Discerning Edge Influence for Network Embedding

Yaojing Wang
State Key Laboratory for Novel Software Technology, Nanjing University, China
wyj@smail.nju.edu.cn

Yuan Yao
State Key Laboratory for Novel Software Technology, Nanjing University, China
y.yao@nju.edu.cn

Hanghang Tong*
University of Illinois at Urbana-Champaign, United States
htong@illinois.edu

Feng Xu
State Key Laboratory for Novel Software Technology, Nanjing University, China
xf@nju.edu.cn

Jian Lu
State Key Laboratory for Novel Software Technology, Nanjing University, China
lj@nju.edu.cn

ABSTRACT

Network embedding, which learns the low-dimensional representations of nodes, has gained significant research attention. Despite its superior empirical success, often measured by the prediction performance of downstream tasks (e.g., multi-label classification), it is unclear *why* a given embedding algorithm outputs the specific node representations, and *how* the resulting node representations relate to the structure of the input network. In this paper, we propose to discern the edge influence as the first step towards understanding skip-gram basd network embedding methods. For this purpose, we propose an auditing framework NEAR, whose key part includes two algorithms (NEAR-ADD and NEAR-DEL) to effectively and efficiently quantify the influence of each edge. Based on the algorithms, we further identify high-influential edges by exploiting the linkage between edge influence and the network structure. Experimental results demonstrate that the proposed algorithms (NEAR-ADD and NEAR-DEL) are significantly faster (up to $2,000\times$) than straightforward methods with little quality loss. Moreover, the proposed framework can efficiently identify the most influential edges for network embedding in the context of downstream prediction task and adversarial attacking.

CCS CONCEPTS

• **Mathematics of computing** → **Graph algorithms**; **Approximation algorithms**; • **Theory of computation** → *Dynamic graph algorithms*; • **Networks** → Topology analysis and generation.

KEYWORDS

Network Embedding, Edge Influence, Network Topological Properties

*This work was partly done when the author was at Arizona State University.

ACM Reference Format:
Yaojing Wang, Yuan Yao, Hanghang Tong, Feng Xu, and Jian Lu. 2019. Discerning Edge Influence for Network Embedding. 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/10.1145/3357384.3358044

1 INTRODUCTION

Network embedding, which learns the low-dimensional representations of nodes, has gained significant research attention due to its strong empirical performance and wide applicability. The learned node representations have been successfully applied in a variety of downstream tasks including node classification [37], node clustering [46], link prediction [18], and visualization [42].

To date, many network embedding methods have been proposed. The basic idea of existing work is to first construct the context structure for each node and then design an appropriate objective function to preserve this structure. For example, the classic DeepWalk [37] uses truncated random walk to construct the context and applies the skip-gram model [31] to learn the node representations. Many sophisticated extensions have also been proposed including whether to preserve community structure, whether to incorporate node attributes, and whether to use supervision information (see the related work section for a review). Generally speaking, the existing work focuses on *what* kind of node representations can better improve the performance of the downstream prediction tasks (e.g., multi-label classification and link prediction). However, it is unclear *why* a given embedding algorithm outputs the specific node representations, and *how* the resulting node representations relate to the structure of the input network.

In this paper, instead of developing a new network embedding method, we propose a network embedding auditing framework (NEAR) to discern the edge influence, which serves as the first step towards understanding skip-gram based network embedding methods. Specially, we propose two algorithms (NEAR-DEL and NEAR-ADD) which can effectively and efficiently quantify the influence of a few edges on the learned embeddings[1], when edges are deleted or added, respectively. These algorithms can be used to efficiently quantify the edge influence and update the node embeddings when directly retraining is computationally too expensive. Furthermore, we propose

[1] We interchangeably use 'node representations' and 'embeddings' in this paper.

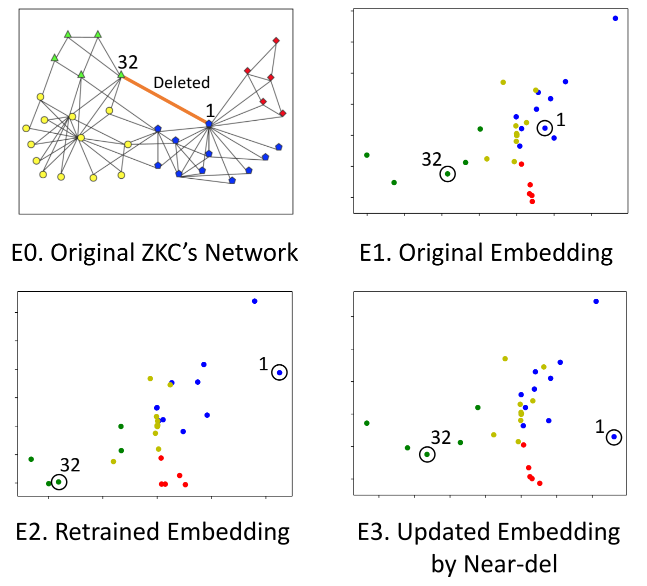


Figure 1: *E0*, deleting the edge between node 1 and node 32 in the Zachary's karate club social network. The nodes are colored according to their communities in the network. *E1*, two-dimensional visualization of node representations learned by the skip-gram model. *E2*, node representations learned by retraining the skip-gram model after the edge is deleted. *E3*, node representations learned by the proposal NEAR-DEL. The proposed method leads to similar node representations compared to the retraining method, while it is 10 times faster.

to exploit the linkage between the edge influence and key network topological properties, and use such linkage/correlations to identify the high-influential edges.

An illustrative example is shown in Fig. 1 where we use the famous Zachary's karate club network as the input. In this example, we first apply a skip-gram model on the network to learn the node embeddings (E1 of Fig. 1). Suppose that we are interested in the influence of the edge between node 1 and node 32. Intuitively, if we delete this edge, the learned embeddings of the two terminal nodes would drift apart from each other. A straightforward way to verify this is to retrain the network embedding model on the new network (the result is in E2 of Fig. 1). Alternatively, we can directly apply the proposed NEAR-DEL on the original embeddings and approximately obtain the updated embeddings (E3 of Fig. 1). As we can see in the example, for both the retraining method and the proposed NEAR-DEL, the learned embeddings of node 1 and node 32 drift apart from each other as compared to E1. Moreover, the learned embeddings of NEAR-DEL are similar to those of the retraining method. For instance, the embeddings of node 1 move from a central place to a peripheral place. Meanwhile, the proposed method is 10 times faster in terms of training time than the retraining method.

In experiments, we first evaluate the effectiveness and efficiency of the proposed algorithms (NEAR-ADD and NEAR-DEL) on two benchmark datasets. The results show that when the network changes, compared to the *retraining method* which retrains the network embedding model to obtain the updated node representations, the proposed methods are significantly faster (up to $2,000\times$ speedup) with little quality loss. Furthermore, we evaluate the effectiveness of the proposed auditing framework in terms of identifying the most influential edges in the context of downstream prediction task and adversarial attacking. The results demonstrate that the proposed method is indeed helpful to better understand the embedding results via effectively identifying influential edges in close relation to important network topological properties. For example, by considering edges with certain topological properties (e.g., node degrees and structural holes [3]), we find that even a single structural hole edge has a significant impact on the learned node embeddings. This will not only lead to an efficient pruning strategy for identifying high-influential edges, but also shed critical light to understand the key driving factor of a network embedding algorithm as well as design effective attacking and defending algorithms in the adversarial setting.

The main contributions of this paper include:

- We propose to discern the influence of edges as the first step towards understanding network embedding. We propose a network embedding auditing framework with two specific algorithms, which efficiently computes the influence of different edges and links edge influence with key network topological properties to identify high-influential edges.
- We conduct experimental evaluations on real datasets showing the effectiveness and efficiency of the proposed auditing framework NEAR. Specially, we demonstrate that the proposed algorithms NEAR-ADD and NEAR-DEL are significantly faster with little quality loss, and that the proposed framework can identify high-influential edges for the multi-label classification task and the adversarial attacking task.

The rest of the paper is organized as follows. Section 2 presents the problem statement, and Section 3 describes the proposed auditing framework. Section 4 shows the experimental results. Section 5 reviews related work, and Section 6 concludes.

2 PROBLEM STATEMENT

Before describing the auditing framework, we first introduce the problem statement. In this work, we focus on auditing the network embedding methods built upon the skip-gram model. Examples that adopt this model include [8, 14, 18, 35, 37, 39, 55].

Preliminary: Skip-gram Model based Network Embedding. Consider a network $G = (V, E)$, where V is the set of nodes and E is the set of edges. The skip-gram model with negative sampling typically optimizes the following objective function [18, 37],

$$\max_{\Theta} \sum_{(u,v)\in E'} \log\sigma(\theta_u^T \cdot \theta_v) + k \cdot \mathbb{E}_{v'\sim P}[\log\sigma(-\theta_u^T \cdot \theta_{v'})], \tag{1}$$

where E' is the edge set of the context network constructed from the original network, $\sigma(x) = 1/(1 + exp(-x))$, k is the negative sampling rate, and P is the negative sampling distribution. Note that the context of a node can be generated by applying various random walks on the original network. The goal of the above equation is to learn a d-dimensional representation $\theta_v \in \mathbb{R}^d$ for each node v. We use a $|V| \times d$ matrix Θ to denote the representations of all nodes.

Network Embedding Auditing Problem. In this work, we are interested in efficiently identifying the edges that have a relatively large influence on the learned embeddings. Given the input network G and the current learned embeddings $\hat{\Theta}$, we aim to (1) quantify the influence of a small change ΔG of the network structure, and (2) identify the high-influential edges.

We use $\hat{\Theta}'$ to denote the updated embeddings. Once we efficiently obtain the new embeddings $\hat{\Theta}'$, we use it to quantify the influence of ΔG, i.e., to what extent ΔG affects the embedding results. Although we can simply retrain the network embedding model to obtain its influence when the network changes, such a straightforward method would be time-consuming especially when the network is large. Furthermore, even if we could efficiently measure the edge influence, finding the most influential edges could still be time-consuming if we have to compute the influence over every edge of the input network. To this end, we propose to exploit the linkage between edge influence and key network topological properties, and use such structural properties as the indicators to identify high-influential edges.

3 THE AUDITING FRAMEWORK

In this section, we present the proposed network embedding auditing framework NEAR. We start with quantifying the edge influence, and then discuss how to identify high-influential edges.

3.1 Quantifying Edge Influence

We first present how to quantify the influence of each edge. The key intuition behind the proposed NEAR is based on the observation that the skip-gram model can be naturally decomposed into a set of subproblems over each edge. Take the model in Eq. (1) as an example. Given a network $G = (V, E)$, let us start by assuming $E' = E$ and defining its edges as $e_1, e_2, ..., e_{|E|}$. In this case, the objective function in Eq. (1) can be rewritten as

$$\hat{\Theta} = \text{argmax}_{\Theta} \frac{1}{|E|} \sum_{i=1}^{|E|} \mathcal{J}(e_i, \Theta), \tag{2}$$

which can be decomposed into subproblems $\mathcal{J}(e, \Theta)$ on each edge $e = (u, v)$, with $\mathcal{J}(e, \Theta) = \log \sigma(\theta_u^T \cdot \theta_v) + k \cdot \mathbb{E}_{v' \sim P}[\log \sigma(-\theta_u^T \cdot \theta_{v'})]$.

Based on Eq. (2), we can compute the influence of adding or deleting a few certain edges. For example, when slightly perturbing the weight of a certain edge e with some small ϵ (ϵ can be both positive and negative values), we have the following objective,

$$\hat{\Theta}_{\epsilon,e} = \text{argmax}_{\Theta} \frac{1}{|E|} \sum_{i=1}^{|E|} \mathcal{J}(e_i, \Theta) + \epsilon \mathcal{J}(e, \Theta). \tag{3}$$

Adding/deleting multiple edges can be seen as adding/deleting the edges one by one. In order to obtain $\hat{\Theta}_{\epsilon,e}$ based on $\hat{\Theta}$, we resort to the influence functions [25]. Specially, we obtain the influence $\mathcal{I}(e)$, which determines the difference between $\hat{\Theta}_{\epsilon,e}$ and $\hat{\Theta}$, of perturbing edge e as

$$\mathcal{I}(e) \overset{\text{def}}{=} \frac{\mathrm{d}\hat{\Theta}_{\epsilon,e}}{\mathrm{d}\epsilon}\Big|_{\epsilon=0} = -H_{\hat{\Theta}}^{-1} \nabla_{\Theta} \mathcal{J}(e, \hat{\Theta}), \tag{4}$$

Algorithm 1 The HVP4NE algorithm

```
Input: network $G = (V, E)$, embedding $\hat{\Theta}$, edge $e$, vector $g$
Output: estimated influence $-H^{-1}g$
 1: $list = [\ ]$
 2: for $i = 1 \to r$ do
 3:     define $H_0^{-1} = I$
 4:     for $j = 1 \to l$ do
 5:         $e_s \leftarrow$ EdgeSample$(E, e)$
 6:         $H_j^{-1}g \leftarrow g + (I - \nabla_{\Theta}^2 \mathcal{J}(e_s, \hat{\Theta}))H_{j-1}^{-1}g$
 7:     end for
 8:     $list[i] = H_t^{-1}g$
 9: end for
10: return $-$Average$(list)$
```

where $H_{\hat{\Theta}} = \frac{1}{|E|} \sum_{i=1}^{|E|} \nabla_{\Theta}^2 \mathcal{J}(e_i, \hat{\Theta})$ is the Hessian matrix[2]. Notice that both $H_{\hat{\Theta}}$ and $\mathcal{J}(e, \hat{\Theta})$ are computed based on the current learned node embeddings $\hat{\Theta}$.

Scale up the Computation. Directly computing the edge influence based on Eq. (4) is computationally expensive. Here, we provide an approximate algorithm to compute $H_{\hat{\Theta}}^{-1} \nabla_{\Theta} \mathcal{J}(e, \hat{\Theta})$ in Eq. (4). Specially, we apply Hessian-vector products (HVPs) [2, 36] to efficiently approximate $H_{\hat{\Theta}}^{-1}g$ where $g = \nabla_{\Theta} \mathcal{J}(e, \hat{\Theta})$, $\nabla_{\Theta}^2 \mathcal{J}(e_s, \hat{\Theta})h$, and $h = H_{j-1}^{-1}g$ are all vectors. Here, the subtle point of HVPs is that we actually do not need to compute the multiplication step between matrix and vector, but rather treat the $H_j^{-1}g$ as one vector, and then update this vector over iterations. In detail, when computing the HVPs between H^{-1} and vector g, we first define $H_0^{-1}g \leftarrow g$ and recursively compute $H_j^{-1}g \leftarrow g + (I - \nabla_{\Theta}^2 \mathcal{J}(e_s, \hat{\Theta}))H_{j-1}^{-1}g$ to approximate $H^{-1}g$, where e_s is a sampled edge. By doing so, we do not need to compute the products of $H_j^{-1}g$ but only need to treat it as a vector and update it over iterations. As for the sampling of edge e_s, we sample the edges having a common node with edge e. In other words, we consider the influence of perturbing edge e in the local neighborhood of edge e.

The algorithm of HVPs computation for network embedding is summarized in Alg. 1. In the algorithm, we first iterate r times and return the average influence to reduce noise (Line 2 and Line 10). In each iteration, we compute the $H^{-1}g$. We pick l to be large enough such that $H^{-1}g$ stabilizes (Line 4). The key steps of the algorithm are Line 5 and Line 6 which update $H^{-1}g$. In Line 5, we sample the edge e_s in the local neighborhood of edge e. Therefore, instead of updating all the $d \times |V|$ parameters, we only need to update the learned embeddings of the nodes related to edge e_s. In Line 6, we directly update $H_j^{-1}g$ without computing the products. The overall time complexity of Alg. 1 is summarized in the following lemma, which can be further reduced to $O(d)$ if the number of iterations r and the number of sampled edges l are small constants. This is much more efficient than directly computing Eq. (4).

LEMMA 1. **Time complexity of HVP4NE.** *Alg. 1 requires $O(rld)$ time to estimate the Hessian-vector product $H^{-1}g$.*

[2]When deleting edges, we make sure that the network is still connected so that the Hessian matrix is not singular.

Algorithm 2 The NEAR-DEL algorithm

```
Input: network $G = (V, E)$, original embedding $\hat{\Theta}$, edge $e$
Output: new embedding $\hat{\Theta}_{-e}$
1: $g \leftarrow \nabla_{\hat{\Theta}} \mathcal{J}(e, \hat{\Theta})$
2: $\mathcal{I}(e) \leftarrow$ HVP4NE$(G, g, e, \hat{\Theta})$
3: $\hat{\Theta}_{-e} \leftarrow \hat{\Theta} - \frac{1}{|E|}\mathcal{I}(e)$
4: return $\hat{\Theta}_{-e}$
```

PROOF. Omitted for brevity. □

Remarks. Alternatively, we can directly compute the gradients of the added/deleted edges and then update the related node embeddings accordingly. However, as will shown in our experiments, we found that updating the node embeddings in such a stochastic way would lead to much less accurate results. Additionally, note that in the above solution for quantifying edge influence, we assume that $E' = E$, i.e., the context network equals to the original network. We can also extend the solution for more general context networks such as those generated by truncated random walks. We will leave such extensions as future work.

NEAR-DEL: Deleting Edges. Next, we present the proposed NEAR-DEL algorithm to deal with the case when edges are deleted in the network. This would help characterize the importance of different edges.

Formally, we use $\hat{\Theta}_{-e} - \hat{\Theta}$ to denote the influence (or embedding differences) of removing an edge $e = (u, v)$. By setting $\epsilon = -\frac{1}{|E|}$ in Eq. (3), and substituting the skip-gram model in Eq. (1) into it, we approximate the influence as follows,

$$\hat{\Theta}_{-e} - \hat{\Theta} \approx -\frac{1}{|E|}\mathcal{I}(e) = \frac{1}{|E|} H_{\hat{\Theta}}^{-1} \nabla_{\Theta}(\log \sigma(\hat{\theta}_u^T \cdot \hat{\theta}_v) + k \cdot \mathbb{E}_{v' \sim P}[\log \sigma(-\hat{\theta}_u^T \cdot \hat{\theta}_{v'})]).$$

Based on the above equation, deleting multiple edges can be handled by iteratively deleting one edge each time. For example, we use $\hat{\Theta}_{-E_s} - \hat{\Theta}$ to denote the influence of removing a set E_s of m edges, where $E_s = \{e_{s_1}, ..., e_{s_m}\}$ contains the edges to delete. By defining

$$\hat{\Theta}_{\epsilon,-E_s} = \text{argmin}_{\Theta} \frac{1}{|E|} \sum_{i=1}^{|E|} \mathcal{J}(e_i, \Theta) + \sum_{i=1}^{m} \epsilon_i \mathcal{J}(e_{s_i}, \Theta), \qquad (5)$$

and setting $\epsilon_i = -\frac{1}{|E|}$ for $i \in \{1, ..., m\}$, we can have the following approximation.

$$\hat{\Theta}_{-E_s} - \hat{\Theta} \approx \frac{1}{|E|} \sum_{i=1}^{m} H_{\hat{\Theta}}^{-1} \nabla_{\Theta} \mathcal{J}(e_{s_i}, \hat{\Theta}). \qquad (6)$$

Remarks. The proposed algorithm can also be used for the case of deleting nodes. For example, when deleting a node, we can treat it as deleting all the edges that connect to this node. Therefore, we only present the algorithm for deleting a single edge e in Alg. 2, and the algorithms for deleting nodes and multiple edges can be obtained in a similar way. In the algorithm, a key step is to invoke Alg. 1 to obtain the influence.

NEAR-ADD: Adding Edges. Next, we present NEAR-ADD which can estimate the learned embeddings when some new edges are inserted into the network. NEAR-ADD can be used to incrementally update the existing network embedding model when the network dynamically grows.

Suppose we are going to add an edge $\tilde{e} = (u, v) \notin E$, and the learned embeddings of $\tilde{G} = (\tilde{V}, \tilde{E})$ is $\hat{\Theta}_{+\tilde{e}}$, where $\tilde{E} = E \cup \{\tilde{e}\}$ and $\tilde{V} = V \cup \{u, v\}$. The basic idea is to first assume that the original objective function in Eq. (2) contains edge $\tilde{e}$ with weight 0, and then tune the weight from 0 to 1 by setting $\epsilon = \frac{1}{|E|}$. The remaining steps are similar to the case of deleting edges, and we omit the detailed algorithms and equations for brevity.

Algorithm Analysis. Next, we analyze the proposed algorithms in terms of both effectiveness and efficiency.

A - Approximation Error analysis. We first analyze the effectiveness of NEAR-DEL and NEAR-ADD. As stated in the following lemma, there are two possible places where we could introduce the approximation error: one is from the approximation of the parameter change when edges are removed/inserted, and the other is from HVPs computation in Alg. 1.

LEMMA 2. **Effectiveness of NEAR-DEL and NEAR-ADD.** *Let $\Theta^*_{\epsilon,e}$ be the parameters learned by retraining the network embedding model after we tune edge e. We have that $\hat{\Theta}_{\epsilon,e} = \Theta^*_{\epsilon,e}$ if $\epsilon \rightarrow 0$ in Eq.* (3)*, and it samples all possible edges with iteration number $t \rightarrow \infty$ in Alg. 1.*

PROOF. Omitted for brevity. □

The above lemma means that when ϵ is sufficiently small (or when the network size |E| is sufficiently large), and the iteration number t and sampled edge number l are sufficiently large, our approach will produce nearly the same embeddings as the retraining method.

B - Complexity analysis. The time complexity of NEAR-DEL and NEAR-ADD is summarized in the following lemma. It states that the proposed algorithms scale linearly w.r.t. the network size (i.e., the node number) when deleting/adding edges.

LEMMA 3. **Time complexity of NEAR-DEL and NEAR-ADD.** *When m edges are deleted or added,* Near-del/Near-add *requires $O(md|V|)$ time to generate the updated embeddings.*

PROOF. Omitted for brevity. □

3.2 Identifying High-influential Edges

Although the proposed algorithms in the previous subsection can efficiently *quantify* the influence of each edge with an $O(md|V|)$ time complexity, *identifying* the most influential edges for network embedding could still be time-consuming when the network is large. This is because the overall time complexity would be $O(md|V||E|)$ ($|V|$ and $|E|$ being numbers of nodes and edges of the input network), if we have to compute influence for every edge of the input network. In this subsection, we exploit the correlation between network topological properties and edge influence to design efficient filtering method. Specially, we study the edge properties that are related to node degree, PageRank value, structural hole, and triadic formation. In addition to speeding-up the computation, the correlation/linkage between network topology and edge influence can also help answer key questions to better understand the network embedding, e.g., what kinds of network topology make or break a network embedding algorithm (explainability)? how can we design effective attacking and defending algorithms for network embedding (adversarial mining)?

For completeness, we briefly explain how to compute these topological properties. The first property of node degree (which means how many edges are connected to the node) is relatively straightforward to understand. We compute two features related to the node degree for an edge $e = (u, v)$, i.e., $f_1 = d_u + d_v, f_2 = |d_u - d_v|$, where d_u and d_v are the degrees of node u and v, respectively. Here, f_1 reflects the overall degree related to the edge, and f_2 reflects the balance level of the two terminal nodes. For example, an edge with both large f_1 and f_2 means that this edge connects to a high-degree node and a low-degree node.

The second property PageRank ranks the importance of each node in a network. We use $\mathbb{PR}_u$ to indicate the PageRank value of node u. Similar to the first property, we define $f_3 = \mathbb{PR}_u + \mathbb{PR}_v$ and $f_4 = |\mathbb{PR}_u - \mathbb{PR}_v|$ for edge $e = (u, v)$.

As to the third property, a structural hole edge/node indicates that the edge/node serves as a mediator (to some extent) among two or more closely connected sub-networks. Following [3], we use the effective size $\mathbb{ES}_u$ to evaluate the strength of structural hole for each node u. Then, for each edge $e = (u, v)$, we compute its structural hole features as $f_5 = \mathbb{ES}_u + \mathbb{ES}_v$ and $f_6 = |\mathbb{ES}_u - \mathbb{ES}_v|$.

For the fourth property, we study the triadic formation of edges. The triadic formation value $\mathbb{TF}_e$ for edge $e = (u, v)$ is defined as the number of closed triads that the edge belongs to, and a larger $\mathbb{TF}$ indicates that the edge is strongly clustered in the neighborhood. We directly use $f_7 = \mathbb{TF}_e$ as the feature.

With the above topological properties computed, we correlate them with the corresponding edge influence, and build a filtering pipeline for efficiently identifying high-influential edges. To efficiently compute the correlation, we sample a subset of edges for each property/feature before computing the edge influence. Take node degree feature f_1 as an example. We randomly select 100 edges, and then compute the Spearman correlation between the f_1 values and the corresponding edge influence. We found that the node degree (f_1 and f_2) and the structural hole (f_5) are two significant positive indicators. Here, both f_1 and f_2 are positively correlated to edge influence, meaning that an edge connecting to a high-degree node and a low-degree node would be more influential than an edge connecting to two high-degree nodes. The readers may refer to Table 2 in the next section for detailed correlation results.

To build a filtering pipeline for identifying high-influential edges, we first filter out most (potentially) low-influential edges based on the node degree properties (i.e., f_1 and f_2, which can be computed very fast), and then compute the structural hole property for the remaining edges. In detail, if we aim to select K most influential edges, we first compute f_1 and f_2 for each edge, and select $5K$ edges with the highest f_1 and f_2 values. Next, we compute the f_5 value for these $5K$ edges and return the top K of them.

4 EXPERIMENTAL EVALUATIONS

In this section, we present the experimental results.

4.1 Experimental Setup

Datasets. We use two widely-used datasets in network embedding research: PPI and BlogCatalog [18]. PPI is a sub-graph of the Protein-Protein Interaction network for Homo Sapiens. The labels stand for biological states, and each protein may have multiple states. BlogCatalog is a network of social relationships among the bloggers. The labels represent topic categories, and each blogger may have multiple topic categories. Both datasets are publicly available, and the statistics of the datasets are shown in Table 1.

Table 1: Statistics of datasets.

Dataset	# of vertex	# of edges	# of labels
PPI	3,890	76,584	50
BlogCatalog	10,312	333,983	39

Effectiveness Evaluation Protocol for Quantifying Edge Influence. To verify the effectiveness of the proposed auditing method for quantifying edge influence, we use edge deletion as the evaluation scenario. Specially, we consider two aspects to verify the effectiveness. The first aspect is about the accuracy of the updated embeddings compared to the retraining method. That is, we first randomly select an edge and delete it in the original network, and use NEAR-DEL to learn the new embeddings $\hat{\Theta}_C$. We also use the *retraining method* which retrains the network embedding model on the new network to obtain the embeddings $\hat{\Theta}_B$. The accuracy is defined over the similarity between $\hat{\Theta}_C$ and $\hat{\Theta}_B$. Ideally, $\hat{\Theta}_C$ should be the same with $\hat{\Theta}_B$. However, due to the non-convex nature of the problem and the stochastic nature in both the network embedding model and the proposed algorithms, the learned embeddings of two repetitive training processes may deviate from each other. Therefore, we do not directly compare the absolute embedding differences between $\hat{\Theta}_B$ and $\hat{\Theta}_C$. Instead, inspired by the kernel method in SVM, we define a metric called Relative Node Distance (RND) to measure their relative distance compared to the original embedding $\hat{\Theta}_A$.

Take Fig. 2 as an example. We assume that we are interested in whether the representation update is accurate for node y. Instead of directly measuring the distance between the two embeddings of node y, we compare the relative position of node y in the spaces of $\hat{\Theta}_B$ and $\hat{\Theta}_C$. Specially, we first normalize the learned embeddings, and then calculate three distance values (i.e., $d_{x,y,A}$, $d_{x,y,B}$, and $d_{x,y,C}$ in Fig. 2) between node y and a randomly selected landmark node x in the embeddings learned from the original network, from the retraining method, and from the proposed NEAR-DEL, respectively. Ideally, the three values should satisfy $d_{x,y,A} > d_{x,y,B} \approx d_{x,y,C}$ or $d_{x,y,A} < d_{x,y,B} \approx d_{x,y,C}$. Therefore, we use Γ_y to denote the RND measurement for the target node y with landmark node x, which is defined as

$$\begin{aligned} \Gamma_{x,y} &= \left| \frac{d_{x,y,B} - d_{x,y,C}}{d_{x,y,A} - d_{x,y,C}} \right|, \\ \Gamma_y &= \frac{1}{L} \sum_{x=1}^{L} \Gamma_{x,y}, \end{aligned} \tag{7}$$

where we compute the average $\Gamma_{x,y}$ value over L landmark nodes. In this work, we randomly select 100 landmark nodes for each target node (i.e., $L = 100$). Smaller RND value indicates better quality of the proposed method, meaning that the distance between $\hat{\Theta}_C$ and $\hat{\Theta}_B$ is relatively smaller than that between $\hat{\Theta}_C$ and $\hat{\Theta}_A$.

The second aspect to verify the effectiveness is to use the embeddings in the downstream prediction tasks. In this work, we study the

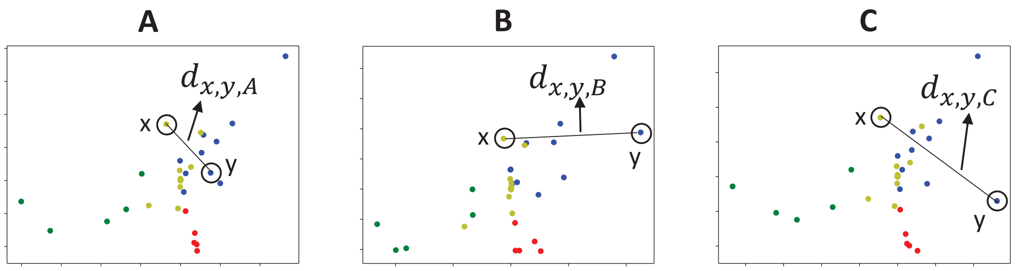


Figure 2: An illustration of the Relative Node Distance measurement. *A*, node embeddings learned on original network. *B*, node embeddings learned by the retraining method. *C*, node embeddings learned by NEAR-DEL.

multi-label classification task. That is, we first calculate the Micro-F1 and Macro-F1 scores of accurately classifying the nodes based on the two embeddings of $\hat{\Theta}_B$ and $\hat{\Theta}_C$, and then compute the absolute differences between the scores. To reduce noise, we repeat the multi-label classification 10 times and return the averaged results of Micro-F1 and Macro-F1 scores. Further, we repeat the above process 100 times by randomly deleting 100 different edges (deleting one edge each time)[3]. Then, we report the mean absolute error (MAE) as well as its standard deviation over the deleted edges as the final results. In this experiment, we will also compare the proposed method with a baseline that updates the node embeddings by directly computing the gradients of the deleted edges.

For the second aspect, we also verify the proposed extension for more general context networks. We take $p = 2$ in Eq. (**??**) as an example, and use the updated embeddings in the multi-label classification task. In this experiment, we delete/add one edge in the original network, and sample q edges based on NEAR in the difference of the context network. Note that the q edges are sampled based on the filter pipeline described in Subsection 3.2. We set q=10, 20, 30, 40, and 50. Similarly, we repeat the experiments 10 times and report the averaged MAE results of the prediction differences between the retrained embeddings ($\hat{\Theta}_B$) and the updated embeddings ($\hat{\Theta}_C$).

Effectiveness Evaluation Protocol for Identifying High-influential Edges. Here, we check if our method can identify more influential edges than the baselines in the context of both multi-label classification and adversarial attacking.

For the multi-prediction task, we check if the downstream prediction performance of the learned embeddings can be significantly affected by deleting the identified edges by NEAR. For this purpose, we select 1, 10, and 20 edges returned by the proposed auditing framework, and randomly select the corresponding numbers of edges for comparison (the latter one is denoted as 'random method'). We compute the differences of Micro-F1 and Macro-F1 scores on the multi-label classification task before and after these edges are deleted. We repeat this process 50 times and report the average differences of Micro-F1 and Macro-F1 scores to see the effect of these deleted edges. We also test the significance of the differences between the results returned by NEAR and the random method using paired t-test.

[3]Choosing more edges would be too time-consuming for the retraining method.

For the adversarial attacking task, we conduct a vulnerability analysis by poisoning the training data. Specially, the attacking goal is to change a target node's prediction by manipulating this node (i.e., adding a few edges). Suppose $\hat{\Theta}$ contains the embeddings learned from the original network G, and $\hat{\Theta}'$ contains the embeddings learned from the network $G + \Delta G$, where ΔG contains the edges we aim to add. Defining $f(\hat{\Theta})$ and $f(\hat{\Theta}')$ as the prediction results of the corresponding embeddings, the attacking goal is to maximize the following objective function,

$$\begin{aligned} &h(f(\hat{\Theta}), f(\hat{\Theta}')) \\ &s.t.\ |\Delta G| \leq K, \end{aligned} \tag{8}$$

where h is the function that measures the difference between the two predictions, and we constrain that there could be at most K edges to be added. We use the multi-label classification task for the f function. For the h function, we define two metrics related to attacking ability (i.e., *AA* and *CAA*) as follows.

The first metric is *AA*, which is defined as the ratio between the number of misclassified labels and the number of correct labels,

$$AA = 1 - \frac{|C_c \cap C_m|}{|C_c|}, \tag{9}$$

where C_c contains the correctly classified classes by the original embeddings and C_m contains the predicted classes after manipulation. We also define the second h function *CAA*, which measures the ability to modify any predicted classes compared with the original ones.

$$CAA = \begin{cases} 1, & C_c \neq C_m; \\ 0, & otherwise. \end{cases} \tag{10}$$

Basically, these metrics measure the ability to mislead the predictions, and larger *AA* and *CAA* mean better attacking ability.

Efficiency Evaluation Protocol. For efficiency, we record and compare the wall-clock time of NEAR-DEL and the retraining method. All the methods are run on the same machine with 64G memory and 4 CPU cores (at 4.0GHZ using 8 threads). For the parameters of the proposed algorithms, we set $l = 1,000$ and $r = 10$ in Alg. 1.

4.2 Effectiveness Results

(A) Relative Node Distance Measurement for NEAR-DEL. We first present the Relative Node Distance (RND) results of the proposed algorithm NEAR-DEL in terms of updating the node embeddings. We randomly delete one edge each time, and then repeat this process 100

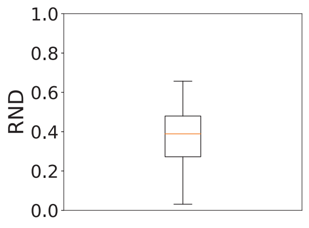
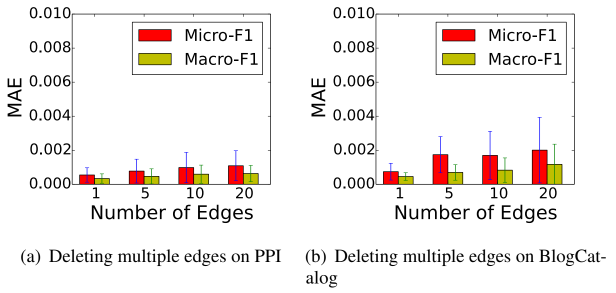


Figure 3: Relative node distance measurement on the node embeddings. NEAR-DEL can accurately update the node representations.

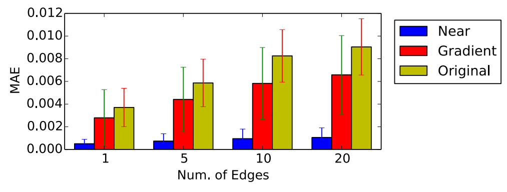


(a) Deleting multiple edges on PPI (b) Deleting multiple edges on BlogCatalog

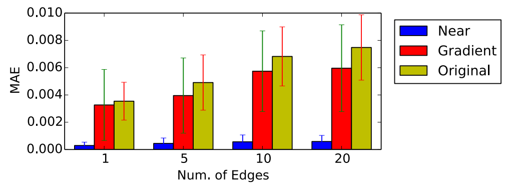
Figure 4: The effectiveness results of deleting multiple edges. The proposed method has little quality loss compared to the retraining method.

times to obtain 100 different RND values. For each RND value, we compute it 100 times and report the average result. The result on the PPI data with embedding dimension 128 is shown in Fig. 3. Similar results are observed on the BlogCatalog data, and are omitted for brevity. RND = 1 is a reference line where we do not update the embeddings at all. As we can see from the figure, the median RND value is around 0.4 and almost all the RND values are between 0 and 0.7, which means that the distance between the embedding updated by NEAR-DEL and the retrained embedding is relatively smaller than the distance between the embedding updated by NEAR-DEL and the original embedding. In other words, NEAR-DEL is accurate in terms of updating the embedding results from the original place to the retrained place.

(B) Downstream Prediction Task Measurement for NEAR-DEL. Next, we measure the effectiveness of the proposed algorithm NEAR-DEL based on the performance on the downstream prediction task. The MAE results of the prediction differences between the retrained embeddings ($\hat{\Theta}_B$) and the updated embeddings ($\hat{\Theta}_C$) are shown in Fig. 4. We fix the percentage of training data to 50%, and report the average results of 100 experiments. The x-axis of the figure indicates the number of edges to delete each time, and the y-axis is MAE. We can observe that as the number of deleted edges increases, the MAE values also increase. This is consistent with our error analysis as the approximation error might be accumulated over time. Nonetheless, even with 20 edges deleted, the relative error is smaller than 1% on both datasets (0.18%-0.96% on PPI and 0.19%-0.74% on BlogCatalog).



(a) Micro-F1 score



(b) Macro-F1 score

Figure 5: The effectiveness comparisons of NEAR-DEL and baseline methods on deleting edges. The proposed method outperforms the baseline methods in terms of obtaining the updated node embeddings.

Table 2: Spearman correlations between edge property and edge influence.

Property	Description	Correlation	p-value
f_1	$d_u + d_v$	0.6604	<0.001
f_2	$\|d_u - d_v\|$	0.3803	<0.01
f_3	$\mathbb{PR}_u + \mathbb{PR}_v$	0.4084	-
f_4	$\|\mathbb{PR}_u - \mathbb{PR}_v\|$	0.1526	-
f_5	$\mathbb{ES}_u + \mathbb{ES}_v$	0.8995	<0.001
f_6	$\|\mathbb{PR}_u - \mathbb{PR}_v\|$	0.1254	-
f_7	$\mathbb{TF}_e$	0.2153	-

To further show the effectiveness of the proposed algorithm, we compare it with two baseline methods. The first baseline (denoted as 'Gradient') directly computes the gradients of the deleted edges and updates the embeddings accordingly [42]. The second baseline (denoted as 'Original') assumes no update with edges deleted. That is, it computes the MAE between the prediction results of the original embedding $\hat{\Theta}_A$ and retrained embedding $\hat{\Theta}_B$. The results on the PPI data are shown in Fig. 5. As we can see, the MAE values between the original embeddings and retrained embddings are significantly larger than the proposed NEAR-DEL. This indicates that even a small number of edges are deleted/added, it would be crucial to retrain/update the embeddings instead of directly using the original model. The MAE values of the Gradient baseline are also relatively larger than NEAR. This result indicates the effectiveness of the proposed NEAR-DEL. Overall, the proposed algorithm bears little quality loss compared to the retraining method in terms of quantifying the edge influence.

(C) Correlation Results of Network Topological Properties. Next, we check whether the identified high-influential edges would have

Table 3: The effectiveness of NEAR for identifying influential edges for the multi-label classification task. The selected edges by the proposed NEAR have a relatively larger effect on the learned embeddings. That is, when a few edges are deleted by NEAR, the performance of the learned emebddings in terms of the downstream prediction task would significantly decrease. (● indicates the result is significantly smaller than the randomly selected edges with p-value< 0.01, and ○ indicates no significant difference.)

Edges	1 edge		10 edges		20 edges	
	Micro-F1 (%)	Macro-F1 (%)	Micro-F1 (%)	Macro-F1 (%)	Micro-F1 (%)	Macro-F1 (%)
random edges	-0.0322	-0.0856	-0.0388	-0.0641	-0.0351	0.1151
Top ND edges	-0.0193 ○	-0.0635 ○	0.0445 ○	0.1201 ○	0.0468 ○	0.0514 ○
selected edges by NEAR	-0.1634 ●	-0.1975 ●	-0.1462 ●	-0.1972 ●	-0.2653 ●	-0.2256 ●

significant impact in terms of multi-label classification and adversarial attacking. Before that, we first present the correlation results between the edge properties and the edge influence. We compute the Spearman correlation between each edge property/feature and the corresponding edge influence. The results are shown in Table 2. As we can see from the table, the structural hole feature f_5 has the highest correlation with edge influence (the correlation coefficient is 0.8995). This result indicates that the structural hole is perhaps the most indicative property for high edge influence. In addition, the node degree features (f_1 and f_2) are also positively correlated. This result indicates that edges connecting to a high-degree node and a low-degree node tend to have a larger influence on the learned embeddings, than edges connecting to two high-degree nodes.

(D) Multi-label Classification Results for NEAR. Here, we check if the identified high-influential edges by NEAR have a significant effect on the multi-label classification task. That is, we delete the selected edges by NEAR, update the learned embeddings, and test the performance of the embeddings in the multi-label classification task. The results are shown in Table 3, where we also report the results of the random method and the method of deleting edges based on node degree feature f_1. The random method randomly delete edges, and the 'ND' method deletes edges with highest f_1 values.

We can observe from Table 3 that when the deleted high-influential edges are selected by NEAR, the performance of the learned emebddings in the multi-label classification task would significantly decrease. Recall that NEAR selects edges based on the structural hole property (f_5) in the last step. This means that the structural hole property has a relatively large impact on the learned embeddings, i.e., when a few such edges are deleted, the performance of the learned emebddings would significantly decrease. Moreover, we can observe from Table 3 that even when a single structural hole edge is deleted, it can significantly decrease the performance of the learned embeddings. In contrast, deleting, for instance, the 'Top ND edges' would have little impact of the learned embeddings. This is consistent with our intuition, i.e., deleting an edge that belongs to a high-degree node would have less influence on the embedding results compared to deleting a structural hole edge, as the latter has a larger influence on the network connectivity.

(E) Adversarial Attacking Results for NEAR. Next, we use NEAR to add some high-influential edges for a given target node, and check if such manipulations could mislead the node prediction results in the multi-label classification task. For a target node, we are allowed to add K edges where K is set to $1, 3, 7$, and 10 in our experiments. We randomly choose 50 target nodes whose degree is between 30 and 50, and report the average attacking ability results on the PPI data in Table 4. For comparison, we also report the results of the 'random' method and the 'degree' method. The random method randomly selects edges to add. The degree method selects edges that have larger f_1 values. Note that directly computing the edge influence for each target node would be computationally expensive as the time complexity is $O(md|V|^2)$.

Table 4: The effectiveness of NEAR for identifying influential edges for adversarial attacking. NEAR can effectively find the vulnerable edges to manipulate.

		K			
		1	3	7	10
Add K edges	AA_{rand}	0.287	0.303	0.311	0.345
	AA_{degree}	0.371	0.384	0.457	0.496
	AA_{NEAR}	**0.403**	**0.554**	**0.527**	**0.631**
	CAA_{rand}	0.320	0.340	0.340	0.400
	CAA_{degree}	0.380	0.400	0.480	0.520
	CAA_{NEAR}	**0.500**	**0.640**	**0.680**	**0.740**

As we can see from the table, choosing the influential edges as quantified by the proposed method can significantly mislead the classification results. For example, when adding one single edge, half of the node prediction results will be changed, and 40.3% labels cannot be correctly predicted. When adding 10 edges, 74% node predictions will be changed, and 63.1% labels cannot be correctly predicted. As to the competitors, when 10 edge are added, the random method can only change the predictions of 40% nodes and mislead 34.5% labels, while the degree method can change the predictions of 52% nodes and mislead 49.6% labels.

Overall, NEAR can identify high-influential edges that effectively affect the multi-label classification performance, and locate vulnerable edges to manipulate for adversarial attacking.

4.3 Efficiency Results

Finally, we present the efficiency results. In particular, we delete several edges each time and report the average wall-clock time of the retraining method and our methods for quantifying edge influence. The results are shown in Fig. 6 where the y-axis is in log scale. As we can see, the proposed methods are significantly faster than the retraining method. For example, on the BlogCatalog data, the proposed method achieves $100\times$ to $2,000\times$ speedup when 1 - 20 edges are deleted each time. Additionally, the proposed method scales linearly w.r.t. the number of deleted edges, which is consistent with the algorithm analysis in Subsection 3.1.

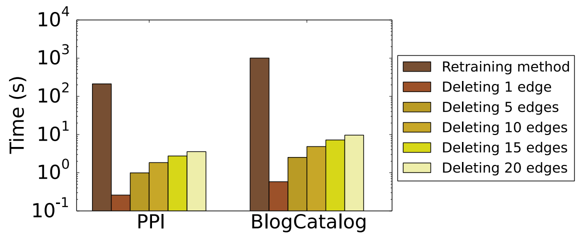


Figure 6: Efficiency results of NEAR. The proposed method achieves up to $2,000\times$ speedup compared to the retraining method.

5 RELATED WORK

In this section, we briefly review the related work, including network embedding models, explainable learning methods, and adversarial learning methods on network embedding. Since network embedding is a very active research area, here we only cover the most relevant existing network embedding models. We refer readers to some recent surveys [4, 9, 20, 47] for more details.

Network Embedding Models. The basic network embedding methods take the network structure as input and aim to learn the node representations via context network construction and different objectives. For example, several researchers propose to apply the skip-gram model [31, 32] to learn the node representations (e.g., DeepWalk [37], LINE [42], Node2Vec [18] and DP-Walker [14]). Others apply low-rank matrix approximation (e.g., GraRep [5] and NetMF [38]), or neural network models such as autoencoders (e.g., DNGR [6], NetRA [54] and SDNE [43]) and adversarial networks (e.g., ANE [11] and GraphGAN [44]).

Built upon the basic network embedding models, various extensions have been proposed in literature. The most studied extensions include handling node attributes during the embedding process [7, 16, 19, 22, 27, 28, 30, 49–51], and incorporating the supervision labels under a semi-supervised setting [17, 21, 24, 35, 40, 41, 52]. There also exist other extensions. For example, Wang et al. [46] propose to preserve the community structure when such information is available; Ou et al. [34] and Zhou et al. [55] pay special attention to directed networks by computing direction-aware proximities; Wang et al. [45] and Kim et al. [23] focus on signed networks where there exist both positive and negative links; Xu et al. [48], Dong et al. [13], and Fu et al. [15] target on heterogeneous networks with nodes/edges of different types. Orthogonal to the above models, our focus is on how to determine the influence of a small network change on the learned embeddings.

The most related work is DANE [26], HTNE [58], CTDNE [33], DynamicTriad [56], and NetWalk [53]. whose focus is to incrementally learn the node embeddings. However, they build their methods with their own network embedding models (based on eigen-decomposition, Hawkes process, temporal random walk, triadic closure process, and network streams, respectively). In contrast, our proposed method can potentially update many existing network embedding methods that are built upon the widely-used skip-gram model. Liu et al. [29] also aim to interpret network embedding. Their focus is to connect the learned embeddings with the node attributes, while our focus is to quantify the influence of edges.

Explainable Learning. Understanding and interpreting the complex machine learning models have been receiving tremendous attention in recent years. Typically, existing methods treat machine learning models as black-box systems and study how a fixed model leads to particular prediction. For example, researchers propose to perturb either the test points [1, 12] or the training points [25] to see how the prediction changes. In this work, we propose a network embedding auditing framework to help better understand network embedding models. Based on this framework, we can efficiently analyze the influence of different edges on the embedding results, and incrementally update the learned embeddings when the network changes.

Adversarial Learning. Some recent work has focused on the adversarial learning of network embedding models. For example, Zügner et al. [57] learn the adversarial attacks on a surrogate of the semi-supervised neural network based network embedding models (e.g., GCN [24]); Dai et al. [10] formulate the problem as a Markov decision process and propose a reinforcement learning method to attack against supervised graph neural networks. In contrast to these two proposals, we study the adversarial attacks on the unsupervised skip-gram based network embedding.

6 CONCLUSIONS

In this paper, we propose to discern edge influence as the first step towards understanding network embedding methods built upon the skip-gram model. We propose a network embedding auditing framework NEAR to efficiently quantify edge influence and identify high-influential edges. We instantiate two algorithms supporting edge deletion and addition, and correlate edge influence with network topological properties to efficiently identify high-influential edges. The experimental results demonstrate that the proposed method is accurate in terms of quantifying the edge influence, and in the meanwhile it is much more efficient than the retraining method. Moreover, the proposed method helps better understand network embedding and identify high-influential edges in the tasks of multi-label classification and adversarial attacking. There are also some interesting findings in the experiments. For example, even when a single structural hole edge is deleted, it could significantly decrease the performance of the learned embeddings in the downstream prediction task.

The proposed auditing method could provide a critical building block for future research in multiple directions, including (1) auditing embedding on attributed and/or dynamic networks, (2) more comprehensive attacking strategies for network embedding, and (3) robust network embedding.

7 ACKNOWLEDGEMENT

This work is supported by the National Natural Science Foundation of China (No. 61632021, 61672274, 61702252) and the Collaborative Innovation Center of Novel Software Technology and Industrialization. Hanghang Tong is partially supported by NSF (IIS-1651203, IIS-1715385), and DHS (2017-ST-061-QA0001). Yuan Yao is the corresponding author.

REFERENCES

[1] Philip Adler, Casey Falk, Sorelle A Friedler, Tionney Nix, Gabriel Rybeck, Carlos Scheidegger, Brandon Smith, and Suresh Venkatasubramanian. 2018. Auditing black-box models for indirect influence. *Knowledge and Information Systems* 54, 1 (2018), 95–122.

[2] Naman Agarwal, Brian Bullins, and Elad Hazan. 2016. Second-order stochastic optimization in linear time. *stat* 1050 (2016), 15.

[3] Ronald S Burt. 2009. *Structural holes: The social structure of competition*. Harvard university press.

[4] Hongyun Cai, Vincent W Zheng, and Kevin Chang. 2018. A comprehensive survey of graph embedding: problems, techniques and applications. *IEEE Transactions on Knowledge and Data Engineering* (2018).

[5] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *CIKM*. 891–900.

[6] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2016. Deep neural networks for learning graph representations. In *AAAI*. 1145–1152.

[7] Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C Aggarwal, and Thomas S Huang. 2015. Heterogeneous network embedding via deep architectures. In *KDD*. ACM, 119–128.

[8] Jifan Chen, Qi Zhang, and Xuanjing Huang. 2016. Incorporate group information to enhance network embedding. In *CIKM*. 1901–1904.

[9] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. 2018. A survey on network embedding. *IEEE Transactions on Knowledge and Data Engineering* (2018).

[10] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. 2018. Adversarial Attack on Graph Structured Data. In *ICML*. 1123–1132.

[11] Quanyu Dai, Qiang Li, Jian Tang, and Dan Wang. 2018. Adversarial Network Embedding. In *AAAI*.

[12] Anupam Datta, Shayak Sen, and Yair Zick. 2016. Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems. In *SP*. 598–617.

[13] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *KDD*. ACM, 135–144.

[14] Rui Feng, Yang Yang, Wenjie Hu, Fei Wu, and Yueting Zhuang. 2018. Representation Learning for Scale-free Networks. In *AAAI*.

[15] Tao-yang Fu, Wang-Chien Lee, and Zhen Lei. 2017. HIN2Vec: Explore Metapaths in Heterogeneous Information Networks for Representation Learning. In *CIKM*. ACM, 1797–1806.

[16] Hongchang Gao and Heng Huang. 2018. Deep Attributed Network Embedding.. In *IJCAI*. 3364–3370.

[17] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. 2018. Large-scale learnable graph convolutional networks. In *KDD*. ACM, 1416–1424.

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

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

[20] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584* (2017).

[21] Xiao Huang, Jundong Li, and Xia Hu. 2017. Label Informed Attributed Network Embedding. In *WSDM*.

[22] Hiroyoshi Ito, Takahiro Komamizu, Toshiyuki Amagasa, and Hiroyuki Kitagawa. 2018. Network-Word Embedding for Dynamic Text Attributed Networks. In *Semantic Computing (ICSC), 2018 IEEE 12th International Conference on*. IEEE, 334–339.

[23] Junghwan Kim, Haekyu Park, Ji-Eun Lee, and U Kang. 2018. SIDE: Representation Learning in Signed Directed Networks. In *WWW*. 509–518.

[24] Thomas Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

[25] Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. *arXiv preprint arXiv:1703.04730* (2017).

[26] Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. 2017. Attributed network embedding for learning in a dynamic environment. In *CIKM*. ACM, 387–396.

[27] Lizi Liao, Xiangnan He, Hanwang Zhang, and Tat-Seng Chua. 2018. Attributed social network embedding. *IEEE Transactions on Knowledge and Data Engineering* 30, 12 (2018), 2257–2270.

[28] Jie Liu, Zhicheng He, Lai Wei, and Yalou Huang. 2018. Content to node: Self-translation network embedding. In *KDD*. ACM, 1794–1802.

[29] Ninghao Liu, Xiao Huang, Jundong Li, and Xia Hu. 2018. On Interpretation of Network Embedding via Taxonomy Induction. KDD.

[30] Jianxin Ma, Peng Cui, and Wenwu Zhu. 2018. DepthLGP: Learning Embeddings of Out-of-Sample Nodes in Dynamic Networks. AAAI.

[31] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[32] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*. 3111–3119.

[33] Giang H Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. 2018. Dynamic Network Embeddings: From Random Walks to Temporal Random Walks. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 1085–1092.

[34] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric transitivity preserving graph embedding. In *KDD*. ACM, 1105–1114.

[35] Shirui Pan, Jia Wu, Xingquan Zhu, Chengqi Zhang, and Yang Wang. 2016. Tri-party deep network representation. In *IJCAI*. 1895–1901.

[36] Barak A Pearlmutter. 1994. Fast exact multiplication by the Hessian. *Neural computation* 6, 1 (1994), 147–160.

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

[38] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network Embedding as Matrix Factorization: UnifyingDeepWalk, LINE, PTE, and node2vec. In *WSDM*.

[39] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. 2017. struc2vec: Learning node representations from structural identity. In *KDD*. ACM, 385–394.

[40] Yu Shi, Qi Zhu, Fang Guo, Chao Zhang, and Jiawei Han. 2018. Easing Embedding Learning by Comprehensive Transcription of Heterogeneous Information Networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2190–2199.

[41] Jian Tang, Meng Qu, and Qiaozhu Mei. 2015. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *KDD*. ACM, 1165–1174.

[42] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW*. International World Wide Web Conferences Steering Committee, 1067–1077.

[43] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *KDD*. ACM, 1225–1234.

[44] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. 2018. GraphGAN: Graph Representation Learning with Generative Adversarial Nets. In *AAAI*.

[45] Suhang Wang, Charu Aggarwal, Jiliang Tang, and Huan Liu. 2017. Attributed signed network embedding. In *CIKM*. ACM, 137–146.

[46] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. 2017. Community Preserving Network Embedding. In *AAAI*. 203–209.

[47] Yaojing Wang, Yuan Yao, Hanghang Tong, Feng Xu, and Jian Lu. 2019. A Brief Review of Network Embedding. *Big Data Mining and Analytics* 2, 1 (2019), 35–47.

[48] Linchuan Xu, Xiaokai Wei, Jiannong Cao, and Philip S Yu. 2017. Embedding of Embedding (EOE): Joint Embedding for Coupled Heterogeneous Networks. In *WSDM*. 741–749.

[49] Linchuan Xu, Xiaokai Wei, Jiannong Cao, and Philip S Yu. 2018. On Exploring Semantic Meanings of Links for Embedding Social Networks. In *WWW*. 479–488.

[50] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y Chang. 2015. Network Representation Learning with Rich Text Information.. In *IJCAI*. 2111–2117.

[51] Dejian Yang, Senzhang Wang, Chaozhuo Li, Xiaoming Zhang, and Zhoujun Li. 2017. From Properties to Links: Deep Network Embedding on Incomplete Graphs. In *CIKM*. ACM, 367–376.

[52] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. 2016. Revisiting Semi-Supervised Learning with Graph Embeddings. In *ICML*. 40–48.

[53] Wenchao Yu, Wei Cheng, Charu C Aggarwal, Kai Zhang, Haifeng Chen, and Wei Wang. 2018. Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2672–2681.

[54] Wenchao Yu, Cheng Zheng, Wei Cheng, Charu C Aggarwal, Dongjin Song, Bo Zong, Haifeng Chen, and Wei Wang. 2018. Learning deep network representations with adversarially regularized autoencoders. In *KDD*. ACM, 2663–2671.

[55] Chang Zhou, Yuqiong Liu, Xiaofei Liu, Zhongyi Liu, and Jun Gao. 2017. Scalable Graph Embedding for Asymmetric Proximity. In *AAAI*. 2942–2948.

[56] Lekui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. 2018. Dynamic Network Embedding by Modeling Triadic Closure Process. In *AAAI*.

[57] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial attacks on neural networks for graph data. In *KDD*. ACM, 2847–2856.

[58] Yuan Zuo, Guannan Liu, Hao Lin, Jia Guo, Xiaoqian Hu, and Junjie Wu. 2018. Embedding Temporal Network via Neighborhood Formation. In *KDD*. 2857–2866.