From node embedding to community embedding on graphs

Cavallari, Sandro

2019

Cavallari, S. (2019). From node embedding to community embedding on graphs. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/85349>

<https://doi.org/10.32657/10220/48265>

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

From Node Embedding To Community Embedding On Graphs

Sandro Cavallari

MAIN SUPERVISOR: Erik Cambria
CO-SUPERVISOR: Vincent W. Zheng

**SCHOOL OF COMPUTER SCIENCE AND
ENGINEERING**

A Dissertation Submitted to the Nanyang Technological University
in Partial Fulfillment of the Requirements of the Degree of Doctor of
Philosophy

2019

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

24/04/2019

.....

Date

.....

Sandro Cavallari

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

24/04/2019

.....
Date

.....
Erik Cambria

Authorship Attribution Statement

Please select one of the following; *delete as appropriate:

*(B) This thesis contains material from 3 papers published in the following peer-reviewed journals / from papers accepted at conferences in which I am listed as an author.

Chapter 3 is published as: Sandro Cavallari, Vincent W. Zheng, Hongyun Cai, Kevin Chen-Chuan Chang, and Erik Cambria. 2017. Learning Community Embedding with Community Detection and Node Embedding on Graphs. In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (CIKM '17). ACM, New York, NY, USA, 377-386. DOI: <https://doi.org/10.1145/3132847.3132925>

The contributions of the co-authors are as follows:

- Ph.D. Vincent W. Zheng provided the initial idea of community embedding and edited the draft paper.
- After some revision of the idea, I have designed, executed and analysed the experiments, also all the coding work was my responsibility.
- Ph.D. Hongyun Cai helped in the evaluation of the community detection experiments; also, she helped in polishing the experiment section of the final manuscript.
- Prof. Kevin Chen-Chuan Chang gave some initial intuition on the gaps present in the graph embedding research field.
- Prof. Erik Cambria supervised the experiment execution and provided some guidance in the experiments interpretation.

Chapter 4 is published as Sandro Cavallari, Erik Cambria, Hongyun Cai, Kevin Chen-Chuan Chang, and Vincent W. Zheng. 2019. Embedding Finite and Infinite Communities on Graphs. IEEE Computational Intelligence Magazine 14.5 (2019) Special Issue.

The contributions of the co-authors are as follows:

- After reviewing the experiments in Chapter 3. I started the new problem settings. I also have done the entire implementation work and conducted all the experiments

(from community embedding to parameter sensitivity). Finally, I have also done most of the writing work.

- Ph.D. Vincent W. Zheng provided the initial idea for a Bayesian approach to community embedding based on the well-known IGM algorithm. He also provided the first draft of the publication and supervised the drawing up of the paper.
- Ph.D. Hongyun Cai helped with the analysis of the related works. She provided a comprehensive review of all the graph embedding works done in the last years. Yet, I have done Bayesian embedding review and conducted the complexity analysis.
- Prof. Kevin Chen-Chuan Chang supervised the execution process.
- Prof. Erik Cambria was in charge of the publication process. He provided the final polish of the manuscript and manages the review process.

Chapter 5 is published as Sandro Cavallari, Vincent W Zheng, Hongyun Cai, Soujanya Poria and Erik Cambria. 2019. An Attention-Based Model for Learning Dynamic Interaction Networks. International Joint Conference on Neural Networks (IJCNN).

The contributions of the co-authors are as follows:

- As for the previous works, I was in charge of doing the experiments as well as writing the code. Moreover, I'm the one who proposes the message passing architecture. Finally, I was the one in charge of writing the paper
- Ph.D. Vincent W Zheng provided the initial problem formulation and the used semantic parametrization of different problem factors
- Ph.D. Hongyun Cai provided some insight on the design of the toy example used to evaluate and study the model performance in a controlled environment
- Ph.D. Soujanya Poria supervised the experiment interpretation and the paper's drawing up
- Prof. Erik Cambria helped with the paper writing and took control of the publication and review process.

24/04/2019

.....
Date

.....
Sandro Cavallari

Acknowledgment

First, I would like to express my sincere gratitude to my advisor Prof. Erik Cambria and my co-supervisor Dr. Vincent W. Zheng for their patience, motivation, and immense knowledge, as well as for the continuous support during my PhD study and related research. Their guidance helped me all the time during my research and while writing this thesis. I could not have imagined having a better advisor and mentor for my PhD.

In addition to my advisor, I would like to thank the rest of my thesis committee: Prof. Quek Hiok Chai, Dr. Antonio Feraco, and Dr. Stefano Mazzoni for their insightful comments and encouragement, and for posing the difficult questions that motivated me to widen my research from various perspectives.

I would like to thank my fellow lab mates for the stimulating discussions, for the sleepless nights we spent working together before deadlines, and for all the fun we have had in the last four years. Last but not the least, I would like to thank my family: my parents and my brother and my cousins for supporting me throughout this long journey.

Contents

Abstract	12
1 Introduction	14
1.1 Background	16
1.1.1 Graphs, Nodes, and Edge Properties	16
1.1.2 Matrix Representation of a Graph	18
1.1.3 Paths and Random Walks	18
1.2 Graph Embedding	19
1.3 Objectives and Major Contributions	22
2 Literature Review	25
2.1 Background	25
2.2 Factorization Methods	26
2.2.1 Graph Laplacian Eigenmaps	27
2.2.2 Matrix Factorization	27
2.3 Deep Learning Models	28
2.3.1 Random Walk Models	28
2.3.2 Non-Random Walk Models	30
2.3.3 Interactive Networks	31
2.4 Knowledge Graph	32
2.5 Community Detection	32
2.6 Bayesian Embedding	33
2.7 Conclusion	33
3 ComE	35
3.1 Introduction	35
3.2 Problem Formulation	38
3.2.1 Community Detection and Embedding	39
3.2.2 Node Embedding	39
3.2.3 Closing the Loop	40
3.3 Inference	41
3.3.1 Algorithm and Complexity	43
3.4 Experiments	44
3.4.1 Experiment Setting	44
3.4.2 Graph Visualization	47
3.4.3 Community Detection	49
3.4.4 Node Classification	51

3.4.5	Model Study	53
3.5	Conclusion	57
4	ComE+	58
4.1	Introduction	58
4.2	Problem Formulation	60
4.2.1	Community Detection and Embedding	61
4.2.2	Closing the Loop	62
4.3	Bayesian Inference	63
4.3.1	Algorithm and Complexity	65
4.4	Experiments	66
4.4.1	Datasets	66
4.4.2	Baselines	66
4.4.3	Parameters and Environment	66
4.4.4	Graph Visualization	67
4.4.5	Community Detection	68
4.4.6	Node Classification	70
4.4.7	Parameter Sensitivity	72
4.4.8	Convergence and Efficiency	76
4.5	Conclusion	77
5	An Attention-Based Model for Learning Dynamic Interaction Networks	78
5.1	Introduction	78
5.2	Dynamic Interaction Attention Network	80
5.2.1	Problem Formulation	80
5.2.2	Model Architecture	81
5.3	Experiments	83
5.3.1	Datasets	83
5.3.2	Baselines	85
5.3.3	Experiments Setup	87
5.3.4	Results	88
5.3.5	Sensitivity	90
5.4	Conclusion	92
6	Conclusion and Future Work	93
6.1	ComE Limitations	94
6.1.1	Random Walk Generative Process	94
6.1.2	Distance-aware Graph Embedding for Semi-supervised Learning	96
6.2	DIAN Limitations	98
	Appendix A	111
	Appendix B	112

Acronyms

Table 1: Acronyms used in this thesis.

Acronyms	Description
e.g.	Exempli gratia, “For example”
et al.	Et alia, “and others”
i.e.	Id est, “In other words”
etc.	Et cetera
w.r.t.	With respect to
s.t.	Subject to
Fig.	Figure
Tab.	Table
Ch.	Chapter
Eq.	Equation
Def.	Definition
Alg.	Algorithm
Appx.	Appendix
POS	Part-of-speech
PPI	Protein–protein interaction graph
MLP	Multilayer perceptron
RNN	Recurrent neural network
CNN	Convolutional neural network
GCN	Graph convolutional neural network
IN	Interactive neural network
NMF	Non-negative matrix factorization
SVD	Singular value decomposition
DL	Deep learning
RNN	Recurrent neural network
PMI	Pointwise mutual information
GMM	Gaussian mixture model
Seq2Seq	Sequence-to-sequence
GAT	Graph attention network
GNN	Graph neural network
NGM	Neural graph model

Symbols

Table 2: Notations used in this thesis.

Notation	Description
$G(V, E)$	Graph G , nodes V , and edges E
ℓ	Random walk length
γ	Number of random walk for each node
ζ	Context size
$\phi_i \in \mathbb{R}^d$	Node embedding of node i
m	Negative context size
$\phi'_i \in \mathbb{R}^d$	Context embedding of node i
$\mathcal{N}(\psi_k, \Sigma_k)$	Community embedding of community k
$\psi_k \in \mathbb{R}^d$	Community k 's mean vector
$\Sigma_k \in \mathbb{R}^{d \times d}$	Community k 's covariance matrix
$\pi_{ik} \in [0, 1]$	Membership of node i to community k
ρ	Dirichlet concentration prior
$P_n(\cdot)$	Negative sampling probability
ν	Wishart distribution's degree of freedom
K	Number of communities to detect
N_i	Set of direct neighborhood of node v_i
α	Trade-off parameter for context embedding
C_i	Set of context node of node v_i
β	Trade-off parameter for community embedding
a	graph's average degree
$\Phi \in \mathbb{R}^{ V \times d}$	Node embedding matrix
$\Phi' \in \mathbb{R}^{ V \times d}$	Context embedding matrix
$\mathbf{A} \in \mathbb{N}^{ V \times V }$	Adjacency matrix
$\mathbf{J} \in \mathbb{N}^{ V \times V }$	Second-order proximity matrix
$\mathbf{L} \in \mathbb{R}^{ V \times V }$	Laplacian matrix
$\mathbf{D} \in \mathbb{N}^{ V \times V }$	Degree diagonal matrix
$\mathbf{P} \in \mathbb{R}^{ V \times V }$	Transition probability matrix
$\mathbf{S} \in \mathbb{R}^{ V \times V }$	Similarity matrix
$\mathbf{H} \in \mathbb{R}^{ V \times K}$	Membership matrix
$\mathbf{B} \in \mathbb{R}^{ V \times V }$	Modularity matrix
$\mathbf{W} \in \mathbb{R}^{ V \times V }$	Weight matrix

List of Figures

1.1	Example of equivalent representation for a graph $G = (V, E)$, formed by a vertex set $V = \{1, 2, 3, 4, 5, 6\}$ and an edge set $E = \{(1,2), (1,3), (2,3), (3,5), (5,6)\}$	15
1.1a	Diagram of a graph G	15
1.1b	Another diagram of graph G	15
1.2	Example of a path $P = (\{1, 2, 3, 5\}, \{e_{1,2}, e_{2,3}, e_{3,5}\})$ on a graph G . Note that only the bold lines represent the edges that form path P	19
1.3	2D toy example of graph embedding at different granularities	21
1.3a	Input graph	21
1.3b	Node embedding	21
1.3c	Edge embedding	21
1.3d	Sub-graph embedding	21
1.3e	Whole graph embedding	21
1.4	Simple graph with three communities enclosed in dashed circles	22
1.5	2D nodes and communities embedding	23
1.5a	Karate club graph	23
1.5b	Visualization based on our model [25]	23
1.5c	Visualization based on M-NMF model [144]	23
3.1	Embedding nodes and communities in a 2D space.	36
3.1a	Karate club graph	36
3.1b	Visualization based on our model	36
3.2	A closed loop for learning community embedding.	37
3.3	Degree distribution for all the graphs used for evaluation. Note that most of them appear to follow the power-law distribution.	46
3.3a	BlogCatalog	46
3.3b	Wikipedia	46
3.3c	DBLP	46
3.3d	Rochester	46
3.3e	Mich	46
3.3f	Amherst	46
3.4	Graph visualization on the Karate Club graph. Note that the community embedding is obtained using GMM with $K = 2$ while Fig. 3.4f is obtained using the spectral clustering's community assignment to fit two bivariate Gaussian distributions.	48
3.4a	DeepWalk	48
3.4b	LINE	48

3.4c	Node2Vec	48
3.4d	GraRep	48
3.4e	MNMF	48
3.4f	SF	48
3.5	Graph visualization on the DBLP graph (better viewed in color). Different node colors indicate different communities; red is <i>NLP</i> , blue is <i>Computer Vision</i> , green is <i>Data Mining</i> , yellow is <i>Database</i> and black is <i>Networking</i>	49
3.5a	DeepWalk / SF	49
3.5b	LINE	49
3.5c	Node2Vec	49
3.5d	GraRep	49
3.5e	M-NMF	49
3.5f	ComE	49
3.6	Community detection results. The smaller Conductance (denoted as Cond.) is, the better. The bigger NMI is, the better.	50
3.6a	BlogCatalog	50
3.6b	DBLP	50
3.6c	Wikipedia	50
3.6d	Rochester	50
3.6e	Mich	50
3.6f	Amherst	50
3.7	Impact of the parameter d on the node classification task.	54
3.7a	BlogCatalog	54
3.7b	Wikipedia	54
3.7c	DBLP	54
3.7d	Rochester	54
3.7e	Mich	54
3.7f	Amherst	54
3.8	Impact of the parameter d on the clustering task.	54
3.8a	BlogCatalog	54
3.8b	Wikipedia	54
3.8c	DBLP	54
3.8d	Rochester	54
3.8e	Mich	54
3.8f	Amherst	54
3.9	Model's convergence and efficiency. Note that the results are done for $d = 2$.	55
3.9a	Convergency	55
3.9b	Efficiency	55
3.10	Impact of the parameters α and β .	56
3.10a	BlogCatalog	56
3.10b	Wikipedia	56
3.10c	DBLP	56
3.10d	Rochester	56
3.10e	Mich	56

3.10f	Amherst	56
4.1	Comparison between the ComE and ComE+ nodes and communities embedding in a 2D space.	59
4.1a	Karate club graph	59
4.1b	ComE	59
4.1c	ComE+	59
4.2	Graphical model for ComE+. The solid arrows indicate a generative process. The dotted arrows indicate a neural network modeling for node embedding.	60
4.3	Generative process of ComE+.	61
4.4	Graph visualization on Karate Club.	67
4.4a	ComE	67
4.4b	ComE+	67
4.4c	PRUNE	67
4.5	Graph visualization on the DBLP graph (better viewed in color). Different node colors indicate different communities; red is <i>NLP</i> , blue is <i>Computer Vision</i> , green is <i>Data Mining</i> , yellow is <i>Database</i> , and black is <i>Networking</i>	68
4.5a	ComE	68
4.5b	ComE+	68
4.5c	PRUNE	68
4.6	ComE+ community detection results. The smaller Conductance is, the better. The bigger NMI is, the better.	69
4.6a	BlogCatalog	69
4.6b	Wikipedia	69
4.6c	DBLP	69
4.6d	Rochester	69
4.6e	Mich	69
4.6f	Amherst	69
4.7	Impact of the parameters α and β for ComE+ and ComE.	73
4.7a	BlogCatalog	73
4.7b	Wikipedia	73
4.7c	DBLP	73
4.7d	Rochester	73
4.7e	Mich	73
4.7f	Amherst	73
4.8	Impact of k parameter in terms of Micro-F1 and Macro-F1.	74
4.8a	BlogCatalog	74
4.8b	DBLP	74
4.8c	Wikipedia	74
4.8d	Rochester	74
4.8e	Mich	74
4.8f	Amherst	74
4.9	Impact of k parameter in terms of Conductance and NMI.	74
4.9a	BlogCatalog	74

4.9b	DBLP	74
4.9c	Wikipedia	74
4.9d	Rochester	74
4.9e	Mich	74
4.9f	Amherst	74
4.10	Impact of the node embedding's dimension in terms of Micro-F1 and Macro-F1.	75
4.10a	BlogCatalog	75
4.10b	DBLP	75
4.10c	Wikipedia	75
4.10d	Rochester	75
4.10e	Mich	75
4.10f	Amherst	75
4.11	Impact of the node embedding's dimension in terms of Conductance and NMI.	75
4.11a	BlogCatalog	75
4.11b	DBLP	75
4.11c	Wikipedia	75
4.11d	Rochester	75
4.11e	Mich	75
4.11f	Amherst	75
4.12a	Model convergence.	78
4.12b	Model efficiency. Note that the average time per iteration is reported.	78
5.1	A motivational example Fig. 5.1b showing the implicit graph structure learned w.r.t. the target station v_i . Note that among all the surrounding stations, v_j is the one that has the highest impact on v_i because it is the closest one and is located on the same freeway lane.	79
5.1a	Some traffic stations present in the bay area.	79
5.1b	Interaction graph learned	79
5.2	Overall model architecture.	81
5.3	A training example for each synthetic dataset. Note that the neighbors are represented by a thinner line w.r.t to the node and label time series. Moreover, noise neighbors have the thinnest line.	85
5.3a	Dynamic Noise dataset with 1 irrelevant neighbor and 3 relevant	85
5.3b	Time-shifted dataset	85
5.3c	Noise Time-shifted dataset with 2 irrelevant neighbors	85
5.4	The attention results of the DIAN model on the toy tasks. Note that y-axes report the target node input at different time-step, while the x-axes report the neighbors' inputs.	89
5.4a	Dynamic Noise. Note that in the reported example are only present two relevant neighbors and a random neighbor.	89
5.4b	Time-shifted attention output	89
5.4c	Noise Time-shifted	89
5.5	The attention results of the DIAN model on the real-world datasets.	90

5.5a	EnerNOC	90
5.5b	PeMS	90
5.6a	Impact of the number of attention head on toy datasets	93
5.6b	Impact of hidden dimension on real-world datasets	93
5.6c	Impact of the neighbors numbers on toy datasets	93
6.1	Given a graph $G(V, E)$, it is possible to extract some fixed length paths, for example [1,4,6,9]. The obtained paths are then used to train a NGM in an unsupervised way to obtain a meaningful graph embedding. Note that in Fig. 6.1b, we used a square shape to represent the nodes' vectorial representation rather than the node itself. .	95
6.1a	Graph $G = (V, E)$	95
6.1b	Neural Graph Model	95
6.2	Given a graph $G(V, E)$ in Fig. 6.2a and a node embedding in Fig. 6.2b: at each step, the embedding model should accost the nodes with the most different embedding representation.	96
6.2a	Graph $G = (V, E)$	96
6.2b	Node Embedding	96

List of Tables

1	Acronyms used in this thesis.	4
2	Notations used in this thesis.	5
2.1	Comparison with related work on graph embedding. Here, “●” indicates an property as explicitly defined, and “o” indicates an property as implicitly derived. Complexity is defined over only graph-related factors.	34
2.2	Comparison with related work on interactive network. Here, “●” indicates an property as explicitly defined, and “o” indicates an property as implicitly derived.	34
3.1	Datasets used for evaluation.	45
3.2	DBLP dataset labels.	45
3.3	Node classification results.	52
3.4	ComE p-values w.r.t. the baselines using 80% of the lables.	53
4.1	Node classification results.	71
4.2	ComE+ p-values w.r.t. the baselines using 80% of the lables.	72
5.1	Datasets summary. Note that for all the datasets, all the examples related to a given node and associated neighbors are present in the training, evaluation, or test set. Thus, we are in an inductive learning setting.	86
5.2	Parameters setting used during the experiments.	87
5.3	Results obtained on the toy tasks. Note that all the reported results are the average of four executions.	88
5.4	Results obtained on the real world tasks. Note that all the reported results are the average of four executions.	89

Abstract

In the last few years, graphs have become an instinctive representative tool to better study complex structures. For example, in chemistry, it is common to represent and study the interaction between different proteins as a graph, reducing the experimental costs. Similarly, with the consolidation of Web 2.0 and the rise of Web 3.0, social networks have become one of the most popular sources of information, wherein users not only utilize but also provide content. On such platforms, the users’ information is explicitly provided and most of the information is implicitly contained in the links between different users. In robotics, it is possible to achieve more a “human-like” movement of an artificial skeleton if we represent the interaction of each major joint rather than adopting an independent approach.

In addition to its representation capabilities, effectively extracting valuable information from graph data remains an open problem. While, in the last few years, machine learning, particularly deep learning, has achieved outstanding results in many complex applications such as image processing or natural language processing, researchers have not yet found a legitimate technique to leverage the structural information contained in the graph data type. Moreover, it is not yet possible to fully unleash the learning capabilities of deep models. Thus, in this thesis, we will introduce the concept of graph embedding, a robust set of techniques able to automatically extract meaningful information from network structures. First, we will provide an understanding of the problem setting by formally defining each component. Subsequently, we provide a review of all the major studies related to this topic. Then, we study an important yet largely under-explored setting of graph embedding, i.e., embedding communities rather than individual nodes. We identify that community embedding is not only useful for community-level applications such as graph visualization but also provides an exciting opportunity to improve community detection and node classification. To learn such embedding, our insight hinges on a closed loop among community embedding, community detection, and node embedding. On the one hand, node embedding can help improve community detection, which outputs good communities for fitting better community embedding. On the other hand, community embedding can be used to optimize the node embedding by introducing a community-aware high-order proximity. Based on this insights, we propose a novel Community Embedding (ComE) model that jointly solves the three tasks together.

In practice, the number of communities can be unknown beforehand; therefore, we further proposed ComE+, a new framework that handles both uncertainty of unknown ground truth community assignments and an unknown number of communities. We further developed an efficient inference algorithm for ComE+ by

maintaining the complexity as low as $O(|V| + |E|)$. We evaluated the proposed algorithms on multiple real-world data sets and showed that they improve graph visualization and outperform the state-of-the-art baselines in various application tasks, e.g., community detection and node classification.

Finally, given the proliferation of dynamic interactive structure in many socio-economic problems, we argue that a new set of embedding techniques are required. Recently, spatio-temporal graphs and interaction networks have proven to be useful in abstracting the underlying interactive structure that defines the influence between the various aspects of a problem, but most of the previous studies have developed architecture that is able to only preserve manually crafted interrelation. However, we argue that it is necessary to automatically learn from such interaction structures if we want such models to fit in real-world problems. We also require that the discovered latent structure should be easily interpretable to facilitate the understanding of the modern world. Thus, we propose an efficient neural mixture model able to project a dynamic interaction in a fixed size space, preserving the influence that each node undergoes because of its neighbors. The proposed architecture is able to outperform strong baselines in various data sets ranging from the prediction of the household’s energy load to traffic congestion, and many others.

Keywords: Graph embedding, Community embedding, Interaction networks.

Chapter 1

Introduction

In nature, many events are the consequence of the cooperation and synergy among different components of a complex structure. Similarly, the modern society is characterized by interactions among economic, political, and other forces, which are not completely understood. Being able to define such underlying elements is an essential step toward the understanding of the modern world. For example, with the rise of Web 2.0 and the development of platforms, such as Facebook, Twitter, and Reddit, users, for the first time, have become both content creators and content consumers. Still, a considerable amount of information is contained in the connections among the different users/customers. By analyzing such links, it is feasible to classify users, detect communities, recommend friends, and predict new links between two users; thus, it is possible to extract insights from the modern society. How can you adequately represent such kind of knowledge? Also, which are the most efficient methods to process such non-Euclidean structures? These questions and certain others related to the graph mining field are common queries that researchers are trying to resolve.

Given their expressive power and their ability to preserve interconnected conformation, *graphs* have emerged as an intuitive representation tool for mathematically abstract complex systems. According to [15], it is possible to define a graph as follows:

Definition 1.0.1. A **Graph** G comprises an ordered pair of sets (V, E) , where $V = \{v_1, \dots, v_n\}$ is a non-empty set of *vertex* (or nodes, or points) and $E = \{e_{i,j} = (v_i, v_j)\}_{i,j=1}^n \subseteq V^2$ is the set of *edges*.

Despite its formal definition, it is possible to picturize a graph by drawing circles to represent the vertices and lines to describe the edges that connect the two vertices. As shown in Fig. 1.1, the way these circles and lines are drawn is considered irrelevant: all that matters is the information of which pair of vertices forms an edge and which does not.

Traditionally, to leverage such structured knowledge for analytical purposes, researchers proposed multiple recursive or iterative algorithms. Deep First Search [33] and Page Rank [107] are just two of the most well-known examples. However, more complicated analytics tasks require sophisticated algorithms that do not scale well to the size of modern problem settings. **Graph embedding**, which proposes to project the topological graph structure into a Euclidean space and preserves some

Figure 1.1: Example of equivalent representation for a graph $G = (V, E)$, formed by a vertex set $V = \{1, 2, 3, 4, 5, 6\}$ and an edge set $E = \{(1, 2), (1, 3), (2, 3), (3, 5), (5, 6)\}$

well-defined properties, has emerged as a valuable alternative. Over the years, researchers have proposed various metrics to retain the nodes' connectivity in the embedding space.

While most of the proposed formulation focused on the presence of direct links between different vertices, many networks are better defined by the mesoscopic structures created by a set of densely connected nodes. Such structures are commonly associated with the concept of communities. Thus, in this thesis, we explore the concept of **community embedding** by analyzing the associated challenges and opportunities. Communities are one of the most prominent and complex features of a network because they are characterized by blurred overlapping boundaries and multiple membership problem. Still, community embedding remains a largely under-explored setting even if it is useful for various community-level applications such as community visualization, to help generate insights from big graphs or community recommendation, and to search for similar communities. Moreover, community embedding can be applied to different types of graphs. For example, it is possible to cluster World Wide Web pages based on their topics or modeling the word distribution of certain documents to provide a summary. Similarly, it is possible to cluster IP address and generate compact routing tables. We argue that community embedding provides a *higher-order community-aware proximity* metric that can enrich the graph embedding through a closed-loop process. Thus, it can also improve multiple node-level tasks, i.e., we can improve node classification, node clustering, or node ranking performances.

We provide a review of the most renowned graph embedding studies in Ch. 2; in Ch. 3 we have formally defined the community embedding problem and proposed ComE, which is the first model that provides a probabilistic interpretation of community embedding. As ComE has certain practical limitations, in Ch. 4, we enhance the original algorithm by adopting a Bayesian approach to better handle the uncertainty related to the community structure that is present in a graph. While ComE and ComE+ only leverage the topological information, in many real-world scenarios, communities are better defined by the nodes' attribute. Thus, in Ch. 5, we investigate the possibility of learning an implicit graph structure based on the attribute of each node. As a result, we empirically demonstrate that the learned network is useful to solve multiple regression tasks. Finally, in Ch. 6, we provide certain possible future developments of this research.

1.1 Background

1.1.1 Graphs, Nodes, and Edge Properties

As a consequence of being intuitive for illustration, most of the graph theory definitions are derived from their graph's representation. Thus, it is possible to differentiate between *infinite* or *finite* graphs, which are formed by a finite or an infinite number of nodes and edges, respectively. Please note that in this thesis, we will focus only on finite graphs. Furthermore, graphs that are composed of one single vertex are called *trivial* graphs, while all other graphs are called *non-trivial*. Graphs can be differentiated between two main categories according to their edge set: if E is formed by ordered pairs, i.e., $e_{i,j} \neq e_{j,i}$, we have a **directed** graph; otherwise, if E is formed by unordered pairs, we have an **undirected** graph. For example, Twitter¹ is a directed graph, while Facebook² is an undirected graph. Similarly, if a weight $w_{i,j}$ is associated to each edge $e_{i,j}$, we have a **weighted** graph; however, if the edges are not weighted, we have an **unweighted** graphs. Note that we will refer to weighted graphs as $G = (V, E, W)$, where W is the set containing the weights associated to each edge. It is also possible that a graph is composed of nodes and edges of a different type. In this case, a node type mapping function $f_v : V \rightarrow T^v$ and an edge type mapping function $f_e : E \rightarrow T^e$ define different graphs types. We can distinguish between three types of graphs as follows:

1. When both nodes and edges belong to a single category, we have a homogenous graph. Formally, it is defined as follows:

Definition 1.1.1. Homogenous Graph: a graph $G = (V, E)$ where all nodes and edges belong to a single type ($|T^v| = |T^e| = 1$)

On the one hand, unweighted and undirected homogenous graphs are considered as the most basic graph structures since they preserve only the interactive structure. On the other hand, weights and directions provide additional information that has to be carefully exploited during the analytical process.

2. If different types of nodes or edges form a graph structure, we have a heterogeneous graph. Formally, it is defined as follows:

Definition 1.1.2. Heterogeneous Graph: a graph $G = (V, E)$, where nodes and edges can have a different type ($|T^v| \geq 1$ and/or $|T^e| \geq 1$)

According to [21], it is possible to further classify heterogeneous graphs as follows:

- (a) **Multimedia network:** as Yelp³, these graphs are formed by node or edge, representing the different types of multimedia content, e.g., image or text.

---

¹<https://twitter.com/>

²<https://www.facebook.com/>

³<https://www.yelp.com/>

- (b) **Knowledge graphs:** Knowledge graphs are networks formed by entities (nodes) that are connected by relations (edges). Usually, each entry of the graph represents some factual knowledge, and it is represented as a triple (*head entity, relation, tail entity*), or (h, r, t) . Given their ability to represent knowledge in a concise form, researchers have focused on developing an efficient inference algorithm to solve multiple reasoning tasks.
- (c) **Community-based question answering** is a crowdsourcing platform where users are allowed to post some questions or give some answers. The primary aim is to provide a ranking between the possible answers given an input query; thus, it has been proven useful to represent such setting using graphs that connect questions, answers, and users.

Moreover, these graphs are usually associated with additional information. For instance, some graphs provide **labels** for their nodes or edges to better distinguish their functional roles. Other graphs provide **attribute** values rather than labels. Note that we refer to labels as categorical values, while attributes can represent any features such as the pixels of an image or the age of a user. Attributes usually contain unstructured information with a finer granularity; however, how to leverage this different type of object and relation is still an answered question. In addition to node and edge-level information, because of the rise of e-commerce and social marketing, considerable efforts have focused on studying the **information propagation** problem. Thus, the associated concept of *cascade graph* has been proposed to describe how the information flows in a given network.

3. Using the well-developed spectral graph theory [31], it is possible to create abstract graphs from non-relational datasets, assuming that the input data lie in a high-dimensional manifold.

Definition 1.1.3. Abstract graphs from a non-relational dataset are only defined by the similarity matrix $G = \mathbf{S} \in \mathbb{R}^{|V| \times |V|}$, computed as the distance between each pair datapoint of an input dataset $X \in \mathbb{R}^{|V| \times d'}$

Works like [64] interpret the constructed similarity matrix as the adjacent matrix of a graph, while other works like Isomap [8] utilize the geodesic distance to alleviate the problem of curvature in manifolds that are not projectable in a Euclidean space.

In addition to different graph types, there are also different node and edge properties. On the one hand, given any graph, we define two vertices that form an edge as its *ends*, while an edge *joins* its ends. A vertex v_i is said to be *incident* with an edge $e_{i,j}$ if $v_i \in e_{i,j}$, while two vertices that are incident with a common edge are said to be *neighbors*. According to Tab. 2, we denote all the neighbors of a vertex v_i as N_i and degree of v_i as its number of neighbors ($|N_i|$). However, if we have a directed graph, each vertex will have an in-degree and an out-degree, defined as the number of in-coming edges and out-coming edges, respectively. It is also worth

defining the graph's average degree as $a = \frac{1}{|N|} \sum_{v_i \in V} N_i$. Note that we refer to two incident edges on a common vertex as *adjacent*. While we call *loop* an edge with same ends, we refer to *link* as an edge with distinct ends.

1.1.2 Matrix Representation of a Graph

Although drawing a graph could give some intuition on its property, a more analytical representation is given by its **adjacency matrix** $\mathbf{A} \in \mathbb{N}^{|V| \times |V|}$:

$$a_{i,j} = \begin{cases} 1, & \text{if } e_{i,j} \in E \\ 0, & \text{otherwise} \end{cases}$$

For example, the adjacency matrix for the graph G in Fig. 1.1 is as follows:

$$\begin{matrix} & \begin{matrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{matrix} \\ \begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{matrix} & \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{matrix}$$

In computer science, the adjacency matrix has become a commonly used representation for graphs because networks are known to be sparse and thus storable in a sparse matrix. Moreover, if we have a weighted graph, it is possible to represent it through its *weight matrix* $\mathbf{W} \in \mathbb{R}^{|V| \times |V|}$:

$$w_{i,j} = \text{weight}(e_{i,j})$$

Furthermore, closely related to the adjacency matrix is the *transition matrix* $\mathbf{P} \in \mathbb{R}^{|V| \times |V|}$, defined as follows:

$$p_{ij} = \frac{a_{i,j}}{\sum_{1 \leq j \leq |V|} a_{i,j}}$$

Rather than describing the structure of the graph, the transition matrix represents the probability of transitioning to node v_j from node v_i .

1.1.3 Paths and Random Walks

A *path* is a non-empty graph $P = (V, E)$ formed by distinct nodes connected by subsequent edges. Formally, a path is defined as $V = \{v_1, v_2, \dots, v_k\}$ $E = \{e_{1,2}, e_{2,3}, \dots, e_{k-1,k}\}$. We say that vertices v_1 and v_k are *linked* by P, while the number of edges determine the path's *length*. Figure 1.2 shows an example of a path $P = (\{1, 2, 3, 5\}, \{e_{1,2}, e_{2,3}, e_{3,5}\})$ on the graph G, originally depicted in Fig. 1.1.

A particular type of path, which is used in many mathematical and computer science models such as the Markov chain model, is the **random walk**. Given a

Figure 1.2: Example of a path $P = (\{1, 2, 3, 5\}, \{e_{1,2}, e_{2,3}, e_{3,5}\})$ on a graph G . Note that only the bold lines represent the edges that form path P .

graph G and a starting vertex v_1 , we randomly select a neighbor of v_1 , and we move on it. Then, for a given number of times, we iterate the same procedure from the randomly selected point. The (random) path generated is commonly known as *random walk* and is closely related to the steady-state probability distribution of a Markov chain. A fundamental property of a random walk is that in the limit, the long-term average probability of being at a particular vertex is independent of the starting vertex or an initial probability distribution over vertices, provided that the underlying graph is strongly connected.

1.2 Graph Embedding

In the past few decades, researchers have had to face escalations that social networks, biology networks, citations networks, and other graph structures have led to the advancement of the Big Data era. The continuous requirement to identify patterns among large amounts of data has driven the development of a distributed framework, e.g., Pregel [96], GraphX [51], GraphLab [94], and Giraph [39], or dynamic graph processing tools such as GraphIn [122] and BLADYG [4]. All these frameworks focus on providing an implementation, efficient in space and time, of the classical graph traversing algorithms (e.g., breadth-first or deep-first search), of the computation of basic graph statistics (e.g., triangles counting, average degree, diameter, etc.), or of the traditional pattern mining algorithms such as collaborative filtering. However, there are two main challenges that these frameworks face. First, there is no consensus on the “building blocks” that are required by a graph analytics framework. Second, such frameworks suffer from high training and setup costs, while they are not easily adaptable to the requirements of the, always more popular, deep learning (DL) algorithms.

In addition to the abovementioned frameworks, graph embedding provides another efficient, yet effective way to facilitate the execution of analytical tasks. For example, graph embedding has been proven useful to solve the following tasks:

- Node classification in which we aim to classify each node based on the label of its neighbors;
- Link prediction, which aims to detect the missing edges between two nodes;
- Node clustering, which is used to group single node in communities; and

- Graph classification, which aims at assigning a label according to the entire graph structure.

According to the general embedding definition, graph embedding seeks a vectorial representation that is able to preserve as much as possible of the original information contained in the input graph. Then, the vectorial representation obtained is used to efficiently execute mining algorithms to solve complex analytical tasks.

Previous graph embedding studies such as [8, 10, 11, 64] focused on finding node embedding of non-relational data located in a high-dimensional manifold (Def. 1.1.3). While the graph adjacency matrix is artificially created as the pairwise similarity between the feature vector of each datapoint, the final representation is obtained as matrix decomposition where "similar" nodes in the adjacency matrix will be scattered closer in the lower dimensional space. Nevertheless, such embedding techniques were known to be expensive to compute. Only in 2014, the pioneering work performed by Perozzi et al. proposed a simple yet elegant solution to obtain a meaningful node embedding using an efficient sampling procedure. Subsequently, the state-of-the-art DL architecture (usually in the form of auto-encoders) and graph kernels were demonstrated to be capable of obtaining accurate edge [28] and graph representations [124].

Cai et al. formally defines graph embedding as follows:

Definition 1.2.1. Graph embedding: Given an input graph $G = (V, E)$ and a predefined embedding dimension d ($d \ll |V|$), graph embedding aims to identify a representation of G in the d -dimensional space such that the graph properties are preserved as much as possible. The obtained output could be either one d -dimensional vector (for a whole graph) or a set of d -dimensional vectors in which each vector represents the embedding of a part of the graph (e.g., node, edge, or substructure).

As per definition, graph embedding provides many application-dependent challenges to overcome. On the one hand, there are challenges related to the different types of input graphs. For example, in certain applications, an homogenous graph is provided; thus, it is possible to preserve only its structural information [97, 110, 162] according to its weight [23, 24, 131, 141]. Instead, when a heterogeneous graph is provided, different features associated with each node or edge have been considered. For instance, [41, 95, 163] exploit the different types of edges present in a community-based question answering graph to obtain a more accurate embedding. However, [28, 46, 159] use the different node and edge types present in the multimedia network to improve the obtained low-dimensional representation. Furthermore, given the vast literature, it is essential to consider all the different distance metric associated with each graph's property because the performance may depend on the final application. Note that scalability issues are related to the type of input graph: e.g., social and citation networks are known to be large graphs; thus, they require a scalable algorithm. Instead, protein-protein interaction (PPI) graphs usually have smaller size; therefore, they can be processed by more complex algorithms.

On the other hand, the output embedding format is task dependent. To summarize, we distinguish four types of possible embedding outputs:

Figure 1.3: 2D toy example of graph embedding at different granularities

1. **Node embedding:** As shown in Fig. 1.3b, node embedding aims to find a meaningful vectorial representation for each node. Thus, it is the most common form of embedding that has been created. Forcing “close” nodes to have similar representation in the embedding space, it has proven useful in tasks such as node classification, node clustering, and node ranking. There is no consensus on how to define two close nodes; however, most methods rely on the similarity between two adjacent nodes (*first-order proximity*) and on the similarity of the neighborhood structure between the two vertices (*second-order proximity*) [53, 110, 129]. Other models have attempted to leverage the information between the neighbors at k -step ($k = 1, 2, 3, 4, \dots$) distance, thus introducing the concept of *higher-order proximity* [23].
2. **Edge Embedding:** It focuses on identifying a low-dimensional embedding for each edge (Fig. 1.3c); therefore, it is more suitable for tasks such as link prediction [53], network reconstruction [28], and knowledge graph entity/relation prediction [16, 18, 151]. Edge embedding preserves the asymmetry of directed edges, which is necessary for the development of special techniques. For instance, [61] proposed using multi-dimensional Gaussian distributions with a KL-divergence loss to represent the asymmetry in entities and relations.
3. **Sub-graph Embedding:** Fig. 1.3d shows an example of sub-graph embedding, which is supposed to represent substructure such as small communities or node-edge structure in a vectorial form. Sub-graph embedding has been studied in multiple studies. For example, to support the semantic proximity search task on a heterogeneous graph, [93] proposed to leverage the well-known LSTM model to learn the sub-structure between two possibly distant nodes. Similarly, [91] demonstrated that explicitly preserving the mined sub-graph

Figure 1.4: Simple graph with three communities enclosed in dashed circles

structure through stacked auto-encoders improves the performance of proximity embedding. Compared to sub-graph embedding, community embedding has received limited attention in the research community. In fact, the concept of community embedding has been explored only in [144].

4. **Whole graph embedding:** Certain applications, such as the study of protein and molecular interactions, require the computation of similar patterns between different graphs; thus, in the last decade, various methods ([104, 123]) have been proposed to represent an entire graph as a vector (e.g. Fig. 1.3e).

1.3 Objectives and Major Contributions

In graph theory, it is possible to define a community as a set of densely connected nodes or formally as follows:

Definition 1.3.1. Community: a non-empty set of nodes z_i where $\cup_{i=1}^K z_i \subseteq V$ with dense intra-group connections [143].

Consider the simple graph composed of 12 nodes organized in three communities and an outlier (node N) shown in Fig. 1.4. Detecting the communities is an important task because it allows the identification of communities' leaders, i.e., the node with the highest degree inside a cluster. Communities' leaders provide good intra-community information spread; thus, they are a key factor for multiple tasks. Moreover, nodes at the border of a community are essential too as they work as mediators and spread the information contained in the community across the entire network (nodes L, M, E, and B).

Providing a compact parametrization of a community is a challenging task because there is no consensus on its quantitative definition. On the one hand, many different approaches have been proposed for community detection ranging from distance-based methods to model-based methods [62]. Yes, there is no consensus on the most optimal approach [109, 138, 158]. Distance-based models (e.g., spectral clustering, non-negative matrix factorization, and K-means) are known to produce a hard partition of the dataset. To the best of our knowledge, [144] is the first

Figure 1.5: 2D nodes and communities embedding

study to perform both community embedding and node embedding; however, the proposed matrix factorization formulation lacks a human-interpretable community representation and is subject to the multi-membership problem. Instead, model-based methods (e.g., GMM and IGM [114]) produce label’s assignment based on the conditional probability of a mixture of distributions. In such models, each node affects, in big or small proportion, the representation of each community, while a community provides an explicit representation of how the nodes spread in the latent space. According to Def. 1.3.1, community embedding is expected to characterize how its member nodes distribute in the low-dimensional space. As a result, we cannot merely define a community embedding as a vector; rather, we need to define it as a *distribution* over the node embedding space. Thus, in Ch. 3, we have proposed a novel probabilistic formulation of the community embedding problem. Inspired by the Gaussian mixture model [13], we see each community as a multivariate Gaussian distribution. Such a formulation enables us to define a self-enforcing closed loop among node embedding, community detection, and community embedding. First, node embedding has been proven to improve community detection performances because it can effectively preserve the network’s structure [23, 76, 110, 131]. Then, the obtained community assignment can be used to fit a meaningful community embedding. Finally, community embedding represents densely connected nodes; therefore, it can provide a higher order proximity value for node embedding while providing a human-readable community representation. For example, in Fig. 1.5 we report the embedding in a 2D space of the well-known Karate Club graph⁴. This network is known to have two communities: one led by the club instructor (node 1) and the other managed by the club administrator (node 34) [157]. As shown in Fig. 1.5b, our probabilistic formulation enables the creation of a community-aware node embedding in which each community is explicitly visualized as two overlapping ellipses, each of which is defined by a 2×2 covariance matrix and a 2D mean vector; however, the matrix factorization formulation proposed in Fig. 1.5c remains a challenge to interpret.

A recent study [109] emphasizes the little agreement between the nodes’ meta data used as community ground true and the actual community structure present

⁴<https://networkdata.ics.uci.edu/data.php?id=105>

in the graph. Thus, in many real-world applications, the number of communities present in a graph is unknown in advance. Therefore, in Ch. 4 we propose ComE+, a method that combines Bayesian optimization with Maximum A Posteriori (MAP) estimation to automatically determine the number of communities from the data distribution. Empirically, we demonstrate that such an approach can better handle the uncertainty related to the community structure while maintaining an efficient algorithm of complexity $O(|V| + |E|)$.

While in ComE and ComE+ we only consider homogeneous graphs, many socio-economic problems are represented by heterogeneous graphs in which a node or an edge is associated with certain attributes. In Ch. 5, we argue that it is possible to identify attribute-based structures that better explain the causal relationship between the different factors of a real-world problem. For example, the traffic prediction of an urban area is usually modeled as a directed graph in which each vertex represents a traffic station, and the vertices' attributes represent the car's stream for a given time interval. Thus, it is possible to leverage the attribute information of each vertex to identify a weighted network structure that better describes the flow of cars. To accomplish this, we propose the DIAN model, a graph neural network that leverages a novel symmetric attention mechanism to learn an implicit graph based on the evolution of the nodes' attributes. The proposed architecture has been proven able to effectively solve multiple structured regression tasks.

Finally, in Ch. 6, we outlined certain possible future studies that are necessary to solve some of the problems present in the current state-of-the-art graph embedding models. For example, most of the graph-embedding algorithms rely on random walks as the fundamental method to represent the graph structure. However, it is not trivial to condition the sampling process according to the current embedding performances; thus, most models are subject to sub-optimal performances.

Chapter 2

Literature Review

2.1 Background

In this chapter, we are going to review important studies related to graph embedding. In particular, we will focus on node embedding models because they are the most relevant ones when considering the algorithms proposed in this thesis. The first node embedding technique dates back to the early 2000s when graph embedding was perceived as a branch of the dimensionality reduction field. The earliest approaches [8, 10, 11, 64] generated an artificial similarity matrix $\mathbf{S}$ from non-relational datasets located in a high-dimensional manifold. Each node, or data point, was then projected in a vectorial space based on the similarity w.r.t. its neighbors. Other solutions differentiate on the properties they try to preserve in the embedding space or on the construction of matrix $\mathbf{S}$. For example, based on the insight that two nodes connected by a high weight edge have to be similar, the initial approach exploits the local pairwise similarity between a node and its neighborhood as the most basic structural information. Formally, we define such proximity as follows:

Definition 2.1.1. First-order proximity: Given a graph $G = (V, E)$, the edge's weight $w_{i,j}$ represents the first-order proximity between the pair of nodes (v_i, v_j) .

Minimizing the weighted L-2 distance between two adjacent nodes is a straightforward approach to obtain node embedding:

$$\arg \min_{\Phi} \mathcal{L}(\phi_i, \phi_j | G) = \arg \min_{\Phi} \sum_{e_{i,j} \in E} w_{i,j} \|\phi_i - \phi_j\|^2 \quad (2.1)$$

where ϕ_i represents the node embedding of vertex v_i and Φ is the embedding of all the nodes. According to Eq. 2.1, a nodes' pair has a similar vector representation proportional to their edge weight $w_{i,j}$.

As a natural extension of the first-order proximity, it is possible to evaluate the neighborhood structural similarity between a pair of vertices. Formally, we define the second-order proximity as follows:

Definition 2.1.2. Second-order proximity: Given a pair (v_i, v_j) , we refer to the second-order proximity as the similarity between the neighborhood structure of v_i and v_j ; i.e., let $\mathbf{j}_i = [w_{i,1}, \dots, w_{i,n}]$ denote the first-order proximity between vertex v_i

and all the other vertices of V , then the second-order proximity is defined as the similarity between $\mathbf{j}_i$ and $\mathbf{j}_j$ for whichever similarity metric is used.

Based on the same notation, the higher-order proximity can be defined in a similar manner:

Definition 2.1.3. Higher-order proximity: Let $\mathbf{j}_i^{k-1}$ be the proximity between node v_i and its k -th-step neighbors; the similarity between $\mathbf{j}_i^{k-1}$ and $\mathbf{j}_j^{k-1}$ represents the k -order proximity between node v_i and v_j .

Next, we are going to introduce the primary technique used to produce meaningful node embedding. To provide a better understanding of such a popular field, we will classify each method according to the following taxonomy:

1. **Factorization-based methods:** Matrix factorization methods initially represent a graph G in a matrix format (e.g., Adjacency matrix, Laplacian matrix, etc.). Node embedding is then obtained as a factorization of the initial graph matrix. Thus, it was interpreted as a dimensionality reduction technique. According to [21], such models are differentiated in *Graph Laplacian Eigenmaps* and *Matrix Factorization*.
2. **Deep Learning methods:** Because of their demonstrated capability to automatically learn useful data representation in multiple tasks, DL algorithms have been successfully applied to the graph embedding problem. Given the efficiency and effectiveness of random walk (Ch. 1.1.3) for preserving the graph structure, multiple models utilize randomly sampled paths as fundamental representation units. Thus, we will differentiate between *Random Walk* and *Non-Random Walk* based models. Moreover, the model proposed in Ch. 5 is affine to the recently proposed *Interactive Network* (IN) architecture: a review of such neural network design is included.
3. **Knowledge graph methods:** Because of the peculiar nature of these graphs, knowledge graph embedding models have caused specific challenges and attracted considerable attention from the research community; therefore, we decided to handle them separately.

Given that communities are a key concept of this dissertation, a short review of the major *community detection* algorithms is also provided. Subsequently, *Bayesian embedding* approaches are covered because of the resemblance with the ComE+ model proposed in Ch. 4. Finally, Tab. 2.1 provides a brief view of the characteristics of the most relevant models w.r.t. ComE (Ch. 3) and ComE+ (Ch. 4).

2.2 Factorization Methods

At its early stage, the problem of graph embedding was treated as a structure-preserving dimension reduction techniques based on the assumption that the input data lie in a well-defined manifold.

2.2.1 Graph Laplacian Eigenmaps

Inspired by the first-order proximity (Def. 2.1.1) and Eq. 2.1, initial models penalize the neighbor nodes that are embedded far apart from each other. Defining the Laplacian matrix as $\mathbf{L} = \mathbf{D} - \mathbf{A}$ and assuming $\mathbf{D}$ being the diagonal matrix where $D_{i,j} = \sum_{i \neq j} W_{i,j}$, it is possible to reduce Eq. 2.1 to the following equation:

$$\begin{aligned} \arg \min \mathcal{L}(\phi_i, \phi_j | G) &= \arg \min_{\text{s.t. } \Phi^T \mathbf{D} \Phi = 1} \Phi^T \mathbf{L} \Phi \\ &= \arg \min \frac{\Phi^T \mathbf{L} \Phi}{\Phi^T \mathbf{D} \Phi} \\ &= \arg \min \frac{\Phi^T \mathbf{W} \Phi}{\Phi^T \mathbf{D} \Phi} \end{aligned} \quad (2.2)$$

where the optimal node embedding Φ is obtained as the eigenvectors associated with the top eigenvalues of the problem $\mathbf{W} \Phi = \lambda \mathbf{D} \Phi$.

While the problem formulation is standard, the proposed models differentiate in the computation of the weighting matrix $\mathbf{W}$ if they refer to a non-relational dataset or in the definition of a linear function $\phi_i = X_i^T a$, where X_i represents the feature vector of the input i , which is used to overcome the *transductive* training problem. For example, MDS [64], the first work on this line, used the Euclidean distance to compute $\mathbf{W}$, but did not consider the neighborhood of each node. Subsequently, [8, 63, 117] overcame this issue by constructing a K-nearest neighbor graph from the data feature and proposed a more appropriate metric to compute $\mathbf{W}$. Specifically, regularization terms are used to overcome the problem related to the projection in a Euclidean space of the curvature surface present in a manifold. To process heterogeneous graphs, [20, 88] proposed to construct two different weight matrix: one to encode the graph structure (labels) and one to encode the additional information (user's feedback). Finally, SDP [136] and MVU [149] argue that the weight matrix $\mathbf{W}$ has to be learned instead of being computed as the pairwise similarity between any couple of nodes to better capture the relationship between the different data points.

2.2.2 Matrix Factorization

Matrix factorization methods leverage the dimensionality reduction knowledge to identify a low-dimensional representation of a graph (initially described via a matrix such as the adjacency matrix). Among the others, *non-negative matrix factorization* (NMF), which aims to find two non-negative matrixes whose product approximate the original one, is the most popular technique. It is worth mentioning that w.r.t. other factorization algorithms such as *singular value decomposition* (SVD) [50], the non-negative constraint allows only additive interaction between the various components; thus, it is believed that NMF is able to learn a *parts-based* representation [80] of the dataset. Formally, given an input proximity matrix $\mathbf{W} \in \mathbb{R}^{|V| \times |V|}$, the intent of NMF algorithms is to compute two non-negative matrixes $\mathbf{U} \in \mathbb{R}^{|V| \times d}$

and $\mathbf{V} \in \mathbb{R}^{|V| \times d}$, with $d \ll |V|$, which minimize:

$$\min \|\mathbf{W} - \mathbf{U}\mathbf{V}^T\|^2 \quad (2.3)$$

where $\mathbf{U}$ represents the node embedding and $\mathbf{V}$ is the context embedding. Based on Eq. 2.3, NMF models generate a compressed approximation of the original data matrix, $\mathbf{W} \approx \mathbf{U}\mathbf{V}^T$, where each node is represented by a linear combination of its node embedding and context embedding. Different NMF methods propose various regularization terms for the loss function (Eq. 2.3) or various proximity matrix $\mathbf{W}$. For example, graph factorization [3] adds a Gaussian regularization term to better preserve the manifold structure and TADW [153] adds text features to the matrix $\mathbf{W}$; while [132] proposes to jointly optimize Eq. 2.3 and a SVM loss to obtain better the node classification results. An honorable mention goes to M-NMF [144], which jointly factorizes the node embedding and the modularity matrix to obtain a community-preserving embedding. Such a method is most similar to the approaches presented in Ch. 3 and 4. However, the learned community embedding is vectorial, and thus there is a lack of a human-interpretable visualization; moreover, its complexity of $O(|V|^2)$ is generally higher than our complexity of $O(|V| + |E|)$ because in practice, graphs are sparse with $|E| \ll |V|^2$.

As an alternative to NMF, SVD received considerable attention in the past years because of its excellent performance in recommendation systems. Precisely, SVD is used to represent $\mathbf{W}$ using its singular values, i.e.:

$$\mathbf{U} = [\sqrt{\sigma_1}u_1, \dots, \sqrt{\sigma_d}u_d] \quad (2.4)$$

$$\mathbf{V} = [\sqrt{\sigma_1}v_1, \dots, \sqrt{\sigma_d}v_d] \quad (2.5)$$

where $\{\sigma_1, \dots, \sigma_{|V|}\}$ are the singular values sorted in decreasing order. Such singular values are then used to generate a low-dimensional node embedding. Note that the existing solutions differentiate in the preserved proximities. For example, both GraRep [23] and HOPE [106] preserve a higher-order proximity, while [105] proposes a novel distance metric to compute $\mathbf{W}$. However, SVD is known to be expensive to compute $O(|V|^3)$.

2.3 Deep Learning Models

Since 2012 [32], DL shows impressive performances on various tasks such as image processing/pattern recognition and natural language processing. Because of its ability in leveraging huge amount of data to learn useful representation, researchers have successfully proposed DL embedding models for a vast range of application ranging from words or documents to nodes or graphs. When applied to graph embedding field, it is possible to distinguish between DL models based on random walks (Ch. 1.1.3) or directly based on the graph structure.

2.3.1 Random Walk Models

Random walks are known to preserve the network structure and have proven useful for multiple ranking tasks [1, 2, 7]. Pioneering graph embedding approaches utilized

sampled paths to capture the second-order proximity between nodes. Such paths are then processed by the DL methods to produce a useful graph description in the Euclidean space.

Inspired by the outstanding performances of the Word2Vec [102] model on NLP tasks, DeepWalk [110] first proposed to use the SkipGram model [100] on randomly sample paths to obtain a valuable graph embedding. Each path is seen as a sentence, in which the nodes are perceived as the sentence’s words. Then, the SkipGram model is employed to maximize the co-occurrence between neighbors that appear within a window of size w on a sample path and the embedding of the current node. Note that in such models, each node is exploiting two roles: as a node for itself and as context for a node present in the path window w ; therefore, an extra context embedding $\phi' \in \mathbb{R}^d$ is introduced. According to its definition, DeepWalk satisfies the second-order proximity (Def. 2.1.2) because nodes with similar neighborhood would share a similar embedding. Formally, it is possible to formulate the DeepWalk’s loss as follows:

$$\begin{aligned} \min_{\Phi} \mathcal{L}(\Phi|G) &= \min_{\Phi} -\log P(\{v_{i-w}, \dots, v_i, \dots, v_{i+w}\}|\phi_i) \\ &= \min_{\Phi} -\log \sum_{-w \leq j \leq w} P(v_{i+j}|\phi_i) \end{aligned} \quad (2.6)$$

where ϕ_i represents the node embedding of vertex v_i and Φ is the embedding of all the nodes present in the graph. The conditional probability $P(v_{i+j}|\phi_i)$ is then defined as a softmax function:

$$P(v_{i+j}|\phi_i) = \frac{\exp(\phi_{i+j}^T \phi_i)}{\sum_{k=1}^{|V|} \exp(\phi_k^T \phi_i)} \quad (2.7)$$

However, computing Eq. 2.7 is unfeasible on a graph; thus, two alternatives have been proposed:

1. **Hierarchical softmax** [103]: A balanced binary tree is constructed such that each leaf represents a graph node. The probability distribution of any graph’s node v_i is computed as the product of the sigmoid activations between the tree’s nodes along the path connecting the root of the tree to the leaf associate to v_i . Because the maximum depth of a balanced binary tree is $\log_2(|V|)$, we only need to evaluate at most $\log_2(|V|)$ nodes of the tree to obtain the final probability of a given graph’s vertex. Formally, let $n(v_i, j)$ be the j -th node along the hierarchal path from the root to v_i ; let $L(v_i)$ be the length of the path from the root to v_i ; let $ch(n)$ be the child of the tree’s node n ; and let $\langle x \rangle$ be 1 if x is true and -1 otherwise. Then, the hierarchical softmax is defined as follows:

$$P(v_{i+j}|\phi_i) = \prod_{k=1}^{L(v_{i+j})-1} \sigma(\langle n(v_{i+j}, k+1) = ch(n(v_{i+j}, k)) \rangle \phi_{n(v_{i+j}, k)}^T \phi_i) \quad (2.8)$$

2. **Negative sampling** [55]: Based on the noise contrastive estimation, negative sampling assumes that a good model should differentiate between true

neighbors and noise neighbors of a node v_i by means of logistic regression. According to this principle, it is possible to define $P(v_{i+j}|\phi_i)$ as

$$P(v_{i+j}|\phi_i) = \log \sigma(\phi_{i+j}^T \phi_i) + \sum_{t=1}^K \mathbb{E}_{v_t \sim P_n(v_i)} [\log \sigma(-\phi_t^T \phi_i)] \quad (2.9)$$

where K represents the number of negative example used for training, and v_t is a negative neighbor sampled from a negative distribution $P_n(v_i)$ (such as a uniform distribution, i.e., $P_n(v_i) = \frac{1}{|V|}, \forall v_i \in V$).

Driven by the success of DeepWalk, many other DL approaches leverage random walk to learn a graph distribution. For example, Node2Vec [53] leverages additional structural information by guiding the path sampling process at a higher computational cost ($O(|V|a^2)$ where a is the graph’s average degree). LINE [129] preserves both first- and second-order proximity maximizing the edge reconstruction probability ($\max \sum_{e_{i,j} \in E} \log \sigma(\phi_i^T \phi_j)$) at the price of a higher complexity $O(a|E|)$. Metapath2Vec [37] and Struct2Vec [116] also consider second-order proximity for node embedding. On the one hand, Metapath2Vec extended the DeepWalk approach using a meta-paths sampling process to preserve the semantics of a heterogeneous network. On the other hand, Struct2Vec better preserved the structural functionality of each node. Similarly, the SkipGram model has been applied to a semi-supervised setting: Planetoid [155] leverage the second-order proximity as well as node label to improve the classification performance; while TriDNR [108], a tri-party deep network representation model, leverage node structure, node content, and node labels (if available) to jointly learn optimal node representation.

Besides the SkipGram model, inspired by the success of recurrent neural networks (RNNs) in NLP tasks such as neural language modeling, there has been attempt to apply LSTM and GRU architecture to learn the network structure. More precisely, the random walks are seen as sequences modeled by an RNN. For example, [42, 92] proposed to used LSTM to learn a proximity embedding leveraging sub-graph structure. Instead, [83] uses the GRU architecture to embed information cascade paths.

2.3.2 Non-Random Walk Models

Besides using sampled paths as a 3-rd party to learn the graph structure, there have been models that work directly on the networks or the matrix representation of the network. Among the others, autoencoders have been proven useful to preserve the graph property in a low-dimensional space. [141] uses autoencoders to preserve first- and second-order proximity. Instead, [24] used denoizing autoencoder on a pointwise mutual information (PMI) matrix to obtain graph embedding; sparse autoencoders are shown to be related to spectral clustering in [131].

Other than autoencoders, PRUNE [77] proposes a Siamese architecture able to model global node ranking and first-order proximity by $O(|E|)$. However, the authors argue that the edges representation obtained is also able to *implicitly* preserve the community structure as well as the second-order proximity by representation propagation.

Recently, the convolutional operator, initially defined only on Euclidean space, has been extended to the manifold as well as graph domain; more in [99]. On the one hand, Bruna et al. were the first to propose an extension of the convolutional operator on the *spectral domain*. To the best of our knowledge, this is the first research to propose the graph convolutional network (GCN). However, only in [35, 73] a solid theoretical background for localized graph filters has been proposed. Such non-trivial extension is known to be expensive to compute ($O(|V|^2)$); thus a polynomial approximation, which considers only the K -nearest neighbors, is adopted. Still, the GCN architecture is not able to assign different importance to nodes of the same neighborhood, thus it is subject to overfitting.

On the other hand, the first deep learning approach directly applied to the *vertex domain* of a graph was proposed in [121]. The suggested graph neural network (GNN) utilizes an iterative process, named message passing, to maintain the neighbors' invariance. Thus, the authors developed an efficient method to convolve all the neighbors directly in the vertex domain. Subsequently, [85] enhanced the original architecture with a gating mechanism. Recently, similar iterative message passing architecture was proposed in [48, 72, 140] to study the graph structure based on the network attribute; however, [139] presented an alternative approach wherein an attention mechanism is used to dynamically determine the importance of each neighbor w.r.t. a given node, showing promising results on both inductive as well as transductive learning tasks. Finally, a comprehensive review of the deep learning model applied to graphs can be found in [161].

2.3.3 Interactive Networks

Introduced initially in [9], interaction networks became a powerful architecture that can effectively represent the relations between the components of different systems. Nevertheless, in their initial proposition, interaction networks require to manually craft those relations to obtain a meaningful view of the problem. Under a similar setting, [66] successfully demonstrates that it is possible to simplify a problem by allowing semantically similar vertices to share their parameters, reducing the complexity and the training time of the model. Moreover, the authors argued that thanks to its feedforward architecture, the Structural-RNN model can learn a meaningful feature representation that maintains the overall spatio-temporal graph structure. Subsequently, [48, 120] demonstrate that it is possible to infer such object interaction automatically, but the proposed message passing architecture appears to be inefficient since it performs an all-vs-all comparison if no prior information is provided. Finally, Kipf et al. shows that predicting different edge types better represents an implicit graph structure, but in the presence of a dynamic system.

Similar approaches are also explored under reinforcement learning settings, where the coordination between multiple agents is a crucial functionality to solve for most of the real-world problems. In [79], a novel variational inference model is shown to be capable of learning an interaction graph between multiple agents while a traditional learning algorithm is used to maximize the future expected reward. Applying coordinated stochastic gradient descent, the overall model effectively learns to imitate a given trajectory while improving the hidden role assignment of each agent.

A similar achievement is also obtained in [125] where the authors proposed a continuous communication channel between different agents to achieve coordination at a system level.

2.4 Knowledge Graph

As stated in Ch. 1.1.1, each knowledge graph entity is represented as a triple (h, r, t) . Thus, given a knowledge graph G , its embedding is recognized as a ranking problem where true triple (h, r, t) have a higher score than false triple (h', r', t') that does not exist in G . Let $f_r(h, t)$ be the distance between the head and tail entities based on relation r ; it is possible to obtain a graph embedding minimizing the following loss function:

$$\min \mathcal{L}(G) = \min \sum_{(h,r,t) \in G} \sum_{(h',r',t') \notin G} \max\{0, \gamma + f_r(h, t) - f_r(h', t')\} \quad (2.10)$$

where γ is a parameter defining the minimum distance between the true and false triples. To the best of our knowledge, TransE [17] is the first work proposing such embedding strategy, defining $f_r(h, t) = ||h + r - t||$. Subsequently, other methods propose different distance functions [67, 68, 87, 146] or they jointly optimize Eq. 2.10 and other objective function to preserve additional information. For example, [150] combined a topic model loss with a triple ranking loss, while [54] added a regularization term to obtain semantically smooth embedding.

2.5 Community Detection

Community detection aims to discover groups of nodes in a graph, such that the intra-group connections are denser than the inter-group ones [143]. Most of the existing work first embeds the nodes in a low-dimensional space by either feature engineering or graph embedding. Then they apply clustering, e.g., spectral clustering [130], Laplacian-regularized GMM [62], or neural networks [76, 131, 154] on the nodes for community detection. Despite the success of these methods, their results are sub-optimal since their node embedding does not explicitly consider community structure.

Recent work advances community detection by exploiting rich types of interaction on a social network such as content [119], attributes [126], and information diffusion [22]. Moreover, some methods exploit opportunities of coupling community detection with other tasks, such as hole spanner detection [60] or social role detection [58]. Comparatively, the proposed models are different in that:

1. ComE and ComE+ are graph embedding models, whereas the above community detection work generally does not rely on an embedding process;
2. our models focus on the most basic graph setting where we have a homogeneous graph as input, whereas some community detection work may require additional inputs of the graph.

2.6 Bayesian Embedding

As overstated, graph embedding tries to embed a node into a “point” vector space [16, 110]. Instead, a community has to characterize both what its member nodes are and how they spread; we consider a Bayesian definition of community embedding. That is, we define each community embedding as a distribution rather than a point in the low-dimensional space. Few studies consider Bayesian embedding. For example, KG2E [61] models each entity/relation with a multivariate Gaussian distribution in the low-dimensional space as its embedding and uses these distributions as representations for the triples of a knowledge graph. GenVector [156] embeds each social network user and each word as a vector, and then tries to infer latent topics from each user’s social posts, which are assumed to generate the user/word embedding. TransG [151] considers each relation in a knowledge graph as having multiple semantics and models an up to an infinite number of multivariate Gaussian distributions for relation embedding. Our work is different from the above work:

1. KG2E and TransG consider knowledge graphs and GenVector considers graphs with content; however we consider basic homogeneous graphs;
2. they do not consider community or try to improve node embedding by community detection.

Some studies have considered communities in Bayesian graph embedding. For example, LPCM (latent position cluster model) [59] embeds each node as a vector and models the first-order proximity by a logistic function. It also models each node’s embedding as generated by its communities’ multivariate Gaussian distributions. However, its number of communities is finite and given. An extension of LPCM is considered in [113] with a more efficient inference algorithm, and a different likelihood function of LPCM is explored in [128] for quantifying the residual uncertainty of model parameters. Our work is different from the above work:

1. they cannot handle an infinite number of communities;
2. they overlook the second-order proximity in node embedding.

2.7 Conclusion

In this chapter, we have overviewed the major works done on node embedding and interaction graph. On the one hand, due to the high amount of work done in this field, a summary of the most similar works done w.r.t. ComE and ComE+ is given in Tab. 2.1. As is possible to notice, besides its relevance for multiple applications, community embedding remains an under-explored research area and only few works address this problem explicitly, preserving first-, second-, and community-aware higher-order of proximity. However, this dissertation aims to define some guidelines on how to approach such problem.

On the other hand, Tab. 2.2 gives a brief overview of the work done on interactive networks. Few studies have considered dynamic attributed graphs; thus, in Ch. 5, we aim to fill this gap in the current literature.

Table 2.1: Comparison with related work on graph embedding. Here, “•” indicates an property as explicitly defined, and “o” indicates an property as implicitly derived. Complexity is defined over only graph-related factors.

	Node embedding			Comm. embed.	Comm. detect.	Infinite comm.	Complexity
	1st-order	2nd-order	higher-order				
DeepWalk [110]		•					$O(V \log V)$
Node2Vec [53]		•					$O(V \log V + V a^2)$
LINE [129]	•	•					$O(a E)$
SDNE [141]	•	•					$O(a V)$
GraRep [23]			•				$O(V ^3)$
Struc2Vec [116]		•					$O(V ^3)$
Metapath2Vec [37]		•					$O(V)$
Spectral [130]					•		$O(V ^3)$
PRUNE [77]	•	o	o				$O(E)$
DNR [154]			•		•		$O(V ^2)$
M-NMF [144]	•	•	•	o	•		$O(a V ^2)$
LPCM [59]	•		•	•	•		$O(V ^2)$
ComE	•	•	•	•	•		$O(V + E)$
ComE+	•	•	•	•	•	•	$O(V + E)$

Table 2.2: Comparison with related work on interactive network. Here, “•” indicates an property as explicitly defined, and “o” indicates an property as implicitly derived.

	Task level	Time series align	Node partition	Node attention
VIAN [65]	Node level			•
GAT [139]	Node level			•
GAM [81]	Graph level			•
Structural-RNN [66]	Node level		•	
NRI [72]	Node level		o	
DIAN	Node level	•	•	•

Chapter 3

ComE

3.1 Introduction

Graphs are an instinctive representation framework for many real-world applications, e.g., social media analytics, user’s interest diffusion, knowledge representation/competition/extraction, and protein interaction. Analyzing such graphs is a key factor to take a strategic decision; thus, it has received significant attention in the last few decades. Traditionally, graph embedding, which aims to represent a graph structure on a d -dimensional space preserving some predefined graph’s properties, became a popular technique to extract hidden information present in such a rich structure. However, most studies focused on a node level representation in which each vertex is associated to a vector such that two “related” nodes have similar vector representations [21]. Such a representation has proven successful for many different applications, e.g., node classification [23, 45, 74, 110], node clustering [131, 154], node recommendation/ranking [160, 164], link prediction [53, 77, 106], graph visualization [129, 141], and other similar applications [16, 104, 133, 142].

While the abovementioned studies mostly focused on the microscopic architectures formed between nodes, this dissertation aims to study another important yet largely under-explored setting of graph embedding, which focused on preserving the *community* structures formed by densely connected nodes. Community embedding is useful for many community-level applications, e.g., for community visualization, to help generate insights from big graphs or community recommendation to search for similar communities.

Although the concept of community is self-explanatory, there is no commonly accepted quantitative definition for it. Indeed, communities are one of the most prominent and complex networks’ feature because blurred overlapping boundaries characterize them, whereas nodes can simultaneously belong to different communities at the same time. According to Def. 1.3.1, a community is a group of densely connected nodes; thus, community embedding is expected to characterize how its member nodes are distributed in the low-dimensional space. As a consequence, it is impossible to represent the communities as a vector in a d' space; rather, it is more appropriate to formulate a community as a *distribution* in the same node embedding space. Similarly, traditional clustering algorithms used for community detection are based on distance metrics; consequently, they are affected by the multi-membership

Figure 3.1: Embedding nodes and communities in a 2D space.

problem while producing artificial and hard to explain outcomes [158]. To this end, we were inspired by the Gaussian mixture model (GMM) [13] to obtain a probabilistic representation in which each community is defined as a multivariate Gaussian distribution in the node embedding space. In this manner, each node affects, in big or small proportion, the representation of each community, while a community provides an explicit formulation of how the nodes spread in the latent space.

In Fig. 3.1, a community-preserving 2D embedding is reported for the well-studied Karate Club graph¹. Note that the Karate Club network has 34 nodes and 78 edges; moreover, it is known to have two communities formed after the club partitioned because of certain internal management divergence [157]. During the partitioning phase, some members are declared supporters of the instructor’s partition (node 1), while others are supporters of the club administrator (node 34). Among the others, some nodes (i.e. node 9) are known to support both groups. Thus, they are commonly recognized as “weak supporters” of both communities because they neither fully support the administrator nor the instructor’s partition. In Fig. 3.1b, the network is visualized in a 2D space, in which each node embedding is a 2D vector. Because each community is a group of densely connected nodes, we examine each community embedding as a *multivariate Gaussian distribution* in the 2D space. Consequently, the two overlapping communities are envisioned as two overlapping ellipses, each of which is characterized by a 2D mean vector and a 2×2 covariance matrix. Community embedding is useful for many community-level applications; e.g., community visualization to help generate insights from big graphs or community recommendation to search for similar communities.

Note that learning community embedding is non-trivial. On the one hand, to have meaningful community embedding, it is necessary to identify the communities well. Then, a straightforward approach for community embedding requires the following:

1. run community detection, such as spectral clustering [130], on the graph to obtain community assignments for each node;
2. apply node embedding, such as DeepWalk [110] or LINE [129], on the graph to obtain an embedding vector for each node;

¹<https://networkdata.ics.uci.edu/data.php?id=105>

Figure 3.2: A closed loop for learning community embedding.

3. aggregate the node embedding vectors in each community to fit a (multivariate Gaussian) distribution as its community embedding.

Such a pipeline approach is suboptimal because its community detection is independent of its node embedding.

As mentioned in Ch. 2, there are few studies that consider both node embedding and community detection; they either require extra supervision (e.g., must-links) [154] or high computational complexity (e.g., quadratic to the number of nodes in a graph) [144]. Moreover, they under-characterize a community in the low-dimensional space as a vector; thus, it is difficult to accurately visualize overlapping communities. Our insight for learning community embedding is that there exists a closed loop among community detection, community embedding, and node embedding, as shown in Fig. 3.2. On the one hand, as discussed earlier, node embedding can help improve community detection (i.e., ①), which outputs good communities for fitting meaningful community embedding (i.e., ②). On the other hand, community embedding can be used to optimize node embedding (i.e., ③). Suppose for a community k , we already have its community embedding as a multivariate distribution in a low-dimensional space. Then, we can enforce community k ’s member nodes to closely scatter near its community embedding’s mean vector in that low-dimensional space. As a result, these same community nodes tend to have similar node embedding vectors. Compared with first- and second-order proximity, community embedding does not focus on well-defined microscopic structure but instead focuses on the complex mesoscopic construction generated by densely connected nodes, thus introducing a *community-aware high-order proximity* to the node embedding. This feedback from community embedding to node embedding helps us to close the loop. Hopefully, the community-aware node embedding can serve as better inputs for the subsequent community detection, thus leading to more meaningful community embedding results.

Guided by the closed loop insight, we propose **ComE**, a novel Community Embedding framework that jointly solves community embedding, community detection, and node embedding. We define community embedding as a multivariate Gaussian distribution and use it to empower community detection from the node embedding results using a Gaussian mixture formulation. We denote a graph as $G = (V, E)$, where V is the set of nodes and E is the set of edges. This Gaussian mixture formulation has proven to be able to efficiently detect the communities [62] and infer their community embedding distributions from G in $O(|V|)$ time. Given community assignments and community embedding, the neural network formulations of DeepWalk and LINE are extended to preserve first-, second- and high-order (community-aware) proximity together. For this neural network formulation, a scal-

able inference algorithm is derived whose complexity is linear to the graph size $O(|V| + |E|)$.

We summarize our contributions as follows:

- We introduce a novel joint modeling framework, which leverages the closed loop among node embedding, community detection, and community embedding, to learn graph embedding.
- We contribute with a scalable inference algorithm that complexity of $O(|V| + |E|)$ is often lower than the existing higher order of proximity-aware methods (Tab. 2.1).
- We evaluate ComE on multiple real-world datasets with various application tasks. It renders better graph visualization results and improves the state-of-the-art baselines by at least 1.8%–6.7% (NMI) and 0.3%–17.1% (Conductance) in community detection, 0.8%–20.1% (Macro-F1), and 0.7%–30.4% (Micro-F1) in node classification.

3.2 Problem Formulation

A graph $G = (V, E)$ is provided as an input, where V is the node set and E is the edge set. Traditional graph embedding aims to learn a node embedding for each $v_i \in V$ as $\phi_i \in \mathbb{R}^d$; however, in this dissertation, we aim to learn a community embedding. Suppose there are K communities on the graph G . For each node v_i , its community assignment is denoted as $z_i \in \{1, \dots, K\}$. Motivated by the Gaussian mixture formulation [13], we defined community embedding as a multivariate Gaussian distribution in a low-dimensional space. The GMM is proven to be capable of detecting communities in complex networks [62, 158] while providing a sound probabilistic formulation suitable to model the uncertainty that is typical of unsupervised tasks.

Definition 3.2.1. Community embedding of a community k (with $k \in \{1, \dots, K\}$) in a d -dimensional space is a multivariate Gaussian distribution $\mathcal{N}(\psi_k, \Sigma_k)$, where $\psi_k \in \mathbb{R}^d$ is a mean vector and $\Sigma_k \in \mathbb{R}^{d \times d}$ is a covariance matrix.

As output, we aim to learn the following:

1. node embedding ϕ_i for each node $v_i \in V$;
2. community membership π_{ik} , such that $\sum_{k=1}^K \pi_{ik} = 1$, for each node $v_i \in V$ and each community $k \in \{1, \dots, K\}$;
3. community embedding parameters (ψ_k, Σ_k) for each community $k \in \{1, \dots, K\}$.

We summarize all our notations in Tab. 2.

3.2.1 Community Detection and Embedding

Given node embedding, one straightforward way to detect communities and learn their community embedding is to consider a pipeline approach. For example, as shown in Fig. 3.2, we can run spectral clustering to detect communities, and then fit a Gaussian component for each community. However, such a pipeline approach lacks a unified objective function; thus, it is difficult to optimize later with node embedding.

Alternatively, we performed community detection and embedding together in one single objective function based on GMM, i.e., we consider each node v_i 's embedding ϕ_i as generated by a multivariate Gaussian distribution from a community $z_i = k$. Then, for all the nodes in V , we have the likelihood as follows:

$$\prod_{i=1}^{|V|} \sum_{k=1}^K p(z_i = k) p(v_i | z_i = k; \phi_i, \psi_k, \Sigma_k) \quad (3.1)$$

where $p(z_i = k)$ is the probability of node v_i belonging to community k . For notation simplicity, we denote $p(z_i = k)$ as π_{ik} ; thus, we have $\pi_{ik} \in [0, 1]$ and $\sum_{k=1}^K \pi_{ik} = 1$. Moreover, $p(v_i | z_i = k; \phi_i, \psi_k, \Sigma_k)$ is a multivariate Gaussian distribution defined as follows:

$$\begin{aligned} p(v_i | z_i = k; \phi_i, \psi_k, \Sigma_k) &= \mathcal{N}(\phi_i | \psi_k, \Sigma_k) \\ &= \frac{\exp\left(-\frac{1}{2}(\phi_i - \psi_k)^T \Sigma_k^{-1} (\phi_i - \psi_k)\right)}{\sqrt{(2\pi)^d |\Sigma_k|}} \end{aligned} \quad (3.2)$$

Such problem formulations entail the usage of the GMM for two distinct scopes:

1. for community detection: by seeing π_{ik} as an unknown variable, the intent is to learn a *mixed community membership* for each node v_i ;
2. for community embedding: by delineating (ψ_k, Σ_k) as unknown, it is possible to represent each community by its mean vector and covariance matrix.

By optimizing Eq. 3.1 w.r.t. π_{ik} 's and (ψ_k, Σ_k) 's, we achieve community detection and embedding at the same time.

3.2.2 Node Embedding

Traditionally, node embedding focuses on preserving first- or second-order proximity. For example, to preserve first-order proximity, LINE [129] enforces two neighboring nodes to have similar embedding by minimizing

$$\begin{aligned} O_1 &= - \sum_{(v_i, v_j) \in E} \log \sigma(\phi_j^T \phi_i) \\ &= - \sum_{(v_i, v_j) \in E} \log \frac{1}{1 + \exp(-\phi_j^T \phi_i)} \end{aligned} \quad (3.3)$$

where $\sigma(x)$ is a sigmoid function. To preserve second-order proximity, both LINE and DeepWalk [110] enforce two nodes sharing many ‘‘contexts’’ (i.e., neighbors within ζ hops) to have similar embedding. In this case, each node has two roles: as

a node for itself and as a context for some other nodes. To differentiate such roles, DeepWalk introduces an extra context embedding for each node v_j as $\phi'_j \in \mathbb{R}^d$. We denoted C_i as the set of contexts for v_i . Then, we adopted *negative sampling* [102] to define a function for measuring how well v_i generates each of its contexts $v_j \in C_i$ as follows:

$$\Delta_{ij} = \log \sigma(\phi_j'^T \phi_i) + \sum_{t=1}^m \mathbb{E}_{v_l \sim P_n(v_i)} [\log \sigma(-\phi_l'^T \phi_i)] \quad (3.4)$$

where $v_l \sim P_n(v_i)$ denotes sampling a node $v_l \in V$ as a “negative context” of v_i according to a probability $P_n(v_l)$. We set $P_n(v_l) \propto r_l^{3/4}$ as proposed in [102], where r_i is v_i ’s degree. In total, there are m negative contexts. Generally, maximizing Eq. 3.4 enforces node v_i ’s embedding ϕ_i to best generate its positive contexts ϕ_j' ’s but not its negative contexts ϕ_l' ’s. Then, we can minimize the following objective function to preserve the second-order proximity:

$$O_2 = -\alpha \sum_{v_i \in V} \sum_{v_j \in C_i} \Delta_{ij} \quad (3.5)$$

where $\alpha > 0$ is a trade-off parameter.

3.2.3 Closing the Loop

To close the loop in Fig. 3.2, we need to enable the feedback from community detection and community embedding to node embedding. Suppose we have identified the mixed community membership π_{ik} ’s and the community embedding (ψ_k, Σ_k) ’s in Ch. 3.2.1. Then, we can re-use Eq. 3.1 to enable such feedback but rather than using the traditional GMM approach, the feedback is achieved by seeing the node embedding ϕ_i as the *unknown* variable. Effectively, optimizing Eq. 3.1 w.r.t. ϕ_i enforces the nodes within the same community to get closer to the corresponding community center ψ_k , i.e., two nodes sharing a community are likely to have a similar embedding. Compared with the first- and second-order proximity, this design enforces community-aware high-order proximity on node embedding, which is later useful for community detection and embedding. For example, in Fig. 3.1, node 3 and node 10 are directly linked, but they tend to belong to two different communities according to [157]. Therefore, by only preserving first-order proximity, we may not be able to identify their community membership’s difference well. For another example, node 9 and node 10 share a number of one-hop and two-hop neighbors, but compared with node 10, node 9 tend to be closer to community led by node 1 according to [157]. Therefore, by only preserving second-order proximity, we may not be able to identify their community membership’s difference well either.

Based on the closed loop, we optimized community detection, community embedding, and node embedding together. We have three types of proximity to consider for node embedding, including first-, second-, and high-order proximity. In general, there are two approaches to combine different types of proximity for node embedding:

1. “concatenation”, e.g., LINE first separately optimizes O_1 and O_2 , and then it concatenates the two resulting embedding for each node into a long vector as the final output;

2. “unification”, e.g., SDNE [141] learns a single node embedding for each node to preserve both first- and second-order proximity at the same time.

In this study, to encourage the node embedding to unify multiple types of proximity, we adopt the unification approach and leave the other approach as future work. Consequently, based on Eq. 3.1, we define the objective function for community detection and embedding as well as enforce the high-order proximity for node embedding as follows:

$$O_3 = -\frac{\beta}{K} \sum_{i=1}^{|V|} \log \sum_{k=1}^K \pi_{ik} \mathcal{N}(\phi_i | \psi_k, \Sigma_k), \quad (3.6)$$

where $\beta \geq 0$ is a trade-off parameter. We denote $\Phi = \{\phi_i\}$, $\Phi' = \{\phi'_i\}$, $\Pi = \{\pi_{ik}\}$, $\Psi = \{\psi_k\}$, and $\Sigma = \{\Sigma_k\}$ for $i = 1, \dots, |V|$ and $k = 1, \dots, K$. Then, we unify the first- and second-order proximity for node embedding. The ultimate objective function for ComE is as follows:

$$\begin{aligned} \mathcal{L}(\Phi, \Phi', \Pi, \Psi, \Sigma) = & O_1(\Phi) \\ & + O_2(\Phi, \Phi') \\ & + O_3(\Phi, \Pi, \Psi, \Sigma) \end{aligned} \quad (3.7)$$

Our final optimization problem becomes as follows:

$$(\Phi^*, \Phi'^*, \Pi^*, \Psi^*, \Sigma^*) \leftarrow \arg \min_{\forall k, \text{diag}(\Sigma_k) > \mathbf{0}} \mathcal{L}(\Phi, \Phi', \Pi, \Psi, \Sigma) \quad (3.8)$$

where $\text{diag}(\Sigma_k)$ returns the diagonal entries of Σ_k . We particularly introduce a constraint of $\text{diag}(\Sigma_k) > \mathbf{0}$ for each $k \in \{1, \dots, K\}$ to avoid the singularity issue of optimizing $\mathcal{L}$. Similar to the Gaussian mixture model, there exists degenerated solutions for optimizing $\mathcal{L}$ without any constraint [12], i.e., when a Gaussian component collapses to a single point, the $\text{diag}(\Sigma_k)$ becomes zero, which makes O_3 become negative infinity.

3.3 Inference

Since our objective in Eq. 3.8 can be seen as consisted of node embedding and community embedding, we decompose the optimization into two parts and follows an iterative approach to solve it. In particular, we consider iteratively optimizing (Π, Ψ, Σ) with a constrained minimization given (Φ, Φ') , and optimizing (Φ, Φ') with an unconstrained minimization given (Π, Ψ, Σ) . Empirically, this iterative optimization algorithm quickly converges with a reasonable initialization; e.g., we initialize (Φ, Φ') by DeepWalk results in our experiments. We report the convergence in Ch. 3.4.5. Next, we detail this iterative optimization.

Fix (Φ, Φ') , optimize (Π, Ψ, Σ) . In this case, Eq. 3.8 is simplified as inferring a GMM with the constraints of $\text{diag}(\Sigma_k) > \mathbf{0}$ for each $k \in \{1, \dots, K\}$. To solve this constrained optimization, we adopt the approach as suggested by [13]; i.e., we use *expectation maximization* (EM) algorithm [36] to infer (Π, Ψ, Σ) and meet the constraint via suitable heuristics of randomly resetting $\Sigma_k > \mathbf{0}$ and $\psi_k \in \mathbb{R}^d$

whenever a $\text{diag}(\Sigma_k)$ starts to have zero. In particular, using EM, we can iteratively update the (Π, Ψ, Σ) as follows:

$$\pi_{ik} = \frac{N_k}{|V|}, \quad (3.9)$$

$$\psi_k = \frac{1}{N_k} \sum_{i=1}^{|V|} \gamma_{ik} \phi_i, \quad (3.10)$$

$$\Sigma_k = \frac{1}{N_k} \sum_{i=1}^{|V|} \gamma_{ik} (\phi_i - \psi_k)(\phi_i - \psi_k)^T, \quad (3.11)$$

where $\gamma_{ik} = \frac{\pi_{ik} \mathcal{N}(\phi_i | \psi_k, \Sigma_k)}{\sum_{k'=1}^K \pi_{ik'} \mathcal{N}(\phi_i | \psi_{k'}, \Sigma_{k'})}$ and $N_k = \sum_{i=1}^{|V|} \gamma_{ik}$. Note that in practice, if (Φ, Φ') is reasonably initialized (e.g., by DeepWalk in our experiments), the constraints of $\text{diag}(\Sigma_k) > \mathbf{0}$ are easily satisfied; thus, the inference of (Π, Ψ, Σ) can quickly converge.

Fix (Π, Ψ, Σ) , optimize (Φ, Φ') . In this case, Eq. 3.8 is simplified as an unconstrained optimization over the node embedding with three types of proximity. Because of the summation within the logarithmic term of O_3 , it is inconvenient to compute the gradient of ϕ_i . Thus, we try to minimize an upper bound of $\mathcal{L}(\Phi, \Phi' | \Pi, \Psi, \Sigma)$ instead. Specifically, we introduce the following:

$$O'_3 = -\frac{\beta}{K} \sum_{i=1}^{|V|} \sum_{k=1}^K \pi_{ik} \log \mathcal{N}(\phi_i | \psi_k, \Sigma_k) \quad (3.12)$$

It is easy to prove that $O'_3(\Phi | \Pi, \Psi, \Sigma) \geq O_3(\Phi | \Pi, \Psi, \Sigma)$ because of the following log-concavity

$$\begin{aligned} \sum_{i=1}^{|V|} \log \sum_{k=1}^K \pi_{ik} \mathcal{N}(\phi_i | \psi_k, \Sigma_k) \\ \geq \\ \sum_{i=1}^{|V|} \sum_{k=1}^K \log \pi_{ik} \mathcal{N}(\phi_i | \psi_k, \Sigma_k) \end{aligned} \quad (3.13)$$

As a result, we define:

$$\begin{aligned} \mathcal{L}'(\Phi, \Phi' | \Pi, \Psi, \Sigma) &= O_1(\Phi) \\ &\quad + O_2(\Phi, \Phi') \\ &\quad + O'_3(\Phi | \Pi, \Psi, \Sigma) \end{aligned} \quad (3.14)$$

and thus $\mathcal{L}'(\Phi, \Phi' | \Pi, \Psi, \Sigma) \geq \mathcal{L}(\Phi, \Phi' | \Pi, \Psi, \Sigma)$. We optimize $\mathcal{L}'(\Phi, \Phi')$ by *stochastic gradient descent* (SGD) [102]. For each $v_i \in V$, we have the following:

$$\frac{\partial O_1}{\partial \phi_i} = - \sum_{(i,j) \in E} \sigma(-\phi_j^T \phi_i) \phi_j \quad (3.15)$$

$$\frac{\partial O_2}{\partial \phi_i} = -\alpha \sum_{v_j \in C_i} \left[\sigma(-\phi_j'^T \phi_i) \phi_j' + \sum_{t=1}^m \mathbb{E}_{v_l \sim P_n(v_l)} [\sigma(\phi_l'^T \phi_i) (-\phi_l')] \right] \quad (3.16)$$

$$\frac{\partial O'_3}{\partial \phi_i} = \frac{\beta}{K} \sum_{k=1}^K \pi_{ik} \Sigma_k^{-1} (\phi_i - \psi_k) \quad (3.17)$$

We also compute the gradient for context embedding as follows:

$$\begin{aligned} \frac{\partial O_2}{\partial \phi_j'} &= -\alpha \sum_{v_i \in V} \left[\delta(v_j \in C_i) \sigma(-\phi_j'^T \phi_i) \phi_i \right. \\ &\quad \left. + \sum_{t=1}^m \mathbb{E}_{v_l \sim P_n(v_l)} [\delta(v_l = v_j) \sigma(\phi_l'^T \phi_i) (-\phi_i)] \right] \end{aligned} \quad (3.18)$$

3.3.1 Algorithm and Complexity

ALGORITHM 1: Inference algorithm for ComE

Input: graph $G = (V, E)$, k , $\#(\text{paths per node}) \gamma$, walk length ℓ , context size ζ , embedding dimension d , negative context size m , parameters (α, β)

Output: node embedding Φ , context embedding Φ' , community embedding (Ψ, Σ)

```

1  $\mathcal{P} \leftarrow \text{SamplePath}(G, \ell);$ 
2 Initialize  $\Phi$  and  $\Phi'$  by DeepWalk [110] with  $\mathcal{P}$ ;
3 for  $iter = 1 : T_1$  do
4   for  $subiter = 1 : T_2$  do
5     Update  $\pi_{ik}$ ,  $\psi_k$  and  $\Sigma_k$  by Eq. 3.9–3.11;
6     for  $k = 1 : K$  do
7       if there exists zero in  $\text{diag}(\Sigma_k)$  then
8         Randomly reset  $\Sigma_k > \mathbf{0}$  and  $\psi_k \in \mathbb{R}^d$ ;
9     foreach  $edge (i, j) \in E$  do
10      SGD on  $\phi_i$  and  $\phi_j$  by Eq. 4.3;
11    foreach  $node v_i \in V$  do
12      SGD on  $\phi_i$  by Eq. 3.17;
13    foreach  $path p \in \mathcal{P}$  do
14      foreach  $v_i$  in  $path p$  do
15        SGD on  $\phi_i$  by Eq. 3.16;
16        SGD on  $\phi'_j$ 's within  $\zeta$  hops by Eq. 3.18;
```

We summarize the inference algorithm of ComE in Alg. 1. In line 1, for each $v_i \in V$, we sample γ paths starting from v_i with length ℓ on G . In line 2, we initialize (Φ, Φ') by DeepWalk. In lines 4–8, we fix (Φ, Φ') and optimize (Π, Ψ, Σ) for community detection and embedding. In lines 9–16, we fix (Π, Ψ, Σ) and optimize (Φ, Φ') for node embedding. In particular, we update node embedding by first-order proximity (lines 9–10), second-order proximity (lines 11–14), and community-aware high-order proximity (lines 15–16).

We analyze the complexity of Alg. 1. In line 1, path sampling takes $O(|V|\gamma\ell)$. In line 2, parameter initialization by DeepWalk takes $O(|V|)$. In line 5, community detection and embedding takes $O(|V|K)$. In lines 6–8, checking constraint takes $O(K)$. In lines 9–10, node embedding w.r.t. first-order proximity takes $O(|E|)$. In lines 11–14, node embedding w.r.t. second-order proximity takes $O(|V|\gamma\ell)$. In lines 15–16, node embedding w.r.t. community-aware high-order proximity takes $O(|V|K)$. In total, the complexity is $O(|V|\gamma\ell + |V| + T_1 \times (T_2|V|K + K + |E| + |V|\gamma\ell + |V|K))$, which is linear to the graph size (i.e., $|V|$ and $|E|$). Thus, the proposed algorithm is efficient. A proof of convergency is given in Appx. 6.2.

3.4 Experiments

As community embedding is useful for visualizing communities in a low-dimensional space and helping both community detection and node embedding, we design three evaluation tasks for experiments: *graph visualization*, *community detection*, and *node classification*. Moreover, in this section, we empirically study the model convergence and parameter sensitivity.

3.4.1 Experiment Setting

To fully understand the obtained results, we are going to define the settings used for all the experiments. Initially, we report a review of the datasets and the evaluation metrics that have been employed. Subsequently, a summary of the baselines used for comparison is provided in addition to Ch. 2. Finally, a description of the parameters setting adopted by each algorithm is given.

Datasets

We use seven public graph datasets for evaluation. These graphs are of various types, ranging from social networks to word co-occurrence network and academic paper citation network.

- *BlogCatalog*² is a social network for users to publish blogs. In this dataset, each node is a BlogCatalog user and each edge is a friendship connection. Each node has multiple labels, indicating the topics of the user’s blog topics.
- *Wikipedia*³ is a co-occurrence network of words appearing in the first million bytes of the Wikipedia dump. In this dataset, each node is a word and each edge is a word co-occurrence relationship. Each node has a label, indicating the word’s Part-of-Speech (POS) tag.
- *Rochester*, *Mich*, and *Amherst* are three Facebook networks for students from the same universities. Each node is a student, and an edge is their friendship connection. The matriculation year is used as the node’s label. We remove all the isolated nodes and unlabeled nodes for evaluations.
- *DBLP*⁴ is an academic paper citation network built upon the DBLP repository. We extracted the papers from 19 selected conferences from five areas, as shown in Tab. 3.2. In this dataset, each node is a paper and each edge is a citation. Each node has a label, indicating one of the five areas for its paper’s conference venue.
- *Karate Club* is a social network of a university karate club [157].

The statistics for each dataset are summarized in Tab. 3.1. Moreover, Fig. 3.3 shows the degree distribution for the six main graphs. It is interesting to observe that all the graphs appear to follow the power-law degree distribution, typical of

---

²<http://socialcomputing.asu.edu/datasets/BlogCatalog3>

³<http://snap.stanford.edu/node2vec/POS.mat>

⁴<https://aminer.org/billboard/aminernetwork>

Table 3.1: Datasets used for evaluation.

	# (node)	# (edge)	# (class)	# (label)	Avg. Degree	Avg. Clustering	Avg. Shortest Path
BlogCatalog	10,312	333,983	39	≥ 1	64.77	0.46	2.38
Wikipedia	4,777	92,406	40	≥ 1	38.69	0.53	2.03
DBLP	13,184	48,018	5	1	3.64	0.17	6.71
Rochester	4,140	145,305	19	1	70.19	0.28	2.63
Mich	2,924	54,903	13	1	37.55	0.27	2.83
Amherst	2,021	81,492	15	1	80.64	0.31	2.37
Karate Club	34	78	2	≥ 1	4.58	0.57	2.41

Table 3.2: DBLP dataset labels.

Conference	Label
EMNLP, ACL, CoNLL, COLING	NLP
CVPR, ICCV, ICIP, SIGGRAPH	Computer Vision
KDD, ICDM, CIKM, WSDM	Data Mining
SIGMOD, VLDB/PVLDB, ICDE	Database
INFOCOM, SIGCOM, MobiHoc, MobiCom	Networking

scale-free networks [111]. Such behavior is common in social and citation networks where the “preferential attachment” phenomenon is observable.

Another common characteristic of social networks is the “small-world” [147] effect. Small-world networks are characterized by a small average shortest path length and a “high clustering coefficient”. Wikipedia and BlogCatalog appear to follow such a definition, while the other datasets do not fully match such characteristics. Note that small-world networks are known to be a robust network structure, but they are subject to the presence of multiple hubs. Hubs allow a fast propagation of the information because they interconnect multiple nodes; however, they also interconnect nodes with different labels. Thus, they can affect the overall performance of a graph embedding algorithm, which erroneously forces node of different categories to be similar.

Evaluation Metrics

In community detection, we use both *Conductance* [75] and *normalized mutual information* (NMI) [131]. Conductance is basically a ratio between the number of edges leaving a community and that within the community. NMI measures the closeness between the predicted communities with ground truth based on the node labels. In node classification, we use *Micro-F1* and *Macro-F1* [110]. Micro-F1 is the overall F1 w.r.t. all types of labels. Macro-F1 is the average of F1 scores w.r.t. each type of label.

Figure 3.3: Degree distribution for all the graphs used for evaluation. Note that most of them appear to follow the power-law distribution.

Baselines

We design baselines to back up our arguments in Ch. 3.1 that it is non-trivial to learn community embedding.

- *SF*: We first design a straightforward approach that separates community detection and node embedding. Then, we fit the community embedding from the detection and node embedding results. We use spectral clustering [130] for community detection and use DeepWalk for node embedding; finally, we use GMM to fit community embedding. Note that because its node embedding is the same as DeepWalk, we will only evaluate SF in community detection and community visualization for the Karate dataset.

Moreover, we consider the other approach that runs community detection on the node embedding results from the following state-of-the-art baselines, and then runs GMM to detect communities and fit community embedding.

- *DeepWalk* [110]: it models second-order proximity in the embedding process.
- *LINE* [129]: it considers both first- and second-order proximity.
- *Node2Vec* [53]: it extends DeepWalk by exploiting homophily and structural roles in embedding.
- *GraRep* [23]: it models random walk-based high-order proximity.
- *M-NMF* [144]: it jointly models node and community embedding using non-negative matrix factorization.

We compared our model with all the baselines on all the datasets using the author-published codes whenever possible. The same grid-search strategy used to search the parameters for ComE has been used on all the baselines to fine-tune the

optimal setting. Similarly, all the embedding is generated providing a comparable amount of training data and iterations. Finally, all the experiments are performed on a Linux machine with eight 3.50 GHz Intel Xeon(R) CPUs and 16 GB memory.

Parameters and Environment

Our ComE has only two more parameters (i.e., α and β) than DeepWalk. To obtain a fair comparison, we follow the DeepWalk and Node2Vec works to set the parameters. Unless stated otherwise, for all the methods, we set the embedding dimension $d = 128$. For DeepWalk, Node2Vec, and ComE, we set $\gamma = \zeta = 10$, $\ell = 80$, and $m = 5$; however, for all the clustering experiments, we set K as the number of unique labels.

- **Node2Vec:** According to [53], for BlogCatalog $p = 0.25$ and $q = 0.25$ have been used; however, on Wikipedia, the used settings are $p = 4$ and $q = 1$. We followed the same tuning procedure for DBLP, Rochester, Mich, and Amherst and observed the following:

1. like on BlogCatalog, $p = 0.25$ and $q = 0.25$ works at best for DBLP, Rochester, and Mich;
2. on Amherst, $p = 0.25$ and $q = 1$ is the outstanding setting.

- **M-NMF:** we tune α and β in the range $[0.1, 1, 5, 10]$ while ensuring the other parameter is fixed. The final setting is as follows:

1. $\alpha = 0.1$ and $\beta = 5$ for BlogCatalog and Wikipedia;
2. $\alpha = 1$ and $\beta = 5$ for Rochester, Mich, and Amherst;
3. for DBLP, we used $\alpha = 10$ and $\beta = 5$.

3.4.2 Graph Visualization

We compared ComE with the baselines on both a small Karate Club graph and a bigger DBLP paper citation graph.

Karate Club results. We visualize the Karate Club graph in Fig. 3.1a based on the node and community embedding results of our baselines. As we can see, LINE (Fig. 3.4b) does not present community structures because it does not consider the community in its node embedding. DeepWalk, Node2Vec, GraRep, and M-NMF tend to present two separate communities, which cannot identify those weak supporters for the communities (e.g., node 9); however, SF detects noise communities because of the hard clustering results obtained from spectral clustering. In contrast, as shown in Fig. 3.1b, our ComE can clearly identify such weak supporters because of its joint modeling of overlapping community detection, community embedding, and node embedding.

Figure 3.4: Graph visualization on the Karate Club graph. Note that the community embedding is obtained using GMM with $K = 2$ while Fig. 3.4f is obtained using the spectral clustering’s community assignment to fit two bivariate Gaussian distributions.

DBLP results. We visualized the DBLP paper citation graph in Fig. 3.5. We used the t-SNE toolkit [134] for graph visualization. Rather than plotting community embedding eclipses, we use different node colors to visualize different communities and see if the algorithms are able to correctly preserve the communities present in the graphs. Although SF and DeepWalk have different ways to generate the community embedding, they use the same node embedding; thus, they have the same visualization results in Fig. 3.5a. First, note that no method is able to separate the green and the yellow classes (Data Mining and Database); we believe that this is related to the dataset itself given the similarity of the two topics. Moreover, DeepWalk and Node2Vec generate representations with many overlaps among the colors and the overall embedding is quite similar. This could be possible because both the models only consider the second-order proximity. LINE presents a bit more cohesive visualization probably because of its ability in preserving first and second-order proximity. We also observe that, in GraRep’s results, green nodes are spread in different places and mixed with red and yellow nodes, indicating a lack of clear community structure in the embedding. This might be possible because of its ability in decomposing a high-order transition probability matrix, which could dominate both Data Mining and Database nodes. However, GraRep is able to correctly detect Networking and Computer vision communities. M-NMF (Fig. 3.5e) instead presented a blurred community structure and a centre of gravity which attract all the nodes. This could be related to the fact that M-NMF relies on the

Figure 3.5: Graph visualization on the DBLP graph (better viewed in color). Different node colors indicate different communities; red is *NLP*, blue is *Computer Vision*, green is *Data Mining*, yellow is *Database* and black is *Networking*

modularity matrix to learn the community embedding, which could contains noise caused by the overlap between the green and yellow nodes. This hypothesis is also supported by the Karate results (Fig. 3.4e), in which M-NMF already demonstrate to struggle in modeling overlapping communities. Compared with all these baseline methods, our ComE can correctly detect three classes because of its joint modeling; however, as for the other methods, it is not able to differentiate between green and yellow nodes, which are clustered in the mixed communities.

3.4.3 Community Detection

In community detection, our aim is to predict the most likely community assignment for each node. As an unsupervised task, we use the whole graph for learning embedding, and then predict the communities for each node using GMM for all the methods that natively do not provide a community assignment. Note that BlogCatalog and Wikipedia are multi-labels datasets; therefore, we compute only the Conductance of each node because there is no clear way to calculate NMI for such a multi-label setting.

Remark: During the experiments, we set $\alpha = 0.1$ and $\beta = 0.1$ in ComE for all the datasets. Moreover, the reported results are the average of 10 independent executions.

As shown in Fig. 3.6, ComE outperforms all the baselines for both Conductance

Figure 3.6: Community detection results. The smaller Conductance (denoted as Cond.) is, the better. The bigger NMI is, the better.

and NMI on BlogCatalog, DBLP, Wikipedia, and Amherst. Specifically, ComE can improve the best baseline by relatively 0.3%–17.1% in terms of Conductance and 1.8%–6.7% in terms of NMI in the abovementioned datasets. Interestingly, on Mich, ComE can obtain a relative improvement of 1.8% on NMI compared to the best baseline; however, GraRep outperforms ComE in terms of Conductance by 0.9%. On Rochester, ComE can outperform all the baseline only for Conductance (2.3%), although it is comparable regarding NMI (-0.3%). However, according to Fig. 3.10d with the setting of $\alpha = 0.1$ and $\beta = 0.001$ ComE is able to consistently outperform the baselines also in NMI. Such phenomenon is related to the presence of multiple connected components and communities composed by a single node in Rochester and Mich. Thus, ComE is penalized by the constraint on Σ_k (Fig. ??), which prevents it from learning single-node communities; still a smaller value for β helps in overcoming such issue.

Overall, the results suggest that modeling community detection together with node embedding is better than solving them separately. The poor performance of the SF method also supports the hypothesis of having a closed loop model because the dense node representation obtained by the closed loop can better separate different communities. Moreover, we observe that on average, the graph embedding methods perform better than SF, which suggest the usefulness of graph embedding for community detection. However, it is not straightforward to identify an optimal approach among all the baselines: Node2Vec happens to be a good baseline for the community detection problem, since it shows the best performance in 3 datasets,

while on Wikipedia and Amherst GraRep and N-MNF shows some competitive advantage.

Finally, it is observed that graphs with a smaller clustering coefficient provide the best results. This is reasonable since on such graphs, it is easier for an embedding algorithm to preserve the overall structure in an embedding space. Instead, it is challenging to preserve (in a vector) all the closed triplets present in a graph with a high clustering coefficient.

3.4.4 Node Classification

In node classification, our aim is to categorize each node into one or more classes, depending on whether it is a single-label or multi-label setting. Based on [110], a graph embedding is initially generated for the whole graph, and then training sets are randomly sampled using from 40% up to 90% of the available nodes. The training sets have been used to train a LibSVM classifier ($c = 1$ for all the methods) [27]. Finally, the classifier’s performance has been tested using the remaining nodes.

Remark: Each reported result is the average of 10 independent evaluation with different training-testing splits. Note that Tab. 3.4 reports the p -values w.r.t. the experiment performed with 80% of the dataset as training.

ComE has been compared w.r.t. all the baselines using $\alpha = 0.1$ and $\beta = 0.1$ for all the datasets. Tab. 3.3 shows that ComE is generally better than the baselines in terms of both Macro-F1 and Micro-F1. In particular, ComE improves the best baselines by relatively 0.8%–20.1% (Macro-F1) and 0.7%–30.4% (Micro-F1) on BlogCatalog, Wikipedia, and DBLP when using 80% of the nodes as training set. However, on Rochester, Mich, and Amherst, the improvements only range from 1.7% to 4.5% (Macro-F1) and from 0.11% to 0.3% (Micro-F1). Our student t -tests (Tab. 3.4) show that almost all the improvements obtained on BlogCatalog, Wikipedia, and DBLP are significant, with one-tailed p -values always less than 0.01 except for GraRep on the Wikipedia dataset. Instead, the Micro-F1 performance for the remaining datasets is comparable w.r.t. GraRep and M-NMF.

This is partially a consequence of single-node communities present in the dataset, which affects the ComE performance. It is possible to argue that Facebook datasets and Wikipedia contain fewer nodes than the rest of the datasets. This affects the path sampling process because it leads to a comparatively smaller set of random walks. Moreover, GraRep and M-NMF leverage the adjacency matrix, which could contain more information than a fixed amount of sampled path. This provides supplemental information to train the classifiers with limited training labels, but comes at the cost of higher complexity ($O(|V|^3)$) for GraRep. While a straightforward solution to this problem would be the increase in the number of sampled paths, it would not be fair w.r.t. the other baselines; thus, we keep γ , ζ , and ℓ fixed. Still ComE can partially overcome such limitations since is able to leverage a higher-order of proximity.

From Tab. 3.3, it is also possible to obtain some interesting observations. First, it is possible to notice how GraRep is generally the best baseline; however, on DBLP Node2Vec is the best performer. As mentioned previously, this is a consequence of DBLP’s low density, which makes it easier for Node2Vec to explore the most mean-

Table 3.3: Node classification results.

BlogCatalog								Wikipedia					
	% Labels	40%	50%	60%	70%	80%	90%	40%	50%	60%	70%	80%	90%
Macro-F1 (%)	ComE	24.9	25.2	26.2	26.4	26.5	27.2	4.1	5.3	7.0	8.9	9.8	11.1
	DeepWalk/SF	20.6	20.9	21.5	21.5	22.2	22.5	3.1	3.3	3.4	3.6	4.6	8.1
	Line	10.1	10.2	10.8	10.8	10.9	10.9	2.5	2.8	3.3	3.9	5.3	8.9
	Node2Vec	22.8	23.1	23.4	23.6	24.1	24.6	3.9	3.9	4.4	5.1	6.1	10.9
	GraRep	22.9	22.9	23.3	23.5	23.6	24.0	4.8	5.0	5.8	6.3	8.1	11.5
	M-NMF	15.2	15.3	15.3	15.6	15.7	15.8	4.9	5.0	5.4	5.8	7.0	9.8
Micro-F1 (%)	ComE	38.0	38.5	40.5	41.1	41.7	42.8	24.0	27.9	32.9	40.8	44.0	51.1
	DeepWalk/SF	35.4	35.9	37.9	38.0	38.3	39.0	21.6	22.2	22.2	23.9	28.0	41.0
	Line	27.2	27.6	29.5	29.8	30.2	30.5	22.5	23.0	24.4	26.1	30.5	44.2
	Node2Vec	36.9	37.4	39.4	39.6	39.9	40.5	23.9	24.1	25.1	26.8	31.1	45.3
	GraRep	38.2	38.6	40.2	40.7	40.9	42.0	22.2	26.3	27.5	29.1	33.4	46.6
	M-NMF	31.0	31.5	33.2	33.4	33.8	34.4	22.9	27.1	28.2	29.6	33.7	46.1

DBLP								Rochester					
	% Labels	40%	50%	60%	70%	80%	90%	40%	50%	60%	70%	80%	90%
Macro-F1 (%)	ComE	92.1	92.1	92.2	92.2	92.2	92.5	31.8	34.3	37.8	43.0	49.7	57.8
	DeepWalk/SF	91.0	91.1	91.2	91.2	91.2	91.4	28.3	29.5	33.3	38.4	44.1	51.8
	Line	90.2	90.3	90.3	90.4	90.4	90.7	29.5	31.5	34.6	41.1	47.4	57.1
	Node2Vec	91.2	91.4	91.4	91.5	91.5	91.7	28.6	30.9	34.6	40.0	46.6	55.2
	GraRep	90.4	90.5	90.5	90.5	90.6	90.8	31.9	34.0	37.5	42.5	48.8	57.4
	M-NMF	89.3	89.4	89.5	89.5	89.6	89.8	31.4	33.9	37.3	44.2	48.3	57.3
Micro-F1 (%)	ComE	92.4	92.5	92.6	92.6	92.6	92.8	84.7	85.1	85.4	85.8	86.6	87.7
	DeepWalk/SF	91.4	91.6	91.6	91.6	91.6	91.8	81.2	81.8	82.3	82.6	82.9	83.3
	Line	90.8	90.9	91.0	91.0	91.0	91.4	82.9	84.1	84.5	85.1	85.4	85.8
	Node2Vec	91.7	91.9	91.9	91.9	92.0	92.2	80.9	81.6	81.8	82.2	82.6	83.2
	GraRep	90.9	91.0	91.1	91.1	91.1	91.4	85.1	85.4	85.5	86.1	86.1	87.4
	M-NMF	90.0	90.1	90.2	90.2	90.3	90.5	84.9	85.3	85.8	86.4	86.4	87.8

Mich								Amherst					
	% Labels	40%	50%	60%	70%	80%	90%	40%	50%	60%	70%	80%	90%
Macro-F1 (%)	ComE	28.4	30.6	31.6	33.2	36.9	42.8	53.9	57.5	58.9	61.4	65.7	73.8
	DeepWalk/SF	28.3	30.3	30.6	32.7	33.2	37.8	46.8	50.1	51.8	53.3	57.6	64.1
	Line	28.9	30.1	30.5	33.5	34.1	38.0	48.7	52.4	55.2	55.5	59.5	67.9
	Node2Vec	27.3	29.7	30.7	33.1	34.4	39.1	49.6	52.3	53.5	55.0	57.6	66.4
	GraRep	29.0	30.2	30.9	32.9	35.4	40.9	54.3	55.7	60.3	61.5	62.9	69.9
	M-NMF	29.4	31.1	32.5	33.8	34.3	40.1	50.5	52.5	56.4	57.3	60.0	68.1
Micro-F1 (%)	ComE	60.9	61.7	61.9	62.3	63.2	64.6	90.3	90.4	90.6	90.8	91.2	91.5
	DeepWalk/SF	57.6	59.2	59.7	59.8	60.9	62.5	87.8	87.9	88.2	88.4	88.5	88.5
	Line	58.7	59.9	60.4	61.2	61.5	63.7	88.6	88.6	88.7	88.8	88.9	89.1
	Node2Vec	57.5	58.9	59.7	60.0	61.6	63.2	88.1	88.6	88.9	89.2	89.4	89.9
	GraRep	60.5	60.9	61.4	61.6	63.0	63.5	90.8	90.8	90.9	91.0	91.0	91.5
	M-NMF	60.6	61.2	61.4	61.5	61.6	62.9	89.9	90.3	90.5	90.6	90.8	91.3

ingful neighbors. A counter-proof is also provided by the random walk-based methods which work relatively better than the factorization-based methods on DBLP. This suggests that the node size and the density of a graph can impact the performance of some algorithms because of the length and the window size limitation of the random walk methods.

Second, Wikipedia appears to be the most challenging dataset to solve. This is a consequence of its creation process. Wikipedia is a word co-occurrence graph generated by a dump of the homonym website in which the labels of each node reflect the POS-tag associate to each word in the original text corpus. Given that each word

Table 3.4: ComE p-values w.r.t. the baselines using 80% of the lables.

		BlogCatalog	Wikipedia	DBLP	Rochester	Mich	Amherst
Macro-F1	DeepWalk/SF	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	Line	< 0.01	< 0.01	< 0.01	< 0.05	< 0.01	< 0.01
	Node2Vec	< 0.01	< 0.01	< 0.01	< 0.02	< 0.05	< 0.01
	GraRe	< 0.01	= 0.21	< 0.01	= 0.39	= 0.38	< 0.06
	M-NMF	< 0.01	< 0.01	< 0.01	= 0.29	= 0.12	< 0.01
Micro-F1	DeepWalk/SF	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	Line	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	Node2Vec	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	GraRep	< 0.01	< 0.01	< 0.01	= 0.15	= 0.61	= 0.98
	M-NMF	< 0.01	< 0.01	< 0.01	= 0.67	< 0.03	= 0.57

can have multiple syntactic functions according to the sentence structure, Wikipedia is a multi-label dataset. However, identifying the correct tag without knowing the sentence structure is difficult, which explains the poor general performances obtained over this dataset.

Third, the performance obtained by all the algorithms on the single-label datasets is higher than the performance of a multi-label dataset. On the one hand, Tab. 3.1 and Fig. 3.3 show that BlogCatalog and Wikipedia present multiple hubs and generally a dense structure. Thus, it is easier for the nodes of different categories to be a part of the same neighborhood. As a consequence, it is challenging to differentiate between them only based on the homophily principle. On the other hand, single-label datasets present a more fragile structure, but it is easier to identify well-defined communities inside them. Note that as mentioned previously, M-NMF appears to be a valuable approach over the dataset with a well-defined structure.

Finally, it is interesting to see that ComE improves the baselines on node classification because it is unsupervised and it does not directly optimize the classification loss. This indicates that the high-order proximity from community embedding does contribute to node embedding.

3.4.5 Model Study

In this section, the impact of the main ComE’s hyper-parameters is evaluated. First, the performance of ComE is calculated w.r.t. the dimension of the embedding. Subsequently, the execution time and the convergence of ComE are empirically assessed to validate the inference algorithm proposed in Ch. 3.3. Finally, the impact of α and β is graded because they are the parameters that mostly differentiate ComE from the baselines.

Representation dimension

The graph embedding’s scope is to identify a low-dimension representation of the input graph ($d \ll |V|$). Thus, we evaluate the performance of ComE for different representation size in terms of node classification and node clustering, i.e., Figs. 3.7 and 3.8 report the performance for $d \in [32, 64, 128, 256]$ for all the datasets.

Figure 3.7: Impact of the parameter d on the node classification task.

Figure 3.8: Impact of the parameter d on the clustering task.

On the one hand, Fig. 3.7 shows that the classification performances are more subject to the value of d . As expected, almost in every experiment, we achieve a performance increase if we augment the representation size. It is worth mentioning that the performances do not increase linearly with the dimensionality; however, it is possible usually to detect a monotonic increase in the node classification achievement. Mich and Amherst report the best performance when $d = 128$ rather than $d = 256$. This validates the general embedding hypothesis, which assumes that a dense vector space is able to preserve structural information better.

On the other hand, Fig. 3.8 shows that clustering algorithms are subject to overfitting if the embedding dimension exceeds a specific value. This could be related to the GMM algorithm, which is not able to leverage the complex graph structure preserved in a bigger embedding, but it performs better on a compact representation. However, as for other algorithms, $d = 128$ seems a reasonable setting for ComE.

Convergency and Efficiency

The convergence and efficiency of ComE are validated in Fig. 3.9. The loss function (Eq. 3.7) at the end of every iteration (line 18 in Alg. 1) is recorded to show the convergence of the proposed algorithm. As shown in Fig. 3.9a, the loss of ComE consistently converges quickly within 2–3 iterations. This validates the robustness of Alg. 1 w.r.t. different datasets.

Figure 3.9: Model’s convergence and efficiency. Note that the results are done for $d = 2$.

To demonstrate the efficiency of ComE, we tested it on six datasets at different scales. For each dataset, we generated three subsets in which we maintained 25%, 50%, and 75% of the total number of edges and nodes, respectively. Note that to speed up the computation time of those experiments, we set $d = 2$ and $\zeta = 5$. The diagram in Fig. 3.9b shows the processing time of ComE (line 2–line 19 in Alg. 1) in different datasets. Clearly, the processing time of ComE is linear to the graph size (i.e., $|V|$ and $|E|$). This empirically validates the complexity analysis reported at the end of Ch. 3.3.

Impact of α and β

Fig. 3.10 reports the impact of the model parameters α (trade-off parameter for context embedding) and β (trade-off parameter for community embedding). In each figure, we tune one parameter within the range of $[0.001, 1]$ while fixing the other parameter as 0.1. In general, the model performance is quite robust when α and β are within the studied interval range. In particular, $\alpha = 0.1$ gives the best trade-off for the second-order proximity in the objective function as it provides the best results in terms of Conductance for all the experiment except for Amherst. The tuning of β is not as sensitive and easy as α . In terms of node classification task, $\beta = 0.01$ and $\beta = 1$ are valid candidates for node classification. For example, on Wikipedia, $\beta = 0.001$ produces a relative improvement of 1.9% in terms of Macro-F1 while $\beta = 0.01$ is the best setting in terms of Conductance on BlogCatalog. This could be related to the multiple membership problems related to the multi-label dataset, in which a softer clustering effect is less prone to the noise in the communities’ structure. Similarly, on Rochester, $\beta = 0.001$ and 0.01 are valid candidates in terms of Macro-F1 and NMI, where single-node communities can affect the generated embedding.

Figure 3.10: Impact of the parameters α and β .

3.5 Conclusion

In this chapter, we studied the important, yet largely under-explored problem of embedding communities on graphs. We developed a closed loop among community embedding, community detection, and node embedding. Therefore, we tried to jointly optimize all these three tasks to allow them to reinforce each other.

A scalable inference algorithm, which only requires a complexity of $O(|V| + |E|)$, has been proposed (Alg. 1). We evaluated ComE on multiple real-world datasets and showed that it outperforms the state-of-the-art baselines by at least 1.8%–6.7% (NMI) and 0.3%–17.1% (Conductance) in community detection and 0.8%–20.1% (Macro-F1) and 0.7%–30.4% (Micro-F1) in node classification. It also improved the baselines in the task of graph visualization.

Chapter 4

ComE+

4.1 Introduction

In Ch. 3, we have formulated the problem of generating a community-aware higher-order proximity node embedding, combining the multivariate Gaussian distributions with simple neural networks architectures. Empirically, such an embedding method has been proven to be capable of improving node as well as community-related tasks, corroborating the importance of preserving the mesoscopic structure present in a graph. Because of the closed loop formulation (Fig. 3.2), the representation of the communities obtained is both congruent with the node embedding (they are in the same low-dimensional space) and is also able to better solve the multi-membership problem because of its probabilistic formulation. Moreover, the efficient inference algorithm proposed in Ch. 3.3 ensures that ComE can scale to large problem settings.

To achieve a rich embedding, ComE requires a proper community assignment. Most of the clustering algorithms, such as the GMM or spectral clustering, require knowing the exact number of communities present in the dataset to infer meaningful labels. However, in most real-world applications, such prior information is unavailable. A recent study [109] also emphasized the discrepancy between the meta-data commonly used as ground truth for the community assignment and the real generative process of the network. For example, it could be possible that a graph is too sparse to contain any community structure or the community detected is capturing different aspects w.r.t. the meta-data. On the one hand, on artificially generated networks, it is possible to define a ground truth communities' assignment according to the link generative process. On the other hand, on real-world data, neither the ground truth nor its link generative process is known. Therefore, it is not possible to reach an optimum consensus on the network partition and consequently on the number of existing clusters. Based on this argument, we propose an entirely data-driven approach for community detection in which both the embedding and the number of clusters are derived from the actual data distribution.

For instance, in Fig. 4.1a, the Karate Club network has been reported. Note that the original study [157] partitioned the graph into two factions according to the administrative evolution of the club. Other studies suggested that this is not the only scientifically reasonable way to partition the network; e.g., [40, 49, 110] proposed partitioning the graph into more than two clusters. Similarly, it is possible

Figure 4.1: Comparison between the ComE and ComE+ nodes and communities embedding in a 2D space.

to generalize this result on a more complex network, assuming that there could be more than a single legitimate clustering result. On the one hand, based on the same study performed in Ch. 3.4.5 for the parameters α and β , it is possible to manually tune the number of communities (K). However, such an approach is tedious or impractical for large-scale graphs. On the other hand, in this chapter, we propose a Bayesian inference algorithm that is able to automatically detect the number of communities from the data and certain carefully designed prior distributions.

We extend the ComE model to **ComE+**, an algorithm for community embedding with an unknown number of communities (i.e., K is unknown and can be infinite), which is a typical scenario in practice. For ComE+, we aim to model the closed loop in Fig. 3.2, but with special consideration of modeling an infinite number of communities in detection and embedding. To realize this infinite community embedding, we were inspired by the infinite Gaussian mixture (IGM) [114] and adopted a Bayesian inference framework. We first considered a simple stick-breaking construction approach [14] to generate up to an infinite number of communities and accordingly sample each node’s community assignment. Then, we sampled the node embedding from the community embedding, and later optimized it w.r.t. the first- and second-order proximity. We then developed an efficient iterative inference algorithm for ComE+. Given a finite dataset, our inference algorithm can let the data automatically determine the exact number of communities for embedding. In Fig. 4.1, we compared ComE w.r.t. ComE+, showing that ComE+ is able to correctly infer $K = 2$ from the data distribution and it embeds the two overlapping communities in the 2D space.

We summarize our contributions as follows:

- We extend ComE to ComE+ to handle an unknown number of communities for community embedding. As far as we know, this is the first node embedding model that is able to automatically infer the number of communities from the data distribution while preserving such a structure during the embedding process.
- We designed a new variational inference model for ComE+ while maintaining the inference algorithm complexity to be as low as $O(|V| + |E|)$.
- We evaluated ComE+ w.r.t. ComE and other baselines on seven public real-world datasets with various application tasks. We relatively improved the baselines across

Figure 4.2: Graphical model for ComE+. The solid arrows indicate a generative process. The dotted arrows indicate a neural network modeling for node embedding.

datasets by at least 0.8%–5.2% (Conductance) and 0.6%–4.7% (NMI) in community detection, 0.8%–25.4% (Macro-F1), and 0.3%–34.7% (Micro-F1) in node classification. The detailed settings for these improvements are reported in Ch. 4.4.

4.2 Problem Formulation

Based on the same setting of ComE (Ch. 3.2), we were given a graph $G = (V, E)$ as input. The goal of ComE+ is to jointly learn node embedding and community embedding, assuming there are K communities on the graph G , where K can be either known (i.e., finite) or unknown (i.e., infinite). For each node v_i , we denote its community assignment as $z_i \in \{1, \dots, K\}$. We define community embedding as a multivariate Gaussian distribution, describing how nodes spread in the low-dimensional space.

Similar to ComE, as *output* we aim to learn the following:

1. node embedding ϕ_i for each node $v_i \in V$;
2. community membership π_{ik} such that $\sum_{k=1}^K \pi_{ik} = 1$ for each node $v_i \in V$ and each community $k \in \{1, \dots, K\}$;
3. community embedding parameters (ψ_k, Σ_k) for each community $k \in \{1, \dots, K\}$.

To handle the unknown number of communities, we resort to the infinite generative approach for community detection and community embedding. We summarize our graphical model for ComE+ in Fig. 4.2. Similar to ComE, to learn the node embedding ϕ_i 's, we considered both first- and second-order proximity. For the first-order proximity, we introduced a dotted arrow from a node v_i 's embedding ϕ_i to its direct neighbors' embedding ϕ_l , which indicates that neighbor nodes have a similar embedding. For the second-order proximity, we introduced a dotted arrow from ϕ_i to its context nodes' embedding ϕ'_j , which indicates two nodes with similar context have similar embedding. For each node v_i , there are in total $|N_i|$ direct neighbors and $|C_i|$ context nodes. Furthermore, we leverage the community embedding to provide higher-order proximity. In particular, we introduced solid arrows from ψ_k and Σ_k to ϕ_i , where $k = z_i$ is node v_i 's community assignment. These solid arrows enforce that the nodes in the same community k are closer to the community center

- Draw $\pi_k \sim \mathcal{B}(1, \rho)$, for $k = 1, 2, \dots$;
- Draw $\psi_k \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, for $k = 1, 2, \dots$;
- Draw $\Sigma_k^{-1} \sim \mathcal{W}(\mathbf{I}, \nu)$, for $k = 1, 2, \dots$;
- For each node v_i :
  - Draw $z_i \sim SBC(\boldsymbol{\pi})$;
  - Draw $\phi_i \sim \mathcal{N}(\psi_{z_i}, \Sigma_{z_i})$.

Figure 4.3: Generative process of ComE+.

(i.e., ψ_k). Finally, we learn the Bayesian infinite community embedding (ψ_k, Σ_k) 's by inferring their posterior distributions based on the node embedding ϕ_i 's and their prior distributions' hyperparameters (depicted by the solid arrows from ρ and ν). Because of this infinite generative process, ComE+ assumes that there could be up to an infinite number of communities in the graph G and the data distribution will automatically select the correct value of K . However, the node embedding formulation is equal to the one defined in Ch. 3.2.2; thus, we are not going to rewrite it.

4.2.1 Community Detection and Embedding

Following Ch. 3.2.1, we see each node embedding to be generated by a multivariate Gaussian distribution $\mathcal{N}(\psi_k, \Sigma_k)$ from a community $z_i = k$. However, in ComE+, we are inspired by IGM [114] to sample the node embedding from an infinite number of (ψ_k, Σ_k) 's. This indicates that we have both ψ_k and Σ_k governed by some prior distributions. Note that we are technically different from IGM in two aspects. First, rather than trying to derive the posterior under an infinity limit, we take a more straightforward *stick-breaking construction* approach [14] to generate an infinite number of communities and sample each node's corresponding community assignment. Second, rather than using MCMC for inference, we take a *variational inference* approach (which is more suitable for our joint modeling, as discussed in Ch. 4.2.2) to approximate the posterior for the community embedding random variables. Next, we summarize the generative process for community detection and embedding.

Generative process. We summarize the generative process of ComE+ in Fig. 4.3. In particular, we denote $\mathcal{B}(a, b)$ as a beta distribution, where $a > 0$ and $b > 0$ are the parameters. We denote $\mathcal{W}(A, \nu)$ as a Wishart distribution with a positive definite scale matrix $A \in \mathbb{R}^{d \times d}$ and degrees of freedom $\nu > d - 1$. Denote $\mathbf{I}$ as a $d \times d$ dimensional identity matrix. We denote $\rho > 0$ and $\nu > 0$ as hyperparameters for the prior distributions of ψ_k and Σ_k , respectively, describing how many latent components to activate and how small the Gaussian covariance can be. Furthermore, $z_i \sim SBC(\boldsymbol{\pi})$ is a stick-breaking construction, where given an infinite dimensional vector $\boldsymbol{\pi} = [\pi_1, \pi_2, \dots]$, we compute the mixture probability for each community k

as follows:

$$\eta_k = \pi_k \prod_{j=1}^{k-1} (1 - \pi_j), \quad \text{for } k = 1, 2, \dots \quad (4.1)$$

Based on the resulting infinite dimensional vector $\boldsymbol{\eta} = [\eta_1, \eta_2, \dots]$, we draw the community assignment indicator $z_i \in \{1, 2, \dots\}$ for each node v_i using a multinomial distribution:

$$z_i \sim \text{Multi}(\boldsymbol{\eta}), \quad \text{for } i = 1, \dots, |V| \quad (4.2)$$

For notation simplicity, we denote $Z = \{z_1, \dots, z_N\}$ as the community assignments for N nodes and $\Omega = \{(\boldsymbol{\psi}_1, \Sigma_1), (\boldsymbol{\psi}_2, \Sigma_2), \dots\}$ as the infinite community embeddings. Finally, we denote $W = \{\boldsymbol{\pi}, \Omega, Z\}$. Given node embeddings $\Phi = [\boldsymbol{\phi}_1, \dots, \boldsymbol{\phi}_N]$ for graph G , our goal is to infer the posterior distribution $p(W|\Phi; \rho, \nu)$, which identifies the community assignments and community embeddings.

4.2.2 Closing the Loop

In ComE+, we aim to realize the same closed loop defined in Ch. 3.2.3 but in an infinite community setting. For node embedding, we follow the same modeling as introduced in Ch. 3.2.2. For infinite community detection and embedding, as well as for enabling their feedback to node embedding, we use the generative process in Fig. 4.3. Next, we introduced how to combine node embedding with infinite community detection and embedding.

Effectively, learning node embedding requires a maximum a posteriori (MAP) estimation, where we see O_1 (Eq. 3.3) and O_2 (Eq. 3.5) as modeling the likelihood of $\boldsymbol{\phi}_i$'s, and $\boldsymbol{\phi}_i \sim \mathcal{N}(\boldsymbol{\psi}_{z_i}, \Sigma_{z_i})$ as modeling the prior. In contrast, to handle the infinite number of communities, we required Bayesian inference. To combine node embedding with infinite community detection and embedding, we were required to manage both MAP estimation and Bayesian inference. In general, there can be different approaches for Bayesian inference. For example, [59, 128, 156] adopted a MCMC sampling method, whereas [151] optimized the likelihood function using *stochastic gradient descent* (SGD). In this study, we proposed to use a variational inference formulation as the objective function of ComE+. However, we resorted to variational inference to develop an analytical form, which directly infers the posterior distribution of the Gaussian random variables and encodes the node embedding prior using $\mathcal{N}(\boldsymbol{\psi}_{z_i}, \Sigma_{z_i})$.

In particular, we introduced a *variational distribution* $q(W)$ to approximate the posterior $p(W|\Phi; \rho, \nu)$ by minimizing the KL-divergence between them:

$$\begin{aligned} KL(q(W) \parallel p(W|\Phi; \rho, \nu)) &\stackrel{1}{=} \mathbb{E}_q \left[\log \frac{q(W)}{p(W|\Phi; \rho, \nu)} \right] \\ &= \mathbb{E}_q[\log q(W)] - \mathbb{E}_q[\log p(W|\Phi; \rho, \nu)] \\ &\stackrel{2}{=} \mathbb{E}_q[\log q(W)] - \mathbb{E}_q[\log p(W, \Phi|\rho, \nu)] + \log p(\Phi|\rho, \nu), \end{aligned} \quad (4.3)$$

where at step 1, we use the definition of KL-divergence; at step 2, we use

$$p(W|\Phi; \rho, \nu) = \frac{p(W, \Phi|\rho, \nu)}{p(\Phi|\rho, \nu)}$$

As $\log p(\Phi|\rho, \nu)$ does not depend on q , minimizing Eq. 4.3 is equivalent to the following:

$$\max_q \mathbb{E}_q[\log p(W, \Phi|\rho, \nu)] - \mathbb{E}_q[\log q(W)]. \quad (4.4)$$

Notice that $p(W, \Phi|\rho, \nu)$ is the joint probability describing our generative process in Ch. 4.2.1. Thus, it naturally includes the formulation of $\mathcal{N}(\boldsymbol{\psi}_{z_i}, \Sigma_{z_i})$, which can be used to optimize together with O_1 and O_2 . To this end, we define the objective function for inferring the community embedding posterior distribution, which also enables the higher-order proximity for node embedding, as:

$$O_3^{ComE+} = -\beta (\mathbb{E}_q[\log p(W, \Phi|\rho, \nu)] - \mathbb{E}_q[\log q(W)]) \quad (4.5)$$

where $\beta > 0$ is a trade-off parameter. Finally, we define the overall objective function for ComE+ as follows:

$$\begin{aligned} \mathcal{L}^{ComE+}(\Phi, \Phi', q) = & O_1(\Phi) \\ & + O_2(\Phi, \Phi') \\ & + O_3^{ComE+}(\Phi, q) \end{aligned} \quad (4.6)$$

4.3 Bayesian Inference

To optimize the ComE+ objective in Eq. 4.6, we also applied iterative optimization between (Φ, Φ') and q . Given (Φ, Φ') , q is optimized to infer the infinite community embedding. Given q , (Φ, Φ') is optimized to learn the node embedding with all types of proximity. Empirically, this iterative optimization algorithm converges quickly, and we report the convergence in Ch. 4.4.7.

Fix (Φ, Φ') , optimize q . For variational inference, we attempted to define $q(W)$ using a tractable form. Note that $q(W)$ is a distribution over an infinite set of $(\pi_k, z_k, \boldsymbol{\psi}_k, \Sigma_k)$'s. To make $q(W)$ tractable, we follow [14] to truncate $q(W)$ at a value K by setting $q(\pi_K = 1) = 1$. According to Eq. 4.1, we have all $\eta_k(\boldsymbol{\pi}) = 0$ for $k > K$. Because of this truncation, we revise $q(W)$ as $q(W, K)$, where K is a variational parameter. We will empirically evaluate the model performance under different K values in the experiments. Note that the number of represented communities can be smaller than K , and it is to be automatically determined by the data. In other words, K only serves as a maximum number of communities to be detected for the graph. Then, we use *mean-field approximation* to factorize $q(W, K)$ as follows:

$$q(W, K) = \prod_{k=1}^K q_{\gamma_{k,1}, \gamma_{k,2}}(\pi_k) \prod_{k=1}^K q_{\boldsymbol{\tau}_k}(\boldsymbol{\psi}_k) \prod_{k=1}^K q_{B_k, c_k}(\Sigma_k^{-1}) \prod_{i=1}^{|V|} q_{\boldsymbol{\xi}_i}(z_i) \quad (4.7)$$

where we define $q_{\gamma_{k,1}, \gamma_{k,2}}(\pi_k)$ as a Beta distribution parameterized with $\gamma_{k,1} > 0$; $\gamma_{k,2} > 0$, $q_{\boldsymbol{\tau}_k}(\boldsymbol{\psi}_k)$ as a multivariate Gaussian distribution parameterized with mean $\boldsymbol{\tau}_k \in \mathbb{R}^d$; $q_{B_k, c_k}(\Sigma_k^{-1})$ as a Wishart distribution parameterized with a positive definite scale matrix $B_k^{-1} \in \mathbb{R}^{d \times d}$ and degrees of freedom $c_k > 0$; and $q_{\boldsymbol{\xi}_i}(z_i)$ as a multinomial distribution parameterized with $\boldsymbol{\xi}_i \in \Delta$ (where Δ is a simplex). Formally, we have for $q(W, K)$:

- $q_{\gamma_{k,1}, \gamma_{k,2}}(\pi_k) : \pi_k \sim \mathcal{B}(\gamma_{k,1}, \gamma_{k,2})$, for $k = 1, \dots, K$;
- $q_{\tau_k}(\psi_k) : \psi_k \sim \mathcal{N}(\tau_k, \mathbf{I})$, for $k = 1, \dots, K$;
- $q_{B_k, c_k}(\Sigma_k^{-1}) : \Sigma_k^{-1} \sim \mathcal{W}(B_k^{-1}, c_k)$, for $k = 1, \dots, K$;
- $q_{\xi_i}(z_i) : z_i \sim \text{Multi}(\xi_i)$, for $i = 1, \dots, |V|$.

With the truncated variational distribution $q(W, K)$, we revise Eq. 4.5 as follows:

$$\begin{aligned}
O_3^{ComE+'}(\Phi, q) &= -\frac{\beta}{K} (\mathbb{E}_q[\log p(W, \Phi | \rho, \nu)] - \mathbb{E}_q[\log q(W, K)]) \\
&= -\frac{\beta}{K} \left(\sum_{k=1}^K (\mathbb{E}_q[\log p(\pi_k | \rho)] - \mathbb{E}_q[\log q_{\gamma_{k,1}, \gamma_{k,2}}(\pi_k)]) \right. \\
&\quad + \sum_{k=1}^K (\mathbb{E}_q[\log p(\psi_k)] - \mathbb{E}_q[\log q_{\tau_k}(\psi_k)]) \\
&\quad + \sum_{k=1}^K (\mathbb{E}_q[\log p(\Sigma_k^{-1} | \nu)] - \mathbb{E}_q[\log q_{B_k, c_k}(\Sigma_k^{-1})]) \\
&\quad + \sum_{i=1}^{|V|} (\mathbb{E}_q[\log p(z_i | \boldsymbol{\pi})] - \mathbb{E}_q[\log q_{\xi_i}(z_i)]) \\
&\quad \left. + \sum_{i=1}^{|V|} \mathbb{E}_q[\log p(\phi_i | \psi_{z_i}, \Sigma_{z_i})] \right) \tag{4.8}
\end{aligned}$$

Ultimately, we replace O_3^{ComE+} in Eq. 4.6 with $O_3^{ComE+'}$ for inference. Due to the space limit, we skip the derivation details and summarize the updates for each variational parameter in $q(W, K)$ as follows:

$$\gamma_{k,1} = 1 + \sum_{i=1}^{|V|} \xi_{i,k}, \tag{4.9}$$

$$\gamma_{k,2} = \rho + \sum_{i=1}^{|V|} \sum_{j=k+1}^K \xi_{i,j} \tag{4.10}$$

$$\tau_k = \left(\mathbf{I} + B_k^{-1} c_k \sum_{i=1}^{|V|} \xi_{i,k} \right)^{-1} \left(B_k^{-1} c_k \sum_{i=1}^{|V|} \xi_{i,k} \phi_i \right) \tag{4.11}$$

$$B_k = \left(\sum_{i=1}^{|V|} \xi_{i,k} + 1 \right) \mathbf{I} + \sum_{i=1}^{|V|} \xi_{i,k} (\phi_i - \tau_k)(\phi_i - \tau_k)^T \tag{4.12}$$

$$c_k = 2 + \nu + \sum_{i=1}^{|V|} \xi_{i,k} \tag{4.13}$$

$$\xi_{i,k} \propto e^{\Lambda(\gamma_{k,1}, \gamma_{k,2}) + \sum_{j=1}^k \Lambda(\gamma_{j,1}, \gamma_{j,2}) + \mathbb{E}_q[\log p(\phi_i | z_i = k)]} \tag{4.14}$$

where we define $\Lambda(\gamma_{l,1}, \gamma_{l,2}) = \Psi(\gamma_{l,1}) - \Psi(\gamma_{l,1} + \gamma_{l,2})$, given $\Psi(\cdot)$ as a digamma function.

Fix q , optimize (Φ, Φ') . We used MAP estimation to learn the node embedding with O_1 and O_2 as the likelihood terms and $O_3^{ComE+'}$ as the prior term for ϕ_i 's. Optimizing O_1 and O_2 is straightforward and is similar to ComE's inference in Ch. 3.3. Hence, we focused on $O_3^{ComE+'}$ next. In Eq. 4.8, only the last term depends on ϕ_i 's. Then, given the community embedding, we can simplify $O_3^{ComE+'}$ as follows:

$$\begin{aligned}
O_3^{ComE+''}(\Phi) &= -\frac{\beta}{K} \sum_{i=1}^{|V|} \mathbb{E}_q[\log p(\phi_i | \psi_{z_i}, \Sigma_{z_i})] \\
&\propto \frac{\beta}{K} \sum_{i=1}^{|V|} \sum_{k=1}^K \xi_{i,k} \cdot \frac{1}{2} (\phi_i - \tau_k)^T (B_k^{-1} c_k) (\phi_i - \tau_k)
\end{aligned} \tag{4.15}$$

As a result, we minimize:

$$\mathcal{L}^{ComE+''}(\Phi, \Phi') = O_1(\Phi) + O_2(\Phi, \Phi') + O_3^{ComE+''}. \tag{4.16}$$

The gradients of O_1 and O_2 over ϕ_i are given in Eq. and 3.16. The gradient of $O_3^{ComE+''}$ over ϕ_i is as follows:

$$\frac{\partial O_3^{ComE+''}}{\partial \phi_i} = \frac{\beta}{K} \sum_{k=1}^K \xi_{i,k} (B_k^{-1} c_k) (\phi_i - \tau_k). \quad (4.17)$$

Then, we have the total gradient for each ϕ_i as follows:

$$\frac{\partial \mathcal{L}'}{\partial \phi_i} = \frac{\partial O_1}{\partial \phi_i} + \frac{\partial O_2}{\partial \phi_i} + \frac{\partial O_3^{ComE+''}}{\partial \phi_i} \quad (4.18)$$

while the gradient w.r.t. the context embedding is as follows:¹:

$$\frac{\partial \mathcal{L}'}{\partial \phi'_j} = \frac{\partial O_2}{\partial \phi'_j} \quad (4.19)$$

4.3.1 Algorithm and Complexity

ALGORITHM 2: Inference algorithm for ComE+

Input: graph $G = (V, E)$, variational parameter k , #(paths per node) γ , walk length ℓ , context size ζ , embedding dimension d , negative context size m , parameters (α, β) .

Output: node embedding Φ , context embedding Φ' , community embedding posterior distribution q

```

1  $\mathcal{P} \leftarrow \text{SamplePath}(G, \ell);$ 
2 Initialize  $\Phi$  and  $\Phi'$  by DeepWalk [110] with  $\mathcal{P}$ ;
3 for  $iter = 1 : T_1$  do
4   for  $subiter = 1 : T_2$  do
5     Optimize  $q(W, K)$  by Eq. 4.9–4.14 given  $\Phi$ ;
6     foreach  $edge (i, j) \in E$  do
7       SGD on  $\phi_i$  and  $\phi_j$  by Eq. 4.3;
8     foreach  $node v_i \in V$  do
9       SGD on  $\phi_i$  by Eq. 4.15;
10    foreach  $path p \in \mathcal{P}$  do
11      foreach  $v_i$  in  $path p$  do
12        SGD on  $\phi_i$  by Eq. 3.16;
13        SGD on  $\phi'_j$ 's within  $\zeta$  hops by Eq. 3.18;
```

We summarize the inference algorithm of ComE+ in Alg. 2. In line 1, for each $v_i \in V$, we sampled γ paths starting from v_i with length ℓ on G . In line 2, we initialized (Φ, Φ') using DeepWalk. In lines 4–5, we fixed (Φ, Φ') and optimized $q(W, K)$ for community detection and embedding. Note that we did not have to check the constraint of $diag(\Sigma_k) > 0$ since they are easily satisfied by the sampling procedure of the variational approach that was proposed. In lines 6–11, we fixed $q(W, K)$ and optimized (Φ, Φ') for node embedding. Moreover, we analyzed the complexity of Alg. 2. In line 1, path sampling takes $O(|V|\gamma\ell)$; in line 2, parameter initialization using DeepWalk takes $O(|V|)$; and in line 5, optimizing the variational parameters $q(W, K)$ takes $O(|V|K)$. Similarly, in line 7, node embedding w.r.t. first-order proximity takes $O(|E|)$; in lines 9–11, node embedding w.r.t. second-order

---

¹Already given in Eq. 3.18

proximity takes $O(|V|\gamma\ell)$; and in lines 12–13, node embedding w.r.t. community-aware high-order proximity takes $O(|V|K)$. Overall, the complexity of Alg. 2 is $O(|V|\gamma\ell + |V| + T_1 \times (T_2|V|K + K + |E| + |V|\gamma\ell + |V|K))$, which is still linear to the graph size (i.e., $|V|$ and $|E|$).

4.4 Experiments

In this section, we extensively evaluate ComE+ on the same tasks used in Ch. 3.4; however, three more datasets have been added to obtain a comprehensive evaluation. Note that the same metric that was used to evaluate ComE is used in this section to obtain a fair comparison. Moreover, to better evaluate the difference between ComE and ComE+, a sensitivity analysis is conducted on the major parameters used by the algorithms in Ch. 4.4.7.

4.4.1 Datasets

For consistency, we used the same datasets as described in Ch. 3.4.1 to compare ComE+. Such a procedure guarantees a better comparison between all the tested methods.

4.4.2 Baselines

In addition to the baselines described in Ch. 3.4.1, we added PRUNE [77] to compare ComE+ w.r.t. all the state-of-the-art embedding methods for homogeneous graphs. PRUNE models first-order proximity and estimates the global ranking of each node using Siamese neural networks. Indirectly, the proposed architecture is argued to preserve the second-order as well as the communities' structure via the modularity matrix; thus, it allows us to compare the explicit formulation that was proposed with a more implicit one.

4.4.3 Parameters and Environment

For a better comparison, the test performed on ComE+ follows the same general parameter setting used in Ch. 3.4.1. All experiments, if not otherwise specified, were performed using an embedding dimension of $d = 128$. Moreover, ComE+ used the same setting of γ , ζ , ℓ , and m as specified for ComE.

- **PRUNE:** With respect to the others methods, PRUNE uses as input only the edges of a network. To make the comparison fairer, we set the number of training epochs in such a way that the training example received is comparable w.r.t. all the other models. Moreover, as suggested in the original study, we used Adam optimization with a learning rate $lr = 0.001$ and we set $\alpha = 5$ and $\lambda = 0.01$ for all the experiments.

- **ComE+:** As for ComE, ComE+ has two main parameters α and β . A comprehensive evaluation of such parameters is performed in Ch. 4.4.7; however, if not

Figure 4.4: Graph visualization on Karate Club.

otherwise stated, we used $\alpha = 0.1$ and $\beta = 0.1$ for BlogCatalog, Wikipedia, and Karate Club, whereas for the remaining datasets, $\alpha = 0.1$ and $\beta = 1$ gave the best results. Note that for all the datasets, we set K as the number of unique labels for ComE. Instead, on ComE+ experiments, we used $K = 50$ for all but the DBLP dataset, in which 20 is used as an upper bound for the number of communities.

4.4.4 Graph Visualization

As for ComE, to obtain a first qualitative result of how our models are performing w.r.t. the baselines, we performed graph visualization of the Karate Club graph and the DBLP citation graph. Note that Fig. 4.4 and 4.5 report only the ComE, ComE+, and PRUNE results, while the other baselines are shown only in Ch. 3.4.2.

Karate Club results. PRUNE and LINE (Fig. 3.4b) do not present any proper community structure. While LINE, using a concatenation approach, which produces an embedding that is difficult to project consistently in 2D, PRUNE is affected by only preservice, i.e., the limited first-order proximity. In contrast, ComE can identify such weak supporters because of its joint modeling of overlapping community detection, community embedding, and node embedding. Compared to the baseline, ComE+ is also able to detect the weak supporters. However, it presents a “soften” clustering effect compared to ComE. We believe that this is related to the sampling procedure of the membership function Eq. 4.14.

DBLP results. We visualize the **DBLP** paper citation graph in Fig. 4.5 using the t-SNE visualization toolkit [134] and by adopting different colors to identify different communities. As for the results reported in Fig. 3.5 ComE+ is not able to properly separate the green and the yellow classes (Data Mining and Database), thus enforcing our assumption that these two classes present a high overlapping and that additional knowledge is required to achieve more accurate results. Similar to M-NMF, PRUNE uses a derivate of the modularity matrix to preserve the community structure implicitly. However, it is probably affected by the overlapping between the communities; thus, it is not able to correctly learn a network structure. Moreover, the few edges present in the networks limit its ability to indirectly preserve the second-order of proximity. Compared with all baselines, the proposed ComE and

Figure 4.5: Graph visualization on the DBLP graph (better viewed in color). Different node colors indicate different communities; red is *NLP*, blue is *Computer Vision*, green is *Data Mining*, yellow is *Database*, and black is *Networking*

ComE+ methods can detect three classes because of the community-aware higher order proximity; however, they are not able to differentiate between the Database and Data Mining studies.

4.4.5 Community Detection

In community detection, we are looking to predict the most likely community assignment for each node. Based on the same setting that was used for the experiments in Ch. 3.4.3, first, we learn the node embedding for the entire graph; then, we apply a clustering algorithm to derive the associated communities for each node. As for chapter 3, because there is no consensus in the research community on how to compute the NMI on *multi-label* datasets, for BlogCatalog and Wikipedia, only the Conductance w.r.t. the first label has been reported. Please note that we cannot compare the ComE+’s results with those reported in Ch. 3.4.3, because in Fig. 3.6, GMM is used to obtain the communities labels, while we assumed to know the number of clusters present in the dataset. Instead, in those experiments, we use IGM and only an upper bound for K . Thus, it would be unfair to compare ComE and ComE+ for community detection because ComE+ requires lesser hypotheses. Similarly, we do not compare the straightforward (SF) approach described in Ch. 3.4.1, since spectral clustering requires to know the number of partition present in the dataset. For all the ComE+ experiments, we set $\alpha = 0.1$ and $\beta = 0.1$ for BlogCatalog and Wikipedia, respectively, while $\alpha = 0.1$ and $\beta = 1$ are used for the remaining datasets. The maximum number of communities K is set as 50 for all datasets except for DBLP, which uses $k = 20$.

The results reported in Fig. 4.6 demonstrate that ComE+ can outperform all the baselines regarding Conductance by 0.75% to 5.17%. In terms of NMI, ComE+ can outperform all the baselines on all the datasets except for Amherst, where GraRep can obtain significant improvements (relative -3.7%).

On the one hand, on Amherst (and also on Rochester) all the random walk-based models report poor results due to the limited amount of sampled paths (see Ch. 3.4.4 and 3.4.3). Such limited information combined with the ComE+ tendency of producing smaller clusters composed of homogeneous nodes makes it challenging for the IGM to infer the correct number of clusters; thus, it affects the NMI perfor-

mance. Instead, Tab. 4.1 shows that the additional supervision provided during the training stage of the node classification task can easily overcome such a limitation. Rochester and Mich are characterized by single-node communities which, by definition, premise a noise community structure. Nevertheless, ComE+, because of its Bayesian approach, can overcome such a limitation in the Mich dataset. However, it is comparable to Node2Vec on Rochester (according to the NMI score). This is related to the guided sampling process used by Node2Vec, which can better explore and preserve the structural information in the random walks.

Overall, it is noticeable that the NMI improvements are relatively smaller than the one obtained for Conductance. According to Fig. 3.5 and 4.5, this suggests that ComE+ tends to create multiple sub-clusters for the same community. Hence, it can create small homogeneous clusters (i.e., for Conductance), but it becomes less accurate when inferring the value of K (i.e., for NMI).

Finally, we notice that even for these experiments it is not straightforward to fine an optimal approach since both Node2Vec and GraRep show some pros. and cons. However, the robust performance of ComE+ suggests that a Bayesian approach for community embedding can better handle the noise structures found in a graph and produce a richer node embedding.

Figure 4.6: ComE+ community detection results. The smaller Conductance is, the better. The bigger NMI is, the better.

4.4.6 Node Classification

As a supervised task, we evaluated ComE and ComE+ on the node classification task, in which we aim to categorize each node into one or more classes, depending on whether it is a single-label or multi-label setting. We followed the same methodology used in Ch. 3.4.4 and in [110]; first, a graph embedding was generated for the whole graph, and then 40% to 90% (in the interval of 10%) of nodes were randomly split as different training/testing sets. The randomly generated split is used as cross-validation dataset to train an SVM classifier [27] for evaluation. We reported the results with the SVM classifier parameter $c = 1$, which was suggested by [110] for all the methods. The remaining nodes were used for testing.

Remark: Tab. 4.1 reports the average of 10 experiments executed with different random sampling, while 4.2 report the p-values for ComE+ experiments. Note that we use the same parameter setting that was used for the clustering experiments (Ch. 4.4.5).

From Tab. 4.1, it is possible to make the following observations. First, our ComE and ComE+ are generally better than all the baselines in terms of both Macro-F1 and Micro-F1. In particular, when using 80% of the data for training, ComE and ComE+ can improve the baselines by 0.8%–25.4% on Macro-F1 and 0.3%–34.7% on Micro-F1.

Second, ComE and ComE+ have comparable performance on DBLP. On the one hand, in the multi-label datasets, ComE+ is able to achieve better performance, indicating that it can solve the multiple membership problems more appropriately or detect a more meaningful community structure [109] because of its variational inference algorithm. Similarly, on Rochester and Mich, ComE+ is able to outperform ComE because it can handle single-node communities better. On the other hand, ComE remains the best performer on DBLP, which is the dataset that presents the best separation among the different communities.

Third, on Amherst and Rochester, all the random walk-based methods present poor performance w.r.t. the matrix factorization ones. As mentioned above, this is related to the smaller number of sampled paths. Still, it is possible to notice that ComE and ComE+ can partially overcome this limitation because of their ability to leverage community information. Yet, ComE+ can outperform ComE since is able to better handle the uncertainty related to the community structure detected. Note that SVM can leverage the small homogeneous clusters generated by ComE+ and the additional label information provided at training time to obtain outstanding results. In community detection, such additional information was not provided; thus, it is difficult for IGM to detect the correct number of labels present in Amherst.

Fourth, as for the results in Ch. 3.4.4, GraRep appears to be the best baseline for the node classification task. This suggests that the higher-order transition probability matrix contains valuable information, but it comes at the price of overall higher complexity ($O(|V|^3)$). ComE+ overcome such limitation leveraging communities information.

Fifth, w.r.t. ComE, ComE+ shows better performances on the multi-label datasets and on the dataset with single-node communities. This proves the validity of the proposed variational inference algorithm, since it is able to better handle the uncertainty related to the community structure. Moreover, from a practical point of

Table 4.1: Node classification results.

BlogCatalog								Wikipedia					
	% Labels	40%	50%	60%	70%	80%	90%	40%	50%	60%	70%	80%	90%
Macro-F1 (%)	ComE	24.9	25.2	26.2	26.4	26.5	27.2	4.1	5.3	7.0	8.9	9.8	11.1
	ComE+	25.7	26.2	26.3	26.6	27.1	27.4	4.3	5.4	7.2	9.0	10.2	11.1
	DeepWalk/SF	20.6	20.9	21.5	21.5	22.2	22.5	3.1	3.3	3.4	3.6	4.6	8.1
	Line	10.1	10.2	10.8	10.8	10.9	10.9	2.5	2.8	3.3	3.9	5.3	8.9
	Node2Vec	22.8	23.1	23.4	23.6	24.1	24.6	3.9	3.9	4.4	5.1	6.1	10.9
	GraRep	22.9	22.9	23.3	23.5	23.6	24.0	4.8	5.0	5.8	6.3	8.1	11.5
	M-NMF	15.2	15.3	15.3	15.6	15.7	15.8	4.9	5.0	5.4	5.8	7.0	9.8
	PRUNE	4.21	4.3	4.5	4.5	4.6	4.7	1.9	2.6	3.5	4.5	4.9	5.6
Micro-F1 (%)	ComE	38.0	38.5	40.5	41.1	41.7	42.8	24.0	27.9	32.9	40.8	44.0	51.1
	ComE+	40.0	40.8	41.3	41.8	42.5	43.8	24.2	28.7	33.2	41.2	45.4	51.2
	DeepWalk/SF	35.4	35.9	37.9	38.0	38.3	39.0	21.6	22.2	22.2	23.9	28.0	41.0
	Line	27.2	27.6	29.5	29.8	30.2	30.5	22.5	23.0	24.4	26.1	30.5	44.2
	Node2Vec	36.9	37.4	39.4	39.6	39.9	40.5	23.9	24.1	25.1	26.8	31.1	45.3
	GraRep	38.2	38.6	40.2	40.7	40.9	42.0	22.2	26.3	27.5	29.1	33.4	46.6
	M-NMF	31.0	31.5	33.2	33.4	33.8	34.4	22.9	27.1	28.2	29.6	33.7	46.1
	PRUNE	14.3	14.8	15.	15.4	15.6	16.3	16.6	20.2	24.7	31.9	35.0	40.9

DBLP								Rochester					
	% Labels	40%	50%	60%	70%	80%	90%	40%	50%	60%	70%	80%	90%
Macro-F1 (%)	ComE	92.1	92.1	92.2	92.2	92.2	92.5	31.8	34.3	37.8	43.0	49.7	57.8
	ComE+	91.9	92.0	92.1	92.2	92.2	92.4	32.4	34.8	38.1	43.7	53.7	60.2
	DeepWalk/SF	91.0	91.1	91.2	91.2	91.2	91.4	28.3	29.5	33.3	38.4	44.1	51.8
	Line	90.2	90.3	90.3	90.4	90.4	90.7	29.5	31.5	34.6	41.1	47.4	57.1
	Node2Vec	91.2	91.4	91.4	91.5	91.5	91.7	28.6	30.9	34.6	40.0	46.6	55.2
	GraRep	90.4	90.5	90.5	90.5	90.6	90.8	31.9	34.0	37.5	42.5	48.8	57.4
	M-NMF	89.3	89.4	89.5	89.5	89.6	89.8	31.4	33.9	37.3	44.2	48.3	57.3
	PRUNE	21.3	21.3	21.7	22.3	22.4	22.7	8.1	8.7	9.7	11.0	13.2	15.7
Micro-F1 (%)	ComE	92.4	92.5	92.6	92.6	92.6	92.8	84.7	85.1	85.4	85.8	86.6	87.7
	ComE+	92.3	92.4	92.5	92.5	92.6	92.7	85.3	85.7	86.0	86.5	86.8	88.3
	DeepWalk/SF	91.4	91.6	91.6	91.6	91.6	91.8	81.2	81.8	82.3	82.6	82.9	83.3
	Line	90.8	90.9	91.0	91.0	91.0	91.4	82.9	84.1	84.5	85.1	85.4	85.8
	Node2Vec	91.7	91.9	91.9	91.9	92.0	92.2	80.9	81.6	81.8	82.2	82.6	83.2
	GraRep	90.9	91.0	91.1	91.1	91.1	91.4	85.1	85.4	85.5	86.1	86.1	87.4
	M-NMF	90.0	90.1	90.2	90.2	90.3	90.5	84.9	85.3	85.8	86.4	86.4	87.8
	PRUNE	37.8	37.9	38.0	38.3	38.4	38.4	26.0	26.6	27.4	27.9	29.1	30.2

Mich								Amherst					
	% Labels	40%	50%	60%	70%	80%	90%	40%	50%	60%	70%	80%	90%
Macro-F1 (%)	ComE	28.4	30.6	31.6	33.2	36.9	42.8	53.9	57.5	58.9	61.4	65.7	73.8
	ComE+	29.3	32.0	33.2	34.2	37.3	43.2	53.8	57.2	60.4	62.4	66.6	75.4
	DeepWalk/SF	28.3	30.3	30.6	32.7	33.2	37.8	46.8	50.1	51.8	53.3	57.6	64.1
	Line	28.9	30.1	30.5	33.5	34.1	38.0	48.7	52.4	55.2	55.5	59.5	67.9
	Node2Vec	27.3	29.7	30.7	33.1	34.4	39.1	49.6	52.3	53.5	55.0	57.6	66.4
	GraRep	29.0	30.2	30.9	32.9	35.4	40.9	54.3	55.7	60.3	61.5	62.9	69.9
	M-NMF	29.4	31.1	32.5	33.8	34.3	40.1	50.5	52.5	56.4	57.3	60.0	68.1
	PRUNE	10.0	10.3	10.8	11.3	11.6	13.7	10.4	11.1	11.9	12.0	12.7	15.5
Micro-F1 (%)	ComE	60.9	61.7	61.9	62.3	63.2	64.6	90.3	90.4	90.6	90.8	91.2	91.5
	ComE+	61.1	61.9	62.3	63.1	64.1	65.7	90.3	90.7	90.9	91.0	91.6	91.8
	DeepWalk/SF	57.6	59.2	59.7	59.8	60.9	62.5	87.8	87.9	88.2	88.4	88.5	88.5
	Line	58.7	59.9	60.4	61.2	61.5	63.7	88.6	88.6	88.7	88.8	88.9	89.1
	Node2Vec	57.5	58.9	59.7	60.0	61.6	63.2	88.1	88.6	88.9	89.2	89.4	89.9
	GraRep	60.5	60.9	61.4	61.6	63.0	63.5	90.8	90.8	90.9	91.0	91.0	91.5
	M-NMF	60.6	61.2	61.4	61.5	61.6	62.9	89.9	90.3	90.5	90.6	90.8	91.3
	PRUNE	22.2	22.5	22.8	23.2	23.2	23.6	25.7	26.5	27.0	27.1	27.3	27.9

Table 4.2: ComE+ p-values w.r.t. the baselines using 80% of the lables.

		BlogCatalog	Wikipedia	DBLP	Rochester	Mich	Amherst
Macro-F1	DeepWalk	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	Line	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	Node2Vec	< 0.01	< 0.01	< 0.01	< 0.01	< 0.05	< 0.01
	GraRep	< 0.01	= 0.05	< 0.01	< 0.01	= 0.29	< 0.07
	M-NMF	< 0.01	< 0.01	< 0.01	< 0.01	< 0.05	< 0.01
	PRUNE	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
Micro-F1	DeepWalk	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	Line	< 0.01	< 0.01	< 0.01	< 0.05	< 0.01	< 0.01
	Node2Vec	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	GraRep	< 0.01	< 0.01	< 0.01	< 0.01	= 0.13	< 0.08
	M-NMF	< 0.01	< 0.01	< 0.01	= 0.29	< 0.05	< 0.01
	PRUNE	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01

view, detecting the exact number of communities in a graph is challenging and can impact the overall performances in terms of both community detection and node classification (more in Figs. 4.8 and 4.9).

Lastly, PRUNE, which explicitly models only the first-order proximity while the second-order proximity is only derived as representation propagation, produces suboptimal results on the tested datasets. Another possible reason for the poor performance of PRUNE is the usage of undirected graphs that could affect the *PMI* matrix used during inference. In undirected graphs, the in-degree and the out-degree are too homogeneous, preventing PRUNE to properly learn a valuable representation.

4.4.7 Parameter Sensitivity

In this section, we are going to evaluate the impact that different parameters have on the performance of ComE+. Note that to better evaluate the impact of the proposed variational inference algorithm, we extensively compared ComE+ w.r.t. ComE.

Impact of α and β

We tune the model parameters α (trade-off parameter for context embedding) and β (trade-off parameter for community embedding) in Fig. 4.7. In each figure, we tune one parameter within the range of $[0.001, 1]$ while fixing the other parameter as 0.1. Please note that the Macro-F1 and Micro-F1 results are obtained using 80% of the dataset as training set, and the results reported are the average of 10 different executions. Moreover, the considerable difference in Conductance and NMI between ComE and ComE+ is related to the different algorithms used for community detection.

On the one hand, as stated in Ch. 3.4.5, $\alpha = 0.1$ and $\beta = 0.1$ is a reasonable setting for ComE on almost all the datasets and all the metrics. However, note that on Rochester, $\beta = 0.01$ obtains a relative improvement of 1.5% in terms of Micro-F1 score, but this could be related to the presence of communities formed

Figure 4.7: Impact of the parameters α and β for ComE+ and ComE.

by a single member. This is a challenging problem setting for ComE because it can produce a divergent loss function (look at Fig. ?? for a better understanding).

Figure 4.8: Impact of k parameter in terms of Micro-F1 and Macro-F1.

Figure 4.9: Impact of k parameter in terms of Conductance and NMI.

Finally, it is interesting to see that Amherst dataset provides a better NMI value with $\alpha = 0.1, \beta = 1$. This could be related to the fact that we use more community embeddings' information to overcome the lack of random walks' information.

On the other hand, ComE+, in most of the cases, performs better when it is set to $\alpha = 0.1$ and $\beta = 1$. Compared to ComE, ComE+ is less sensitive to the settings of α and β , which validates the robustness of ComE+ for real-world datasets. On multi-label datasets, we observe the same noise effect as in ComE. For example, setting $\beta = 0.1$ on BlogCatalog gives a relative improvement of 4.5% and 2.7% in terms of Macro-F1 and Micro-F1, respectively. As for the performance of community detection (Conductance and NMI), the less sensitivity of ComE+ to different parameter settings is probably because of IGM's sampling scheme, which reduces the effect of the noise introduced by different configurations.

Impact of K

Since ComE+ is able to automatically learn the number of communities in a graph, we also study the impact of K on ComE and ComE+. We denote K' as the real number of communities in each dataset. We generate different embedding using ComE and ComE+ by assigning K to the values of $[3, K', 10, 20]$ for DBLP and $[5, 10, K', 50, 100]$ for the other datasets. Again, we used 80% of the nodes for training and the rest 20% for testing. The impact of K in terms of node classification and community detection are reported in Fig. 4.9 and Fig. 4.8, respectively. We made

Figure 4.10: Impact of the node embedding's dimension in terms of Micro-F1 and Macro-F1.

Figure 4.11: Impact of the node embedding's dimension in terms of Conductance and NMI.

the following observations from these two Figures. First, on DBLP, both ComE and ComE+ are robust w.r.t. the setting of K in terms of node classification; however, both of them are sensitive to K in terms of community detection. The performance of ComE reaches the top when $K = K'$; therefore, it suffers a relative 140% to 189% performance decrease in terms of Conductance and 17.6%–25.6% decrease in terms of NMI when $K \in [3, 10, 20]$. As expected, ComE+ is more robust when $K \geq K'$. The relative differences of performance varies between 0%, +0.2%] (NMI), and $[-2.3\%, 0\%]$ (Conductance). Second, the models usually perform the best when $K = K'$ in most cases; however, on rare occasions (e.g., on Amherst), ComE+ has better performance when $K > K'$. Although the improvement is not significant, it is indicative of community structures that do not fully match the labels assignment process. Finally, as expected, the performance of ComE+ is more robust than ComE for both community detection and node classification w.r.t. the parameter K . This validates the ability of the proposed Bayesian inference algorithm (Ch. 4.3) to automatically detect the correct number of labels from the data distribution.

Impact of d

Finally, we studied the impact of the node embedding's dimension on the two different models. In Fig. 4.10 and 4.11 the classification and clustering performance for

(a) Model convergence.

(b) Model efficiency. Note that the average time per iteration is reported.

— BlogCatalog — DBLP — Wikipedia — Rochester — Mich — Amherst

ComE and ComE+ is shown if we vary the dimensionality of $\phi_i \in [32, 64, 128, 256]$.

As for the results in Ch. 3.4.5, Fig. 4.10 shows that the node classification performances of the proposed algorithms are largely consistent w.r.t. the dimensionality of the embedding created. Only in Figs. 4.10e and 4.10f, it is possible to detect an anomaly for ComE+, where $d = 64$ seems to significantly outperform larger values in terms of Micro-F1 score. While this is consistent with ComE’s results, it indicates that under some performance metrics, SVM can define better decision boundaries in a dense vector space where nodes of the same class are inherently closer.

Instead, from Fig. 4.11, it is possible to notice how the community detection performances are affected by the embedding dimension. On the one hand, BlogCatalog and Wikipedia are subject to over-fitting when $d > 128$. This could be related to the fact that in a higher dimensional space, communities are more separate; thus, it is more challenging to detect overlapping communities. On the other hand, the DBLP dataset (Fig. 4.11b) does not present any significant changes in performance w.r.t. the dimensionality of ϕ_i neither for ComE nor for ComE+.

4.4.8 Convergence and Efficiency

As for ComE and ComE+, we tested the convergence and efficiency. Note that for these experiments, we set $d = 2$ and $\zeta = 5$.

We recorded the value of the loss functions (Eq. 4.6) at the end of every iteration (i.e., line 13 in Alg. 2). Note that we normalized the loss value by $|V|$ for different datasets to better illustrate them in one figure. As shown in Fig. 4.12a and similar to ComE, the loss ComE+ converged quickly within 2–3 iterations.

To demonstrate the efficiency of ComE+, we tested them on all the six datasets at different scales. For each dataset, we generated three subsets in which we maintained 25%, 50%, and 75% of the total number of edges and nodes. The diagram in Fig. 4.12b shows the processing time of ComE+ on different datasets. Moreover, the processing time of ComE+ is linear to the graph size (i.e., $|V|$ and $|E|$). This validates our complexity analysis at the end of Ch. 4.3.1.

4.5 Conclusion

In this chapter, we propose a novel model for node and community embedding. We also propose a new problem setting w.r.t. Ch. 3. To the best of our knowledge, this is the first algorithm that explicitly preserves first-, second- and community-aware higher order of proximity without knowing in advance the number of partitions present in the network. Therefore, ComE+ is able to handle an unknown and up to an infinite number of communities. In ComE+, we also try to model the closed loop, yet with a special focus on using a stick-breaking construction approach to generate the communities and sample each node’s community assignment. We also designed an efficient iterative inference algorithm for ComE+, which can still retain a low complexity of $O(|V| + |E|)$. We evaluated ComE+ on seven real-world datasets with multiple application tasks. ComE+ is shown to be capable of outperforming all the baselines and achieving new state-of-the-art results for classification and clustering tasks. When possible, we further compared ComE+ with ComE to better evaluate the achievement of the proposed variational inference algorithm. We also showed that ComE+ improves the baselines in the task of graph visualization. Finally, we studied the parameter sensitivity, model convergence, and model efficiency. Our models can quickly converge and scale well w.r.t. the graph size $|V| + |E|$.

While ComE+ relaxes a strict hypothesis for community embedding (knowing the number of communities in advance), we argue that the path sampling approach is likely to produce suboptimal results because there is no feedback from the embedding process. Thus, an end-to-end approach would provide a more sound solution.

Chapter 5

An Attention-Based Model for Learning Dynamic Interaction Networks

5.1 Introduction

In the previous chapters, we have considered the problem of community embedding under the assumption of having a homogeneous graph. However, biological systems, as well as the modern society, are better represented by heterogeneous networks in which each vertex is not only defined by its links but also by its attributes. While a community is defined as a mesoscopic structure that arises between densely connected nodes (Def. 1.3.1), we explore the possibility of learning a complex interconnected structure based on the attributes of each system's component in this chapter, i.e., we explore the possibility of learning an *implicit graph structure based on the vertices' content*. Thus, it would be possible to leverage such an implicit network to perform attribute-based community embedding. We argue that the combination of traditional community embedding and attribute-based community embedding methods would eventually provide a better understanding of the modern world. However, we want to emphasize that in this chapter, we are not directly addressing the attribute-based community embedding problem, but we are rather investigating the possibility of using the nodes' attributes to learn an implicit network.

Despite the popularity of graphs as representational frameworks, they remain an impractical data-structure because they are characterized by non-Euclidean space in which it is challenging to define localized operations. On the one hand, RNNs have been proven capable to preserve the long-term dependencies present in sequential tasks [44, 52, 101, 127]. However, such recurrent architecture can only handle array-like data, while many problems are better expressed as a graph [140]. On the other hand, traditional graph embedding techniques [25, 110, 129] assume the existence of a well-defined structure. Nevertheless, many real-world scenarios are characterized by dynamic networks in which the structure or the attributes might change over time [37, 148].

Lately, the research community has focused on solving well-structured tasks such as modeling physical system [9, 72, 120], protein-to-protein interaction [43, 115], or

Figure 5.1: A motivational example Fig. 5.1b showing the implicit graph structure learned w.r.t. the target station v_i . Note that among all the surrounding stations, v_j is the one that has the highest impact on v_i because it is the closest one and is located on the same freeway lane.

knowledge-based completion [56]. In this work, we address the problem of jointly learning the *evolution of a complex system* and the *implicit relational structure* that connects the components of the system. In other words, we focus on the scenario wherein the influence weights that the neighbors exert on a given vertex are not explicitly defined, but they are learned in an unsupervised way based on the system’s dynamic. Note that an edge between node v_j and node v_i with weight equal to 0 indicates that v_j does not influence v_i ; thus, it is equal to removing that edge from the graph.

Considering the traffic prediction problem shown in Fig. 5.1a and after analyzing the data reported by each traffic station, it is possible to predict the road congestion at a given time and derive the causal relationships that governs the car flow. Ultimately, the traffic jam is better captured by measuring a multitude of vehicles that transit across a set of geographically close stations rather than independently considering each traffic terminal. A graph structure (Fig. 5.1b) that describes the station’s interdependence could help the transportation authorities to better understand and tackle all the factors that cause bottlenecks on the roads. While this motivational example focuses on the traffic prediction problem, such a setting is common in multiple applications such as image semantic segmentation [78, 86], semantic role labeling [98], and event extraction [90].

In this chapter, we propose a novel *attention-based neural network* that generates a fixed-size embedding representative of the evolution of the nodes’ attributes and the interactive structure of the considered problem. We extensively evaluate this approach on two real-world datasets and on three synthetic datasets. The primary contributions of this chapter are as follows:

1. We propose a model that is able to automatically infer the interactive structure of a problem.
2. To better preserve the problem’s semantic, we propose a different parametrization for the target vertex and its neighbors.

3. We show that our attention architecture is comparable or outperforms structured and unstructured algorithms.

5.2 Dynamic Interaction Attention Network

Spatio-temporal graphs (st-graphs) [47, 82, 84, 89] have received considerable attention from the research community because they provide a natural framework to represent complex systems. Structural-RNN [66] is the first model that combines the modeling ability of a deep neural network with the expressive power of st-graphs. Overall, the authors demonstrate that this setting better preserves the causal structure of multiple real-world scenarios such as human motion forecasting and autonomous car driving. However, such an approach requires a human defined interactive structure; thus, it is subject to suboptimal solutions when the problem architecture is unknown. Therefore, we are looking for a model that can automatically infer the problem structure based on the nodes' attributes.

5.2.1 Problem Formulation

According to [9], given a set of atomic elements V and a set of relations E , it is possible to define a complex system as a graph $G = (V, E)$. Nevertheless, we claim that such a definition is *not applicable* to many real-world scenarios in which the set of relations is not known a priori or where it changes over time. To this end, we argue that a complex system is better represented by a dynamic network structure $G = (V, E, W^t)$, where V is the nodes' set, E is the edges' set, and W^t is the set of the edge's weight at time t ($w_{j,i}^t$), which is representative of the influence exerted by node v_j on node v_i . For example, according to the traffic forecasting problem, to better predict the car's flow through a target station, it is necessary to consider the traffic through its neighboring stations. However, as the status of the neighboring stations is dynamic, we assume the influence changes over time.

More formally, let us define $\mathbf{x}_i^t \in \mathbb{R}^d$ as the feature vector of an element $v_i \in V$ at time t , e.g. the average car speed and the car flow through a traffic station. Moreover, let us define $\mathbf{x}_i = (\mathbf{x}_i^1, \dots, \mathbf{x}_i^T)$ as the time-series of v_i 's attributes and N_i as the neighbors of v_i . Then, our objective is to predict the attributes' evolution of a target object v_i conditioned on the behaviors of its neighbors.

We formalizes our model following a two-step approach common in various message passing architectures [48] (MPNN). First, we computed a fixed-size edge representation as follows:

$$\mathbf{h}_{N_i}^t = f_e(\phi(\mathbf{x}_i^t), \psi(\mathbf{x}_j^t)) \quad (5.1)$$

where $\phi(\cdot)$ and $\psi(\cdot)$ are encoding functions. Then, based on the implicit graph structure captured in $\mathbf{h}_{N_i}$, we predicted the behavior of a target node v_i :

$$\hat{\mathbf{x}}_i^{t+1} = f_v(\phi(\mathbf{x}_i^1), \dots, \phi(\mathbf{x}_i^t), \mathbf{h}_{N_i^1}^1, \dots, \mathbf{h}_{N_i^t}^t) \quad (5.2)$$

As we are interested in predicting the evolution of the attribute associated to a target node, we formalized our problem as a regression task, i.e., we trained the

Figure 5.2: Overall model architecture.

model in an end-to-end mode and minimized the mean squared error (MSE):

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{|V|} \frac{1}{T-1} \sum_{t=1}^{T-1} (\mathbf{x}_i^{t+1} - \hat{\mathbf{x}}_i^{t+1})^2 \quad (5.3)$$

5.2.2 Model Architecture

While we have formalized the problem as a regression task over each vertex, in this section, we are going to define the **Dynamic Interaction Attention Network** (DIAN) architecture.

First, most of the related work encodes the graph’s edges as the concatenation of the node features and then applies a graph attention network (GAT) to learn a proper network structure [38, 65, 135, 139] or processes each problem aspects with different RNNs organized in a feedforward structure [66]. However, the first approach does not properly handle dynamic graphs and does not adequately differentiate between the node attributes as well as the neighbors’ attribute. Instead, the second approach is not able to automatically learn the influence of each neighbor. To preserve the best of both worlds, we followed [66] for partitioning the nodes according to their semantics, i.e., we differentiated between the target node v_i and its neighbors $v_j \in N_i$. As shown in Fig. 5.2 and defined in Eq. 5.1, we employed two distinct parametrizations to encode target node features ($\phi(\cdot)$) and the neighbors’ attributes ($\psi(\cdot)$). In this manner, we reduced the number of parameters to learn because all the neighbors share the same parametrization, but we conserve the semantic difference between a target node and its neighbors.

Given that for each vertex $v_i \in V$ we have an associated sequence of attributes, we defined $\phi(\cdot)$ and $\psi(\cdot)$ as two distinct RNNs. RNNs are a deep learning model that has been proven capable of solving various sequential problems [44, 52, 101] because of their ability to preserve a smooth transition between subsequent time-steps. An RNN is a hidden state relative to the input sequence $\mathbf{x}_i$, and it can be expressed as

follows:

$$\mathbf{h}_i^t = f_{\text{RNN}}(\mathbf{x}_i^t, \mathbf{h}_i^{t-1})$$

where f_{RNN} is a non-linear function that preserves the time causality of the input sequence. Note that, we used the well-known GRU model [30] as a recurrent function.

We defined the node and neighbor feature encoding as $\mathbf{h}_i = \phi(\mathbf{x}_i)$ and $\mathbf{h}_j = \psi(\mathbf{x}_j)$. We can then formalize the message passing (Eq. 5.1) and vertex update (Eq. 5.2) functionalities of the DIAN model. To obtain a meaningful information flow, we followed [145] for defining a message passing formulation as follows:

$$\mathbf{h}_{N_i}^t = \sigma\left(\sum_{j \in N_i} \sum_{t'=1}^t w_{j,i}^{t'} g(\mathbf{h}_j^{t'})\right) \quad (5.4)$$

$$g(\mathbf{h}_j^t) = \mathbf{W}_g \mathbf{h}_j^t \quad (5.5)$$

$$f(\mathbf{h}_i^t, \mathbf{h}_j^t) = e^{\left(\frac{\mathbf{h}_i^t \mathbf{W}_s \mathbf{h}_j^{t\top}}{\tau}\right)} \quad (5.6)$$

$$w_{j,i}^t = \frac{1}{\sum_{k \in N_i} \sum_{t=1}^T f(\mathbf{h}_i^t, \mathbf{h}_j^t)} f(\mathbf{h}_i^t, \mathbf{h}_j^t) \quad (5.7)$$

where $\mathbf{W}_s$ and $\mathbf{W}_g$ are learnable parameters, $\mathbf{h}_j^{t\top}$ stands for the transpose of $\mathbf{h}_j^t$, τ is a temperature parameter to be hand-tuned, and $\sigma(\cdot)$ is an activation function such as the ReLU function. Such formulation enables the DIAN architecture to learn a fixed-size embedding of the implicit interaction graph representative of the studied problem. Note that Eq. 5.7 is a *softmax* formulation similar to the scaled dot product proposed in [137]; however, it is computed over space (neighbors) and time dimensions. Moreover, we propose a bilinear transformation to preserve the interaction between two nodes. Note that a similar transformation has been used in [152] to efficiently solve the link prediction problem. In Ch. 5.3, we demonstrated that such a simple formulation provides competitive results w.r.t. more complex models. Moreover, we found it beneficial to apply K attention heads [137] to better preserve the graph's structure. Thus, Eq. 5.4 is revised as a linear combination of each head output:

$$\mathbf{h}_{N_i,k}^t = \sigma\left(\sum_{j \in N_i} \sum_{t'=1}^t w_{j,i}^{t'} g(\mathbf{h}_j^{t'})\right) \quad (5.8)$$

$$\mathbf{h}_{N_i}^t = f_{\text{ln}}\left(\sum_{k=1}^K \mathbf{W}_h^k \mathbf{h}_{N_i,k}^t\right) \quad (5.9)$$

where $\mathbf{W}_h^k$ are learnable parameters and $f_{\text{ln}}(\cdot)$ stands for the layer normalization function [6].

Finally, to update the vertex representation (Eq. 5.2), we used the concatenation approach proposed for multiplet graph neural networks:

$$\hat{\mathbf{x}}_i^{t+1} = f_a(\mathbf{W}_o[\mathbf{h}_i^t, \mathbf{h}_{N_i}^t]) \quad (5.10)$$

where $\mathbf{W}_o$ is a matrix of learnable parameters, $[\cdot, \cdot]$ stands for the concatenation operator, and $f_a(\cdot)$ is any activation function.

Note that

- We have found it useful to add a recurrent connection from the error computed at the three previous time-steps ($[\hat{\mathbf{x}}_i^{t-1} - \mathbf{x}_i^{t-1}, \hat{\mathbf{x}}_i^{t-2} - \mathbf{x}_i^{t-2}, \hat{\mathbf{x}}_i^{t-3} - \mathbf{x}_i^{t-3}]$) to the next target node input ($\mathbf{x}_i^t$). Such a connection has proven effective when modeling a volatile time-series.
- Since we focus on temporal data: during the computation of $w_{j,i}^t$, it is crucial to mask all the $t+1, \dots, T$ time-steps to maintain the problem’s time causality.
- The proposed DIAN model is different from the structural-RNN [66] architecture because it does not require a prior interaction structure. It automatically learns the influence that each neighbor exerts on a given node. Such a mechanism allows the DIAN architecture to better generalize on new tasks as it does not require handcrafting the possible relations. As a side effect, because of the attention module used, the interaction structure learned becomes explicit and straightforward to interpret.
- The proposed DIAN model is technically different from the GAT model because we focus on dynamic interactive graphs, while in [139], the authors focus only on explicit graphs structure. Moreover, the GAT model is designed to perform only structural learning and does not align the nodes’ sequences.
- Our model is also different from NRI [72] because we consider the edge weights and not the different edge types.

5.3 Experiments

To evaluate the insight of Ch. 5.1 and 5.2, we designed a series of real-world experiments and toy tasks. The designed toy tasks culminated in synthetically generated datasets created with the purpose of studying the convergence of the DIAN model in certain specific scenarios. The real-world datasets assessed the performance of the proposed architecture on existing problems. Note that during all the experiments, we used the mean squared error (MSE) as loss function (Eq. 5.3) and the rooted mean squared error (RMSE) as evaluation metrics.

5.3.1 Datasets

Synthetic Datasets

The synthetic datasets have been designed to replicate the same problem setting of the real-world tasks. The primary purpose is to compare the performances of different models in a controlled environment to evaluate certain desired properties. Thus, we generate three datasets each of them composed by 10K examples for training, 2K for evaluating, and 2K for testing.

- We designed the *Dynamic Noise dataset* specifically to test if a model is capable of detecting the relevant neighbors given the entire set N_i . Detecting relevant neighbors is one of the most fundamental characteristics required by our model because it reflects the capability of learning an implicit graph structure.

Furthermore, we assigned a monotonically increasing sequence to each vertex. Then, each target node was connected with a random number of neighbors ranging between two and four. While an irregular sequence is assigned to one neighbor, the remaining neighbors have a time series correlated to the target node. At each time-step, the goal is to compute the mean between the values of the target node and the relevant neighbors.

- Structural learning ability is crucial in relational inference. Dynamic interactive graphs characterize our problem setting because the vertices' attributes evolve over time. However, in some cases, the neighbors' sequences might not be aligned with the target node sequence. For example, it is reasonable to think about the traffic flow through a target station as compared to the throughput of the prior stations but for an offset. Thus, we want to investigate if the DIAN model can align the sequence of a target node with one of its neighbors.

To this end, the *Time-shifted dataset* associates a triangular wave-formed sequence to each target node, while each of its four neighbors has a similar wave-formed series shifted ahead in time and amplitude scale. The objective is to predict the one-step-ahead value of the target node. To solve this task, a given model has to align the node time series with its neighbors to correctly detect the shift point of the triangular wave-form sequences.

- The *Noise Time-shifted dataset* combines the two previous task to investigate which model is able to jointly align and detect a variable number of relevant neighbors. For this purpose, this synthetic dataset presents a set of target nodes characterized by a triangular wave-form time series, up to two irrelevant neighbors, and at least two relevant neighbors having a sequence similar to the one of the target node, but shifted in time and amplitude scale by a random offset. Note that in this task, the final aim is to predict the next value of the node's sequence.

An example for each handcrafted datasets is reported in Fig. 5.3.

Real-World Datasets

We also use two real-world datasets to evaluate the proposed architecture. Note that while in Ch. 5.1 we used the traffic flow prediction problem as a motivational example, we test our model on different tasks such as the energy load forecast. We argue that all tasks are related to an interactive graph structure. For example, buildings closely geo-located are subject to the same climatic factors; thus, they share a common consumption pattern.

Figure 5.3: A training example for each synthetic dataset. Note that the neighbors are represented by a thinner line w.r.t to the node and label time series. Moreover, noise neighbors have the thinnest line.

- The *EnerNOC GreenButton*¹ dataset contains anonymized geo-located energy usage data from 100 commercial/industrial sites for 2012. The energy consumption is re-sampled at 3-h intervals, anomalous readings are eliminated, and the energy consumption is divided by the building’s square footage to obtain a more comparable metric. Note that all the time series are normalized through the softplus function ($\text{softplus}(x) = \log(1 + e^x)$) and categorical features such as the building type, the time of day, and the day of the week are encoded in a one-hot format. We assume that buildings that are closely geo-located are subject to the same climatic factors; therefore, they share a common consumption pattern. To this end, for each building, we compute the top four geographically closest sites to form our neighbor’s set.
- The *PeMS bay-area traffic*² dataset, as shown in Fig. 5.1a, comprises traffic data crawled from 3665 traffic-control stations around the San Francisco Bay area. Traffic data comprise car flow (car/10 min) and average speed (mph) through each station for the weekdays ranging from May 2017 to July 2017. Each training example is formed by one-day data for a given station, and the objective is to predict traffic congestion for the next hour. We found it useful to normalize the time series with a scaled softlog function ($\text{softlog}(x) = \log(1 + \frac{x}{10})$). Our intuition is that geographically close stations suffer from similar traffic condition; therefore, for each traffic-control terminal, we computed the top six closest stations as neighbors.

Finally, we have provide a summary of the used datasets in Tab. 5.1.

5.3.2 Baselines

We designed our baselines to accurately evaluate how different neural networks can learn and exploit structured information in the data.

¹<https://open-enernoc-data.s3.amazonaws.com/anon/index.html>

²<https://bit.ly/2sB27Pa>

Table 5.1: Datasets summary. Note that for all the datasets, all the examples related to a given node and associated neighbors are present in the training, evaluation, or test set. Thus, we are in an inductive learning setting.

	EnerNOC	PeMS	Synthetic
Node number	100	3665	14000
Neighbors for each node	4	6	4
Sequence lenght	16	138	10
Training examples	9570	50000	10000
Eval examples	4176	12000	2000
Test examples	4002	11300	2000

- *Structural-RNN* [66]: directly models a graph structure given in input because of its feed-forward architecture, but it is not able to learn an implicit graph structure and dynamically adapt to the changes in such a structure.
- *Simple-RNN* [30]: to investigate the impact that the neighbors’ information has on the final performance, we used a single GRU to only model the node sequences.
- *Structural-FF*: to evaluate the importance of modeling the smooth transition present in our time series, we proposed a baseline that uses the same feed-forward architecture of the S-RNN architecture, but it employed multilayer perceptrons (MLPs) rather than RNNs.
- *Simple-FF*: to understand how difficult it is to solve the proposed tasks, a simple MLP that only processes the node information is used in the evaluation. Note that such a network architecture can neither model the smooth time transition nor the structured information.
- *GAT* [139]: The GAT architecture leverages graph structure via a self-attention mechanism. However, the proposed architecture is mostly designed to leverage explicit network structure, while we focus on dynamic interactive graphs. Moreover, GAT is not able to fully leverage the temporal transition in the node’s attribute.
- *RNNGAT*: Because we focus on dynamic graphs where each node is associated to a sequence $\mathbf{x}_i$, we decided to enhance the original GAT architecture by encoding each node using an RNN rather than the originally proposed MLP. Note that in this architecture, the well-known GRU design is also used as the RNN implementation.

In addition to the above-mentioned baselines, we also designed certain degenerated versions of our DIAN model to evaluate the impact of the proposed edge encoding, the recurrent connection, and the attention module:

- *DIAN^{rec}*: is a degenerated version the DIAN model. It does not present the recurrent connection for the caused error and uses the traditional scaled dot

Table 5.2: Parameters setting used during the experiments.

	Symbol	Synthetic dataset	Real-world datasets
Hidden units	u	5	128
Attention heads	h	4	4
Learning rate	lr	0.01	0.001
Temperature	τ	0.45	0.45

product formulation [137] to compute the attention weight (Eq. 5.7). Overall, this results in a more complex architecture; however, as shown in Tabs. 5.3 and 5.4, it does not provide a sensible advantage.

- *DIAN^{trans}*: extends the results of the work done in [137] to the graph setting. In this model, we try to evaluate if the transformer architecture can be directly used to encode the node’s attribute. On the one hand, this drastically reduces the number of parameters to learn w.r.t. an RNN model. On the other hand, it is known that to achieve good soft-alignment performances, a stack of multiple transformers is necessary. To avoid the vanishing gradient problem, the authors used residual connections. However, it is not completely clear how to implement skip connections on a message passing architecture. Thus, we use a single transformer layer to learn the graph architecture, to align the time series, and to preserve the smooth transition between subsequent time-steps.

5.3.3 Experiments Setup

We report a summary of the parameters used during the experiments in Tab. 5.2. To obtain a fair comparison among all the baselines, we set the same number of hidden units u in all the evaluated models. For all the experiments on the synthetic datasets, we set $u = 5$, while in the real-world forecasting problems, we set $u = 128$. We used four attention heads ($h = 4$) in all experiments because the results reported in Fig. 5.6a demonstrate that four heads give the best trade-off between the network complexity and performance improvements. All the reported results are achieved with Adam [71] as an optimizer and 0.01 and 0.001 as the learning rate on the toy and real-world tasks, respectively.

It is also worth mentioning that for the PeMS dataset, a temporal window (tw) is used to reduce the number of time-steps considered during back-propagation and to speed up the learning process of the different models. In particular, for the PeMS dataset, we use a window size of 20 time-steps ($tw = 20$). Finally, note that all the models are trained for 100 epochs while the evaluation is performed after every ten epochs. The parameter setting that better performs on the eval set is then used for testing.

Remark: All the reported results are the average of four independent executions.

Table 5.3: Results obtained on the toy tasks. Note that all the reported results are the average of four executions.

Model	Dynamic Noise	Time-shifted	Noise Time-shifted
	<i>RMSE</i>	<i>RMSE</i>	<i>RMSE</i>
Structural-RNN	0.56 ± 0.0038	0.20 ± 0.0064	0.36 ± 0.0015
Simple-RNN	0.67 ± 0.0014	0.53 ± 0.0076	0.51 ± 0.0028
Structural-FF	0.68 ± 0.0078	0.63 ± 0.0052	1.19 ± 0.016
Simple-FF	0.73 ± 0.029	1.43 ± 0.0086	1.44 ± 0.0099
DIAN	0.07 ± 0.0013	0.20 ± 0.0034	0.24 ± 0.0084
DIAN ^{rec}	0.09 ± 0.0083	0.19 ± 0.0025	0.25 ± 0.0059
DIAN ^{trans}	0.32 ± 0.0099	1.03 ± 0.018	1.02 ± 0.21
GAT	0.3 ± 0.024	0.62 ± 0.0079	1.25 ± 0.25
RNNGAT	0.12 ± 0.0055	0.32 ± 0.0019	0.77 ± 0.0014

5.3.4 Results

Toy Tasks

Tab. 5.3 reports the performance of every tested model on the toy tasks.

Note that the proposed DIAN model can outperform almost all the baselines obtaining a relative improvement between 4.1% and 29.6% in two out of three datasets, but the DIAN^{rec} architecture outperforms it by a relative 5% in the Time-shifted dataset. However, the Time-shifted dataset is the only one that does not require structural learning since all the neighbors are relevant. Moreover, the reported standard deviation indicates that the DIAN^{rec} improvements are not significant. Thus, we claim that the proposed attention formulation in Eq. 5.6 better preserves the problem’s structure, but it does not perform as well as the scaled dot product on the time alignment task.

From the results, we also argue that having a different parametrization for the target node and its neighbors provides a better problem’s representation since the DIAN model and its degenerated versions show better performances than the other tested architecture. The poor performance on the Dynamic Noise and Noise Time-shifted datasets indicate that the Structural-RNN model can only preserve hand-crafted relations, which are difficult to define in real-world scenarios. Thus, it is not straightforward to scale such architecture on new tasks without a deep understanding of the different scenarios. The GAT model is not able to capture the temporal evolution of the system, and it shows suboptimal results on the toy datasets. The RNNGAT results validate the ideas of jointly perform spatio-temporal attention to learn a dynamic graph structure since it does not show comparable performances with the proposed DIAN model. Finally, the small standard deviation reported in Tab. 5.3 by the DIAN model demonstrates that the proposed attention formulation can consistently solve complex tasks that require structural and temporal learning.

As shown in Fig. 5.4, because of the attention mechanism, we can also investigate the behaviors of the DIAN model to validate the learned problem’s representation. For example, Fig. 5.4a shows how the DIAN architecture can correctly detect the relevant neighbors and focus only on the required time-steps, i.e., it shadows neighbor

(a) Dynamic Noise. Note that in the reported example are only present two relevant neighbors and a random neighbor. (b) Time-shifted attention (c) Noise Time-shifted

Figure 5.4: The attention results of the DIAN model on the toy tasks. Note that y-axes report the target node input at different time-step, while the x-axes report the neighbors’ inputs.

Table 5.4: Results obtained on the real world tasks. Note that all the reported results are the average of four executions.

Model	EnerNOC	PeMS
	<i>RMSE</i>	<i>RMSE</i>
Structural-RNN	0.045 $\pm$ 0.019	55.83 $\pm$ 0.019
Simple-RNN	0.049 $\pm$ 0.0041	58.14 $\pm$ 0.86
Structural-FF	0.067 $\pm$ 0.00167	69.41 $\pm$ 4.95
Simple-FF	0.063 $\pm$ 0.0034	70.8 $\pm$ 0.85
DIAN	0.027 $\pm$ 0.0013	38.68 $\pm$ 0.26
DIAN ^{rec}	0.046 $\pm$ 0.00014	55.83 $\pm$ 0.25
DIAN ^{trans}	0.055 $\pm$ 0.0011	67.15 $\pm$ 0.47
GAT	0.061 $\pm$ 0.0025	252.22 $\pm$ 1.3
RNNGAT	0.055 $\pm$ 0.0022	247.59 $\pm$ 1.2

2, and it mostly focuses on the last input. Fig. 5.4b demonstrates how the proposed architecture can downsample the neighbors’ sequences and correctly aligns them. Moreover, Fig. 5.4c proves that the proposed attention formulation (Eq. 5.9) can jointly solve the two previously tested tasks (align and detect irrelevant neighbors).

Real-World Tasks

According to Tab. 5.4, the proposed DIAN significantly outperform all the baselines on the real-world datasets.

First, note that the DIAN^{rec} and the structural-RNN are the best baselines for both the real-world datasets. However, the results suggest that the DIAN^{rec} is not able to learn an appropriate problem structure for the real-world datasets since its performance is comparable to the handcrafted relations used in the structural-RNN model. This validates the proposed symmetric attention formulation as the

Figure 5.5: The attention results of the DIAN model on the real-world datasets.

DIAN model significantly outperforms all the other methods. Moreover, this result suggests that learning an appropriate network structure provides more information than aligning different time series.

Second, neither the GAT nor the RNN-GAT models can learn a meaningful problem’s embedding. This could be a consequence of their design, which is not optimized to learn a smooth temporal transition of the dynamic interactive graph. Moreover, they do not use a different parametrization for a target node and its neighbors. Thus, they are subject to suboptimal results concerning the RMSE metric.

Third, note that the $\text{DIAN}^{\text{trans}}$, structural-FF, and simple-FF are not able to fully capture the problem’s structure. We argue that such effects are the consequence of their shallow architecture, which is not able to jointly model the smooth temporal transition and the network structure. While a deep transformer architecture recently demonstrated new state-of-the-art results in modeling temporal sequences [112], it is not yet evident how to leverage such a design in the graph domain.

Finally, Fig. 5.5 shows the attention output of the DIAN model for real world tasks. Overall, both the figures highlight the DIAN ability in detecting the relevant neighbors and learning an appropriate problem structure. Therefore, we conclude that the attentive mechanism proposed in Eq. 5.9 appears to fit a neural message passing architecture, which reduces the number of parameters to learn but provides competitive performances for dynamic interactive networks.

5.3.5 Sensitivity

We tested the impact of the main parameters of the DIAN model.

In Fig. 5.6a, we evaluated the performances of the DIAN model when changing the number of attention heads. Overall, the results demonstrate that adding additional attention heads enables a better representation of the studied problem because each head can focus on a single problem’s aspect. However, on the toy tasks, $h = 4$ provides the best trade-off between the model complexity and the performance’s gain in two out of three datasets.

Fig. 5.6b shows the impact that the hidden dimension (parameter u) has on the final output. As expected, a bigger hidden layer provides better performances. However, $u = 128$ provides significant improvements w.r.t. other settings.

A skeptical reader might argue that executing an attention mechanism to learn

(a) Impact of the number of attention head on toy datasets

(b) Impact of hidden dimension on real-world datasets

(c) Impact of the neighbors numbers on toy datasets

a graph embedding would not scale to bigger interactive networks. Thus, we evaluated the DIAN model over other synthetically generated datasets. In Fig. 5.6c, we reported the performances over the same toy tasks described in Ch. 5.3.1, but we changed the number of neighbors for each target node in the range $[4, 100, 1000, 3000]$. Note that for the dynamic noise and the noise Time-shifted datasets, only the total amount of neighbors change, while the number of relevant neighbors remained constant to four. On the one hand, on the dynamic noise and the noise time-shifted datasets, the model reports a significant decrease in the performances but it is still able to correctly solve the task; moreover, it outperforms most of the tested baselines. On the other hand, on the time-shifted dataset, the performance increases w.r.t. the number of neighbors. Note that such a behavior is expected since the time-shifted dataset presents only relevant neighbors; thus, the model has more knowledge to leverage.

5.4 Conclusion

In this chapter, we proposed a novel attentive model for dynamic interactive graphs. Unlike previous studies, our model can fulfill the following:

1. it can jointly learn the underlying graph and align the nodes' time series;
2. it gives a human-readable interpretation of the actual underlying graph structure;
3. it can scale to reasonably sized interaction graphs.

While we did not experience scalability issues even on challenging toy tasks, an alternative solution to this issue is developing a neighbor's sampling process as proposed in [29, 57], but such an alternative will be explored in future studies. About comparable architectures, the DIAN model can preserve the graph structure better. This is related to the different parametrization proposed for a node and its neighbors and to the novel symmetric attention formulation that better captures the node-neighbor interaction.

Overall, the results obtained suggest that a similar approach can be leveraged to learn an attribute-based community embedding in future. In this manner, researchers would obtain a deeper understanding of the modern society by unveiling relationships that are hidden in the graph structure but captured in the nodes' or edges' attributes.

Chapter 6

Conclusion and Future Work

The primary topic of this thesis is graph embedding. In Ch. 1, we introduced the graph theory’s basic concepts that explore the different challenges related to graph analytics. While distributed frameworks provide a valuable solution, *graph embedding* shows promising results in multiple tasks at the node, edge, sub-graph, and graph level. To date, most of the studies have focused on preserving different proximity degree between adjacent nodes; however, an extensive literature review (Ch. 2) shows how very few studies have explored the mesoscopic community structure present in a graph. A meaningful community representation is valuable to multiple community-level application such as community recommendation, community classification, and community visualization. Nevertheless, we argued that a higher-order community-aware proximity metric could enrich node embedding via a closed loop process. To the best of our knowledge, [144] is the first work to jointly solve community detection and community-aware node embedding, but the proposed matrix factorization formulation lacks a human-interpretable community representation.

Instead, in **ComE** (Ch. 3), we proposed a new probabilistic formulation of the community embedding problem. Because a community is a group of densely connected nodes, a community embedding is expected to characterize how its member nodes distribute in the low-dimensional space. Thus, inspired by the Gaussian mixture model [13], we considered each community as a multivariate Gaussian distribution. Such a formulation enables us to define a self-enforcing closed loop between node embedding, community detection, and community embedding. First, node embedding has proven to be capable in improving the community detection performances because it can effectively preserve the network’s structure [23, 76, 110, 131]. Then, the obtained community assignment can be used to fit a meaningful community embedding. Finally, community embedding represents densely connected nodes; therefore, it can provide a higher-order proximity value for the node embedding while providing a human-readable community representation. Such formulation has been shown to be capable of improving the baselines by at least 6.6% (NMI) and 2.2%–16.9% (Conductance) in community detection, and 0.8%–26.9% (Macro-F1) and 0.7%–30% (Micro-F1) in node classification.

6.1 ComE Limitations

After an extensive evaluation process, ComE showed certain limitations. For example, ComE requires that the exact number of communities present in the graphs before the embedding process should be known. Such information is not present in many real-world applications; moreover, a recent study [109] argued that a global optimal node partition is not reachable. Therefore, we proposed a Bayesian approach that is able to overcome this limitation in Ch. 4. Always leveraging the closed loop among community detection, node embedding, and community embedding: **ComE+** considers an infinite number of communities for detection and embedding. To realize this infinite community embedding, we were inspired by infinite Gaussian mixture (IGM) [114]. A simple stick-breaking construction approach [14] was used to generate up to an infinite number of communities and accordingly sample each node’s community assignment. Then, ComE+ samples the node embedding from the community embedding and later optimizes it w.r.t. the first- and second-order proximity. Thus, given a finite dataset, the efficient inference algorithm proposed can let the data automatically determine the exact number of communities for embedding and shows promising results for multiple applications, especially for datasets with a noise community structure.

6.1.1 Random Walk Generative Process

In addition to the number of communities, another limitation related to most of the random walk-based models (Ch. 2.3.1) is the inability of guiding the path sampling process based on the embedding performances. Node2Vec [53] proposed to bias the truncated path generative process to better explore the graphs and force a trade-off between a deep-first search or a breadth-first search behavior. However, such an approach is not advised by the embedding process, but it is expensive to compute and require manual fine-tuning. Other approaches have been proposed to carefully construct the walks to preserve the sub-graph structures better [91, 92]. However, we claim that to properly solve such a problem an end-to-end approach is required.

Neural Graph Model. Inspired by the flourishing research on neural language model [26, 34, 69, 70] in which an RNN is used to learn the world generative process of a given language, we propose the new concept of neural graph model (NGM). As shown in Fig. 6.1, given a set of path $\mathbf{P}$ generated from a graph $G(V, E)$, NGM employs the same RNN architecture to learn the graph generative process.

In particular, suppose that we are given a path $[v_1, \dots, v_{i-1}, v_i, \dots, v_N]$, then it is possible to formally define the learning process of the NGM, shown in Fig. 6.1, as follows:

$$\mathbf{h}^i = f(\phi_{i-1}, \mathbf{h}^{i-1}) \quad (6.1)$$

$$y_i = \text{Softmax}(\mathbf{W}\mathbf{h}^i) \quad (6.2)$$

where $f(\cdot, \cdot)$ in Eq. 6.1 represents the LSTM or GRU function; $\mathbf{W}$ represents a linear transformation; ϕ_{i-1} is the embedding associated with the i -th -1 node in the truncated random walk; and y_i is the predicted probability distribution of the i -th

Figure 6.1: Given a graph $G(V, E)$, it is possible to extract some fixed length paths, for example $[1, 4, 6, 9]$. The obtained paths are then used to train a NGM in an unsupervised way to obtain a meaningful graph embedding. Note that in Fig. 6.1b, we used a square shape to represent the nodes' vectorial representation rather than the node itself.

node in the path. In a traditional neural language model setting, such a network is trained using the cross-entropy loss between the *predicted node distribution* y_i and the *one-hot target distribution* representative of the i -th vertex v_i (we will refer to such target distribution as $\hat{v}_i$):

$$\mathcal{L}_{CE}(\hat{v}_i, y_i) = - \sum_{k \in V} \hat{v}_i[k] \log y_i[k] \quad (6.3)$$

where $y_i[k]$ stands for the k -th dimension of the vector y_i . However, such a learning approach is still based on randomly sampled paths to preserve the graph structure.

Differentiable routing. To overcome such limitations, we further leveraged the recent CapsuleNet architecture [118] to achieve a differentiable dynamic routing process that allowed us to train our model in an end-to-end manner. A node from the same neighborhood has to have a similar vector representation; therefore, at each step, an ideal embedding model aims to accost the neighbor with the most different embedding w.r.t. the current node. For example, as shown in Fig. 6.2, suppose that we are currently computing the node embedding for vertex v_1 . To better preserve the graph structure, the model should focus on the juxtaposition between vertex v_1 and v_3 because v_3 is the most different neighbor. Thus, we can formulate the training loss (Eq. 6.3) according to the above-mentioned routing principle as follows:

$$v_j = \max_{j \in N_i} \Delta(\phi_i, \phi_j) \quad (6.4)$$

$$\mathcal{L}_{CE}(\hat{v}_j, y_j) \quad (6.5)$$

where $\Delta(\cdot, \cdot)$ represents any difference metric and v_j is the most different neighbor of vertex v_i according to Δ . Unfortunately, Eq. 6.4 is not differentiable; however, inspired by the dynamic routing process proposed in [118], we can derive a soft formulation of such a routing procedure. In particular, we focus on deriving a differentiable formulation for the target node distribution that incorporates our routing

Figure 6.2: Given a graph $G(V, E)$ in Fig. 6.2a and a node embedding in Fig. 6.2b: at each step, the embedding model should accost the nodes with the most different embedding representation.

principle. At step i on the random walk, let us define the weight function:

$$\mathbf{w}_{N_i} = \begin{cases} 1 - \sigma(\phi_j^T \phi_i) & \text{if } j \in N_i, \\ 0 & \text{otherwise.} \end{cases} \quad (6.6)$$

where $\sigma(\cdot) = \frac{1}{1+e^{-x}}$ and $\mathbf{w}_i$ represents a weight dissimilarity vector $\mathbb{R}^{V \times 1}$. Then, we can compute a routing-aware neighbor distribution as follows:

$$\Phi_{N_i} = \sum_{i \in V} (\Phi \odot \mathbf{w}_{N_i}) \quad (6.7)$$

$$\hat{v}_j^* = \text{Softmax}(\Phi \Phi_{N_i}^T) \quad (6.8)$$

where $\odot$ represents the element-wise multiplication; Φ is the entire node embedding; Φ_{N_i} is a neighbor embedding $\mathbb{R}^{1 \times d}$; and $\hat{v}_i^*$ is the target distribution that we aim to approximate in Eq. 6.2. Note that the matrix product $\Phi \Phi_{N_i}^T$ is an unnormalized probability distribution of the estimated neighbors of the vertex v_i according to the node embedding. With such a formulation, the NGM can be trained end-to-end using the KL-divergence between y_i and $\hat{v}_i^*$ as follows:

$$\mathcal{L}_{KL}(\hat{v}_j^*, y_j) = - \sum_{k \in V} \hat{v}_j^*[k] \log \frac{y_j[k]}{\hat{v}_j^*[k]}$$

Note that we proposed to use the KL-divergence rather than the cross-entropy to better preserve the difference between the two probability distributions. Such a procedure guarantees to soft-route the sampling process to the most different neighbors so as to better preserve the graph structure.

6.1.2 Distance-aware Graph Embedding for Semi-supervised Learning

Another limitation related to most random walk methods that use the second-order proximity is the selection of parameter ζ , namely, the context size. Such fixed and manually tuned context size for graph embedding has a significant limitation, i.e.,

it overlooks the distance between a context node to its target node. Therefore, it is unable to truly reflect the importance of a node being a context for a target node. In other words, the random sampling does not select the right context for each target node. Instead, a more accurate approach is to automatically learn how to select the right context for each target node by context weighting. In particular, it is possible to model the distance from a target node v_i to each context node v_j with the stationary probability of walking from v_i to v_j on the graph, where the graph edges are weighted and parameterized. We then jointly learn the edge weight parameters and the node embedding from supervision.

In such a context, given a graph $G = (V, E)$, we denote the distance between two nodes v_i and v_j as $\delta_{i,j}$. Note that $e_{i,j}$ may not necessarily be in E . $\delta_{i,j}$ is then interpreted as the stationary probability of walking from v_i to v_j on the graph. We will discuss how to define $\delta_{i,j}$ later. For now, let us assume $\delta_{i,j}$ is given. Then, it is possible to redefine the **first-order** proximity as follows:

$$O_1 = - \sum_{(v_i, v_j) \in E} \log \delta_{i,j} \sigma(\phi_j^T \phi_i) \quad (6.9)$$

and the **second-order** proximity as:

$$\begin{aligned} \Delta_{ij} &= \log \sigma(\phi_j^T \phi_i) + \sum_{t=1}^m \mathbb{E}_{v_l \sim P_n(v_i)} [\log \sigma(-\phi_l^T \phi_i)] \\ O_2 &= -\alpha \sum_{v_i \in V} \sum_{v_j \in C_i} \delta_{i,j} \Delta_{ij} \end{aligned} \quad (6.10)$$

Distance modeling. As stated above, we define $\delta_{i,j}$ as the stationary probability of walking from v_i to v_j on the graph. In particular, we denote $\boldsymbol{\delta}_i = [\delta_{i,1}, \dots, \delta_{i,N}]$ as the stationary probability vector of walking from v_i to all the N nodes on the graph. For an $e_{i,j} \in E$, we define its weight as follows:

$$\alpha_{i,j} = \sigma(\mathbf{w}^T \mathbf{f}_{i,j}) \quad (6.11)$$

where $\mathbf{f}_{i,j}$ is a feature vector for edge $e_{i,j}$ and $\mathbf{w}$ is a feature weight vector. Then, we define the *transition matrix* for a random walk on the graph as follows:

$$p_{i,j} = (1 - \alpha) p'_{i,j} + \alpha \mathbb{1}[i] \quad (6.12)$$

where $\alpha \in [0, 1]$ is a restart probability and $\mathbb{1}[i]$ is a length- N vector with its i -th dimension as one and the other dimensions as zeros; moreover,

$$p_{i,j} = \begin{cases} \frac{\alpha_{i,j}}{\sum_k \alpha_{i,k}} & \text{if } e_{i,j} \in E, \\ 0 & \text{O.W.} \end{cases}$$

Finally, we define $\boldsymbol{\delta}_i$ as follows:

$$\boldsymbol{\delta}_i^T = \boldsymbol{\delta}_i^T \mathbf{P}. \quad (6.13)$$

Note that we have to run a personalized random walk starting on each node v_i , which can be tedious.

Semi-supervision. In general, we can plug in any supervision to guide the embedding and the distance modeling. Herein, we consider two representative types of supervision based on node classification and link prediction.

For node classification, we have supervision as the labels on the nodes. Suppose there are in total C different classes. For a labeled node v_l , we denote its label as an indicator vector $\mathbf{y}_l \in +1, -1^C$, where $y_l^c = +1$ if v_l has class c . Suppose we have T labeled nodes $\{v_{l_1}, \dots, v_{l_T}\}$, given each $l_t \in \{1, \dots, N\}$. We denote their labels as $L = \{\mathbf{y}_{l_1}, \dots, \mathbf{y}_{l_T}\}$. We then denote a loss function for classification as $h(\phi_l, \mathbf{y}_l; \theta)$, which can be defined in different forms, e.g. Softmax or SVM. Then, we define the node classification objective function as follows:

$$O_3 = \sum_{\mathbf{y}_l \in L} h(\phi_l, \mathbf{y}_l; \theta) \quad (6.14)$$

For link prediction, we can leverage the presence of the real edges in the graph. We denote the labels for link prediction as a set of tuples $L' = \{(v_i, v_j, v_k) \mid e_{i,j} \in E, e_{i,k} \notin E\}$. We denote a loss function for ranking as $h(\phi_i, \phi_j, \phi_k; \theta)$, which can also be defined in different forms, e.g. Euclidean distance. Then, we define the link prediction objective function as follows:

$$O_3 = \sum_{(v_i, v_j, v_k) \in L'} h(\phi_i, \phi_j, \phi_k; \theta). \quad (6.15)$$

Finally, let us denote $\Theta = \{\mathbf{w}, \theta\}$ as the parameter set; we aim to minimize:

$$\mathcal{L} = O_1(\Phi, \Theta) + \lambda_1 O_2(\Phi, \Phi', \Theta) + \lambda_2 O_3(\Phi, \Theta) \quad (6.16)$$

Note that such loss can be first learned by coordinated stochastic gradient descent on (Φ, Φ') and then on Θ ; thus, in this manner, we learn to weight the different context size (gradient w.r.t. $\mathbf{w}$) and the optimal Φ and Φ' .

6.2 DIAN Limitations

Finally, in Ch. 5, we proposed a model that is able to learn the dynamic influence that a neighbor exerts on the node's attributes to better understand the evolution of a complex graph structure. The proposed neural architecture has been proven useful to align the time series of adjacent nodes and infer the edge weight and provide a human-interpretable graph structure; however, it cannot scale to high degree graphs.

Recently, researchers have placed considerable efforts in the development of linear attention formulations, which could scale to bigger problem settings. Combining the advantages of the transformer neural architecture [137] with the fast-weight RNN [5] would provide a scalable solution. However, such architecture is not yet proposed in the current literature.

Bibliography

- [1] Agarwal, A. and Chakrabarti, S. (2007). Learning random walks to rank nodes in graphs. In *Proceedings of the 24th international conference on Machine learning*, pages 9–16. ACM.
- [2] Agarwal, A., Chakrabarti, S., and Aggarwal, S. (2006). Learning to rank networked entities. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 14–23. ACM.
- [3] Ahmed, A., Shervashidze, N., Narayanamurthy, S., Josifovski, V., and Smola, A. J. (2013). Distributed large-scale natural graph factorization. In *WWW*, pages 37–48.
- [4] Aridhi, S., Montresor, A., and Velegrakis, Y. (2017). Bladyg: A graph processing framework for large dynamic graphs. *Big Data Research*, 9:9–17.
- [5] Ba, J., Hinton, G., Mnih, V., Leibo, J. Z., and Ionescu, C. (2016a). Using fast weights to attend to the recent past. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS’16, pages 4338–4346, USA. Curran Associates Inc.
- [6] Ba, L. J., Kiros, R., and Hinton, G. E. (2016b). Layer normalization. *CoRR*, abs/1607.06450.
- [7] Backstrom, L. and Leskovec, J. (2011). Supervised random walks: predicting and recommending links in social networks. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 635–644. ACM.
- [8] Balasubramanian, M. and Schwartz, E. L. (2002). The isomap algorithm and topological stability. *Science*, 295(5552):7–7.
- [9] Battaglia, P., Pascanu, R., Lai, M., Rezende, D. J., and others (2016). Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems*, pages 4502–4510.
- [10] Belkin, M. and Niyogi, P. (2001). Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*, pages 585–591.
- [11] Belkin, M. and Niyogi, P. (2003). Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396.

- [12] Biernacki, C. and Chretien, S. (2003). Degeneracy in the maximum likelihood estimation of univariate Gaussian mixtures with EM. *Statistics & Probability Letters*, 61(4):373–382.
- [13] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., NJ, USA.
- [14] Blei, D. M. and Jordan, M. I. (2006). Variational inference for dirichlet process mixtures. *Bayesian Analysis*, 1(1):121–144.
- [15] Bollobas, B. (1998). *Modern Graph Theory*. Springer.
- [16] Bordes, A., Usunier, N., García-Durán, A., Weston, J., and Yakhnenko, O. (2013a). Translating embeddings for modeling multi-relational data. In *NIPS*, pages 2787–2795.
- [17] Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., and Yakhnenko, O. (2013b). Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795.
- [18] Bordes, A., Weston, J., Collobert, R., and Bengio, Y. (2011). Learning structured embeddings of knowledge bases. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*.
- [19] Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. (2013). Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*.
- [20] Cai, D., He, X., and Han, J. (2007). Spectral regression: a unified subspace learning framework for content-based image retrieval. In *Proceedings of the 15th ACM international conference on Multimedia*, pages 403–412. ACM.
- [21] Cai, H., Zheng, V. W., and Chang, K. C. (2017a). A comprehensive survey of graph embedding: Problems, techniques and applications. *CoRR*, abs/1709.07604.
- [22] Cai, H., Zheng, V. W., Zhu, F., Chang, K. C., and Huang, Z. (2017b). From community detection to community profiling. *PVLDB*, 10(7):817–828.
- [23] Cao, S., Lu, W., and Xu, Q. (2015). Grarep: Learning graph representations with global structural information. In *CIKM*, pages 891–900.
- [24] Cao, S., Lu, W., and Xu, Q. (2016). Deep neural networks for learning graph representations. In *AAAI*, pages 1145–1152.
- [25] Cavallari, S., Zheng, V. W., Cai, H., Chang, K. C., and Cambria, E. (2017). Learning community embedding with community detection and node embedding on graphs. In *CIKM*, pages 377–386.

- [26] Chan, W., Jaitly, N., Le, Q., and Vinyals, O. (2016). Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 4960–4964. IEEE.
- [27] Chang, C.-C. and Lin, C.-J. (2011). Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27.
- [28] Chang, S., Han, W., Tang, J., Qi, G., Aggarwal, C. C., and Huang, T. S. (2015). Heterogeneous network embedding via deep architectures. In *KDD*, pages 119–128.
- [29] Chen, J., Ma, T., and Xiao, C. (2018). Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*.
- [30] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.
- [31] Chung, F. R. and Graham, F. C. (1997). *Spectral graph theory*. Number 92. American Mathematical Soc.
- [32] Cireşan, D., Meier, U., and Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. *arXiv preprint arXiv:1202.2745*.
- [33] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition.
- [34] Daniluk, M., Rocktäschel, T., Welbl, J., and Riedel, S. (2017). Frustratingly short attention spans in neural language modeling. *arXiv preprint arXiv:1702.04521*.
- [35] Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852.
- [36] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38.
- [37] Dong, Y., Chawla, N. V., and Swami, A. (2017). metapath2vec: Scalable representation learning for heterogeneous networks. In *KDD*, pages 135–144.
- [38] Duan, Y., Andrychowicz, M., Stadie, B. C., Ho, J., Schneider, J., Sutskever, I., Abbeel, P., and Zaremba, W. (2017). One-Shot Imitation Learning. *arXiv:1703.07326 [cs]*. arXiv: 1703.07326.
- [39] Elser, B. and Montresor, A. (2013). An evaluation study of bigdata frameworks for graph processing. In *Big Data, 2013 IEEE International Conference on*, pages 60–67. IEEE.

- [40] Evans, T. S. (2010). Clique graphs and overlapping communities. *Journal of Statistical Mechanics: Theory and Experiment*, 2010(12):P12037.
- [41] Fang, H., Wu, F., Zhao, Z., Duan, X., Zhuang, Y., and Ester, M. (2016a). Community-based question answering via heterogeneous social network learning. In *Thirtieth AAAI Conference on Artificial Intelligence*.
- [42] Fang, Y., Lin, W., Zheng, V. W., Wu, M., Chang, K. C., and Li, X. (2016b). Semantic proximity search on graphs with metagraph-based learning. In *ICDE*, pages 277–288.
- [43] Fout, A., Byrd, J., Shariat, B., and Ben-Hur, A. (2017). Protein interface prediction using graph convolutional networks. In *Advances in Neural Information Processing Systems*, pages 6530–6539.
- [44] Fragkiadaki, K., Levine, S., Felsen, P., and Malik, J. (2015). Recurrent network models for human dynamics. In *Computer Vision (ICCV), 2015 IEEE International Conference on*, pages 4346–4354. IEEE.
- [45] Garcia Duran, A. and Niepert, M. (2017). Learning graph representations with embedding propagation. In *NIPS*, pages 5125–5136.
- [46] Geng, X., Zhang, H., Bian, J., and Chua, T.-S. (2015). Learning image and user features for recommendation in social networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4274–4282.
- [47] George, B. and Shekhar, S. (2008). Time-aggregated graphs for modeling spatio-temporal networks. In *Journal on Data Semantics XI*, pages 191–212. Springer.
- [48] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*.
- [49] Girvan, M. and Newman, M. E. (2002). Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826.
- [50] Golub, G. H. and Reinsch, C. (1970). Singular value decomposition and least squares solutions. *Numerische mathematik*, 14(5):403–420.
- [51] Gonzalez, J. E., Xin, R. S., Dave, A., Crankshaw, D., Franklin, M. J., and Stoica, I. (2014). Graphx: Graph processing in a distributed dataflow framework. In *OSDI*, volume 14, pages 599–613.
- [52] Graves, A. and Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks. In *International Conference on Machine Learning*, pages 1764–1772.
- [53] Grover, A. and Leskovec, J. (2016). node2vec: Scalable feature learning for networks. In *KDD*.

- [54] Guo, S., Wang, Q., Wang, B., Wang, L., and Guo, L. (2017). Sse: Semantically smooth embedding for knowledge graphs. *IEEE Transactions on Knowledge and Data Engineering*, 29(4):884–897.
- [55] Gutmann, M. U. and Hyvärinen, A. (2012). Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research*, 13(Feb):307–361.
- [56] Hamaguchi, T., Oiwa, H., Shimbo, M., and Matsumoto, Y. (2017). Knowledge transfer for out-of-knowledge-base entities: a graph neural network approach. *arXiv preprint arXiv:1706.05674*.
- [57] Hamilton, W., Ying, Z., and Leskovec, J. (2017). Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034.
- [58] Han, Y. and Tang, J. (2015). Probabilistic community and role model for social networks. In *KDD*, pages 407–416.
- [59] Handcock, M. S., Raftery, A. E., and Tantrum, J. M. (2007). Model-based clustering for social networks. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 170(2):301–354.
- [60] He, L., Lu, C.-T., Ma, J., Cao, J., Shen, L., and Yu, P. S. (2016). Joint community and structural hole spanner detection via harmonic modularity. In *KDD*.
- [61] He, S., Liu, K., Ji, G., and Zhao, J. (2015). Learning to represent knowledge graphs with gaussian embedding. In *CIKM*, pages 623–632. ACM.
- [62] He, X., Cai, D., Shao, Y., Bao, H., and Han, J. (2011). Laplacian regularized gaussian mixture model for data clustering. *IEEE Trans. Knowl. Data Eng.*, 23(9):1406–1418.
- [63] He, X. and Niyogi, P. (2004). Locality preserving projections. In *Advances in neural information processing systems*, pages 153–160.
- [64] Hofmann, T. and Buhmann, J. (1995). Multidimensional scaling and data clustering. In *Advances in Neural Information Processing Systems*, pages 459–466.
- [65] Hoshen, Y. (2017). Vain: Attentional multi-agent predictive modeling. In *Advances in Neural Information Processing Systems*, pages 2701–2711.
- [66] Jain, A., Zamir, A. R., Savarese, S., and Saxena, A. (2016). Structural-RNN: Deep learning on spatio-temporal graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5308–5317.
- [67] Ji, G., He, S., Xu, L., Liu, K., and Zhao, J. (2015). Knowledge graph embedding via dynamic mapping matrix. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint*

- Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 687–696.
- [68] Ji, G., Liu, K., He, S., and Zhao, J. (2016). Knowledge graph completion with adaptive sparse transfer matrix. In *AAAI*, pages 985–991.
  - [69] Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., and Wu, Y. (2016). Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*.
  - [70] Kim, Y., Jernite, Y., Sontag, D., and Rush, A. M. (2015). Character-aware neural language models. *arXiv preprint arXiv:1508.06615*.
  - [71] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
  - [72] Kipf, T. N., Fetaya, E., Wang, K., Welling, M., and Zemel, R. S. (2018). Neural relational inference for interacting systems. *CoRR*, abs/1802.04687.
  - [73] Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907.
  - [74] Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *ICLR*.
  - [75] Kloster, K. and Gleich, D. F. (2014). Heat kernel based community detection. In *KDD*, pages 1386–1395.
  - [76] Kozdoba, M. and Mannor, S. (2015). Community detection via measure space embedding. In *NIPS*, pages 2890–2898.
  - [77] Lai, Y.-A., Hsu, C.-C., Chen, W. H., Yeh, M.-Y., and Lin, S.-D. (2017). Prune: Preserving proximity and global ranking for network embedding. In *NIPS*, pages 5263–5272.
  - [78] Landrieu, L. and Simonovsky, M. (2017). Large-scale point cloud semantic segmentation with superpoint graphs. *arXiv preprint arXiv:1711.09869*.
  - [79] Le, H. M., Yue, Y., and Carr, P. (2017). Coordinated multi-agent imitation learning. *arXiv preprint arXiv:1703.03121*.
  - [80] Lee, D. D. and Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788.
  - [81] Lee, J. B., Rossi, R., and Kong, X. (2018). Graph classification using structural attention. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1666–1674. ACM.
  - [82] Lezama, J., Alahari, K., Sivic, J., and Laptev, I. (2011). Track to the future: Spatio-temporal video segmentation with long-range motion cues. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3369–3376. IEEE.

- [83] Li, C., Ma, J., Guo, X., and Mei, Q. (2017). Deepcas: An end-to-end predictor of information cascades. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 577–586, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.
- [84] Li, Y. and Nevatia, R. (2008). Key object driven multi-category object recognition, localization and tracking using spatio-temporal context. In *ECCV (4)*, pages 409–422.
- [85] Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. S. (2015). Gated graph sequence neural networks. *CoRR*, abs/1511.05493.
- [86] Liang, X., Shen, X., Feng, J., Lin, L., and Yan, S. (2016). Semantic object parsing with graph lstm. In *European Conference on Computer Vision*, pages 125–143. Springer.
- [87] Lin, Y., Liu, Z., Sun, M., Liu, Y., and Zhu, X. (2015). Learning entity and relation embeddings for knowledge graph completion. In *AAAI*, volume 15, pages 2181–2187.
- [88] Lin, Y.-Y., Liu, T.-L., and Chen, H.-T. (2005). Semantic manifold learning for image retrieval. In *Proceedings of the 13th annual ACM international conference on Multimedia*, pages 249–258. ACM.
- [89] Liu, W., Zheng, Y., Chawla, S., Yuan, J., and Xing, X. (2011). Discovering spatio-temporal causal interactions in traffic data streams. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1010–1018. ACM.
- [90] Liu, X., Luo, Z., and Huang, H. (2018a). Jointly multiple events extraction via attention-based graph information aggregation. *arXiv preprint arXiv:1809.09078*.
- [91] Liu, Z., Zheng, V. W., Zhao, Z., Yang, H., Chang, K. C.-C., Wu, M., and Ying, J. (2018b). Subgraph-augmented path embedding for semantic user search on heterogeneous social network. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, pages 1613–1622, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.
- [92] Liu, Z., Zheng, V. W., Zhao, Z., Zhu, F., Chang, K. C., Wu, M., and Ying, J. (2017a). Semantic proximity search on heterogeneous graph by proximity embedding. In *AAAI*, pages 154–160.
- [93] Liu, Z., Zheng, V. W., Zhao, Z., Zhu, F., Chang, K. C.-C., Wu, M., and Ying, J. (2017b). Semantic proximity search on heterogeneous graph by proximity embedding. In *AAAI*, pages 154–160.
- [94] Low, Y., Bickson, D., Gonzalez, J., Guestrin, C., Kyrola, A., and Hellerstein, J. M. (2012). Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 5(8):716–727.

- [95] Lu, H. and Kong, M. (2017). Community-based question answering via contextual ranking metric network learning. In *AAAI*, pages 4963–4964.
- [96] Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., and Czajkowski, G. (2010). Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM.
- [97] Man, T., Shen, H., Liu, S., Jin, X., and Cheng, X. (2016). Predict anchor links across social networks via an embedding approach. In *IJCAI*, volume 16, pages 1823–1829.
- [98] Marcheggiani, D. and Titov, I. (2017). Encoding sentences with graph convolutional networks for semantic role labeling. *arXiv preprint arXiv:1703.04826*.
- [99] Masci, J., Rodolà, E., Boscaini, D., Bronstein, M. M., and Li, H. (2016). Geometric deep learning. In *SIGGRAPH ASIA 2016 Courses*, page 1. ACM.
- [100] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.
- [101] Mikolov, T., Karafiát, M., Burget, L., Černocký, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*.
- [102] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119.
- [103] Morin, F. and Bengio, Y. (2005). Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pages 246–252. Citeseer.
- [104] Niepert, M., Ahmed, M., and Kutzkov, K. (2016). Learning convolutional neural networks for graphs. In *ICML*, pages 2014–2023.
- [105] Nikolentzos, G., Meladianos, P., and Vazirgiannis, M. (2017). Matching node embeddings for graph similarity. In *AAAI*, pages 2429–2435.
- [106] Ou, M., Cui, P., Pei, J., Zhang, Z., and Zhu, W. (2016). Asymmetric transitivity preserving graph embedding. In *KDD*, pages 1105–1114.
- [107] Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab.
- [108] Pan, S., Wu, J., Zhu, X., Zhang, C., and Wang, Y. (2016). Tri-party deep network representation. *Network*, 11(9):12.
- [109] Peel, L., Larremore, D. B., and Clauset, A. (2017). The ground truth about metadata and community detection in networks. *Science advances*, 3(5):e1602548.
- [110] Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). Deepwalk: Online learning of social representations. In *KDD*, pages 701–710.

- [111] Price, D. d. S. (1976). A general theory of bibliometric and other cumulative advantage processes. *Journal of the American society for Information science*, 27(5):292–306.
- [112] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2018). Language models are unsupervised multitask learners.
- [113] Raftery, A. E., Niu, X., Hoff, P. D., and Yeung, K. Y. (2012). Fast inference for the latent space network model using a case-control approximate likelihood. *J. of Computational and Graphical Stats*, 21(4):901–919.
- [114] Rasmussen, C. E. (2000). The infinite gaussian mixture model. In *NIPS*, pages 554–560.
- [115] Rhee, S., Seo, S., and Kim, S. (2017). Hybrid approach of relation network and localized graph convolutional filtering for breast cancer subtype classification. *arXiv preprint arXiv:1711.05859*.
- [116] Ribeiro, L. F., Saverese, P. H., and Figueiredo, D. R. (2017). Struc2vec: Learning node representations from structural identity. In *KDD*, pages 385–394.
- [117] Roweis, S. T. and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326.
- [118] Sabour, S., Frosst, N., and Hinton, G. E. (2017). Dynamic routing between capsules. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 3859–3869.
- [119] Sachan, M., Dubey, A., Srivastava, S., Xing, E. P., and Hovy, E. (2014). Spatial compactness meets topical consistency: Jointly modeling links and content for community detection. In *WSDM*, pages 503–512.
- [120] Santoro, A., Raposo, D., Barrett, D. G., Malinowski, M., Pascanu, R., Battaglia, P., and Lillicrap, T. (2017). A simple neural network module for relational reasoning. *arXiv preprint arXiv:1706.01427*.
- [121] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2009). The graph neural network model. *Trans. Neur. Netw.*, 20(1):61–80.
- [122] Sengupta, D., Sundaram, N., Zhu, X., Willke, T. L., Young, J., Wolf, M., and Schwan, K. (2016). Graphin: An online high performance incremental graph processing framework. In *European Conference on Parallel Processing*, pages 319–333. Springer.
- [123] Shervashidze, N., Schweitzer, P., Leeuwen, E. J. v., Mehlhorn, K., and Borgwardt, K. M. (2011a). Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561.

- [124] Shervashidze, N., Schweitzer, P., van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. (2011b). Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12:2539–2561.
- [125] Sukhbaatar, S., Szlam, A., and Fergus, R. (2016). Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 2244–2252.
- [126] Sun, Y., Aggarwal, C. C., and Han, J. (2012). Relation strength-aware clustering of heterogeneous information networks with incomplete attributes. *PVLDB*, 5(5):394–405.
- [127] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.
- [128] Suwan, S., Lee, D. S., Tang, R., Sussman, D. L., Tang, M., Priebe, C. E., et al. (2016). Empirical bayes estimation for the stochastic blockmodel. *Electronic Journal of Statistics*, 10(1):761–782.
- [129] Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., and Mei, Q. (2015). Line: Large-scale information network embedding. In *WWW*, pages 1067–1077.
- [130] Tang, L. and Liu, H. (2011). Leveraging social media networks for classification. *Data Min. Knowl. Discov.*, 23(3):447–478.
- [131] Tian, F., Gao, B., Cui, Q., Chen, E., and Liu, T. (2014). Learning deep representations for graph clustering. In *AAAI*, pages 1293–1299.
- [132] Tu, C., Zhang, W., Liu, Z., Sun, M., et al. (2016). Max-margin deepwalk: Discriminative learning of network representation. In *IJCAI*, pages 3889–3895.
- [133] Tu, C., Zhang, Z., Liu, Z., and Sun, M. (2017). Transnet: Translation-based network representation learning for social relation extraction. In *IJCAI*, pages 2864–2870.
- [134] van der Maaten, L. and Hinton, G. (2008). Visualizing high-dimensional data using t-sne. *JMLR*, 9:2579–2605.
- [135] van Steenkiste, S., Chang, M., Greff, K., and Schmidhuber, J. (2018). Relational neural expectation maximization: Unsupervised discovery of objects and their interactions. *CoRR*, abs/1802.10353.
- [136] Vandenberghe, L. and Boyd, S. (1996). Semidefinite programming. *SIAM review*, 38(1):49–95.
- [137] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010.

- [138] Veldt, N., Gleich, D. F., and Wirth, A. (2018). A correlation clustering framework for community detection. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, pages 439–448.
- [139] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.
- [140] Venkatakrisnan, S. B., Alizadeh, M., and Viswanath, P. (2018). Graph2seq: Scalable learning dynamics for graphs. *CoRR*, abs/1802.04948.
- [141] Wang, D., Cui, P., and Zhu, W. (2016). Structural deep network embedding. In *KDD*, pages 1225–1234.
- [142] Wang, J., Zheng, V. W., Liu, Z., and Chang, K. C.-C. (2017a). Topological recurrent neural network for diffusion prediction. In *ICDM*, pages 475–484.
- [143] Wang, M., Wang, C., Yu, J. X., and Zhang, J. (2015). Community detection in social networks: An in-depth benchmarking study with a procedure-oriented framework. *PVLDB*, 8(10):998–1009.
- [144] Wang, X., Cui, P., Wang, J., Pei, J., Zhu, W., and Yang, S. (2017b). Community preserving network embedding. In *AAAI*, pages 203–209.
- [145] Wang, X., Girshick, R., Gupta, A., and He, K. (2018). Non-local neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, page 4.
- [146] Wang, Z., Zhang, J., Feng, J., and Chen, Z. (2014). Knowledge graph embedding by translating on hyperplanes. In *AAAI*, volume 14, pages 1112–1119.
- [147] Watts, D. J. and Strogatz, S. H. (1998). Collective dynamics of ‘small-world’ networks. *nature*, 393(6684):440.
- [148] Wei, X., Xu, L., Cao, B., and Yu, P. S. (2017). Cross view link prediction by learning noise-resilient representation consensus. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1611–1619. International World Wide Web Conferences Steering Committee.
- [149] Weinberger, K. Q., Sha, F., and Saul, L. K. (2004). Learning a kernel matrix for nonlinear dimensionality reduction. In *Proceedings of the twenty-first international conference on Machine learning*, page 106. ACM.
- [150] Xiao, H., Huang, M., Meng, L., and Zhu, X. (2017). Ssp: Semantic space projection for knowledge graph embedding with text descriptions. In *AAAI*, volume 17, pages 3104–3110.
- [151] Xiao, H., Huang, M., and Zhu, X. (2016). Transg: A generative model for knowledge graph embedding. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 2316–2325.

- [152] Yang, B., Yih, W.-t., He, X., Gao, J., and Deng, L. (2014). Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*.
- [153] Yang, C., Liu, Z., Zhao, D., Sun, M., and Chang, E. Y. (2015). Network representation learning with rich text information. In *IJCAI*, pages 2111–2117.
- [154] Yang, L., Cao, X., He, D., Wang, C., Wang, X., and Zhang, W. (2016a). Modularity based community detection with deep learning. In *IJCAI*, pages 2252–2258.
- [155] Yang, Z., Cohen, W. W., and Salakhutdinov, R. (2016b). Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861*.
- [156] Yang, Z., Tang, J., and Cohen, W. W. (2016c). Multi-modal bayesian embeddings for learning social knowledge graphs. In *IJCAI*, pages 2287–2293.
- [157] Zachary, W. W. (1977). An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33(4):452–473.
- [158] Zhang, H., Giles, C. L., Foley, H. C., and Yen, J. (2007). Probabilistic community discovery using hierarchical latent gaussian mixture model. In *AAAI*, volume 7, pages 663–668.
- [159] Zhang, H., Shang, X., Luan, H., Wang, M., and Chua, T.-S. (2017). Learning from collective intelligence: Feature learning using social images and tags. *ACM transactions on multimedia computing, communications, and applications (TOMM)*, 13(1):1.
- [160] Zhang, Q. and Wang, H. (2016). Not all links are created equal: An adaptive embedding approach for social personalized ranking. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 917–920. ACM.
- [161] Zhang, Z., Cui, P., and Zhu, W. (2018). Deep learning on graphs: A survey. *arXiv preprint arXiv:1812.04202*.
- [162] Zhao, X., Chang, A., Sarma, A. D., Zheng, H., and Zhao, B. Y. (2013). On the embeddability of random walk distances. *PVLDB*, 6(14):1690–1701.
- [163] Zhao, Z., Yang, Q., Cai, D., He, X., and Zhuang, Y. (2016). Expert finding for community-based question answering via ranking metric network learning. In *IJCAI*, pages 3000–3006.
- [164] Zhou, C., Liu, Y., Liu, X., Liu, Z., and Gao, J. (2017). Scalable graph embedding for asymmetric proximity. In *AAAI*, pages 2942–2948.

Appendix A

List of Publications

- **Sandro Cavallari**, Vincent W Zheng, Hongyun Cai, Soujanya Poria and Erik Cambria. “An Attention-Based Model for Learning Dynamic Interaction Networks.” International Joint Conference on Neural Networks (IJCNN), 2019.
- Ranjan Satapathy, Yang Li, **Sandro Cavallari** and Erik Cambria. “Seq2Seq Deep Learning Models for Microtext Normalization.” International Joint Conference on Neural Networks (IJCNN), 2019.
- Iti Chaturvedi, **Sandro Cavallari**, Erik Cambria and Roy Welsch. “Genetic Programming for Domain Adaptation in Product Reviews.” 2019-20th International Conference on Computational Linguistics and Intelligent Text Processing (CICLing), 2019.
- **Sandro Cavallari**, Vincent W Zheng, Hongyun Cai, Kevin Chen-Chuan Chang and Erik Cambria. “Learning community embedding with community detection and node embedding on graphs.” ACM on Conference on Information and Knowledge Management (CIKM), 2017.
- Iti Chaturvedi, **Sandro Cavallari**, Erik Cambria and Vincent W Zheng. “Learning Word Vectors in Deep Walk Using Convolution.” In FLAIRS conference, 2017.
- Vincent W Zheng, **Sandro Cavallari**, Hongyun Cai, Kevin Chen-Chuan Chang and Erik Cambria. “From node embedding to community embedding.” KDD WISDOM, 2016.

List of Submission

- **Sandro Cavallari**, Erik Cambria, Hongyun Cai, Kevin Chen-Chuan Chang and Vincent W Zheng. “Embedding Both Finite and Infinite Communities on Graphs”. IEEE Computational Intelligence Magazine (CIM), 2019. Under publication.
- Iti Chaturvedi, Ranjan Satapathy, **Sandro Cavallari** and Erik Cambria. “Fuzzy Commonsense Reasoning for Multimodal Sentiment Analysis”. Pattern Recognition Letters, Under review.

Appendix B

Proof of Convergence

We will try to provide a convergency proof by induction, assuming we want to demonstrate that the final $\Phi^*, \Phi'^*, \Pi^*, \Psi^*, \Sigma^*$ leads to minimum $\mathcal{L}^{ComE}(\Phi^*, \Phi'^*, \Pi^*, \Psi^*, \Sigma^*)$.

1. Fix $\Phi^{(t)}, \Phi'^{(t)}$, if we optimize Π, Ψ, Σ we have the following:

$$\begin{aligned} & \mathcal{L}^{ComE}(\Phi^{(t)}, \Phi'^{(t)}, \Pi^{(t+1)}, \Psi^{(t+1)}, \Sigma^{(t+1)}) \\ & \leq \\ & \mathcal{L}^{ComE}(\Phi^{(t)}, \Phi'^{(t)}, \Pi^{(t)}, \Psi^{(t)}, \Sigma^{(t)}) \end{aligned} \tag{6.17}$$

because of the EM algorithm used by the GMM, which is known to converge. Thus, given $\Pi^{(t+1)}, \Psi^{(t+1)}, \Sigma^{(t+1)}$, $\mathcal{L}^{ComE}$ decreases.

2. Fix $\Pi^{(t+1)}, \Psi^{(t+1)}, \Sigma^{(t+1)}$, we optimize Φ, Φ' . Due to the SGD computation, we can ensure the following:

$$\begin{aligned} & \mathcal{L}^{ComE'}(\Phi^{(t+1)}, \Phi'^{(t+1)}, \Pi^{(t+1)}, \Psi^{(t+1)}, \Sigma^{(t+1)}) \\ & \leq \\ & \mathcal{L}^{ComE'}(\Phi^{(t)}, \Phi'^{(t)}, \Pi^{(t+1)}, \Psi^{(t+1)}, \Sigma^{(t+1)}) \end{aligned} \tag{6.18}$$

However, $\mathcal{L}^{ComE'}(\Phi^{(t+1)}, \Phi'^{(t+1)}, \Pi^{(t+1)}, \Psi^{(t+1)}, \Sigma^{(t+1)})$ is just an upper bound of $\mathcal{L}^{ComE}(\Phi^{(t+1)}, \Phi'^{(t+1)}, \Pi^{(t+1)}, \Psi^{(t+1)}, \Sigma^{(t+1)})$, which is used to make the loss tractable. Thus, to guarantee that $\mathcal{L}^{ComE}(\Phi^{(t+1)}, \Phi'^{(t+1)}, \Pi^{(t+1)}, \Psi^{(t+1)}, \Sigma^{(t+1)})$ converge we have to add the following condition:

ALGORITHM 3: Convergence condition

```

1 if  $\mathcal{L}^{ComE'}(\Phi^{(t+1)}, \Phi'^{(t+1)}, \Pi^{(t+1)}, \Psi^{(t+1)}, \Sigma^{(t+1)}) \geq$ 
    $\mathcal{L}^{ComE}(\Phi^{(t+1)}, \Phi'^{(t+1)}, \Pi^{(t+1)}, \Psi^{(t+1)}, \Sigma^{(t+1)})$  then
2   | return  $\Phi^* \leftarrow \Phi^{(t+1)}, \Phi'^* \leftarrow \Phi'^{(t+1)}, \Pi^* \leftarrow \Pi^{(t+1)}, \Psi^* \leftarrow \Psi^{(t+1)}, \Sigma^* \leftarrow \Sigma^{(t+1)}$ 
3 else
4   | foreach  $edge (i, j) \in E$  do
5   |   | SGD on  $\phi_i$  and  $\phi_j$ ;
6   | foreach  $path p \in \mathcal{P}$  do
7   |   | foreach  $v_i$  in path  $p$  do
8   |   |   | SGD on  $\phi_i$ ;
9   |   |   | SGD on  $\phi'_j$ 's within  $\zeta$  hops;
10  | foreach  $node v_i \in V$  do
11  |   | SGD on  $\phi_i$ ;

```

Please note that for line 4 – 11 of Alg. 3, we are just executing the ComE inference algorithm, but such if-then-else statement clearly guarantees that $\mathcal{L}^{ComE}$ is decreasing. Moreover, since $\mathcal{L}^{ComE} \geq 0$, it ensures convergence.