SCE: Scalable Network Embedding from Sparsest Cut

Shengzhong Zhang
Fudan University
szzhang17@fudan.edu.cn

Zengfeng Huang*
Fudan University
huangzf@fudan.edu.cn

Haicang Zhou
Fudan University
haicang.zhou@outlook.com

Ziang Zhou
Fudan University
zhouza15@fudan.edu.cn

ABSTRACT

Large-scale network embedding is to learn a latent representation for each node in an unsupervised manner, which captures inherent properties and structural information of the underlying graph. In this field, many popular approaches are influenced by the skip-gram model from natural language processing. Most of them use a contrastive objective to train an encoder which forces the embeddings of similar pairs to be close and embeddings of negative samples to be far. A key of success to such contrastive learning methods is how to draw positive and negative samples. While negative samples that are generated by straightforward random sampling are often satisfying, methods for drawing positive examples remains a hot topic.

In this paper, we propose SCE for unsupervised network embedding only using negative samples for training. Our method is based on a new contrastive objective inspired by the well-known sparsest cut problem. To solve the underlying optimization problem, we introduce a Laplacian smoothing trick, which uses graph convolutional operators as low-pass filters for smoothing node representations. The resulting model consists of a GCN-type structure as the encoder and a simple loss function. Notably, our model does not use positive samples but only negative samples for training, which not only makes the implementation and tuning much easier, but also reduces the training time significantly.

Finally, extensive experimental studies on real world data sets are conducted. The results clearly demonstrate the advantages of our new model in both accuracy and scalability compared to strong baselines such as GraphSAGE, G2G and DGI.

CCS CONCEPTS

• **Computing methodologies** → **Learning latent representations**; *Feature selection*; • **Information systems** → *Data mining*; • **Mathematics of computing** → **Graph theory**.

*Corresponding author

KDD '20, August 23–27, 2020, Virtual Event, CA, USA

ACM ISBN 978-1-4503-7998-4/20/08…$15.00
https://doi.org/10.1145/3394486.3403068

KEYWORDS

Network embedding; Graph neural networks; Graph partition

ACM Reference Format:
Shengzhong Zhang, Zengfeng Huang, Haicang Zhou, and Ziang Zhou. 2020. SCE: Scalable Network Embedding from Sparsest Cut. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3394486.3403068

1 INTRODUCTION

Graph analytics is of central importance in many applications including the analysis of social, biological, and financial networks. Many conventional methods focus on measures and metrics that are carefully designed for graphs, and their efficient computation [26]. Recently, machine learning based approaches have received great amount of attention, e.g., [6, 8, 13, 20, 32, 35–37], which aim to automatically explore and learn network's structural information. One prominent paradigm is network embedding. The idea is to learn a latent representation (a.k.a node embedding) for each node in an unsupervised manner, which captures inherent properties and structural information of the underlying graph. Then various downstream tasks can be solved in the latent embedding space with conventional learning and mining tools.

A successful line of research in network embedding has been inspired by the skip-gram model from the NLP community [25]. Deepwalk [28] and node2vec [13] first generate a set of paths using various random walk models, then treat these paths as if they were sentences and use them to train a skip-gram model. Instead of random walks, LINE [32] directly uses the first and second order proximity between nodes as positive examples. The intuition behind these methods is to make sure nodes "close" in the graph are also close in the embedding space. The key to such methods is how to measure the "closeness" between nodes.

Recently, there has been great successes in generalizing convolutional neural networks (CNNs) to graph data, e.g., [9, 12]. Graph convolutional networks (GCNs) proposed by Kipf and Welling [20] and their variants achieve state-of-the-art results in many tasks, which are considered as better encoder models for graph data. However, most successful methods in this regime use (semi-)supervised training and require lots of labeled data. The performance of GCNs degrades quickly as the train set shrinks [22]. Thus, it is desirable to design appropriate unsupervised objectives that are compatible with convolution based models.

Hamilton et al. [14] suggest to use similar contrastive objectives as in skip-gram to train GCN and its extensions, which is shown to

be empirically effective. A recent work [36] proposes a more complicated objective function based on mutual information estimation and maximization [7, 17], which achieves highly competitive results even compared to state-of-the-art semi-supervised models. This objective is also contrastive, but instead of measuring distance between node pairs, it measures the similarities between single nodes with respect to the representation of the entire graph. Although achieving better performance, the model is more complicated and the computational costs are higher than simpler contrastive objectives, as mutual information estimation and graph-level representations are used in the training process.

1.1 Our Method

In this paper, we provide a new approach called *Sparsest Cut network Embedding* (*SCE*) for unsupervised network embedding based on well-known graph partition problems. The resulting model is simple, effective and easy to train, often outperforming both semi-supervised and unsupervised baselines in terms of accuracy and computation time.

Contributions. First, we propose a novel contrastive-type optimization formulation for network embedding, which draws inspiration from the classic sparsest cut problem. We then propose to use Laplacian smoothing operators (or filters) to simplify the objective function, which aims to eliminate the need for positive examples. As in prior works, a small random sample of node pairs is used as negative examples, but in our framework, we show that this is theoretically justified by spectral graph sparsification theory [31]. We use graph convolutional networks to implement Laplacian smoothing operators, since it has been proved by Wu et al. [38] that graph convolutions behavior much like low-pass filters on graph signals. In a nutshell, the resulting learning model consists of a GCN structure as the encoder and a loss function that only involves negative samples. Finally, extensive empirical studies are conducted, which clearly demonstrate the advantages of our new model in both accuracy and scalability.

Note that we only use negative samples to train the graph convolutional network, which may seem counter-intuitive. The main reason why it works is that the iterative neighborhood aggregation schemes in GCN implicitly forces nearby nodes in the graph to be close in the embedding space. Thus, explicit use of positive examples in the objective function could be redundant. A critical detail in implementation of previous methods is how to draw positive samples. Change of sampling schemes may lead to dramatic drop in performance. Moreover, effective schemes such as random walk based method introduce extra computation burden. On the contrary, there is no need for positive samples in our approach, which not only makes the implementation and tuning much easier, but also reduces the training time significantly.

1.2 Preliminaries

Graph notations. We will consider an undirected graph denoted as $G = (V, E, F)$, where V is the vertex set, E is the edge set, and $F \in \mathbb{R}^{n\times f}$ is the feature matrix (i.e., the i-th row of F is the feature vector of node v_i). Let $n = |V|$ and $m = |E|$ be the number of vertices and edges respectively. We use $A \in \{0,1\}^{n\times n}$ to denote the adjacency matrix of G, i.e., the (i, j)-th entry in A is 1 if and only if their is an edge between v_i and v_j. The degree of a node v_i, denoted as d_i, is the number of edges incident on v_i. The degree matrix D is a diagonal matrix and the its i-th diagonal entry is d_i.

Laplacian matrix. The Laplacian matrix of a graph G is defined as $L_G = D - A$. A useful property of L is that its quadratic form measures the "smoothness" of a vector with respect to the graph structure. More formally, for any vector $x \in \mathbb{R}^n$, it is easy to verify that

$$x^T L_G x = \frac{1}{2}\sum_{i,j} A_{ij}(x_i - x_j)^2 = \sum_{(v_i,v_j)\in E} (x_i - x_j)^2. \quad (1)$$

This can be extended to multi-dimensional cases. For any matrix $X \in \mathbb{R}^{n\times d}$, let X_i be the i-th row of X, then we have

$$\mathrm{Tr}(X^T L_G X) = \frac{1}{2}\sum_{i,j} A_{ij}\|X_i - X_j\|^2 = \sum_{(v_i,v_j)\in E} \|X_i - X_j\|^2. \quad (2)$$

Here $\mathrm{Tr}(\cdot)$ is the trace function, i.e., the sum of the diagonal entries, and $\|\cdot\|$ is the Euclidean norm.

Graph cuts. Let $S \subset V$ be a subset of vertices and $\overline{S} = V \setminus S$ be its complement. We use $E(S, \overline{S})$ to denote the set of crossing edges between S and $\overline{S}$, which are also called *cut edges* induced by S. One task in many applications is to partition the graph into two or more disjoint parts such that the number of crossing edges is minimized. When studying such graph partition problems, it is useful to investigate the laplacian matrix of the graph, because the laplacian quadratic form applied on the indicator vector of S is exactly the cut size. Let $x_S \in \{0,1\}^n$ be the indicator vector of set S, i.e., the i-th entry is:

$$x_{S,i} = \begin{cases} 1, & \text{if } v_i \in S \\ 0, & \text{otherwise} \end{cases}.$$

Then it follows from (1) that

$$x_S^T L_G x_S = \sum_{(v_i,v_j)\in E} (x_{S,i} - x_{S,j})^2 = |E(S, \overline{S})|. \quad (3)$$

2 THEORETICAL MOTIVATION AND ANALYSIS

In this section, we derive our model SCE and discuss its theoretical connections to graph partition problems.

2.1 Sparsest Cut

Our model is motivated by the *Sparsest Cut* problem (see e.g., [4]). The goal in this problem is to partition the graph into two disjoint parts such that the cut size is small; but we also wish the two disjoint parts to be balanced in terms of their sizes. For the standard minimum cut problem, when there are low degree nodes, min-cuts are likely to be formed by a single node, which are not very useful cuts in typical applications. The balance requirement in sparsest cut problems avoids such trivial cuts and could produce more interesting solutions. Next, we define the problem more formally.

For any subset of nodes S, its *edge expansion* is defined as

$$\phi(S) = \frac{|E(S, \overline{S})|}{\min(|S|, |\overline{S}|)}.$$

The sparsest cut problem asks to find a set S^* with smallest possible edge expansion. We define $\phi(G) = \min_{S \subset V} \frac{|E(S,\bar{S})|}{\min(|S|,|\bar{S}|)}$. In this paper, we consider a slight variant of the above definition, which is to find a set S^* with smallest $\phi'(S^*)$, where

$$\phi'(S) = \frac{|E(S,\bar{S})|}{|S||\bar{S}|}.$$

Similarly, we define $\phi'(G) = \min_{S \subset V} \frac{|E(S,\bar{S})|}{|S||\bar{S}|}$.

The above two sparsest cut formulations are equivalent up to an approximation factor of 2 because

$$\frac{|S||\bar{S}|}{n} \le \min(|S|,|\bar{S}|) \le \frac{2|S||\bar{S}|}{n}.$$

Unfortunately, both of the above two formulations are NP-hard [24]. Hence most researches focus on designing efficient approximation algorithms. Currently, algorithms based on *SDP relaxations* achieve the best theoretical guarantee in terms of approximation ratio [4].

2.2 A Parameterized Relaxation

Another way to represent edge expansion is

$$\phi'(S) = \frac{2x_S^\top L_G x_S}{\sum_{i=1}^n \sum_{j=1}^n (x_{S,i} - x_{S,j})^2}.$$

Indeed, the numerator $x_S^\top L_G x_S = |E(S,\bar{S})|$ by (3). From (1), the denominator is exactly $2x_S^\top L_K x_S$, where K is the *complete graph* defined on the same vertex set as G and L_K is the corresponding Lapalacian matrix. This is twice the cut size of $(S,\bar{S})$ on the complete graph (by (3)), which is exactly $2|S||\bar{S}|$. Therefore, we have

$$\phi'(G) = \min_{S \in V} \frac{x_S^\top L_G x_S}{x_S^\top L_K x_S} = \min_{x \in \{0,1\}^n} \frac{x^\top L_G x}{x^\top L_K x}.$$

This algebraic formulation is still intractable, mainly due to the integral constraints $x \in \{0,1\}^n$. It is thus natural to relax these and only require each $x_i \in [0,1]$. More powerful relaxations usually lift each x_i to high-dimensions and consider the optimization problem

$$\min_{X \in \mathbb{R}^{n \times d}} \frac{\mathrm{Tr}(X^\top L_G X)}{\mathrm{Tr}(X^\top L_K X)},$$

or equivalently (by (2))

$$\min_{X \in \mathbb{R}^{n \times d}} \frac{\sum_{(v_i,v_j) \in E} \|X_i - X_j\|^2}{\sum_{i=1}^n \sum_{j=1}^n \|X_i - X_j\|^2}.$$

Here X_i can be viewed as a d-dimensional embedding of node v_i.

The drawback of the above relaxation is that it doesn't utilize node features, which usually contains important information. In this paper, we propose the following relaxation:

$$\min_{\theta} \frac{\sum_{(v_i,v_j) \in E} \|g_\theta(F_i) - g_\theta(F_j)\|^2}{\sum_{i=1}^n \sum_{j=1}^n \|g_\theta(F_i) - g_\theta(F_j)\|^2}. \tag{4}$$

Here $g_\theta(\cdot) : \mathbb{R}^f \to \mathbb{R}^d$ is a parameterized function mapping feature vectors to d-dimensional embeddings, and F_i is the feature of node v_i. The goal is to find optimal parameters θ such that the objective function is minimized.

2.3 Approximation with Graph Convolutional Networks

The optimization problem (4) is highly nonconvex, especially when $g_\theta()$ is modeled by neural networks, and thus could be very difficult to optimize. In this section, we provide effective heuristics based on Graph Convolution operations [20] to facilitate the optimization.

The problem (4) can be viewed as a contrastive game between two players: the denominator wants to maximize pair-wise distances between all pairs, while the numerator tries to make neighboring pairs close. To simplify the problem, we model the behavior of the numerator player by a Laplacian smoothing filter and then remove the numerator from the objective function. Let Π_G be a smoothing matrix. For a signal x, the value of $x^\top L_G x$ become smaller after applying a smoothing operater on it. Thus, the objective of the numerator player is implicitly encoded in Π_G and will be removed from (4), which greatly eases the optimization process.

This trick can also be motivated by stochastic optimization. To minimize objective (4), the algorithm randomly sample some positive examples and negative examples from E and $V \times V$ respectively in each round, then performs mini-batch update via gradient descent. Consider a step of gradient updates from positive samples. This essentially reduces the value of the Laplacian quadratic form, hence performing smoothing. Instead of perform gradient updates, our method directly applies a low-pass filter to smooth the signal.

Let $g_\theta(F) \in \mathbb{R}^{n \times d}$ denote the matrix whose i-th row is $g_\theta(F_i)$, and $\Pi_G g_\theta(F) \in \mathbb{R}^{n \times d}$ be the matrix after smoothing, which contains the embeddings of all nodes. Let z_i denote the embedding of v_i, which is the i-th row in $\Pi_G g_\theta(F)$, and $Z = \Pi_G g_\theta(F)$ be the output embedding matrix. Our loss will be of the form

$$L = \frac{2}{\sum_{i=1}^n \sum_{j=1}^n \|z_i - z_j\|^2} = \frac{1}{\mathrm{Tr}(Z^\top L_K Z)}. \tag{5}$$

It is observed in [22] that the graph convolution operation from [20] is a special form of Laplacian smoothing [33]. Moreover, it is proved in [38] that the effect of iteratively applying this graph convolution is similar to a low-pass-type filter, which projects signals onto the space spanned by low eigenvectors approximately. This is exactly what we need for Π_G. Therefore, in our model, we implement Π_G as a multilayer graph convolution network. Since we will use a multilayer linear network to model $g_\theta(\cdot)$, our network structure, i.e. $\Pi_G g_\theta(F)$, is similar to SGC from [38]. See section 3 for the details.

2.4 Negative Sampling and Spectral Sparsification

One disadvantage of the above loss function is that it contains n^2 terms, and thus too time consuming even just to make one pass over them. Thus, we will only randomly sample a small set of pairs $\mathcal{N} \subset V \times V$ in the beginning, which are called *negative samples*. We use $H = (V, \mathcal{N})$ to denote the graph with edge set $\mathcal{N}$ and L_H be its Laplacian. Then the loss becomes

$$L' = \frac{1}{\sum_{(i,j) \in \mathcal{N}} \|z_i - z_j\|^2} = \frac{1}{\mathrm{Tr}(Z^\top L_H Z)}. \tag{6}$$

In fact, well-know graph sparsification results show that minimizing L' is almost equivalent to minimizing L. More specifically,

if we sample each possible pair independently with probability p, then the spectral sparsification theorem from [31] claims that $x^\top L_K x \approx x^\top L_H x/p$ holds for all $x \in \mathbb{R}^n$ simultaneously with high probability provided that the number of sampled edges is $\Theta(n \log n)$ in expectation (or $\Theta(\log n)$ per node). By this result, the two loss functions (5) and (6) are approximately equivalent. Assume Z is the optimal embedding for L', then its loss with respect to L is

$$L(Z) \approx \frac{L'(Z)}{p} \leq \frac{L'(Z^*)}{p} \approx L(Z^*).$$

Here Z^* is the optimal embedding with respect to L and the inequality follows from the optimality of Z for L'. We refer to [31] for the quantitative bounds on graph sparsification.

2.5 Remarks

The sparsest cut problem considered above is usually called uniform sparsest cut, which can be formulated as $\min_{x\in\{0,1\}^n} \frac{x^\top L_G x}{x^\top L_K x}$. A natural generalization is to use different a graph, G', rather than complete graph in the denominator and consider the problem $\min_{x\in\{0,1\}^n} \frac{x^\top L_G x}{x^\top L_{G'} x}$. This more general problem is called non-uniform sparsest cut and many graph partition problems are special cases of this formulation. For instance, if there is only one edge (s, t) in G', then this is equivalent to minimum s-t cut. This objective is also contrastive and each edge in G' can be viewed as a negative example. Although we show in this work that the simplest choice of complete graph has already achieves impressive results, in general, more prior information can be encoded in G' to further improve the prediction accuracy.

The sparsest cut problem only considers bi-partitions. However, it can be extended to multi-partitions and hierarchical partitions by applying a top-down recursive partitioning scheme [10]. It would be interesting to encode such recursive paradigms into network structure.

Our method is inspired from sparsest cut problem, but is not intended to solve it. It is unclear whether graph neural networks could be helpful for solving such graph partition problems.

3 IMPLEMENTATION DETAILS OF OUR MODEL

In this section, we provide more implementation details of SCE, and an extension of the basic model will also be considered.

3.1 Graph Convolutions

The graph convolutional filters used in our implementation is originated from [20]. Let A be the adjacency matrix of the underlying graph, D be its degree matrix, and I be the identity matrix. $\tilde{A}$ is defined as the adjacency matrix of the graph after adding self-loops for each node, i.e., $\tilde{A} = A + I$, and similarly $\tilde{D} = D + I$ is the adjusted degree matrix. The graph convolution proposed in [20] is $\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$. In this work, we will use a asymmetric variant suggested in [14], which is $\tilde{D}^{-1}\tilde{A}$.

3.2 Architecture

The architecture conceptually consists of two components. The first component is a graph convolutional filter used for smoothing, and the second component is a multilayer linear network modeling the parameterized mapping $g_\theta(\cdot)$ described in the previous section. The input is a set of node features represented as a matrix $F \in \mathbb{R}^{n\times f}$, in which n is the number of nodes and f is the dimension of the input feature.

Algorithm 1 Message Passing of SCE

Input: Input feature matrix F; Graph adjacent matrix with self-loop $\tilde{A}$; Linear network weights $\{W^{(i)}\}_{i=0}^{l}$
Output: Output embedding matrix Z

```
Initialize F^(0) = F;
Compute the degree matrix D̃ by D̃_ii = Σ_j Ã_ij;
// Feature Smoothing;
for i = 1, 2, ··· , k do
    F^(i) = D̃^{-1} Ã F^(i−1);
end for
// Linear Network;
Initialize X^(0) = F^(k);
for j = 1, 2, ··· , l do
    X^(j) = X^(j−1) W^(j);
end for
return Z = X^(l);
```

Feature Mapping. We use multilayer linear network to model the feature mapping $g_\theta(\cdot)$, i.e.,

$$g_\theta(F) = FW^{(1)} \cdots W^{(l)},$$

and θ consists of all the parameter matrices $W^{(1)}, \cdots, W^{(l)}$. In terms of expressive power, multilayer linear networks are the same as single layer ones. However, if the network is trained with gradient-based method, it has been proved that the optimization trajectory of deep linear networks could differ significantly. In particular, there is a form of implicit regularization induced from training deep linear networks with gradient-based optimization [3]. In our experiments, we also observe multilayer structures often perform better.

Smoothing Matrix. The basic implementation of the smoothing matrix Π_G in our model contains k simple iterations. The i-th iteration is formulated as

$$F^{(i)} = \tilde{D}^{-1}\tilde{A}F^{(i-1)}, i = 1, 2, ..., k.$$

More compactly, the smoothing operater can be written as

$$\Pi_G F = (\tilde{D}^{-1}\tilde{A})^k F.$$

So the encoder structure in SCE is

$$(\tilde{D}^{-1}\tilde{A})^k FW^{(1)} \cdots W^{(l)}.$$

Detailed illustration can be seen in Algorithm 1. For large data sets, we also propose a mini-batch version to save computation and memory costs, which can be found in Algorithm 2.

MoSCE. To aggregate information, we also use multi-scale graph filters using similar ideas as in [1, 2]. The model with multi-scale filters is called *Multi-order Sparsest Cut network Embedding* (*MoSCE*). A graphical illustration of MoSCE is presented in Figure 1. We compute representations $\{Z^{(i)}\}_{i=1}^{k}$ with different smoothing levels,

Algorithm 2 Message Passing of Mini-batch SCE

Input: Input feature matrix F; Graph adjacent matrix with self-loop $\tilde{A}$; Linear network weights $\{W^{(i)}\}_{i=0}^{l}$; Mini-batch size b;

Output: Output embedding matrix Z

Initialize $F_0 = F$;
Compute the degree matrix $\tilde{D}$ by $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$;
// *Feature Smoothing*;
for $i = 1, 2, \cdots, k$ **do**
    $F^{(i)} = \tilde{D}^{-1}\tilde{A}F^{(i-1)}$;
end for
Sample b rows from $F^{(k)}$ and concatenate them into a matrix Sample($F^{(k)}$)
// *Linear Network*;
Initialize $X_0 = \text{Sample}(F^{(k)})$;
for $j = 1, 2, \cdots, l$ **do**
    $X^{(j)} = X^{(j-1)}W^{(j)}$;
end for
return $Z = X^{(l)}$;

i.e. $Z^{(i)} = (\tilde{D}^{-1}\tilde{A})^i FW^{(1)} \cdots W^{(l_i)}$ and aggregate these smoothed features with concatenation, mean-pooling or max-pooling:

$$Z = aggregate(Z^{(1)}, Z^{(2)}, \cdots, Z^{(k)}).$$

Since $Z^{(1)}, Z^{(2)}, \cdots, Z^{(k)}$ are computed in SCE, the computational cost of MoSCE is roughly the same.

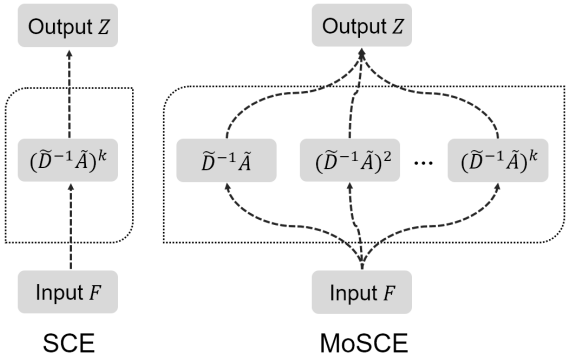


Figure 1: The structure of SCE and MoSCE

3.3 Loss Function

We use θ to denote the set of all trainable parameters. With the *negative samples* denoted as $\mathcal{N} \subset V \times V$, the unsupervised loss used for training is

$$\mathcal{L}_{unsup}(\theta; A, X) = \frac{1}{\sum_{(v_i,v_j)\in\mathcal{N}} \| z_i - z_j \|^2}.$$

Here z_i and z_j are the output embeddings of node v_i and v_j. Each negative pair in $\mathcal{N}$ is randomly sampled from $V \times V$. To suppress overfitting, a L_2 regularizer is introduced in our loss function as $\mathcal{L}_2 = \|\theta\|_2^2$. Thus, the whole loss function of our models is

$$\mathcal{L} = \alpha\mathcal{L}_{unsup} + \beta\mathcal{L}_2,$$

where α and β are hyper-parameters to balance the effect of regularization.

4 RELATED WORK

Many methods have been proposed for learning node representations in the past few years. One major class of methods use random walk to generate a set of paths, then treat these paths as if they were sentences and use them to train a skip-gram model. These class includes Deepwalk [28], LINE [32], node2vec [13], VERSE [34] and many others. Other class is based on matrix factorization [27, 42]. These methods first (approximately) compute a proximity matrix $M \in \mathbb{R}^{n\times n}$, which represents the pair-wise node similarities, and then use singular value decomposition (SVD) or eigen-decomposition to obtain the desired embeddings. For instance, personalized page rank was used as the proximity [41, 43]. See [34] for more choices of proximity. Recently, it has been shown that many random walk based approaches can be unified into the matrix factorization framework with closed forms [29].

Other approaches are also widely studied. SDNE [37] uses an Autoencoder structure. A recent work of [8] proposes Graph2Gauss. This method embeds each node as a Gaussian distribution according to a novel ranking similarity based on the shortest path distances between nodes. With the introduction of graph neural networks, GNN based embedding methods are also widely studied [14, 19, 36], which use unsupervised contrastive-type objective functions.

A critical building of our framework is Laplacian smoothing filters. In our implementation, we use recent graph convolutional networks [20, 38] for this purpose, since the effect of iteratively applying this graph convolution is similar to a low-pass-type filter, which projects signals onto the space spanned by low eigenvectors approximately [18, 22, 38]. GCNs generalize convolutions to the graph domain, working with a spectral operator depending on the graph structure, which learns hidden layer representations that encode both graph structure and node features simultaneously [9, 11, 16]. Deeper versions of GCN often lead to worse performance [20, 39]. It is believed that this might be caused by over-smoothing, i.e., some critical feature information may be "washed out" via the iterative averaging process (see e.g., [39]). Many approaches are proposed to cope with oversmoothing, e.g., [2, 22, 23, 39], and such structures are potentially better filter for our framework.

Our framework draws inspirations from the sparsest cut problem, which is well-known to be NP-hard [24]. On the other hand, efficient approximation algorithms are known for this problem, most of which are based on Linear Programming or Semi-definite Programming relaxations [5, 21]. Currently, algorithms based on *SDP relaxations* achieve $O(\sqrt{\log n})$ approximation, which is conjectured to be the best possible [5]. The $O(\log n)$-approximation algorithm from [5] has high computational complexity and the time is later improved to $O(n^2\text{polylog}(n))$ in [4].

5 EXPERIMENTS

We evaluate the performance of SCE and MoSCE on both transductive and inductive node classification tasks. We compare our methods with state-of-the-art unsupervised models, while we also list the results of strong supervised methods for reference. For unsupervised models, the learned representations are evaluated by

Table 1: Summary of the datasets used in our experiments, as reported in [8, 15, 40]

Dataset	Nodes	Edges	Features	Classes
Cora	2,708	5,429	1,433	7
Citeseer	3,327	4,732	3,703	6
Pubmed	19,717	44,338	500	3
Cora Full	19,793	65,311	8,710	70
Flicker	89,250	899,756	500	7
Reddit	232,965	11,606,919	602	41

training and testing a logistic regression classifier. We detail the experimental setup and results in following parts.

5.1 Experimental Setup

Datasets. The results are evaluated on four citation networks [8, 20] Cora, Citeseer, Pubmed, Cora Full for transductive task and two community networks [15] Flicker, Reddit for inductive learning. The citation networks contain sparse bag-of-words feature vectors for each document and a list of citation links between documents. We follow previous literature to construct undirected graphs on them. Models are trained to classify each document (node) to a corresponding label class. For inductive learning on community networks, the task on Flickr dataset is to categorize images into difference types based on their descriptions and properties, while the task on Reddit is to predict which community each post belongs to. See Table 1 for a concise summary of the six datasets.

Metrics. In the literature, models are often tested on fixed data splits for transductive tasks. However, such experimental setup may favors the model that overfits the most [30]. In order to get thorough empirical evaluation on transductive tasks, results are averaged over 50 random splits for each dataset and standard deviations are also reported. Moreover, we also test the performances under different label rates. Performances for inductive tasks are tested on relatively larger graphs, so we choose fixed data splits as in previous papers [14, 15] and report the Micro-F1 scores averaged on 10 runs. To evaluate the scalability of different models, we also report their training time.

Baseline models. Baseline models used are list below. We follow the common practice suggested in the original papers for hyperparameter settings.

- Deepwalk [28]: Deepwalk is the representative random walk based method. This method preserves higher-order proximity between nodes by maximizing the probability of the observed random walk.
- GCN-unsupervised: To demonstrate the effectiveness of our loss function, we include an unsupervised version of GCN. This method simply use a GCN as the encoder and is trained with the same unsupervised loss function used in LINE [32], GraphSAGE [14]:

$$L_u = -\log(\sigma(z_u^T z_v)) - Q \cdot E_{v_n \sim P_n} \log(\sigma(-z_u^T z_{v_n})). \quad (7)$$

  Here node v is the neighbor of node u and P_n is a negative sampling distribution of node u.
- GraphSAGE [14]: GraphSAGE use stochastic training techniques. By applying a neighbor sampling strategy, it can process very large graphs. There are several variants with different aggregators proposed in the paper. The loss function for unsupervised training is (7).
- G2G [8]: This method embeds each node as a Gaussian distribution according to a novel ranking similarity based on the shortest path distances between nodes. A distribution embedding naturally captures the uncertainty about the representations.
- DGI [36]: DGI uses GCNs as its encoder and more complicated unsupervised loss for training. The loss function is supposed to maximizes the mutual information between "patch representations and a corresponding summary vector".

Methods listed above are unsupervised models, which are the main competitors. We also compare SCE with some representative semi-supervised models, namely GCN [20], SGC [38], GAT [35], MixHop [2].

General model setup. For all the supervised models, we utilize early stopping strategy on each random split and output their corresponding classification result. For all the unsupervised models, we choose embedding dimension to be 512 on Cora, Citeseer, Cora Full and 256 on Pubmed. After the embeddings of nodes are learned, a classifier is trained by applying logistic regression in the embedding space. For inductive learning, SCE uses 512-dimensional embedding space. Other settings of hyper-parameters for the baseline models are the same as suggested in pervious papers.

Hyperparameter for our models. In the transductive experiments, the detailed hyperparameter settings for Cora, Citeseer, Pubmed, and Cora Full are listed below. For SCE, we use Adam optimizer with learning rates of $[0.001, 0.0001, 0.02, 0.01]$ and a L_2 regularization with weights [5e-4, 1e-3, 5e-4, 5e-4]. The number of training epochs are $[20, 200, 50, 20]$. For MoSCE, we use Adam optimizer with learning rates of $[0.001, 0.0001, 0.02, 0.01]$ and a L_2 regularization with weight [5e-4, 1e-3, 0, 0]. The number of training epochs are $[20, 50, 100, 30]$. We sample 5 negative samples for each nodes on each dataset before training, and the hyperparameter α is set to $[15000, 15000, 50000, 100000]$ respectively. For the multi-scale filter in MoSCE, the number of levels used are 3 for Cora and Pubmed, 2 for Cora Full and Citeseer. As different variants of our model produce almost the same results on inductive tasks, we only list results of the basic version. We train SCE with an Adam optimizer with a learning rate of 0.001 and a L_2 regularization with weight 0.02; the number of training epochs are $[4, 20]$ for Reddit and Flickr respectively.

Implementation details. Our models are implemented with Python3 and PyTorch, while for other baseline methods, we use the public release with settings of hyper-parameters the same as suggested in original papers. Experiments are mostly conducted on a NVIDIA 1080 Ti GPU. However, for inductive learning, because the official code of Deepwalk [28] can only run on cpu and due to the huge memory usage, DGI [36] cannot run on GPU, training time of all the models for inductive learning is tested on Intel(R) Xeon(R) CPU E5-2650 v4 (48 cores).

5.2 Results and Analysis

The numerical results are summarized in Table 2 (Transductive learning) and Table 5 (Inductive learning).

Table 2: Summary of results in terms of mean classification accuracy and standard deviation (in percent) over 50 random splits on different datasets. The size of training set are [5, 20] per class for each dataset respectively. The highest accuracy in each column is highlighted in bold and the top 1 unsupervised are underlined. We group all models into three categories: GNN variants(GCN, GAT, SGC, MixHop), unsupervised embedding methods (DeepWalk, GCN-unsupervised, G2G, DGI) and our models.

Method		Cora		Citeseer		Pubmed		Cora Full	
		5	20	5	20	5	20	5	20
Supervised	GCN	67.5±4.8	79.4±1.6	57.7±4.7	69.4±1.4	65.4±5.2	**77.2**±2.1	49.3±1.8	**61.5**±0.5
	SGC	63.9±5.4	78.3±1.9	59.5±3.4	69.8±1.4	65.8±4.4	76.3±2.3	46.0±2.2	57.7±1.2
	GAT	71.2±3.5	79.6±1.5	54.9±5.0	69.1±1.5	65.5±4.6	75.4±2.3	43.9±1.5	56.9±0.6
	MixHop	67.9±5.7	80.0±1.4	54.5±4.3	67.1±2.0	64.4±5.6	75.7±2.7	47.5±1.5	61.0±0.7
Unsupervised	Deepwalk	60.3 ±4.0	70.5 ±1.9	38.3±2.9	45.6±2.0	60.3 ±5.6	70.8 ±2.6	38.9±1.4	51.1±0.7
	GCN(unsup)	61.3±4.3	74.3±1.6	42.3±3.4	56.8±1.9	60.9±5.7	70.3±2.5	32.7±1.9	45.2±0.9
	G2G	72.7 ±2.0	76.2 ±1.1	60.7±3.5	65.7±1.5	**67.6**±3.9	74.1±2.1	38.9±1.3	49.3±0.5
	DGI	72.9 ±4.0	78.1±1.8	65.7±3.6	**71.1**±1.1	65.3±5.7	73.9±2.3	50.5±1.4	58.4±0.6
Ours	SCE	74.3±2.7	80.2±1.1	65.4±2.9	70.7±1.2	65.0±4.9	75.8±2.2	51.3±1.5	60.6±0.6
	MoSCE	**74.6**±2.9	**80.4**±1.2	**66.1**±2.5	70.8±1.3	64.8±4.6	75.9±2.3	**51.7**±1.4	61.1±0.5

5.2.1 Transductive Learning. In this section we consider transductive learning where the information of the whole dataset is available in the training process.

Comparison Between Unsupervised Embedding Baselines and Our Models. We observe that the performance of SCE and its extension MoSCE are better than other unsupervised models under most experimental settings. In particular, our model outperforms GCN-unsupervised for all four datasets. We also observe in our experiments that GCN-unsupervised is very difficult to train, and in many cases it works better without training. This shows that a carefully designed unsupervised objective is critical to train GCN-type encoders. In addition, our models typically outperform the best baseline DGI by a margin of 1%-2%.

To test the scalability of unsupervised models, we also test the training time for SCE, DGI and G2G, which are listed in Table 3. The training time of our models is orders of magnitude faster than DGI and G2G. For SCE, we use a larger number of training epochs on Citeseer and Pubmed, due to slower convergence on these two datasets. Even so, our training time is still less than 3 seconds, which is roughly 10 times faster than DGI. Among baseline methods, DGI often has both higher accuracy and lower training time in our experiments.

To better investigate the tradeoff between training time and testing accuracy, we record the testing accuracy after each training epoch, then plot the wall clock time vs accuracy curve. The results of SCE and DGI are presented in Figure 2. As we clearly see from the figure, for all the four datasets, SCE converges much faster than DGI does.

Comparison Between semi-supervised GNN Variants and Our Models. The results clearly demonstrate the advantage of our models across four datasets. In particular, our models outperform all GNN variants on Cora, Citeseer and Cora Full. When the number of labels per class is 5, our models outperform GCN by a margin of 10.1% on Cora, 8.4% on Citeseer and 2.4% on Cora Full. Even though SCE is slightly worse than MoSCE, it still surpasses GNN variants in most cases. On the one hand, semi-supervised models are greatly affected by the size of the training set, which is also observed in [22], especially for more sophisticated models such as GAT and MixHop. On the other hand, the accuracy of SCE and MoSCE are not affected as significantly as others when the size of training set become smaller.

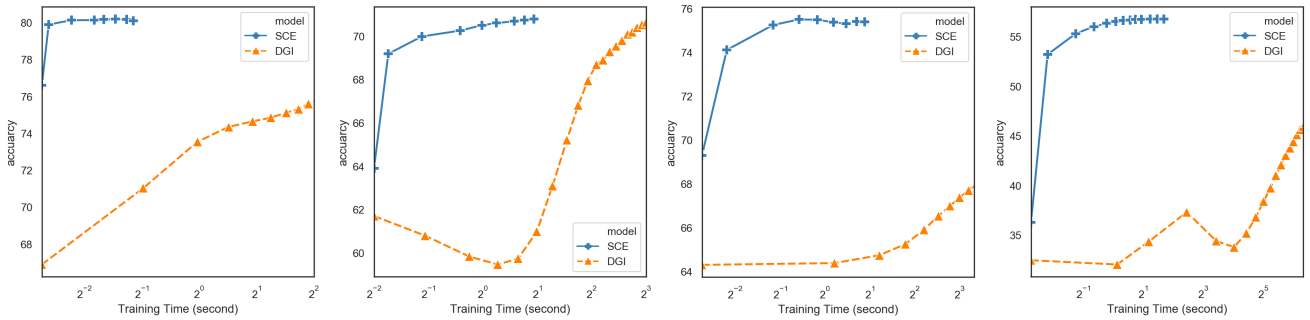
Table 3: The average training time of each model on 10 runs. We train SCE for a fixed number of epochs (20 on Cora, 200 on Citeseer, 100 on Pubmed, 20 on Cora Full). Note that G2G and DGI need to use a large amount of memory. In order to compare fairly on the GPU, we tested the time to learn 128-dimensional vectors on Pubmed and Cora Full.

Method	Cora	Citeseer	Pubmed	Cora Full
G2G	451.5s	89.3s	715.6s	491.1s
DGI	15.7s	16.8s	16.3s	548.2s
SCE	**0.1**s	**1.4**s	**2.3**s	**0.9**s

Effectiveness of our loss function. Here we also study the efficacy of proposed loss function for training graph networks. In our framework, we try to maximize pair-wise distances for negative samples, i.e., $\sum_{(v_i,v_j)\in\mathcal{N}} \| z_i - z_j \|^2$. Our loss function is the inverse of this. Another natural idea is to minimize the negative of these pair-wise distances between negative pairs. So we compare our loss function against the negative sum of Euclidean distances loss:

$$\mathcal{L}_{unsup}(\theta; A, X) = -\sum_{(v_i,v_j)\in\mathcal{N}} \| z_i - z_j \|^2$$

We also test the model accuracy when it is not trained (simply uses randomly initialized parameters). For these experiments, the encoder structures are the same as in SCE . Table 4 show the result of different loss functions. Our loss function has a large improvement over both the untrained model and the model trained by the negative Euclidean distance loss. Comparing the negative sum of Euclidean distance loss function with the untrained model, training



(a) Cora (b) Citeseer (c) Pubmed (d) Cora Full

Figure 2: Test average accuracies on random splits in one training process.

Table 4: The performance of different loss functions.

Method	Cora	Citeseer	Pubmed	Cora Full
SCE (no loss)	76.6±1.3	63.9±1.6	69.4±2.1	51.7±0.8
SCE (negative)	79.0±1.5	67.9±1.6	65.4±3.3	57.6±0.6
SCE (ours)	**80.2**±1.1	**70.7**±1.2	**75.8**±2.2	**60.6**±0.6

usually leads to better performance. These results demonstrate the effectiveness of our loss function.

5.2.2 Inductive Learning. Next we present the results of inductive learning. In this setting, models have no access to test set. The model with higher ability on generalization will perform better on this task.

Accuracy. Experimental results can be found in Table 5, where we only list training time for unsupervised models. It's obvious that our model gets a significant performance gain (1% - 5% in Micro-F1 scores), producing a score very close to supervised methods [1]. Meanwhile, it is also much faster (around 1-2 orders of magnitude) than all the previous unsupervised methods. As stated before, one of the most impressive attributes of our method is its fast training time with limited memory usage.

DGI uses large amount of memory for processing large graphs, so we were not be able to test the performance of DGI on Reddit under our system condition. The Micro-F1 score of 94.0 is reported in the original paper [36].

Scalability. Previous unsupervised methods like GraphSAGE and DGI use GCN as their building block, and in order to scale to large graph, they need neighbor sampling strategies under limited GPU memory. Under this space-time trade-off strategy, several redundant forward and backward propagations are conducted on each single node during each epoch, which results in their very long training time.

However, using a simplified convolution layer as in SGC [38], our method does not need message aggregation during training, which tremendously reduces the memory usage and training time. For graphs with more than 100 thousands nodes and 10 millions edges (Reddit), our model can also run smoothly on one NVIDIA 1080 Ti GPU. For even larger graph datasets, stochastic batch training methods are inevitable. For these setting, the simplified convolution layer is also friendly to mini-batch training and does not need complicated subsampling strategy as needed for other models.

[1] On large graph dataset like Reddit and Flickr more than 60% data is in training set, which means supervised method can get much more information than unsupervised methods.

Table 5: Summary of results in terms of F1 score and runtime on Reddit and Flicker. Because GraphSAGE-LSTM and DGI use too much memory and runtime during training, we cannot reproduce GraphSAGE-LSTM and DGI on Reddit dataset.

Method	Reddit		Flickr	
	Micro-F1	Time	Micro-F1	Time
Raw features	58.5	-	46.2	-
DeepWalk + features	69.1	1277.9s	46.1	447.5s
GraphSAGE-GCN	90.8	477.2s	43.9	29.6s
GraphSAGE-mean	89.7	487.6s	47.3	30.5s
GraphSAGE-LSTM	90.7	-	46.5	2784.7s
GraphSAGE-pool	89.2	22328.2s	46.4	469.0s
DGI	94.0	-	44.7	2444.6s
SCE	**94.6**	**13.7s**	**50.6**	**16.0s**

6 CONCLUSION

In this paper, we provide a new approach for unsupervised network embedding based on graph partition problems. The resulting model is simple, fast and outperforms DGI [36] in terms of accuracy and computation time. Our method is based on a novel contrastive objective inspired from the well-known sparsest cut problem. To solve the underlying optimization problem, we introduce a Laplacian smoothing trick, which uses graph convolutional operators as low-pass filters for smoothing node representations. The resulting model consists of a GCN-type structure as the encoder and a simple loss function. Notably, our model does not use positive samples but only negative samples for training, which not only makes the implementation and tuning much easier, but also reduces the training time significantly. Extensive experimental studies on real world data sets clearly demonstrate the advantages of our new model on both accuracy and scalability.

Future work. The sparsest cut problem considered in the paper is called the uniform sparsest cut. A natural generalization is to use different a graph, G', rather than complete graph, as the negative

sample graph and consider the problem $\min_{x\in\{0,1\}^n} \frac{x^T L_G x}{x^T L_{G'} x}$. We show in this work that the simplest choice of complete graph has already achieves impressive results. We believe more prior information can be encoded in G' to further improve the embedding performance. The sparsest cut problem only considers bi-partitions. However, it can be extended to multi-partitions and hierarchical partitions by applying a top-down recursive partitioning scheme [10]. It would be interesting to encode such recursive paradigms into network structure.

7 ACKNOWLEDGMENTS

This work is supported by Shanghai Science and Technology Commission Grant No. 17JC1420200, National Natural Science Foundation of China Grant No. 61802069, Science and Technology Commission of Shanghai Municipality Project Grant No. 19511120700, and by Shanghai Sailing Program Grant No. 18YF1401200.

REFERENCES

[1] Sami Abu-El-Haija, Amol Kapoor, Bryan Perozzi, and Joonseok Lee. 2018. N-gcn: Multi-scale graph convolution for semi-supervised node classification. *arXiv preprint arXiv:1802.08888* (2018).

[2] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. 2019. MixHop: Higher-Order Graph Convolutional Architectures via Sparsified Neighborhood Mixing. In *International Conference on Machine Learning*. 21–29.

[3] Sanjeev Arora, Nadav Cohen, Wei Hu, and Yuping Luo. 2019. Implicit regularization in deep matrix factorization. In *Advances in Neural Information Processing Systems*. 7411–7422.

[4] Sanjeev Arora, Elad Hazan, and Satyen Kale. 2010. $O(\sqrt{\log n})$ Approximation to SPARSEST CUT in $\tilde{O}(n^2)$ Time. *SIAM J. Comput.* 39, 5 (2010), 1748–1771.

[5] Sanjeev Arora, Satish Rao, and Umesh Vazirani. 2009. Expander flows, geometric embeddings and graph partitioning. *Journal of the ACM (JACM)* 56, 2 (2009), 1–37.

[6] James Atwood and Don Towsley. 2016. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems*. 1993–2001.

[7] Mohamed Ishmael Belghazi, Aristide Baratin, Sai Rajeshwar, Sherjil Ozair, Yoshua Bengio, Aaron Courville, and Devon Hjelm. 2018. Mutual Information Neural Estimation. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, Stockholmsmässan, Stockholm Sweden, 531–540.

[8] Aleksandar Bojchevski and Stephan Günnemann. 2018. Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking. In *International Conference on Learning Representations*.

[9] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations*.

[10] Moses Charikar and Vaggos Chatziafratis. 2017. Approximate hierarchical clustering via sparsest cut and spreading metrics. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 841–854.

[11] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*. 3844–3852.

[12] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*. 2224–2232.

[13] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 855–864.

[14] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*. 1024–1034.

[15] Ajitesh Srivastava Rajgopal Kannan Hanqing Zeng, Hongkuan Zhou and Viktor Prasanna. 2020. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *International Conference on Learning Representations*.

[16] Mikael Henaff, Joan Bruna, and Yann LeCun. 2015. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163* (2015).

[17] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. 2018. Learning deep representations by mutual information estimation and maximization. *arXiv preprint arXiv:1808.06670* (2018).

[18] NT Hoang and Takanori Maehara. 2019. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550* (2019).

[19] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).

[20] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*.

[21] Tom Leighton and Satish Rao. 1999. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM (JACM)* 46, 6 (1999), 787–832.

[22] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[23] Sitao Luan, Mingde Zhao, Xiao-Wen Chang, and Doina Precup. 2019. Break the Ceiling: Stronger Multi-scale Deep Graph Convolutional Networks. In *Advances in Neural Information Processing Systems*. 10943–10953.

[24] David W Matula and Farhad Shahrokhi. 1990. Sparsest cuts and bottlenecks in graphs. *Discrete Applied Mathematics* 27, 1-2 (1990), 113–123.

[25] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[26] Mark Newman. 2018. *Networks*. Oxford university press.

[27] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 1105–1114.

[28] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 701–710.

[29] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. 459–467.

[30] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868* (2018).

[31] Daniel A Spielman and Nikhil Srivastava. 2011. Graph sparsification by effective resistances. *SIAM J. Comput.* 40, 6 (2011), 1913–1926.

[32] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1067–1077.

[33] Gabriel Taubin. 1995. A signal processing approach to fair surface design. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. 351–358.

[34] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. 2018. Verse: Versatile graph embeddings from similarity measures. In *Proceedings of the 2018 World Wide Web Conference*. 539–548.

[35] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *International Conference on Learning Representations*.

[36] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2018. Deep graph infomax. In *International Conference on Learning Representations*.

[37] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1225–1234.

[38] Felix Wu, Tianyi Zhang, Amaur Holanda de Souza, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. 2019. Simplifying graph convolutional networks. In *International Conference on Machine Learning*.

[39] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. In *International Conference on Machine Learning*.

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

[41] Yuan Yin and Zhewei Wei. 2019. Scalable graph embeddings via sparse transpose proximities. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1429–1437.

[42] Ziwei Zhang, Peng Cui, Xiao Wang, Jian Pei, Xuanrong Yao, and Wenwu Zhu. 2018. Arbitrary-order proximity preserved network embedding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2778–2786.

[43] Chang Zhou, Yuqiong Liu, Xiaofei Liu, Zhongyi Liu, and Jun Gao. 2017. Scalable graph embedding for asymmetric proximity. In *Thirty-First AAAI Conference on Artificial Intelligence*.