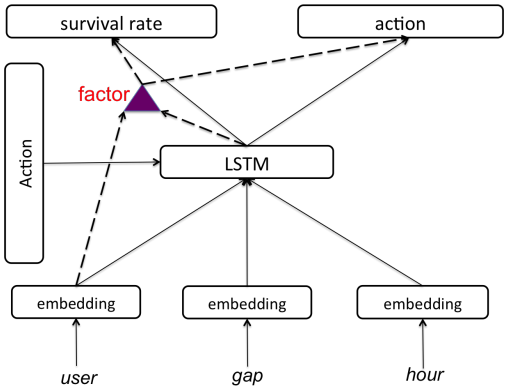


**Figure 4: Final architecture for our Neural Survival Recommender. Different data modalities are aggregated then fed into the LSTM layer. The model then predicts survival rate and the action for the next session.**

matrices $W_{fu}, W_{fh}, W_{fr}$ with the following equations,

$$\mathcal{F}_u = W_{fu}\mathbf{u}$$
$$\mathcal{F}_h = W_{fh}h \tag{14}$$
$$a = softrelu\Big(W_{rec}h + W_{fr}\Big(\mathcal{F}_u \circ \mathcal{F}_h\Big) + b_{rec}\Big) \tag{15}$$
$$\Lambda = softrelu\Big(W_{sur}h + W_{fs}\Big(\mathcal{F}_u \circ \mathcal{F}_h\Big) + b_{sur}\Big) \tag{16}$$

where $\circ$ is the Hadamard product. We show this final architecture in Fig. 4.

# 4. EXPERIMENTS

## 4.1 Datasets

We used two real-world datasets to evaluate our models, namely, *lastfm*, and *tianchi-mobile*. These two are publicly available datasets with timestamps of users' actions on the recommended items that fit our scenario:

- *lastfm* is a music recommendation dataset that contains roughly 1,000 users with entire listening history of 19,150,868 (user, time, artist,song) tuples from 2004 to 2009. We use the top 20000 most popular artists, decided by the received listening counts, as the recommendation targets.
- *tianchi-mobile* collects users' behavioral data on Alibaba's M-Commerce platforms. It records users' actions on items with timestamps, rounded to the nearest hour. We use the item category as our recommendation targets. The dataset contains 10000 users and 8917 item categories.

It should be noted that most explicit datasets also contain timestamps (e.g. Netflix, Yahoo! music). But such timestamps do not indicate the time for which the purchase/click actions were performed, hence it's not reasonable to use that for return time predictions, as users may rate items way after actual consumption.

For both datasets, there is no clear notion of session provided, as it is hard to assert that the user has left the service by the impression times (e.g. scrolling the webpage incurs different impressions, but obviously it's the same session). We thus segment the impressions into sessions by defining a minimum gap – if two events are separated apart by more than such quantity, we say that the two events belong to different sessions. For both datasets, we set it to be 1 hour. It means that, say at $t_i$, we record a listening event for a user, if the next event happens at $t_j$ such that $t_j - t_i \leq 1.0$, then two events belong to the same session. Otherwise, they belong to the different sessions. This way of defining a session was also used in [19]. The choice of 1 hour is not of any particular reason, and it should depend on business need to set this parameter for real production applications.

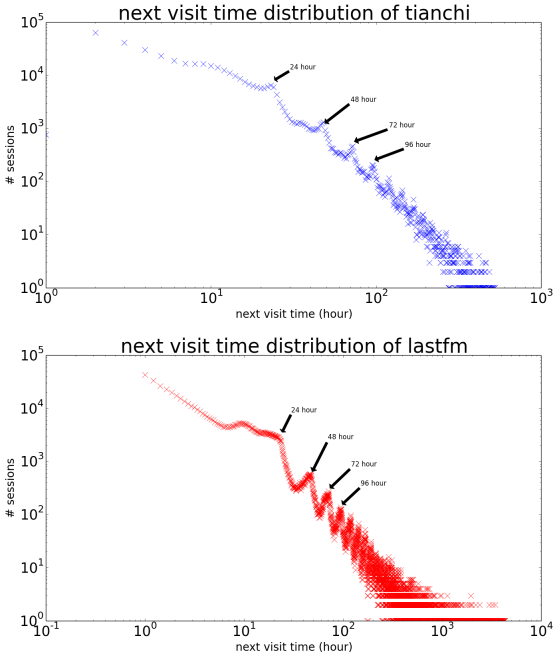


**Figure 5: Distribution of visiting behavior of the two datasets. Top - *tianchi-mobile*, bottom - *lastfm*.**

## 4.2 Evaluation

We split the data into training/testing set by selecting a time point $T$ which is assumed to be the point where we have the data up to (i.e. right-censored). This mimics the real-world situation where we observed users' behavior from the past up to the current time, and the model is asked to predict the future given all the available data at hand. For *last.fm*, we set $T$ to be 20090101, which results in 632,304 training sessions and 97,395 testing sessions; for *tianchi-mobile*, we set $T$ to be 20141210 resulting in 289,399 and 122,119 training and testing sessions respectively.

We use *mean absolute error* (MAE), measured in hour, to assess the model performance for return time prediction task. To predict a single number out of the predicted $\Lambda$, we search for a point $T^{'}$ where the survival function $S(T^{'}) = Pr(t_{return} \geq T^{'}) = 0.5$. That is, we predict the median of the survival distribution (one might also use mean as the prediction). For recommendation, we use precision at 15 (Prec@15), Recall@15, mean reciprocal rank (MRR) and mean average rank (MAR) to evaluate the quality of the top-K recommendation. We compute each metric for each

session and take the average over the number of test sessions.

Assume that there are $|S_{test}|$ sessions in the test set. For a user $u$ in a test session $s$, let $ret(u,s), b_s^u$ be the predicted, true return time, $rank(u,v,s)$ be the rank that the model produces for item $v$ and $a_s^u$ denotes the set of user $u$'s action on in that session. The five metrics are then calculated as follows,

$$MAE = \frac{1}{|S_{test}|} \sum_{u,s \in S_{test}} |ret(u,s) - b_s^u| \quad (17)$$

$$Prec@k = \frac{1}{|S_{test}|} \sum_{u,s \in S_{test}} \sum_{v \in a_s^u} \frac{\mathcal{I}(rank(u,v,s) \leq K)}{K} \quad (18)$$

$$Recall@k = \frac{1}{|S_{test}|} \sum_{u,s \in S_{test}} \sum_{v \in a_s^u} \frac{\mathcal{I}(rank(u,v,s) \leq K)}{|a_s^u|} \quad (19)$$

$$MRR = \frac{1}{|S_{test}|} \sum_{u,s \in S_{test}} \frac{1}{|a_s^u|} \sum_{v \in a_s^u} \frac{1}{rank(u,v,s)} \quad (20)$$

$$MAR = \frac{1}{|S_{test}|} \sum_{u,s \in S_{test}} \frac{1}{|a_s^u|} \sum_{v \in a_s^u} rank(u,v,s) \quad (21)$$

where $K = 15$ in our experiments.

## 4.3 Baselines

We compare our model with several strong baseline methods, as described below:

**Hierarchical Poisson Factorization (HPF)** [11] improves upon the traditional matrix factorization model by modeling the observed data as Poisson distributed instead of standard Gaussian variable. It has more theoretical benefits for the implicit feedbacks datasets as well as superior performance and scalability.

**Dynamic Poisson Factorization (DPF)** [3] DPF is a variant of Poisson factorization where temporal dynamics was brought into the model. A state-space process is superposed on the user latent factor and item latent factor to capture the evolving preferences/attributes for user and items respectively.

**Hierarchical Representation Model (HRM)** [32] is a neural network based model that takes the items representation from the previous session, and preforms pooling to get feature vector which is then used to predict a ranked list for the next session.

**Linear Regression with Huber Loss (LR-H)** Instead of survival analysis technique, we directly regress on the return time using linear regression. We used Huber loss as we found it performs much better due to its robustness to outliers.

**Poisson Process (PP)** Poisson process is a baseline predictor for survival analysis where the rate $\lambda$ is assumed to be a simple constant function.

**Hawkes Process (HP)** Hawkes process superposes on the Poisson process with a self-exciting recurrence. We use the following intensity function in our experiments,

$$\lambda(t) = \lambda + \lambda_u + \sum_{t_j < t} \alpha_u \exp(-\beta_u (t - t_j)) \quad (22)$$

where $\lambda$ is a shared base rate across all users and $\lambda_u, \alpha_u, \beta_u$ are parameters to be learned for each user.

| | *Prec@15* | *Recall@15* | *MRR* | *MAR* |
|---|---|---|---|---|
| HPF | 0.092 | 0.290 | 0.110 | 366.01 |
| DPF | 0.0836 | 0.2369 | 0.0825 | 506.86 |
| HRM | 0.0925 | 0.3064 | 0.1261 | 418.83 |
| Our Model | **0.107** | **0.365** | **0.167** | **356.02** |

**Table 1: Recommendation results on *tianchi-mobile* dataset for different models.**

| | *Prec@15* | *Recall@15* | *MRR* | *MAR* |
|---|---|---|---|---|
| HPF | 0.0612 | 0.124 | 0.044 | 1608.88 |
| DPF | 0.0643 | 0.134 | 0.0562 | 1753.90 |
| HRM | 0.0840 | 0.2134 | 0.0923 | 1263.93 |
| Our Model | **0.100** | **0.247** | **0.113** | **1077.01** |

**Table 2: Recommendation results on *lastfm* dataset for different models.**

| | *tianchi-mobile* | *lastfm* |
|---|---|---|
| Poisson Process | 8.99 | 17.12 |
| LR-Huber | 8.37 | 17.12 |
| Hawkes Process | 8.26 | 15.64 |
| Our Model | **7.53** | **14.66** |

**Table 3: Return time prediction results measured in MAE on *tianchi-mobile* and *lastfm* for different models.**

We do not compare against FPMC [30] and NMF as they were already shown to perform worse than our competitors.

## 4.4 Accuracy

For our own implementation of deep networks, we use *mxnet*, a scalable deep learning library [4]. We train our model with 50 iterations over the training set with learning rate 0.001. We use a single LSTM layer with the number of hidden units set to 300 and $\delta t = 1.0$. The size of the embedding and the dimension of all the factorization-based competing methods are also set to 300. For all the compared models, we tune the parameters to give the best performance.

The results on recommendation tasks are presented in table 1 and 2 and the result for return time prediction is summarized in table 3, where our model outperforms competitors by sizable margins.

## 4.5 Performance Analysis

### *Multitask Learning.*

We study the effect of trading off both tasks, i.e. return time prediction vs. interest estimation. The trade-off is controlled by $\alpha$ in (13). Fig. 6 shows that it is possible to improve accuracy by a combination between both tasks over a wide range of parameters. In fact, a default of $\alpha = \frac{1}{2}$ yields good results on both datasets.

### *Performance with Varying Observed Sessions.*

Intuitively, the more training sessions we have, the better we can predict future behavior for a user. In Fig. 7, we show varying performance with different number of sessions observed for a user in the training period. By bucketizing

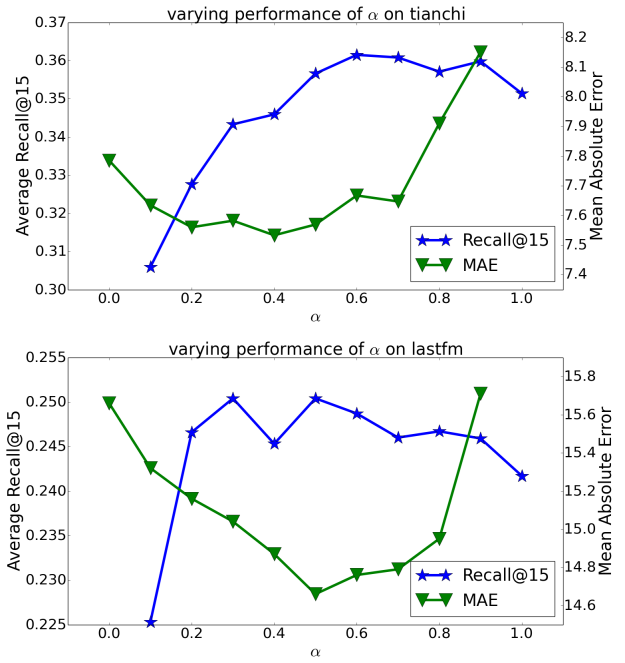


**Figure 6: Average Recall@15 and MAE for a trade-off between return prediction and relevance for recommendation, as parametrized by $\alpha$. Top - *tianchi-mobile*, bottom - *lastfm*.**

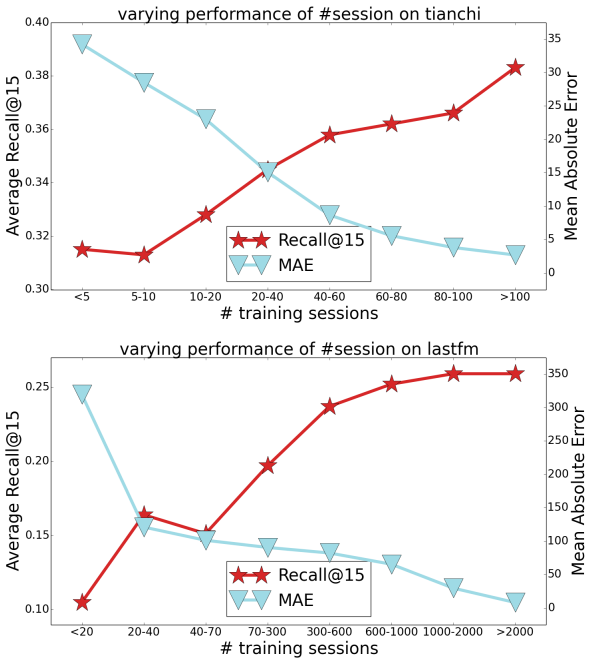


**Figure 7: Performance on Recall@15 and MAE for varying number of observed training sessions. Top - *tianchi-mobile*, bottom - *lastfm*.**

users into different bins, the performance for that bucket is then averaged to produce a single number.

### *Performance with Varying Gap Return Time.*

We further investigate the effect of gap return time ($g_{i-1}$ for the $i$th session) on the performance by showing the performance on varying such quantity in Fig. 8. We observe that the smaller the $g_{i-1}$ is, the better we can predict user's behavior well. This is not surprising as the recurrent interests of user has to do with how long it has been after the previous event. The closer the two sessions, the more dependent they should be. As show in the figure, if two session are separated for greater than 500 hours, temporal information becomes almost useless as previous events barely correlates to what user will do in this session.

### *Predicted Survival Function.*

We show several examples of how our model is doing on predicting the survival function $S(T) = Pr(t_{survival} \geq T)$, which is approximated by $\exp\Big(-\sum_0^T \lambda(t)\delta t\Big)$. In fig 9, survival function, true/predicted return time as well as the mean absolute error are presented for the lastfm dataset.

## 5. CONCLUSION AND FUTURE WORK

We have described an LSTM-based model to learn recurring user activity activities by learning the inter-session dependency. We performed multitask learning on the return time prediction and the item recommendation and we achieved significant improvements over strong baselines.

There are at least two future directions to consider. First, how to model both inter-session relations *and* intra-session

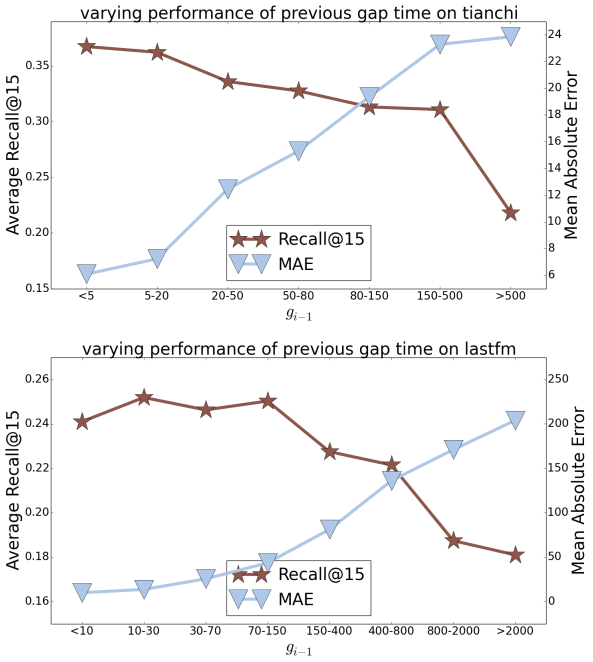


**Figure 8: Performance on Recall@15 and MAE for varying gap return time. Quite unsurprisingly, the longer it takes for a user to return, the less confident we are in the user's interest. Top - *tianchi-mobile*, bottom - *lastfm*.**

relations? We would like to learn not only a ranked list for each session but also the *sequence of songs* within a session.

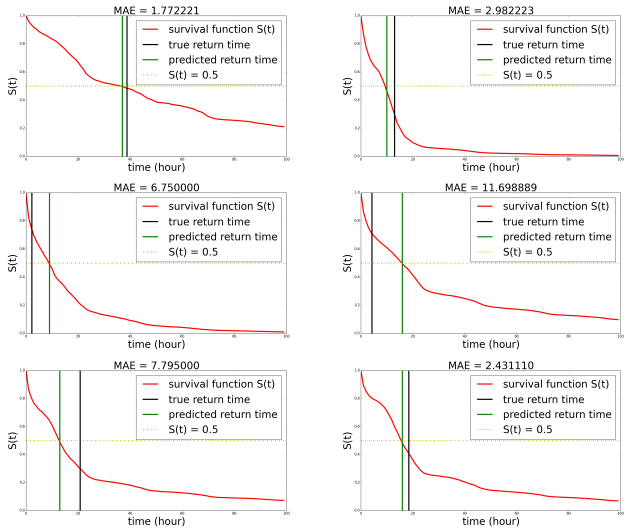


**Figure 9: Six examples from lastfm of model's prediction as well as true return time. Predicted return time corresponds to the time $t$ where $S(t) = 0.5$.**

Second, it will be interesting to see whether we can further improve the performance by including the global temporal dynamics into the model and not just modeling inter-session dynamics. That is, if an artist is very popular during some period of time, do users' interests shift due to this global trend? We leave these to future research.

## 6. REFERENCES


[1] L. Baltrunas and X. Amatriain. Towards time-dependant recommendation based on implicit feedback, 2009.

[2] J. Bennett, S. Lanning, and N. Netflix. The netflix prize. In *In KDD Cup and Workshop in conjunction with KDD*, 2007.

[3] L. Charlin, R. Ranganath, J. McInerney, and D. M. Blei. Dynamic poisson factorization. *CoRR*, 2015.

[4] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, 2015.

[5] E. Choi, N. Du, R. Chen, L. Song, and J. Sun. Constructing disease network and temporal progression model via context-sensitive hawkes process. In *2015 IEEE International Conference on Data Mining, ICDM 2015, Atlantic City, NJ, USA, November 14-17, 2015*, pages 721–726, 2015.

[6] C. Danescu-Niculescu-Mizil, R. West, D. Jurafsky, J. Leskovec, and C. Potts. No country for old members: User lifecycle and linguistic change in online communities. In *Proceedings of the 22nd international conference on World Wide Web*, pages 307–318. International World Wide Web Conferences Steering Committee, 2013.

[7] N. Du, H. Dai, R. Trivedi, U. Upadhyay, M. Gomez-Rodriguez, and L. Song. Recurrent marked temporal point processes: Embedding event history to vector. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1555–1564, New York, NY, USA, 2016. ACM.

[8] N. Du, M. Farajtabar, A. Ahmed, A. J. Smola, and L. Song. Dirichlet-hawkes processes with applications to clustering continuous-time document streams. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, 2015.

[9] N. Du, Y. Wang, N. He, and L. Song. Time sensitive recommendation from recurrent user activities. In *NIPS*, 2015.

[10] M. Gomez-Rodriguez, J. Leskovec, and B. Schölkopf. Modeling information propagation with survival theory. *CoRR*, abs/1305.3616, 2013.

[11] P. Gopalan, J. M. Hofman, and D. M. Blei. Scalable recommendation with hierarchical poisson factorization. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence, UAI 2015, July 12-16, 2015, Amsterdam, The Netherlands*, 2015.

[12] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[13] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber. LSTM: A search space odyssey. *CoRR*, abs/1503.04069, 2015.

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

[15] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, pages 263–272, 2008.

[16] J. G. Ibrahim, M.-H. Chen, and D. Sinha. *Bayesian survival analysis*. Wiley Online Library, 2005.

[17] R. Kalman. A new approach to linear filtering and prediction problems. 82:35–45, 1960.

[18] K. Kapoor, M. Sun, J. Srivastava, and T. Ye. A hazard based approach to user return time prediction. In *Knowledge discovery and data mining*, pages 1719–1728. ACM, 2014.

[19] K. Kapoor, K. uian, J. Srivastava, and P. Schrater. Just in time recommendations: Modeling the dynamics of boredom in activity streams. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, 2015.

[20] R. Kiros, R. Salakhutdinov, and R. S. Zemel. Unifying visual-semantic embeddings with multimodal neural language models. *CoRR*, 2014.

[21] N. Koenigstein, G. Dror, and Y. Koren. Yahoo! music recommendations: Modeling music ratings with temporal dynamics and item taxonomy. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, pages 165–172, 2011.

[22] Y. Koren. Collaborative filtering with temporal dynamics. In *Knowledge discovery and data mining KDD*, pages 447–456, 2009.

[23] H. Ma, I. King, and M. Lyu. Learning to recommend with social trust ensemble. In *ACM SIG Information Retrieval*, pages 203–210, New York, NY, USA, 2009. ACM.

[24] L. v. d. Maaten and G. Hinton. Visualizing

high-dimensional data using t-sne. *jmlr*, 9:2579–2605, 2008.

[25] J. McAuley and J. Leskovec. Hidden Factors and Hidden Topics: Understanding Rating Dimensions with Review Text. In *Proceedings of the 7th ACM Conference on Recommender Systems*, pages 165–172, 2013.

[26] R. G. Miller Jr. *Survival analysis*, volume 66. John Wiley & Sons, 2011.

[27] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng. Multimodal deep learning. 2011.

[28] T. Ozaki. Maximum likelihood estimation of hawkes' self-exciting point processes. *Annals of the Institute of Statistical Mathematics*, 1979.

[29] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 1989.

[30] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, 2010.

[31] Y. K. Tan, X. Xu, and Y. Liu. Improved recurrent neural networks for session-based recommendations. *CoRR*, 2016.

[32] P. Wang, J. Guo, Y. Lan, J. Xu, S. Wan, and X. Cheng. Learning hierarchical representation model for nextbasket recommendation. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2015.

[33] L. Xiong, X. Chen, T.-K. Huang, J. G. Schneider, and J. G. Carbonell. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *SDM*, pages 211–222. SIAM, 2010.

[34] S.-H. Yang, B. Long, A. J. Smola, H. Zha, and Z. Zheng. Collaborative competitive filtering: learning recommender using context of user choice. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 295–304. ACM, 2011.

[35] Y. Zhang, A. Ahmed, V. Josifovski, and A. J. Smola. Taxonomy discovery for personalized recommendation. In *Proceedings of the 7th ACM international conference on Web search and data mining*, 2014.