NActSeer: Predicting User Actions in Social Network using Graph Augmented Neural Network

Mohammad Raihanul Islam
Virginia Tech, USA
raihan8@cs.vt.edu

Sathappan Muthiah
Virginia Tech, USA
sathap1@cs.vt.edu

Naren Ramakrishnan
Virginia Tech, USA
naren@cs.vt.edu

ABSTRACT

Nowadays social network platforms like Twitter, Facebook, Weibo have created a new landscape to communicate with our friends and the world at large. In this landscape our social activities, purchase decisions, check-ins etc. become available immediately to our friends/followers and thus encouraging them to involve in the same activity. This gives rise to the question, given a user and her friends' previous actions, can we predict what is she going to do next? This problem can serve as a good indicator enabling policy research, targeted advertising, assortment planning etc. To capture such sequential mechanism two broad classes of methods have been proposed in the past. First one is the Markov Chain (MC), which assumes user's next action can be predicted based on her most recently taken actions while the second type of approach i.e. Recurrent Neural Network (RNN) tries to model both long and short term preferences of a user. However, none of the two classes of models contain any integrated mechanism to capture the preferences of neighbor's actions. To fill this gap, we propose a social network augmented neural network model named *NActSeer* which takes the neighbors' actions into account in addition to the user's history. To achieve this *NActSeer* maintains a dynamic user embedding based on the activities within a time window. It then learns a feature representation for each user which is augmented by her neighbors. Empirical studies on four real-world datasets show that *NActSeer* is able to outperform several classical and state-of-the-art models proposed for similar problems and achieves up to 71% performance boost.

CCS CONCEPTS

• **Information systems** → **Data mining**; • **Human-centered computing** → **Social network analysis**; *Social media*; • **Computing methodologies** → *Neural networks*.

KEYWORDS

Social Network, User Activity Prediction, Representation Learning

ACM Reference Format:
Mohammad Raihanul Islam, Sathappan Muthiah, and Naren Ramakrishnan. 2019. *NActSeer*: Predicting User Actions in Social Network using Graph Augmented Neural Network. In *The 28th ACM International Conference on Information and Knowledge Management (CIKM '19), November 3–7, 2019, Beijing, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3357384.3358032

CIKM '19, November 3–7, 2019, Beijing, China

ACM ISBN 978-1-4503-6976-3/19/11…$15.00
https://doi.org/10.1145/3357384.3358032

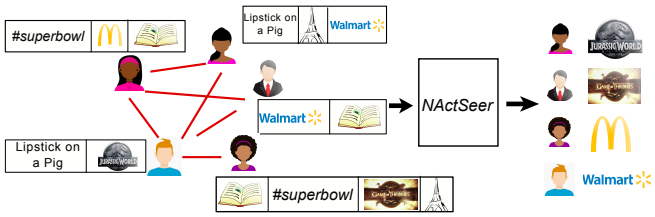


Figure 1: A example of our *NActSeer* model functionality wherein the sequence of actions by a user e.g. reading a book, watching a movie, checking into a location or sharing specific memes (hashtag) on social media is taken as input along with the users' network connections. *NActSeer* then makes use of these information to predict the actions likely to be performed next by the user (best viewed in color).

1 INTRODUCTION

With the proliferation of social network platforms such as Facebook, Twitter, Weibo, Instagram into our daily life, millions of people can now connect with their friends and keep track of the person they are interested in. One of the common features of social network platforms is to provide different mechanisms for interacting with a user's content such as like, comment, share (retweet) etc. These mechanisms help inform our friends about our new activities and also help discover their activities which can have a significant influence on our future actions. This constant sharing of information about one's activity in social media encourages some sort of "bandwagon" effect. Here activities/actions refer to any type of involvement in a social network site such as checking in a new location, publishing a new meme, or commenting on a social media post etc. An example of such phenomena is shown in Fig. 1, wherein users pick up actions from their friends. Modeling and predicting user activity in social media is important as it has a wide array of applications e.g. in targeted advertising [26], policy making [36] etc. Moreover it has a far reaching effect on different aspects of the society e.g. in politics [4], scientific discovery [9], commercial advertising [20].

Modeling user behavior in social network has been a long studied problem in the literature [1, 8, 24, 28, 39]. For instance, Cheng et al. [8] used a combination of structural, temporal and textual features to predict the final cascade growth. Li et al. [24] introduced an end-to-end model to estimate cascade size using a Recurrent Neural Network (RNN). Arguably majority of these works are focused on finding the macroscopic pattern in social network and

not designed towards more granular level prediction. Very recently several works have been proposed to predict user behavior on personal level [33, 41] by leveraging representation learning, wherein the problem is cast as a binary classification problem. However, such a formulation does not take the sequential nature into account therefore evidently loses the information on dependency between the actions. This motivates us to push this problem into a sequential domain. Specifically, given a set of users' action history and the connectivity links between them our objective is to predict their next actions.

Given a sequence of items or entities predicting the next entity is a long studied problem in many domains e.g. text mining [37], time-series prediction [7], traffic forecasting [25] etc. Generally there are two broad classes of methods to model such problems. First one is the markov chain model which exploits only the most recent entities to predict the next entity. Secondly in the recent past, deep learning models like RNN have been utilized to perform this type of task. RNN can capture both short and long term transitions between entities within a sequence at the cost of having more parameters. Yet, the problem is more complicated in our context as users continuously adopt their friends' actions. However, RNN and its variants are not geared towards that.

Present Work: Inspired by the success of representation learning we present a deep learning architecture in this paper. We fruitfully integrate the concept of feature representation for node in graph and sequential prediction into a unified architecture to design our proposed model *NActSeer*. For this purpose we introduce an additional gate to aggregate the recent actions of a user's neighbors and fuse it with his/her action representation. We integrate two components to achieve this. First we generate a representation for each user for each time span in our observation window. This representation summarizes user's actions and can be treated as user's feature vector for that time span. Then we predict the user's next action from her current state and her neighbors' embedding.

Our contributions are as follows:

(1) **Fine-grained Problem Formulation:** We extend the binary classification problem of user activity prediction in social network into a sequential domain. The underlying problem is to predict a user's next action given all the user's previous actions and the connectivity between them.

(2) **Novel Deep Architecture:** Given this problem formulation we propose a deep learning model called *NActSeer*. Our model can aggregate the previous actions of a user's neighbors and can use this summarized representation to reinforce the prediction of next action. In this work we build upon the Long short-term memory (LSTM), however, this can be generalized to similar RNN variants. *NActSeer* facilitates a hitherto uncombined components of RNN, log-bilinear model, graph convolution, node representation etc. A general comparison of features between *NActSeer* and other methods proposed for similar problems in shown in Table 1.

(3) **Rigorous Empirical Evaluation:** We conduct a thorough empirical evaluation of *NActSeer* with several state-of-the-art methods in different experimental settings to illustrate the effectiveness of *NActSeer*. *NActSeer* achieves up to 71% performance boost over the state-of-the-art. Our analysis also shows *NActSeer* is able to capture a better action-action correlation that is helpful in predicting the actions of new users. We also found out having more neighbors improves overall performance with a diminishing return.

Table 1: Comparison of various features between *NActSeer* and state-of-the-art methods proposed for similar problems.

Methods	Sequential Prediction	Compute User Embedding	Aggregate Network Information	Dynamic User Embedding
GCN [21]	✘	✔	✔	✘
GraphSage [13]	✘	✔	✔	✘
LSTM	✔	✘	✘	✘
Caser [40]	✔	✔	✘	✘
SASRec [19]	✔	✘	✘	✘
NActSeer	✔	✔	✔	✔

2 PROBLEM FORMULATION

In this section, we introduce the necessary concepts and provide a formal problem definition. Consider an undirected social network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of users and $\mathcal{E} = \mathcal{V} \times \mathcal{V}$ represents the connections between them. Suppose, in the network a user $v \in \mathcal{V}$ can perform a fixed set of *actions* $\mathcal{A}$. Actions represents any activity performed by the user e.g. location check-ins, publishing new memes or commenting on a particular photo etc. User can repeat the same action many times e.g. commenting on the same post or checking into the same place multiple times. An action performed by a user $v \in \mathcal{V}$ can be represented as a tuple $c_l^v = \langle a_l^v, s_l^v \rangle$, where $a_l^v \in \mathcal{A}$ and s_l^v represents the timestamp. l is index of this action i.e. user v's l^{th} action. The timestamp represents the actual time when user performs the action. Let us assume the entire period of all the user history is L. Now for each user we can obtain an action history $\mathbf{p}_v$ for each $v \in \mathcal{V}$. Now suppose all the users' history is stored in $\mathcal{H}$.

Problem Statement: Given a static social network $\mathcal{G}$ and all the users' history the goal of *Network Activity Prediction* is to predict the next action user v is going to perform. Formally given $\mathbf{p}_v, v \in \mathcal{V}$, where $\mathbf{p}_v = \{c_1^v, c_2^v, \dots c_l^v\}$ our objective is to predict the next action a_{l+1}^v.

3 *NACTSEER* DESCRIPTION

Predicting the next action of a user can be modeled by a sequence capturing method like Recurrent Neural Network (RNN). However RNN or its variants e.g. Long short-term memory (LSTM) or Gated Recurrent Unit (GRU) can only take the sequence of user's previous actions to predict her next probable action. There is no mechanism to integrate her neighbors' action history into these models. However, it is very much possible that a user can be influenced by her friends or even friends of friends. As a result, we have the following intuitions towards solving our *Network Activity Prediction* problem. First, user's next action can be determined by her own previous actions. And secondly, neighbors' recent actions can also play a key role in the prediction. In order to take our assumptions into account, our proposed *NActSeer* model considers both user's previous action and her neighbors to predict the next action. Now in describing the mechanism of *NActSeer* we first describe how to generate a

Table 2: Important notations used in the paper

Notation	Description
$\mathcal{G}$	The network between the users
$\mathcal{V}$	The set of users
$\mathcal{A}$	Set of all the actions a user can take
$\mathcal{H}$	The entire action history of all the users
$\mathbf{p}_v$	The original action history for user v
$\mathbf{q}_v$	The time span tagged user history for user v
$\mathbf{X} \in \mathbb{R}^{\lvert\mathcal{A}\rvert \times D_a}$	The action embedding matrix
$\mathbf{U} \in \mathbb{R}^{\lvert\mathcal{V}\rvert \times T \times \lvert D_a\rvert}$	User embedding for all the time spans
$\mathbf{U}_t \in \mathbb{R}^{\lvert\mathcal{V}\rvert \times \lvert D_a\rvert}$	User embedding for time span t
$\mathbf{U}_{v,t} \in \mathbb{R}^{\lvert D_a\rvert}$	User v's embedding for time span t
$\tilde{\mathbf{U}} \in \mathbb{R}^{\lvert\mathcal{V}\rvert \times T \times \lvert D_a\rvert}$	Network aggregated user embedding for all time spans
$\tilde{\mathbf{U}}_t \in \mathbb{R}^{\lvert\mathcal{V}\rvert \times \lvert D_a\rvert}$	Aggregated user embedding for time span t
$\mathbf{D}$	Diagonal degree matrix
$\mathbf{W}$	Adjacency weight matrix
L	The entire period of user history
T	The number of time spans
P	The number of diffusion steps
K	User history size for each time span
D_a, D_m	action embedding size and state size

summary of the user's actions. Then we outline how the neighbor's summary is aggregated with the user's previous action. The set of important notations we use is listed in Table 2.

3.1 Modeling User History

Representing Actions: Inspired from the recent success in representation learning we represent each action by a fixed length vector. Suppose the embedding of all the actions $\mathcal{A}$ are stored in a matrix $\mathbf{X} \in \mathbb{R}^{|\mathcal{A}| \times D_a}$, where D_a is the size of embedding for actions.

Discretizing the Timestamp of Actions: To integrate neighbor's previous actions into *NActSeer* we need a way to summarize the actions for each user. To accomplish this for every action of a user, we have to compute the summary of her neighbors' recent actions. However, since social network usually contains huge number of users with lots of activities this would impose a huge computational overhead. To resolve this we divide the entire period of user actions into a fixed number of time spans. For example if the entire time period (i.e. our observation window) (L) ranges over multiple years then we can divide it by months or if it spans over months then division by days can be a suitable choice. Suppose we divide the entire time period L into T fixed time spans. Recall that user's entire history is $\mathbf{p}_v = \{c_1^v, c_2^v, \dots c_l^v\}$, where $c_l^v = \langle a_l^v, s_l^v \rangle$. Now if we divide the user action by time span then it becomes $\mathbf{q}_v = \{\langle a_1^v, t_1^v \rangle, \langle a_2^v, t_2^v \rangle, \dots \langle a_l^v, t_l^v \rangle\}$. Here t_l^v represents the discrete time span in which s_l^v falls i.e. $t_{l-1}^v < s_l^v \leq t_l^v, t_l^v \in \mathbb{N}$.

Representing Users by Their Actions: After tagging each user's action with a discrete time span we can compute the user embedding for each time span using the action embedding matrix $\mathbf{X}$. We propose in our case user embedding should be computed for each time span as user preferences are continuously changing. Now suppose the user embedding matrix is $\mathbf{U} \in \mathbb{R}^{|\mathcal{V}| \times T \times |D_a|}$. To compute the user embedding $\mathbf{U}_{v,t}$ for $v \in \mathcal{V}$ at time span t we use the actions she takes on time span t. However, within a time span a user can take a lot of actions, this is especially true for active users. Therefore we consider K most recent actions for a time span t i.e. $\langle c_{t,n_t-K+1}^v, c_{t,n_t-K+2}^v \dots c_{t,n_t}^v \rangle$, where n_t is the number of actions within the time span t. On the other hand, if the user performs less than K actions in a time span we can pad it with actions from the previous time span. Here K is a parameter to be set manually. Given this sequence of actions we can employ any sequence model to compute the user embedding. However, a complex model would require more computational power. Therefore we use the log-bilinear model (LBL) [22, 29] to accomplish this. LBL is a feed forward neural network model with one hidden layer. Now given the K most recent actions of a user v at time span t the user embedding $\mathbf{U}_{v,t}$ can be computed using the following equation:

$$\mathbf{U}_{v,t} = \sum_{k=1}^{K} Y_k e_{t,k}^v \tag{1}$$

where $Y \in \mathbb{R}^{K \times |D_a| \times |D_a|}$ is a trainable parameter of the model and $e_{t,k}^v$ is the embedding of k^{th} action in time span t for user v. In this way given the entire user history $\mathcal{H}$ of all the users we can compute the user embedding for every time span. For a given time span t, $\mathbf{U}_{v,t}$ summarizes user v's activity during t.

Leveraging action embedding to compute the user embedding has several benefits. Since the embedding of a user will be used in predicting her neighbors next action, computing user embedding this way is more beneficial as it directly comes from the action embedding matrix $\mathbf{X}$. Static user features (e.g profile information) are less valuable here as they are hardly correlated with user's next action preference. Moreover this type of information may not be always available. However, if static feature for each user is also available they can also be incorporated here by concatenation. Lastly, in using such mechanism no retraining is required to obtain embedding for a new user.

3.2 Aggregating Neighbor's Preferences

After computing the user embedding for each time span we can now aggregate it with the neighbor's action preference. Recently several Convolutional Neural Network (CNN) models have been proposed which aggregate neighbor's feature to generate representation for each node [5, 21, 25]. We can leverage similar architectures to compute an aggregated representation of the neighbor's recent actions. Suppose $A(\mathcal{G})$ captures the network structure of graph $\mathcal{G}$. Now given all users' embedding $\mathbf{U}_t \in \mathbb{R}^{|\mathcal{V}| \times D_a}$ on a time span t, the network aggregated user representation $\tilde{\mathbf{U}}_t \in \mathbb{R}^{|\mathcal{V}| \times D_a}$ for the P^{th} order neighborhood can be computed as follows:

$$\begin{aligned} \tilde{\mathbf{U}}_t^P &= NetAgg(\mathbf{U}_t) \\ &= \left[\sum_{i=0}^{P-1} [A(\mathcal{G})]^i \, \mathbf{U}_t \right] \\ \tilde{\mathbf{U}}_t &= [\tilde{\mathbf{U}}_t^1; \dots; \tilde{\mathbf{U}}_t^P] W_n + b_w \end{aligned} \tag{2}$$

Here, $W_n \in \mathbb{R}^{P \times D_a \times D_a}$ and $b_w \in \mathbb{R}^{D_a}$ are the model parameters. Specifically $\sum_{p=0}^{P-1} [A(\mathcal{G})]^p \, \mathbf{U}_t$ results in a matrix of user embedding for the P^{th} diffusion step. We repeat this process for each diffusion

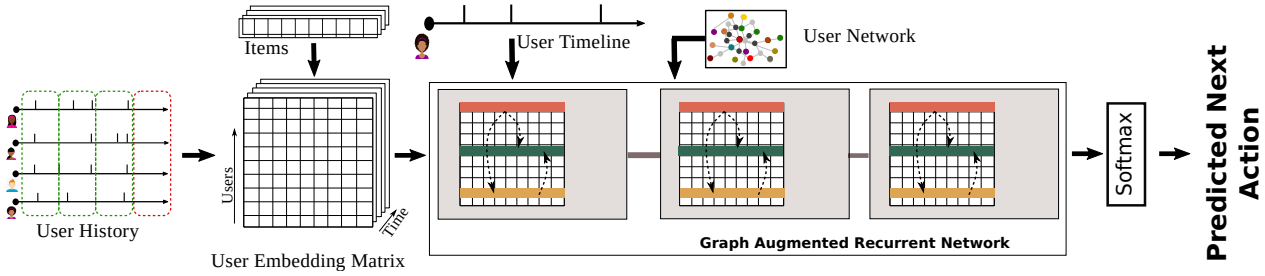


Figure 2: The overall architecture of *NActSeer*. First the timestamp of the entire action history of all the users is discretized into time spans. Then we apply log-bilinear model to obtain user embedding for each time span. The individual user action history along with the user embedding is then fed to the LSTM part of the *NActSeer* model, where user embedding is aggregated given the social network $\mathcal{G}$ (shown in dashed line). At the end of the sequence the hidden state of the LSTM is passed through a softmax layer for next action prediction. We illustrate several possible actions a user can take in Fig. 1.

step to obtain P such user embeddings (i.e. each represents p^{th} order neighborhood where $0 \leq p < P$). Then we concatenate them and pass it to a fully-connected network to obtain $\tilde{\mathbf{U}}_t$. This way we can aggregate not just immediate neighbors but higher-order neighborhoods as well. Now for the choice of $A(\mathcal{G})$ we use normalized graph Laplacian matrix. Given $\mathbf{D}$ the diagonal degree matrix and $\mathbf{W}$ the adjacency weight matrix of $\mathcal{G}$ the normalized graph Laplacian matrix is

$$A(\mathcal{G}) = \mathbf{D}^{-\frac{1}{2}}(\mathbf{D} - \mathbf{W})\mathbf{D}^{-\frac{1}{2}}$$

3.3 Complete Architecture of *NActSeer*

The input to *NActSeer* is the time span tagged user action history $\mathbf{q}_v = \{\langle a_1^v, t_1^v\rangle, \langle a_2^v, t_2^v\rangle, \ldots \langle a_l^v, t_l^v\rangle\}$. This is passed to the embedding layer to convert it to a fixed length vector input.

Embedding Layer: Given $\mathbf{q}_v = \{c_1^v, c_2^v, \ldots c_l^v\}$, where $c_l^v = \langle a_l^v, t_l^v\rangle$ we represent each action $a_l^v \in \mathcal{A}$ in the sequence as a vector. Recall that embedding of all the actions are stored in a matrix $\mathbf{X}$. To extract the embedding a_q of a specific action we can use the corresponding one-hot vector q of size $|\mathcal{A}|$. Now given $\mathbf{q}_v$ we can obtain the action embedding sequence $\mathbf{x}_v = \{x_{v_1}, x_{v_2}, \ldots x_{v_l}\}$ where $x_i \in \mathbb{R}^{D_a}$. Additionally we can gather the corresponding user embedding for each $t_i^v, \forall i \in l$. As a result, we obtain the embedding sequence for a user, $\mathbf{r}_v = \{\mathbf{U}_{t_1}, \mathbf{U}_{t_2}, \ldots \mathbf{U}_{t_l}\}$ using Eqn. 1. It is to be noted that for each single user v the corresponding embedding sequence consists of the embedding of all other users i.e. $\mathbb{R}^{|\mathcal{V}|\times D_a}$ at each corresponding time span (as it is required for $NetAgg$ in Eqn 2). To avoid confusion between $\mathbf{U}_t$ and $\mathbf{U}_{v,t}$ (which only refers to a single user's embedding i.e. $\mathbb{R}^{D_a}$) we use the notation $\mathbf{U}_{t_l^v}$ when discussing about the user sequence r_v. So from here on $\mathbf{r}_v = \{\mathbf{U}_{t_1^v}, \mathbf{U}_{t_2^v}, \ldots \mathbf{U}_{t_l^v}\}$, where $\mathbf{U}_{t_l^v} \in \mathbb{R}^{|\mathcal{V}|\times D_a}$. Finally, both the user sequence and the action sequence are concatenated and given as input to the *NActSeer* model.

Layer Normalization: We adopt the layer normalization [2] technique to normalize the inputs across features i.e. zero mean and unit variance to help improve stability and accelerate the training process. Suppose the input vector is $\mathbf{x}$ then layer normalization is defined as

$$\text{LayerNorm}(\mathbf{x}) = \alpha \odot \frac{\mathbf{x} - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta$$

Here μ and the σ are the mean and variance of $\mathbf{x}$ and α and β are learned parameters (i.e. scaling factors and bias). We apply the layer normalization on the concatenated input sequences.

***NActSeer* Operation:** *NActSeer* takes layer-normed action sequence embedding $\mathbf{x}_v$ and user embedding $\mathbf{r}_v$. The *NActSeer* equations are:

$$\tilde{\mathbf{U}} = NetAgg(\mathbf{U}_{t_j^v}) \tag{3a}$$
$$\tilde{x}_{v_j} = x_{v_j} \oplus \tilde{\mathbf{U}}_{v,t_j^v} \tag{3b}$$
$$i_j = \sigma(W^i \tilde{x}_{v_j} + Q^i h_{j-1} + b_i) \tag{3c}$$
$$f_j = \sigma(W^f \tilde{x}_{v_j} + Q^i h_{j-1} + b_f) \tag{3d}$$
$$\tilde{C}_j = \tanh(W^c \tilde{x}_{v_j} + Q^c h_{j-1} + b_c) \tag{3e}$$
$$C_j = \tilde{C}_j \odot i_j + f_j \odot C_{j-1} \tag{3f}$$
$$o_j = \sigma(W^o \tilde{x}_{v_j} + Q^o h_{j-1} + b_o) \tag{3g}$$
$$h_j = o_j \odot \tanh(C_j) \tag{3h}$$

Here $W^* \in \mathbb{R}^{D_m \times D_a}$ and $Q^* \in \mathbb{R}^{D_m \times D_m}$ are the parameters of different gates within the RNN cell of *NActSeer* and $\oplus$ represents element-wise addition.

Next Action Prediction: The score of an action to be taken next can be computed by the following equation: $w_{a_{j+1}} = V_a h_j + b_a$. Here $V_a \in \mathbb{R}^{|\mathcal{A}|\times D_m}$ and $b_a \in \mathbb{R}^{|\mathcal{A}|}$. Now we can use a softmax layer to compute the probability of each action a being adopted.

$$p_{\tilde{a}_{j+1}|h_j} = \frac{\exp(w_{a_{j+1}})}{\sum_{z\in\mathcal{A}} \exp(w_{z_{j+1}})} \tag{4}$$

The final objective function the becomes

$$O = \underset{\theta}{\text{argmin}} \left(- \sum_{v\in\mathcal{V}} \sum_{j=1}^{l} \left[\log \left(p_{\tilde{a}_{j+1}=a_{j+1}|h_j} \right) \right] \right) \tag{5}$$

where θ is the set of all the model parameters.

3.4 Expediting User Embedding Computation:

From Eqn. 3 we see that at each step of our recurrent network we have to compute Eqn. 2. This imposes a huge computational burden on the overall training process. So, we modify the computational

Algorithm 1: Computing User Embedding for a mini-batch

Input: a mini-batch of size B of time span tagged user activity $\mathbf{q}_v$ for $v \in \mathcal{V}$
Output: Network aggregated user embedding matrix $\tilde{\mathbf{U}}_t$ for the mini-batch

```
for t ← 1 to T do
    foreach v ∈ V do
        U_{v,t} = Σ_{k=1}^{K} Y_k c^v_{t,k}
for t ← 1 to T do
    calculate Ũ_t from Eqn. 2
BatchUserEmbedding ← list()
for b ← 1 to B do
    for j ← 1 to l do
        extract user embedding Ũ_{v,t_j^v} from Ũ_t given time span t_j^v
        add Ũ_{v,t_j^v} to BatchUserEmbedding
return BatchUserEmbedding ∈ R^{B×l×D_a}
```

Table 3: Statistics of the datasets used in the experiments.

Datasets	# of actions	# of users	# of edges	avg. # of actions / user	avg. # of users / action
Flickr	7568	12173	506283	30.081	27.0163
Flixster	12246	17532	114345	29.105	32.679
Gowalla	14897	18712	189206	29.031	29.691
Digg	3553	20002	47466	20.795	24.009

flow to pre-compute $\tilde{\mathbf{U}}$. For a single mini-batch we compute the user embedding matrix $\mathbf{U}$ only one time then compute Eqn. 2 for every time span t to obtain $\tilde{\mathbf{U}}$. We then extract the network aggregated user embedding for corresponding time span as and when necessary. A pseudocode is shown in Algorithm 1. The output of Algorithm 1 is a matrix of network aggregated user embedding $BatchUserEmbedding \in \mathbb{R}^{B \times l \times D_a}$ for a mini-batch of size B and sequence length l. Now in the *NActSeer* cell we do not need to compute Eqn. 3(a). We can just add the embedding (i.e. Eqn. 3(b)) and then execute Eqn. 3(c–h). This removes the burden of computing Eqn. 2 too many times and expedite training process. The code for *NActSeer* is public available[1].

4 EXPERIMENTAL FINDINGS

4.1 Experimental Setup

Datasets: We evaluate our model on four real-world datasets from different domains. Some statistical information regarding the datasets is shown in Table 3. The description of the datasets are:

- **Flickr** [6] is a photo sharing website where a user can mark a photo as favorite and also make friends with other users. We consider the act of marking a photo as an action. Here the task is to predict which photo a user will mark next as her favorite.
- **Flixster** [18] is a movie rating website where users can become friends with each other. Here giving rating to a movie is considered as an action. Our goal here is to predict which movie a user will rate next.
- **Gowalla** [27] is a location sharing website where users can share their current location. Users can also make friends with other users. Each location checked-in by a user is considered to be an action in our study. The objective here is to predict a user's next location.
- **Digg** [16] is a popular news aggregator. Here users can vote the news they like, which is treated as an action. Users can also form friendship here which constitute the social network. We predict the news post a user will vote for.

Competing Methods: We compare our model with state-of-the-art methods proposed for related problems. In order to have a thorough evaluation, we compare it against methods that can model sequential input and also methods that can make use of network structure.

- **Bayesian Personalized Ranking** (BPR) [34] is a popular method in recommender system. We use a binary vector of size $|\mathcal{A}|$, wherein actions performed by a user in the past take the value of 1.
- **LSTM** is a widely popular RNN model for sequence modeling. For this model we use the user's previous action sequence to predict the next action.
- **GCN** [21] is a CNN based model to learn feature representation for nodes in graph. For input feature representation for users we use similar input as used for BPR.
- **GraphSage** [13] is one of the state-of-the-art models for generating node embedding in large networks. It uses sampling technique and various types of aggregators to learn features from a node's local neighborhood. Specifically we use the mean aggregator which shows the best result for this problem. Here we use the same feature representation as GCN for input.
- **Caser** [40] is a recently proposed CNN-based method for sequential recommendation. It captures the sequential nature of actions by applying convolutional operations on the embedding matrix of the most recently accessed items.
- **SASRec** [19] is a state-of-the-art model for sequential recommendation. It uses positional embedding and multi-head attention mechanism to detect the most relevant items.

For GCN, GraphSage, SASRec and Caser we use the codes published by the original authors. We also tried Decision Tree and SVM classifier using the actions as features. However they show very poor results (e.g. lower than 2% percentage value), so we skip them.

Parameter Settings: Unless otherwise specified we set the state size of the model $D_m = 64$, user/action embedding size $D_a = 64$, context size $K = 1$, dropout prob. $\Delta = 0.3$. We run 100 iterations of each model and report the best result. We set the value of T to 12 empirically.

Evaluation Metric: Given a users action history, acquiring the next action can be treated as a retrieval task since an arbitrarily large number of actions can be selected. Therefore an intuitive way for evaluation is to apply ranking metrics used in information retrieval. For this to work we rank all the actions by their probabilities and consider the relevant action to be the actual actions taken by the users. We use two widely popular ranking methods:

- **map@κ:** This represents the Mean Average Precision used in information retrieval.
- **hits@κ:** The rate of top κ ranked actions containing the actual next action taken by the user.

In the following sections we report and discuss about the performance of *NActSeer* under different experimental settings.

[1] https://github.com/raihan2108/NActSeer

4.2 How Effective Is *NActSeer* in Predicting the Next User Action?

We show the performance of different methods in Fig. 3. We make the following observations:
(1) *NActSeer* outperforms all the methods by a very good margin. Performance gain ranges from around 5–71% in terms of hits@20 across different datasets. On the other hand for map@20 it is around 5–34%. For lower values of κ sometimes the gain is even higher. This demonstrates that *NActSeer* fruitfully combines the user's previous action and her neighbors' actions to predict effectively.
(2) Although methods like GCN and GraphSage uses neighborhood information they cannot exploit the sequential nature. As a result they do not perform well.
(3) Though LSTM, SASRec and Caser can model the sequential nature of the input they all suffer from the network aggregation problem. Therefore their performance also hurts.
(4) We also notice that the non-neural method like BPR is not able to perform quite well in this setting as it optimizes a very low number of parameters compared to neural models. This demonstrates that neural models are capable of capturing the action transition better than the non-neural methods in this case.
(5) We observe that performance improves as κ increases. This is expected since the target actions is more likely to be included if more candidate actions are considered. Moreover, map@κ scores are relatively lower as they also consider the position of the true action among the candidate actions.

4.3 How Effective Is *NActSeer* in Predicting the Actions of New Users?

To evaluate how does *NActSeer* perform in predicting the actions of new users we train all the models on the action history of 70% of users. Then we predict the actions of 20% of users (10% is used for validation). During training we only use the induced subgraph of users of the training set while in testing we use the entire graph. The comparison is shown in Fig. 4. Here our observations are as follows:
(1) *NActSeer* outperforms other methods with a gain of 17–33% for hits@20 and 23–68% for map@20. *NActSeer* is able to make use of the neighbors' action to predict future actions for new users. Moreover, since *NActSeer* represents a user by the actions she takes, even the new user embedding is computed by the action embedding ($\mathbf{X}$) learned during the training. Hence *NActSeer* does not have to learn the nuances of a new user allowing the model to improve its prediction capability.
(2) SASRec is able to perform better in this experiment than Caser because it does not have a separate user embedding matrix as Caser. We believe since Caser cannot learn the user embedding for new users its performance suffers. We note this observation is in line with the results obtained in Kang et al. [19].
(3) GCN and GraphSage exhibits similar problems as before, though they use a feature vector based on previous actions, they are unable to learn the relationships between actions due to the non-sequential mechanism. As a result they do not perform well.
(4) Finally similar as before BPR cannot perform as well as other methods in this experiment as it lacks both the sequential and network aggregation mechanisms.

Table 4: Comparison of *NActSeer* and other baselines when *NActSeer* is trained only on the training graph. Best result is shown in boldface while second best is shown in underlined italics. We can see the event when *NActSeer* is trained only on the training graph it can outperform other baselines very well.

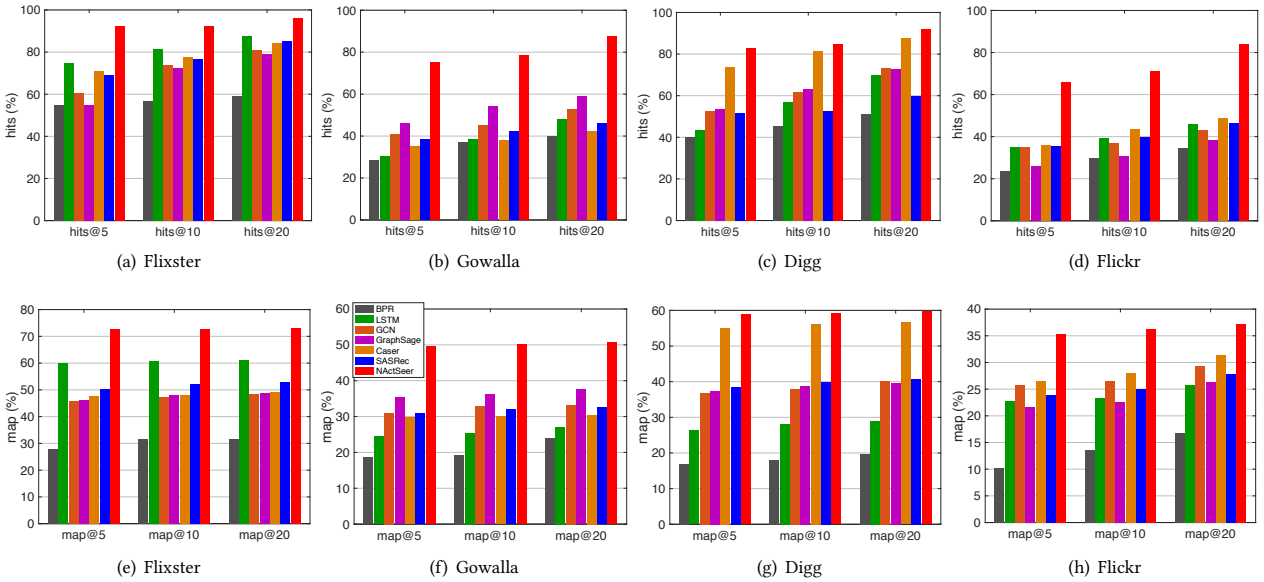
Datasets	Methods	hits@5	hits@10	hits@20
Flickr	BPR	2.365	5.061	9.686
	LSTM	8.311	10.211	11.941
	GCN	12.551	13.831	*16.927*
	GraphSage	*13.676*	*15.672*	16.719
	Caser	13.228	13.968	14.661
	SASRec	12.178	14.255	15.497
	NActSeer (Train graph only)	**16.356**	**18.684**	**20.811**
Gowalla	BPR	27.844	30.708	35.763
	LSTM	38.355	43.328	47.078
	GCN	39.722	45.857	51.078
	GraphSage	41.355	*46.328*	48.911
	Caser	41.646	45.821	48.651
	SASRec	*43.572*	45.962	*51.175*
	NActSeer (Train graph only)	**51.389**	**51.950**	**55.262**

4.4 How Useful Is the Expedited User Embedding Computation?

Recall that in Algorithm 1 we show a faster way of computing the user embedding for a mini-batch. Now we evaluate how does it expedite the training process. The comparison between the execution time of normal and expedited computation is shown in Fig. 5(a). We observe that the expedited computation method achieves up to 8.5% speed-up for each training iteration.

4.5 How Fruitful Is the Action Embedding for Users With No Connection?

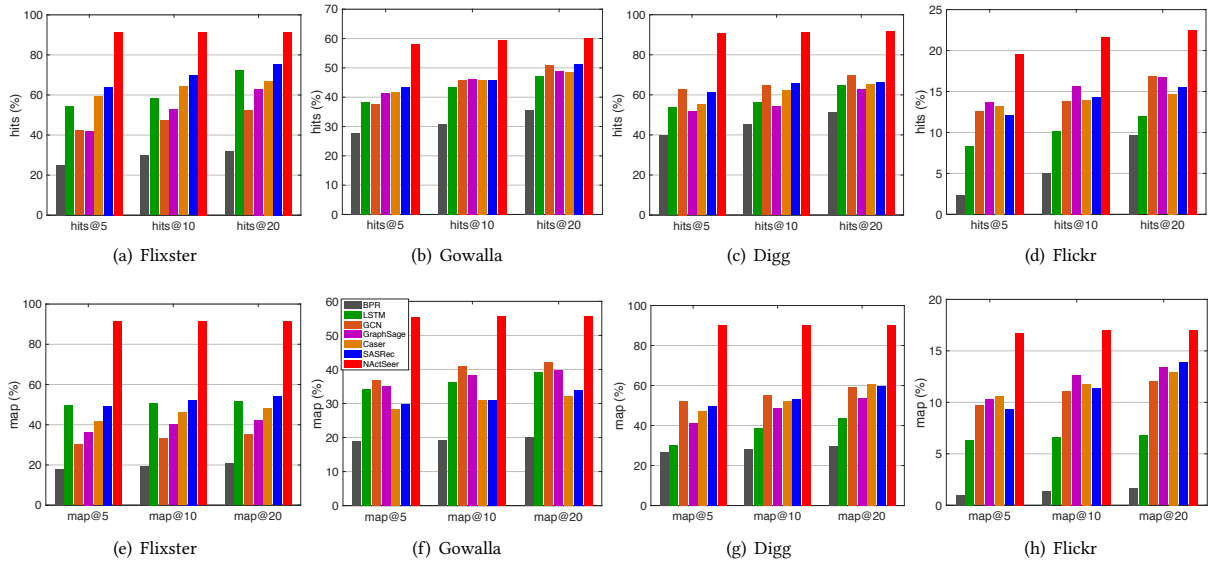
Generally when a new user joins the social network he/she does not have any connections/neighbors. The lack of connectivity impose a challenge as the model only have to rely on action-action correlation to predict user action. As a result, action embedding matrix $\mathbf{X}$ plays a crucial role here as the model only have to rely on this to make prediction for new users.
To evaluate how effective is the learned action embedding we use the experimental setting described in Section 4.3 (i.e. predicting the actions of new users). Only difference is during testing instead of using the entire graph, which includes the connectivity from all the users we use the training graph (i.e. the induced subgraph from the users whose data is used for training). By using the training graph we make sure that new users have no neighbors thereby forcing the models to only leverage the learned action embedding i.e. action-action correlation for prediction. The result is shown in Table 4. We observe that this modified version of *NActSeer* is still able to outperform other models by a very good margin. This shows that the learned action embedding matrix of *NActSeer* is encoding richer vocabulary of features that is helpful to predict the actions of new users who does not have any connections with the existing users.



(a) Flixster (b) Gowalla (c) Digg (d) Flickr

(e) Flixster (f) Gowalla (g) Digg (h) Flickr

Figure 3: Comparison of *NActSeer* with some state-of-the-art and classical methods proposed for related problems. Top and bottom row show the HITS and MAP scores respectively. We can see *NActSeer* outperforms all other methods in terms of HITS and MAP score.



(a) Flixster (b) Gowalla (c) Digg (d) Flickr

(e) Flixster (f) Gowalla (g) Digg (h) Flickr

Figure 4: Comparison of *NActSeer* with the competing methods in case of predicting the actions of new users. We can see *NActSeer* outperforms other methods in both HITS and MAP scores. *NActSeer* outshines other models because 1) new user embedding can immediately be calculated from their actions and 2) receiving rich signal from user's neighbors.

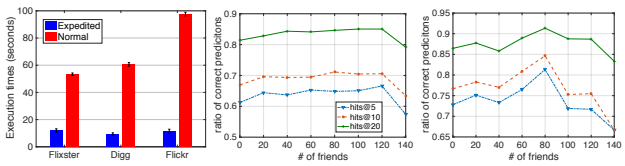
4.6 Ablation Study

As our model contains components such as layer normalization and dropout, we evaluate how our model performs without them. For this we remove the components one at a time and finally both and then compare the results with our default version of *NActSeer*. The result is shown in Table 5 for Gowalla and Flickr dataset. We see that removing layer normalization degrades the overall performance a bit (almost 3% loss in case of map@20 for Gowalla). Removing the dropout layer also decreases performance (~8% reduction). As expected removing both components results in even

more decreased performance (∼ 30% reduction). But even then our model outperforms all competing methods. It is also to be noted that models like SASRec also uses these components but can not perform as well as *NActSeer*. As a result, we can conclude that *NActSeer* provides superior performance even without layer normalization and dropout.

4.7 Does the Number of Friends Have Any Effect on Performance?

To see whether the number of friends has an effect on overall performance we plot the number of correct predictions i.e. the number of users for whom a correct prediction has been made vs number of neighbors. We normalize the correct prediction count by dividing the total number of users that has the corresponding number of neighbors. The result is shown in Fig 5 (b) and (c). We can see that having more neighbors improves the overall performance of the users. This shows *NActSeer* successfully leverage the neighbors activity to achieve better performance. However, this trend has a diminishing return in the end.



(a) Execution Time (b) Flickr (c) Gowalla

Figure 5: (a) Comparison of execution time between expedited and normal user embedding computation for different datasets. We can see that expedited user embedding computation significantly lowers the training time. (b) and (c) Normalized count of correct predictions vs number of neighbors/friends. We observe an increasing trend of performance as the number of friends increases showing that including neighbors actions improves performance. However, the trend has an diminishing return.

4.8 Parameter Sensitivity

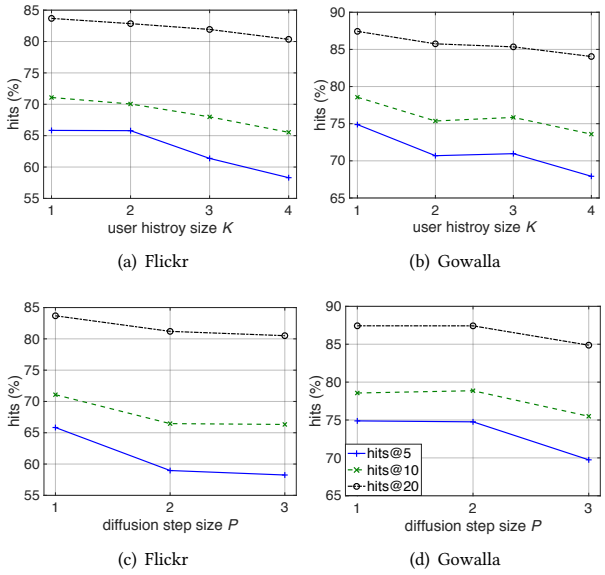
Here we show the sensitivity of *NActSeer* to its two most influential parameters: the size of user's history K and the of size diffusion step P. For this we only change the values of a parameter keeping other to default value. We observe that higher values of K leads to lower performance (Fig. 6(a–b)). This means only the most recently taken action is sufficient for the prediction. This phenomena has also been observed in Kang et al. [19] for recommender system. Next we vary the value of P and show the result in Fig. 6(c–d). We notice that increasing P exhibits stable performance initially however, it has a diminishing return in the end. Additionally it also requires more computational time and power. As a result we set the default setting of both parameters to 1 for the best performance.

4.9 Case Studies

Observing the Effect of Neighbors: Here we present two case studies in two different datasets where neighbor's previous actions

Table 5: Ablation study for Flickr and Gowalla dataset. Removing layer normalization and dropout reduces the performance.

Datasets	Model Variants	HITS			MAP		
		5	10	20	5	10	20
Flickr	Default	65.84	71.08	83.69	35.42	36.2	37.14
	(-) Dropout	48.69	49.87	51.09	15.25	28.39	31.96
	(-) Norm.	41.55	48.23	53.89	21.11	25.32	30.88
	(-) Both	25.37	39.86	61.56	9.76	11.77	13.23
Gowalla	Default	74.88	78.56	87.43	49.47	50.01	50.66
	(-) Dropout	46.99	57.57	71.75	41.54	42.01	46.58
	(-) Norm.	55.15	63.29	69.33	43.07	48.25	48.98
	(-) Both	45.53	53.98	62.32	32.89	34.34	35.29

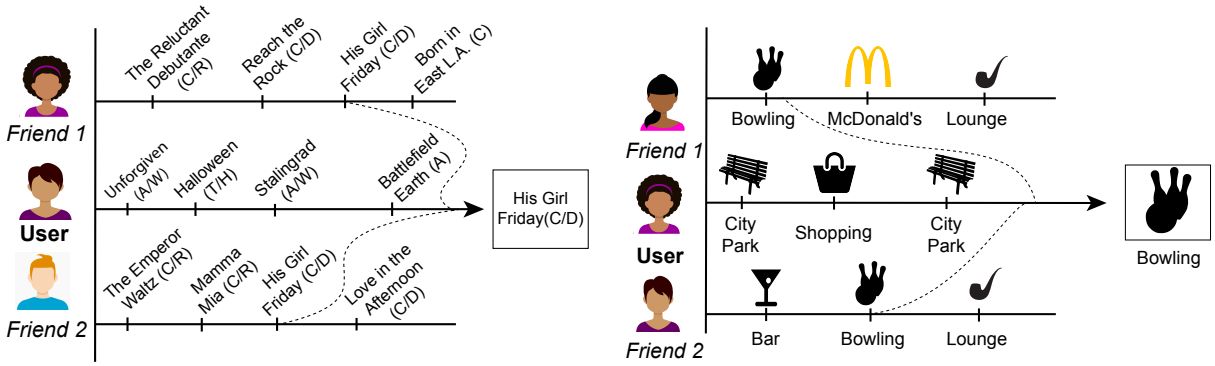


(a) Flickr (b) Gowalla

(c) Flickr (d) Gowalla

Figure 6: Parameter sensitivity study. We show hits@5, hits@10 and hits@20 for different values of K (a and b) and P (c and d) in case of Flickr and Gowalla dataset.

affect the current user's behavior. In Fig. 7 (a) we show one user and her neighbor's recently rated movies with the movie genre shown in parenthesis for Flixster dataset. We observe that the friends of the 'User' usually watch/rate movies of comedy/drama genre while she prefers action/thriller genre. However, influenced by her friends she chooses her next movie "His Girl Friday", which was recently watched by her friends. *NActSeer* is able to correctly predict this instance as it incorporates the neighbor actions. We show a similar case study for Gowalla dataset. We observe that two friends of the 'User' frequently visit a bowling alley however, the 'User' never visited this one. Influenced by her friends she checks-in into the place. This case is also correctly detected by *NActSeer*. In both



(a) Flixster (b) Gowalla

Figure 7: How does incorporating neighbor's actions benefits *NActSeer* in (a) Flixster and (b) Gowalla. In (a) we show one recent user who enjoys watching action/thriller movies while her two neighbors usually watch comedy/drama movies. We show the movie genre in parenthesis (A, C, D, R, T, W represents action, comedy, drama, romantic, thriller and war respectively). Influenced by the neighbors' behavior she picks her next movie to watch as "His Girl Friday", which was previously watched by her friends. *NActSeer* is able to leverage information from the neighbors (shown in dashed line) and make the correct prediction. Similarly in (b) user deviates from her usual check-in behavior and visits a bowling alley where her friend's frequently visit, which is correctly predicted by *NActSeer*. We only show the category of the place for privacy reasons.

cases we anonymize the name and place for privacy and only show the category of the location. Note that in both cases, the correctly predicted action achieved the top ranking (i.e. hits@1).

To summarize from these examples it can be seen that *NActSeer* is successful in capturing the propagation of action preferences between friends. This helps the model to identify break in trends of a user behavior and effectively make the correct prediction.

Friend Recommendation: One of the usabilities of user embedding is that it can be used for friend recommendation. To illustrate this we draw a case study from Flixster dataset. For a correctly classified next action (movie rating) we obtain the users (shown in red) who have rated the same movie in recent times. We plot the 2D embedding space of such users and their immediate neighbors (i.e. first order) in Fig. 8. We observe that although the users (shown in red) are not immediate neighbors to one another (shown in blue) they are relatively closer than their first order neighbors. Due to their similar nature they can be recommended to one another. Therefore we can exploit the user embedding as a metric for friend recommendation.

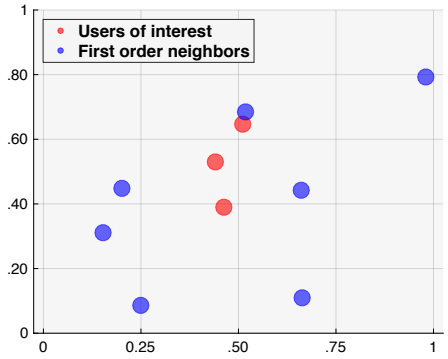


Figure 8: 2D user embedding of several non-neighboring users who rated the same movie in the same time span and their neighbors for Flixster dataset. We observe that despite having no edge between them they position themselves quite close to each other. However, their immediate neighbors (i.e first order) are quite far from them. As a result they can be recommended to one another as friends. We use TSNE for dimension reduction and remaining user embeddings are not shown to avoid clutter.

5 RELATED WORK

In this section we discuss concepts related to user action prediction.

Sequential Recommendation: Sequential recommender systems seek to model item-item transitions to capture the sequential patterns between consecutive items [3, 19, 40]. FPMC [35] is a first-order markov chain model with a matrix factorization term along with an item-item transition matrix. Higher order chain can also be incorporated to capture a deeper user history, however it is often not necessary as discussed in [14]. Recently deep learning based-models are also used to capture the sequential nature of user preferences [3, 15, 19, 40]. For instance, GRU4Rec [15] uses GRU for session based recommendation whereas Caser [40], a CNN based model, treats the embedding matrix of previous items as an image and applies convolutional operation to capture the sequence. Finally SASRec [19] uses self-attention mechanism to predict the next item from user's history. Although the above mentioned model tries to predict the next item given user's history, none of them take into account the connection among the users' as ours.

User Behavior Analysis: Existing works generally analyze the user behavior in the form of cascade analysis. One of the research interests in this topic is to predict the final size of the cascade [8, 24,

44]. These methods uses a wide range of handcrafted features [8, 43] or deep learning methods [17, 24] for the prediction. Another line of works develop diffusion models and methods to maximize influence in a social network [20, 28]. Apart from these, inferring the social ties between users from information cascade has also been analyzed [11, 42] along with studying the impact of external influence in information propagation [30].

Representation Learning in Graphs: The representation learning in graphs can be categorized in two ways. The first line of work tries to generate representation for the nodes in the network. Several node embedding methods have been proposed in [12, 32, 38]. Recently CNN has been generalized to represent graph structure to compute feature for nodes [5, 10, 21]. This type of model is first introduced in [5], which is improved in [10] using fast localized convolution filters. The second line of work tries to represent the entire graph or a portion of it as a fixed length vector [23, 31]. GAM [23] exploits attention mechanism along with reinforcement learning to compute representation for the entire graph, on the other hand Patchy-San [31] uses a CNN architecture. Although these models work on graph structure, none of the models deal with predicting future actions like ours.

6 CONCLUSION

In this paper we present an intriguing problem of user action prediction. To solve this problem we develop a network augmented RNN model *NActSeer* which can take user's neighborhood into account to predict the user's next action. We compare our model against several state-of-the-art methods on several real world datasets of various domains. *NActSeer* clearly outperforms all the other models. The *Network Activity Prediction* problem opens several new directions for research in social network mining. One important extension can be to extend the problem definition so that action can have attributes. Another interesting concept is to extend the *NActSeer* model so that it can handle additional input signals pertinent to action. Finally predicting how user's activities lead to a change in the overall state of the network can be an interesting research direction.

Acknowledgements: This work is supported in part by the National Science Foundation via grants DGE-1545362, IIS-1633363.

REFERENCES

[1] Aris Anagnostopoulos, Ravi Kumar, and Mohammad Mahdian. 2008. Influence and correlation in social networks. In *KDD*. 7–15.
[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).
[3] Alex Beutel et al. 2018. Latent Cross: Making Use of Context in Recurrent Recommender Systems. In *WSDM*. 46–54.
[4] Robert M Bond et al. 2012. A 61-million-person experiment in social influence and political mobilization. *Nature* 489, 7415 (2012), 295.
[5] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral Networks and Locally Connected Networks on Graphs. In *NIPS*.
[6] Meeyoung Cha, Alan Mislove, and Krishna Gummadi. 2009. A Measurement-driven Analysis of Information Propagation in the Flickr Social Network. In *WWW*. 721–730.
[7] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. 2018. Recurrent neural networks for multivariate time series with missing values. *Scientific reports* 8, 1 (2018), 6085.
[8] Justin Cheng, Lada Adamic, P. Dow, Jon Kleinberg, and Jure Leskovec. 2014. Can Cascades Be Predicted?. In *WWW*. 925–936.
[9] Manlio De Domenico, Antonio Lima, Paul Mougel, and Mirco Musolesi. 2013. The anatomy of a scientific rumor. *Scientific reports* 3 (2013), 2980.
[10] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*. 3844–3852.
[11] Manuel Gomez-Rodriguez, Jure Leskovec, and Andreas Krause. 2012. Inferring networks of diffusion and influence. *TKDD* 5, 4 (2012), 1–37.
[12] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *KDD*. 855–864.
[13] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeuroIPS*. 1024–1034.
[14] Ruining He, Wang-Cheng Kang, and Julian McAuley. 2017. Translation-based recommendation. In *RecSys*. 161–169.
[15] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based recommendations with recurrent neural networks. In *ICLR*.
[16] Tad Hogg and Kristina Lerman. 2012. Social dynamics of digg. *EPJ Data Science* 1, 1 (2012), 5.
[17] Mohammad Raihanul Islam, Sathappan Muthiah, Bijaya Adhikari, B. Aditya Prakash, and Naren Ramakrishnan. 2018. DeepDiffuse: Predicting the'Who'and'When'in Cascades. In *ICDM*. 1055–1060.
[18] Mohsen Jamali and Martin Ester. 2010. A matrix factorization technique with trust propagation for recommendation in social networks. In *RecSys*. 135–142.
[19] Wang-Cheng Kang and Julian McAuley. 2018. Self-Attentive Sequential Recommendation. In *ICDM*. 197–206.
[20] D. Kempe, J. Kleinberg, and É. Tardos. 2003. Maximizing the Spread of Influence Through a Social Network. In *KDD*. 137–146.
[21] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.
[22] Ryan Kiros, Richard Zemel, and Ruslan Salakhutdinov. 2014. A multiplicative model for learning distributed text-based attribute representations. In *NIPS*. 2348–2356.
[23] John Boaz Lee, Ryan Rossi, and Xiangnan Kong. 2018. Graph classification using structural attention. In *KDD*. 1666–1674.
[24] Cheng Li, Jiaqi Ma, Xiaoxiao Guo, and Qiaozhu Mei. 2017. DeepCas: An End-to-end Predictor of Information Cascades. In *WWW*. 577–586.
[25] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2018. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *ICLR*.
[26] Yuchen Li, Dongxiang Zhang, and Kian-Lee Tan. 2015. Real-time targeted influence maximization for online advertisements. *VLDB* 8, 10 (2015), 1070–1081.
[27] Yong Liu, Wei Wei, Aixin Sun, and Chunyan Miao. 2014. Exploiting geographical neighborhood characteristics for location recommendation. In *CIKM*. 739–748.
[28] Yasuko Matsubara, Yasushi Sakurai, B. Aditya Prakash, Lei Li, and Christos Faloutsos. 2012. Rise and fall patterns of information diffusion: model and implications. In *KDD*. 6–14.
[29] Andriy Mnih and Geoffrey Hinton. 2007. Three new graphical models for statistical language modelling. In *ICML*. 641–648.
[30] Seth Myers, Chenguang Zhu, and Jure Leskovec. 2012. Information Diffusion and External Influence in Networks. In *KDD*. 33–41.
[31] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *ICML*. 2014–2023.
[32] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online Learning of Social Representations. In *KDD*. 701–710.
[33] Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. 2018. DeepInf: Social Influence Prediction with Deep Learning. In *KDD*. 2110–2119.
[34] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *UAI*. 452–461.
[35] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *WWW*. 811–820.
[36] Pawel Sobkowicz, Michael Kaschesky, and Guillaume Bouchard. 2012. Opinion mining in social media: Modeling, simulating, and forecasting political opinions in the web. *Government Information Quarterly* 29, 4 (2012), 470–479.
[37] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*. 3104–3112.
[38] Jian Tang et al. 2015. Line: Large-scale information network embedding. In *WWW*. 1067–1077.
[39] Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. 2009. Social influence analysis in large-scale networks. In *KDD*. 807–816.
[40] Jiaxi Tang and Ke Wang. 2018. Personalized top-n sequential recommendation via convolutional sequence embedding. In *WSDM*. 565–573.
[41] Daheng Wang, Meng Jiang, Qingkai Zeng, Zachary Eberhart, and Nitesh V. Chawla. 2018. Multi-Type Itemset Embedding for Learning Behavior Success. In *KDD*. 2397–2406.
[42] Senzhang Wang, Xia Hu, Philip S Yu, and Zhoujun Li. 2014. MMRate: inferring multi-aspect diffusion networks with multi-pattern cascades. In *KDD*. 1246–1255.
[43] Lilian Weng, Filippo Menczer, and Yong-Yeol Ahn. 2014. Predicting Successful Memes Using Network and Community Structure. In *ICWSM*. 535–544.
[44] Linyun Yu et al. 2015. From Micro to Macro: Uncovering and Predicting Information Cascading Process with Behavioral Dynamics. In *ICDM*. 559–568.