Biased Walk: A network embedding algorithm and applications

Duong Nguyen
Computer Science and Engineering
The University of California, San Diego
duong@eng.ucsd.edu

Mike Liu
Computer Science and Engineering
The University of California, San Diego
mikeywayglax@gmail.com

ABSTRACT

Network embedding algorithms are powerful tools that can learn latent feature representations of nodes and transform networks into lower dimensional representations. Based on such network embeddings or sets of representation vectors of nodes we can solve a variety graph mining tasks through statistical machine learning techniques. Some typical problems which have been solved effectively based on network embeddings include network clustering, link prediction and multilabel node classification [1], [4]. Though the network embedding problem has been well studied, the existing embedding methods still have limitations. For example, traditional methods such as Isomap[8], Laplacian eigenmap[2], MDS[3], which are based on eigendecomposition of a quadratic-size matrix are not scalable so they cannot handle large networks in practice. Some recent works those are graph-based methods have better performance and are scalable but they only work on specific types of networks (unweighted networks only [1]) or do not perform well on some networks in specific applications such as citation, protein and blog networks [7], [4]. In this paper we propose an embedding algorithm called BiasedWalk, which can be considered as an extension of DeepWalk [1]. Our method can be applied for many kinds of networks (including undirected and weighted ones) on a wide range of network domains. The algorithm's performance is compared to the state-of-the-art network embedding methods on various network datasets.

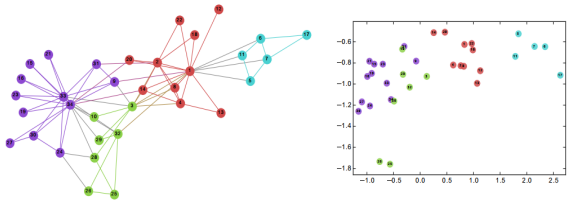
KEYWORDS

Network embedding, representation learning, feature learning in graphs, dimensionality reduction.

ACM Reference format:
Duong Nguyen and Mike Liu. 2016. Biased Walk: A network embedding algorithm and applications. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 6 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Network embedding is a useful tool for representing network nodes in a low-dimensional space. Since many typical machine learning algorithms can work with representation vectors in embedding spaces, we can adopt them to solve various graph mining tasks such as node classification, link prediction and recommendation based on representation results of network embedding methods. This reason makes network embedding receiving much attention of the science community in graph mining even though there have already been many related works on this topic.



(a) Input: Karate Graph (b) Output: Representation

Figure 1: An embedding sample of a karate network (source: Perozzi et al. KDD 2014: Deepwalk).

Formally, a network can be represented as a graph $G = (V, E)$ where V and E is respectively the set of nodes and the set of edges of the network. Depending on specific applications the representation graphs can be different. They can be weighted or unweighted, directed or undirected, for example, graphs representing user friendship in a social network are undirected but graphs of website hyper links are directed, and graphs of fights between countries should be directed and weighted. An embedding of network $G = (V, E)$ is a mapping function $f : V \rightarrow R^d$ where $d \ll |V|$. That means each node of the network would be represented by a vector of d dimensions. Figure 1 gives an example of a network which represents relationship between members in a karate club and an embedding of the network in R^2. Intuitively, an embedding of a network can be considered as a compact and implicit representation of the set of nodes that still preserves information of the network topology.

In general, an embedding function should be satisfied following conditions. First and foremost, an embedding needs to preserve information of network structure. After transforming networks to another space we commonly loss some information of network structure. It is challenging to find an embedding which can preserve network topology that means nodes which are close together in the network should also have small distance to each other in their embedding space. Secondly, embedding methods should be scalable. We expect that they have linear time complexity of the graph size (e.g. linear with the number of nodes and edges) to work with large networks. Thirdly, embedding methods should be able to handle different kinds of networks, for example, directed, undirected, weighted and unweighted networks. In addition, embedding methods which support online learning are highly prefered. In practice

Conference'17, Washington, DC, USA

DOI: 10.1145/nnnnnnn.nnnnnnn

most networks are dynamic and large, calculating embedding for all nodes are expensive so we should avoid repeat such operation.

In this paper we propose a network embedding algorithm called Biased Walk. The algorithm is based on DeepWalk[1] and Skip-gram model[6]. To evaluate the proposed algorithm we compare it to the state of the art embedding methods DeepWalk[1], Line[7] and Node2vec[4] through applying their embeddings to solve the network clustering, link prediction and multilabel classification problems. Their performances are measured on variety kinds of networks including directed/undirected, weighted/unweighted, sparse/very sparse and labeled/unlabeled ones.

2 RELATED WORKS

Existing network embedding methods can be classified into two categories: graph and non-graph based. Traditional methods such as Isomap[8], MDS[3] and Laplacian eigenmap[2] are typical for the first category. These traditional ones find eigen vectors of a certain representation matrix associated to the network and use the eigen vectors to construct embedding vectors for the network nodes. Because finding eigen vectors of a quadratic-size matrix (in the number of network nodes) is very expensive in terms of running time so these methods are not scalable. In addition, methods which use eigen decomposition are often sensitive with noise. Therefore, in some applications since we do not have an accurate and complete networks the embedding results will be strongly affected.

Recent methods such as Deepwalk[1], Line[7] and Node2vec[4] belong to the second category and can handle large networks with million nodes and billion edges. The authors of these methods model network embedding problem as an optimization one which aims to maximize conditional probabilities of observing the neighborhood nodes given network nodes but each of these methods has its own strategy to sample the neighborhood node set for each node. Specifically, Deepwalk uses uniform random walks from a node to sample its neighborhood nodes while Line considers nodes at 1-hop and 2-hops futher and Node2vec adopts biased random walks from the source node. Both Deepwalk and Line are considered as a special case of Node2vec since Node2vec use additional parameters to control probability to select next nodes during random walks. By adjusting those parameters Node2vec is able to guide its walks traveling futher (similar to Deepwalk's manner) or locally going around the source node (as Line's manner). Although Node2vec can give better results than the two methods but that requires efforts to tune parameters manually or doing cross-validation on a training data to select the best ones. Moreover, in some networks the experiment results on [4] show that Node2vec does not give us significant improvements.

3 PROBLEM DEFINITION

Given a network $G = (V, E)$, our aim is to find a mapping $f : V \rightarrow R^d$ where d is the desired number of dimensions. We follow the optimization model on [1], [7] and [4] for the network embedding problem. Let N_u be the set of neighborhood nodes of $u \in V$ and depending on each sampling stategy we can define this set for each node u. The mapping that we are seeking maximizes the sum of log-probabilities of observing network neighborhood $N(u)$ for every node u given its feature representation on f:

$$\arg\max_f \sum_{u \in V} \log Pr(N_u | f(u)). \quad (1)$$

Assuming that the probability of observing a neighborhood node is independent of observing any other neighborhood one given the reresentation of the source node, we have: $Pr(N_u|f(u)) = \prod_{v \in N_u} Pr(v|f(u))$. To make the maximum likelihood optimization problem can be efficiently solve (by the Skip-gram architecture [6]), Node2vec [4] considers the conditional probability of every pair of source-neighborhood nodes as a softmax function of the dot product of their representation features: $Pr(v|f(u)) = \frac{\exp(f(u).f(v))}{\sum_{z \in V} \exp(f(u).f(z))}$. So that the objective function (1) is equivalent to:

$$\arg\max_f \sum_{u \in V} (-N_u \log L_u + \sum_{v \in N_u} f(u).f(v)).$$

Where $L_u = \sum_{z \in V} \exp(f(u).f(z))$.

4 METHODOLOGY

In this section, we will propose a method for learning latent representations of network nodes, which is named BiasedWalk. The method is based on the Skip-gram model [6] and can be considered as an extension of DeepWalk. The Skip-gram model was originally proposed for learning vector representations for words that can preserve a high number of precise syntactic and semantic word relationships. More formally, given a sequence of words the model aims to maximize the sum of probabilities of predicting surrounding words on each window of the same specific size (called context words) on the sequence conditioned to the word at the center of the window. We follow DeepWalk model which considers nodes of a network as words on a vocabulary and sample paths on the network to generate sentences (sequences of words) on a corpus. The difference between BiasedWalk and DeepWalk is that BiasedWalk does not use uniform random walks as DeepWalk but uses biased random walks to sample paths on networks.

More specifically, given a network $G = (V, E)$ with weight function $w : E \rightarrow R^+$. We use random walks starting from every nodes in V. Assuming that a biased walk currently stops at node u, it will consider all neighbor nodes of u as candidates for the next node to visit. Let $\Gamma(u)$ be the set of neighbor nodes of u. The probability p_v of selecting $v \in \Gamma(u)$ as the next node is proportional to $f_{uv} = L_{uv}.w_{uv}$, where L_{uv} is the value measuring the similarity between neighborhood structures of u and v. That means some candidate node becoming the next node depends on two factors, one is how strong the directed relationship between this node and the previously visiting node is and how similar neighborhoods of the two nodes is. That biases our walks from random transitions of DeepWalk's random walks but helps us have "better walks" where nodes on a window of a specific length have a stronger relationship. Then the formulation for calculating the probability p_v is as follows: $p_u = \frac{f_{uv}}{\sum_{z \in \Gamma(u)} f_{uz}}$.

Now we are using following indices to measure the neighborhood structure similarity.

(1) Jaccard index extended: $L^J_{uv} = \frac{|\widetilde{\Gamma}(u) \cap \widetilde{\Gamma}(v)|}{|\widetilde{\Gamma}(u) \cup \widetilde{\Gamma}(v)|}$, where $\widetilde{\Gamma}(u) = \Gamma(u) \cup \{u\}$.

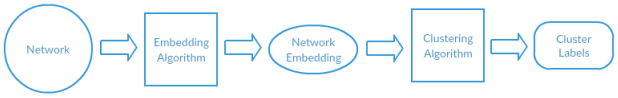


Figure 2: Evaluation Framework

(2) Adamic-Adar index extended: $L^A_{uv} = \sum_{z \in \widetilde{\Gamma}(u) \cap \widetilde{\Gamma}(v)} \frac{1}{\log |\widetilde{\Gamma}(z)|}$.

(3) Triangle counting-based index: $L^{\Delta}_{uv} = |\Gamma(u) \cap \Gamma(v)| + 1$.

It should be noted that we are avoiding using the regular Jacard and Adamic-Adar index as in some cases there is an edge connecting u and v but the two nodes do not have any common neighbor nodes. That means their Jacard and Adamic-adard indices are equal to zero and then v is never selected as the next node. In the third index, the number of triangles which contain (u, v) (also is the number of common neighbor nodes of the two nodes) is used and for the similar reason, we need add one because there may not be any triangles contaning these nodes.

The set of paths (or sequences of nodes) obtained from biased and same-length walks is then fed into the Skip-gram model. The set of word embedding vectors returned by Skip-gram is an one-to-one correspondence for our set of node embedding vectors.

5 EXPERIMENTS AND EVALUATION

For evaluation, we use the framework as show in figure 2. In this framework, we can control three components: the network, the embedding algorithm, and the clustering algorithm. The remaining two components: the network embedding and clustering labels are outputs. This framework allows us to compare the output against previous works done on community detection, thus by fixing the networks and clustering algorithms. We uses 7 datasets, 3 embedding algorithms and 3 prediction algorithms for clustering, link prediction and multi-label classification. We evaluate the performance of our embedding algorithm using the F1-scores of the prediction we get at the end of this frame work. In this section, we will cover in general the first 3 components of the framework, and for the last 2 components, we will cover prediction algorithm and results together for each of the prediction algorithm used.

5.1 Network Datasets

We chose a variety of networks aiming to test the robustness of the embedding algorithms. These networks includes Zachary's Karate Club, Facebook ego network, Citation Network from Kaggle, Astrophy Co-authorship network, Blog Catalog 3 network, PPI (Protein-Protein Interaction) Network, and Wikipedia Network. Some basic details of these networks are listed in 1.

5.2 Embedding Algorithms

We use DeepWalk[1], Node2vec[4] and Biasedwalk on the seven network datasets mentioned in the previous section. These embedding created by Deepwalk[1] and Node2vec[4] are used to in our experiments to measure the performance again our Biasedwalk approach in clustering, link prediction, and multi-label classification. The results are shown in Section 5.4, 5.5 and 5.6 respectively.

Table 1: Network Detail

Network	#Nodes	#Edges	Directed?	Weigthed?
Zachary's Karate Club	34	78	no	no
Facebook Ego	747	60050	no	no
Citiation	27684	335130	yes	no
Coauthorship	18772	396160	no	no
Blog Catalog 3	10312	333983	no	no
PPI	19706	390633	no	no
Wikipedia	4777	184812	no	no

5.3 Network Embeddings

We create embedding files from all of the network datasets using the Deepwalk, Node2vec and Biasedwalk embedding techniques. We set the length of the feature vector for all three embedding algorithms used to $d = 128$, which produce very compact representation since (with the exception of Karate and Facebook ego network) $128 << |V|$ for our datasets.

5.4 Clustering

There are many well established clustering algorithms based on distance between points, such as K-means, Mean-shift, Ward hierarchical clustering, DBSCAN, etc. To keep the evaluation process simple, we use K-means method, which work well in general and has good scalability.

The k-means algorithm takes an input K to separate the data points X into K disjoint clusters C, each described by the mean μ_j of the cluster. The K-means algorithm aims to choose the means that minimize sum of squared within each cluster.

$$\sum_{i=0}^{n} \min_{\mu_j \in C} (\|x_j - \mu_i\|^2) \tag{1}$$

We will used the k-mean method provided in the scikit-learn library in Python to produce the node cluster label from the network embeddings.

As a preliminary check for the validity of our framework, we use Zachary's Karate club and the Facebook ego network of a user as the input networks. We embedded the network using Node2vec method, and we feed the embedding into K-mean clustering technique with parameter $k = 2$ and 4 for the karate network, and $k = 2, 3$ and 6 for the ego network. We generated our graph with the original network and the community labels generated by K-mean visualized the graph using Cytoscape, a graph visualization software. As we can see in figures 3, 4, the nodes nicely clustered into 2 communities, where the centroids seems to be node 1 (the club president) and node 34 (the karate instructor). Although it is arguable if there are actually four communities in this graph, we see in figure 5 that this framework can separate the network into more clusters by adjusting the input parameter to k-means algorithm.

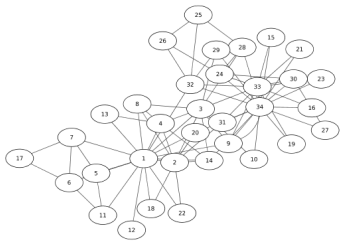


Figure 3: karate original

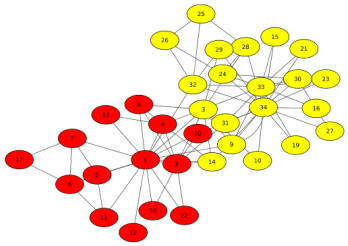


Figure 4: karate k=2

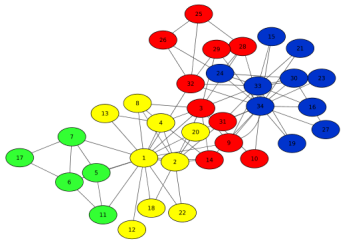


Figure 5: karate k=4

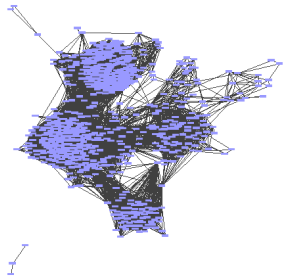


Figure 6: Original ego network of a Facebook user

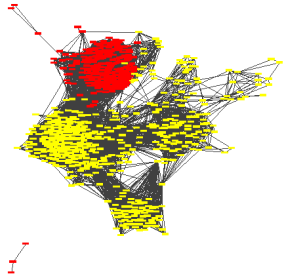


Figure 7: Ego network k=2

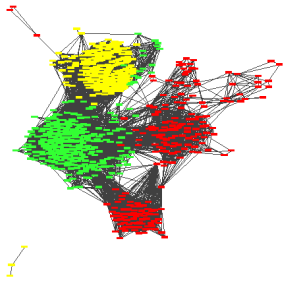


Figure 8: Ego network k=3

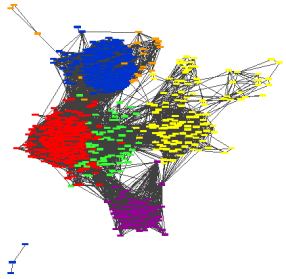


Figure 9: Ego network k=6

5.5 Link Prediction

Inspired by a Kaggle competition on link prediction, we to measure our embedding approach via link prediction. In this experiment, we uses two data datasets, citation network from the Kaggle competition and AstroPhy collaboration network. These two networks are embedded using Deepwalk[1], Node2vec[4], and Biasedwalk approach. For Deepwalk, we generate one embedding as the baseline. For Biasedwalk, we generate one embedding for file for each type of biased used, resulting in 3 embedded files. For Node2vec, we generate 25 different embedding files over the parameters $p, q \in \{0.25, 0.5, 1, 2, 4\}$, and the best result is reported, borrowing the same approach used in the Node2vec Paper[4]. The embedded lower dimension vectors are used as the only feature, passed into a random forest classifier to learn if a link exist between two given nodes in the network. We used 5 percent training labels to train the classifier and evaluate the results. For Kaggle citation network, we evaluate the result by submitting the predictions to the Kaggle competition to get the F1-scores. In the case of Astrophy collaboration network, we calculated the F1-marco and F1-micro scores using the sklearn package in Python.

The results of our experiment is shown in Table 2 and 3. For the Citation network, our method, using the best bias method, outperform Deepwalk by 3.17% and Node2vec by 3.14%. It is worthwhile to note that for all three bias methods, Jaccard index, Adamic Adard, and triangle counting, Biased walk outperform both Deepwalk and Node2vec. This seems to suggest that Biasedwalk works better that the other embedding methods when directed. For the Astrophy network, we outperform Deepwalk by 0.95% in F1-macro score

Table 2: Citation Network from Kaggle

Kaggle citation network		F1 score
DeepWalk		82.12%
Our method	Jaccard index based	84.72%
	Adamic Adard index based	83.56%
	Triangle counting based	83.90%
node2vec	p = 0.5, q = 2	82.14%

Table 3: Astrophy Collaboration Network

AstroPhy collaboration network		Macro-F1	Micro-F1
DeepWalk		90.75%	90.77%
Our method	Jaccard index based	91.61%	91.64%
	Adamic Adard index based	91.15%	91.18%
	Triangle counting based	91.60%	91.64%
node2vec	p = 2, q = 4	92.05%	92.07%

and 0.96% in F1-micro score. However, Node2vec outperform our approach by 0.48% in F1-Macro score and 0.47% in F1-micro score.

We can see that for the task of link prediction, our biased walk approach outperform Deepwalk on both network datasets. Our method which uses Jaccard similarity extended can perform even better than Node2vec.

5.6 Multi-Label Classification

For multi-label classification, we uses the Blog Catalog 3 network, PPI (Protein-Protein Interaction) network, and Wikipedia network. The Blog Catalog 3 network this is a network of social relationships of the bloggers listed on the BlogCatalog website. The 39 different labels represent blogger interests inferred through the meta-data provided by the bloggers (i.e. political, educational). The PPI network corresponds to the graph with nodes which could be labeled by the hallmark gene sets[5]. There are 50 different labels for this network. The Wikipedia network is a cooccurrence network of words appearing in the first million bytes of the Wikipedia dump. The 40 different labels for the words are part of speech associated with each word. Again, we embed these networks using the same procedure as those in link prediction, which gives us 1 embedded file for Deepwalk, 3 files for Biasedwalk, and 25 files for Node2vec. We pass the embedding as the feature vector into a SVM classifier, and training using 5 percent of the training data. A predicted result is considered correct if and only if all labels associated with a node are predicted correctly. Again we calculate the F1-macro and F1-micro scores and the result are show in figure 4, 5, and 6.

In general the prediction accuracy is much lower for this application since predicting all the labels correctly is a rather hard problem, however, we can still examine the relative different between the embedding algorithms. For Blog Catalog 3 network, the F1-macro score of Biasedwalk is 0.20% better than Deepwalk but 11.25% worse than Node2vec. the F1-micro score are 10.36% and 11.25% worse than Deepwalk and Node2vec respectively. An possible explanation of Biasedwalk's inferior performance is the

Table 4: Blog Catalog 3 Network

Blog catalog network		Macro-F1	Micro-F1
DeepWalk		16.05%	29.63%
Our method	Jaccard index based	16.09%	26.56%
	Adamic Adard index based	8.33%	16.70%
	Triangle counting based	7.62%	14.96%
node2vec	p = 0.25, q = 0.25	18.13%	31.69%

Table 5: PPI Network

PPI network		Macro-F1	Micro-F1
DeepWalk		10.35%	12.60%
Our method	Jaccard index based	11.25%	13.50%
	Adamic Adard index based	10.65%	12.78%
	Triangle counting based	10.07%	12.24%
node2vec	p = 4, q = 1	10.35%	12.41%

Table 6: Wikipeida Network

Wikipedia		Macro-F1	Micro-F1
DeepWalk		7.91%	37.51%
Our method	Jaccard index based	4.49%	29.87%
	Adamic Adard index based	7.32%	33.58%
	Triangle counting based	7.06%	32.85%
node2vec	p = 4, q = 0.5	8.44%	36.69%

unusually low Jaccard similiar index among nodes in this network, as we found that all Jaccard indices extended between its node pairs are less than 0.6. For PPI network, the F1-macro score for Biasedwalk is 8.70% better than Deepwalk and Node2vec. The F1-micro score is 7.14% better than Deepwalk and 8.78% better than Node2vec. Again, all three bias methods does as well as or better than both Deepwalk and Node2vec. For Wikipedia network, the F1-macro score for Biasedwalk is 7.46% worse than Deepwalk and 13.27% worth than Node2vec. The F1-micro score is 10.48% worse than Deepwalk and 8.48% worse than Node2vec.

Now, considering all the results from link prediction and multi-label classification, it seems that in general Biasedwalk using Jaccard similiar index performs better than Adam-Adar and Triangle counting methods and Biasedwalk performs better than Deepwalk, and is comparable to Node2vec.

6 CONCLUSIONS AND FUTURE WORKS

We explored two state of the art network embedding algorithms, Deepwalk[1] and Node2vec[4]. These embedding techniques reduce the dimensionality of network graph, preserve graphical structure and properties. The reduced representation can be used in clustering algorithms and machine learning algorithms to solve real life problems such as community detection and link prediction. We also proposed BiasedWalk, a network embedding method and we tested the proposed method against these state-of-art embedding

algorithms in link prediction and multi-label classification. Our results have shown that our proposed method generated embedding that outperform Deepwalk and is comparable to Node2vec on most networks with different structures of our experiment.

In the future, we would like to use other network indices for calculating neigborhood similarity between nodes and use higher-order random walks to sample paths on networks. We will also investigate specific reasons which make Biasedwalk working well on some kind of networks such as directed or high clustering coefficient networks.

ACKNOWLEDGMENTS

The project is proposed and completed under Dr. Fragkiskos Malliaros' suggestions and advice.

REFERENCES

[1] R. Al-Rfou B. Perozzi and S. Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.

[2] Mikhail Belkin and Partha Niyogi. 2002. Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering. In *Advances in Neural Information Processing Systems 14*, T. G. Dietterich, S. Becker, and Z. Ghahramani (Eds.). MIT Press, 585–591. http://papers.nips.cc/paper/1961-laplacian-eigenmaps-and-spectral-techniques-for-embedding-and-clustering.pdf

[3] Michael A. A. Cox and Trevor F. Cox. 2008. *Multidimensional Scaling*. Springer Berlin Heidelberg, Berlin, Heidelberg, 315–347. https://doi.org/10.1007/978-3-540-33037-0_14

[4] Aditya Grover and Jure Leskovec. 2016. Node2Vec: Scalable Feature Learning for Networks. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. ACM, New York, NY, USA, 855–864. https://doi.org/10.1145/2939672.2939754

[5] M. Mahoney. 2011. Large text compression benchmark. (2011). www.mattmahoney.net/dc/textdata

[6] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and Their Compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems (NIPS'13)*. Curran Associates Inc., USA, 3111–3119. http://dl.acm.org/citation.cfm?id=2999792.2999959

[7] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale Information Network Embedding. In *Proceedings of the 24th International Conference on World Wide Web (WWW '15)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 1067–1077. https://doi.org/10.1145/2736277.2741093

[8] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. 2000. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science* 290, 5500 (2000), 2319–2323. https://doi.org/10.1126/science.290.5500.2319 arXiv:http://science.sciencemag.org/content/290/5500/2319.full.pdf