Improve Network Embeddings with Regularization

Yi Zhang
School of Computer Science,
University of Windsor
Windsor, Ontario
zhang18f@uwindsor.ca

Jianguo Lu
School of Computer Science,
University of Windsor
Windsor, Ontario
jlu@uwindsor.ca

Ofer Shai
Chan Zuckerberg Initiative Inc.
Toronto, Ontario
ofer.shai@gmail.com

ABSTRACT

Learning network representations is essential for many downstream tasks such as node classification, link prediction, and recommendation. Many algorithms derived from SGNS (skip-gram with negative sampling) have been proposed, such as LINE, DeepWalk, and node2vec. In this paper, we show that these algorithms suffer from norm convergence problem, and propose to use L2 regularization to rectify the problem. The proposed method improves the embeddings consistently. This is verified on seven different datasets with various sizes and structures. The best improvement is 46.41% for the task of node classification.

CCS CONCEPTS

• **Information systems** → **Data mining**;

KEYWORDS

Network embeddings, Skip-gram, Regularization, LINE, DeepWalk, node2vec

ACM Reference Format:
Yi Zhang, Jianguo Lu, and Ofer Shai. 2018. Improve Network Embeddings with Regularization. In *2018 ACM Conference on Information and Knowledge Management (CIKM'18), October 22–26, 2018, Torino, Italy.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3269206.3269320

1 INTRODUCTION

Recent years, many network embedding algorithms have been proposed. Among them, skip-gram with negative sampling (hereafter SGNS) based algorithms, such as LINE[8], DeepWalk[6], and node2vec[4], are widely discussed. Despite the popularity of these algorithms, the repeatability of the experiments is a common problem. For example, the macro-F1 of multi-label classification on BlogCatalog, a widely used dataset for benchmarking network embeddings[4, 6, 8], is reported at 0.273 in [6], but at 0.211 in [4]. It is partially due to the randomness of the algorithms, the numerous hyper-parameters involved. One of the hyper-parameters is the iteration, i.e., the number of times to scan the data to train the model.

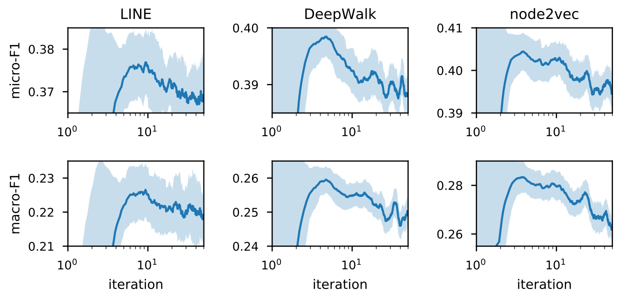


Figure 1: Performance on node classification – BlogCatalog.

1.1 Performance degeneration over iteration

We observe that with the increase of iteration, the performance decreases consistently for these algorithms over various datasets. Figure 1 shows such phenomenon on the BlogCatalog data for those three network embedding algorithms. The micro and macro F1 scores of node classification are plotted. To ameliorate the variation caused by the randomness of the algorithms, for each algorithm, we train five models independently, and report the mean of the their performance. For better observation, we set the learning rate to a fixed value of 0.025 instead of a decaying learning rate. During the training, when every 10^6 samples have been trained, we take a snapshot of the model and evaluate the embeddings in the node classification task.

From the plot, we can see that the performances rise to the peak after five iterations, then drop down continuously. In practice, a grid-search of this hyper-parameter may help to find the best stop time. However, a better solution is needed for producing a stable result.

1.2 The norms of the embeddings

Next, we examine the evolution of the L2-norms of the vectors. Overall, the norms will increase over iterations. Intuitively, large nodes, the nodes that have higher degrees, should have larger norms so that they can have a higher impact during the training. To observe the norm evolution of different types of nodes, we sample four categories of nodes according to their degrees. The smallest nodes are the ones with a degree between one and four. The second smallest nodes have degrees between $2^2 + 1$ and 2^4. The third category has degrees between $2^4 + 1$ and 2^8, and the largest nodes have degrees greater than $2^8 + 1$.

For each group, we randomly select 25 nodes and record their average L2-norms. Figure 2 shows the change of L2-norms over training iterations that ranges from 1 to 50. Row 1 is for embedding rectors, row 2 for output vectors. For all three algorithms (LINE,

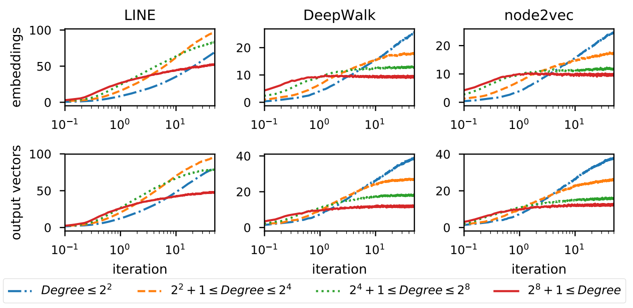


Figure 2: L2-norm of vectors – BlogCatalog.

DeepWalk, node2vec), we observe that the norms of larger nodes are indeed larger than smaller nodes in the first few iterations. With the growth of iterations, the norms of large nodes tend to converge. However, the norms of smaller nodes keep growing and surpass the norms of larger nodes. After that cross-point, the performance degenerates as shown in Figure 1.

2 NETWORK EMBEDDINGS WITH L2 REGULARIZATION

Given a network $G = (V, E)$, where V is a set of nodes and E is a set of edges. An embedding is a dense d-dimension vector v_i for a node $n_i \in V$. The embedding v_i should retain the information of node n_i in the network such as similarity and structure.

To learn the embeddings from a network, for each node n in the network, existing works train the SGNS model with a specific sampling strategy $N_+(n)$ that captures the node-neighborhood information. More specifically, DeepWalk and node2vec use the dynamic window on the uniform and biased random walk paths, and LINE uses random edge sampling. In our work, we add L2 regularization in the model to improve the embeddings. The objective function is:

$$O = \frac{1}{S} \sum_{n_i \in V} \sum_{n_j \in N_+(n_i)} [\log \sigma(u_j \cdot v_i) + \sum_{k=1}^{K} \mathbb{E}_{u_k \sim P_n(u)} \log \sigma(-u_j \cdot v_i)] \\ - \lambda \sum_{i=1}^{|V|} \|v_i\|_2{}^2 - \lambda \sum_{i=1}^{|V|} \|u_i\|_2{}^2, \tag{1}$$

where S is the number of the observed training pairs. v_i is the embedding vector for node n_i. u_i is the output vector for node n_i. λ is the weight for L2 regularization. $|V|$ is the number of nodes in the network. P_n is a noise distribution which is the frequency of nodes raised to the power of 0.75. Note that for LINE, this frequency follows the degree distribution, while in DeepWalk and node2vec, it is the frequency of a node in the training samples. These works use SGD to update their models. Thus the model updated immediately when a training sample arrives. To ensure a smooth update, we evenly distribute the regularization weight λ into each local objective function by frequency. More specifically, for each training sample (n_i, n_j), the local objective is:

$$\log \sigma(u_j \cdot v_i) + \sum_{k=1}^{K} \mathbb{E}_{u_k \sim P_n(u)} \log \sigma(-u_j \cdot v_i) \\ -\lambda_1 \|v_i\|_2^2 - \lambda_2 \|u_j\|_2^2 - \sum_{k=1}^{K} \mathbb{E}_{u_k \sim P_n} \lambda_3 \|u_k\|_2^2. \tag{2}$$

Here λ_1, λ_2 and λ_3 are the regularization weight for embedding vector v_i, output vector u_j and negative sample u_k divided from λ, which are defined as below:

$$\begin{aligned} \lambda_1 &= \frac{\lambda}{Freq_i(n_i)}, \\ \lambda_2 &= \frac{\lambda}{Freq_o(n_j) + Freq_n(n_j)}, \\ \lambda_3 &= \frac{\lambda}{Freq_o(n_k) + Freq_n(n_k)}. \end{aligned} \tag{3}$$

Here $Freq_i(\cdot)$, $Freq_o(\cdot)$ and $Freq_n(\cdot)$ denote the frequency of a node trained as an input, output and negative sample per training iteration respectively.

Some works adopt the subsampling strategy to reduce the training samples for frequent items[5]. With subsampling, the model randomly discards a sample with probability $P(i) = max(0, 1 - \sqrt{\frac{t}{f(i)}})$, where t usually sets to 10^{-4} to 10^{-5} in word embeddings based on different sizes of the corpus. In our experiment, we find that the model is sensitive to this parameter. Thus, we discard subsampling in our experiment and keep all observed training samples. One can easily adopt subsampling into our model and search for the best parameter to improve the embeddings and training speed.

3 L2 REGULARIZATION ON EMBEDDING VECTORS

Our regularization applies on both embedding vectors and output vectors. It is necessary to compare with the approach that regularizes the embedding vectors only [1, 2]. Embeddings are updated according to $\sigma(u \cdot v)$ and u. Training with unrestricted u will also lead to larger update weight during the training.

For example, assume that there is a training sample pair (n_i, n_j). v_i and u_j are the corresponding embedding vector and output vector. Without the regularizer, the update weight for v_i is

$$(1 - \sigma(u_j \cdot v_i)) \cdot u_j + \sum_{k=1}^{K} \mathbb{E}_{u_k \sim P_n} - \sigma(u_k \cdot v_i) \cdot u_k.$$

The first term is the weight that learnt from the output sample n_j. When u_j is an unrestricted vector with large L2-norm, $\sigma(u_j \cdot v_i)$ can be larger or smaller than expected, leading $\sigma(u_j \cdot v_i)$ closer to 0 or 1. Moreover, when u_j is unrestricted, the final update weight could be larger than expected. Thus, we here argue that the applying L2 regularization on embedding vectors are insufficient.

To support our claim. we train these algorithms by setting λ_2 and λ_3 to 0, which leaves the output vector unrestricted. This equals to the LogSig model proposed in [2]. The experiment results are illustrated in Figure 3.

We can see that the norms for embedding vectors are restricted. However, the norms for output vectors are still growing, even larger

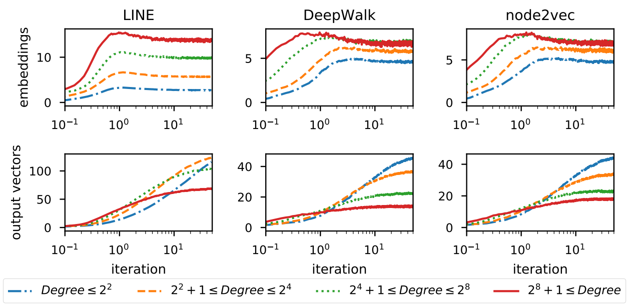


Figure 3: Norms of the vectors in network embeddings – L2 regularization on embedding vectors.

than before – For example, the norms for small nodes in LINE increased from 80.29 to 115.97. Meanwhile, we found that after we apply regularization on embedding vectors, there is a strange peak in the early stage of the training for the embedding vectors, indicating un-smooth learning curve during the training.

Overall, we can see that applying regularization on embeddings will stabilize the embedding vectors, but the convergence problem still exists for output vectors, even severer than before. Compare to our model, as showed in Figure 4, we can see the norms of embeddings and output vectors converge for all three algorithms.

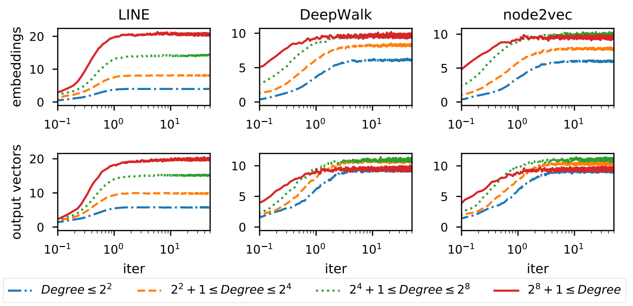


Figure 4: Norms of the vectors in network embeddings – L2 regularization on embedding vectors and output vectors.

Intuitively, the norm of a vector reflects the importance of the corresponding node. Larger norm will bring large update step. By adding the regularization properly, we see that the norms of small nodes are restricted and smaller than large nodes.

4 EXPERIMENTS

We use seven datasets in our experiment. Some data is widely used to benchmark the network embedding algorithms[2–4, 6–9]. The statistics of the datasets are listed in Table 1. WebKB is the smallest one, which has only 877 nodes and 1,608 edges. YouTube is the largest one, which contains 1.14 million nodes and 2.99 million edges. In the experiment, we test the original LINE, DeepWalk, and node2vec, and compare them with their regularized counterparts. Methods with subscript **R** are our regularized versions, methods with subscript **RE** regularize embedding vectors only.

4.1 Experimental setup

We reimplement these algorithms in Cython with BLAS acceleration. For a fair comparison, all algorithms are implemented in the same framework. The code and data are available on our webpage[1]. The learning rate is set to decaying from 0.025 to 0.00001 linearly during the training. The window size is set to 5 for DeepWalk and node2vec, and the number of negative samples for each training sample is set to 5. We set lambda=1 for LINE and lambda=5 for DeepWalk and node2vec. The difference in lambda is necessary because DeepWalk and node2vec take five times more pairs per iteration due to the window size.

Dataset	# Nodes	# Edges	# Labels	Multi-label
WebKB	877	1,608	5	no
Cora	2,708	5,429	7	no
CiteSeer	3,319	4,722	6	no
BlogCatalog	10,312	333,983	39	yes
PubMed	19,717	44,338	3	no
Flickr	80,513	5,899,882	195	yes
Youtube	1,138,499	2,990,443	47	yes

Table 1: Statistics of seven datasets.

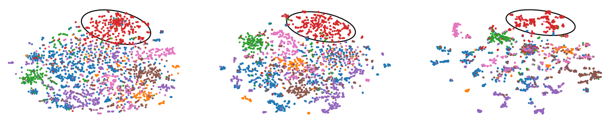
4.2 Node classification

We evaluate the models with the node classification task. Following the same method in [3, 4, 6, 8], we train a Logistic Regression classifier with default parameters using scikit-learn in Python. To evaluate the performance, we take r proportion of nodes to train the classifier, then use the rest $1-r$ nodes to test the performance of the classifier, where $r = 0.08$ Flickr and Youtube and $r = 0.8$ for the rests. for Since the datasets are unbalanced, we report both micro and macro F1 in our experiment. Intuitively, micro-F1 evaluates the overall performance over all classes, while macro-F1 takes the unweighted mean of F1 for each class.

Due to the randomness of the algorithms, we train five models for each algorithm on each dataset, and report the average of these five models. Table 2 and Table 3 list the results.

Overall, smaller datasets gain more improvement compared to larger datasets. The biggest improvement is LINE$_R$ with CiteSeer by 42.73% on micro-F1 and 46.4132% on macro-F1 over LINE. While LINE$_{RE}$ gains 8.2938% and 10.59% improvement over LINE. YouTube is the largest one, in which we can see that LINE$_R$ improves the micro and macro-F1 by 7.78% and 10.53% over LINE. On the other hand, the micro-F1 of LINE$_{RE}$ drops -0.13% compare to LINE.

Among three algorithms, LINE benefits most from the regularization, especially on CiteSeer. While DeepWalk and node2vec also receive around 5% improvements.



(a) LINE (b) LINE$_{RE}$ (c) LINE$_R$

Figure 5: Visualization for Cora.

[1]http://zhang18f.myweb.cs.uwindsor.ca/n2v_r/

	WebKB		Cora		CiteSeer		BlogCatlog		PubMed		Flickr		YouTube	
	F1	Imp(%)	F1	Imp(%)	F1	Imp(%)	F1	Imp(%)	F1	Imp(%)	F1	Imp(%)	F1	Imp(%)
LINE	0.5114	–	0.5959	–	0.3819	–	0.3699	–	0.7186	–	0.3362	–	0.2729	–
LINE$_{\text{RE}}$	0.5545	8.44	0.6450	8.24	0.4136	8.29	0.3812	3.05	0.7208	0.31	0.3352	-0.29	0.2726	-0.13
LINE$_{\text{R}}$	0.5761	12.67	0.7875	32.14	0.5451	42.73	0.3847	4.00	0.8092	12.61	0.3415	1.55	0.2942	7.78
DeepWalk	0.4034	–	0.8077	–	0.5541	–	0.3919	–	0.7956	–	0.3505	–	0.3123	–
DeepWalk$_{\text{RE}}$	0.4409	9.30	0.8011	-0.82	0.5587	0.82	0.4074	3.97	0.7997	0.51	0.3524	0.54	0.3137	0.44
DeepWalk$_{\text{R}}$	0.4466	10.70	0.8292	2.65	0.5756	3.87	0.4146	5.80	0.8024	0.85	0.3545	1.13	0.3149	0.81
node2vec	0.4375	–	0.7993	–	0.5535	–	0.4029	–	0.7970	–	–	–	–	–
node2vec$_{\text{RE}}$	0.4341	-0.78	0.7952	-0.51	0.5493	-0.76	0.4054	0.64	0.7991	0.27	–	–	–	–
node2vec$_{\text{R}}$	0.4875	11.43	0.8207	2.68	0.5725	3.43	0.4170	3.51	0.8067	1.22	–	–	–	–

Table 2: Micro-F1 on classification task, training ratio is 8% for Flickr and YouTube and 80% for the resets.

	WebKB		Cora		CiteSeer		BlogCatlog		PubMed		Flickr		YouTube	
	F1	Imp(%)	F1	Imp(%)	F1	Imp(%)	F1	Imp(%)	F1	Imp(%)	F1	Imp(%)	F1	Imp(%)
LINE	0.3286	–	0.5821	–	0.3412	–	0.2189	–	0.6963	–	0.1760	–	0.1903	–
LINE$_{\text{RE}}$	0.3477	5.84	0.6320	8.57	0.3773	10.59	0.2322	6.06	0.7007	0.64	0.1830	3.96	0.1936	1.71
LINE$_{\text{R}}$	0.3459	5.27	0.7736	32.90	0.4996	46.41	0.2376	8.54	0.7951	14.19	0.1941	10.28	0.2103	10.53
DeepWalk	0.2215	–	0.7958	–	0.5026	–	0.2543	–	0.7805	–	0.2103	–	0.2431	–
DeepWalk$_{\text{RE}}$	0.2697	21.76	0.7925	-0.42	0.5095	1.38	0.2700	6.14	0.7849	0.57	0.2164	2.92	0.2436	0.22
DeepWalk$_{\text{R}}$	0.2919	31.79	0.8241	3.55	0.5261	4.67	0.2812	10.57	0.7893	1.13	0.2170	3.17	0.2463	1.31
node2vec	0.2702	–	0.7912	–	0.5025	–	0.2637	–	0.7818	–	–	–	–	–
node2vec$_{\text{RE}}$	0.2568	-4.95	0.7879	-0.41	0.4960	-1.28	0.2720	3.15	0.7838	0.26	–	–	–	–
node2vec$_{\text{R}}$	0.2869	6.17	0.8120	2.62	0.5178	3.04	0.2875	9.05	0.7939	1.55	–	–	–	–

Table 3: Macro-F1 on classification task, training ratio is 8% for Flickr and YouTube and 80% for the resets.

Figure 5 demonstrates the difference of the embeddings produced by LINE, LINE$_{\text{RE}}$, and LINE$_{\text{R}}$. The dataset is Cora, a citation network. The dimension of the embeddings is further reduced from 100 to two using t-SNE, so that they can be plotted. Each dot represents a paper, and its color reflects its category. For example, the red dots represent *Genetic Algorithms*. From the plot, we can observe that our method LINE$_{\text{R}}$ can separate groups better.

Tasks	Textcase	SGNS	SGNS$_{\text{R}}$	Imp(%)
Similarity	RW	0.3331	0.3913	17.47
	SimLex-999	0.2800	0.3060	9.32
Analogy	Google	0.4468	0.4628	3.58
	MSR	0.5056	0.5072	0.32

Table 4: SGNS and SGNS$_{\text{R}}$ on Text8.

5 WORD EMBEDDINGS

We also applied our L2 regularization SGNS, and observed improvements as reported in Table 4. Both the data (Text8) and evaluation tasks (word similarity and analogy) are the same as the ones provided in the implementation of [5]. Hyper-parameters are also the same except iteration, which is 50 in our experiment. On similarity task, our model improves the performance by 17% on rare words test cases and 9.32% on SimLex-999. The analogy tasks also receive 3.58% and 0.32% on Google and MSR respectively.

6 CONCLUSIONS

In this paper, we study the norm convergence problem of SGNS based network embedding algorithms. Due to the unrestricted weight of the vectors, the L2 norm of small nodes will continue growing during the training. Insufficient regularization in the previous works does not fix the issue. Our experiment shows that the improper regularization will make the embeddings worse in some cases. Based on our observation, we apply the L2 regularizer on both input and output vectors to improve the embeddings. The experiment shows that the new model can improve the embeddings in node classification task. We verify our model on seven datasets in size of hundreds to millions.

REFERENCES

[1] Qingyao Ai, Liu Yang, Jiafeng Guo, and W. Bruce Croft. 2016. Analysis of the Paragraph Vector Model for Information Retrieval. In *Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval (ICTIR '16)*. ACM, New York, NY, USA, 133–142. https://doi.org/10.1145/2970398.2970409

[2] Yihan Gao, Chao Zhang, Jian Peng, and Aditya G. Parameswaran. 2018. Low-Norm Graph Embedding. (2018). arXiv:CoRR/1802.03560

[3] Palash Goyal and Emilio Ferrara. 2018. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems* 151 (2018), 78 – 94. https://doi.org/10.1016/j.knosys.2018.03.022

[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] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 3111–3119.

[6] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '14)*. ACM, New York, NY, USA, 701–710. https://doi.org/10.1145/2623330.2623732

[7] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network Embedding As Matrix Factorization: Unifying DeepWalk, LINE, PTE, and Node2Vec. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining (WSDM '18)*. ACM, New York, NY, USA, 459–467. https://doi.org/10.1145/3159652.3159706

[8] 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

[9] D. Zhang, J. Yin, X. Zhu, and C. Zhang. 2018. Network Representation Learning: A Survey. *IEEE Transactions on Big Data* (2018), 1–1. https://doi.org/10.1109/TBDATA.2018.2850013