DeepInf: Modeling Influence Locality in Large Social Networks

Jiezhong Qiu[†], Jian Tang[♯♮], Hao Ma[‡], Yuxiao Dong[‡], Kuansan Wang[‡], and Jie Tang[†]

[†]Department of Computer Science and Technology, Tsinghua University
[‡]Microsoft Research, Redmond
[♯]HEC Montreal, Canada
[♮]Montreal Institute for Learning Algorithms, Canada
qiujz16@mails.tsinghua.edu.cn,jian.tang@hec.ca
{haoma,yuxdong,kuansanw}@microsoft.com,jietang@tsinghua.edu.cn

ABSTRACT

Online communities such as Facebook, Twitter, WeChat, and Weibo have become an indispensable part of our everyday life, where we can easily access the behaviors of our friends and are influenced by their behaviors. Effectively predicting social influence for each user is critical for many applications such as recommendation and online advertising. For example, for a specific user, if some of her friends retweeted a tweet, can we predict whether she will retweet it or not in the future?

Most existing approaches typically design various hand-crafted rules to extract user-specific and network-specific features. However, these approaches heavily rely on the knowledge of domain experts. As a result, they are usually difficult to generalize to different domains. Inspired by the recent success of deep neural networks in a variety of applications, in this paper we design an end-to-end framework for feature representation learning to predict social influence. Specifically, each user is represented with a local sub-network she embedded in. A graph neural network is used to learn the representation of the sub-network, which effectively integrates the user-specific features and network structures. We conduct extensive experiments with large-scale user behaviors in several online communities. Experimental results show that our proposed end-to-end approach significantly outperforms previous feature engineering-based approaches.

CCS CONCEPTS

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

KEYWORDS

Social Influence, Social Networks, Representation Learning, Graph Convolution, Attention

, ,

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

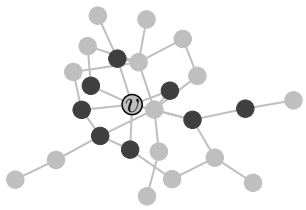


Figure 1: A motivating example. We want to predict the action status of user v, given the observed action status of her near neighbors (we use black and gray to indicate "active" and "inactive", respectively) and the local network she embedded in.

1 INTRODUCTION

Social influence is everywhere around us, not only in our daily physical lives but also in the virtual Web space. The term social influence typically refers to the phenomenon that a person's emotions, opinions, or behaviors are affected by others. With the emergence and rapid proliferation of online social platforms, people have seen the impact of social influence in every field, including but not limited to advertising, innovation adoption, presidential elections, stock markets, and civil unrest. There is little doubt that social influence has become a prevalent, subtle, and complex force that drives our social decisions and activities. Therefore, there is a clear need for methods and frameworks to characterize, understand, and quantify the underlying mechanism and dynamics of social influence.

Indeed, extensive work has been done on social influence prediction in the literature [10, 23, 27]. For example, [10] studied predicting the sizes of cascades with different kinds of hand-crafted features extracted from networks while [23] proposed an end-to-end approach for the problem. All these approaches mainly study predicting the global patterns of social influence, i.e., macro-level social influence prediction. However, in many online applications such as advertising and recommendation, how to effectively predict the social influence for each user is critical, i.e., micro-level social influence prediction.

In this paper, we focus on micro-level social influence prediction. We formulate it as a classification problem based on the definition of influence locality inherited from Zhang et al. [50]. Under this formulation, we want to predict the action status of a user given

the action status of her near neighbors and her local structural information. For example, for a specific user, if some of her friends bought a product, will she buy the product as well? Figure 1 shows a motivating example. In this example, we are interested in the action status of user v, given the action status of her near neighbors and the local network she embedded in.

Traditional social influence locality models consider complicated hand-crafted features which require extensive domain knowledge and are usually hard to generalize to new domains. Inspired by the recent success of deep learning in a variety of applications, which provide end-to-end solutions for automatic feature representation learning, we design an end-to-end approach to discover hidden and predictive signals in social influence automatically. By incorporating recently developed network embedding [17, 31, 35], graph convolution [22], and attention mechanism [44], we expect the end-to-end model can achieve better performance than well-engineered hand-crafted features.

Therefore, after formulating the social influence locality prediction problem, we propose a novel framework called DeepInf to consider both influence dynamics and network structure. To predict the action status of a specific user v, we first sample her near neighbors through random walk with restart. After obtaining a local network as shown in Figure 1, we use graph convolution and graph attention to learn predictive signals. We test the proposed framework on four social networks from different domains — OAG, Digg, Twitter, and Weibo. Experimental results show that DeepInf can significantly improve the prediction performance. We also compare DeepInf with several methods including simple linear models with hand-crafted features and the state-of-the-art graph classification model [29].

Organization The rest of this paper is organized as follows: Section 2 formulates social influence locality problem. Section 3 introduces the proposed framework in detail. In Section 4 and 5, we conduct extensive experiments and case studies. Finally, Section 6 summarizes related work and Section 7 concludes this work.

2 PROBLEM FORMULATION

In this section, we first introduce several necessary definitions and then formulate the problem of predicting social influence locality.

Definition 2.1. **r-neighbors and r-ego network** Let $G = (V, E)$ be a static social network, where V denotes the set of users and $E \subseteq V \times V$ denotes the set of relationships.[1] For a user v, its r-neighbors are defined to be $\Gamma_v^r = \{u : d(u, v) \leq r\}$ where $d(u, v)$ is the shortest distance (in terms of number of hops) between u and v in network G. The r-ego network of user v is the sub-network induced by Γ_v^r, denoted by G_v^r.

Definition 2.2. **Social Action** Users in social network G perform social actions. For a social action, a user u, and a timestamp t, we observe a binary action status, $s_u^t \in \{0, 1\}$, where $s_u^t = 1$ indicates user u has performed this action before time t, and $s_u^t = 0$ indicates that the user has not performed this action yet. Such an action log can be available from many social networks, e.g., the "retweet" action in Twitter and the citation action in academic social networks.

[1] In this work, we consider undirected relationship.

Given the above definitions, we introduce social influence locality, which amounts to a kind of *closed world assumption*: people take action only due to influence from their near neighbors, while other external sources are assumed to be not present.

Problem 1. ***Social Influence Locality[50]*** *Social influence locality models the probability of v's action status condition on her r-ego network G_v^r and the action states of her r-neighbors. More formally, given G_v^r and $S_v^t = \{s_u^t : u \in \Gamma_v^r \setminus \{v\}\}$, social influence locality aims at quantifying the activation probability of v after a given time interval Δt:*

$$Prob\left(s_v^{t+\Delta t} \middle| G_v^r, S_v^t\right).$$

Practically, suppose we have N observations (instances), each observation is a 3-tuple (v, a, t), where v is a user, a is a social action and t is a timestamp. For such a 3-tuple (v, a, t), we also know v's r-ego network $-$ $G_{v_i}^r$, the action status of v's r-neighbors $-$ S_v^t, and v's action status at $t + \Delta t$ $-$ $s_v^{t+\Delta t}$. The influence locality problem can be formulated as a binary classification problem which can be solved by minimizing the following negative log likelihood objective w.r.t model parameters Θ:

$$\mathcal{L}(\Theta) = -\sum_{i=1}^{N} \log\left(P_\Theta\left(s_{v_i}^{t_i+\Delta t} \middle| G_{v_i}^r, S_{v_i}^t\right)\right). \quad (1)$$

Especially, in this work, we assume Δt is sufficiently large ($\Delta t = \infty$), i.e., we want to predict the action status of ego user v at the end of our observation window.

Discussion on Problem Formulation We formulate social influence locality prediction as a graph classification problem. Each observation (v, a, t) is associated with a graph (r-ego network G_v^r), where each vertex has several features, e.g., action status. Given that, we want to predict the label of this graph. In our specific setting, the label is defined to be the action status of the ego user at timestamp $t + \Delta t$, i.e., $s_v^{t+\Delta t}$.

3 MODEL FRAMEWORK

Our goal is to design an end-to-end unified framework to capture the subtle micro-mechanism and dynamics behind social influence locality. In this section, we formally propose DeepInf, as shown in Figure 2, to automatically detect these latent factors. The framework firstly samples fixed-size small networks as proxies for r-ego networks (see Section 3.1), then feed the sampled sub-networks into a graph convolutional network or graph attention network with mini-batch learning (see Section 3.2). Finally, the model output is compared with ground truth to minimize the negative log-likelihood loss.

3.1 Sampling Near Neighbors

Consider a user v and her r-ego network G_v^r, a straightforward way to extract G_v^r from G is to perform Breadth-First-Search (BFS) starting from user v. However, for different ego user, G_v^r's may have different size. Meanwhile, the size (regarding the number of vertices) of r-ego networks can be very large due to the small world phenomenon [45]. To remedy these issues, we sample a fixed-size sub-network from v's r-ego network, instead of directly dealing with the r-ego network itself. A nature choice of the sampling method is to perform random walk with restart (RWR) [39]. Inspired by [2, 40] which indicates that active neighbors are more important

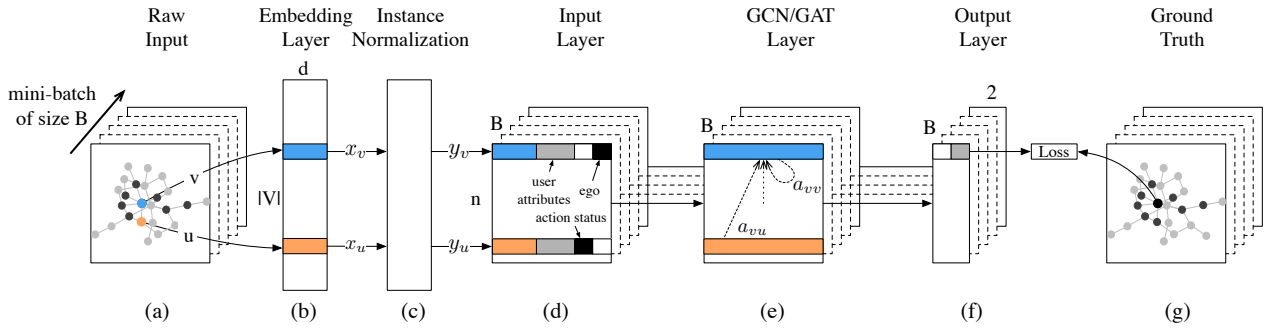


Figure 2: Model Framework of DeepInf. (a) Raw input which consists of a mini-batch of B instances; Each instance is a sub-network comprised of n users who are sampled using random walk with restart as described in Section 3.1. In this example, we keep our eyes on ego user v (marked as blue) and one of her active neighbor u (marked as orange). (b) An embedding layer which map each user to her D-dimensional representation; (c) An instance normalization layer [41]. For each instance, this layer normalizes users' embedding $\boldsymbol{x}_u$'s according to Eq. 3. The output embedding $\boldsymbol{y}_u$'s have zero mean and unit variance within each instance. (d) The formal input layer which concatenates together network embedding, two dummy features (one indicates whether the user is active, the other indicates whether the user is the ego), and other user attributes. (e) A GCN or GAT layer. a_{vv} and a_{vu} indicate the attention coefficients along self-loop (v, v) and edge (v, u), respectively; The value of these attention coefficients can be chosen between Eq. 4 and Eq. 6 according to the choice between GCN and GAT. (f) and (g) Compare model output and ground truth, we get the negative log likelihood loss. In this example, ego user v was finally activated (marked as black).

than inactive neighbors, we start the random walk from either ego user v or one of her active neighbors randomly. Next, the random walk iteratively travels to its neighborhood with the probability that is proportional to their edge weights. Also at each step, it has some probability to return to the starting points (ego user v or one of v's active neighbors). The RWR runs until it successfully collects a fixed number of vertices, denoted by $\bar{\Gamma}_v^r$ with $\left|\bar{\Gamma}_v^r\right| = n$. For simplicity, we rearrange the sampled r-neighbors $\bar{\Gamma}_v^r$ such that the ego user v always ranks first. We then regard the sub-network $\bar{G}_v^r$ induced by $\bar{\Gamma}_v^r$ as a proxy of r-ego network G_v^r, and we denote $\bar{S}_v^t = \{s_u^t : u \in \bar{\Gamma}_v^r \setminus \{v\}\}$ to be the action status of sampled users. Therefore, we re-define the optimization objective in Eq. 1 to be:

$$\mathcal{L}(\Theta) = -\sum_{i=1}^{N} \log\left(P_\Theta\left(s_{v_i}^{t_i+\Delta t}\middle|\bar{G}_{v_i}^r, \bar{S}_{v_i}^t\right)\right). \tag{2}$$

3.2 Neural Network Model

Once we have retrieved $\bar{G}_v^r$ and $\bar{S}_v^t$ for each instance, we want to design an effective neural network model which incorporates both structural properties in $\bar{G}_v^r$ and action status in $\bar{S}_v^t$. The neural network model outputs a hidden representation for the ego user, which can be used to predict her action status — $s_v^{t+\Delta t}$. Our neural network model consists of a network embedding layer, an instance normalization layer, an input layer, several graph convolutional or graph attention layer, and finally an output layer. In this section, we introduce these layers one by one and build the model step by step.

Embedding Layer With the recent emergence of representation learning [4], network embedding technique has been extensively studied to automatically discover and encode network's structural properties into a low-dimensional latent space. More formally, network embedding methods learn an embedding matrix $\boldsymbol{X} \in \mathbb{R}^{D\times|V|}$, with each column corresponding to the representation of a vertex (user) in network G. In our model, the embedding layer maps a user u to her D-dimensional representation $\boldsymbol{x}_u \in \mathbb{R}^D$.

Instance Normalization [41] Instance normalization is a recently proposed technique in image style transfer [41, 42]. In image stylization, this layer normalizes an image such that its pixels have zero mean and unit variance. We adopt this technique in predicting social influence locality. For each user $u \in \bar{\Gamma}_{v_i}^r$, after retrieving her representation $\boldsymbol{x}_u$ from the embedding layer, the instance normalization is given by

$$\boldsymbol{\mu}_d = \frac{1}{n}\sum_{u\in\bar{\Gamma}_v^r} \boldsymbol{x}_{ud},\ \boldsymbol{\sigma}_d^2 = \frac{1}{n}\sum_{u\in\bar{\Gamma}_v^r} (\boldsymbol{x}_{ud} - \boldsymbol{\mu}_d)^2,\ \boldsymbol{y}_{ud} = \frac{\boldsymbol{x}_{ud} - \boldsymbol{\mu}_d}{\sqrt{\boldsymbol{\sigma}_d^2 + \epsilon}}, \tag{3}$$

for each embedding deminsion $d = 1, \cdots, D$. Here $\boldsymbol{\mu}_d$ and $\boldsymbol{\sigma}_d$ are the mean and variance, ϵ is a small number for numerical stability. Intuitively, such normalization can remove instance-specific mean and variance, which encourages the down-stream model to focus on users' relative positions in latent embedding space, rather than their absolution positions.

Input Layer The input layer constructs a feature vector for each user. Besides the normalized low-dimensional representation comes from up-stream instance normalization layer, it also considers two binary variables. One variable indicates users' action status and the other indicates whether the user is the ego user. Furthermore, the input layer is open to any other user attributes, e.g., structural features, demographics information, and other rich attributes such as text and image [24].

GCN [22] Based Network Encoding Graph Convolutional Network (GCN) is a semi-supervised learning algorithm for graph-structured data. The GCN model is built by stacking multiple GCN layers. The input to each GCN layer is a vertex feature matrix, $\boldsymbol{H} \in \mathbb{R}^{n \times F}$, where n is the number of vertices, and F is the number of features. Each row of $\boldsymbol{H}$, denoted by $\boldsymbol{h}_i^\top$, is associated with a vertex. Generally speaking, the essence of GCN layer is a nonlinear transformation which outputs $\boldsymbol{H'} \in \mathbb{R}^{n \times F'}$ as follows:

$$\boldsymbol{H'} = \text{GCN}(\boldsymbol{H}) = g\left(\boldsymbol{A}(G)\boldsymbol{H}\boldsymbol{W}^\top + \boldsymbol{b}\right), \tag{4}$$

where $\boldsymbol{W} \in \mathbb{R}^{F' \times F}, \boldsymbol{b} \in \mathbb{R}^{F'}$ are model parameters, g is a non-linear activation function, $\boldsymbol{A}(G)$ is a $n \times n$ matrix that captures structural information of graph G. GCN instantiates $\boldsymbol{A}(G)$ to be a static matrix closely related to normalized graph Laplaican [11]:

$$\boldsymbol{A}_{\text{GCN}}(G) = \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}, \tag{5}$$

where $\boldsymbol{A}$ is the adjacency matrix of G [2], and $\boldsymbol{D} = \text{diag}(\boldsymbol{A}\boldsymbol{1})$.

Multi-head Graph Attention [44] Graph Attention (GAT) is a recent proposed technique which introduce attention mechanism into GCN. GAT defines matrix $\boldsymbol{A}_{\text{GAT}}(G) = [a_{ij}]_{n \times n}$ through a self-attention mechanism. More formally, an attention coefficient e_{ij} is firstly computed by an attention function $\text{attn} : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \to \mathbb{R}$, which measure the importance of vertex j to vertex i:

$$e_{ij} = \text{attn}\left(\boldsymbol{W}\,\boldsymbol{h}_i, \boldsymbol{W}\,\boldsymbol{h}_j\right).$$

Different from traditional self-attention mechanism where the attention coefficient between each pair of instances will be computed, GAT only evaluate e_{ij} for $(i,j) \in E(\tilde{G}_v^r)$ or $i = j$, i.e., (i,j) is either an edge or a self-loop, in order to better capture and leverage the graph structural information. After that, to make coefficients comparable among vertices, a softmax function is adopted to normalize attention coefficients:

$$a_{ij} = \text{softmax}(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \tilde{\Gamma}_i^1} \exp(e_{ik})}.$$

Following Velickovic et al. [44], the attention function is instantiated with a dot product and a LeakyReLU [26, 47] nonlinearity. For an edge or a self-loop (i,j), The dot product is performed between parameter $\boldsymbol{c}$ and the concatenation of the feature vectors of the two end points $-\ \boldsymbol{W}\boldsymbol{h}_i$ and $\boldsymbol{W}\boldsymbol{h}_j$, i.e., $e_{ij} = \text{LeakyReLU}\left(\boldsymbol{c}^\top\left[\boldsymbol{W}\boldsymbol{h}_i || \boldsymbol{W}\boldsymbol{h}_j\right]\right)$, where the LeakyReLU has negative slop 0.2. To sum up, the normalized attention coefficients can be expressed as:

$$a_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\boldsymbol{c}^\top\left[\boldsymbol{W}\,\boldsymbol{h}_i || \boldsymbol{W}\,\boldsymbol{h}_j\right]\right)\right)}{\sum_{k \in \tilde{\Gamma}_i^1} \exp\left(\text{LeakyReLU}\left(\boldsymbol{c}^\top\left[\boldsymbol{W}\,\boldsymbol{h}_i || \boldsymbol{W}\,\boldsymbol{h}_k\right]\right)\right)}, \tag{6}$$

where $||$ indicates vector concatenation.

Once obtained the normalized attention coefficients, i.e., a_{ij}'s, we can plugin $\boldsymbol{A}_{\text{GAT}}(G) = [a_{ij}]_{n \times n}$ into Eq. 4. This completes the definition of a single-head graph attention. In addition, we apply multi-head graph attention as suggested by Velickovic et al. [44] and Vaswani et al. [43]. The multi-head attention mechanism performs K independent single attention in parallel, i.e., we have K independent parameters $\boldsymbol{W}_1, \cdots, \boldsymbol{W}_K$ and attention matrix $\boldsymbol{A}_1, \cdots, \boldsymbol{A}_K$. Multi-head attention aggregate the output of K single attention together through an aggregation function:

$$\boldsymbol{H'} = g\left(\text{Aggregate}\left(\boldsymbol{A}_1(G)\boldsymbol{H}\boldsymbol{W}_1^\top, \cdots, \boldsymbol{A}_K(G)\boldsymbol{H}\boldsymbol{W}_K^\top\right) + \boldsymbol{b}\right). \tag{7}$$

[2] GCN applies self-loop trick on graph G by adding self-loos on each vertex, i.e., $A \leftarrow A + I$

Table 1: Summary of datasets. $|V|$ and $|E|$ indicates the number of vertices and edges in graph $G = (V, E)$, while N is the number of social influence locality instances (observations) as described in Section 2.

	OAG	Digg	Twitter	Weibo
$\|V\|$	953,675	279,630	456,626	1,776,950
$\|E\|$	4,151,463	1,548,126	12,508,413	308,489,739
N	499,848	24,428	499,160	779,164

Following [44], we concatenate the outputs of each single-head attention to aggregate them except an average operator for the last layer.

Output Layer and Loss Function This layer output a 2-dimension representation for each user, we compare the representation of the ego user with ground truth, and then optimize the log-likelihood loss as described in Eq. 2.

Mini-batch Learning When sampling from r-ego network, we force the sampled sub-networks have fixed size n. Benefiting from such homogeneity, we can apply mini-batch learning. In each iteration, we sample B observations to be a mini-batch. Then we optimize our model w.r.t. the sampled B observations. Such method runs much faster than full-batch learning and still introduces enough noise during optimization.

4 EXPERIMENT SETUP

We set up our experiments with large-scale real-world data sets to quantitatively evaluate the proposed DeepInf framework.

4.1 Data Sets

Our experiments are conducted on four social networks from difference domains —- OAG, Digg, Twitter, and Weibo. Table 1 lists statistics of the four data sets.

OAG[3] OAG (Open Academic Graph) data set is generated by linking two large academic graphs: Microsoft Academic Graph [34] and AMiner [38]. We choose 20 popular conferences from data mining, information retrial, machine learning, natural language processing, computer vision, and database research communities [4]. The social network is defined to be the co-author network, and the social action is defined to be citation behaviors — a researcher cites a paper from the above conferences. We are interested in how one's citation behaviors are influenced by her collaborators.

Digg [20] Digg is a news aggregator which allows people to vote web content, a.k.a, story, up or down. The data set contains data about stories promoted to Digg's front page over a period of a month in 2009. For each story, it contains the list of all Digg users who have voted for the story up to the time of data collection and the time stamp of each vote. The voters' friendship links are also retrieved.

[3] www.openacademic.ai/oag/

[4] KDD, WWW, ICDM, SDM, WSDM, CIKM, SIGIR, NIPS, ICML, AAAI, IJCAI, ACL, EMNLP, CVPR, ICCV, ECCV, MM, SIGMOD, ICDE, and VLDB.

Twitter [13] The Twitter data set was built after monitoring the spreading processes on Twitter before, during and after the announcement of the discovery of a new particle with the features of the elusive Higgs boson on Jul. 4th, 2012. The social network is defined to be the Twitter friendship network, and the social action is defined to be whether a user retweets "higgs" related tweets.

Weibo [50, 51] Weibo is the most popular Chinese microblogging service. The dataset is from [50] and can be downloaded here.[5] The complete dataset contains the directed following networks and tweets (posting logs) of 1,776,950 users between Sep. 28th, 2012 and Oct. 29th, 2012. The social action is defined as retweeting behaviors in Weibo — a user forwards (retweets) a posts (tweets).

Data Preparation We process the above four data sets following the practice in existing work [50, 51]. More concretely, for a user v who was influenced at some timestamp t, we generate a positive instance. For each neighbor of the influenced user, if she was never observed to be active in our observation window, we create a negative instance. Our target is to distinguish positive observations from negative ones. However, the achieved data sets are facing data imbalance problems in the two respects. The first problem comes from the number of active neighbors. As observed by Zhang et al. [51], structural features become significantly correlated with social influence locality when the ego user has a relatively large number of active neighbors. However, the number of active neighbors is imbalanced in most social influence data sets. For example, in Weibo, around 80% observations only have one active neighbor and the instances with the number of active neighbors ≥ 3 only occupies 8.57%. Therefore, when we train our model on such imbalanced data sets, the model will be dominated by observations with few active neighbors. To deal with the imbalance issue and show the superiority of our model in capturing local structural information, we filter out observations with few active neighbors. Especially, in each data set, we only consider instances where ego users have ≥ 3 active neighbors. The second problem comes from label imbalance. For example, in Weibo data sets, the ratio of negative instances to positive instances is about 300:1. To address this issue, we sample a more balanced data sets with ratio between negative and positive to be 3:1.

4.2 Evaluation Metrics

To evaluate our framework quantitatively, we use the following performance metrics:

Prediction Performance We evaluate the prediction performance of DeepInf in terms of Area Under Curve (AUC) [7], Precision (Prec.), Recall (Rec.), and F1-Measure (F1).

Parameter Sensitivity We analyze several hyper-parameters in our model and test how different hyper-parameter choices can influence prediction performance.

Case Study We use case studies to further demonstrate and explain the effectiveness of our proposed framework.

[5] http://aminer.org/Influencelocality

Table 2: List of features used in this work.

Name	Description
Vertex	Coreness [3].
	Pagerank [30].
	Hub score and authority score [8].
	Eigenvector Centrality [5].
	Clustering Coefficient [46].
	Rarity (reciprocal of ego user's degree) [1].
	Network embedding (DeepWalk [31], 64-dim).
Ego	The number/ratio of active neighbors [2].
	Density of subgnetwork induced by active neighbors [40].
	#Connected components formed by active neighbors [40].

4.3 Comparison Methods

We compare DeepInf with several baselines.

Logistic Regression (LR) We use logistic regression (LR) to train a classification model. The model considers two categories of features: (1) vertex features for the ego-user; (2) hand-crafted ego-network features. As listed in Table 2, the first category of features includes page rank score, clustering coefficient, etc. And the second category includes the number/ratio of active neighbors, structural diversity, and the number of connected components formed by active neighbors.

Support Vector Machine (SVM) We also use support vector machine (SVM) with linear kernel as the classification model. The model use the the same features as logistic regression (LR).

PSCN [29] As we model social influence locality prediction as a graph classification problem, we compare our framework with the state-of-the-art graph classification models, PSCN [29]. For each graph, PSCN selects w vertices according to a user-defined ranking function, e.g., degree and betweenness centrality . Then for each selected vertex, it assembles its top k near neighbors according to breadth-first search order. For each graph, The above process constructs a vertex sequence of length $w \times k$ with F channels, where F is the number of features for each vertex. Finally, PSCN applies 1-dimensional convolutional layers on it.

In our experiments, we find that the recommended betweenness centrality ranking function does not work well in predicting social influence locality. We turn to use breadth-first search order starting from the ego user as the ranking function. When BFS order is not unique, we break ties by ranking active users first. We select $w = 16$ and $k = 5$ by validation, and then apply two 1-dimensional convolutional layers. The first conv layer has 16 output channels, a stride of 5, and a kernel size of 5. The second conv layer has 8 output channels, a stride of 1, and a kernel size of 1. The outputs of the second layer are then fed into a fully-connected layer to predict labels.

DeepInf and its Variants We implement two variants of DeepInf, denoted by DeepInf-GCN and DeepInf-GAT, respectively. **DeepInf-GCN** uses graph convolutional layer as building blocks of our framework, i.e., setting $A(G) = D^{-1/2}AD^{-1/2}$ in Eq. 4. **DeepInf-GAT** uses graph attention as showed in Eq. 6. However, both DeepInf and PSCN accept vertex-level features only. Due to this limitation,

Table 3: Prediction performance of different methods on the four data sets (%).

Data	Model	AUC	Prec.	Rec.	F1
OAG	LR	65.55	32.26	69.97	44.16
	SVM	65.48	32.17	**69.82**	44.04
	PSCN	67.70	36.24	60.46	45.32
	DeepInf-GAT	**70.59**	**38.93**	61.29	**47.61**
Digg	LR	84.72	56.78	73.12	63.92
	SVM	86.01	63.42	67.34	65.32
	PSCN	83.96	62.16	67.34	64.65
	DeepInf-GAT	**88.97**	**68.80**	**73.79**	**71.21**
Twitter	LR	78.07	45.86	**69.81**	55.36
	SVM	79.42	49.12	67.31	56.79
	PSCN	79.40	48.43	68.06	56.59
	DeepInf-GAT	**80.01**	**49.39**	67.47	**57.03**
Weibo	LR	77.10	42.34	72.88	53.56
	SVM	77.11	43.27	70.79	53.71
	PSCN	79.54	44.89	73.48	55.73
	DeepInf-GAT	**82.75**	**48.86**	**74.13**	**58.90**

Table 4: Relative gain of DeepInf-GAT in terms of AUC (%).

Method	OAG	Digg	Twitter	Weibo
LR	65.66	84.72	78.07	77.10
SVM	65.48	86.01	79.42	77.11
PSCN	67.70	83.96	79.40	79.54
DeepInf-GAT	70.59	88.97	80.01	82.75
Gain	4.27%	3.44%	0.74%	4.04%

we do not use the ego-network features in these two models. Instead, we want that they can discover hidden and predictive signals automatically.

Hyper-parameter Setting & Implementation Details As for our framework, DeepInf, we first perform random walk with a restart probability 0.8, and the size of sampled sub-network is set to be 50. For the embedding layer, a 64-dimension network embedding is pre-trained using DeepWalk [31]. Then we choose to use a three-layer GCN or GAT structure for DeepInf, both the first and second GCN/GAT layers contain 128 hidden units, while the third layer (output layer) contains 2 hidden units. Especially, for DeepInf with multi-head graph attention, both the first and second layer consists of $K = 8$ attention heads with each computing 16 hidden units (for a total of $8 \times 16 = 128$ hidden units). For detailed model configuration, we adopt exponential linear units (ELU) [12] as nonlinearity (function g in Eq. 4). All the parameters are initialized with Glorot initialization [15] and trained using the Adagrad [14] optimizer with learning rate 0.01, weight decay $5e^{-4}$, and dropout rate 0.2. We use 75%, 12.5%, 12.5% instances for training, validation and test, respectively; the mini-batch size is set to be 1024 across all data sets.

5 EXPERIMENTAL RESULTS

We compare the prediction performance of all methods across the four data sets in Table 3 and list the relative performance gain in Table 4, where the gain is over the closest baselines. In addition, we compare the variants of DeepInf and list the results in Table 5. We have several interesting observations and insights.

Table 5: Prediction performance of variants of DeepInf (%).

Data	Model	AUC	Prec.	Rec.	F1
OAG	DeepInf-GCN	63.89	30.44	**73.06**	42.97
	DeepInf-GAT	**70.59**	**38.93**	61.29	**47.61**
Digg	DeepInf-GCN	75.32	50.42	56.18	53.15
	DeepInf-GAT	**88.97**	**68.80**	**73.79**	**71.21**
Twitter	DeepInf-GCN	76.53	45.00	66.61	53.71
	DeepInf-GAT	**80.01**	**49.39**	**67.47**	**57.03**
Weibo	DeepInf-GCN	75.41	41.95	68.10	51.92
	DeepInf-GAT	**82.75**	**48.86**	**74.13**	**58.90**

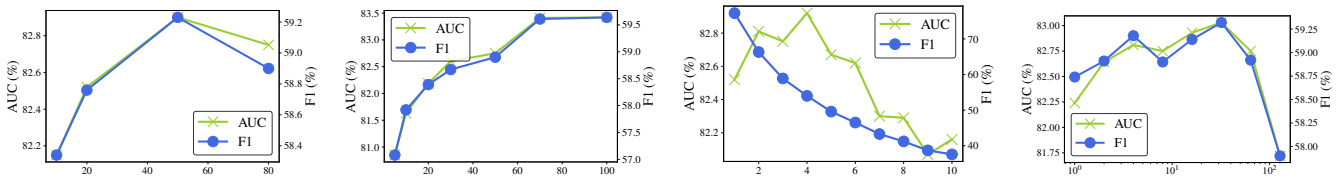
Table 6: Prediction performance of DeepInf-GAT (%) with/without vertex features as introduced in Table 2.

Data	Features	AUC	Prec.	Rec.	F1
OAG	×	67.97	34.45	**66.94**	45.49
	√	**70.59**	**38.93**	61.29	**47.61**
Digg	×	88.93	66.47	**74.06**	70.06
	√	**88.97**	**68.80**	73.79	**71.21**
Twitter	×	78.61	46.77	**68.29**	55.52
	√	**80.01**	**49.39**	67.47	**57.03**
Weibo	×	82.10	47.01	**76.71**	58.29
	√	**82.75**	**48.86**	74.13	**58.90**

(1) As shown in Figure 3, DeepInf-GAT achieves significantly better performance over baselines methods in terms of both AUC and F1, demonstrating the effectiveness of our proposed framework. In OAG, DeepInf-GAT discovers the hidden mechanism and dynamics of social influence locality, giving us 4.72% relative performance gain w.r.t. AUC.

(2) For PSCN, it selects a subset of vertices according to a user-defined ranking function. As mentioned in Section 4, instead of using betweenness centrality, we propose to use BFS order-based ranking function. Such ranking function can be regarded as a special graph attention mechanism where the ego user pays much more attention to her active neighbors! PSCN achieves comparable results in the Twitter data set. While in other three data sets, DeepInf-GAT still outperforms PSCN significantly (up to 4.27% relative gain regarding AUC).

(3) An interesting observation is the inferiority of DeepInf-GCN, as shown in Table 5. Previously, we have seen the success of GCN in may label classification tasks [22]. However, in this application, DeepInf-GCN achieves the worst performance over all comparison methods. We attribute its inferiority to the homophily assumption of GCN — similar vertices are more likely to link with each other than dissimilar ones. Under such assumption, for a specific vertex, GCN computes its hidden representation by taking an average over its neighbors' representations. However, in our application, the homophily assumption may not be true. By averaging over neighbors, GCN may mix predictive signals with noise. On the other hand, as pointed out by [2, 40], active neighbors are more important than inactive neighbors, which also encourages us to use graph attention which treats neighbors differently.



(a) Return Probability of RWR (b) Size of Sampled Networks (c) Negative Positive Ratio (d) Number of Heads

Figure 3: Parameter analysis. (a) Return probability of random walk with restart. (b) Size of sampled networks. (c) Negative positive ratio. (d) The number of heads used for multi-head attention.

(4) In experiments shown in Table 3, 4, and 5, we still rely on several vertex features, such as page rank score and clustering coefficient, which are commonly used in network mining tasks. However, we want to avoid using any hand-crafted features and make DeepInf a "pure" end-to-end learning framework. Quite surprisingly, we can still achieve comparable performance (as showed in Table 6), even we do not consider any hand-crafted features except the pre-trained network embedding.

5.1 Parameter Analysis

In this section, we investigate how the prediction performance varies with the hyper-parameters in sampling near neighbors and the neural network model. We conduct all the parameter analyses on the Weibo data set.

Return Probability of Random Walk with Restart When sampling near neighbors, the return probability of random walk with restart (RWR) controls the "shape" of the sampled r-ego network. Figure 3(a) shows the prediction performance (in terms of AUC and F1) by varying the return probability from 10% to 80%. As the increasing of return probability, the prediction performance first increases and then slightly decreases, with the best performance achieved at around 50%.

Size of Sampled Networks Another parameter that controls the sampled r-ego network is the size of sampled networks. Figure 3(b) shows the prediction performance (in terms of AUC and F1) by varying the size from 5 to 100. We can observe a slow increase of prediction performance when we sample more near neighbors.

Negative Positive Ratio As we mentioned in Section. 5, the positive and negative observations are imbalanced in our data sets. To investigate how such imbalance influence the prediction performance, we vary the ratio between negative and positive instances from 1 to 10 , and show the performance in Figure 3(c). Interestingly, we observe a decreasing trend w.r.t. the F1 measure, while the AUC score stays stable.

5.2 Discussion on GAT and Case Study

#Head for Multi-head Attention Another hyper-parameter we analyze is the number of heads used for multi-head attention. For fair comparison, we fixed the number of total hidden units to be 128. We vary the number of heads to be 1, 2, 4, 8, 16, 32, 64, 128, i.e., each head has 128, 64, 32, 16, 8, 4, 2, 1 hidden units, respectively. We see

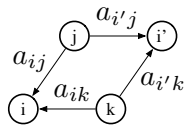


Figure 4: Illustration of order-preserving property of GAT.

that DeepInf benefits from the multi-head mechanism. However, as the decreasing of the number of hidden units correspond to each head, the prediction performance decreases.

Besides the concatenation-based attention used in GAT (Eq. 6), we also try other popular attention mechanisms, e.g., the dot product attention or the bilinear attention as summarized in [25]. However, those attention mechanisms do not perform as well as the concatenation-based one. In this section, we introduce the order-preserving property of GAT [44]. Based on the property, we attempt to explain the effectiveness of DeepInf-GAT through case studies.

Observation 1. ***Order-preserving of Graph Attention*** *Suppose (i,j), (i,k), (i',j) and (i',k) are either edges or self-loops, and a_{ij}, a_{ik}, $a_{i'j}$, $a_{i'k}$ are the attention coefficients associated with them (as shown in Figure 4). If $a_{ij} > a_{ik}$ then $a_{i'j} > a_{i'k}$.*

Proof. As introduced in Eq. 6, the graph attention coefficient for edge (or self-loop) (i,j) is defined as $a_{ij} = \text{softmax}(e_{ij})$, where

$$e_{ij} = \text{LeakyReLU}\left(\boldsymbol{c}^\top \left[\boldsymbol{W}\boldsymbol{h}_i || \boldsymbol{W}\boldsymbol{h}_j\right]\right).$$

If we rewrite $\boldsymbol{c}^\top = \left[\boldsymbol{p}^\top \quad \boldsymbol{q}^\top\right]$, we have

$$e_{ij} = \text{LeakyReLU}\left(\boldsymbol{p}^\top \boldsymbol{W}\boldsymbol{h}_i + \boldsymbol{q}^\top \boldsymbol{W}\boldsymbol{h}_j\right).$$

Due to the strict monotonicity of softmax and LeakyReLU, $a_{ij} > a_{ik}$ implies $\boldsymbol{q}^\top \boldsymbol{W}\boldsymbol{h}_j > \boldsymbol{q}^\top \boldsymbol{W}\boldsymbol{h}_k$. Apply the strict monotonicity of LeakyReLU and softmax again, we get $a_{i'j} > a_{i'k}$. □

The above observation shows the following fact — although each vertex only pay attention to its neighbors in GAT (local attention), the attention coefficients have a global ranking, which is determined by $\boldsymbol{q}^\top \boldsymbol{W}\boldsymbol{h}_j$. Thus we can define a score function $\text{score}(j) = \boldsymbol{q}^\top \boldsymbol{W}\boldsymbol{h}_j$. Then each vertex pays attention to its neighbors according to this score function — a higher score function value indicates a higher attention coefficient. Thus, plotting the value of the scoring function can illustrate where are the "popular areas" or "important areas" of the network. Furthermore, multi-head attention provides a multi-view mechanism — for K heads, we have K score functions, $\text{score}_k(j) = \boldsymbol{q}_k^\top \boldsymbol{W}_k \boldsymbol{h}_j$, $k = 1, \cdots, K$, highlighting

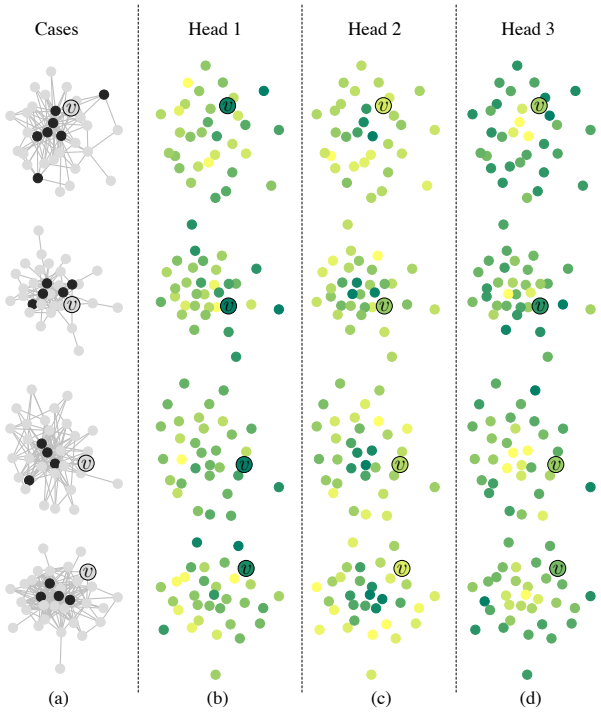


Figure 5: Case study. How different graph attention heads highlight different areas of the network. (a) Four selected cases from Digg data set. Active and inactive users are marked as black and gray, respectively. User v is the ego-user that we are interested in. (b)(c)(d) Three representative attention heads.

different areas of the network. To better illustrate this mechanism, we perform a few case studies. As shown in Figure 5, we choose four instances from Digg data sets (each row corresponding to one instance) and select three representative attention heads from the first GAT layer. Quite interestingly, we can observe significant heterogeneity across multiple attention heads. For example, as shown in Figure 5, the first attention head tend to focus on the ego-user, while the second and the third highlight active users and inactive users, respectively. However, this property does not hold for other attention mechanisms. Due to the page limit, we do not discuss them here.

6 RELATED WORK

Our study is closely related to a large body of literature on social influence analysis [9] and graph representation learning [19].

Social Influence Analysis Most existing work has focused on social influence modeled as a macro-social process (a.k.a., cascade), with a few that have explored the alternative micro-level mechanism that considers the locality of social influence in practice. At the macro-level, researchers are interested in global patterns of social influence. Such global patterns includes various respects of a cascade and their correlation with the final cascade size, e.g., the rise-and-fall patterns [27], external influence sources [28], and conformity phenomenon [37]. Recently, there have been efforts to detect those global patterns automatically using deep learning, e.g., the DeepCas model [23] which formulate cascade prediction as a sequence problem and solve it with Recurrent Neural Network.

Another line of studies focuses on the micro-level mechanism in social influence where each user is only influenced by her near neighbors. Examples of such work include pairwise influence [16, 32], topic-level influence [36], and structural diversity [40]. Such micro-level models act as fundamental building blocks of many real-world problems and applications. For example, in the influence maximization problem [21], both IC mode (independent cascade) and LT model (Linear threshold) assume a pairwise influence model; Another example is a large-scale field experiment by Facebook Bond et al. [6] during the 2010 US congressional elections, the results showed how online social influence changes offline voting behavior.

Graph Representation Learning Representation learning [4] has been a hot topic in research communities. In the context of graph mining, there have been many efforts to graph representation learning. One line of studies focus on vertex (node) embedding, i.e., to learn a low-dimensional latent factors for each vertex. Examples include DeepWalk [31], LINE [35], node2vec [17], etc. Another line of studies pay attention to representation of graphs, i.e., to learn latent representations of sub-structures for graphs, including, graph kernel [33], deep graph kernel [48], and state-of-the-art method PSCN [29]. Recently, there have been several attempts to incorporate semi-supervised information into graph representation learning. Typical examples include Planetoid [49], GCN [22], GraphSAGE [18], and the state-of-the-art model GAT [44].

7 CONCLUSION

In this work, we study the social influence locality problem. We formulate this problem from a deep learning perspective and propose a graph-based learning framework which incorporates recently developed network embedding, graph convolution, and self-attention techniques. We test the proposed framework on four social networks — OAG, Digg, Twitter, and Weibo. Our extensive experimental analysis shows the proposed framework can achieve a better performance in predicting social influence locality among baseline methods which leverage rich hand-craft features. This work explores the possibility of feature learning instead of feature engineering in social influence analysis and attempts to explain the dynamics behind social influence.

Future Work The idea behind our proposed DeepInf can be extended to a lot of network mining tasks. Our DeepInf can effectively and efficiently summarize a local area in a network. Such summarized representation can then be feed into various down-stream applications, e.g., link prediction, similarity search, network alignment, etc. We would like to explore this promising direction in the future. Another exciting direction is the sampling of near neighbors. In this work, we perform naive random walk with restart without taking any vertex features into consideration. Also, the sampling procedure is loosely-coupled with the neural network model. It is existing to combine sampling and learning together using reinforcement learning techniques.

Acknowledgements. We thank Linjun Zhou, Yutao Zhang, and Jing Zhang for their comments. Jiezhong Qiu and Jie Tang are supported by NSFC 61561130160.

REFERENCES

[1] Lada A Adamic and Eytan Adar. 2003. Friends and neighbors on the web. *Social networks* 25, 3 (2003), 211–230.

[2] Lars Backstrom, Dan Huttenlocher, Jon Kleinberg, and Xiangyang Lan. 2006. Group formation in large social networks: membership, growth, and evolution. In *KDD '06*. ACM, 44–54.

[3] Vladimir Batagelj and Matjaz Zaversnik. 2003. An O(m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049* (2003).

[4] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35, 8 (2013), 1798–1828.

[5] Phillip Bonacich. 1987. Power and centrality: A family of measures. *American journal of sociology* 92, 5 (1987), 1170–1182.

[6] Robert M Bond, Christopher J Fariss, Jason J Jones, Adam DI Kramer, Cameron Marlow, Jaime E Settle, and James H Fowler. 2012. A 61-million-person experiment in social influence and political mobilization. *Nature* 489, 7415 (2012), 295.

[7] Chris Buckley and Ellen M Voorhees. 2004. Retrieval evaluation with incomplete information. In *SIGIR '04*. ACM, 25–32.

[8] Soumen Chakrabati, B Dom, D Gibson, J Kleinberg, S Kumar, P Raghavan, S Rajagopalan, and A Tomkins. 1999. Mining the link structure of the World Wide Web. *IEEE Computer* 32, 8 (1999), 60–67.

[9] Wei Chen, Laks VS Lakshmanan, and Carlos Castillo. 2013. Information and influence propagation in social networks. *Synthesis Lectures on Data Management* 5, 4 (2013), 1–177.

[10] Justin Cheng, Lada Adamic, P Alex Dow, Jon Michael Kleinberg, and Jure Leskovec. 2014. Can cascades be predicted?. In *WWW '14*. ACM, 925–936.

[11] Fan RK Chung. 1997. *Spectral graph theory*. Number 92. American Mathematical Soc.

[12] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289* (2015).

[13] Manlio De Domenico, Antonio Lima, Paul Mougel, and Mirco Musolesi. 2013. The anatomy of a scientific rumor. *Scientific reports* 3 (2013), 2980.

[14] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR* 12, Jul (2011), 2121–2159.

[15] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS '10*. 249–256.

[16] Amit Goyal, Francesco Bonchi, and Laks VS Lakshmanan. 2010. Learning influence probabilities in social networks. In *WSDM '10*. ACM, 241–250.

[17] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD '16*. ACM, 855–864.

[18] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS '17*. 1025–1035.

[19] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation Learning on Graphs: Methods and Applications. *arXiv preprint arXiv:1709.05584* (2017).

[20] Tad Hogg and Kristina Lerman. 2012. Social dynamics of digg. *EPJ Data Science* 1, 1 (2012), 5.

[21] David Kempe, Jon Kleinberg, and Éva Tardos. 2003. Maximizing the spread of influence through a social network. In *KDD '03*. 137–146.

[22] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *ICLR '16* (2016).

[23] Cheng Li, Jiaqi Ma, Xiaoxiao Guo, and Qiaozhu Mei. 2017. DeepCas: An end-to-end predictor of information cascades. In *WWW '17*. 577–586.

[24] Huijie Lin, Jia Jia, Jiezhong Qiu, Yongfeng Zhang, Guangyao Shen, Lexing Xie, Jie Tang, Ling Feng, and Tat-Seng Chua. 2017. Detecting stress based on social interactions in social networks. *TKDE* 29, 9 (2017), 1820–1833.

[25] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *EMNLP '15* (2015).

[26] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. 2013. Rectifier nonlinearities improve neural network acoustic models. In *ICML '13*. 3.

[27] 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 '12*. 6–14.

[28] Seth A Myers, Chenguang Zhu, and Jure Leskovec. 2012. Information diffusion and external influence in networks. In *KDD '12*. ACM, 33–41.

[29] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *ICML '16*. 2014–2023.

[30] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report. Stanford InfoLab.

[31] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD '14*. ACM, 701–710.

[32] Kazumi Saito, Ryohei Nakano, and Masahiro Kimura. 2008. Prediction of information diffusion probabilities for independent cascade model. In *KES '08*. Springer, 67–75.

[33] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. 2009. Efficient graphlet kernels for large graph comparison. In *AISTATS' 09*. 488–495.

[34] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-june Paul Hsu, and Kuansan Wang. 2015. An overview of microsoft academic service (mas) and applications. In *WWW '15*. ACM, 243–246.

[35] 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.

[36] Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. 2009. Social influence analysis in large-scale networks. In *KDD '09*. ACM, 807–816.

[37] Jie Tang, Sen Wu, and Jimeng Sun. 2013. Confluence: Conformity influence in large social networks. In *KDD '13*. ACM, 347–355.

[38] 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.

[39] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. 2006. Fast Random Walk with Restart and Its Applications. In *ICDM '06*. 613–622.

[40] Johan Ugander, Lars Backstrom, Cameron Marlow, and Jon Kleinberg. 2012. Structural diversity in social contagion. *PNAS* 109, 16 (2012), 5962–5966.

[41] Dmitry Ulyanov, Vedaldi Andrea, and Victor Lempitsky. 2016. Instance Normalization: The Missing Ingredient for Fast Stylization. *arXiv preprint arXiv:1607.08022* (2016).

[42] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor S Lempitsky. 2016. Texture Networks: Feed-forward Synthesis of Textures and Stylized Images.. In *ICML '16*. 1349–1357.

[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS '17*. 6000–6010.

[44] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Y Bengio. 2018. Graph Attention Networks. *ICLR '18* (2018).

[45] Duncan J Watts. 1999. *Small worlds: the dynamics of networks between order and randomness*. Princeton university press.

[46] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of 'small-world' networks. *nature* 393, 6684 (1998), 440–442.

[47] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. 2015. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853* (2015).

[48] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. In *KDD '15*. 1365–1374.

[49] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. 2016. Revisiting semi-supervised learning with graph embeddings. *ICML '16* (2016).

[50] Jing Zhang, Biao Liu, Jie Tang, Ting Chen, and Juanzi Li. 2013. Social Influence Locality for Modeling Retweeting Behaviors.. In *IJCAI' 13*.

[51] Jing Zhang, Jie Tang, Juanzi Li, Yang Liu, and Chunxiao Xing. 2015. Who influenced you? predicting retweet via social influence locality. *TKDD* 9, 3 (2015), 25.