Report - Machine Learning Advanced DD2434
1.4 Large Project - Network Representation Learning

groupe 11
Falconnier Pierre, Flandre Corentin, Goupil Benoît, Le Dez Michel
piefal@kth.se, flandre@kth.se, goupil@kth.se, micld@kth.se

from November 28, 2023 to January 8, 2024

Contents

1 Introduction

With significant progress in unsupervised-network representation learning (UNRL), there has been a resurgence of unsupervised methods for network embeddings for graphs. One of the major problems of this resurgence in UNRL domain is the lack of comprehensive large-scale studies to compare approaches under different experimental conditions.

The paper on which this report is based, "A comparative study for unsupervised network representation learning" [5], proposes to conduct a large comparative study of URNL by a large-scale empirical study of 9 recent UNRL techniques on 11 real-world dataset. The comparative study used the two commons tasks of node classification and link prediction to compare these techniques. It is important to highlight the multitude and diversity of this study which use different type of large networks datasets (5 undirected and 6 directed) with different properties (social, citation and collaboration networks). The comparison aims to highlights the strengths and weakness of approaches under different graph and task.

In this report, we reimplemented the primary methods from the "report's based paper" [5], using standard Machine Learning (ML) and Deep Learning (DL) languages. We applied these methods to the datasets suggested in the paper, as well as to an additional dataset, with the aim of reevaluating these methods in the context of two specific tasks: link prediction and node classification. Additionally, we will discuss the original paper, the reproducibility of its experiments, and the insights we gained through this process.

2 Methods

First, we implemented differents methods discussed in the paper: DeepWalk [6], Node2vec [3], Line-1 [8], NetMF [7] and GraphSAGE [4]. Secondly, we collect some datasets (presented in the paper and additional) of similar size for revelant comparison. Finally, we evaluate the methods on two commons tasks: link prediction and node classification.

2.1 Implementation of the methods

2.1.1 GraphSAGE

GraphSAGE [4] introduces an efficient sampling technique for aggregating neighborhood information. For each node, it samples a fixed-size neighborhood and applies a neural network to aggregate features from these neighbors. Various aggregator functions—mean, pooling, and LSTM aggregators—can be employed to combine neighbor features. Sampling a fixed-size neighborhoods instead of the entire neighborhoods enhances the scalability for large graphs.

Our implementation of the model uses Pytorch Geometric [1] for handling graph data and operations, and Pytorch [2] for optimizing the model.

The loss function adopted is the negative sampling objective from the original paper. We aimed to replicate as closely as possible the hyperparameters from the comparatuve paper[5]: a two-layer architecture with the mean aggregator, a batch size of 512, a learning rate of 0.0001, and a dropout rate of 0.4. The model samples 25 neighbors in the first layer and 10 in the second layer. We implemented the "big" version of the model as described in the original paper, which means the hidden linear layer has a size of 1024, and the output layer a size of 128. The value of Q in the loss expression is unspecified, so we set $Q = 1$, giving equal importance to positive and negative edges. We empirically sampled 10 negative edges per positive edge to compute the expectation term in the negative loss component. The number of epochs is not specified, so we ran the model for 10 epochs, consistent with the original paper's approach for supervised tasks.

GraphSAGE requires node features, but some datasets, like the Epinions dataset, do not provide these. The original paper does not specify how to handle such cases, so we computed each node's features as the mean of its neighbors' degrees.

2.1.2 Deepwalk

DeepWalk performs uniform random walks on a graph. The similarity between two vertices in the graph is quantified by the ratio of the weight of the edge connecting them to the total positive edge weight in the context graph. This ratio indicates how similar two vertices are, with the similarity being proportional to the value of this ratio. DeepWalk uses truncated random walks of a specified length from each vertex to create sequences of vertices. For each pair of vertices within a window of a given size around a particular vertex in the sequence, an edge is formed in the context graph. This context graph is crucial for defining the relationships between different vertices in the network. The key hyperparameters in DeepWalk are the walk length (here 40), window size (here 10), and the number of negative samples (here 5). These parameters determines how the similarities between vertices are encoded and how the context matrix is formed.

2.1.3 Node2vec

The Node2Vec method is described in the paper entitled "node2vec: Scalable feature learning for networks" [3]. Unlike DeepWalk, which performs uniform random walks, Node2Vec follows a 2nd order random walk. This approach quantifies the similarity of vertices based on the ratio of the weight of an edge (connecting two vertices) to the total positive edge weight in the context graph. Essentially, the similarity between two vertices is proportional to this ratio. Node2Vec introduces bias in its random walks, characterized by two hyperparameters p and q. These parameters significantly influence the performance of Node2Vec, especially in graphs with high reciprocity and clustering coefficients. They influence the likelihood of revisiting nodes or exploring outward: - p controls the likelihood of revisiting the node where the random walk just came from. - q controls the likelihood of moving to a node that is far away from the current node.

The method employs truncated random walks of a specified length from each vertex to create a sequence of vertices. For each pair of vertices within this sequence, an edge is formed in the context graph. This graph helps define the relationships and context between different vertices in the network

In order to implement Node2Vec, we used Pytorch Geometric for graph process and Torch. The node embeddings are learned based on walks via negative sampling (NS) optimization. NS is a way used for the optimization as used in GraphSAGE and LINE-1.

To get the best quality of resulting embeddings on our task (NC or LP), we can perform a grid search over p and q. The grid 0.25, 0.50, 1, 2, 4 is used in the experiments by the authors of the original paper using 10-fold cross-validation.

We used the same hyperparameters as given in [5], in the tables 1 and 2 of the 3 Parameter Settings part.

2.1.4 NetMF

The method NetMF, introduced in the paper by Jiezhong Qiu et al. [7], presents a unified framework for network embedding, which aims to learn low-dimensional representations for nodes in a graph. It bridges the gap between two popular approaches: matrix factorization methods and random walk-based methods (like DeepWalk). NetMF works by approximating the matrix obtained from DeepWalk's random walks with a matrix factorization technique. Essentially, it first computes a matrix representation that encodes the local and global structure of the graph of node pairs from random walks. Then, it factorizes this matrix using Singular Value Decomposition (SVD) to obtain dense, low-dimensional embeddings for each node. These embeddings can then be used for various downstream tasks like node classification, link prediction, and visualization.

We then proceeded to implement the method, setting the parameters as outlined in the original paper by Jiezhong Qiu et al. We evaluated the algorithm's performance across different graph sizes and complexities. The set of parameters includes the rank of the matrix used for DeepWalk (i.e., the number of non-zero eigenvalues), the window size, the final embedding dimension, and the negative sampling rate. We configured the parameters to the values recommended in the main paper. These values are a window size of 10, a matrix rank of 256, an embedding dimension of 128, and a negative sampling rate of 1. While this configuration proved efficient for graphs of smaller sizes, running the algorithm on larger graphs presented memory challenges. Specifically, we encountered issues with loading the entire matrix into RAM for larger networks

2.1.5 Line-1

LINE-1 is a method using first order proximity to create embedding for the vertices of the graph. For a pair of vertices, first order proximity equals 1 if there is an edge, 0 otherwise. Therefore, it considered that two vertices are similar if and only if there is an edge between them. Then, to find the embedding of the vertices, it maximises the log-likelihood among the existing edges, which can be written as: $\sum_{(i,j)\in E} \log p(v_i, v_j)$, where the probability of finding an edge between v_i and v_j is simply the dot product similarity between their representations, normalized by the sigmoid function. A trivial solution to this optimisation problem can be reached by setting all the embedding' components to ∞, which effectively yields to $p(v_i, v_j) = 1$ for all $(i, j) \in E$. To avoid this, negative sampling is used as a regularization term in the loss. Concretely, suppose we are currently maximising the likelihood of finding an edge between v_i and v_j, then, to avoid the embedding of diverging, negative sampling samples a number K of negative vertices (vertex that are not in the neighborhood of v_i) and add a term to the objective function which try to maximise the likelihood of not finding these non-existing edges. To sample the negative vertices, a multinomial distribution whose support is the set of vertices and whose probability distribution is proportional to the degree of the vertices to the power 3/4 is used. The probability distribution is not uniform in order to reduce the odds of sampling a high degree vertices which have more chance of being in the neighborhood of the actual vertex, which we do no want.

To implement this method, we used Pytorch, which has efficient tools for optimising embedding's weights. In the original paper, they use a number of negative samples $K = 5$, Asynchronous Stochastic Gradient Descent (ASGD) with an initial learning rate of 0.025, a linear decrease of the learning rate, mini-batch of size 1 and a maximum number of mini-batch sampled of 10 billions. Given the gap of computational power with the author of the paper, it was not possible to use ASGD and mini-batch of size 1, therefore, we tried different optimizer along with different size of mini-batch, initial learning rate and decaying technique but it was really difficult to optimise these hyper-parameters because the execution time was too long for each trial.

Notice that this method only learns source representation and doesn't do the difference between the source representation of a vertex and its context representation, therefore, it is not recommended to use it for directed graphs but here we used it anyway to compare its performance with the other methods.

2.2 The datasets

We used a majority of the datasets proposed in the paper. Some datasets (like Cora, PubMed and CiteSeer) are available in the class torch_geometric.datasets, so we use the class torch_geometric.data.Data as standard to store them.

- BlogCatalog, Flickr, Youtube, Reddit, Twitter and Epinion for the social category
- Cora and PubMed for the citation category

We also used an additional dataset: CiteSeer which represents a graph of citations, similar to PubMed.

2.3 Task prediction evaluation

To evaluate the quality of the embeddings obtained by each method, we tested them on two tasks: link prediction (LP) and node classification (NC).

2.3.1 Link Prediction

For this task, we followed the method described in the paper. First, we randomly removed 50% of the existing edges in the network to create a test set. This left the remaining graph as our training graph for learning embeddings with our model. We were careful not to isolate nodes, as this would cause them to disappear from the training graph and lack an embedding representation.

Then, we balanced the test set with negative edges, i.e., edges that do not exist in the network, to evaluate our classifier's ability to detect these negative edges. To do this, we randomly selected pairs of vertices, ensuring that no edge existed between them. For directed graphs, we also included in our test set the opposite direction of existing edges, enabling our classifier to distinguish between one direction of an existing edge and another that does not exist. We accomplished this by randomly inverting a fraction of the existing edges. As proposed in the paper, we tested three different settings for directed graphs. In the first setting, where the fraction is 0%, all negative edges came from randomly selected non-existing edges. In the second setting, with a fraction of 50%, the negative edges were equally composed of randomly selected non-existing edges and reversed existing edges. Finally, in the third setting, where 100% of the negative edges came from reversed existing edges.

To compute the probability of finding an edge in the test set, we used the dot product similarity between the embeddings of the two vertices of the edge, normalized by the sigmoid function. Then, to assess the quality of our embeddings, we calculated the area under the ROC curve, derived from our probability predictions and the true labels of the test set. The results for the various graphs and methods can be found in Table 1 and Table 2.

2.3.2 Node classification

For this task, we follow the paper's method. Basically, for a given model and graph, we compute the graph embeddings and use them for a 5-fold multi-label classification using one-vs-rest logistic regression models. We then evaluate the classification by computing the mean of both Micro-F1 and Macro-F1 over the folds.

Because the paper does not give any additional information, we used the classes LogisticRegression and OneVsRestClassifier from scikit-learn with default hyperparameters.

3 Results

3.1 Link Prediction Task

See Table 1 and Table 2 for the results of each model on various datasets.

	Cora			**CiteSeer**			**Twitter**			**Epinion**		
	0%	50%	100%	0%	50%	100%	0%	50%	100%	0%	50%	100%
DeepWalk	0.8127	0.5770	0.5376	0.8452	0.5901	0.5101	0.5000	0.5000	0.5	0.5839	0.5302	0.5377
Node2vec	0.8312	0.6439	0.5000	0.9028	0.6784	0.5003	0.5000	0.5000	0.5000	0.9040	0.7163	0.7169
Line-1	0.9994	0.7226	0.4998	0.9375	0.7100	0.4947	0.5919	0.5469	0.4997	0.6017	0.5502	0.5020
GraphSAGE	0.9624	0.7464	0.4991	0.9624	0.7193	0.4849	0.4966	0.4975	0.5000	0.6531	0.5536	0.5003

Table 1: Link Prediction Results on directed graphs - ROC-AUC

	BlogCatalog	**Youtube**	**Reddit**	**Flickr**
DeepWalk	0.5098	0.5973	×	×
Node2Vec	0.5120	0.6102	×	×
Line-1	0.8162	×	×	×
NetMF	0.6812	×	×	×
GraphSAGE	0.5352	×	×	×

Table 2: Link Prediction Results on undirected graphs - ROC-AUC

3.2 Node Classifictation Task

See Table 3 for the results of each model on various datasets.

	BlogCatalog		**PubMed**		**Cora**		**Reddit**		**Flickr**		**Youtube**		**CiteSeer**	
	mic.	mac.	mic.	mac.	mic.	mac.	mic.	mac.	mic.	mac.	mic.	mac.	mic.	mac.
DeepWalk	38.89	27.46	72.83	66.45	61.98	42.89	×	×	12.42	9.090	×	×	40.71	25.43
Node2vec	39.82	28.76	72.85	67.42	62.39	43.66	×	×	19.26	11.29	×	×	56.80	49.81
Line-1	33.21	23.91	55.29	48.94	17.21	12.14	×	×	07.17	00.53	×	×	41.84	36.76
NetMF	34.38	18.65	81.65	80.36	55.64	42.25	×	×	×	×	×	×	61.17	56.21
GraphSAGE	07.49	06.41	80.96	80.40	39.79	28.63	×	×	00.71	00.06	×	×	74.63	69.33

Table 3: Node Classification Results - Micro-F1 and Macro-F1 scores

4 Discussion

4.1 About our results

About link prediction, our results are coherent with the paper's in the way that, the greater the fraction of reversed edges is in the directed test graph, the worse the predictions are. Some ROC-AUV are 0.5 or close to it, which means that the predictions on the considered dataset are not better than random ones. It might indicate that the training was not effective or that something went wrong. However, it remains consistent with some of the paper's ROC-AUV. However, we are surprised to see that Line-1 outperforms the other methods on the link prediction of BlogCatalog.

The node classification results highlight differences in performance across datasets, especially for the Flickr dataset for which all the methods struggled to make accurate predictions. The same happened for the Youtube and Reddit datasets, showing the difficulty of our methods to scale to large datasets.

The paper [5] gives hints on which approach might be more suitable given the structural properties of the CiteSeer graph (like its density, degree distribution, assortativity, number of triangles, clustering coefficient, etc.) given here: https://networkrepository.com/citeseer.php. Then, the most appropriate approaches could be Node2Vec or DeepWalk, which can better capture the asymmetrical relationships between nodes and effectively capture the low-density and the degree distribution might be more suitable. But this is not what we observe on the node classification task, as the other approaches led to better F1-scores.

4.2 About reproducibility of experiments

The main difficulty in the reproducibility of experiments was to manage larges graphs during the node classification taks due to a problem we get about batch processing. Effectively, we get really better results without batch iteration but the time for training was too long for larger datasets. Then, we were unable to produce results for every datasets and methods.

Overall, the discrepancies between our results and those reported in [5] can be attributed to several factors. Firstly, there is no assurance that the dataset we used is identical, as its precise origin is not disclosed in the paper [5]. Secondly, many essential hyperparameters (epochs, loss parameters...) are not specified, which could lead to substantial variances in results. Moreover, the methods for the NC and LP tasks are not detailed and we had to make some assumptions that may lead to different results. Regarding the underperformance of GraphSAGE on BlogCatalog and Flickr datasets, our approach of generating features based on neighborhood mean degree may not be suitable.

Another difficulty in the reproducibility of the paper's experiments was the lack of memory and computational power of our computers. The papers describes the resources and version of ML/DL languages used, it's a good thing in terms of comparison.

Apart from these few difficulties, experiments was relatively reproducible because the paper provides good tracks for experiments even if it's correspond to lot of paper's reconstitution.

4.3 About the paper

We found that the paper serving as the basis for this project is highly relevant for comparing different UNRL algorithms, as it is a recent publication that compares various methods across multiple graphs—an approach not previously undertaken with these algorithms. Significantly, it presents results for an array of algorithms applied to a diverse range of datasets, focusing on different tasks. This aims to identify the most suitable method depending on the properties of our graph (such as directed/undirected, high/low reciprocity, high/low spectral separation, high/low clustering coefficient, etc.) or our labels.

The proposed "9. ACKNOWLEDGEMENTS" section provides a summary of the main findings discovered during the experiments. We were able to leverage this to form hypotheses about our results and performance on new datasets, guided by the characteristics of the corresponding graphs. One limitation clearly stated in the paper is the challenge in decisively determining which method to adopt for new cases, attributed to the complexity of the study. Indeed, the choice of an appropriate UNRL method depends heavily on the specific case at hand, and there is no single method that outperforms the others in all scenarios. A thorough analysis of the properties of the graph representing our network is essential.

4.4 About this project

In terms of ML knowledge, we each delved deeply into the intricacies of the Unsupervised Learning algorithms proposed in the subject matter. This immersion led us to utilize powerful ML tools like Torch and adhere to certain standards. Additionally, it necessitated close collaboration on these standards to ensure that we could compare our methods under optimal conditions, using precisely the same task evaluation approaches.

However, we found this project to be quite stressful, challenging, and demanding in every aspect. There are numerous methods that are difficult to implement from scratch, along with many datasets to locate and process, particularly for those of us using Pytorch and Pytorch Geometric for the first time. We frequently encountered memory issues and faced time-consuming runs. The difficulty gap between this project and the other two large projects offered in this course was surprising and, in our view, somewhat unfair. Nevertheless, this was our first foray into working with graphs, and we believe that this project was the most interesting due to its objective of comparing different approaches rather than focusing on a single one.

Our code can be found here: https://github.com/PierreFalconnier/advanced-ml-kth.git . Feel free to contact us if you have some questions.

References

[1] PyG Documentation — pytorch_geometric documentation - https://pytorch-geometric.readthedocs.io/en/latest/.

[2] PyTorch - https://pytorch.org/.

[3] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks, 2016.

[4] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018.

[5] Megha Khosla, Vinay Setty, and Avishek Anand. A comparative study for unsupervised network representation learning. *IEEE Transactions on Knowledge and Data Engineering*, page 1–1, 2020.

[6] 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*, KDD '14. ACM, August 2014.

[7] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 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 2018. ACM, February 2018.

[8] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 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, May 2015.