Deep Coevolutionary Network: Embedding User and Item Features for Recommendation

Hanjun Dai*
Georgia Institute of Technology
hanjundai@gatech.edu

Yichen Wang*
Georgia Institute of Technology
yichen.wang@gatech.edu

Rakshit Trivedi
Georgia Institute of Technology
rstrivedi@gatech.edu

Le Song
Georgia Institute of Technology
lsong@cc.gatech.edu

ABSTRACT

Recommender systems often use latent features to explain the behaviors of users and capture the properties of items. As users interact with different items over time, user and item features can influence each other, evolve and co-evolve over time. The compatibility of user and item's feature further influence the future interaction between users and items.

Recently, point process based models have been proposed in the literature aiming to capture the temporally evolving nature of these latent features. However, these models often make strong parametric assumptions about the evolution process of the user and item latent features, which may not reflect the reality, and has limited power in expressing the complex and nonlinear dynamics underlying these processes.

To address these limitations, we propose a novel deep coevolutionary network model (DeepCoevolve), for learning user and item features based on their interaction graph. DeepCoevolve use recurrent neural network (RNN) over evolving networks to define the intensity function in point processes, which allows the model to capture complex mutual influence between users and items, and the feature evolution over time. We also develop an efficient procedure for training the model parameters, and show that the learned models lead to significant improvements in recommendation and activity prediction compared to previous state-of-the-arts parametric models.

KEYWORDS

Deep coevolutionary networks, Point processes, Time-sensitive recommendation

ACM Reference format:
Hanjun Dai*, Yichen Wang*, Rakshit Trivedi, and Le Song. 2017. Deep Coevolutionary Network: Embedding User and Item Features for Recommendation. In *Proceedings of ACM conference, Halifax, Canada, August 2017 (KDD'17),* 9 pages.
DOI: 10.475/123_4

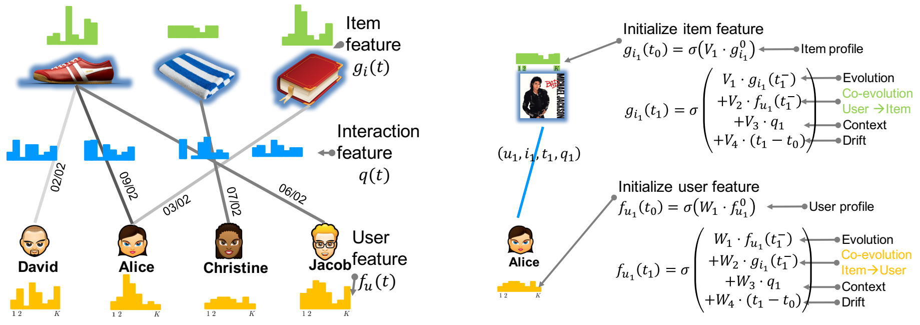
1 INTRODUCTION

Making proper recommendation of items to users at the right time is a fundamental task in e-commerce platforms and social service websites. The compatibility between user's interest and item's property is a good predictive factor on whether the user will interact with the item in future. Conversely, the interactions between users and items further drives the evolution of user interests and item features. As users interact with different items, users' interests and items' features can also co-evolve over time, *i.e.*, their features are intertwined and can influence each other:

- From *User* to *item*. In discussion forums such as Reddit, although a group (item) is initially created for statistics topics, users with very different interest profiles can join this group. Hence, the participants can shape the features of the group through their postings. It is likely that this group can finally become one about deep learning if most users discuss about deep learning.
- From *Item* to *user*. As the group is evolving towards topics on deep learning, some users may become more interested in such topics, and they may participate in other specialized groups. On the contrary, some users may gradually gain interests in math groups, lose interests in statistics group.

Such coevolutionary nature of user and item features raises very interesting and challenging questions: How to model coevolutionary features? How to efficiently train such models on large scale data? There are previous attempts which divide time into epochs, and perform tensor factorization to learn the latent features [7, 25, 38]. These methods are not able to capture the fine grained temporal dynamics of user-item interactions, and can not answer the query related to time of interaction. Recent, point processes which treat event times as random variables have emerged as a good framework for modeling such temporal feature evolution process [13, 35]. However, these previous work make strong parametric assumptions about the functional form of the generative processes, which may not reflect the reality, and is not accurate enough to capture the complex and *nonlinear* co-evolution of user and item features in real world.

To address the limitation in previous point process based methods, we propose a novel deep coevolutionary network model (DeepCoevolve) which defines point process intensities using recurrent neural network (RNN) over evolving interaction networks. Our model can generate an effective representation/embedding of the underlying user and item latent feature without assuming a fixed parametric forms in advance. Figure 1 summarizes our framework. In particular, our work makes the following contributions:


Item feature $g_i(t)$
Interaction feature $q(t)$
User feature $f_u(t)$
02/02
09/02
03/02
07/02
06/02
David
Alice
Christine
Jacob
Initialize item feature
$g_{i_1}(t_0) = \sigma(V_1 \cdot g^0_{i_1})$
Item profile
$g_{i_1}(t_1) = \sigma \begin{pmatrix} V_1 \cdot g_{i_1}(t_1^-) \\ +V_2 \cdot f_{u_1}(t_1^-) \\ +V_3 \cdot q_1 \\ +V_4 \cdot (t_1 - t_0) \end{pmatrix}$
Evolution
Co-evolution User → Item
Context
Drift
(u_1, i_1, t_1, q_1)
Initialize user feature
$f_{u_1}(t_0) = \sigma(W_1 \cdot f^0_{u_1})$
User profile
$f_{u_1}(t_1) = \sigma \begin{pmatrix} W_1 \cdot f_{u_1}(t_1^-) \\ +W_2 \cdot g_{i_1}(t_1^-) \\ +W_3 \cdot q_1 \\ +W_4 \cdot (t_1 - t_0) \end{pmatrix}$
Evolution
Co-evolution Item→User
Context
Drift
Alice

(a) (b)

Figure 1: Model illustration. (a) User-item interaction as evolving bipartite graph. Each edge stands for a tuple and contains the information of user, item, interaction time, and interaction feature. (b) Deep coevolutionary feature embedding processes. The embeddings of users and items are updated at each event time, by a nonlinear activation function $\sigma(\cdot)$ and four terms: self evolution, co-evolution, context (interaction feature), and self drift.

- **Novel model.** We propose a novel deep learning model that captures the *nonlinear* coevolution nature of users' and items' embeddings in a nonparametric way. It assigns an evolving feature embedding process for each user and item, and the co-evolution of these latent feature processes is modeled with two parallel components: (i) *item* → *user* component, a user's latent feature is determined by the nonlinear embedding of latent features of the items he interacted with; and (ii) *user* → *item* component, an item's latent features are also determined by the latent features of the users who interact with the item.
- **Efficient Training.** We use RNN to parametrize the interdependent and intertwined user and item embeddings. The increased flexibility and generality further introduces technical challenges on how to train RNN on the *evolving networks*. The co-evolution nature of the model makes the samples inter-dependent and not identically distributed, which is significantly more challenging than the typical assumptions in training deep models. We propose an efficient stochastic training algorithm that makes the BTPP tractable in the co-evolving network.
- **Strong performance.** We evaluate our method over multiple datasets, verifying that our method leads to significant improvements in user behavior prediction compared to state-of-the-arts. It further verifies the importance of using nonparametric point process in recommendation systems. Precise time prediction is especially novel and not possible by most prior work.

2 BACKGROUND ON TEMPORAL POINT PROCESSES

A temporal point process [1, 8, 9] is a random process whose realization consists of a list of discrete events localized in time, $\{t_i\}$ with $t_i \in \mathbb{R}^+$. Equivalently, a given temporal point process can be represented as a counting process, $N(t)$, which records the number of events before time t. An important way to characterize temporal point processes is via the conditional intensity function $\lambda(t)$, a stochastic model for the time of the next event given all the previous events. Formally, $\lambda(t)\mathrm{d}t$ is the conditional probability of observing an event in a small window $[t, t + \mathrm{d}t)$ given the history $\mathcal{H}(t)$ up to t and that the event has not happen before t, *i.e.*,

$\lambda(t)\mathrm{d}t := \mathbb{P}\{\text{event in } [t, t + \mathrm{d}t)|\mathcal{H}(t)\} = \mathbb{E}[\mathrm{d}N(t)|\mathcal{H}(t)]$

where one typically assumes that only one event can happen in a small window of size $\mathrm{d}t$, *i.e.*, $\mathrm{d}N(t) \in \{0, 1\}$. Then, given a time $t \geqslant 0$, we can also characterize the conditional probability that no event happens during $[0, t)$ as: $S(t) = \exp\left(-\int_0^t \lambda(\tau)\,\mathrm{d}\tau\right)$ and the conditional density $p(t)$ that an event occurs at time t is defined as

$$p(t) = \lambda(t) \exp\left(-\int_0^t \lambda(\tau)\,\mathrm{d}\tau\right) \qquad (1)$$

The function form of the intensity $\lambda(t)$ is often designed to capture the phenomena of interests. We present some representative examples of typical point processes where the intensity has a particularly specified parametric forms.

For example, **Poisson processes** has a constant intensity $\lambda(t) = \mu$. **Hawkes processes** [18] models the mutual excitation between events, *i.e.*, $\lambda(t) = \mu + \alpha \sum_{t_i \in \mathcal{H}(t)} \kappa_\omega(t - t_i)$, where $\kappa_\omega(t) := \exp(-\omega t)$ is an exponential triggering kernel, $\mu \geqslant 0$ is a baseline intensity. Here, the occurrence of each historical event increases the intensity by a certain amount determined by the kernel κ_ω and the weight $\alpha \geqslant 0$, making the intensity history dependent and a stochastic process by itself. **Rayleigh process** [1] models an increased tendency over time

$$\lambda(t) = \alpha t,$$

where $\alpha > 0$ is the weight parameter.

3 DEEP COEVOLUTIONARY FEATURE EMBEDDING PROCESSES

We present Deep Coevolutionary Network (DeepCoevolve): a generative model for modeling the interaction dynamics between users and items. The high level idea of our model is that we first use RNN to explicitly capture the coevolving nature of users' and items' latent features. Then, based on the compatibility between the users' and items' latent feature, we model the user-item interactions by a multi-dimensional temporal point process. We further parametrize the intensity function by the compatibility between users' and items' latent features.

Given m users and n items, we denote the ordered list of N observed events as $O = \{e_j = (u_j, i_j, t_j, \boldsymbol{q}_j)\}_{j=1}^N$ on time window $[0, T]$, where $u_j \in \{1, \ldots, m\}$, $i_j \in \{1, \ldots, n\}$, $t_j \in \mathbb{R}^+$, $0 \leqslant t_1 \leqslant t_2 \ldots \leqslant T$. This represents the interaction between user u_j, item i_j at time t_j, with the interaction context $\boldsymbol{q}_j \in \mathbb{R}^d$. Here $\boldsymbol{q}_j$ can be a high dimension vector such as the text review, or simply the embedding of static user/item features such as user's profile and item's categorical features. For notation simplicity, we define $O^u = \{e_j^u = (i_j^u, t_j^u, \boldsymbol{q}_j^u)\}$ as the ordered listed of all events related to user u, and $O^i = \{e_j^i = (u_j^i, t_j^i, \boldsymbol{q}_j^i)\}$ as the ordered list of all events related to item i. We also set $t_0^i = t_0^u = 0$ for all the users and items.

3.1 Deep coevolutionary network

As the standard setting in recommendation systems, for each user u and item i, we use the low-dimension vector $\boldsymbol{f}_u(t) \in \mathbb{R}^k$ and $\boldsymbol{g}_i(t) \in \mathbb{R}^k$ to represent their latent feature embedding at time t, *i.e.*, user's interest and item's property.

These embedding vectors change over time. Specifically, we model the evolution of embeddings $\boldsymbol{f}_u(t)$ and $\boldsymbol{g}_i(t)$ using two update functions. These embeddings are initialized as 0, and then the updates are carried out whenever a user interacts with an item. The amount of the updates are determined by neural networks which aggregates historical information of user and item embeddings, and interaction time and context. The specific form of the two parallel feature embedding updates are described below.

DeepCoevolve:
Users' embedding update. For each user u, we formulate the corresponding embedding $\boldsymbol{f}_u(t)$ after user u's k-th event $e_k^u = (i_k^u, t_k^u, \boldsymbol{q}_k^u)$ as follows

$$\boldsymbol{f}_u(t_k^u) = \sigma\Bigg(\underbrace{\boldsymbol{W}_1(t_k^u - t_{k-1}^u)}_{\text{temporal drift}} + \underbrace{\boldsymbol{W}_2 \boldsymbol{f}_u(t_{k-1}^u)}_{\text{self evolution}} + \underbrace{\boldsymbol{W}_3 \boldsymbol{g}_{i_k}(t_k^u-)}_{\text{co-evolution: item feature}} + \underbrace{\boldsymbol{W}_4 \boldsymbol{q}_k^u}_{\text{interaction feature}}\Bigg). \quad (2)$$

Items' embedding update. For each item i, model $\boldsymbol{g}_i(t)$ after item i's k-th event $e_k^i = (u_k^i, t_k^i, \boldsymbol{q}_k^i)$ as follows:

$$\boldsymbol{g}_i(t_k^i) = \sigma\Bigg(\underbrace{\boldsymbol{V}_1(t_k^i - t_{k-1}^i)}_{\text{temporal drift}} + \underbrace{\boldsymbol{V}_2 \boldsymbol{g}_i(t_{k-1}^i)}_{\text{self evolution}} + \underbrace{\boldsymbol{V}_3 \boldsymbol{f}_{u_k}(t_k^i-)}_{\text{co-evolution: item feature}} + \underbrace{\boldsymbol{V}_4 \boldsymbol{q}_k^i}_{\text{interaction feature}}\Bigg), \quad (3)$$

where $t-$ means the time point just before time t, $\boldsymbol{W}_4, \boldsymbol{V}_4 \in \boldsymbol{R}^{k \times d}$ are the embedding matrices mapping from the explicit high-dimensional feature space into the low-rank latent feature space and $\boldsymbol{W}_1, \boldsymbol{V}_1 \in \mathbb{R}^k$, $\boldsymbol{W}_2, \boldsymbol{V}_2, \boldsymbol{W}_3, \boldsymbol{V}_3 \in \mathbb{R}^{k \times k}$ are weights parameters. $\sigma(\cdot)$ is activation function, such as commonly used Tanh or Sigmoid function. For simplicity, we use basic recurrent neural network to formulate the recurrence structure, but it is also straightforward to design more sophisticated structured using GRU or LSTM to gain more expressive power. Figure 1 summarizes the key update equations of our model given a new event. Each update incorporates four terms, namely terms related to temporal drift, self evolution, coevolution and interaction features.

The rationale for designing these terms is explained below:

- **Temporal drift**. The first term is defined based on the time difference between consecutive events of specific user or item. It allows the users' basic feature (*e.g.*, personalities) and items' basic property (*e.g.*, price, description) to smoothly change over time. Such changes of basic features normally are caused by external influences.
- **Self evolution**. The current user feature should also be influenced by its feature at the earlier time. This captures the intrinsic evolution of user/item features. For example, a user's current interest should be related to his/her interest two days ago.
- **User-item coevolution**. This term captures the phenomenon that user and item embeddings are mutually dependent on each other. First, a user's embedding is determined by the latent features of the items he interacted with. At each event time t_k, the item embedding influences the update of the user embedding. Conversely, an item's embedding is determined by the feature embedding of the user who just interacts with the item.
- **Interaction features**. The interaction feature is the additional information happened in the user-item interactions. For example, in online discussion forums such as Reddit, the interaction features are the posts and comments. In online review sites such as Yelp, the features are the reviews of the businesses. This term models influence of interaction context on user and item embeddings. For example, the contents of a user's post to a Reddit science group are really original and interesting, which sets the future direction of the group.

3.2 Understanding coevolutionary embeddings

Although the recurrent updates in (2) and (3) involve only the user and item pairs directly participating in that specific interaction, the influence of a particular user or item can propagate very far into the entire bipartite network. Figure 2a provides an illustration of such cascading effects. It can be seen that a user's feature embedding can influence the item feature embedding he directly interacts with, then modified item feature embedding can influence a different user who purchases that item in a future interaction event, and so on and so forth through the entire network.

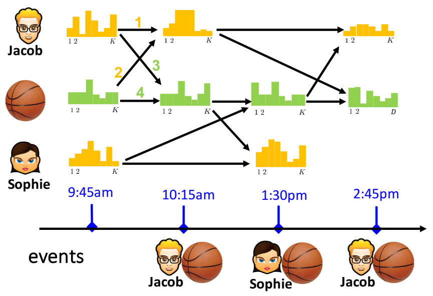
Since the feature embedding updates are event driven, both the user and item's feature embedding processes are piecewise constant functions of time. These embeddings are *changed only if an interaction event happens.* That is a user's attribute changes only when he has a new interaction with some item. This is reasonable since a user's taste for music changes only when he listens to some new or old musics. Similarly, an item's attribute changes only when some user interacts with it. Such piecewise constant feature embeddings are illustrated in Figure 2b.

Next we show how to use the feature embeddings to model the complex user-item interaction event dynamics.

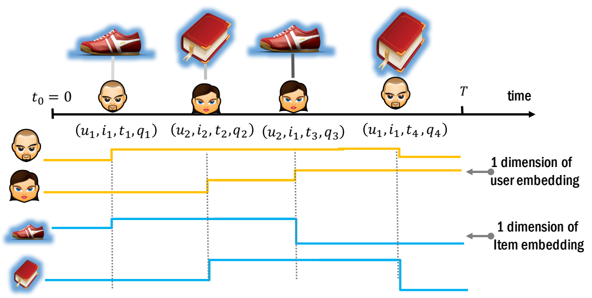
3.3 Intensity function as the compatibility between embeddings

We model the repeated occurrences of all users interaction with all items as a multi-dimensional temporal point process, with each user-item pair as one dimension. Mathematically, we model the intensity function in the (u, i)-th dimension (user u and item i) as a Rayleigh process:

$$\lambda^{u,i}(t|t') = \underbrace{\exp\left(\boldsymbol{f}_u(t')^\top \boldsymbol{g}_i(t')\right)}_{\text{user-item compatibility}} \cdot \underbrace{(t - t')}_{\text{time lapse}} \quad (4)$$



(a) Graph of embedding computation



(b) Dependency between events

Figure 2: (a) The arrows indicate the dependency structures in the embedding updates, *e.g.*, Jacob interacts with basketball at 10:15am. Then the feature embeddings are updated: the new feature embedding at 10:15 am is influenced by his previous feature embedding and the basketball's previous feature embedding at 9:45am (arrow 1 and 2); the basketball's feature embedding is also influenced by Jacob's previous feature embedding and its previous embedding feature (arrow 3 and 4). (b) A user or item's feature embedding is piecewise constant over time and will change *only* after an interaction event happens. Only one dimension of the feature embedding is shown.

where $t > t'$, and t' is the last time point where either user u's embedding or item i's embedding changes before time t. The rationale behind this formulation is as follows:

- **Time as a random variable.** Instead of discretizing the time into epochs as traditional methods [4, 15, 20, 28, 33], we explicitly model the timing of each interaction event as a random variable, which naturally captures the heterogeneity of the temporal interactions between users and items.
- **Short term preference.** The probability for user u to interact with item i depends on the compatibility of their instantaneous embeddings, which is evaluated through the inner product at the last event time t'. Because $\boldsymbol{f}_u(t)$ and $\boldsymbol{g}_i(t)$ co-evolve through time, their inner-product measures a general representation of the cumulative influence from the past interactions to the occurrence of the current event. The $\exp(\cdot)$ function ensures the intensity is positive and well defined.
- **Rayleigh time distribution.** The user and item embeddings are *piecewise constant*, and we use the time lapse term to make the intensity *piecewise linear*. This form leads to a Rayleigh distribution for the time intervals between consecutive events in each dimension. It is well-adapted to modeling fads [1], where the likelihood of generating an event rises to a peak and then drops extremely rapidly. Furthermore, it is computationally easy to obtain an analytic form of this likelihood. One can then use it to make item recommendation by finding the dimension that the likelihood function reaches the peak.

4 EFFICIENT LEARNING FOR DEEP COEVOLUTIONARY NETWORK

In this section, we will first introduce the objective function, and then propose an efficient learning algorithm.

4.1 Objective function

With the parameterized intensity function in (4), we can sample events according to it. Due to the interdependency between the feature embeddings and the propagation of influence over the interaction network, the different dimensions of the point process can intricate dependency structure. Such dependency allows sophisticated feature diffusion process to be modeled. Figure 3a illustrates the dependency structures of the generated events.

Given a sequence of events observed in real world, we can further estimate the parameters of the model by maximizing the likelihood of these observed events. Given a set of N events, the joint negative log-likelihood can be written as [11]:

$$\ell = \underbrace{-\sum_{j=1}^{N} \log\left(\lambda^{u_j, i_j}(t_j | t_j')\right)}_{\text{happened events}} + \underbrace{\sum_{u=1}^{m}\sum_{i=1}^{n}\int_0^T \lambda^{u,i}(\tau|\tau')\, d\tau}_{\text{survival of not-happened events}} \qquad (5)$$

We can interpret it as follows: (i) the negative intensity summation term ensures the probability of all interaction events is maximized; (ii) the second survival probability term penalizes the *non-presence* of an interaction between all possible user-item pairs on the observation window. Hence, our framework not only explains why an event happens, but also why an event did not happen.

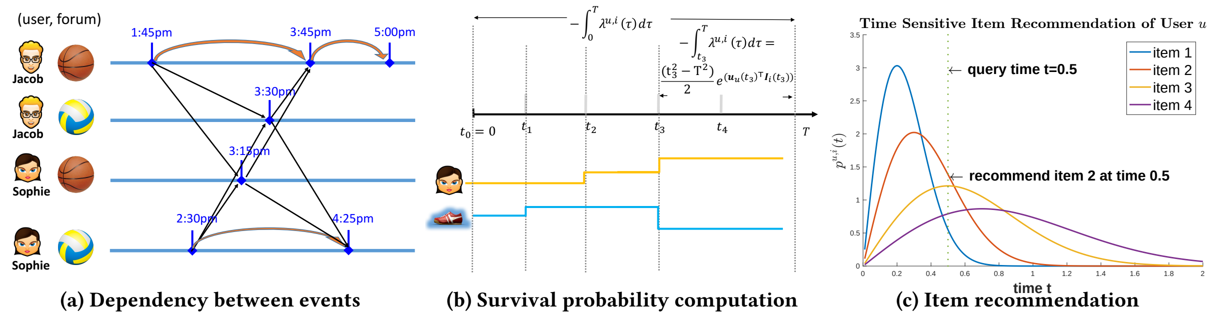
Due to the co-evolution nature of our model, it is a very challenging task to learn the model parameters since the bipartite interaction network is time-varying. Next, we will design an efficient learning algorithm for our model.

4.2 Efficient learning algorithm

We propose an efficient algorithm to learn the parameters $\{V_i\}_{i=1}^4$ and $\{\boldsymbol{W}_i\}_{i=1}^4$. The Back Propagation Through Time (BPTT) is the standard way to train a RNN. To make the back propagation tractable, one typically needs to do truncation during training. However, due to the novel co-evolutionary nature of our model, all the events are related to each other by the user-item bipartite graph (Figure 3a), which makes it hard to decompose.

Hence, in sharp contrast to works [12, 19] in sequential data where one can easily break the sequences into multiple segments to make the BPTT trackable, it is a *challenging* task to design BPTT in our case. To efficiently solve this problem, we first order all the events globally and then do mini-batch training in a sliding window fashion. Each time when conducting feed forward and back propagation, we take the consecutive events within current sliding window to build the computational graph. Thus in our case the truncation is on the global timeline, compared with the truction on individual independent sequences in prior works.

Next, we explain our procedure in detail. Given a mini-batch of M ordered events $\tilde{O} = \{e_j\}_{j=1}^M$, we set the time span to be $[T_0 =$



(a) Dependency between events **(b) Survival probability computation** **(c) Item recommendation**

Figure 3: (a) The events dependency for two users and two forums (items). It shows how event at one dimension influence other dimensions. Each orange arrow represents the dependency within each dimension, and the black arrow denotes the cross-dimension dependency. For example, Sophie interacts with volleyball at 2:30pm, and this event changes the volleyball embedding, thus will affect Jacob's visit at 3:30pm. (b) Survival probability for a user-item pair (u, i). The integral $\int_0^T \lambda^{u,i}(\tau|\tau')\mathrm{d}\tau$ is decomposed into 4 inter-event intervals separated by $\{t_0, \cdots, t_3\}$. (c) Illustration of item recommendation.

$t_1, T = t_M]$. Below we show how to compute the intensity and survival probability term in the objective function (5) respectively.

- **Computing the intensity function.** Each time when a new event e_j happens between u_j and i_j, their corresponding feature embeddings will evolve according to a computational graph, illustrated in Figure 2a. Due to the change of feature embedding, all the dimensions related to u_j or i_j are also influenced and the intensity functions for these dimensions change consequently. Such cross-dimension dependency of influence is further shown in Figure 3a. In our implementation, we first compute the corresponding intensity $\lambda^{u_j,i_j}(t_j|t_j')$ according to (4), and then update the embedding of u_j and i_j. This operation takes $O(M)$ complexity, and is independent to the number of users or items.
- **Computing the survival function.** To compute the survival probability $-\int_{T_0}^{T} \lambda^{u,i}(\tau|\tau')\mathrm{d}\tau$ for each pair (u, i), we first collect all the time stamps $\{t_k\}$ that have events related to either u or i. For notation simplicity, let $|\{t_k\}| = n_{u,i}$ and $t_1 = T_0, t_{n_{u,i}} = T$. Since the embeddings are piecewise constant, the corresponding intensity function is piecewise linear according to (4). Thus, the integration is decomposed into each time interval where the intensity is linear, *i.e.*,

$$\int_{T0}^{T} \lambda^{u,i}(\tau|\tau')\mathrm{d}\tau = \sum_{k=1}^{n_{u,i}-1} \int_{t_k}^{t_{k+1}} \lambda^{u,i}(\tau|\tau')\mathrm{d}\tau \quad (6)$$

$$= \sum_{k=1}^{n_{u,i}-1} (t_{k+1}^2 - t_k^2) \exp\left(f_u(t_k)^\top g_i(t_k)\right) \quad (7)$$

Figure 3b illustrates the details of computation.

Although the survival probability term exists in closed form, it is still expensive to compute it for each user item pair. Moreover, since the user-item interaction bipartite graph is very sparse, it is not necessary to monitor each dimension in the stochastic training setting. To speed up the computation, we propose a novel random-sampling scheme as follows.

Note that the intensity term in the objective function (5) tries to maximize the inner product between user and item that has interaction event, while the survival term penalizes over all other pairs of inner products. We observe that this is similar to Softmax computing for classification problem. Hence, inspired by the noise-contrastive estimation method (NCE) [17] that is widely used in language models [27], we keep the dimensions that have events on them, while randomly sample dimensions without events in current mini-batch to speed up the computation.

Finally, another challenge in training lies in the fact that the user-item interactions vary a lot across mini-batches, hence the corresponding computational graph also changes greatly. To make the training efficient, we use the graph embedding framework [10] which allows training deep learning models where each term in the objective has a different computational graphs but with shared parameters. The Adam Optimizer [23] and gradient clip is used in our experiment.

5 PREDICTION WITH DEEPCOEVOLVE

Since we use DeepCoevolve to model the intensities of multivariate point processes, our model can make two types of predictions, namely next item prediction and event time prediction. The precise event time prediction is especially novel and not possible by most prior work. More specifically,

- *Next item prediction.* Given a pair of user and time (u, t), we aim at answering the following question: *what is the item the user u will interact at time t?* To answer this problem, we rank all items in the descending order in term of the value of the corresponding conditional density at time t,

$$p^{u,i}(t) = \lambda^{u,i}(t)S^{u,i}(t) \quad (8)$$

and the best prediction is made to the item with the largest conditional density. Using this formulation, at different point in time, a different prediction/recommendation can be made, allowing the prediction to be time-sensitive. Figure 3c illustrates such scenario.
- *Time prediction.* Given a pair of user and item (u, i), we aim at answering the following question: *When this user will interact with this item in the future?* We predict this quantity by computing the expected next event time under $p^{u,i}(t)$. Since the intensity model is a Rayleigh model, the expected event time can be computed in closed form as

$$\mathbb{E}_{t\sim p^{u,i}(t)}[t] = \sqrt{\frac{\pi}{2\exp\left(f_u(t-)^\top g_i(t-)\right)}} \quad (9)$$

Table 1: Comparison with different methods.

Method	DeepCoevolve	LowRankHawkes	Coevolving	PoissonTensor	TimeSVD++	FIP	STIC
Continuous time	√	√	√				√
Predict Item	√	√	√	√	√	√	
Predict Time	√	√	√	√			√
Computation	RNN	Factorization	Factorization	Factorization	Factorization	Factorization	HMM

6 COMPLEXITY ANALYSIS

In this section, we provide an in depth analysis of our approach in terms of the model size, the training and testing complexity. We measure these terms as functions of the number of users, number of items, and the number of events. Other factors, such as dimensionality of latent representations, are treated as constant.

- **Model Size**. If the baseline profile feature for user and items are not available, we can use one-hot representation of user and items, and the basic feature embedding takes $O(\#\text{user} + \#\text{item})$ parameters. The interaction features (e.g., bag of words features for reviews) are independent of the number of users and number of items. Moreover, the parameters of RNN are also independent of the dataset. Thus, our model is as scalable as the traditional matrix factorization methods.
- **Training Complexity**. Using BPTT with a budget limit, each mini-batch training will only involve consecutive M samples. When an event happens, the embeddings of the corresponding user and item are updated. Thus we need $O(M)$ operations for updating embeddings. For each user item pair, $(n+m)$ dimensions that are correlated with this user-item pair will update their constant intensity functions. Hence, ideally we need $O((n+m)\times M)$ operations to forward the intensity function and survival probability. As mentioned in Section 4, using NCE to sample C dimensions that survive from last event to current event, we can further reduce the computational cost to $C \times M$. Thus in summary, each stochastic training step takes $O(M)$ cost, which is linear to the number of samples in mini-batch.
- **Test/Prediction Complexity**. The item prediction in (8) requires comparing each item with the current user embedding. Since (8) has a closed form, the complexity is $O(N)$ where N is the number of items. This can further be improved by other methods such as fast inner product search [3]. Since the event time prediction in (9) is in closed form, the complexity for this prediction task is $O(1)$.

7 EXPERIMENTS

We evaluate our method on three real-world datasets:

- **IPTV.** It contains 7,100 users' watching history of 436 TV programs in 11 months (Jan 1 - Nov 30 2012), with around 2M events, and 1,420 movie features (including 1,073 actors, 312 directors, 22 genres, 8 countries and 5 years).
- **Yelp.** This data was available in Yelp Dataset challenge Round 7. It contains reviews for various businesses from October, 2004 to December, 2015. The dataset we used here contains 1,005 users and 47,924 businesses, with totally 291,716 reviews.
- **Reddit.** We collected discussion related data on different subreddits (groups) for the month of January 2014. We removed all bot users' and their posts from this dataset. Furthermore, we randomly selected 1,000 users, 1,403 groups, and 14,816 discussions.

To study the sparsity of these datasets, we visualize the three datasets in Figure 4 from two perspectives: (i) the number of events per user, and (ii) the user-item interaction graph.

Sparsity in terms of the number of events per user. Typically, the more user history we have, the better results we should expect in the prediction tasks. In IPTV dataset, users have longer length of history than other two datasets. Thus we expect different methods will have the best performance on this dataset.

Sparsity in terms of diversity of items to recommend. If users has similar tastes, the distinct number of items in the union of their history should be small. From the user-item bipartite graph, it is easy to see that Yelp dataset has higher density than the other two datasets, hence higher diversity. The density of the interaction graph also reflects the *variety* of history event for each user. For example, the users in IPTV only have 436 programs to watch, but the users in Yelp dataset can have 47,924 businesses to choose. Also, the Yelp dataset has 9 times more items than IPTV and Reddit dataset in the bipartite graph. This means the users in Yelp dataset has more diverse tastes than users in other two datasets.

Based on the above two facts, we expect that the Yelp dataset is the most challenging one, since it has shorter length of history per user, and much more diversity of the items.

7.1 Competitors

We compared our DeepCoevolve with the following state-of-arts. Table 1 summarizes the differences between methods.

- **LowRankHawkes** [13]: This is a low rank Hawkes process model which assumes user-item interactions to be independent of each other and does not capture the co-evolution of user and item features.
- **Coevolving** [35]: This is a multi-dimensional point process model which uses a simple linear embedding to model the co-evolution of user and item features.
- **PoissonTensor** [7]: Poisson Tensor Factorization has been shown to perform better than factorization methods based on squared loss [22, 34, 37] on recommendation tasks. The performance for this baseline is reported using the average of the parameters fitted over all time intervals.
- **TimeSVD++** [25] and **FIP** [38]: These two methods are only designed for explicit ratings, the implicit user feedbacks (in the form of a series of interaction events) are converted into the explicit ratings by the respective frequency of interactions with users.
- **STIC** [21]: it fits a semi-hidden markov model (HMM) to each observed user-item pair and is only designed for time prediction.

7.2 Overall Performance Comparison

For each sequence of user activities, we use all the events up to time $T \cdot p$ as the training data, and the rest events as the testing data, where T is the observation window. We tune the latent rank

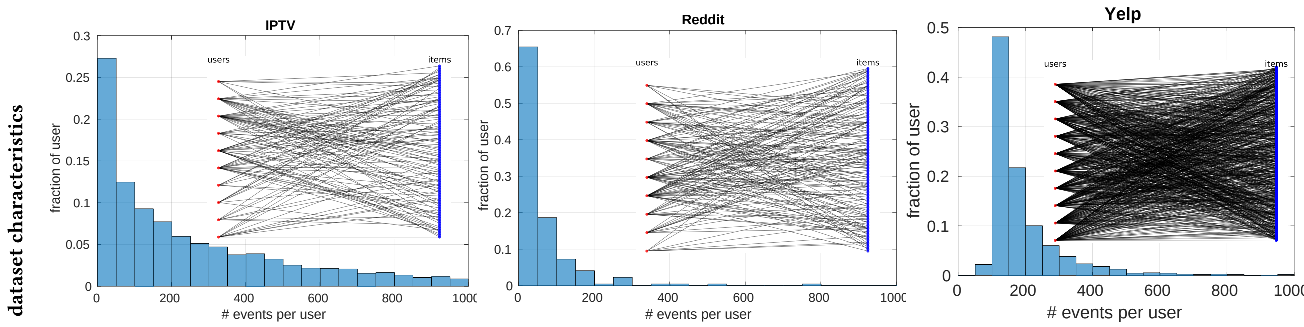
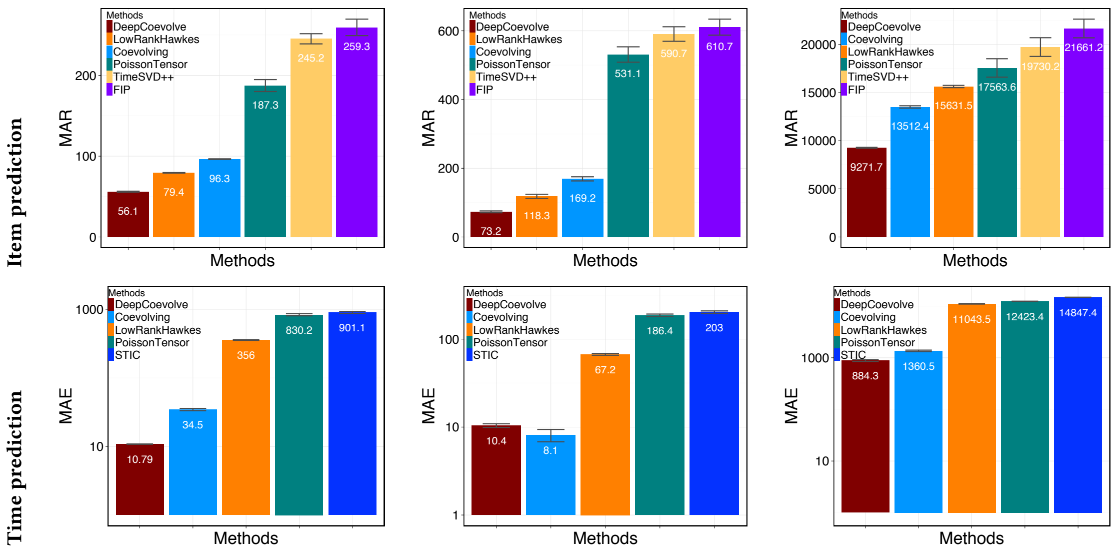
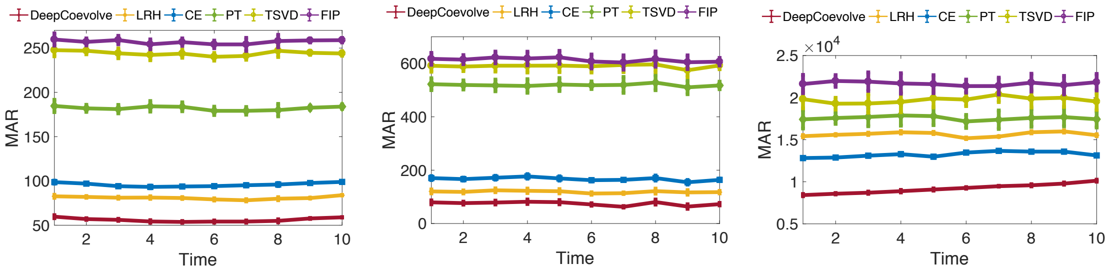


Figure 4: Visualization of the sparsity property in each dataset. The bar plot shows the distribution of number of events per user. Within each plot there is a user-item interaction graph. This graph is generated as follows. For each dataset, we randomly pick 10 users with 100 history events each user and collect all items they have interacted with. The interaction graph itself is a bipartite graph, and we put users on left side, and items on the right side.



(a) IPTV (b) Reddit (c) Yelp

Figure 5: Prediction results on three real world datasets. The MAE for time prediction is measured in hours.



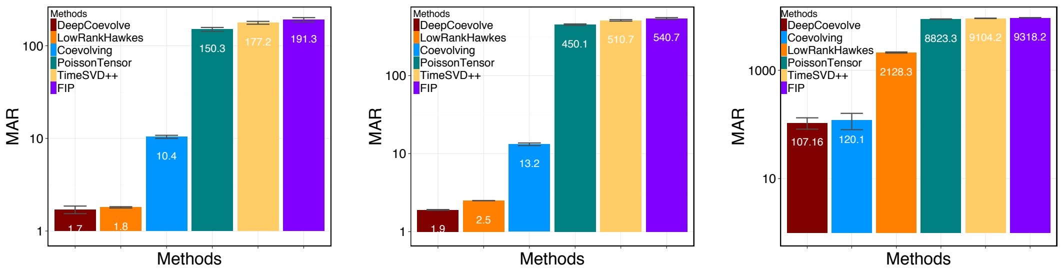
(a) IPTV (b) Reddit (c) Yelp

Figure 6: Item prediction over different time windows. We divide the entire test set into 10 bins, and report MAR in each bin.

of other baselines using 5-fold cross validation with grid search. We vary the proportion $p \in \{0.7, 0.72, 0.74, 0.76, 0.78\}$ and report the averaged results over five runs on two tasks (we will release code and data once published):

- *Item prediction metric.* We report the Mean Average Rank (MAR) of each test item at the test time. Ideally, the item associated with the test time t should rank one, hence smaller value indicates better predictive performance.
- *Time prediction metric.* We report the Mean Absolute Error (MAE) between the predicted and true time.

Figure 5 shows that DeepCoevolve significantly outperforms both epoch-based baselines and state-of-arts point process based methods. LowRankHawkes has good performance on item prediction but not on time prediction, while Coevolving has good performance on time prediction but not on item prediction. We discuss the performance regarding these two metrics below.



(a) IPTV (b) Reddit (c) Yelp

Figure 7: Item prediction performance on recurring events.

Item prediction. Top row of Figure 5 shows the overall item recommendation performance across 5 different splits of datasets. Though the characteristics of datasets varies significantly, our proposed DeepCoevolve is doing consistently better than competitors. Note LowRankHawkes also achieves good item prediction performance, but not as good on the time prediction task. Since one only needs the rank of conditional density $p(\cdot)$ in (1) to conduct item prediction, LowRankHawkes may still be good at differentiating the conditional density function, but could not learn its actual value accurately, as shown in the time prediction task where the value of the conditional density function is needed for precise prediction.

Time prediction. Figure 5 also shows that DeepCoevolve outperforms other methods. Compared with LowRankHawkes, our method has 6× improvement on Reddit, it has 10× improvement on Yelp, and 30× improvement on IPTV. The time unit is hour. Hence it has 2 weeks accuracy improvement on IPTV and 2 days on Reddit. This is important for online merchants to make time sensitive recommendations. An intuitive explanation is that our method accurately captures the *nonlinear* pattern between user and item interactions. The competitor LowRankHawkes assumes specific parametric forms of the user-item interaction process, hence may not be accurate or expressive enough to capture real world temporal patterns. Furthermore, it models each user-item interaction dimension independently, which may lose the important affection from user's interaction with other items while predicting the current item's reoccurrence time. Our work also outperforms Coevolving, *e.g.*, with around 3× MAE improve on IPTV. Moreover, the item prediction performance is also much better than Coevolving. It shows the importance of using RNN to capture the nonlinear embedding of user and item latent features, instead of the simple parametrized linear embedding in Coevolving.

7.3 Refined Performance Comparison

Comparison in different time windows. Figure 6 further shows the item prediction performance on different time windows. Specifically, we divide the timeline of test set into 10 consecutive nonintersecting windows, and report MAR on each of them. Our DeepCoevolve is consistently better in different time periods. Moreover, all the point process based methods have stable performance with small variance. Since these models use new events to update the intensity function, the likelihood of unseen user-item interactions in the training set will have chance to get adjusted accordingly.

Performance on recurring events. For the businesses like restaurants, it is also important to understand whether/when your customers will come back again. Here we compare the performance on those recurring events that appear both in training and testing. As expected in Figure 7, all the algorithms get better performance on this portion of events, compared to the overall results in Figure 5. Moreover, the point process models benefit more from predicting recurring events. This justifies the fact that the more events we observe for a particular dimension (user-item pair), the better we can estimate its intensity and likelihood of future events.

8 RELATED WORK

Recent work predominantly fix the latent features assigned to each user and item [2, 6, 14, 26, 29, 36, 38, 39]. In more sophisticated methods, the time is divided into epochs, and static latent feature models are applied to each epoch to capture some temporal aspects of the data [4, 7, 15, 16, 20, 22, 22, 25, 28, 33, 37, 37]. For these methods, it is not clear how to choose the epoch length since different users may have very different timescale when they interact with items. Moreover, since the predictions are only in the resolution of the chosen epoch length, these methods typically are not good at time-sensitive question such as when a user will return to the item. A recent low-rank point process model [13] overcomes these limitations. However, it fails to capture the heterogeneous coevolutionary properties of user-item interactions.

Wang et al. [35] models the coevolutionary property, but uses a simple linear representation of the users' and items' latent features, which might not be expressive enough to capture the real world patterns. Our model is fundamentally different from [35]. We use RNN to model the complex dynamics of the feature embedding, which is more powerful due to the *nonparametric* nature of our work and much improved prediction performance. More importantly, the model size is only *linear* in the number of users and items, making our algorithm more scalable. Figure 8 contains more details.

As demonstrated in Du et al. [12], the nonlinear RNN unit is quite flexible to approximate many point process models. However, RMTPP is limited to learn only in one-dimensional point process setting. Our model is significantly different from RMTPP since we focus on the recommendation system setting with the idea of using multivariate point process and RNN to capture coevolutionary dynamics of latent features over a temporally evolving network. We further demonstrate that our model is very efficient even with the presence of RNN related parameters and can therefore be potentially applied to online setting.

In the deep learning community, very few work model the coevolution of users' and items' latent features and are still extensions of epoch based methods. [32] proposed a hierarchical Bayesian model that jointly performs learning for the content features and

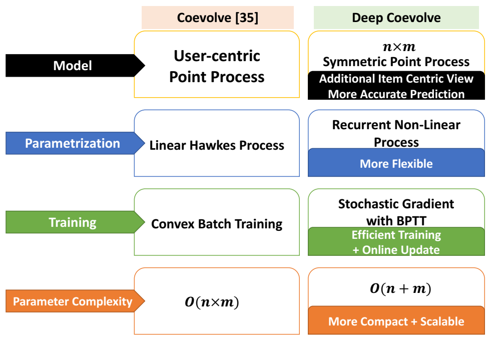


Figure 8: Comparison with the linear Hawkes model.

collaborative filtering for the ratings matrix. [19] applied RNN and adopt item-to-item recommendation approach with session based data. [31] improved this model with techniques like data augmentation, temporal change adaptation. [24] proposed collaborative RNN that extends collaborative filtering method to capture history of user behavior. Specifically, they used static global latent factors for items and assign separate latent factors for users that are dependent on their past history. [30] extended the deep semantic structured model to capture multi-granularity temporal preference of users. They use separate RNN for each temporal granularity and combine them with feed forward network which models users' and items' long term static features. [5] models the time change with piecewise constant function, but is not capable of predicting the future time point. Our work is unique in the sense that we explicitly treat time stamps as random variables and model the coevolution of users' and items' latent features using temporal point processes and deep learning model over evolving graph.

9 CONCLUSIONS

We proposed an expressive and efficient framework to model the *nonlinear* coevolution nature of users' and items' embeddings. Moreover, the user and item's evolving and coevolving processes are captured by the RNN. We further developed an efficient stochastic training algorithm for the coevolving user-item netowrks. We demonstrate the superior performance of our method on both the time and item prediction task, which is not possible by most prior work. Future work includes extending to other social applications, such as group dynamics in message services.

REFERENCES

[1] Odd Aalen, Ornulf Borgan, and Hakon Gjessing. 2008. *Survival and event history analysis: a process point of view.* Springer.
[2] D. Agarwal and B.-C. Chen. 2009. Regression-based latent factor models. In *KDD*, J.F. Elder, F. Fogelman-Soulié, P.A. Flach, and M.J. Zaki (Eds.).
[3] Grey Ballard, Tamara G Kolda, Ali Pinar, and C Seshadhri. 2015. Diamond sampling for approximate maximum all-pairs dot-product (MAD) search. In *Data Mining (ICDM), 2015 IEEE International Conference on.* IEEE, 11–20.
[4] Laurent Charlin, Rajesh Ranganath, James McInerney, and David M Blei. 2015. Dynamic Poisson Factorization. In *RecSys.*
[5] Tianqi Chen, Hang Li, Qiang Yang, and Yong Yu. 2013. General Functional Matrix Factorization using Gradient Boosting. In *Proceeding of 30th International Conference on Machine Learning (ICML'13)*, Vol. 1. 436–444.
[6] Y. Chen, D. Pavlov, and J.F. Canny. 2009. Large-scale behavioral targeting. In *KDD*, J.F. Elder, F. Fogelman-Soulié, P.A. Flach, and M. J. Zaki (Eds.).
[7] Eric C Chi and Tamara G Kolda. 2012. On tensors, sparsity, and nonnegative factorizations. *SIAM J. Matrix Anal. Appl.* 33, 4 (2012), 1272–1299.
[8] D.R. Cox and V. Isham. 1980. *Point processes.* Vol. 12. Chapman & Hall/CRC.
[9] D.R. Cox and P.A.W. Lewis. 2006. Multivariate point processes. *Selected Statistical Papers of Sir David Cox: Volume 1, Design of Investigations, Statistical Methods and Applications* 1 (2006), 159.
[10] Hanjun Dai, Bo Dai, and Le Song. 2016. Discriminative Embeddings of Latent Variable Models for Structured Data. In *ICML.*
[11] D.J. Daley and D. Vere-Jones. 2007. *An introduction to the theory of point processes: volume II: general theory and structure.* Vol. 2. Springer.
[12] Nan Du, Hanjun Dai, Rakshit Trivedi, Utkarsh Upadhyay, Manuel Gomez-Rodriguez, and Le Song. 2016. Recurrent Marked Temporal Point Processes: Embedding Event History to Vector. In *KDD.*
[13] Nan Du, Yichen Wang, Niao He, and Le Song. 2015. Time Sensitive Recommendation From Recurrent User Activities. In *NIPS.*
[14] Michael D Ekstrand, John T Riedl, and Joseph A Konstan. 2011. Collaborative filtering recommender systems. *Foundations and Trends in Human-Computer Interaction* 4, 2 (2011), 81–173.
[15] Prem Gopalan, Jake M Hofman, and David M Blei. 2015. Scalable Recommendation with Hierarchical Poisson Factorization. *UAI* (2015).
[16] San Gultekin and John Paisley. 2014. A collaborative kalman filter for time-evolving dyadic processes. In *ICDM.* 140–149.
[17] Michael U Gutmann and Aapo Hyvärinen. 2012. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research* 13, Feb (2012), 307–361.
[18] Alan G Hawkes. 1971. Spectra of some self-exciting and mutually exciting point processes. *Biometrika* 58, 1 (1971), 83–90.
[19] Balazs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based Recommendations With Recurrent Neural Networks. In *ICLR.*
[20] Balázs Hidasi and Domonkos Tikk. 2015. General factorization framework for context-aware recommendations. *Data Mining and Knowledge Discovery* (2015), 1–30.
[21] Komal Kapoor, Karthik Subbian, Jaideep Srivastava, and Paul Schrater. 2015. Just in Time Recommendations: Modeling the Dynamics of Boredom in Activity Streams. In *WSDM.*
[22] Alexandros Karatzoglou, Xavier Amatriain, Linas Baltrunas, and Nuria Oliver. 2010. Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *Recsys.*
[23] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[24] Young-Jun Ko, Lucas Maystre, and Matthias Grossglauser. 2016. Collaborative Recurrent Neural Networks for Dynamic Recommender Systems. *Journal of Machine Learning Research* (2016), 1–16.
[25] Y. Koren. 2009. Collaborative filtering with temporal dynamics. In *KDD.*
[26] Yehuda Koren and Joe Sill. 2011. OrdRec: an ordinal model for predicting personalized item rating distributions. In *RecSys.*
[27] Andriy Mnih and Koray Kavukcuoglu. 2013. Learning word embeddings efficiently with noise-contrastive estimation. In *Advances in Neural Information Processing Systems.* 2265–2273.
[28] Jiayu Zhou Juhan Lee Preeti Bhargava, Thomas Phan. 2015. Who, What, When, and Where: Multi-Dimensional Collaborative Recommendations Using Tensor Factorization on Sparse User-Generated Data. In *WWW.*
[29] R. Salakhutdinov and A. Mnih. 2008. Bayesian probabilistic matrix factorization using Markov chain Monte Carlo. In *ICML.*
[30] Yang Song, Ali Mamdouh Elkahky, and Xiaodong He. 2016. Multi-Rate Deep Learning for Temporal Recommendation. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval.* 909–912.
[31] Yong K Tan, Xinxing Xu, and Yong Liu. 2016. Improved Recurrent Neural Networks for Session-based Recommendations. *arXiv:1606.08117v2* (2016).
[32] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. 2015. Collaborative Deep Learning for Recommender Systems. In *KDD.* ACM.
[33] Xin Wang, Roger Donaldson, Christopher Nell, Peter Gorniak, Martin Ester, and Jiajun Bu. 2016. Recommending Groups to Users Using User-Group Engagement and Time-Dependent Matrix Factorization. In *AAAI.*
[34] Yichen Wang, Robert Chen, Joydeep Ghosh, Joshua C Denny, Abel Kho, You Chen, Bradley A Malin, and Jimeng Sun. 2015. Rubik: Knowledge Guided Tensor Factorization and Completion for Health Data Analytics. In *KDD.*
[35] Yichen Wang, Nan Du, Rakshit Trivedi, and Le Song. 2016. Coevolutionary Latent Feature Processes for Continuous-Time User-Item Interactions. In *NIPS.*
[36] Yichen Wang and Aditya Pal. 2015. Detecting Emotions in Social Media: A Constrained Optimization Approach. In *IJCAI.*
[37] Liang Xiong, Xi Chen, Tzu-Kuo Huang, Jeff G. Schneider, and Jaime G. Carbonell. 2010. Temporal Collaborative Filtering with Bayesian Probabilistic Tensor Factorization.. In *SDM.*
[38] Shuang-Hong Yang, Bo Long, Alex Smola, Narayanan Sadagopan, Zhaohui Zheng, and Hongyuan Zha. 2011. Like like alike: joint friendship and interest propagation in social networks. In *WWW.*
[39] Xing Yi, Liangjie Hong, Erheng Zhong, Nanthan Nan Liu, and Suju Rajan. 2014. Beyond Clicks: Dwell Time for Personalization. In *RecSys.*