FACULTY OF ENGINEERING
AND ARCHITECTURE

# Accelerating Progress in Network Representation Learning: Systematic Evaluations and New Approaches

**Alexandru Cristian Mara**

Doctoral dissertation submitted to obtain the academic degree of
Doctor of Computer Science Engineering

**Supervisors**
Prof. Tijl De Bie, PhD - Prof. Jefrey Lijffijt, PhD
Department of Electronics and Information Systems
Faculty of Engineering and Architecture, Ghent University

January 2023

GHENT
UNIVERSITY

ISBN 978-94-6355-671-2
NUR 984
Wettelijk depot: D/2023/10.500/3

## Members of the Examination Board

### Chair

Prof. Em. Hendrik Van Landeghem, PhD, Ghent University

### Other members entitled to vote

Prof. Thomas Demeester, PhD, Ghent University
Prof. Stephan Günnemann, PhD, Technische Universität München, Germany
Prof. Raul Santos-Rodriguez, PhD, University of Bristol, United Kingdom
Prof. Willem Waegeman, PhD, Ghent University

### Supervisors

Prof. Tijl De Bie, PhD, Ghent University
Prof. Jefrey Lijffijt, PhD, Ghent University

# Acknowledgements

Firstly, I would like to express my most sincere gratitude to my supervisors Tijl De Bie and Jefrey Lijffijt, without whom I might have never obtained the title of 'Doctor of Computer Science Engineering'. As I have said many times before, I have the utmost respect for both of you as persons and as scientists. I admire your work ethic, your passion for research and your ability to guide students and help them discover their own academic paths, interests and goals. I honestly believe I could not have chosen a better place to pursue my doctoral studies than with you, here, at Ghent University. Tijl, you have managed to bring together a diverse group of people with different cultural and ethnic backgrounds yet somehow make it feel like a second family to me (and I'm sure to the rest of the group as well), so I thank you for that. Of course, the group hangouts, dinners, and those yearly 'Brainbuilding' escapes with pools and saunas might have helped a bit in this regard ;). Thank you also for your open-door policy and for always finding time for us, no matter how packed your calendar was (sorry I called you to sign a document while you were in a meeting with the minister of education...). We may not always show it, but we greatly appreciate being so high up on your priority list. Thank you also for the many discussions we've had over the past four years and the late nights working on papers right before a deadline. Jef, thank you for your advice and guidance during these years. Perhaps because of the more applied nature of my work, I've had many doubts about its scientific value. However, your reassurance and trust in me have made a significant positive impact. Thank you also for encouraging me to think outside the box. I aspire to one day be able to find connections between diverse areas of research and to come up with interesting questions and remarks with the same ease as you do now. And yes, I will also remember to use 'that' instead of 'which' in defining clauses from now on. Lastly, thank you both for sharing your contact networks and introducing me to so many interesting researchers and academics and for the opportunities to attend many conferences, summer schools, research stays and lectures.

I would like to continue by thanking all the examination committee members for the time and effort they have dedicated to assessing and helping me improve the present dissertation. I also want to thank professor Günnemann, in particular, for the opportunity to conduct a research stay and to meet his amazing group at TUM as part of my PhD. I also thank my co-authors and colleagues from the AIDA group, present and past (as in, those who have left the group). It has been an honor to work alongside you, to share ideas, and to enjoy relaxed chats over a drink and a board game. To Bo, thank you for your guidance, your advice and the

amazing Brainbuilding meals. To Florian, thanks for all the good times, chats, and laughs (except when you lied about being a ‘mason’ while playing ‘werewolves’, I haven’t forgotten that one). To Raphael and Sander, thank you for joining me in those long after-conference evenings. To Maarten, thank you for being such a good friend, a ruthless board-game adversary and for translating my summary, I owe you one. To everyone else, thank you and best of luck in your careers!

Not done yet, but we are almost there. I want to also express my gratitude to the Ghent University PhD Community, and in particular to Michael. Your events provided an invaluable opportunity for meeting other PhDs, discussing work-related issues, and staying sane during the pandemic and post-pandemic months (years?). Thanks to you, I’ve met so many smart and talented people, and also my current friends. Nina, Alyssa, thanks for all the inappropriate jokes and the unforgettable trip to a certain beverage factory. The climbing and post-climbing refreshments with you were also fun, I guess. Svieta, sorry I messed up your nationality when we first met, but that marked the start of an amazing friendship. I’m not mentioning Paddy, Bea, Laura or Mitchel, because I’m sure they won’t add me to their acknowledgments either (but they were pretty cool). To these and so many others I have not named, thank you! You made these four years very special (special good).

And one more paragraph, I promise. Vivian, thank you for your love and support during these years and for giving me the motivation to write this thesis in two weeks. A mis amigos de España, Alberto, Carlos, Guille, gracias por hacerme sentir como si nunca me hubiera ido cada vez que volvía a casa por navidad. Adina, gracias por esas charlas de hermanos y por estar siempre de mi parte. Sigues siendo mi hermana favorita. În ultimul dar şi cel mai important loc, mama, tata, vă mulţumesc pentru tot ce a-ţi făcut pentru mine. Nu aş fi ajuns unde sunt acum dacă nu era pentru toate sacrificiile pe care le-aţi făcut voi. Realizarea mea e, în mare parte, si a voastră. Vă mulţumesc încă o dată şi vă iubesc.

*Ghent, December 2022*
*Alexandru Cristian Mara*

# Table of Contents

**Acknowledgements** **i**

**Samenvatting** **xix**

**Summary** **xxiii**

**1 Introduction** **1**
1.1 Background . . . 1
1.2 Motivation . . . 3
1.3 Contributions . . . 4
1.4 Outline . . . 7
1.5 Publications . . . 8
1.5.1 Publications in international journals (listed in the Science Citation Index) . . . 8
1.5.2 Publications in international conferences (listed in the Science Citation Index) . . . 9
1.5.3 Publications in other international conferences . . . 9
References . . . 10

**2 Reproducibility in Network Representation Learning** **13**
2.1 Introduction . . . 14
2.2 Related work . . . 16
2.3 Methods . . . 17
2.3.1 Evaluation framework . . . 17
2.3.2 Other building blocks and baselines . . . 20
2.4 Experiments . . . 22
2.4.1 Experimental setups . . . 22
2.4.2 Experimental results . . . 24
2.5 Conclusions . . . 27
References . . . 29

**3 EvalNE: A Framework for Network Embedding Evaluation** **33**
3.1 Motivation and significance . . . 34
3.2 Related software . . . 35
3.3 Software description . . . 36
3.3.1 Software architecture . . . 36

3.3.2 Software functionalities . . . 37
3.4 Illustrative examples . . . 38
3.4.1 Command line example . . . 38
3.4.2 API example . . . 39
3.5 Impact . . . 39
3.6 Conclusions . . . 39
References . . . 41

**4 Performance Evaluation of Network Representations 45**
4.1 Introduction . . . 46
4.2 Methods . . . 49
4.2.1 Network embedding methods . . . 49
4.2.2 Baseline heuristics . . . 51
4.2.3 Implementations and hyperparameters . . . 53
4.3 Datasets . . . 53
4.4 Experimental setup . . . 54
4.4.1 Link prediction . . . 54
4.4.2 Network reconstruction . . . 55
4.4.3 Evaluation setup . . . 56
4.4.4 Reproducibility notes . . . 56
4.5 Experimental results . . . 57
4.5.1 Link prediction performance . . . 57
4.5.2 Hyperparameter tuning . . . 60
4.5.3 Embedding dimensionality . . . 62
4.5.4 Train-test split size . . . 62
4.5.5 Edge sampling . . . 62
4.5.6 Prediction pipeline . . . 65
4.5.7 Network reconstruction performance . . . 67
4.5.8 Method performance and network structure . . . 67
4.5.9 Limits to the scope of this evaluation . . . 69
4.5.10 External validation of the results . . . 70
4.6 Conclusions . . . 70
References . . . 72

**5 Robustness Evaluation of Network Representations 77**
5.1 Introduction . . . 78
5.2 Related work . . . 80
5.3 Methods . . . 80
5.3.1 Node embedding methods . . . 81
5.3.2 Network attacks . . . 82
5.4 Experiments . . . 83
5.4.1 Preliminaries and setup . . . 84
5.4.2 Data . . . 85
5.4.3 Experimental results . . . 86
5.5 Conclusions . . . 92

References . . . 93

**6 Learning Representations of Signed Networks 97**
6.1 Introduction . . . 98
6.2 Related work . . . 102
6.3 Method . . . 103
6.3.1 Concepts and notation . . . 103
6.3.2 A MaxEnt distribution for edge signs . . . 104
6.3.3 Conditional Signed Network Embedding . . . 107
6.4 Experimental setup . . . 109
6.4.1 Datasets . . . 109
6.4.2 Baseline methods . . . 110
6.4.3 Sign prediction evaluation . . . 110
6.4.4 Reproducibility notes . . . 112
6.5 Experimental results . . . 112
6.5.1 Sign prediction performance . . . 112
6.5.2 Hyperparameter sensitivity . . . 115
6.5.3 Convergence analysis . . . 117
6.5.4 A case study: Visualization . . . 117
6.6 Conclusions . . . 118
References . . . 120

**7 Conclusions and Future Work 125**
7.1 Conclusions . . . 125
7.2 Future Work . . . 127
7.2.1 Reproducibility . . . 127
7.2.2 Evaluation . . . 127
7.2.3 Representation learning models . . . 128

**A Appendix 131**
A.1 EvalNE: A Framework for Network Embedding Evaluation . . . 131
A.1.1 Configuration file example . . . 131
A.1.2 EvalNE code snippet . . . 133
A.2 Robustness Evaluation of Network Representations . . . 135
A.2.1 Additional dataset details . . . 135
A.2.2 Randomized attacks: additional results . . . 135
A.2.3 Other attacks: additional results . . . 135
A.3 Learning Representations of Signed Networks . . . 137
A.3.1 Additional experiments . . . 137

# List of Figures


2.1 Diagram showing the types of methods that can be evaluated using EvalNE. Gray blocks represent modules provided by the library and white blocks are the user-specified methods to be evaluated. The library can evaluate end-to-end predictors (including link prediction heuristics), node-pair embedding and node embedding methods. . . . . 17
2.2 Execution times of proposed and naive edge sampling strategies w.r.t. proportion of train edges. Solid lines correspond to the proposed method, while dashed lines correspond to the typically used (naive) sampling. . . . . 27
2.3 Execution times of proposed and naive edge sampling strategies on sub-graphs of BlogCatalog of increasing sizes. The results are reported for a 50-50 train-test split. . . . . 28

3.1 Block diagram of EvalNE showing the methods and tasks that can be evaluated. 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. . . . . 37

4.1 Improvement in AUC of tuning model hyperparameters. Only methods with tuned parameters are shown. . . . . 60
4.2 Best performing node-pair embedding operators on setup LP1. Only methods for which the node-pair operator was tuned are shown. 60
4.3 Execution times in seconds of setups LP1 and LP2. . . . . 61
4.4 Method performance (AUC) on setup LP1 for $d \in \{8, 32, 128\}$. . 61
4.5 Method performance as $-\log(1 - AUC)$ on setup LP1 for train-test edge splits 20-80, 50-50 and 80-20. A darker color indicates better performance. . . . . 63
4.6 Execution times of different train-test split algorithms. . . . . 64
4.7 Performance of the heuristics and NE methods compared for the spanning tree and timestamp-based sampling strategies. . . . . 64
4.8 End-to-end prediction pipelines compared to node-pair embeddings and binary classification. . . . . 65

4.9 Log odds for each coefficient of the logistic regression binary classifier. Results are presented for the GR-QC dataset but identical patterns are observed for other networks and experimental setups. 66
4.10 Network reconstruction performance in terms of precision@$Np$ shown only for the top-performing approaches. . . . . 69

5.1 Robustness to randomized attacks for different budget values. The x-axis shows budgets as a fraction of all edges in the graph. Negative values represent edge deletion and positives edge addition attacks. Figure 5.1a presents f1_micro scores for node classification on Citeseer. Figure 5.1b shows AUCs for network reconstruction on Facebook. In Figures 5.1c and 5.1d we show average node classification f1_micro scores for different fractions of labeled nodes $N_{tr}$ on Citeseer and Cora, respectively. Shaded areas denote 95% confidence intervals. . . . . 87
5.2 Comparison of adversarial edge deletion and addition attacks for $b = 0.2$. Figures 5.2a and 5.2b show deletion and addition attacks on node classification for Citeseer. Colors indicate the fraction of train nodes $N_{tr}$. Figures 5.2c and 5.2d show similar results for network reconstruction on both Facebook and PolBlogs networks combined (colors indicate the network). . . . . 89
5.3 Comparison of edge addition, rewiring and deletion attacks for both downstream tasks. The leftmost and center figures present f1_micro scores for random and node label based attacks on node classification. The rightmost figure shows AUC results for random attacks on network reconstruction. . . . . 90
5.4 Correctly and incorrectly classified nodes for the homophilic IIP network (top row) and the heterophilic StudentDB network (bottom row) for varying attack budgets. . . . . 91

6.1 Execution times of all methods for sign prediction on each evaluated network. Gray boxes indicate the total runtime per method on all datasets. . . . . 114
6.2 Relative execution times of all methods compared to the fastest approach on each dataset (lower is better). . . . . 115
6.3 Average sign prediction AUC over all evaluated networks for varying sizes of $E_{train}$ - $E_{test}$. A black line over each bar denotes the 95% confidence interval for the estimate of the average performance. 116
6.4 Sign prediction AUC scores of CSNE(pol.+tri.) for (a) different embedding dimensions $d$ and (b) different values of $\sigma_2$. . . . . 117
6.5 Convergence of (a) the MaxEnt prior and (b) CSNE. . . . . 118
6.6 Plot showing the CSNE embeddings of the Harry Potter network. Enemity relations between characters are highlighted in red, while friendships are highlighted in blue. A subset of protagonists and antagonists are shown in blue and red, respectively. . . . . 119

A.1 EvalNE configuration file example. . . . . 132
A.2 Code snippet showing the use of EvalNE from python code. . . . 134
A.3 Node classification f1_micro scores on Cora (a), and network reconstruction AUC scores on PolBlogs (b). Negative attack budgets indicate edge deletion. . . . . 135
A.4 Node classification performance for the Citeseer network (a), and the Cora network (b). Y-axis indicate f1_macro scores. Negative attack budgets indicate edge deletion. . . . . 136
A.5 Network reconstruction performance for the Facebook network (a), and the PolBlogs network (b). Y-axis indicate average precision scores. Negative attack budgets indicate edge deletion. . . 136
A.6 Comparison of edge addition, rewiring and deletion attacks for both downstream tasks. The leftmost and center figures present f1_macro scores for random and node label based attacks on node classification. The rightmost figure shows average precision results for random attacks on network reconstruction. . . . . 136

# List of Tables


2.1 Datasets used for evaluation. . . . . 24
2.2 Reported AUC scores in the node2vec paper and difference to our reproduced values. . . . . 25
2.3 Reported precision@k scores in the SDNE paper and difference to our reproduced values. . . . . 25
2.4 Reported AUC scores in the PRUNE paper and difference to our reproduced values. . . . . 26
2.5 Reported AUC scores in the CNE paper and difference to our reproduced values . . . . . 26
2.6 Absolute difference in recall between results obtained using the naive and proposed edge sampling for different networks and link prediction heuristics. . . . . 27

4.1 Implementation used and hyperparameters tuned for each method. Except for AROPE, CNE, GAE, VGAE, GIN, GatedGCN and the heuristics, the edge embedding operator is also tuned as a hyperparameter. . . . . 52
4.2 Summary of dataset statistics where $2|\mathbf{E}|/|\mathbf{V}|$ denotes the average node degrees of the undirected networks. . . . . 54
4.3 AUC scores and standard deviations over 3 experiment repetitions for setup LP1 where hyperparameters are tuned and $d = 128$ for all methods except GIN, GatedGCN and 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 gray background. . . . . 58
4.4 Network reconstruction performance shown as precision@100 where method hyperparameter are tuned and $d = 128$ for all methods except CNE, GIN and GatedGCN where $d = 8$. The average precision@100 over all networks and average rank for each method are also presented. . . . . 68

5.1 Poison attacks evaluated and their types: (D) deterministic, (ND) non-deterministic. . . . . 82

5.2 Main statistics of the networks used for evaluation. The average degree is indicated by $\langle k \rangle$, the assortativity coefficient by $r$, and 'Viz' represents the network visualization task in Section 5.4.3.4. . 86

6.1 Main statistics of the networks used for sign prediction evaluation. 110
6.2 Sign prediction AUC for all networks. Best performing method per dataset is highlighted in bold. . . . . 113

A.1 Average sign prediction AUC and standard deviations over 3 repetitions of the experiment using 50/50 train/test edge splits. Best performing method per dataset is highlighted in bold. Note that $0.000$ in the table means $< 0.0005$. . . . . 138

# List of Acronyms

## A

| | |
|---|---|
| **AA** | Adamic-Adar Index |
| **AI** | Artificial Intelligence |
| **API** | Application Programming Interface |
| **AUC** | Area Under the ROC Curve |
| **AUROC** | Area Under the ROC Curve |
| **AUC-ROC** | Area Under the ROC Curve |

## B

| | |
|---|---|
| **BFS** | Breadth First Search |

## C

| | |
|---|---|
| **CE** | Connect Externally |
| **CN** | Common Neighbors |
| **CNE** | Conditional Network Embedding |
| **CSNE** | Conditional Signed Network Embedding |

## D

| | |
|---|---|
| **DA** | Degree Assortativity |

**DAG** Direct Acyclic Graph
**DD** Degree Disassortativity
**DFT** Depth First Tree
**DI** Disconnect Internally
**DICE** Disconnect Internally Connect Externally
**DRU** Distributed Radomization Unit
**DT** Decision Tree

# E

**ERC** European Research Council
**EU** European Union

# F

**FN** False Negatives
**FNR** False Negative Rate
**FP** False Positives
**FPR** False Positive Rate
**FWO** Fonds Wetenschappelijk Onderzoek

# G

**GAE** Graph Auto-Encoder
**GCN** Graph Convolutional Network
**GEM** Graph Embedding Methods
**GF** Graph Factorization
**GIN** Graph Isomorphism Network
**GNN** Graph Neural Network
**GUI** Graphical User Interface

# I

**IIP** Internet Industry Partnership

# J

**JC** Jaccard Coefficient

# L

**LE** Laplacian Eigenmaps
**LLE** Locally Linear Embeddings
**LP** Link Prediction
**LR** Logistic Regression
**LRCV** Logistic Regression with Cross Validation

# M

**MAP** Maximum a Posteriori
**MaxEnt** Maximum Entropy
**MIT** Massachusetts Institute of Technology
**ML** Machine Learning
**MLE** Maximum Likelihood Estimation

# N

**NC** Node Classification

**ND** Non-Deterministic
**NE** Network Embeddings
**NR** Network Reconstruction
**NRL** Network Representation Learning

## O

**OGB** Open Graph Benchmark
**OOD** Out of Distribution

## P

**PA** Preferential Attachment
**PPI** Protein-Protein Interaction

## R

**RAM** Random-Access Memory
**RAI** Resource Allocation Index
**ROC** Receiver Operating Characteristic

## S

**SDNE** Structural Deep Network Embedding
**SNE** Signed Network Embedding
**SP** Sign Prediction
**ST** Spanning Tree
**SVD** Singular Value Decomposition

## T

**TN** True Negatives
**TNR** False Negative Rate
**TP** True Positives
**TPR** True Positive Rate

## V

**VGAE** Variational Graph Auto-Encoder

## W

**WYS** Watch Your Step

# Samenvatting

**Achtergrond** Netwerken zijn datastructuren die natuurlijkerwijs in veel wetenschappelijke en industriële toepassingen voorkomen. Netwerken stellen entiteiten (bv. gebruikers, proteïnen) voor als knopen en hun relaties (bv. vriendschap, chemische interactie) als verbindingen. Deze structuren kunnen ook aanvullende informatie over entiteiten en relaties bevatten, die gecodeerd wordt als attributen van de knopen, respectievelijk de verbindingen. Deze universaliteit en grote uitdrukkingskracht zijn de basis van het belang en het uitgebreide gebruik van netwerken. De meeste traditionele methoden voor machinaal leren ondersteunen echter geen gestructureerde netwerkdata. Om deze problematiek aan te pakken zijn aanzienlijke onderzoeksinspanningen geleverd om geschiktere representaties te leren.

Het vakgebied van het leren van netwerkrepresentaties, genaamd 'netwerkinbedding', is gewijd aan het bestuderen van zulke representaties. Het doel is netwerkknopen te coderen als vectoren, meestal in een continue ruimte (bv. de Euclidische ruimte), waarbij de belangrijkste eigenschappen van de oorspronkelijke graaf behouden blijven. Een bijzonder voorbeeld is het behoud van similariteit in de graaf als nabijheid in de inbeddingsruimte. Deze inbeddingen van knooppunten kunnen impliciet worden geleerd, als bijproduct van het oplossen van een vervolgtaak (bv. in graaf-neurale netwerken, genoemd GNNs), of expliciet, waarbij de inbedding zelf het einddoel is (bv. random walk-gebaseerde methoden). De eerste worden over het algemeen beschouwd als semi-gesuperviseerde benaderingen omdat aanvullende informatie over de vervolgtaak (meestal knooplabels) wordt gebruikt om het leerproces van de inbeddings te sturen. Het tweede type methoden wordt aangeduid als ongesuperviseerde methoden, waarbij geen expliciete labels worden gebruikt in het leerproces, buiten de netwerkstructuur zelf. In dit werk richten wij ons voornamelijk op de laatste soort om twee belangrijke redenen (hoewel sommige resultaten voor beide soorten gelden). Ten eerste bieden deze methoden universele inbeddingen die geschikt zijn voor een verscheidenheid van taken, van het visualiseren van netwerken in twee- of driedimensionale ruimten tot het detecteren van ontbrekende informatie of het voorspellen van attribuut- en labelwaarden. Ten tweede is de evaluatie van deze methoden minder goed gedefinieerd, aangezien zij niet gekoppeld zijn aan specifieke vervolgtaken. Ongesuperviseerde benaderingen kunnen verder worden ingedeeld op basis van de gebruikte interpretaties over de similariteit van knopen, de gebruikte afstandsmetriek in de inbeddingsruimte en de wiskundige principes die als hefboom dienen om de representaties te verkrijgen (bv. random walks, matrixfactorisatie, diepe neurale netwerken).

De kwaliteit van de geleerde representaties kan visueel worden beoordeeld of, in een algemener geval, in termen van prestatie op vervolgtaken. Van betere representaties wordt dus verwacht dat zij resulteren in hogere prestaties op vervolgtaken. De complexiteit van de netwerkdata in combinatie met de verscheidenheid aan inbeddingsbenaderingen en het gebruik van proxy-vervolgtaken om de kwaliteit van inbeddings te bepalen, heeft tot verschillende uitdagingen geleid. Deze uitdagingen hebben op hun beurt de vooruitgang in dit vakgebied vertraagd.

**Uitdagingen** Ondanks aanzienlijke collectieve inspanningen voor betere onderzoekspraktijken en het uitbrengen van referentie-implementaties voor nieuwe methoden, blijven er verschillende belangrijke uitdagingen bestaan. Ten eerste is er een gebrek aan gestandaardiseerde procedures voor de evaluatie van netwerkinbedding. Dit heeft ertoe geleid dat auteurs verschillende benchmarkdatasets, voorbewerking- en samplingtechnieken, voorspellingsmodellen en trainingsprocedures, evaluatiemetrieken, enz. hebben voorgesteld. Deze en andere ontwerpkeuzes, die in de praktijk vaak niet worden besproken in onderzoeksartikelen, hebben een reproduceerbaarheidscrisis veroorzaakt en het vergelijken van verschillende methoden bemoeilijkt. Bovendien is er een gebrek aan empirische vergelijkingsstudies, die licht zouden kunnen werpen op de stand van het onderzoek naar het leren van netwerkrepresentaties en het vaststellen van gestandaardiseerde evaluatieprotocollen. Er is enig werk verricht op het gebied van GNN's, terwijl ongesuperviseerde inbeddingsbenaderingen grotendeels over het hoofd zijn gezien. Ook de robuustheid van niet gesuperviseerde modellen tegen willekeurige datafouten en vijandige aanvallen wordt slecht begrepen. Dit is bijzonder problematisch omdat sommige van hun belangrijkste toepassingen plaatsvinden op gebieden waar vijanden veel voorkomen. Samengevat hebben deze uitdagingen geleid tot moeilijk te kwantificeren incrementele vooruitgang op dit vakgebied en de onvoldoende ontwikkeling van andere representatieparadigma's ten gunste van traditionele benaderingen zoals random walks en matrixfactorisatie.

**Bijdragen** In dit proefschrift verkennen we eerst de belangrijkste oorzaken en presenteren we bewijs van een voortdurende reproduceerbaarheidscrisis op het gebied van het leren van netwerkrepresentaties. In het bijzonder richten wij ons op uitdagingen in verband met de evaluatie van netwerkrepresentaties op linkvoorspelling ('link' is jargon voor een verbinding tussen knopen). Wij laten zien hoe het gebrek aan gestandaardiseerde evaluaties en ongedocumenteerde ontwerpkeuzes, naast andere factoren, kunnen leiden tot slechte reproduceerbaarheid. Onze experimentele evaluatie, waarbij we proberen de resultaten van verschillende papers in het veld te repliceren met verschillende maten van succes, ondersteunt onze beweringen. Om deze situatie te verlichten introduceren wij EvalNE, een raamwerk voor reproduceerbare evaluatie van linkvoorspellingen.

Voortbouwend op onze codebasis voor reproduceerbaarheid, breiden we de functies ervan uit en presenteren we EvalNE als een uitgebreide evaluatietoolbox. Het raamwerk omvat alle belangrijke vervolgtaken die gewoonlijk worden

gebruikt om de kwaliteit van de representatie te evalueren. Dit omvat linkvoorspelling, tekenvoorspelling, multi-class classificatie van knopen en netwerkreconstructie. EvalNE biedt automatisering en abstractie voor taken variërend van datasampling tot hyperparametertuning en modelvalidatie. Ons raamwerk zorgt voor consistentie en reproduceerbaarheid via deelbare configuratiebestanden waarin alle experimentele ontwerpkeuzes worden gedetailleerd. De optionele grafische gebruikersinterface is ontworpen om gebruikers buiten de academische wereld aan te trekken. Hiermee kunnen ingenieurs en onderzoekers op een intuïtieve en interactieve manier evaluaties ontwerpen en uitvoeren.

Verder gebruiken we het voorgestelde EvalNE-raamwerk om een uitgebreide empirische evaluatie uit te voeren van 23 methoden voor het ongesuperviseerd leren van representaties. Deze benaderingen omvatten alle belangrijke paradigma's, namelijk Skip-Gram, matrixfactorisatie, neurale netwerken en probabilistische modellen. Wij evalueren de prestaties op linkvoorspelling en netwerkreconstructie en beoordelen het effect van verschillende ontwerpkeuzes. Deze omvatten de inbeddingsdimensionaliteit, de train- en teststrategie voor het samplen van randen en de hyperparameters van de methode, naast vele andere. Het doel van dit werk is licht te werpen op de state-of-the-art op dit gebied en een duidelijke standaard te introduceren voor nieuwe benaderingen. Onze experimentele resultaten tonen slechts geringe vooruitgang, waarbij veel modellen niet beter presteren dan eenvoudige heuristieken. Bovendien vonden wij geen statistisch significante verschillen tussen de prestaties van de meeste benaderingen. Tenslotte wijzen onze resultaten ook op slechts geringe prestatiewinst door tijdrovende hyperparameterafstemming.

Ons werk aan de prestatie-evaluatie heeft ook correlaties aan het licht gebracht tussen de voor de linkvoorspelling gebruikte datasamplingstrategieën en de inbeddingskwaliteit. Om verder te begrijpen hoe veranderingen in de invoerdata de geleerde representaties beïnvloeden, hebben we een systematische analyse uitgevoerd van de robuustheid van de netwerkinbedding. In dit werk hebben we willekeurige veranderingen en vijandige aanvallen op de invoergrafen uitgevoerd en de variaties in de prestatie op latere taken gevolgd. We evalueerden in totaal 14 aanvallen die specifiek gericht waren op de netwerkverbindingen. De voorgestelde vijandige aanvallen zijn gebaseerd op topologische eigenschappen van de grafiek, zoals graad, assortativiteit en verschillende centraliteitsmetingen. Onze evaluatie heeft ook betrekking op zowel de classificatie van knopen met meerdere labels als netwerkreconstructie. De resultaten wijzen op grotere variaties in de classificatie van knopen dan in de reconstructie van netwerken over de hele linie. Wij stellen ook vast dat de meest effectieve aanvallen variëren op basis van de aanvalssterkte. Niettemin blijken heuristische aanvallen op basis van knoopgraden verrassend effectief.

Ten slotte stellen wij CSNE voor, een nieuwe methode voor ondertekende netwerkinbedding en tekenvoorspelling. Deze aanpak is gebaseerd op Conditional Network Embedding (CNE), een niet gesuperviseerd model dat uitstekende prestaties leverde in onze benchmarkstudie. Met dit werk laten we zien hoe technieken die zijn ontwikkeld voor specifieke netwerkstructuren met succes kunnen worden aangepast aan andere domeinen. CSNE is een probabilistisch model dat een prior

combineert die structurele informatie in het netwerk vastlegt met een inbedding die gericht is op andere fijnkorrelige details. De prior modelleert tegelijkertijd: knooppolariteit (neiging van knooppunten om meer positieve/negatieve verbindingen in het netwerk te maken) en evenwichtige driehoektellingen (een maat voor de mate waarin de structurele evenwichtstheorie geldt in het netwerk). Een inbedding wordt dan onafhankelijk van de voorkennis geleerd, wat resulteert in minder overfitting en een nauwkeuriger model. Experimentele resultaten bevestigen dat CSNE beter presteert dan de state-of-the-art. Bovendien kan de CSNE-prior ook rechtstreeks worden gebruikt voor tekenvoorspelling. Individueel behaalt de prior uitstekende resultaten met uitvoeringstijden die tot 50x lager liggen dan de snelste baseline.

**Vooruitzichten** De resultaten die in dit proefschrift worden gepresenteerd, hebben het potentieel om de vooruitgang in het leren van netwerkrepresentaties aanzienlijk te versnellen. Doorheen ons werk hebben we het belang benadrukt van goede evaluatieprocedures en reproduceerbaarheid van de resultaten. Het voorgestelde EvalNE-kader vormt een eerste stap naar geautomatiseerde, reproduceerbare en gestandaardiseerde evaluaties. In dit proefschrift geven we ook een eerste overzicht van de prestaties en robuustheid van methoden voor het leren van representaties. Dit werk vormt een duidelijke basislijn voor nieuwe benaderingen. Later kunnen kaders zoals EvalNE worden gebruikt om de vooruitgang in het veld voortdurend te volgen. Met instrumenten zoals die welke in dit proefschrift worden voorgesteld, kunnen de onderzoeksinspanningen worden verlegd van evaluatiehindernissen naar nieuwe aanpakken en paradigma's voor het leren van representaties.

# Summary

**Background** Networks are data structures that arise naturally in many scientific and industrial applications. They represent entities (e.g., users, proteins) as nodes and their relations or interactions (e.g., friendship, chemical interaction) as edges. These structures also support additional information regarding entities and interactions encoded as node or edge attributes, respectively. This universality and high expressive power are the cornerstones of network importance and widespread use. Most traditional machine learning methods, however, do not efficiently support network structured data. To overcome this challenge significant research efforts have been directed towards learning more suitable network representations.

The field of network representation learning or network embedding is devoted to studying such representations. The aim is to encode network nodes as vectors, typically in a continuous space (e.g., Euclidean space), where key properties of the original graph can be preserved. One particular example is preserving connectivity in the graph as closeness in the embedding space. These node embeddings can be learned implicitly as the byproduct of solving a downstream task (e.g., in GNNs) or explicitly, where the embedding itself is the final goal (e.g., random walk-based methods). The former are generally regarded as semi-supervised approaches since additional information pertaining to the downstream task (commonly node labels) is used to guide the embedding learning process. The second type of methods are dubbed unsupervised and no explicit labels beyond the network structure itself are used in the learning process. In this work, we focus mainly on the latter for two main reasons (although some results do apply to both types). Firstly, these methods provide universal embeddings suitable for a variety of tasks, from visualizing networks in two or three-dimensional spaces to detecting missing information or predicting attribute and label values. Secondly, the evaluation of these methods is less well-defined as they are not attached to particular downstream tasks. Unsupervised approaches can be further classified based on the notions of node similarity used, the distance metrics in the embedding space adopted, and the mathematical principles leveraged to obtain the representations (e.g., random walks, matrix factorization, deep neural networks).

The quality of the learned representations can be assessed visually or in a more general case in terms of downstream task performance. Better representations are, thus, expected to result in higher downstream performance. The complexity of network data together with the variety of embedding approaches and the use of proxy downstream tasks to determine embedding quality has given rise to several challenges. These challenges have, in turn, slowed down progress in the field.

**Challenges** Despite significant community efforts towards better research practices and the release of reference implementations for new methods, several important challenges remain. Firstly, there is a lack of standardized procedures for network embedding evaluation. This has led to authors proposing different benchmark datasets, data preprocessing and sampling techniques, downstream prediction models and training procedures, evaluation metrics, etc. These and other design choices, often not discussed in research publications, have sparked a reproducibility crisis and made comparing different methods challenging. Additionally, there is a lack of empirical studies shedding light on the state of research in network representation learning and establishing standardized evaluation protocols. Some work has been done in the area of GNNs while unsupervised approaches have been largely overlooked. Similarly, the robustness to random failures and adversarial attacks of unsupervised models is also poorly understood. This is particularly problematic as some of their applications are in areas where adversaries are common. In summary, these challenges have led to hard to quantify incremental progress in the field and the underdevelopment of other representation paradigms in favor of traditional approaches such as random walks and matrix factorization.

**Contributions** In this thesis, we first explore the main causes and present evidence of an ongoing reproducibility crisis in the field of network representation learning. In particular, we focus on challenges related to the evaluation of network representations on downstream link prediction. We show how the lack of standardized evaluations and undocumented design choices, among other factors, can lead to poor reproducibility. Our experimental evaluation, where we attempt to replicate the results of several papers in the field with different levels of success, supports our claims. To alleviate this situation we introduce EvalNE as a framework for reproducible link prediction evaluations.

Building upon our reproducibility code base, we extend its capabilities and present EvalNE as a comprehensive evaluation toolbox. The framework covers all major downstream tasks commonly used to evaluate representation quality. These include link prediction, sign prediction, node classification, and network reconstruction. EvalNE provides automation and abstraction for tasks ranging from data sampling to hyperparameter tuning and model validation. Our framework ensures consistency and reproducibility via shareable configuration files detailing all experimental design choices. The optional graphical user interface is designed to expand our user base beyond the academic circles. It allows practitioners to design and run evaluations in an intuitive and interactive manner.

Further, we leverage the proposed EvalNE framework to conduct an extensive empirical evaluation of 23 unsupervised representation learning methods. These approaches cover all main paradigms, namely Skip-Gram, matrix factorization, neural networks, and probabilistic models. We evaluate performance on downstream link prediction and network reconstruction and assess the impact of various design choices. These include embedding dimensionality, train and test edge sampling strategy, and method hyperparameters, among many others. The goal of

this work is to shed light on the state-of-the-art in the field and establish a clear baseline for new approaches. Our experimental results show only minor progress with many models performing on par with simple heuristics. Furthermore, we did not find statistically significant differences between the performances of most approaches. Lastly, our results also indicate only minor gains in performance through time-consuming hyperparameter tuning operations.

Our work on performance evaluation also revealed correlations between the data sampling strategies used for link prediction and the embedding quality. To further understand how changes to the input data affect the representations learned we conducted a systematic analysis of network embedding robustness. In this work, we performed random changes and adversarial attacks on the input graphs and monitored the variations in downstream task performance. We evaluated a total of 14 attacks targeting specifically the network edges. The proposed adversarial attacks are based on topological graph properties such as degree, assortativity, and different centrality measures. Our evaluation also covers both node multi-class classification and network reconstruction tasks. Results indicate higher variations in node classification than network reconstruction performance across the board. We also find that the most effective attacks vary based on the attack strength. Nevertheless, heuristic attacks based on node degrees prove surprisingly effective.

Finally, we propose CSNE, a new signed network embedding and sign prediction model. CSNE is based on Conditional Network Embedding, an unsupervised model exhibiting excellent performance in our benchmark study. This work, thus, constitutes an example of how experimental studies can help identify new high-potential models and paradigms. CSNE is a probabilistic model that combines a prior capturing structural information in the network and an embedding focused on other fine-grained detail. The prior models simultaneously: node polarity (tendency of nodes to make more positive/negative connections in the network) and balanced triangle counts (a measure of the extent to which structural balance theory holds in the network). An embedding is then learned independently of the prior resulting in less overfitting and a more accurate model. Experimental results confirm that CSNE outperforms the state-of-the-art. Additionally, the CSNE prior can also be used directly for sign prediction. Independently, the prior achieves excellent results with execution times up to $50\times$ lower than the fastest baseline.

**Outlook** Results presented in this dissertation have the potential to significantly accelerate progress in network representation learning. Throughout our work we have identified important issues related to evaluation and reproducibility of results. The proposed EvalNE framework constitutes a first step towards automated, reproducible and standardized evaluations. In this thesis we also provide large-scale analysis of the performance and robustness of representation learning methods and set clear baselines for new approaches. Later, frameworks such as EvalNE can be used to continuously monitor progress in the field. With more tools similar to the ones proposed in this thesis research efforts can shift away from evaluation hurdles and towards novel representation learning approaches and paradigms.

# 1 Introduction

This chapter situates the conducted research work, summarizes the main contributions and outlines the structure of this dissertation. It also provides an overview of the publications authored during this research period.

## 1.1 Background

The fields of machine learning and data mining extract knowledge from a variety of data sources including tabular, image, sound and text. One particularly interesting data type is network structured data, which arises naturally in many scientific and industrial applications. These applications include logistical networks, social networks, biological networks, telecommunication networks, and span fields from social sciences to computer science. Network data encodes entities as nodes and their interactions as edges. For instance, one can represent cities as nodes and the roads connecting them as edges. Another example is representing users as nodes and their interactions on a particular social network as edges. Additional information can be also represented through node attributes (e.g., city population, user age) and edge weights (e.g., road length, number of user interactions). The importance of networks or graphs[1] stems primarily from their universality and representation power. Networks, unlike metric spaces, are not bounded by properties such as positivity, symmetry or triangle inequality. For instance, the distance be-

[1] In this thesis, as it is often done in the field, we will use the terms network/graph, node/vertex and edge/link interchangeably.

tween two points in a metric space must be the same regardless of direction. In networks, the number of communications from user A to user B can be different than those from B to A. Despite their qualities, however, information cannot be extracted efficiently from networks using traditional machine learning models. Networks are commonly encoded as lists of connected entities or matrices of interaction, which are not optimal inputs for downstream classification, regression or clustering. Thus, new and efficient representations of network data are required.

The field of network representation learning or network embedding aims specifically at bridging this gap. It studies network representations as low-dimensional vectors in an embedding space. This space is generally Euclidean [1–3], but others such as hyperbolic spaces have also been proposed [4]. The main idea is to encode nodes as vectors that preserve specific network properties. One example is encoding similar nodes in the graph into close-by vectors in the embedding space. Traditionally, these approaches relied on the factorization of matrices representing the network structure such as the adjacency matrix or the graph Laplacian [5, 6]. Later, NLP-inspired techniques proposed capturing node similarities in the graph through random walks and generating embeddings reflecting node co-occurrences in said walks using Skip-Gram [1, 7] (see further details in Chapter 4). Deep neural networks have also been proposed for learning embeddings [8, 9], and more recently graph neural networks (GNNs) are reshaping the state-of-the-art [10, 11]. With the exception of GNNs, the remaining techniques learn embeddings in an unsupervised fashion by relying exclusively on the network structure. Although the existence and absence of edges can be considered as some form of supervision, these methods are generally regarded as unsupervised. On the other hand, GNNs use, in addition to the network structure, labels from downstream tasks (generally node classification) to guide the embedding learning process. As such, GNNs are considered to be semi-supervised methods [9]. In this thesis, we will focus on unsupervised representation learning techniques for two main reasons. Firstly, they are extensively used in practice and provide general-purpose embeddings suitable for data visualization and a variety of downstream tasks. Secondly, the evaluation of unsupervised representation learning approaches is more complex and presents many more design choices, as we discuss below and in Section 1.3. We also note that despite a focus on unsupervised approaches some of our work applies to GNNs as well. Specifically, our evaluation framework introduced in Chapter 3 can also be used to assess the performance of these types of approaches.

The quality of network representations can be assessed visually or through downstream task performance. Visual evaluations are limited to small-sized networks and low embedding dimensionalities [12]. Evaluations via proxy tasks, on the other hand, are more generally applicable. In these evaluations, network embeddings are first learned for a given graph and then used as inputs for a subsequent classification or regression task. Commonly evaluated downstream tasks include

node classification, sign and link prediction, network reconstruction, graph classification and attribute forecasting. In this dissertation, we mainly focus on the first four tasks, which we briefly introduce next. Formal task definitions are provided in the chapters where each task is evaluated. In node classification one aims to predict missing node labels given all node representations and a subset of labeled nodes. In sign prediction, the goal is to determine if given pairs of linked nodes should be positively or negatively connected. Similarly, in link prediction and network reconstruction we use vector representations to predict if given pairs of nodes should be linked or not and to recover the link structure of the input network, respectively. In all the above mentioned cases, better representations of the networks are expected to result in higher downstream performances. Unfortunately, the complexity of the evaluation pipelines has given rise to a number of challenges which have, in turn, slowed down progress in the field [13].

## 1.2 Motivation

To design better network representation learning approaches we must first be able to accurately determine the state-of-the-art in the field. This, in turn requires a certain level of consistency in the evaluation processes and reproducibility of results [14, 15]. These desirable properties are directly hindered by the complexities associated with the evaluation of representations via proxy downstream tasks. Currently, authors face an array of evaluation design choices ranging from benchmark networks to data preprocessing, sampling of train and test edges, hyperparameter tuning, downstream classifiers, performance metrics and more (these choices are discussed extensively in Section 4.1). As a result, evaluations are generally not comparable across papers. Moreover, some design choices are rarely discussed in practice compromising the reproducibility of results. In addition to this, the field also suffers from a lack of empirical studies proposing standardized evaluation setups and comparing existing approaches. Some work has been done in the area of GNNs [16], but unsupervised approaches have been largely overlooked. This uncertainty in the state-of-the-art also results in promising new paradigms receiving far less attention than other well established models. Lastly, the robustness of embedding methods to changes in the input data is poorly understood. Nevertheless, these approaches are being consistently applied to fields where adversaries are common, such as the web [17, 18], and critical applications e.g., job recommendations [19], medical data analysis [20], etc.

In summary, to accelerate progress in network representation learning we need to ensure reproducibility of results, simplify and automate evaluations, and provide empirical studies that survey and summarize the state of research.

## 1.3 Contributions

In this dissertation we aim to accelerate progress in network representation learning through a series of contributions. First, we explore the main challenges related to the evaluation of representation learning methods and experimentally uncover an ongoing reproducibility crisis. Second, we provide a software framework to automate evaluation and ensure reproducibility. Third, we perform an extensive empirical evaluation of the state-of-the-art in the field. Fourth, we examine the robustness of representation learning approaches under random and adversarial attacks. Fifth, we propose a new approach for signed network embedding and sign prediction. Next, we describe each of these contributions in more detail, but first we provide some necessary notation.

In what follow we will denote an undirected graph as $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with vertex set $\mathbf{V} = \{1, \ldots, N\}$ and edge set $\mathbf{E} \subseteq \binom{\mathbf{V}}{2}$. Edges or connected node-pairs are represented as unordered pairs $\{i, j\} \in \mathbf{E}$. Non-edges or disconnected pairs are represented as $\{i, j\} \in \mathbf{D}$. We use $\mathbf{E}_{tr}$ and $\mathbf{D}_{tr}$ to refer to the sets of train edges and non-edges and $\mathbf{E}_{te}$ and $\mathbf{D}_{te}$ to refer to the test node-pair sets. We use $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$ with $\mathbf{X} \in \mathbb{R}^{\mathrm{N} \times \mathrm{d}}$ to denote a $d$-dimensional node embedding ($d$ is a user-defined parameter that varies between works). Node-pair embeddings are represented as $\mathbf{Z}_{tr} \in \mathbb{R}^{\mathrm{M_{tr}} \times \mathrm{d}}$, and $\mathbf{Z}_{te} \in \mathbb{R}^{\mathrm{M_{te}} \times \mathrm{d}}$, where $M_{tr} = |\mathbf{E}_{tr}| + |\mathbf{D}_{tr}|$ and $M_{te} = |\mathbf{E}_{te}| + |\mathbf{D}_{te}|$, respectively.

**Reproducibility in Network Representation Learning** We begin our contributions with an analysis of reproducibility of research in the field of network representation learning. Our evaluation focuses specifically on downstream link prediction as it requires one of the most complex evaluation setups. In particular, this task generally requires the following steps. Given a graph $\mathbf{G}$ a holdout set of edges, $\mathbf{E}_{te}$, must first be sampled. Node embeddings $\mathbf{X}$ are then learned from the remaining graph $\mathbf{G_{tr}} = (\mathbf{V}, \mathbf{E}_{tr})$. Non-edges are sampled to form the sets $\mathbf{D}_{tr}$ and $\mathbf{D}_{te}$. A binary operator is then generally applied on the node embeddings to obtain node-pair representations i.e. $\mathbf{z}_{\{i,j\}} = \mathbf{x}_i \circ \mathbf{x}_j$. The above operation is performed on the sets of train and test edges and non-edges to obtain $\mathbf{Z}_{tr}$ and $\mathbf{Z}_{te}$. Using $\mathbf{Z}_{tr}$ and edge/non-edge as labels, a binary classifier is trained.[2] Finally, link prediction performance is evaluated using $\mathbf{Z}_{te}$. This process involves various designs choices such as edges sampling techniques, binary operators, binary classifiers, etc. These choices are not always documented leading to poor reproducibility. In this work we propose EvalNE as a framework for reproducible link prediction evaluations and use it to replicate the results of four papers. Our evaluation shows that with access to all the abovementioned design choices, EvalNE can

[2] We note that an alternative pipeline to node-pair embeddings plus binary classification exists. One could directly compute node similarities via dot product of the corresponding node embeddings. A detailed discussion and comparison of these two pipelines is provided in Section 4.5.6.

accurately replicate results. On the other hand, relying exclusively on the information reported in manuscripts and the authors' reference implementations, results can differ significantly.

**EvalNE: A Framework for Network Embedding Evaluation** Building on our reproducibility code base, we extend EvalNE to become a comprehensive evaluation framework covering all major downstream tasks. These include link prediction, sign prediction, node multi-class classification and network reconstruction. EvalNE automates evaluation for any number of embedding methods and networks. Performance can be assessed independently on the implementation language of the models and computed from node embeddings $\mathbf{X}$, node-pair embeddings $\mathbf{Z}$, or final model predictions (this last pipeline also allows for the evaluation of GNNs). The framework also implements several sampling and negative sampling techniques, four different binary operators, supports any downstream binary and multi-class classifier, includes hyperparameter tuning and much more. Complete experimental setups are described in configuration files used to run evaluations and can be shared to ensure reproducibility. An additional GUI is provided to simplify the creation and management of these configuration files, run evaluations and monitor progress.

**Performance Evaluation of Network Representations** Leveraging the capabilities of EvalNE, we also perform an extensive empirical evaluation of 23 unsupervised representation learning methods on seven real-world networks. The evaluated approaches include NLP-inspired techniques based on random walks and Skip-Gram, matrix factorization approaches, deep and shallow neural models and probabilistic methods. Representation quality is assessed through link prediction and network reconstruction results. In the evaluation we analyze different embedding dimensionalities ($d$) and observe that some approaches perform better for lower values of this parameter. This finding directly challenges the common practice of setting $d = 128$. We also compare different strategies for sampling edges from $\mathbf{E}$ in terms of efficiency and effect on downstream performance. Here, we find that our proposed *spanning_tree* approach is significantly faster than the naive sampling used in the literature. Negative sampling techniques for $\mathbf{D}_{tr}$ and $\mathbf{D}_{te}$ are also compared as well as the effect on performance of tuning method hyperparameters (including the binary operator in $\mathbf{x}_i \circ \mathbf{x}_j$). In this case, we find only small improvements in performance as a result of the costly hyperparameter tuning operations. Finally, we also compare distinct implementations for the same embedding approaches and observe important differences. Overall, our evaluation shows only minor progress in the fields with many methods performing on par with simple baseline heuristics. Moreover, we did not find statistically significant differences in performance between the studied approaches.

**Robustness Evaluation of Network Representations** Our performance evaluation also revealed that edge sampling strategies (to obtain the set $\mathbf{E}_{tr}$) could results in train graphs $\mathbf{G}_{tr}$ with significantly different properties than $\mathbf{G}$. This, in turn, would affect downstream performance of the evaluated methods to various degrees. To further investigate how changes in the input graph affect embedding quality and, thus, downstream performance, we conducted a systematic analysis of network robustness. In this work we used 14 different random and adversarial attacks to poison the input networks and measured changes in node classification and network reconstruction predictions. For efficiency and interpretability of results we limited our adversarial attacks to heuristics targeting network properties such as degree, assortativity and other centrality measures. Our experimental results showed that most embedding approaches provide robust network reconstruction scores while node classification results vary significantly. We also made another interesting observation. For relatively low attack budgets ($< 0.2 \cdot |\mathbf{E}|$) edge addition attacks impact performance more than edge deletions. For values above this threshold, edge deletions become more damaging. We also found that commonly used baseline attacks for node classification that leverage full knowledge of the node labels do not significantly outperform other types of structural attacks such as those based on node degree.

**Learning Representations of Signed Networks** Lastly, we direct our attention towards signed networks where edges can represent positive or negative interactions between entities. In this context we introduce a novel sign network embedding and sign prediction model. The proposed Conditional Signed Network Embedding (CSNE) builds upon the CNE framework [2], which showed excellent performance in our benchmark evaluation (see Chapter 4). This chapter, thus, acts as a case in point of the importance of benchmark studies and evaluation tools for boosting progress in research. In this case, by uncovering a new model with properties that can address some of the main challenges in the related field of signed network embeddings. CSNE is a two-part probabilistic model. First, a maximum-entropy prior (see [21]) is used to model structural information about the network, such as the tendency of users to make more positive or more negative connections, or the extent to which structural balance theory holds in the network (i.e. to what extent statements such as *'the enemy of my enemy is my friend'* hold [22]). This prior is then combined with an embedding capturing additional fine-grained detail about the node interactions. Finally, the Bayes formula is used to combine these two elements. The modular nature of CSNE presents two main advantages. On one hand, it avoids overfitting of the embedding by encoding part of the information in the prior. On the other hand, unlike the state-of-the-art, CSNE does not enforce structural balance by design, instead it learns the extent to which it applies in every network. Our experimental results confirm that CSNE consistently out-

performs the state-of-the-art for sign prediction. We also show that the MaxEnt prior can be directly used for sign prediction. Independently, this model achieves comparable results to our baselines with runtimes $50\times$ lower than the fastest competing method.

## 1.4 Outline

This dissertation comprises a number of publications that were realized within the scope of this PhD. The selected publications are presented as different chapters and provide an integral and consistent overview of the work performed. The main research contributions are detailed in Section 1.3 and the complete list of publications that resulted from this work is presented in Section 1.5. In what follows, we give an overview of the remainder of this dissertation and explain the links between different chapters.

We start in Chapter 2 with our work on reproducibility in the field of network representation learning. More specifically, we discuss reproducibility and evaluation challenges related to the use of link prediction as a downstream task to assess representation quality. In Chapter 4, we will further expand upon the evaluation challenges presented here as well as others related to network reconstruction. In Chapter 2, we also introduce a preliminary version of the EvalNE framework and use it to reproduce the link prediction evaluations in several papers. Specific software design details of the latest version or EvalNE are presented later in Chapter 3.

In Chapter 3, we introduce EvalNE, our network embedding evaluation toolbox. The software was initially designed for reproducing evaluations of network embeddings on link prediction in Chapter 2. Currently, the framework has evolved into a comprehensive evaluation toolbox (extensively used in our work) covering all major downstream evaluation tasks. These tasks include link prediction and network reconstruction, evaluated in our benchmark study in Chapter 4; node classification, used in our robustness evaluation in Chapter 5; and sign prediction and data visualization, leveraged to assess the performance of the CSNE model proposed in Chapter 6. The robustness evaluation module developed for our work in Chapter 5 will be released in an upcoming version of EvalNE.

Chapter 4 details an extensive empirical evaluation of the performance of network representation learning methods. We investigate two downstream tasks, link prediction and network reconstruction as well as the impact of a range of design choices commonly adopted in the literature. The experimental evaluation is performed using the EvalNE framework proposed in Chapter 3. The aim is to shed light on the state-of-the-art in this field, thus, addressing one of the main challenges discussed in Chapter 1.

In Chapter 5 we provide a systematic analysis of the robustness of node embedding approaches, a particular type of representation learning methods. More

specifically, we investigate changes in downstream performance for various models under random corruptions and adversarial attacks on the input data. This research builds on the observations from Chapter 4 regarding differences in downstream link prediction performance when the training networks are sampled according to different strategies. Our experimental evaluation leverages the EvalNE framework presented in Chapter 3.

Chapter 6 introduces a novel signed network embedding and sign prediction model. The proposed method is based on Conditional Network Embedding, an unsupervised representation model which has shown excellent results in our experimental evaluation in Chapter 4. Hereby, this chapter highlights two key benefits of the contributions in this dissertation. On one hand, CNE's potential was identified through the benchmark study in Chapter 4, on the other, the model was evaluated using the EvalNE framework proposed in Chapter 3. The experimental evaluation in this chapter comparing signed network embedding models, was also conducted using EvalNE.

Lastly, in Chapter 7, we summarize the main conclusions of this dissertation and discuss avenues for future research.

## 1.5 Publications

The research results presented in this thesis have been published in scientific journals and presented at a series of international conferences. The following list provides an overview of all publications grouped by type. Subsequent chapters in this book will begin with the name of the relevant publication.

### 1.5.1 Publications in international journals (listed in the Science Citation Index [3] )

1. **Alexandru Mara**, Jefrey Lijffijt and Tijl De Bie. *EvalNE: A Framework for Network Embedding Evaluation.* SoftwareX, 17, Jan. 2022.

2. **Alexandru Mara**, Jefrey Lijffijt and Tijl De Bie. *An Empirical Evaluation of Network Representation Learning Methods.* Big Data, Mar. 2022.

   This journal paper is a substantial extension of the following conference manuscript:

   - **Alexandru Mara**, Jefrey Lijffijt and Tijl De Bie. *Benchmarking Network Embedding Models for Link Prediction: Are we Making*

[3] The publications listed are recognized as 'A1 publications', according to the following definition used by Ghent University: A1 publications are articles listed in the Science Citation Index Expanded, the Social Science Citation Index or the Arts and Humanities Citation Index of the ISI Web of Science, restricted to contributions listed as article, review, letter, note or proceedings paper.

*Progress?* In 2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA 2020), pages 138–147, Oct. 2020.

### 1.5.2 Publications in international conferences (listed in the Science Citation Index [4] )

1. **Alexandru Mara**, Jefrey Lijffijt and Tijl De Bie. *EvalNE: A Framework for Evaluating Network Embeddings on Link Prediction.* Proceedings of the 1st Workshop on Evaluation and Experimental Design in Data Mining and Machine Learning (EDML 2019) co-located with SIAM International Conference on Data Mining (SDM 2019), pages 5–13, Jan. 2019.

2. **Alexandru Mara**, Yoosof Mashayekhi, Jefrey Lijffijt and Tijl De Bie. *CSNE: Conditional Signed Network Embedding*. Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM 2020), Oct. 2020.

### 1.5.3 Publications in other international conferences

1. **Alexandru Mara**, Jefrey Lijffijt, Stephan Günnemann and Tijl De Bie. *A Systematic Evaluation of Node Embedding Robustness.* To appear: Proceedings of the First Learning on Graphs Conference (LoG 2022), PMLR 198, Dec. 2022.

[4]The publications listed are recognized as 'P1 publications', according to the following definition used by Ghent University: P1 publications are proceedings listed in the Conference Proceedings Citation Index - Science or Conference Proceedings Citation Index - Social Science and Humanities of the ISI Web of Science, restricted to contributions listed as article, review, letter, note or proceedings paper, except for publications that are classified as A1.

# References

[1] B. Perozzi, R. Al-Rfou, and S. Skiena. *DeepWalk: Online Learning of Social Representations*. In Proc. of KDD, pages 701–710, 2014.

[2] B. Kang, J. Lijffijt, and T. D. Bie. *Conditional Network Embeddings*. In Proc. of ICLR, 2019.

[3] A. Tsitsulin, M. Munkhoeva, D. Mottin, P. Karras, I. Oseledets, and E. Müller. *FREDE: Anytime Graph Embeddings*. Proc. VLDB Endow., 14(6):1102–1110, feb 2021.

[4] G. Alanis-Lobato, P. Mier, and M. Andrade. *Efficient embedding of complex networks to hyperbolic space via their Laplacian*. Scientific reports, 6:Art. 30108, 2016. doi:http://doi.org/10.25358/openscience-7398.

[5] S. T. Roweis and L. K. Saul. *Nonlinear Dimensionality Reduction by Locally Linear Embedding*. Science, 290(5500):2323–2326, 2000. doi:10.1126/science.290.5500.2323.

[6] M. Belkin and P. Niyogi. *Laplacian eigenmaps and spectral techniques for embedding and clustering*. In Proc. of NIPS, pages 585–591, 2002.

[7] A. Grover and J. Leskovec. *node2vec: Scalable feature learning for networks*. In Proc. of KDD, pages 855–864, 2016.

[8] D. Wang, P. Cui, and W. Zhu. *Structural deep network embedding*. In Proc. of KDD, pages 1225–1234, 2016.

[9] T. N. Kipf and M. Welling. *Semi-Supervised Classification with Graph Convolutional Networks*, 2016. arXiv:1609.02907.

[10] W. L. Hamilton, R. Ying, and J. Leskovec. *Inductive Representation Learning on Large Graphs*. In Proc. of NIPS, page 1025–1035, Red Hook, NY, USA, 2017. Curran Associates Inc.

[11] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. *Graph Attention Networks*. Proc. of ICLR, 2018.

[12] E. Heiter, B. Kang, T. De Bie, and J. Lijffijt. *Evaluating Representation Learning and Graph Layout Methods for Visualization*. IEEE Computer Graphics and Applications, 42(3):19–28, 2022. doi:10.1109/MCG.2022.3160104.

[13] A. Mara, J. Lijffijt, and T. De Bie. *Benchmarking Network Embedding Models for Link Prediction : Are we Making Progress?* In Proc. of DSAA, pages 138–147, 2020. doi:10.1109/DSAA49011.2020.00026.

[14] M. F. Dacrema, P. Cremonesi, and D. Jannach. *Are We Really Making Much Progress? A Worrying Analysis of Recent Neural Recommendation Approaches*. In Proc. of RecSys, page 101–109, 2019. doi:10.1145/3298689.3347058.

[15] M. Hutson. *Core progress in AI has stalled in some fields*. Science, 368(6494):927–927, 2020. doi:10.1126/science.368.6494.927.

[16] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson. *Benchmarking Graph Neural Networks*. Arxiv, 2020.

[17] D. Zügner, O. Borchert, A. Akbarnejad, and S. Günnemann. *Adversarial Attacks on Graph Neural Networks: Perturbations and Their Patterns*. ACM Trans. Knowl. Discov. Data, 14(5), jun 2020.

[18] F. Monti, F. Frasca, D. Eynard, D. Mannion, and M. M. Bronstein. *Fake News Detection on Social Media using Geometric Deep Learning*. ArXiv, abs/1902.06673, 2019.

[19] Mashayekhi, Yoosof and Kang, Bo and Lijffijt, Jefrey and De Bie, Tijl. *Quantifying and Reducing Imbalance in Networks*. In Proc. of FEAST, ECML-PKDD 2021 Workshop, page 10, 2021.

[20] L. Rasmy, Y. Xiang, Z. Xie, C. Tao, and D. Zhi. *Med-BERT: pretrained contextualized embeddings on large-scale structured electronic health records for disease prediction*. In npj Digital Medicine, 2021.

[21] F. Adriaens, A. Mara, J. Lijffijt, and T. De Bie. *Block-Approximated Exponential Random Graphs*. In Proc. of DSAA, pages 70–80, 2020. doi:10.1109/DSAA49011.2020.00019.

[22] F. Heider. *Attitudes and Cognitive Organization*. The Journal of Psychology, 21(1):107–112, 1946. PMID: 21010780.

# 2

# Reproducibility in Network Representation Learning

*In this chapter, we focus on reproducibility in network representation learning. Specifically, we discuss reproducibility and evaluation challenges related to the use of link prediction as a downstream task to assess representation quality. In Chapter 4 we will further expand upon the evaluation challenges presented here as well as others related to network reconstruction. In the current chapter, we also introduce a preliminary version of the EvalNE framework geared towards reproducibility and use it to replicate link prediction results of seminal papers in the field. Specific software design details of an extended version or EvalNE will be presented later in Chapter 3. This chapter is based on the manuscript referenced below. Changes to the text have been made throughout in order to avoid repetitions with other chapters and emphasize the reproducibility aspect of this work. Some section names, tables and figures have also been altered to ensure consistency across chapters.*

⋆ ⋆ ⋆

**A. Mara, J. Lijffijt, and T. De Bie.** ***EvalNE: A Framework for Evaluating Network Embeddings on Link Prediction.***

**Proceedings of EDML (SDM), Jan. 2019.**

**Abstract** Network embedding (NE) methods aim to learn low-dimensional representations of network nodes as vectors, typically in Euclidean space. These representations are then validated on a variety of downstream prediction tasks. Link prediction is one of the most popular choices for assessing NE performance. However, evaluating representation quality through link prediction requires a carefully designed pipeline to ensure consistent, reproducible and comparable results. We argue this has not been considered sufficiently in many recent works. In this chapter, we introduce EvalNE, a new framework for reproducible evaluations of NE methods on link prediction. We also show empirical evidence of an ongoing reproducibility crisis which could be alleviated by tools such as EvalNE. The proposed framework is freely available as a Python package and provides automation and abstraction for tasks such as hyper-parameter tuning, model validation, edge sampling and computation of node-pair embeddings. The framework integrates efficient procedures for edge and non-edge sampling and can be used to evaluate any off-the-shelf embedding method. In our experimental evaluation, we attempt to replicate the results of several influential papers in the field. We show that provided all design choices, EvalNE can perfectly replicate results. On the other hand, relying exclusively on the information reported in the manuscripts and reference implementations, the results vary significantly.

## 2.1 Introduction

Link prediction is an important task with applications in a wide range of fields such as computer science, social sciences, biology, and medicine [1–4]. It amounts to estimating the likelihood for the existence of edges, between pairs of nodes that do not form an edge in the input graph.

Many recent Network Embedding (NE) methods, e.g., [5–13], have been applied to solving link prediction problems, showing promising results. These methods map nodes in the network to $d$-dimensional vectors, typically in Euclidean space. The obtained representations are then used as features for a variety of tasks such as visualization, multi-label classification, clustering or link prediction.

**The challenges of evaluating NE methods for link prediction** Unfortunately, the practical performance of most NE methods is poorly understood and not comparable due to inconsistent evaluation procedures. In this chapter, *we focus on a number of difficulties specific to the evaluation of NE methods for link prediction*. Link prediction is particularly challenging as the evaluation involves a number of design choices that can confound the results and are prone to errors.

For example, the implicit assumption is that the input graph is not complete and the purpose is to predict the missing edges as accurately as possible. To evaluate the performance of a NE method for link prediction, one thus needs an (in-

complete) training graph along with a (more) complete version of that graph for testing. Much research has been devoted to determining the best approach to generate these training graphs [1, 2, 4]. Theoretical and empirical evidence suggest that in order to fairly evaluate link prediction methods, snapshots of the networks at different points in time should be used for training and testing. In this way, the methods are tested on natural evolutions of the networks. Nonetheless, the availability of such snapshots is uncommon and raises additional questions, such as how to choose the time intervals for splitting the networks.

For these reasons, in most cases [7–10] authors resort in their evaluations to sampling sets of edges from the input graphs and using the remaining subgraphs for training. The sampled edges are later used as positive test examples. The process of sampling edges is not standardized and varies in several ways between studies. The relative sizes of the train and test sets, for example, is a user-defined parameter which varies significantly between experimental setups. In [8, 9] the authors use a 50-50 train-test split, in [7] a 60-40, in [10] an 80-20 and in [14] values ranging from 30-70 up to 80-20. Moreover, certain methods have requirements such as connectivity of the training graph [8], while others do not [10]. In addition, the more commonly employed sampling strategies lack efficiency and scalability and can, thus, rapidly become evaluation bottlenecks.

A related problem is the fact that, in addition to 'positive' train and test edges, in most cases 'negative' edges (also called non-edges) are required as well. Non-edges can be leverage by some NE implementations to derive better network representations, while in a more general case they are used to assess downstream link prediction performance. The sets of non-edges can be selected according to different strategies [15] and can be of various sizes.

Furthermore, most NE implementations only provide node embeddings. In order to derive link prediction from these embeddings two pipelines are commonly used. On one hand, link predictions can be computed directly from the node embeddings via e.g., dot products. On the other hand, node-pair embeddings can be first computed by applying a binary operator on the node embeddings. Subsequently a binary classifier can be trained to distinguish between the node-pair representations of edges and non-edges. In the literature several approaches for deriving node-pair embeddings have been proposed [16], and have been shown to have a high impact on link prediction performance [8].

Finally, it appears to be common practice in recent literature to use recommended default settings for existing methods, while tuning hyperparameters for the method being introduced. Especially when the recommended default settings were informed by experiments on other graphs than those used in the study at hand, this can paint an unduly unfavorable picture.

**Contributions** In this chapter we propose a new framework for reproducible evaluations of NE methods on link prediction. The framework coined EvalNE automates many parts of the evaluation process from hyperparameter tuning to edge sampling. Specifically, the framework: (1) Implements guidelines from research in the area of link prediction evaluation and sampling [1, 2, 4]. (2) Includes (novel) efficient edge and non-edge sampling algorithms. (3) Provides the most widely used node-pair embedding operators. (4) Evaluates the scalability and accuracy of methods, through wall clock time and a wide range of fixed-threshold metrics and threshold curves. (5) Integrates in a single operation the evaluation of any set of NE methods and networks. (6) Ensures reproducibility and comparability of evaluations and results through shareable configuration files. Using EvalNE we replicate the experimental link prediction evaluations of several key works in the field and find evidence of an ongoing reproducibility crisis. Finally, we also compare the novel edge sampling strategy included in EvalNE to the naive approach commonly used in the literature. Experimental results confirm the higher efficiently and scalability of the proposed approach.

The remainder of this chapter is organized as follows. In Section 2.2 we give a brief overview of the related work. In Section 2.3 we present our evaluation framework. Our experiments reproducing the results of several papers and an empirical analysis of the proposed edge sampling strategy are reported in Section 2.4. Finally, Section 2.5 concludes this chapter.

## 2.2 Related work

Our work is related to the evaluation of link prediction which has been studied in several recent works, i.e., [1, 2, 4, 17]. These studies focus on the impact of different edge sampling strategies, negative sampling and fair evaluations criteria.

Link prediction as a downstream evaluation task for the representations learned by NE algorithms was first introduced in the pioneering work of Grover et al. [8]. In a later survey, Zhang et al. [18] point out the importance of link prediction as an application of network representation learning. The authors also signal inconsistencies in the evaluation of different NE approaches and conduct an empirical analysis of several models on a wide range of datasets. Their empirical study, however, focuses exclusively on vertex classification and clustering. The importance of standard evaluation frameworks as tools to bridge the gap between research and application of NEs is discussed by Hamilton et al. [19].

To the best of our knowledge only two frameworks for the evaluation of NE methods currently exist. `OpenNE` is a recently proposed toolbox for evaluating NE methods on multi-label classification exclusively. The toolbox also includes implementations of several state-of-the-art embedding methods. `GEM` [20] is a similar framework which also implements a variety of embedding methods and

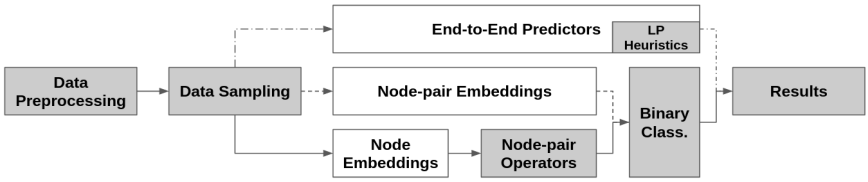



*Figure 2.1: Diagram showing the types of methods that can be evaluated using EvalNE. Gray blocks represent modules provided by the library and white blocks are the user-specified methods to be evaluated. The library can evaluate end-to-end predictors (including link prediction heuristics), node-pair embedding and node embedding methods.*

includes basic evaluation on multi-label classification, visualization and link prediction tasks. These frameworks, however, are focused on the implementations of embedding methods rather than the evaluation pipeline. Thus, these libraries, unlike EvalNE, are limited to the evaluation of the NE methods they implement.

## 2.3 Methods

In this section, we discuss the key aspects of our proposed framework. EvalNE has been designed as a pipeline of interconnected and interchangeable building blocks, as illustrated in Figure 2.1. These blocks have specific functions such as providing sets of train and test edges or computing node-pair embeddings from given node embeddings. The modular structure of our framework simplifies code maintenance and the addition of new features, and allows for flexible model evaluation. The framework can be used to evaluate methods providing node embeddings, node-pair embeddings, or node similarity scores (we include in this category the link prediction heuristics).

EvalNE is available as an open-source Python toolbox on GitHub[1] and the documentation and examples are hosted by ReadTheDocs[2]. The framework supports all major operating systems and can be used as a command line tool (providing appropriate configuration files) and as an API. Further details regarding the software architecture are provided in Chapter 3. In what follows we describe the main building blocks used in downstream link prediction evaluation and required for our evaluation in Section 2.4.

### 2.3.1 Evaluation framework

The core building blocks of our framework are the data split and model evaluation. These blocks constitute the most basic pipeline for assessing the quality of link

[1] https://github.com/Dru-Mara/EvalNE
[2] https://evalne.readthedocs.io/en/latest/index.html

predictions. Before describing these blocks, we first introduce some notation.

We represent a directed network as $G = (V, E)$ with vertex set $V = \{1, \ldots, n\}$ and edge set $E \subseteq V \times V$. Edges are represented as ordered pairs $e = (u, v) \in E$ with $|E| = m$. $E_{train}$ and $E_{test}$ denote the training and testing edge sets. We represent the embedding of network nodes into a $d$-dimensional space as $\mathbf{X} = (x_1, x_2, \ldots, x_n)$ where $\mathbf{X} \in \mathbb{R}^{\mathrm{n} \times \mathrm{d}}$. In what follows, we restrict our evaluation to directed and undirected networks with no attributes, labels or edge weights. The latest version of EvalNE, however, does cover the evaluation of labeled and weighed graphs as well (see Chapter 3).

#### 2.3.1.1 Data split

As pointed out in Section 2.1, in order to perform link prediction on a given input graph $G$, sets of train and test edges are required. The set of train edges is generally required to span all nodes in $G$ and induce a training graph $G_{train}$ with a single connected component, because embeddings of independent components will be far away from and unrelated to each other.

Many recent studies resort to a naive algorithm to derive a connected $G_{train}$. The procedure removes edges from an input graph iteratively until a desired number of train edges remain. The removed edges are used as test samples. In each iteration, the connectivity of the graph needs to be checked and only remove an edge if it does not result in a disconnected graph. This requirement is generally satisfied by running a Breadth First Search (BFS) on the graph after each edge removal, which is a costly operation ($\mathcal{O}(n + m)$).

Integrated in our evaluation framework, we include a novel algorithm (Algorithm 1) to perform the train-test splits which, as we will show in Section 2.4, is orders of magnitude faster yet equally simple. Algorithm 1 also accounts for the fact that the training graph $G_{train}$ must span all nodes in $G$ and contain a single connected component.

**Algorithm 1:** Given an undirected graph $G = (V, E)$ and a target number of train edges $m_{tr}$, the proposed algorithm to split train and test edges proceeds as follows:

1. Obtain a uniform spanning tree $ST$ of $G$ using Algorithm 2.
2. Initialize the set of training edges $E_{train}$ to all edges in $ST$.
3. Select an edge uniformly at random from $E \setminus E_{train}$.
4. Add the sampled edge to $E_{train}$.
5. Repeat steps 3 and 4 until $|E_{train}| = m_{tr}$ then output $E_{train}$ and $E \setminus E_{train}$.

**Algorithm 2 (Broder's algorithm [21]):** Given an undirected graph $G$ select a spanning tree uniformly at random from the set of all possible ones:

1. Select a random vertex $s$ of $G$ and start a random walk on the graph until every vertex is visited. For each vertex $i \in V \setminus \{s\}$ collect the edge $e = (j, i)$ that corresponds to the first entrance to vertex $i$. Let ST be this collection of edges.

2. Output the set ST.

The complexity of Algorithm 2 is $\mathcal{O}(n \log n)$ on expectation [21]. The same bound applies for the proposed Algorithm 1 as the sampling and addition of random edges to the train set can be done in constant time. A more efficient alternative to Algorithm 2 for obtaining a uniformly sampled spanning tree of a given graph exists. This method, named Wilson's algorithm [22], was introduced in a later version of EvalNE (see Chapter 3) and evaluated in Chapter 4.

For directed graphs $G$, we first construct an equivalent undirected version $G^*$ by adding reciprocals for every edge in $E$. We then run Algorithm 1 on $G^*$ and include in the train set only those edges present in the initial directed graph $G$. This method results in a weakly connected training graph spanning the same nodes as the original $G$.

In addition to train and test edges, sets of train and test *non-edges* (also referred to as *negative samples*) are required in order to evaluate link prediction. Non-edges represent pairs of vertices $u, v$ such that $e = (u, v) \notin E$. The proposed toolbox can compute these non-edges according to either the *open world* or the *closed world* assumption. The *open world* assumption models a scenario where non-edges can correspond to either unobserved or non-existent connections between nodes (equivalent to a collaborative filtering scenario with implicit feedback). Thus, under the *open world* assumption, train non-edges are selected such that they are not in $E_{train}$ and thus overlapping between train non-edges and test true edges can occur. The *closed world* assumption, on the other hand, models a scenario where the user has certainty about the status of some non-edges as non-existent connections (akin to collaborative filtering with explicit feedback). In this case non-edges are sampled such that they are not in $E_{train}$ nor in $E_{test}$.

The number of train and test edges and non-edges are user-controlled parameters. The minimum number of train edges is determined by the size of the spanning tree and the combined sets of non-edges have a trivial upper limit of $(n \cdot n) - m$. However, for the train set size, fractions between 50% and 90% of $m$ are recommended. For values below 50%, the resulting training graph will often not preserve the properties of the original graph $G$ as shown in [23].

#### 2.3.1.2 Model evaluation

Our proposed framework can evaluate the scalability, parameter sensitivity and accuracy of embedding methods. We assess scalability by directly measuring wall clock time. Method performance can be easily reported for different embedding dimensionalities and hyperparameter values. Finally, the link prediction accuracy is reported using two types of metrics: fixed-threshold metrics and threshold curves. Fixed-threshold metrics summarize the performance of the methods to single values. EvalNE provides the following measures: confusion matrix (TP, FN, FP, TN), precision, recall, fallout, miss, accuracy, F-score and AUC. Threshold curves present the performance of the methods for a range of threshold values in $[0, 1]$. EvalNE includes precision-recall curves [2] and receiver operating curves [24].

### 2.3.2 Other building blocks and baselines

In addition to the core building blocks and in order to extend the evaluation process to different types of embedding methods, our framework also provides data preprocessing, data manipulation, and different node-pair embedding functions. Moreover, EvalNE integrates a binary classification pipeline to obtain link predictions from node-pair embeddings. Furthermore, the toolbox contains a series of built-in link prediction heuristics as baselines.

#### 2.3.2.1 Preprocessing

The toolbox offers a variety of functions to load, store, and manipulate networks. These include methods to prune nodes based on degree, remove self-loops, relabel nodes, obtain sets of specific types of edges, restrict networks to their main connected components and obtain common network statistics. The preprocessing functions of EvalNE build on top of and can be used in combination with those provided by other mainstream software packages (e.g., NetworkX [25]).

#### 2.3.2.2 Link prediction heuristics

A training graph $G_{train} = (V, E_{train})$ spanning the same set of vertices as the initial graph $G$ but containing only the edges in the train set is provided as input to the link prediction heuristics. Depending on the input graph type we use one of the following definitions. For undirected graphs we use the ones presented below, where $\Gamma(u)$ denotes the neighborhood of a node $u$. For directed graphs we restrict our analysis to either the in-neighborhood ($\Gamma_i(u)$) or the out-neighborhood ($\Gamma_o(u)$) of the nodes.

*Common neighbors (CN):*

$$\mathrm{CN}(u, v) = |\Gamma(u) \cap \Gamma(v)|$$

*Jaccard coefficient (JC):*

$$\mathrm{JC}(u, v) = \frac{|\Gamma(u) \cap \Gamma(v)|}{|\Gamma(u) \cup \Gamma(v)|}$$

*Adamic-Adar index (AA):*

$$\mathrm{AA}(u, v) = \sum_{w \in \Gamma(u) \cap \Gamma(v)} \frac{1}{\log |\Gamma(w)|}$$

*Resource allocation index (RAI):*

$$\mathrm{RAI}(u, v) = \sum_{w \in \Gamma(u) \cap \Gamma(v)} \frac{1}{|\Gamma(w)|}$$

*Preferential attachment (PA):*

$$\mathrm{PA}(u, v) = |\Gamma(u)| \cdot |\Gamma(v)|$$

*Katz similarity:*

$$\mathrm{Katz}(u, v) = \sum_{l=1}^{\infty} \beta^l \cdot |\rho(u, v|l)|$$

For Katz, the parameter $\beta \in [0, 1]$ is a damping factor and $|\rho(u, v|l)|$ represents the number of paths of length $l$ from node $u$ to node $v$.

In addition to these methods, a random prediction model is provided for reference. This method simply outputs a uniform random value in $[0, 1]$ as the likelihood of a link between any pair of given vertices $(u, v)$.

#### 2.3.2.3 Node-pair embedding operators

Unlike for the aforementioned heuristics where the output can be directly used for link prediction, for NE methods this is generally not the case. Most implementations of NE methods produce node embeddings only. Thus, an additional step of learning node-pair embeddings is required in order to predict links via binary classification.

The node-pair representations are typically derived in an unsupervised fashion. For example, to derive the embedding $y_{(u,v)}$ for node-pair $(u, v)$, we apply a binary operator $\circ$ over the embeddings of nodes $u$ and $v$, i.e. $x_u$ and $x_v$ respectively:

$$y_{(u,v)} = x_u \circ x_v.$$

In [8] the authors propose the following alternatives for the operator $\circ$ which we include in our evaluation framework. Any other user-defined operators can be also easily integrated.

*Average (Avg.):*

$$x_u \oplus x_v \equiv \frac{x_{u,i} + x_{v,i}}{2}$$

*Hadamard (Had.):*

$$x_u \odot x_v \equiv x_{u,i} * x_{v,i}$$

*Weighted $L_1$:*

$$||x_u \cdot x_v||_{\bar{1}} \equiv |x_{u,i} - x_{v,i}|$$

*Weighted $L_2$:*

$$||x_u \cdot x_v||_{\bar{2}} \equiv |x_{u,i} - x_{v,i}|^2$$

#### 2.3.2.4 Binary classification

Several NE approaches (e.g., [7–10]) rely on a logistic regression classifier to predict link probabilities from node-pair embeddings. In EvalNE we include logistic regression with 10-fold cross validation. The framework, however, is flexible and allows for any other binary classifier to be used.

## 2.4 Experiments

In this section, we use EvalNE to replicate the link prediction results of the following four manuscripts from the NE literature i.e. Node2vec, SDNE, PRUNE and CNE. We have selected these methods due to their high citation counts and the wide array of evaluation pipelines they exhibit. The main goals of this experimental evaluation are showcasing the capabilities of EvalNE and highlighting the need to extensively discuss design choices. We also conduct a series of additional experiments in which we compare our edge sampling strategy (Algorithm 1) to the naive approach used in the literature. We perform these comparisons in terms of both scalability and link prediction accuracy. All experiments were run on a single machine equipped with a Intel® Core™ i5 processor and 16GB of RAM.

### 2.4.1 Experimental setups

Next, we present the main characteristics of the experimental settings replicated and some important remarks:

*Node2vec* [8]: In this work the authors evaluate the following embedding methods and link prediction heuristics: Node2vec, DeepWalk [12], LINE [13], Spectral Clustering [26], CN, JC, AA, and PA. Experiments are performed on the Facebook [27], PPI [8], and AstroPh [27] datasets with 50-50 train-test splits. The same number of edges and non-edges are used throughout the experiment. The results are reported in terms of AUC and the node-pair operators used are average, Hadamard, weighted $L_1$ and weighted $L_2$.

*SDNE* [14]: The authors report in this case experiments using the following methods: DeepWalk, LINE, SDNE, GraRep [6], LapEig [5] and CN. The link prediction experiments are performed on the GR-QC dataset [27] with a 0.85 train-test fraction. The results are reported only for the Hadamard operator and in terms of precision@k for a collection of values between 2 and 10000. In this setup, the number of train non-edges is the same as that of train edges while all the remaining non-edges in the graph are used for testing.

*PRUNE* [10]: This experimental setup includes Node2vec, DeepWalk, LINE, SDNE, PRUNE and NRCL [28]. The networks used are HepPH [27], FB-wallpost [29] and Webspam and the train-test fraction is 0.8 with similar sizes of the non-edge sets. The node-pair embedding operator used is not reported by the authors and the results are presented in terms of AUC. In this setting, the networks used are directed.

*CNE* [9]: The evaluation setup in this case is similar to that of Node2vec with the following differences: CNE and Metapath2vec [30] are also evaluated; BlogCatalog [31], Wikipedia [27] and StudentDB are included in the list of evaluated networks and the results are only reported for the Hadamard operator. It should be noted that two authors of this work are co-authors of CNE, which granted us access to all major design choices and the complete evaluation code.

In our experiments we have replicated these setting with the following exceptions (1) for node2vec we did not include spectral clustering and (2) for PRUNE we could not obtain NRCL and lacked the computational resources to evaluate the NE methods on Webspam. In all cases we report the same metrics as the original authors and average results over five independent executions (original values used by the authors are not reported in the manuscripts). We used the *open world* assumption for the non-edges (original choice is also not reported), logistic regression as binary classifier and tuned the same hyperparameters. Each evaluation was run using a different EvalNE configuration file which we make available online. We note that for all methods, with the exception of CNE, we only used the publicly available information regarding experimental design choices, that is, the manuscript contents and reference implementations.

Regarding method implementations, we evaluated the link prediction heuris-

*Table 2.1: Datasets used for evaluation.*

| Dataset | Category | $\|V\|$ | $\|E\|$ |
|---|---|---|---|
| Facebook | Social | 4039 | 88234 |
| FB-wallpost | Social | 43953 | 262631 |
| BlogCatalog | Social | 10312 | 333983 |
| StudentDB | Social | 395 | 3423 |
| HepPh | Citation | 34546 | 421578 |
| AstroPh | Collaboration | 17903 | 196972 |
| GR-QC | Collaboration | 4158 | 26844 |
| PPI | Biological | 3852 | 37841 |
| Wikipedia | Language | 4777 | 92295 |

tics included in EvalNE; original code by the authors for Deepwalk[3] , Node2vec[4], LINE[5], PRUNE[6], CNE[7], and Metapath2vec[8], and for the remaining ones, the implementations in the `OpenNE`[9] library. Table 2.1 contains the main characteristics of all the networks used. Network sizes are those corresponding to the main (weakly) connected components of each graph.

## 2.4.2 Experimental results

### 2.4.2.1 Reproducibility results

For each of the experimental settings reproduced we present a table containing the original values reported in the paper, and in parenthesis the difference between our result and these values (Tables 2.2–2.5). Positive values in parenthesis, thus, indicate that our results are higher than the ones reported in the original papers by that margin, while negative values indicate the opposite.

Our results in Table 2.2 and Table 2.3 show some significant differences with respect to the values reported in the Node2vec and SDNE manuscripts (differences over $0.15$ are marked in bold in all tables). Possible explanations for these inconsistencies are the use of different implementations of NE methods, different non-reported default parameters or the use of parallelization. In addition, we have also observed important differences when computing the AUC directly from class labels or from class probabilities.[10] To replicate the Node2vec experiments we used class labels as they resulted in closer values to those reported by the authors.

[3] https://github.com/phanein/deepwalk
[4] https://github.com/aditya-grover/node2vec
[5] https://github.com/tangjianpku/LINE
[6] https://github.com/ntumslab/PRUNE
[7] https://bitbucket.org/ghentdatascience/cne/
[8] https://ericdongyx.github.io/metapath2vec/m2v.html
[9] https://github.com/thunlp/OpenNE
[10] In some cases authors compute class assignments by thresholding prediction probabilities prior to computing the AUC. In these cases the AUCs reported equal accuracy scores at the fixed thresholds.

*Table 2.2: Reported AUC scores in the node2vec paper and difference to our reproduced values.*

| | | Facebook | PPI | AstroPh |
|---|---|---|---|---|
| | CN | 0.81 (+0.14) | 0.71 (+0.06) | 0.82 (+0.13) |
| | JC | 0.88 (+0.04) | 0.70 (+0.04) | 0.81 (+0.12) |
| | AA | 0.83 (+0.13) | 0.71 (+0.06) | 0.83 (+0.12) |
| | PA | 0.71 (+0.04) | 0.67 (+0.13) | 0.70 (+0.08) |
| Avg. | DeepWalk | 0.72 (−0.01) | 0.69 (+0.08) | 0.71 (+0.01) |
| | LINE | 0.70 (−0.03) | 0.63 (+0.12) | 0.65 (+0.14) |
| | node2vec | 0.73 (−0.01) | 0.75 (−0.01) | 0.72 (−0.02) |
| Had. | DeepWalk | 0.97 (−0.03) | **0.74 (−0.2)** | 0.93 (−0.12) |
| | LINE | 0.95 (−0.06) | 0.72 (−0.01) | 0.89 (+0.05) |
| | node2vec | 0.97 (0.0) | **0.77 (−0.17)** | 0.94 (−0.05) |
| W $L_1$ | DeepWalk | 0.96 (−0.01) | 0.60 (+0.14) | 0.83 (+0.09) |
| | LINE | **0.95 (−0.33)** | 0.70 (−0.01) | **0.88 (−0.28)** |
| | node2vec | 0.96 (−0.01) | 0.63 (−0.03) | 0.85 (+0.03) |
| W $L_2$ | DeepWalk | 0.96 (−0.01) | 0.61 (+0.14) | 0.83 (+0.09) |
| | LINE | **0.95 (−0.33)** | 0.71 (−0.02) | **0.89 (−0.27)** |
| | node2vec | 0.96 (0.0) | 0.62 (−0.02) | 0.85 (+0.03) |

*Table 2.3: Reported precision@k scores in the SDNE paper and difference to our reproduced values.*

| | prec@100 | prec@200 | prec@300 | prec@500 | prec@800 | prec@1000 | prec@10000 |
|---|---|---|---|---|---|---|---|
| SDNE | 1 (0.0) | 1 (0.0) | 1 (+0.01) | 0.99 (0.0) | 0.97 (+0.03) | 0.91 (+0.09) | 0.25 (−0.12) |
| LINE | 1 (0.0) | 1 (0.0) | 0.99 (+0.01) | 0.93 (+0.06) | 0.74 (+0.26) | 0.79 (+0.11) | 0.21 (−0.13) |
| DeepWalk | **0.6 (+0.40)** | **0.55 (+0.45)** | **0.44 (+0.56)** | **0.34 (+0.65)** | **0.29 (+0.70)** | **0.29 (+0.71)** | 0.15 (+0.01) |
| GraRep | **0.04 (+0.96)** | **0.03 (+0.97)** | **0.03 (+0.97)** | **0.04 (+0.96)** | **0.03 (+0.97)** | **0.03 (+0.97)** | **0.19 (+0.16)** |
| CN | 1 (0.0) | 0.96 (+0.03) | 0.96 (+0.03) | 0.98 (0.0) | 0.87 (+0.05) | 0.79 (+0.09) | 0.19 (0.0) |
| LapEig | 0.93 (+0.07) | 0.85 (+0.15) | **0.82 (+0.17)** | **0.66 (+0.34)** | **0.46 (+0.53)** | **0.39 (+0.48)** | 0.05 (+0.15) |

The remaining small differences between approaches can be due to the use of different edge sampling strategies (naive in the original manuscripts and our proposed approach in the results reported here).

The results in Table 2.4 show that EvalNE can also replicate evaluations of directed networks. The differences observed are mainly related to unreported design choices and the use of different edge sampling techniques.

Finally, the results presented in Table 2.5 show that with access to all evaluation design choices, as in the case of CNE, results can be replicated almost perfectly. The only exception are the Metapath2vec results on the Facebook network, which were related to a typo in the original manuscript. For CNE, we computed AUC scores from the class probabilities of the final binary classifier, as reported by the authors.

These results illustrate the existence of a reproducibility crisis in the field and an urgent need for: (a) reproducible evaluation pipelines, and (b) detailed reports of experimental design choices such as data sampling, method hyperparameters, and method implementations used.

*Table 2.4: Reported AUC scores in the PRUNE paper and difference to our reproduced values.*

| | DeepWalk | LINE | Node2vec | SDNE | PRUNE |
|---|---|---|---|---|---|
| HepPh | 0.80 (−0.05) | 0.80 (−0.09) | 0.81 (−0.05) | 0.75 (−0.04) | 0.86 (−0.03) |
| FB-wallpost | 0.83 (+0.01) | 0.78 (+0.09) | 0.85 (−0.09) | 0.86 (−0.02) | 0.88 (−0.01) |

*Table 2.5: Reported AUC scores in the CNE paper and difference to our reproduced values*

| | Facebook | PPI | AstroPh | BlogCatalog | Wikipedia | StudentBD |
|---|---|---|---|---|---|---|
| CN | 0.97 (+0.01) | 0.77 (0.0) | 0.94 (+0.01) | 0.92 (+0.01) | 0.84 (0.0) | 0.42 (−0.01) |
| JS | 0.97 (+0.01) | 0.76 (0.0) | 0.94 (+0.01) | 0.78 (0.0) | 0.50 (−0.01) | 0.42 (−0.01) |
| AA | 0.98 (0.0) | 0.77 (+0.01) | 0.94 (+0.01) | 0.93 (0.0) | 0.86 (+0.01) | 0.42 (−0.01) |
| PA | 0.83 (+0.01) | 0.89 (+0.01) | 0.86 (+0.01) | 0.95 (0.0) | 0.91 (+0.01) | 0.91 (+0.01) |
| DeepWalk | 0.98 (0.0) | 0.64 (+0.01) | 0.92 (+0.01) | 0.61 (−0.01) | 0.56 (0.0) | 0.76 (+0.03) |
| LINE | 0.95 (0.0) | 0.75 (+0.01) | 0.98 (0.0) | 0.76 (+0.01) | 0.71 (0.0) | 0.86 (−0.02) |
| Node2vec | 0.99 (0.0) | 0.68 (+0.02) | 0.97 (0.0) | 0.73 (−0.05) | 0.67 (−0.08) | 0.83 (0.0) |
| Metapath | **0.74 (+0.17)** | 0.85 (0.0) | 0.83 (+0.02) | 0.91 (0.0) | 0.83 (+0.02) | 0.92 (−0.01) |
| CNE(unif.) | 0.99 (0.0) | 0.89 (+0.01) | 0.99 (0.0) | 0.92 (+0.01) | 0.84 (0.0) | 0.93 (0.0) |
| CNE(deg.) | 0.99 (0.0) | 0.91 (0.0) | 0.99 (0.0) | 0.96 (0.0) | 0.92 (−0.01) | 0.94 (0.0) |

#### 2.4.2.2 Sampling strategy evaluation

We also compared the proposed edge sampling strategy (Algorithm 1) in terms of accuracy and scalability to the naive approach. To assess accuracy we selected the Facebook, PPI, and AstroPh datasets, and performed link prediction with the following heuristics: CN, JC, AA, and PA. We collected all metrics available in EvalNE and computed for each the absolute difference between the naive and proposed approach. The recall presents the highest deviation, so we show these results in Table 2.6. Although recall differs up to $0.0393$, we note that it is traded off with precision. The maximum difference in accuracy is of $0.0154$ and in AUC, one of the most widely used metrics, $0.015$ attained by PA on AstroPh. Across methods and datasets, the results obtained with the naive approach were slightly higher than those using Algorithm 1. The random edge sampling in the naive approach leads naturally to test edges incident to higher degree nodes. In turn, higher node degrees imply less uncertainty in a particular region of the embedding space and, thus, higher link prediction performance. Algorithm 1, on the other hand, leverages random edge sampling to select train edges (to be added to those already in the spanning tree). This leads to test edges that, on average, connect lower degree nodes and, thus, are harder to predict.

Finally, regarding scalability, in Figure 2.2 we summarize the execution times in seconds of both methods on four different datasets. This experiment shows that Algorithm 1 is several order of magnitude faster than the naive edge sampling and is independent on the numbers of train and test edges required. The execution time of the naive approach decreases, as expected, as the number of test edges requested becomes smaller. In Figure 2.3 we compare the execution times of both

*Table 2.6: Absolute difference in recall between results obtained using the naive and proposed edge sampling for different networks and link prediction heuristics.*

| | Facebook | PPI | AstroPh |
|---|---|---|---|
| CN | 0.0226 | 0.0031 | 0.0189 |
| JC | 0.0134 | 0.0066 | 0.0178 |
| AA | 0.0245 | 0.0031 | 0.0189 |
| PA | 0.0242 | 0.0321 | 0.0393 |

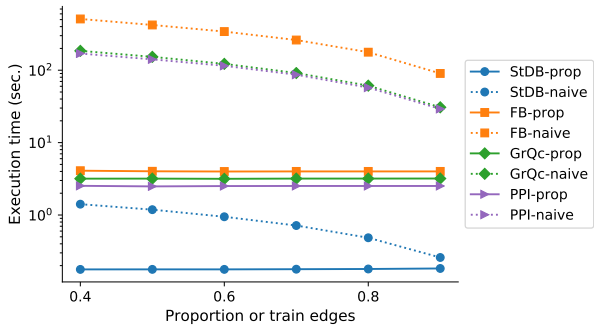



*Figure 2.2: Execution times of proposed and naive edge sampling strategies w.r.t. proportion of train edges. Solid lines correspond to the proposed method, while dashed lines correspond to the typically used (naive) sampling.*

approaches on sub-graphs of the BlogCatalog network of different sizes ranging from 400 up to 10000 nodes. These results also highlight the higher scalability of the proposed sampling strategy.

## 2.5 Conclusions

The recent surge of research in the area of network embeddings has resulted in a wide variety of data sets, metrics, and setups for evaluating and comparing the utility of embedding methods. Comparability across studies is lacking and not all evaluations are equally sound. This motivates the need for specific tools and pipelines to ensure correct and reproducible evaluation of NE approaches. Furthermore, the use of link prediction to assess representation quality induces additional complexities such as train and test edge sampling, non-edge sampling, and in many cases selection of node-pair embedding operators and binary classifiers. The evaluation procedures, thus, becomes an ensemble of tasks which allow for many errors or inconsistencies.

In this chapter we have proposed EvalNE, a novel framework for reproducible evaluations of network embedding methods on link prediction. EvalNE simplifies

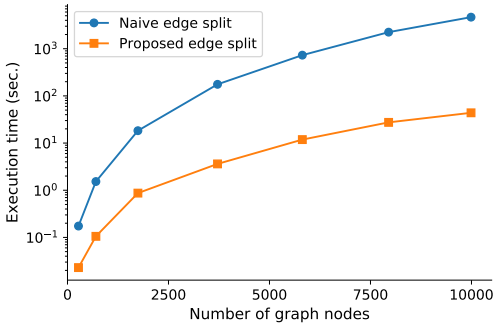



*Figure 2.3: Execution times of proposed and naive edge sampling strategies on sub-graphs of BlogCatalog of increasing sizes. The results are reported for a 50-50 train-test split.*

edge and non-edge sampling, automates model hyperparameter tuning and reports method performance according to many criteria. In our experiments, we have also revealed signs of an ongoing reproducibility crisis in network embeddings sparked by non-standardized evaluations, undocumented design choices, and limited evaluation frameworks. Finally, we have introduced a novel edge sampling strategy and empirically showed that is orders of magnitude faster than the naive approach commonly used in the literature.

## Acknowledgment

The research leading to these results has received funding from the ERC under the EU's Seventh Framework Programme (FP7/2007-2013) / ERC Grant Agreement no. 615517, from the FWO (project no. G091017N, G0F9816N), and from the EU's Horizon 2020 research and innovation programme and the FWO under the Marie Sklodowska-Curie Grant Agreement no. 665501.

# References

[1] D. Garcia-Gasulla, C. U. Cortes, E. Ayguade, and J. J. Labarta. *Evaluating link prediction on large graphs*. In Proc. of CAAI, pages 90–99, 2015.

[2] R. N. Lichtenwalter and N. V. Chawla. *Link Prediction: Fair and Effective Evaluation*. In Proc. of ASONAM, pages 376–383, 2012.

[3] L. Lü and T. Zhou. *Link prediction in complex networks: A survey*. Physica A, 390(6):1150–1170, 2011.

[4] Y. Yang, R. N. Lichtenwalter, and N. V. Chawla. *Evaluating Link Prediction Methods*. KAIS, 45(3):751–782, 2015.

[5] M. Belkin and P. Niyogi. *Laplacian eigenmaps and spectral techniques for embedding and clustering*. In Proc. of NIPS, pages 585–591, 2002.

[6] S. Cao, W. Lu, and Q. Xu. *GraRep: Learning graph representations with global structural information*. In Proc. of CIKM, pages 891–900, 2015.

[7] M. Gao, L. Chen, X. He, and A. Zhou. *BiNE: Bipartite Network Embedding*. In Proc. of SIGIR, pages 715–724, 2018.

[8] A. Grover and J. Leskovec. *node2vec: Scalable feature learning for networks*. In Proc. of KDD, pages 855–864, 2016.

[9] B. Kang, J. Lijffijt, and T. De Bie. *Conditional Network Embeddings*. arXiv:1805.07544, 2018.

[10] Y.-A. Lai, C.-C. Hsu, W. H. Chen, M.-Y. Yeh, and S.-D. Lin. *PRUNE: Preserving Proximity and Global Ranking for Network Embedding*. In Proc. of NIPS, pages 5257–5266, 2017.

[11] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu. *Asymmetric transitivity preserving graph embedding*. In Proc. of KDD, pages 1105–1114, 2016.

[12] B. Perozzi, R. Al-Rfou, and S. Skiena. *DeepWalk: Online Learning of Social Representations*. In Proc. of KDD, pages 701–710, 2014.

[13] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. *LINE: Large-scale information network embedding*. In Proc. of WWW, pages 1067–1077, 2015.

[14] D. Wang, P. Cui, and W. Zhu. *Structural deep network embedding*. In Proc. of KDD, pages 1225–1234, 2016.

[15] B. Kotnis and V. Nastase. *Analysis of the Impact of Negative Sampling on Link Prediction in Knowledge Graphs*. CoRR, abs/1708.06816, 2017. arXiv:1708.06816.

[16] H. Chen, B. Perozzi, R. Al-Rfou, and S. Skiena. *A Tutorial on Network Embeddings*. arXiv:1808.02590, 2018.

[17] V. Martínez, F. Berzal, and J.-C. Cubero. *A Survey of Link Prediction in Complex Networks*. ACM Comp. Surv., 49(4), 2016.

[18] D. Zhang, J. Yin, X. Zhu, and C. Zhang. *Network Representation Learning: A Survey*. arXiv:1801.05852, 2018.

[19] W. L. Hamilton, R. Ying, and J. Leskovec. *Representation Learning on Graphs: Methods and Applications*. arXiv:1709.05584, 2017.

[20] P. Goyal and E. Ferrara. *GEM: A Python package for graph embedding methods*. JOSS, 3(29):876, 2018.

[21] A. Broder. *Generating Random Spanning Trees*. In Proc. of FOCS, pages 442–447, 1989.

[22] D. B. Wilson. *Generating random spanning trees more quickly than the cover time*. In Proc. of STOC, pages 296–303, 1996.

[23] J. Leskovec and C. Faloutsos. *Sampling from Large Graphs*. In Proc. of KDD, pages 631–636, 2006.

[24] T. Fawcett. *ROC Graphs: Notes and Practical Considerations for Researchers*. Technical report, HP Laboratories, 2004.

[25] A. A. Hagberg, D. A. Schult, and P. J. Swart. *Exploring Network Structure, Dynamics, and Function using NetworkX*. In Proc. of the Python in Science Conference, pages 11 – 15, Pasadena, CA USA, 2008.

[26] L. Tang and H. Liu. *Leveraging social media networks for classification*. Data Mining and Knowledge Discovery, 23(3):447–478, Nov 2011. doi:10.1007/s10618-010-0210-x.

[27] J. Leskovec and A. Krevl. *SNAP Datasets: Stanford Large Network Dataset Collection*. http://snap.stanford.edu/data, June 2014.

[28] X. Wei, L. Xu, B. Cao, and P. S. Yu. *Cross View Link Prediction by Learning Noise-resilient Representation Consensus*. In Proc. of WWW, pages 1611–1619, 2017.

[29] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi. *On the Evolution of User Interaction in Facebook*. In Proc. of WOSN, August 2009.

[30] Y. Dong, N. V. Chawla, and A. Swami. *Metapath2Vec: Scalable Representation Learning for Heterogeneous Networks*. In Proc. of KDD, pages 135–144, New York, NY, USA, 2017. ACM. doi:10.1145/3097983.3098036.

[31] R. Zafarani and H. Liu. *Social Computing Data Repository at ASU*, 2009. Available from: http://socialcomputing.asu.edu.

# 3

# EvalNE: A Framework for Network Embedding Evaluation

*In Chapter 1 we established the need for comprehensive evaluation toolboxes. In Chapter 2 we highlighted challenges related to reproducibility of research and introduced EvalNE as a framework for reproducible link prediction evaluations. In this chapter, we present the latest iteration of EvalNE which has evolved into a comprehensive evaluation toolbox covering five important downstream tasks. These tasks include link prediction and network reconstruction, evaluated in our benchmark study in Chapter 4; node classification, used in our robustness evaluation in Chapter 5; and sign prediction and data visualization, leveraged to assess the performance of the CSNE model proposed in Chapter 6. The robustness evaluation module developed in Chapter 5 will be added to EvalNE in version v0.5.0. This chapter is based on the manuscript referenced below. Some section headlines and figures have been altered to ensure consistency across thesis chapters and minor changes have been made to the text to reflect the recent addition of a graphical user interface.*

⋆ ⋆ ⋆

**A. Mara, J. Lijffijt, and T. De Bie.** ***EvalNE: A Framework for Network Embedding Evaluation.***

**SoftwareX, 17, 2022.**

**Abstract** In this chapter we introduce EvalNE, a Python toolbox for network embedding evaluation. The main goal of EvalNE is to aid researchers and practitioners in performing consistent and reproducible evaluations of new embedding methods, replicating existing evaluations, and conducting benchmark studies. The toolbox can evaluate models independently of their programming language and assess the quality of learned representations through data visualization and downstream tasks such as sign and link prediction, network reconstruction, and node 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. 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 minimizing the risk of evaluation errors. Finally, the framework includes a GUI that allows users to set up and execute evaluations in an intuitive and interactive manner, monitor system resources and browse previous evaluations.

## 3.1 Motivation and significance

Network embedding (NE) or network representation learning methods aim to learn low-dimensional representations of network nodes as vectors, typically in 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 [1, 2], link prediction [3, 4], network reconstruction [5, 6], and node classification [7, 8]. A higher performance on these downstream tasks is then generally associated with a better representation of the network.

Evaluating NE methods based on downstream task performance, however, is challenging and requires a number of additional steps and design choices which can confound the results, harm reproducibility, and are prone to errors. Firstly, for each downstream task one must ensure that the input data is preprocessed consistently, such that all methods are correctly evaluated. For instance, many NE methods require the input network to contain a single connected component (e.g., [4]) while some do not (e.g., [6]). 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 [9]. 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 [10]. This technique is commonly used in link prediction evaluation to generate the *negative class* for the binary classification of edges and non-edges. The negative

sampling approach and the relative sizes of the train and test non-edge sets vary between scientific studies. Yet another challenge in NE evaluation is that embedding methods provide different types of outputs. While some approaches only output node embeddings [11], others can provide node-pair embeddings [12] or even node similarities [4]. 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 embedding quality [3]. Finally, additional complexity stems from hyperparameter tuning, not only for the evaluated embedding methods, but also for the downstream task itself.

To address the above-mentioned challenges and simplify the evaluation of NE methods on downstream tasks, we introduce EvalNE. The toolbox can be used both as a standalone application or integrated in existing software.

**EvalNE as a command line tool** EvalNE leverages configuration files that allow the user to describe complete experimental setups, 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, detailed performance reports and graphics are generated as well as log files detailing the issues encountered.

**EvalNE as an API** The toolbox provides access to a series of classes and functions implementing various components required for method evaluation on different downstream tasks (see Fig. 3.1). Users can combine and modify these building blocks to produce new evaluation pipelines. The library does not implement any NE method but provides the necessary accessories to conveniently evaluate any off-the-shelf approach. The framework does, however, implement a number of link prediction heuristics (see Section 3.3.2).

## 3.2 Related software

To the best of our knowledge, no other libraries provide similar levels of automation and flexibility as EvalNE for network embedding evaluation. A recently proposed framework [13], which draws inspiration from EvalNE, is geared towards evaluating embeddings on semantic tasks and is limited to node embedding approaches only. Other libraries such as `OpenNE` [14] and `GEM` [15] focus mainly on providing implementations of NE methods and present limited evaluation capabilities. Their evaluation pipelines are rigid and tailored specifically to the approaches they implement, particular datasets and small sets of downstream tasks (node classification in both cases and additionally link prediction for `GEM`). The

`Karateclub` [16] library is designed to aid users in coding their own evaluations. Finally, other initiatives such as the Open Graph Benchmark [17] collect challenging datasets for NE evaluation. The associated `OGB` Python package also offer support for data preprocessing and a set of evaluators specifically tailored to the benchmark datasets.

Unlike these frameworks, EvalNE can evaluate methods with different types of output, from node to node-pair embeddings and node-pair similarities. Our framework is not limited to a predefined set of networks or methods available in a particular library. Instead, it can evaluate any available approach (note that this includes all implementations in the above-mentioned libraries). EvalNE also does not require users to write their own evaluation pipelines nor significantly modify their code to fit a particular API.

## 3.3 Software description

EvalNE is available on GitHub [1] and is released under the MIT free software license. 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, macOS, and Microsoft Windows. EvalNE only depends on a small number of popular open-source packages and others such as `OpenNE` and `GEM` are optional and can provide implementations of different NE methods. The additional `EvalNE-GUI` package [2] provides a user interface for our framework and makes managing and running evaluations more intuitive. The toolbox documentation is automatically managed and hosted online by Read the Docs [3]. It provides detailed instructions on the installation and main functionalities as well as a range of examples for both command line and API use. Finally, the library also contains a set of pre-filled configuration files allowing users to reproduce the experimental link prediction evaluations of the approaches introduced in Chapter 2.

### 3.3.1 Software 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 Fig. 3.1, simplifies code maintenance and addition of new features and allows users evaluate methods with different output types (node embeddings, node-pair embeddings, node-pair similarities, etc.) on different downstream tasks.

[1] https://github.com/Dru-Mara/EvalNE
[2] https://github.com/Dru-Mara/evalne-gui
[3] https://evalne.readthedocs.io/en/latest/

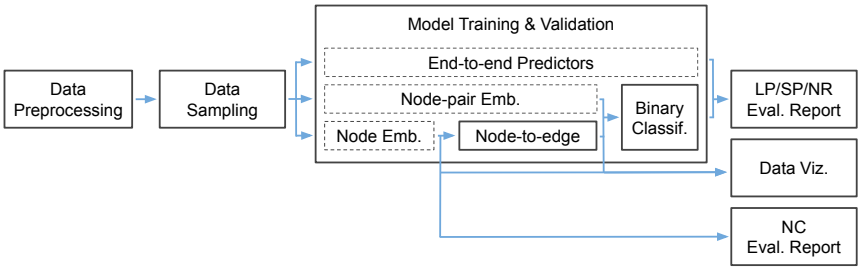



*Figure 3.1: Block diagram of EvalNE showing the methods and tasks that can be evaluated. Downstream link prediction, sign prediction, network reconstruction and node classification tasks are abbreviated as LP, SP, NR, and NC, respectively. Dashed blocks correspond to user-specified methods with different output types while the remaining blocks are provided by EvalNE.*

### 3.3.2 Software functionalities

The building blocks introduced in Section 3.3.1 are composed of a variety of classes and methods offering the following functionalities.

**Data preprocessing** Building on the popular `NetworkX` [18] framework, EvalNE offers additional functions for loading and storing networks, pruning and relabeling nodes, removing self-loops, sampling edges, restricting networks to the main connected component and obtaining common statistics.

**Data sampling** For evaluation, sets of train, validation, and test 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 already discussed in Section 2.3.1.1. The *EvalSplit* class encapsulates all 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 via grid search and collection of results. For sign and link 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 formal definitions in Section 2.3.2).

**Evaluation report & visualization** The proposed framework can evaluate method scalability through wall clock time, and performance through fixed-threshold metrics and threshold curves. The library implements over 10 different fixed-threshold metrics including AUC, precision, recall, and F-score. Integrated threshold curves, which present method performance for a range of threshold values, include precision-recall [19] and ROC [20] curves. Methods to visualize embeddings and graphs are also provided as well as dimensionality reduction techniques to map higher-dimensional embeddings to 2D. Method specific performance is recorded in *Results* objects while global evaluation summaries are provided through *Scoresheet* objects.

**Baseline heuristics** Specifically for the link prediction task, the toolbox provides a set of heuristic methods for both directed and undirected networks. These heuristics are based on: i) node-pair similarities, i.e., common neighbors, Jaccard coefficient, Adamic-Adar index, preferential attachment, resource allocation, cosine similarity, Leicht-Holme-Newman index and topological overlap, ii) paths, i.e., Katz similarity, and iii) embeddings, i.e., all_baselines (the concatenation of five heuristics as an embedding).

## 3.4 Illustrative examples

In this section, we present two examples that showcase the use of EvalNE as a command line application and as a Python API.

### 3.4.1 Command line example

When using EvalNE as a command line application a configuration file describing the evaluation setup is required. An example of one such file is presented in Appendix A.1.1. The evaluation parameters in the configuration file are grouped into 8 categories or sections also described in Appendix A.1.1. These sections cover general task-related parameters, data ingestion and preprocessing, edge sampling, methods to evaluate and results reporting.

Once the configuration file is filled, an evaluation can be started using the following command: `python -m evalne <config_file>.ini`. The toolbox will automatically log issues such as failed method executions, and provide a

tabular output and two pickle files containing the train and test evaluation scoresheets after execution.

### 3.4.2 API example

A code snippet showing the use of EvalNE as an API to evaluate five different methods on downstream link prediction is presented in Appendix A.1.2. In this example, we first read and preprocess a new dataset, crate an *LPEvalSplit* object that will contain the train and test edges and non-edges and initialize a link prediction evaluator. Then, we create a scoresheet object to store the evaluation results and select three baseline heuristics and two NE methods to be evaluated. Finally, we launch the evaluation and present the results.

## 3.5 Impact

Recent research has found that progress is stalling in many areas within AI [21–23], and that at the core of this phenomenon is the so called reproducibility crisis. Tools such as EvalNE and others inspired by it e.g., [13], have the potential to minimize this crisis by enabling reproducible and consistent evaluations and large-scale benchmarks. One such large-scale evaluation using EvalNE has already been conducted, specifically for the link prediction task [9]. Moreover, the ability to use configuration files to describe rich experimental setups presents two particular benefits. On the one hand, evaluations are simplified as many repetitive tasks such as data preprocessing or hyperparameter tuning are automated. On the other hand, ensuring that other practitioners can replicate an evaluation, reduces to making the configuration files accessible. The authors of [24] and [2] have already explored these benefits when conducting a link prediction evaluation on large-scale networks and a sign prediction evaluation, respectively. We also note that the software has attracted some attention in other research areas with BSc and MSc thesis in the fields of Mathematics [25], Biology [26] and Computer Science [27] making use of its capabilities. Finally, the user base of EvalNE has been continuously growing for the past months as shown by several popularity indicators of our GitHub repository.

## 3.6 Conclusions

In this chapter we have introduced EvalNE, a Python toolbox for consistent and reproducible evaluation of network embedding methods. The toolbox allows users to compare NE methods and perform benchmarks on a variety of downstream tasks using automated yet highly flexible evaluation pipelines. In the near future, we expect to broaden the impact of our framework with support for additional tasks

such as clustering and node classification through link prediction, direct integration with libraries providing NE method implementations and extended visualization capabilities.

## Acknowledgment

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] S. Wang, J. Tang, C. Aggarwal, Y. Chang, and H. Liu. *Signed network embedding in social media*. In Proc. of SDM, pages 327–335, 2017. doi:10.1137/1.9781611974973.37.

[2] A. Mara, Y. Mashayekhi, J. Lijffijt, and T. de Bie. *CSNE: Conditional Signed Network Embedding*. In Proc. of CIKM, pages 1105–1114, 2020. doi:10.1145/3340531.3411959.

[3] A. Grover and J. Leskovec. *node2vec: Scalable feature learning for networks*. In Proc. of KDD, pages 855–864, 2016. doi:10.1145/2939672.2939754.

[4] B. Kang, J. Lijffijt, and T. De Bie. *Conditional Network Embeddings*. In Proc. of ICLR, 2019.

[5] D. Wang, P. Cui, and W. Zhu. *Structural deep network embedding*. In Proc. of KDD, pages 1225–1234, 2016. doi:10.1145/2939672.2939753.

[6] Z. Zhang, P. Cui, X. Wang, J. Pei, X. Yao, and W. Zhu. *Arbitrary-Order Proximity Preserved Network Embedding*. In Proc. of KDD, pages 2778–2786, 2018. doi:10.1145/3219819.3219969.

[7] B. Perozzi, R. Al-Rfou, and S. Skiena. *DeepWalk: Online Learning of Social Representations*. In Proc. of KDD, pages 701–710, 2014. doi:10.1145/2623330.2623732.

[8] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. *LINE: Large-scale information network embedding*. In Proc. of WWW, pages 1067–1077, 2015. doi:10.1145/2736277.2741093.

[9] A. Mara, J. Lijffijt, and T. De Bie. *Benchmarking network embedding models for link prediction : are we making progress?* In Proc. of DSAA, pages 138–147, 2020. doi:10.1109/DSAA49011.2020.00026.

[10] B. Kotnis and V. Nastase. *Analysis of the Impact of Negative Sampling on Link Prediction in Knowledge Graphs*. CoRR, abs/1708.06816, 2017.

[11] A. Tsitsulin, D. Mottin, P. Karras, and E. Müller. *VERSE: Versatile Graph Embeddings from Similarity Measures*. In Proc. of WWW, pages 539–548, 2018. doi:10.1145/3178876.3186120.

[12] W. Song, S. Wang, B. Yang, Y. Lu, X. Zhao, and X. Liu. *Learning node and edge embeddings for signed networks*. Neurocomputing, 319:42–54, 2018. doi:10.1016/j.neucom.2018.08.072.

[13] M. A. Pellegrino, M. Cochez, M. Garofalo, and P. Ristoski. *A Configurable Evaluation Framework for Node Embedding Techniques*. In Proc. of ESWC Satellite Events, pages 156–160, 2019. doi:10.1007/978-3-030-32327-1_31.

[14] C. TU, C. YANG, Z. LIU, and M. SUN. *Network representation learning: an overview*. SCIENTIA SINICA Informationis, 47(8):980–996, 2017. doi:10.1360/N112017-00145.

[15] P. Goyal and E. Ferrara. *GEM: A Python package for graph embedding methods*. Journal of Open Source Software, 3(29):876, 2018. doi:10.21105/joss.00876.

[16] B. Rozemberczki, O. Kiss, and R. Sarkar. *Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs*. In Proc. of CIKM, pages 3125–3132, 2020. doi:10.1145/3340531.3412757.

[17] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec. *Open Graph Benchmark: Datasets for Machine Learning on Graphs*. CoRR, abs/2005.00687, 2020.

[18] A. A. Hagberg, D. A. Schult, and P. J. Swart. *Exploring Network Structure, Dynamics, and Function using NetworkX*. In Proc. of Python in Science Conference, pages 11 – 15, 2008.

[19] R. N. Lichtenwalter and N. V. Chawla. *Link Prediction: Fair and Effective Evaluation*. In Proc. of ASONAM, pages 376–383, 2012. doi:10.1109/ASONAM.2012.68.

[20] T. Fawcett. *ROC Graphs: Notes and Practical Considerations for Researchers*. Technical report, Personal communication, 2004.

[21] M. Hutson. *Core progress in AI has stalled in some fields*. Science, 368(6494):927–927, 2020. doi:10.1126/science.368.6494.927.

[22] M. F. Dacrema, P. Cremonesi, and D. Jannach. *Are We Really Making Much Progress? A Worrying Analysis of Recent Neural Recommendation Approaches*. In Proc. of RecSys, page 101–109, 2019. doi:10.1145/3298689.3347058.

[23] D. Blalock, J. J. G. Ortiz, J. Frankle, and J. Guttag. *What is the State of Neural Network Pruning?* In Proc. of MLSys, pages 129–146, 2020.

[24] F. Adriaens, A. Mara, J. Lijffijt, and T. De Bie. *Block-Approximated Exponential Random Graphs*. In Proc. of DSAA, pages 70–80, 2020. doi:10.1109/DSAA49011.2020.00019.

[25] K. N. Petro-de Chalendar and A. Benczúr. *Node embedding algorithms and their evaluation through link prediction*. Technical report, Eotvos Lorand University, 2020.

[26] P.-P. Strybol. *Using Deep Learning Approaches to Unveil Key Biological Processes in Human Tumours*. Technical report, Ghent University, 2019.

[27] S. Mbungu and M. Saerens. *Graph embedding with application to semi-supervised classification, visualization, reconstruction and neighborhood preservation tasks: an experimental comparison*. Technical report, Ecole polytechnique de Louvain, 2019.

# 4

# Performance Evaluation of Network Representations

*One of the main challenges discussed in Chapters 1 and 2 is the lack of benchmark studies summarizing and comparing different representation learning approaches. In this chapter we present one such empirical study. Our extensive evaluation covers two downstream tasks, link prediction and network reconstruction as well as the impact on performance of a range of design choices commonly adopted in the literature. The experimental evaluation is performed using the EvalNE framework proposed in Chapter 3. This chapter is based on the manuscript referenced below. Some section headlines, tables and figures have been altered to ensure consistency across thesis chapters.*

⋆ ⋆ ⋆

**A. Mara, J. Lijffijt, and T. De Bie.** ***An Empirical Evaluation of Network Representation Learning Methods.***

**Big Data, Mar. 2022.**

**Abstract** Network representation learning methods map network nodes to vectors in an embedding space that can preserve specific properties and enable traditional downstream prediction tasks. The quality of the representations learned is then generally showcased through results on these downstream tasks. Commonly

used benchmark tasks such as link prediction or network reconstruction, however, present complex evaluation pipelines and an abundance of design choices. This, together with a lack of standardized evaluation setups can obscure the real progress in the field. In this chapter, we aim to investigate the impact on performance of a variety of such design choices and perform an extensive and consistent evaluation that can shed light on the state-of-the-art on network representation learning. Our evaluation reveals that only limited progress has been made in recent years, with embedding-based approaches struggling to outperform basic heuristics in many scenarios.

## 4.1 Introduction

Networks are data representations which arise naturally in many fields from social sciences to medicine, biology or computer science [1–4]. They model entities as graph nodes and their relations as edges. These particular structures, while highly expressive, do not directly support traditional machine learning tasks such as classification, regression or clustering. In this context, network representation learning or network embedding (NE) methods learn mappings from network nodes to low dimensional vectors of an embedding space [5–9]. The obtained vector representations generally preserve certain network properties and constitute convenient transformations that support downstream prediction and inference tasks.

To assess the quality of the learned representations, downstream tasks such as data visualization [10, 11], node classification [12, 13], link prediction [14, 15] and network reconstruction [8, 16] are evaluated. Among these tasks, data visualization is an effective evaluation tool that is limited to small sized networks. Node classification is a relatively straightforward task that can be performed on larger networks and, thus, has been extensively studied in recent literature [17–19]. Link prediction and network reconstruction, on the other hand, have received far less attention. The main reason for this is their need for more complex evaluation pipelines with several preprocessing steps and design choices that can confound the results and are prone to errors. A recent empirical study [20] does, however, consider the link prediction task. In this study the authors unify various representative NE models under the same framework and aim to explain their differences using a small set of experimental setups.

In this chapter, we focus on link prediction[1], where we aim to estimate the likelihood of the existence of edges between given node-pairs not connected in the input graph, and on network reconstruction, which amounts to predicting the edges

[1]Note that *link reconstruction* would be a more appropriate term for referring to the task of predicting missing edges of a static network. *Link prediction*, on the other hand, should specifically denote the prediction of new links in time evolving networks. This distinction, however, is rarely made in the representation learning literature where the latter term is used to refer to predictions in both static and dynamic networks.

of a given network from its node embeddings. We also cover a broad range of experimental setups and implementations of unsupervised NE methods in a transductive scenario. The two evaluated tasks are strongly related and presents similar challenges as we discuss next:

*1) Data preprocessing:* To evaluate the generalization performance of NE methods, specifically for link prediction, sets of train and test/validation edges (i.e connected node-pairs) are required. These sets can be obtained using different approaches that might impact the evaluation results. For instance, a typical assumption is that the train edges span a (train) sub-network of a more complete (test) network. Thus, a principled approach is to use snapshots of the network at two different points in time. The first snapshot is used for training and the difference between the two snapshots for testing/validation [2–4]. Unfortunately, networks with such a temporal component are scarce, and therefore authors ordinarily resort to *sampling edges* from a network as test/validation examples and using the remaining edges for training [9, 15, 21, 22]. The sampling process varies between scientific works in different aspects. The sizes of the train and test/validation sets varies from 50-50 [15, 21], to 60-40 [9] or 80-20 [22]. The algorithms used to generate these splits also differ and while some aim to construct training networks with similar, scaled-down, properties such as, e.g., node degrees [23], others generate training graphs that preserve the general topology of the original network [24].

*2) Prediction pipeline:* Different NE methods require different pipelines to obtain link predictions. While some embedding methods directly compute link probabilities [15, 16], for others, these need to be learned on top of the node embeddings. Two common approaches are: (i) interpreting some notion of similarity between two node embeddings (e.g., dot product) as the link probability and (ii) casting the problem as a binary classification task. The latter requires as a pre-step the computation of node-pair embeddings. Thus, an operator must be applied on the node embeddings to obtain node-pair representations that are, in turn, fed into a binary classifier to perform link prediction or network reconstruction. The choice of operator not only varies between scientific works [13, 21] but is, in some cases, completely overlooked in the experimental setup discussion [10, 22]. Moreover, many valid choices exist for the binary classifier, and classifier training requires—in addition to 'positive' examples or edges—'negative' samples or non-edges. These non-edges can also be selected using different strategies and be of varying sizes [25].

*3) Hyperparameter tuning:* When comparing new methods with the state-of-the-art, for the baseline methods the default parameter settings are often used [13, 16], yet care is taken in tuning the parameters of the introduced method. When the recommended default settings were informed by experiments on other graphs than those used in the study at hand, this can paint an unduly unfavorable picture of the

baseline methods.

*4) Evaluation metrics:* Finally, no consensus exists on which evaluation criteria should be used for comparing different methods. While some papers advocate for the use of precision@$Np$ for a range of $Np$ values [8, 10, 16], others use area under the ROC curve (AUC) [15, 21] or precision and recall at fixed thresholds [26].

These challenges have led to an inconsistency in evaluation procedures throughout papers. Hence, the practical performance of NE methods is poorly understood. Moreover, as several recent studies have found [27, 28], the inconsistency of evaluation procedures is a central issue that has stalled progress in many sub-fields within AI. As such, in this chapter we aim to clarify the impact of these design choices on the evaluation pipelines and ultimately on method performance. At the same time, we aim to shed light on the real state-of-the-art by means of a standard evaluation process.

**Contributions** We provide an in depth empirical analysis of the state-of-the-art on network representation learning. We focus on link prediction and network reconstruction and evaluate a total of 29 implementations of 23 different NE methods on 7 popular real-world networks. We study the impact on method performance of different evaluation pipelines and specific components such as hyperparameter tuning, embedding dimensionality, train set size, edge sampling strategy, node-pair embedding operator, and binary classifier. Finally, there are inevitable limitations to the scope of our evaluation, such as an exclusive focus on undirected and unweighted networks without attributes, moderate-sized networks, and a representative but finite set of evaluated methods. To alleviate these limitations, we based our evaluation on `EvalNE` [2] [24] a framework that ensures reproducibility of results and allows for direct extensions of our work in all the aforementioned aspects.

This chapter is an extension of [29]. The main difference is the extended experimental evaluation: we have included one additional downstream task (network reconstruction; Section 4.4.2, Section 4.5.7, Table 4.4, Fig. 4.10), one new heuristic baseline and six new NE methods (changes throughout), included a comparison of evaluation pipelines using dot product similarities, and single layer neural decoders against node-pair embeddings followed by binary classification (Section 4.5.6, Fig. 4.8), and deepened the analysis of the link prediction performance and the use of embedding dimensions by different approaches (Section 4.5.6, Section 4.5.8, Fig. 4.9).

The remainder of this chapter is organized as follows. In Section 4.2 we present the methods evaluated. Section 4.3 introduces the real-world datasets used. A detailed description of the evaluation setup is given in Section 4.4. In Section 4.5 we summarize our findings and finally, Section 4.6 concludes this chapter.

[2] https://github.com/Dru-Mara/EvalNE

## 4.2 Methods

In this section we present the NE methods (Section 4.2.1) and baseline heuristics (Section 4.2.2) evaluated, and discuss the specific implementations used and method hyperparameters tuned (Section 4.2.3).

Before continuing with method descriptions we first lay down some notation. We represent an undirected network or graph as $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with vertex set $\mathbf{V} = \{1, \ldots, N\}$ and edge set $\mathbf{E} \subseteq \binom{\mathbf{V}}{2}$. Edges or connected node-pairs are represented as unordered pairs $\{i, j\} \in \mathbf{E}$. Non-edges or disconnected pairs are represented as $\{i, j\} \in \mathbf{D}$. We refer to the sets of train node-pairs as $\mathbf{E}_{train}$ and $\mathbf{D}_{train}$ and to the test node-pair sets as $\mathbf{E}_{test}$ and $\mathbf{D}_{test}$. The adjacency matrix of a graph $\mathbf{G}$ is represented as $\mathbf{A} \in \{0, 1\}^{N \times N}$, with element at row $i$ and column $j$ equal to 1 if $\{i, j\} \in \mathbf{E}$ and to 0 otherwise. We use $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)$ with $\mathbf{X} \in \mathbb{R}^{\mathrm{N} \times \mathrm{d}}$ to denote a $d$-dimensional node embedding and $\Gamma(i)$ to refer to the neighborhood of node $i$.

### 4.2.1 Network embedding methods

NE methods can be broadly categorized into four classes according to the strategy used for learning node similarities. In this taxonomy we can distinguish methods based on the Skip-Gram model [30], matrix factorization, neural architectures, and probabilistic approaches. Next, we describe the methods from each family included in our evaluation.

#### 4.2.1.1 Methods based on Skip-Gram

These methods leverage the Skip-Gram model to compute node representations such that the posterior probability of observing a neighboring node in a random walk is maximized.

*DeepWalk* [12] is the first method to use language modeling inspired techniques for NE. It uses random walks with fixed transition probabilities to measure node similarities, and embeddings are derived using Skip-Gram approximated via hierarchical softmax.

*Node2vec* [21] is a generalization of DeepWalk which uses truncated random walks for node neighborhood exploration and the Skip-Gram model, approximated via negative sampling, for embedding generation. The random walk properties are controlled by a return parameter $p$ and an in-out parameter $q$.

*Struc2vec* [31] extracts structural information from graphs through node-pair similarities for a range of neighborhood sizes. This information is then summarized as a multi-layer weighted graph $\hat{\mathbf{G}}$. Subsequently, random walks on $\hat{\mathbf{G}}$ are used to generate the embeddings.

*Metapath2vec* [11] is designed to learn embeddings from heterogeneous networks. The authors extend the concept of random walks to account for nodes of different types and use a heterogeneous Skip-Gram model to learn the embeddings.

*LINE* [6] uses joint and conditional probability distributions to model the first and second order adjacencies between linked nodes in $\mathbf{G}$. Embeddings are then obtained by maximizing the postulated objective function.

*VERSE* [13] learns embeddings by training a single layer neural network that minimizes the Kullback-Leibler (KL) divergence between node similarities in $\mathbf{G}$ and vector similarities in $\mathbf{X}$. Noise Contrastive Estimation is used for scalability.

*Watch Your Step (WYS)*[3] [32] is an attention model learned on the power series of the transition matrix of $\mathbf{G}$. The importance of different terms in the power series is parameterized by a probability distribution learned from data via backpropagation.

#### 4.2.1.2 Methods based on matrix factorization

These approaches use representations of node similarities such as high-order proximities expressed as polynomials of $\mathbf{A}$, the incidence matrix, Katz similarity or the graph Laplacian. Node embeddings are then obtained by factorizing the selected matrix.

*Graph Factorization (GF)* [33] uses regularized Gaussian matrix factorization to recover a matrix $\mathbf{Z}$ such that $\mathbf{Z}\mathbf{Z}^T$ is close to $\mathbf{A}$ in terms of observed non-zeros.

*GraRep* [7] is based on factorization of different polynomials of the adjacency matrix $\mathbf{A}$ which identify relations between nodes in $\mathbf{G}$ at different resolutions.

*HOPE* [8] preserves high-order proximities in graphs and accounts for the asymmetric transitivity property of directed networks.

*Laplacian Eigenmaps (LE)* [5] constructs a weighted representation $\hat{\mathbf{A}}$ of $\mathbf{A}$ by leveraging first order proximities on $\mathbf{G}$. The Laplacian $\mathbf{L}$ computed from $\hat{\mathbf{A}}$ is then factorized to obtain node embeddings.

*Locally Linear Embeddings (LLE)* [34] operates under the assumption that nodes and their neighbors lie on locally linear patches of a high-dimensional manifold. The embedding of a node can, thus, be derived from the linear coefficients that better reconstruct the node from the embeddings of its neighbors.

*FREDE* [35] uses Personalized PageRank as a node similarity matrix and adapts the Frequent Directions sketching algorithm [36] to produce embeddings.

*NetMF* [37] factorizes the DeepWalk transition probability matrix via SVD to reach the global optimum. To address the method's limited scalability, the authors have proposed an approximation method in [38].

*M-NMF* [39] incorporates community structure information in the embedding learning process via modularized non-negative matrix factorization.

[3] https://github.com/benedekrozemberczki/AttentionWalk

*AROPE* [16], similarly to GraRep, proposes embeddings as found by the truncated singular value decomposition of polynomials of $\mathbf{A}$. The authors describe a fast eigen-decomposition for these polynomials based on shifting or reweighing the decomposition of $\mathbf{A}$.

#### 4.2.1.3 Methods based on neural networks

Neural approaches return as embeddings the latent representations learned by deep layers of the networks. In this category we include models with a wide range or architectures, from shallow to deep models and from *classical* convolutional networks to graph convolutional approaches.

*Structural Deep Network Embedding (SDNE)* [10] uses a deep neural network for learning embeddings that capture first and second order graph proximities.

*PRUNE* [22] relies on a deep siamese neural network for learning node embeddings and can incorporate node ranking as additional information.

*GAE/VGAE* [14] are based on the variational auto-encoder model and compute embeddings from latent variables. The inference model is parameterized by a two layer graph convolutional network and the generative model computes inner products of latent representations. VGAE is a non-probabilistic variant of GAE.

*Graph Isomorphism Network (GIN)*[4] [40] is a neural model based on the Weisfeiler-Lehman test [41] that achieves maximum discriminative power among graph neural networks.

*GatedGCN* [42] belongs to the subclass of message passing graph convolutional approaches where node representations are learned by performing local updates only. The model incorporates edge gating mechanisms into vanilla graph convolutional networks.

#### 4.2.1.4 Probabilistic approaches

These methods use probabilistic approaches to model node similarities and learn embeddings.

*CNE* [15] uses a Bayesian approach to generate embeddings that model the observed network while taking prior information into account. The prior can incorporate structural graph properties such as node degrees or block densities for clustered or multi-partite networks.

### 4.2.2 Baseline heuristics

In addition to the NE approaches we evaluate a set of heuristics as baselines. Among these, we include heuristics that derive similarity scores —which can be interpreted as link probabilities— from the neighborhoods $\Gamma(i)$ and $\Gamma(j)$ of nodes

[4] https://github.com/graphdeeplearning/benchmarking-gnns

*Table 4.1: Implementation used and hyperparameters tuned for each method. Except for AROPE, CNE, GAE, VGAE, GIN, GatedGCN and the heuristics, the edge embedding operator is also tuned as a hyperparameter.*

| Methods | Implementations | Hyperparameters tuned |
|---|---|---|
| Heuristics | `EvalNE` | - |
| DeepWalk | Orig., `OpenNE` | $num\_walks = walk\_len = [5, 10, 20, 40, 80]$, $window\_size = [5, 10, 20]$ |
| Node2vec | Orig., `OpenNE` | $num\_walks = walk\_len = [5, 10, 20, 40, 80]$, $window\_size = [5, 10, 20]$, $p = q = [0.5, 1, 2]$ |
| Struc2vec | Orig. | $num\_walks = walk\_len = [5, 10, 20, 40, 80]$, $window\_size = [5, 10, 20]$ |
| Metapath2vec | Orig. | $\alpha = [0.01, 0.025]$, $negative = [5, 10]$ |
| LINE | Orig. `OpenNE` | $\rho = [0.01, 0.025]$, $negative\_ratio = [5, 10]$ |
| VERSE | Orig. | $nsamples = [3, 5, 10]$ |
| WYS | Other | $lr = [0.01, 0.05]$, $num\_walks = [20, 40, 80]$, $window\_size = [5, 10, 20]$ |
| GF | `OpenNE` | - |
| GraRep | `OpenNE` | $kstep = [2, 4, 8]$ |
| HOPE | `OpenNE`, `GEM` | $\beta = [0.1, 0.01, 0.001, 0.0001]$ |
| LE | `OpenNE`, `GEM` | - |
| LLE | `GEM` | - |
| FREDE | Orig. | $log\_transform = [log, add, max]$ |
| NetMF | Orig. | $window\_size = [1, 10]$ |
| M-NMF | Orig. | $clusters = [10, 20, 50]$ |
| AROPE | Orig. | $weights = [[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1], [1, 0.1, 0.01, 0.001], [1, 0.5, 0.05, 0.005]]$ |
| SDNE | `OpenNE`, `GEM` | $\beta = [2, 5, 10]$, $encoder\_list = [[128], [512, 128], [1024, 512, 128]]$ |
| PRUNE | Orig. | $\lambda = [0.01, 0.05]$ |
| GAE | Orig., | - |
| VGAE | Orig., | - |
| GIN | Other, | $lr = [0.01, 0.001]$, $step\_size = [50, 100]$ |
| GatedGCN | Orig., | $lr = [0.01, 0.001]$, $step\_size = [50, 100]$ |
| CNE | Orig. | $lr = [0.01, 0.05]$ |

pairs $\{i, j\}$. Specifically, we evaluate Common Neighbors defined as: $CN_{\{i,j\}} = |\Gamma(i) \cap \Gamma(j)|$; Jaccard Coefficient, $JC_{\{i,j\}} = |\Gamma(i) \cap \Gamma(j)|/|\Gamma(i) \cup \Gamma(j)|$; Adamic-Adar Index, $AA_{\{i,j\}} = \sum_{k \in \Gamma(i) \cap \Gamma(j)} 1/\log|\Gamma(k)|$; Resource Allocation Index, $RA_{\{i,j\}} = \sum_{k \in \Gamma(i) \cap \Gamma(j)} 1/|\Gamma(k)|$; and Preferential Attachment, $PA_{\{i,j\}} = |\Gamma(i)| \cdot |\Gamma(j)|$. We also include a method based on paths, i.e. Katz similarity. This method is defined as $Katz_{\{i,j\}} = \sum_{l=1}^{\infty} \beta^l \cdot |\rho(i, j|l)|$ where $\beta \in [0, 1]$ is a damping factor and $|\rho(i, j|l)|$ represents the number of paths of length $l$ from node $i$ to node $j$.

Lastly, we generate a 'heuristics embedding' by concatenating CN, JC, AA, RA, and PA as a 5-dimensional node-pair embedding. Logistic regression is then used to obtain link predictions. We will refer to this method as *NE_heuristics*.

### 4.2.3 Implementations and hyperparameters

Regarding the code used in our evaluation, we primarily rely on reference implementations by the original authors of different manuscripts. In addition to these, we also use code from other sources such as the extensively used `OpenNE` [43] and `GEM` [19] libraries. Table 4.1 summarizes the implementations used and hyperparameters tuned.

In the table all heuristics are summarized in a single *Heuristics* field and have no tunable hyperparameters (for Katz we fix $\beta = 0.001$ throughout our evaluation). Detailed hyperparameter descriptions are provided in the supplementary material [5]. For all methods where this is relevant, the node-pair embedding operator is tuned as an additional hyperparameter (see Section 4.4). The evaluation metrics used and hyperparameter tuning strategy are task dependent and, thus, are discussed later in Section 4.4.1–4.4.3.

Lastly, some notable differences between method implementations and important method specifics are: for LINE, the original implementation uses second order proximity by default (LINE-2) while the `OpenNE` implementation uses a concatenation of 1st and 2nd orders (LINE-1+2). For HOPE, the `OpenNE` implementation uses Common Neighbors as a similarity matrix which is then factorized, while the `GEM` implementation uses Katz similarity. For VERSE the reference implementation provides an unsupervised version of the method with Personalized PageRank as similarity metric. In NetMF, context window sizes of 1 and 10 correspond to evaluating NetMF-small and NetMF-large, respectively. Finally, for GIN and GatedGCN end-to-end link prediction is performed by adding a new layer to each network trained to minimize the cross-entropy between the logits of this layer and the ground truth labels (existence or not of an edge).

## 4.3 Datasets

We conduct our experimental evaluation on 7 undirected real-world networks from different domains. These networks are medium-sized to ensure successful execution of all methods and constrain the computational resources needed. Next, we present a short description of each network and in Table 4.2, we summarize their main statistics.

*StudentDB* [44] represents a snapshot of Antwerp University's relational student database. Nodes in the network represent entities such as students, professors, tracks, etc. and edges constitute binary relations, i.e. student-in-track, student-in-program, student-take-course, professor-teach-course, course-in-room. *Facebook* [45] and *BlogCatalog* [46] are online social networks where nodes represent

[5] The supplemental PDF file can be found in our GitHub repository: https://github.com/aida-ugent/NRLBenchmark.

*Table 4.2: Summary of dataset statistics where $2|\mathbf{E}|/|\mathbf{V}|$ denotes the average node degrees of the undirected networks.*

| Dataset | Category | $\lvert\mathbf{V}\rvert$ | $\lvert\mathbf{E}\rvert$ | $2\lvert\mathbf{E}\rvert/\lvert\mathbf{V}\rvert$ |
|---|---|---|---|---|
| StudentDB | Relational | 395 | 3,423 | 17.33 |
| Facebook | Social | 4,039 | 88,234 | 43.69 |
| BlogCatalog | Social | 10,312 | 333,983 | 64.77 |
| GR-QC | Collaboration | 4,158 | 26,844 | 6.45 |
| AstroPH | Collaboration | 18,772 | 396,160 | 22.00 |
| PPI | Biological | 3,852 | 37,841 | 19.64 |
| Wikipedia | Language | 4,777 | 92,295 | 38.64 |

different users and edges indicate friendships. *GR-QC* [45] and *AstroPh* [45] describe collaboration networks in the fields of General Relativity and Astrophysics. Nodes represent papers and edges denote citations between them. *PPI* [47] is a biological protein-protein interaction network and constitutes a subset of the Homo Sapiens PPI network. Finally, *Wikipedia* [48] contains nodes representing words in Wikipedia pages and links denoting co-occurrences.

## 4.4 Experimental setup

In this section we give details on the link prediction and network reconstruction tasks, present the experimental setups used and discuss the limitations to the scope of the evaluation and reproducibility of results.

### 4.4.1 Link prediction

As pointed out in Section 4.1 the objective in link prediction is to identify missing links in an incomplete graph $\mathbf{G}$. Thus, the first step for evaluation is to preprocess $\mathbf{G}$ and obtain sets of train and test node-pairs. For our main experiments we use the standard approach of generating an incomplete training graph $\mathbf{G}_{train} = (\mathbf{V}, \mathbf{E}_{train})$ from a more complete graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ where the connected node-pairs $\{i, j\} \in \mathbf{E} \setminus \mathbf{E}_{train}$ are used for testing. The proportion $f = |\mathbf{E}_{train}|/|\mathbf{E}|$ (or train fraction) is a user-defined parameter.

For obtaining $\mathbf{E}_{train}$ and $\mathbf{E}_{test}$ we evaluate 3 approaches, namely *random* [23], *spanning tree (ST)*[6] [24] and *depth first tree (DFT)*. The first starts by randomly sampling a set of 'preliminary' test edges ($\mathbf{E}_{test}$). The main connected component of the remaining graph is then computed and all edges in it are considered train edges ($\mathbf{E}_{train}$). Finally, the test set is refined such that $\{i, j\} \in \mathbf{E}_{test}$

[6]This approach is similar to Algorithm 1 introduced in Section 2.3.1.1 with the exception that in step 1, a more efficient approach for sampling a spanning tree, i.e. Wilson's Algorithm, is used.

only if $i \in \mathbf{V}_{train}$ and $j \in \mathbf{V}_{train}$. The ST and DFT approaches are very similar. These construct a spanning tree of $\mathbf{G}$ and add randomly selected edges until the desired size of $\mathbf{E}_{train}$ is reached. All remaining edges are used for testing ($\mathbf{E}_{test}$). The main difference between these methods lies in the approach taken for computing the spanning tree. In the case of ST, this is done by selecting a spanning tree uniformly at random from all possible ones, while in DFT a faster depth first search is used.

In addition to splitting connected pairs, we also generate sets of train and test non-edges, $\mathbf{D}_{train}$ and $\mathbf{D}_{test}$. The node-pairs in these sets are randomly selected using an open world assumption where any pair $\{i, j\} \notin \mathbf{E}_{train}$ is considered a valid train non-edge. Test non-edges are selected as pairs $\{i, j\} \notin (\mathbf{E} \cup \mathbf{D}_{train})$. In our experiments, we set $|\mathbf{D}_{train}| = |\mathbf{E}_{train}|$ and $|\mathbf{D}_{test}| = |\mathbf{E}_{test}|$.

For hyperparameter tuning the train sets need to be further split into train and validation. We fix their proportions to 90% train and 10% validation in all our experiments and the validation split is always performed using the same algorithm as the initial train-test split. Grid search is adopted as the strategy for learning the best model hyperparameters.

For most methods, with the exception of CNE, AROPE, GAE, VGAE, GIN, GatedGCN and the heuristics, for which node-pair similarities are directly computed, the link predictions are learned through binary classification. First, node-pair embeddings for $\mathbf{E}_{train}$, $\mathbf{D}_{train}$, $\mathbf{E}_{test}$ and $\mathbf{D}_{test}$ need to be obtained from the node embeddings $\mathbf{X}$ learned by a method on $\mathbf{G}_{train}$. The embedding of a pair $\{i, j\}$ can be computed by applying different operators $\circ$ to the embeddings of the incident nodes $i$ and $j$ i.e. $\mathbf{x}_{\{i,j\}} = \mathbf{x}_i \circ \mathbf{x}_j$. In our evaluation, we use the operators introduced in [21], namely *Average* ($(\mathbf{x}_i + \mathbf{x}_j)/2$), *Hadamard* ($\mathbf{x}_i \cdot \mathbf{x}_j$), *Weighted* $L_1$ ($|\mathbf{x}_i - \mathbf{x}_j|$) and *Weighted* $L_2$ ($|\mathbf{x}_i - \mathbf{x}_j|^2$). A binary classifier is then fitted with the node-pair embeddings and labels $\{0, 1\}$ representing non-edges and edges, respectively.

### 4.4.2 Network reconstruction

Network reconstruction evaluates how accurately an embedding method captures the link structure of a network. In other words, it evaluates how well the adjacency matrix $\mathbf{A}$ of the input graph $\mathbf{G}$ can be recovered from the embeddings generated by an NE method. Similarly to the link prediction setting, we formulate network reconstruction as a binary classification problem where we expect connected node-pairs in $\mathbf{G}$ to be ranked higher than non-connected pairs. First, for each method and network we perform hyperparameter tuning via grid search. The best set of parameters is then used to compute node embeddings and node-pair embeddings are derived using the same operators $\circ$ as for link prediction. Finally, binary classification is performed and results are reported.

We note that for this task there are two fundamental differences with respect to the link prediction pipeline. First, since the objective is to determine how well a method captures the structure of a graph, training of the NE methods, training of the binary edge classifier, hyperparameter tuning and model assessment are all performed on the complete input graph $\mathbf{G}$ rather than on train and validation graphs, i.e., $\mathbf{G} = \mathbf{G}_{train}$ and $\mathbf{E} = \mathbf{E}_{train}$. Second, this task requires us to provide a prediction for all $(N \cdot (N-1))$ entries in $\mathbf{A}$ to determine if the probabilities of edges $\{i, j\} \in \mathbf{E}$ are higher that those of non-edges $\{i, j\} \in \mathbf{D}$. As this is unfeasible even for moderate size networks, we randomly sample $1\%$ of node-pairs ($0.1\%$ for AstroPH and BlogCatalog) and approximate the method performance by these samples. Note that for the sparse graphs considered in our evaluation, this random sampling results in many more non-edges than edges to predict and, thus, AUC would not be an adequate performance metric. In this case we report precision@$Np$ for a range of $Np$ values.

### 4.4.3 Evaluation setup

For link prediction we use two experimental setups, called LP1 and LP2, with model hyperparameter tuning as only difference. While in LP1 we tune all hyperparameters presented in Table 4.1 as well as the node-pair operator, in LP2 we use default recommended parameters for all methods while still tuning the node-pair embedding strategy. We quantify method performance in terms of AUC and also report execution times. For visualization purposes, we use $-\log(1 - AUC)$, as it better reflects differences in performance for AUC values $\approxeq 1$. Unless otherwise specified we use embedding dimensionality $d = 128$, train fraction $f = 0.8$, ST as edge sampling strategy and logistic regression (with 5-fold cross-validation for tuning the regularization parameter) as binary classifier. Finally, all methods are run for their pre-defined number of iterations.

Setup LP1 is used to study the performance of NE methods with respect to parameters $d \in \{8, 32, 128\}$ and $f \in \{0.2, 0.5, 0.8\}$. Setup LP2, on the other hand, is used to investigate the effect of the remaining pipeline components.

For network reconstruction we use a single experimental setup where method hyperparameters and node-pair embedding operators are tuned. We present method results for the best embedding dimension $d \in \{8, 32, 128\}$ and use logistic regression with 5-fold cross validation as binary classifier.

### 4.4.4 Reproducibility notes

In order to ensure the reproducibility of our results and foster further research in this area, we have based our experimental evaluation upon the `EvalNE` framework. This open source Python toolbox aims to simplify the evaluation of NE

methods for link prediction, network reconstruction, node classification, and visualization. The toolbox automates tasks such as hyperparameter tuning, selection of train and test edges or non-edge sampling. It implements widely used node-pair embedding operators and can incorporate any classifier for prediction. Moreover, its design ensures that common errors, such as the computation of features on $\mathbf{G}$ rather than just on $\mathbf{G}_{train}$, or other forms of label leakage, are ruled out. Finally, for maximum reproducibility, configuration files describing our complete evaluation setups and which can be used directly in `EvalNE` to replicate our results, are made available online [7].

## 4.5 Experimental results

In this section we present the results of our empirical study. First, we focus on link prediction and discuss performance in Section 4.5.1, the effect of hyperparameter tuning in Section 4.5.2, embedding dimensionality in Section 4.5.3, train-test splits in Section 4.5.4, edge sampling in Section 4.5.5 and finally prediction pipelines in Section 4.5.6. Then, in Section 4.5.7 we present the results for the network reconstruction task and in Section 4.5.8, the relation between method performance and network structure. All experiments were conducted on a machine equipped with two 12 Core Intel(R) Xeon(R) Gold processors and 256GB of RAM.

### 4.5.1 Link prediction performance

We start in Table 4.3 by presenting the best AUC scores for each method on the link prediction task. These results were obtained using setup LP1 where for each method we selected the embedding dimension $d \in \{8, 32, 128\}$ that resulted in the highest scores. GIN, GatedGCN and CNE performed best for $d = 8$ while all the remaining methods performed best for $d = 128$ (see Section 4.5.3 for a detailed discussion). In the table we group methods according to the taxonomy in Section 4.2 with the best performing method per group highlighted in bold and the overall best for each network on gray background. The last two columns show averages over the 7 evaluated networks of: i) the AUC performance and ii) the ranking of each method among all other approaches.

The results in Table 4.3 show the excellent performance, on most datasets, of the baseline heuristics and in particular of RA, Katz and the NE_heuristics. The latter, together with GraRep, achieves the highest average AUC amongst all methods of $0.963$. In addition, although NE_heuristics is not a top performer on any particular network, its AUC scores are consistently high across all networks and never more than $2\%$ lower than the best scores in each case. This effect is also reflected by the method's excellent average AUC rank ($2^{nd}$ overall). Amongst

[7] https://github.com/aida-ugent/NRLBenchmark

*Table 4.3: AUC scores and standard deviations over 3 experiment repetitions for setup LP1 where hyperparameters are tuned and $d = 128$ for all methods except GIN, GatedGCN and 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 gray 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±0.003 | 0.959±0.000 | 0.990±0.000 | 0.863±0.002 | 0.900±0.001 | 0.897±0.210 | 20.79 |
| JC | 0.630±0.011 | 0.990±0.001 | 0.770±0.001 | 0.959±0.000 | 0.990±0.002 | 0.839±0.005 | 0.624±0.001 | 0.829±0.260 | 26.93 |
| AA | 0.630±0.011 | 0.993±0.001 | 0.952±0.003 | 0.959±0.000 | 0.991±0.000 | 0.867±0.002 | 0.919±0.001 | 0.902±0.211 | 16.14 |
| PA | 0.922±0.008 | 0.842±0.003 | 0.955±0.002 | 0.839±0.001 | 0.878±0.001 | 0.905±0.001 | 0.920±0.006 | 0.894±0.041 | 18.93 |
| RA | 0.630±0.011 | **0.994±0.001** | **0.958±0.003** | 0.959±0.000 | 0.991±0.000 | 0.867±0.002 | **0.931±0.001** | 0.904±0.212 | 12.93 |
| Katz | 0.737±0.008 | 0.994±0.000 | 0.954±0.001 | **0.988±0.003** | **0.995±0.000** | 0.920±0.000 | 0.914±0.002 | 0.929±0.086 | 9.79 |
| NE_heuristics | **0.966±0.004** | 0.993±0.000 | 0.956±0.001 | 0.976±0.000 | 0.993±0.001 | **0.927±0.004** | 0.929±0.003 | **0.963±0.026** | **5.71** |
| DeepWalk | 0.906±0.005 | 0.990±0.000 | 0.943±0.001 | 0.986±0.000 | 0.984±0.000 | 0.905±0.002 | 0.903±0.001 | 0.945±0.040 | 15.57 |
| DeepWalk_opne | 0.906±0.010 | 0.991±0.000 | 0.943±0.001 | 0.985±0.000 | 0.983±0.000 | 0.905±0.001 | 0.904±0.002 | 0.945±0.039 | 15.43 |
| Node2vec | 0.948±0.009 | 0.994±0.000 | 0.938±0.006 | 0.985±0.001 | 0.989±0.001 | 0.840±0.002 | 0.893±0.003 | 0.941±0.054 | 16.29 |
| Node2vec_opne | 0.897±0.004 | 0.991±0.001 | 0.929±0.001 | 0.986±0.000 | 0.992±0.001 | 0.900±0.001 | 0.901±0.002 | 0.942±0.043 | 16.00 |
| Struc2vec | 0.933±0.010 | 0.833±0.004 | 0.953±0.002 | 0.842±0.001 | 0.874±0.001 | 0.904±0.001 | 0.918±0.005 | 0.894±0.042 | 20.57 |
| Metapath2vec | **0.981±0.005** | 0.942±0.003 | 0.948±0.003 | 0.803±0.002 | 0.858±0.000 | 0.880±0.001 | 0.903±0.006 | 0.902±0.058 | 22.29 |
| LINE | 0.963±0.004 | 0.993±0.001 | 0.931±0.002 | 0.984±0.000 | 0.991±0.000 | 0.877±0.002 | 0.882±0.002 | 0.946±0.048 | 15.14 |
| LINE_opne | 0.850±0.010 | 0.991±0.000 | 0.931±0.002 | 0.933±0.001 | 0.963±0.000 | 0.895±0.003 | 0.894±0.001 | 0.923±0.045 | 22.86 |
| VERSE | 0.935±0.010 | **0.994±0.001** | **0.956±0.002** | **0.990±0.000** | **0.996±0.002** | **0.919±0.002** | **0.918±0.002** | **0.959±0.033** | **5.14** |
| WYS | 0.819±0.016 | 0.940±0.003 | 0.915±0.005 | 0.833±0.004 | 0.855±0.002 | 0.853±0.010 | 0.864±0.008 | 0.868±0.042 | 30.00 |
| GF_opne | 0.868±0.007 | 0.983±0.000 | 0.897±0.004 | 0.933±0.001 | 0.947±0.001 | 0.837±0.003 | 0.834±0.005 | 0.900±0.054 | 28.14 |
| GraRep_opne | 0.969±0.003 | **0.993±0.000** | **0.962±0.001** | **0.984±0.000** | **0.990±0.001** | **0.921±0.001** | **0.922±0.002** | **0.963±0.029** | **6.14** |
| HOPE_gem | **0.989±0.001** | 0.990±0.000 | 0.955±0.002 | 0.952±0.001 | 0.950±0.000 | 0.909±0.001 | 0.919±0.002 | 0.952±0.029 | 13.00 |
| HOPE_opne | 0.914±0.002 | 0.989±0.000 | 0.944±0.005 | 0.920±0.000 | 0.947±0.000 | 0.872±0.001 | 0.915±0.005 | 0.929±0.034 | 21.43 |
| LE_gem | 0.906±0.010 | 0.992±0.000 | 0.800±0.005 | 0.975±0.000 | 0.934±0.003 | 0.760±0.005 | 0.767±0.003 | 0.876±0.097 | 24.64 |
| LE_opne | 0.906±0.011 | 0.992±0.000 | 0.802±0.003 | 0.977±0.002 | 0.932±0.005 | 0.764±0.006 | 0.771±0.001 | 0.878±0.092 | 24.14 |
| LLE_gem | 0.889±0.008 | 0.990±0.000 | 0.704±0.008 | 0.970±0.006 | 0.894±0.002 | 0.726±0.005 | 0.741±0.004 | 0.845±0.114 | 28.43 |
| FREDE | 0.815±0.005 | 0.747±0.000 | 0.627±0.001 | 0.604±0.001 | 0.548±0.002 | 0.621±0.002 | 0.910±0.002 | 0.696±0.125 | 32.71 |
| NetMF | 0.942±0.016 | 0.990±0.000 | 0.952±0.001 | 0.970±0.003 | 0.968±0.000 | 0.890±0.001 | 0.914±0.001 | 0.947±0.033 | 17.00 |
| M-NMF | 0.944±0.009 | 0.992±0.000 | 0.936±0.008 | 0.983±0.000 | 0.983±0.001 | 0.878±0.001 | 0.913±0.002 | 0.947±0.040 | 15.79 |
| AROPE | 0.982±0.002 | 0.991±0.001 | 0.955±0.001 | 0.968±0.000 | 0.967±0.001 | 0.910±0.002 | 0.918±0.001 | 0.956±0.029 | 12.50 |
| SDNE_gem | **0.987±0.004** | 0.979±0.002 | 0.952±0.002 | 0.945±0.001 | 0.971±0.000 | **0.909±0.001** | 0.918±0.002 | **0.952±0.028** | 15.14 |
| SDNE_opne | 0.985±0.002 | 0.987±0.000 | 0.953±0.005 | 0.957±0.002 | 0.969±0.000 | 0.898±0.001 | 0.917±0.007 | **0.952±0.032** | 15.71 |
| PRUNE | 0.901±0.010 | 0.838±0.002 | **0.956±0.003** | 0.836±0.001 | 0.874±0.000 | 0.903±0.001 | **0.920±0.003** | 0.890±0.042 | 20.14 |
| GAE | 0.955±0.012 | **0.993±0.001** | 0.943±0.005 | 0.977±0.003 | **0.991±0.001** | 0.856±0.002 | 0.913±0.002 | 0.947±0.047 | **14.36** |
| VGAE | 0.952±0.007 | 0.993±0.000 | 0.935±0.003 | **0.978±0.002** | 0.989±0.000 | 0.881±0.003 | 0.898±0.007 | 0.946±0.042 | 15.86 |
| GIN | 0.820±0.055 | 0.855±0.011 | 0.725±0.012 | 0.748±0.013 | 0.826±0.005 | 0.843±0.005 | 0.897±0.002 | 0.816±0.059 | 31.00 |
| GatedGCN | 0.713±0.045 | 0.823±0.040 | 0.756±0.014 | 0.776±0.005 | 0.812±0.008 | 0.852±0.008 | 0.871±0.036 | 0.800±0.054 | 32.43 |
| CNE | **0.946±0.009** | **0.994±0.000** | **0.967±0.001** | **0.980±0.000** | **0.976±0.001** | **0.928±0.001** | **0.922±0.000** | **0.959±0.026** | **6.93** |

Skip-Gram based approaches, VERSE is a clear top performer. GraRep is the matrix factorization-based method with highest scores and presents a consistent performance across networks. Nonetheless, AROPE and HOPE also exhibit good performance. The neural architecture-based approaches present similar performances with SDNE and GAE obtaining marginally higher scores.

Overall, according to their average AUCs and AUC ranks, the best performing methods are: VERSE, NE_heuristics, GraRep and CNE. It is worth noting that for these top performers we tuned at most one hyperparameter (see Table 4.2) and, thus, their excellent performance can not be ascribed to an exhaustive model selection process. Also noteworthy is the fact that CNE achieves state-of-the-art results with an 8-dimensional embedding as compared to the 128 dimensions required by other methods.

In the absence of additional data, such as node or edge attributes, our experiments suggest that performance generally improves as the order of proximity between nodes captured by a model increases (i.e. as a node's neighborhood includes vertices that are further and further away). GraRep, the best performing NE method, captures relations up to order 8. HOPE and AROPE, also top performers, capture relations up to order 4 while the remaining first and second order methods exhibit lower performance. Two interesting exceptions to this pattern are VERSE and CNE. In the case of VERSE, the second-order method employs a nonlinear transformation able to preserve more information from the original network. CNE, on the other hand, finds an embedding based on first-order information. Its excellent performance can be explained by the fact that it additionally models structural information (node degrees) in a prior, leaving more flexibility to the embedding.

Another important observation from the results in Table 4.3 is that method performance varies significantly between different implementations of the same NE methods (up to 11.7% for LINE on StundentDB). These differences are especially large for implementations of LINE (3.3% difference, on average, over all networks), HOPE (2.4%) and Node2vec (2%). The main factors causing these differences are numerical instability, approximation of computations and dependencies on different software versions with varying default parameters. In our supplementary material [8] we also show fluctuations in method performance of up to 44.5% for `GEM` implementations due to package dependencies and up to 5% for metapath2vec due to multi-core execution.

Also interesting is the difference in performance between Node2vec and DeepWalk. In this case, one would expect the former (which is a generalization of the latter) to perform consistently better. This would indeed be the case if both methods would approximate Skip-Gram via negative sampling [21]. In our experiments, however, the Skip-Gram is approximated via hierarchical softmax for DeepWalk (as proposed by the original authors) and via negative sampling for

[8] https://github.com/aida-ugent/NRLBenchmark

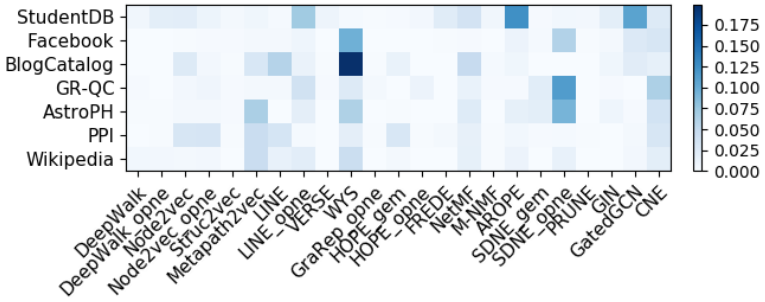



*Figure 4.1: Improvement in AUC of tuning model hyperparameters. Only methods with tuned parameters are shown.*

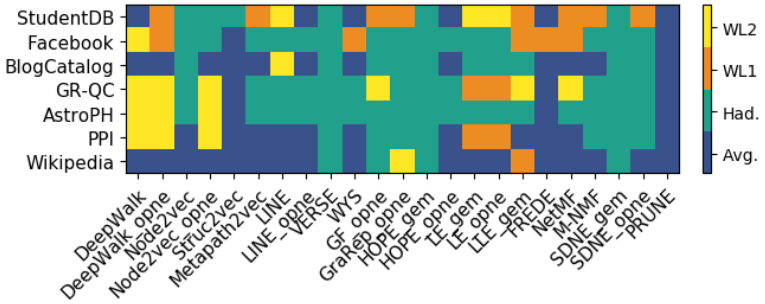



*Figure 4.2: Best performing node-pair embedding operators on setup LP1. Only methods for which the node-pair operator was tuned are shown.*

Node2vec (also as proposed by the authors), which explains the observed differences. Further experiments comparing the two Skip-Gram approximations for both methods reveal significant fluctuations in AUC of up to $0.02$ with a standard deviation of $0.02$. Moreover, we find that no approximation provides consistently better results than the other across all datasets. Ultimately, this experiment reveals the importance of accounting for the approximation used when comparing methods based on Skip-Gram.

### 4.5.2 Hyperparameter tuning

Fig. 4.1 presents a heatmap of the increment in link prediction accuracy obtained through hyperparameter tuning, i.e. the difference in AUC between setups LP1 and LP2. Only methods with tuned hyperparameters are shown in this figure. The results reveal limited improvements in most cases. For approximately 7% of the network-method combinations, however, we observe increments in AUC greater that 5%. WYS, SDNE and CNE benefit the most from hyperparameter tuning. Popular random walk methods such as DeepWalk and Node2vec, for which

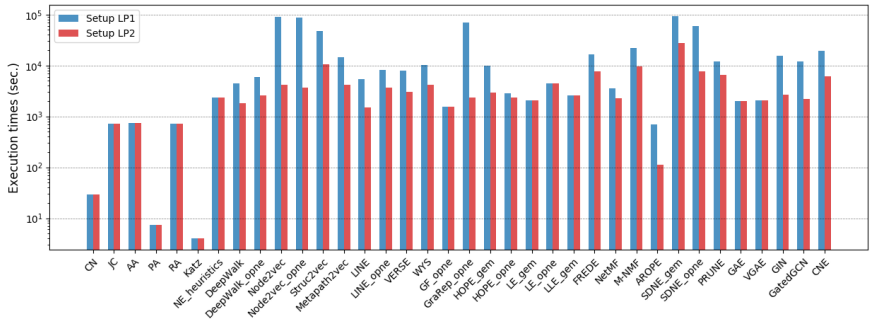



*Figure 4.3: Execution times in seconds of setups LP1 and LP2.*

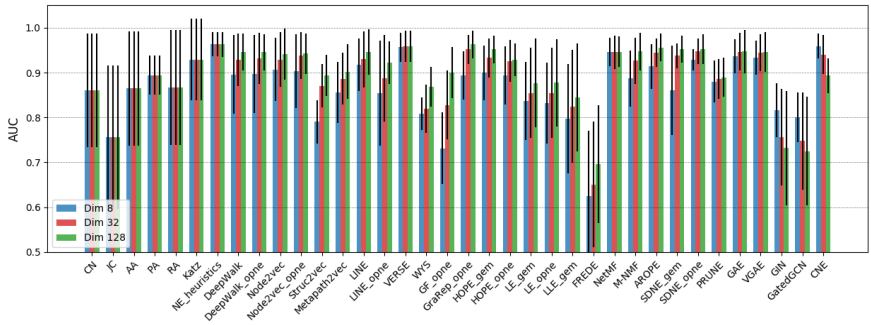



*Figure 4.4: Method performance (AUC) on setup LP1 for $d \in \{8, 32, 128\}$.*

parameter tuning is tedious, show minimal gains in AUC.

In Fig. 4.2 we present the node-pair embedding operator selected using hyperparameter tuning for each method and network on experimental setup LP1. The results indicate that Hadamard is the most frequently selected (in 71/161 cases) and that some methods prefer specific operators. This is: Hadamard in the case of VERSE, HOPE and SDNE and average for PRUNE. The remaining methods present a mix of operators. On average, over all methods and datasets we observe a difference in validation AUC between the best and the worst performing operator of $0.141$ with a standard deviation of $0.092$. This clearly highlights the need to tune the node-pair embedding operator as a method hyperparameter.

For each method, the sum of execution times on the 7 evaluated networks for experimental setups LP1 and LP2 are presented in Fig. 4.3. As expected, runtimes for LP1 are larger than those of LP2 and the differences are especially significant for the methods with more tuned hyperparameters e.g., Node2vec, Struc2vec, SDNE (as this implies computing an embedding of the validation graph for each combination of hyperparameters). Nevertheless, the highest increase in runtime can be found for GraRep, for which a single hyperparameter was tuned (the *k*-

*step*). This indicates that computing high order proximities in GraRep is a slow process and, as shown in Fig. 4.1, does not result in a large improvement in performance for *k-step*$> 4$.

We also observe from Fig. 4.3 that naive sequential implementations of the heuristics are still faster than heavily optimized and parallelized NE methods with only AROPE presenting comparable runtimes. Finally, with no hyperparameter tuning, similar patterns can be observed within each method family with factorization methods being the fastest.

### 4.5.3 Embedding dimensionality

We have evaluated the effect of embedding dimensionality on method performance by modifying setup LP1 and computing the AUCs of all methods for $d \in \{8, 32, 128\}$. In Fig. 4.4 we summarize the average AUC and standard deviation for each method and embedding dimension over all evaluated networks. The heuristics do not depend on $d$ but are shown for reference. These results show that most methods significantly improve in performance as $d$ increases. The opposite effect can be seen for GIN, GatedGCN and CNE, which may indicate that these methods are underfitting.

### 4.5.4 Train-test split size

We further used experimental setup LP1 to analyze the effect on method performance of the train fraction $f$. These results are summarized as three heatmaps in Fig. 4.5 showing performance as $-\log(1 - AUC))$ for $f \in \{0.2, 0.5, 0.8\}$. An interesting observation here is that for $f > 0.5$ most methods capture well the network structures while this is not the case for $f < 0.5$. This is reflected by an average increase in AUC over all methods of 6.7% between $f = 0.2$ and $f = 0.5$ while only a 1.9% difference can be observed between $f = 0.5$ and $f = 0.8$.

From this experiment we also observe that probabilistic approaches present the highest robustness to varying train set sizes followed by Skip-Gram, factorization, neural approaches and finally the heuristics. VERSE, GraRep, CNE and NetMF are the methods that best capture the network structure when only 20% of the edges are used for training. Lastly, regarding method runtimes, these increase by approximately 60% from evaluations with $f = 0.2$ to $f = 0.8$. Some notable exceptions are Metapath2vec, Struc2vec and SDNE_gem, for which the runtimes increase by 84.9%, 83.7% and 81.1% respectively.

### 4.5.5 Edge sampling

We also conducted an experiment to compare the three strategies introduced in Section 4.4.1, namely random, ST and DFT for splitting $\mathbf{E}$ into $\mathbf{E}_{train}$ and $\mathbf{E}_{test}$

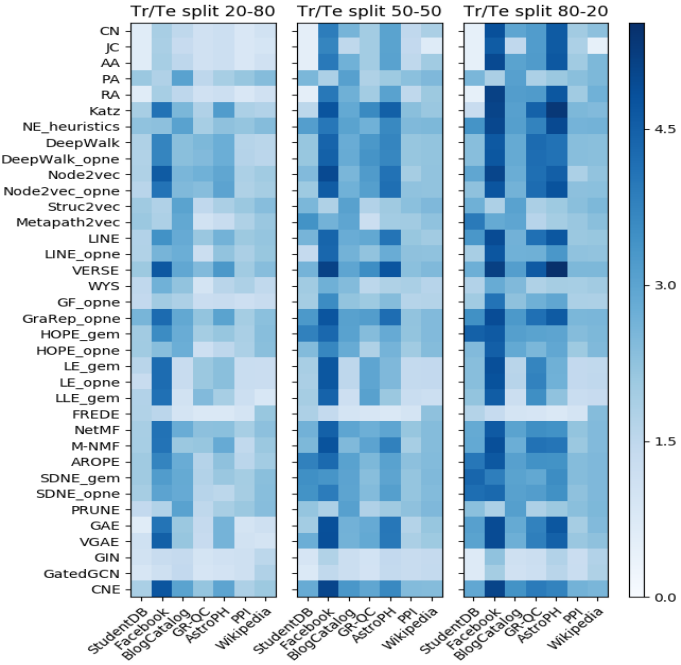



*Figure 4.5: Method performance as* $-\log(1-AUC)$ *on setup LP1 for train-test edge splits 20-80, 50-50 and 80-20. A darker color indicates better performance.*

and determine their impact on method performance. We used experimental setup LP2 where no method hyperparameters, apart from the node-pair embedding operator, were tuned. Our results show minimal differences in AUC between the three strategies. More precisely, the average AUC and standard deviation over all methods and datasets for random edge sampling is $0.894 \pm 0.114$, for the ST strategy is $0.897 \pm 0.114$ and for DFT is $0.894 \pm 0.109$.

For large networks edge sampling can become a bottleneck and, thus, sampling runtimes were also worth investigating. Our results, depicted in Fig. 4.6, show that the random strategy is the slowest followed by ST and finally, DFT which is up to one order of magnitude faster on specific networks. The slower runtimes of the random strategy are mainly due to set intersections to obtain the correct $\mathbf{E}_{train}$ and $\mathbf{E}_{test}$. In addition, we have found that the random edge split strategy does not preserve all nodes from the input graph $\mathbf{G}$ in the training graph $\mathbf{G}_{train}$. On average, over all datasets 2.5% of the nodes in $\mathbf{G}$ are lost. This effect is especially severe for networks with lower average degrees, such as StudentDB and GR-QC, which lose up to 8% of their nodes. The ST and DFT strategies, on the other hand, preserve all nodes.

We also compared the ST strategy to a split based on edge timestamps in or-

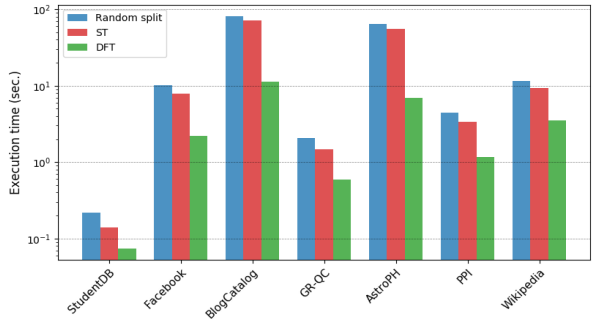



*Figure 4.6: Execution times of different train-test split algorithms.*

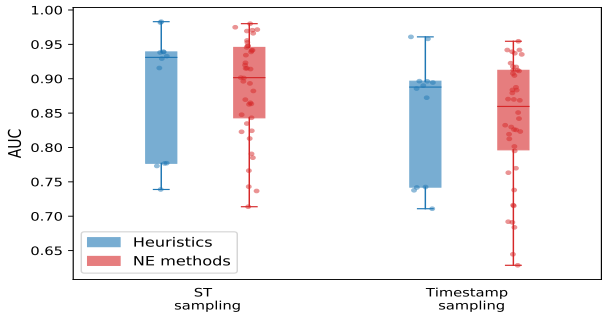



*Figure 4.7: Performance of the heuristics and NE methods compared for the spanning tree and timestamp-based sampling strategies.*

der to determine if the excellent performance of the heuristics is a result of the edge sampling strategy used. For this, we selected two temporal networks CollegeMsg [9] and MathOverflow [10] from the SNAP repository [45]. The first network encodes messages sent between students at UC Irvine while the second captures interactions between users of the Math Overflow social platform. As both networks present temporal information on the edges, we can compare a timestamp-based edge sampling to the random ST strategy. In this experiment we used setup LP2 with a fixed train fraction of $0.8$. In the timestamp sampling we selected the $20\%$ most recent edges such that, when these were removed, the remaining training graph was still connected.

Our results are summarized in Fig. 4.7. We observe that the ST strategy significantly boosts the performance of both heuristics and NE methods when compared to the timestamp sampling. We also observe that for ST sampling the performances of heuristics and NE methods are very similar, while for the timestamp sampling,

[9] https://snap.stanford.edu/data/CollegeMsg.html
[10] https://snap.stanford.edu/data/sx-mathoverflow.html

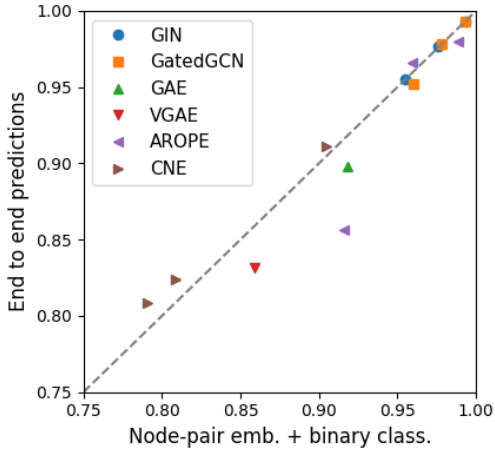



*Figure 4.8: End-to-end prediction pipelines compared to node-pair embeddings and binary classification.*

the latter perform slightly better. Ultimately, this experiment indicates that the sampling strategy induces properties on the training graphs that some approaches might exploit better than others.

Finally, we also monitored the variation in performance when embeddings are learned from different initial training graphs $\mathbf{G}_{train}$. For each of the three edge split strategies we performed 3 different evaluations with varying initial random seeds and fixed $f = 0.8$. Our results show no significant differences with a maximum standard deviation observed across all datasets, methods and strategies of only $0.003$. The largest variations are observed for the smallest network, StudentDB. This experiment, thus, indicates that averaging link prediction results over several sets $\mathbf{E}_{train}$ and $\mathbf{E}_{test}$ generated using different random seeds —in order to obtain unbiased estimates of method performance— becomes less necessary as the train network sizes ($\mathbf{G}_{train}$) exceed a few thousand nodes.

### 4.5.6 Prediction pipeline

For all end-to-end methods considered in our experiments, (i.e., GAE, VGAE, GIN, GatedGCN, AROPE and CNE) we compared their original pipelines with the same pipeline used for other methods, i.e., computation of node-pair embeddings followed by binary classification with logistic regression. The results are summarized in Fig. 4.8 and show that for most methods and networks, computing node-pair embeddings and performing logistic regression can result in a higher overall performance.

We have conducted additional experiments using setup LP2 comparing all

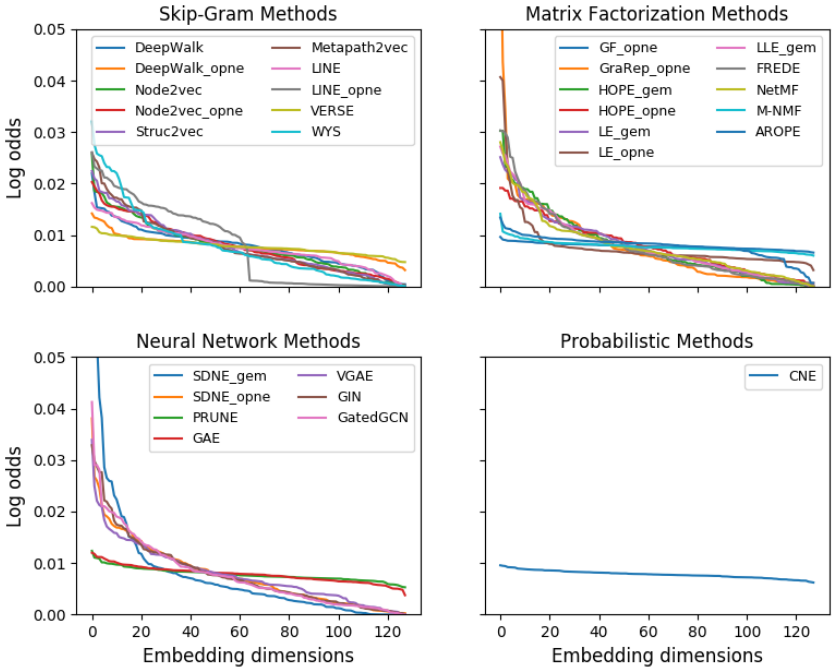



*Figure 4.9: Log odds for each coefficient of the logistic regression binary classifier. Results are presented for the GR-QC dataset but identical patterns are observed for other networks and experimental setups.*

methods with different binary classifiers: logistic regression (LR), logistic regression with 5-fold cross validation (LRCV) and decision trees (DT). Our experiments do not reveal any significant differences in performance for most methods with the exception of LE and LLE which are significantly boosted by the DT classifier. For LE we observe an increment in average AUC over all networks of $6.5\%$, from $0.87$ using LRCV to $0.93$ with DT. For LLE the increment is of $7.8\%$, from an AUC of $0.84$ with LRCV to $0.91$ with DT.

The LR binary classifier can also be used to gain insights on how NE methods distribute information over the available embedding dimensions. In Fig. 4.9 we show, for each method, the sorted log odds of the LR coefficients. These are presented for a specific network (GR-QC) and train-test split on setup LP2 with $d = 128$, but remain consistent across different settings. We observe that most methods present similar patterns with few dimensions encoding most of the information and the remaining ones capturing progressively less. Notable exceptions are some of the top performers i.e. CNE, VERSE and AROPE and also other methods such as GF, M-NMF, GAE or PRUNE which spread the information more evenly. A second interesting finding is that the `OpenNE` implementation of LINE

evaluated appears to encode no information beyond dimension 64. This could indicate an implementation error when calculating the second order proximities (recall that this implementation uses concatenated 1st and 2nd orders). We do not observe this pattern in the original implementation of LINE (LINE-2) which takes full advantage of all 128 dimensions.

### 4.5.7 Network reconstruction performance

For the network reconstruction task we first summarize method performance in terms of precision@$Np$, for $Np = 100$, in Table 4.4. We chose this threshold as it best highlights the differences between evaluated methods. We observe from the table that Katz is the top performer closely followed by VGAE, AROPE, HOPE_-opne and VERSE.

In Fig.4.10 we plot the precision@$Np$ for a range of $Np$ values for a subset of methods and networks. The methods shown are the top performers from Table 4.4, i.e. Katz, VERSE, HOPE_opne, AROPE and VGAE. We observe that Katz is the only methods that maintains a very high precision on all networks as the threshold increases. AROPE and HOPE perform very similarly on all networks and maintain overall good precision scores. For VERSE and VGAE we observe large variations between different networks and overall lower performances.

### 4.5.8 Method performance and network structure

The results in Tables 4.3 and 4.4 also show that most NE methods and heuristics perform well on Facebook, AstroPH and GR-QC (a graphical representation of the results in Table 4.3, is provided in the rightmost heatmap in Fig. 4.5[11]). These networks share similar topologies with large diameters and high clustering coefficients which indicate the presence of community structures within the networks. These observations are consistent with the data represented by the networks i.e. groups of friends in Facebook and citations between papers in different research sub-fields for AstroPH and GR-QC. The community structures with high intra-community edge densities and low inter-community densities could explain the higher link prediction performance of most methods.

On the other hand, the lowest method performance can be observed for StudentDB, BlogCatalog, Wikipedia and PPI. For StudentDB, the k-partite structure of the network poses a challenge to both heuristics and NE methods. Firstly, due to the fact that, in this case, similar node neighborhoods do not necessarily imply that two nodes should be connected. For instance, two students following the same courses will have identical node neighborhoods, yet, they should never be

[11] With the exception of GIN, GatedGCN and CNE presented in Fig. 4.4 for $d = 128$ and in Table 4.3 with $d = 8$.

*Table 4.4: Network reconstruction performance shown as precision@100 where method hyperparameter are tuned and $d = 128$ for all methods except CNE, GIN and GatedGCN where $d = 8$. The average precision@100 over all networks and average rank for each method are also presented.*

| Methods | StudentDB | Facebook | BlogCatalog | GR-QC | AstroPH | PPI | Wikipedia | Avg. prec@Np | Avg. Rank |
|---|---|---|---|---|---|---|---|---|---|
| CN | 0.00 | 0.95 | 0.52 | 0.81 | 0.91 | 0.41 | 0.73 | 0.61 | 18.50 |
| JC | 0.00 | 0.88 | 0.03 | 0.69 | 0.80 | 0.04 | 0.03 | 0.35 | 27.29 |
| AA | 0.00 | 0.95 | 0.51 | 0.93 | 0.93 | 0.44 | 0.77 | 0.64 | 17.00 |
| PA | 0.20 | 0.41 | 0.43 | 0.22 | 0.07 | 0.30 | 0.87 | 0.35 | 28.36 |
| RA | 0.00 | 0.97 | 0.59 | 0.95 | 0.98 | 0.40 | 0.90 | 0.68 | 13.64 |
| Katz | **0.30** | **1** | **1** | **1** | **1** | **1** | **1** | **0.90** | **3.43** |
| NE_heuristics | 0.22 | 0.95 | 0.50 | 0.94 | 0.97 | 0.41 | 0.83 | 0.68 | 15.71 |
| DeepWalk | 0.30 | 0.89 | 0.47 | 0.86 | 0.86 | 0.68 | 0.90 | 0.70 | 14.21 |
| DeepWalk_opne | 0.30 | 0.89 | 0.45 | 0.89 | 0.84 | 0.59 | 0.87 | 0.69 | 16.00 |
| Node2vec | 0.30 | 0.94 | 0.62 | 0.98 | 0.86 | 0.58 | 0.69 | 0.71 | 13.71 |
| Node2vec_opne | 0.30 | 0.86 | 0.56 | 0.89 | 0.87 | 0.65 | 0.72 | 0.69 | 15.43 |
| Struc2vec | 0.23 | 0.29 | 0.50 | 0.38 | 0.19 | 0.30 | **0.92** | 0.40 | 25.43 |
| Metapath2vec | 0.30 | 0.31 | 0.39 | 0.04 | 0.11 | 0.10 | 0.68 | 0.27 | 29.36 |
| LINE | 0.30 | 0.87 | 0.24 | 0.96 | 0.94 | 0.43 | 0.76 | 0.64 | 16.36 |
| LINE_opne | 0.29 | 0.87 | 0.53 | 0.67 | 0.79 | 0.63 | 0.81 | 0.65 | 18.00 |
| VERSE | 0.30 | **0.97** | **0.71** | **0.99** | **0.97** | **0.87** | 0.80 | **0.80** | **9.00** |
| WYS | 0.30 | 0.40 | 0.26 | 0.36 | 0.29 | 0.17 | 0.79 | 0.36 | 27.07 |
| GF_opne | 0.29 | 0.87 | 0.33 | **0.95** | 0.88 | 0.72 | 0.61 | 0.66 | 18.43 |
| GraRep_opne | 0.23 | 0.84 | 0.57 | 0.72 | 0.73 | 0.51 | 0.70 | 0.61 | 20.21 |
| HOPE_gem | 0.30 | 0.98 | 0.48 | 0.65 | 0.70 | 0.64 | **1** | 0.67 | 13.21 |
| HOPE_opne | 0.30 | 0.99 | 0.88 | 0.78 | 0.89 | 0.96 | **1** | **0.82** | 8.14 |
| LE_gem | 0.00 | 0.93 | 0.45 | 0.55 | 0.20 | 0.30 | 0.45 | 0.48 | 27.00 |
| LE_opne | 0.30 | 0.93 | 0.45 | 0.55 | 0.20 | 0.31 | 0.45 | 0.45 | 13.79 |
| LLE_gem | 0.30 | 0.73 | 0.36 | 0.62 | 0.44 | 0.23 | 0.88 | 0.50 | 23.64 |
| FREDE | 0.26 | 0.12 | 0.06 | 0.00 | 0.02 | 0.05 | 0.92 | 0.20 | 30.50 |
| NetMF | 0.30 | 0.98 | 0.92 | 0.66 | 0.58 | 0.99 | **1** | 0.77 | 10.00 |
| M-NMF | 0.30 | 0.83 | 0.32 | 0.61 | 0.44 | 0.44 | 0.75 | 0.52 | 23.00 |
| AROPE | 0.30 | **1** | **1** | 0.50 | **0.94** | **1** | **1** | **0.82** | **7.71** |
| SDNE_gem | 0.29 | 0.97 | 0.53 | 0.62 | **0.68** | 0.36 | 0.91 | 0.62 | 17.29 |
| SDNE_opne | 0.29 | 0.96 | 0.46 | 0.84 | 0.58 | 0.33 | 0.91 | 0.62 | 18.14 |
| PRUNE | 0.29 | 0.56 | 0.46 | 0.23 | 0.06 | 0.28 | 0.77 | 0.37 | 27.79 |
| GAE | **0.60** | 0.80 | **0.90** | **1** | 0.60 | 0.70 | 0.90 | 0.78 | 11.07 |
| VGAE | **0.60** | **1** | **0.90** | **1** | 0.60 | **0.90** | **1** | **0.85** | **5.64** |
| GIN | **0.60** | 0.80 | **0.90** | 0.10 | 0.20 | 0.50 | 0.30 | 0.48 | 21.29 |
| GatedGCN | 0.40 | 0.90 | **0.90** | 0.10 | 0.20 | 0.60 | 0.20 | 0.47 | 19.50 |
| CNE | **0.29** | **0.83** | **0.27** | **0.91** | **0.33** | **0.24** | **0.63** | **0.50** | **25.64** |

connected, as links between nodes of the same types do no exist in the data. Secondly, NE methods must at the same time represent similarity between nodes of different types and dissimilarity for those of the same type. For example, students following the same course must be embedded close to said course yet far from each other. Methods able to capture high order proximities between nodes (e.g GraRep, HOPE, AROPE) or those that learn node roles (e.g., Metapath2vec) maintain high AUC scores in this case.

The BlogCatalog and Wikipedia networks present very similar structures with small diameters, high clustering coefficients and large average degrees. This can result in cluttered representations where all nodes are close-by making downstream classification tasks harder. Lastly, the PPI network presents the opposite scenario with a large diameter and small clustering coefficient. This can result in embeddings where most nodes lie equally far from each other and, thus, overall lower AUCs.

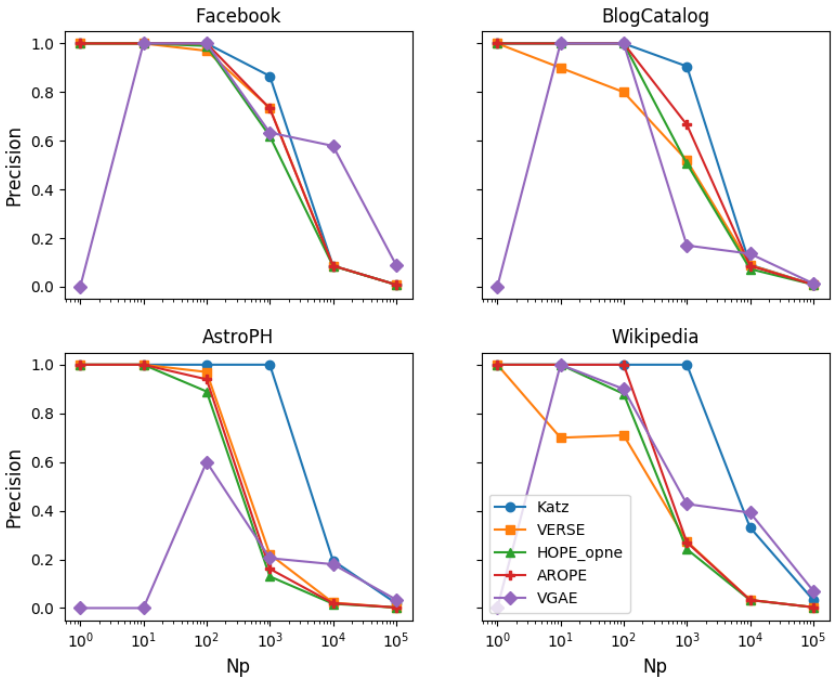



*Figure 4.10: Network reconstruction performance in terms of precision@$Np$ shown only for the top-performing approaches.*

We further analyze the reasons behind the large variations in method performance on different networks by computing the average true positive rates (TPR) and false positive rates (FPR) over all methods for each network. As threshold, we used the value in the AUC curve that maximized the accuracy of predictions. We omit showing the full statistics for brevity. We observe that while most TPRs at this threshold vary around $0.8$, the FPRs for StudentDB, BlogCatalog, PPI and Wikipedia are three orders of magnitude higher that those of other networks. This indicates that, indeed, a clear community structure simplifies the link prediction task while cluttered graphs negatively impact the prediction of non-edges.

### 4.5.9 Limits to the scope of this evaluation

Constraints on overall computation time (the results in this section required approximately 65 days to compute) inevitably imply boundaries to the scope of this evaluation: It is limited to popular mid-sized undirected and unweighted networks. It evaluates a representative but non-exhaustive set of NE methods. It only evaluates methods purely exploiting the network structure (and not node/edge metadata), such that some methods are not used to their full potential (but note that the

same holds for the heuristic baselines). Furthermore, the selected heuristics are very simple, other more powerful similarity metrics between network nodes have been proposed e.g., in [49, 50]. These are not included, as a thorough analysis of link prediction heuristic performance is not the main goal of our work. For a detailed overview and experimental evaluation of different heuristics we refer the reader to [51].

### 4.5.10 External validation of the results

Due to differences in the evaluations, we can only validate our results, to some extent, against the empirical results in the Node2vec and CNE papers. In the first case, although our AUC scores are consistently higher, the same main conclusion holds: node2vec outperforms its competitors on the three networks originally evaluated. However, a properly tuned LINE exhibits similar performance or even outperforms node2vec on other networks. In the second case, our results are very similar to those reported by the authors of CNE. Finally, our evaluation also corroborates the conclusions reported by the authors of AROPE regarding network reconstruction performance and the choice of node-pair operator for VERSE.

## 4.6 Conclusions

In this chapter we have conducted an extensive empirical study on NE methods. Our results show that, despite the surge of interest in the field in recent years, thin progress has been made. Most of the NE methods evaluated perform similarly, and in most cases not significantly better than simple heuristics for both tasks studied. On average, the proposed NE_heuristics baseline outperforms all other methods for the link prediction task. Nevertheless, some embedding-based methods such as GraRep, VERSE and CNE do show promising results. We have also shown that clear community structures in the graphs, high embedding dimensionalities and capturing high-order proximities between nodes all impact method performance positively. In contrast with recently published results, our experiments indicate no significant difference between LR and LRCV as binary classifiers in this setting. We also highlight the need to tune the node-pair embedding operator as model hyperparameter and that a single train-test split provides a good estimation of method accuracy resulting in higher evaluation efficiency. Methods such as Katz, VERSE, HOPE, AROPE and VGAE present exceptionally good results for network reconstruction. Finally, we hope this study and evaluation toolboxes such as `EvalNE` serve as initial steps towards the creation of standard evaluation pipelines for NE methods, and shed light on the current state-of-the-art. Specifically, we plan to use these results as a basis for an online resource that is continuously augmented with results for newly developed methods and/or other networks.

## Acknowledgment

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] L. Lu and T. Zhou. *Link prediction in complex networks: A survey*. Physica A, 390(6):1150–1170, 2011.

[2] R. N. Lichtenwalter and N. V. Chawla. *Link Prediction: Fair and Effective Evaluation*. In Proc. of ASONAM, pages 376–383, 2012.

[3] D. Garcia-Gasulla, C. U. Cortes, E. Ayguade, and J. J. Labarta. *Evaluating link prediction on large graphs*. In Proc. of CAAI, pages 90–99, 2015.

[4] Y. Yang, R. N. Lichtenwalter, and N. V. Chawla. *Evaluating Link Prediction Methods*. KAIS, 45(3):751–782, 2015.

[5] M. Belkin and P. Niyogi. *Laplacian eigenmaps and spectral techniques for embedding and clustering*. In Proc. of NIPS, pages 585–591, 2002.

[6] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. *LINE: Large-scale information network embedding*. In Proc. of WWW, pages 1067–1077, 2015.

[7] S. Cao, W. Lu, and Q. Xu. *GraRep: Learning graph representations with global structural information*. In Proc. of CIKM, pages 891–900, 2015.

[8] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu. *Asymmetric transitivity preserving graph embedding*. In Proc. of KDD, pages 1105–1114, 2016.

[9] M. Gao, L. Chen, X. He, and A. Zhou. *BiNE: Bipartite Network Embedding*. In Proc. of SIGIR, pages 715–724, 2018.

[10] D. Wang, P. Cui, and W. Zhu. *Structural deep network embedding*. In Proc. of KDD, pages 1225–1234, 2016.

[11] Y. Dong, N. V. Chawla, and A. Swami. *Metapath2Vec: Scalable Representation Learning for Heterogeneous Networks*. In Proc. of KDD, pages 135–144, 2017. doi:10.1145/3097983.3098036.

[12] B. Perozzi, R. Al-Rfou, and S. Skiena. *DeepWalk: Online Learning of Social Representations*. In Proc. of KDD, pages 701–710, 2014.

[13] A. Tsitsulin, D. Mottin, P. Karras, and E. Muller. *VERSE: Versatile Graph Embeddings from Similarity Measures*. In Proc. of WWW, page 539–548, 2018. doi:10.1145/3178876.3186120.

[14] T. N. Kipf and M. Welling. *Variational Graph Auto-Encoders*. NIPS Workshop on Bayesian Deep Learning, 2016.

[15] B. Kang, J. Lijffijt, and T. D. Bie. *Conditional Network Embeddings*. In Proc. of ICLR, 2019.

[16] Z. Zhang, P. Cui, X. Wang, J. Pei, X. Yao, and W. Zhu. *Arbitrary-Order Proximity Preserved Network Embedding*. In Proc. of KDD, pages 2778–2786, 2018. doi:10.1145/3219819.3219969.

[17] W. L. Hamilton, R. Ying, and J. Leskovec. *Representation Learning on Graphs: Methods and Applications*. Arxiv, 2017.

[18] D. Zhang, J. Yin, X. Zhu, and C. Zhang. *Network Representation Learning: A Survey*, 2018. arXiv:1801.05852.

[19] P. Goyal and E. Ferrara. *Graph embedding techniques, applications, and performance: A survey*. Knowledge-Based Systems, 151:78–94, 2018. doi:10.1016/j.knosys.2018.03.022.

[20] M. Khosla, V. Setty, and A. Anand. *A Comparative Study for Unsupervised Network Representation Learning*. IEEE Transactions on Knowledge and Data Engineering, 33(5):1807–1818, 2021.

[21] A. Grover and J. Leskovec. *node2vec: Scalable feature learning for networks*. In Proc. of KDD, pages 855–864, 2016.

[22] Y.-A. Lai, C.-C. Hsu, W. H. Chen, M.-Y. Yeh, and S.-D. Lin. *PRUNE: Preserving Proximity and Global Ranking for Network Embedding*. In Proc. of NIPS, pages 5257–5266, 2017.

[23] S. Gurukar, P. Vijayan, A. Srinivasan, G. Bajaj, C. Cai, M. Keymanesh, S. Kumar, P. Maneriker, A. Mitra, V. Patel, B. Ravindran, and S. Parthasarathy. *Network Representation Learning: Consolidation and Renewed Bearing*. Arxiv, 2019.

[24] A. C. Mara, J. Lijffijt, and T. De Bie. *EvalNE : A Framework for Evaluating Network Embeddings on Link Prediction*. In Proc. of EDML, (SDM), volume 2436, pages 5–13, 2019.

[25] B. Kotnis and V. Nastase. *Analysis of the Impact of Negative Sampling on Link Prediction in Knowledge Graphs*. Arxiv, 2017.

[26] X. Wei, L. Xu, B. Cao, and P. S. Yu. *Cross View Link Prediction by Learning Noise-resilient Representation Consensus*. In Proc. of WWW, pages 1611–1619, 2017.

[27] M. Hutson. *Core progress in AI has stalled in some fields*. Science, 368(6494):927–927, 2020. doi:10.1126/science.368.6494.927.

[28] M. F. Dacrema, P. Cremonesi, and D. Jannach. *Are We Really Making Much Progress? A Worrying Analysis of Recent Neural Recommendation Approaches*. In Proc. of RecSys, page 101–109, 2019. doi:10.1145/3298689.3347058.

[29] A. C. Mara, J. Lijffijt, and T. d. Bie. *Network Embedding Models for Link Prediction: Are We Making Progress?* In Proc. of DSAA, pages 138–147, 2020.

[30] T. Mikolov, K. Chen, G. Corrado, and J. Dean. *Efficient Estimation of Word Representations in Vector Space*. In Y. Bengio and Y. LeCun, editors, Proc. of ICLR, 2013.

[31] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo. *struc2vec: Learning node representations from structural identity*. In Proc. of KDD, pages 385–394, 2017.

[32] S. Abu-El-Haija, B. Perozzi, R. Al-Rfou, and A. Alemi. *Watch Your Step: Learning Node Embeddings via Graph Attention*. In Proc. of NIPS, page 9198–9208, 2018.

[33] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola. *Distributed large-scale natural graph factorization*. In Proc. of WWW, pages 37–48, 2013.

[34] S. T. Roweis and L. K. Saul. *Nonlinear Dimensionality Reduction by Locally Linear Embedding*. Science, 290(5500):2323–2326, 2000. doi:10.1126/science.290.5500.2323.

[35] A. Tsitsulin, M. Munkhoeva, D. Mottin, P. Karras, I. Oseledets, and E. Müller. *FREDE: Anytime Graph Embeddings*. Proc. VLDB Endow., 14(6):1102–1110, feb 2021.

[36] M. Ghashami, E. Liberty, J. M. Phillips, and D. P. Woodruff. *Frequent Directions: Simple and Deterministic Matrix Sketching*. SIAM Journal on Computing, 45(5):1762–1792, 2016. doi:10.1137/15M1009718.

[37] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang. *Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and Node2vec*. In Proc. of WSDM, page 459–467, 2018. doi:10.1145/3159652.3159706.

[38] J. Qiu, Y. Dong, H. Ma, J. Y. Li, C. Wang, K. Wang, and J. Tang. *NetSMF: Large-Scale Network Embedding as Sparse Matrix Factorization*. Proc. of WWW, 2019.

[39] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang. *Community Preserving Network Embedding*. In Proc. of AAAI, pages 203–209, 2017.

[40] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. *How Powerful are Graph Neural Networks?* In Proc. of ICLR, 2019.

[41] B. Weisfeiler and A. Leman. *The reduction of a graph to canonical form and the algebra which appears therein*. NTI, Series, 2(9):12–16, 1968.

[42] X. Bresson and T. Laurent. *Residual Gated Graph ConvNets*, 2018. arXiv:1711.07553.

[43] C. TU, C. YANG, Z. LIU, and M. SUN. *Network representation learning: an overview*. SCIENTIA SINICA Informationis, 47(8):980–996, 2017.

[44] B. Goethals, W. Le Page, and M. Mampaey. *Mining Interesting Sets and Rules in Relational Databases*. In Proc. of SAC, pages 997–1001, 2010. doi:10.1145/1774088.1774299.

[45] J. Leskovec and A. Krevl. *SNAP Datasets: Stanford Large Network Dataset Collection*, 2015.

[46] R. Zafarani and H. Liu. *Social computing data repository at ASU*, 2009.

[47] B.-J. Breitkreutz, C. Stark, T. Reguly, L. Boucher, A. Breitkreutz, M. Livstone, R. Oughtred, D. H. Lackner, J. Bahler, V. Wood, et al. *The BioGRID interaction database: 2008 update*. Nucleic acids research, 36:D637–D640, 2007.

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

[49] W. Cukierski, B. Hamner, and B. Yang. *Graph-based features for supervised link prediction*. In Proc. of IJCNN, pages 1237–1244, 2011.

[50] A. Hagberg, P. Swart, and D. S Chult. *Exploring network structure, dynamics, and function using networkx*. In Proc. of SciPy, pages 11–15, 2008.

[51] A. Ghasemian, H. Hosseinmardi, A. Galstyan, E. M. Airoldi, and A. Clauset. *Stacking Models for Nearly Optimal Link Prediction in Complex Networks*. Arxiv, 2019.

# 5

# Robustness Evaluation of Network Representations

*Results presented in Chapter 4 indicate differences in downstream link prediction results when the training networks are sampled according to different strategies. In this chapter, we expand upon this observation by performing a systematic analysis of the robustness of node embedding approaches, a particular type of representation learning methods. More specifically, we investigate changes in downstream performance for various models under random corruptions and adversarial attacks on the input data. Our experimental evaluation was performed, once again, leveraging the EvalNE framework presented in Chapter 3. This chapter is based on the manuscript referenced below. Some section headlines, tables and figures have been altered to ensure consistency across thesis chapters.*

⋆ ⋆ ⋆

**A. Mara, J. Lijffijt, S. Günnemann, and T. De Bie.** ***A Systematic Evaluation of Node Embedding Robustness.***

**To appear: Proceedings of LoG Conference, Dec. 2022.**

**Abstract** Node embedding methods map network nodes to low dimensional vectors that can be subsequently used in a variety of downstream prediction tasks. The popularity of these methods has grown significantly in recent years, yet, their

robustness to perturbations of the input data is still poorly understood. In this chapter, we assess the empirical robustness of node embedding models to random and adversarial poisoning attacks. Our systematic evaluation covers representative methods based on Skip-Gram, matrix factorization, and deep neural networks. We compare edge addition, deletion and rewiring attacks computed using network properties and node labels. We also investigate the performance of popular node classification attack baselines that assume full knowledge of the node labels. We report qualitative results via embedding visualization and quantitative results in terms of downstream node classification and network reconstruction performances. We find that node classification results are impacted more than network reconstruction ones, that degree-based and label-based attacks are on average the most damaging and that label heterophily can influence attack performance.

## 5.1 Introduction

In recent years, the design of robust machine learning models has become an important topic and attracted significant amounts of research attention [1–4]. The term 'robust' refers to the ability of a model to provide consistent and accurate predictions under small perturbations of the input data. These perturbations can appear in the form of random noise, out of distribution (OOD) data, or partially observed inputs [5]. They can affect models at train or evaluation times and be random or adversarial in nature. For a more complete overview of robustness in machine learning we refer the reader to [6]. In this chapter, we empirically study both random and adversarial attack scenarios where perturbations are either a consequence of noise or specifically crafted to reduce model performance. We further focus our analysis on attacks affecting the models at training time exclusively, also know as the poisoning scenario [7].

Simultaneously, node representation learning or node embedding models have become increasingly popular for bridging the gap between traditional machine learning and network structured data [8–10]. These approaches map network nodes to real-valued vectors that can be subsequently used in downstream prediction tasks, such as classification [11] and regression [12]. Training of these models can be performed in a semi-supervised or unsupervised fashion. In the former, embeddings are optimized for a particular downstream task while in the latter, general purpose embeddings are obtained. Robustness is an important feature for representation learning models as well. One would generally prefer small changes in the input networks to have a minimal impact on the vector representations learned and on downstream task performance. Moreover, with the deployment of these models in safety-critical environments (e.g., [13]) and on the web (where adversaries are common [14, 15]), robustness evaluation has become ever more essential. Unfortunately, the robustness of unsupervised node embedding approaches is poorly

understood. Some recent studies have analyzed particular semi-supervised models based on the Graph Neural Network (e.g., [16]) and on shallow models (e.g., [17]). Others, have evaluated specific unsupervised random walk approaches under poison attacks [7]. Methods leveraging unsupervised embedding learning paradigms such as matrix factorization and deep neural networks, have not received much attention yet. Additionally, there is a lack of studies providing broader robustness evaluations and comparing multiple models.

We perform a systematic empirical analysis of the robustness of foundational works in the field of node embeddings. Among the 9 unsupervised approaches evaluated we include Node2vec [18], GraRep [19], and SDNE [20], which have inspired many other methods based on similar principles, e.g., [21–23]. The models considered can be categorized into Skip-Gram, matrix factorization, and deep neural networks, and their robustness is compared on two downstream tasks: node classification and network reconstruction. We evaluate robustness under randomized and adversarial attacks targeting the network edges. For adversarial attacks we limit the scope to heuristic-based approaches where edges are targeted based on topological network properties (e.g. assortativity, degree). In contrast, optimization-based attacks (e.g. [7, 24–26]) solve a multi-level optimization problem to identify the most promising targets. Heuristic attacks are, thus, weaker yet simpler and more computationally efficient making them more easily accessible to an attacker. Additionally, they do not require tailoring to specific embedding models and downstream tasks –as many optimization-based approaches do– and provide intuitive and explainable targets. Moreover, the heuristic attacks considered in this chapter have already shown to effectively lead to structural collapse in networks [27]. The analysis of stronger optimization-based models is left for future work. Lastly, we focus our evaluation on global attack scenarios where changes can be made to the entire graph structure provided a fixed attack budget.

**Contributions** Our main contribution is a systematic analysis of node embedding robustness. We evaluate a total of 9 unsupervised node embedding approaches based on three learning paradigms. We employ 6 small and mid-sized networks and compare 14 different poison attack strategies. Further, we investigate differences between randomized and adversarial attacks and compare edge addition, deletion and rewiring strategies. We also investigate attacks leveraging full knowledge of the node labels, commonly used as baselines, and show that network homophily (tendency of nodes with similar labels to be connected) and heterophily (where nodes of different labels are more often connected) have a strong impact on their performance. This work constitutes the first empirical evaluation of its magnitude on node embedding robustness.

The remainder is organized as follows: in Section 5.2 we present the related work and in Section 5.3 we introduce the embedding methods and attack strategies

evaluated. In Section 5.4, we discuss the experimental evaluation and results and finally, in Section 5.5 we outline our main conclusions.

## 5.2 Related work

A large body of research has shown that traditional machine learning models and more recently deep neural models can be easily misled into providing wrong answers with high confidence [28, 29]. Work on identifying and protecting against these adversarial attacks has particularly developed in the field of computer vision [30]. Works in this field, including [31, 32], have also shown how changes unperceivable to the human eye can result in dramatic performance drops or misclassifications. Later, adversarial attacks were introduced in the field of network science [5]. In [27], the authors show how structural properties of networks can collapse as a result of attacks. The authors further provide a framework for simulating attacks and defenses on networks. With the popularization of node embedding methods authors have also investigated adversarial attacks on particular semi-supervised [16, 33] and unsupervised [17] approaches. A survey reviewing various adversarial attacks and defense strategies, specifically designed for semi-supervised representation learning models, is presented in [34]. The authors also collect implementations of representative attack and defense strategies and make them publicly available as a PyTorch toolbox. For unsupervised representation learning models, while there are some empirical studies comparing performance (e.g., [35]), there is little research evaluating robustness. With the present chapter, our aim is to fill this gap and provide a fist empirical study and overview on the robustness to random and adversarial attacks of unsupervised node embedding approaches. Lastly, akin to the results in [36] for graph classification using GNN models, we also investigate patterns in the attacks and relations to the network structure. This analysis reveals an interesting effect of label heterophily on node classification attacks.

## 5.3 Methods

In this section we introduce the node embedding approaches evaluated and the attack strategies used to poison the input networks. Regarding notation, in what follows we will use $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ to refer to an undirected graph with vertex set $\mathbf{V} = \{v_1, \ldots, v_N\}$, $N = |\mathbf{V}|$ and edge set $\mathbf{E} \subseteq (\mathbf{V} \times \mathbf{V})$, $M = |\mathbf{E}|$. We will represent edges or connected node-pairs as unordered pairs $\{v_i, v_j\} \in \mathbf{E}$. And refer to pairs $\{v_i, v_j\} \notin \mathbf{E}$ as non-edges or unconnected node-pairs. Node embeddings are denoted as $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)$, $\mathbf{X} \in \mathbb{R}^{\mathrm{N} \times \mathrm{d}}$ where $\mathbf{x}_i$ is the d-dimensional vector representation corresponding to node $v_i$.

### 5.3.1 Node embedding methods

For our experimental evaluation we have selected 9 representative methods spanning three different embedding learning paradigms, namely Skip-Gram, matrix factorization and deep neural networks. Next, we introduce each paradigm and the corresponding methods.

**Skip-Gram** These approaches capture node similarities in the graph through random walks and leverage the Skip-Gram model [37] to obtain node representations that maximize the posterior probability of observing neighboring nodes in the walks. From this category we evaluate: Deepwalk [38], the seminal work that proposed fixed length random walks to capture node similarities and Skip-Gram (approximated via hierarchical softmax) for learning the embedding matrix $\mathbf{X}$; Node2vec [18], which introduced more flexible random walks controlled by in/out and return parameters and approximates Skip-Gram via negative sampling; LINE [39], where the authors leverage first and second order proximities to learn representations; And finally, VERSE [11], which minimizes the KL-divergence between a similarity metric on $\mathbf{G}$ (by default Personalized PageRank) and a vector similarity on $\mathbf{X}$.

**Matrix factorization** Factorization methods take as input node similarities encoded in the graph Laplacian, incidence matrices, adjacency matrices ($\mathbf{A}$) and their polynomials, etc. and compute low dimensional embeddings by factorizing the selected matrix. We evaluate the following methods based on this paradigm: GraRep [19], HOPE [40], NetMF [41] and M-NMF [42]. GraRep factorizes high order polynomials of $\mathbf{A}$, HOPE can factorize different similarity matrices provided they can be expressed as a composition of two sparse proximity matrices. NetMF decomposes the DeepWalk transition matrix via SVD and lastly, M-NMF computes embeddings via non-negative matrix factorization and incorporates community structure in this process.

**Deep neural networks** Deep neural models, from auto-encoders to Siamese networks or CNNs, have also been used to obtain node representations from a graph's link structure in an unsupervised fashion. Among these types of methods we evaluate SDNE [20], a deep neural model that captures first and second order proximity in the graph.[1]

[1] We also evaluated PRUNE [12] but despite our best efforts the method severely underperformed on all tasks.

*Table 5.1: Poison attacks evaluated and their types: (D) deterministic, (ND) non-deterministic.*

| Edge addition | | Edge deletion | | Edge rewiring | |
|---|---|---|---|---|---|
| Name | Type | Name | Type | Name | Type |
| add_rand | ND | del_rand | ND | rew_rand | ND |
| add_deg | ND | del_deg | D | - | - |
| add_pa | ND | del_pa | D | - | - |
| add_da | ND | del_da | D | - | - |
| add_dd | ND | del_dd | D | - | - |
| add_ce | ND | del_di | ND | DICE | ND |

### 5.3.2 Network attacks

We subdivide network attacks into randomized and adversarial and further into three main types based on the changes to the network structure. These changes are edge addition, edge deletion and edge rewiring. Table 5.1 summarizes all attacks and below we briefly describe each one.

**Randomized attacks** These attacks are designed to simulate random errors or failures in the networks. We consider edge addition (*add_rand*), deletion (*del_rand*) and rewiring (*rew_rand*). In the first case, pairs of nodes, $v_i, v_j \in \mathbf{V}$ are selected uniformly at random and added to $\mathbf{E}$ iff $v_i \neq v_j$ and $\{v_i, v_j\} \notin \mathbf{E}$. For deletion attacks, edges $\{v_i, v_j\} \in \mathbf{E}$ are selected uniformly at random and removed from $\mathbf{E}$ iff $d_i \geq 2 \wedge d_j \geq 2$. Here $d_i$ and $d_j$ represent the degrees of nodes $v_i$ and $v_j$, respectively. In rewire attacks we use *del_rand* to remove a budget of edges $\{v_i, v_j\} \in \mathbf{E}$ and then reconnect each $v_i$ to a new node $v_k$ such that $v_k \neq v_j$ and $\{v_i, v_k\} \notin \mathbf{E}$.

**Adversarial attacks** We also consider a particular type of heuristic-based adversarial attacks which target specific network properties such as node degrees, assortativity, and node labels. Despite their lower effectiveness compared to optimization-based attacks, we evaluate these approaches due to their lower computational complexity, applicability to different embedding methods and downstream tasks, and explainable attack targets. Moreover, they aim to modify key structural properties commonly captured by representation models and can thus lead to worse representations. The heuristics considered have also been successfully used as baselines in previous works e.g., [7, 27].

For all edge addition attacks we ensure that newly generated pairs do not already exist in the graph, i.e., $\{v_i, v_j\} \notin \mathbf{E}$, and they do not form selfloops, i.e., $v_i \neq v_j$. For degree-based (*add_deg*) and preferential attachment (*add_pa*) edge addition strategies we sample nodes uniformly and based on degree, re-

spectively, and connect them to destination nodes sampled based on degree. For the degree assortativity (*add_da*) and disassortativity (*add_dd*) attacks, we generate edges which increase/decrease this property. We define assortativity ($r$) akin to [43] and compute it per edge as the product of standard scores of $d_i$ and $d_j$, i.e. $r_{\{v_i,v_j\}} = (d_i - \mu)/\sigma \cdot (d_j - \mu)/\sigma$. Where $\mu = \frac{1}{M}\sum_{l=1}^{M} d_l^2$ and $\sigma = (\frac{1}{M}\sum_{i=1}^{M} d_i \cdot (d_i - \mu)^2)^{1/2}$. Thus, to increase assortativity we sample nodes $v_i$ with probability $p_i \propto |d_i - \mu|$ and connect them to nodes $v_j$ sampled with probability $p_j \propto \frac{1}{|d_i - d_j|}$. To increase disassortativity we sample nodes $v_i$ as above and $v_j$ with $p_j \propto |d_i - d_j|$. The *add_ce* strategy applies to labeled graphs only and adds a set of random edges connecting nodes of dissimilar labels, exclusively.

Unless otherwise specified, edge deletion attacks ensure that input networks do not become disconnected after the attack. For *del_deg* and *del_pa* we first sort all edges based on the appropriate metric, i.e., $d_i + d_j$ for *del_deg* and $d_i \times d_j$ for *del_pa*, and later remove the top edges that do not disconnect the network. For *del_da* and *del_dd* we compute $r_{\{v_i,v_j\}}$ and $-r_{\{v_i,v_j\}}$ as described above. Then, we sort the edges based on these properties and take the top candidates in each case while avoiding disconnections. The *del_di* strategy applies exclusively to labeled graphs and randomly selects edges for removal where the incident nodes share the same label.

Finally, *DICE* [7] is an adversarial attack where edges are removed or added to a network with equal probability. Edges are removed according to the *del_di* strategy and added following *add_ce*. It is important to note that all edge deletion attacks with the exception of *del_di* are deterministic while the remaining addition and rewire attacks are non-deterministic (see Table 5.1).

## 5.4 Experiments

In this section we present the experimental setup, networks used and results obtained. All our experiments were carried out on a machine equipped with two 12 Core Intel(R) Xeon(R) Gold processors, 1TB of RAM and an RTX 3090 GPU.

To ensure reproducibility of results, we have employed and extended the capabilities of the EvalNE toolbox [44]. This Python framework allows users to assess the performance and robustness of network embedding approaches for downstream node classification, network reconstruction, link prediction and sign prediction. In the framework we have integrated a variety of random and adversarial poison attack strategies, including those introduced in Section 5.3.2 and Table 5.1. In EvalNE, complete evaluation pipelines and hyperparameters are specified through configuration files which can be used at any time to replicate results. These configuration files together with the rest of our code are available online at https://github.com/aida-ugent/EvalNE-robustness.

### 5.4.1 Preliminaries and setup

As pointed out in Section 5.1, the main goal of this chapter is to investigate the robustness of node embedding approaches to poison attacks. To this end we report changes in downstream node classification and network reconstruction performances for different attacks on the input graphs. Next, we summarize the main goals and evaluation pipelines for both tasks and the overall evaluation setup.

**Node classification** Given an input graph and labels for a subset of vertices, the goal in node classification is to infer the labels of the remaining vertices. To evaluate node classification robustness we proceed as follows. (1) We start by attacking an input network $\mathbf{G}$ with a specific strategy (from Table 5.1) and budget $b$. The budget defines the number of edges an attacker can add, delete or rewire in the network, expressed as a fraction of the total edges. For example, $b = 0.1$ indicates 10% of all edges in $\mathbf{E}$. (2) The attacked network $\hat{\mathbf{G}} = (\mathbf{V}, \hat{\mathbf{E}})$ is then provided as input to a node embedding approach which yields a representation matrix $\mathbf{X}$ containing vertex representations as its rows. As shown by Mara et. al. [35], gains from optimizing the hyperparameters of these models are marginal, and thus, we resort to fixed default values.[2] We also fix the embedding dimensionality $d = 128$. (3) Given a number of training nodes $N_{tr}$ (also defined as a fraction of all nodes in $\mathbf{V}$), a multi-class one-versus-rest Logistic Regression classifier with 5-fold cross validation is trained to predict node labels from node representations. (4) We repeat the previous step 3 times with different node samples and report average results. For some experiments we will report results independent of the value of $N_{tr}$. In these cases we additionally average results over several values of $N_{tr}$. (5). Finally, and unless otherwise specified, for the non-deterministic attacks listed in Table 5.1 we repeat the complete process 3 times with varying random seeds resulting in different sets of edges being removed in step 1). We report node classification performance in terms of f1_micro and f1_macro.

**Network reconstruction** In this task the aim is to investigate how well the link structure of an input network can be recovered from the node representations. To this end node representations are first learned from the input network. Then, node-pair representations are derived by applying a binary operator on the node representations. Finally, a binary classifier is trained to discriminate edges from non-edges. High quality representations are expected to result in the classifier scores of edges being higher than those of non-edges.

We evaluate robustness on this task akin to node classification. (1) We attack the input network $\mathbf{G}$ with a given strategy and budget $b$. (2) We compute

[2] Exact hyperparameter values and method implementations are reported in the EvalNE configuration files.

node representations for $\hat{\mathbf{G}}$ with different methods for which we use fixed default hyperparameters. (3) Representations of node pairs $\{v_i, v_j\}$ are combined into node-pair representations using the Hadamard product, i.e., $\mathbf{x}_{i,j} = \mathbf{x}_i \cdot \mathbf{x}_j$. (4) A binary Logistic Regression with 5-fold cross validation is trained using representations corresponding to edges and non-edges in $\hat{\mathbf{G}}$. (5) Classification performance is tested using representations of edges and non-edges of the original non-attacked graph $\mathbf{G}$. For computational efficiency, we approximate the performance using 5% of all possible node-pairs in $\mathbf{G}$. (6) We again repeat the complete process 3 times for non-deterministic attacks. For this task we report AUCs and average precision.

**Experimental setup** Our evaluation setup is structured as follows. First, in Section 5.4.3.1 we investigate the performance of node embedding approaches under random attacks. In this case, we use the *add_rand* and *del_rand* strategies and vary the attack budget $b \in [0.1, 0.2, ..., 0.9]$.[3] For node classification specifically, we report average results over $N_{tr} \in [0.1, 0.5, 0.9]$, 3 node shuffles for each $N_{tr}$ value, and 3 experiment repetitions for non-deterministic attacks. For network reconstruction we only perform the 3 experiment repetitions for non-deterministic attacks. We then also investigate the effect of the number of labeled nodes for node classification by comparing the results obtained for $N_{tr} = 0.1$ to $N_{tr} = 0.5$ and $N_{tr} = 0.9$. Second, in Section 5.4.3.2 we evaluate adversarial robustness. We use a similar setup with the following exceptions: we compare all attacks from Table 5.1 (random attacks are used as baselines) and the budget is fixed to $b = 0.2$. Third, in Section 5.4.3.3 we compare addition, deletion and rewiring attacks. For both downstream tasks we compare *add_rand*, *del_rand* and *rew_rand* and for node classification we additionally compare *add_ce*, *del_di* and *DICE*. Other parameters are set as for the adversarial attack experiment. In this section we also investigate differences between deletion attacks that disconnect and those that do not disconnect the input networks. Lastly, in Section 5.4.3.4 we investigate the performance of common node classification baselines such as *DICE* that leverage full knowledge of the node labels.

## 5.4.2 Data

To conduct our experiments we use a total of 6 small and mid sized networks from different domains. Specifically, for node classification we use Citeseer [45] and Cora [46], two citation networks where nodes denote publications, edges represent citations between them and node labels indicate the main research field of each paper. For network reconstruction we use PolBlogs [47], a network of political blogs connected to each other via hyperlinks, and Facebook [48], a network of indi-

[3] We acknowledge the impracticality of extreme budgets but find these edge cases theoretically interesting.

*Table 5.2: Main statistics of the networks used for evaluation. The average degree is indicated by $\langle k \rangle$, the assortativity coefficient by $r$, and 'Viz' represents the network visualization task in Section 5.4.3.4.*

| Network | Type | Task | # Nodes | # Edges | # Labels | $\langle k \rangle$ | $r$ |
|---|---|---|---|---|---|---|---|
| Citeseer | Citation | NC | 2110 | 3668 | 6 | 3.48 | 0.01 |
| Cora | Citation | NC | 2485 | 5069 | 7 | 4.08 | -0.07 |
| PolBlogs | Web | NR | 1222 | 16714 | - | 27.35 | -0.22 |
| Facebook | Social | NR | 4039 | 88234 | - | 43.69 | 0.06 |
| IIP | Collaboration | Viz | 219 | 630 | 3 | 5.75 | -0.22 |
| StudentDB | Relational | Viz | 395 | 3423 | 7 | 17.33 | -0.34 |

viduals and their social relations on the platform. Lastly, we perform qualitative and visualization experiments on the internet industry partnership (IIP) [49] and the StudentDB [50] networks. In the former, nodes represent companies, edges represent relations such as alliance or partnership and node labels indicate the company's main business area, i.e., user content, infrastructure or commerce. The latter, StudentDB, is a k-partite network representing a snapshot of the Antwerp University relational database. Nodes represent entities such as students, courses, tracks, etc., and edges are binary relations, e.g., student-in-track, course-in-track, etc. Node labels indicate the type of each node (see Appendix A.2.1 for more details). In Table 5.2 we summarize the main statistics of the networks used.

## 5.4.3 Experimental results

### 5.4.3.1 Randomized attacks

We start in Figure 5.1a with node classification performance under random edge attacks and varying attack budgets. In the chart, negative budgets indicate edge deletion and positives indicate edge addition. In this case we allow edge deletions to disconnect the original networks. We report f1_micro scores for the Citeseer network (f1_macro results as well as those for the Cora network are similar and provided in Appendix A.2.2). From the figure we first note different general behaviors for edge deletion and addition attacks. Deletions cause a consistent performance degradation until complete network collapse at $b = 0.9$. Additions cause a sharper loss in performance for relatively low budget values ($b \leq 0.2$) which become less severe around $b = 0.4$. Thus, in the low budget regime commonly analyzed in the literature ($-0.2 < b < 0.2$), edge addition attacks are superior to edge deletion. Outside of this range, however, edge deletions are more damaging. This observation is reasonable given the asymmetry in the attack budgets. Removing 90% of the graph edges leaves significantly less information to learn an embedding from than adding 90% of spurious edges. We also observe from Figure 5.1a that NetMF and M-NMF are slightly more robust to edge additions than

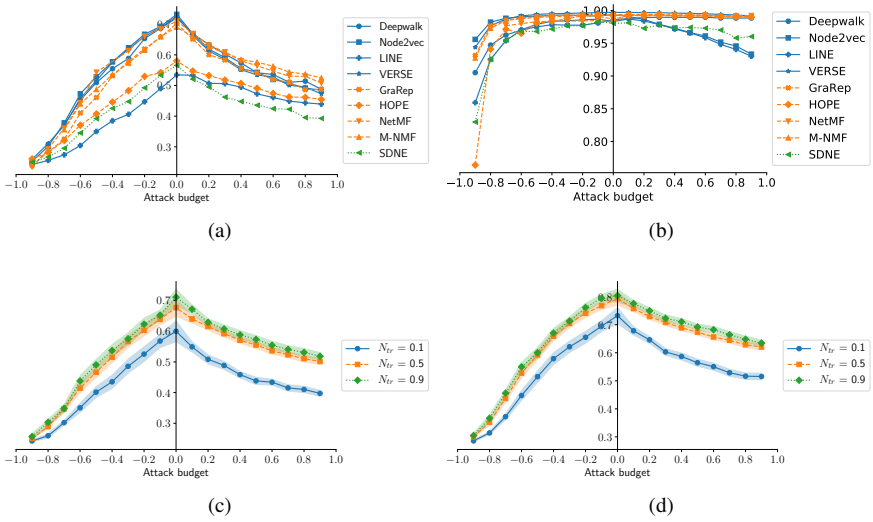



*Figure 5.1: Robustness to randomized attacks for different budget values. The x-axis shows budgets as a fraction of all edges in the graph. Negative values represent edge deletion and positives edge addition attacks. Figure 5.1a presents f1_micro scores for node classification on Citeseer. Figure 5.1b shows AUCs for network reconstruction on Facebook. In Figures 5.1c and 5.1d we show average node classification f1_micro scores for different fractions of labeled nodes $N_{tr}$ on Citeseer and Cora, respectively. Shaded areas denote 95% confidence intervals.*

other approaches while edge deletion performance is similar across the board.

In Figure 5.1b we present the AUC scores for reconstructing the original Facebook network $\mathbf{G}$, from an attacked graph $\hat{\mathbf{G}}$. The plot indicates high edge recovery with AUCs $\approxeq 1$ despite the random attacks. Most methods maintain high robustness for a wide range of budget values. Some notable exceptions are Node2vec, LINE, and SDNE which consistently lose performance the more adversarial edges are added. One possible explanation is that these methods are not only affected by the addition of spurious edges but also by the removal of potentially informative negative samples, used by all three approaches to learn embeddings. For the PolBlogs dataset presented in Appendix A.2.2, we observe similar patterns. An exception in both networks is HOPE, which significantly degrades performance for strong edge deletion attacks ($b \leq -0.6$). This indicates the method is less suited to learning embeddings of highly sparse networks. The high robustness exhibited by the evaluated approaches on this task is particularly interesting given the double impact of the attacks. Unlike in node classification, attacks on network reconstruction affect the models both at embedding learning time and binary classifier training (edge and non-edge train labels are obtained from the attacked $\hat{\mathbf{G}}$).

We now focus our attention on the impact of the number of train labels available for node classification ($N_{tr}$). In Figures 5.1c and 5.1d we compare the average performance over all methods and experiment repetitions for $N_{tr} \in [0.1, 0.5, 0.9]$ on Citeseer and Cora, respectively. For both networks we observe equally low performances when few labeled nodes are available i.e., $N_{tr} = 0.1$. For larger values ($N_{tr} \geq 0.5$) the performances are very similar. We also observe that as networks become denser (as we move right on the x-axis in each plot) the difference between low and high values of $N_{tr}$ become more significant. This indicates that node embedding methods will generally not provide robust predictions when few labeled nodes are available and this situation will worsen the denser the network.

#### 5.4.3.2 Adversarial attacks

We now compare the effect of different heuristic-based adversarial attacks on node classification. Figures 5.2a and 5.2b summarize the results on the Citeseer network for edge deletion and addition attacks, respectively. In both cases we present decreases in f1_micro caused by different attacks with budget $b = 0.2$, as compared to the performance on the non-attacked graph. Firstly, if we compare across graphs we observe that edge additions decrease performance more than deletions across all methods for this particular budget value. This is also consistent with our observations from Figure 5.1a for random attacks on node classification. Among the edge deletion attacks we see that *del_dd* is, from an adversarial perspective, the most effective strategy. With this attack, we are targeting edges from high degree to low degree nodes further increasing the uncertainty regarding the latter. On the other hand, for edge addition the most effective strategies are connecting edges with different labels together (*add_ce*) or connecting nodes with similar degrees to each other (*add_deg*). It is interesting to note that attacks with full knowledge of the node labels *del_di* and *add_ce* are not significantly stronger than others e.g., degree based attacks. The colors in both figures indicate different fractions of labeled nodes. We observe that most of the variance in performance comes from the experiments with $N_{tr} = 0.1$ (blue points) and that these are also mostly concentrated in the lower ends of the boxplots. The variances for $N_{tr} \geq 0.5$ are very similar across different attack strategies and networks.

In Figures 5.2c and 5.2d we present similar results for network reconstruction. In these cases we show the combined performances for both Facebook and PolBlogs datasets. The experiments reveal that edge deletion attacks are marginally stronger than edge addition. In particular, deleting edges based on degree is the most effective adversarial technique of the ones we have evaluated. Overall, we also observe much less variance in performance compared to the results on node classification.

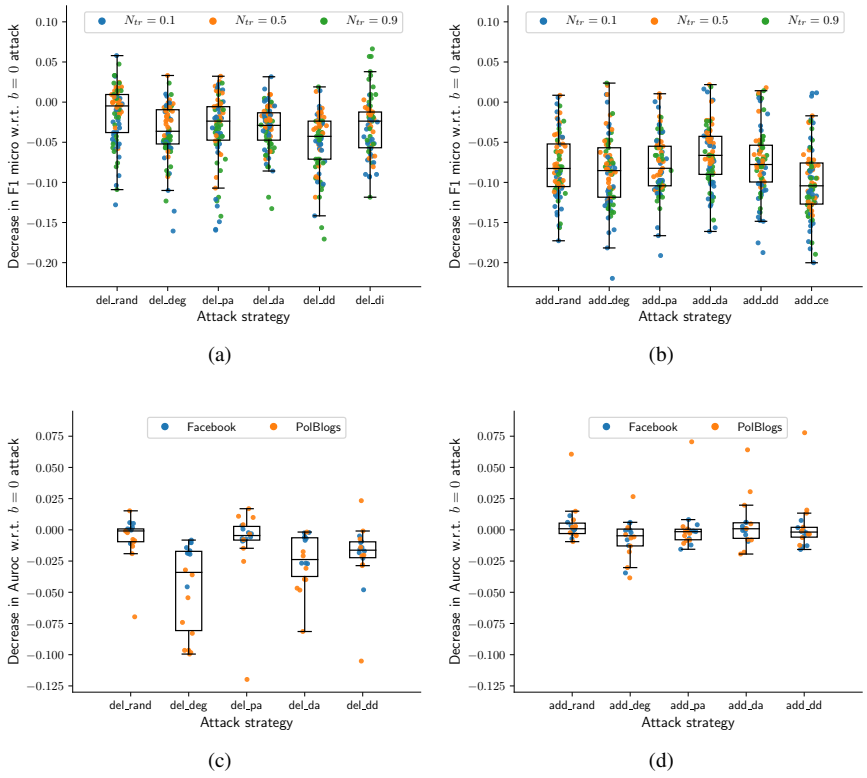



*Figure 5.2: Comparison of adversarial edge deletion and addition attacks for $b = 0.2$. Figures 5.2a and 5.2b show deletion and addition attacks on node classification for Citeseer. Colors indicate the fraction of train nodes $N_{tr}$. Figures 5.2c and 5.2d show similar results for network reconstruction on both Facebook and PolBlogs networks combined (colors indicate the network).*

#### 5.4.3.3 Addition, deletion and rewiring attacks

In Figure 5.3 we compare edge addition, rewiring and deletion attacks on both downstream tasks. The attack budget is fixed to $b = 0.2$ and we show combined results for the two networks used in each task (marker color denotes the data used). We observe that for node classification rewiring attacks perform best (central boxes in the left and middle plots in Figure 5.3). This is also the case if we look at each individual dataset with results for Cora (orange dots) being significantly higher than those on Citeseer (blue dots). For network reconstruction we have much less data available, considering that we do not need to test different train sizes and shuffles per size. In this case the results indicate similar performances for all attack types. We further observe that results on the Facebook network are overall higher

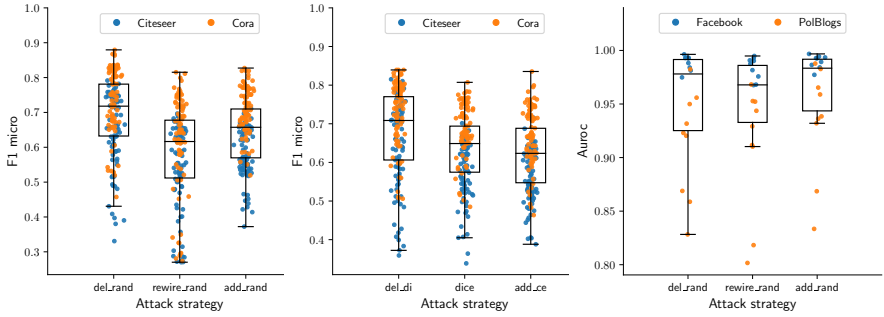



*Figure 5.3: Comparison of edge addition, rewiring and deletion attacks for both downstream tasks. The leftmost and center figures present f1_micro scores for random and node label based attacks on node classification. The rightmost figure shows AUC results for random attacks on network reconstruction.*

than on PolBlogs. The f1_macro and average precision scores for each task also corroborate this findings and are presented in Appendix A.2.3.

We further investigate how strong a role network connectivity plays in adversarial attacks. We compare random and degree attacks constrained to not disconnecting the input networks and their unconstrained counterparts. We find that constrained attacks are on average, over all methods and networks 5% less effective. Specifically, for random attacks the f1_micro performance without disconnections is $0.651 \pm 0.166$ (mean and standard deviation) and with disconnections $0.612 \pm 0.161$. Similarly, for degree based attacks average performance reaches $0.637 \pm 0.163$ when disconnections are prevented and $0.606 \pm 0.164$ when they are not.

#### 5.4.3.4 Attacks exploiting node labels

In this section we investigate adversarial attacks on node classification where the attacker has full access to the node labels. These types of attacks e.g., DICE, are commonly used as baselines under the assumption that access to the node labels inevitably leads to stronger attacks. Here, we demonstrate that the above assumption does not always hold. Specifically, we find that node label homophily/heterophily has a strong impact on the performance of these types of attacks.

In this experiment we use the IIP and StudentDB datasets. The former is an example of a homophilic network where 70.9% of edges connect nodes of the same label. On the other hand, StudentDB is a strongly heterophilic network where no edges connect nodes sharing the same label. We summarize the results for node2vec, although our findings apply to other methods capturing high order proximities in graphs. We use *DICE* as an attack strategy.

We start our evaluation by attacking both networks with budgets $b \in [0.0, 0.2, 0.6]$.

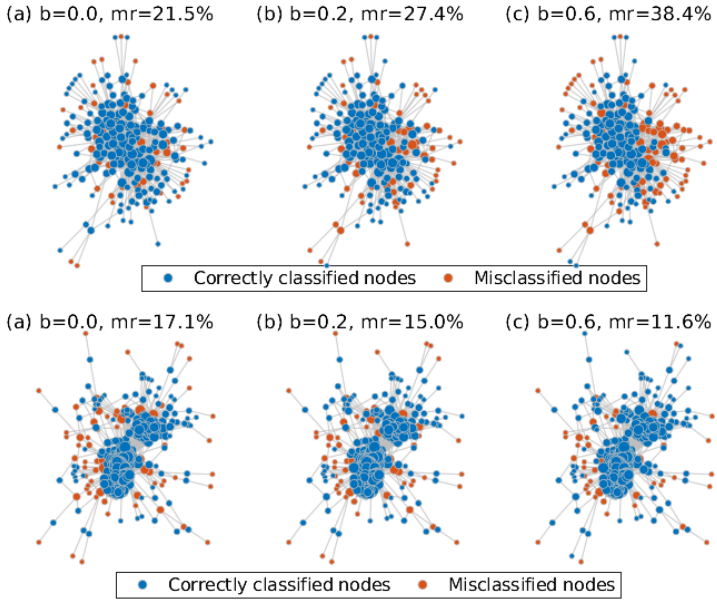



*Figure 5.4: Correctly and incorrectly classified nodes for the homophilic IIP network (top row) and the heterophilic StudentDB network (bottom row) for varying attack budgets.*

We then learn node embeddings and perform downstream node classification for each network and attack budget. Correctly and incorrectly classified nodes at validation time are recorded for each case. The top row in Figure 5.4 presents a spring-layout representation of the IIP network for each attack budget where nodes are colored based on their prediction status, correct (blue) or incorrect (orange). From the figure we can visually confirm that, as the attack strength increases, the misclassification rate (*mr* in the figure) also increases. This is also confirmed numerically by the *mr* value presented above each plot.

The bottom row in Figure 5.4 presents the same information for the StudentDB network. In this case, as the attack strength increases the misclassification rate decreases (as can be seen visually and through the mr values). This seemingly counter intuitive behavior can be explained by the fact that *DICE* introduces additional information in the network reinforcing the heterophily schema (similar nodes remain unconnected while dissimilar ones are more connected). This dilutes the local network structure and makes nodes of the same type more similar to each other. Methods such as node2vec able to capture high order proximities between nodes can capitalize on this additional information to provide embeddings more suitable for node classification.

## 5.5 Conclusions

In this chapter we have demonstrated that node embedding approaches, regardless of their underlying representation mechanisms, are sensitive to random and adversarial poison attacks. We have shown that results on downstream node classification are significantly less robust compared to those on network reconstruction. Our experiments also revealed that for low attacks budgets (below 20% of edges in the graph) edge addition attacks are generally stronger than edge deletions. Outside of this range, the opposite is true. Surprisingly, our empirical evaluation showed no significant differences between the heuristic-based adversarial attacks evaluated. Leveraging full knowledge of the node labels when attacking node classification does also not yield significantly stronger attacks. Finally, we have also shown that the number of labeled nodes plays a fundamental role in node classification robustness, that rewiring attacks are generally stronger than addition or deletion independently, and that attacks leveraging node label information can result in improved representations of heterophilic networks. With this work and our extension to robustness evaluation for the EvalNE software, we hope to lay the foundations for further research in this area.

## Acknowledgment

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] K. Bhatia, P. Jain, P. Kamalaruban, and P. Kar. *Consistent Robust Regression*. In Proc. of NeurIPS, volume 30. Curran Associates, Inc., 2017.

[2] M. Ren, W. Zeng, B. Yang, and R. Urtasun. *Learning to Reweight Examples for Robust Deep Learning*. In Proc. of ICML, volume 80, pages 4334–4343. PMLR, 2018.

[3] D. Bertsimas, J. Dunn, C. Pawlowski, and Y. D. Zhuo. *Robust Classification*. INFORMS Journal on Optimization, 1(1):2–34, 2019.

[4] W. Jin, Y. Ma, X. Liu, X. Tang, S. Wang, and J. Tang. *Graph Structure Learning for Robust Graph Neural Networks*. In Proc. of KDD, page 66–74. ACM, 2020.

[5] S. Günnemann. *Graph Neural Networks: Adversarial Robustness*, pages 149–176. Springer Nature Singapore, 2022.

[6] C.-J. H. Pin-Yu Chen. *Adversarial Robustness for Machine Learning*. Elsevier, 2022.

[7] A. Bojchevski and S. Günnemann. *Adversarial Attacks on Node Embeddings via Graph Poisoning*. In Proc. of ICML, pages 695–704, 2019.

[8] B. Kang, J. Lijffijt, and T. D. Bie. *Conditional Network Embeddings*. In Proc. of ICLR, 2019.

[9] J. Qiu, Y. Dong, H. Ma, J. Y. Li, C. Wang, K. Wang, and J. Tang. *NetSMF: Large-Scale Network Embedding as Sparse Matrix Factorization*. Proc. of WWW, 2019.

[10] A. Mara, Y. Mashayekhi, J. Lijffijt, and T. de Bie. *CSNE: Conditional Signed Network Embedding*. In Proc. of CIKM, page 1105–1114. ACM, 2020.

[11] A. Tsitsulin, D. Mottin, P. Karras, and E. Muller. *VERSE: Versatile Graph Embeddings from Similarity Measures*. In Proc. of WWW, page 539–548, 2018.

[12] Y.-A. Lai, C.-C. Hsu, W. H. Chen, M.-Y. Yeh, and S.-D. Lin. *PRUNE: Preserving Proximity and Global Ranking for Network Embedding*. In Proc. of NeurIPS, pages 5257–5266, 2017.

[13] L. Rasmy, Y. Xiang, Z. Xie, C. Tao, and D. Zhi. *Med-BERT: pretrained contextualized embeddings on large-scale structured electronic health records for disease prediction*. In npj Digital Medicine, 2021.

[14] L. Guo, Y. Wen, and X. Wang. *Exploiting Pre-Trained Network Embeddings for Recommendations in Social Networks*. Journal of Computer Science and Technology, 33:682–696, 2018.

[15] F. Monti, F. Frasca, D. Eynard, D. Mannion, and M. M. Bronstein. *Fake News Detection on Social Media using Geometric Deep Learning*. ArXiv, abs/1902.06673, 2019.

[16] D. Zügner, O. Borchert, A. Akbarnejad, and S. Günnemann. *Adversarial Attacks on Graph Neural Networks: Perturbations and Their Patterns*. ACM Trans. Knowl. Discov. Data, 14(5), 2020.

[17] X. Chen, B. Kang, J. Lijffijt, and T. D. Bie. *Adversarial Robustness of Probabilistic Network Embedding for Link Prediction*. In Proc. of PKDD/ECML Workshops, 2021.

[18] A. Grover and J. Leskovec. *node2vec: Scalable feature learning for networks*. In Proc. of KDD, pages 855–864, 2016.

[19] S. Cao, W. Lu, and Q. Xu. *GraRep: Learning graph representations with global structural information*. In Proc. of CIKM, pages 891–900, 2015.

[20] D. Wang, P. Cui, and W. Zhu. *Structural deep network embedding*. In Proc. of KDD, pages 1225–1234, 2016.

[21] Y. Qian, Y. Zhang, Q. Wen, Y. Ye, and C. Zhang. *Rep2Vec: Repository Embedding via Heterogeneous Graph Adversarial Contrastive Learning*. In Proc. of KDD, page 1390–1400. ACM, 2022.

[22] J. Cao, W. Xu, D. Jin, X. Zhang, A. Miller, L. Liu, and W. Ding. *A Network Embedding-Enhanced NMF Method for Finding Communities in Attributed Networks*. IEEE Access, pages 1–1, 2022.

[23] A. Agibetov. *Neural graph embeddings as explicit low-rank matrix factorization for link prediction*. Pattern Recognition, 133:108977, 2023.

[24] H. Chang, Y. Rong, T. Xu, W. Huang, H. Zhang, P. Cui, W. Zhu, and J. Huang. *A Restricted Black-Box Adversarial Framework Towards Attacking Graph Embedding Models*. In Proc. of AAAI, volume 34, 2020.

[25] S. Geisler, T. Schmidt, H. Şirin, D. Zügner, A. Bojchevski, and S. Günnemann. *Robustness of Graph Neural Networks at Scale*. In Proc of NeurIPS, 2021.

[26] L. Lin, E. Blaser, and H. Wang. *Graph Structural Attack by Perturbing Spectral Distance*. In Proc. of KDD, 2022.

[27] S. Freitas, D. Yang, S. Kumar, H. Tong, and D. H. Chau. *Evaluating Graph Vulnerability and Robustness using TIGER*. Proc. of CIKM, 2021.

[28] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. *Intriguing properties of neural networks*. In Proc. of ICLR, 2014.

[29] I. J. Goodfellow, J. Shlens, and C. Szegedy. *Explaining and Harnessing Adversarial Examples*. CoRR, abs/1412.6572, 2015.

[30] A. Schwarzschild, M. Goldblum, A. Gupta, J. P. Dickerson, and T. Goldstein. *Just How Toxic is Data Poisoning? A Unified Benchmark for Backdoor and Data Poisoning Attacks*. In Proc. of ICML, 2021.

[31] I. Evtimov, K. Eykholt, E. Fernandes, T. Kohno, B. Li, A. Prakash, A. Rahmati, and D. Song. *Robust Physical-World Attacks on Machine Learning Models*. CoRR, abs/1707.08945, 2017. arXiv:1707.08945.

[32] Y. Vorobeychik and M. Kantarcioglu. *Adversarial machine learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning, 12(3):1–169, 2018.

[33] H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, and L. Song. *Adversarial Attack on Graph Structured Data*. In Proc. of ICML, volume 80, pages 1115–1124. PMLR, 2018.

[34] W. Jin, Y. Li, H. Xu, Y. Wang, S. Ji, C. C. Aggarwal, and J. Tang. *Adversarial Attacks and Defenses on Graphs*. ACM SIGKDD Explorations Newsletter, 22:19–34, 2020.

[35] A. Mara, J. Lijffijt, and T. De Bie. *An Empirical Evaluation of Network Representation Learning Methods*. Big Data, 2022. doi:10.1089/big.2021.0107.

[36] X. Wan, H. Kenlay, R. Ru, A. Blaas, M. A. Osborne, and X. Dong. *Adversarial Attacks on Graph Classifiers via Bayesian Optimisation*. In Proc. of NeurIPS, volume 34, pages 6983–6996. Curran Associates, Inc., 2021.

[37] T. Mikolov, K. Chen, G. Corrado, and J. Dean. *Efficient Estimation of Word Representations in Vector Space*. In Proc. of ICLR, 2013.

[38] B. Perozzi, R. Al-Rfou, and S. Skiena. *DeepWalk: Online Learning of Social Representations*. In Proc. of KDD, pages 701–710, 2014.

[39] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. *LINE: Large-scale information network embedding*. In Proc. of WWW, pages 1067–1077, 2015.

[40] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu. *Asymmetric transitivity preserving graph embedding*. In Proc. of KDD, pages 1105–1114, 2016.

[41] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang. *Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and Node2vec*. In Proc. of WSDM, page 459–467, 2018.

[42] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang. *Community Preserving Network Embedding*. In Proc. of AAAI, pages 203–209, 2017.

[43] M. E. J. Newman. *Mixing patterns in networks*. Physical Review E, 67 026126:1024–1034, 2003.

[44] A. Mara, J. Lijffijt, and T. De Bie. *EvalNE: A framework for network embedding evaluation*. SoftwareX, 17, 2022. doi:10.1016/j.softx.2022.100997.

[45] C. L. Giles, K. D. Bollacker, and S. Lawrence. *Citeseer: an automatic citation indexing system*. In Proc. of ICDL, pages 89–98. ACM Press, 1998.

[46] A. K. Mccallum, K. Nigam, J. Rennie, and K. Seymore. *Automating the Construction of Internet Portals with Machine Learning*. Information Retrieval, 3:127–163, 2000.

[47] L. A. Adamic and N. Glance. *The Political Blogosphere and the 2004 U.S. Election: Divided They Blog*. In Proc. of LinkKDD, page 36–43. ACM, 2005.

[48] J. Leskovec and A. Krevl. *SNAP Datasets: Stanford Large Network Dataset Collection*, 2015.

[49] R. A. Rossi and N. K. Ahmed. *The Network Data Repository with Interactive Graph Analytics and Visualization*. In Proc. of AAAI, 2015. Available from: https://networkrepository.com.

[50] B. Goethals, W. Le Page, and M. Mampaey. *Mining Interesting Sets and Rules in Relational Databases*. In Proc. of SAC, pages 997–1001, 2010.

# 6

# Learning Representations of Signed Networks

*Our experimental evaluation in Chapter 4 revealed the excellent performance of a particular probabilistic model, i.e. Conditional Network Embedding (CNE). In this chapter, we leverage the same principles introduced in CNE and propose CSNE a conditional signed network embedding and sign prediction model. Hereby, the present chapter highlights two key benefits of the contributions in this dissertation. On one hand, CNE's potential was identified through our benchmark study in Chapter 4, on the other, the model was evaluated using the EvalNE framework proposed in Chapter 3. The experimental evaluation and comparison with state-of-the-art models in the present chapter, was also conducted using EvalNE. This work is based on the manuscript referenced below. Some section headlines, tables and figures have been altered to ensure consistency across thesis chapters.*

⋆ ⋆ ⋆

**A. Mara, Y. Mashayekhi, J. Lijffijt, and T. De Bie.** ***CSNE: Conditional Signed Network Embedding.***

**Proceedings of CIKM, Oct. 2020.**

**Abstract** Signed networks are mathematical structures that encode positive and negative relations between entities such as friend/foe or trust/distrust. Recently,

several papers studied the construction of useful low-dimensional representations (embeddings) of these networks for the prediction of missing relations or signs. Existing network embedding methods for sign prediction, however, generally enforce different notions of status or balance theories in their optimization function. These theories, are often inaccurate or incomplete which negatively impacts method performance.

In this context, we introduce conditional signed network embedding (CSNE). Our novel probabilistic approach models structural information about the signs in the network separately from fine-grained detail. Structural information is represented in the form of a prior, while the embedding itself is used for capturing fine-grained information. These components are then integrated in a rigorous manner. CSNE's accuracy depends on the existence of sufficiently powerful structural priors for modeling signed networks, currently unavailable in the literature. Thus, as a second main contribution, which we find to be highly valuable in its own right, we also introduce a novel approach to construct priors based on the Maximum Entropy (MaxEnt) principle. These priors can model node *polarity* (tendency to make more positive/negative connections) and signed *triangle counts* (a measure of the degree to which structural balance holds in a network).

Experiments on a variety of real-world networks confirm that CSNE outperforms the state-of-the-art on sign prediction tasks. Moreover, the MaxEnt priors on their own, while less accurate than full CSNE, achieve accuracies competitive with the state-of-the-art at very limited computational cost. Thus, they provide an excellent runtime-accuracy trade-off in resource-constrained scenarios.

## 6.1 Introduction

In recent years, signed networks have become prominent in online media representing friend/foe relations [1], trust networks describing trust/distrust [2], natural language processing for modeling synonyms and antonyms [3], and sentiment analysis denoting positive or negative opinions [4]. Signed networks are powerful data representations describing complex interactions between heterogeneous entities represented as nodes (or vertices). Generally, we refer to these interactions or node-pair connection as edges or links. In contrast to classical networks, where links simply denote a relation between entities, in signed networks these relations can be 'positive' (e.g., friend, trust) or 'negative' (e.g., foe, distrust).

An important task on signed networks is the prediction of signs between node-pairs for which this sign is unknown. This contrasts with link prediction, where the purpose is to predict whether any given node-pair with unknown link status should be linked or not. Sign prediction is an important problem in practice owing to the high cost of acquiring information on the signs in many domains. In social networks, for example, interactions between users (e.g., messages exchanged, or

connections made) can be efficiently tracked. Unveiling the positive or negative nature of the relations between users (i.e. edge signs) from these interactions, however, is more challenging. In a trust network it may be easy to keep track of who engaged in a transaction with whom, but much less easy to understand the mutual trust relations underlying these transactions. Note that sign prediction not only allows one to predict the sign of a link in the network, it also allows one to predict the sign of a link the existence of which is not established. In social networks, for example, this would allow one to assess whether two people are likely to become friends or foes if introduced to each other. Sign prediction has thus been the topic of many recent studies [5–8].

A particular class of approaches extensively used for the analysis of *unsigned* networks consists of representation learning methods, also called network embedding methods [9, 10]. These techniques model nodes as low-dimensional vectors in $\mathbb{R}^d$. The underlying idea is that similar nodes in the graph are mapped to close-by vectors in the embedding space. Using these representations, traditional machine learning methods can be applied on network data to perform downstream tasks such as link prediction [11, 12], information diffusion [13–15], and node classification [16].

Despite their success, however, unsigned network embedding techniques are not directly applicable to signed networks. Indeed, in signed networks node-pairs can have three possible states: unlinked, positively linked, and negatively linked. Unsigned network embedding methods only consider two possible states (unlinked and linked). Thus, to apply them unaltered to signed networks, one would have to ignore the distinction between two of the three states, e.g., ignoring the difference between negatively linked and unlinked. This is problematic, as the semantics of these states are often very different. For example, unlinked node-pairs are often to be interpreted as having an *unknown* sign. Moreover, for many signed networks it may be useful to account for balance theory [17] in the model, which is conceptually impossible without distinguishing these three possible states of node-pairs. Balance theory suggests that triads with an odd number of negative connections are unstable and, thus, less likely to form or persist over time. For example, in a trust network structural balance means one is less likely to trust someone who is distrusted by a friend.

To overcome the aforementioned difficulties, several embedding methods for signed networks have been proposed, such as SiNE [6], Signet [3] and SNE [5]. These methods adopt random walk or probabilistic strategies to learn the structure of signed networks and have been shown to outperform unsigned embedding methods for downstream tasks, such as sign prediction. Nevertheless, these methods present two major drawbacks. On one hand, they often impose constraints on the learning process that *strictly* enforce status or balance theories, even though these theories generally only hold to a certain extent [18]. On the other hand,

they often suffer from overfitting as only the connected node-pairs are used in the embedding learning process.

In this context, we propose a novel probabilistic approach which we coin Conditional Signed Network Embedding (CSNE). The idea of CSNE is to find an embedding for the nodes in the network that is maximally informative of the signs in the network *conditioned* on a probabilistic prior for these signs. These two distributions are then combined in a rigorous manner. The rationale is the observation that embeddings are often unsuited for representing complex structural information, while they are strong at representing fine-grained local information [19]. Thus, if we are able to use a probabilistic prior to represent structural information about the network, both types of information can be effectively leveraged for sign prediction. Representing structural and fine-grained information separately has the additional advantage of allowing us to conveniently learn the prior from data, i.e. learn the extent to which structural balance and other properties hold for each network. Moreover, this approach reduces the amount of information the embedding must represent. This, in turn, allows CSNE to use lower embedding dimensionalities, without compromising accuracy, and effectively prevent overfitting.

More specifically, inspired by Conditional Network Embedding (CNE) [10], we construct a likelihood function for the embedding (i.e. a distribution for the signs conditional on the embeddings), and infer the embedding using the Maximum Likelihood principle. The key idea of CSNE is to construct this likelihood function using Bayes rule from a prior distribution for the signs, and a conditional distribution for the embedding conditioned on the signs. This approach allows us to conveniently model structural information in a suitable prior while the embedding itself can focus exclusively on learning fine-grained local connectivity relations between nodes that are more easily expressed in a metric space.

CSNE could work, at least in principle, with any prior for the signs. Yet, in this chapter we propose a novel approach for inferring such priors. Our flexible approach can accommodate different types of structural information, however, in this chapter we focus on two particular types that have the potential to significantly improve sign prediction performance. The first is that within a network, some nodes may have a more positive (or negative) inclination than others. When modeling, for instance, social interactions, certain users will be liked more often, while others are disliked more often depending on their personality, ideology etc. For new connections formed by these users, we may expect such tendency to persist. We refer to this underlying tendency as the *polarity* of a node. The second is that structural balance rarely holds exactly, but it may hold to a certain extent. The extent to which it holds can be quantified in terms of the number of balanced triangles (where structural balance holds) versus the number of unbalanced triangles (where it does not hold) in the network. For example, if trust in a trust network is being breached at a high rate, structural balance will hold less strongly. Indeed, even if in

an equilibrium situation structural balance would hold, occasional breach of trust can disrupt this balance.

Unfortunately, no approaches have been proposed in the literature modeling such information, for use as a prior in CSNE. Thus, building upon recent work on efficient Maximum Entropy (MaxEnt) models for *unsigned* graphs [20], we show how both polarity and the degree to which structural balance holds can be modeled in an accurate and highly scalable manner. This approach is a second main result in this chapter.

The **main contributions** of this chapter are:

- Overcoming conceptual and technical challenges we propose a new probabilistic method for signed network embedding and sign prediction. Our method, which we coin Conditional Signed Network Embedding (CSNE) computes the most informative embeddings w.r.t. a prior on the network structure.

- We design a new edge-independent MaxEnt prior specifically for signed networks. This prior models two key structural properties of signed networks: the proportion of positive to negative connections of each node or node *polarity* and the number of balanced and unbalanced triangles formed by each edge or *triangle counts*.

- We provide extensive experiments, comparing with existing methods for signed network embedding and sign prediction.

The **benefits of these contributions** are as follows:

- CSNE is a *conceptually novel* and *mathematically rigorous* probabilistic approach to signed network embedding, capable of modeling both structural and fine-grained information. Its *modular design* means that future research developments on priors for signed networks have the potential to further boost CSNE's performance.

- The proposed MaxEnt prior, which is *conceptually simple and intuitive*, is *extremely efficient* to compute (an order of magnitude faster than SIGNet, the fastest state-of-the-art method for sign prediction). When used on its own for sign prediction, it already *uniformly outperforms* all state-of-the-art baselines evaluated across a diversity of commonly used benchmark networks.

- CSNE with the proposed MaxEnt prior *further improves upon this accuracy* for sign prediction, while requiring computation times equal to or only slightly higher than the fastest state-of-the-art baseline method (SIGNet).

The remainder of the chapter is organized as follows. In Section 6.2, we briefly discuss the related work. In Section 6.3, we introduce our proposed method. For ease of exposition, we first discuss the MaxEnt prior, before discussing CSNE and how the prior can be used as a building block in CSNE. Our experimental setup is described in Section 6.4. We report the results in Section 6.5. Finally, Section 6.6 concludes this chapter and summarizes open questions.

## 6.2 Related work

This chapter is primarily related to the analysis of signed networks and the prediction of missing signs [5–8]. Our modular approach, additionally, incorporates ideas from two related fields.

First, CSNE connects to a large body of research in the field of unsigned network representation learning. Early approaches such as Laplacian Eigenmaps [21] and Locally Linear Embeddings [22] were motivated by dimensionality reduction. More recently, embedding methods have been used to bridge the gap between traditional machine learning and network structured data [23–25]. Methods such as DeepWalk [9] and Node2vec [11] have been proposed to learn embeddings by leveraging a random walk strategy on the graphs. Other approaches such as GraRep [26] and AROPE [27] aim to model high order proximities between nodes in the networks. A recent probabilistic approach, CNE, can efficiently incorporate prior information in the embedding learning process. This method has also been shown in a recent empirical study to largely outperform other embedding methods for the task of link prediction [28]. For these reasons, our research incorporates ideas introduced in CNE to the context of signed network embedding. As mentioned in the previous section, however, these methods are not directly applicable to the analysis of signed network.

Second, the proposed probabilistic approach for modeling structural properties of signed networks borrows ideas from the field of maximum entropy random graph models [29–31], and most directly from [32]. Our approach additionally incorporates recent results from [20] to address a fundamental challenge of these models: the difficulty of modeling structural properties, which tend to introduce dependencies between the variables involved, without losing computational tractability.

Several methods for the specific task of signed network embedding have been proposed. One of the first approaches in this field is SNE [5], which computes embeddings through a log-bilinear model that incorporates link types in a given path in the network. More recent approaches [3, 6, 33], impose constraints in the optimization process to enforce different notions of structural balance. SiNE [6] proposes a deep learning framework for signed network embedding and incorporates the extended structural balance theory [34]. SIGNet [3] is a scalable node

embedding method for signed networks. Akin to CSNE, this method aims to incorporate the notion of balance theory introduced in [17]. Targeted node sampling [35], which extends negative sampling techniques from classical word2vec algorithms, is used to maintain structural balance in higher order neighborhoods. And in [33], the authors propose SSNE, a method for embedding directed signed networks that considers status theory ( [2, 36]) in its learning phase. The status of a node is determined based on positive and negative links and a ranking is computed. In contrast to these methods, CSNE does not assume a notion of balance must hold exactly. Instead, our method learns the extent to which balance holds for every network. Finally, in another recent work, two methods nSNE and lSNE were proposed for learning node and edge embeddings in signed networks [8].

## 6.3 Method

In this section we introduce CSNE. We start in Section 6.3.1 by introducing basic concepts and notation for signed networks and sign prediction. In Section 6.3.2, we recap some prior research on MaxEnt graph models and introduce a novel edge-independent MaxEnt distribution that allows one to efficiently model polarity and different types of triangle counts (to model the extent to which structural balance holds) in signed networks. Finally, in Section 6.3.3 we describe the overall CSNE method, and how the MaxEnt model can be used as a building block thereof. In what follows, we limit our analysis exclusively to undirected networks.

### 6.3.1 Concepts and notation

**Signed networks** We represent an undirected signed network by $G = (V, E)$ with $|V| = n$ nodes and $|E| = m$ edges where $E \subseteq \binom{V}{2}$. Each node $i \in V$ denotes an entity and edges $\{i, j\} \in E$ represent unordered relations between two entities $i$ and $j$. Edges $\{i, j\}$ can describe positive or negative relations in signed networks. A function $s : E \to \{-1, +1\}$ is used to map edges to their respective signs. We denote the set of positive links i.e. $\{\{i, j\} \in E \mid s(\{i, j\}) = 1\}$ by $E^+$, and negative links $\{\{i, j\} \in E \mid s(\{i, j\}) = -1\}$ by $E^-$. The adjacency matrix of a signed network $G$ is represented as $\hat{\mathbf{A}} \in \mathcal{A} = \{-1, 0, 1\}^{n \times n}$ with entries $\hat{a}_{ij} \in \{-1, 0, 1\}$, and with $\hat{a}_{ij} = 0$ if $\{i, j\} \notin E$ and $\hat{a}_{ij} = s(\{i, j\})$ otherwise.[1]

**Sign prediction** This task amounts to, given an observed signed network $G = (V, E)$, inferring the signs $s(\{i, j\}) \in \{-1, 1\}$ of unobserved node-pairs $\{i, j\} \in$

[1] In some settings, it might be useful to consider a fourth possible state of a node-pair $\{i, j\} \in \binom{V}{2}$: linked (i.e. $\{i, j\} \in E$), but with unknown sign (i.e. $s(\{i, j\})$ unknown). Although semantically this is different, from a methodological perspective the distinction between such node-pairs and unlinked node-pairs is irrelevant, as the only node-pairs used for training in any existing method (including ours) are those for which the sign is known.

$\binom{V}{2} \setminus E$.

### 6.3.2 A MaxEnt distribution for edge signs

Maximum entropy random graph models are statistical models used for network inference based on the *principle of maximum entropy* [29]. This principle states that the best estimate of a distribution given certain constraints is the one with highest entropy amongst those satisfying said constraints. In practice, these constraints are often derived from empirical data.

In the context of *unsigned* networks, MaxEnt modeling works as follows. Given an observed *binary* adjacency matrix $\hat{\mathbf{A}} \in \mathcal{A}_{bin} = \{0,1\}^{n \times n}$ and a set of statistics (e.g., node degrees, assortativity, densities of particular blocks, etc.) considered characteristic for this network, one seeks the MaxEnt distribution $P$ over random $\mathbf{A} \in \mathcal{A}_{bin}$ such that the expected values of these statistics are equal to the values empirically observed on $\hat{\mathbf{A}}$. For instance, one can seek the MaxEnt distribution $P$ which preserves the node degrees of $\hat{\mathbf{A}}$ i.e. $\mathbb{E}_P[\sum_i \mathbf{A}_{i,j}] = \sum_i \hat{\mathbf{A}}_{i,j}$.

Here, our objective is to learn a MaxEnt distribution $P(\mathbf{A})$ for the *signs* of all node-pairs $\{i,j\} \in \binom{V}{2}$. We use a similar formulation to that of MaxEnt models for unsigned graphs, with a key difference: the random variables are now the signs $s(\{i,j\})$ instead of whether a node-pair is connected or not. In summary, we aim to find a distribution over the set $\mathcal{A}_{sign} = \{-1,1\}^{n \times n}$ of all possible signed matrices $\mathbf{A}$ of size $n$, with expected values for certain important statistics equal to their values on the empirically observed signs. Of course, as the signs are known only for the linked node-pairs, the statistics are computed based on the linked node-pairs only.

As observed by various authors [20, 37], two major drawbacks when fitting MaxEnt models are scalability and generalization to arbitrary constraints. Thus, a key challenge in the development of MaxEnt models is to identify statistics that are characteristic for the data, while using them as constraints is computationally tractable. We already argued that node polarity statistics, and statistics on the number of balanced/unbalanced triangles, are both useful to use as constraints. Now, we will show that their use as constraints in MaxEnt modeling is computationally tractable as well. Our approach is based on recent work which identifies a broad class of statistics that can be used to formulate these constraints [20].

Let $\mathbf{F} \in \mathbb{R}^{n \times n}$ denote a real-valued matrix with $f_{ij}$ the element on row $i$ and column $j$, referred to as a *feature matrix*. Rephrased for our current context, in [20] the authors then demonstrate that constraints on statistics of the form

$$\gamma(\mathbf{A}) \triangleq \sum_{\{i,j\} \in E} f_{ij} a_{ij}, \tag{6.1}$$

can be used efficiently for MaxEnt modeling. Note that these statistics must satisfy the requirement of pertaining to the edges only such that their empirical values

can be computed. They can also be specialized to compute polarity and triangle statistics. Next, we introduce $n$ feature matrices which we specialize to compute the polarity of each node and three additional matrices for computing the number of triangles with particular sign patters.

**Polarity** We formally define the polarity of a node as the sum of the signs of edges incident to this node. To model the polarity of each node $l \in V$, we propose to use one feature matrix $\mathbf{F}_l$ as an indicator function of the $l$th row, i.e.

$$f_{ij}^{l} = \begin{cases} 1, & \text{iff } i = l \\ 0, & \text{otherwise.} \end{cases} \tag{6.2}$$

Indeed, for such a feature matrix, $\sum_{\{i,j\}\in E} f_{ij}^{l} a_{ij} = \sum_{j:\{l,j\}\in E} a_{lj}$, equal to the polarity of node $l$ as required. Let us denote the corresponding polarity statistic for node $l$ as $\gamma_l$.

**Counting triangles with various sign patterns** In addition to the polarity statistics (one per node), we also use three statistics that jointly describe how many triangles the network contains with three positive signs (+++ triangles), with two positive signs (++- triangles), with one positive sign (+-- triangles), and with all negative signs (--- triangles). Note that the total number of triangles is fixed, so three of these constraints will suffice. Moreover, using any linear combination of these statistics would be equivalent. In particular, it is mathematically convenient to count the following three statistics which jointly can be used:

- **Triangle statistic** $\gamma_{++}$ This statistic is defined as the number of +++ triangles minus the number of ++- triangles. This can be counted in the form of Eq. (6.1) by defining $f_{ij}$ as the number of wedges, or paths of length two between $i$ and $j$, with two positive signs, or formally:

$$f_{ij}^{++} = \sum_{k:\{k,i\}\in E \wedge \{k,j\}\in E} \frac{\hat{a}_{ik}+1}{2}\frac{\hat{a}_{kj}+1}{2}. \tag{6.3}$$

  Then $\sum_{\{i,j\}\in E} f_{ij}^{++}\hat{a}_{ij} = \sum_{\{i,j\}\in E^+} f_{ij}^{++} - \sum_{\{i,j\}\in E^-} f_{ij}^{++}$, which is indeed equal to the number of +++ triangles minus the number of ++- triangles.

- **Triangle statistic** $\gamma_{+-}$ This statistic is defined as the number of ++- triangles minus the number of +-- triangles. It can be computed as in Eq. (6.1) by defining $f_{ij}^{+-}$ as the number of wedges with differing signs that connect $i$ and $j$, or formally:

$$f_{ij}^{+-} = \sum_{k:\{k,i\}\in E \wedge \{k,j\}\in E} \frac{1-\hat{a}_{ik}\hat{a}_{kj}}{2}. \tag{6.4}$$

- **Triangle statistic** $\gamma_{--}$ This statistic is defined as the number of +-- triangles minus the number of --- triangles. It can be computed as in Eq. (6.1) by defining $f_{ij}^{--}$ as the number of wedges with two minus signs connecting $i$ and $j$, or formally:

$$f_{ij}^{--} = \sum_{k:\{k,i\}\in E \wedge \{k,j\}\in E} \frac{1-\hat{a}_{ik}}{2}\frac{1-\hat{a}_{kj}}{2}. \tag{6.5}$$

Note that these three statistics jointly, together with the fixed overall triangle count, indeed uniquely define the number of triangles with any sign pattern. For example, denoting the total number of triangles as $t$, it is easy to verify that the number of ++- triangles can be computed as $\frac{t+\gamma_{--}+2\gamma_{+-}-\gamma_{++}}{4}$.

Thus the MaxEnt distribution can be found by solving the following convex optimization problem:

$$\begin{aligned} &\arg\max_{P(\mathbf{A})} \quad -\mathbb{E}_P[\log P(\mathbf{A})], \\ \text{s.t.} \quad &\mathbb{E}_P[\gamma_l(\mathbf{A})] = c_l \quad \forall l = 1,\dots,n, \\ &\mathbb{E}_P[\gamma_{++}(\mathbf{A})] = c_{++}, \\ &\mathbb{E}_P[\gamma_{+-}(\mathbf{A})] = c_{+-}, \\ &\mathbb{E}_P[\gamma_{--}(\mathbf{A})] = c_{--}, \end{aligned} \tag{6.6}$$

where $c_l = \gamma_l(\hat{\mathbf{A}})$, $c_{++} = \gamma_{++}(\hat{\mathbf{A}})$, $c_{+-} = \gamma_{+-}(\hat{\mathbf{A}})$, and $c_{--} = \gamma_{--}(\hat{\mathbf{A}})$ denote the empirically observed values of the statistics.

As shown in [20], the formulation in Eq. (6.6) factorizes as a product of independent Bernoulli distributions. The solution, adapted to the case of signed networks, is of the form:

$$P(\mathbf{A}) = \prod_{\{i,j\}\in E} P(a_{ij}=1)^{\frac{\hat{a}_{ij}+1}{2}}(1-P(a_{ij}=1))^{1-\frac{\hat{a}_{ij}+1}{2}}. \tag{6.7}$$

Moreover, the 'success probabilities' $P(a_{ij}=1)$ for node-pairs $\{i,j\}$ are equal to:

$$P(a_{ij}=1) = \frac{\exp(\sum_{l=1}^{n} f_{ij}^{l}\lambda_l + f_{ij}^{++}\lambda_{++} + f_{ij}^{+-}\lambda_{+-} + f_{ij}^{--}\lambda_{--})}{1+\exp(\sum_{l=1}^{n} f_{ij}^{l}\lambda_l + f_{ij}^{++}\lambda_{++} + f_{ij}^{+-}\lambda_{+-} + f_{ij}^{--}\lambda_{--})}, \tag{6.8}$$

where $\lambda_l, \lambda_{++}, \lambda_{+-}$, and $\lambda_{--}$ denote the Lagrange multipliers associated with the respective constraints in Eq. (6.6). The optimal values of these Lagrange multipliers can be found by unconstrained minimization of the convex Lagrange dual

function:

$$L(\lambda_1, \ldots, \lambda_M) = \tag{6.9}$$
$$\sum_{\{i,j\}\in E} \log(1 + \exp(\sum_{l=1}^{n} f_{ij}^{l}\lambda_l + f_{ij}^{++}\lambda_{++} + f_{ij}^{+-}\lambda_{+-} + f_{ij}^{--}\lambda_{--}))$$
$$- \sum_{l=1}^{n} c_l\lambda_l - c_{++}\lambda_{++} - c_{+-}\lambda_{+-} - c_{--}\lambda_{--}.$$

This optimization problem can be solved efficiently for very large networks by using equivalences between the Lagrange multipliers [20]. In our proposed model, this reduces the number of free variables to $O(\sqrt{n})$. Moreover, we can further improve efficiency by limiting our computations to the connected node-pairs for which signs are known. Finally, we observe that after fitting the MaxEnt prior, a probability for each node-pair of being positive or negative is obtained. These unique probabilities, determined by the polarities of the connected nodes and triangle counts, can be directly used for sign prediction.

### 6.3.3 Conditional Signed Network Embedding

As introduced in Section 6.1, the MaxEnt model is required as a prior for CSNE. For ease of exposition, the prior was discussed in the previous section. In the current section, we introduce CSNE proper and explain how the MaxEnt prior is an integral component. However, before diving into the details of CSNE, we formally define the task of signed network embedding.

Embedding approaches learn a function $g : V \rightarrow \mathbb{R}^d$ which maps nodes in the network to d-dimensional real-valued vectors. These representations are generally denoted as $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)' \in \mathbb{R}^{n\times d}$, where $\mathbf{x}_i$ is the embedding corresponding to node $i$. A common modeling assumption for NE methods is that similar nodes in the network must be mapped to close-by representations in the embedding space. For the particular case of signed network embedding, we additionally require the function $g : V \rightarrow \mathbb{R}^d$ to map pairs $\{i, j\} \in E^-$ to more distant representations $\mathbf{x}_i$ and $\mathbf{x}_j$ in the embedding space, than pairs $\{i, j\} \in E^+$. At the same time, unlinked node-pairs are commonly not used in the embedding learning process i.e. their distance in unimportant.

CSNE aims to learn the most informative embedding $\mathbf{X}$ for a given signed network $G = (V, E)$ with adjacency matrix $\hat{\mathbf{A}}$. In simple terms, the objective is to find the embedding $\mathbf{X}$ that maximizes the likelihood of observing the signs on the edges in $\hat{\mathbf{A}}$. We formulate this optimization task as a Maximum Likelihood Estimation (MLE) problem, $argmax_{\mathbf{X}} P(\mathbf{A}|\mathbf{X})$. Akin to [10], we do not postulate the likelihood function $P(\mathbf{A}|\mathbf{X})$ directly. Instead, we *do* postulate the density function of the embedding $\mathbf{X}$ conditioned on the signed network, i.e. $p(\mathbf{X}|\mathbf{A})$. Then, we combine $p(\mathbf{X}|\mathbf{A})$ with the MaxEnt prior discussed in Section 6.3.2, $P(\mathbf{A})$, by means of the Bayes formula. The likelihood function, thus, follows

as $P(\mathbf{A}|\mathbf{X}) = \frac{p(\mathbf{X}|\mathbf{A})P(\mathbf{A})}{p(\mathbf{X})}$. Independently modeling the density function of the embedding conditioned on the network and the MaxEnt distribution on the signs of $G$, has one major advantage. The prior, can encode properties of the network which do not have to be reflected by the learned embedding $\mathbf{X}$. Effectively, this means that CSNE can a make better use of the embedding space.

The MaxEnt prior $P(\mathbf{A})$ has already been introduced in Section 6.3.2 and its expression given in Eq. 6.7. For postulating the conditional density $p(\mathbf{X}|\mathbf{A})$, we only need to model the distances between connected node-pairs $\{i, j\}$. As metric, we use the Euclidean distance between the embeddings of the end nodes, i.e. $d_{ij} \triangleq ||\mathbf{x}_i - \mathbf{x}_j||_2$. As already discussed, we require the distances between positively connected node-pairs $\{i, j\} \in E^+$ to be lower than those between negatively connected pairs $\{i, j\} \in E^-$. To model this, we use two half-normal distributions with locations at 0 and different spread parameters. Distances between pairs $\{i, j\} \in E^+$ are generated from a half-normal distribution with spread parameter $\sigma_1$ and between pairs $\{i, j\} \in E^-$ from a similar distribution with parameter $\sigma_2$, where $\sigma_1 < \sigma_2$. Thus, we have:

$$p(d_{ij} \mid \{i, j\} \in E^+) = \mathcal{N}_+\left(d_{ij}|\sigma_1^2\right), \tag{6.10}$$
$$p(d_{ij} \mid \{i, j\} \in E^-) = \mathcal{N}_+\left(d_{ij}|\sigma_2^2\right). \tag{6.11}$$

The conditional density $p(\mathbf{X}|\mathbf{A})$ can then be expressed as follows:

$$p(\mathbf{X}|\mathbf{A}) = \prod_{\{i,j\}\in E^+} \mathcal{N}_+\left(d_{ij}|\sigma_1^2\right) \cdot \prod_{\{i,j\}\in E^-} \mathcal{N}_+\left(d_{ij}|\sigma_2^2\right). \tag{6.12}$$

The resulting likelihood function to optimize is:

$$\begin{aligned} P(\mathbf{A}|\mathbf{X}) &= \frac{p(\mathbf{X}|\mathbf{A})P(\mathbf{A})}{p(\mathbf{X})} = \frac{p(\mathbf{X}|\mathbf{A})P(\mathbf{A})}{\sum_{\mathbf{A}} p(\mathbf{X}|\mathbf{A})P(\mathbf{A})} \\ &= \prod_{\{i,j\}\in E^+} \frac{\mathcal{N}_+\left(d_{ij}|\sigma_1^2\right) P(a_{ij}=1)}{\mathcal{N}_+\left(d_{ij}|\sigma_1^2\right) P(a_{ij}=1) + \mathcal{N}_+\left(d_{ij}|\sigma_2^2\right)(1-P(a_{ij}=1))} \\ &\cdot \prod_{\{i,j\}\in E^-} \frac{\mathcal{N}_+\left(d_{ij}|\sigma_2^2\right)(1-P(a_{ij}=1))}{\mathcal{N}_+\left(d_{ij}|\sigma_1^2\right) P(a_{ij}=1) + \mathcal{N}_+\left(d_{ij}|\sigma_2^2\right)(1-P(a_{ij}=1))}. \end{aligned} \tag{6.13}$$

In order to maximize the likelihood function Eq. (6.13) we use block stochastic gradient descent. The derivation of the gradient follows closely the one in [10]. Thus, we refer the user to this manuscript for more details. The gradient of the log-likelihood function in CSNE w.r.t. the embedding $\mathbf{x}_i$ of node $i$ reads:

$$\frac{\partial\, log(P(\mathbf{A}|\mathbf{X}))}{\partial\, \mathbf{x_i}} = 2 \sum_{j:\{i,j\}\in E^+} (\mathbf{x}_i - \mathbf{x}_j) P(a_{ij} = -1|\mathbf{X}) \left(\frac{1}{\sigma_2^2} - \frac{1}{\sigma_1^2}\right) + 2 \sum_{j:\{i,j\}\in E^-} (\mathbf{x}_i - \mathbf{x}_j) P(a_{ij} = 1|\mathbf{X}) \left(\frac{1}{\sigma_1^2} - \frac{1}{\sigma_2^2}\right). \quad (6.14)$$

Intuitively, the first summation in the gradient expression pulls the embedding of each node close to the embeddings of positively connected neighbors. The second summation, on the other hand, pushes this embedding far from those of negatively connected nodes. Finally, after fitting the model, sign predictions can be directly extracted from the posterior probability matrix.

## 6.4 Experimental setup

To evaluate the performance of CSNE, we conducted experiments on six networks from four sources, which we introduce in Section 6.4.1. We compared the empirical observations for CSNE with four recent methods for signed network embedding that are discussed in Section 6.4.2. Specifics of the test setup, including hyperparameters settings for all methods are outlined in Section 6.4.3. Reproducibility details are provided in Section 6.4.4.

### 6.4.1 Datasets

We performed sign prediction evaluation using 6 different real-world datasets. The first two networks constitute snapshots from the Slashdot [2] [36] social news website, where users can create friends (positive links) and foes (negative links). These snapshots, Slashdot(a) and Slashdot(b), were obtained in November 2008 and February 2009, respectively. Epinions [36] is a product review website, where users can trust (positive links) or distrust (negative links) each other. Wiki-rfa [38] contains votes of Wikipedia users endorsing or opposing candidates for adminship. Neutral votes, also present in the data, are not used. The dataset contains information on votes between 2003 and 2013. Lastly, we used two Bitcoin cryptocurrency trust networks, Bitcoin-$\alpha$ [3] and Bitcoin-otc [4]. These networks, obtained from [39], were gathered in order to identify transactions with fraudulent users.

Directed networks were preprocessed to remove the directions of links. In the experiments, we also used the largest connected components only and ignored self-loops. In Table 6.1 we summarize some key network statistics.

[2] http://slashdot.org
[3] http://www.btcalpha.com
[4] https://www.bitcoin-otc.com

*Table 6.1: Main statistics of the networks used for sign prediction evaluation.*

| Data | Slashdot(a) | Slashdot(b) | Epinions | Wiki-rfa | Bitcoin-$\alpha$ | Bitcoin-otc |
|---|---|---|---|---|---|---|
| $\|V\|$ | 77350 | 82140 | 131828 | 11258 | 3783 | 5881 |
| $\|E\|$ | 468554 | 500481 | 711210 | 171562 | 14124 | 21492 |
| $\|E\|/\|V\|$ | 12.11 | 12.18 | 10.78 | 30.47 | 7.46 | 7.30 |
| % $\|E^+\|/\|E\|$ | 75% | 76% | 83% | 77% | 90% | 85% |
| % Bal. Tri. | 85% | 86% | 89% | 73% | 83% | 85% |

### 6.4.2 Baseline methods

To be able to interpret the performance of CSNE in the context of existing methods, we performed the same sign prediction experiments using the following methods:

*SiNE* [6] uses a deep neural network architecture to learn node embeddings. The objective function optimized by SiNE satisfies the structural balance theory. In this method, nodes are expected to be closer to their friends than their foes. If no negative connections exist for specific nodes, virtual nodes and negative links to these are generated.

*nSNE* [8], similarly to SiNE, uses a deep neural network to learn node embeddings by leveraging second order proximities in the graph. At the same time, the method learns a mapping from node embeddings to edge embeddings.

*lSNE* [8] is a simplified version of nSNE where the mapping function from node embeddings to edge embeddings is assumed to be linear. The function can therefore be learned via gradient descent.

*SIGNet* [3] uses a random walk strategy to determine node similarity on the graph and the Skip-Gram model to obtain node embeddings. The authors propose an extension to the negative sampling used in word2vec models to perform targeted node sampling and accommodate the main concepts of structural balance theory (i.e., balanced triangles are more likely than unbalanced triangles).

### 6.4.3 Sign prediction evaluation

**General setup** As introduced in Section 6.3.1, sign prediction amounts to identifying the signs of unobserved connections between nodes in a given network. For performance evaluation of sign prediction, it is common to divide the given set of edges ($E$) into two disjoint subsets: the train edges ($E_{train}$) are used in the model learning phase, while the test edges ($E_{test}$) are used for assessing the prediction performance of the methods. The train and test sets are constructed such that $E_{test} \cup E_{train} = E$, and $E_{test} \cap E_{train} = \emptyset$. At training time, edges in $E_{test}$ are removed and the corresponding value in the adjacency matrix is set to 0. At test time, the model is evaluated on all $\{i, j\} \in E_{test}$ and predictions compared to $\hat{a}_{ij}$. In our evaluation, we selected 80% of the total edges $\{i, j\} \in E$ for

training ($E_{train}$) and the remaining 20% for testing ($E_{test}$). Train edges were selected regardless of their sign, using the default sampling strategy of the `EvalNE` toolbox [40], which ensures that the training network remains connected. The sets $E_{train}$ and $E_{test}$ are expected to contain similar proportions of positive and negative edges as the original graph. Unless otherwise specified, all results reported are averages over three independent repetitions of the experiment with different train and test sets.

**Edge embeddings** For prediction, nSNE, lSNE and our proposed methods can directly return the probability of an edge $\{i, j\}$ of being positive or negative. For SiNE and SIGNet, however, this is not the case. These methods only return node embeddings from which predictions must be derived. As shown in [41] an effective approach for obtaining predictions, in this case, is through binary classification on the edge embeddings derived from node embeddings. A classifier is first trained on the embeddings of edges in $E_{train}$ and their corresponding signs $\{-1, 1\}$. Then, predictions for node-pairs in $E_{test}$ can be computed. To obtain the embedding of a link $\{i, j\}$ from node representations one can apply different operators $\circ$ to the embeddings of the incident nodes $i$ and $j$ i.e. $\mathbf{x}_{ij} = \mathbf{x}_i \circ \mathbf{x}_j$. In our evaluation, we selected the operators introduced in [11], namely *Average* ($(\mathbf{x}_i + \mathbf{x}_j)/2$), *Hadamard* ($\mathbf{x}_i \cdot \mathbf{x}_j$), *Weighted* $L_1$ ($|\mathbf{x}_i - \mathbf{x}_j|$) and *Weighted* $L_2$ ($|\mathbf{x}_i - \mathbf{x}_j|^2$). The choice of operator was tuned as additional method hyperparameters for SiNE and SIGNet and Logistic Regression with 5 fold cross validation of the regularization parameter was used as binary classifier.

**Hyperparameters** Throughout our evaluation, we set number of dimensions $|d| = 20$ for all methods. For CSNE we fixed the spread parameters $\sigma_1 = 1$ and $\sigma_2 = 2$. We ran SiNE, nSNE, lSNE and CSNE for 500 iterations. This value was empirically found to provide best results in preliminary experiments. Fitting a MaxEnt prior is a convex problem for which we use second order information, therefore, we limited the number of iterations in this case to 20. We performed method hyperparameter tuning on a validation set obtained by further splitting $E_{train}$ in 80% training and 20% validation. The specific method hyperparameters we tuned are as follows. For SiNE, we tuned $\delta = \delta_0 \in \{0.5, 1\}$ and the edge embedding operator. For SIGNet we only tuned the edge embedding operator. For nSNE and lSNE, we varied $\lambda \in \{5e-5, 2.5e-5, 1e-5\}$ and $\beta \in \{0.5, 0.05, 0.005, 0\}$. Finally, for CSNE, including the MaxEnt priors, we did not tune any hyperparameters.

**Evaluation metric** We evaluated the methods in terms of Area Under the Curve for the Receiver Operating Characteristic (AUC-ROC). This metric is popular for

binary classification tasks and well suited for prediction in the case of class imbalance. Let TP, TN, FP and FN be the elements of a confusion matrix. Then, we can compute the true positive rate as $TPR = \frac{TP}{TP+FN}$ and the false positive rate as $FPR = \frac{FP}{FP+TN}$. The AUC is then the area under the ROC curve created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various thresholds.

### 6.4.4 Reproducibility notes

To ensure the reproducibility of our experimental evaluation we used the `EvalNE` toolbox [40]. This Python toolbox aims to simplify and standardize the evaluation of network embedding methods on various downstream tasks. `EvalNE` uses configuration files that detail the complete evaluation setup. Our configuration file together with an implementation of CSNE and instructions for replicating our experiments are available online [5]. This file, together with the datasets and methods reported in Sections 6.4.1 and 6.4.2 allows full reproduction of our experiments.

## 6.5 Experimental results

In this section we present and discuss quantitative and qualitative experimental results. Quantitative results, on one hand, are shown for the task of sign prediction. A qualitative evaluation, on the other, is performed through visualization of the signed embeddings learned by CSNE on a small network representing relations between characters in the Harry Potter novels.

### 6.5.1 Sign prediction performance

We start in Table 6.2 by presenting the AUC scores for each method on all evaluated datasets. For CSNE we present the results using two MaxEnt priors with different sets of constraints, i.e. node polarity only (pol.) and node polarity combined with balanced/unbalanced triangle counts (pol.+tri.). These MaxEnt priors, as discussed in Section 6.3.2, can be independently used for sign prediction. Therefore, we also include their performance in Table 6.2 as *Prior(pol.)* and *Prior(pol.+tri.)*, respectively.

Our results showcase the superior performance of CSNE over the baseline methods on all datasets. The largest difference in AUC to the best performing baseline (2.7%) can be seen for Slashdot(b) while the lowest (0.4%) can be found for Epinions. Additionally, CSNE exhibits a more consistent performance on different networks as compared to other baselines. SIGNet, the best performing baseline on Bitcoin-$\alpha$, performs significantly worse than CSNE, nSNE and lSNE on

[5] https://github.com/aida-ugent/CSNE

*Table 6.2: Sign prediction AUC for all networks. Best performing method per dataset is highlighted in bold.*

| Methods | Slashdot(a) | Slashdot(b) | Epinions | Wiki-rfa | Bitcoin-$\alpha$ | Bitcoin-otc | Avg. AUC Rank |
|---|---|---|---|---|---|---|---|
| SiNE | 0.850±0.002 | 0.856±0.002 | 0.898±0.001 | 0.816±0.002 | 0.835±0.012 | 0.857±0.005 | 8 |
| nSNE | 0.895±0.001 | 0.894±0.002 | 0.950±0.001 | 0.879±0.002 | 0.810±0.024 | 0.868±0.010 | 4.5 |
| lSNE | 0.886±0.002 | 0.893±0.002 | 0.941±0.002 | 0.872±0.001 | 0.854±0.022 | 0.907±0.005 | 6 |
| SIGNet | 0.887±0.002 | 0.893±0.002 | 0.943±0.001 | 0.848±0.005 | 0.893±0.002 | 0.928±0.006 | 4.5 |
| Prior(pol.) | 0.879±0.001 | 0.885±0.001 | 0.929±0.001 | 0.870±0.001 | 0.858±0.014 | 0.891±0.002 | 7 |
| Prior(pol.+tri.) | 0.888±0.001 | 0.894±0.001 | 0.937±0.001 | 0.875±0.001 | 0.874±0.009 | 0.914±0.004 | 3 |
| CSNE(pol.) | 0.902±0.001 | 0.908±0.001 | 0.952±0.001 | 0.886±0.002 | 0.896±0.009 | 0.930±0.002 | 2 |
| CSNE(pol.+tri.) | **0.904±0.001** | **0.909±0.001** | **0.954±0.000** | **0.887±0.002** | **0.899±0.006** | **0.936±0.002** | 1 |

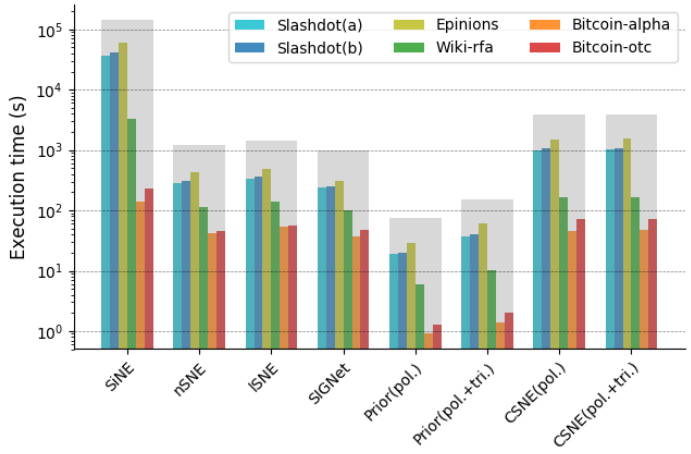



*Figure 6.1: Execution times of all methods for sign prediction on each evaluated network. Gray boxes indicate the total runtime per method on all datasets.*

Wikipedia. Similarly, nSNE, the best baseline on Slashdot(a), Slashdot(b) and Epinions obtains poor results on Bitcoin-$\alpha$ and Bitcoin-otc.

The results in Table 6.2 also show that using balanced and unbalanced triangle counts as additional structural constraints to the node polarity in the MaxEnt prior, always results in improved AUC scores. This effect is most prominent when comparing *Prior(pol.)* and *Prior(pol.+tri.)*, where the latter obtains higher accuracy scores across the board. A similar, yet less prominent effect can be observed when these priors are used as part of CSNE. Another interesting observation from Table 6.2 is that the proposed *Prior(pol.+tri.)* already provides state-of-the-art results while being much faster than other methods, as shown by our runtime experiments.

In Figure 6.1 we present the execution times in seconds (all experiments were run on a machine equipped with an Intel(R) Core i7-7700K processor and 32GB of RAM), including hyperparameter tuning, for all methods. For CSNE, the results include prior computation. Each colour in the figure represents a different network and the gray boxes indicate, per method, the corresponding cumulative execution times on all networks (i.e. sum of all colored bars). An immediate observation from Figure 6.1 is that the total execution time of SiNE (gray bar in the Figure 6.1), is approximately two orders of magnitude larger than those of other methods. The cumulative execution times on all networks of the two MaxEnt priors, on the other hand, are approximately one order of magnitude lower than those of the fastest evaluated baseline, SIGNet. We also observe that the two CSNE variants are not significantly slower, on most networks, than other baseline methods.

In Figure 6.2 we group the methods by dataset and present their execution

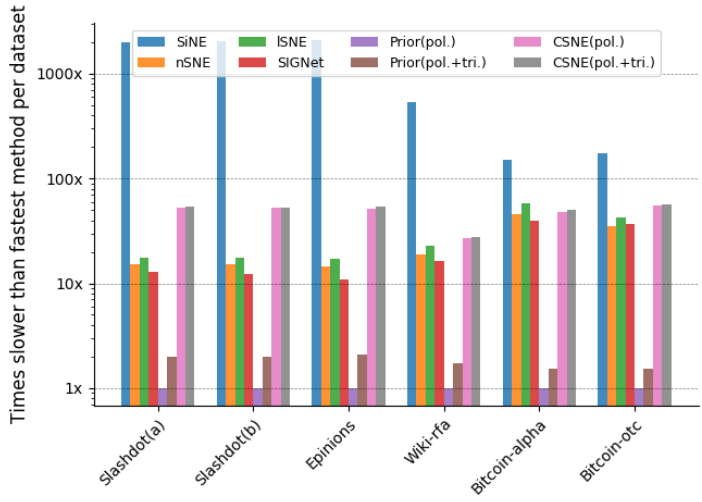



*Figure 6.2: Relative execution times of all methods compared to the fastest approach on each dataset (lower is better).*

times relative to the fastest method in each case. Firstly, we observe that *Prior(pol.)* is the fastest method on all networks and that *Prior(pol.+tri.)* is never more than 2x slower. The remaining methods, with the exception of SiNE, are between 10x and 50x slower. For SiNE, we observe execution times that are up to 2000 times those our proposed *Prior(pol.)* approach on the two Slashdot and the Epinion networks. Finally, we observe that the relative speedup of *Prior(pol.)* over the baselines becomes smaller as the network sizes increase. This effect is due to parallelization, present in the baseline implementations and not in the MaxEnt prior, which becomes more relevant for larger networks. Parallelization of the MaxEnt prior is left as future work.

### 6.5.2 Hyperparameter sensitivity

We evaluated the sensitivity of CSNE with respect to three hyperparameters, i.e. train set size, embedding dimensionality and the spread parameter $\sigma$.

**Train set size** First, we assessed the generalization performance of the proposed method from different amounts of initial training data. We did this by dividing the set of all graph edges $E$ in sets $E_{train}$ and $E_{test}$ of different sizes. We started by using 35% of all edges for training and 65% for testing. We then gradually increased the size of $E_{train}$ to 50%, 65% and finally 80% while the size of $E_{test}$ scaled accordingly 50%, 35% and 20%. Using the same setting as in Section 6.4.3 we performed sign prediction evaluation for these different edge splits. For com-

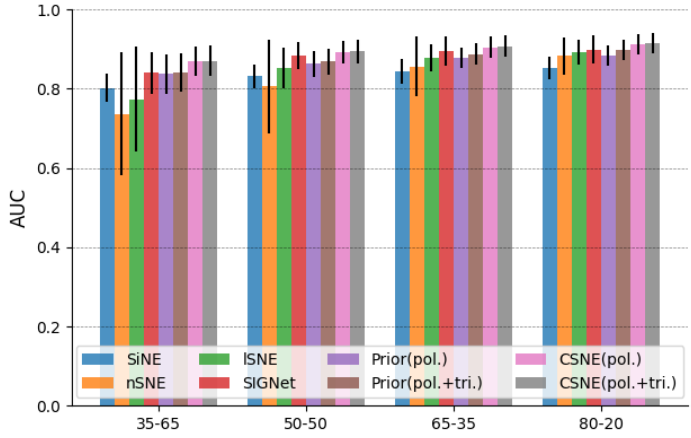



*Figure 6.3: Average sign prediction AUC over all evaluated networks for varying sizes of $E_{train}$ - $E_{test}$. A black line over each bar denotes the 95% confidence interval for the estimate of the average performance.*

parison, we also included the AUC scores of the baseline methods. The average AUC scores of each method over all evaluated networks, are summarized in Figure 6.3. The 95% confidence intervals are also presented for each case.

The results show that the performance of lSNE, nSNE and SIGNet degrades significantly as the size of $E_{train}$ decreases. CSNE, the MaxEnt prior and SiNE, on the other hand, are more robust to changes in the size of $E_{train}$. The tight confidence intervals for these three methods also indicate a consistent performance across different datasets. For lSNE, nSNE and SIGNet these ranges are larger, especially when little training data is available.

**Embedding dimensionality** Another fundamental parameter for signed network embedding methods is the size of the resulting embeddings. We studied the performance of CSNE w.r.t. the dimensionality for $d \in \{2, 4, 8, 16, 32\}$. The results, depicted in Figure 6.4a, show a consistent performance of the method for all values of this parameter. Even for values as low $d = 2$, the performance of out method is excellent. This indicates that CSNE can be directly used for visualization without the need to use additional tools.

**Spread parameter $\sigma$** The two $\sigma$ parameters introduced in Section 6.3.3 determine, on one hand, the spread of the Gaussian distributions from which the distances between positively and negatively connected pairs are generated in Eq. (6.12). These parameters also control the strength of the pull and push effects in Eq. (6.14). Therefore, to understand the effect on performance of different values for these parameters, we conducted and additional experiment. In this experiment, we set

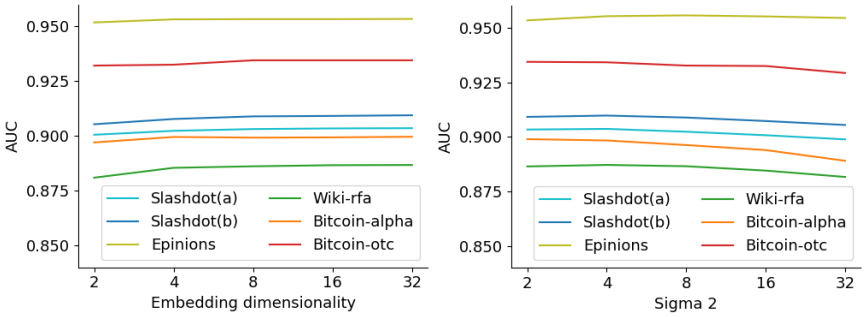



*Figure 6.4: Sign prediction AUC scores of CSNE(pol.+tri.) for (a) different embedding dimensions $d$ and (b) different values of $\sigma_2$.*

$\sigma_1 = 1$, as this simply fixes the scale, and varied $\sigma_2 \in \{2, 4, 8, 16, 32\}$. The results, summarized in Figure 6.4b, show the robustness of CSNE to changes of this parameter. Only very large values of $\sigma_2$ i.e. $\sigma_2 = 16$ and $\sigma_2 = 32$ appear to have a slight effect on method performance. As such, $\sigma_1 = 1$ and $\sigma_2 = 2$ appear to be good default values in most cases.

### 6.5.3 Convergence analysis

In this section we present and discuss the convergence of the MaxEnt prior and of the complete CSNE approach. In Figures 6.5a and 6.5b we plot, for each method, the sign prediction AUC against the gradient descent iteration number on all datasets introduced in Table 6.1. In both cases, we used the joint polarity and triangle count prior. Computing the MaxEnt prior amounts to optimizing a convex function as discussed in Section 6.3.2. By leveraging second order information, convergence is achieved in less than 20 iterations for all datasets as shown in Figure 6.5a. In CSNE, maximizing the likelihood function is a non-convex optimization problem solved via block stochastic gradient descent. Convergence is achieved in approximately 200 iterations as shown in Figure 6.5b.

### 6.5.4 A case study: Visualization

We also performed a qualitative evaluation of the CSNE embeddings on the popular Harry Potter network [6]. Nodes in the graph correspond to characters in the novels while edges denote friend or enemy relations extrapolated from the character interactions throughout the novels. We preprocessed the original directed network to obtain an undirected representation, extracted the main connected component

[6] https://github.com/efekarakus/potter-network/tree/master/data

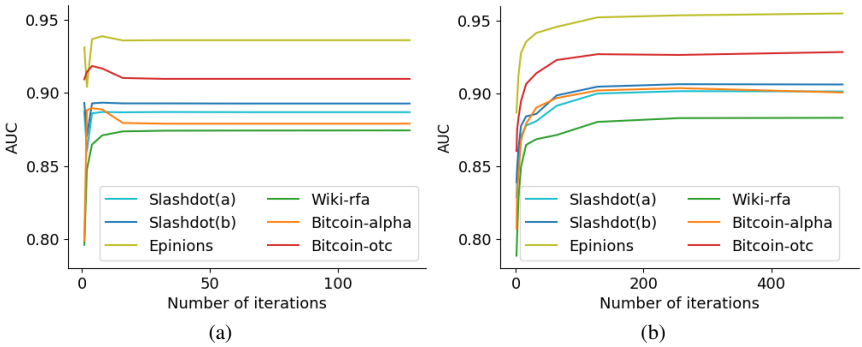



*Figure 6.5: Convergence of (a) the MaxEnt prior and (b) CSNE.*

and removed self loops. The resulting network $G$ contained $n = 65$ nodes and $m = 453$ edges with an average degree of 14.

We computed a 2-dimensional embedding $\mathbf{X}$ of $G$ using CSNE with a MaxEnt structural prior encoding node polarity and the number of balanced and unbalanced triangles. We used 100 iterations to fit the prior and another 100 iterations to learn the embeddings. We obtained an initial assessment of the embedding quality by performing sign prediction for all edges $E$ of $G$. The resulting AUC score is $0.994$.

The embeddings learned by CSNE are presented in Figure 6.6 where blue links denote friendship relations and red links denote enemy relations. The main protagonists and antagonists of the novels are presented as blue and red circles, respectively. Two clear clusters can be identified, a larger one corresponding to the protagonists and allies and a smaller one for the antagonists. The relations within each cluster are mostly positive while between clusters are negative. To verify that, as expected, positively connected nodes are, on average, closer to each other than negatively connected ones, we used the Euclidean distance. The average Euclidean distance obtained for all positively connected pairs $\{i, j\} \in E^+$ is $0.745$ with a standard deviation of $0.525$. For the negatively connected pairs, $\{i, j\} \in E^-$, the obtained distance is $3.360 \pm 1.014$. This experiment shows that CSNE is able to effectively capture the structure of a signed network in dimensionalities as low as $d = 2$. This also showcases the potential of CSNE for signed network visualization.

## 6.6 Conclusions

In this chapter, we have presented a new probabilistic approach for learning representations of signed networks with applications to visualization and sign prediction. The proposed CSNE method solves an MLE problem which seeks the

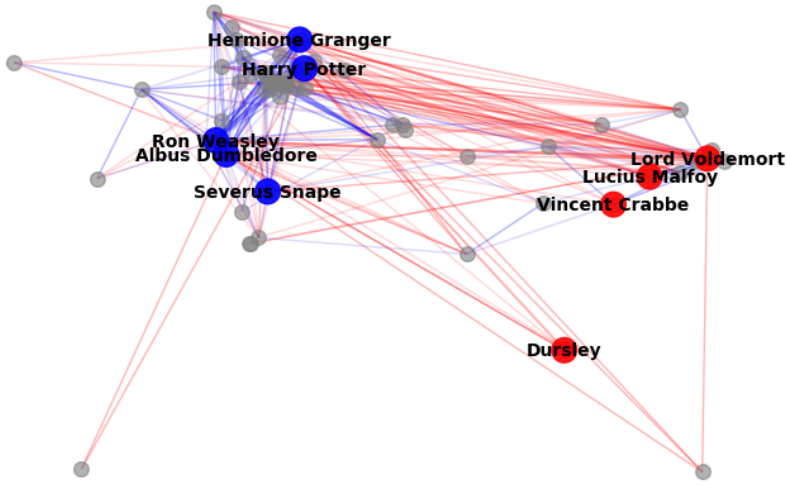



*Figure 6.6: Plot showing the CSNE embeddings of the Harry Potter network. Enemity relations between characters are highlighted in red, while friendships are highlighted in blue. A subset of protagonists and antagonists are shown in blue and red, respectively.*

embeddings that maximize the probability of observing the signs on the edges of an input graph. Our optimization process models certain structural properties of the data as a MaxEnt prior. Particularly, this prior captures node polarity and structural balance i.e. as counts of balanced and unbalanced triangles. Our experimental results indicate that CSNE can adequately model the specific properties of signed networks and outperforms other baselines for sign prediction. Additionally, we have showed that the proposed MaxEnt priors can also be directly used for sign prediction, resulting in state-of-the-art AUC scores with runtimes up to 50x lower than those of other baselines. Our work opens up several avenues for further research and improvements. On one hand, more sophisticated MaxEnt priors can be designed specifically for networks with particular structures, such as k-partiteness. Another possible line of work is to tackle the extension of CSNE to directed signed networks.

## Acknowledgment

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] J. Kunegis, A. Lommatzsch, and C. Bauckhage. *The slashdot zoo: mining a social network with negative edges*. In Proc. of WWW, pages 741–750, 2009.

[2] R. Guha, R. Kumar, P. Raghavan, and A. Tomkins. *Propagation of trust and distrust*. In Proc. of WWW, pages 403–412. ACM, 2004.

[3] M. R. Islam, B. A. Prakash, and N. Ramakrishnan. *SIGNet: scalable embeddings for signed networks*. In Proc. of PAKDD, pages 157–169. Springer, 2018.

[4] H. Wang, F. Zhang, M. Hou, X. Xie, M. Guo, and Q. Liu. *SHINE: Signed Heterogeneous Information Network Embedding for Sentiment Link Prediction*. In Proc. of WSDM, page 592–600, New York, NY, USA, 2018. ACM.

[5] S. Yuan, X. Wu, and Y. Xiang. *SNE: signed network embedding*. In Proc. of PAKDD, pages 183–195. Springer, 2017.

[6] S. Wang, J. Tang, C. Aggarwal, Y. Chang, and H. Liu. *Signed network embedding in social media*. In Proc. of SDM, pages 327–335. SIAM, 2017.

[7] J. Kim, H. Park, J.-E. Lee, and U. Kang. *Side: representation learning in signed directed networks*. In Proc. of WWW, pages 509–518, 2018.

[8] W. Song, S. Wang, B. Yang, Y. Lu, X. Zhao, and X. Liu. *Learning node and edge embeddings for signed networks*. Neurocomputing, 319:42–54, 2018.

[9] B. Perozzi, R. Al-Rfou, and S. Skiena. *DeepWalk: Online Learning of Social Representations*. In Proc. of KDD, pages 701–710, 2014.

[10] B. Kang, J. Lijffijt, and T. De Bie. *Conditional Network Embeddings*. Proc. of ICLR, 2019.

[11] A. Grover and J. Leskovec. *node2vec: Scalable feature learning for networks*. In Proc. of KDD, pages 855–864, 2016.

[12] H. Wang, F. Zhang, M. Hou, X. Xie, M. Guo, and Q. Liu. *Shine: Signed heterogeneous information network embedding for sentiment link prediction*. In Proc. of WSDM, pages 592–600. ACM, 2018.

[13] S. Gao, H. Pang, P. Gallinari, J. Guo, and N. Kato. *A novel embedding method for information diffusion prediction in social network big data*. IEEE Transactions on Industrial Informatics, 13(4):2097–2105, 2017.

[14] C. Li, J. Ma, X. Guo, and Q. Mei. *Deepcas: An end-to-end predictor of information cascades*. In Proc. of WWW, pages 577–586, 2017.

[15] S. Bourigault, S. Lamprier, and P. Gallinari. *Representation learning for information diffusion through social networks: an embedded cascade model*. In Proc. of WSDM, pages 573–582. ACM, 2016.

[16] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. *LINE: Large-scale information network embedding*. In Proc. of WWW, pages 1067–1077, 2015.

[17] F. Heider. *Attitudes and Cognitive Organization*. The Journal of Psychology, 21(1):107–112, 1946. PMID: 21010780. doi:10.1080/00223980.1946.9917275.

[18] J. Leskovec, D. Huttenlocher, and J. Kleinberg. *Predicting Positive and Negative Links in Online Social Networks*. In Proc. of WWW, page 641–650, New York, NY, USA, 2010. ACM.

[19] J. Lu, J. Xuan, G. Zhang, and X. Luo. *Structural property-aware multilayer network embedding for latent factor analysis*. Pattern Recognit., 76:228–241, 2018.

[20] F. Adriaens, A. Mara, J. Lijffijt, and T. D. Bie. *Scalable Dyadic Independence Models with Local and Global Constraints*, 2020. arXiv:2002.07076.

[21] M. Belkin and P. Niyogi. *Laplacian eigenmaps and spectral techniques for embedding and clustering*. In Proc. of NIPS, pages 585–591, 2002.

[22] S. T. Roweis and L. K. Saul. *Nonlinear Dimensionality Reduction by Locally Linear Embedding*. Science, 290(5500):2323–2326, 2000.

[23] D. Liben-Nowell and J. Kleinberg. *The link-prediction problem for social networks*. Journal of the American society for information science and technology, 58(7):1019–1031, 2007.

[24] S. Bhagat, G. Cormode, and S. Muthukrishnan. *Node classification in social networks*. In Social network data analytics, pages 115–148. Springer, 2011.

[25] S. Papadopoulos, Y. Kompatsiaris, A. Vakali, and P. Spyridonos. *Community detection in social media*. Data Mining and Knowledge Discovery, 24(3):515–554, 2012.

[26] S. Cao, W. Lu, and Q. Xu. *GraRep: Learning graph representations with global structural information*. In Proc. of CIKM, pages 891–900, 2015.

[27] Z. Zhang, P. Cui, X. Wang, J. Pei, X. Yao, and W. Zhu. *Arbitrary-Order Proximity Preserved Network Embedding*. In Proc. of KDD, pages 2778–2786, New York, NY, USA, 2018. ACM.

[28] A. Mara, J. Lijffijt, and T. D. Bie. *Network Representation Learning for Link Prediction: Are we improving upon simple heuristics?*, 2020. arXiv:2002.11522.

[29] E. T. Jaynes. *Information Theory and Statistical Mechanics*. Phys. Rev., 106:620–630, May 1957.

[30] J. Park and M. E. J. Newman. *Statistical mechanics of networks*. Physical review. E, Statistical, nonlinear, and soft matter physics, 70:066117, 01 2005.

[31] M. Duijn, K. Gile, and M. Handcock. *A Framework for the Comparison of Maximum Pseudo Likelihood and Maximum Likelihood Estimation of Exponential Family Random Graph Models*. Social Networks, 31:52–62, 01 2009.

[32] T. De Bie. *Maximum Entropy Models and Subjective Interestingness: An Application to Tiles in Binary Databases*. Data Min. Knowl. Discov., 23(3):407–446, November 2011.

[33] C. Lu, P. Jiao, H. Liu, Y. Wang, H. Xu, and W. Wang. *SSNE: Status Signed Network Embedding*. In Proc. of PAKDD, pages 81–93. Springer, 2019.

[34] M. Cygan, M. Pilipczuk, M. Pilipczuk, and J. O. Wojtaszczyk. *Sitting closer to friends than enemies, revisited*. In Proc. of MFCS, pages 296–307. Springer, 2012.

[35] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. *Distributed representations of words and phrases and their compositionality*. In Proc. of NIPS, pages 3111–3119, 2013.

[36] J. Leskovec, D. Huttenlocher, and J. Kleinberg. *Signed networks in social media*. In Proc. of SIGCHI, pages 1361–1370. ACM, 2010.

[37] S. M. Goodreau. *Advances in exponential random graph (p*) models applied to a large social network*. Social networks, 29 2:231–248, 2007.

[38] R. West, H. S. Paskov, J. Leskovec, and C. Potts. *Exploiting social network structure for person-to-person sentiment analysis*. Transactions of the Association for Computational Linguistics, 2:297–310, 2014.

[39] S. Kumar, F. Spezzano, V. Subrahmanian, and C. Faloutsos. *Edge weight prediction in weighted signed networks*. In Proc. of ICDM, pages 221–230. IEEE, 2016.

[40] A. Mara, J. Lijffijt, and T. De Bie. *EvalNE: A Framework for Evaluating Network Embeddings on Link Prediction*. arXiv preprint arXiv:1901.09691, 2019.

[41] S. Gurukar, P. Vijayan, A. Srinivasan, G. Bajaj, C. Cai, M. Keymanesh, S. Kumar, P. Maneriker, A. Mitra, V. Patel, B. Ravindran, and S. Parthasarathy. *Network Representation Learning: Consolidation and Renewed Bearing*. ArXiv, abs/1905.00987, 2019.

# 7 Conclusions and Future Work

In this chapter, we first present the main conclusions of this dissertation. Then, we summarize some of the limitations of the research conducted, propose a series of possible extensions and improvements, and discuss other avenues for future research opened by our work.

## 7.1 Conclusions

Throughout the chapters in this dissertation we have discussed some of the main challenges related to the evaluation of network representation learning methods. In particular, we focused on the ongoing reproducibility crisis, the complexity of evaluation pipelines, lack of standardized evaluations and uncertainties regarding the state-of-the-art. Then, we empirically assessed the effect of these challenges on the progress in the field ranging from non-reproducible evaluations to limited improvements in performance and robustness over an extended time frame.

More specifically, our work on the reproducibility of link prediction evaluations highlighted the importance of discussing design choices such as data preprocessing, edge and non-edge sampling, method implementations and evaluation pipelines. Experimentally, we demonstrated that when these choices are not reported, reproduced results can differ from published ones by up to 30% in AUC scores (e.g. node2vec results in Table 2.2). With access to most of these design choices, on the other hand, results are much more in line with those reported with AUCs varying less than 1% on average (e.g. CNE results in Table 2.5). To ensure

reproducibility and facilitate comprehensive reporting of all experimental design choices, we introduced the EvalNE framework. Later, we presented an extension of this toolbox to streamline performance and robustness evaluation of network representations on all major downstream tasks including: node classification, link prediction, sign prediction, network reconstruction and embedding visualization. To further expand the EvalNE user-base, we also developed a graphical user interface. This GUI allows practitioners to design and monitor evaluation in an intuitive and interactive manner. Then, leveraging the EvalNE framework, we conducted extensive performance and robustness evaluations where we attempted to clarify the current state of research. In terms of performance we only found minor improvements over the last few years. In our experiments, most of the evaluated approaches performed similarly, even when method hyperparameters were tuned. Moreover, for downstream link prediction and network reconstruction, simple heuristics based on topological network properties performed on par with the more complex representation learning models evaluated. Regarding robustness, we found that most of the evaluated approaches are sensitive to poison attacks on the input data. We also found only small differences in robustness between different methods on the same downstream task, but large differences across different tasks. Specifically, we found node classification results to be less robust than network reconstruction predictions. Simultaneously, these performance and robustness evaluations also highlighted the potential of some new models. Building upon one such model, we also contributed a new probabilistic approach for signed network embedding and sign prediction coined CSNE. The proposed model combines ideas from statistical modeling, particularly exponential random graph models, with first order embeddings to achieve state-of-the-art results.

To date, the results presented in this dissertation have already attracted significant research attention. For instance, the EvalNE framework has been used to assess the performance of representation learning models in areas such as, mathematics, biology, and computer science. Our framework has also inspired the creation of other benchmark and reproducibility tools, e.g., GEval, GraphWorld. The concerns we raised in Chapters 2 and 4 regarding reproducibility and the slow progress in network representation learning have also received a fair amount of attention. In these works, we showed that network representation learning is also tainted by the same reproducibility and evaluation crisis affecting other fields. A growing number of authors have since investigated these issues in related research areas such as link prediction and semi-supervised embedding learning. The research works and tools presented in this dissertation constitute a necessary first step towards boosting progress in network representation learning. Inevitably, this dissertation also presents some limitations that should be addressed by future work. In what follows, we will discuss some of these limitations, possible extensions of our work as well as new research directions.

## 7.2 Future Work

In this section, we cover avenues for future research from three different perspectives, namely: reproducibility, evaluation, and representation learning models. In each case, we start with possible extensions to our contributions followed by a broader discussion on each topic.

### 7.2.1 Reproducibility

Firstly, in our reproducibility work in Chapter 2, there are several directions which we did not investigate. For instance, the evaluation is focused exclusively on downstream link prediction. One possible extension could address other tasks, such as node classification or attribute forecasting and assess the extent to which our conclusions continue to apply there. The evaluation also focuses on unsupervised learning methods only. Therefore, a second clear extension would be towards evaluating the reproducibility of results for semi-supervised models such as GNNs.

In addition to the concrete extensions mentioned above, more work highlighting the importance of reproducibility of research is needed. The computer science community, unlike others such as medicine, biology or chemistry, continues to undervalue the importance of reproducible research. Works in this area are often seen as *a waste of resources* that merely retrain existing models or as *channels to undermine researchers credibility* by exposing evaluation issues. In reality, reproducibility is a key aspect that can significantly boost progress. It can help us better understand existing models, identify their strengths and weaknesses, compare with other models and, in the long run, encourage better research practices, higher quality method implementations and standardized evaluations. Only recently, has the scientific community started to take steps towards better reproducibility practices. Major conferences such as ICML, NeurIPS, AAAI or ECML have implemented reproducibility checklists or included reproducibility, in some form or another, in their review processes. At the same time, workshops on reproducibility have also become more prominent. Unfortunately, as long as novelty continues to be the leading factor behind publishability, research on reproducibility will continue to be actively discouraged and an exception to the norm.

### 7.2.2 Evaluation

In this dissertation, we have explored and proposed solutions to important engineering and scientific challenges related to experimental design and evaluation. We have done this, on one hand, through EvalNE and on the other, through the benchmark studies in Chapters 4 and 5. These works, however, present several limitations.

For EvalNE, a major limitation is the lack of integration with other libraries that provide method implementations. Examples of such libraries include `OpenNE`, `GEM` and `Karateclub`. Currently, EvalNE users are required to personally install each library and individual method to be evaluated. This, in turn, requires knowledge of system administration, virtual environments, etc. and can lead to tedious and error-prone setups. By directly integrating these libraries in EvalNE, environment setups can become fully automated while prerequisites on user knowledge disappear. These benefits, however, would come at the cost of increased EvalNE maintenance and excessive reliance on the implementations provided by the integrated libraries (which may not always be the best choice, as we have already shown in Chapter 4). Other possible extensions for EvalNE are towards new downstream tasks (e.g., graph classification), other node-pair embedding operators (e.g., concatenation), and meta-learning automated hyperparameter tuning (e.g., SMAC). The library could also benefit from a more streamlined API.

Our empirical studies on the performance and robustness of network embedding methods also present some limitations and could be further extended. One possibility is expanding the breadth of the evaluations with new models, networks and attack strategies. Another possibility is targeting evaluation depth by comparing models that can leverage additional information such as edge and node attributes, node labels and community memberships. For the robustness study, in particular, one could additionally investigate stronger optimization-based adversarial attacks, local attacks, node manipulations, etc.

Indeed, new evaluation tools and more extensive benchmarking can have a significant positive impact on the progress in network representation learning. With access to reliable, flexible and comprehensive evaluation tools fewer and fewer researchers would opt for developing their own pipelines. This would not only contribute to a higher evaluation consistency across papers, but also to the development and extension of the evaluation tools themselves. Similarly, empirical evaluations provide a variety of benefits, as we already explored in Chapters 1 and 4. Yet these evaluations are still mostly focused on comparing representation learning methods. Works comparing and proposing new benchmark datasets, downstream tasks and evaluation pipelines are required. Projects such as the Open Graph Benchmark, are already taking steps in this direction, yet more research efforts are needed.

### 7.2.3 Representation learning models

Our work on signed network embeddings is used as an example to showcase some of the benefits of our evaluation tool and empirical studies. The proposed model can be further improved in terms of scalability, specifically for the embedding computation part of CSNE. And by investigating new priors capable of capturing

additional structural information. This work also tries to highlight the interesting dynamics of signed networks, their applicability to many real-world problems, and the specific challenges they present (e.g., data sparsity, negative sampling, etc.). Unfortunately, to date these models are still understudied. For instance, there are no empirical evaluations comparing signed embedding approaches, few studies investigate negative sampling in this context, and structural balance continues to be primarily enforced instead of modeled. More work in these areas and on signed network embeddings in general, could result in significant advances in natural language processing, sociology and computer science.

In addition to CSNE, the conditional network embedding framework has also inspired other projects. For example, we are currently investigating the use of MaxEnt priors combined with embeddings for improving user recommendations. The priors could be well suited for capturing 'weak' interactions between users and the system, such as clicks, views, etc. This information can then be combined with embeddings modeling the purchase network in order to provide better recommendations. One could also investigate the use of MaxEnt priors for modeling other types of node information such as sensitive attributes (e.g., for fairness and privacy research) or geographical location (e.g., for efficient modeling of IoT and sensor networks).

Finally, we hope the tools and models provided in this dissertation will inspire better practices on reproducibility, help researchers focus on developing new models instead of evaluation hurdles, and overall accelerate progress in the field of network representation learning.

# A Appendix

*In this Appendix, we provide further details regarding methods, datasets, reproducibility of evaluations as well as additional experimental results for different chapters in this thesis. In what follows, top level headlines indicate the names of different chapters and subheadders contain the relevant additional information.*

⋆ ⋆ ⋆

## A.1 EvalNE: A Framework for Network Embedding Evaluation

### A.1.1 Configuration file example

In this section, we present an example configuration file for EvalNE. This file replicates the experimental setup of the node2vec manuscript (Grover et. al., 2016). A high level description of the parameters shown in Fig. A.1 is given below and more detailed explanations can be found in the EvalNE repository under *examples/conf.ini*.

**General** In this category the user can indicate parameters such as the downstream task to be evaluated (LP, SP, NR, NC) the number of repetitions of the experiment with different edge splits over which results are averaged or the embedding dimensionality, among several others.

```
; conf.ini

# Conf file that replicates the experimental section of the paper:
# "node2vec: Scalable Feature Learning for Networks"

[GENERAL]
TASK = lp
LP_NUM_EDGE_SPLITS = 3
NC_NUM_NODE_SPLITS =
NC_NODE_FRACS =
NR_EDGE_SAMP_FRAC =
# Run once for each ee method: average, hadamard, weighted_l1 and weighted_l2
EDGE_EMBEDDING_METHODS = average
LP_MODEL = LogisticRegression
EMBED_DIM = 128
TIMEOUT = None
VERBOSE = True
SEED = 42

[NETWORKS]
NAMES = Facebook PPI ArXiv
INPATHS = ../data/Facebook/facebook_combined.txt
          ../data/PPI/ppi.edgelist
          ../data/Astro-PH/CA-AstroPh.txt
DIRECTED = False
SEPARATORS = '\s' ',' '\t'
COMMENTS = '#' '#' '#'
LABELPATHS =

[PREPROCESSING]
RELABEL = True
DEL_SELFLOOPS = True
SAVE_PREP_NW = False
WRITE_STATS = True
DELIMITER = ','

[EDGESPLIT]
TRAINTEST_FRAC = 0.5
TRAINVALID_FRAC = 0.9
SPLIT_ALG = spanning_tree
OWA = False
FE_RATIO = 1

[BASELINES]
LP_BASELINES = common_neighbours
               jaccard_coefficient
               adamic_adar_index
               preferential_attachment
NEIGHBOURHOOD = in out

[OPENNE METHODS]
NAMES_OPNE =
METHODS_OPNE =
TUNE_PARAMS_OPNE =

[OTHER METHODS]
NAMES_OTHER = node2vec deepWalk line
EMBTYPE_OTHER = ne ne ne
WRITE_WEIGHTS_OTHER = False False True
WRITE_DIR_OTHER = True True True
METHODS_OTHER = python ../methods/node2vec/main.py --input {} --output {} --dimensions {}
--walk-length 80 --num-walks 10 --window-size 10 --workers 8
                python ../methods/node2vec/main.py --input {} --output {} --dimensions {}
--walk-length 80 --num-walks 10 --window-size 10 --workers 8 --p 1 --q 1
                ../methods/LINE/linux/line -train {} -output {} -size {} -order 2 -
samples 100 -negative 5 -rho 0.025 -threads 1
TUNE_PARAMS_OTHER = --p 0.25 0.5 1 2 4 --q 0.25 0.5 1 2 4
INPUT_DELIM_OTHER = '\s' '\s' '\s'
OUTPUT_DELIM_OTHER = '\s' '\s' '\s'

[REPORT]
MAXIMIZE = auroc
SCORES = all
CURVES =
PRECATK_VALS =
```

*Figure A.1: EvalNE configuration file example.*

**Networks** This section contains the parameters related to the experimental data. The user is expected to provide the names, locations, type (directed or undirected) and separator characters of the networks to be evaluated.

**Preprocessing** Common preprocessing functions such as node relabeling or main connected component computations can be indicated in this section.

**Edgesplit** The approach used to split a network's edges in train, validation, and test sets, as well as the relative sizes of these, are indicated in this section.

**Baselines** Heuristics to be used as baselines for the link prediction and network reconstruction tasks can be indicated in this section.

**OpenNE methods** This section contains parameters that indicate the names, and the tunable and fixed hyperparameters for methods in the `OpenNE` library.

**Other methods** Parameters for other methods not part of `OpenNE` can be indicated in this section. This includes their names, output type, command line call (as one would execute the methods on the command line), and method hyperparameters to be tuned.

**Reports** Lastly, in this section the user is expected to provide the score to be maximized when performing model validation, the scores to be reported in the final output tables and the curves to plot for each evaluated method.

### A.1.2 EvalNE code snippet

In this section we present a code snippet showing the use of EvalNE as an API. A high level description of this example is provided in Section 3.4.2 of the manuscript.

```
# Load and preprocess the network
G = pp.load_graph('../../evalne/tests/data/network.edgelist')
G, _ = pp.prep_graph(G)

# Split edges in train and test sets
traintest_split = LPEvalSplit()
traintest_split.compute_splits(G)

# Create an Evaluator and Scoresheet for the results
nee = LPEvaluator(traintest_split)
scoresheet = Scoresheet()

# Evaluate baselines
methods = ['random_prediction', 'common_neighbours', 'jaccard_coefficient']
for method in methods:
    result = nee.evaluate_baseline(method=method)
    scoresheet.log_results(result)

# Set NE method parameters
methods    = ['PRUNE', 'metapath2vec++']
m_types    = ['ne', 'ne']
ee_methods = ['average', 'hadamard']
commands   = [
        'python ../methods/PRUNE/src/main.py --inputgraph {} --output {} --dimension {}',
        '../methods/metapath2vec/metapath2vec -min-count 1 -train {} -output {} -size {}']
input_delim  = [' ', ' ']
output_delim = [',', ' ']

# Evaluate NE methods
for i in range(len(methods)):
    results = nee.evaluate_cmd(methods[i], m_types[i], commands[i],
                               ee_methods, input_delim[i], output_delim[i])
    scoresheet.log_results(results)

# Get output
scoresheet.print_tabular()
```

*Figure A.2: Code snippet showing the use of EvalNE from python code.*

## A.2 Robustness Evaluation of Network Representations

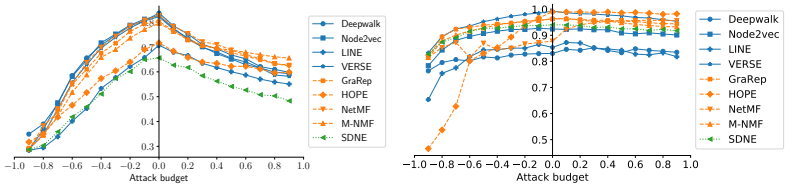
### A.2.1 Additional dataset details

The IIP network represents a set of companies competing in the internet industry between 1998 and 2001. Nodes in the graph denote companies and edges represent business relations such as joint venture, strategic alliance or other type of partnership. The associated node labels denote the company's main business area i.e., content, infrastructure of commerce.

The StudentDB network represents a snapshot of Antwerp University's relational student database. Nodes in the network represent entities, more specifically: students, professors, tracks, programs, courses and rooms. Edges constitute binary relations between them, that is, student-in-track, student-in-program, student-takes-course, professor-teaches-course, and course-in-room. Numerical node labels are assigned according to each node's type.

### A.2.2 Randomized attacks: additional results

In this section we present our additional experiments regarding randomized attacks on node embeddings. We start in Figures A.3a and A.3b by presenting the node classification f1_micro results for the Cora dataset and the network reconstruction AUC scores for PolBlogs.



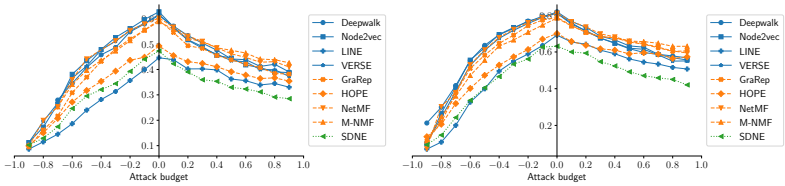

(a) (b)

*Figure A.3: Node classification f1_micro scores on Cora (a), and network reconstruction AUC scores on PolBlogs (b). Negative attack budgets indicate edge deletion.*

In Figures A.4a and A.4b we summarize the f1_macro scores for both Citeseer and Cora and Figures A.5a and A.5b present the average precision on Facebook and PolBlogs.

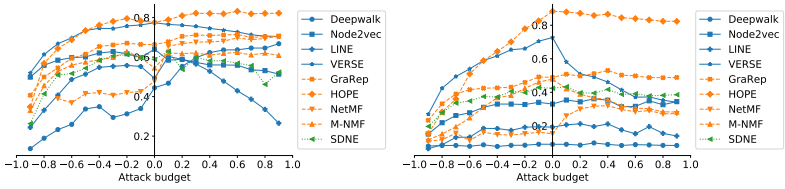
### A.2.3 Other attacks: additional results

We also compare the performance of edge addition, rewiring and deletion on both downstream tasks in terms of f1_micro and average precision. These results sup-




(a) (b)

*Figure A.4: Node classification performance for the Citeseer network (a), and the Cora network (b). Y-axis indicate f1_macro scores. Negative attack budgets indicate edge deletion.*




(a) (b)

*Figure A.5: Network reconstruction performance for the Facebook network (a), and the PolBlogs network (b). Y-axis indicate average precision scores. Negative attack budgets indicate edge deletion.*

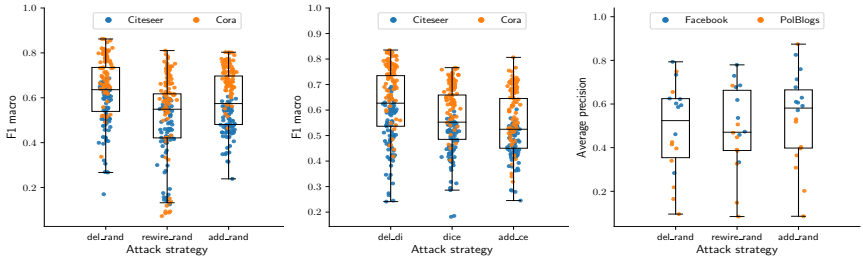



*Figure A.6: Comparison of edge addition, rewiring and deletion attacks for both downstream tasks. The leftmost and center figures present f1_macro scores for random and node label based attacks on node classification. The rightmost figure shows average precision results for random attacks on network reconstruction.*

port our conclusions in Section 5.4.3.3 (see Figure A.6).

## A.3 Learning Representations of Signed Networks

### A.3.1 Additional experiments

In Section 6.4.3 we summarized the AUC scores of different methods for the task of sign prediction using 80% of the edges for training and 20% for testing. In Table A.1 we present similar results for the case of 50% training and 50% testing edges. The results indicate a good performance of CSNE across the board. For the two Bitcoin networks, where CSNE with polarization and triangle counts does not obtain the best AUC scores, the difference to the best performing method (SIGNet) are of only 1% and 0.04%, respectively. The proposed Maxent prior with polarization and triangle counts largely outperforms the SiNE, nSNE and lSNE baselines on all but one dataset –where it obtains comparable results.

*Table A.1: Average sign prediction AUC and standard deviations over 3 repetitions of the experiment using 50/50 train/test edge splits. Best performing method per dataset is highlighted in bold. Note that* 0.000 *in the table means* $< 0.0005$.

| Methods | Slashdot(a) | Slashdot(b) | Epinions | Wiki-rfa | Bitcoin-alpha | Bitcoin-otc | Avg. AUC Rank |
|---|---|---|---|---|---|---|---|
| SiNE | 0.832±0.002 | 0.838±0.002 | 0.882±0.001 | 0.797±0.002 | 0.819±0.019 | 0.826±0.003 | 8 |
| nSNE | 0.865±0.005 | 0.853±0.002 | 0.929±0.000 | 0.851±0.006 | 0.628±0.014 | 0.712±0.007 | 7 |
| lSNE | 0.863±0.003 | 0.849±0.002 | 0.928±0.000 | 0.851±0.002 | 0.776±0.011 | 0.848±0.008 | 6 |
| SIGNet | 0.872±0.000 | 0.879±0.001 | 0.934±0.001 | 0.841±0.003 | **0.875±0.006** | **0.906±0.004** | 3 |
| Prior(pol.) | 0.863±0.001 | 0.870±0.001 | 0.916±0.000 | 0.862±0.001 | 0.821±0.013 | 0.844±0.001 | 5 |
| Prior(pol.+tri.) | 0.868±0.001 | 0.874±0.001 | 0.923±0.000 | 0.865±0.001 | 0.825±0.014 | 0.860±0.002 | 4 |
| CSNE(pol.) | 0.883±0.001 | 0.890±0.000 | 0.942±0.000 | 0.874±0.001 | 0.865±0.013 | 0.902±0.002 | 2 |
| CSNE(pol.+tri.) | **0.884±0.001** | **0.890±0.000** | **0.944±0.000** | **0.874±0.001** | 0.865±0.012 | 0.905±0.002 | 1 |