DeepBehave: Learning Deep Representations for Social Behavioral Data

Jiezhong Qiu
Tsinghua University
qiujz16@mails.tsinghua.edu.cn

Xinyang Zhang
Tsinghua University
zhangxinyang14@mails.tsinghua.edu.cn

Jie Tang
Tsinghua University
jietang@tsinghua.edu.cn

Tracy Xiao Liu
Tsinghua University
liuxiao@sem.tsinghua.edu.cn

Shaoxiong Wang
Tsinghua University
wsxiong13@gmail.com

ABSTRACT

In online social networks, users generate a huge volume of behavioral data, which offers us an unprecedented opportunity to understand users' dynamic behavioral patterns. Meanwhile, this also brings a number of new challenges. First, users' behavior is quite dynamic and interdependent: one's behavior may be strongly influenced by family members or friends. Second, the behavioral data is huge, containing multiple different actions performed by users at different time. In addition, users behavior data may contain different latent structures.

In this paper, we formalize the problem as learning deep representations for social behaviors. We propose the DeepBehave to address these challenges by considering both users' behavior history and the structure of users' ego networks. To handle the huge volume of data, DeepBehave uses a context sampling strategy, and for extracting the latent structures, we present a heterogeneous embedding learning.

We apply DeepBehave to learn behavior representations on two large datasets — Weibo and AMiner, to support user behavior prediction. Experiments show that the proposed approach significantly improve the prediction performance compared to several alternative methods (+2.3% by F1-score; $p < 0.01$, t-test).

CCS CONCEPTS

•**Theory of computation** →**Social networks;** •**Information systems** →*Data mining;* •**Applied computing** →*Sociology;*

KEYWORDS

social behavior, social networks, representation learning

ACM Reference format:
Jiezhong Qiu, Xinyang Zhang, Jie Tang, Tracy Xiao Liu, and Shaoxiong Wang. 2018. DeepBehave: Learning Deep Representations for Social Behavioral Data . In *Proceedings of , ACM Location, February (ACM Conference),* 9 pages.
DOI: 10.475/123_4

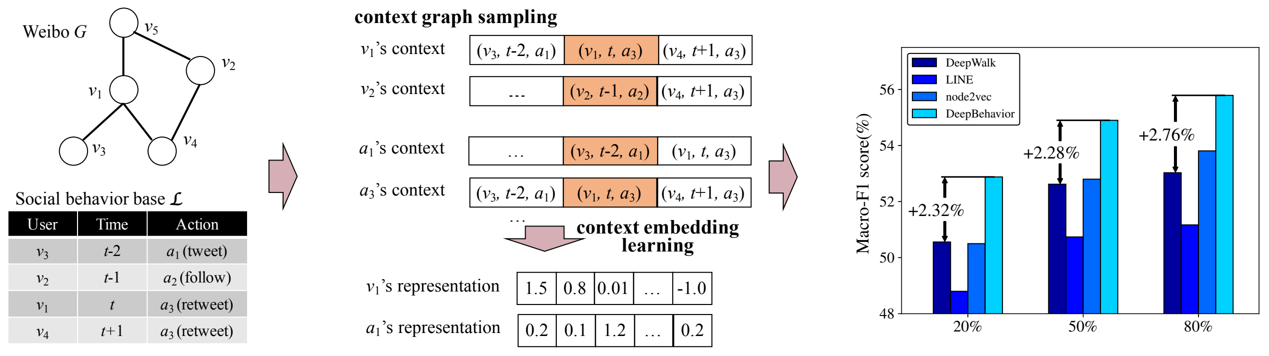
1 INTRODUCTION

Online social networks have become a bridge connecting our physical daily life and the virtual Web space. A huge volume of user behavioral data has been generated by users in online social networks. For example, Alibaba, the largest online shopping website, records 1 trillion user behavioral logs per month. Twitter, the most popular microblogging service, with 646 million active users, has generated more than 1.8 billion tweets. The large quantity of available social data offers an unprecedented opportunity to understand user behavioral patterns and network dynamics.

A user's action in the behavioral data can be defined as a triple (v, t, a), representing that user v performs an action a at time t. One interesting yet challenging question is to uncover how and why users behave as the data exhibits. Considerable research has studied different perspectives of the problem, such as user modeling [7, 8, 13], social tagging [10, 16], and social recommendation [14, 19, 27]. User modeling aims to understand users' specific needs so that system built upon the user modeling results might "say the 'right' thing at the 'right' time in the 'right' way" [8]. Social tagging is concerned with assigning a set of tags to users to represent user preferences [10]. Social recommendation helps users find relevant information by suggesting information of potential interest to them [27].

Considering all of these, we need to solve the most fundamental issue, which is how to find a good representation of users in a social network. Statistical models, such as matrix factorization [14, 25], probabilistic topic model [3, 12], and random walk [18], are widely used models for learning user representations. However, social behavioral data is becoming more and more complicated, e.g., social users are highly connected and the generated data is time-dependent. Recently, a few efforts have been made to deal with these problems. For instance, Perozzi et al. [24] presented a deep learning approach to learn latent representations of users in a network, although their method does not consider time information.

To further illustrate our work, we present Figure 1(a), which shows an example of learning representation for social behavioral


Weibo G
v_5
v_2
v_1
v_3
v_4
Social behavior base $\mathcal{L}$
context graph sampling
v_1's context
$(v_3, t\text{-}2, a_1)$ (v_1, t, a_3) $(v_4, t\text{+}1, a_3)$
v_2's context
… $(v_2, t\text{-}1, a_2)$ $(v_4, t\text{+}1, a_3)$
a_1's context
… $(v_3, t\text{-}2, a_1)$ (v_1, t, a_3)
a_3's context
$(v_3, t\text{-}2, a_1)$ (v_1, t, a_3) $(v_4, t\text{+}1, a_3)$
context embedding learning
v_1's representation
1.5 0.8 0.01 … -1.0
a_1's representation
0.2 0.1 1.2 … 0.2
DeepWalk
LINE
node2vec
DeepBehavior
Macro-F1 score(%)
+2.32%
+2.28%
+2.76%
20%
50%
80%

User	Time	Action
v_3	t-2	a_1 (tweet)
v_2	t-1	a_2 (follow)
v_1	t	a_3 (retweet)
v_4	t+1	a_3 (retweet)

(a) Input (b) Our DeepBehave model (c) Prediction performance improvement

Figure 1: Example of learning behavior representations on Weibo. (a) An example of the input in our problem: a social network and a social behavior base. (b) An example of how DeepBehave learns the representations of users and actions — context graph sampling and context embedding learning. (c) Prediction performance of publication behavior prediction by comparison methods with different percentages of training data.

data. The input of our problem includes a network of users and a historical sequence of their social actions. Then the goal is to leverage network structure and time-dependent action logs to learn a continuous representation vector for each user v and each action a. The learned representations can be used to understand the probability that a user v would perform action a at time t, and to understand which factors trigger her to perform the action.

This problem is not trivial in the following dimensions. First, since the data is time-dependent and socially connected, how to define the context for each (v, a) is thorny. Second, the social behavior data is huge, containing multiple different actions performed by different users at different time. It is important to develop an efficient approach to represent different entities in a shared space so that they can be computationally integrated. Third, how to capture the latent structures between users and actions from the big social behavior data is another challenge.

Therefore, we formalize the problem of *learning behavior representations*, and propose a novel approach called DeepBehave to consider both network structure and time information. Given the input of the problem, we first build a behavioral context graph, with each vertex representing a behavior (v, t, a)—user v performs an action a at time t, and each edge represents a potential influence relationship between the two connected behaviors. The influence relationship implies that a precedent action has a probability to induce the following action. Figure 1(b) shows an example of the heterogeneous context graph constructed in our approach. As the graph would be rather complicated if we consider all possible relationships between behaviors, we present a sampling-based method to sample context for each user and action. After obtaining the sampled behavioral context, we use an embedding learning method to learn representations of users and actions.

The learned behavioral representations can be applied to various applications. In our experiments, we use two different genres of datasets: AMiner and Weibo. AMiner.org is a free online academic search service [30]. Weibo.com is the most popular Chinese microblogging service. On each dataset, we use the learned representations to help predict user behavior. Experiments show that DeepBehave can significantly improve the prediction accuracy. We also compare DeepBehave with several embedding learning methods for networked data including DeepWalk [24], LINE [29] and node2vec [9]. Figure 1(c) shows the prediction performances for each method on the AMiner dataset with different percentages of training data, and our proposed algorithm clearly outperforms (p-value$<$ 0.01, t-test) the alternative methods (Cf. § 4 for more experimental results).

2 PROBLEM DEFINITION

Let $G = (V, E)$ denote a social network, where $V = \{v_1, \cdots, v_N\}$ is a set of N users and $E \subset V \times V$ is a set of relationships between users. Each relationship $e_{ij} \in E$ between users v_i and v_j is associated with a weight $w_{ij} \in \mathbb{R}$ indicating the strength of the relationship. The social network can be directed or undirected. If G is undirected, we have $w_{uv} \equiv w_{vu}$. Each user may perform a number of different actions in the social network. When user v performs an action a at time t, we record a behavior log, i.e., a triple (v, t, a), where $a \in A$ and $t \in [0, T]$. This yields the following definition.

Definition 2.1. **Social Behavior Base:** A social behavior base records a history of all users' actions: a sequence of behavior logs $\mathcal{L} = \{L_1, L_2, \ldots\}$, where $L_i = (v_i, a_i, t_i)$ indicates the i^{th} behavior log recorded in the base $\mathcal{L}$.

An action can be defined in different ways. For example, on Twitter, we can define the action as tweeting or retweeting a message on a specific topic (containing a hashtag like "NBAvine"). On Ebay, the action can be defined as buying a specific item. In an academic network, an action can be defined as publishing a paper on specific topics such as "data mining" and "machine learning". Our general goal is to model and understand how and why users behave as the data exhibits. In particular, we formalize the problem of learning behavior representation below.

PROBLEM 1. **Learning Behavior Representations.** *Given a social network $G = (V, E)$ and a social behavior base $\mathcal{L} = \{L_1, L_2, \ldots\}$, the goal of learning behavior representations is to learn distributed latent representations $\mathbf{X}_V \in \mathbb{R}^{|V| \times d}$ for social users and latent representations $\mathbf{X}_A \in \mathbb{R}^{|A| \times d}$ for social actions, where d is the number of latent dimensions.*

It is worthwhile to note that the i^{th} row of the matrix $\mathbf{X}_V$ corresponds to the latent representation of user v_i, and i^{th} row of matrix $\mathbf{X}_A$ corresponds to the latent representation of action a_i. Essentially, we hope to find a way to explain why a user performs a specific action at a particular time. The situation is very complicated, because a user's action may be triggered by some special event, influenced by friends, or it is simply a continuation of a previous action. Another issue worth discussing is the space of actions. It could be either defined as a specific action, or as a category of actions. The former results in a large and sparse space of actions, while the latter faces the risk of accuracy loss.

Table 1: Definition of weights of different types of edges from two behavior (v_i, a_i, t_i) and (v_j, a_j, t_j), and their sampling time complexity in context graph.

	social-based	**action-based**
Strength	$w_{v_i v_j} e^{-\lambda_1 \lvert t_i - t_j \rvert}$	$1(a_i, a_j) e^{-\lambda_2 \lvert t_i - t_j \rvert}$
Complexity	$O(\lvert N(v_i) \rvert \log \lvert \mathcal{L} \rvert)$	$O(\log \lvert \mathcal{L} \rvert)$

* Hyper parameters λ_1 and λ_2 are positive rates called the exponential time decay constants.

3 DEEPBEHAVE: LEARNING BEHAVIOR REPRESENTATIONS

We propose a context-embedding approach named DeepBehave for learning social behavioral representations by considering both network structure and time information. Specifically, our input consists of a social network and a social behavior base, and our goal is to learn embedding representations for users and actions. The technical challenges include (1) how to extract context for each entities (users and actions) from the massive behavioral logs, and (2) how to learn the representations from the extracted contexts. We first derive a heterogeneous context graph from the input data. As there are many links in a context graph for those users with large number of friends, the graph is very unbalanced. To solve this problem, we present a context sampling algorithm to select a graph context for each user and action. The sampling algorithm considers both network structure and time information. Then we present a context embedding learning algorithm to the sampled contexts to learn representations for social behaviors.

3.1 Context Graph Building

We first discuss how to build a heterogeneous context graph from an input social network and a social behavior base. The heterogeneous context graph is defined as follows:

Definition 3.1. **Heterogeneous Context Graph:** Let $\mathcal{G} = (\mathcal{L}, \mathcal{E})$ denote a heterogeneous context graph, where $\mathcal{L}$ is the set of behavior logs and $\mathcal{E}$ is the set of edges between the behavior logs. Each edge in $\mathcal{E}$ is an ordered pair (L_i, L_j), connecting two behaviors $L_i = (v_i, t_i, a_i)$ and $L_j = (v_j, t_j, a_j)$. We associate each edge with a weight $\mathcal{W}_{ij} > 0$ to represent the closeness (or strength) of two behaviors.

In our definition, there are two types of edges in $\mathcal{G}$. Each type of edge is associated with a weight (strength), which captures time dynamics, social tie, and action relatedness. Now we introduce these edges in greater detail. Table 1 summarizes the two types of edges and their corresponding strengths we defined in this work.

- **Social-based Edge.** The social-based edge is designed to model social influence and temporal behavior in social networks. For two behaviors (v_i, t_i, a_i) and (v_j, t_j, a_j), we create a social-based edge between them if the two social users are related in social networks, i.e., $(v_i, v_j) \in E$. Since we have added self-edges (v_i, v_i) in edge set E, the social-based edge could also link behaviors performed by the same user. In addition, we adopt exponential time decay to model influences of behaviors over time.
- **Action-based Edge.** If two behaviors share the same action, we define an action-based edge between them. This actually corresponds to the situation of action diffusion, i.e., one user's behavior triggers the other user to perform the same action.

In different applications, one can define various edges based on domain knowledge. For example, on Twitter, when a user tweets a message, all followers' responses to the message can be considered as the context of the user's tweet behavior [1]. In a citation network, when an author publishes a paper, all references to this paper can be used to build a context graph [26].

Once we have built a heterogeneous context graph, we can apply an embedding learning algorithm to learn the representations for social users and social actions. However, the challenge is that the context graph is huge (with billions of nodes) and very unbalanced—e.g., some users in Twitter are connected with many followers, while some others have very few. Moreover, this method cannot consider long-distance contexts, e.g., friend's friends' behavior. Therefore, we introduce context graph sampling in the next section to solve these problems.

3.2 Context Graph Sampling

The obtained heterogeneous context graph is often massive. For example, the number of nodes in the context graph generated from the Weibo data is up to 1 billion by considering users, actions, and different time stamps (Cf. § 4 for more data statistics). This makes it infeasible to apply embedding approaches. We propose a graph sampling method to select a subset of vertices and/or edges as the context for each node. In general, there are two categories of graph sampling methods: node sampling and edge sampling. The former independently and uniformly chooses nodes from the original graph, while the latter randomly selects edges instead of nodes. We use a random walk (RW) based method in this paper.

Random walk on the context graph $\mathcal{G}$ can be considered as a stochastic process rooted in L_i with random variables: $L_i^{(0)}, L_i^{(1)}, \ldots, L_i^{(k)}$. In the k^{th} iteration, a random variable is selected from the neighbors of chosen vertex $L_i^{(k-1)}$ in the $(k-1)^{th}$ iteration, with $L_i^{(0)} = L_i$. Finally, we obtain a sequence of behaviors visited by the random walking process: $L_i^{(0)}, L_i^{(1)}, \ldots, L_i^{(k)}$. We can also consider time information and social relationships in the random walking process. In our problem, we consider the random

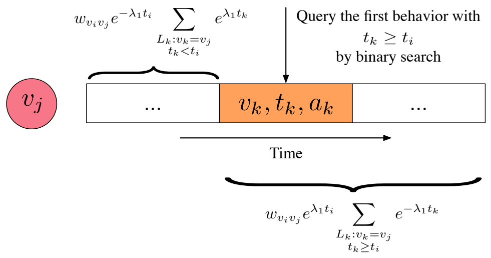

$w_{v_i v_j} e^{-\lambda_1 t_i} \sum_{\substack{L_k:v_k=v_j \\ t_k<t_i}} e^{\lambda_1 t_k}$
Query the first behavior with $t_k \geq t_i$ by binary search
v_j
...
v_k, t_k, a_k
...
Time
$w_{v_i v_j} e^{\lambda_1 t_i} \sum_{\substack{L_k:v_k=v_j \\ t_k\geq t_i}} e^{-\lambda_1 t_k}$

Figure 2: Implementation details for sampling social users/actions by Eq. 5. All the behaviors performed by v_j are preprocessed by sorting in chronological order and computing the prefix sum of $e^{\lambda_1 t_k}$ and the suffix sum of $e^{-\lambda_1 t_k}$.

walk on a context graph. Thus, for user of the k-th behavior $L_i^{(k)}$ sampled by one random walk, we define its context as:

$$\mathrm{C}\left(v_i^{(k)}\right) = \left\{L_i^{(k-C)}, \dots, L_i^{(k-1)}, a_i^{(k)}, L_i^{(k+1)}, \dots, L_i^{(k+C)}\right\},$$

where C is the parameter to control the length of the context. Similarly, for the social action of the k-th behavior $L_i^{(k)}$ sampled by one random walk, we define its context as

$$\mathrm{C}\left(a_i^{(k)}\right) = \left\{L_i^{(k-C)}, \dots, L_i^{(k-1)}, v_i^{(k)}, L_i^{(k+1)}, \dots, L_i^{(k+C)}\right\}$$

We now detail how we perform the context sampling over the heterogeneous context graph. In § 3.1, we discussed two types of edges in context graphs. Since different types of edges are unbalanced in practice, we introduce a weight parameter θ to control the effect of different types of edges ($\theta \geq 0$). Moreover, $N(v_i)$ is the set of one-hop neighbors of social user v_i in a social network. $N(a_i)$ is the set of one-hop neighbors of social action a_i in an action knowledge base. $N_0(L_i)$ and $N_1(L_i)$ are the set of one-hop neighbors of social behavior L_i under the social-based and action-based edges in a context graph, respectively. Context graph sampling is performed as follows:

Sampling the Type of Edges m. In a context graph, we have two kinds of edges. We first sample the type of edge and then sample the edge within each edge type. Here a natural choice is to treat θ as the parameters of the Bernoulli distribution over the m-type edges. Thus, we sample an edge type m: $m \sim \text{Bernoulli}(\theta)$. Next, we define the probability of sampling a social behavior L_j from $N_m(L_i)$ as:

$$P(L_j|L_i, m) = \frac{\mathcal{W}_{ij}^m}{\sum_{L_k \in N_m(L_i)} \mathcal{W}_{ik}^m} \propto \mathcal{W}_{ij}^m \tag{1}$$

Eq. 1 can be rewritten as:

$$P(L_j|L_i, m) = P(v_j|L_i, m) P(t_j, a_j|L_i, v_j, m) \tag{2}$$

or

$$P(L_j|L_i, m) = P(a_j|L_i, m) P(v_j, t_j|L_i, a_j, m) \tag{3}$$

Eq. 2 and 3 correspond to similar but slightly different sampling processes. Eq. 2 implies that we first sample a social user from V, and then sample a social action from A according to the defined probabilities. Eq. 3 corresponds to a different situation: we first sample an action from A, and then sample the specific user from those who have performed the action.

Sampling Social User/Action. If we sample $m = 0$ (social-based edge) in the previous step, it requires that we sample the next social behavior L_j which satisfies $(v_i, v_j) \in E$. Furthermore, we sample the entity of the behavior v_j (Eq. 2,) as follows. Specifically, by marginalizing out t and a in Eq. 2, we get:

$$\begin{aligned} P(v_j|L_i, m=0) &= \sum_{L_k:v_k=v_j} P(L_k|L_i, m=0) \\ &\propto \sum_{L_k:v_k=v_j} \mathcal{W}_{ik}^0 \end{aligned} \tag{4}$$

Since $(v_i, v_j) \in E$, we have:

$$\begin{aligned} P(v_j|L_i, m=0) &\propto \sum_{L_k:v_k=v_j} w_{v_i v_k} e^{-\lambda_1 |t_i - t_k|} \\ &= w_{v_i v_j} \left(e^{-\lambda_1 t_i} \sum_{\substack{L_k:v_k=v_j \\ t_k<t_i}} e^{\lambda_1 t_k} + e^{\lambda_1 t_i} \sum_{\substack{L_k:v_k=v_j \\ t_k \geq t_i}} e^{-\lambda_1 t_k} \right) \end{aligned} \tag{5}$$

Figure 2 illustrates how we compute the weight of social users based on Eq. 5. For each social user v_j, we preprocess it by sorting its behaviors in chronological order and computing the prefix sum of $e^{\lambda_1 t_k}$ and the suffix sum of $e^{-\lambda_1 t_k}$. To compute $P(v_j|L_i, m)$, we first perform a binary search to get the earliest behavior performed by v_j with its time t_k satisfying the constraint $t_k \geq t_i$, then we use the preprocessed prefix sum and suffix sum to compute the sampling probability.

Sampling Social Behavior. When we obtain a sample user in previous step v_j, we continue to sample a social behavior that is performed by v_j. Based on Bayes' theorem, social behavior $L_j = (v_j, t_j, a_j)$ is sampled with the following probability:

$$P(L_j|L_i, v_j, m=0) = \frac{P(L_j|L_i, m=0)}{P(v_j|L_i, m=0)} \propto \mathcal{W}_{ij}^0. \tag{6}$$

Since $\mathcal{W}_{ij}^0$ depends on both i and j, it is challenging to perform a weighted random sampling efficiently. Therefore, we need to first decide whether to sample a behavior performed before t_i or after t_i. Without loss of generality, the probability that we sample v_j's behavior after t_i is:

$$\begin{aligned} P(t_j \geq t_i|L_i, v_j, m=0) &\propto \sum_{\substack{L_k:v_k=v_j \\ t_k \geq t_i}} \mathcal{W}_{ik}^0 \\ &= w_{v_i v_j} e^{\lambda_1 t_i} \sum_{\substack{L_k:v_k=v_j \\ t_k \geq t_i}} e^{-\lambda_1 t_k} \end{aligned} \tag{7}$$

Again, we perform a binary search to get the first behavior performed by v_j with its time $t_k \geq t_i$. Then we can utilize our preprocessed suffix sum of $e^{-\lambda_1 t_k}$ to quickly compute the probability in Eq. 7. Similarly, the probability that we sample v_j's behavior before t_i can be also computed quickly by using our preprocessed

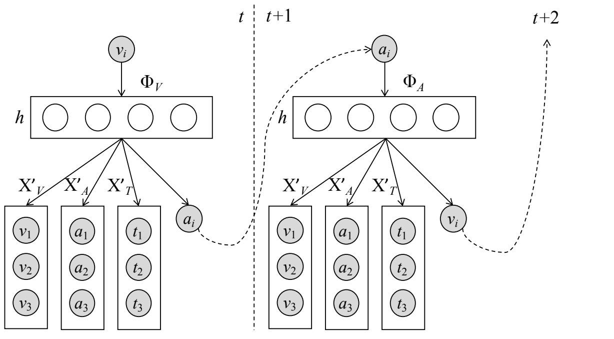


Figure 3: Graphical representation of the proposed embedding learning model. From input layer to hidden layer, the mapping function, Φ_V or Φ_A, maps the current social user/action to its representation. From hidden layer to output layer, three weight matrices X'_V, X'_A, and X'_T are used to compute probabilities in Eq. 9 and Eq. 10.

prefix sum of $e^{\lambda_1 t_k}$. Furthermore, assuming we have chosen to sample a behavior after t_i, we rewrite Eq. 6 as follows:

$$\begin{aligned} &P(L_j|L_i, t_j \geq t_i, v_j, m = 0) \\ &= w_{v_i v_j} e^{\lambda_1 t_i} \times e^{-\lambda_1 t_j} \propto e^{-\lambda_1 t_j} \end{aligned} \tag{8}$$

and it only depends on L_j itself. Next, we use the roulette-wheel selection [17] to sample the behavior with $O(\log|\mathcal{L}|)$ complexity.

Complexity. The time complexity for sampling a behavior when $m = 0$ is $O(|N(v_i)| \log |\mathcal{L}|)$. The situations for $m = 1$ are discussed in Appendix, and also in Table 1, we list the time complexity for sampling different edge types.

3.3 Context Embedding Learning

After discussing how to sample the contexts of social user and social action in § 3.2, we discuss how to learn the social representations for social users/actions based on the sampled contexts. According to the context graph building/sampling methods described in § 3.1-3.2, we can obtain a sequence of "related" behaviors as social context for each social user and action. Afterwards, our task is to learn a "good" representation for each user/action. Specifically, given a behavior and its sampled context, our goal is to learn a function to represent the joint probability of the context: $P(\mathbf{C}(v_i)|v_i)$ and $P(\mathbf{C}(a_i)|a_i)$. In this paper, we estimate the representations by maximizing the two aforementioned joint probabilities for all actions/users. Straightforwardly, we can use word2vec [20] or similar algorithms [2] to compute the vector representations of actions/users. However, they are not applicable directly, because we are dealing with heterogeneous graphs consisting of users and actions. Moreover,besides a universal representation, we also want to learn a time-specific representation which capture dynamic information.

To address the problems of traditional representation learning method in modeling time dynamics, we propose a DeepBehave model which enables a user or an action to have distributed representations when involved in different timestamps.

We first introduce two mapping functions. The first is $\Phi_V : V \longmapsto \mathbb{R}^d$. This mapping Φ_V represents the latent representation associated with each social user in social networks. The second mapping is $\Phi_A : A \longmapsto \mathbb{R}^d$, which represents the latent representation associated with each action. In practice, we represent Φ_V and Φ_A by a $|V| \times d$ matrix and a $|A| \times d$ matrix of free parameters, which will serve as our $\mathbf{X}_V$ and $\mathbf{X}_A$ later.

From input layer to hidden layer, we apply the mapping functions, i.e., Φ_V or Φ_A, to the current social user/action, and get the representations for the social user or the social action. Moreover, from hidden layer to output layer, there are three weight matrices $\mathbf{X}'_V \in \mathbb{R}^{d \times |V|}$, $\mathbf{X}'_A \in \mathbb{R}^{d \times |A|}$ and $\mathbf{X}'_T \in \mathbb{R}^{d \times |T|}$ associated with user, action and time to be predicted. A softmax layer is used in the output layer. Figure 3 presents a graphical representation of the proposed embedding learning model. For each user v_i, we try to learn a good representation to maximally generate its sampled context, which consists of a set of users $\{v\}$, a set of actions $\{a\}$, a set of times $\{t\}$ appearing in the context, and the action performed by user v_i. For each action a_i, analogously, we try to learn a representation to maximally generate users, actions, and times in the sampled context, and also the user who performs the action. The framework is very general and flexible. We can leverage the learned results to iteratively improve the representations of users and actions. Meanwhile, we can also use the presentations learned in previous time stamp to guide the representation learning of users and actions in the current time stamp. The joint probability of $P(\mathbf{C}(v_i)|v_i)$ and $P(\mathbf{C}(a_i)|a_i)$ can be rewritten as:

$$\begin{aligned} P(\mathbf{C}(v_i^{(k)})|v_i^{(k)}) &= P(a_i^{(k)}|v_i^{(k)}) \prod_{\substack{c=-C \\ c\neq 0}}^{C} P(L_i^{(k+c)}|v_i^{(k)}) \\ &= \prod_{c=-C}^{C} \frac{\exp\left(\mathbf{q}_{a_i^{(k+c)}}\right)}{\sum_{j=1}^{|A|} \exp\left(\mathbf{q}_j\right)} \times \prod_{\substack{c=-C \\ c\neq 0}}^{C} \frac{\exp\left(\mathbf{p}_{v_i^{(k+c)}}\right)}{\sum_{j=1}^{|V|} \exp\left(\mathbf{p}_j\right)} \\ &\times \prod_{c=-C}^{C} \frac{\exp\left(\mathbf{r}_{t_i^{(k+c)}}\right)}{\sum_{j=1}^{|T|} \exp\left(\mathbf{r}_j\right)} \end{aligned} \tag{9}$$

$$\begin{aligned} P(\mathbf{C}(a_i^{(k)})|a_i^{(k)}) &= P(v_i^{(k)}|a_i^{(k)}) \prod_{\substack{c=-C \\ c\neq 0}}^{C} P(L_i^{(k+c)}|a_i^{(k)}) \\ &= \prod_{c=-C}^{C} \frac{\exp\left(\mathbf{p}_{v_i^{(k+c)}}\right)}{\sum_{j=1}^{|V|} \exp\left(\mathbf{p}_j\right)} \times \prod_{\substack{c=-C \\ c\neq 0}}^{C} \frac{\exp\left(\mathbf{q}_{a_i^{(k+c)}}\right)}{\sum_{j=1}^{|A|} \exp\left(\mathbf{q}_j\right)} \\ &\times \prod_{c=-C}^{C} \frac{\exp\left(\mathbf{r}_{t_i^{(k+c)}}\right)}{\sum_{j=1}^{|T|} \exp\left(\mathbf{r}_j\right)} \end{aligned} \tag{10}$$

where $\mathbf{p} = \mathbf{h} \times \mathbf{X}'_V$, $\mathbf{q} = \mathbf{h} \times \mathbf{X}'_A$ and $\mathbf{r} = \mathbf{h} \times \mathbf{X}'_T$. The basic idea here is to find the latent representations $\mathbf{h}$ by applying mapping function Φ_V (or Φ_A), and then multiply $\mathbf{h}$ by the weight matrix $\mathbf{X}'_V$ (or $\mathbf{X}'_A$, $\mathbf{X}'_T$) to obtain $\mathbf{p} \in \mathbb{R}^{|V|}$ (or $\mathbf{q} \in \mathbb{R}^{|A|}$, $\mathbf{r} \in \mathbb{R}^{|T|}$).

Thus, to solve the log-likelihood objective function of Eqs. 9-10, we first randomly initialize the latent representations $\mathbf{h}$ for users and actions, and then use a gradient-based algorithm to update the weight matrices $\mathbf{X}'_V$, $\mathbf{X}'_A$, $\mathbf{X}'_T$, $\mathbf{X}_A$ and $\mathbf{X}_V$. Take Eq. 9 as example, the derived gradients are as follows: (Detailed derivation is omitted due to space limitation.)

$$
\begin{aligned}
\frac{\partial E_{v_i^{(k)}}}{\partial \mathbf{X}'_{Vdj}} &= \sum_{\substack{c=-C\\c\neq 0}}^{C} \left(\frac{\exp(\mathbf{p}_j)}{\sum_{j'=1}^{|V|}\exp(\mathbf{p}_{j'})} - 1[v_i^{(k+c)} = j] \right) \mathbf{h}_d \\
\frac{\partial E_{v_i^{(k)}}}{\partial \mathbf{X}'_{Adj}} &= \sum_{c=-C}^{C} \left(\frac{\exp(\mathbf{q}_j)}{\sum_{j'=1}^{|A|}\exp(\mathbf{q}_{j'})} - 1[a_i^{(k+c)} = j] \right) \mathbf{h}_d \\
\frac{\partial E_{v_i^{(k)}}}{\partial \mathbf{X}'_{Tdj}} &= \sum_{c=-C}^{C} \left(\frac{\exp(\mathbf{r}_j)}{\sum_{j'=1}^{|T|}\exp(\mathbf{r}_{j'})} - 1[r_i^{(k+c)} = j] \right) \mathbf{h}_d \\
\frac{\partial E_{v_i^{(k)}}}{\partial \mathbf{X}_{V v_i^{(k)} d}} &= \sum_{\substack{c=-C\\c\neq 0}}^{C} \sum_{j=1}^{|V|} \left(\frac{\exp(\mathbf{p}_j)}{\sum_{j'=1}^{|V|}\exp(\mathbf{p}_{j'})} - 1[v_i^{(k+c)} = j] \right) \mathbf{X}'_{Vdj} \\
&+ \sum_{c=-C}^{C} \sum_{j=1}^{|A|} \left(\frac{\exp(\mathbf{q}_j)}{\sum_{j'=1}^{|A|}\exp(\mathbf{q}_{j'})} - 1[a_i^{(k+c)} = j] \right) \mathbf{X}'_{Adj} \\
&+ \sum_{c=-C}^{C} \sum_{j=1}^{|T|} \left(\frac{\exp(\mathbf{r}_j)}{\sum_{j'=1}^{|T|}\exp(\mathbf{r}_{j'})} - 1[r_i^{(k+c)} = j] \right) \mathbf{X}'_{Tdj}
\end{aligned}
$$

where $E_{v_i^{(k)}} = -\log P(\mathbf{C}(v_i^{(k)})|v_i^{(k)})$ and $1[.]$ is an indicator function, which returns 1 if the condition in [.] is true and 0 otherwise.

Parallel Learning. To speed up learning process for the proposed DeepBehave method, we develop a parallel learning algorithm. We use two different parallel programming APIs in our implementation: OpenMP [5] and PThreads [15, 23]. OpenMP is a high-level and portable API for multi-platform shared-memory parallel programming. More specifically, in context graph sampling, we launch random walks starting from different behaviors in different threads. PThreads (POSIX Threads) is a low-level API for threaded programming which offers fine-grained control over thread management. For embedding learning, we use PThreads to distribute train batches to multiple threads. PThreads was also used in Word2Vec [20].

Implementation Note. We also give the implementation details for the other components in our methods. As calculating the partition function (normalization factor) is expensive, we are not able to directly obtain the probabilities in Eq. 9 and 10, To speed up the training process, we use Hierarchical Softmax [21, 22] to approximate the probability distribution. Moreover, as our goal is to learn representations for both social users and social actions, which come from two different spaces, we build two hierarchical softmax trees for users and actions respectively.

Additionally, we perform a grid search to reach a good configuration of model hyper-parameters. In particular, for weight parameters θ, we search them from 0 to 1.0 with an interval of 0.1, by keeping all other parameters constant. For exponential time decay hyper-parameters λ_m, we search them according to the nature of datasets. For example, the granularity of timestamps in AMiner dataset is year, which ranges from 1975 to 2014, so we search λ_1 and λ_2 in the range of $\frac{1}{50}$ to 1 in our experiment.

Table 2: Summary of datasets.

Metric	AMiner	Weibo
$\|V\|$	512,000	1,776,950
$\|A\|$	593	300,000
$\|E\|$	2,930,000	308,489,739
$\|\mathcal{L}\|$	54,000	23,755,810
Time span	1975 - 2014	Sep 2009 - Sep 2012

4 EXPERIMENTS

4.1 Experimental Setup

Datasets. We apply DeepBehave to two large-scale datasets. Table 2 summarizes the datasets we used in this study.

AMiner. AMiner[1] is a network of academic scholars. The dataset, crawled from AMiner.org [30], comprises of 512,000 authors and 2,930,000 co-authorships during 1975 - 2014. On AMiner, we define a social action as publishing a paper on a specific topic (e.g., social networks). Hence, a behavior log (v, t, a) indicates that author v publishes a paper on topic a in year t.

Weibo. Weibo is the most popular Chinese microblogging service. The dataset is from [33] and can be downloaded here.[2] The complete dataset contains the directed following networks and tweets (posting logs) of 1,776,950 users. Similar to Twitter, there are two types of posts in Weibo, namely original posts (tweets) and reposts (or retweets). The social action is defined as posting a microblog—i.e., the triple (v, t, a) means that user v tweets or retweets the microblog a at time t.

Comparison Methods. We compare DeepBehave with several alternative methods which can handle large-scale networks.

DeepWalk. DeepWalk [24] is an approach for social network embedding. It also uses random walk to extract contextual information from each vertex.

LINE. LINE [29] is an approach proposed for information network embedding. There are two variants: LINE (1st) and LINE (2nd). We choose LINE (2nd) in our experiments because it is applicable to both directed and undirected networks.[3]

node2vec. node2vec [9] is an approach for network node embedding. It optimizes a neighborhood preserving objective by simulating biased random walks.

DeepWalk, LINE, and node2vec can not directly handle the time-dependent social behavior data. To make the comparison more comparable, when training DeepWalk, LINE, and node2vec we treat the social behavior (v, t, a) as an edge from social user v to social action a, which is compatible with the two comparison methods.

We apply the comparison methods for learning social behavior representations and use the learned representations to predict user behaviors. In all experiments, we vary the percentage of training

[1]http://aminer.org/citation
[2]http://aminer.org/Influencelocality
[3]We also did experiments to compare with PTE [28], a LINE-based approach for heterogeneous word-document network embedding. However, the performance of PTE always underperforms LINE on our datasets. Limited by space, we only show performance of LINE in this paper.

Table 3: Prediction performance of publishing behavior on the AMiner dataset as a multi-label classification task.(%)

Metric	Algorithm	10%	20%	30%	40%	50%	60%	70%	80%	90%
Micro-F1	Deepwalk	51.75	54.25	55.18	55.71	55.94	56.14	56.31	56.53	56.96
	LINE	50.13	52.69	53.66	54.25	54.53	54.80	55.07	55.16	55.62
	node2vec	51.41	53.46	54.37	54.81	55.03	55.17	55.56	55.99	56.31
	DeepBehave	**52.99***	**55.49***	**56.34***	**56.87***	**57.20***	**57.42***	**57.74***	**57.98***	**58.24***
Macro-F1	Deepwalk	48.09	50.60	51.76	52.45	52.62	52.80	53.02	53.03	53.27
	LINE	46.23	48.80	49.88	50.40	50.74	51.03	51.32	51.17	51.36
	node2vec	48.20	50.50	51.75	52.36	52.58	52.80	53.34	53.81	54.00
	DeepBehave	**50.00***	**52.88***	**53.85***	**54.65***	**54.90***	**55.10***	**55.53***	**55.79***	**55.76***

Two-sided p-values are significant at: * 1%.

Table 4: Prediction performance of (re)tweet behavior on the Weibo dataset as a binary classification task.(%)

Metric	Algorithm	10%	20%	30%	40%	50%	60%	70%	80%	90%
Precision	DeepWalk	28.07	41.94	48.88	53.80	56.71	58.24	55.82	54.12	53.92
	LINE	30.30	43.80	51.26	55.93	59.49	60.69	**61.96**	**60.08**	52.91
	node2vec	29.35	43.65	52.10	55.01	**59.64**	60.69	60.93	56.88	53.88
	DeepBehave	**30.38**	**45.37**	**51.49**	**58.02**	59.02	**61.63**	61.26	58.77	**57.56**
Recall	DeepWalk	78.05	77.89	78.59	76.50	76.82	77.49	76.34	72.94	71.15
	LINE	77.48	78.02	77.49	76.93	78.18	76.84	79.46	**79.76**	**78.05**
	node2vec	**80.21**	**78.87**	**78.80**	77.01	78.57	77.86	78.36	76.72	75.29
	DeepBehave	79.81	78.26	78.68	**79.07**	**78.98**	**79.50**	**79.60**	77.02	76.88
F1-score	DeepWalk	41.27	54.67	60.25	63.15	65.23	66.48	64.47	62.12	61.33
	LINE	43.54	55.62	61.66	64.76	67.56	67.80	**69.62**	**68.53**	63.05
	node2vec	42.94	56.17	**62.70**	64.17	67.79	68.20	68.55	65.31	62.79
	DeepBehave	**43.99***	**57.69***	62.23	**67.01***	**67.90***	**69.39***	69.21	66.60	**65.35***

Two-sided p-values are significant at: * 1%.

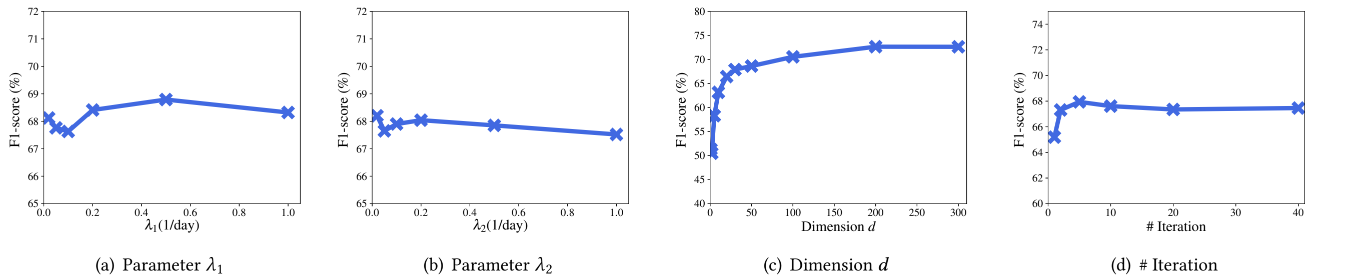
data from 10% to 90% and use 10% as the interval. For each experiment, we randomly split the data into training and testing datasets for 30 times and report the average performance.

4.2 Prediction Performance

On each dataset, we apply the proposed method and the comparison methods to learn social representations, and then apply the learned representations to predict users' behaviors. Tables 3-4 list the prediction performance of the comparison methods on two datasets.

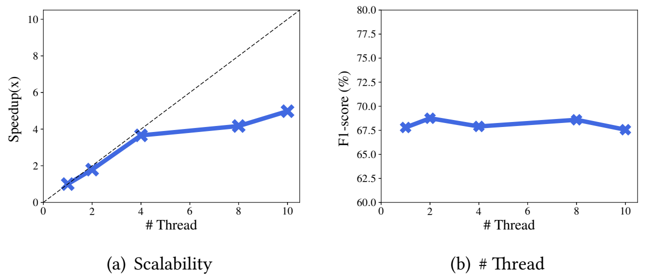
AMiner. We choose seven popular conferences including AAAI, CIKM, ICML, KDD, NIPS, SIGIR, and WWW as the classification categories. We only consider authors who published on those conferences. If an author publishes a paper on any of the conferences, we say that the author performs an action. We use the learned representation for authors and publication keywords as features of a paper, and train a classifier for conference prediction. This task has also been used in [29] for embedding evaluation. The prediction task is a multi-label classification task, so we report Micro-F1 and Macro-F1 measure to show performance in each label. For fair comparison, DeepWalk, LINE, and node2vec are given both co-author network and the behavior information as inputs. Table 3 suggests that the proposed DeepBehave method clearly outperforms all the comparison methods across all metrics. The proposed DeepBehave method is better at incorporating the network structure and the time information, thus achieves the best performance ($\theta = 0.8$). We also conduct the sign test for model comparison and the p-value is smaller than 0.01, which confirms the significance of the performance improvements by the proposed method over the comparison methods.

Weibo. We use different comparison methods to predict whether users will tweet (or retweet) a microblog. The task can also be considered as a binary classification problem. Instead of using Micro/Macro-F1 (which are for multi-label classification) in AMiner dataset, we report Precision, Recall, and F1-score here. The positive instances are randomly chosen from the retweet behavior data, and those instances are not used to train the embedding model. The negative instances are chosen in the following way: (1) randomly sample a microblog a in Weibo; (2) sample a user v from all Weibo users who do not retweet the microblog but have at least one follower or followee who retweet the microblog a. The probability associated with each user is proportional to the number of followers and followees who has retweets the microblog. Finally, we treat each pair (v, a) as a negative instance. In this task, Deepwalk, LINE, and node2vec use social network (following network) information and social behavior (tweet/retweet behavior) information to learn the representations. It is observed from Table 4 that the proposed DeepBehave method outperforms DeepWalk significantly across all metrics, and outperforms LINE and node2vec in most cases. Our method accommodates well to various size of behavior data. For DeepBehave , the best prediction performances are reached with $\theta = 0.4$. In comparison to the experiment on AMiner dataset, with a smaller θ setting on Weibo dataset, the context sampling process is more biased against sampling behaviors sharing the same action.



(a) Parameter λ_1 (b) Parameter λ_2 (c) Dimension d (d) # Iteration

Figure 4: Parameter sensitivity analysis. (a)(b) Parameters λ_1, λ_2 that control time decay ratio; (c) Representation dimension d (d) Algorithm iterations.



(a) Scalability (b) # Thread

Figure 5: Model scalability analysis. (a) Speed-up radio w.r.t the number of threads; (b) Performance w.r.t the number of threads.

4.3 Parameter Analyses

We now investigate how the performance varies with the parameters in the context graph building/sampling and embedding learning. We conduct all the parameter analyses on the Weibo dataset.

Time decay ratio λ. In context graph building, λ_1 and λ_2 controls the time decay ratio for social-based and action-based edges respectively. Figure 4(a) and Figure 4(b) show the prediction performance (in terms of F1-score) by varying the two parameters. The prediction performance is not sensitive to time decay ratio in the evaluated task.

Dimension d. We evaluate how the latent dimension d affects the quality of the learned representations by DeepBehave. We perform an analysis by varying the dimension of latent space in the proposed DeepBehave method. Figure 4(c) shows its prediction performance (in terms of F1-score) with different numbers of latent dimension. We find that performance of DeepBehave gradually stabilized when the latent dimension is larger than 30. Higher latent dimension results in slightly better performance, but requires large training graph.

Number of Iterations. We further investigate the convergence of the learning algorithm for DeepBehave. Figure 4(d) presents the convergence analysis of the algorithm. The algorithm converges within 10 iterations. Furthermore, this rapid convergence enables us to do efficient training of the model on large scale datasets.

4.4 Scalability Performance

We now evaluate the scalability performance of the parallel learning algorithm on the Weibo dataset. Figure 5(a) shows the speedup of the parallel algorithm with different number of computer cores (up to 10 cores). The parallel learning algorithm achieves $\sim 5\times$ speedup with 10 cores. We also evaluate the prediction performance by the parallel learning algorithm. On average, as shown in Figure 5(b), the prediction accuracy under the parallel learning algorithm across different numbers of cores only results in slight difference, which demonstrates the effectiveness of the parallel learning algorithm.

5 RELATED WORK

Our work is closely related to the following two topics including (1) temporal pattern in social networks; (2) representation learning for social networks.

Temporal Pattern in Social Networks Regarding temporal pattern in social networks, most studies focus on developing different methods to capture the key factors that represent temporal patterns. For example, Wei et al. [32] propose two classes of dynamic metrics to model the Recency-Primacy effect, a characterization of human cognitive processes, and their experimental results show that the proposed dynamic metrics can successfully measure temporal evolution patterns and an optimal length of historical information for predicting the existence of future activities. Aggarwal et al. [1] study the event detection problem in social streams. Fakhraei et al. [6] propose a statistical relational model to detect spammers in evolving multi-relational networks by considering both the multi-relational nature of networks and activity sequences. In addition, Costa et al. [7] analyze the distribution of time intervals between posting behaviors, and propose a generative model to match their discovered patterns. Jiang et al. [13] study the behavior prediction and pattern-mining problem, and propose a flexible evolutionary multi-faceted analysis framework.

Representation Learning for Social Networks Distributed representations have been proposed to model structural relationships between concepts [11]. The idea of using distributed representations of words [2, 20] has been widely used in natural language processing tasks, such as speech recognition and machine translation. Moreover, a growing literature has been studying the embedding of large-scale networks. For example, Perozzi et al. [24] propose DeepWalk, which deploys truncated random walks for social network. Tang et al. [29] propose LINE, a network embedding method that preserves both the local and global network structures, they further propose a word embedding model PTE [28] for heterogeneous word-document network. Change et al. [4] propose neural network model with deep architecture for embedding learning in heterogeneous network. Wang et al. [31] propose embedding learning model that preserve detailed structural information by using deep neural network. Grover et al. [9] proposed a node2vec model that capture both homophily and structural equivalence.

6 CONCLUSION

In this paper, we study a novel problem of learning deep representations for social behavior data, and propose an approach called DeepBehave by considering both network structure and dynamic information in the behavioral data. Given the input data, we first build a heterogeneous context graph in which nodes represent user behaviors and edges represent different types of relationships (or correlations) between user behaviors. Then we present a sampling method to construct the graph context for each vertex in context graph and apply embedding learning to the sampled contexts. Our experiments on several large datasets show that the proposed approach significantly improves the performance of behavior prediction over other methods.

APPENDIX: CONTEXT SAMPLING

In context graph sampling, if we sampled $m = 1$ (action-based edge) in the first step, which means we want to sample next behavior L_j through action-based edges by satisfying $a_i = a_j$. L_j could be sampled from all behaviors with action a_i with the following probability based on Bayes' theorem:

$$P(L_j|L_i, a_j, m) = \frac{P(L_j|L_i, m)}{P(a_j|L_i, m)} \propto \mathcal{W}^1_{ij} \tag{11}$$

To compute $\mathcal{W}^1_{ij}$ efficiently, we follow a similar process to sampling with $\mathcal{W}^0_{ij}$. We first decide whether to sample behavior before or after t_i. Without loss of generality, the probability of sampling behavior after t_i is given by:

$$\begin{aligned} P(t_j \geq t_i|L_i, a_j, m = 1) &\propto \sum_{\substack{L_k : a_k = a_j \\ t_k \geq t_i}} \mathcal{W}^1_{ij} \\ &= e^{\lambda_2 t_i} \sum_{\substack{L_k : a_k = a_j \\ t_k \geq t_i}} e^{-\lambda_2 t_k} \end{aligned} \tag{12}$$

We utilize preprocessed suffix sum of $e^{-\lambda_2 t_k}$ and binary search to accelerate computation of Eq. 12. Meanwhile, given $t_k > t_i$, Eq. 11 can be written as:

$$\begin{aligned} &P(L_j|L_i, t_j \geq t_i, v_j, m = 1) \\ &= e^{\lambda_2 t_i} \times e^{-\lambda_2 t_j} \propto e^{-\lambda_2 t_j} \end{aligned} \tag{13}$$

and depends on only L_j. Thus we can yield sampling result with roulette-wheel selection.

REFERENCES

[1] Charu C Aggarwal and Karthik Subbian. 2012. Event Detection in Social Streams.. In *SDM*, Vol. 12. 624–635.
[2] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *JMLR* 3 (2003), 1137–1155.
[3] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet Allocation. *JMLR* 3 (2003), 993–1022.
[4] Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C Aggarwal, and Thomas S Huang. 2015. Heterogeneous network embedding via deep architectures. In *KDD'15*. 119–128.
[5] Leonardo Dagum and Ramesh Menon. 1998. OpenMP: an industry standard API for shared-memory programming. *IEEE computational science and engineering* 5, 1 (1998), 46–55.
[6] Shobeir Fakhraei, James Foulds, Madhusudana Shashanka, and Lise Getoor. 2015. Collective Spammer Detection in Evolving Multi-Relational Social Networks. In *KDD'15*. 1769–1778.
[7] Alceu Ferraz Costa, Yuto Yamaguchi, Agma Juci Machado Traina, Caetano Traina Jr, and Christos Faloutsos. 2015. RSC: Mining and Modeling Temporal Activity in Social Media. In *KDD'15*. 269–278.
[8] Gerhard Fischer. 2001. User Modeling in Human&Ndash;Computer Interaction. *User Modeling and User-Adapted Interaction* 11, 1-2 (2001), 65–86.
[9] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *KDD*.
[10] Paul Heymann, Daniel Ramage, and Hector Garcia-Molina. 2008. Social Tag Prediction. In *SIGIR'08*. 531–538.
[11] Geoffrey E Hinton. 1986. Learning distributed representations of concepts. In *CogSci'06*, Vol. 1. 12.
[12] Thomas Hofmann. 1999. Probabilistic Latent Semantic Indexing. In *SIGIR'99*. 50–57.
[13] Meng Jiang, Peng Cui, Fei Wang, Xinran Xu, Wenwu Zhu, and Shiqiang Yang. 2014. FEMA: flexible evolutionary multi-faceted analysis for dynamic behavioral pattern discovery. In *KDD'14*. 1186–1195.
[14] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.
[15] Bil Lewis and Daniel J Berg. 1998. *Multithreaded programming with Pthreads*. Prentice-Hall, Inc.
[16] Xirong Li, Cees GM Snoek, and Marcel Worring. 2009. Learning social tag relevance by neighbor voting. *IEEE TMM* 11, 7 (2009), 1310–1322.
[17] Adam Lipowski and Dorota Lipowska. 2012. Roulette-wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and its Applications* 391, 6 (2012), 2193–2196.
[18] László Lovasz. 1993. Random walks on graphs: A survey. *Combinatorics* 2, 1 (1993), 1–6.
[19] Hao Ma, Haixuan Yang, Michael R Lyu, and Irwin King. 2008. Sorec: social recommendation using probabilistic matrix factorization. In *CIKM'08*. 931–940.
[20] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS'13*. 3111–3119.
[21] Andriy Mnih and Geoffrey E Hinton. 2009. A scalable hierarchical distributed language model. In *NIPS'09*. 1081–1088.
[22] Frederic Morin and Yoshua Bengio. 2005. Hierarchical Probabilistic Neural Network Language Model.. In *AISTATS*, Vol. 5. 246–252.
[23] Bradford Nichols, Dick Buttlar, and Jacqueline Farrell. 1996. *Pthreads programming: A POSIX standard for better multiprocessing.* " O'Reilly Media, Inc.".
[24] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD'14*. 701–710.
[25] Steffen Rendle. 2012. Factorization machines with libFM. *ACM TIST* 3, 3 (2012), 57.
[26] Chenhao Tan, Jie Tang, Jimeng Sun, Quan Lin, and Fengjiao Wang. 2010. Social Action Tracking via Noise Tolerant Time-varying Factor Graphs. In *KDD'10*. 1049–1058.
[27] Jiliang Tang, Xia Hu, and Huan Liu. 2013. Social recommendation: a review. *SNAM* 3, 4 (2013), 1113–1133.
[28] Jian Tang, Meng Qu, and Qiaozhu Mei. 2015. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *KDD'15*. 1165–1174.
[29] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale Information Network Embedding. In *WWW'15*. 1067–1077.
[30] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. ArnetMiner: Extraction and Mining of Academic Social Networks. In *KDD'08*. 990–998.
[31] Daixin Wang and Wenwu Cui, Peng amd Zhu. 2016. Structural Deep Network Embedding. In *KDD'16*.
[32] Wei Wei and Kathleen M Carley. 2015. Measuring temporal patterns in dynamic social networks. *ACM TKDD* 10, 1 (2015), 9.
[33] Jing Zhang, Biao Liu, Jie Tang, Ting Chen, and Juanzi Li. 2013. Social Influence Locality for Modeling Retweeting Behaviors.. In *IJCAI'13*, Vol. 13. 2761–2767.