dynnode2vec: Scalable Dynamic Network Embedding

Sedigheh Mahdavi[1], Shima Khoshraftar[1] and Aijun An[1]

***Abstract*— Network representation learning in low dimensional vector space has attracted considerable attention in both academic and industrial domains. Most real-world networks are dynamic with addition/deletion of nodes and edges. The existing graph embedding methods are designed for static networks and they cannot capture evolving patterns in a large dynamic network. In this paper, we propose a dynamic embedding method, *dynnode2vec*, based on the well-known graph embedding method *node2vec*. *Node2vec* is a random walk based embedding method for static networks. Applying static network embedding in dynamic settings has two crucial problems: 1) Generating random walks for every time step is time consuming 2) Embedding vector spaces in each timestamp are different. In order to tackle these challenges, *dynnode2vec* uses evolving random walks and initializes the current graph embedding with previous embedding vectors. We demonstrate the advantages of the proposed dynamic network embedding by conducting empirical evaluations on several large dynamic network datasets.**

Keywords-Dynamic networks, Graph embedding , Network embedding, Representation learning

I. Introduction

In the last few decades, graph embedding methods have achieved remarkable success in the analysis of large networks. The basic aim is to represent nodes of a large, high-dimensional graph in low-dimensional vectors that preserve the neighbourhood information of the graph. Several static graph embedding algorithms [1], [2], [3], [4], [5], [6], [7], [8] have been developed for a variety of machine learning tasks such as visualization [9], node classification [10], link prediction [11], and recommendation [12]. Random walk and edge sampling graph embedding approaches [2], [3], [4] have a remarkable performance in large networks which contain more than thousand nodes and edges.

Large real-world networks evolve naturally over time, i.e., nodes and edges appear or disappear, or edges change. For example, in co-authorship networks new edges may emerge (new collaborations may be formed) or new nodes can be added (new authors) and in social networks, users may delete friends (delete edges) or some users may leave the network (delete nodes). The static graph embedding methods are not capable of dealing with the critical challenge involved in dynamic networks. The disadvantage of using static embedding methods at each timestamp independently are as follows. First, embedding vectors for each timestamp are in different spaces. Second, learning embedding vectors separately is a time-consuming process.

[1]Department of Electrical Engineering and Computer Science, York University, Toronto, Canada {smahdavi, khoshraf, aan}@cse.yorku.ca

In this paper, we propose *dynnode2vec*, a scalable dynamic network embedding for large evolving networks. In order to handle dynamic networks, *dynnode2vec* modifies the well-known static embedding method, *node2vec* by employing the previous learned embedding vectors as initials weights for the skip-gram model. This is motivated by dynamic language models, especially dynamic Skip-gram models [13], [14]. In addition, we utilizes evolving random walks for updating the trained skip-gram from previous timestamp. The evloving random walks are only generated for nodes that have changed in consecutive times. As random walk generation is the time consuming part of graph embeddings, we are able to significantly reduce the running time. Our main contributions in this paper are:

- We develop a dynamic embedding method *dynnode2vec* that captures evolving patterns in large dynamic networks
- *dynnode2vec* is a fast and accurate method for dynamic graph embedding
- We evaluate the performance of our method in variety of tasks including link prediction, node classification and anomaly detection on large real-world graphs

II. *Dynnode2vec*: Scalable dynamic network embedding

We represent a dynamic network G as a sequence of graphs $G_1, G_2, \ldots, G_T$ from timestamps 1 to T. Each graph at time t is represented by $G_t = (V_t, E_t)$ where V_t and E_t are the vertices and edges of the graph respectively. Our goal is to modify the static *node2vec* embedding method for learning representation of a dynamic network. We begin with a brief description of the static *node2vec* embedding method. Then, we explain the steps of *dynnode2vec* which are summarized in Algorithm 1.

node2vec is an extension of *deepwalk* algorithm [4] and has two main subcomponents; *node2vec* random walk and Skip-gram model. *node2vec* random walk is a flexible random walk method which samples neighbourhoods of a source node by Breadth-first Sampling (BFS) and Depth-first Sampling (DFS). *node2vec* first generates a corpus by sampling a number of random walks γ of length *t* starting at each vertex. Then, the Skip-gram uses these random walks to learn the representation vector for each node.

A. Description of dynnode2vec steps

The main challenge of modification of the static *node2vec* is how to learn embedding at time t by updating embedding vectors in time step $t-1$. For a dynamic network $G = G_1, G_2, \ldots, G_T$, we run the static *node2vec* for the first

978-1-5386-5035-6/18/$31.00 ©2018 IEEE

graph G_1 separately, extract the embedding vectors and keep the structure of trained Skip-gram for the next timestamp. For all other subsequent timestamps $2, \ldots, T$, the following steps are performed between two consecutive timestamps t and $t+1$.

1) Evolving Walk generation: In the static *node2vec*, random walks are generated independently for each timestamp for all nodes which is very time-consuming process. In *dynnode2vec*, we generate an effective set of random walks for only evolving nodes instead of generating random walks for all nodes in the current timestamp. Therefore, new random walks from changed regions in the graph G_t can efficiently update the embedding vectors according to temporal evolution of networks over time. Assume the structure evolution of the graph G_t from G_{t-1} includes sets of new edges/nodes that are added (E_{add}/V_{add}), deleted (E_{del}/V_{del}) and their weights have changed (E_{change}). The evolving nodes in the timestamp t are defined as follows:

$$\Delta V_t = V_{add} \cup \{v_i \in V_t | \exists e_i = (v_i, v_j) \in (E_{add} \cup E_{del})\} \quad (1)$$

Generating evolving random walks is fast and time-efficient since dynamic networks are evolved gradually and neighbourhoods of most nodes are kept unchanged. However, in the worst case, evolving nodes include all nodes as all nodes have changed in the timestamp t.

2) Dynamic Skip-gram model: In natural language processing domain, several dynamic word embedding [15], [16], [13], [14] have been proposed to track word evolution such as new words are created (internet), and some words die out throughout time. In [14], for learning the representation vectors in the timestamp t they train the Skip-gram by initializing word vectors obtained from previous timestamp $t-1$. In this dynamic Skip-gram model, the vocabulary set is updated and the Skip-gram is retrained by new documents in timestamp t. In network embedding domains, DynGEM [17] incrementally learns embedding vectors for the timestamp t by using embedding vectors from the timestamp $t-1$ as initial weights for SDNE atuoencoder in a dynamic network.

dynnode2vec also takes advantage of dynamic Skip-gram model for obtaining embedding vectors at time t and uses the pre-trained Skip-gram model (Skip-gram$_{t-1}$) as initial weight for (Skip-gram$_t$). In order to do that, first the vocabulary set of Skip-gram$_t$ is updated according to new evolving walks. Then, Skip-gram$_t$ is trained by new evolving walks generated on the evolving nodes.

Algorithm 1 :**Algorithm: Dynnode2vec**

Input: Graphs $G = G_1, G_2, \ldots, G_T$
Output: Embedding vectors $Z_1, Z_2, \ldots, Z_T$
Run static *node2vec* for the Graph G_1
for $t = 2$ to N **do**
    Find a set of evolving nodes,ΔV_t,
    Sample new random walks ($Walk_n$) for ΔV_t
    Train Skip-Gram $Skip_t$ with $Walk_n$ and obtain Z_t
end for

III. EXPERIMENTS

A. Datasets

The performance of *dynnode2vec* is evaluated on following datasets.

- Hep-th [20]: This dataset is the coauthorship network of researchers in High energy physics theory conference. Hep-th has $34k$ nodes, $421k$ edges divided into 60 graphs.
- Autonomous Systems (AS) [21]: AS is built from logs of the BGP (Border Gateway Protocol) which shows the communication between users. Number of nodes, edges and time steps are $6k$, $13k$ and 100 respectively.
- Enron [18]: Enron is the email communication network between Enron company employees. It has $87k$ nodes and $1.1M$ edges over 175 months.
- StackOverflow (St-Ov) [22]: This dataset is derived from question and answers in Math Overflow website. Each edge shows users interactions. This dataset contains $14k$ nodes and $195k$ edges over 2350 days. We divided the datastream into 58 graphs.
- dblp [23]: This is the coauthorship network dataset among researchers of different fields. It consists of $90k$ nodes and $749k$ edges over 18 years. Each node in dblp has one of the two class labels, database and data mining (VLDB, SIGMOD, PODS, ICDE, EDBT, SIGKDD, ICDM, DASFAA, SSDBM, CIKM, PAKDD, PKDD, SDM and DEXA) and computer vision and pattern recognition (CVPR, ICCV, ICIP, ICPR, ECCV, ICME and ACM-MM).
- Darpa [19]: Darpa is a dataset consisting of communications between source IPs and destination IPs. This dataset contains different attacks between IPs. We used a subset of darpa consisting of $12k$ nodes and $22k$ edges over 100 hours.

B. Baselines

We compared the performance of our method against the following existing methods:

- DeepWalk[4]: This is the first node embedding method based on random walks. DeepWalk uses Skip-gram model and uniform random walks to learn the neighborhood structure of the graph.
- *node2vec*[3]: This method learns node representation in networks by preserving network neighborhood of nodes. It explores the neighborhood of a node by generating DFS and BFS random walks for that node.
- DynGEM [17]: This approach is a stable dynamic node embedding method that works for growing dynamic graphs. It incrementally builds the embedding vectors at each time using the embedding vectors from previous time. In dynamic networks with a large number of nodes like Enron and DBLP, the memory requirement of Dyngem signicantly increases. Therefore, we are only able to run it on three datasets; AS, Hep-th, and St-Ov.

TABLE I

LINK PREDICTION RESULTS USING FOUR OPERATORS A) WEIGHTED-L1 B) WEIGHTED-L2 C) HADAMARD D) AVERAGE

Op	Algorithm	Dataset			
		AS	Hep-th	Enron	St-Ov
A	DeepWalk	0.957	0.9887	0.8489	0.5595
	node2vec	0.9554	0.9893	0.8533	0.5617
	DynGEM	0.7855	0.6246	-	0.6242
	dynnode2vec	**0.9625**	**0.996**	**0.8922**	**0.66**
B	DeepWalk	0.9577	0.9882	0.805	0.5561
	node2vec	0.9561	0.9894	0.8649	0.5608
	DynGEM	0.7701	0.6161	-	0.6246
	dynnode2vec	**0.9635**	**0.9977**	**0.8981**	**0.6698**
C	DeepWalk	0.888	0.9749	0.8167	0.7456
	node2vec	0.8935	0.9762	0.8236	0.7478
	DynGEM	0.8005	0.5882	-	0.6387
	dynnode2vec	**0.9512**	**0.9975**	**0.9012**	**0.8886**
D	DeepWalk	0.6197	0.5571	0.5271	0.5182
	node2vec	0.6258	0.5577	0.5083	0.5149
	DynGEM	**0.7949**	0.555	-	0.6175
	dynnode2vec	0.7718	**0.6269**	**0.6147**	**0.6824**
node2vec settings (p,q)		(0.5,1)	(0.5,1)	(0.5,1)	(1,2)

C. Link Prediction

Link prediction is an important application of graph embeddings. In this task, future edges are predicted given the previous edges. We consider the link prediction as a classification task similar to [3]. For example, if we have a sequence of graphs $G_0, G_1, ..., G_t$, we predict edges at time t using edges from time 0 to $t-1$. For instance, edges of G_1 are predicted using the positive and negative edges of G_0. For G_2, we use edges from G_0 and G_1. We do the same for all the graphs at future time points.

Edge embedding. The embedding of an edge (u, v) can be computed using embedding vectors of nodes u and v. In the literature, different operators are applied on node embedding vectors to compute edge embeddings including Weighted-L1, Weighted-L2, Hadamard and average as defined in [3].

We report the link prediction results for all the four aforementioned operators in Table I. The results are the average AUC (Area Under Curve) with a grid search over $p, q \in \{0.5, 1, 2, 4\}$. We evaluated all methods on four datasets: AS, Hep-th, Enron and St-Ov. The results show that *dynnode2vec* outperforms baselines in almost all datasets. Among different operators, *dynnode2vec* has the best performance with Hadamard operator and the worst performance with average operator.

D. Node Classification

Another application of node embeddings is in node classification. In supervised classification, nodes have class labels in the dataset. We used logistic regression as the classification method. Similar to link prediction, node embeddings of previous time points are used in predicting class labels of future time points. We compared the results of our method with baselines on dblp dataset. The Micro-F_1 and Macro-F_1 results are reported in Table II. Our result are better than other baselines in terms of Micro-F_1 scores.

TABLE II

NODE CLASSIFICATION RESULTS

Metric	Algorithm	Dataset
		dblp
Micro-F1	DeepWalk	0.5337
	node2vec	0.5272
	dynnode2vec	**0.5415**
Macro-F1	DeepWalk	**0.4141**
	node2vec	0.3847
	dynnode2vec	0.4056
node2vec settings (p,q)		(1,2)

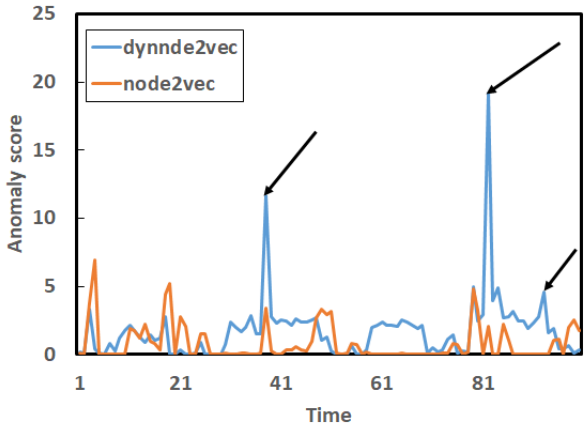

Fig. 1. Anomaly detection results

E. Anomaly Detection

Anomalies are any deviations from normal behavior. There are different categories of anomaly detection methods in dynamic settings including detecting anomalous nodes, edges and change detection. Inspired by DynGEM, we used *dynnode2vec* to detect changes in the dynamic network. We computed the norm of the differences between embedding vectors of common nodes at consecutive time points. High values for the norm signals a significant change in the structure of the graph. We applied *dynnode2vec* and node2vec on a subset of Darpa dataset[19]. Figure 1 shows that there are three major peaks in the dynamic node2vec curve. These spikes correspond to time points that three important attacks occurred in the dataset.

F. Effects of evolving walk generation

In this work, we only generate walks for the nodes in the graph that have changed compared to previous time point. This leads to a significant speedup in *dynnod2vec* running time. In order to show the time efficiency of *dynnod2vec*, we compared the running time of *dynnod2vec* with two methods: node2vec and *dynnod2vec* version denoted by *dynnod2vec-all* that samples walks for all the nodes. The comparison results on three datasets: AS (the first fifty time steps), Hep-th and Enron (the first forty time steps) are shown in Table III. *dynnod2vec* was faster than other methods. All experiments are performed on a windows X-64 with 7 cores, 64 GB RAM and a clock speed of 3.6 GHz.

Additionally in terms of accuracy, we compared the AUC

TABLE III

RUNNING TIME COMPARISON

	AS	Hep-th
node2vec	23.22 min	14.42 min
dynnode2vec-all	23.10 min	13.52 min
dynnode2vec	4.49 min	1.25 min

TABLE IV

COMPARISON OF DYNNODE2VEC VS DYNNODE2VEC-ALL

OP	**Algorithm**	**Dataset**			
		AS	Hep-th	Enron	St-Ov
a	dynnode2vec-all	0.9799	0.9973	0.9114	0.6674
	dynnode2vec	0.9625	0.996	0.8922	0.66
b	dynnode2vec-all	0.9803	0.9976	0.9042	0.6734
	dynnode2vec	0.9635	0.9977	0.8981	0.6698
c	dynnode2vec-all	0.9099	0.9966	0.9022	0.8825
	dynnode2vec	0.9512	0.9975	0.9012	0.8886
d	dynnode2vec-all	0.6488	0.5982	0.6364	0.6798
	dynnode2vec	0.7718	0.6269	0.6147	0.6824
node2vec setting (p,q)		(0.5,1)	(0.5,1)	(0.5,1)	(1,2)

scores of *dynnod2vec* with *dynnod2vec-all*. Table IV indicates the AUC scores in link prediction task. In most cases the AUC for these two methods are not significantly different.

IV. CONCLUSION

In this paper, we propose *dynnode2vec*, a scalable dynamic network embedding that learns representation vectors for dynamic networks. *dynnode2vec* employs the dynamic Skip-gram model and evolving random walks to discover information changes in temporal networks. In the dynamic Skip-gram model, the previous learned embedding vectors are transferred to the next timestamp as initial weights. This results in smooth embedding vectors for graphs over times. Furthermore, the evolving random walks are generated to efficiently reflect the changes in dynamic graph structure. By only considering subset of random walks, *dynnode2vec* can obtain embedding vectors in notably less time without sacrificing accuracy. Our experiments demonstrate the superiority of *dynnode2vec* as compared with the state-of-the-art embedding methods in various tasks. Future work will investigate using other dynamic Skip-gram models [15], [13] for dynamic graph embedding.

V. ACKNOWLEDGEMENTS

We would like to thank Palash Goyal for kindly sharing the source code for the DynGEM method.

REFERENCES

[1] S. Cao, W. Lu, and Q. Xu, "Grarep: Learning graph representations with global structural information," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 2015, pp. 891–900.

[2] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2015, pp. 1067–1077.

[3] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2016, pp. 855–864.

[4] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 701–710.

[5] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations." in *AAAI*, 2016, pp. 1145–1152.

[6] S. Chang, W. Han, J. Tang, G.-J. Qi, C. C. Aggarwal, and T. S. Huang, "Heterogeneous network embedding via deep architectures," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 119–128.

[7] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems*, vol. 151, pp. 78–94, 2018.

[8] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[9] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[10] S. Bhagat, G. Cormode, and S. Muthukrishnan, "Node classification in social networks," in *Social network data analytics*. Springer, 2011, pp. 115–148.

[11] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *Journal of the American society for information science and technology*, vol. 58, no. 7, pp. 1019–1031, 2007.

[12] X. Yu, X. Ren, Y. Sun, Q. Gu, B. Sturt, U. Khandelwal, B. Norick, and J. Han, "Personalized entity recommendation: A heterogeneous information network approach," in *Proceedings of the 7th ACM international conference on Web search and data mining*. ACM, 2014, pp. 283–292.

[13] R. Bamler and S. Mandt, "Dynamic word embeddings," *arXiv preprint arXiv:1702.08359*, 2017.

[14] Y. Kim, Y.-I. Chiu, K. Hanaki, D. Hegde, and S. Petrov, "Temporal analysis of language through neural language models," *arXiv preprint arXiv:1405.3515*, 2014.

[15] N. Kaji and H. Kobayashi, "Incremental skip-gram model with negative sampling," *arXiv preprint arXiv:1704.03956*, 2017.

[16] C. May, K. Duh, B. Van Durme, and A. Lall, "Streaming word embeddings with the space-saving algorithm," *arXiv preprint arXiv:1704.07463*, 2017.

[17] P. Goyal, N. Kamra, X. He, and Y. Liu, "Dyngem: Deep embedding method for dynamic graphs," *arXiv preprint arXiv:1805.11273*, 2018.

[18] Enron network dataset, http://konect.uni-koblenz.de/networks/

[19] R. Lippmann, R. K. Cunningham, D. J. Fried, I. Graf, K. R. Kendall, S. E. Webster, and M. A. Zissman, Results of the DARPA 1998 offline intrusion detection evaluation, in Recent Advances in Intrusion Detection, 1999

[20]] Johannes Gehrke, Paul Ginsparg, and Jon Kleinberg. Overview of the 2003 kdd cup. ACM SIGKDD Explorations Newsletter, 5(2):149151, 2003.

[21] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection, 2014

[22] Math overflow network dataset, http://snap.stanford.edu/data/sx-mathoverflow.html/

[23] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. ArnetMiner: Extraction and Mining of Academic Social Networks. In Proceedings of the Fourteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD'2008).