Large Project - Network Representation Learning

Nouhaila Agouzal, Abdessamad Badaoui, Nasr Allah Aghelias, Hajar Rouchdi

Abstract. In our project, we conducted a thorough exploration of the Khosla, Setty, and Anand's (2019) paper on unsupervised network representation learning [3]. Our objectives encompassed the implementation of key methods, such as DeepWalk, Node2vec, Line-1, NetMF, and GraphSage, as detailed in the original paper. While replicating experiments with specified datasets, we broadened our scope by incorporating additional datasets from diverse domains. However, due to time and memory constraints, exact reproducibility of results was challenging. Despite these limitations, our results yielded reasonable outcomes, aligning with the conclusions drawn in the original paper. It's worth noting that not all datasets mentioned in the paper were tested for similar reasons. Our article critically discusses these constraints, offering valuable insights and lessons learned from our project.

1. Introduction

The paper, titled "Comparative study of unsupervised network embeddings," delves into the notable advancements in unsupervised network representation learning (UNRL) approaches for graphs. It underscores the diverse methodologies employed, encompassing random-walk techniques, novel optimization objectives, and deep architectures. Despite this diversity, there is a lack of a standardized framework for systematically comparing these embeddings.

The authors contend that UNRL approaches generally revolve around modeling and leveraging the neighborhood or context information of a node. However, the definitions and exploitation of context vary significantly among methods. To address this issue, the paper introduces a framework that unifies random walk-based, matrix factorization, and deep learning-based approaches under a shared context-based optimization function. The objective is to systematically categorize these methods based on their commonalities and differences.

The study conducts an extensive empirical analysis, considering nine prevalent UNRL techniques and eleven real-world datasets with diverse structural properties. Node classification and link prediction, two common tasks, serve as the basis for evaluation. The findings underscore that there is no universally superior method for non-attributed graphs. The choice of an appropriate method relies on specific properties of the embedding methods, the task at hand, and the structural characteristics of the underlying graph.

In this project, our objectives include implementing DeepWalk, Node2vec, Line-1, NetMF, and GraphSage, which are some of the methods mentioned in the original paper. We aim to collect datasets from the original paper and additional ones from diverse domains, evaluating methods in link prediction and node classification tasks. Our focus extends to discussing result reproducibility, generalizability to new datasets, and analyzing the original paper's approach, methodology, and conclusions. A concise summary will conclude our project, highlighting key insights gained.

2. Collected Datasets

From all the datasets mentioned in the paper, we collected 6 datasets, including BlogCatalog, PubMed, Cora, Flickr, Reddit, and Epinions. Additionally, we gathered new datasets not mentioned in the paper, such as POS (Part Of Speech - Wikipedia) and the Protein-Protein Interactions (PPI) dataset (subgraph of the PPI network for Homo Sapiens). Figure 1 provides a summary of their properties.

3. Methods

3.1. DeepWalk

DeepWalk [5] is an algorithm designed for learning low-dimensional representations of vertices in a graph, specifically tailored for social networks. It introduces an unsupervised approach to capture graph structure independently of label distribution, aiming to adapt to evolving social networks and provide community-aware, low-dimensional, and continuous representations. The algorithm utilizes a stream of short random walks in the graph, employing SkipGram and Hierarchical Softmax techniques to optimize and learn representations efficiently. DeepWalk achieves scalability through parallelization and asynchronous stochastic gradient descent, demonstrating its adaptability to large-scale social networks. The resulting representations encode latent community membership and enable robust classification in machine-learning tasks.

The DeepWalk algorithm comprises two main components: a random walk generator and an update procedure. The random walk generator initiates walks from a uniformly sampled vertex, exploring neighbors until a maximum length (t) is reached. The algorithm's core involves an outer loop determining the number of random walks (γ) initiated at each vertex and an inner loop iterating over the vertices. The SkipGram algorithm is employed within the inner loop to update

Dataset	Type	$\|V\|$	$\|E\|$	$\|\mathcal{L}\|$	D	Avg clus	T
POS	Undir	4,777	184,812	40	3	0.538	0.046
PPI[7]	Undir	3,890	76,584	50	∞	0.146	0.092

Table 1. number of nodes ($|V|$), number of edges ($|E|$), number of labels ($|\mathcal{L}|$), diameter (D), average clustering coefficient (Avg clus), transitivity (T)

representations based on the co-occurrence probability of vertices within a window. The optimization involves stochastic gradient descent (SGD), utilizing the back-propagation algorithm. To handle computational complexity, Hierarchical Softmax is introduced, assigning vertices to a binary tree and reducing the complexity of calculating probabilities. Parallelizability is achieved through asynchronous SGD, demonstrating scalability. The overall process involves random walk generation, representation mapping, and hierarchical softmax, optimizing for efficient learning of low-dimensional vertex representations in large-scale social networks.

3.2. Node2Vec

Node2Vec [1] is a semi-supervised algorithm designed for scalable feature learning in networks, particularly effective for tasks like node classification and link prediction. The algorithm formulates feature learning as a maximum likelihood optimization problem, aiming to learn continuous feature representations for nodes that preserve network neighborhoods. It extends the Skip-Gram model to networks, optimizing a novel network-aware, neighborhood-preserving objective function using stochastic gradient ascent. Notably, Node2Vec introduces a flexible notion of a node's network neighborhood, allowing it to learn representations that capture both homophily and structural equivalence in diverse network patterns. The algorithm utilizes biased random walks to sample neighborhoods efficiently, providing control over the search space through tunable parameters. Node2Vec demonstrates superior performance in multi-label classification and link prediction tasks compared to state-of-the-art methods, showcasing its effectiveness, robustness, and scalability on various real-world networks.

Node2Vec, a scalable feature learning algorithm tailored for network analysis tasks like node classification and link prediction, introduces a sophisticated approach to neighborhood sampling through biased random walks. This innovation enables a seamless transition between Breadth-First Search (BFS) and Depth-First Search (DFS) behaviors by incorporating parameters p and q. These parameters govern exploration speed and the distinction between "inward" and "outward" nodes. The algorithm's key features include random walks, guided by transition probabilities, efficient sampling using alias sampling, and an execution process divided into preprocessing, random walk simulations, and optimization phases with Stochastic Gradient Descent (SGD). Node2Vec's parallelizable execution contributes to its scalability and efficiency, making it an effective tool for capturing homophily and structural equivalence patterns in diverse real-world networks.

3.3. NetMF

The NETMF (Network Embedding as Matrix Factorization) method, as detailed in Qiu et al. (2018) document [6], provides a unified matrix factorization framework for network embedding, particularly focusing on improving the methodologies of DeepWalk and LINE. This unification is accomplished by representing various network embedding approaches, including LINE, PTE, DeepWalk, and node2vec, within a matrix factorization framework where the factorized matrices have specific closed forms, as shown in equations 4, 5, 7, and 8 in the same document [6]. NETMF primarily examines the DeepWalk matrix (Eq. 7 [6]). Then, the framework begins by establishing a con-

nection between the DeepWalk matrix and the graph Laplacian. It then proceeds to present the NetMF framework for network embedding based on this matrix factorization approach.

NETMF offers two distinct algorithms for network embedding based on window size T: small (algorithm 3) and large (algorithm 4). For small window sizes, NETMF employs direct computation and factorization of the DeepWalk matrix, suitable for scenarios with manageable computational complexity. To address memory constraints, we utilized sparse matrices in this algorithm, significantly reducing memory usage while preserving essential data. Conversely, for large window sizes, NETMF adopts an approximation algorithm, crucial for larger datasets where direct computation is impractical due to high computational costs.

NETMF requires a symmetric adjacency matrix (A) for its operation, which is crucial for its effective performance. In tasks like node classification, directed graphs are converted to undirected to maintain this symmetry and compatibility with NETMF. However, for tasks like link prediction, where edge directionality is key, this conversion isn't suitable. Thus, we will not compute it for this task as done in the original paper.

Remark: In the original paper, 'T' is defined as the walk length. Accordingly, we have set T to 40 for all the results presented using the NETMF algorithm despite the fact that in Qiu et al. (2018) document [6], they consider it as window size and used T = 1 or 10.

3.4. LINE-1

Large-scale Information Network Embedding is designed to learn embeddings by preserving first-order proximity information in a network. In the context of LINE, first-order proximity refers to the immediate neighbors of a node within the network. LINE optimizes an objective function that balances the likelihood of preserving the proximity of connected nodes and the likelihood of separating the embeddings of unrelated nodes. The advantages of LINE's focus on first-order proximity lie in its scalability to handle large, arbitrary types of networks: undirected, directed and/or weighted, efficiently while capturing local structure. This approach allows LINE to generate embeddings representing the local context of nodes, making it well-suited for various tasks such as link prediction, node classification, and visualization in large-scale information networks. An edge sampling algorithm was used to optimize the objective, which tackles the limitation of the classical stochastic gradient descent and improves the effectiveness and efficiency of the inference.

To optimize the model, we initially employ a batch alias sampler for edge sampling, a technique commonly used in graph-related tasks, especially within the context of machine learning applied to graphs. The alias method serves as a probabilistic data structure facilitating efficient random sampling. In the specific case of edge sampling, this method is frequently employed to select edges from a graph while preserving specific structural properties. Once a given percentage of edges is sampled, we proceed to iterate over them individually. For each edge, we draw K negative samples from the noise distribution $P_n(v) \propto d_v^{3/4}$, where d_v represents the degree of node v, as outlined in the original paper on this method [8]. Subsequently, in each iteration, we update the embeddings for both the current node and the negative samples.

LINE requires a significant amount of time to execute, particularly when dealing with larger datasets. Therefore, we typically sample only 1 to 20% of the network's size for its execution using the edge sampling technique. Unfortunately, due to time constraints and hardware limitations, we were unable to run it on all datasets.

3.5. GraphSAGE

The GraphSAGE algorithm, introduced in 'Inductive Representation Learning on Large Graphs' [2] offers an inductive approach to node embedding in dynamic and large-scale graphs, which effectively handle graphs with emerging and unseen nodes. It employs a neural network framework to integrate node features and those of their neighbors, reaching up

to K layers through various aggregation functions like Mean, MeanPool, MaxPool, LSTM, and Graph Convolution Network (GCN).

The GraphSAGE algorithm starts with initial node features, denoted as x_v, and iteratively updates the embeddings. During the backward step, it learns weight matrices, which are then applied along with a non-linearity function up to a specified depth K. This procedure involves aggregating the embeddings from a node's neighbors and its own embeddings, culminating in refined node embeddings z_v that effectively capture local neighborhood information.

The parameters of GraphSAGE are learned via a graph-based loss function in an unsupervised setting, promoting similarity among nearby nodes and distinctness among distant nodes, using a sigmoid function and negative sampling.

4. Tasks

4.1. Link Predection

To implement the link prediction algorithm for directed graphs, we utilized the NetworkX library for graph manipulation and the scikit-learn library for evaluation metrics. The process involves creating a training graph and a test graph by randomly removing a specified percentage of edges from the original graph (G). The removed edges are stored in the test set, which is balanced by randomly selecting some negative edges. We ensured that no node is isolated during this process to guarantee the learning of embeddings for all nodes. Following the training on the modified graph, link prediction is assessed on the test set using the inner product of node embeddings normalized by a sigmoid function. Model performance is evaluated using the Receiver Operating Characteristic Area Under the Curve (ROC AUC) score, indicating the model's ability to distinguish between true and false links in the test graph. For undirected graphs, some adjustments were made in constructing the test set, particularly with negative edges, employing three different methods described in the paper: Method 1 - Random negative edges in the test set, Method 2 - 50% of test negative edges created by reversing true edges, and Method 3 - All true edges of the test set are reversed to create negative edges.

4.2. Node Classification

We followed the guidelines of the original paper that used the one-vs-rest Logistic Regression. However, we altered the method slightly since the phrasing was vague with respect to how to do the multi-label node classification. We used the classes OneVsRestClassifier and LogisticRegression from the scikit-learn library. To predict labels on the test input, we compute the probabilities of each class and pick the classes that have the largest values. The number of classes to expect is pre-determined from the true labels of the true test output set. This way, we know beforehand how many classes we need to predict. After computing the prediction, we calculate the Micro and Macro scores (using f1_score from scikit-learn) over 5-fold cross validation similarly to the original paper.

5. Results

The results of our experiments are displayed in Figures 2, 3, and 4. Symbols T and M represent, respectively, issues with runtime and memory when executing the corresponding method. Both DeepWalk and Node2Vec utilize the skip-gram algorithm which could be imported from the gensim library as Word2vec [4]. Skip-gram is a natural language processing algorithm that embeds words close to each other in sentences using stochastic gradient descent. We assume that the version utilized by the original paper [3] is older than the one we did use, hence the slight differences between our results and those of the paper. Furthermore, since the bigger the number of random walks (in both DeepWalk and Node2Vec), the more time consuming the algorithm is. Consequently, we used less

[1] For GraphSAGE algorithm, the calculations are conducted for just the second graph of the dataset due to memory limitations.

Table 2. Link Prediction on undirected graphs using the ROC-AUC score

Method	BlogCatalog	Flickr	POS	PPI
DeepWalk	0.492	0.737	0.556	0.678
Node2Vec	0.514	0.795	0.470	0.681
NetMF	0.631	M	0.617	0.690
LINE-1	0.476	T	0.2	0.52

Table 3. Link Prediction on directed graphs using the ROC-AUC score

Method	Cora			PubMed		
	0%	50%	100%	0%	50%	100%
DeepWalk	0.789	0.625	0.516	0.862	627	0.500
Node2Vec	0.756	0.614	0.507	0.876	0.645	0.500
LINE-1	0.480	0.475	0.5	T	T	T

Table 4. Node Classification using the Micro and Macro F1-score

Method	Cora		PubMed		BlogCatalog		Flickr		POS		PPI[1]	
	mic	mac	mic	mac	mic	mac	mic	mac	mic	mac	mic	mac
DeepWalk	59.49	48.53	69.55	67.32	41.56	27.34	39.13	28.2	46.97	8.49	20.56	17.99
Node2Vec	60.24	46.98	69.01	66.55	41.24	26.93	38.87	27.48	47.38	8.88	19.48	16.60
NetMF	62.15	43.65	M	M	40.52	24.01	M	M	48.98	9.99	24.33	19.64
LINE-1	T	T	T	T	13.51	3.72	T	T	38.17	4.28	17.21	15.11
GraphSAGE	54.70	35.65	67.80	67.35	12.22	3.70	M	M	-	-	25.79	19.99

walks than recommended in larger datasets like Flickr.

For NetMF method, we applied both algorithms with small and large window across different datasets and for the two tasks (LP and NC), however, the results presented are from the algorithm that demonstrated the best performance.

For large datasets, we try to reduce T to 1 and use sparse matrices in the NetMF algorithm but, unfortunately, this strategy does not resolve memory issues perfectly, as observed with the Flicker dataset (h= 16384) and with PubMed dataset. Conversely, for the remaining datasets, we adhered to the original paper's specifications by setting the walk length T to 40 (and h = 256 for large window algorithm).

LINE-1 demands a significant amount of runtime, hindering its execution on all collected datasets. For the datasets we did test, we only sampled a small portion of the entire graph in each epoch so that the algorithm finishes in a reasonable time. This limitation may explain the poor results observed in certain datasets, such as PPI in the node classification task, or the Cora dataset in the link prediction task.

In our implementation of the GraphSAGE algorithm, we used the mean aggregation function with K=2, indicating that the node embeddings were generated by aggregating feature information from each node's immediate neighbors and their neighbors in turn, up to two hops away. For minibatch processing, NeighborLoader, a powerful tool in PyTorch Geometric that facilitates the efficient handling of large graphs by creating mini-batches for training, was utilized for neighbor sampling, and the Adam optimizer was employed for training. However, due to time constraints, the link prediction part was not completed.

6. Evaluating reproducibility and generalization of paper results in our experiments

We did not reproduce exact numerical results from the original paper; however, we closely

approximated them most of the time, particularly for methods that are less affected by time and memory constraints. Here is a summary of the main insights we derived from our experiments in comparison with the original paper :

We noticed that in most of the datasets, we were able to reproduce the fact that DeepWalk is competetive in terms of Node Classification even though it is the oldest. We hence agree with the paper on the fact that this puts into question of the biased random walks in Node2Vec.

In the context of Link Prediction on the BlogCatalog dataset, the NETMF algorithm performs better than the other algorithms, aligning with the assertions made in the original paper. On the other hand, when it comes to Node Classification, NETMF's performance varies across datasets. Specifically, while it registers a superior F1 micro score for the Cora dataset, it does not maintain this superiority for BlogCatalog where the DeepWalk and Node2Vec algorithms show better performance. Generally, NETMF's F1 scores are comparable to those of DeepWalk and Node2Vec, even when applied to directed datasets like as Cora dataset.

The conclusion drawn in the original paper regarding LINE outperforming DeepWalk and Node2Vec was not corroborated by our experiments. This discrepancy can be attributed to the fact that our study exclusively utilized the first-order proximity of LINE, whereas the conclusions in the paper were based on the combined LINE-1+2 approach.

For the GraphSAGE algorithm, our experiments primarily focused on employing the mean aggregator, diverging from the methodology in the original paper that experimented with various aggregators to determine the most effective one for each dataset. This specific focus on the mean aggregator in our approach might contribute to the differences noted between our experimental outcomes and those detailed in the paper for node classification tasks. The limited scope of our aggregator selection implies that a broader exploration, similar that of the original study, could potentially bring our results more in line with those findings. Furthermore, there is a possibility for enhancing our results by more rigorously fine-tuning the hyperparameters, an aspect that holds potential for further optimization in our experiments

7. The approach, methodology, and conclusions of the original paper

Since there exists multiple embedding algorithms, and some of them significantly differ from each other on a conceptual level (e.g. DeepWalk and Node2Vec can be considered similar DeepWalk and LINE-1 can not), we need a universal framework to compare their performance. This is the main objective of this study and we think they have done a great job and succeeded. However, there a few points we would like to expand on:

7.1. Datasets

It was commendable that they managed to test their methods on multiple and large datasets. However, since the hardware they worked with was too advanced for us, we couldn't reproduce their results on some datasets. Furthermore, whenever they mentioned a particular dataset they only cited the paper that primarily exploit it without directly referencing the dataset itself. Consequently, we had to search for the datasets separately, and each had its own format (matlab, csv, etc).

7.2. Tasks

Testing the algorithms on both Node Classification and Link Prediction was a sound approach. Furthermore, having a baseline for the Node Classification was brilliant idea. It is always good to have an estimate on what to expect from the algorithms before running them. One issue, however with the Node Classification, is that they mentioned that they used one-vs-rest Logistic Regression but they didn't specify how they assigned multiple labels to the input datasets.

8. Key takeaways from the project.

We learned that it was difficult to reproduce the results of this scientific study, it should be even more difficult to produce the results the first time and compile them into a paper. This made us appreciate more the efforts put into scientific research. Throughout the process, we acquired valuable skills, navigating each stage of a machine or deep learning algorithm, encompassing data collection, preprocessing, implementation, and evaluation. Addressing numerous bugs and errors encountered enhanced our understanding of the strengths and weaknesses inherent in each method. Furthermore, this project was the best opportunity to learn how to distribute tasks efficiently between group members, otherwise it would have been significantly more difficult to achieve the results we obtained. What is more, aside from the technical component of the project, it was psychologically demanding i.e. we needed to persevere through hardships and adapt to dead ends since they are to be expected during a project of this caliber.

References

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

[2] Will Hamilton, Zhitao Ying, and Jure Leskovec. "Inductive representation learning on large graphs". In: *Advances in neural information processing systems* 30 (2017).

[3] Megha Khosla, Vinay Setty, and Avishek Anand. "A comparative study for unsupervised network representation learning". In: *IEEE Transactions on Knowledge and Data Engineering* 33.5 (2019), pp. 1807–1818.

[4] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[5] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. "Deepwalk: Online learning of social representations". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014, pp. 701–710.

[6] Jiezhong Qiu et al. "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*. 2018, pp. 459–467.

[7] Chris Stark et al. "The BioGRID interaction database: 2011 update". In: *Nucleic acids research* 39.suppl_1 (2010), pp. D698–D704.

[8] Jian Tang et al. "Line: Large-scale information network embedding". In: *Proceedings of the 24th international conference on world wide web*. 2015, pp. 1067–1077.