Dynamic Network Embedding : An Extended Approach for Skip-gram based Network Embedding

Lun Du*, Yun Wang*, Guojie Song†, Zhicong Lu, Junshan Wang
Peking University
{dulun, wangyun94, gjsong, phyluzhicong, wangjunshan}@pku.edu.cn

Abstract

Network embedding, as an approach to learn low-dimensional representations of vertices, has been proved extremely useful in many applications. Lots of state-of-the-art network embedding methods based on Skip-gram framework are efficient and effective. However, these methods mainly focus on the static network embedding and cannot naturally generalize to the dynamic environment. In this paper, we propose a stable dynamic embedding framework with high efficiency. It is an extension for the Skip-gram based network embedding methods, which can keep the optimality of the objective in the Skip-gram based methods in theory. Our model can not only generalize to the new vertex representation, but also update the most affected original vertex representations during the evolvement of the network. Multi-class classification on three real-world networks demonstrates that, our model can update the vertex representations efficiently and achieve the performance of retraining simultaneously. Besides, the visualization experimental result illustrates that, our model is capable of avoiding the embedding space drifting.

1 Introduction

Network embedding, as an approach to learn low-dimensional representations of vertices, has been proved extremely useful in many applications [Hamilton *et al.*, 2017]. As the input of machine learning models, the low-dimensional features help to complete specific tasks efficiently, such as vertex classification, clustering, graph visualization, link prediction and social influence analysis [He *et al.*, 2012; Song *et al.*, 2015].

Network embedding can be categorized into the structure-preserving methods and property-preserving methods [Cui *et al.*, 2017]. Our paper belongs to the former. In terms of the static network embedding, some of them are based on matrix factorization [Belkin and Niyogi, 2001]. Some are based on a deep autoencoder [Wang *et al.*, 2016; Cao *et al.*, 2016]. Other important methods [Perozzi *et al.*, 2014; Grover and Leskovec, 2016; Tang *et al.*, 2015; Cao *et al.*, 2015] are inspired by skip-gram in word2vec [Mikolov *et al.*, 2013]. They are more efficient than the former ones, and also have good performance.

However, there is nothing permanent except change. Many real-world networks are not static but are continuously evolving, especially social networks, such as a new user joins the network as an unseen vertex, or two users make friends as a new link in the network. With the evolvement of networks, the representations of vertices become stale and need to be updated to keep freshness. Naive dynamic network embedding methods apply static embedding algorithms to each snapshot of the dynamic networks, which may lead to the following unsatisfactory situations: one is the retrained embedding spaces will drift and it is hard to align. The other is the time complexity of training increases linearly with the number of vertices in networks. Actually, a network may not change much during a short time in dynamic situations, thus the embedding spaces should not change too much, and retraining is not necessary as well.

There are few researches on dynamic network embedding currently [Zhu *et al.*, 2018; Zhang *et al.*, 2017; Ma *et al.*, 2018], which can be categorized into structure preserving and property preserving. Among the former ones, [Tang *et al.*, 2015; Perozzi *et al.*, 2014] are skip-gram based methods designed for static networks, which introduce a way to obtain the vector representations of new vertices briefly. But they can't handle the complicated situations in dynamic settings, e.g. the removal of vertices, the addition and removal of edges, and the update of edge weights. Another type of dynamic network embedding methods is preserving attributes [Li *et al.*, 2017; Hamilton *et al.*, 2017]. For example, SAGE can generalize the vertex representations efficiently with the neighbor feature information based on a general inductive framework. Meanwhile, it can also handle the network only with structural information. Actually, the performance of SAGE is inferior to models based on skip-gram framework in plain network only with structural information.

Considering the outstanding performance of Skip-gram based network embedding (SGNE) methods, we hope to extend the SGNE methods to dynamic setting and keep the optimality of the objective in theory. The dynamic changes of networks are complex, including the addition and removal of

*These authors contributed equally to the work.
†Corresponding Author

vertices and edges, and the update of edge weights. It's challenging to design a framework for all these changes. Secondly, in order to improve efficiency, only the most affected vertices should be updated. Sometimes, the nearest vertices from the new vertex are not always the most affected vertices. So how to measure the influence on the representations of vertices is also a challenge. Thirdly, when the network has great changes, it's challenging to keep the accuracy of network embedding from decreasing obviously.

In this paper, we propose an efficient and stable embedding framework for dynamic networks. It is an extension of the network embedding methods based on skip-gram in a dynamic setting. All SGNE methods are applicable to our framework and achieve the optimal solution of retraining theoretically. In this paper, we apply our framework to extend LINE into dynamic settings as an example. Besides, our model can also be applied to multiple dynamic changes and keep the learning effect when the dynamic network changes a lot.

In details, we divide the dynamic network embedding into two tasks: calculating the representations of new vertices and adjusting the representations of original vertices that are affected greatly. Due to the changes of dynamic network at each time step is small comparing with the network size, we hope to learn new vertices and only update the representations of a part of vertices to improve the efficiency. Therefore, we firstly propose a decomposable objective equivalent to the Skip-gram objective, which can learn the representation of each vertex separately. Secondly, we analyze the influence of dynamic network changes on the original vertex representations quantificationally. A selection mechanism is proposed to choose the original vertices affected greatly and update the representations of them. In addition, through smooth regularization, our model ensures the stability of the embedding results.

To summarize, we make the following contributions:

- We propose Dynamic Network Embedding (DNE), an extended dynamic network embedding framework for Skip-gram based network embedding methods, which can approximately achieve the performance of retraining more efficiently.
- We present a decomposable objective which can learn the representation of each vertex separately. In theory, we quantitatively analyze the degree of impact on original vertices during the network evolvement, and propose a selection mechanism to select the greatly affected vertices to update.
- We conduct extensive experiments on four real networks to validate the effectiveness of our model in vertex classification and network layout. The results demonstrate that DNE can approximately achieve the performance of LINE retraining, and it is about 4 times faster than LINE. Besides, DNE shows strong layout stability.

2 Related Work

Static Network Embedding Network embedding can be categorized into structure-preserving methods [Henderson *et al.*, 2012; Ribeiro *et al.*, 2017] and property-preserving methods [Li *et al.*, 2015; Kipf and Welling, 2016] . Inspired by the skip-gram in word2vec [Mikolov *et al.*, 2013], some approaches represent a node with its nearby nodes. Deepwalk [Perozzi *et al.*, 2014] generalizes the word embedding and employs a truncated random walk to learn latent representations of a network. LINE [Tang *et al.*, 2015] designs an optimized objective function to preserve first-order and second-order proximities to learn network representations. Besides the structure-preserving, many property-preserving works [Li *et al.*, 2017] specialize to design for attributes network. However, all the aforementioned methods are designed for the static network embedding specially.

Dynamic Network Embedding Similar to the static network embedding, the dynamic network embedding can also be categorized into structure-preserving methods and property-preserving methods. Actually, DeepWalk [Perozzi *et al.*, 2014] LINE [Tang *et al.*, 2015] can also be regarded as a structure-preserving dynamic network embedding. The two methods handle new vertices based on static embedding. But the new vertices do not update the original vertices and the relationship among the new vertices will not be preserved into the representations. [Zhou *et al.*, 2018] focuses on mining the pattern of network evolvement to predicts whether there will be a link between two vertices at the next time step. But it can't handle the addition of vertices. In term of property-preserving dynamic network embedding, SAGE [Hamilton *et al.*, 2017] proposes a inductive method to learn the projection between the node features and the node representations. But the parameters in the model are fixed after training, which greatly limits the scalability of the model. Besides, it is proved in the paper that the model is still structure-preserving when dealing with high-dimensional embeddings. But our experiments show that its ability of preserving structure is inferior to our skip-gram based structure-preserving framework.

3 Problem Definition and Analysis

3.1 Problem Definition

Definition 1 (***Dynamic Network***)**.** A dynamic network $\mathcal{G}$ is a sequence of network snapshots within a time interval and evolving over time: $\mathcal{G} = (G_1, ..., G_T)$. Each $G_\tau = (V_\tau, E_\tau) \in \mathcal{G}$ is a weighted and directed network snapshot recorded at time τ, where V_τ is the set of vertices and E_τ is the set of edges within the time interval τ. Each edge $(i, j) \in E_\tau$ is associated with a weight $w_{ij} > 0$. For each $(i, j) \notin E_\tau$, w_{ij} is set to 0.

As undirected networks and unweighted networks are special cases of the network we defined, they are included in our problem definition. In addition, $\Delta G_\tau = (\Delta V_\tau, \Delta E_\tau)$ denotes the change of the whole network, where ΔV_τ and ΔE_τ are the sets of the vertices and edges to be added (or removed) at time τ. According to the definition of dynamic network, we define the dynamic network embedding:

Definition 2 (***Dynamic Network Embedding***)**.** Given a dynamic network $\mathcal{G}$, the dynamic network embedding is a sequence of functions $\Phi = (\Phi_1, ..., \Phi_T)$, where each function

$\Phi_\tau \in \Phi$ is $V_\tau \to \mathbb{R}^d (d \ll \min_\tau |V_\tau|)$ which can preserve the structure property of G_τ.

Since we focus on the vector representations of vertices, the dynamic embedding process can be divided into two parts, the learning of the representations of the new vertices and the adjustment of the original ones. All situations of dynamic changes mentioned above will influence the representations of original vertices, while only the addition of new vertices needs to learn new vectors. In our framework, other changes are considered as a special case of adding vertices. The addition of vertices is introduced as the example of our model, and how to handle other situations will be presented briefly later.

3.2 Analysis

Our method is an extended framework for the Skip-gram network embedding (SGNE) methods in a dynamic setting. We firstly generalize about the main idea of SGNE. Since we use LINE as an example of SGNE to introduce our method, the objective of LINE [Tang *et al.*, 2015] is introduced secondly.

Main Idea of SGNE The SGNE methods are to learn useful vertex representations for predicting the neighbor vertices. Thus, they have the same essential objective that is to maximize the log probability:

$$\sum_{v_i \in V} \sum_{v_j \in N_S(v_i)} \log p(v_j | v_i), \tag{1}$$

where $N_S(v_i)$ is the neighbor set of v_i and $p(v_j|v_i)$ is modeled using Softmax. Different methods have different definitions of neighbour set N_S. Among several SGNE methods, the neighborhood definition of LINE is the most direct. So we use LINE as an example to introduce our model, and we will also present how other methods are applied to our model later.

Objective of LINE In LINE, the neighborhood definition of the vertex v_i is direct, namely, $N_S(v_i) = \{v_j | (i,j) \in E\}$. LINE introduces two definitions of objectives. One of them, LINE with Second-order Proximity (LINE-SP), is based on Skip-gram which we mainly talk about.

LINE-SP learns the vertex representations to preserve the similarity between the neighborhood network structures of vertex pairs. The objective is defined as:

$$\begin{aligned} \max_{\vec{u},\vec{c}} L = \sum_{(i,j) \in E} & w_{ij} (\log \sigma(\vec{c}_j \cdot \vec{u}_i) \\ & + k \cdot \mathbb{E}_{v_n \sim P_n(v)} [\log \sigma(-\vec{c}_n \cdot \vec{u}_i)]), \end{aligned} \tag{2}$$

where, $\sigma(\cdot) = 1/(1 + e^{-x})$ is the sigmoid function, w_{ij} is the weight of the edge (i,j), $\vec{u}_i$ is the representation of v_i when it is treated as a central vertex and $\vec{c}_i$ is the representation of v_i when it is treated as a specific "context" (i.e a specific successor vertex). $P_n(v) \propto d_v^\alpha$ is the noise distribution for negative sampling, where d_v is the out-degree of the vertex v and α is a hyper parameter which is set to $3/4$ in LINE.

In our model, we have two adjustments regarding the d_v^α. To facilitate the theoretical proof, we set $\alpha = 1$, the same as the setting of [Levy and Goldberg, 2014]. In fact, sampling according to $P^{3/4}$ produces somewhat superior results on some of the semantic evaluation tasks [Levy and Goldberg, 2014] and it has not been proved to have a better effect in the network representation. The other one is that we set d_v to the in-degree of v instead of the out-degree. As the meaning of $P_n(v)$ is the probability of a vertex to be sampled as a negative sample when v is a specific successor node, we believe the in-degree is a better choice.

4 Dynamic Network Embedding Model

In this section, we firstly propose a decomposable objective equivalent to the objective of LINE (Eq.(2)), which can be optimized for $\vec{u}_i$, $\vec{c}_i$ on the basis that the representations of most vertices don't need to be adjusted. Based on the objective, we secondly introduce a method to learn the representations of new vertices. Thirdly, we present an embedding adjustment mechanism for original vertices, analyzing the influence of the dynamic changes on original vertices quantitatively. Finally, we discuss the applicability of the framework on other SGNE methods, and to deal with other dynamic changes of the network.

4.1 Decomposable Objective Equivalent to LINE

The objective of LINE-SP Eq.(2) can not decompose into local objective for $\vec{u}_i$ and $\vec{c}_i$ simultaneously. We give the decomposable objective:

$$\begin{aligned} \sum_{(i,j) \in E} w_{ij} \Big(& \log \sigma(\vec{c}_j \cdot \vec{u}_i) + k \big(\mu \cdot \mathbb{E}_{v_n \sim P_{in}(v)} [\log \sigma(-\vec{c}_n \cdot \vec{u}_i)] \\ & + (1-\mu) \cdot \mathbb{E}_{v_n \sim P_{out}(v)} [\log \sigma(-\vec{c}_j \cdot \vec{u}_n)] \big) \Big), \end{aligned} \tag{3}$$

where, μ is an arbitrary real number between 0 and 1. $P_{in}(v) \propto d_v^{(in)}$ and $P_{out}(v) \propto d_v^{(out)}$ are the noise distributions for negative sampling, where $d_i^{(in)}$ is the in-degree of vertex v_i and $d_i^{(out)}$ is the out-degree of vertex v_i, i.e. $d_i^{(in)} = \sum_j w_{ji}$ and $d_i^{(out)} = \sum_j w_{ij}$.

Lemma 1. For any real number $\mu \in [0,1]$, the objective Eq.(3) is equivalent to Eq.(2), i.e.the objective of LINE-SP.

Proof. According to the $w_{ij} = 0$ for any $(i,j) \notin E$, Eq.(2) can be rewritten:

$$\begin{aligned} L = & \sum_{v_i \in V} \sum_{v_j \in V} w_{ij} (\log \sigma(\vec{c}_j \cdot \vec{u}_i)) \\ & + \sum_{v_i \in V} \sum_{v_j \in V} w_{ij} (k \cdot \mathbb{E}_{v_n \sim P_{in}(v)} [\log \sigma(-\vec{c}_n \cdot \vec{u}_i)]) \\ = & \sum_{v_i \in V} \sum_{v_j \in V} w_{ij} (\log \sigma(\vec{c}_j \cdot \vec{u}_i)) \\ & + \sum_{v_i \in V} d_i^{(out)} (k \cdot \mathbb{E}_{v_n \sim P_{in}(v)} [\log \sigma(-\vec{c}_n \cdot \vec{u}_i)]), \end{aligned} \tag{4}$$

where the expectation term can be explicitly expressed:

$$\mathbb{E}_{v_n \sim P_{in}(v)} [\log \sigma(-\vec{c}_n \cdot \vec{u}_i)] = \sum_{v_n \in V} \frac{d_n^{(in)}}{D} \log \sigma(-\vec{c}_n^T \cdot \vec{u}_i),$$

where $D = \sum_{(i,j)\in E} w_{ij}$.

We focus on the latter term of Eq.(4):

$$\begin{aligned}
&\sum_{v_i\in V} d_i^{(out)}(k\cdot\mathbb{E}_{v_n\sim P_{in}(v)}[\log\sigma(-\vec{c}_n\cdot\vec{u}_i)]) \\
&=\sum_{v_i\in V}\sum_{v_j\in V} k\cdot d_i^{(out)}\cdot\frac{d_j^{(in)}}{D}\log\sigma(-\vec{c}_j^{\,T}\cdot\vec{u}_i) \\
&=\sum_{v_j\in V} d_j^{(in)}\left(k\cdot\mathbb{E}_{v_n\sim P_{out}(v)}[\log\sigma(-\vec{c}_j\cdot\vec{u}_n)]\right).
\end{aligned} \tag{5}$$

Combine Eq.(4) and Eq.(5):

$$\begin{aligned}
\ell=&\sum_{v_i\in V}\sum_{v_j\in V} w_{ij}(\log\sigma(\vec{c}_j\cdot\vec{u}_i)) \\
&+\mu\sum_{v_i\in V} d_i^{(out)}(k\cdot\mathbb{E}_{v_n\sim P_{in}(v)}[\log\sigma(-\vec{c}_n\cdot\vec{u}_i)]) \\
&+(1-\mu)\sum_{v_j\in V} d_j^{(in)}\left(k\cdot\mathbb{E}_{v_n\sim P_{out}(v)}[\log\sigma(-\vec{c}_j\cdot\vec{u}_n)]\right) \\
=&\sum_{(i,j)\in E} w_{ij}\Big(\log\sigma(\vec{c}_j\cdot\vec{u}_i)+k\big(\mu\cdot\mathbb{E}_{v_n\sim P_{in}(v)}[\log\sigma(-\vec{c}_n\cdot\vec{u}_i)] \\
&+(1-\mu)\cdot\mathbb{E}_{v_n\sim P_{out}(v)}[\log\sigma(-\vec{c}_j\cdot\vec{u}_n)]\big)\Big).
\end{aligned}$$

Thus, the lemma has been proved. □

The objective can decompose on $\vec{w}$ when μ is set to 1:

$$\max_{\vec{u}_i}\sum_{v_j\in N_{out}(v_i)} w_{ij}(\log\sigma(\vec{c}_j\cdot\vec{u}_i)+k\mathbb{E}_{v_n\sim P_{in}(v)}[\log\sigma(-\vec{c}_n\cdot\vec{u}_i)]),$$

and the objective can decompose on $\vec{c}$, when μ is set to 0:

$$\max_{\vec{c}_i}\sum_{v_j\in N_{in}(v_i)} w_{ji}(\log\sigma(\vec{c}_i\cdot\vec{u}_j)+k\mathbb{E}_{v_n\sim P_{out}(v)}[\log\sigma(-\vec{c}_i\cdot\vec{u}_n)]).$$

[Levy and Goldberg, 2014] derives the theoretical optimal solution of Skip-gram with negative sampling. Based on the work, we give the theoretical optimal solution of LINE-SP:

$$x_{ij}=\vec{c}_j\cdot\vec{u}_i=\log(\frac{w_{ij}\cdot D}{d_i^{(out)}\cdot d_j^{(in)}})-\log k. \tag{6}$$

It shows that the optimal solution is for the inner product $\vec{c}_j\cdot\vec{u}_i$ of the vertex pairs rather than every specific $\vec{u}_i$ and $\vec{c}_j$. Thus, with a fixed $\vec{u}_i$ (or $\vec{c}_j$), we can optimize the objective only for $\vec{c}_j$ (or $\vec{u}_i$), which makes $\vec{c}_j\cdot\vec{u}_i$ tend to x_{ij}.

4.2 New Vertex Representation

We discuss the situation of adding several vertices at time τ. For the new vertices, their edges can be categorized into three types. For any edges $(i,j)\in\Delta E_\tau$: $\Delta E_\tau^{(1)}=\{(i,j)|v_i\in\Delta V_\tau\wedge v_j\in\Delta V_\tau\}, \Delta E_\tau^{(2)}=\{(i,j)|v_i\in\Delta V_\tau\wedge v_j\notin\Delta V_\tau\}, \Delta E_\tau^{(3)}=\{(i,j)|v_i\notin\Delta V_\tau\wedge v_j\in\Delta V_\tau\}$. They correspond to different values of μ in the objective: $\frac{1}{2}$, 1 or 0. Thus, the objective for new vertices is defined as:

$$\min_{\vec{u}^{(\tau)},\vec{c}^{(\tau)}}\ell=\ell_1+\ell_2+\ell_3, \tag{7}$$

where,

$$\begin{aligned}
\ell_1=&\sum_{(i,j)\in\Delta E_\tau^{(1)}}-w_{ij}(\log\sigma(\vec{c}_j^{(\tau)}\cdot\vec{u}_i^{(\tau)})+\frac{k}{2}\mathcal{F}_{in}(\vec{u}_i^{(\tau)})+\frac{k}{2}\mathcal{F}_{out}(\vec{c}_j^{(\tau)})), \\
\ell_2=&\sum_{(i,j)\in\Delta E_\tau^{(2)}}-w_{ij}(\log\sigma(\vec{c}_j^{(\tau-1)}\cdot\vec{u}_i^{(\tau)})+k\cdot\mathcal{F}_{in}(\vec{u}_i^{(\tau)})), \\
\ell_3=&\sum_{(i,j)\in\Delta E_\tau^{(3)}}-w_{ij}(\log\sigma(\vec{c}_j^{(\tau)}\cdot\vec{u}_i^{(\tau-1)})+k\cdot\mathcal{F}_{out}(\vec{c}_j^{(\tau)})),
\end{aligned}$$

$$\begin{aligned}
\mathcal{F}_{in}(\vec{u}_i^{(\tau)})&=\mathbb{E}_{v_n\sim P_{in}(v)}[\log\sigma(-\vec{c}_n^{(\tau_n)}\cdot\vec{u}_i^{(\tau)})], \\
\mathcal{F}_{out}(\vec{c}_j^{(\tau)})&=\mathbb{E}_{v_n\sim P_{out}(v)}[\log\sigma(-\vec{c}_j^{(\tau)}\cdot\vec{u}_n^{(\tau_n)})],
\end{aligned}$$

where $\vec{u}_i^{(\tau)},\vec{c}_i^{(\tau)}$ is the vector representations of v_i at time τ. τ_n equals to τ when $v_n\in\Delta V_\tau$, otherwise τ_n equals to $\tau-1$. It shows that the objective only learns the vector representations of the new vertices, but doesn't adjust the original ones.

4.3 Adjustment of Original Vertex Representation

Generally, for a dynamic change in the network, a few vertices will be influenced. Therefore, in order to improve the efficiency, we don't have to adjust the representations of all vertices. We only adjust the vertices influenced greatly. We analyze theoretically the changes of the optimal solution for the original vertex representations when the network changes dynamically.

Based on the Eq.(6), we can calculate the delta of the theoretical optimal solution Δx_{ij} between two snapshots. When a new vertex v_* is added at time τ, the delta is

$$\begin{aligned}
&\Delta x_{ij}^{(\tau)}=x_{ij}^{(\tau)}-x_{ij}^{(\tau-1)} \\
=&\log\frac{D+d_*^{(out)}+d_*^{(in)}}{D}+\log\frac{d_i^{(out)}}{d_i^{(out)}+w_{i*}}+\log\frac{d_j^{(in)}}{d_j^{(in)}+w_{*j}}.
\end{aligned}$$

In reality, we are unable to obtain the optimal solution. Therefore, we give the standard to justify whether v_i should be adjusted combing the edge weights:

$$\begin{aligned}
\epsilon_i=\frac{1}{Z_i^{(\tau-1)}}(&\sum_{j\in N_{out}^{(\tau-1)}(v_i)} w_{ij}(x_{ij}^{(\tau)}-\vec{c}_j^{(\tau-1)}\cdot\vec{u}_i^{(\tau-1)}) \\
&+\sum_{j\in N_{in}^{(\tau-1)}(v_i)} w_{ji}(x_{ji}^{(\tau)}-\vec{c}_i^{(\tau-1)}\cdot\vec{u}_j^{(\tau-1)})),
\end{aligned} \tag{8}$$

where $x_{ij}^{(\tau)}$ is calculated according to Eq.(6) and $Z_i^{(\tau-1)}$ is a normalization factor, i.e. $Z_i^{(\tau-1)}=\sum_{j\in N_{out}^{(\tau-1)}(v_i)} w_{ij}+\sum_{j\in N_{in}^{(\tau-1)}(v_i)} w_{ji}$.

We adjust the top m original vertices with the largest ϵ_i. For the original vertices needed to be adjusted, we add them into ΔG_τ with their edges, and adopt the objective Eq.(7) to learn them.To guarantee the stability of original vertex representations, we introduce a smooth regularization term.

The **overall objective** is defined as:

$$\max_{\vec{u}^{(\tau)},\vec{c}^{(\tau)}}\mathcal{L}=\ell+\lambda\ell_{reg}, \tag{9}$$

where,

$$\ell_{reg} = \sum_{v_i \in \Delta V_{origin}} (||\vec{u}_i^{(\tau)} - \vec{u}_i^{(\tau-1)}|| + ||\vec{c}_i^{(\tau)} - \vec{c}_i^{(\tau-1)}||),$$

$$\Delta V_{origin} = \{v \mid v \in \Delta V \wedge v \in V^{(\tau-1)}\}.$$

We adopt Adam optimizer [Rushing *et al.*, 2005] to optimize the objective. The computational complexity of our model is the same as LINE in each batch, if the sizes of mini-batch are the same. But the number of parameters in our model is $O(|\Delta V|)$, while in LINE it's $O(|V|)$. So the convergence speed of our model is faster than LINE.

4.4 Applicability Analysis

Various Dynamic Changes For any type of dynamic change within the interval τ, we have two networks $G_{\tau-1}$ and G_τ. According to the two networks, we first calculate two theoretical optimal solutions $x_{ij}^{\tau-1}$ and x_{ij}^{τ} by Eq.(6). Next, the impact degree ϵ_i of each vertices v_i is obtained by Eq.(8). Finally, we optimize the objective Eq.(9) to adjust the top m original vertices with the largest ϵ_i. These steps above is suitable for all dynamic changes.

Various SGNE Methods Other Skip-gram based network embedding methods can be extended to dynamic setting with the framework of DNE. The main difference among these methods is the definition of the vertex neighborhood $N_S(v)$. Based on $N_S(v)$, we define $E_S = \{(i,j)|v_j \in N_S(v_i)\}$. In order to extend these method with the help of our approach, we replace E in the aforementioned equations (in section 4) with E_S. When new vertices arrive, we first obtain the neighborhoods of the new vertices and the affected ones, and then we optimize the objective Eq.(9) to learn the representations of these vertices.

5 Experiments

In this section, we evaluate our model in the dynamic setting from three aspects: the performance on the multi-class vertices classification, the time cost and the embedding stability. Experimental analyses are presented as follows.

5.1 Experiment Setup

Data Sets We employed the following three real datasets in Facebook social networks [Traud *et al.*, 2012] for vertex classification, which comprises 100 colleges and universities in US. We choose the social networks in Amherst College (2235nodes, 90954edges), Duke University (9895nodes, 506443edges) and Auburn University (18448nodes, 973918edges). Besides, in order to validate the embedding stability of our model, Karate [Zachary, 1977] are applied to visualize the network.

Next, we introduce how to generate a sequence of network snapshots $(G_1, ..., G_T)$ from the original network. The initial network G_1 is a sub-network sampling from the original network. Each time step τ, G_τ can be obtained by adding a fixed size of new vertices and edges to $G_{\tau-1}$

Compared Methods We compared the DNE against the following three network embedding algorithms:

- LINE [Tang *et al.*, 2015]. A static network embedding method which optimizes a carefully designed objective function to preserve the local and global network structures. In dynamic scenarios, as the representative of the static embedding method, LINE can be retrained for new vertices. In this paper, LINE with second-order proximity is employed in experiment.
- LINE for New Vertices (LINE-NV) [Tang *et al.*, 2015]. A simply extended method on LINE for new vertices through updating the embedding of the new vertex and keeping the embeddings of existing vertices.
- GraphSAGE [Hamilton *et al.*, 2017]. A general inductive framework of the dynamic network embedding that leverages vertex feature information (e.g. text attributes) to efficiently generate vertex embeddings for previously unseen data. Besides, GraphSAGE is capable of maintaining modest performance by leveraging graph structure, even with completely random feature inputs.

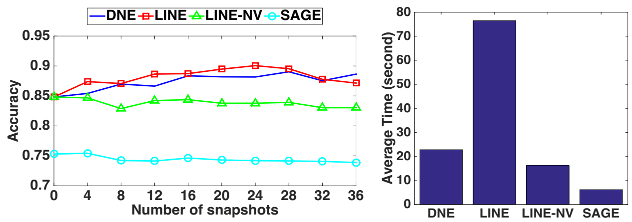
Parameters Settings For fair comparison, the parameters settings of all models in experiments are consistent. The dimensionality of embeddings is set to 20 for classification and 2 for visualization. The mini-batch size of the stochastic gradient descent is set as 300. The learning rate is set as 0.003. There are some special parameters in DNE: the number of original vertices that need to be updated at each time step is set as 10. Other special parameters settings for SAGE follows the recommended settings in the relevant code package[1].

5.2 Vertex Classification

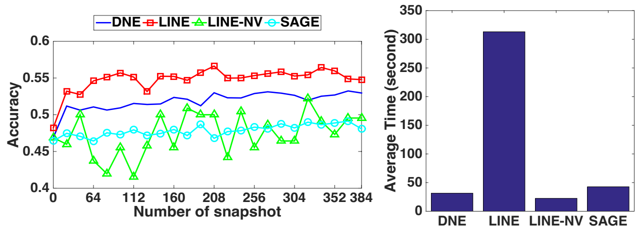
In order to verify the effectiveness of DNE, we employ three real social networks to validate the accuracy of vertex classification and time cost. The learned representations are used to classify the vertices into a set of labels. The classifier we used is Logistic Regression with sklearn package, and the evaluation metric is Accuracy. For all models, the size of network at the first snapshot (i.e. the initial network G_1) is set to 200 nodes for Duke and Auburn, 500 nodes for Amherst. The number of new arriving nodes at each snapshot is set to 200. The dimensionality of embeddings is set to 20. The results are averaged over 10 different runs.

Fig.1 illustrates the effect of number of snapshots on the vertex classification and the average time cost on every snapshot. In terms of accuracy, it can be seen from the Fig.1 that the performance of our model DNE almost keeps up with LINE. Obviously, since LINE-NV does not update the original vertices during the network evolvement, the performance of the model is unstable and inferior to DNE and LINE. Besides, the experiment results illustrate that SAGE is weak in structure preservation. In terms of time cost, since the LINE-NV does not refresh the original vertex embeddings during the evolvement of the network, LINE-NV is the fastest. Compared with LINE, fewer parameters and faster convergence make DNE several times faster than LINE. Compared with

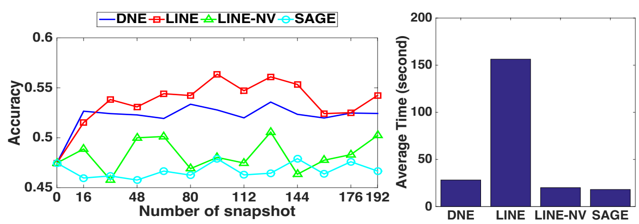
[1]https://github.com/williamleif/GraphSAGE



(a) The comparison of vertex classification accuracy and time cost of different models on the Amherst dataset

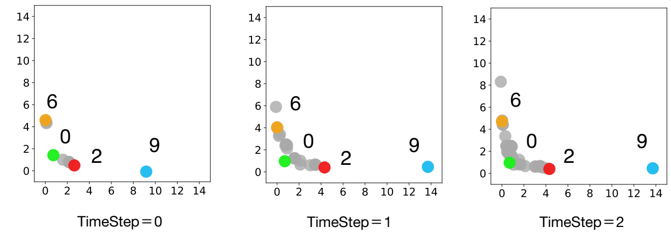


(b) The comparison of vertex classification accuracy and time cost of different models on the Duke dataset

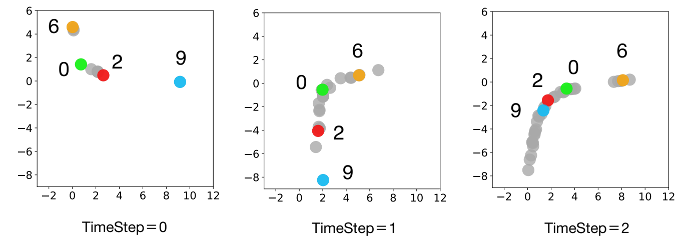


(c) The comparison of vertex classification accuracy and time cost of different models on the Auburn dataset

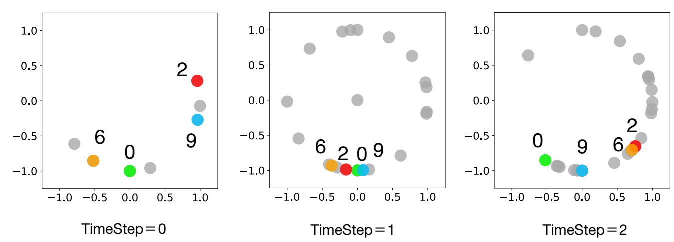
Figure 1: The performance comparison of vertex classification on three real networks. For each sub-graph, the left is the effect of number of snapshots on the vertex classification, and the right is the average time cost on each snapshot.



(a) Karate network layout with DNE



(b) Karate network layout with LINE



(c) Karate network layout with SAGE

Figure 2: 2D visualization on dynamic network. The Karate dataset are embedded into 2-dimensional space at each snapshot. In order to evaluate the layout stability of different model, four vertices (0,2,6,9) in Karate are tracked. According to the changes of the tracked vertex positions, DNE is superior to other model in layout stability.

SAGE, it can be found that the time cost of DNE will be more stable, especially when the network becomes larger and larger. This is because the sampling strategy of SAGE will lead more and more original vertices need to be updated while the network is becoming larger and larger. However, DNE presents an embedding adjustment mechanism for original vertices to adjust the top m original vertices. In general, our model performs better, especially in the large-scale network.

5.3 Network Layouts

Network layout is an important application of network embedding which projects a network into two-dimensional space. Especially, in dynamic scenarios, layout stability is an important and challenging problem. The embedding of networks at consecutive time steps should not differ too much when the networks do not change much.

In this experiment, the Karate data set with 34 nodes is generated into a sequence of three network snapshots $G = (G_1, G_2, G_3)$, where G_1 is initialized as a network with 10 nodes, and 12 nodes are added into G at each snapshot. In order to evaluate the layout stability of models, G are embedded into 2-dimensional space. Besides, we track four point (0, 2, 6, 9), which can be seen in Fig.2, to observe their positions at different snapshots. From the visualization, LINE, as the representative of the static embedding method, appears obvious embedding space drift. Our model DNE is superior to other two models, because just the most affected m nodes will be updated at each snapshot.

6 Conclusion

We propose an efficient and stable embedding framework for dynamic networks, which is an extension of SGNE methods in a dynamic setting. On three real social networks, we validate the effect of our model in vertex classification. We can achieve the approximate performance as retraining and DNE are obviously more efficient. Besides, DNE shows strong layout stability. The idea that we propose to extend the static method in dynamic setting is a general method. A particularly interesting direction for future work is to extend other static network embedding methods in dynamic setting based on our idea.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant No. 61572041).

References

[Belkin and Niyogi, 2001] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. pages 585–591, 2001.

[Cao *et al.*, 2015] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *ACM International on Conference on Information and Knowledge Management*, pages 891–900, 2015.

[Cao *et al.*, 2016] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In *Association for the Advancement of Artificial Intelligence Conference*, pages 1145–1152, 2016.

[Cui *et al.*, 2017] P. Cui, X. Wang, J. Pei, and W. Zhu. A Survey on Network Embedding. *ArXiv e-prints*, November 2017.

[Grover and Leskovec, 2016] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. *Knowledge Discovery and Data mining*, pages 855–864, 2016.

[Hamilton *et al.*, 2017] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1025–1035, 2017.

[He *et al.*, 2012] Xinran He, Guojie Song, Wei Chen, and Qingye Jiang. Influence blocking maximization in social networks under the competitive linear threshold model. pages 463–474, 2012.

[Henderson *et al.*, 2012] Keith Henderson, Brian Gallagher, Tina Eliassi-Rad, Hanghang Tong, Sugato Basu, Leman Akoglu, Danai Koutra, Christos Faloutsos, and Lei Li. Rolx:structural role extraction & mining in large graphs. In *ACM Knowledge Discovery and Data Mining*, pages 1231–1239, 2012.

[Kipf and Welling, 2016] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[Levy and Goldberg, 2014] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pages 2177–2185, 2014.

[Li *et al.*, 2015] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated graph sequence neural networks. *CoRR*, abs/1511.05493, 2015.

[Li *et al.*, 2017] Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. Attributed network embedding for learning in a dynamic environment. *CoRR*, abs/1706.01860, 2017.

[Ma *et al.*, 2018] Jianxin Ma, Peng Cui, and Wenwu Zhu. Depthlgp: Learning embeddings of out-of-sample nodes in dynamic networks. In *Association for the Advancement of Artificial Intelligence Conference*, pages 1–9, 2018.

[Mikolov *et al.*, 2013] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *Neural Information Processing Systems*, pages 3111–3119, 2013.

[Perozzi *et al.*, 2014] Bryan Perozzi, Rami Alrfou, and Steven Skiena. Deepwalk: online learning of social representations. *Knowledge Discovery and Data mining*, pages 701–710, 2014.

[Ribeiro *et al.*, 2017] Leonardo F.R. Ribeiro, Pedro H.P. Saverese, and Daniel R. Figueiredo. Struc2vec: Learning node representations from structural identity. In *ACM Knowledge Discovery and Data Mining*, pages 385–394, 2017.

[Rushing *et al.*, 2005] John Rushing, Udaysankar Nair, Udaysankar Nair, Ron Welch, Ron Welch, and Hong Lin. Adam: a data mining toolkit for scientists and engineers. *Computers and Geosciences*, 31(5):607–618, 2005.

[Song *et al.*, 2015] Guojie Song, Xiabing Zhou, Yu Wang, and Kunqing Xie. Influence maximization on large-scale mobile social network: A divide-and-conquer method. *IEEE Transactions on Parallel and Distributed Systems*, 26(5):1379–1392, 2015.

[Tang *et al.*, 2015] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *International Conference on World Wide Web*, pages 1067–1077, 2015.

[Traud *et al.*, 2012] Amanda L. Traud, Peter J. Mucha, and Mason A. Porter. Social structure of facebook networks. *Social Science Electronic Publishing*, 391(16):4165–4180, 2012.

[Wang *et al.*, 2016] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *ACM Knowledge Discovery and Data Mining*, pages 1225–1234, 2016.

[Zachary, 1977] Wayne W Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 33(4):452–473, 1977.

[Zhang *et al.*, 2017] Ziwei Zhang, Peng Cui, Jian Pei, Xiao Wang, and Wenwu Zhu. Timers: Error-bounded svd restart on dynamic networks. In *Association for the Advancement of Artificial Intelligence Conference*, pages 1–8, 2017.

[Zhou *et al.*, 2018] Lekui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. Dynamic network embedding by modeling triadic closure process. In *Association for the Advancement of Artificial Intelligence Conference*, 2018.

[Zhu *et al.*, 2018] Dingyuan Zhu, Peng Cui, Ziwei Zhang, Jian Pei, and Wenwu Zhu. High-order proximity preserved embedding for dynamic networks. *IEEE Transactions on Knowledge & Data Engineering*, PP(99):1–1, 2018.