NETWORK EMBEDDING AS MATRIX FACTORIZATION

Jiezhon Qiu[1], **Yuxiao Dong**[2], **Hao Ma**[3]*, **Jian Li**[1], **Chi Wang**[2], **Kuansan Wang**[2], **Jie Tang**[1]
[1]Tsinghua University
[2]Microsoft Research, Redmond
[3]Facebook AI
qiujz16@mails.tsinghua.edu.cn
{yuxdong, wang.chi, kuansanw}@microsoft.com
haom@fb.com
{lijian83, jietang}@tsinghua.edu.cn

ABSTRACT

In this work, we analyze modern skip-gram with negative sampling base network embedding models, and show that all of them can be unified into the matrix factorization framework with closed forms. We then present network embedding as matrix factorization (NetMF) method which aims to factorize the closed form matrix directly. However, constructing and factorizing this matrix—which is dense—is prohibitively expensive in terms of both time and space, making it not scalable for large networks. We further present the algorithm of large-scale network embedding as sparse matrix factorization (NetSMF). NetSMF leverages theories from spectral sparsification to efficiently sparsify the aforementioned dense matrix, enabling significantly improved efficiency in embedding learning. The sparsified matrix is spectrally close to the original dense one with a theoretically bounded approximation error, which helps maintain the representation power of the learned embeddings. We conduct experiments on networks of various scales and types. Results show that among both popular benchmarks and factorization based methods, NetSMF is the only method that achieves both high efficiency and effectiveness.

1 INTRODUCTION

Recent years have witnessed the emergence of network embedding, which offers a revolutionary paradigm for modeling graphs and networks (Hamilton et al., 2017). The goal of network embedding is to automatically learn latent representations for objects in networks, such as vertices and edges. Over the course of its development, the DeepWalk (Perozzi et al., 2014), LINE (Tang et al., 2015), and node2vec (Grover & Leskovec, 2016) models have been commonly considered as powerful benchmark solutions for evaluating network embedding research.

We first shows that both the DeepWalk and LINE methods can be viewed as implicit factorization of a closed-form matrix (Qiu et al., 2018). Building upon this theoretical foundation, the NetMF method was instead proposed to explicitly factorize this matrix, achieving more effective embeddings than DeepWalk and LINE. Unfortunately, it turns out that the matrix to be factorized is an $n \times n$ dense one with n being the number of vertices in the network, making it prohibitively expensive to directly construct and factorize for large-scale networks.

In light of these limitations of existing methods (See the summary in Table 1), we propose to study representation learning for large-scale networks with the goal of achieving efficiency, capturing global structural contexts, and having theoretical guarantees. Our idea is to find a sparse matrix that is spectrally close to the dense NetMF matrix implicitly factorized by DeepWalk. The sparsified matrix requires a lower cost for both construction and factorization. Meanwhile, making it spectrally close to the original NetMF matrix can guarantee that the spectral information of the network is maintained, and the embeddings learned from the sparse matrix is as powerful as those learned from the dense NetMF matrix.

*Work performed while at Microsoft Research

Table 1: The comparison between NetSMF and other popular network embedding algorithms.

	LINE	DeepWalk	node2vec	NetMF	NetSMF
Efficiency	√				√
Global context		√	√	√	√
Theoretical guarantee				√	√
High-order proximity			√		

In this work, we present the solution to network embedding learning as sparse matrix factorization (NetSMF). NetSMF comprises three steps. First, it leverages the spectral graph sparsification technique (Cheng et al., 2015b;a) to find a sparsifier for a network's random-walk matrix-polynomial. Second, it uses this sparsifier to construct a matrix with significantly fewer non-zeros than, but spectrally close to, the original NetMF matrix. Finally, it performs randomized singular value decomposition to efficiently factorize the sparsified NetSMF matrix, yielding the embeddings for the network.

With this design, NetSMF offers both efficiency and effectiveness with guarantees, as the approximation error of the sparsified matrix is theoretically bounded. We conduct experiments in five networks, which are representative of different scales and types. Experimental results show that for million-scale or larger networks, NetSMF achieves orders of magnitude speedup over NetMF, while maintaining competitive performance for the vertex classification task. In other words, both NetSMF and NetMF outperform well-recognized network embedding benchmarks (i.e., DeepWalk, LINE, and node2vec), but NetSMF addresses the computation challenge faced by NetMF.

2 Network Embedding as Matrix Factorization (NetMF)

In this section, we analyze DeepWalk (Perozzi et al., 2014) and illustrate the essence of DeepWalk is actually performing an implicit matrix factorization (See the matrix form solution in Thm. 1).

DeepWalk first generates a "corpus" $\mathcal{D}$ by random walks on graphs (Lovász et al., 1993). To be formal, the corpus $\mathcal{D}$ is a multiset that counts the multiplicity of vertex-context pairs. DeepWalk then trains a skip-gram model on the multiset $\mathcal{D}$. In this work, we focus on skip-gram with negative sampling (SGNS). 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 the multiset $\mathcal{D}$. Once we have $\mathcal{D}$, we run an SGNS to attain the network embedding (Line 8). Next, we introduce some necessary background about the SGNS technique, followed by our analysis of the DeepWalk method.

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 $w_1, \cdots w_L$ and the contexts for word w_i are words surrounding it in a T-sized window $w_{i-T}, \cdots, w_{i-1}, w_{i+1}, \cdots, w_{i+T}$. Following the work by Levy & Goldberg (2014), SGNS is implicitly factorizing

$$\log\left(\frac{\#(w,c)\,|\mathcal{D}|}{\#(w)\cdot\#(c)}\right) - \log b, \tag{1}$$

where $\#(w,c)$, $\#(w)$ and $\#(c)$ denote the number of times word-context pair (w,c), word w and context c appear in the corpus, respectively; b is the number of negative samples.

Proofs and Analysis Our analysis of DeepWalk is based on the following key assumptions. Firstly, assume the used graph is undirected, connected, and non-bipartite, making $P(w) = d_w/\operatorname{vol}(G)$ a unique stationary distribution of the random walks. Secondly, suppose the first vertex of a random walk is sampled from the stationary distribution $P(w) = d_w/\operatorname{vol}(G)$. To characterize DeepWalk, we want to reinterpret Eq. 1 by using graph theory terminologies. We have the following theorem. The detailed proofs is shown in Appendix (See section 6.1).

Theorem 1. *For DeepWalk, when $L \to \infty$,*

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

Algorithm 1: DeepWalk

```
for n = 1, 2, ..., N do
    Pick w_1^n according to a probability distribution P(w_1);
    Generate a vertex sequence (w_1^n, ..., w_L^n) of length L by a random walk on network G;
    for j = 1, 2, ..., L − T do
        for r = 1, ..., T do
            Add vertex-context pair (w_j^n, w_{j+r}^n) to multiset D;
            Add vertex-context pair (w_{j+r}^n, w_j^n) to multiset D;
Run SGNS on D with b negative samples.
```

In matrix form, DeepWalk is equivalent to factorize

$$\log^{\circ}\left(\frac{\mathrm{vol}(G)}{bT}\left(\sum_{r=1}^{T}\boldsymbol{P}^{r}\right)\boldsymbol{D}^{-1}\right). \tag{2}$$

The matrix in equation 2 offers an alternative view of the skip-gram based network embedding methods. We propose an explicit matrix factorization approach named NetMF to learn the embeddings. Note that the matrix in equation 2 would be ill-defined if there exist a pair of vertices unreachable in T hops, because $\log(0) = -\infty$. So following Levy & Goldberg (2014), NetMF uses the logarithm truncated at point one, that is, $\mathrm{trunc\_log}(x) = \max(0, \log(x))$. Thus, NetMF targets to factorize the matrix

$$\mathrm{trunc\_log}^{\circ}\left(\frac{\mathrm{vol}(G)}{b}\boldsymbol{M}\right), \tag{3}$$

where $\boldsymbol{M} = \frac{1}{T}\sum_{r=1}^{T}(\boldsymbol{D}^{-1}\boldsymbol{A})^{r}\boldsymbol{D}^{-1}$. In the rest of this work, we refer to the matrix in equation 3 as *the NetMF matrix*.

3 Network Embedding as Sparse Matrix Factorization (NetSMF)

In this section, we develop network embedding as sparse matrix factorization (NetSMF) (Qiu et al., 2019). We present the NetSMF method to construct and factorize a sparse matrix that approximates the dense NetMF matrix. The main technique we leverage is random-walk matrix-polynomial (molynomial) sparsification.

3.1 Random-Walk Molynomial Sparsification

We first introduce the definition of spectral similarity and the theorem of random-walk molynomial sparsification.

Definition 1. *(Spectral Similarity of Networks) Suppose $G = (V, E, \boldsymbol{A})$ and $\widetilde{G} = (V, \widetilde{E}, \widetilde{\boldsymbol{A}})$ are two weighted undirected networks. Let $\boldsymbol{L} = \boldsymbol{D}_G - \boldsymbol{A}$ and $\widetilde{\boldsymbol{L}} = \boldsymbol{D}_{\widetilde{G}} - \widetilde{\boldsymbol{A}}$ be their Laplacian matrices, respectively. We define G and $\widetilde{G}$ are $(1+\epsilon)$-spectrally similar if*

$$\forall \boldsymbol{x} \in \mathbb{R}^{n}, (1-\epsilon)\cdot \boldsymbol{x}^{\top}\widetilde{\boldsymbol{L}}\boldsymbol{x} \leq \boldsymbol{x}^{\top}\boldsymbol{L}\boldsymbol{x} \leq (1+\epsilon)\cdot \boldsymbol{x}^{\top}\widetilde{\boldsymbol{L}}\boldsymbol{x}.$$

Theorem 2. *(Spectral Sparsifiers of Random-Walk Molynomials (Cheng et al., 2015a;b)) For random-walk molynomial*

$$\boldsymbol{L} = \boldsymbol{D} - \sum_{r=1}^{T}\alpha_r \boldsymbol{D}\left(\boldsymbol{D}^{-1}\boldsymbol{A}\right)^{r}, \tag{4}$$

where $\sum_{r=1}^{T}\alpha_r = 1$ and α_r non-negative, one can construct, in time $O(T^2 m\epsilon^{-2}\log^2 n)$, a $(1+\epsilon)$-spectral sparsifier, $\widetilde{\boldsymbol{L}}$, with $O(n\log n\epsilon^{-2})$ non-zeros. For unweighted graphs, the time complexity can be reduced to $O(T^2 m\epsilon^{-2}\log n)$.

To achieve a sparsifier $\widetilde{\boldsymbol{L}}$ with $O(\epsilon^{-2}n\log n)$ non-zeros, the sparsification algorithm consists of two steps: The first step obtains an initial sparsifier for $\boldsymbol{L}$ with $O(Tm\epsilon^{-2}\log n)$ non-zeros. The second

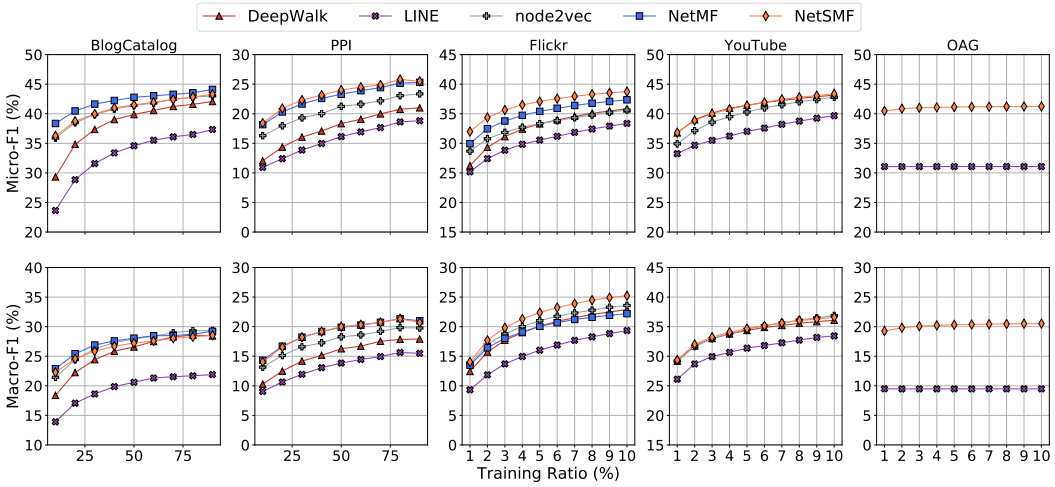

Figure 1: Predictive performance w.r.t. the ratio of training data. The x-axis represents the ratio of labeled data (%), and the y-axis in the top and bottom rows denote the Micro-F1 and Macro-F1 scores (%) respectively. For methods which fail to finish computation in one week or cannot handle the computation, their results are not available and thus not plotted in this figure.

step then applies the standard spectral sparsification algorithm (Spielman & Srivastava, 2011) to further reduce the number of non-zeros to $O(\epsilon^{-2} n \log n)$. In this work, we only adopt the first step because a sparsifier with $O(Tm\epsilon^{-2} \log n)$ non-zeros is sparse enough for our task. Thus we skip the second step that involves additional computations. From now on, when referring to the random-walk molynomial sparsification algorithm in this work, we mean its first step only.

One can immediately observe that, if we set $\alpha_r = \frac{1}{T}, r \in [T]$, the matrix $\boldsymbol{L}$ in Eq. equation 4 has a strong connection with the desired matrix $\boldsymbol{M}$ in Eq. equation **??**. Formally, we have the following equation

$$\boldsymbol{M} = \boldsymbol{D}^{-1} \left(\boldsymbol{D} - \boldsymbol{L} \right) \boldsymbol{D}^{-1}. \tag{5}$$

Thm. 2 can help us construct a sparsifier $\widetilde{\boldsymbol{L}}$ for matrix $\boldsymbol{L}$. Then we define $\widetilde{\boldsymbol{M}} \triangleq \boldsymbol{D}^{-1}(\boldsymbol{D} - \widetilde{\boldsymbol{L}})\boldsymbol{D}^{-1}$ by replacing $\boldsymbol{L}$ in Eq. equation 5 with its sparsifier $\widetilde{\boldsymbol{L}}$. One can observe that matrix $\widetilde{\boldsymbol{M}}$ is still a sparse one with the same order of magnitude of non-zeros as $\widetilde{\boldsymbol{L}}$. Consequently, instead of factorizing the dense NetMF matrix in Eq. equation 3, we can factorize its sparse alternative, i.e.,

$$\text{trunc\_log}^{\circ} \left(\frac{\text{vol}(G)}{b} \widetilde{\boldsymbol{M}} \right). \tag{6}$$

In the rest of this work, the matrix in equation 6 is referred to as *the NetMF matrix sparsifier*. Limited by space, we introduce our NetSMF algorithm in Appendix (see section 6.2).

4 Experimental Results

We evaluate the proposed NetSMF method on the multi-label vertex classification task, against NetMF (Qiu et al., 2018), LINE (Tang et al., 2015), DeepWalk (Perozzi et al., 2014), and node2vec (Grover & Leskovec, 2016). Among these four methods, DeepWalk achieves neither efficiency nor effectiveness superiority; node2vec achieves relatively good performance at the cost of efficiency; NetMF achieves effectiveness at the expense of significantly increased time and space costs; NetSMF is the only method that achieves both high efficiency and effectiveness, empowering it to learn effective embeddings for billion-scale networks (e.g., the OAG network with 0.9 billion edges) in one day on one modern server.

5 CONCLUSION

In this work, we study network embedding with the goal of achieving both good efficiency and effectiveness. To address the efficiency and scalability challenges faced by the NetMF model, we propose to study large-scale network embedding as sparse matrix factorization. We present the NetSMF algorithm, which achieves a sparsification of the (dense) NetMF matrix. In the future, we would like to efficiently and accurately learn embeddings for large-scale directed Cohen et al. (2016), dynamic Kapralov et al. (2017), and/or heterogeneous networks.

REFERENCES

Raymond B Cattell. The scree test for the number of factors. *Multivariate behavioral research*, 1 (2):245–276, 1966.

Dehua Cheng, Yu Cheng, Yan Liu, Richard Peng, and Shang-Hua Teng. Efficient sampling for gaussian graphical models via spectral sparsification. In *COLT '15*, pp. 364–390, 2015a.

Dehua Cheng, Yu Cheng, Yan Liu, Richard Peng, and Shang-Hua Teng. Spectral sparsification of random-walk matrix polynomials. *arXiv preprint arXiv:1502.03496*, 2015b.

Michael B Cohen, Jonathan Kelner, John Peebles, Richard Peng, Aaron Sidford, and Adrian Vladu. Faster algorithms for computing the stationary distribution, simulating random walks, and more. In *FOCS '16*, pp. 583–592. IEEE, 2016.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD '16*, pp. 855–864. ACM, 2016.

Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53 (2):217–288, 2011.

William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data(base) Engineering Bulletin*, 40:52–74, 2017.

Michael Kapralov, Yin Tat Lee, CN Musco, CP Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. *SIAM Journal on Computing*, 46(1):456–477, 2017.

Omer Levy and Yoav Goldberg. Neural Word Embedding as Implicit Matrix Factorization. In *NIPS '14*, pp. 2177–2185. 2014.

László Lovász et al. Random walks on graphs: A survey. *Combinatorics, Paul erdos is eighty*, 2(1): 1–46, 1993.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *KDD '14*, pp. 701–710. ACM, 2014.

Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *WSDM '18*, pp. 459–467. ACM, 2018.

Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Chi Wang, Kuansan Wang, and Jie Tang. Netsmf: Large-scale network embedding as sparse matrix factorization. In *WWW '19*, 2019.

A.N. Shiryaev and A. Lyasoff. *Problems in Probability*. Problem Books in Mathematics. Springer New York, 2012. ISBN 9781461436881.

Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *WWW '15*, pp. 1067–1077, 2015.

6 APPENDIX

6.1 ANALYSIS AND PROOFS FOR DEEPWALK

To characterize DeepWalk, we want to reinterpret Eq. 1 by using graph theory terminologies. Our general idea is to partition the multiset $\mathcal{D}$ into several sub-multisets according to the way in which vertex and its context appear in a random walk sequence. More formally, for $r = 1, \cdots, T$, we define

$$\mathcal{D}_{\overrightarrow{r}} = \left\{(w, c) : (w, c) \in \mathcal{D}, w = w_j^n, c = w_{j+r}^n\right\},$$
$$\mathcal{D}_{\overleftarrow{r}} = \left\{(w, c) : (w, c) \in \mathcal{D}, w = w_{j+r}^n, c = w_j^n\right\}.$$

That is, $\mathcal{D}_{\overrightarrow{r}}$/$\mathcal{D}_{\overleftarrow{r}}$ is the sub-multiset of $\mathcal{D}$ such that the context c is r steps after/before the vertex w in random walks. Moreover, we use $\#(w, c)_{\overrightarrow{r}}$ and $\#(w, c)_{\overleftarrow{r}}$ to denote the number of times vertex-context pair (w, c) appears in $\mathcal{D}_{\overrightarrow{r}}$ and $\mathcal{D}_{\overleftarrow{r}}$, respectively. The following three theorems characterize DeepWalk step by step.

Theorem 3. *Denote $\boldsymbol{P} = \boldsymbol{D}^{-1}\boldsymbol{A}$, when $L \to \infty$, we have*

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

Lemma 1. *(S.N. Bernstein Law of Large Numbers (Shiryaev & Lyasoff, 2012)) Let $Y_1, Y_2 \cdots$ be a sequence of random variables with finite expectation $\mathbb{E}[Y_j]$ and variance $\mathrm{Var}(Y_j) < K$, $j \geq 1$, and covariances are s.t. $\mathrm{Cov}(Y_i, Y_j) \to 0$ as $|i - j| \to \infty$. Then the law of large numbers (LLN) holds.*

Proof. First consider the special case when $N = 1$, thus we only have one vertex sequence $w_1, \cdots, w_L$ generated by random walk as described in Algorithm 1. Consider one certain vertex-context pair (w, c), let Y_j ($j = 1, \cdots, L - T$) be the indicator function for event that $w_j = w$ and $w_{j+r} = c$. We have the following two observations:

- The quantity $\#(w, c)_{\overrightarrow{r}} / |\mathcal{D}_{\overrightarrow{r}}|$ is the sample average of Y_j's, i.e.,

$$\frac{\#(w, c)_{\overrightarrow{r}}}{|\mathcal{D}_{\overrightarrow{r}}|} = \frac{1}{L - T} \sum_{j=1}^{L-T} Y_j.$$

- Based on our assumptions about the graph and the random walk, $\mathbb{E}[Y_j] = \frac{d_w}{\mathrm{vol}(G)} \left(\boldsymbol{P}^r\right)_{w,c}$, and when $j > i + r$,

$$\begin{aligned}\mathbb{E}(Y_i Y_j) &= \mathrm{Prob}(w_i = w, w_{i+r} = c, w_j = w, w_{j+r} = c) \\ &= \frac{d_w}{\mathrm{vol}(G)} \left(\boldsymbol{P}^r\right)_{w,c} \left(\boldsymbol{P}^{j-i-r}\right)_{c,w} \left(\boldsymbol{P}^r\right)_{w,c}.\end{aligned}$$

In this way, we can evaluate the covariance $\mathrm{Cov}(Y_i, Y_j)$ when $j > i + r$ and calculate its limit when $j - i \to \infty$ by using the fact that our random walk converges to its stationary distribution:

$$\begin{aligned}\mathrm{Cov}(Y_i, Y_j) &= \mathbb{E}(Y_i Y_j) - \mathbb{E}(Y_i)\mathbb{E}(Y_j) \\ &= \frac{d_w}{\mathrm{vol}(G)} \left(\boldsymbol{P}^r\right)_{w,c} \underbrace{\left(\left(\boldsymbol{P}^{j-i-r}\right)_{c,w} - \frac{d_w}{\mathrm{vol}(G)}\right)}_{\text{goes to 0 as } j-i\to\infty} \left(\boldsymbol{P}^r\right)_{w,c} \to 0.\end{aligned}$$

Then we can apply Lemma 1 and conclude that the sample average converges in probability towards the expected value, i.e.,

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

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

For the general case when $N > 1$, we define Y_j^n ($n = 1, \cdots, N$, $j = 1, \cdots, L - T$) to be the indicator function for event $w_j^n = w$ and $w_{j+r}^n = c$, and organize Y_j^n's as $Y_1^1, Y_1^2, \cdots, Y_1^N, Y_2^1, Y_2^2, \cdots, Y_2^N, \cdots$. This r.v. sequence still satisfies the condition of S.N. Bernstein LLN, so the above conclusion still holds. □

Theorem 4. *When $L \to \infty$, we have*

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

Proof. Note the fact that $\frac{|\mathcal{D}_{\overrightarrow{r}}|}{|\mathcal{D}|} = \frac{|\mathcal{D}_{\overleftarrow{r}}|}{|\mathcal{D}|} = \frac{1}{2T}$. By using Thm. 3 and the continuous mapping theorem, we get

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

Further, marginalizing w and c respectively reveals the fact that $\frac{\#(w)}{|\mathcal{D}|} \xrightarrow{p} \frac{d_w}{\mathrm{vol}(G)}$ and $\frac{\#(c)}{|\mathcal{D}|} \xrightarrow{p} \frac{d_c}{\mathrm{vol}(G)}$, as $L \to \infty$. □

Theorem 1. *For DeepWalk, when $L \to \infty$,*

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

In matrix form, DeepWalk is equivalent to factorize

$$\log^{\circ}\left(\frac{\mathrm{vol}(G)}{bT}\left(\sum_{r=1}^{T}\boldsymbol{P}^r\right)\boldsymbol{D}^{-1}\right). \tag{2}$$

Proof. Using the results in Thm. 4 and the continuous mapping theorem, we get

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

Write it in matrix form:

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

□

6.2 THE NETSMF ALGORITHM

In this section, we formally describe the NetSMF algorithm, which consists of three steps: random-walk molynomial sparsification, NetMF sparsifier construction, and truncated singular value decomposition.

Step 1: Random-Walk Molynomial Sparsification. To achieve the sparsifier $\widetilde{\boldsymbol{L}}$, we adopt the algorithm in Cheng et al. (2015b). The algorithm starts from creating a network $\widetilde{G}$ that has the same vertex set as G and an empty edge set (Alg. 2, Line 1). Next, the algorithm constructs a sparsifier with $O(M)$ non-zeros by repeating the PathSampling algorithm for M times. In each

Algorithm 2: NetSMF

Input : A social network $G = (V, E, \boldsymbol{A})$ which we want to learn network embedding; The number of non-zeros M in the sparsifier; The dimension of embedding d.
Output: An embedding matrix of size $n \times d$, each row corresponding to a vertex.

1 $\widetilde{G} \leftarrow (V, \emptyset, \widetilde{\boldsymbol{A}} = \mathbf{0})$
`/* Create an empty network with` $E = \emptyset$ `and` $\widetilde{\boldsymbol{A}} = 0$. `*/`
2 **for** $i \leftarrow 1$ **to** M **do**
3 Uniformly pick an edge $e = (u, v) \in E$
4 Uniformly pick an integer $r \in [T]$
5 $u', v', Z \leftarrow$ `PathSampling` (e, r)
6 Add an edge $\left(u', v', \frac{2rm}{MZ}\right)$ to $\widetilde{G}$
`/* Parallel edges will be merged into one edge, with their weights summed up together. */`
7 **end**
8 Compute $\widetilde{\boldsymbol{L}}$ to be the unnormalized graph Laplacian of $\widetilde{G}$
9 Compute $\widetilde{\boldsymbol{M}} = \boldsymbol{D}^{-1}\left(\boldsymbol{D} - \widetilde{\boldsymbol{L}}\right)\boldsymbol{D}^{-1}$
10 $\boldsymbol{U}_d, \boldsymbol{\Sigma}_d, \boldsymbol{V}_d \leftarrow$ `RandomizedSVD` $\left(\text{trunc\_log}^{\circ}\left(\frac{\text{vol}(G)}{b}\widetilde{\boldsymbol{M}}\right), d\right)$
11 **return** $\boldsymbol{U}_d\sqrt{\boldsymbol{\Sigma}_d}$ as network embeddings

Algorithm 3: PathSampling algorithm as described in Cheng et al. (2015b).

Procedure `PathSampling` $(e = (u, v), r)$
Uniformly pick an integer $k \in [r]$
Perform $(k-1)$-step random walk from u to u_0
Perform $(r-k)$-step random walk from v to u_r
Keep track of $Z(\boldsymbol{p})$ along the length-r path $\boldsymbol{p}$ between u_0 and u_r according to Eq. equation 7
return $u_0, u_r, Z(\boldsymbol{p})$

iteration, it picks an edge $e \in E$ and an integer $r \in [T]$ uniformly (Alg. 2, Line 3-4). Then, the algorithm uniformly draws an integer $k \in [r]$ and performs $(k-1)$-step and $(r-k)$-step random walks starting from the two endpoints of edge e respectively (Alg. 3, Line 3-4). The above process samples a length-r path $\boldsymbol{p} = (u_0, u_1, \cdots, u_r)$. At the same time, the algorithm keeps track of $Z(\boldsymbol{p})$, which is defined by

$$Z(\boldsymbol{p}) = \sum_{i=1}^{r} \frac{2}{\boldsymbol{A}_{u_{i-1}, u_i}}, \tag{7}$$

and then adds a new edge (u_0, u_r) with weight $\frac{2rm}{MZ(\boldsymbol{p})}$ to $\widetilde{G}$ (Alg. 2, Line 6). Parallel edges in $\widetilde{G}$ will be merged into one single edge, with their weights summed up together. Finally, the algorithm computes the Laplacian of $\widetilde{G}$, which is the sparsifier $\widetilde{\boldsymbol{L}}$ as we desired (Alg. 2, Line 8). This step gives us a sparsifier with $O(M)$ non-zeros.

Step 2: Construct a NetMF Matrix Sparsifier. As we have discussed at the end of Section 3.1, after constructing a sparsifier $\widetilde{\boldsymbol{L}}$, we can plug it into Eq. equation 5 to obtain a NetMF matrix sparsifier as shown in Eq. equation 6 (Alg. 2, Line 9-10). This step does not change the order of magnitude of non-zeros in the sparsifier.

Step 3: Truncated Singular Value Decomposition. The final step is to perform truncated singular value decomposition (SVD) on the constructed NetMF matrix sparsifier (Eq. equation 6). However, even the sparsifier only has $O(M)$ number of non-zeros, performing exact SVD is still time consuming. In this work, we leverage a modern randomized matrix approximation technique—Randomized SVD—developed by Halko et al. (2011). Due to space constraint, we cannot include many details. Briefly speaking, the algorithm projects the original matrix to a low-dimensional space through a Gaussian random matrix. One only needs to perform traditional SVD (e.g. Jacobi SVD) on a $d \times d$ small matrix. We list the pseudocode algorithm in Alg. 4. Another advantage of SVD is that we can determine the dimensionality of embeddings by using, for example, Cattell's Scree test Cattell

Algorithm 4: Randomized SVD on NetMF Matrix Sparsifier

```
/* In this work, the matrix to be factorized (Eq. equation 6) is an n × n
   symmetric sparse matrix. We store this sparse matrix in a row-major
   way and make use of its symmetry to simplify the computation.      */
1  Procedure RandomizedSVD(A, d)
2      Sampling Gaussian random matrix O                    // O ∈ R^{n×d}
3      Compute sample matrix Y = A^⊤ O = AO                 // Y ∈ R^{n×d}
4      Orthonormalize Y
5      Compute B = AY                                       // B ∈ R^{n×d}
6      Sample another Gaussian random matrix P              // P ∈ R^{d×d}
7      Compute sample matrix of Z = BP                      // Z ∈ R^{n×d}
8      Orthonormalize Z
9      Compute C = Z^⊤ B                                    // C ∈ R^{d×d}
10     Run Jacobi SVD on C = UΣV^⊤
11     return ZU, Σ, YV
/* Result matrices are of shape n × d, d × d, n × d resp.          */
```

Table 2: Time and Space Complexity of NetSMF.

	Time	**Space**
Step 1	$O(MT \log n)$ for weighted networks $O(MT)$ for unweighted networks	$O(M + n + m)$
Step 2	$O(M)$	$O(M + n)$
Step 3	$O(Md + nd^2 + d^3)$	$O(M + nd)$

(1966). In the test, we plot the singular values and select a rank d such that there is a clear drop in the magnitudes or the singular values start to even out. More details will be discussed in Section **??**.

Complexity Analysis. Now we analyze the time and space complexity of NetSMF, as summarized in Table 2. As for step 1, we call the PathSampling algorithm for M times, during each of which it performs $O(T)$ steps of random walks over the network. For unweighted networks, sampling a neighbor requires $O(1)$ time, while for weighted networks, one can use roulette wheel selection to choose a neighbor in $O(\log n)$. It taks $O(M)$ space to store $\widetilde{G}$, while the additional $O(n+m)$ space comes from the storage of the input network. As for step 2, it takes $O(M)$ time to perform the transformation in Eq. equation 5 and the element-wise truncated logarithm in Eq. equation 6. The additional $O(n)$ space is spent in storing the degree matrix. As for step 3, $O(Md)$ time is required to compute the product of a row-major sparse matrix and a dense matrix (Alg. 4, Lines 3 and 5); $O(nd^2)$ time is spent in Gram-Schmidt orthogonalization (Alg. 4, Lines 4 and 8); $O(d^3)$ time is spent in Jacobi SVD (Alg. 4, Line 10).