UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

INFORMATION ENGINEERING DEPARTMENT

MASTER'S DEGREE IN COMPUTER ENGINEERING

APEROL from Networks: Analyzing Pipeline and Embedding Representations for Optimized Learning (from Networks)

Professor
Fabio Vandin

Students
Marco Annunziata, 2160851
Silvia Mondin, 2141201
Sveva Turola, 2160852

ACADEMIC YEAR 2025-2026

Contents

1 Introduction

Graphs have emerged as a natural model for the representation and analysis of data and complex systems in various domains. For instance, link prediction in social networks can help to understand the spreading process of rumors or epidemics [1], and in biological networks, it is commonly used for predicting novel interactions between proteins [2].
However, most traditional machine learning algorithms are not designed to work directly with graph-structured data, as they require numeric vectors or matrices as inputs. To address this challenge, embedding techniques have been developed to learn low-dimensional vector representations of nodes, edges, or entire graphs while preserving their topological properties.
Consequently, in order to design an effective embedding technique, it is necessary to consider several criteria [3]. Two of these are adaptability and scalability. An embedding method is considered adaptable if it can be utilized for various data and tasks. The scalability of an embedding technique is determined by its capacity to process large-scale networks within a reasonable amount of time.

The objective of this study is to evaluate known embedding methods from the perspective of these two criteria.
From the perspective of adaptability, the objective is to determine the potential impact of the graph domain on the precision of an embedding method. In other words, the objective is to ascertain whether there exist approaches that are better suited for a particular field, whether there is one embedding technique that consistently outperforms the others, or if the methods are comparable. In the context of scalability, the objective is to identify the embedding methods that are better suited to process large graphs. This is accomplished by evaluating the tradeoff between the accuracy of the predictions and the time required for training and inference.

2 Node embedding

Node embedding is the task of mapping nodes into an embedding space, where each node v is represented by a vector $\mathbf{z_v} \in \mathbb{R}^d$ using the mapping function $ENC : v \to \mathbf{z}_v$. The encoding of nodes into embeddings has to be such that similarity between each pair of nodes is kept in the correspondent embeddings. In order to achieve this, a similarity score $S(u, v)$ is computed between pairwise nodes and a decoder function $DEC : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is used for computing a similarity score for the embeddings. The discrepancy between the two scores is then compared and minimized by means of a loss function, such as the empirical loss:

$$\mathcal{L} = \sum_{i,j} \ell(DEC(\mathbf{z}_u, \mathbf{z}_v), S(u, v))$$

The node embedding methods are defined by the characteristics of the encoder and decoder.

In this project, four of these methods are considered:

- Node2Vec [4], a shallow encoding method and essentially a tunable version of the DeepWalk algorithm, which is often used in the scientific literature as the baseline for comparing node embedding methods;
- LINE [5], a shallow encoding which is an highly-scalable approximation algorithm;
- Deep Variational Network Embedding in Wasserstein Space [6], a deep encoding method using a Variational Autoencoder. This method has a particular approach of encoding each node to a probability distribution, so in this case the similarity score between embeddings is a distance between probability distributions, which can be measured using the Wasserstein distance;
- GraphSage [7], a deep encoding method making use of a GNN that aggregates the features of each node's neighborhood and learns to generate node embeddings from that.

3 Link Prediction

In this study, the main task considered is link prediction. The link prediction task is defined as the estimation of the existence of a link between two nodes in a network, based on the observed links. Formally,

given a graph $G = (V, E)$, where V is the set of nodes and E the set of edges, the link prediction task aims to predict whether an edge $(u, v) \in (V \times V) \setminus E$ exists or not, based on the observed edges in E.

In order to perform link prediction using node embeddings, a common approach is to compute a feature vector for each pair of nodes (u, v) based on their embeddings $\mathbf{z}_u$ and $\mathbf{z}_v$. Subsequently, the feature vectors are utilized as input to a binary classifier that predicts the presence or absence of a link between the two nodes.

It is important to note that the quality of the predictions is potentially dependent on two factors: the embeddings themselves and the classifier utilized. To this end, classifiers of varying complexity have been considered:

- Support Vector Machines [8], where eventually the kernel trick could be used to handle non-linear relations between embeddings;
- Random Forest [9], for its really fast computation;
- Multilayer Perceptron [10], for accurate predictions.

4 Datasets

The datasets that used in this project are nine and they are divided into three categories: road networks, social networks, and biological networks. For each of these categories, three datasets of increasing size were selected: one small, one medium, and one large.

The datasets, with their main properties (i.e., the number of nodes, the number of edges, if it is directed, and if it is weighted), are presented in Table 1.

Network	$\|V\|$	$\|E\|$	**Directed**	**Weighted**
Pennsylvania [11, 12]	1,088,092	3,083,796	✓	✗
Padua (province) [13]	122,728	304,184	✓	✓
Hong Kong (city) [14]	43,620	91,542	✓	✓
Italian Covid-19 Retweet Network [15]	221,574	2,332,721	✓	✓
Deezer [16, 17]	54,573	846,915	✗	✗
GitHub Developers [18, 19]	37,700	289,003	✗	✗
Mus Musculus Protein Interactions [20] [1]	20,969	808,988	✗	✓
Saccharomyces cerevisiae Protein Interactions [20] [2]	5,786	103,831	✗	✓
Bio-grid-fission-yeast [21]	2,026	12,637	✗	✗

Table 1: Datasets used in the project

5 Experiments

The link prediction task requires a sample of edges from the graph (positive edges) and a sample of edges from the complement graph (negative edges, in this case, do not include self-loops). Consequently, each dataset will be partitioned as follows. Consider graph $G = (V, E)$, where V denotes the set of vertices and E is the set of edges. A first split of the set of edges E is needed for the embedding algorithms, creating E_{embed} and E_{pred}. E_{embed} is used for the subgraph $G_{\text{embed}} = (V, E_{\text{embed}})$ given as input to the embedding methods: the shallow embedding methods will use it to learn a lookup table Z of node embeddings, whereas the deep embedding methods aim to learn a function f that can generalize well. The remaining edges in E_{pred} are instead reserved for the link-prediction task, where the second split occurs: from E_{pred} the sets E_{train}, E_{val}, E_{test} are created for the training, validation and test of the prediction model.

In order to achieve a balanced distribution of negative edges, each set contains an equal number of

[1] Confidence score of > 0.4
[2] Confidence score of > 0.7

positive and negative examples. Furthermore, in order to avoid data leakage, the sampled negative edges must be distinct from those used internally by the embedding algorithms during training.

The input of the prediction models are edge embeddings. For each edge (u, v), its corresponding embedded representation is constructed by concatenating the embeddings z_u, z_v of its incident nodes.

The split used in this study is: (from E) 80% for E_{embed} and 20% for E_{pred}; (from E_{pred}) 60% for E_{train}, 20% for E_{val} and 20% for E_{test}.

Two metrics that are generally used for link prediction are the Area Under the Receiver Operating Characteristic curve (AUROC), and the Area Under the Precision-Recall curve (AUPR). AUROC is historically considered the primary performance metric used to evaluate the performances of link prediction methods [22]. However, AUROC favors accurate classification of positive examples, at the cost of misclassifying the negative ones. In a scenario like the link prediction problem, which is inherently biased towards the negative class, this approach may not be optimal and can overestimate the performance of the methods. Consequently, the AUPR curve was also selected, as it can provide better evaluation of link prediction in the presence of class imbalance.

In regard to the implementation of the embedding techniques mentioned in Sec 2, the following implementations will be adopted: [23] for Node2Vec, [24] for LINE, [25] for DVNE, [26] for GraphSage. Depending on specific needs, some of these could be modified or used as an inspiration for different implementations. In particular, DVNE was not evaluated on directed graphs in the original work [6], meaning it may be necessary to explore adaptations for this setting. If no satisfactory solution is found, DVNE will be restricted only for undirected graphs to mantain a fair comparison.

The experiments will be conducted using the DEI cluster *"Blade"*. The specifications for the nodes of the cluster are available in the open documentation [27]. More information about the project's memory usage and runtime will be available in the final report.

5.1 Results

...

6 Conclusions

...

References

[1] Romualdo Pastor-Satorras et al. "Epidemic processes in complex networks". In: *Reviews of modern physics* 87.3 (2015), pp. 925–979.

[2] Björn H Junker and Falk Schreiber. *Analysis of biological networks*. Vol. 2. Wiley Online Library, 2008.

[3] Anthony Baptista et al. "Zoo guide to network embedding". In: *Journal of Physics: Complexity* 4.4 (2023), p. 042001.

[4] Aditya Grover and Jure Leskovec. *node2vec: Scalable Feature Learning for Networks*. 2016. arXiv: 1607.00653 [cs.SI]. URL: https://arxiv.org/abs/1607.00653.

[5] Jian Tang et al. "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, pp. 1067–1077. DOI: 10.1145/2736277.2741093. URL: http://dx.doi.org/10.1145/2736277.2741093.

[6] Dingyuan Zhu et al. "Deep Variational Network Embedding in Wasserstein Space". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '18. London, United Kingdom: Association for Computing Machinery, 2018, pp. 2827–2836. ISBN: 9781450355520. DOI: 10.1145/3219819.3220052. URL: https://doi.org/10.1145/3219819.3220052.

[7] William L. Hamilton, Rex Ying, and Jure Leskovec. *Inductive Representation Learning on Large Graphs*. 2018. arXiv: 1706.02216 [cs.SI]. URL: https://arxiv.org/abs/1706.02216.

[8] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20.3 (1995), pp. 273–297.

[9] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[10] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), pp. 533–536.

[11] Jure Leskovec et al. "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters". In: *Internet Mathematics* 6.1 (2009), pp. 29–123.

[12] Jure Leskovec and Andrej Krevl. *SNAP Datasets: Stanford Large Network Dataset Collection*. https://snap.stanford.edu/data/roadNet-PA.html.

[13] Marco Annunziata. *Padua_Network_dataset_2025*. 2025. URL: https://github.com/Remdox/Padua_Network_dataset_2025.

[14] yzengal. *RoadNetwork-China-City*. 2021. URL: https://github.com/yzengal/RoadNetwork-China-City/blob/main/Hongkong.road-d.tar.gz.

[15] V. Mesina et al. *Italian Covid-19 Retweet Network (2020-2022) [Data set]*. 13th International Conference on Complex Networks and Their Applications (CNA2024), Istanbul, Turkey. Zenodo. 2024. URL: https://doi.org/10.5281/zenodo.13909011.

[16] Benedek Rozemberczki et al. "GEMSEC: Graph Embedding with Self Clustering". In: *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2019*. ACM. 2019, pp. 65–72.

[17] Jure Leskovec and Andrej Krevl. *SNAP Datasets: Stanford Large Network Dataset Collection*. https://snap.stanford.edu/data/gemsec-Deezer.html.

[18] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. *Multi-scale Attributed Node Embedding*. 2019. arXiv: 1909.13021 [cs.LG].

[19] Jure Leskovec and Andrej Krevl. *SNAP Datasets: Stanford Large Network Dataset Collection*. http://snap.stanford.edu/data/github-social.html.

[20] Damian Szklarczyk et al. "The STRING database in 2023: protein–protein association networks and functional enrichment analyses for any sequenced genome of interest". In: *Nucleic Acids Research* 51.D1 (2023), pp. D638–D646. DOI: 10.1093/nar/gkac1000. URL: https://string-db.org/cgi/download.

[21] Ryan A. Rossi and Nesreen K. Ahmed. "The Network Data Repository with Interactive Graph Analytics and Visualization". In: *AAAI*. 2015. URL: https://networkrepository.com.

[22] I. Bhargavi Kalyani, A. Rama Prasad Mathi, and Niladri Sett. "Evaluating link prediction: new perspectives and recommendations". In: *International Journal of Data Science and Analytics* 20.7 (2025), pp. 6855–6886. ISSN: 2364-4168. DOI: 10.1007/s41060-025-00858-0. URL: https://doi.org/10.1007/s41060-025-00858-0.

[23] palash1992. 2022. URL: https://github.com/eliorc/node2vec.

[24] tangjianpku et al. 2019. URL: https://github.com/tangjianpku/LINE.

[25] Lakshya-99. 2020. URL: https://github.com/Lakshya-99/Deep_Variational_Network_Embedding/tree/master.

[26] William L. Hamilton et al. 2018. URL: https://github.com/williamleif/graphsage-simple/.

[27] Università degli Studi di Padova. *Cluster "Blade"*. URL: https://docs.dei.unipd.it/en/CLUSTER/Overview.

Contribution of Authors and AI Usage

The contributions for this project are:

- ...

The following AI tools were used:

- Copilot was used to suggest code and report snippets.
- DeepL was used to correct the grammar and syntax of the text.