Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and node2vec

ABSTRACT

Since the invention of word2vec [25, 26], the skip-gram model has significantly advanced the research of network embedding, such as the recent emergence of DeepWalk, LINE, PTE, and node2vec approaches. In this work, we show that all of the aforementioned models with negative sampling can be unified into the matrix factorization framework with closed forms. Our analysis and proofs reveal that: (1) DeepWalk [28] empirically produces a low-rank transformation of a network's normalized Laplacian matrix; (2) LINE [35], in theory, is a special case of DeepWalk when the size of vertices' context is set to one; (3) As an extension to LINE, PTE [34] can be viewed as the joint factorization of multiple networks' Laplacians; (4) node2vec [15] is factorizing a matrix related to the stationary distribution and transition probability tensor of a 2nd-order random walk. We further provide the theoretical connections between skip-gram based network embedding algorithms and the theory of graph Laplacian. Finally, we present the NetMF method as well as its approximation algorithm for computing network embedding. Our method offers significant improvements over DeepWalk and LINE (up to 38% relatively) in conventional network mining tasks. This work lays the theoretical foundation for skip-gram based network embedding methods, leading to a better understanding of latent network representations.

KEYWORDS

Network Embedding; Matrix Factorization; Graph Spectral

1 INTRODUCTION

The conventional paradigm of mining and learning with networks usually starts from the explicit exploration of their structural properties [12, 30]. But many of such properties, such as betweenness centrality, triangle count, and modularity, require extensive domain knowledge and expensive computation to handcraft. In light of these issues, as well as the opportunities offered by the recent emergence of representation learning [1], learning latent representations for networks, a.k.a., network embedding, has been recently extensively studied in order to automatically discover and map a network's structural properties into a latent space.

Formally, the problem of network embedding is often formalized as follows: Given an undirected and weighted graph $G=(V, E)$ with V as the node set and E as the edge set, the goal is to learn a function $V \to \mathbb{R}^{|V|\times d}$ that maps each vertex to a d-dimensional ($d \ll |V|$) latent representation that captures the structural properties of G.

ACM Conference 2018, Los Angeles, California, USA

DOI: 10.475/123_4

Table 1: The closed matrix forms that are implicitly approximated and factorized by DeepWalk, LINE, PTE, and node2vec.

Algorithm	Closed Matrix Form
DeepWalk	$\log\left(\mathrm{vol}(G)\left(\frac{1}{T}\sum_{r=1}^{T}(\boldsymbol{D}^{-1}\boldsymbol{A})^r\right)\boldsymbol{D}^{-1}\right)-\log b$
LINE	$\log\left(\mathrm{vol}(G)\boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{D}^{-1}\right)-\log b$
PTE	$\log\left(\begin{bmatrix}\alpha\,\mathrm{vol}(G_{\mathrm{ww}})(\boldsymbol{D}_{\mathrm{row}}^{\mathrm{ww}})^{-1}\boldsymbol{A}_{\mathrm{ww}}(\boldsymbol{D}_{\mathrm{col}}^{\mathrm{ww}})^{-1}\\ \beta\,\mathrm{vol}(G_{\mathrm{dw}})(\boldsymbol{D}_{\mathrm{row}}^{\mathrm{dw}})^{-1}\boldsymbol{A}_{\mathrm{dw}}(\boldsymbol{D}_{\mathrm{col}}^{\mathrm{dw}})^{-1}\\ \gamma\,\mathrm{vol}(G_{\mathrm{lw}})(\boldsymbol{D}_{\mathrm{row}}^{\mathrm{lw}})^{-1}\boldsymbol{A}_{\mathrm{lw}}(\boldsymbol{D}_{\mathrm{col}}^{\mathrm{lw}})^{-1}\end{bmatrix}\right)-\log b$
node2vec	$\log\left(\frac{\frac{1}{2T}\sum_{r=1}^{T}\left(\sum_u \boldsymbol{X}_{w,u}\underline{\boldsymbol{P}}^r_{c,w,u}+\sum_u \boldsymbol{X}_{c,u}\underline{\boldsymbol{P}}^r_{w,c,u}\right)}{(\sum_u \boldsymbol{X}_{w,u})(\sum_u \boldsymbol{X}_{c,u})}\right)-\log b$

See detailed notations in Section 2. Those used in DeepWalk and LINE are briefly introduced below:
$\boldsymbol{A}$: $\boldsymbol{A}\in\mathbb{R}_+^{|V|\times|V|}$ is G's adjacency matrix with $\boldsymbol{A}_{i,j}$ as the edge weight between vertices i and j;
$\boldsymbol{D}_{\mathrm{col}}$: $\boldsymbol{D}_{\mathrm{col}}=\mathrm{diag}(\boldsymbol{A}^\top\boldsymbol{e})$ is the diagonal matrix with column sum of $\boldsymbol{A}$;
$\boldsymbol{D}_{\mathrm{row}}$: $\boldsymbol{D}_{\mathrm{row}}=\mathrm{diag}(\boldsymbol{A}\boldsymbol{e})$ is the diagonal matrix with row sum of $\boldsymbol{A}$;
$\boldsymbol{D}$: For undirected graphs ($\boldsymbol{A}^\top=\boldsymbol{A}$), $\boldsymbol{D}_{\mathrm{col}}=\boldsymbol{D}_{\mathrm{row}}$. For brevity, $\boldsymbol{D}$ represents both $\boldsymbol{D}_{\mathrm{col}}$ & $\boldsymbol{D}_{\mathrm{row}}$. $\boldsymbol{D}=\mathrm{diag}(d_1,\cdots,d_{|V|})$, where d_i represents generalized degree of vertex i;
$\mathrm{vol}(G)$: $\mathrm{vol}(G)=\sum_i\sum_j \boldsymbol{A}_{i,j}=\sum_i d_i$ is the volume of an weighted graph G;
T & b: The context window size and the number of negative sampling in skip-gram, respectively.

The output representations can be used as the input of mining and learning algorithms for a variety of network science tasks, such as label classification and community detection.

The attempt to address this problem can date back to spectral graph theory [10] and social dimension learning [36]. Its very recent advances have been largely influenced by the skip-gram model originally proposed for word embedding [25, 26], whose input is a text corpus composed of sentences in natural language and output is the latent vector representation for each word in the corpus. Notably, inspired by this setting, DeepWalk [28] pioneers network embedding by considering the vertex paths traversed by the random walks over networks as the sentences and leveraging skip-gram for learning latent node representations. With the advent of DeepWalk, many network embedding models have been developed, such as LINE [35], PTE [34], and node2vec [15].

The above models have thus far been demonstrated quite effective empirically. However, the theoretical mechanism behind them is much less well-understood. We note that the skip-gram with negative-sampling for word embedding has been shown to be an implicit factorization of certain word-context matrix [21]. But it is unclear what is the relation between the word-context matrix and the network. Moreover, there was an early attempt to theoretically analyze DeepWalk's behavior [47]. However, their main theoretical results are not fully consistent with the setting of the original DeepWalk paper (See the detailed discussion in Section 2.2). In addition, despite the superficial similarity between DeepWalk, LINE, PTE, and node2vec, there is a lack of deeper understanding of their underlying connections.

Contributions In this work, we provide several theoretical results concerning several existing skip-gram inspired network embedding methods. More concretely, we first show that all of four models we

mentioned—DeepWalk, LINE, PTE, and node2vec—are in theory performing implicit matrix factorization. We derive the closed form of the matrix for each model (see Table 1 for a summary). For example, DeepWalk (random walks over a graph + skip-gram) is in essence implicitly first approximating the closed-form matrix[1] and then factorizing the approximated matrix.

Second, observed from their matrices' closed forms, we find that, interestingly, LINE can be seen as a special case of DeepWalk, when the window size T of surrounding contexts is set to 1 in skip-gram. Further, we demonstrate that PTE, as an extension of LINE, is actually an implicit factorization of the joint matrix of multiple networks.

Third, we discover new theoretical connection between DeepWalk's implicit matrix and graph Laplacians. Based on the connection, we propose a new algorithm *NetMF* to approximate the closed form of DeepWalk's implicit matrix. By explicitly factorizing this sparse matrix based on SVD, our extensive experiments in four networks (used in the DeepWalk and node2vec approaches) demonstrate NetMF's outstanding performance (relative improvements by up to 38%) over DeepWalk and LINE.

2 THEORETICAL ANALYSIS AND PROOFS

In this section, we present the detailed theoretical analysis and proofs for four popular network embedding approaches: LINE, PTE, DeepWalk and node2vec.

2.1 LINE and PTE

LINE [35] We start our analysis of LINE with the second order proximity (aka LINE (2nd)). LINE (2nd) aims to learn two representation matrices $\boldsymbol{X}, \boldsymbol{Y} \in \mathbb{R}^{|V| \times d}$, where their rows are denoted by $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$, $i = 1, \cdots, n$. The objective of LINE (2nd) is to maximize

$$\ell = \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} \boldsymbol{A}_{i,j} \left(\log g\left(\boldsymbol{x}_i^\top \boldsymbol{y}_j\right) + b \mathbb{E}_{j' \sim P_N} \left[\log g\left(-\boldsymbol{x}_i^\top \boldsymbol{y}_{j'}\right) \right] \right),$$

where $g(\cdot) = 1/(1 + \exp(\cdot))$ is the sigmoid function; b is the parameter for negative sampling; P_N is known as the noise distribution that generates negative samples. In LINE, the authors empirically set $P_N(j) \propto d_j^{3/4}$. In our analysis, we assume $P_N(j) \propto d_j$ because the normalization factor has a closed form solution in graph theory, i.e., $P_N(j) = d_j/\mathrm{vol}(G)$.[2] Rewrite the objective function to be

$$\ell = \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} \boldsymbol{A}_{i,j} \log g\left(\boldsymbol{x}_i^\top \boldsymbol{y}_j\right) + b \sum_{i=1}^{|V|} d_i \mathbb{E}_{j' \sim P_N} \left[\log g\left(-\boldsymbol{x}_i^\top \boldsymbol{y}_{j'}\right) \right] \quad (1)$$

and explicitly express the expectation term as

$$\begin{aligned} \mathbb{E}_{j' \sim P_N} \left[\log g\left(-\boldsymbol{x}_i^\top \boldsymbol{y}_{j'}\right) \right] &= \sum_{j'} \frac{d_{j'}}{\mathrm{vol}(G)} \log g\left(-\boldsymbol{x}_i^\top \boldsymbol{y}_{j'}\right) \\ &= \frac{d_j}{\mathrm{vol}(G)} \log g\left(-\boldsymbol{x}_i^\top \boldsymbol{y}_j\right) + \sum_{j' \neq j} \frac{d_{j'}}{\mathrm{vol}(G)} \log g\left(-\boldsymbol{x}_i^\top \boldsymbol{y}_{j'}\right). \end{aligned} \quad (2)$$

By combining Eq. 1 and Eq. 2, and considering the local objective function for a specific pair of vertices (i, j), we have

$$\ell(i, j) = \boldsymbol{A}_{i,j} \log g\left(\boldsymbol{x}_i^\top \boldsymbol{y}_j\right) + b \frac{d_i d_j}{\mathrm{vol}(G)} \log g\left(-\boldsymbol{x}_i^\top \boldsymbol{y}_j\right).$$

[1] The implicit approximation is performed by random walk sampling.

[2] A similar result could be achieved if we use $P_N(j) \propto d_j^{3/4}$.

Let us define $z = \boldsymbol{x}_i^\top \boldsymbol{y}_j$ and take the derivative w.r.t. z, we have

$$\frac{\partial \ell(i, j)}{\partial z} = \boldsymbol{A}_{i,j} g(-z) - b \frac{d_i d_j}{\mathrm{vol}(G)} g(z).$$

Setting the derivative to be zero reveals

$$e^{2z} - \left(\frac{\mathrm{vol}(G) \boldsymbol{A}_{i,j}}{b d_i d_j} - 1 \right) e^z - \frac{\mathrm{vol}(G) \boldsymbol{A}_{i,j}}{b d_i d_j} = 0.$$

The above quadratic equation has two solutions (1) $e^z = -1$, which is invalid; and (2) $e^z = \frac{\mathrm{vol}(G) \boldsymbol{A}_{i,j}}{b d_i d_j}$, i.e.,

$$\boldsymbol{x}_i^\top \boldsymbol{y}_j = z = \log \left(\frac{\mathrm{vol}(G) \boldsymbol{A}_{i,j}}{b d_i d_j} \right). \quad (3)$$

Writing Eq. 3 in matrix form, LINE (2nd) is factoring matrix

$$\log \left(\mathrm{vol}(G) \boldsymbol{D}^{-1} \boldsymbol{A} \boldsymbol{D}^{-1} \right) - \log b = \boldsymbol{X} \boldsymbol{Y}^\top, \quad (4)$$

where $\log(\cdot)$ denotes element-wise matrix logarithm. Similarly, LINE with first order proximity (aka LINE (1st)) is factorizing the same matrix under the constraint that $\boldsymbol{X} = \boldsymbol{Y}$, i.e.,

$$\log \left(\mathrm{vol}(G) \boldsymbol{D}^{-1} \boldsymbol{A} \boldsymbol{D}^{-1} \right) - \log b = \boldsymbol{X} \boldsymbol{X}^\top.$$

PTE [34] PTE is an extension of LINE (2nd) in heterogeneous text networks. To achieve this, we first adapt our analysis of LINE (2nd) to bipartite networks. Consider a bipartite network $G = (V_1 \cup V_2, E)$ where V_1, V_2 are two sets of vertices and $E \subseteq V_1 \times V_2$. Similarly, we have the bipartite adjacency matrix $\boldsymbol{A} \in \mathbb{R}_+^{|V_1| \times |V_2|}$ and $\mathrm{vol}(G) = \sum_{i=1}^{|V_1|} \sum_{j=1}^{|V_2|} \boldsymbol{A}_{i,j}$. The goal is to learn a representation $\boldsymbol{x}_i$ for each vertex $v_i \in V_1$ and a representation $\boldsymbol{y}_j$ for each vertex $v_j \in V_2$. The objective function is

$$\ell = \sum_{i=1}^{|V_1|} \sum_{j=1}^{|V_2|} A_{i,j} \left(\log g\left(\boldsymbol{x}_i^\top \boldsymbol{y}_j\right) + b \mathbb{E}_{j' \sim P_N} \left[\log g\left(-\boldsymbol{x}_i^\top \boldsymbol{y}_{j'}\right) \right] \right).$$

Applying the same procedure for LINE, we can see that maximizing ℓ is actually factorizing

$$\log \left(\mathrm{vol}(G) \boldsymbol{D}_{\mathrm{row}}^{-1} \boldsymbol{A} \boldsymbol{D}_{\mathrm{col}}^{-1} \right) - \log b = \boldsymbol{X} \boldsymbol{Y}^\top$$

where we use $\boldsymbol{D}_{\mathrm{row}}$ and $\boldsymbol{D}_{\mathrm{col}}$ because they are not the same in bipartite networks. Furthermore, the heterogeneous text network is composed of three sub-networks — word-word network G_{ww}, document-word network G_{dw}, and label-word network G_{lw}, where G_{dw} and G_{lw} are bipartite. Let the adjacency matrix of the three sub-networks be $\boldsymbol{A}_{\mathrm{ww}} \in \mathbb{R}^{\#\mathrm{word} \times \#\mathrm{word}}$, $\boldsymbol{A}_{\mathrm{dw}} \in \mathbb{R}^{\#\mathrm{doc} \times \#\mathrm{word}}$ and $\boldsymbol{A}_{\mathrm{lw}} \in \mathbb{R}^{\#\mathrm{label} \times \#\mathrm{word}}$. Take the diagonal degree matrices of G_{dw} as example, we use $\boldsymbol{D}_{\mathrm{row}}^{\mathrm{dw}}$ and $\boldsymbol{D}_{\mathrm{col}}^{\mathrm{dw}}$ to denote its diagonal matrices with row/column sum. Given the above notations, PTE is factorizing

$$\log \left(\begin{bmatrix} \alpha\, \mathrm{vol}(G_{\mathrm{ww}}) (\boldsymbol{D}_{\mathrm{row}}^{\mathrm{ww}})^{-1} \boldsymbol{A}_{\mathrm{ww}} (\boldsymbol{D}_{\mathrm{col}}^{\mathrm{ww}})^{-1} \\ \beta\, \mathrm{vol}(G_{\mathrm{dw}}) (\boldsymbol{D}_{\mathrm{row}}^{\mathrm{dw}})^{-1} \boldsymbol{A}_{\mathrm{dw}} (\boldsymbol{D}_{\mathrm{col}}^{\mathrm{dw}})^{-1} \\ \gamma\, \mathrm{vol}(G_{\mathrm{lw}}) (\boldsymbol{D}_{\mathrm{row}}^{\mathrm{lw}})^{-1} \boldsymbol{A}_{\mathrm{lw}} (\boldsymbol{D}_{\mathrm{col}}^{\mathrm{lw}})^{-1} \end{bmatrix} \right) - \log b, \quad (5)$$

where the factorized matrix is of shape (#word + #doc + #label) × #word, b is the parameter for negative sampling, and $\{\alpha, \beta, \gamma\}$ are non-negative hyper-parameters to balance the weights of the three networks. In PTE, $\{\alpha, \beta, \gamma\}$ satisfy $\alpha\, \mathrm{vol}(G_{\mathrm{ww}}) = \beta\, \mathrm{vol}(G_{\mathrm{dw}}) = \gamma\, \mathrm{vol}(G_{\mathrm{lw}})$. This is because the authors perform edge sampling in training where edges from three different networks are sampled alternatively (see Section 4.2 in [34]).

2.2 DeepWalk

In this section, we analyze DeepWalk [28] and illustrate the essence of DeepWalk is actually performing an implicit matrix factorization (See the matrix form solution in Thm. 2.3).

DeepWalk assumes a "corpus" $\mathcal{D}$ generated by random walks on graphs [23], and then trains a skip-gram with negative sampling (SGNS) on the corpus. For clarity, we summarize the DeepWalk method in Algorithm 1. The outer loop (Line 1-7) specifies the total number of times, N, for which we should run random walks. For each random walk of length L, the first vertex is sampled from a prior distribution $P(w)$. The inner loop (Line 4-7) specifies the construction of corpus $\mathcal{D}$. Once we have the corpus $\mathcal{D}$, we run an SGNS to attain the network embedding (Line 8). The number of negative samples is set to be b. Next, we introduce some necessary background about the SGNS technique, followed by our analysis of the DeepWalk method.

Algorithm 1: DeepWalk

1 **for** $i = 1, 2, \ldots, N$ **do**
2 Pick w_1 according to a probability distribution $P(w_1)$;
3 Generate a vertex sequence $(w_1, \cdots, w_n)$ of length L by random walk;
4 **for** $j = 1, 2, \ldots, L - T$ **do**
5 **for** $r = 1, \ldots, T$ **do**
6 Add vertex-context pair (w_j, w_{j+r}) to corpus $\mathcal{D}$;
7 Add vertex-context pair (w_{j+r}, w_j) to corpus $\mathcal{D}$;
8 Run SGNS on $\mathcal{D}$ with b negative samples.

Preliminary on Skip-gram with Negative Sampling (SGNS) The skip-gram model assumes a corpus of words w and their context c. More concretely, the words come from a textual corpus of words $w_1, \cdots w_n$ and the contexts for word w_i are words surrounding it in an T-sized window $w_{i-T}, \cdots, w_{i-1}, w_{i+1}, \cdots, w_{i+T}$. Following the work by Levy and Goldberg[21], the SGNS is actually factorizing

$$\log\left(\frac{\#(w,\, c)\,|\mathcal{D}|}{\#(w)\cdot\#(c)}\right) - \log b, \tag{6}$$

where b is the number of negative samples.

Proofs and Analysis Our analysis of DeepWalk is based on the following key assumptions:

- The graph is undirected, connected, and non-bipartite, which makes $P(w) = \frac{d_w}{\mathrm{vol}(G)}$ the stationary distribution.
- The first vertex of random walk is sampled from the stationary distribution $P(w) = \frac{d_w}{\mathrm{vol}(G)}$. Note that in the DeepWalk paper [28], $P(w)$ is the uniform distribution over the vertices. We discuss the difference in Remark 1.

To characterize DeepWalk, we want to reinterpret Eq. 6 using graph theoretic terminology. The corpus $\mathcal{D}$ is actually a multiset which counts the multiplicity of vertex-context pairs. To be consistent with the discussion in [21], we also use term $\#(w, c)$ to denote the number of times the vertex-context pair (w, c) appears in corpus $\mathcal{D}$, $\#(w) = \sum_{c'} \#(w, c')$ and $\#(c) = \sum_{w'} \#(w', c)$. Moreover, we define some necessary notations here. First, we divide the vertex-context pairs (w, c) in multiset $\mathcal{D}$ into $2T$ categories (sub-multiset). More concretely, for $r = 1, \cdots, T$, we define

$$\mathcal{D}_{\overrightarrow{r}} = \{(w, c) \in \mathcal{D} : w_i = w,\, w_{i+r} = c\},$$
$$\mathcal{D}_{\overleftarrow{r}} = \{(w, c) \in \mathcal{D} : w_i = w,\, w_{i-r} = c\}.$$

That is, $\mathcal{D}_{\overrightarrow{r}}/\mathcal{D}_{\overleftarrow{r}}$ is the sub-multiset of $\mathcal{D}$ such that context c is r-step after/before vertex w in random walks. Second, for a specific vertex-context pair (w, c), we use $\#(w, c)_{\overrightarrow{r}}$ and $\#(w, c)_{\overleftarrow{r}}$ to denote the number of times such pair appears in $\mathcal{D}_{\overrightarrow{r}}$ and $\mathcal{D}_{\overleftarrow{r}}$, respectively.

Theorem 2.1. *Denote $\boldsymbol{P} = \boldsymbol{D}^{-1}\boldsymbol{A}$, we have*

$$\frac{\#(w,\, c)_{\overrightarrow{r}}}{\left|\mathcal{D}_{\overrightarrow{r}}\right|} = \frac{d_w}{\mathrm{vol}(G)}\left(\boldsymbol{P}^r\right)_{w,c} \text{ and } \frac{\#(w,\, c)_{\overleftarrow{r}}}{\left|\mathcal{D}_{\overleftarrow{r}}\right|} = \frac{d_c}{\mathrm{vol}(G)}\left(\boldsymbol{P}^r\right)_{c,w}.$$

Proof. Let's consider the vertex sequence generated by random walk as described in Algorithm 1. Easy to see that $\frac{\#(w,c)_{\overrightarrow{r}}}{\left|\mathcal{D}_{\overrightarrow{r}}\right|}$ is an estimator of the probability of visiting vertex w at some position and vertex c at another position r steps after. Thus, we have

$$\begin{aligned}\frac{\#(w,\, c)_{\overrightarrow{r}}}{\left|\mathcal{D}_{\overrightarrow{r}}\right|} &= \frac{1}{L-T}\sum_{j=1}^{L-T} P(w_j = w,\, w_{j+r} = c)\\ &= \frac{1}{L-T}\sum_{j=1}^{L-T} P(w_j = w)P(w_{j+r} = c|w_j = w).\end{aligned}$$

Because the random walk starts from the stationary distribution, we know $P(w_j = w) = \frac{d_w}{\mathrm{vol}(G)}$. According to the property of random walk that $P(w_{j+r} = c|w_j = w) = (\boldsymbol{P}^r)_{w,c}$, we have

$$\frac{\#(w,\, c)_{\overrightarrow{r}}}{\left|\mathcal{D}_{\overrightarrow{r}}\right|} = \frac{1}{L-T}\sum_{j=1}^{L-T}\frac{d_w}{\mathrm{vol}(G)}\left(\boldsymbol{P}^r\right)_{w,c} = \frac{d_w}{\mathrm{vol}(G)}\left(\boldsymbol{P}^r\right)_{w,c},$$

Similarly, we get $\frac{\#(w,c)_{\overleftarrow{r}}}{\left|\mathcal{D}_{\overleftarrow{r}}\right|} = \frac{d_c}{\mathrm{vol}(G)}\left(\boldsymbol{P}^r\right)_{c,w}$. □

Remark 1. *What if we start the random walk with other distributions (for example, uniform distribution in origin DeepWalk [28])? Actually, for a connected, undirected, non-bipartite graph, $P(w_j = w) \to \frac{d_w}{\mathrm{vol}(G)}$ as $j \to \infty$ (see section 3 in [23]). So when the length of random walk is sufficiently large, the theorem above still holds.*

Theorem 2.2.

$$\frac{\#(w,\, c)}{|\mathcal{D}|} = \frac{1}{2T}\sum_{r=1}^{T}\left(\frac{d_w}{\mathrm{vol}(G)}\left(\boldsymbol{P}^r\right)_{w,c} + \frac{d_c}{\mathrm{vol}(G)}\left(\boldsymbol{P}^r\right)_{c,w}\right).$$

Proof. Note that $\frac{\left|\mathcal{D}_{\overrightarrow{r}}\right|}{|\mathcal{D}|} = \frac{\left|\mathcal{D}_{\overleftarrow{r}}\right|}{|\mathcal{D}|} = \frac{1}{2T}$. Thus, we get

$$\begin{aligned}\frac{\#(w,\, c)}{|\mathcal{D}|} &= \frac{\sum_{r=1}^{T}\left(\#(w,\, c)_{\overrightarrow{r}} + \#(w,\, c)_{\overleftarrow{r}}\right)}{\sum_{r=1}^{T}\left(\left|\mathcal{D}_{\overrightarrow{r}}\right| + \left|\mathcal{D}_{\overleftarrow{r}}\right|\right)}\\ &= \frac{\sum_{r=1}^{T}\left(\frac{\#(w,c)_{\overrightarrow{r}}}{\left|\mathcal{D}_{\overrightarrow{r}}\right|}\frac{\left|\mathcal{D}_{\overrightarrow{r}}\right|}{|\mathcal{D}|} + \frac{\#(w,c)_{\overleftarrow{r}}}{\left|\mathcal{D}_{\overleftarrow{r}}\right|}\frac{\left|\mathcal{D}_{\overleftarrow{r}}\right|}{|\mathcal{D}|}\right)}{\sum_{r=1}^{T}\left(\frac{\left|\mathcal{D}_{\overrightarrow{r}}\right|}{|\mathcal{D}|} + \frac{\left|\mathcal{D}_{\overleftarrow{r}}\right|}{|\mathcal{D}|}\right)}\\ &= \frac{1}{2T}\sum_{r=1}^{T}\left(\frac{d_w}{\mathrm{vol}(G)}\left(\boldsymbol{P}^r\right)_{w,c} + \frac{d_c}{\mathrm{vol}(G)}\left(\boldsymbol{P}^r\right)_{c,w}\right).\end{aligned}$$

Further, we have $\frac{\#(w)}{|\mathcal{D}|} = \frac{d_w}{\mathrm{vol}(G)}$ and $\frac{\#(c)}{|\mathcal{D}|} = \frac{d_c}{\mathrm{vol}(G)}$. □

Theorem 2.3. *For DeepWalk,*

$$\frac{\#(w,\, c)\,|\mathcal{D}|}{\#(w)\cdot\#(c)} = \frac{\mathrm{vol}(G)}{2T}\left(\frac{1}{d_c}\sum_{r=1}^{T}(\boldsymbol{P}^r)_{w,c} + \frac{1}{d_w}\sum_{r=1}^{T}(\boldsymbol{P}^r)_{c,w}\right).$$

In matrix form, DeepWalk is equivalent to factorizing

$$\log\left(\frac{\mathrm{vol}(G)}{T}\left(\sum_{r=1}^{T}\boldsymbol{P}^r\right)\boldsymbol{D}^{-1}\right) - \log(b). \tag{7}$$

Proof. Using the results in Thm. 2.2, we get

$$\begin{aligned}\frac{\#(w,\, c)\,|\mathcal{D}|}{\#(w)\cdot\#(c)} &= \frac{\frac{\#(w,c)}{|\mathcal{D}|}}{\frac{\#(w)}{|\mathcal{D}|}\cdot\frac{\#(c)}{|\mathcal{D}|}}\\ &= \frac{\frac{1}{2T}\sum_{r=1}^{T}\left(\frac{d_w}{\mathrm{vol}(G)}(\boldsymbol{P}^r)_{w,c} + \frac{d_c}{\mathrm{vol}(G)}(\boldsymbol{P}^r)_{c,w}\right)}{\frac{d_w}{\mathrm{vol}(G)}\cdot\frac{d_c}{\mathrm{vol}(G)}}\\ &= \frac{\mathrm{vol}(G)}{2T}\left(\frac{1}{d_c}\sum_{r=1}^{T}(\boldsymbol{P}^r)_{w,c} + \frac{1}{d_w}\sum_{r=1}^{T}(\boldsymbol{P}^r)_{c,w}\right).\end{aligned}$$

Write it in matrix form:

$$\begin{aligned}&\frac{\mathrm{vol}(G)}{2T}\left(\sum_{r=1}^{T}\boldsymbol{P}^r\boldsymbol{D}^{-1} + \sum_{r=1}^{T}\boldsymbol{D}^{-1}(\boldsymbol{P}^r)^\top\right)\\ =&\frac{\mathrm{vol}(G)}{2T}\left(\sum_{r=1}^{T}\underbrace{\boldsymbol{D}^{-1}\boldsymbol{A}\times\cdots\times\boldsymbol{D}^{-1}\boldsymbol{A}}_{k\text{ terms}}\boldsymbol{D}^{-1} + \sum_{r=1}^{T}\boldsymbol{D}^{-1}\underbrace{\boldsymbol{A}\boldsymbol{D}^{-1}\times\cdots\times\boldsymbol{A}\boldsymbol{D}^{-1}}_{k\text{ terms}}\right)\\ =&\frac{\mathrm{vol}(G)}{T}\sum_{r=1}^{T}\underbrace{\boldsymbol{D}^{-1}\boldsymbol{A}\times\cdots\times\boldsymbol{D}^{-1}\boldsymbol{A}}_{k\text{ terms}}\boldsymbol{D}^{-1} = \mathrm{vol}(G)\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{P}^r\right)\boldsymbol{D}^{-1}.\end{aligned}$$

□

Corollary 2.4. *Comparing Eq. 4 and Eq. 7, we can easily observe that LINE (2nd) is a special case of DeepWalk when $T = 1$.*

By slightly modifying the Eq. 7, DeepWalk can be further generalized to model different weights on different hops. For a T-dimension multinomial distribution with parameters $\pi_1, \cdots, \pi_T$ such that $\pi_i \geq 0$ and $\sum \pi_i = 1$, we can define the following matrix:

$$\log\left(\mathrm{vol}(G)\left(\sum_{r=1}^{T}\pi_r\boldsymbol{P}^r\right)\boldsymbol{D}^{-1}\right) - \log b.$$

This matrix assigns different weights to different hops in random walk. DeepWalk could be regarded as a special case of uniform distribution, i.e., $\pi_1 = \cdots = \pi_r = 1/T$. In fact, the authors of DeepWalk proposed a model named WalkLets [29] recently where they assign higher weights to distant vertex-context pairs in random walk. WalkLets can be explained by our generalization.

Discussion on Yang *et al.* [46, 47] Two related works we want to discuss here are [46] and [47] by the same authors, where they claim they prove a closed form matrix factorized by DeepWalk. The conclusion is then referred and used in a few following papers [29, 42, 48]. Here we want to point out their limitations. The first issue comes from negative sampling. In [46] and [47], a strong assumption is the use of softmax instead of negative sampling. In softmax, all samples are of equal weights. However, negative sampling performs weighted sampling according to the frequency of samples. The second issue comes from random walk and the construction of the "corpus" $\mathcal{D}$. In DeepWalk, the vertex-context pairs are "undirected" — for one vertex, both its predecessors and successors in random walk will be treated as its context. In [46] and [47], however, the authors only consider vertex-context pair whose context comes after vertex. This makes their results limited.

2.3 node2vec

Node2vec [15] is a recent and state-of-the-art network embedding method. We briefly summarize node2vec in Algorithm 2. Different from DeepWalk, node2vec defines a 2nd-order random walk. The authors first define an unnormalized transition probability tensor $\underline{\boldsymbol{T}}$ with two parameters — return parameter p and in-out parameter q, and then normalize it to be a transition probability (Line 1):

$$\underline{\boldsymbol{T}}_{i,j,k} = \begin{cases}\frac{1}{p} & k = i,\\ 1 & i,\, j,\, k \text{ form a closed-triangle},\\ \frac{1}{q} & i,\, j,\, k \text{ form a open-triangle}.\end{cases}$$

$$\underline{\boldsymbol{P}}_{i,j,k} = \mathrm{Prob}\,(w_{t+1} = i | w_t = j,\, w_{t-1} = k) = \frac{\underline{\boldsymbol{T}}_{i,j,k}}{\sum_i \underline{\boldsymbol{T}}_{i,j,k}}.$$

Similar to DeepWalk, node2vec performs this 2nd-order random walk to generate a "corpus" $\mathcal{D}$ (Line 2-8) and then train an SGNS model (Line 9) on the corpus.

Algorithm 2: node2vec

1 Construct transition probability tensor $\underline{\boldsymbol{P}}$;
2 **for** $i = 1, 2, \ldots, N$ **do**
3 Pick w_1, w_2 according to a distribution $P(w_1, w_2)$;
4 Generate a vertex sequence $(w_1, \cdots, w_n)$ of length L by the 2nd-order random walk;
5 **for** $j = 2, 3, \ldots, L - T$ **do**
6 **for** $r = 1, \ldots, T$ **do**
7 Add vertex-context pair (w_j, w_{j+r}) to corpus $\mathcal{D}$;
8 Add vertex-context pair (w_{j+r}, w_j) to corpus $\mathcal{D}$;
9 Run SGNS on $\mathcal{D}$ with b negative samples.

Notations We define some necessary notations to characterize the 2nd-order random walk. Firstly, we categorize vertex-context pairs into the following classes, i.e., for $r = 1, \cdots, T$

$$\begin{aligned}\mathcal{D}_{\overrightarrow{r}} &= \{(w, c, u) : w_{i-1} = u,\, w_i = w,\, w_{i+r} = c\}\\ \mathcal{D}_{\overleftarrow{r}} &= \{(w, c, u) : w_i = w,\, w_{i-r} = c,\, w_{i-r-1} = u\}.\end{aligned}$$

Similarly, for a certain triplet (w, c, u), we use $\#(w, c, u)_{\overrightarrow{r}}$ and $\#(w, c, u)_{\overleftarrow{r}}$ to denote the number of times such triplets appear in $\mathcal{D}_{\overrightarrow{r}}$ and $\mathcal{D}_{\overleftarrow{r}}$. Secondly, the higher-order transition probability tensor is defined to be $(\underline{\boldsymbol{P}}^r)_{i,j,k} = \mathrm{Prob}\,(w_{t+r} = i | w_t = j, w_{t-1} = k)$, which is analogous to $\boldsymbol{P}^r$ in DeepWalk.

Stationary Distribution Tensor $\underline{\boldsymbol{P}}$ defines a 2nd-order Markov chain. The stationary distribution $\boldsymbol{X}_{i,j}$ of the 2nd-order Markov chain satisfies $\sum_k \underline{\boldsymbol{P}}_{i,j,k}\boldsymbol{X}_{j,k} = \boldsymbol{X}_{i,j}$. The existence of such $\boldsymbol{X}$ is guaranteed by Perron-Frobenius theorem [3]. In our analysis of node2vec, we assume the first two vertices of the 2nd-order random walks are sampled from the stationary distribution $\boldsymbol{X}$. Moreover, for any $r \geq 1$, $\underline{\boldsymbol{P}}$ and $\boldsymbol{X}$ satisfy $\sum_j \sum_k (\underline{\boldsymbol{P}}^r)_{i,j,k}\boldsymbol{X}_{j,k} = \sum_j \boldsymbol{X}_{i,j}$, which can be proved by induction on r.

Main Results Limited by space, we list the main results of node2vec without proofs. The idea is similar to our analysis of DeepWalk.

- $\frac{\#(w,c,u)_{\overrightarrow{r}}}{|\mathcal{D}_{\overrightarrow{r}}|} = \boldsymbol{X}_{w,u}(\underline{\boldsymbol{P}}^r)_{c,w,u}$ and $\frac{\#(w,c,u)_{\overleftarrow{r}}}{|\mathcal{D}_{\overleftarrow{r}}|} = \boldsymbol{X}_{c,u}(\underline{\boldsymbol{P}}^r)_{w,c,u}$.
- $\frac{\#(w,c)_{\overrightarrow{r}}}{|\mathcal{D}_{\overrightarrow{r}}|} = \frac{\sum_u \#(w,c,u)_{\overrightarrow{r}}}{|\mathcal{D}_{\overrightarrow{r}}|} = \sum_u \boldsymbol{X}_{w,u}(\underline{\boldsymbol{P}}^r)_{c,w,u}$.
- $\frac{\#(w,c)_{\overleftarrow{r}}}{|\mathcal{D}_{\overleftarrow{r}}|} = \frac{\sum_u \#(w,c,u)_{\overleftarrow{r}}}{|\mathcal{D}_{\overleftarrow{r}}|} = \sum_u \boldsymbol{X}_{c,u}(\underline{\boldsymbol{P}}^r)_{w,c,u}$.
- $\frac{\#(w,c)}{|\mathcal{D}|} = \frac{1}{2T}\sum_{r=1}^{T}\left(\sum_u \boldsymbol{X}_{w,u}(\underline{\boldsymbol{P}}^r)_{c,w,u} + \sum_u \boldsymbol{X}_{c,u}(\underline{\boldsymbol{P}}^r)_{w,c,u}\right)$.
- $\frac{\#(w)}{|\mathcal{D}|} = \sum_u \boldsymbol{X}_{w,u}$ and $\frac{\#(c)}{|\mathcal{D}|} = \sum_u \boldsymbol{X}_{c,u}$.

Combine the above results together, we conclude that

$$\frac{\#(w,\,c)\,|\mathcal{D}|}{\#(w)\cdot\#(c)} = \frac{\frac{1}{2T}\sum_{r=1}^{T}\left(\sum_u \boldsymbol{X}_{w,u}\underline{\boldsymbol{P}}^r_{c,w,u} + \sum_u \boldsymbol{X}_{c,u}\underline{\boldsymbol{P}}^r_{w,c,u}\right)}{\left(\sum_u \boldsymbol{X}_{w,u}\right)\left(\sum_u \boldsymbol{X}_{c,u}\right)}. \quad (8)$$

However, we can not achieve a matrix form solution for node2vec like what we have done for DeepWalk in section 2.2.

As we see, modeling the full 2nd-order dynamics is expensive because we have to compute and store the stationary distribution $\boldsymbol{X}$ and transition probability tensor $\underline{\boldsymbol{P}}^r$. We have noticed some recent works [2, 3, 14] that try to understand or approximate 2nd-order random walk by assuming a rank-one factorization $\boldsymbol{X}_{i,j} = \boldsymbol{x}_i\boldsymbol{x}_j$ for stationary distribution $\boldsymbol{X}$. Due to the page limitation, in the next section, we mainly focus on the matrix factorization framework depends on the 1st-order random walk. We leave the solution over the 2nd-order random walk for future research.

3 NetMF: NETWORK EMBEDDING AS MATRIX FACTORIZATION

Based on the analysis in Section 2, we unify LINE, PTE, DeepWalk and node2vec in the framework of matrix factorization, where the factorized matrices have closed forms as showed in Eq. 4, Eq. 5, Eq. 7, and Eq. 8, respectively. In this section, we study the DeepWalk matrix (Eq. 7) because it is more general than LINE matrix and computationally more efficient than node2vec matrix. we first discuss the connection between DeepWalk matrix and graph Laplacian in section 3.1. Then in section 3.2, we present NetMF for small window size T and its approximation version for large T.

3.1 Connection between DeepWalk Matrix and Normalized Graph Laplacian

In this section, we will show that the factorized matrix has a close relationship with the normalized graph Laplacian. Ignoring the element-wise matrix logarithm and the constant term, the matrix we analyze is $\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{P}^r\right)\boldsymbol{D}^{-1}$.

Preliminaries The normalized graph Laplacian is defined to be $\mathcal{L} := \boldsymbol{I} - \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$ and has the following desired properties.

Theorem 3.1. *([10]) The normalized graph Laplacians $\mathcal{L}$ satisfies:*

- *All eigenvalues are real numbers and lie in the interval* $[0, 2]$
- *The multiplicity of 0 is the number of connected components of the graph.*
- *The multiplicity of 2 is the number of bipartite components with at least two vertices.*

Besides Thm. 3.1, we also use the following two helpful theorems.

Theorem 3.2. *([39]) Singular values of a real symmetric matrix are the absolute values of its eigenvalues.*

Theorem 3.3. *([16], pp. 178-180) Let $\boldsymbol{B}, \boldsymbol{C}$ be two $n \times n$ symmetric matrices. Then for the decreasingly ordered singular values of $\boldsymbol{B}, \boldsymbol{C}$ and $\boldsymbol{BC}$, the following inequality*

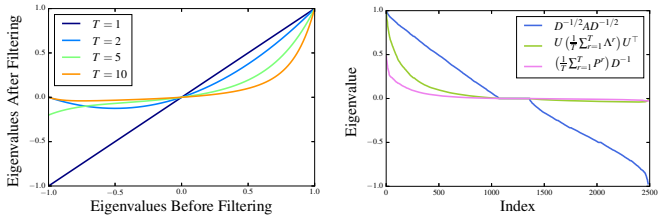
$$\sigma_{i+j-1}(\boldsymbol{BC}) \le \sigma_i(\boldsymbol{B}) \times \sigma_j(\boldsymbol{C})$$

holds for any $1 \le i, j \le n$ and $i + j \le n + 1$.

Proofs and Analysis By Thm. 3.1, $\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2} = \boldsymbol{I} - \mathcal{L}$ has eigen-decomposition $\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$ such that $\boldsymbol{U}\boldsymbol{U}^\top = \boldsymbol{I}, \boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \cdots, \lambda_n)$, where $1 = \lambda_1 \ge \lambda_2 \ge \cdots \ge \lambda_n \ge -1$. Based on the eigen-decomposition of $\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$, we can see that $\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{P}^r\right)\boldsymbol{D}^{-1}$ has the decomposition

$$\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{P}^r\right)\boldsymbol{D}^{-1} = \boldsymbol{D}^{-1/2}\boldsymbol{U}\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{\Lambda}^r\right)\boldsymbol{U}^\top\boldsymbol{D}^{-1/2},$$

which is the product of the following three symmetric matrices: $\boldsymbol{D}^{-1/2}$, $\boldsymbol{U}\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{\Lambda}^r\right)\boldsymbol{U}^\top$, and $\boldsymbol{D}^{-1/2}$. For either the first or the third matrix, $\boldsymbol{D}^{-1/2}$, it is a diagonal matrix with eigenvalues $1/\sqrt{d_i}$, for $i = 1, \cdots, n$. For the second matrix, $\boldsymbol{U}\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{\Lambda}^r\right)\boldsymbol{U}^\top$ has eigenvalues $\frac{1}{T}\sum_{r=1}^{T}\lambda_i^r$, for $i = 1, \cdots n$. We can treat the new eigenvalue $\frac{1}{T}\sum_{r=1}^{T}\lambda_i^r$ as the output of a transformation applied on old eigenvalue λ_i, i.e., a kind of filter! We plot the effect of this transformation (filter) $f(x) = \frac{1}{T}\sum_{r=1}^{T}x^r$ in Figure 1(a). This filter has two properties. Firstly, it prefers positive large eigenvalues; Secondly, the preference becomes stronger as the window size T increases. In conclusion, as T grows, this filter tries to approximate a low rank positive semi-definite matrix by keeping large positive eigenvalues.



(a) (b)

Figure 1: Comprehend DeepWalk Matrix as Filtering: (1) Function $f(x) = \frac{1}{T}\sum_{r=1}^{T}x^r$ with domain $[-1, 1]$. The window size T is set to be $1, 2, 5, 10$, respectively; (2) Eigenvalues of $\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$, $\boldsymbol{U}\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{\Lambda}^r\right)\boldsymbol{U}^\top$, and $\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{P}^r\right)\boldsymbol{D}^{-1}$ for Cora network.

Guided by Thm. 3.2, the decreasingly ordered singular values of the matrix $\boldsymbol{U}\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{\Lambda}^r\right)\boldsymbol{U}^\top$ can be constructed by sorting the absolute value of its eigenvalues in non-increasing order such that

$$\left|\frac{1}{T}\sum_{r=1}^{T}\lambda_{p_1}^r\right| \ge \left|\frac{1}{T}\sum_{r=1}^{T}\lambda_{p_2}^r\right| \ge \cdots \ge \left|\frac{1}{T}\sum_{r=1}^{T}\lambda_{p_n}^r\right|, \quad (9)$$

where $\{p_1, p_2, \cdots, p_n\}$ is a permutation of $\{1, 2, \cdots, n\}$. Similarly, since every d_i is positive, the decreasingly ordered singular values of the matrix $\boldsymbol{D}^{-1/2}$ can be constructed by sorting $1/\sqrt{d_i}$ in non-increasing order such that $1/\sqrt{d_{q_1}} \ge 1/\sqrt{d_{q_2}} \ge \cdots 1/\sqrt{d_{q_n}}$ where $\{q_1, q_2, \cdots, q_n\}$ is a permutation of $\{1, 2, \cdots, n\}$. Applying

Thm. 3.3 twice reveals the fact that the s-th singular value satisfies

$$\sigma_s\left(\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{P}^r\right)\boldsymbol{D}^{-1}\right) \le \sigma_i\left(\boldsymbol{D}^{-\frac{1}{2}}\right)\sigma_j\left(\boldsymbol{U}\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{\Lambda}^r\right)\boldsymbol{U}^\top\right)\sigma_k\left(\boldsymbol{D}^{-\frac{1}{2}}\right) = \frac{\left|\frac{1}{T}\sum_{r=1}^{T}\lambda_{p_j}^r\right|}{\sqrt{d_{q_i}d_{q_k}}}$$

for any $1 \le i, j, k \le n$ and $s := i+j+k-2$. This could be interpreted as another filter, since the magnitudes of eigenvalues are reduced according to the degree sequence, i.e., d_i's. In particular, an upper bound of this filter can be showed by simply fixing $i = k = 1$, i.e.,

$$\sigma_s\left(\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{P}^r\right)\boldsymbol{D}^{-1}\right) \le \frac{\left|\frac{1}{T}\sum_{r=1}^{T}\lambda_{p_s}^r\right|}{d_{q_1}}, \tag{10}$$

for any $1 \le s \le n$. According to our construction, $d_{q_1} = d_{\min}$ is the smallest generalized degree.

Illustrative Example: Cora In order to illustrate the filtering effect we discuss above, we analyze a small citation network Cora [24]. To meet our assumption that the network is connected, we choose the largest connected components from Cora, with 2,485 nodes (90.8% nodes), 5,069 edges (93.4% edges), and minimum degree $d_{\min} = 1$. We also assume the citation links are undirected. As showed in Figure 1(b), we plot the decreasingly ordered eigenvalues of matrices $\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$, $\boldsymbol{U}\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{\Lambda}^r\right)\boldsymbol{U}^\top$, and $\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{P}^r\right)\boldsymbol{D}^{-1}$, respectively. For $\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$, the largest eigenvalue $\lambda_1 = 1$ with multiplicity one, and the smallest eigenvalue $\lambda_n = -0.971 > -1$. For $\boldsymbol{U}\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{\Lambda}^r\right)\boldsymbol{U}^\top$, we observe that all negative eigenvalues and small positive eigenvalues are "filtered out" in spectrum. Finally, for matrix $\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{P}^r\right)\boldsymbol{D}^{-1}$, we observe that the magnitudes of eigenvalues are further reduced.

3.2 NetMF

In this section, we propose NetMF. For simplicity, we denote $\boldsymbol{M} = \frac{\mathrm{vol}(G)}{bT}\left(\sum_{r=1}^{T}\boldsymbol{P}^r\right)\boldsymbol{D}^{-1}$, and we call $\log \boldsymbol{M}$ the DeepWalk matrix.

NetMF for Small Window Size T NetMF for small T is quite straightforward and intuitive. The basic idea is to compute DeepWalk matrix exactly and then perform matrix factorization. The details in listed in Algorithm 3. In the first step (Line 1-2), we compute matrix power from $\boldsymbol{P}^1$ to $\boldsymbol{P}^T$ and then get $\boldsymbol{M}$. However, the factorization of $\log \boldsymbol{M}$ presents computation challenges due to the element-wise matrix logarithm. The matrix is not only ill-defined (since $\log 0 = -\infty$), but also dense. Inspired by an approach from skip-gram model named *Shifted* PPMI (SPPMI) [21], we define $\boldsymbol{M}'$ such that $\boldsymbol{M}'_{i,j} = \max(\boldsymbol{M}_{i,j}, 1)$ (Line 3). In this way $\log \boldsymbol{M}'$ is a sparse and consistent version of $\log \boldsymbol{M}$. We then factorize $\log \boldsymbol{M}'$ using Singular Value Decomposition and construct network embedding using its top-d singular values/vectors.

NetMF for Large Window Size Direct computation of matrix $\boldsymbol{M}$ presents some computation challenges for large window size T. First, the time complexity of exact computation of matrix $\boldsymbol{M}$ is $O(T|V||E|)$. Second, it also leads to numerical precision problems. Here we propose an approximation algorithm as listed in Algorithm 4. The basic idea is from our analysis in section 3.1,

Algorithm 3: NetMF for Small Window Size T

1 Compute $\boldsymbol{P}^1, \cdots, \boldsymbol{P}^T$;
2 Compute $\boldsymbol{M} = \frac{\mathrm{vol}(G)}{bT}\left(\sum_{r=1}^{T}\boldsymbol{P}^r\right)\boldsymbol{D}^{-1}$;
3 Compute $\boldsymbol{M}' = \max(\boldsymbol{M}, 1)$;
4 Rank-d approximation by SVD: $\log \boldsymbol{M}' = \boldsymbol{U}_d\boldsymbol{\Sigma}_d\boldsymbol{V}_d^\top$;
5 **return** $\boldsymbol{U}_d\sqrt{\boldsymbol{\Sigma}_d}$ as network embedding.

Algorithm 4: NetMF for Large Window Size T

1 Eigen-decomposition $\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2} \approx \boldsymbol{U}_h\boldsymbol{\Lambda}_h\boldsymbol{U}_h^\top$;
2 Approximate $\boldsymbol{M}$ with
$\hat{\boldsymbol{M}} = \frac{\mathrm{vol}(G)}{b}\boldsymbol{D}^{-1/2}\boldsymbol{U}_h\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{\Lambda}_h^r\right)\boldsymbol{U}_h^\top\boldsymbol{D}^{-1/2}$;
3 Compute $\hat{\boldsymbol{M}}' = \max(\hat{\boldsymbol{M}}, 1)$;
4 Rank-d approximation by SVD: $\log \hat{\boldsymbol{M}}' = \boldsymbol{U}_d\boldsymbol{\Sigma}_d\boldsymbol{V}_d^\top$;
5 **return** $\boldsymbol{U}_d\sqrt{\boldsymbol{\Sigma}_d}$ as network embedding.

where we reveal $\boldsymbol{M}$'s relationship with normalized graph Laplacian and show its low-rank nature theoretically and empirically. In our algorithm, we first approximate $\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$ with its top-h eigenpairs $\boldsymbol{U}_h\boldsymbol{\Lambda}_h\boldsymbol{U}_h^\top$ (Line 1). Since only top-h eigenpairs are required and the matrix is sparse, we can perform decomposition using the Arnoldi method [19] which achieves significant time reduction [45]. In step two (Line 2), we approximate $\boldsymbol{M}$ with $\hat{\boldsymbol{M}} = \frac{\mathrm{vol}(G)}{b}\boldsymbol{D}^{-1/2}\boldsymbol{U}_h\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{\Lambda}_h^r\right)\boldsymbol{U}_h^\top\boldsymbol{D}^{-1/2}$. The final step is the same as that in Algorithm 3. We define $\hat{\boldsymbol{M}}' = \max(\hat{\boldsymbol{M}}, 1)$ and then perform SVD on $\log \hat{\boldsymbol{M}}'$ to get network embedding.

The largest obstacle is the element-wise matrix logarithm. Since we do not have a good tool to analyze this operator, we have to compute it explicitly even we already have a good low-rank approximation of $\boldsymbol{M}$. We leave this problem for further research.

Proofs and Analysis For NetMF with large window size, we develop the following error bound theorem for the approximation of $\boldsymbol{M}$ and the approximation of $\log \boldsymbol{M}'$.

Theorem 3.4. *Let $\|\cdot\|_F$ be matrix Frobenius norm. Then*

$$\left\|\boldsymbol{M} - \hat{\boldsymbol{M}}\right\|_F \le \frac{\mathrm{vol}(G)}{bd_{\min}}\sqrt{\sum_{j=k+1}^{n}\left|\frac{1}{T}\sum_{r=1}^{T}\lambda_j^r\right|^2};$$

$$\left\|\log \boldsymbol{M}' - \log \hat{\boldsymbol{M}}'\right\|_F \le \left\|\boldsymbol{M}' - \hat{\boldsymbol{M}}'\right\|_F \le \left\|\boldsymbol{M} - \hat{\boldsymbol{M}}\right\|_F.$$

Proof. See Appendix. □

Thm. 3.4 reveals the fact that, even we do not have a tool to analyze element-wise matrix logarithm, the error bound for the approximation of $\log \boldsymbol{M}'$ is bounded by the error bound for the approximation of $\boldsymbol{M}$.

4 EXPERIMENTS

In this section, we evaluate the proposed NetMF method on the node multi-label classification task, which was also used in the works of DeepWalk, LINE, and node2vec.

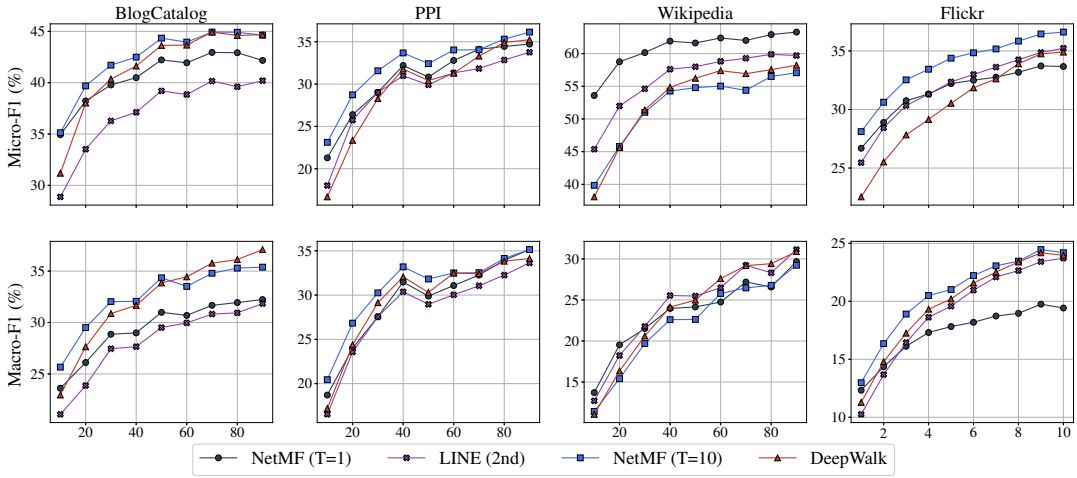


Figure 2: Predictive performance on varying the ratio of traing data. The x-axis represents the ratio of labeled data (%), and the y-axis in the top and bottom rows denote the Micro-F1 and Macro-F1 scores respectively.

Table 2: Statistics of Datasets.

Dataset	BlogCatalog	PPI	Wikipedia	Flickr
$\|V\|$	10,312	3,890	4,777	80,513
$\|E\|$	333,983	76,584	184,812	5,899,882
#Labels	39	50	40	195

Datasets We employ four widely-used datasets for this task. The statistics of these datasets are listed in Table 2.

BlogCatalog [36] is a network of social relationships of the online bloggers. The node labels represent interests of bloggers.

Protein-Protein Interactions (PPI) [32] is a subgraph of PPI network for Homo Sapiens. The labels are obtained from the hallmark gene sets and represent biological states.

Wikipedia[3] is a co-occurrence network of words appearing in the first million bytes of the Wikipedia dump. The labels are the Part-of-Speech (POS) tags inferred by Stanford POS-Tagger [38].

Flickr [36] is a network of the contacts between users in Flickr. The labels represent the interest groups of the users.

Baseline Methods We compare our methods NetMF ($T = 1$) and NetMF ($T = 10$) with LINE (2nd) [35] and DeepWalk [28], which we have introduced in Sections 2.1 and 2.2, respectively. For NetMF ($T = 10$), we set $h = 1024$. For DeepWalk, we present its results with the authors' preferred parameters—window size 10, walk length 40, and the number of walks starting from each vertex to be 80. Finally, for all methods, the embedding dimension is set to be 128.

Prediction Setting Following exactly the same experimental procedure and treatment in DeepWalk [28], we randomly sample a portion of labeled vertices for training, and use the rest for testing. For BlogCatalog, PPI, and Wiki datasets, the training ratio is varied from 10% to 90%. For Flickr, the training ratio is varied from 1% to 10%. We use the one-vs-rest logistic regression model implemented by LIBLINEAR [13] for the multi-label classification task. In the test phase, the one-vs-rest model yields a ranking of labels rather than an exact label assignment. To avoid the thresholding effect [37], we assume that the number of labels for test data is given [28, 37]. We repeat the prediction procedure 10 times and evaluate the performance of different approaches in terms of average Micro-F1 and average Macro-F1 [40].

Experimental Results Figure 2 summarizes the prediction performance of all methods on the four datasets and Table 3 lists the quantitative and relative gaps between our methods and baselines. In specific, we show NetMF ($T = 1$)'s relative performance gain over LINE (2nd) and NetMF ($T = 10$)'s relative improvements over DeepWalk, respectively, as each pair of them share the same parameter setting. In general, we have the following key observations and insights:

(1) As one can see from Figure 2, in most of cases, the proposed NetMF methods achieve better predictive performance over baseline approaches as measured by both Micro-F1 and Macro-F1, demonstrating the effectiveness of the theoretical foundation we lay out for network embedding.

(2) As shown in Table 3, the proposed methods outperform DeepWalk and LINE by large margins when sparse labeled vertices are provided. Take the PPI dataset with 10% training data as an example, NetMF ($T = 1$) achieves relatively 18.15% and 13.19% performance gains over LINE (2nd) in terms of Micro-F1 and Macro-F1 scores, respectively. More impressively, NetMF ($T = 10$) outperforms DeepWalk by 38.61% and 18.96% relatively as measured by two metrics.

(3) DeepWalk and LINE (2nd) become more comparable to NetMF as the increase of training data, especially in BlogCatalog and PPI. However, in other datasets such as Flickr, as the training ratio increases, NetMF ($T = 10$) achieves >4.9% improvements over DeepWalk in terms of Micro-F1.

(4) In Wikipedia, NetMF ($T = 1$) shows better performance than NetMF ($T = 10$), DeepWalk and LINE, which implies that short-term dependence is enough to model its network structure. This is because Wikipedia is a dense word co-occurrence network (average degree is 77.11) in which edges exist between words co-occurring in a two-length window in the Wikipedia corpus.

(5) DeepWalk actually tries to estimate the true vertex-context joint distribution with an empirical one through random walk sampling. Although the consistency can be guaranteed by the law of

[3] http://mattmahoney.net/dc/text.html

Table 3: Micro/Macro-F1 Score(%) for Multilabel Classification on BlogCatalog, PPI, Wikipedia, and Flickr datasets. In Flickr, 1% of vertices are labeled for training, and in the other three datasets, 10% of vertices are labeled for training.

Algorithm	BlogCatalog (10%)		PPI (10%)		Wikipeida (10%)		Flickr (1%)	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
LINE (2nd)	10.25	21.07	18.02	16.53	45.38	12.71	25.46	10.25
NetMF ($T = 1$)	12.33	23.61	21.29	18.71	53.59	13.71	26.69	12.33
Relative Gain of NetMF ($T = 1$)	20.29%	12.06%	18.15%	13.19%	18.09%	7.87%	4.83%	20.29%
DeepWalk	11.28	22.95	16.68	17.19	38.09	11.06	22.54	11.28
NetMF ($T = 10$)	12.98	25.66	23.12	20.45	39.85	11.41	28.11	12.98
Relative Gain of NetMF ($T = 10$)	15.07%	11.81%	38.61%	18.96%	4.62%	3.16%	24.71%	15.07%

large numbers, the gap between the true and estimated distributions may still negatively affect their prediction performance due to the large size of real-world networks.

5 RELATED WORK

The story of network embedding stems from Spectral Clustering [4, 43], a general data clustering technique, which selects eigenvalues/eigenvectors of a data affinity matrix to obtain a data representation that can be clustered or embedded in a low-dimensional space. Spectral Clustering has been widely used in fields such as community detection [20] and image segmentation [31]. In recent years, there is an increasing interest in network embedding. Following a few pioneer works such as SocDim [36], DeepWalk [28], LINE [35], a growing number of literature has paid the attention to this area, for example, heterogeneous network embedding [7, 11, 17, 34], direct network embedding [27], semi-supervised network embedding [18, 42, 49], network embedding with rich vertex attributes [41, 47], network embedding with high order structure [5, 15], network embedding via deep neural network [6, 22, 44], signed network embedding [9], etc.

Among the above papers, a wildly used technique is to define "context" for each vertex, and then train a predictive model to perform context prediction. For example, DeepWalk [28] and node2vec [15] define vertices' context by the 1st and 2nd-order random walks, respectively; LINE defines a vertex's context by its neighbors; Metapath2vec [11] defines vertex's context by meta path [33]. These ideas of using "context" are mostly motivated by a wildly used predictive model, i.e., skip-gram with negative sampling (SGNS) [26]. Most recently, an excellent work written by Levy and Goldberg [21] proves that SGNS is actually conducting implicit matrix factorization, which provides us a tool to analyze the above network embedding models. Besides the analysis in Levy and Goldberg [21], we also utilize the conclusions from random walk on graph [23], spectral graph theory [10] and matrix analysis [16, 39] to help understand network embedding models.

6 CONCLUSION

In this work, we provide a theoretical analysis of four impactful network embedding methods—DeepWalk, LINE, PTE, and node2vec—that were recently proposed between year 2014 and year 2016. We show that all of the four methods are essentially performing implicit matrix factorization and the closed forms of their matrices offer not only the relationships between those methods but also their intrinsic connection with graph Laplacian. We further propose NetMF—a framework to explicitly factorize the closed-form matrices that DeepWalk and LINE aim to implicitly approximate and factorize. Our extensive experiments suggest that NetMF's direct factorization achieves consistent performance improvement over DeepWalk and LINE.

In the future, we would like to further explore promising directions to advance network embedding. It would be necessary to investigate whether and how the development in random-walk polynomials [8] can support fast approximations of the closed-form matrices. The computation and approximation of the 2nd-order random walks employed by node2vec is another interesting topic to follow. In addition, it is exciting to study the nature of skip-gram based dynamic and heterogeneous network embedding.

APPENDIX:

Theorem 3.4. *Let $\|\cdot\|_F$ be matrix Frobenius norm. Then*

$$\left\|\boldsymbol{M}-\hat{\boldsymbol{M}}\right\|_F \leq \frac{\mathrm{vol}(G)}{b d_{\min}} \sqrt{\sum_{j=k+1}^{n}\left|\frac{1}{T}\sum_{r=1}^{T}\lambda_j^r\right|^2};$$

$$\left\|\log \boldsymbol{M}' - \log \hat{\boldsymbol{M}}'\right\|_F \leq \left\|\boldsymbol{M}' - \hat{\boldsymbol{M}}'\right\|_F \leq \left\|\boldsymbol{M}-\hat{\boldsymbol{M}}\right\|_F.$$

Proof. The first inequality can be seen by applying the definition of Frobenius norm and Eq. 10.

For the second inequality, first to show $\left\|\log \boldsymbol{M}' - \log \hat{\boldsymbol{M}}'\right\|_F \leq \left\|\boldsymbol{M}' - \hat{\boldsymbol{M}}'\right\|_F$. According to the definition of Frobenius norm, sufficient to show $\left|\log \boldsymbol{M}'_{i,j} - \log \hat{\boldsymbol{M}}'_{i,j}\right| \leq \left|\hat{\boldsymbol{M}}'_{i,j} - \boldsymbol{M}'_{i,j}\right|$ for any i, j. Without loss of generality, assume $\boldsymbol{M}'_{i,j} \leq \hat{\boldsymbol{M}}'_{i,j}$.

$$\left|\log \boldsymbol{M}'_{i,j} - \log \hat{\boldsymbol{M}}'_{i,j}\right| = \log \frac{\hat{\boldsymbol{M}}'_{i,j}}{\boldsymbol{M}'_{i,j}} = \log\left(1 + \frac{\hat{\boldsymbol{M}}'_{i,j} - \boldsymbol{M}'_{i,j}}{\boldsymbol{M}'_{i,j}}\right)$$

$$\leq \frac{\hat{\boldsymbol{M}}'_{i,j} - \boldsymbol{M}'_{i,j}}{\boldsymbol{M}'_{i,j}} \leq \hat{\boldsymbol{M}}'_{i,j} - \boldsymbol{M}'_{i,j} = \left|\hat{\boldsymbol{M}}'_{i,j} - \boldsymbol{M}'_{i,j}\right|,$$

where the first inequality is because $\log(1+x) \leq x$ for $x \geq 0$, and the the second inequality is because $\boldsymbol{M}'_{i,j} = \max\left(\boldsymbol{M}_{i,j}, 1\right) \geq 1$. Next to show $\left\|\boldsymbol{M}' - \hat{\boldsymbol{M}}'\right\|_F \leq \left\|\boldsymbol{M} - \hat{\boldsymbol{M}}\right\|_F$. Sufficient to show $\left|\boldsymbol{M}'_{i,j} - \hat{\boldsymbol{M}}'_{i,j}\right| \leq \left|\boldsymbol{M}_{i,j} - \hat{\boldsymbol{M}}_{i,j}\right|$ for any i, j. Recall the definition of $\boldsymbol{M}'$ and $\hat{\boldsymbol{M}}'$, we get $\left|\boldsymbol{M}'_{i,j} - \hat{\boldsymbol{M}}'_{i,j}\right| = \left|\max(\boldsymbol{M}_{i,j}, 1) - \max(\hat{\boldsymbol{M}}_{i,j}, 1)\right| \leq \left|\boldsymbol{M}_{i,j} - \hat{\boldsymbol{M}}_{i,j}\right|$. □

REFERENCES

[1] Yoshua Bengio, Aaron Courville, and Pierre Vincent. 2013. Representation learning: A review and new perspectives. *IEEE TPAMI* 35, 8 (2013), 1798–1828.

[2] Austin R Benson, David F Gleich, and Jure Leskovec. 2015. Tensor spectral clustering for partitioning higher-order network structures. In *SDM*. SIAM, 118–126.

[3] Austin R Benson, David F Gleich, and Lek-Heng Lim. 2017. The Spacey Random Walk: A stochastic Process for Higher-Order Data. *SIAM Rev.* 59, 2 (2017), 321–345.

[4] Matthew Brand and Kun Huang. 2003. A unifying theorem for spectral embedding and clustering.. In *AISTATS*.

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

[6] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2016. Deep Neural Networks for Learning Graph Representations.. In *AAAI*. 1145–1152.

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

[8] Dehua Cheng, Yu Cheng, Yan Liu, Richard Peng, and Shang-Hua Teng. 2015. Efficient sampling for Gaussian graphical models via spectral sparsification. In *COLT*. 364–390.

[9] Kewei Cheng, Jundong Li, and Huan Liu. 2017. Unsupervised Feature Selection in Signed Social Networks. In *KDD*. ACM, n/a.

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

[11] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable Representation Learning for Heterogeneous Networks. In *KDD*. n/a.

[12] David Easley and Jon Kleinberg. 2010. *Networks, Crowds, and Markets: Reasoning about a Highly Connected World*. Cambridge University Press.

[13] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A library for large linear classification. *JMLR* 9, Aug (2008), 1871–1874.

[14] David F Gleich, Lek-Heng Lim, and Yongyang Yu. 2015. Multilinear pagerank. *SIAM J. Matrix Anal. Appl.* 36, 4 (2015), 1507–1541.

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

[16] Roger A. Horn and Charles R. Johnson. 1991. *Topics in Matrix Analysis*. Cambridge University Press. https://doi.org/10.1017/CBO9780511840371

[17] Yann Jacob, Ludovic Denoyer, and Patrick Gallinari. 2014. Learning latent representations of nodes for classifying in heterogeneous social networks. In *WSDM*. ACM, 373–382.

[18] Thomas N Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv preprint arXiv:1609.02907* (2016).

[19] Richard B Lehoucq, Danny C Sorensen, and Chao Yang. 1998. *ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods*. SIAM.

[20] Jure Leskovec, Kevin J Lang, and Michael Mahoney. 2010. Empirical comparison of algorithms for network community detection. In *WWW*. ACM, 631–640.

[21] Omer Levy and Yoav Goldberg. 2014. Neural Word Embedding as Implicit Matrix Factorization. In *NIPS*. 2177–2185.

[22] Hang Li, Haozheng Wang, Zhenglu Yang, and Masato Odagaki. 2017. Variation Autoencoder Based Network Representation Learning for Classification. In *ACL*. 56.

[23] László Lovász. 1993. Random walks on graphs. *Combinatorics, Paul erdos is eighty* 2 (1993), 1–46.

[24] Qing Lu and Lise Getoor. 2003. Link-based Classification. In *ICML*.

[25] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *CoRR* abs/1301.3781 (2013). http://arxiv.org/abs/1301.3781

[26] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *NIPS*. 3111–3119.

[27] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric Transitivity Preserving Graph Embedding.. In *KDD*. 1105–1114.

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

[29] Bryan Perozzi, Vivek Kulkarni, and Steven Skiena. [n. d.]. Don't Walk, Skip! Online Learning of Multi-scale Network Embeddings. In *ASONAM*. IEEE/ACM.

[30] S Yu Philip, Jiawei Han, and Christos Faloutsos. 2010. *Link mining: Models, algorithms, and applications*. Springer.

[31] Jianbo Shi and Jitendra Malik. 2000. Normalized cuts and image segmentation. *IEEE PAMI* 22, 8 (2000), 888–905.

[32] Chris Stark, Bobby-Joe Breitkreutz, Andrew Chatr-Aryamontri, Lorrie Boucher, Rose Oughtred, Michael S Livstone, Julie Nixon, Kimberly Van Auken, Xiaodong Wang, Xiaoqi Shi, et al. 2010. The BioGRID interaction database: 2011 update. *Nucleic acids research* 39, suppl_1 (2010), D698–D704.

[33] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S Yu, and Tianyi Wu. 2011. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. In *PVLDB*, Vol. 4. 992–1003.

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

[35] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW*. 1067–1077.

[36] Lei Tang and Huan Liu. 2009. Relational learning via latent social dimensions. In *KDD*. ACM, 817–826.

[37] Lei Tang, Suju Rajan, and Vijay K Narayanan. 2009. Large scale multi-label classification via metalabeler. In *WWW*. ACM, 211–220.

[38] Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *NAACL*. Association for Computational Linguistics, 173–180.

[39] Lloyd N Trefethen and David Bau III. 1997. *Numerical linear algebra*. Vol. 50. Siam.

[40] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. 2009. Mining multi-label data. In *Data mining and knowledge discovery handbook*. Springer, 667–685.

[41] Cunchao Tu, Han Liu, Zhiyuan Liu, and Maosong Sun. 2017. Cane: Context-aware network embedding for relation modeling. In *ACL*. n/a.

[42] Cunchao Tu, Weicheng Zhang, Zhiyuan Liu, and Maosong Sun. 2016. Max-Margin DeepWalk: Discriminative Learning of Network Representation.. In *IJCAI*. 3889–3895.

[43] Ulrike Von Luxburg. 2007. A tutorial on spectral clustering. *Statistics and computing* 17, 4 (2007), 395–416.

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

[45] Christopher KI Williams and Matthias Seeger. 2001. Using the Nyström method to speed up kernel machines. In *NIPS*. 682–688.

[46] Cheng Yang and Zhiyuan Liu. 2015. Comprehend deepwalk as matrix factorization. *arXiv preprint arXiv:1501.00358* (2015).

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

[48] Cheng Yang, Maosong Sun, Zhiyuan Liu, and Cunchao Tu. 2017. Fast Network Embedding Enhancement via High Order Proximity Approximation. In *IJCAI*.

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