Representation Learning for Large-scale Dynamic Networks

Yanwei Yu[1,2], Huaxiu Yao[1], Hongjian Wang[1], Xianfeng Tang[1], and Zhenhui Li[1]

[1] College of Information and Science Technology, Pennsylvania State University
{yuy174,huaxiuyao,hxw186,xianfeng,jessieli}@ist.psu.edu
[2] School of Computer and Control Engineering, Yantai University
yuyanwei@ytu.edu.cn

Abstract. Representation leaning on networks aims to embed networks into a low-dimensional vector space, which is useful in many tasks such as node classification, network clustering, link prediction and recommendation. In reality, most real-life networks constantly evolve over time with various kinds of changes to the network structure, e.g., creation and deletion of edges. However, existing network embedding methods learn the representation vectors for nodes in a static manner, which are not suitable for dynamic network embedding. In this paper, we propose a dynamic network embedding approach for large-scale networks. The method incrementally updates the embeddings by considering the changes of the network structures and is able to dynamically learn the embedding for networks with millions of nodes within a few seconds. Extensive experimental results on three real large-scale networks demonstrate the efficiency and effectiveness of our proposed methods.

1 Introduction

Networks are ubiquitous in our daily life, such as social networks, communication networks, biological networks, academic networks and the World Wild Web. People have studied many important data mining problems on networks, including network visualization [21], node classification [3], community detection [12], link prediction [20] and recommendation [37]. A typical way to tackle these problems is based on hand-crafted features of networks, which requires a lot of manual efforts on feature engineering and usually is constricted to a specific problem. Network embedding techniques provide an alternative way to learn features automatically. The basic idea of network embedding is to learn the low-dimensional representation of nodes by preserving the network structure. Following the initial ideas in network embedding [2, 31, 10], recent techniques such as DeepWalk [25] and node2vec [13] learn node representation using random walks sampled in the network. A limitation of such random walk based method is the high computational cost. To scale up to large-scale network with millions of nodes, LINE [28] utilizes edge sampling in the network to learn representations that preserve the first-order and the second-order proximities.

However, existing studies mostly focus on static networks. In the real world, the networks could vary over time with creation and deletion of edges [1]. For instance, in social networks, users may add a user as a new friend or unfriend a user who used to be a friend. In co-author network, people build new co-author relationships over time. In co-location graph where edges as two people being within certain distance, people gather and depart dynamically.

Though dynamic networks widely exist, the studies of representation learning on dynamic networks are limited. A naive method is to re-run the embedding methods on the whole network when the network is updated. However, learning network representation is costly, especially for the large-scale network. Re-computing on the whole network with every batch of updates may not be feasible in the real-world setting. Naturally, we ask the question, "can we learn network embedding dynamically in a more efficient way?"

To address these challenges, in this paper, we propose an efficient embedding method, DLNE, for dynamic network embedding. Our intuition is that, we update previous embeddings by considering the changes of the networks, i.e., newly added (or deleted) edges. The method will be much more efficient compared with re-computing on the whole network because the changes in a large network could be relatively small. More specifically, our proposed method is built based on the LINE method [28], which has been shown to be significantly faster than other embedding methods. We use LINE to obtain the initial representations on the current network. With the new batch of updates on the network structure, we update the representations of corresponding affected nodes by optimizing the loss function defined based on first-order and second-order proximities.

Our method is validated on three large-scale real world networks, including social networks and citation networks. We conduct extensive experiments to verify the effectiveness and efficiency of our method by comparing with state-of-the-art methods via a multi-label classification task. The results suggest that DLNE is able to incrementally update the representations for nodes on a dynamic network with millions of edges in time scale of seconds.

To summarize, the major contributions of this paper are as follows:

- We formally study the problem of dynamic large-scale network embedding. To the best of our knowledge, we are the first method to efficiently learn embeddings dynamically on a large-scale network.
- We propose a novel method DLNE to solve the problem of dynamic large-scale network embedding. We design the loss function to learn the updated representations by considering the changes of the network structure.
- We conduct extensive evaluations through a multi-label classification task on three real-world networks. Experimental results demonstrate the effectiveness and efficiency of our proposed method.

The rest of paper is organized as follows. Related work is discussed in Section 2. Then, we formally state our problem definition in Section 3 and the proposed online embedding method is described in Section 4. Experimental results are reported in Section 5. Finally, we conclude the paper in Section 6.

2 Related Work

Dimension reduction or low-dimension graph representation learning have been studied extensively in the literature. Many methods are proposed in various fields, such as multidimensional scaling [10], IsoMap [31], LLE [27], and Laplacian Eigenmaps [2]. Chen et al. [8] propose the network embedding for directed network. They use Markov random walks to measure the locality link structure of directed networks. Following Chen's work and motivated by the success of word2vec technique [23, 22], Perozzi et al. [25] propose DeepWalk for social network embedding. They use a truncated random walk to construct the context of a vertex, then they employ word2vec to learn latent representations for all vertices in social network. Grover and Leskovec [13] further propose node2vec, which improve DeepWalk by enabling a controlled random walk. Tang et al. [28] propose LINE to learn embedding for both undirected and directed large-scale information networks with unweighted or weighted edges, which is particularly designed to preserve both the first-order and second-order proximities. Cao et al. [4] extend LINE to support high-order graph representation learning by capturing different k-step local relational information. Chen and Wang [9] propose an heterogeneous information network embedding that considers local and global semantic among multi-typed entities. In addition, other deep learning based approaches [5, 32] are proposed to enhance the network representation.

Another line of work aims to learn the graph embedding while considering additional information other than graph connectivity. For example, Yang et al. [36] propose a matrix factorization based method to learn network representations that incorporates text feature into network structure. Chen et al. [7] incorporate group information to learn network embedding. Most recently, Huang et al. [14] propose LANE framework for learning representation vectors for attributed networks. They aim to learn better feature representation incorporating label information into network embedding while preserving their correlations. Xu et al. [35] propose Embedding of Embedding (EOE) framework for coupled heterogeneous networks. They incorporate a harmonious embedding matrix to further embed the embeddings that only encode intra-network edges. Wang et al. [34] propose a Modularized Nonnegative Matrix Factorization (MNMF) model to incorporate the community structure into network embedding. Network embedding techniques have attracted more and more attentions in network science community. Li et al. [17] study on how to leverage node order information and annotation data to improve network embedding in a sem-supervised manner. *However, all the approaches mentioned so far only handle static networks. They are not applicable to learn representations on dynamic evolving information networks.*

In practice, many real world networks (e.g., social networks and co-occurrence networks) are dynamic networks whose edges and vertices change over time. In dynamic network analysis, Ning et al. [24] propose an incremental spectral clustering for evolving networks by updating the eigenvalue system continuously. Chen and Tong [6] propose an online approach to track the eigen-functions of adjacency matrix for a dynamic network. Li et al. [19] propose a unsupervised

feature selection for dynamic networks, which leverages the temporal evolution property of dynamic networks to update the feature results incrementally. *However, such work focus on online analysis of network structure change in dynamic networks, while our work aims to online representation learning for dynamic evolving networks.*

Wang et al. [33] propose a graph embedding method to learn the temporal dynamics of urban region graph. *Although they study a dynamic temporal graph, the embedding learning is not conducted in an incremental manner. Instead, they construct the whole evolving network, and learn different embedding for regions at different timestamp simultaneously. In our method, the embedding vectors of networks are updated continuously and efficiently, rather than re-learned from scratch.*

Most recently, Li et al. [18] propose dynamic network embedding for attributed network. They first use an off-line Laplacian Eigenmaps-based model to learn graph embeddings. Then they update the embedding by updating the top eigenvectors and eigenvalues according to the updated graph matrix. Jian et al. [15] propose an online network embedding algorithm for node classification on streaming network. They use same Laplacian Eigenmaps-based model to update embedding representations for newly arrived nodes. *However, the graph factorization-based method only considers the one-hop relationships in adjacency matrix, and it is difficult to scale because of the use of laplacian eigenmaps. Meanwhile, our proposed method considers both local and global network structure and is able to handle large-scale dynamic networks with millions of vertices and edges in an online fashion.*

3 Problem Definition

In this section, we first introduce some concepts used in this paper. Then we define the problem of dynamic network embedding.

Definition 1 (Network) *A network is denoted as $\mathcal{G} = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of vertices, and $E = \{e_{ij}\}$, where $i, j \in \{1, 2, \cdots, n\}$, is the set of edges. Each edge e_{ij} connects two vertices v_i and v_j, and the weight of this edge is w_{ij}.*

In practice, networks can be categorized as unweighted (e.g., social networks) or weighted (e.g., word co-occurrence network) networks. And networks can also be directed (e.g., citation networks) or undirected (e.g., co-author networks) networks. In unweighted network, $w_{ij} = 1$ if e_{ij} exists, while w_{ij} takes continuous values in weighted network. In undirected network $e_{ij} = e_{ji}$ with the same weight, while $e_{ij} \neq e_{ji}$ and $w_{ij} \neq w_{ji}$ in a directed network.

The structure of networks often evolves over time by adding or deleting edges and vertices. Without loss of generaliy, we partition the time dimension into discrete timestamps $t = \{1, 2, \cdots, T\}$ with fixed interval τ. We use $\mathcal{G}_t$ to denote the dynamic evolving network at time t. Correspondingly, the set of edges and vertices are denoted as E_t and V_t.

For simplicity, we track network updates by the addition and deletion of edges, because the addition (deletion) of vertex could be implemented by the addition (deletion) of edges. More specifically, one vertex is deleted when all edges connecting to this vertex are deleted. Also, if a vertex is added to into current network, at least one edge should be created to connect the new vertex with an existing vertex. To this end, within each time interval τ, we use E_a and E_d to denote the set of edges that are added and deleted, respectively.

Network embedding aims to represent each vertex of the networks as a low dimensional vector, and vertices should have similar embedding vectors if they are connected. To achieve such an embedding result, the network structures must be preserved. Next, we formally define the first-order proximity and the second-order proximity to preserve the local and global network structure, respectively.

Definition 2 (First-Order Proximity) *The first-order proximity in a network is defined as the local pairwise proximity between two directly connected vertices. For each pair of vertices u and v, if there exists $e_{uv} \in E$, the weight w_{uv} indicates the first-order proximity between them. Otherwise, the first-order proximity between u and v is* 0.

Definition 3 (Second-Order Proximity) *The second-order proximity between a pair of vertices is defined to account for their neighborhood structure. Let $\mathcal{N}_u = \{v_{u1}, v_{u2}, v_{u3}, \cdots\}$ denote the set of direct neighbors of vertex u. The second-order proximity between u and v is determined by the similarity of two sets $\mathcal{N}_u$ and $\mathcal{N}_v$.*

In this paper, we aim to learn embedding vectors to preserve the first-order proximity and the second-order proximity among vertices, meanwhile the learned embeddings could be updated to account for the temporal dynamics of networks. The formal definition of dynamic network embedding problem is given as follows:

Problem 1 (Dynamic Network Embedding) *Given a dynamic network $\mathcal{G}_t$, the current embeddings Φ_t of all the vertices V_t in $\mathcal{G}_t$, the problem of dynamic network embedding aims to efficiently calculate the embeddings Φ_{t+1} for all vertices in $\mathcal{G}_{t+1}$ from Φ_t. $\Phi : V \to \mathbb{R}^d$ can also be regarded as the mapping function from vertices to d-dimension vector representations.*

4 DLNE: Dynamic Large-scale Network Embedding

In this section, we provide details for our proposed Dynamic Large-scale Network Embedding (DLNE). First, we present the technical details on how to update the embeddings according to the addition and deletion edge sets E_a and E_d. More specifically, we account for first-order and second-order proximity information while update the embedding learning. Finally, we give the algorithm and optimization steps.

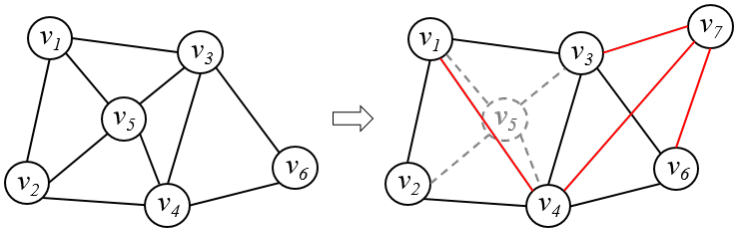


Fig. 1: An example of evolving network: solid red edges denote the added edges E_a, ane dashed grey edges denote the deleted edges E_d.

4.1 Overall Framework.

In Figure 1 we show an example of evolving network, where the network evolves from left to the right. In the given time window, vertex v_5 is deleted, which is equivalent to delete edges $\{e_{15}, e_{25}, e_{35}, e_{45}\}$. Similarly, the addition of v_7 is equivalent to addition of $\{e_{37}, e_{47}, e_{67}\}$. We use Φ_l to denote the embedding function for the graph on the left. The goal of our problem is to calculate Φ_r with the addition edge set E_a and deletion edge set E_d.

Intuitively, the edge embedding is used to account for the local network structure information. Given each edge within the updated edge sets E_a and E_d, the network structure only changes locally. Driven by this intuition, we propose to update the vertex embedding locally. Namely, for each edge $e_{ij} \in E_a \cup E_d$, we only update the embedding vecter $\Phi(v)$, where v is in the local structure of v_i and v_j. The problem becomes how to define local structure, and update the embeddings accordingly. In this paper, we define the local structure with the first-order and the second-order proximity. In the following sections, we give the technical details on how to update the embedding to account for the first-order and the second-order proximity, respectively.

DLNE with First-Order Proximity. First, we consider the first-order proximity when there is an edge added into current network. For each edge $e_{ij} \in E_a$, the probability of the connection between vertex v_i and v_j is formulated as follows:

$$p_1(v_i, v_j) = \sigma(\Phi(v_j)^T \cdot \Phi(v_i)), \tag{1}$$

where $\sigma(x) = 1/(1 + \exp(-x))$ is the sigmoid function and $\Phi(v_i) \in \mathbb{R}^d$ is the embedded vector of vertex v_i in the current timestamp. Namely, the probability of v_j being a neighbor of vertex v_i is correlated to the similarity of their potential representation vectors.

However, the influence of added and deleted edge on the corresponded vertices is different. For each newly added edge $e_{ij} \in E_a$, in order to maintain the influence of both connected and non-connected vertices, we utilize negative edge sampling from the noise distribution $P_n(V)$ to model the influence of

non-connected vertices and then the loss function is:

$$\mathcal{L}_{fa}(v_i, v_j) = -\log\, \sigma(\Phi(v_j)^T \cdot \Phi(v_i)) - \sum_{x=1}^{k} \mathbb{E}_{v_x \sim P_n(V)} \Big[\log\, \sigma(-\Phi(v_x)^T \cdot \Phi(v_i))\Big], \tag{2}$$

where k is the number of sampled non-connected edges. Based on the loss function, we maximize the probability of v_j being a neighbor of vertex v_i and minimize the probability of each negative vertex v_x being a neighbor of vertex v_i. We set $P_n(V) \propto d_v^{3/4}$ as suggested in [23], where d_v is the degree of vertex v.

For each deleted edge $e_{kh} \in E_d$, we use the negative edge sampling shown as Eq. (3) to minimize the probability of vertex v_k being a neighbor of v_h, which reduces the similarity between v_k and v_h in the latent representation.

$$\mathcal{L}_{fd}(v_k, v_h) = -\log(1 - \sigma(\Phi(v_k)^T \cdot \Phi(v_h))). \tag{3}$$

Then, in order to update the embedding vectors of all vertices which correspond to the created and deleted edges, we maximize the joint probability over all evolved vertices. The loss function for preserving the first-order proximity is formulated as follows:

$$\mathcal{L}_1 = \sum_{e_{ij} \in E_a} w_{ij} \mathcal{L}_{fa}(v_i, v_j) + \sum_{e_{kh} \in E_d} w_{kh}\ \mathcal{L}_{fd}(v_k, v_h), \tag{4}$$

where w_{ij} is the weight of edge e_{ij}, representing the importance of edge e_{ij} in constructing the embeddings of v_i and v_j.

DLNE with Second-Order Proximity. Second, the second-order proximity is determined by the similarity of neighbors between two vertices. The intuition is that two vertices are more similar if they share more common neighbors. The second-order proximity has been demonstrated to be a good metric to measure the similarity of a pair of vertices, even if they are not connected [20]. In this work, we employ the conditional probability of "context" v_j linked with vertex v_i [28]:

$$p_2(v_j|v_i) = \frac{exp(\Psi(v_j)^T \cdot \Phi(v_i))}{\sum_{k=1}^{|V|} exp(\Psi(v_k)^T \cdot \Phi(v_i))}, \tag{5}$$

where $|V|$ is the number of neighbors of v_i, and $\Psi(v_j) \in \mathbb{R}^d$ is an auxiliary vector of v_j that needs to be learned when v_j is treated as "context". We can see that $p_2(\cdot|v_i)$ defines a conditional distribution of vertex v_i among its contexts.

Similar to the first-order proximity, the influence of created and deleted edges are different. For each added edge $e_{ij} \in E_a$, we also use the negative edge sampling method to model the influence of the vertices that are not in the

context set of v_i, and then the loss function is defined as:

$$\mathcal{L}_{sa}(v_i, v_j) = -\log\sigma(\Psi(v_j)^T \cdot \Phi(v_i)) - \sum_{x=1}^{k} \mathbb{E}_{v_x \sim P_n(V)} \Big[\log\sigma(-\Psi(v_x)^T \cdot \Phi(v_i))\Big]. \tag{6}$$

The loss function $\mathcal{L}_{sa}(v_i, v_j)$ is to maximize the log-probability of observing the context of each vertex v_i that connect with the created edges.

For each deleted edge $e_{kh} \in E_d$, one negative edge sampling process is used to model the influence of edge e_{kh} and then the loss function $\mathcal{L}_{sd}$ is defined as:

$$\mathcal{L}_{sd}(v_k, v_h) = -\log(1 - \sigma(\Psi(v_h)^T \cdot \Phi(v_k))). \tag{7}$$

By combining the influence of each added and deleted edge, the loss function for preserving the second-order proximity is defined as:

$$\mathcal{L}_2 = \sum_{e_{ij} \in E_a} w_{ij} \mathcal{L}_{sa}(v_i, v_j) + \sum_{e_{kh} \in E_d} w_{kh} \mathcal{L}_{sd}(v_k, v_h). \tag{8}$$

The representation Φ and Ψ can be learned by training the empirical distribution $p_2(v_j|v_i) = \frac{w_{ij}}{\sum_{k \in \mathcal{N}_i} w_{ik}}$ that can be observed in the network, where the denominator is the out-degree of vertex v_i.

Finally, we jointly consider the influence of the first-order proximity and the second-proximity, and the joint loss function is defined as follows:

$$\mathcal{L} = \mathcal{L}_1 + \mathcal{L}_2. \tag{9}$$

4.2 Algorithm and Optimization

The Algorithm 1 is the pseudo-code for our DLNE that preserves both first-order and second-order proximities. We first perform edge sampling in the set of added edges E_a to generate the training vertices that are associated with the sampled edges. In particular, we use negative sampling to implement our loss function (Eq. (4) and Eq. (8)) for each sampled new edge. Then, we process the deleted edges by edge sampling in E_d. The sampled deleted edges is addressed in form of a negative edge in consistent with Eq. (3) for the first-order proximity and Eq. (7) for the second-order proximity. Note that AddedEdgeSample(E_a) and DeletedEdgeSample(E_d) only perform one edge sampling from the set of added edges E_a and the set of deleted edges E_d, respectively. $NEG_k(v_i)$ represents the set of k sampled negative vertices w.r.t v_i. To optimize the loss function, we employ the asynchronous stochastic gradient algorithm (ASGD) [26]. The learning rate η for ASGD is initially set to 0.025 and then decreased linearly with the number of vertices that have been trained.

The gradients in Algorithm 1 for edge $e_{ij} \in E_a$ are calculated as follows:

$$\frac{\partial \mathcal{L}_1}{\partial \Phi(v)} = \begin{cases} -w_{ij}(\sigma(-\Phi(v)^T \Phi(v_i)))\Phi(v_i) & v = v_j \\ w_{ij}(\sigma(\Phi(v)^T \Phi(v_i)))\Phi(v_i) & v \in NEG_k(v_i) \end{cases}$$

Algorithm 1 DLNE: Dynamic Large-scale Network Embedding

Input: Dynamic network $\mathcal{G}_t$; embeddings Φ_t and auxiliary vectors Ψ_t of $\mathcal{G}_t$; network updates E_a and E_d; embedding dimension d; the number of edge sampling s_n; the number of negative sampling k; learning rate η.

Output: Updated embedding vectors Φ_{t+1} of $\mathcal{G}_{t+1}$.

$num_s \leftarrow 0$;
while $(num_s < s_n)$ **do**
  $e_{ij} \leftarrow$ AddedEdgeSample(E_a);
  for all $v \in v_j \cup NEG_k(v_i)$ **do**
    $\Phi(v) \leftarrow \Phi(v) - \eta \frac{\partial \mathcal{L}_1}{\partial \Phi(v)}$;
    $\Psi(v) \leftarrow \Psi(v) - \eta \frac{\partial \mathcal{L}_2}{\partial \Psi(v)}$;
  end for
  $\Phi(v_i) = \Phi(v_i) - \eta(\frac{\partial \mathcal{L}_1}{\partial \Phi(v_i)} + \frac{\partial \mathcal{L}_2}{\partial \Phi(v_i)})$;
  $e_{kh} \leftarrow$ DeletedEdgeSample(E_d);
  $\Phi(v_h) \leftarrow \Phi(v_h) - \eta \frac{\partial \mathcal{L}_1}{\partial \Phi(v_h)}$;
  $\Phi(v_k) \leftarrow \Phi(v_k) - \eta(\frac{\partial \mathcal{L}_1}{\partial \Phi(v_k)} + \frac{\partial \mathcal{L}_2}{\partial \Phi(v_k)})$;
  $\Psi(v_h) \leftarrow \Psi(v_h) - \eta \frac{\partial \mathcal{L}_2}{\partial \Psi(v_h)}$;
  $num_s + +$;
end while

$$\frac{\partial \mathcal{L}_1}{\partial \Phi(v_i)} = -w_{ij}(\sigma(-\Phi(v_j)^T \Phi(v_i)))\Phi(v_j) + w_{ij} \sum_{x=1}^{k} (\sigma(\Phi(v_x)^T \Phi(v_i)))\Phi(v_x)$$

$$\frac{\partial \mathcal{L}_2}{\partial \Psi(v)} = \begin{cases} -w_{ij}(\sigma(-\Psi(v)^T \Phi(v_i)))\Phi(v_i) & v = v_j \\ w_{ij}(\sigma(\Psi(v)^T \Phi(v_i)))\Phi(v_i) & v \in NEG_k(v_i) \end{cases}$$

$$\frac{\partial \mathcal{L}_2}{\partial \Phi(v_i)} = -w_{ij}(\sigma(-\Psi(v_j)^T \Phi(v_i)))\Psi(v_j) + w_{ij} \sum_{x=1}^{k} (\sigma(\Psi(v_x)^T \Phi(v_i)))\Psi(v_x)$$

The gradient for edge $e_{kh} \in E_d$ are calculated as follows:

$$\frac{\partial \mathcal{L}_1}{\partial \Phi(v_h)} = w_{kh}(\sigma(\Phi(v_h)^T \Phi(v_k)))\Phi(v_k)$$

$$\frac{\partial \mathcal{L}_1}{\partial \Phi(v_k)} = w_{kh}(\sigma(\Phi(v_h)^T \Phi(v_k)))\Phi(v_h)$$

$$\frac{\partial \mathcal{L}_2}{\partial \Psi(v_h)} = w_{kh}(\sigma(\Psi(v_h)^T \Phi(v_k)))\Phi(v_k)$$

$$\frac{\partial \mathcal{L}_2}{\partial \Psi(v_k)} = w_{kh}(\sigma(\Psi(v_h)^T \Phi(v_k)))\Phi(v_h)$$

We use the same strategy as proposed in [28] to sample edges with probabilities in proportional to the original edge weights in the created or deleted slides. Intuitively, the edges with large weight would be sampled more times. In this way, the embedding method can support the weighted graph. Specifically, we

use the alias table method [16] to draw an edge sample, which only takes $O(1)$ time. Therefore, we are able to efficiently update the embedding vectors for each vertex in the current window.

It is worthy to mention that if the previous embedding Φ_t and Ψ_t are not available, our method can still apply. In this case, we treat previous graph G_t as an empty graph, and randomly initialize Φ and Ψ. By adding all edges in E_a and runs Algorithm 1, we are able to learn the dynamic network embeddings.

5 Experiments

5.1 Data Description.

We use three real world networks to evaluate our method:

- **YouTube**[3] [30] is a video-sharing website on which users can upload, view, and share videos. Both the user social network and group membership information are included in the dataset. The group is defined by common video genres (e.g. anime and wresting) that the user followed. We use such group information as user labels.
- **DBLP** is an author-paper network[4] [29]. We use the DBLP dataset to construct two citation networks, which are DBLP(Paper) and DBLP(Author). DBLP(Paper) is a directed network, which represents the citation relationships among papers. As a directed weighted network, DBLP(Author) represents the citation relationships among authors, where edge weight is the number of cited papers. The labels of DBLP(Author) and DBLP(Paper) are the research areas of the published papers and authors. We choose 10 research areas in the field of computer science, including AI, computer networks, information security, high-performance computing, software engineering, computer graphics and multimedia, theoretical computer science, human computer interaction and ubiquitous computing, interdisciplinary studies, and database, data mining and information retrieval.

The detailed statistics of these networks are summarized in Table 1. Each network contains at least half million vertices and millions of edges.

We random assign timestamp to edges in YouTuBe dataset due to lack of time information. We rank the edges in the DBLP datasets by the publish time. Without loss of generality, we add the same number of edges E_a and delete the same number of E_d within each time interval.

5.2 Baselines and Evaluation Metrics.

We compare our method with the following baselines:

- **DeepWalk**. This approach learns low-dimensional feature representations for each vertex in the social networks by simulating truncated random walks [25].

[3] Available at http://socialcomputing.asu.edu/pages/datasets
[4] Available at https://aminer.org/citation

Table 1: Statistics of the information networks

Name	YouTube	DBLP(Paper)	DBLP(Author)
$\|V\|$	1,138,499	781,109	524,061
$\|E\|$	2,990,443	4,191,677	20,580,238
Average degree	5.25	10.73	78.54
# labels	47	10	10
# train	31,703	61,257	117,934

- **LINE**. LINE [28] is an approach for large-scale information network embedding. LINE preserves the first-order (LINE(1st)) and the second-order (LINE(2nd)) proximities and supports both weighted and directed networks.
- **node2vec**. node2vec [13] extends DeepWalk by proposing a flexible neighbor selection method for vertices instead of simple random walk.

Note that we did not compare with DANE [18], because DANE employs both network adjacency matrix and node attribute matrix. While it is possible to apply DANE only on network matrix, called DANE-N, it requires to calculate the eigenvalues and thus cannot be applied in the large-scale networks with millions of vertices used in the paper.

To facilitate the comparison between DLNE and baselines, we perform a supervised task – multi-label classification on the embedding results. Specifically, we randomly sample a portion of the labeled vertices as training data, and the rest for testing. We employ a one-vs-rest logistic regression classifier implemented by LibLinear[5] [11]. We repeat this process 10 times, and report the average performance results in terms of *Micro-F1* and *Macro-F1*, which are defined as follows:

$$Micro - F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}, \tag{10}$$

$$Macro - F1 = \frac{\sum_{i=1}^{D} F1(i)}{D}, \tag{11}$$

where D is the number of categories and $F1(i)$ is the *Micro-F1* in the ith category. The *Precision* and *Recall* are calculated on all categories. We evaluate the efficiency of our method on a machine with Core^{TM} i7-6700 (3.4GHz) CPU and 16GB memory. In this experiment, the dimensional d of embedding is set as 128. For each baseline (Deepwalk, LINE and node2vec), we follow the parameter settings used in their original papers.

5.3 Performance Comparison.

Comparison on Incremental Networks. We first evaluate the effectiveness and efficiency of DLNE on multi-label classification compared with baselines in

[5] Available at http://www.csie.ntu.edu.tw/~cjlin/liblinear/

Table 2: Performance of algorithms w.r.t. $\frac{|E_a|}{|E|}$ on incremental networks that only consider newly added edges

		YouTube			DBLP(Paper)			DBLP(Author)		
algorithm	$\frac{\|E_a\|}{\|E\|}$	Mic-F1	Mac-F1	Time(s)	Mic-F1	Mac-F1	Time(s)	Mic-F1	Mac-F1	Time(s)
DeepWalk	1	44.50	35.46	24653	54.41	45.19	16125	61.55	56.72	11997
node2vec	1	44.66	35.94	27756	57.59	46.18	19524	61.91	57.23	15354
LINE	1	45.11	36.21	682	61.16	48.62	340	63.33	59.18	688
DLNE	0.01	41.07	33.07	3.5	52.42	39.23	3.9	56.42	52.89	5.5
	0.05	42.53	34.65	15.5	57.31	45.83	18.5	59.94	55.95	29
	0.1	43.85	35.76	27	60.11	47.73	35	62.08	57.83	61
	0.5	44.58	36.17	125	61.03	48.14	144	62.97	58.76	355
	1	45.35	36.53	255	61.12	48.57	335	63.22	59.13	680

Note: Mic-F1 and Mac-F1 mean Micro-F1 and Macro-F1 scores, and Time(s) denotes runtime (in seconds) of each update. $|E_a|$ is the number of newly added edges at each timestamp.

an incremental environment. Namely, we assume edges are only added into and never deleted from the network.

We simulate different evolving networks by changing the value of $|E_a|/|E|$ from 0.01 to 1, where $|E_a|$ is the number of newly added edges with each time interval and $|E|$ is the number of edges in the dataset. Semantically, $|E_a|/|E|$ defines the evolving speed of a dynamic network. For example, when $|E_a|/|E| = 0.01$, DLNE adds 1% edges of whole network at each timestamp. For the compared baselines, the representation is learned based on the whole network (i.e., $|E_a|/|E| = 1$). Additionally, since the first-order proximity is not applicable on directed graph [28], the evaluation of DLNE on DBLP(Author) and DBLP(Paper) only consider the second-order proximity. On YouTuBe dataset DLNE is able to consider both the first-order and the second-order proximities, and thus LINE(1st+2nd) is compared on YouTuBe dataset only.

We evaluate the final embedding of the dynamic graph with a multi-class classification task. In this task, we choose 20% of the labeled vertices as training, and report the micro-F1 and macro-F1 on the testing data in Table 2. We report the running time for different embedding methods as well.

It is clear that DLNE achieves similar performance compared with other methods, while DLNE is 10-25 times faster than the best baseline. These observations are consistent on all three datasets. Learning the representation on the entire network should achieve the best performance, since more information could be captured, although at the cost of longer running time. We also observe that as $|E_a|/|E|$ increases, the DLNE achieves better prediction performance with longer running time. The reason is that incrementally updating a small portion of newly added edges saves time but loses some information in the process.

Additionally, on each network, we can see that DLNE and LINE run faster than DeepWalk and node2vec. This is because the latter two baselines need to sample and train a huge number of random walks, which is computationally

expensive. The superior performance of DLNE and LINE also indicates that the first-order and second-order proximities effectively captures the local network structure.

Furthermore, on YouTube dataset, although both DLNE and Line consider the first-order and second-order proximities, DLNE achieves better Micro-F1 and Macro-F1 than LINE. The potential reason is that LINE learns the embedding of each vertex by preserving the first-order and second-order proximities separately and then concatenate them. Meanwhile, DLNE accounts for the first-order and second-order proximities by optimizing the joint loss function, which automatically balances the effect of these two proximities.

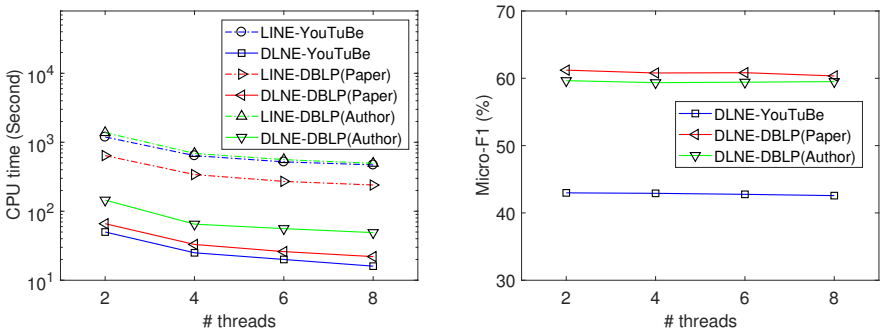
Table 3: Performance of algorithms on dynamic networks that consider both newly added and deleted edges

		YouTube			DBLP(Paper)			DBLP(Author)		
$\frac{\lvert E_0\rvert}{\lvert E\rvert}$	Method	Mic-F1	Mac-F1	Time(s)	Mic-F1	Mac-F1	Time(s)	Mic-F1	Mac-F1	Time(s)
0.9	Deepwalk	41.75	33.51	21687	53.32	44.41	17162	58.41	55.68	14497
	node2vec	41.85	33.85	23980	57.26	45.89	18771	58.26	55.54	17816
	LINE	42.43	35.09	640	60.16	47.72	340	57.95	54.27	688
	DLNE	**42.91**	**35.37**	**25**	**60.79**	**48.01**	**33**	**59.35**	**56.38**	**58**
0.7	Deepwalk	38.71	28.04	15143	45.68	30.49	13645	54.52	50.37	12425
	node2vec	40.15	29.34	16250	45.86	31.56	14350	54.94	50.73	14265
	LINE	41.97	32.76	568	52.66	34.56	274	54.82	50.28	532
	DLNE	**42.70**	**33.85**	**22**	**53.53**	**36.74**	**34**	**56.45**	**52.26**	**50**
0.5	Deepwalk	37.36	27.39	13254	43.62	26.19	12816	52.56	48.36	10797
	node2vec	39.04	28.12	14650	44.94	26.91	13010	53.43	48.45	11971
	LINE	40.12	28.47	410	51.74	32.79	193	51.97	46.62	398
	DLNE	**42.49**	**32.08**	**20**	**52.85**	**34.96**	**32**	**55.06**	**50.20**	**53**

Note: Mic-F1 and Mac-F1 mean Micro-F1 and Macro-F1 scores, and Time(s) denotes running time (in seconds).

Comparison on Dynamic Networks. Next, we evaluate DLNE on a dynamic network, where edges are added and deleted simultaneously. In this experiment, we vary the edge size $|E_0|$ of the initial graph G_0 from $0.5|E|$ to $0.9|E|$, and the numbers of added edges and deleted edges at each timestamp are fixed as $|E_a| = |E_d| = 0.1|E|$. Similar to last experiment, we compare the DLNE output after the last timestamp T with the baseline embeddings of the last snapshot graph G_T.

We report the comparison results in Table 3. The DLNE consistently runs about 6-25 times faster than LINE, because the incremental updating embeddings are time efficient. We also see that DLNE achieves better classification performance than all baselines on both micro-F1 and macro-F1. The reason is that DLNE updates the embeddings of vertices over evolving networks, where the historical network information are also captured. Other baselines only capture the network structure of G_T and overlook the temporal dependency.



(a) time (log scale) (b) Micro-F1

Fig. 2: Performance w.r.t #threads.

Furthermore, we compare the results under different window sizes. Table 3 shows that our proposed method DLNE outperforms other methods consistently with different window size, which demonstrates that our method is robust. In particular, we can see that the performance gap of DLNE over LINE increases (e.g. Macro-F1 gaps are 0.8%, 3.3%, 12.7% on YouTuBe when $|E_0|/|E| = 0.9, 0.7, 0.5$), when the size of initial graph G_0 decreases. Smaller initial graph size means the networks evolves more rounds. When the network evolves more rounds, there are more historical information, which means stronger temporal dependency in the dynamic evolving network. Our DLNE captures more information in evolving networks since it updates the representation incrementally, while other methods only learn the representation on the snapshot. This also demonstrates the superiority of our DLNE since most real-world networks continuously evolve.

5.4 Parallel Computing.

Finally, we evaluate the scalability of our method by running DLNE with different number of threads. The embedding learning parameters are exactly the same as previous experiment. Figure 2 shows the performance comparison w.r.t the number of threads on three datasets. We can see that the speed up of DLNE is stable with the increase of thread number. At the same time, increasing the number of threads does not affect the classification performance, as shown in Figure 2(b). In summary, DLNE exhibits strong parallelism potential in handling large scale network dynamics.

6 Conclusion

This paper proposes an incremental model DLNE to learn the representation of large-scale dynamic networks. DLNE efficiently update the representation of

network in a dynamic environment. The model preserves the first-order and the second-order proximities by optimizing the joint loss function. Extensive evaluations on three real-world networks demonstrate the effectiveness and efficiency of our method. In the future, we plan to explore how to learn the representation on dynamic networks with the changing of the weights of edges. Besides, it is interesting to investigate the efficient deep representation learning for dynamic large-scale networks.

Acknowledgments. This work is partially supported by the National Natural Science Foundation of China under grant Nos. 61773331 and 61403328, the National Science Foundation under grant Nos. 1544455, 1652525, and 1618448, and the China Scholarship Council under grant No. 201608370018.

References

1. Aggarwal, C., Subbian, K.: Evolutionary network analysis: A survey. ACM Computing Surveys (CSUR) 47(1), 10 (2014)
2. Belkin, M., Niyogi, P.: Laplacian eigenmaps and spectral techniques for embedding and clustering. In: Proceedings of NIPS. pp. 585–591 (2002)
3. Bhagat, S., Cormode, G., Muthukrishnan, S.: Node classification in social networks. In: Social network data analytics, pp. 115–148. Springer (2011)
4. Cao, S., Lu, W., Xu, Q.: Grarep: Learning graph representations with global structural information. In: Proceedings of CIKM. pp. 891–900. ACM (2015)
5. Chang, S., Han, W., Tang, J., Qi, G.J., Aggarwal, C.C., Huang, T.S.: Heterogeneous network embedding via deep architectures. In: Proceedings of SIGKDD. pp. 119–128. ACM (2015)
6. Chen, C., Tong, H.: Fast eigen-functions tracking on dynamic graphs. In: Proceedings of SDM. pp. 559–567. SIAM (2015)
7. Chen, J., Zhang, Q., Huang, X.: Incorporate group information to enhance network embedding. In: Proceedings of CIKM. pp. 1901–1904. ACM (2016)
8. Chen, M., Yang, Q., Tang, X.: Directed graph embedding. In: IJCAI. pp. 2707–2712 (2007)
9. Chen, Y., Wang, C.: Hine: Heterogeneous information network embedding. In: Proceedings of DASFAA. pp. 180–195. Springer (2017)
10. Cox, T.F., Cox, M.A.: Multidimensional scaling. CRC press (2000)
11. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: Liblinear: A library for large linear classification. Journal of machine learning research 9(Aug), 1871–1874 (2008)
12. Fortunato, S.: Community detection in graphs. Physics reports 486(3), 75–174 (2010)
13. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: Proceedings of SIGKDD. pp. 855–864. ACM (2016)
14. Huang, X., Li, J., Hu, X.: Label informed attributed network embedding. In: Proceedings of WSDM. pp. 731–739. ACM (2017)
15. Jian, L., Li, J., Liu, H.: Toward online node classification on streaming networks. Data Mining and Knowledge Discovery pp. 1–27 (2017)
16. Li, A.Q., Ahmed, A., Ravi, S., Smola, A.J.: Reducing the sampling complexity of topic models. In: Proceedings of SIGKDD. pp. 891–900. ACM (2014)

17. Li, C., Li, Z., Wang, S., Yang, Y., Zhang, X., Zhou, J.: Semi-supervised network embedding. In: Proceedings of DASFAA. pp. 131–147. Springer (2017)
18. Li, J., Dani, H., Hu, X., Tang, J., Chang, Y., Liu, H.: Attributed network embedding for learning in a dynamic environment. arXiv preprint arXiv:1706.01860 (2017)
19. Li, J., Hu, X., Jian, L., Liu, H.: Toward time-evolving feature selection on dynamic networks. In: Proceedings of ICDM. pp. 1003–1008. IEEE (2016)
20. Liben-Nowell, D., Kleinberg, J.: The link-prediction problem for social networks. Journal of the Association for Information Science and Technology 58(7), 1019–1031 (2007)
21. Maaten, L.v.d., Hinton, G.: Visualizing data using t-sne. Journal of Machine Learning Research 9(Nov), 2579–2605 (2008)
22. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
23. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Proceedings of NIPS. pp. 3111–3119 (2013)
24. Ning, H., Xu, W., Chi, Y., Gong, Y., Huang, T.: Incremental spectral clustering with application to monitoring of evolving blog communities. In: Proceedings of SDM. pp. 261–272. SIAM (2007)
25. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: Proceedings of SIGKDD. pp. 701–710. ACM (2014)
26. Recht, B., Ré, C., Wright, S.J., Niu, F.: Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In: Proceedings of NIPS. pp. 693–701 (2011)
27. Roweis, S.T., Saul, L.K.: Nonlinear dimensionality reduction by locally linear embedding. Science 290(5500), 2323–2326 (2000)
28. Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: Line: Large-scale information network embedding. In: Proceedings of WWW. pp. 1067–1077. ACM (2015)
29. Tang, J., Zhang, J., Yao, L., Li, J., Zhang, L., Su, Z.: Arnetminer: extraction and mining of academic social networks. In: Proceedings of SIGKDD. pp. 990–998. ACM (2008)
30. Tang, L., Liu, H.: Scalable learning of collective behavior based on sparse social dimensions. In: Proceedings of CIKM. pp. 1107–1116. ACM (2009)
31. Tenenbaum, J.B., De Silva, V., Langford, J.C.: A global geometric framework for nonlinear dimensionality reduction. science 290(5500), 2319–2323 (2000)
32. Wang, D., Cui, P., Zhu, W.: Structural deep network embedding. In: Proceedings of SIGKDD. pp. 1225–1234. ACM (2016)
33. Wang, H., Li, Z.: Region representation learning via mobility flow. In: Proceedings of CIKM. ACM (2017)
34. Wang, X., Cui, P., Wang, J., Pei, J., Zhu, W., Yang, S.: Community preserving network embedding. In: Proceedings of AAAI. pp. 203–209 (2017)
35. Xu, L., Wei, X., Cao, J., Yu, P.S.: Embedding of embedding (eoe): Joint embedding for coupled heterogeneous networks. In: Proceedings of WSDM. pp. 741–749. ACM (2017)
36. Yang, C., Liu, Z., Zhao, D., Sun, M., Chang, E.Y.: Network representation learning with rich text information. In: IJCAI. pp. 2111–2117 (2015)
37. Yu, X., Ren, X., Sun, Y., Gu, Q., Sturt, B., Khandelwal, U., Norick, B., Han, J.: Personalized entity recommendation: A heterogeneous information network approach. In: Proceedings of WSDM. pp. 283–292. ACM (2014)