A comparison of network embedding approaches

Nguyen Thanh Tam, Duong Chi Thang

École Polytechnique Fédérale de Lausanne, Switzerland

***Abstract*—Network embedding automatically learns to encode a graph into multi-dimensional vectors. The embedded representation appears to outperform hand-crafted features in many downstream machine learning tasks. There is a plethora of network embedding approaches in the last decade, based on the advances and successes of deep learning. However, there is no absolute winner as the network structure varies from application to application and the notion of connections in a graph has its own semantics in different domains. In this report, we compare different network embedding approaches in real and synthetic datasets, covering different graph structures. Although our prototype currently includes only two network embedding techniques, it can be easily extended due to our systematic evaluation methodology, and available source code.**

I. Introduction

Graphs are structural data and often a natural representation in different domains such as social networks, biological networks, scientific collaboration networks, and recommender systems. Many downstream machine learning tasks use graph-structured data as feature information to make predictions or discover new patterns [1], [2], [3], [7], [8], [9].

The challenging is that there is no unique way to encode graph-structured information as vectorized features. Traditional machine learning approaches rely on hand-crafted graph statistics such as degrees or clustering coefficients. But they require heavy tuning of model parameters and the selection of kernel functions. The feature engineering step becomes a bottle-neck, needs to be re-done if failed, and cannot be shared among different tasks.

In this decade, deep learning has surged as a universal approach to embed data into the vector space in an unsupervised manner. This open research directions for network embedding, which transform the graph elements into high-dimensional vectors such that the structural information can be encoded and inferred in the embedding space. These automatically learned features appear to outperform hand-crafted ones in many downstream machine learning tasks [4], [10], [11], [12], [13].

While many network embedding techniques have been developed, there has been little work on the evaluation of their performance in a comprehensive manner. The main reason is the lack of a universal setting and a universal metric of success. As a result, understanding the performance implications of these techniques is challenging, since each of them has distinct characteristics to deal with different graph structures and application domains. One, for example, may achieve very high accuracy over strongly connected graphs, while another is sensitive to neighborhood information [4].

To this end, we present a comparison of network embedding approaches within a unified benchmarking framework that offers the following salient features:

- We developed a prototype to integrate the most representative state-of-the-art techniques, including node2vec [5] and GraphSAGE [6].
- We designed a generic, extensible benchmarking framework to assist in the evaluation of different network embedding techniques for downstream machine learning tasks such as node classification and link prediction.
- We offer extensive as well as intensive performance analyses on real-world networks and on synthetic random graphs to cover different domain characteristics. We believe that the analyses can serve as a practical guideline for how to select a well-suited network embedding technique on particular application scenarios.

II. Network embedding approaches

A. Model

Problem definition. The network embedding problem can be formulated as follows:

Definition 1: Given a graph $G = (V, E, A)$ where V is a set of nodes, E is a set of vertices and A is the adjacency matrix, the problem of network embedding is to find a mapping $f : V \rightarrow R^d$ from the nodes to a d-dimensional space where $d << |V|$ and the *homophily* property of the graph is preserved.

Graphs that follow the *homophily* property tend to have connected nodes to be similar. In real-world networks such as social networks, this property can be observed as we tend to be similar to our friends.

Approaches. There are different approaches in network embedding. However, these techniques can be summarized by two different components[4]: encoder and similarity function.

The goal of an *encoder* is to map each node in a network G into a low dimensional vector. Formally, an encoder f is defined as follows:

$$f_\theta : V \rightarrow R^d$$

where θ is the parameter of the encoder.

It maps each node $v_i \in V$ to an embedding $e_i = f_\theta(v_i)$ where $e_i \in R^d$. There are two types of encoders which are *embedding-lookup* encoders and *graph convolutional* encoders. We will discuss their represenatives which are node2vec and GraphSAGE in Section II-B.

On the other hand, *similarity functions* allow to enforce network property that we want to preserve in the embedding space. Regarding the homophily property, we want nodes that

are similar in the network to have close embeddings. Tradionally, the "closeness" of embeddings are usually measured by their dot product $e_i^T e_j$. The most prominent way to measure similarity between nodes in the network are based on random walk. More precisely, the similarity between node u and v in the network corresponds to the probability of visiting v on a random walk starting from u using random walk strategy S. In addition to the encoder, different random-walk-based network embedding techniques differ in the random walk strategy they use. Following a random walk strategy S, random walks from v_i generate a multiset $N_S(v_i)$ which contains all the nodes that are visited on random walks starting from v_i. Detail discussion of random-walk-based similarity function is provided in Section II-C.

Network embedding learning process. As a network embedding technique has two components which are the encoders and the similarity functions, the first step in the learning process is to define the encoder f_θ and the similarity function $similarity(u, v)$. The next step is to optimize the parameters θ of the encoder so that our requirement on the embeddings is satisfied:

$$similarity(v_i, v_j) \sim e_i^T e_j$$

For random-walk-based similarity function, the above requirement can be captured by the following loss function:

$$L = \sum_{v_i} \sum_{v_j \in N_S(v_i)} -log(P(v_j|e_i)) \tag{1}$$

where $N_S(v_i)$ is the multiset obtained from random walks starting from v_i using random walk strategy S. By minimizing the loss function, we can find the parameters θ of the encoder.

The probability $P(v_j|e_i)$ can be parametrized using the softmax function:

$$P(v_j|e_i) = \frac{\exp(e_i^T e_j)}{Z_i}$$

where $Z_i = \sum_{v_k \in V} \exp(e_i^T e_k)$ is the normalization term. This normalization term makes the calculation of Equation 1 to be expensive as the complexity would be $O(|V|^2)$. Traditionally, Z_i is approximated using only m nodes (which are called negative samples) instead of all the nodes in V.

B. Encoder

Embedding lookup. The encoder in this case is just an embedding matrix. In order to find an embedding for a node, we just need to perform a "lookup" to find the corresponding column of the node. Formally, an embedding lookup encoder maps a node v_i which is represented as a one-hot vector $\mathbf{v}_i$ as follows:

$$f_\theta(v_i) = \mathbf{Z}\mathbf{v}_i$$

where $\mathbf{Z} \in R^{d \times |V|}$ is an embedding matrix.

Node2vec [5] uses this type of encoder. In this case, the whole embedding matrix $\mathbf{Z}$ is the parameters θ that need to be learned by the network embedding learning process.

Graph convolution. The general idea of graph convolution encoder is that it takes into account the network structure (e.g. a node neighborhood). It aims to compute a node embedding based on the information aggregated from nodes in its neighborhood. Techniques using graph convolution encoders follow the following framework[4]:

$$\begin{aligned} l_i^0 &= a_i \\ l_{N(i)}^k &= \mathbf{aggregate}(\{l_{j|v_j \in N(i)}^{k-1}\}) \\ l_i^k &= \sigma(W_k \mathbf{combine}(l_{N(i)}^k, l_i^{k-1})) \\ e_i &= l_i^K \end{aligned}$$

where $N(i)$ is the neighborhood of node v_i, K is the width of the neighborhood that encoder considers. Different graph convolution network embedding techniques use different aggregate and combine function. The parameters of the encoders following the above framework are $\{W_k, \forall k \in [1, K]\}$.

GraphSAGE [6] uses graph convolution encoder. To aggregate the neighborhood information, it uses different operations such as mean, max-pool, LSTM[4]... while the combine function is just a concatenation between $l_{N(v)}^k$ and l_i^{k-1}.

C. Random walk similarity function

Although there are different ways to measure the similarity between nodes in the network, random-walk-based approaches are the most popular as they are flexible in how to define similarity while efficient in computation. In addition to the encoder, GraphSAGE and node2vec differ in the random walk strategy. While GraphSAGE uses a vanilla strategy, node2vec uses a biased random walk strategy.

Vanilla random walk. GraphSAGE uses this random walk strategy in which a random walk of fixed length h is performed as follows: Given a node u_i which is the i-th node in the random walk from $u_0 = v$, the next node to be visited is computed as follows:

$$P(u_{i+1} = x | u_i = q) = \begin{cases} \frac{\delta_{xq}}{Z}, & \text{if } (x, q) \in E \\ 0, & \text{otherwise} \end{cases}$$

where δ_{xq} is the transition probability between node x and q. Informally, in an unweighted undirected graph, the probability of moving to a neighbor from a node is equally among the neighbors i.e. $\delta_{xq} = A_{xq} = 1$.

Biased random walk. node2vec proposes a biased random walk strategy in which it wants to combine both BFS and DFS through some control parameters a, b. Formally, let u_i, o be the $i, i-1$-th node in the random walk respectively and the random walker is standing at u_i and considering a neighbor q of u_i, the unnormalized transition probability between u_i and q is computed as $\delta_{u_i q} = \sigma(u_{i-1}, q) A_{u_i, q}$ where

$$\sigma(o, q) = \begin{cases} \frac{1}{a}, & \text{if } d_{oq} = 0 \\ 1, & \text{if } d_{oq} = 1 \\ \frac{1}{b}, & \text{if } d_{oq} = 2 \end{cases}$$

where d_{oq} is the length of the shortest path between o and q. d_{oq} can only be $0, 1, 2$ as either q does not connect to o or q connects directly to o or q connects to o through u_i. Informally, higher a increases the chance of the random walker to visit a previous node. On the other hand, b controls the tendency

of the random walker to explore farther or closer to node u_i. $b > 1$ makes the walker tend to visit nodes closer to u_i while $b < 1$ makes the walker to visit nodes far away from u_i.

III. Benchmark Methodology

Framework. The primary goal of this study is to provide a flexible and poweful tool to support the comparison and facilitate the benchmarking analysis of network embedding techniques. Figure 1 illustrates a simplified architecture of the framework. The source code of the framework is available for download [1].

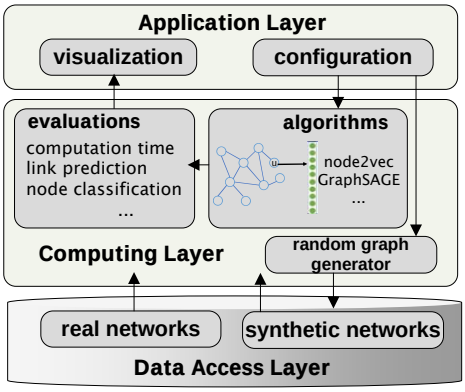


Fig. 1: Benchmarking framework for network embedding

Datasets. We have integrated **real-world networks** from different domains. Table I summarizes the basic statistics of these network datasets. 1) *Facebook* [14]: nodes represent Facebook users, and edges represent a friendship relation between any two users; 2) *Wikipedia* [15]: This is a cooccurrence network of words in Wikipedia articles. The labels represent the Part-of-Speech tags for each word; 3) *BlogCatalog* [16]: This is a network of social relationships of the users listed on the BlogCatalog website. The labels represent users interests inferred through user profiles; 4) *arXiv ASTRO-PH* [17]: This is a collaboration network generated from arXiv papers in the astro-physics field. Nodes represent scientists, and an edge is present between two scientists if they have co-authored a paper; 5) *Protein-Protein Interactions* [18]: This is only a portion of the protein network of Homo Sapiens. The labels represent biological states of the proteins. The network contains several disjoint connected components; 6) *Reddit* [6]: is an online discussion forum. Two posts are connected if the same user comments on both. A node label in this case is the community, or "subreddit", that the post belongs to. The network has several connected components.

Our benchmark also includes **synthetic datasets**, which help users to study unbiased evaluations of network embedding techniques in a wide range of graph models. 1) *Watts-Strogatz random graph model*: $WS(n, k, p)$ attempts to simultaneously capture high clustering and small diameter. Note that with $p = 1$, this model is equivalent to Erdos-Renyi random graph model. Random regular graphs also have a similar structure. We omit them in this report since there is no significant difference in our preliminary results; 2) *Preferential attachment*: It is also known as Barabasi-Albert model $BA(n, m)$, which generates a graph of n nodes such that every node has a fixed m degree.

[1] https://github.com/tamlhp/net_emb

Evaluation Procedure. For a fair comparison, we only use the unsupervised-learning output (which relies only on structural information of the network) of network embedding approaches for downstream machine learning tasks.

- *Node Classification*: is the most common task for evaluating node embeddings. Common applications of node classification include classifying proteins into biological functions and classifying documents, videos, web pages into different categories/communities [5], [6].
- *Link Prediction*: is another downstream classification task that predicts whether a link exists between two nodes in a network. To generate the labeled dataset of edges, we remove 50% of edges chosen randomly from the network while ensuring that the residual network obtained after the edge removals is connected. These removed edges are positive examples. Negative examples are a random subset of node pairs that do not have an edge.

IV. Experimental Evaluation

A. Computation time

This experiment studies the effects of real datasets on computation time, which includes random-walks generation and embedding (training) – two important phases of network embedding methods. In Figure 2, most of the computations of *node2vec* method is generating random-walks, whereas *GraphSAGE* heavily relies on training time. In general, *graphsage* method is slower as it involves an iterative propagation process to collect embedding information from neighbors to a node and pass the embedding information of a current node to the neighbors. However, the *GraphSAGE* method allows us to save the random-walks to disks for reuse [6].

B. Node classification

This experiment studies the predictive power of learned embeddings in classifying a node. To avoid over-fitting, we perform cross-validation by splitting the nodes into three sets: 81% train, 9% validation, 10% test. The best model selected on the validation set is used to evaluate on the test set.

Multi-class. In this setting, each node can be assigned only to a single label. The used classifier is SGD solver with logistic loss function. For simplicity, we calculate the metrics globally by counting the total true positives, false negatives and false positives. Therefore, the precision, recall, F1-score, and accuracy are the same, which is equal to the ratio of true positives over all positives. We also measure Area Under Curve (AUC) score since choosing a cutting threshold for deciding the node labels might be too strict.

In Table II, *node2vec* method is better than *graphsage* in small datasets (wikipedia, blog) but worse in the large dataset (reddit). This is because *graphsage* is designed for embedding dynamic graphs, large-scale graphs, and supervised learning.

Quantity	Dataset					
	(1) facebook	(2) wikipedia	(3) blog	(4) arxiv	(5) protein	(6) reddit
Nodes	4039	4777	10,312	18,772	56,944 [5]	231,443 [5]
Edges	88,234	92,517 [4]	333,983	198,110	818,716	11M
Avg. degree	44	39	65	21	29	100
Diameter	8	3	5	14	N/A [1]	N/A [1]
Avg. cluster coeff.	0.61	0.54	0.46	0.63	0.18	0.17
# Triangles	1.6M	760,505	5.6M	1.4M	2.7M	5.6M
# Node Labels	N/A	40 [2]	39 [2]	N/A	121 [3]	50 [2]

[1] These datasets have disjoint connected components; [2] Multi-class targets; [3] Multi-label targets; [4] Weighted
[5] Each node has hand-crafted features

TABLE I: Statistics for real-world datasets

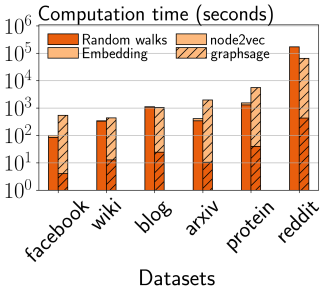


Fig. 2: Computation Time

TABLE II: Single-label node classification

Dataset	Accuracy		AUC	
	node2vec	graphsage	node2vec	graphsage
wikipedia	0.5460	0.4874	0.9589	0.9350
blog	0.3073	0.1595	0.8309	0.7411
reddit	0.6910	0.8085	0.8457	0.9895

When the graph is large, there is much more neighborhood information for *graphsage* to mutually embed the nodes.

Multi-label. In this setting, each node is associated with a set of labels. We rely on the set-based definition of precision and recall to evaluate the individual correctness of each data item. Per item i, *individual precision* P_i is the ratio of correctly predicted labels and the total number of predicted labels, whereas *individual recall* R_i is the ratio of correctly predicted labels and the total number of true labels. For a complete dataset, *precision* P and *recall* R are the respective averages over all items. For the result, we measure the F1-score as a harmonic mean of precision and recall. Only the *protein* dataset has multi-label ground-truth. We also compare *node2vec* and *graphsage* with available hand-crafted features, resulting in F1-scores of 0.7570, 0.7419, and 0.7408 respectively.

C. Link prediction

This experiment predicts whether two nodes have an edge solely based on their embedding vectors. We use Hadamard function to aggregate the vectors of a node pair into an edge vector. We also measure Average Precision (AP) beside AUC, because this metric is not interpolated and is different from computing the AUC with the trapezoidal rule, which uses linear interpolation and can be too optimistic.

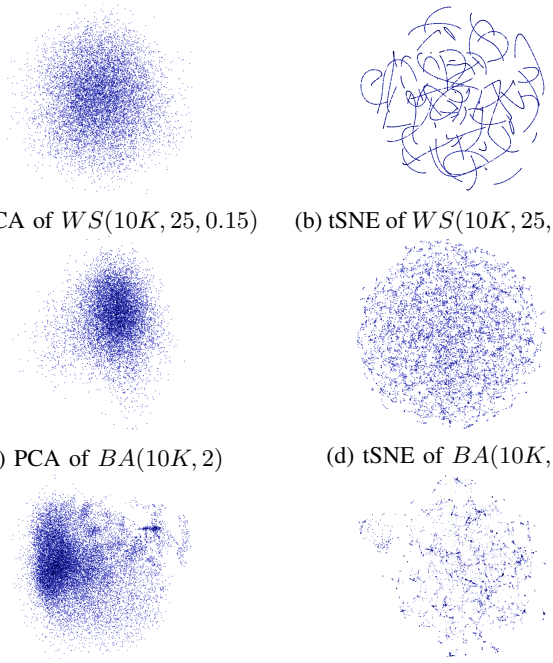
TABLE III: Link prediction

Dataset	AUC		Average Precision	
	node2vec	graphsage	node2vec	graphsage
facebook	0.5372	0.5047	0.5281	0.5116
wikipedia	0.6106	0.6573	0.6065	0.6738
blog	0.5705	0.5599	0.5646	0.5568
arxiv	0.5174	0.5096	0.5139	0.5082

Table III shows that there is no single winner for all cases. The superior performance of *graphsage* over *node2vec* in wikipedia dataset could be explained as this graph is strongly connected (diameter is only 3); and hence, *graphsage* could end up aggregate all the nodes via neighborhood information.

D. Qualitative analysis

Figure 3a-3f highlights the visualization of node embedding vectors of two synthetic datasets and one real dataset (arxiv) by PCA [19] and t-SNE [20]. While the former reveals the relatedness between subgraphs, the latter captures the distance between nodes.



(a) PCA of $WS(10K, 25, 0.15)$ (b) tSNE of $WS(10K, 25, 0.15)$

(c) PCA of $BA(10K, 2)$ (d) tSNE of $BA(10K, 2)$

(e) PCA of arXiv astro-physics (f) tSNE of arXiv astro-physics

References

[1] H. Yin, X. Zhou, B. Cui, H. Wang, K. Zheng, and N. Q. V. Hung, "Adapting to user interest drift for POI recommendation," *TKDE*, pp. 2566–2581, 2016.

[2] H. Yin, Z. Hu, X. Zhou, H. Wang, K. Zheng, N. Q. V. Hung, and S. W. Sadiq, "Discovering interpretable geo-social communities for user behavior prediction," in *ICDE*, 2016, pp. 942–953.

[3] H. Yin, H. Chen, X. Sun, H. Wang, Y. Wang, and Q. V. H. Nguyen, "SPTF: A scalable probabilistic tensor factorization model for semantic-aware behavior prediction," in *ICDM*, 2017, pp. 585–594.

[4] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv:1709.05584*, 2017.

[5] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *KDD*, 2016, pp. 855–864.

[6] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NIPS*, 2017, pp. 1025–1035.

[7] N. Q. V. Hung, N. T. Tam, N. T. Lam, and K. Aberer, "An evaluation of aggregation techniques in crowdsourcing," in *WISE*, 2013, pp. 1–15.

[8] N. Q. V. Hung, D. C. Thang, M. Weidlich, and K. Aberer, "Minimizing efforts in validating crowd answers," in *SIGMOD*, 2015, pp. 999–1014.

[9] N. Q. V. Hung, H. Jeung, and K. Aberer, "An evaluation of model-based approaches to sensor data compression," *TKDE*, pp. 2434–2447, 2013.

[10] H. Yin, L. Chen, W. Wang, X. Du, N. Q. V. Hung, and X. Zhou, "Mobisage: A sparse additive generative model for mobile app recommendation," in *ICDE*, 2017, pp. 75–78.

[11] H. Yin, N. Q. V. Hung, Z. Huang, and X. Zhou, "Joint event-partner recommendation in event-based social networks," in *ICDE*, 2018, pp. 1–12.

[12] H. Chen, H. Yin, W. Wang, H. Wang, Q. V. H. Nguyen, and X. Li, "Pme: projected metric embedding on heterogeneous networks for link prediction," in *KDD*, 2018, pp. 1177–1186.

[13] W. Wang, H. Yin, Z. Huang, Q. Wang, X. Du, and Q. V. H. Nguyen, "Streaming ranking based recommender systems," in *SIGIR*, 2018, pp. 525–534.

[14] J. Leskovec and J. J. Mcauley, "Learning to discover social circles in ego networks," in *NIPS*, 2012, pp. 539–547.

[15] M. Mahoney, "Large text compression benchmark," *URL: http://www. mattmahoney. net/text/text. html*, 2011.

[16] R. Zafarani and H. Liu, "Social computing data repository at asu," 2009.

[17] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *TKDD*, vol. 1, no. 1, p. 2, 2007.

[18] C. Stark, B.-J. Breitkreutz, A. Chatr-Aryamontri, L. Boucher, R. Oughtred, M. S. Livstone, J. Nixon, K. Van Auken, X. Wang, X. Shi *et al.*, "The biogrid interaction database: 2011 update," *Nucleic acids research*, vol. 39, no. suppl_1, pp. D698–D704, 2010.

[19] H. Abdi and L. J. Williams, "Principal component analysis," *Computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

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