Reproducible Evaluations of Network Representation Learning Models Using EvalNE

Alexandru Mara
Ghent University
Ghent, Belgium
alexandru.mara(at)ugent.be

Jefrey Lijffijt
Ghent University
Ghent, Belgium
jefrey.lijffijt(at)ugent.be

Tijl de Bie
Ghent University
Ghent, Belgium
tijl.debie(at)ugent.be

ABSTRACT

In this paper we introduce EvalNE, a Python toolbox for the evaluation of network representation learning methods. The main goal of EvalNE is to help researchers perform consistent and reproducible evaluations of new representation learning methods, compare these with the state-of-the-art or conduct benchmark studies. The toolbox can evaluate models and assess the quality of representations through data visualization and a variety of downstream prediction tasks including sign and link prediction, network reconstruction, and node multi-label classification. EvalNE streamlines evaluation by providing automation and abstraction for tasks such as hyperparameter tuning and model validation, node and edge sampling, node-pair embedding computation and performance reporting. The framework can also evaluate approaches independently of the programming language and with minimal user interaction. As a command line tool, configuration files describe the evaluation setup and guarantee consistency and reproducibility. As an API, EvalNE provides the building blocks to design any evaluation setup while ensuring that common issues, such as data leakage, are ruled out. Finally, to showcase the capabilities of our tool, we present the results of a recent benchmark on representation learning for link prediction conducted using EvalNE.

KEYWORDS

representation learning; evaluation; software; reproducibility

ACM Reference Format:
Alexandru Mara, Jefrey Lijffijt, and Tijl de Bie. 2021. Reproducible Evaluations of Network Representation Learning Models Using EvalNE. In *Proceedings of Workshop on Graph Learning Benchmarks at WWW21 (GLB21).* ACM, New York, NY, USA, 5 pages.

1 INTRODUCTION

Network representation learning or network embedding (NE) methods aim to learn low-dimensional representations of network nodes as vectors, typically in the Euclidean space. These representations can then be directly used for network visualization or to efficiently perform a variety of downstream prediction tasks. These tasks include sign prediction [e.g., 15, 26], link prediction [e.g., 9, 10], network reconstruction [e.g., 25, 28], and node classification [e.g., 17, 22]. A higher performance on these downstream tasks is then generally associated with a better representation of the original network.

The evaluation of embedding methods based on downstream task results, however, is challenging, as it requires a number of additional steps and design choices which can confound the results, harm reproducibility, and are prone to errors. Firstly, for all the above-mentioned tasks one must ensure that the input data is preprocessed in a consistent manner, such that all methods can be correctly evaluated. For instance, many NE methods require the input network to contain a single connected component while some do not. Additionally, the necessary sets of train and test data (nodes or edges) can be selected according to a variety of approaches and be of different sizes [14]. Two popular choices for edge sampling, for example, are pseudo-random train-test split with certain constraints or timestamp-based sampling (i.e. use the most recent edges for testing and the remaining ones for training). Another source of evaluation inconsistencies is negative sampling. This technique is commonly used in link prediction evaluation for generating the *negative class* to be used in a subsequent binary classification of edges and non-edges [11]. The negative sampling approach and the relative sizes of the train and test non-edge sets vary between scientific works. Yet another challenge in NE evaluation through downstream tasks is that embedding methods provide different types of outputs. While some approaches only output node embeddings [23], others can provide node-pair embeddings [21] or even node similarities [10]. As such, specific pipelines for different output types are required. Particularly important in this case is the computation of node-pair embeddings from node embeddings which has been shown to strongly impact method performance [9]. Finally, additional complexity stems from hyperparameter tuning, not only for the evaluated embedding methods, but also for the downstream task itself.

Contributions. To address the above-mentioned challenges and simplify the evaluation of NE methods on downstream prediction tasks, we propose EvalNE. The toolbox can be used both as a standalone application or integrated in existing code. As a command line tool, it leverages configuration files that describe the complete setup, from the tasks, datasets and methods to use, to preprocessing, sampling, hyperparameter tuning and metrics to optimize. These configuration files provide flexibility to define or replicate different experimental setups and are sufficient for complete reproducibility. After evaluation, complete performance reports and graphics are generated as well as log files detailing the issues encountered. When integrated in existing code, EvalNE provides access to a series of classes and functions implementing the various components required for method evaluation on different tasks. We note that the library does not implement any particular NE method but provides the necessary accessories to evaluate any off-the-shelf approach. This is achieved by delegating method execution to the command line interface. The framework does however implement popular

GLB21, Apr 2021, Ljubljana, Slovenia
2021.

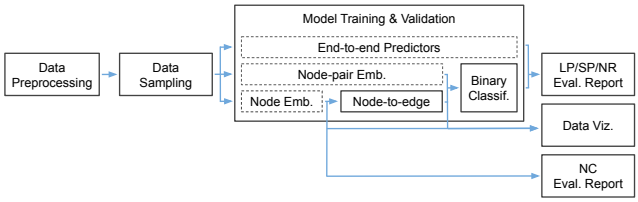

Figure 1: Block diagram of EvalNE showing the methods and tasks that can be evaluated. In the diagram downstream link prediction, sign prediction, network reconstruction and node classification tasks are abbreviated as LP, SP, NR, and NC, respectively. Dashed blocks correspond to user-specified methods with different output types while the remaining blocks are provided by EvalNE.

link prediction heuristics such as Adamic-Adar, preferential attachment, Katz similarity, etc. for both directed and undirected networks. Lastly, the practicality and flexibility of EvalNE are demonstrated by showcasing a recent benchmark study ([14]) conducted using our tool.

2 ARCHITECTURE

Conceptually, the design of EvalNE can be seen as a set of interconnected building blocks which provide all the necessary components for evaluation. This modular structure, shown in Figure 1, simplifies code maintenance and addition of new features and allows one to evaluate methods with different output types (node embeddings, node-pair embeddings, node similarities, etc.) on different downstream tasks. Pragmatically, the blocks correspond to different classes and functions as we summarize next.

Data Preprocessing. The toolbox builds upon the popular NetworkX framework and offers additional functions for loading and storing networks, pruning and relabelling nodes, removing self-loops, sampling edges, restricting networks to the main connected component and obtaining common statistics.

Data Sampling. For evaluation, sets of train, test and validation data (nodes or edges) must be obtained from the input networks. The particular sampling strategy varies for each downstream task. In node classification, for instance, this is relatively straightforward as test/validation nodes can be sampled randomly. For the remaining tasks, which require sets of edges, we provide a variety of sampling methods. From timestamp-based to random sampling that ensures the train edges continue to span a connected subgraph (a key requirement for many NE methods). For negative sampling, EvalNE provides two alternatives: the *open world* and the *closed world* assumptions. The former considers the case where non-edges are *not known* a priori and, thus, they must be sampled from the train graph spanned by the train edges. In this case, the train non-edges may overlap with the set of test edges. The latter considers the case where non-edges are known a priori and, thus, train non-edges cannot overlap with the test edges. The *EvalSplit* class encapsulates the sampling and negative sampling functionalities.

Model Training & Validation. EvalNE provides specific classes called *Evaluators* for each downstream task. These classes manage model training with appropriate input data, hyperparameter tuning based on grid search and collection of results. For link and sign prediction and network reconstruction, methods providing node similarities can be directly evaluated. Those that output node or node-pair embeddings are evaluated through binary classification. EvalNE supports any Scikit-learn binary classifier and implements Logistic Regression with cross-validation as a default. Methods that only output node embeddings additionally require a binary operator to compute node-pair embeddings. For this, EvalNE implements the following operators: average, Hadamard, weighted L_1 and weighted L_2 (see [9]).

Evaluation Report & Visualization. EvalNE can evaluate method scalability through wall clock time, and accuracy through fixed-threshold metrics and threshold curves. The library implements over 10 different fixed-threshold metrics including AUC, precision, recall, and F-score, among others. Integrated threshold curves, which present method performance for a range of threshold values, include precision-recall [13] and ROC [8] curves. Methods for visualizing embeddings and graphs are also provided as well as dimensionality reduction techniques. Method specific performance is recorded in *Results* objects while global evaluation summaries are provided through *Scoresheet* objects.

3 RELATION TO OTHER SOFTWARE

To the best of our knowledge, only three libraries present capabilities similar to those of EvalNE. These are: `OpenNE` [24], `GEM` [6], and `Karateclub` [20]. These frameworks, however, focus on providing implementations of a variety of state-of-the-art NE methods rather than on evaluation. `OpenNE` provides limited functionality for evaluating multi-label classification only, `GEM` additionally includes basic evaluation for node visualization and link prediction, while `Karateclub` aids the user in coding its own evaluation pipelines. Unlike these frameworks, EvalNE is not limited to a pre-defined set of NE method implementations. Instead, our framework can evaluate any off-the-shelf approach (note that this includes all implementations in the above-mentioned libraries). By leveraging the power of the system's command line, EvalNE can efficiently evaluate methods independently of the programming language or interface and with little to no user interaction. This feature is present in both API and command line application uses. Additionally, EvalNE is the only library currently available that provides full automation of the evaluation pipeline including hyperparameter tuning, and edge sampling.

4 SOURCE CODE AND DOCUMENTATION

The EvalNE source code is available on GitHub[1] and is released under the MIT free software license. GitHub also provides a variety of mechanisms for community contribution such as: bug tracking, feedback submission, and pull requests for integrating new features. The library's code style complies with PEP 8 and the docstring documentation follows the standard Numpy format. The toolbox is compatible with Python 2 and Python 3 and can be easily installed using `pip`. Supported platforms include Linux, Apple macOS, and Microsoft Windows. EvalNE only depends on a small number of

[1] EvalNE GitHub repository https://github.com/Dru-Mara/EvalNE

open-source Python packages: NumPy, SciPy, NetworkX, Scikit-learn, and Matplotlib. Other packages such as OpenNE or GEM are optional and can provide implementations of different NE methods.

Regarding the toolbox documentation, this is automatically managed and hosted online by Read the Docs[2]. Detailed instructions on the installation and use both as an API and command line tool are provided. Simple examples of the high-level use are also included as well as more advanced examples of the low-level use and integration with existing Python code. Finally, the library contains pre-filled configuration files which allow one to reproduce the experimental sections of several influential papers on network representation learning.

5 EXPERIMENTAL EVALUATION WITH EVALNE

In this section we summarize the setup and main results of an empirical study on network representation learning for link prediction conducted using EvalNE. For complete evaluation details and results we kindly refer the reader to the original manuscript [14]. The aim of this study was to establish the amount of progress made in the area in recent years and determine the impact on method performance of a variety of factors including: network structure, hyperparameter tuning, train set size, edge sampling strategy, node embedding operators and binary classifiers.

Methods. The experimental evaluation includes 6 link prediction heuristics provided by EvalNE and a total of 23 implementations of 17 different NE methods. Whenever possible, original implementations by the authors were evaluated together with others available in the `OpenNE` and `GEM` libraries. Among the heuristics we find Common Neighbours (CN), Jaccard Coefficient (JC), Adamic-Adar (AA), Resource Allocation (RA), Preferential Attachment (PA) and *NE_heuristics*, a method based on combining the above-mentioned predictors in a 5-dimensional embedding. The NE methods evaluated include Skip-Gram based approaches: DeepWalk [17], Node2vec [9], Struc2vec [18], Metapath2vec [7] and Watch Your Step [1]; matrix factorization approaches: Graph Factorization [3], GraRep [5], HOPE [16], Laplacian Eigenmaps [4], Locally Linear Embeddings [19], M-NMF [27] and AROPE [28]; neural network approaches: SDNE [25], PRUNE [12] and VERSE [23]; and probabilistic approaches: LINE [22] and CNE [10].

Data and Setup. The evaluation was conducted on 7 real-world datasets from different domains including relational (StudentDB), social (Facebook, BlogCatalog), collaboration (GR-QC, AstroPh), biological (PPI) and language networks (Wikipedia). The number of nodes in these graphs ranges between 10^2 and 10^4 and of edges between 10^3 and 10^5. In all cases the networks were treated as undirected and only the network structure was used to learn embeddings. Two experimental setups were designed. The first (LP1), aimed to establish the best possible method performance by tuning model hyperparameters via grid search. The second (LP2), was used to identify the impact on performance of different edge sampling techniques, node embedding operators, and binary classifiers. For setup LP2 method hyperparameters were kept fixed and set to the values recommended by the original authors of each NE method. Each experimental setup was described in one EvalNE configuration file. In setup LP1 up to three different hyperparameters were tuned for each method including the edge embedding operator used to calculate node-pair embeddings from node embeddings (see [9]). Link predictions were obtained through logistic regression on the node-pair embeddings for all methods (*node emb.* pipeline in Figure 1) with the exception of AROPE and CNE, which directly provide node similarities (*end-to-end* pipeline in Figure 1).

Results. Table 1 presents the average AUC performance of each method over three independent repetitions of the experiment with different sets of train and test edges. In this case experimental setup LP1 was used. For the NE methods embeddings of sizes 8, 32 and 128 were trained and the best results are presented. The best performing method within each type of approach is highlighted in bold and the overall best for each network on grey background.

Firstly, we observed that all methods improved in performance as the embedding dimensionality increased. The only exception to this is CNE which provided similar results across all dimensions with the best being achieved for size 8. From the results in Table 1 we highlight the excellent performance, on most datasets, of the RA and NE_heuristics approaches. The latter, together with GraRep, achieves the highest average AUC amongst all methods of 0.963. From their average performance over all networks and the average AUC rank we see that the top performers are: VERSE, NE_heuristics, GraRep and CNE. Another important observation from the results in Table 1 is that method performance varies significantly between different implementations of the same NE methods (e.g. up to 11,7% for LINE on StudentDB).

In the absence of additional data, such as node or edge attributes, our experiments suggest an overall improvement in performance as the order of proximity between nodes, captured by a model, increases. GraRep, the best performing NE method, captures relations of up to order 8. HOPE and AROPE, also top performers, capture relations up to order 4 while the remaining first and second order methods, present a lower performance. Exceptions to this pattern are VERSE and CNE. The former is a second-order method that employs a non-linear transformation able to preserve more information from the original network. The latter is a first order method that models structural information (node degrees) in a prior allowing more flexibility to the embedding.

Regarding the relation between network structure and model performance, we observe that most methods present high AUC scores on networks with clear community structures such as Facebook, AstroPH and GR-QC. On the other hand, k-partite networks such as StudentDB pose a challenge to both heuristics and NE methods. This is mainly due to the fact that similar node neighbourhoods in this case do not imply that two nodes should be connected. Other networks such as BlogCatalog and Wikipedia present similar structures with small diameters, high clustering coefficients and large average degrees which results in cluttered representations and, thus, low link prediction performance.

Additional experiments comparing setups LP1 and LP2 reveal that hyperparameter tuning results in average improvements in link prediction AUC of less that 1% while execution times increases up to 30x. Popular random walk methods such as DeepWalk and

[2]EvalNE documentation https://evalne.readthedocs.io/en/latest/

Table 1: AUC scores and standard deviations over 3 experiment repetitions for setup LP1 where hyperparameters are tuned and $d = 128$ for all methods except CNE where $d = 8$. Note that 0.000 in the table means < 0.0005. The best method within each type of approach is highlighted in bold and the overall best for each column on grey background.

Methods	StudentDB	Facebook	BlogCat.	GR-QC	AstroPH	PPI	Wikipedia	Avg. AUC	Avg. Rank
CN	0.630±0.011	0.992±0.001	0.948±0.000	0.959±0.001	0.990±0.000	0.863±0.003	0.900±0.002	0.860±0.210	17.21
JC	0.630±0.011	0.990±0.001	0.770±0.002	0.959±0.001	0.990±0.000	0.839±0.001	0.623±0.005	0.756±0.260	22.07
AA	0.630±0.011	0.993±0.001	0.952±0.000	0.959±0.001	0.991±0.000	0.867±0.003	0.919±0.002	0.864±0.211	13.57
PA	0.922±0.008	0.842±0.003	0.955±0.001	0.839±0.006	0.879±0.001	0.905±0.002	0.920±0.001	0.894±0.041	16.50
RA	0.630±0.011	**0.994±0.001**	**0.958±0.000**	0.959±0.001	0.991±0.000	0.867±0.003	**0.931±0.002**	0.867±0.212	10.64
NE_heuristics	**0.966±0.004**	0.993±0.000	0.956±0.001	**0.976±0.003**	**0.993±0.000**	**0.927±0.001**	0.929±0.004	**0.963±0.026**	**5.00**
DeepWalk	0.906±0.005	0.990±0.000	0.943±0.000	**0.986±0.001**	0.984±0.000	0.905±0.001	0.903±0.002	0.945±0.040	13.14
DeepWalk_opne	0.906±0.010	0.991±0.000	0.943±0.000	0.985±0.002	0.983±0.000	**0.906±0.001**	0.904±0.001	**0.945±0.039**	**13.00**
Node2vec	0.948±0.009	**0.994±0.000**	0.938±0.001	0.985±0.003	0.989±0.001	0.840±0.006	0.893±0.002	0.941±0.054	13.29
Node2vec_opne	0.897±0.004	0.991±0.001	0.929±0.001	0.986±0.002	**0.992±0.000**	0.900±0.001	0.901±0.001	0.942±0.043	13.57
Struc2vec	0.933±0.010	0.833±0.004	**0.953±0.001**	0.842±0.005	0.874±0.001	0.904±0.002	**0.918±0.001**	0.894±0.042	18.00
Metapath2vec	**0.981±0.005**	0.942±0.003	0.948±0.000	0.804±0.006	0.858±0.002	0.880±0.003	0.903±0.001	0.902±0.058	19.29
WYS	0.819±0.016	0.940±0.003	0.915±0.002	0.833±0.008	0.855±0.004	0.853±0.005	0.864±0.010	0.868±0.042	25.57
GF_opne	0.868±0.007	0.983±0.000	0.898±0.001	0.933±0.005	0.947±0.001	0.837±0.004	0.834±0.003	0.900±0.054	23.57
GraRep_opne	0.969±0.003	**0.993±0.000**	**0.962±0.001**	**0.984±0.002**	**0.990±0.000**	**0.921±0.001**	**0.922±0.001**	**0.963±0.029**	**5.29**
HOPE_gem	**0.989±0.001**	0.990±0.000	0.955±0.000	0.952±0.002	0.950±0.001	0.909±0.002	0.919±0.001	0.952±0.029	11.29
HOPE_opne	0.914±0.002	0.989±0.000	0.944±0.000	0.920±0.005	0.947±0.000	0.872±0.005	0.916±0.001	0.929±0.034	18.57
LE_gem	0.906±0.010	0.992±0.000	0.800±0.003	0.975±0.003	0.934±0.004	0.760±0.005	0.767±0.005	0.876±0.097	20.36
LE_opne	0.906±0.011	0.992±0.000	0.803±0.005	0.977±0.001	0.932±0.002	0.764±0.003	0.771±0.006	0.878±0.092	20.00
LLE_gem	0.890±0.008	0.990±0.000	0.704±0.002	0.970±0.004	0.895±0.006	0.726±0.008	0.741±0.005	0.845±0.114	23.57
M-NMF	0.944±0.009	0.992±0.000	0.936±0.001	0.983±0.002	0.983±0.000	0.878±0.008	0.913±0.001	0.947±0.040	13.36
AROPE	0.982±0.002	0.991±0.001	0.955±0.001	0.968±0.001	0.967±0.000	0.910±0.001	0.918±0.002	0.956±0.029	10.79
SDNE_gem	**0.987±0.004**	0.979±0.002	0.952±0.000	0.945±0.002	0.971±0.001	0.910±0.002	0.918±0.001	0.952±0.028	13.29
SDNE_opne	0.985±0.002	0.987±0.000	0.953±0.000	0.957±0.007	0.969±0.002	0.898±0.005	0.917±0.001	0.952±0.032	13.86
PRUNE	0.901±0.010	0.838±0.002	0.956±0.000	0.836±0.003	0.874±0.001	0.904±0.003	**0.920±0.001**	0.890±0.042	17.71
VERSE	0.935±0.010	**0.994±0.001**	**0.956±0.002**	**0.990±0.002**	**0.996±0.000**	**0.919±0.002**	0.919±0.002	**0.959±0.033**	**4.57**
LINE	**0.963±0.004**	0.993±0.001	0.931±0.000	**0.984±0.002**	**0.991±0.000**	0.877±0.002	0.882±0.002	0.946±0.048	12.64
LINE_opne	0.850±0.010	0.991±0.000	0.932±0.000	0.933±0.001	0.963±0.001	0.895±0.002	0.894±0.003	0.923±0.045	19.29
CNE	0.946±0.009	**0.994±0.000**	**0.967±0.001**	0.980±0.000	0.976±0.000	**0.928±0.001**	**0.922±0.001**	**0.959±0.026**	**6.00**

Node2vec, for which parameter tuning is tedious, show minimal improvements in AUC.

In relation to the size of the training set, we observe that most embedding methods capture the network structure well when presented with 50% or more of the network edges at training time. Values below 50% generally result in poor representations. In this same regime, link prediction heuristics perform significantly worse than NE methods. Regarding the approach used to split edges in train and test, four approaches provided by EvalNE were compared, three pseudo random and one based on edge timestamps. Method performance did not vary significantly when different random sampling approaches were used. Timestamp based sampling, however, impacted the performance of the link prediction heuristics positively and that of the NE methods negatively.

Finally, the evaluation also showed that averaging evaluation results over several sets of train and test edges —in order to obtain unbiased estimates of method performance— becomes less necessary as the train network sizes surpass 10^3 edges. The standard deviation of the AUC between any two repetitions of the experiment with different random seeds was lower that 0.0004.

6 CONCLUSIONS

In this paper we introduced EvalNE, a Python toolbox for consistent and reproducible evaluation of network representation learning methods. In Sec. 5 we also demonstrated the potential of the library by using it to perform a large scale and easily reproducible benchmark, evaluating several of its capabilities including hyperparameter tuning and train set sampling. We note, however, that the use of EvalNE is not restricted to mid sized networks and the link prediction task evaluated in Sec. 5. Indeed, in [2] we already used EvalNE to conduct an experimental evaluation on networks with millions of nodes and edges, and in [15] we used it to empirically evaluate different signed network embedding methods. We thus hope that EvalNE can become a standard for empirical evaluations of graph representation learning methods on the wide range of supported downstream tasks, and believe that this would greatly benefit reproducibility in this fast growing research area. Our future plans include the expansion of framework's visualization capabilities, support for evaluation of attributed networks, addition of new downstream tasks (such as node classification via link prediction) and the creation of a website tracking the benchmark results.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC Grant Agreement no. 615517, from the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme, and from the FWO (project no. G091017N, G0F9816N, 3G042220).

REFERENCES

[1] Sami Abu-El-Haija, Bryan Perozzi, Rami Al-Rfou, and Alex Alemi. [n.d.]. Watch Your Step: Learning Graph Embeddings Through Attention. arXiv:1710.09599

[2] F. Adriaens, A. Mara, J. Lijffijt, and T. De Bie. 2020. Block-Approximated Exponential Random Graphs. In *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*. 70–80. https://doi.org/10.1109/DSAA49011.2020.00019

[3] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. 2013. Distributed large-scale natural graph factorization. In *Proc. of WWW*. 37–48.

[4] Mikhail Belkin and Partha Niyogi. 2002. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Proc. of NIPS*. 585–591.

[5] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. GraRep: Learning graph representations with global structural information. In *Proc. of CIKM*. 891–900.

[6] Siheng Chen, Sufeng Niu, Leman Akoglu, Jelena Kovacevic, and Christos Faloutsos. 2017. Fast, Warped Graph Embedding: Unifying Framework and One-Click Algorithm. *CoRR* abs/1702.05764 (2017).

[7] Yuxiao Dong, Nitesh V. Chawla, and Ananthram Swami. 2017. Metapath2Vec: Scalable Representation Learning for Heterogeneous Networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Halifax, NS, Canada) *(KDD '17)*. ACM, New York, NY, USA, 135–144. https://doi.org/10.1145/3097983.3098036

[8] Tom Fawcett. 2004. *ROC Graphs: Notes and Practical Considerations for Researchers*. Technical Report.

[9] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proc. of KDD*. 855–864.

[10] Bo Kang, Jefrey Lijffijt, and Tijl De Bie. 2019. Conditional Network Embeddings. *Proc. of ICLR*.

[11] Bhushan Kotnis and Vivi Nastase. 2017. Analysis of the Impact of Negative Sampling on Link Prediction in Knowledge Graphs. *CoRR* abs/1708.06816 (2017).

[12] Yi-An Lai, Chin-Chi Hsu, Wen Hao Chen, Mi-Yen Yeh, and Shou-De Lin. 2017. PRUNE: Preserving Proximity and Global Ranking for Network Embedding. In *Proc. of NIPS*. 5257–5266.

[13] Ryan N. Lichtenwalter and N. V. Chawla. 2012. Link Prediction: Fair and Effective Evaluation. In *Proc. of ASONAM*. 376–383.

[14] Alexandru Mara, Jefrey Lijffijt, and Tijl De Bie. 2020. Benchmarking network embedding models for link prediction : are we making progress?. In *Proc. of DSAA* (Sydney, Australia).

[15] Alexandru Mara, Yoosof Mashayekhi, Jefrey Lijffijt, and Tijl de Bie. 2020. CSNE: Conditional Signed Network Embedding. In *Proc. of CIKM* (Virtual Event, Ireland). 1105–1114.

[16] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric transitivity preserving graph embedding. In *Proc. of KDD*. 1105–1114.

[17] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online Learning of Social Representations. In *Proc. of KDD*. 701–710.

[18] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. 2017. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 385–394.

[19] Sam T. Roweis and Lawrence K. Saul. 2000. Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science* 290, 5500 (2000), 2323–2326. https://doi.org/10.1126/science.290.5500.2323 arXiv:https://science.sciencemag.org/content/290/5500/2323.full.pdf

[20] Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar. 2020. Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs. In *Proc of CIKM*. 3125–3132.

[21] Wenzhuo Song, Shengsheng Wang, Bo Yang, You Lu, Xuehua Zhao, and Xueyan Liu. 2018. Learning node and edge embeddings for signed networks. *Neurocomputing* 319 (2018), 42–54.

[22] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale information network embedding. In *Proc. of WWW*. 1067–1077.

[23] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. 2018. VERSE: Versatile Graph Embeddings from Similarity Measures. In *Proc. of WWW* (Lyon, France). 539–548.

[24] Cunchao TU, Cheng YANG, Zhiyuan LIU, and Maosong SUN. 2017. Network representation learning: an overview. *SCIENTIA SINICA Informationis* 47, 8 (2017), 980–996.

[25] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *Proc. of KDD*. 1225–1234.

[26] Suhang Wang, Jiliang Tang, Charu Aggarwal, Yi Chang, and Huan Liu. 2017. Signed network embedding in social media. In *Proc. of SDM*. 327–335.

[27] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. 2017. Community Preserving Network Embedding. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence* (San Francisco, California, USA) *(AAAI'17)*. AAAI Press, 203–209.

[28] Ziwei Zhang, Peng Cui, Xiao Wang, Jian Pei, Xuanrong Yao, and Wenwu Zhu. 2018. Arbitrary-Order Proximity Preserved Network Embedding. In *Proc. of KDD*. 2778–2786.