Graph Representation Learning : from Kernel to Neural Networks

Changmin Wu

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2021IPPAX135

Thèse de doctorat

Graph Representation Learning: from Kernel to Neural Networks

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à l'École Polytechnique

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Palaiseau, le 25/11/2021, par

CHANGMIN WU

Composition du Jury :

M^me^ Catuscia Palamidessi Directrice de recherche, INRIA Saclay (COMETE)	Président
M. Dimitrios Gunopulos Professor, University of Athens (Department of Informatics and Telecommunications)	Rapporteur
M. Jiliang Tang Associate Professor, Michigan State University (Computer Science and Engineering Department)	Rapporteur
M^me^ Florence d'Alché-Buc Professor, Télécom Paris (Laboratoire Traitement et Communication de l'Information - LTCI)	Examinateur
M. Stéphane Gaïffas Professor, Université Paris Diderot (Laboratoire de Probabilités, Statistique et Modélisation - LPSM)	Examinateur
M. Nikos Komodakis Assistant Professor, University of Crete (Computer Science Department)	Examinateur
M. Michalis Vazirgiannis Professor, École Polytechnique (Laboratoire d'informatique - LIX)	Directeur de thèse

IMPRINT

Graph Representation Learning: from Kernels to Neural Networks

COLOPHON

This thesis was typeset using LaTeX and the `memoir` documentclass. It is a mixture of the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić[1], and Diego Di Carlo's thesis *Echo-aware signal processing for audio scene analysis*[2], itself based on Aaron Turon's thesis template re-implemented by Friedrich Wiemer[3].

Graphics and plots are made with `Matplotlib`[4] and `Seaborn`[5]. Drawings and schemes are made with `PGF/TikZ`[6] and `draw.io`[7]. The bibliography was processed by `Biblatex`.

1 https://bitbucket.org/amiede/classicthesis
2 https://github.com/Chutlhu/PhD_manuscript
3 https://github.com/pfasante/phd_thesis
4 https://matplotlib.org
5 https://seaborn.pydata.org
6 https://ctan.org/pkg/pgf
7 https://app.diagrams.net

To my father,
who nurtured my taste for science.

ABSTRACT

Graphs are ubiquitous as most real-world data can be naturally represented in the form of graphs. Capturing information from graph-structured data, i.e. graph mining or graph representation learning, has thus long become and remains an important topic. In this dissertation, we present a series of research contributions on subjects of machine learning on graphs using kernel methods and the emerging graph neural networks (GNNs).

In the first part, we present a novel framework for constructing a valid optimal assignment graph kernel that computes the similarity between two graphs by computing a correspondence of their node embeddings residing in the same space. Using a clustering algorithm, we construct a hierarchy of the vertices in this space, through which the proposed kernel can find an optimal matching of the vertices that maximises the overall similarity of all the pairs. This framework is not limited to graphs. It can be used to compare any objects that are set of vectors. Moreover, the kernel feature map is a more expressive node embedding than the original embedding methods. We demonstrate the efficiency of the proposed kernel empirically on graph classification, link prediction and text categorisation tasks.

The second part of this dissertation is devoted to the GNNs, particularly the Message-Passing Neural Networks (MPNNs), a dominating class of GNNs. As an emerging model, the MPNN soon became a leading tool for graph representation learning, mainly due to its power of projecting attributed graphs into high-level embeddings, making it versatile to different types of graphs and different application areas.

We first demonstrate the power of MPNNs by an application in the field of temporal networks. We tackle the evolution prediction of dynamic graphs by proposing a sequential framework. Precisely, we use MPNNs to encode the sequence of evolving graphs into a sequence of embeddings in a latent space correlated by time. A recurrent architecture then generates the prediction of embedding at the next timestep. Finally, a generative model is employed to reconstruct the graph instance corresponding to that prediction of embedding in the latent space. GNNs significantly improve the performance against traditional models such as random graph models on predicting the topology of evolving graphs.

The following two works move closer to the fundamentals of MPNNs by addressing their limitations in terms of computational cost and robustness against structural noise. We notice that the MPNN can be divided into two disjoint steps: one is related to the graph structure, namely the aggregation step, and the other only concerns node features, namely the update step. Through extensive experiments, we found that the update step seems to play a less important role in model performance, as it can be substantially simplified by sparsifying, or in some cases, even omitting the whole, as long as the non-linear activation stays.

This finding indicates that the MPNN might be vulnerable to graph-structural noise. Indeed, if the main contribution to model performance comes from the aggregation step, then the impact of structural noise would also be amplified. This work proposes a theoretical model based on random matrix theory to analyse the interaction between graph structure information and node feature information, precisely when graph structure is heavily perturbed. The main result is that graph structural noise will heavily overshadow node feature information. When a graph is structurally perturbed enough, node feature information will have no contribution to model performance. Even itself might be informative. This theoretical finding inspires us to robustify MPNNs against graph structural noise with a node feature kernel. Empirical evaluations show the effectiveness of our proposed kernel as it improves the model performance significantly when the graph structure is heavily perturbed.

RÉSUMÉ EN FRANÇAIS

Les graphes sont omniprésents en tant que structure de données, de par leur capacité à modéliser des informations relationnelles entre objets. La capture d'informations à partir de données structurées en graphes, soit l'exploration de graphes ou l'apprentissage de la représentation de graphes, est donc un sujet important. Dans cette thèse, nous présentons une série de contributions de recherche sur les sujets de l'apprentissage automatique sur les graphes en utilisant des méthodes à noyau et les émergents réseaux neuronaux de graphes (RNGs).

Dans la première partie, nous présentons un nouveau modèle pour construire un noyau de graphe optimal en un certain sens, qui calcule la similarité entre deux graphes en calculant une correspondance de leurs plongements de nœuds résidant dans le même espace. En utilisant un algorithme de clustering, nous construisons une hiérarchie des sommets dans cet espace, à travers laquelle le noyau proposé peut trouver une correspondance optimale des sommets qui maximise la similarité globale de toutes les paires. Ce modèle n'est pas limité aux graphes. Il peut être utilisé pour comparer tout objet qui est un ensemble de vecteurs. De plus, la carte de caractéristiques du noyau est un plongement de nœuds plus expressif que les approches de plongement originales. Nous démontrons l'efficacité du noyau proposé de manière empirique sur des tâches de classification de graphes, de prédiction de liens et de catégorisation de textes.

La deuxième partie de cette thèse est consacrée aux RNG, en particulier aux réseaux de neurones à passage de messages (RNPM), une classe dominante de RNG. En tant que modèle émergent, le RNPM est rapidement devenu un des premiers outils pour l'apprentissage de la représentation de graphes, principalement en raison de sa capacité à projeter des graphes attribués dans des plongements de haut niveau, ce qui le rend polyvalent pour différents types de graphes et différents domaines d'application.

Nous démontrons d'abord la puissance des RNPMs par une application dans le domaine des réseaux temporels. Nous nous attaquons à la prédiction de l'évolution de graphes dynamiques en proposant un modèle séquentiel. Précisément, nous utilisons les RNPMs pour encoder la séquence de graphes en évolution en une séquence de plongements dans un espace latent corrélés par le temps. Une architecture récurrente génère ensuite la prédiction du plongement au prochain pas de temps. Enfin, un modèle génératif est utilisé pour reconstru-

ire l'instance de graphe correspondant à cette prédiction de plongement dans l'espace latent. Les RNGs améliorent considérablement les performances par rapport aux modèles traditionnels, tels que les modèles de graphes aléatoires, pour la prédiction de la topologie des graphes évolutifs.

Les deux travaux suivants se rapprochent des principes fondamentaux des RNPMs en abordant leurs limites en termes de coût de calcul et de robustesse au bruit structurel. Nous remarquons que le RNPM peut être divisé en deux étapes disjointes : l'une est liée à la structure du graphe, à savoir l'étape d'agrégation, et l'autre ne concerne que les caractéristiques des nœuds, à savoir l'étape de mise à jour. Grâce à des expériences extensives, nous avons constaté que l'étape de mise à jour semble jouer un rôle moins important en terme de performance du modèle, car elle peut être considérablement simplifiée en sparsifiant, ou dans certains cas, même en omettant la phase de mise à jour, tant que l'activation non linéaire reste.

Ce résultat indique que le RNPM pourrait être vulnérable au bruit de structure des graphes. En effet, si la principale contribution à la performance du modèle provient de l'étape d'agrégation, alors l'impact du bruit structurel serait également amplifié. Ce travail propose un modèle théorique basé sur la théorie des matrices aléatoires pour analyser l'interaction entre l'information sur la structure du graphe et l'information sur les caractéristiques des nœuds, lorsque la structure du graphe est fortement perturbée. Le résultat principal est que le bruit structurel du graphe éclipse fortement l'information sur les caractéristiques des nœuds.Lorsqu'un graphe est suffisamment perturbé sur le plan structurel, l'information sur les caractéristiques des nœuds ne contribuera pas à la performance du modèle, même si elle peut elle-même être informative. Cette découverte théorique nous inspire à renforcer les RNPMs contre le bruit structurel du graphe avec un noyau de caractéristiques de nœuds. Les évaluations empiriques montrent l'efficacité du noyau que nous proposons, car il améliore de manière significative les performances du modèle lorsque la structure du graphe est fortement perturbée.

PUBLICATIONS

The following publications and manuscripts are included in parts or in an extended version in this dissertation:

Lutzeyer*, Johannes F., Changmin Wu*, and Michalis Vazirgiannis (2021). "Sparsifying the Update Step in Graph Neural Networks." In: *CoRR* abs/2109.00909. arXiv: 2109.00909.

Seddik, Mohamed El Amine, Changmin Wu, Johannes F. Lutzeyer, and Michalis Vazirgiannis (2022). "Node Feature Kernels Increase Graph Convolutional Network Robustness." In: *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics (AISTATS)*. Proceedings of Machine Learning Research. PMLR.

Wu, Changmin, Giannis Nikolentzos, and Michalis Vazirgiannis (2019). "Matching Node Embeddings Using Valid Assignment Kernels." In: *Complex Networks and Their Applications VIII - Volume 1 Proceedings of the Eighth International Conference on Complex Networks and Their Applications COMPLEX NETWORKS 2019, Lisbon, Portugal, December 10-12, 2019*. Ed. by Hocine Cherifi, Sabrina Gaito, José Fernendo Mendes, Esteban Moro, and Luis Mateus Rocha. Vol. 881. Studies in Computational Intelligence. Springer, pp. 810–821. DOI: 10.1007/978-3-030-36687-2_67.

Wu, Changmin, Giannis Nikolentzos, and Michalis Vazirgiannis (2020). "EvoNet: A Neural Network for Predicting the Evolution of Dynamic Graphs." In: *Artificial Neural Networks and Machine Learning - ICANN 2020 - 29th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 15-18, 2020, Proceedings, Part I*. Ed. by Igor Farkas, Paolo Masulli, and Stefan Wermter. Vol. 12396. Lecture Notes in Computer Science. Springer, pp. 594–606. DOI: 10.1007/978-3-030-61609-0_47.

ACKNOWLEDGMENTS

Firstly, I would like to express my sincere gratitude to my supervisor Prof. Michalis Vazirgiannis. He guided me through every step of this long journey with his patience and motivation, while always leaving me with sufficient space to explore on my own. I deeply admire his research vision and the scope of his scientific knowledge, which he mobilises well in our meetings and discussions. Above all, he has built a great team with a friendly, open-minded and cooperative atmosphere. I could not imagine a better place to conduct the Ph.D. in.

Besides my supervisor, I would like to thank my rapporteurs, Prof. Dimitrios Gunopulos and Prof. TANG Jiliang, for reviewing my thesis and for their helpful and insightful comments. I am very grateful that they had made every effort to hand in the reports in time so that my defence did not need to postpone, even in a difficult situation where I sent them late the draft of this thesis. I also want to thank all my jury members, Prof. Florence d'Alché-Buc, Prof. Stéphane Gaïffa, Prof. Nikos Komodakis and Prof. Catuscia Palamidessi, for contributing their time and energy to this evaluation process and offering valuable feedback. A special thanks goes to Prof. Florence d'Alché-Buc, who had an injury the night before the defence but still insisted and managed to participate and brought up many thought-provoking questions.

I must not forget to thank my two close collaborators, Dr. Giannis Nikolentzos and Dr. Johannes Lutzeyer. As senior researchers, they took on the role, partially and unofficially, but nonetheless excellently, as my advisor/mentor. Giannis led me into the door of research. He is the first researcher that I have collaborated with. In our first project, he taught me hand by hand how to concrete and tackle a research problem, as well as many other aspects, from technical competencies to writing/presentation skills. I have also learned a lot from Johannes. Despite that he only joined the group near the end of my second year, his integrity, scientific rigour and highly organised and efficient working style soon made a strong impact and left me with a deep impression. Not only a pleasant collaborator, but he is also a sincere friend. I am always grateful for his kindness, availability and support when I need it.

Furthermore, I would like to thank all my friends and colleagues that I had the chance to interacted with, who believe in me and are co-protagonists of this adventure, specifically Hadi Abdine, CHEN Dong, George Dasoulas, FANG Chengran, Christos Giatsidis, GUO Yanzhu, Chrysoula Kosma, Moussa Eddine

Kamal, Sammy Khalife, Stratis Limnios, Olivier Pallanca, George Panagopoulos, QIU Yang, Jesse Read, Maria Rossi, Guillaume Salha-Galvan, Mohamed El Amine Seddik, SHANG Guokan, Konstantinos Skianis, Antoine Tixier, Nikolaos Tziortziotis, Christos Xypolopoulos and many others. Their technical and emotional supports were fundamental. I also want to take this occasion to address my special thanks to Jessica Gameiro, secretary and saviour of the group: French bureaucracy is a nightmare, especially the Direction des Étrangers, but it was much less painful with Jessica's help.

In the end, I would like to thank my parents, WANG Guolan and WU Dejin, for their love and continuous support throughout these three years of Ph.D. study. I owe them so much, and this thesis will not be possible without them.

Changmin Wu
Paris, February 2022

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

x	A scalar (integer or real)
$\boldsymbol{x}$	A vector
$\boldsymbol{X}$	A matrix
$\boldsymbol{I}_n$	Identity matrix with n rows and n columns
$\boldsymbol{I}$	Identity matrix with dimensionality implied by context
$\mathbb{X}$	A set
$\mathbb{R}$	The set of real numbers
$\mathbb{R}^n$	The set of real-valued vectors of dimension n
$\mathbb{R}^{n \times n}$	The set of square, real-valued matrices with n rows and n columns
$\{0, 1\}$	The set containing 0 and 1
$\{0, \dots, n\}$	The set of all integers between 0 and n
$[a, b]$	The real interval including a and b
$(a, b]$	The real interval excluding a but including b
$\mathcal{G}$	A graph
$\lvert \cdot \rvert$	Cardinality of a set or the absolute value of a real number
$\left[\cdot \,\|\, \cdot \right]$	concatenation of two vectors
$\lVert \cdot \rVert$	Euclidean (resp., spectral) norm for vectors (resp., matrices)
$\lVert \cdot \rVert_F$	Frobenius norm
$\odot$	hadarmard product or element-wise product
$\mathbf{1}_n$	n-dimensional all-ones vector

LIST OF ACRONYMS

1 INTRODUCTION

1.1 CONTEXT AND SCOPE

We live now in a networked world. With the fast growth of social networks in the last two decades, we witness a blossom of theories, models and tools in network science. From network property analysis such as small-world phenomenon (Buchanan, 2003; Watts and Strogatz, 1998) to link prediction (Lü and Zhou, 2011), from community detection (Leskovec et al., 2009) to recommendation (Fan et al., 2019; Wang, Tan, and Zhang, 2010), from academic networks of citations or co-authors (Tang et al., 2008) to web and blog networks of hyperlinks (Leskovec et al., 2009), social network analysis is the driving force that pushes forward the research in graph mining or graph representation learning.

Graphs do not only shine in the field of social networks. As a natural representation for modelling relations between objects, graphs are perhaps the most frequent data modality that we retrieve from the real world. They have demonstrated their power of providing meaningful information about relations/interactions in a plethora of applications. In biology, the graph is a straightforward representation of molecules (Kriege and Mutzel, 2012). In bioinformatics, the graph is used to model the precise interaction between biological or chemical compounds, such as protein-protein interaction (Borgwardt et al., 2005). In neural science, connections between neurons in the brain are represented as graphs (networks), which has inspired the design of the first Multi-Layer Perceptron (MLP) (LeCun, Bengio, and Hinton, 2015).

Moreover, graphs can even be extended to areas where data does not contain an inherent graph structure. For example, in natural language processing, a sentence/piece of text can be transformed into a graph, called *Graph-of-Word* (Rousseau and Vazirgiannis, 2013), whose vertices are the unique terms and the edges represent term proximity, i. e., co-occurrence relationships. In computer vision, the images can be converted to k-regular graphs whose vertices are the *super-pixels*, i. e., small regions of homogeneous intensity in images, and edges represent spatial proximity between these *super-pixels*, constructed in a fashion of k-nearest neighbours (Achanta et al., 2012; Knyazev, Taylor, and Amer, 2019).

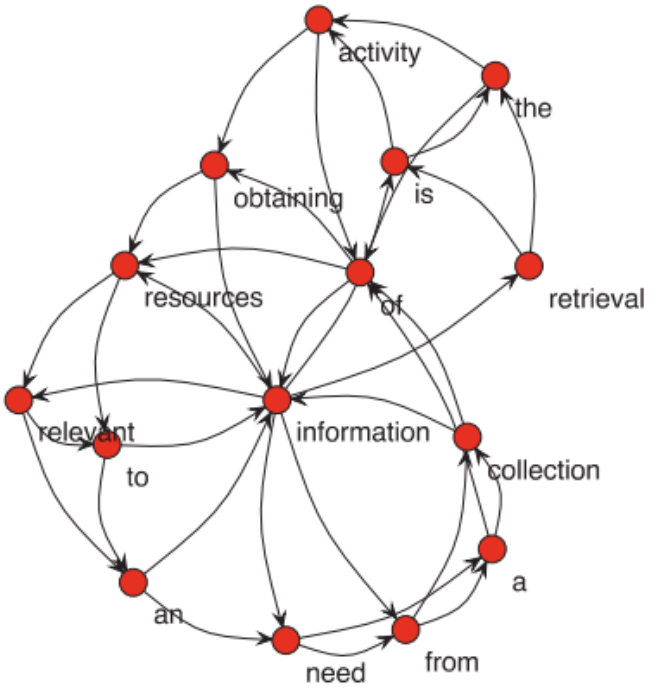

Figure 1.1: Illustration of a *Graph-of-Word* (Rousseau and Vazirgiannis, 2013) which is an unweighted and directed graph. The edge indicates that the two terms (vertices) it connected co-occurred at least once (with the same direction as the edge) in a window of 3 in the text.

However, the omnipresence of graphs has brought many challenges for graph learning as it is tough to analyse the various types of graphs from different areas under a unified framework. Indeed, we have directed/undirected, weighted/unweighted, dynamic/static and heterogeneous/homogeneous graphs. Each requires different methods to process.

Despite the difficulties, researchers were still able to propose some efficient and effective algorithms for specific graph-related problems. These algorithms are usually related to the random graph theory, such as random walks, e. g., the PageRank algorithm (Page et al., 1999), or graph combinatorics, such as graph kernels (Haussler, 1999; Kondor and Lafferty, 2002). Inspired by kernel methods in classic machine learning, graph kernels utilise hand-crafted combinatorial features to measure the similarity between two graphs. They enjoy the advantages of kernel methods as being easy to train and with provable theoretical guarantees.

Seeing the success of deep learning (Goodfellow, 2016) in classic machine learning, such as computer vision, the researchers are also motivated to bring the power of deep learning into graph analysis. However, this migration is not straightforward. Unlike in computer vision, where the data (images) is

euclidean in the form of a regular grid, graph-structured data usually has a non-uniform distribution of vertices, is permutation invariant and often has extra attributes on the edges. These properties make it challenging to define a *convolution* operation on graphs which is the essential part of Convolutional Neural Networks (CNNs). Initial attempts begin from the spectral domain of graphs, as *convolution* is a more natural notion there. The *spectral* Graph Neural Networks (GNNs) have some success (Bruna et al., 2014; Duvenaud et al., 2015; Henaff, Bruna, and LeCun, 2015), but they suffer from high computational cost and a lack of induction ability. Kipf and Welling (2017) proposes Graph Convolutional Neural Network (GCN), which defines the *convolution* operation on the spatial domain of graphs and paves the way to Message-Passing Neural Networks (MPNNs), the current dominating paradigm of GNN designs. The GNNs inherit the expressive power of deep learning and soon take over as the primary tool in graph representation learning. However, just like their parent in classic machine learning, we do not have theoretical guarantees or even understand how GNNs work.

Graph kernels and MPNNs are the two main subjects of this dissertation. The explosion of graph-structured data always calls for more advanced algorithms that are powerful, resource-friendly, and we have a deeper understanding. We will introduce in the next section the improvement we made towards these directions on the two methods.

1.2 OUTLINE AND CONTRIBUTIONS

In this dissertation, we contribute new graph learning algorithms, models and tools with different applications. In particular, we have developed

- a new class of optimal assignment kernel for graph classification which leverages the expressiveness of *bag of vectors* representations for graphs and the validity of histogram intersection kernel,
- an *encoder-decoder* framework for predicting the topology evolution of temporal graphs: this approach takes a GNN as the *encoder*, a graph generation model as the *decoder* and a Recurrent Neural Network (RNN) residing in the *code* space to capture the temporal information,
- a series of parsimonious GNN models which are smaller and cheaper in practice without performance loss for various graph learning tasks: these models reduce the number of parameters by successively sparsifying the *Update* step in existing GNN architectures,

- a robust GNN architecture along with theoretical insights from random matrix theory to improve the model performance against graph structural noise: this approach enhances the information from node features by inserting a node feature kernel in the graph propagation step.

Indeed, the ultimate goal of our research is to provide more *effective*, more *efficient*, more *robust* and more *versatile* algorithms for graph-related tasks. We will elaborate on our contributions, and introduce the outline of this dissertation as follows.

In **Chapter 2**, we begin with some useful notions of graphs and graph theory. We then introduce the machine learning tasks that involve graph-structured data and two major classes of graph representation learning methods that we focus on in our work: *Graph Kernel* and *Graph Neural Network* (GNN). We conclude this chapter by presenting a set of essential libraries that we use in our implementation and a brief introduction to the datasets we experiment on in the following chapters.

Part i is dedicated to the study of graph kernel methods. Most existing kernels belong to the class of R-convolution kernel. It decomposes graphs into substructures and obtains the similarity between each pair of graphs by computing and adding the similarities between every pair of their substructures. This exhaustive computation sometimes causes the problem of *diagonal dominance* where the graph is only similar to itself under the R-convolution kernel.Another class of kernels called assignment kernels improves expressiveness by finding an optimal correspondence between the substructures, but it is not always positive semi-definite. In **Chapter 3**, we propose a new framework to construct an optimal assignment kernel that is guaranteed to be valid. The framework consists of two steps. In the first step, with the help of node embedding methods, we represent each graph as a *bag of vectors* that is assumed to reside in the same embedding space. We recursively split the space into irregular multi-resolution partitions by clustering in the second step. A corresponding hierarchy is then constructed on which the distance between nodes (substructures) measures their correspondence. Finally, we obtain a histogram whose entry corresponds to a vertex of the hierarchy for each graph. A histogram intersection kernel can then be applied. Our proposed framework can measure the similarity between any *set of vectors* object, including but not limited to graphs. Moreover, its kernel feature map is a more expressive embedding for nodes than the original vectors. Experiments on graph classification, link prediction and text categorisation tasks show that the proposed kernel outperforms state-of-the-art baselines and is thus effective.

Part ii is devoted to Graph Neural Networks, with particular emphasis on MPNN, a dominating class of GNN models. We begin by showing the power of GNNs on graph-related tasks with a concrete example in **Chapter 4**. We study the problem of predicting the graph's topology at the next timestep given a sequence of evolving graphs. It is traditionally tackled by random graph models such as Kronecker graphs (Leskovec et al., 2010). These models generate graphs that exhibit specific properties of real-world networks, e. g., shrinking diameter or densification power law (Leskovec, Kleinberg, and Faloutsos, 2005). We propose an *encoder-decoder* framework for a precise prediction of the topology evolution. The *encoder* is a GNN that maps the graph into an embedding, and the *decoder* is an autoregressive graph generation model that reconstructs the graph topology as a sequence of adjacency vectors from an embedding. The choices of GNNs and graph generation models are flexible. This framework takes as input a sequence of graphs, passes them through the GNN and transform the graphs into a sequence of embeddings. Then this sequence is processed by a RNN in order to plug in the temporal relation. The output of RNN is considered as *dynamic graph embedding* as it captures both temporal dependencies of a sequence and structural information from a graph. The *decoder* then reconstructs the topology at the next timestep from this output. The framework is end-to-end trainable with a cross-entropy loss over the prediction of adjacency. Experiments on synthetic and real-world dynamic graphs show that our proposed model has a significant advantage over random graph models on predicting the evolving topology at the next timestep, where its prediction is most similar to the ground truth, measured by a graph kernel.

Chapter 5 focuses on simplifying GNNs, specifically the MPNN. Noticing that the MPNN can be divided into a graph-related *Aggregation* step and a graph-agnostic *Update* step, we study the role of the *Update* step by successively sparsifying its linear transform layer and monitoring the performance change. Our model differs from the current neural network sparsification literature dominated by pruning methods in that the proposed model can identify the sparse sub-network only based on graph properties without training. In our work, we identify the sub-network as an *expander* subgraph from the original bipartite fully connected layer, because of the good property of this type of graph of being sparse and highly connected at the same time. Extensive experiments on three graph learning tasks and various datasets from different domains unanimously drive us to the conclusion that the linear transform layer of the *Update* step can be sparsified almost arbitrarily without performance loss in most cases. Moreover, this layer can even be wholly omitted on some tasks and

datasets, leaving only the nonlinear activation while the model performance remains unharmed.

This observation indicates that the *Update* step plays a less important role than the *Aggregation* step in a MPNN in terms of model performance. A natural concern then arises: *what if the graph structure is perturbed?* Since the graph-related part contributes more to the model performance, the MPNNs are supposed to be vulnerable to graph structural noises. In **Chapter 6**, we further study the interaction between the graph-related step and the graph-agnostic step, particularly under a noisy setting with structural perturbation. We propose *random* GCN, where the weight of the linear transform layer in the *Update* step is a random Gaussian matrix sampled before training and kept unchanged afterwards. We demonstrate empirically that this *random* GCN performs as well as the vanilla GCN in high-dimensional regimes. This model, although lacking practical importance as it can only match the performance of the vanilla model when the dimension is high enough, allows us to introduce tools from random matrix theory and analyse the relation between graph structure information (in the *Aggregation* step) and node feature information (in the *Update* step) theoretically. With certain assumptions on the data, we can conclude that *perturbation on graph structures heavily overshadows information from node features.* Specifically, if the graph structure is completely perturbed, we can not benefit from node feature information. Inspired by the theoretical results, we propose adding a node feature kernel to the message-passing step to robustify the GCN model against structural noise. Experiments demonstrate that our proposed kernel is very effective against graph structural perturbations, especially in the *edge-insertion* case, where random edges are added to the original graph.

Finally, we conclude this dissertation in **Chapter 7** by detailing our contributions and sharing insights on future research directions.

2

BASIC NOTIONS AND PRELIMINARIES

In this chapter, we will provide the definitions and background materials used throughout this dissertation. In Section 2.1, we present the notions and tools of graphs and graph theory. We give an introduction to some problems in the graph learning area in Section 2.2. Then, Section 2.3 and Section 2.4 are dedicated to introducing graph kernels and MPNNs, the two main subjects of this dissertation. Finally, we provide an overview of the software and the major datasets we used, in Section 2.5 and Section 2.6, respectively.

2.1 DEFINITIONS, PROPERTIES AND FUNCTIONS OF GRAPHS

DEFINITION 2.1 (GRAPH). *A graph or a network is a structure amounting to a set of entities with some relationships. Commonly it is represented as an ordered pair* $\mathcal{G} := (\mathbb{V}, \mathbb{E})$ *in which* $\mathbb{V}$ *is a set of vertices and* $\mathbb{E}$ *is a set of edges with* $\mathbb{E} \subseteq \mathbb{V} \times \mathbb{V}$. *Given two nodes* $v_i, v_j \in \mathbb{V}$, *we denote* $e_{ij} = (v_i, v_j) \in \mathbb{E}$ *as the edge pointing from* v_i *to* v_j.

DEFINITION 2.2 (ADJACENCY MATRIX). *The adjacency matrix of a graph* $\mathcal{G} = (\mathbb{V}, \mathbb{E})$ *is defined as* $\boldsymbol{A} = \{\boldsymbol{A}_{ij}\}^{|\mathbb{V}|\times|\mathbb{V}|}$ *with*

$$\boldsymbol{A}_{ij} = \begin{cases} 1 & \text{if } (i,j) \in \mathbb{E}, \\ 0 & \text{otherwise.} \end{cases}$$

DEFINITION 2.3 (LABELLED GRAPH). *A graph* $\mathcal{G} = (\mathbb{V}, \mathbb{E})$ *is a node-labelled graph if it is associated with a function* $\phi : \mathbb{V} \mapsto \Sigma \subset \mathbb{N}$ *that assigns labels (integers) to the vertices of the graph from a discrete set of labels* Σ. *We denote* τ_v *as the label vector for the nodes where* $[\tau_v]_i = l(v_i)$.

Similarly we can define edge-labelled graph endowed with the function $\psi : \mathbb{E} \mapsto \mathcal{R} \subset \mathbb{N}$ *and label vector* τ_e *for the edges.*

DEFINITION 2.4 (ATTRIBUTED GRAPH). *A graph $\mathcal{G} = (\mathbb{V}, \mathbb{E})$ is a node-attributed graph if it is associated with a function $f : \mathbb{V} \mapsto \mathbb{R}^d$ that assigns assigns real-valued vectors to the vertices of the graph. We denote $\boldsymbol{X} \in \mathbb{R}^{|\mathbb{V}| \times d}$ as the feature matrix for the nodes where $\boldsymbol{X}_i = f(v_i)$.*

Similarly we can define edge-attributed graph endowed with the mapping $f' : \mathbb{E} \mapsto \mathbb{R}^{d'}$ and feature matrix $\boldsymbol{E} \in \mathbb{R}^{|\mathbb{V}| \times d'}$ for the edges.

DEFINITION 2.5 (HETEROGENEOUS AND HOMOGENEOUS GRAPH). *A **heterogeneous** graph is a graph of form $\mathcal{G} = (\mathbb{V}, \mathbb{E}, \mathbb{F}, \phi, \psi)$ with $\mathbb{F}$ being the node feature set, $\phi : \mathbb{V} \mapsto \Sigma$ and $\psi : \mathbb{E} \mapsto \mathcal{R}$ being the node/edge label (or type) mappings that satisfy $|\Sigma| + |\mathcal{R}| > 2$. Denote $\mathbb{V}_\sigma$ as the node set of label $\sigma \in \Sigma$, the feature set $\mathbb{F}$ is composed of $|\Sigma|$ feature matrix,*

$$\mathbb{F} = \{\boldsymbol{F}_\sigma | \sigma \in \Sigma\}, \quad \boldsymbol{F} \in \mathbb{R}^{|\mathbb{V}_\sigma| \times d_\sigma}.$$

*where d_σ stands for the feature dimension of nodes with label σ. A **homogeneous** graph is then defined as a graph with a single type of nodes and a single type of edges.*

REMARK 2.1.1. *Apart from $(\mathbb{V}, \mathbb{E})$, a graph can also be represented by its adjacency matrix and/or its node/edge labels and/or features $(\boldsymbol{A}, \tau_v, \tau_e, \boldsymbol{X}, \boldsymbol{E})$. We obtain different classes of graphs, depending on the existence of label mapping or feature mapping and their types. Note that in this dissertation, we focus only on the homogeneous graph. In Part i, our main research object is the node-labelled graph $(\boldsymbol{A}, \tau_v)$, while in Part ii, our main research object is the attributed graph $(\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{E})$. The difference in the complexity of these two objects reflects the difference in the expressive power of the two methods studied respectively in this dissertation.*

DEFINITION 2.6 (DIRECTED AND UNDIRECTED GRAPH). *A directed graph is a graph $\mathcal{G} = (\mathbb{V}, \mathbb{E})$ where every edge $e(v_i, v_j)$ is an ordered pair of nodes that links node v_i to v_j. An undirected graph is a special case of directed graph where if $e(v_i, v_j) \in \mathbb{E}$ then $e(v_j, v_i) \in \mathbb{E}$. Apparently, the adjacency matrix $\boldsymbol{A}$ for an undirected graph is symmetric.*

In this dissertation, we focus mainly on the undirected graphs and thus, all the graphs mentioned afterwards are undirected without specification. Note that however, both methods in Part i and Part ii can be extended to the directed case by choice of node embedding methods and GNNs.

DEFINITION 2.7 (NEIGHBOURHOOD). *Given a node $v_i \in \mathbb{V}$, we define its neighbourhood or the neighbours of v_i as $\mathcal{N}(v_i) = \{v_j \in \mathbb{V} | (v_i, v_j) \in \mathbb{E}\}$.*

DEFINITION 2.8 (DEGREE). *Given an undirected graph $\mathcal{G} = (\mathbb{V}, \mathbb{E})$ and vertex v_i, the degree of v_i is defined as the number of edges incident to v_i,*

$$\text{DEG}_{\mathcal{G}}(v_i) = |\{v_j | e(v_i, v_j) \in \mathbb{E}\}| = \mathcal{N}(v_i).$$

DEFINITION 2.9 (INDUCED SUBGRAPH). *Given a graph $\mathcal{G} = (\mathbb{V}, \mathbb{E})$ and a subset of vertices $\mathbb{S} \subseteq \mathbb{V}$, the subgraph of $\mathcal{G}$ induced by $\mathbb{S}$ is $\mathcal{G}_{\mathbb{S}} = (\mathbb{S}, \mathbb{E}_{\mathbb{S}})$ where the vertex set is $\mathbb{S}$ and the edge set $\mathbb{E}_{\mathbb{S}}$ is*

$$\mathbb{E}_{\mathbb{S}} = \{(v_i, v_j) \in \mathbb{E} | (v_i, v_j) \in \mathbb{S}\}.$$

DEFINITION 2.10 (WALK, PATH, CIRCLE). *A **walk** on a graph $\mathcal{G} = (\mathbb{V}, \mathbb{E})$ is defined as a sequence of vertices $\mathbb{W} = \{v_1, v_2, \ldots, v_l\} \subseteq \mathbb{V}$, where for all $1 \leq i \leq l - 1$, we have $(v_i, v_{i+1}) \in \mathbb{E}$. The length of the walk is equal to the number of edges in the sequence, i. e., l in the above case. If there is no repeated nodes in the sequence, then the walk is called a **path**. If $v_l = v_1$, it is called a **circle**.*

DEFINITION 2.11 (SHORTEST PATH AND DIAMETER). *A **shortest path** from vertex v_i to vertex v_j of a graph $\mathcal{G}$ is a path from v_i to v_j such that there exist no other path between these two vertices with smaller length. The **diameter** of a graph $\mathcal{G}$ is the length of the longest shortest path between any pair of vertices of $\mathcal{G}$.*

DEFINITION 2.12 (TREE). *A tree is an undirected graph in which any two vertices are connected by a unique path, i. e., a connected acyclic undirected graph.*

DEFINITION 2.13 (BIPARTITE GRAPH). *A bipartite graph is a graph whose vertices can be divided into two disjoint and independent sets $\mathbb{U}$ and Γ such that every edge connects a vertex in $\mathbb{U}$ to one in Γ.*

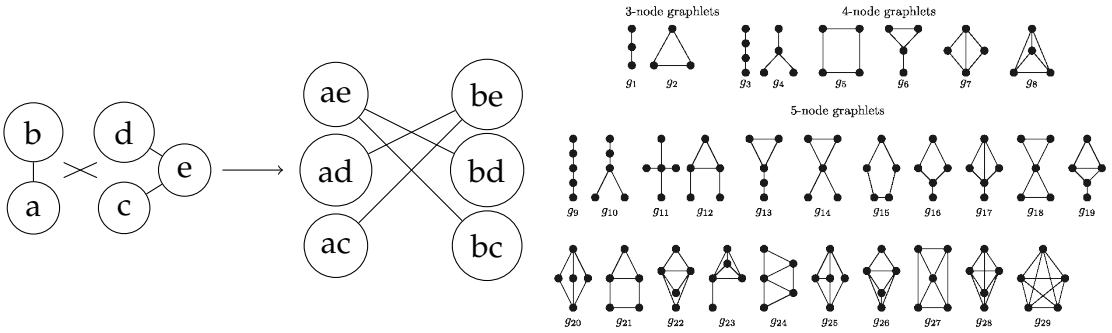

Figure 2.1: **Left:** Illustration of a product graph. **Right:** All 3,4,5-nodes Graphlets (Rahman et al., 2014).

DEFINITION 2.14 (PRODUCT GRAPH). *Given two graphs* $\mathcal{G} = (\mathbb{V}, \mathbb{E})$ *and* $\mathcal{G}' = (\mathbb{V}', \mathbb{E}')$*, the product of these two graphs* $\mathcal{G} \times \mathcal{G}'$ *is defined as* $\mathcal{G}_\times = (\mathbb{V}_\times, \mathbb{E}_\times)$*, with*

$$\begin{aligned} \mathbb{V}_\times &= \{(v_i, v'_j) | v_i \in V \wedge v'_j \in V'\}, \\ \mathbb{E}_\times &= \{((v_i, v'_i), (v_j, v'_j)) \in \mathbb{V}_\times \times \mathbb{V}_\times\}. \end{aligned}$$

$\mathcal{G}_\times$ *is called a product graph. An illustration is given in Figure 2.1.*

DEFINITION 2.15 (GRAPHLET). *A graphlet is a denomination for a small subgraph of a large network parameterised by its size. A network can contain many graphlets of different sizes. An illustration is given in Figure 2.1.*

DEFINITION 2.16 (CONNECTEDNESS). *Given an undirected graph* $\mathcal{G} = (\mathbb{V}, \mathbb{E})$*, two vertices* v_i *and* v_j *are connected if there is a path in* $\mathcal{G}$ *from node* v_i *to node* v_j*.*

DEFINITION 2.17 (BOUNDARY). *The boundary of a set* $\mathbb{S} \subseteq \mathbb{V}$ *denoted* $\partial\mathbb{S}$ *is the set of all vertices, which are connected to a vertex in* $\mathbb{S}$ *by an edge but are not in* $\mathbb{S}$*. The boundary of a single vertex* v *is its neighbourhood* $\mathcal{N}(v)$*.*

DEFINITION 2.18 (EXPANDER GRAPH). *For* $0 < \delta \in \mathbb{R}$*, a graph* $\mathcal{G}$ *is an* δ*-expander graph if for all* $\mathbb{S} \subseteq \mathbb{V}$ *such that* $|\mathbb{S}| \leq \frac{|\mathbb{V}|}{2}$*, we have* $\frac{|\partial\mathbb{S}|}{|\mathbb{S}|} \leq \delta$*.*

DEFINITION 2.19 (EXPANDER RATIO). *The expansion ratio* $h(\mathcal{G})$ *of a graph* $\mathcal{G}$ *is defined to be the minimal* δ *such that* $\mathcal{G}$ *is an* δ*-expander graph.*

In graph theory, *Expander* graphs are a well-studied class of graphs, intuitively understood as highly connected, sparse graphs (Hoory, Linial, and Widgerson, 2006; Lubotzky, 2012), characterised by the expansion ratio. A high expansion ratio indicates that each *small* set of vertices has a relatively large neighbourhood in comparison, which ties in nicely with the qualitative definition as highly connected sparse graphs by Lubotzky (2012). We will demonstrate further in Chapter 5 how the notion of expander graph is helpful in graph learning problems.

DEFINITION 2.20 (GRAPH LAPLACIAN). *Given a graph $\mathcal{G} = (\mathbb{V}, \mathbb{E})$, its adjacency matrix $\boldsymbol{A}$ and the degree matrix $\boldsymbol{D}$ which is a diagonal matrix with $\boldsymbol{D}_{ii} = \sum_j \boldsymbol{A}_{ij}$, we define the graph laplacian as*

$$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A},$$

and the normalised Laplacian as

$$\tilde{\boldsymbol{L}} = \boldsymbol{I}_{|\mathbb{V}|} - \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}},$$

where $\boldsymbol{I}_{|\mathbb{V}|}$ is the identity matrix.

DEFINITION 2.21 (GRAPH FOURIER TRANSFORM AND INVERSE FOURIER TRANSFORM). *Given a graph $\mathcal{G} = (\mathbb{V}, \mathbb{E})$, let $\boldsymbol{X} \in \mathbb{R}^{|\mathbb{V}| \times d}$ denotes a graph signal associated with the node feature mapping $f : \mathbb{V} \mapsto \mathbb{R}^d$, we define the graph Fourier transform on the signal $\boldsymbol{X}$ as*

$$\mathcal{F}(\boldsymbol{X}) = \boldsymbol{U}^\intercal \boldsymbol{X},$$

where $\boldsymbol{U}$ is the matrix of eigenvectors of the normalised graph Laplacian. As $\tilde{\boldsymbol{L}}$ is a real symmetric positive semi-definite matrix, it can be factorised as $\boldsymbol{U}\Lambda\boldsymbol{U}^\intercal$ with Λ being a diagonal matrix of the eigenvalues. The inverse Fourier transform is then

$$\mathcal{F}^{-1}(\boldsymbol{X}') = \boldsymbol{U}\boldsymbol{X}'.$$

2.2 PROBLEMS ON GRAPHS

2.2.1 *Graph Similarity and Isomorphism*

In a graph comparison problem, we measure the *closeness* between two graphs. It can be formulated formally as finding a mapping from graph space to a real value that quantifies the similarity between the two graphs.

DEFINITION 2.22. *(Graph Comparison Problem) Given two graphs $\mathcal{G}, \mathcal{G}' \in \mathbb{G}$, the graph comparison problem aims to find a mapping s*

$$s : \mathbb{G} \times \mathbb{G} \mapsto \mathbb{R}$$

where $s(\mathcal{G}, \mathcal{G}')$ measures the similarity between $\mathcal{G}, \mathcal{G}'$.

The measurement for similarity or closeness is related to the notion of *graph isomorphism*.

DEFINITION 2.23 (GRAPH AND SUBGRAPH ISOMORPHISM). *Given two graphs $\mathcal{G} = (\mathbb{V}, \mathbb{E})$ and $\mathcal{G}' = (\mathbb{V}', \mathbb{E}')$, $\mathcal{G}$ and $\mathcal{G}'$ are isomorphic if and only if there exists a mapping $f : \mathbb{V} \mapsto \mathbb{V}'$ such that $\forall v_i, v_j \in \mathbb{V}$, if $e(v_i, v_j) \in \mathbb{E}$, then $e\big(f(v_i), f(v_j)\big) \in \mathbb{E}'$. The mapping f is called an isomorphism.*

If, without loss of generality, assuming $\mathcal{G}$ contains a subgraph $\mathcal{S}$ that is isomorphic with $\mathcal{G}'$, then $\mathcal{G}$ and $\mathcal{G}'$ are subgraph isomorphic.

However, both graph isomorphism and subgraph isomorphism are NP problems, with subgraph isomorphism being NP-complete. Thus measuring graph similarity with isomorphism is in general infeasible for relatively large graphs. Efficient approximation methods that accurately measure the similarity between two graphs within a reasonable (polynomial) time are needed.

2.2.2 *Machine Learning Tasks on Graphs*

NODE CLASSIFICATION The objective is to find a mapping $f : \mathbb{V} \mapsto \mathbb{N}$, which categorises nodes into discrete classes.

NODE REGRESSION The objective is to find a mapping $f : \mathbb{V} \mapsto \mathbb{R}$, which assigns continuous value to each node.

EDGE CLASSIFICATION The objective is to find a mapping $f : \mathbb{E} \mapsto \mathbb{N}$, similar to node classification.

Figure 2.2: Illustration of a Link Prediction problem. The dash lines between (b, c) and (a, d) in the **right** part indicates potential links.

LINK PREDICTION The objective is to find a mapping $f : \mathbb{V} \times \mathbb{V} \mapsto \{0, 1\}$, which predicts the existence of potential connections when given two nodes. Figure 2.2 gives an illustration of the link prediction problem.

GRAPH CLASSIFICATION The objective is to find a mapping $f : \mathbb{G} \mapsto \mathbb{N}$, which categorises graphs into discrete classes.

GRAPH REGRESSION The objective is to find a mapping $f : \mathbb{G} \mapsto \mathbb{R}$, which predicts continuous value for each graph.

2.3 GRAPH KERNEL

2.3.1 *Kernel Methods*

DEFINITION 2.24 (POSITIVE SEMI-DEFINITE MATRIX). *A real matrix $\boldsymbol{M}^{n\times n}$ is positive semi-definite if*

$$Q(\boldsymbol{v}) = \boldsymbol{v}^\intercal \boldsymbol{M} \boldsymbol{v} \geq 0, \quad \forall \boldsymbol{v} \in \mathbb{R}^n.$$

DEFINITION 2.25 (POSITIVE SEMI-DEFINITE KERNEL). *Let $\mathcal{X}$ be a non-empty set, a symmetric function $K : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is called positive semi-definite kernel if the matrix $\boldsymbol{K}^{n\times n}$ with $\boldsymbol{K}_{ij} = K(x_i, x_j)$ is a positive semi-definite matrix.*

A kernel function is a symmetric continuous function that measures the similarity between two objects. The positive semi-definiteness assures that it can be represented as the inner products between the vector representations of these objects and links the kernel to a Hilbert space. Precisely, if we define a kernel K on $\mathcal{X} \times \mathcal{X}$, then there exists a mapping $\phi : \mathcal{X} \mapsto \mathcal{H}$ from $\mathcal{X}$ to a Hilbert space $\mathcal{H}$ with inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$, such that,

$$K(x_i, x_j) = \langle x_i, x_j \rangle_{\mathcal{H}}, \quad \forall x_i, x_j \in \mathcal{X}.$$

$\mathcal{H}$ is also known as Reproducing Kernel Hilbert Space (RKHS) (Aronszajn, 1950).

The kernel methods can be naturally generalised to the graph space:

DEFINITION 2.26 (GRAPH KERNEL). *The graph kernel K is a positive semi-definite function $\mathbb{G} \times \mathbb{G} \mapsto \mathbb{R}$ defined on $\mathbb{G} \times \mathbb{G}$ with a corresponding Hilbert space $\mathcal{H}$, inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ and a mapping $\phi : \mathbb{G} \mapsto \mathcal{H}$ such that:*

$$K(\mathcal{G}, \mathcal{G}') = \langle \phi(\mathcal{G}), \phi(\mathcal{G}') \rangle_{\mathcal{H}} \quad \forall \mathcal{G}, \mathcal{G}' \in \mathbb{G}.$$

2.3.2 *R-convolution Kernel*

As mentioned above, it is often hard to measure the similarity between two graphs because of the complexity of graph isomorphism. The standard approximation method decomposes the graph into smaller subgraphs and measures similarities between the small pieces instead. A dominating class of graph kernels built upon this concept is the *R*-convolution kernel. It decomposes the input two graphs into two sets of small substructures, computes the similarity between every pair of substructures from these two sets respectively, and then sums the computed values together as the total similarity between two graphs.

DEFINITION 2.27 (DECOMPOSITION). *A decomposition $\mathcal{R}$ of graph $\mathcal{G} = (\mathbb{V}, \mathbb{E})$ is a set of graphs $\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n\}$ where $\mathcal{S}_{i, \forall i=1,\ldots,n}$ is a subgraph of $\mathcal{G}$ such that $\bigcup_{1 \leq i \leq n} \mathcal{S}_i = \mathcal{G}$ and $\mathbb{E}_{\mathcal{S}_i} \bigcap \mathbb{E}_{\mathcal{S}_j} = \emptyset, \forall i, j$.*

DEFINITION 2.28 (*R*-CONVOLUTION KERNEL). *The R-convolution kernel between two graphs $\mathcal{G}$, $\mathcal{G}'$ is defined as*

$$K_{\text{CONV}}(\mathcal{G}, \mathcal{G}') = \sum_{\mathcal{S} \in \mathcal{R}_{\mathcal{G}}} \sum_{\mathcal{S}' \in \mathcal{R}_{\mathcal{G}'}} K_{\text{BASE}}(\mathcal{S}, \mathcal{S}')$$

where $\mathcal{R}$ is the decomposition of graph which decomposes the graph into a set of substructures $\{\mathcal{S}\}$ and K_{BASE} is usually a simple function, i.e.,

$$\begin{aligned} K_{\text{BASE}}(\mathcal{S}, \mathcal{S}') &= 1 \quad \textit{if } \mathcal{S}, \mathcal{S}' \textit{ isomorphic,} \\ &= 0 \quad \textit{otherwise.} \end{aligned}$$

2.3.3 *Some Examples of R-Convolution Kernel*

2.3.3.1 *Random Walk Kernel (Nikolentzos, Siglidis, and Vazirgiannis, 2019)*

The random walk kernel counts common walks (of potentially infinite length) on the input graphs, where the matching walks can be seen as a random walk performed on their product graph. Given input graphs $\mathcal{G}$ and $\mathcal{G}'$, the product graph $\mathcal{G}_\times = \mathcal{G} \times \mathcal{G}'$ and its adjacency matrix $\boldsymbol{A}_\times$, we have

$$K(\mathcal{G}, \mathcal{G}') = \sum_{p,q=1}^{|\mathbb{V}_\times|} \sum_{l=0}^{\infty} [\lambda^l A_\times^l]_{pq} = \mathbf{1}^\intercal (\boldsymbol{I}_{|\mathbb{V}_\times|} - \lambda \boldsymbol{A}_\times)^{-1} \mathbf{1},$$

where the weight coefficient $\lambda \in \mathbb{R}_{\geq 0}$.

2.3.3.2 *Shortest Path Kernel (Borgwardt and Kriegel, 2005)*

The shortest path kernel decomposes input graphs into shortest paths and compares their length and/or labels of the endpoints. We first transfer the input graphs $\mathcal{G} = (\mathbb{V}, \mathbb{E})$ to shortest-path graphs $\mathcal{S} = (\mathbb{V}, \mathbb{E}')$, where

$$\mathbb{E}' = \{(v_i, v_j) | v_i \text{ and } v_j \text{ are connected in } \mathcal{G}\}.$$

Each edge of $\mathcal{S}$ is also assigned with a label that equals the length of the shortest path between the two endpoints in the original graph $\mathcal{G}$. Then, given $\mathcal{G}$, $\mathcal{G}'$, and their shortest-path graphs $\mathcal{S}$, $\mathcal{S}'$, we have

$$K(\mathcal{G}, \mathcal{G}') = \sum_{e \in \mathbb{E}} \sum_{e' \in \mathbb{E}'} K_{\text{BASE}}^{(1)}(e, e'),$$

with $K_{\text{BASE}}^{(1)}(e, e')$ being a positive semi-definite kernel comparing the edge walks of length 1 by,

$$\begin{aligned} K_{\text{BASE}}^{(1)}(e, e') = {} & K_v\big(l(v_i), l(v_i')\big) K_e\big(l(e), l(e')\big) K_v\big(l(v_j), l(v_j')\big) \\ & + K_v\big(l(v_i), l(v_j')\big) K_e\big(l(e), l(e')\big) K_v\big(l(v_j), l(v_i')\big), \end{aligned}$$

where $e = (v_i, v_j)$ and $e' = (v_i', v_j')$. K_v and K_e are simple functions, e. g., Dirac kernel, that compare between node labels and edge labels.

2.3.3.3 *Graphlet Kernel (Shervashidze et al., 2009)*

The graphlet kernel decomposes input graphs into limit-size subgraphs called *graphlets*. Given a set of graphlets $\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{n_k}\}$ the complete set of max k-size graphlets, we count the frequency of occurrence of graphlet $\mathcal{S}_i$ in the original graph $\mathcal{G}$, denoted as f_i. Then the vector $\boldsymbol{f}_{\mathcal{G}} = (f_1, f_2, \ldots, f_{n_k})$ is called the k-spectrum of $\mathcal{G}$ and we have:

$$K(\mathcal{G}, \mathcal{G}') = f_{\mathcal{G}}^{\intercal} f_{\mathcal{G}'}.$$

2.3.3.4 *Weisfeiler-Lehman Subtree Kernel (Shervashidze et al., 2011)*

The Weisfeiler-Lehman Subtree Kernel comparing the number of shared subtrees extracted from input graphs under the Weisfeiler–Lehman framework inspired by the Weisfeiler–Lehman test of graph isomorphism.

Given two inputs $\mathcal{G}$ and $\mathcal{G}'$, let us define $\Sigma_i \subseteq \Sigma$ as the set of letters that occur as node labels at least once in $\mathcal{G}$ or $\mathcal{G}'$ at the end of the i^{th} iteration of the Weisfeiler-Lehman algorithm. Let Σ_0 be the set of original node labels of $\mathcal{G}$ and $\mathcal{G}'$. Assume all Σ_i are pairwise disjoint. Without loss of generality, assume that every $\Sigma_i = \{\sigma_{i,1}, \ldots, \sigma_{i,|\Sigma_i|}\}$ is ordered. Define a map $c_i : \{\mathcal{G}, \mathcal{G}'\} \times \Sigma_i \mapsto \mathbb{N}$ such that $c_i(\mathcal{G}, \sigma_{i,j})$ is the number of occurrences of the letter $\sigma_{i,j}$ in the graph $\mathcal{G}$.

The Weisfeiler-Lehman Subtree Kernel on $\mathcal{G}$ and $\mathcal{G}'$ with h iterations of the Weisfeiler-Lehman algorithm is then defined as

$$K(\mathcal{G}, \mathcal{G}') = \left\langle \phi_h(\mathcal{G}), \phi_h(\mathcal{G}') \right\rangle,$$

with

$$\phi_h(\mathcal{G}) = \left(c_0(\mathcal{G}, \sigma_{0,1}), \ldots, c_0(\mathcal{G}, \sigma_{0,|\Sigma_0|}), \ldots, c_h(\mathcal{G}, \sigma_{h,1}), \ldots, c_h(\mathcal{G}, \sigma_{h,|\Sigma_h|}) \right),$$
$$\phi_h(\mathcal{G}') = \left(c_0(\mathcal{G}', \sigma_{0,1}), \ldots, c_0(\mathcal{G}', \sigma_{0,|\Sigma_0|}), \ldots, c_h(\mathcal{G}', \sigma_{h,1}), \ldots, c_h(\mathcal{G}', \sigma_{h,|\Sigma_h|}) \right).$$

2.3.4 *Optimal Assignment Kernel*

Similar to the R-convolution kernel, the optimal assignment kernel also decomposes the graph into small pieces. However, instead of computing similarity between every pair of them, the optimal assignment kernel finds a bijection (assignment) between the two sets of substructures that maximises the total similarity.

DEFINITION 2.29 (OPTIMAL ASSIGNMENT KERNEL). *The optimal assignment kernel* $K_{\text{ASSIGN}} : \mathbb{G} \times \mathbb{G} \mapsto \mathbb{R}$ *is defined for every* $\mathcal{G}, \mathcal{G}' \in \mathbb{G}$ *as*

$$K_{\text{ASSIGN}}(\mathcal{G}, \mathcal{G}') = \begin{cases} \max_{\pi \in \Gamma_n} \sum_{i=1}^{n} K_{\text{BASE}}(\mathcal{S}_i, \mathcal{S}'_{\pi(i)}) & \text{if } |\mathcal{R}'| > |\mathcal{R}|, \\ \max_{\pi \in \Gamma_{n'}} \sum_{i=1}^{n'} K_{\text{BASE}}(\mathcal{S}_{\pi(i)}, \mathcal{S}'_i) & \text{otherwise.} \end{cases}$$

where $\mathcal{R} = (\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n)$ *is a decomposition of* $\mathcal{G}$ *and* Γ_n *is the set of all possible permutations of* n *elements.*

Comparing to the R-convolution kernel, the optimal assignment kernel can reveal the structural correspondence between two graphs and is less likely to have the *diagonal dominance* problem[1] which is common in the R-convolution kernel.

However, Vert (2008) has given a counterexample showing that the optimal assignment kernel is not in every case positive semi-definite.

THEOREM 2.3.1 (VERT, 2008). *The optimal assignment kernel is not always positive semi-definite.*

2.4 GRAPH NEURAL NETWORK

In recent years, GNNs have become a highly impactful model type for the analysis of graph-structured data. This is mainly due to their ability to process attributed graphs composed of node information and an underlying graph structure, and their dominating empirical performance in various application areas, including chemistry (Duvenaud et al., 2015), social networks (Monti et al., 2019), natural language processing (Yao, Mao, and Luo, 2019) and neural science (Griffa et al., 2017).

Among various GNN models, the MPNN draws our attention as a prominent paradigm that arose recently for performing machine learning tasks on graphs.

2.4.1 *Message-Passing Neural Network*

Given graphs as $\mathcal{G} = (\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{E})$ with $\boldsymbol{A} \in \{0,1\}^{n \times n}$ being the adjacency matrix which contains the information of the graph's vertex set $\mathbb{V}$, $\boldsymbol{X} \in \mathbb{R}^{n \times d}$. being the

1 Mathematically, *diagonal dominance* indicates that for every row i of the kernel matrix $\boldsymbol{K}$, $|\boldsymbol{K}_{ii}| \geq \sum_{i \neq j} |\boldsymbol{K}_{ij}|$ where $|\cdot|$ denotes the absolute value. In a practical sense of graph comparison, the graphs will be only similar to themselves and dissimilar to all the others, which does not yield a meaningful comparison.

node features and $\boldsymbol{E} \in \mathbb{R}^{n \times d'}$. being the edge features, a graph representation learning task aims at learning meaningful embeddings (high-level representations) on the node or graph level that can be used in downstream tasks. MPNNs, probably the current most common class of GNNs, learn such embeddings by iteratively aggregating information from the neighbourhoods of each node and updating their representations based on this information. Precisely, the learning procedure of MPNNs can be divided into the following phases:

INITIAL (OPTIONAL) In this phase, the initial node features $\boldsymbol{X}$ is mapped from the feature space to a hidden space by a parameterised neural network $\Phi^{(0)}$, usually a fully-connected linear layer.

$$\boldsymbol{H}^{(1)} = \Phi^{(0)}(\boldsymbol{X}) = \left(\boldsymbol{h}_1^{(1)}, \ldots, \boldsymbol{h}_n^{(1)}\right),$$

where the hidden representation of node i is denoted as $\boldsymbol{h}_i^{(1)}$, which will be used as the initial point for later iterations. A similar operation could be performed on edge features $\boldsymbol{E}$, but is less common.

AGGREGATION In this phase, MPNNs gather, for each node, information from the node's neighbourhood $\mathcal{N}(i)$, and its incident edges. The gathered pieces of information are called *messages*, denoted by $\boldsymbol{m}_i$. Formally, if $\Psi^{(l)}(\cdot)$ denotes the aggregation function at iteration l, then

$$\boldsymbol{m}_i^{(l)} = \Psi^{(l)}\left(\boldsymbol{h}_i^{(l)}, \left\{\boldsymbol{h}_j^{(l)}, \boldsymbol{E}_{ij} | j \in \mathcal{N}(i)\right\}\right).$$

Due to the isotropic nature of graphs (arbitrary node labelling), this function needs to be permutation equivariant or invariant. It also has to be differentiable so that the framework will be end-to-end trainable.

UPDATE The nodes then update their hidden representations based on their current representations and the received *messages*. Let $\Phi^{(l)}$ denote the update function at iteration l. For node i, we have

$$\boldsymbol{h}_i^{(l+1)} = \Phi^{(l)}\left(\boldsymbol{h}_i^{(l)}, \boldsymbol{m}_i^{(l)}\right).$$

The *Aggregation* step and the *Update* step sometimes are informally referred to as *graph propagation* or *message-passing* procedure (hence the name Message-Passing Neural Network) in this dissertation. After k iterations of the *message passing* procedure, each node obtains a feature vector that captures the structural information within its k-hop neighbourhood.

READOUT (OPTIONAL) After L *Aggregation* and *Update* iterations, depending on the downstream tasks, the MPNN will either output node representations directly or generate a graph representation via a differentiable and permutation invariant readout function,

$$\boldsymbol{g} = \Theta\left(\left\{\boldsymbol{h}_i^{(L)} | i \in \mathbb{V}\right\}\right).$$

2.4.2 *Some examples of Message-Passing Neural Network*

2.4.2.1 *Graph Convolutional Neural Network (Kipf and Welling, 2017)*

The *message-passing* procedure in GCN, at iteration l, can be written as

$$\boldsymbol{m}_i^{(l)} = \frac{1}{\sqrt{d_i}} \sum_{j \in \mathcal{N}(i)} \boldsymbol{h}_j^{(l)} \frac{1}{\sqrt{d_j}}, \tag{2.1}$$

$$\boldsymbol{h}_i^{(l+1)} = \sigma(\boldsymbol{m}_i^{(l)} \boldsymbol{W}^{(l)}), \tag{2.2}$$

where d_i is the degree of node i, $\sigma(\cdot)$ is the activation function and $\boldsymbol{W}^{(l)}$ is the weight matrix of the linear transform layer in the l^{th} *Update* step. Equation 2.1 corresponds to the *Aggregation* step that constructs *message* as a weighted average of the neighbours' embeddings, then equation 2.2, as the *Update* step, updates node i's representation based on the constructed *message*.

2.4.2.2 *Graph Isomorphism Network (Xu et al., 2019)*

The *message-passing* procedure of the Graph Isomorphism Network (GIN) is very similar to that of the GCN, except that, at the *Aggregation* step, the GIN takes into account the central node's representation parameterised by a learnable coefficient. Precisely,

$$\boldsymbol{m}_i^{(l)} = (1 + \epsilon)\boldsymbol{h}_i^{(l)} + \sum_{j \in \mathcal{N}(i)} \boldsymbol{h}_j^{(l)}. \tag{2.3}$$

The *Update* step is the same with equation 2.2.

2.4.2.3 *GraphSage (Hamilton, Ying, and Leskovec, 2017)*

GraphSage also incorporates explicitly the central node's representation in the *Aggregation* step by concatenating with the embeddings from the neighbourhood. The *message-passing* procedure is then,

$$\boldsymbol{m}_i^{(l)} = \left[\boldsymbol{h}_i^{(l)} \Big\| \max_{j \in \mathcal{N}(i)} \sigma(\boldsymbol{h}_j^{(l)} \boldsymbol{W}_1^{(l)}) \right], \tag{2.4}$$

$$\boldsymbol{h}_i^{(l+1)} = \frac{\hat{\boldsymbol{h}}_i^{(l+1)}}{\left\| \hat{\boldsymbol{h}}_i^{(l+1)} \right\|_2}, \quad \hat{\boldsymbol{h}}_i^{(l+1)} = \sigma(\boldsymbol{m}_i^{(l)} \boldsymbol{W}_2^{(l)}). \tag{2.5}$$

where $\max(\cdot)$ takes maximum along each feature dimension, $\sigma(\cdot)$ is the activation function and $\boldsymbol{W}_1^{(l)}, \boldsymbol{W}_2^{(l)}$ are the weight matrices of the linear transform layer in the *Aggregation* and *Update* steps respectively.

2.4.2.4 *Principle Neighborhood Aggregation (Corso et al., 2020)*

In the *Aggregation* step, the Principle Neighborhood Aggregation (PNA) model concatenates the *messages* obtained via different combinations of scalars (coefficients) and aggregators (functions). This step can be written as,

$$\boldsymbol{m}_i^{(l)} = \bigoplus_{j \in \mathcal{N}(i)} \sigma\left(\boldsymbol{h}_j^{(l)} \boldsymbol{W}^{(l)} \right), \tag{2.6}$$

where $\bigoplus$ is a concatenation of the tensor product, denoted by $\otimes$, between the arrays of scalars and aggregators as follow,

$$\underbrace{\begin{bmatrix} \boldsymbol{I} \\ S(\boldsymbol{D}, \alpha = 1) \\ S(\boldsymbol{D}, \alpha = -1) \end{bmatrix}}_{\text{scalars}} \otimes \underbrace{\begin{bmatrix} \text{mean} \\ \text{std} \\ \text{max} \\ \text{min} \end{bmatrix}}_{\text{aggregators}},$$

where $S(\boldsymbol{D}, \alpha)$ is a degree-based normalisation coefficient defined in Corso et al. (2020) and std is the standard deviation aggregation. For example, for combination $S(\boldsymbol{D}, \alpha = 1) * \text{std}(\cdot)$, we have

$$[\boldsymbol{m}_i^{(l)}]_{S(\boldsymbol{D},\alpha=1),\text{std}} = \frac{\log(d_i + 1)}{\delta} \text{std}_{j \in \mathcal{N}(i)} \left[\sigma\left(\boldsymbol{h}_j^{(l)} \boldsymbol{W}^{(l)} \right) \right],$$

where δ is a data-dependent normalisation factor. The *Update* step is the same as equation 2.2.

2.4.3 *Relation between GCN and spectral GNNs*

The MPNN belongs to the class of *spatial* GNN where the *convolution* operation is defined on the spatial domain as the aggregation over neighbourhoods of each node. There exists another class of GNNs called *spectral* GNN where the *convolution* operation is defined on the spectral domain as,

$$\boldsymbol{X} * \boldsymbol{g} = \mathcal{F}^{-1}\left(\mathcal{F}(\boldsymbol{X}) \odot \mathcal{F}(\boldsymbol{g})\right) = \boldsymbol{U}\left(\boldsymbol{U}^\intercal \boldsymbol{X} \odot \boldsymbol{U}^\intercal \boldsymbol{g}\right),$$

where $*$ is the convolution operation following classic signal processing (Mallat, 2009), $\boldsymbol{X}$ and $\boldsymbol{g}$ are the graph signal and the filter respectively. Parameterising the convolution kernel $\boldsymbol{U}^\intercal \boldsymbol{g}$ as $\boldsymbol{\theta}$, we have,

$$\boldsymbol{X} * \boldsymbol{g} = \boldsymbol{U}\left(\boldsymbol{U}^\intercal \boldsymbol{X} \odot \boldsymbol{\theta}\right) = \boldsymbol{U}\left(\boldsymbol{\theta} \odot \boldsymbol{U}^\intercal \boldsymbol{X}\right) = \boldsymbol{U}\boldsymbol{g}_{\boldsymbol{\theta}}\boldsymbol{U}^\intercal \boldsymbol{X},$$

where $\boldsymbol{g}_{\boldsymbol{\theta}}$ in the simplest form is a diagonal matrix with diagonal $\boldsymbol{\theta}$. Intuitively, the procedure of *spectral* GNN can be understood as follow: the graph signal is first transformed from the spatial domain to the spectral domain by graph Fourier transform. It is filtered in the spectral domain with the filter $\boldsymbol{g}_{\boldsymbol{\theta}}$, and then transformed back to the spatial domain by the inverse Fourier transform.

It is interesting to show the relation between GCN, one of the first *spatial* GNNs, and the *spectral* GNNs. Indeed, let us write the *message-passing* procedure of the GCN in a matrix form,

$$\boldsymbol{H}^{(l+1)} = \sigma\left(\tilde{\boldsymbol{A}}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)}\right),$$

where $\tilde{\boldsymbol{A}} = \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$.[2] We have

$$\tilde{\boldsymbol{A}}\boldsymbol{H}^{(l)} = \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{H}^{(l)} = (\boldsymbol{I}_{|\mathbb{V}|} - \tilde{\boldsymbol{L}})\boldsymbol{H}^{(l)} = \boldsymbol{U}(\boldsymbol{I}_{|\mathbb{V}|} - \Lambda)\boldsymbol{U}^\intercal \boldsymbol{H}^{(l)},$$

with $\boldsymbol{g}_{\boldsymbol{\theta}} = \boldsymbol{I}_{|\mathbb{V}|} - \Lambda \in (-1, 1)$. Thus, GCN can be written in the form of the *spectral* GNN and serves as a low-pass filter in the graph spectral domain (Balcilar et al., 2021; NT and Maehara, 2019; Zhou et al., 2020).

2 Note that here $\boldsymbol{A}$ corresponds to the original graph with self-loops $\boldsymbol{A}_{\text{ORIGIN}} + \boldsymbol{I}$.

2.5 SOFTWARE AND LIBRARIES

The primary software and libraries that we use in this thesis are listed below:

- `PyTorch` (Paszke et al., 2019). A Python machine learning library that provides strong Graphics Processing Unit (GPU)-accelerated tensor computation and deep neural network implementations built on a tape-based auto-grad system.
- `PyTorch Geometric` (Fey and Lenssen, 2019). A PyTorch-based library designed to facilitate the implementation and training of Graph Neural Networks.
- `Deep Graph Library` (Wang et al., 2019). Another library designed to facilitate the implementation and scalable training of Graph Neural Networks. Compatible with PyTorch, TensorFlow (Abadi et al., 2015) and Apache MXNet (Chen et al., 2015).
- `NetworkX` (Hagberg, Schult, and Swart, 2008). A Python library for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.
- `Igraph` (Csardi and Nepusz, 2006). A library for the creation and manipulation and fast analysis of (large) graphs.
- `Numpy` (Harris et al., 2020). A Python library for scientific computing that supports large, multi-dimensional arrays and matrices.
- `Scikit-learn` (Pedregosa et al., 2011). A Python machine learning library that provides simple and efficient tools for predictive data analysis.
- `Matplotlib` (Hunter, 2007). A Python-based comprehensive library for the creation of static, animated, and interactive visualisations.

2.6 OVERVIEW OF DATASETS

TU DATASETS TU Benchmark datasets (Morris et al., 2020) are mainly small and medium-sized benchmark datasets for graph classification or regression. In this thesis, we use

- datasets of small molecules including MUTAG, PTC-MR, NCI1 and ZINC, where for each graph, the nodes represent atoms, the edges

are chemical bonds, and they are annotated by the atom and bond types. Additional chemical attributes are possibly available.

- datasets of proteins including D&D, ENZYMES and PROTEINS, where for each graph, the nodes are secondary structure elements and two nodes are connected if they are neighbours along the amino acid sequence or one of the three nearest neighbours in space. The node labels encode their type, i. e., helix, sheet or turn, and several physical and chemical information (Borgwardt et al., 2005).
- datasets of social networks including IMDB-BINARY, IMDB-MULTI, REDDIT-BINARY, REDDIT-MULTI-5K and REDDIT-MULTI-12K, where the IMDB graphs are the ego-network of actor collaborations and the REDDIT graphs are discussion threads where the nodes represent users, and the edges correspond to the responses from one user to the comments of the other.

OGB BENCHMARK DATASETS OGB Benchmark datasets (Hu et al., 2020) is a collection of large-scale and diverse datasets for graph classification or regression, link prediction and node classification. In this thesis, we use

- dataset of the citation network between all Computer Science arXiv papers, ogbn-arxiv, where each node represents a paper, and a directed edge is added between two papers if one cited the other. The nodes are annotated by the average of word embeddings in the paper's title and abstract.

OTHERS Other than the two primary sources of benchmark datasets for graph learning algorithms evaluation, we also use in this thesis,

- datasets of computer vision, including MNIST and CIFAR10, where the images are transferred into graphs following Knyazev, Taylor, and Amer (2019), with the nodes being *super-pixels*, and the edges corresponding to their spatial proximity in the two-dimensional space.
- datasets of the citation networks including Cora, CiteSeer, PubMed (Yang, Cohen, and Salakhutdinov, 2016) and an extended version of Cora, CoraFull (Shchur et al., 2018).
- dataset of the co-author network, CS (Shchur et al., 2018), part of the Microsoft Academic Graph, where the nodes represent authors in the Computer Science area, and two nodes are connected if the two authors co-authored a paper. The nodes are annotated by the keywords of the author's papers.

- datasets of the co-purchase network, Photo, segment of the Amazon co-purchase graph, where the nodes represent goods, and edges indicate that the two goods are frequently bought together. The nodes are annotated by the *bag of words* representation of product reviews.

The statistics of the datasets mentioned above are summarised in Table 2.1. Note that "#Features" corresponds to the dimension of feature vectors if the graph is attributed and the dimension of one-hot embeddings of the categorical value if the graph is labelled.

Table 2.1: Statistics of the datasets used in our experiments

Source	Dataset	#Graphs	#Nodes (avg.)	#Edges (avg.)	#Features	#classes	Task
Small molecules	MUTAG	188	17.93	19.79	7	2	Graph Classification
	PTC-MR	344	25.56	25.96	19	2	
	NCI1	4,110	29.87	32.30	37	2	
Bioinformatics	D&D	1,178	284.32	715.66	82	2	
	ENZYMES	600	32.63	62.14	3	6	
	PROTEINS	1,113	39.06	72.82	4	2	
Social networks	IMDB-BINARY	1,000	19.77	96.53	—	2	
	IMDB-MULTI	1,500	13.00	65.93	—	3	
	REDDIT-BINARY	2,000	429.61	497.75	—	2	
	REDDIT-MULTI-5K	4,999	508.50	594.87	—	5	
	REDDIT-MULTI-12K	11,929	391.40	456.89	—	11	
Computer vision	MNIST	70000	70.57	282.27	—	10	
	CIFAR10	60000	117.63	470.53	—	10	
Small molecules	ZINC	249,456	23.14	24.91	28	—	Graph Regression
Citation	CORA	1	2,708	5,278	1,433	7	Node Classification
	CITESEER	1	3,327	4,552	3,703	6	
	PUBMED	1	19,717	44,324	500	3	
	CORA-Full	1	18,703	62,421	8,710	67	
	ogbn-arxiv	1	169,343	1,166,243	128	40	
Co-author	CS	1	18,333	81,894	6,805	15	
Amazon co-purchase	Photo	1	7,487	119,043	745	8	

Part I

KERNEL METHODS

3

A VALID OPTIMAL ASSIGNMENT KERNEL

As an extension of kernel methods onto graphs, graph kernels were and remain today a favourable approach in graph classification for tackling the graph similarity and learning tasks at the same time. Most graph kernels are instances of the *R*-convolution framework. These kernels decompose graphs into their substructures and sum over all pairs of these substructures. A more promising family of kernels are the assignment kernels, which compute a matching between substructures of two objects such that the total similarity between the matched substructures is maximum.

In this chapter, we present a kernel which compares graphs by computing an assignment of their node embeddings. After embedding the vertices of all graphs in a vector space, we construct a hierarchy of the vertices using a clustering algorithm. Based on this hierarchy, we define a kernel that computes an optimal assignment of the vertices of two graphs.

The proposed kernel is not limited to graph comparison, but can be applied to measure the similarity between any objects represented as sets of vectors. The proposed kernel is evaluated on several graph classification, link prediction and text categorisation datasets. Our results indicate that the proposed approach either outperforms or performs comparably to traditional and the state-of-the-art techniques.

3.1 INTRODUCTION

In recent years, graph-structured data has experienced an enormous growth in many domains, ranging from social networks to bioinformatics. Several problems of increasing interest call for the use of machine learning techniques on graph-structured data. As a consequence, graph classification has emerged as a very important task, and has found applications in several fields such as in computational biology (Schölkopf, Tsuda, and Vert, 2004), in information retrieval (Hermansson et al., 2013) and in cybersecurity (Gascon et al., 2013).

Although in the past few years several neural network models have been generalised to work on graph-structured data (Kondor et al., 2018; Niepert, Ahmed, and Kutzkov, 2016; Zhang et al., 2018), graph kernels are still the

dominant approach for the classification of small and medium-sized graph datasets.

A graph kernel is a symmetric, positive semi-definite function on the set of graphs $\mathbb{G}$. Have we defined such a function k, it is known that there exists a map $\phi : \mathbb{G} \to \mathcal{H}$ from the graph space into a Hilbert space $\mathcal{H}$ such that $k(\mathcal{G}, \mathcal{G}') = \langle \phi(\mathcal{G}), \phi(\mathcal{G}') \rangle_{\mathcal{H}}$ for all $\mathcal{G}, \mathcal{G}' \in \mathbb{G}$. Hence, a graph kernel k implicitly computes the inner product between the representations of the input graphs in the Hilbert space $\mathcal{H}$, and allow kernel classifiers such as Support Vector Machine (SVM) to work directly on graphs.

The majority of graph kernels are instances of the R-convolution framework (Haussler, 1999). These kernels decompose each graph into a set of substructures and compute the similarity of two graphs by comparing each possible pair of these substructures. Different types of substructures give rise to different kernels. Hence, there are kernels that compare graphs based on shortest paths (Borgwardt and Kriegel, 2005), subtrees (Ramon and Gärtner, 2003) or graphlets (Shervashidze et al., 2009), just to name a few.

A more promising family of kernels are the assignment kernels. In general, these kernels compute a matching between substructures of one object and substructures of a second object such that the overall similarity of the two objects is maximised. Such a matching can reveal structural correspondences between the two objects. Furthermore, the emerging kernel matrices do not suffer from the diagonal dominance problem (Yanardag and Vishwanathan, 2015). Unfortunately, the assignment functions are not in general positive semi-definite which complicates their use in kernel methods. For example, an optimal assignment kernel that was proposed in the early days of graph kernels to compute a correspondence between the atoms of molecules (Fröhlich et al., 2005), was later proven not to always be positive semi-definite (Vert, 2008). Despite the difficulty of defining valid assignment kernels, there are some assignment methods that respect the constraint of positive semi-definiteness, such as a method that capitalises on the well-known pyramid match kernel to match the node embeddings of graphs (Nikolentzos, Meladianos, and Vazirgiannis, 2017). More importantly, it was recently shown that there exists a class of base kernels used to compare substructures that guarantees positive semi-definite optimal assignment kernels (Kriege, Giscard, and Wilson, 2016). While the above work paves the way for breaking away from the R-convolution framework and for designing new kernels based on optimal assignments, to the best of our knowledge, no subsequent papers have considered this problem.

In this work, we design an assignment kernel between sets of vectors. We capitalise on the work of Kriege, Giscard, and Wilson (2016), and we propose a base kernel for comparing vectors which approximates the linear kernel. In contrast to the linear kernel, the proposed base kernel guarantees positive semi-definite assignment kernels. The base kernel is obtained from a hierarchical partition of the input space allowing the optimal assignment to be computed in linear time by histogram intersection. We use the proposed kernel to compare graphs and textual documents by comparing their parts (i. e., their node embeddings and word embeddings respectively). We evaluate the proposed approach on several datasets from graph classification, link prediction and text categorisation.

Our main contributions are summarised as follows:

- We take as starting point, the theory of valid optimal assignment kernels developed by Kriege, Giscard, and Wilson (2016), and we propose a kernel that computes a correspondence between two sets of vectors.
- We demonstrate the utility of the proposed kernel in concrete applications, namely, in the tasks of graph classification, link prediction and text categorisation.
- We show that our proposed kernel is very competitive comparing with several state-of-the-art graph kernels.

The rest of this paper is organised as follows. Section 3.2 provides a detailed description of the proposed method that computes an optimal assignment of sets of embeddings. Section 3.3 evaluates the proposed framework on several standard datasets from graph classification, link prediction and text categorisation. Finally, Section 3.4 concludes.

3.2 PROPOSED KERNEL

In this section, we first present the *set of vectors* representation for graphs and how the nodes of the graphs are embedded in the vector space. We then introduce the notion and theory of valid optimal assignment kernels. We finally present the our proposed approach for computing an optimal correspondence between set of vectors (graphs).

3.2.1 *Preliminaries*

3.2.1.1 *Graph as Bag of Vectors*

Bag of words is a common term in Natural Language Processing (NLP), which refers to representing a document as a set of its containing words, ignoring their order. Though one may expect that word order could play an important role in linguistics, this method works surprisingly well in some specific tasks of NLP.

This notion can be naturally generalised to the case of graphs, where each graph is represented as a set of the nodes. In fact, it might be more natural than the notion in linguistics as there is no canonical ordering of the nodes in a graph. As each node is represented by a vector in the Euclidean space, a graph will finally become a *bag of vectors*. The essential part of this method is the vector representation of the vertices, commonly referred to as *Node Embedding*, which corresponds to a mapping that maps the nodes into a low-dimensional vector space while keeping some invariant properties on graphs.

DEFINITION 3.1 (BAG OF VECTORS). *A graph $\mathcal{G} = (\mathbb{V}, \mathbb{E})$ can be represented as a set of vectors $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{|\mathbb{V}|}\}$ with $\boldsymbol{x}_i \in \mathbb{R}^d$ and $\boldsymbol{x}_i = f(v_i), \forall i$, where*

$$f : \mathbb{V} \mapsto \mathbb{R}^d \quad \textit{and} \quad d_{\mathbb{V}}(v_i, v_j) = d_{\mathbb{R}^d}(f(v_i), f(v_j)),$$

$d_{\mathbb{V}}$, $d_{\mathbb{R}^d}$ are similarity measures defined respectively on $\mathbb{V}$ and $\mathbb{R}^d$.

There are various node embedding methods, differing from each other in terms of the invariant properties that they keep. Most of these methods can be unified under a general framework of matrix factorisation (Qiu et al., 2018). In this work, we mainly use a simple embedding method based on eigenvector decomposition of graph adjacency matrix. However, it should be noted that the proposed framework is flexible with the different embedding methods and indeed they have an impact on the expressive ability of the kernel, as demonstrated in Section 3.3.2.

3.2.1.2 *Embeds graph by Adjacency Matrix Decomposition*

Given a graph $\mathcal{G} = (\mathbb{V}, \mathbb{E})$, an embedding algorithm projects the vertices of $\mathcal{G}$ into a vector space. Embedding graphs into low-dimensional vector spaces has proven to be valuable in many tasks such as in node classification and in link prediction. In case these embeddings are consistent across different graphs, it can also be useful for graph comparison.

In this work, we generate embeddings for the vertices of a graph $\mathcal{G} = (\mathbb{V}, \mathbb{E})$ using the eigenvectors of its adjacency matrix $\boldsymbol{A}$. Given the eigenvalue decomposition of the adjacency matrix $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$, the i^{th} row $\boldsymbol{u}_i$ of $\boldsymbol{U}$ corresponds to the embedding of vertex $v_i \in \mathbb{V}$. Since the signs of these eigenvectors are arbitrary, we replace all their components by their absolute values. Specifically, we embed all vertices in the d-dimensional vector space $\mathbb{R}^d$ using the eigenvectors of the d *largest in magnitude* eigenvalues. In case of multiple graphs such as the task in Section 3.3.1, we generate low-dimensional representations for vertices of each graph and assume they reside in the same $\mathbb{R}^d$ space.

Instead of the adjacency matrix, we could have employed other matrices derived from graphs such as the Laplacian matrix, or other more sophisticated methods for generating embeddings (Donnat et al., 2018; Grover and Leskovec, 2016; Ribeiro, Saverese, and Figueiredo, 2017). Note that the operation of the proposed approach remains the same regardless of the embedding algorithm.

3.2.2 *Valid Optimal Assignment Kernel*

3.2.2.1 *Strong Kernels and Hierarchies*

Let $\mathbb{X}$, $\mathbb{X}'$ be two sets containing d-dimensional vectors. In our setting, $\mathbb{X}$ and $\mathbb{X}'$ are the *bag of vectors* representation of two graphs $\mathcal{G}$ and $\mathcal{G}'$. Note, however, that the two sets are not limited to embeddings of vertices, they may contain feature vectors extracted from any substructure of the two graphs. For simplicity, we assume that the size of both sets is the same.

Let $\mathfrak{B}(\mathbb{X}, \mathbb{X}')$ denote the set of all bijections between the two sets. We are interested in computing the overall similarity of the two sets by matching their elements as follow:

$$K_{\mathfrak{B}}(\mathbb{X}, \mathbb{X}') = \max_{\mathbb{B} \in \mathfrak{B}(\mathbb{X}, \mathbb{X}')} \sum_{(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{B}} \langle \boldsymbol{x}, \boldsymbol{x}' \rangle \tag{3.1}$$

A kernel (which we call *base kernel*) is employed to measure the similarity between the elements of the two sets. In the above definition, we use the linear kernel $k(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{x}, \boldsymbol{x}' \rangle$ as our base kernel.

Unfortunately, the above function $K_{\mathfrak{B}}(\mathcal{X}, \mathcal{X}')$ is not always a valid kernel as we mentioned in Section 2.3.4. More specifically, Kriege, Giscard, and Wilson (2016) show that $K_{\mathfrak{B}}(\mathbb{X}, \mathbb{X}')$ can be a valid kernel if the base kernel k is *strong*. They define a strong kernel as follow

DEFINITION 3.2 (STRONG KERNEL). *A function $k : \mathbb{X} \times \mathbb{X} \to \mathbb{R}_{\geq 0}$ is called a strong kernel if*

$$k(\boldsymbol{x}, \boldsymbol{y}) \geq \min\{k(\boldsymbol{x}, \boldsymbol{z}), k(\boldsymbol{z}, \boldsymbol{y})\}, \quad \forall \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{X}.$$

The authors also introduce another class of kernels that is derived from a *hierarchy* on the set $\mathbb{X} \cup \mathbb{X}'$. More specifically, let T be a rooted tree such that the leaves of T are the elements of $\mathbb{X} \cup \mathbb{X}'$. Let $\mathbb{V}(T)$ be the set of vertices of T. Each inner vertex v in T corresponds to a subset of $\mathbb{X} \cup \mathbb{X}'$ comprising all leaves of the subtree rooted at v. Let $w : \mathbb{V}(T) \to \mathbb{R}_{\geq 0}$ be a weight function such that $w(v) \geq w(p(v))$ for all v in T where $p(v)$ is the parent of vertex v. Then, the tuple (T, w) defines a *hierarchy*. Let $\textsc{lca}(u, v)$ be the lowest common ancestor of vertices u and v, that is, the unique vertex with maximum depth that is an ancestor of both u and v.

DEFINITION 3.3 (HIERARCHY-INDUCED KERNEL). *Let $H = (T, w)$ be a hierarchy on $\mathbb{X}$, then the function defined as $k(\boldsymbol{x}, \boldsymbol{y}) = w(\textsc{lca}(\boldsymbol{x}, \boldsymbol{y}))$ for all $\boldsymbol{x}, \boldsymbol{y}$ in $\mathbb{X}$ is the kernel on $\mathbb{X}$ induced by H.*

The authors show that the above two classes of kernels are equivalent.

THEOREM 3.2.1 (KRIEGE, GISCARD, AND WILSON, 2016). *A kernel k on $\mathbb{X}$ is strong if and only if it is induced by a hierarchy on $\mathbb{X}$.*

Sketch of proof. We can prove "a hierarchy-induced kernel is a strong kernel" by contradiction. Suppose a hierarchy-induced kernel is not strong, then there exists at least three leaves x, y, z satisfy $k(\boldsymbol{x}, \boldsymbol{y}) < min\{k(\boldsymbol{x}, \boldsymbol{z}), k(\boldsymbol{z}, \boldsymbol{y})\}$, i. e., $w(\textsc{lca}(\boldsymbol{x}, \boldsymbol{y})) < w(\textsc{lca}(\boldsymbol{x}, \boldsymbol{z}))$ and $w(\textsc{lca}(\boldsymbol{x}, \boldsymbol{y})) < w(\textsc{lca}(\boldsymbol{z}, \boldsymbol{y}))$. By definition of a tree, there is a unique path from $\boldsymbol{x}$ to the root that contains $\textsc{lca}(\boldsymbol{x}, \boldsymbol{z})$ and $\textsc{lca}(\boldsymbol{x}, \boldsymbol{y})$, and a unique path from $\boldsymbol{y}$ to the root that contains $\textsc{lca}(\boldsymbol{z}, \boldsymbol{y})$ and $\textsc{lca}(\boldsymbol{x}, \boldsymbol{y})$. Since $w(\cdot)$ is a decreasing function along the path from leave to root, we have $\textsc{lca}(\boldsymbol{x}, \boldsymbol{y})$ as the ancestor of both $\textsc{lca}(\boldsymbol{x}, \boldsymbol{z})$ and $\textsc{lca}(\boldsymbol{z}, \boldsymbol{y})$. Then we find the contradictory as there exists two paths from $\boldsymbol{z}$ to root: $\boldsymbol{z} \to \textsc{lca}(\boldsymbol{z}, \boldsymbol{y}) \to \textsc{lca}(\boldsymbol{x}, \boldsymbol{y}) \to$ root and $\boldsymbol{z} \to \textsc{lca}(\boldsymbol{x}, \boldsymbol{z}) \to \textsc{lca}(\boldsymbol{x}, \boldsymbol{y}) \to$ root.

We prove the inverse direction by construction. We can show that a hierarchy can be constructed from a strong kernel by successively inserting element of $\mathbb{X}$ into a tree. The idea is to insert the new element to the location closest to the the existing element to which it is the most similar under the given strong kernel, i. e., inserting a parent inner vertex to the tree so that the two nodes become child nodes to this vertex. □

The relation between *strong* kernel and *hierarchy-induced* kernel can also be demonstrated by the *ultrametric*.

DEFINITION 3.4 (ULTRAMETRIC). *A metric space $(\mathbb{X}, d)$ is said to be ultrametric if*

$$d(x, y) \leq \max\{d(x, z), d(z, y)\}, \quad \forall x, y, z \in \mathbb{X}$$

THEOREM 3.2.2 (ISMAGILOV, 1997). *Any ultrametric space admits an isometric embedding in a Hilbert space.*

In fact, for every *ultrametric* d on $\mathbb{X}$, we can find a rooted tree T with leaves $\mathbb{X}$ that satisfies the properties:

- d is the path length between leaves in T,
- the path lengths from a leaf to the root are all equal.

Theorem 3.2.2 tells us that for every *ultrametric*, there also exists an isometric embedding in the Hilbert space, from which we have an associated kernel k and the following relation,

$$\begin{aligned} d^2(x, y) = d^2_{\mathcal{H}}(x, y) &= \langle \phi_{\mathcal{H}}(x) - \phi_{\mathcal{H}}(y), \phi_{\mathcal{H}}(x) - \phi_{\mathcal{H}}(y) \rangle \\ &= k(x, x) + k(y, y) - 2k(x, y). \end{aligned}$$

Set $w(a) \coloneqq k(x, y)$ where a is a inner vertex of T and the LCA of leaves x and y, (T, w) is then a *hierarchy* and k is a (T, w)-induced kernel. k is also a *strong* kernel by the property of *ultrametric*. Precisely, for every (x, y, z) in $\mathbb{X}$, we have

$$d^2(x, y) \leq \max\{d^2(x, z), d^2(z, y)\},$$

which is the same with

$$\begin{aligned} k(x, x) + k(y, y) - 2k(x, y) \leq \max\{&k(x, x) + k(z, z) - 2k(x, z), \\ &k(z, z) + k(y, y) - 2k(z, y)\}. \end{aligned}$$

For the isometric embedding associated with the *ultrametrix*, we also have $k(x, x) = k(y, y) = k(z, z)$, see Ismagilov (1997, pp. 188). Thus, we obtain the *strong* property

$$k(x, y) \geq \min\{k(x, z), k(z, y)\}.$$

3.2.2.2 *Kernel Construction*

In the previous section, we learn that the design of a valid optimal assignment kernel is related to deriving a hierarchy and a strong kernel. In this section, we will propose our approach to find the hierarchy and using the hierarchy-induced (strong) kernel to build a valid kernel that is an optimal assignment between substructures of two graphs[1].

FROM VECTORS TO HIERARCHIES Since we are interested in computing a valid kernel that approximates the function defined in Equation 3.1, we design a hierarchy-induced kernel which approximates the linear kernel.

By constructing a hierarchy on the set of node embeddings, we can derive a strong kernel k and ensure that the emerging function $K^k_{\mathfrak{B}}(\mathbb{X}, \mathbb{X}')$ defined below is a valid kernel:

$$K^k_{\mathfrak{B}}(\mathbb{X}, \mathbb{X}') = \max_{B \in \mathfrak{B}(\mathbb{X}, \mathbb{X}')} \sum_{(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{B}} k(\boldsymbol{x}, \boldsymbol{x}') \tag{3.2}$$

To build the hierarchy, we resort to clustering. By embedding the vertices of all graphs into a common vector space and employing a clustering algorithm, we can create the tree T and then, by defining a valid weight function $w(\cdot)$, a strong kernel on the set of embeddings is directly produced.

Hierarchical clustering techniques, such as agglomerative clustering, are particularly attractive for this task, since they represent the hierarchy of clusters as a tree allowing us to directly generate T. Such methods, however, turn out to be problematic when the number of input graphs and their sizes (i. e., number of vertices) increase, since they require computing the similarities (i. e., inner products) between all vertices of all graphs. In such cases, hierarchical clustering techniques are not feasible. Instead, we generate the tree of the hierarchy by repeatedly performing a variant of k-means clustering which uses the inner product as a similarity measure, and which is known as *spherical* k-means (Dhillon and Modha, 2001).

Given a set of embeddings which are encoded as unit-norm vectors $\boldsymbol{x} \in \mathbb{R}^d$, and a parameter $k \in \mathbb{N}$, spherical k-means maximises the following function:

$$Q(\{\mathbb{C}_i\}_{i=1}^k) = \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in \mathbb{C}_i} \langle \boldsymbol{x}, \boldsymbol{c}_i \rangle$$

1 The substructures correspond to the vertices in our case

where $\mathbb{C}_1, \ldots, \mathbb{C}_k$ and $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_k$ are the k clusters and their normalised centroids, respectively.

Spherical k-means scales well to large number of samples, however, it converges to a local optimum and not necessarily to the global optimum. To generate the tree T, we initially perform spherical k-means on the set of embeddings from all graphs and we cluster them into k top-level groups. Then, the clustering is repeated recursively on each of these clusters until each cluster contains a single vertex. The number of levels L of the emerging tree T is equal to the length of its longest path.

If the initial corpus of embeddings is very large, for computational purposes, we may not execute the algorithm up to the point where each cluster contains a single vertex, but we may terminate it earlier. Each cluster of vertices corresponds to an inner vertex in the generated hierarchy. For example, the set of all data points corresponds to the root r of the tree T, while the k clusters that are produced from those data points correspond to the k children of the root vertex r. The clustering process, and hence, the structure of the emerging hierarchy depends on two parameters:

- the branching factor k,
- the number of levels in the tree L.

The former is a trade-off parameter between the the breadth and the depth of the tree T, while the latter can reduce the number of inner vertices.

Let T be the tree that emerges after performing the clustering process described above. Since a hierarchy H is a tuple (T, w), besides T, it is necessary to define a weight function $w : \mathbb{V}(T) \to \mathbb{R}_{\geq 0}$ such that $w(v) \geq w(p(v))$ for all v in T where $p(v)$ is the parent of vertex v.

We set the weights of all inner vertices of T equal to the minimum of the inner product between the data points of the cluster and its normalised centroid. Therefore, given, an inner node v that corresponds to a cluster $\mathbb{C}$ of data points, its weight is set equal to:

$$w(v) = \min_{x \in \mathbb{C}} \langle \boldsymbol{x}, \boldsymbol{c} \rangle$$

where $\boldsymbol{c}$ is the normalised centroid of the cluster. The weight of the leaf vertices is set equal to 1, that is, if v is a leaf, then $w(v) = 1$.

Let $\mathbb{S}$ be a set of data points and $\boldsymbol{\mu}$ their mean vector. Let also $\mathbb{C}_1, \ldots, \mathbb{C}_k$ and $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_k$ be k clusters and their centroids, respectively, that we obtain after running the spherical k-means clustering algorithm described above on the set $\mathbb{S}$. If v is the vertex of T that corresponds to $\mathbb{S}$ and $u_1, \ldots, u_k$ the vertices that

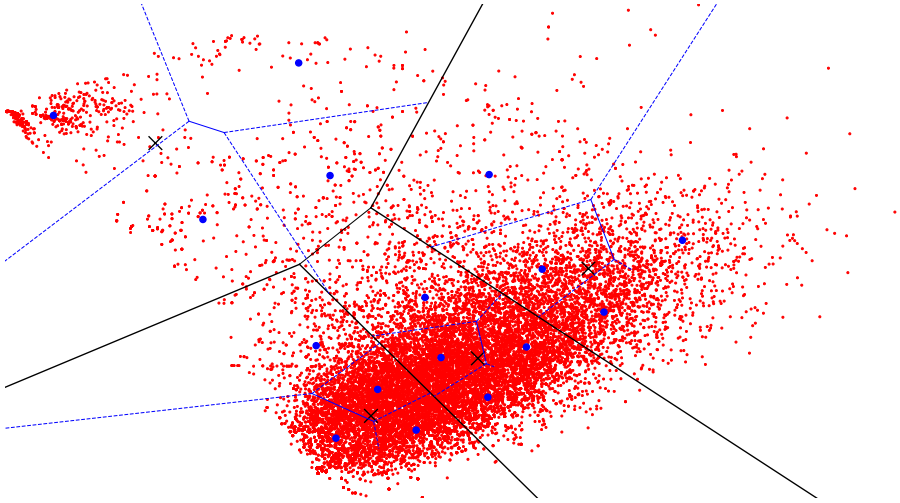

Figure 3.1: An illustration based on real data for the algorithm. The red points represent embeddings in the vector space and they are recursively separated by the dash lines. Blue points are the centroids.

correspond to $\mathbb{C}_1, \ldots, \mathbb{C}_k$, then, in order for $w(v) \leq w(u)$ to hold, it is necessary that

$$\min_{\boldsymbol{x} \in \mathbb{S}} \langle \boldsymbol{x}, \boldsymbol{\mu} \rangle \leq \min_{\boldsymbol{x} \in \mathbb{C}_i} \langle \boldsymbol{x}, \boldsymbol{c}_i \rangle, \quad \forall i = 1, \ldots, k.$$

Although this holds in almost all practical settings, there are some extreme cases where it may not hold. In such cases, we set the weight of the child equal to that of its parent. Note that the weight of all inner vertices of the tree T is at most equal to 1. The equality holds only if the data points of the cluster have equal coordinates. Hence, if v is a leaf and u an inner vertex, it holds $w(v) = 1 \geq w(u)$, and the emerging hierarchy $H = (T, w)$ is valid.

Algorithm 1 shows the pseudo code for constructing the hierarchy with spherical k-means. An illustration of this procedure based on real data is also given in Figure 3.1. Note that the idea of recursively dividing the embedding space is reminiscent of the Pyramid Matching Kernel (Nikolentzos, Meladianos, and Vazirgiannis, 2017) where the authors divide the space with uniform regular grids, which would fail to capture non-uniform distribution of graph embeddings, as shown by the irregular cluster of embeddings on the top-left corner of this illustration.

FEATURE MAP With the built hierarchy, we next derive the explicit feature map of the strong kernel k that is induced by the hierarchy $H = (T, w)$ defined above. Let $\omega : \mathbb{V}(T) \to \mathbb{R}_{\geq 0}$ be an additive weight function defined as $\omega(v) = w(v) - w(p(v))$ and $\omega(r) = w(r)$ for the root r. Note that the property of a hierarchy assures that the values of the $\omega(\cdot)$ function are non-negative. For

Algorithm 1: Hierarchy Construction by Spherical k-means

Data: A set of vectors $\mathbb{X} = \{\boldsymbol{x}_i\}$, branching factor k, tree max depth L.
Result: Constructed hierarchy $H = (T, w)$.
Initialisation: create cluster set $\mathbb{S}^0 = \emptyset$, create tree T with root r and set weight $w(r) = \min_{\boldsymbol{x}\in\mathbb{X}}\langle \boldsymbol{x}, \boldsymbol{\mu}\rangle$, where $\boldsymbol{\mu}$ is the mean vector of $\mathbb{X}$;
while $i \leq L$ **do**
  if $i = 0$ **then**
    Apply spherical k-means on $\mathbb{X}$;
    for $j = 1, \ldots, k$ **do**
      Add $\{\mathbb{C}_j^0, \boldsymbol{c}_j^0\}$ to $\mathbb{S}^0$;
      Add vertex c_j^0 to T and attach to r;
      Set $w(c_j^0) = \min_{\boldsymbol{x}\in\mathbb{C}_j^0}\langle \boldsymbol{x}, \boldsymbol{c}_j^0\rangle$;
    end
    Create cluster set $\mathbb{S}^1 = \emptyset$;
  else
    for *every* $(\mathbb{C}_i^{L-1}, \boldsymbol{c}_i^{L-1}) \in \mathbb{S}^{L-1}$ **do**
      Apply Spherical k-means on C_i^{L-1};
      for $j = 1, \ldots, k$ **do**
        Add $\{\mathbb{C}_{i,j}^{L-1}, \boldsymbol{c}_{i,j}^{L-1}\}$ to $\mathbb{S}^L$;
        Add vertex $c_{i,j}^L$ to T and attach to c_i^{L-1};
        Set $w(c_{i,j}^L) = \min_{\boldsymbol{x}\in\mathbb{C}_{i,j}^L}\langle \boldsymbol{x}, \boldsymbol{c}_{i,j}^{L-1}\rangle$;
      end
    end
  end
end

$v \in \mathbb{V}(T)$ let $P(v) \subseteq \mathbb{V}(T)$ denote the vertices on the path from v to the root r. The strong kernel k induced by the hierarchy H can be defined using the mapping $\phi : \mathbb{X} \to \mathbb{R}^n$, where $n = |\mathbb{V}(T)|$ and the components indexed by $v \in \mathbb{V}(T)$ are

$$\phi(v) = \begin{cases} \sqrt{\omega(u)} & \text{if } u \in P(v), \\ 0 & \text{otherwise.} \end{cases} \tag{3.3}$$

PROPOSITION 3.2.3. *$\phi(\cdot)$ defined in equation 3.3 is a feature map corresponding to the strong kernel $k : \mathbb{X} \times \mathbb{X} \mapsto \mathbb{R}$, such that:*

$$k(\boldsymbol{x}, \boldsymbol{y}) = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y})\rangle_{\mathbb{R}^n} \quad \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{X}$$

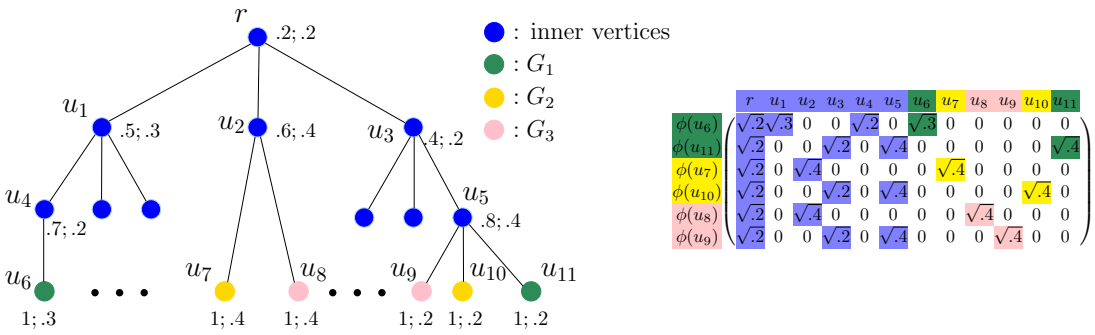

Figure 3.2: **Left:** An example of a hierarchy where each vertex v is annotated by its weights $w(v) : \omega(v)$ and its colour indicates the graph to which it belongs; **Right:** the derived feature vectors.

Proof. By definition,

$$k(\boldsymbol{x}, \boldsymbol{y}) = w(\text{LCA}(\boldsymbol{x}, \boldsymbol{y})) = \sum_{v \in P(\boldsymbol{x}) \cap P(\boldsymbol{y})} \omega(v)$$

From equation 3.3, we have also

$$[\phi(\boldsymbol{x})]_v [\phi(\boldsymbol{y})]_v = \begin{cases} \omega(v) & \text{if } v \in P(\boldsymbol{x}) \cap P(\boldsymbol{y}), \\ 0 & \text{otherwise.} \end{cases}$$

Thus $\phi(x)^\intercal \phi(y) = \sum_{v \in P(\boldsymbol{x}) \cap P(\boldsymbol{y})} \omega(v) = k(\boldsymbol{x}, \boldsymbol{y})$. □

Being the mapping of vertices to the Hilbert space, $\phi(\cdot)$ can also be seen as an embedding method for nodes, which could be useful for some tasks, e. g., Link Prediction, which we will elaborate in Section 3.2.3.2.

Figure 3.2 shows an example of a valid hierarchy and the derived feature vectors.

So far, we have assumed that the embeddings are unit vectors with non-negative components. When dealing with such kind of vectors, there is a close relationship between Euclidean distances and inner products.

LEMMA 3.2.4. *Let $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ be three non-negative valued unit vectors, $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{R}^d$. Then, it holds that*

$$\|\boldsymbol{x} - \boldsymbol{y}\|_2 \geq \|\boldsymbol{x} - \boldsymbol{z}\|_2 \quad \textit{if and only if} \quad \langle \boldsymbol{x}, \boldsymbol{y} \rangle \leq \langle \boldsymbol{x}, \boldsymbol{z} \rangle.$$

Proof. For a unit vector $\boldsymbol{x}$ it holds that $\|\boldsymbol{x}\|_2 = \sqrt{\langle \boldsymbol{x}, \boldsymbol{x} \rangle} = 1$ and therefore, $\langle \boldsymbol{x}, \boldsymbol{x} \rangle = 1$. The Euclidean distance between two unit vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is defined as

$$\begin{aligned} \|\boldsymbol{x} - \boldsymbol{y}\|_2 &= \sqrt{\langle \boldsymbol{x}, \boldsymbol{x} \rangle - 2\langle \boldsymbol{x}, \boldsymbol{y} \rangle + \langle \boldsymbol{y}, \boldsymbol{y} \rangle} \\ &= \sqrt{2(1 - \langle \boldsymbol{x}, \boldsymbol{y} \rangle)}. \end{aligned}$$

Then, if we know that $\boldsymbol{z}$ is closer to $\boldsymbol{x}$ than $\boldsymbol{y}$ to $\boldsymbol{x}$, we have:

$$\begin{aligned} & \|\boldsymbol{x} - \boldsymbol{y}\|_2 \geq \|\boldsymbol{x} - \boldsymbol{z}\|_2 \\ \Leftrightarrow \quad & \sqrt{2(1 - \langle \boldsymbol{x}, \boldsymbol{y} \rangle)} \geq \sqrt{2(1 - \langle \boldsymbol{x}, \boldsymbol{z} \rangle)} \\ \Leftrightarrow \quad & 2(1 - \langle \boldsymbol{x}, \boldsymbol{y} \rangle) \geq 2(1 - \langle \boldsymbol{x}, \boldsymbol{z} \rangle) \\ \Leftrightarrow \quad & \langle \boldsymbol{x}, \boldsymbol{y} \rangle \leq \langle \boldsymbol{x}, \boldsymbol{z} \rangle. \end{aligned}$$

□

Hence, for unit vectors, the squared Euclidean distance is inversely proportional to the inner product.

We next give a lower bound on the inner product between the data points of a cluster.

PROPOSITION 3.2.5. *Let $\mathbb{C}$ be the set of points of a cluster and $\boldsymbol{c}$ its centroid. Let also $\boldsymbol{x}, \boldsymbol{y}$ be any two points of $\mathbb{C}$. Then, it holds that*

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle \geq 4 \min_{\boldsymbol{z} \in \mathbb{C}} \langle \boldsymbol{z}, \boldsymbol{c} \rangle - 3.$$

Proof. From Lemma 3.2.4, it is clear that the lowest inner product between the data points of a cluster is the one between those whose distance is maximum. Let $\boldsymbol{v}, \boldsymbol{w}$ be these two data points. Let also $\boldsymbol{u}$ be the data point whose distance from the centroid of the cluster $\boldsymbol{c}$ is maximum,

$$\boldsymbol{u} = \arg\max_{\boldsymbol{z} \in \mathbb{C}} \|\boldsymbol{z} - \boldsymbol{c}\|_2 = \arg\min_{\boldsymbol{z} \in \mathbb{C}} \langle \boldsymbol{z}, \boldsymbol{c} \rangle,$$

and let d^* be that distance. Then,

$$\|\boldsymbol{u} - \boldsymbol{c}\|_2 = \sqrt{2(1 - \langle \boldsymbol{u}, \boldsymbol{c} \rangle)} = d^*.$$

Note that this data point is not necessarily one of the two most distant data points v and w. Clearly, the distance between v and w will be at most $2d^*$. Then, for any two points $x, y \in \mathbb{C}$,

$$\begin{aligned} \|x - y\|_2 = \sqrt{2(1 - \langle x, y\rangle)} &\leq \|v - w\|_2 \\ &\leq 2d^* \\ &= 2\sqrt{2(1 - \langle u, c\rangle)}. \end{aligned}$$

Therefore,

$$\begin{aligned} & \sqrt{2(1 - \langle x, y\rangle)} \leq 2\sqrt{2(1 - \langle u, c\rangle)} \\ \Leftrightarrow \quad & 2(1 - \langle x, y\rangle) \leq 8(1 - \langle u, c\rangle) \\ \Leftrightarrow \quad & \langle x, y\rangle \geq 4\langle u, c\rangle - 3. \end{aligned}$$

□

The above bound is useful especially for clusters that appear at the low levels of the hierarchy. For such clusters that contain a small number of vertices and the pairwise inner products of all their points are very high, the bound becomes tight. Hence, for data points that are sufficiently close to each other and appear together in the same cluster, we can accurately estimate the real values of the inner products. Since we are interested in matching highly similar data points, in cases where such pairs of data points exist, the proposed kernel can approximate accurately the function shown in equation 3.1, and remain at the same time positive semi-definite.

KERNEL COMPUTATION Let $H = (T, w)$ be a hierarchy on the set of the embeddings of all graphs $\mathbb{X}$. The hierarchy H induces a strong kernel k with feature mapping $\phi(\cdot)$. The kernel $K^k_{\mathfrak{B}}$ defined in equation 3.2 can be computed in linear in the number of vertices of the tree T time by histogram intersection.

DEFINITION 3.5 (HISTOGRAM). *The histogram of a graph with set of vertices $\mathbb{X}$ generated by the feature mapping $\phi(\cdot)$ is*

$$H_{\mathbb{X}} = \sum_{x \in \mathbb{X}} \phi(x)^{\intercal} \phi(x).$$

An illustration is given in Figure 3.3.

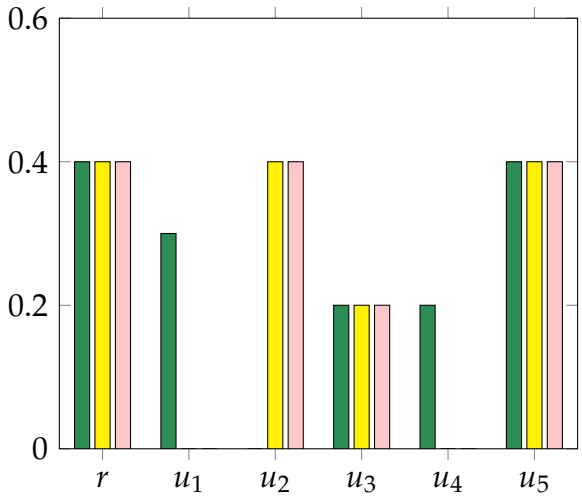

Figure 3.3: Histograms of graph G_1, G_2, G_3 corresponding to the feature mapping in Figure 3.2.

THEOREM 3.2.6 (KRIEGE, GISCARD, AND WILSON, 2016). *Given the hierarchy $H = (T, w)$, two graphs $\mathcal{G}$, $\mathcal{G}'$ and their histograms $\boldsymbol{H}_{\mathbb{X}}$, $\boldsymbol{H}_{\mathbb{X}'}$ computed as above, we define the histogram intersection kernel as,*

$$K(\mathcal{G}, \mathcal{G}') = \sum_{i=1}^{|\mathbb{V}(T)|} \min\left([\boldsymbol{H}_{\mathbb{X}}]_i, [\boldsymbol{H}_{\mathbb{X}'}]_i\right).$$

Then we have,

$$K(\mathcal{G}, \mathcal{G}') = K^k_{\mathfrak{B}}(\mathbb{X}, \mathbb{X}')$$

Proof. As $K^k_{\mathfrak{B}}(\mathbb{X}, \mathbb{X}') = \max_{\mathbb{B}\in\mathfrak{B}(\mathbb{X},\mathbb{X}')} \sum_{(\boldsymbol{x},\boldsymbol{x}')\in\mathbb{B}} k(\boldsymbol{x}, \boldsymbol{x}')$ and $k(\cdot,\cdot)$ is a strong kernel, we can rewritten the equation as,

$$K^k_{\mathfrak{B}}(X, X') = \max_{\mathbb{B}\in\mathfrak{B}(\mathbb{X},\mathbb{X}')} \sum_{(\boldsymbol{x},\boldsymbol{x}')\in\mathbb{B}} \sum_{v\in P(\boldsymbol{x})\cap P(\boldsymbol{x}')} \omega(v) = \max_{\mathbb{B}\in\mathfrak{B}(\mathbb{X},\mathbb{X}')} \sum_{v\in\mathbb{V}(T)} c^{\mathbb{B}}_v \omega(v).$$

where $c^{\mathbb{B}}_v$ is the times of repetition of tree node v in the assignment $\mathbb{B}$.

We have also histogram intersection kernel as,

$$K_{\boldsymbol{H}}(\mathbb{X}, \mathbb{X}') = \sum_{i=1}^{|\mathbb{V}(T)|} \min\left([\boldsymbol{H}_{\mathbb{X}}]_i, [\boldsymbol{H}_{\mathbb{X}'}]_i\right) = \sum_{v\in\mathbb{V}(T)} \min\left(|\mathbb{X}_v|, |\mathbb{X}'_v|\right)\omega(v).$$

Denote $c_v = \min\left(|\mathbb{X}_v|, |\mathbb{X}'_v|\right)$. As $c^{\mathbb{B}}_v$ counts only when v is common ancestor of $\boldsymbol{x}$ and $\boldsymbol{x}'$ (and for all $\boldsymbol{x} \in \mathbb{X}$, $\boldsymbol{x}' \in \mathbb{X}'$ only counted once), while c_v counts when v is an ancestor of $\boldsymbol{x}$ or $\boldsymbol{x}'$, we have naturally $c^{\mathbb{B}}_v \leq c_v, \forall\mathbb{B} \in \mathfrak{B}$. Thus the histogram intersection kernel is the upper bound of the optimal assignment

kernel defined in equation 3.2 and the upper bound is reached when we have the optimal assignment $\mathbb{B}$. □

The histogram intersection kernel is known to be a valid kernel on $\mathbb{R}^n$ (Barla, Odone, and Verri, 2003; Swain and Ballard, 1991). Hence, the proposed optimal assignment kernel is also a valid kernel. Its complexity depends on the size of the tree T, which, as mentioned above, depends on the L parameter. Thus, by setting L to small values, we can reduce the number of vertices of T, and therefore, the complexity of the kernel.

More concretely, given a pair of graphs $\mathcal{G}, \mathcal{G}' \in \mathbb{G}$, let $\mathbb{X}, \mathbb{X}' \subseteq \mathbb{X}$ be their sets of vertex embeddings and $\boldsymbol{H}_{\mathbb{X}}$ and $\boldsymbol{H}_{\mathbb{X}'}$ their histograms. The size of these histograms is equal to the number of vertices of the tree T (denoted by n), and are generated as follows for each graph: each embedding $\boldsymbol{x} \in \mathbb{X}$ corresponds to a leaf v of the tree T and there is a single path from the root r to that leaf. Let $P_{\boldsymbol{x}}$ denote the set of vertices in that path. For each embedding of the graph and for each vertex $u \in P_{\boldsymbol{x}}$, we increase the value of the component of $\boldsymbol{H}_{\mathbb{X}}$ corresponding to vertex u by $\omega(u)$. Then, to compute the kernel between the two graphs, we have,

$$K(\mathcal{G}, \mathcal{G}') = \sum_{i=1}^{n} \min\left([\boldsymbol{H}_{\mathbb{X}}]_i, [\boldsymbol{H}_{\mathbb{X}'}]_i\right).$$

By Theorem 3.2.6, this kernel is the optimal assignment kernel $K^k_{\mathfrak{B}}(\mathbb{X}, \mathbb{X}')$. In what follows, we denote the kernel described above by E-OA.

Besides E-OA, we also computed a variant of it, by designing an alternative hierarchy $H' = (T', w')$. Specifically, to create the tree T', we performed k-means on the unnormalised embeddings (instead of spherical k-means on the normalised embeddings). We defined the weighting function $w'(\cdot)$ implicitly by directly defining an $\omega'(\cdot)$ function.

Regarding the ω' value of the root r, we set it equal to 1. For each vertex $v \in \mathbb{V}(T')$, we directly set $\omega'(v) = \frac{d(v,r)-1}{d(v,r)}$ where $d(v, r)$ is the length of the path between the root r and vertex v. Hence, $\omega'(\cdot)$ weights more vertices appearing lower in the hierarchy than those appearing higher. We denote this variant of the proposed kernel by E-OA-SP.

E-OA-SP seems naive compared to E-OA, since it does not take into account the true similarity between matched embeddings. For example, given two pairs of matched embeddings where the lowest common ancestor of both pairs is at the same level of the hierarchy H', the kernel values of the two pairs are considered to be equal to each other regardless of their true similarity.

3.2.3 *Extensions*

3.2.3.1 *Text Categorisation*

Our approach is not limited to the graphs. It is straightforward to apply it to any case where we need to deal with a set of vectors, e. g., in text mining, we can represent the document as a set of words (vectors). Efficiency of our method in this task will be demonstrated in Section 3.3.3.

3.2.3.2 *Link Prediction*

As stated in Section 2.2.2, the Link Prediction problem is that given an observed network, we want to predict the likelihood of the existence of potential links between two nodes. It has different concrete meanings in different scenarios: in Social Networks, we predict the future association between two nodes (evolution of network); in Biology, we infer links that are likely to exist, but are missing in the observation (invisible) due to certain constraints, e. g., observation noise, resolution of the equipments, etc.

Most common method to tackle this problem is by heuristics, based on the hypothesis that if two nodes share neighbourhood with similar properties, they are likely to be connected. Here, we focused on two heuristics:

- Adamic-Adar Index. $L(u,v) = \sum_{w \in \mathcal{N}(u) \cap \mathcal{N}(v)} \frac{1}{\log |\mathcal{N}(w)|}$
- Jaccard Index. $L(u,v) = \frac{|\mathcal{N}(u) \cap \mathcal{N}(v)|}{|\mathcal{N}(u) \cup \mathcal{N}(v)|}$

where $\mathcal{N}(\cdot)$ denotes the neighbourhood of the node.

Grover and Leskovec (2016) propose that this problem could be formulated as a supervised problem, that given an observed network, we could use the features of existing edges as the input to train a model and then use the model to predict the existence (or nonexistence) of potential links. The feature of an edge can be represented by a combination of its two ends. In Grover and Leskovec (2016), they propose that

$$f(e) = \boldsymbol{x}_v \odot \boldsymbol{x}_w \quad \forall v, w \in \mathbb{V}, e = (v,w) \in \mathbb{E}.$$

where $\boldsymbol{x}$ is the vertex embedding, $\odot$ is the Hadamard product and $f(e)$ is the feature of edge e (or edge embedding).

Seeing the feature map $\phi(\cdot)$ of our approach as a new type of node embedding, it is also useful in Link Prediction problems. With the hierarchy $H = (T, w)$

derived from the original embeddings $\{\boldsymbol{x}\}$, one may naturally think of representing an edge by the path between its two ends in the tree T. This intuition can be unified with Grover and Leskovec (2016)'s framework as:

PROPOSITION 3.2.7. *Define*

$$f(e) = \big(\phi(v) \odot \phi(v)\big) \oplus \big(\phi(w) \odot \phi(w)\big) \quad \forall v, w \in \mathbb{V}, e = (v, w) \in \mathbb{E},$$

where $\phi(\cdot)$ is the vertex embedding induced by the feature map in Section 3.2.2.2, $\oplus$ is the exclusive dis-junction operator (XOR). Then $f(e)$ can be written as

$$f(e = (v, w)) = \begin{cases} \omega(u) & \text{if } u \in \overline{P(v) \cap P(w)}, \\ 0 & \text{otherwise.} \end{cases}$$

which is the path between a and b in tree T.

We demonstrate empirically in Section 3.3.2 that $f(e)$ is also an expressive representation of edges in Link Prediction problem. The common evaluation procedure is that with a given network, we mask a certain percentage of its edges, use edges in residual network and a set of randomly-sampled non-edges[2] as training data, and evaluate by the performance of the predictive model on the existence of previously masked edges. Details can be found in Section 3.3.2.

3.3 EXPERIMENTS AND DISCUSSION

In this section, we empirically evaluate our proposed kernel on real-world data, of three different tasks: graph classification, network link prediction and text categorisation, and compare with several baseline methods.

3.3.1 *Graph Classification*

3.3.1.1 *Datasets*

We evaluate the proposed framework on 10 publicly available graph classification datasets from the TU benchmark, including 5 bioinformatics datasets: MUTAG, ENZYMES, NCI1, PTC-MR and D&D, as well as 5 social network datasets: IMDB-BINARY, IMDB-MULTI, REDDIT-BINARY, REDDIT-MULTI-5K and REDDIT-MULTI-12K.

2 They serve as negative samples, usually the same number with the existing edges

Note that the bioinformatics datasets come with vertex labels, however, we do not take these labels into account in our experimental evaluation. The proposed approach can be easily modified to account for such labels by creating a separate histogram for vertices sharing each label, and computing a different kernel for each type of vertices. Then, the overall kernel can be obtained as the sum of the separate kernels. The summary of statistics of the 10 datasets can be found in Table 2.1.

3.3.1.2 *Experiment Setup*

We compare the proposed kernel against several state-of-the-art graph kernels. More specifically, our baseline competitors are the graphlet kernel (GL) (Shervashidze et al., 2009), the shortest path kernel (SP) (Borgwardt and Kriegel, 2005), the Weisfeiler-Lehman subtree kernel (WL) Shervashidze et al., 2011, the Weisfeiler-Lehman optimal assignment kernel (WL-OA) Kriege, Giscard, and Wilson, 2016, and the pyramid match graph kernel (PM) (Nikolentzos, Meladianos, and Vazirgiannis, 2017).

It should be mentioned that the proposed kernel is very related to the pyramid match graph kernel. However, instead of partitioning the feature space into uniformly-shaped regions, the proposed kernel uses an adaptive grid whose boundaries depend on the distribution of the input data (i. e., node embeddings), allowing it to remain accurate even in the case of high-dimensional data.

To perform graph classification, we employ the LIBSVM implementation of the C-SVM classifier (Chang and Lin, 2011) and perform 10-fold cross-validation. The whole process was repeated 10 times with random fold assignments for each dataset and each kernel. Within each fold all necessary parameters, such as the parameter C of the SVM and any parameters of the kernels, were selected by cross-validation based on the training set.

We chose parameters for the graph kernels as follows. For the Weisfeiler-Lehman subtree kernel and for the Weisfeiler-Lehman optimal assignment kernel, we chose the number of iterations from $h = \{4, 5, 6, 7\}$. For the pyramid match graph kernel, the dimensionality of the embeddings was chosen from $d = \{4, 6, 8, 10\}$ and the number of levels was selected from $L = \{3, 4, 5, 6\}$. Note that the Weisfeiler-Lehman kernels assume node-labelled graphs. Since the graphs contained in our datasets are either unlabelled or we ignore the node labels, we set the label of each vertex equal to its degree. The graphlet kernel that we implemented samples 500 graphlets of size up to 6 from each graph, while the shortest path kernel is a Dirac kernel that counts shortest paths of equal length. As regards the proposed kernel, the dimensionality of

the embeddings was chosen from $d = \{4, 6, 8, 10\}$, the branching factor k of the clustering procedure was chosen from $k = \{2, 5, 10, 20\}$, while we did not restrict the depth of the generated trees (i. e., using parameter L). The proposed kernel was implemented in Python based on Scikit-learn (Pedregosa et al., 2011).

3.3.1.3 *Results*

Table 3.1 illustrates average prediction accuracy and standard deviations. We observe that the proposed assignment kernels that compute a correspondence between node embeddings outperform the baselines on 4 out of the 10 datasets, while they provide the second or third best accuracy on the rest of the datasets. The most successful kernel is WL-OA which performs best on 5 of the 10 datasets, Furthermore, on 3 of them it outperforms the proposed kernels with quite wide margins (on the other 2, the proposed kernels are the second best). Note, however, that we could have combined the proposed kernels with the WL relabelling procedure to also improve their performance.

On most datasets, GL and SP fail to yield accuracy comparable to those of the optimal assignment kernels. This indicates that optimal assignment kernels better capture the similarity between graphs than R-convolution kernels in the considered classification tasks.

As regards the two variants of the proposed kernel, quite surprisingly, E-OA-SP outperforms E-OA on most datasets. One possible explanation for this is that some useful information about the structure of the graph is lost due to the normalisation of the embeddings.

3.3.2 *Link Prediction*

3.3.2.1 *Datasets*

The performance of our proposed kernel on the link prediction task is evaluated on 8 real-world network datasets, same as in Zhang and Chen (2017). They are USAir, NS, PB, Yeast, Celegans, Power, Router, and E.coli. USAir, Power and Router are real networks: USAir is the US Air lines network; Power is the electrical grid network in western US and Router is a router-level Internet network. NS and PB are social networks with NS being the collaboration network of researchers who publish papers on network science and PB being a network of political blogs in the US. The rest are biological networks: Yeast is the protein-protein interaction network in yeast, Celegans is the neural network of Caenorhabditis elegans and E.coli is the pairwise reaction network of metabo-

Table 3.1: 10-fold cross validation accuracy - mean (± standard deviation) - of the graphlet kernel (GL), shortest path kernel (SP), Weisfeiler-Lehman subtree kernel (WL), Weisfeiler-Lehman optimal assignment kernel (WL-OA), pyramid match graph kernel (PM), and the two variants of the proposed kernel that compute a correspondence between sets of embeddings (E-OA-SP and E-OA) on the 10 graph classification datasets. We set the best results to **bold** and underline the second best ones.

Method \ Datasets	MUTAG	ENZYMES	NCI1	PTC-MR	D&D
GL	80.29 (± 0.70)	22.18 (± 0.74)	62.52 (± 0.14)	55.71 (± 0.19)	74.55 (± 0.36)
SP	83.79 (± 1.09)	28.86 (± 0.94)	61.85 (± 0.11)	56.63 (± 0.59)	76.02 (± 0.37)
WL	80.84 (± 1.87)	39.98 (± 0.98)	78.03 (± 0.10)	55.99 (± 0.84)	74.65 (± 0.47)
WL-OA	81.13 (± 2.20)	**40.36** (± 2.30)	**81.22** (± 0.41)	55.47 (± 0.98)	**76.44** (± 0.33)
PM	82.90 (± 1.40)	28.65 (± 0.72)	66.17 (± 0.19)	55.44 (± 1.12)	75.40 (± 0.60)
E-OA-SP	86.64 (± 0.64)	34.98 (± 1.34)	75.25 (± 0.32)	**59.37** (± 1.76)	76.15 (± 0.22)
E-OA	**87.64** (± 0.73)	33.23 (± 0.82)	71.41 (± 0.43)	56.85 (± 1.05)	75.69 (± 0.21)

Method \ Datasets	IMDB BINARY	IMDB MULTI	REDDIT BINARY	REDDIT MULTI-5K	REDDIT MULTI-12K
GL	60.33 (± 0.25)	36.53 (± 0.93)	76.15 (± 0.21)	35.41 (± 0.12)	22.52 (± 0.15)
SP	60.21 (± 0.58)	39.62 (± 0.57)	83.60 (± 0.18)	49.13 (± 0.14)	35.96 (± 0.08)
WL	73.36 (± 0.38)	**51.06** (± 0.47)	75.12 (± 0.44)	49.33 (± 0.28)	33.68 (± 0.10)
WL-OA	**73.61** (± 0.60)	50.48 (± 0.33)	79.34 (± 0.43)	**53.33** (± 0.25)	44.12 (± 0.13)
PM	67.91 (± 0.98)	45.03 (± 0.77)	82.35 (± 0.52)	43.04 (± 0.46)	37.98 (± 0.16)
E-OA-SP	69.16 (± 0.43)	30.47 (± 0.92)	**90.67** (± 0.21)	50.68 (± 0.31)	**44.26** (± 0.08)
E-OA	64.71 (± 0.56)	44.58 (± 1.16)	87.92 (± 0.12)	47.94 (± 0.47)	42.80 (± 0.22)

Table 3.2: Statistics of 8 real-word networks for Link Prediction

	USAir	NS	PB	Yeast	Celegans	Power	Router	E.coli
#Nodes	332	1,589	1,222	2,375	297	4,941	5,022	1,805
#Edges	2,126	2,742	16,714	11,693	2,148	6,594	6,258	14,660

lites in Escherichia coli. A summary of statistics of these datasets is shown in Table 3.2.

3.3.2.2 *Experiment Setup*

We compared our proposed kernel with two commonly used heuristics that we introduce in Section 3.2.3.2: Adamic-Adar index and Jaccard index. We also compared with the framework proposed in Grover and Leskovec (2016) where they use hadamard product over node embeddings to generate vector representation for edges. Three different node embedding methods were implemented for this task in our experiments: wavelet2vec (a spectra-based algorithm named **GraphWave** in Donnat et al. (2018)), node2vec (Grover and Leskovec, 2016), and eigen-vectors of adjacency matrix. These methods are tested in the case that we removed 10%, 50% and 75% edges of the original networks. Their performances were then evaluated under a supervised learning setting, where we employed logistic regression classifier and Area Under Curve (AUC) score as the metric. Hyper-parameters of the logistic regression classifier were optimised using 10-fold cross validation, same with the graph classification task.

3.3.2.3 *Results*

Table 3.3 shows the results of our experiments. One general conclusion we can make is that our framework, the EOA embedding combined with "xor" operator performs better that the original node embedding combined with hadamard product in term of AUC score, sometimes with up to 20% improvement (see **Router**). It achieve the best results on most of the experiments, especially in cases when we have little information (with 50%/75% edges removed), where the embedding methods tend to perform more robustly. Among the embeddings methods, node2vec seems to have a relatively stable performance over all datasets, potentially because it is a method that depend less on the one-hop neighbourhood structure (adjacency matrix).

Table 3.3: AUC scores of heuristics (Adamic-adar, Jaccard) and node embedding frameworks (EOA embedding combined with "xor", original embedding with hadamard product) for Link Prediction with different level of edge removal on 8 real world datasets. The format follows Table 3.1.

Method \ Datasets		USAir			Power			Yeast			Celegans		
		10%	50%	75%	10%	50%	75%	10%	50%	75%	10%	50%	75%
Adamic-adar		**97.22**	91.14	71.41	62.23	60.56	60.40	91.15	84.44	63.89	**84.25**	74.44	59.32
Jaccard		92.84	86.00	69.88	62.23	60.56	60.40	90.93	84.25	63.84	76.51	70.81	58.69
Wavelet2Vec	EOA-xor	93.05	90.97	**88.61**	58.68	58.55	59.46	88.55	86.21	79.35	74.36	72.64	**70.83**
	hadamard	84.18	82.17	75.45	59.80	55.67	55.94	86.93	84.93	77.24	62.75	60.72	56.51
Node2Vec	EOA-xor	95.32	91.02	85.87	80.75	74.76	75.02	**95.37**	**93.20**	86.39	83.40	**79.09**	69.58
	hadamard	88.14	82.51	68.91	**86.74**	**81.68**	**82.10**	94.60	92.82	**86.58**	77.39	73.35	61.95
Adjacency	EOA-xor	93.00	**91.57**	88.05	81.88	80.15	80.97	90.84	88.44	81.37	75.19	72.74	69.31
	hadamard	92.37	89.61	86.72	64.94	70.96	69.79	87.92	86.82	79.83	76.04	71.92	67.54

Method \ Datasets		PB			NS			Router			Ecoli		
		10%	50%	75%	10%	50%	75%	10%	50%	75%	10%	50%	75%
Adamic-adar		**93.36**	87.01	73.87	99.62	83.53	73.93	67.10	52.94	52.97	**96.84**	89.53	78.03
Jaccard		88.46	83.10	72.32	99.61	83.53	73.92	67.02	52.93	52.96	81.32	83.28	75.66
Wavelet2Vec	EOA-xor	90.89	**90.12**	**89.34**	93.67	85.62	86.11	93.38	**92.14**	**91.47**	94.17	93.11	**91.34**
	hadamard	81.22	81.03	75.75	88.64	81.49	78.65	81.37	71.17	73.63	90.11	87.63	82.54
Node2Vec	EOA-xor	91.46	87.89	84.80	**99.63**	97.45	95.51	**96.96**	88.78	88.95	95.52	**93.27**	89.44
	hadamard	84.85	80.17	75.09	99.34	**98.71**	**97.93**	92.01	81.95	81.01	86.21	79.93	71.01
Adjacency	EOA-xor	91.07	89.72	88.49	91.00	90.02	87.79	91.89	76.21	81.05	94.55	93.14	90.99
	hadamard	91.25	89.63	87.67	75.65	77.08	67.64	88.35	63.48	66.63	93.98	92.37	88.35

Table 3.4: Statistics of 5 real-word networks for text categorisation. CV stands for *Cross Validation*.

	BBCSport	Subjectivity	Polarity	TREC	Twitter
#Training examples	348	10,000	10,662	5,452	3,115
#Test examples	389	CV	CV	500	CV
Vocabulary size	14,340	21,335	18,777	9,513	6,266
#Classes	5	2	2	6	3

3.3.3 *Text Categorisation*

3.3.3.1 *Datasets*

To demonstrate the versatility of the proposed kernel to compare any type of objects represented as sets of embeddings, we applied it to the text categorisation problem. We evaluated the variants of the proposed kernel on five standard datasets for text categorisation:

- BBCSport (Greene and Cunningham, 2006), which consists of sports news articles from the BBC Sport website.
- Subjectivity (Pang and Lee, 2004), which consists of subjective sentences gathered from the Rotten Tomatoes website and objective sentences gathered from the Internet Movie Database.
- Polarity (Pang and Lee, 2004), which contains positive and negative snippets acquired from the Rotten Tomatoes website.
- TREC (Li and Roth, 2002), which consists of a set of questions classified into 6 different types.
- Twitter (Sanders, 2011), which contains a set of tweets labelled by its sentiment.

A summary of statistics of the 5 text categorisation datasets can be found in Table 3.4.

3.3.3.2 *Experiment Setup*

In this experiment, each document is represented as a set of word embeddings, which are obtained from a publicly available pretrained model consisting of 300-dimensional vectors (Mikolov et al., 2013). Words that are not contained in this model are initialised to random vectors. Besides pretrained vectors, we also used randomly initialised vectors, denoted as "RAND-OA".

Table 3.5: Classification accuracy of the 3 variants of the proposed kernel (using pretrained and randomly initialised embeddings), the bag-of-words representation with TF-IDF weights (BOW TF-IDF) and the centroid representation (CR) on the 5 text categorisation datasets. The format follows Table 3.1.

Method \ Datasets	BBCSport	Subjectivity	Polarity	TREC	Twitter
BOW TF-IDF	98.38	90.67	77.14	97.00	75.12
CR	**99.59**	90.90	77.79	96.60	72.65
RAND-OA	96.08	89.89	75.72	97.00	75.25
E-OA-SP	99.05	91.25	76.96	97.00	75.41
E-OA	99.45	**91.92**	**77.87**	**97.80**	**76.34**

We compare "RAND-OA" along with the other two *pretrained vectors*-based variants against

BOW TF-IDF where each document is represented as a "Bag Of Words" vector with the entry being 1 if and only if the corresponding word presents in the document, otherwise 0. This vector is then weighted by the Term Frequency–Inverse Document Frequency (TF-IDF) (Leskovec, Rajaraman, and Ullman, 2014, pp. 8-9).

CR where each document is projected into the word embedding space as the centroid of the point cloud that its words form, i. e., the mean vector of the embeddings of a document's terms.

To perform graph classification, both baselines are combined with a linear SVM classifier.

3.3.3.3 *Results*

Table 3.5 illustrates the performance of the proposed kernel and the baselines on the five datasets.

E-OA outperforms the other methods on all datasets except one (BBCSport) where CR yields the best performance. In contrast to the graph classification task, in this setting, E-OA outperforms E-OA-SP on all datasets. BOW TF-IDF achieves very good accuracy on all datasets, considering that it does not utilise word embeddings.

Our baseline kernel with all word embeddings randomly initialised (RAND-OA) is generally outperformed by the two kernels that use pretrained embeddings. However, on the TREC and Twitter datasets, it performs comparably to the

rest of the methods. In general, we expected larger performance gains through the use of pretrained vectors. Hence, our results suggest that the pretrained embeddings may not consistently yield good results across all datasets.

3.4 CHAPTER CONCLUSION

In this work, we propose a kernel for comparing sets of vectors. The kernel computes an optimal assignment between the elements of the sets. We applied the kernel to the problem of graph comparison, where each graph is represented as a set containing the embeddings of its vertices. After embedding the vertices of all graphs in a vector space, we construct a hierarchy of them using clustering. Based on this hierarchy, we define a kernel that computes an optimal assignment of the vertices of two graphs. The proposed kernel can be applied to any problem where instances are represented as sets of vectors.

The kernel is evaluated on standard graph classification, link prediction and text categorisation datasets, where it shows a great performance with respect to traditional and state-of-the-art techniques.

Part II

GRAPH NEURAL NETWORKS

4

SHOWCASE: TOPOLOGY PREDICTION FOR DYNAMIC GRAPHS

Bringing the expressive power of deep learning into non-Euclidean data such as graphs, GNNs have emerged in recent years as an effective tool for analysing graph-structured data (Gilmer et al., 2017; Wu et al., 2021). In this chapter, we demonstrate the power of GNNs over traditional methods on a long-existing problem of topology prediction of dynamic graphs. Predicting the evolution of dynamic graphs is a task of high significance in the area of graph mining as most real-world networks are evolving over time. Despite its practical importance, the task has not been explored in depth so far, mainly due to its challenging nature. In this work, we propose a GNN-based model to tackle this problem. Specifically, we use a MPNN along with a recurrent architecture to capture the temporal evolution patterns of dynamic graphs. Then, we employ a generative model which predicts the topology of the graph at the next (or future) time step and constructs a graph instance that corresponds to that topology. We evaluate the proposed model on several synthetic datasets following common network evolving dynamics, as well as on real-world datasets, and demonstrate the effectiveness of our proposed model over traditional methods.

4.1 INTRODUCTION

Neural networks for structured data such as graphs have been studied extensively in recent years (Wu et al., 2021; Zhou et al., 2020; Zhou, Zheng, and Huang, 2020). They have demonstrated convincing performance in several graph mining tasks, including graph classification (Morris et al., 2019), link prediction (Zhang and Chen, 2017), and community detection (Chen, Li, and Bruna, 2019). So far, the bulk of research GNNs focused mainly on tasks that involve static graphs. However, most real-world networks are dynamic, e. g., nodes and edges are added or removed over time. Despite the success of GNNs in various applications, it is still not clear if these models are useful for learning in dynamic scenarios.

Among the few models that have been applied to this type of data, most studies focused on predicting a low-dimensional representation (i. e., embedding) of the graph for the next time step (Goyal et al., 2018; Li, Guo, and Mei, 2016;

Nguyen et al., 2018; Pareja et al., 2020; Seo et al., 2018). These representations can then be used in downstream tasks (Goyal et al., 2018; Li, Guo, and Mei, 2016; Meng et al., 2018; Pareja et al., 2020). However, predicting the topology of the graph (and not only its low-dimensional representation) is a task that has not been properly addressed yet.

Graph generation, another important task in graph mining, has attracted a lot of attention from the deep learning community in recent years. The objective of this task is to generate graphs that exhibit specific properties, e. g., degree distribution, node triangle participation, community structure, etc. Traditionally, graphs are generated based on some network generation model such as the Erdős-Rényi model (Erdős and Rényi, 1960). These models focus on modelling one or more network properties, and neglect the others. Neural network approaches, on the other hand, can better capture the properties of graphs since they follow a supervised approach (Bojchevski et al., 2018; Grover, Zweig, and Ermon, 2019; You et al., 2018). These architectures minimise a loss function such as the reconstruction error of the adjacency matrix or the value of a graph comparison algorithm.

Capitalising on recent developments in neural networks for graph-structured data and graph generation, we propose in this work, to the best of our knowledge, the first framework for predicting the evolution of the topology of networks in time. The proposed framework can be viewed as an *encoder-predictor-decoder* architecture. The *encoder* network takes a sequence of graphs as input and uses a GNN to produce a low-dimensional representation for each of these graphs. These representations capture structural information about the input graphs. Then, the *predictor* network employs a recurrent architecture which predicts a representation for the future instance of the graph. The *decoder* network corresponds to a graph generation model which utilises the predicted representation, and generates the topology of the graph for the next (or future) time step. The proposed model is evaluated over a series of experiments on synthetic and real-world datasets, and is compared against several baseline methods.

To measure the effectiveness of the proposed model and the baselines, the generated graphs need to be compared with the ground-truth graph instances using some graph comparison algorithm. To this end, we use the Weisfeiler-Lehman subtree kernel which scales to very large graphs and has achieved state-of-the-art results on many graph datasets (Shervashidze et al., 2011). Results show that the proposed model yields good performance, and in most cases, outperforms the competing methods.

The rest of this chapter is organised as follow. Section 4.2 provides an overview of the related work and elaborates our contribution. Section 4.3 introduces some preliminary concepts and definitions related to the graph generation problem, followed by a detailed presentation of the components of the proposed framework. Section 4.4 evaluates the proposed model on several tasks. Finally, Section 4.5 concludes.

4.2 RELATED WORK

Our work is directly linked to the line of research in random graph models. These models are very popular in graph theory and network science. The Erdős-Rényi model (Erdős and Rényi, 1960), the preferential attachment model (Albert and Barabási, 2002), and the Kronecker graph model (Leskovec et al., 2010) are some typical examples of such models. To predict how a graph structure will evolve over time, the values of the parameters of these models can be estimated based on the corresponding values of the observed graph instances, and based on the estimated values, these models are able to generate graphs that have not been observed yet.

Other work along a similar direction includes neural network models which combine GNNs with RNNs (Manessi, Rozza, and Manzo, 2020; Pareja et al., 2020; Seo et al., 2018). These models use GNNs to extract features from a graph and RNNs for sequence learning from the extracted features. Some similar approaches do not use GNNs, but instead they perform random walks or employ deep auto-encoders (Goyal et al., 2018; Nguyen et al., 2018).

All these works focus on predicting how the node representations or the graph representations will evolve over time. However, some applications require predicting the topology of the graph, and not just its low-dimensional representation. The proposed model constitutes the first step towards this objective.

4.3 EVONET: A NEURAL NETWORK FOR PREDICTING GRAPH EVOLUTION

In this Section, we begin by introducing basic concepts of dynamic graph and formalise our task. We then present **EvoNet**, the proposed framework for predicting the evolution of graphs. Since the proposed model comprises of several components, we describe all these components in detail.

4.3.1 *Preliminaries*

Let $\mathcal{G} = (\mathbb{V}, \mathbb{E})$ be an undirected, unweighted graph, where $\mathbb{V}$ is the set of nodes and $\mathbb{E}$ is the set of edges. We will denote by n the number of vertices and by m the number of edges. We define a permutation of the nodes of $\mathcal{G}$ as a bijective function $\pi : \mathbb{V} \to \mathbb{V}$, under which any graph property of $\mathcal{G}$ should be invariant.

We are interested in the topology of a graph which is described by its adjacency matrix $\boldsymbol{A}^{\pi} \in \mathbb{R}^{n \times n}$ with a specific ordering of the nodes π[1]. Each entry of the adjacency matrix is defined as $\boldsymbol{A}^{\pi}_{ij} = \mathbb{1}_{(\pi(v_i), \pi(v_j)) \in \mathbb{E}}$ where $v_i, v_j \in \mathbb{V}$. In what follows, we consider the *topology*, *structure* and *adjacency matrix* of a graph equivalent to each other.

In many real-world networks, besides the adjacency matrix that encodes connectivity information, nodes and/or edges are annotated with a feature vector of dimension d or d', which we denote as $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ and $\boldsymbol{E} \in \mathbb{R}^{m \times d'}$, respectively. Hence, a graph object can be also written in the form of a triplet $\mathcal{G} = (\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{E})$. In this work, we use this triplet to represent all graphs. If a graph does not contain node/edge attributes, we assign to it some artificial attributes based on local properties (e. g., degree, k-core number, number of triangles, etc).

An evolving network is a graph whose topology changes as a function of time. Interestingly, almost all real-world networks evolve over time by adding and removing nodes and/or edges. For instance, in social networks, people make and lose friends over time, while there are people who join the network and others who leave the network. An evolving graph is a sequence of graphs $\{\mathcal{G}_0, \mathcal{G}_1, \ldots, \mathcal{G}_T\}$ where $\mathcal{G}_t = (\boldsymbol{A}_t, \boldsymbol{X}_t, \boldsymbol{E}_t)$ represents the state of the evolving graph at time step t. It should be noted that not only nodes and edges can evolve over time, but also node and edge attributes. However, in this work, we keep attributes of each node and edge fixed, and we allow only the node and edge sets of the graphs to change as a function of time. The sequence can thus be written as $\{\mathcal{G}_t = (\boldsymbol{A}_t, \boldsymbol{X}, \boldsymbol{E})\}_{t \in [0,T]}$ [2].

We are often interested in predicting what "comes next" in a sequence, based on data encountered in previous time steps. In our setting, this is equivalent to predicting $\mathcal{G}_t$ based on the sequence $\{\mathcal{G}_k\}_{k<t}$. In sequential modelling, we usually do not take into account the whole sequence, but only those instances within a fixed small window of size w before $\mathcal{G}_t$, which we denote as

1 For simplicity, the ordering π will be omitted in what follows.

2 Note that here $\boldsymbol{A}_t$ corresponds to the vertex set $\mathbb{V}$ that contains all the nodes which have appeared on the evolving graphs within the time range of interest. Nodes that are not on $\mathcal{G}_t$ at time step t are considered as isolated nodes of $\mathcal{G}_t$.

$\{\mathcal{G}_{t-w}, \mathcal{G}_{t-w+1}, \ldots, \mathcal{G}_{t-1}\}$. We refer to these instances as the graph history. The problem is then to predict the topology of $\mathcal{G}_t$ given its history.

4.3.2 *Proposed Architecture*

The proposed architecture is very similar to a typical sequence learning framework. The main difference lies in the fact that instead of vectors, in our setting, the elements of the sequence correspond to graphs. The combinatorial nature of graph-structured data increases the complexity of the problem and calls for more sophisticated architectures than the ones employed in traditional sequence learning tasks.

Specifically, the proposed model consists of three components:

1. a GNN which generates a vector representation for each graph instance,
2. a RNN for sequential learning,
3. a graph generation model for predicting the graph topology at the next time step.

This framework can also be viewed as an *encoder-predictor-decode* model. The first two components correspond to an *encoder* network which maps the sequence of graphs into a sequence of vectors and another network that predicts a representation for the *next in the sequence* graph. The *decoder* network is the last component of the model, and transforms the above representation into a graph. Figure 4.1 illustrates the proposed model. In what follows, we present the above three components of **EvoNet**.

4.3.2.1 *Encoding Graphs using Graph Neural Networks*

Graph Neural Network (GNN) has recently emerged as a dominant paradigm for performing machine learning tasks on graphs. Its ability of learning meaningful low-dimensional embedding from graph structure makes it a natural choice for the *encoder* component of our framework. From various GNN models, We pick MPNN as the implementation choice. Described in Section 2.4.1, the learning process of MPNN can be divided into three phases: (1) *aggregation*, (2) *update*, and (3) *readout*.

AGGREGATION In this phase, for each node of the graph, the network computes a *message*, which is the aggregation of the representations from its neigh-

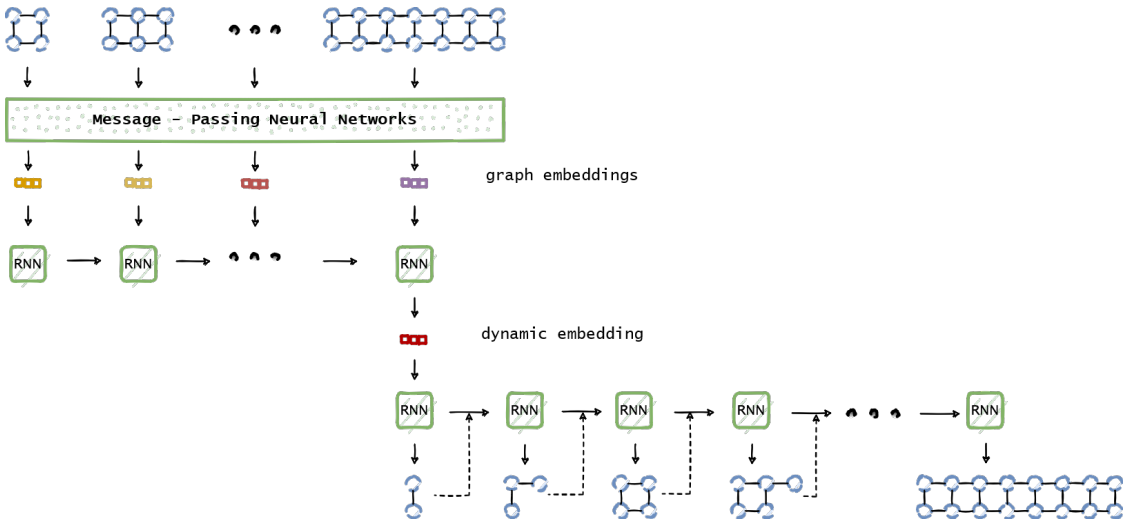

Figure 4.1: Illustration of the proposed architecture

bourhood. Formally, at iteration step $l+1$, a message vector $\boldsymbol{m}_v^{l+1}$ is computed from the representations of the neighbours $\mathcal{N}(v)$ of v by

$$\boldsymbol{m}_v^{l+1} = f_1(\boldsymbol{x}_v^l) + \sum_{w \in \mathcal{N}(v)} \boldsymbol{x}_w^l f_2(\boldsymbol{e}_{vw}), \tag{4.1}$$

where $\boldsymbol{x}_w^l$ is the hidden embedding of node w at iteration l, $\boldsymbol{e}_{vw}$ is the edge embedding between node v and w, and $f_1(\cdot)$, $f_2(\cdot)$ are two MLPs. Suppose $\boldsymbol{m}$ and $\boldsymbol{x}$ are both vectors of the same dimension d, note that $f_2(\cdot)$ transform the edge vector $\boldsymbol{e}_{vw} \in \mathbb{R}^{d'}$ into $\mathbb{R}^{d*d}$.

UPDATE The new representation $\boldsymbol{x}_v^{l+1}$ of v is then computed by combining its current feature vector $\boldsymbol{x}_v^l$ with the message vector $\boldsymbol{m}_v^{l+1}$:

$$\boldsymbol{x}_v^{l+1} = \mathrm{GRU}(\boldsymbol{x}_v^l, \boldsymbol{m}_v^{l+1}), \tag{4.2}$$

where GRU is the Gated Recurrent Unit (GRU) (Cho et al., 2014), a class of RNN whose exact form is:

$$\begin{aligned}
\boldsymbol{z}_v^{l+1} &= \sigma(\boldsymbol{W}_z \boldsymbol{m}_v^{l+1} + \boldsymbol{U}_z \boldsymbol{x}_v^l + \boldsymbol{b}_z) \\
\boldsymbol{r}_v^{l+1} &= \sigma(\boldsymbol{W}_r \boldsymbol{m}_v^{l+1} + \boldsymbol{U}_r \boldsymbol{x}_v^l + \boldsymbol{b}_r) \\
\hat{\boldsymbol{x}}_v^{l+1} &= \phi\big(\boldsymbol{W}_h \boldsymbol{m}_v^{l+1} + \boldsymbol{U}_h (\boldsymbol{r}_v^{l+1} \odot \boldsymbol{x}_v^l) + \boldsymbol{b}_h\big) \\
\boldsymbol{x}_v^{l+1} &= (1 - \boldsymbol{z}_v^{l+1}) \odot \boldsymbol{x}_v^l + \boldsymbol{z}_v^{l+1} \odot \hat{\boldsymbol{x}}_v^{l+1}.
\end{aligned} \tag{4.3}$$

where $\boldsymbol{W}$, $\boldsymbol{U}$, $\boldsymbol{h}$ are the weights and bias, $\sigma(\cdot)$, $\phi(\cdot)$ are sigmoid activation function and hyperbolic tangent activation function respectively. The *aggregation* and *update* steps are similar to those of Li et al. (2016).

READOUT After repeating *aggregation* and *update* steps for L iterations, we get a sequence of node representations $\{\boldsymbol{x}_v^L\}_{v\in\mathbb{V}}$. They are then merged into a single vector which corresponds to the representation of the entire graph at the *readout* step:

$$\boldsymbol{g} = \textsc{set2set}\big(\{\boldsymbol{x}_v^L \mid v \in \mathbb{V}\}\big), \tag{4.4}$$

where SET2SET is a iterative attention-based method that can learn the representation from a set (Vinyals, Bengio, and Kudlur, 2016). Its exact form is:

$$\begin{aligned}
\boldsymbol{q}_k &= \textsc{lstm}(\boldsymbol{q}_{k-1}^*) \\
\alpha_{i,k} &= \textsc{softmax}(\boldsymbol{x}_i \boldsymbol{q}_k^\intercal) = \frac{e^{\boldsymbol{x}_i \boldsymbol{q}_k^\intercal}}{\sum_{j=1}^{n} e^{\boldsymbol{x}_j \boldsymbol{q}_k^\intercal}} \\
\boldsymbol{r}_k &= \sum_{i=1}^{n} \alpha_{i,k} \boldsymbol{x}_i \\
\boldsymbol{q}_k^* &= [\boldsymbol{q}_k, \boldsymbol{r}_k],
\end{aligned} \tag{4.5}$$

where the final output $\boldsymbol{g} = \boldsymbol{q}_K^*$ is twice the dimension of the node representation $\boldsymbol{x}$. This vector captures the topology of the input graph. Of course, as stated in Section 2.4.1, there are many choices for the global pooling function of the *readout* step, as long as it is differentiable and permutation invariant. Other functions, e. g., a simple sum, were indeed considered, but were found less effective in preliminary experiments.

4.3.2.2 *Predicting Graph Representations using Recurrent Neural Networks*

Given an input sequence of graphs, we use the aforementioned MPNN to generate a vector representation for each graph in the sequence. Then, to process this sequence, we use the Recurrent Neural Network (RNN). The RNNs use their internal state, i. e., memory, to preserve sequential information. These networks exhibit temporal dynamic behaviour, and can find correlations between sequential events.

Specifically, a RNN processes the input sequence in a series of time steps, e. g., one for each element in the sequence. For a given time step t, the hidden state $\boldsymbol{h}_{\mathcal{G}_t}$ at that time step is updated as:

$$\boldsymbol{h}_{\mathcal{G}_t} = \boldsymbol{\Theta}(\boldsymbol{h}_{\mathcal{G}_{t-1}}, \boldsymbol{g}_t) \tag{4.6}$$

where $\boldsymbol{\Theta}$ is a non-linear function, whose exact form depends on the choice of the RNN.

The generative RNNs output a probability distribution over the next element of the sequence given its current state. Precisely, they are able to predict the next element (e. g., graph embedding) in the sequence, by learning the conditional distribution $p(\boldsymbol{h}_{\mathcal{G}_t} | \boldsymbol{g}_1, \ldots, \boldsymbol{g}_{t-1})$.

In our implementation, we use Long Short-Term Memory (LSTM) network that reads sequentially the vectors $\{\boldsymbol{g}_i\}_{i \in [t-w, t-1]}$ produced by the MPNN, and generates a vector $\boldsymbol{h}_{\mathcal{G}_t}$ that represents the embedding of $\mathcal{G}_t$. The embedding incorporates topological information and will serve as input to the graph generation module.

The MPNN component presented above can be seen as a form of an *encoder* network. It takes as input a sequence of graphs and projects them into a low-dimensional space. Then, this RNN component takes the sequence of graph representations as input and predicts the representation of the graph at the next time step. We name the predicted representation as *dynamic graph embedding*, for that it captures both the structure of a graph and the temporal dependency of the sequence.

4.3.2.3 *Graph Generation*

To generate a graph that corresponds to the evolution of the current graph instance, we capitalise on a sequential framework for learning generative models of graphs (You et al., 2018). This framework models a graph in an auto-regressive manner (i. e., a sequence of additions of new nodes and edges) to capture the complex joint probability of all nodes and edges in the graph. Formally, given a node ordering π, it considers a graph $\mathcal{G}$ as a sequence of vectors:

$$\boldsymbol{S}_{\mathcal{G}}^{\pi} = (\boldsymbol{s}_1^{\pi}, \boldsymbol{s}_2^{\pi}, \ldots, \boldsymbol{s}_n^{\pi}) \tag{4.7}$$

where $\boldsymbol{s}_i^{\pi} = [a_{1,i}, \ldots, a_{i-1,i}] \in \{0,1\}^{i-1}$ is the adjacency vector between node $\pi(i)$ and the nodes preceding it $(\{\pi(1), \ldots, \pi(i-1)\})$. We adapt this framework to our supervised setting.

The objective of the generative model is to maximise the likelihood of the observed graphs of the training set $p(\mathcal{G}|\boldsymbol{\theta})$. Since a graph can be expressed as a sequence of adjacency vectors (given a node ordering), we consider instead the likelihood $p(\boldsymbol{S}^{\pi}_{\mathcal{G}}|\boldsymbol{\theta})$, which can be decomposed in an auto-regressive manner into the following product:

$$p(S^{\pi}_{\mathcal{G}}|\boldsymbol{\theta}) = \prod_{i=1}^{n} p(\boldsymbol{s}^{\pi}_{i}|\boldsymbol{s}^{\pi}_{k:k<i}, \theta) = \prod_{i=1}^{n}\prod_{j=1}^{i-1} p(a^{\pi}_{ji}|a^{\pi}_{li:l<j}, \boldsymbol{s}^{\pi}_{k:k<i}, \boldsymbol{\theta}) \tag{4.8}$$

This product can be parameterised by a neural network. Following You et al. (2018), we use a hierarchical RNN consisting of two levels:

1. a graph-level RNN which generates new nodes by updating the state of adjacency vectors, i. e., learns the distribution $p(\boldsymbol{s}^{\pi}_{i}|\boldsymbol{s}^{\pi}_{k:k<i})$,
2. a edge-level RNN which generates links between current generated node and those previously-generated, i. e., learns the distribution $p(\hat{a}^{\pi}_{ji}|\hat{a}^{\pi}_{li:l<j})$

More formally, we have:

$$\begin{aligned} \boldsymbol{h}_0 &= \boldsymbol{h}_{\mathcal{G}_T} & \boldsymbol{h}_i &= \boldsymbol{\Theta}_1(\boldsymbol{h}_{i-1}, \boldsymbol{s}^{\pi}_{i-1}) \\ \boldsymbol{e}_{0,i} &= \boldsymbol{h}_i & \boldsymbol{e}_{j,i} &= \boldsymbol{\Theta}_2(\boldsymbol{e}_{j-1,i}, \hat{a}^{\pi}_{j-1,i}) \\ \hat{a}^{\pi}_{j,i} &\sim p & p(\hat{a}^{\pi}_{j,i} &= 1) = f(\boldsymbol{e}_{j,i}) \end{aligned} \tag{4.9}$$

where $\boldsymbol{h}_i$ is the state vector of the graph-level RNN ($\boldsymbol{\Theta}_1$) that encodes the current state of the graph sequence $\{\boldsymbol{s}^{\pi}_{k:k<i}\}$ and is initialised by $h_{\mathcal{G}_T}$, the predicted embedding of the graph at the next time step T. The output of the graph-level RNN corresponds to the initial state of the edge-level RNN ($\boldsymbol{\Theta}_2$), whose state at the last time step is then processed by a function f to produce the probability of existence of an edge $\hat{a}_{j,i}$,[3] e. g., MLP stacked with a *sigmoid* function. In other words, the model learns the probability distribution of the existence of edges and a graph can then be sampled from this distribution, which will serve as the predicted topology for the next time step T.

To train the model, the cross-entropy loss between existence of each edge and its probability of existence is minimised:

$$\mathcal{L} = -\sum_{i=1}^{n}\sum_{j=1}^{i-1}\left[a^{\pi}_{j,i}\log\left(p(\hat{a}^{\pi}_{j,i}=1)\right) + (1-a^{\pi}_{j,i})\log\left(1-p(\hat{a}^{\pi}_{j,i}=1)\right)\right] \tag{4.10}$$

3 We denote the predicted adjacency value as $\hat{a}_{j,i}$ in order to distinguish from the ground truth value $a_{j,i}$.

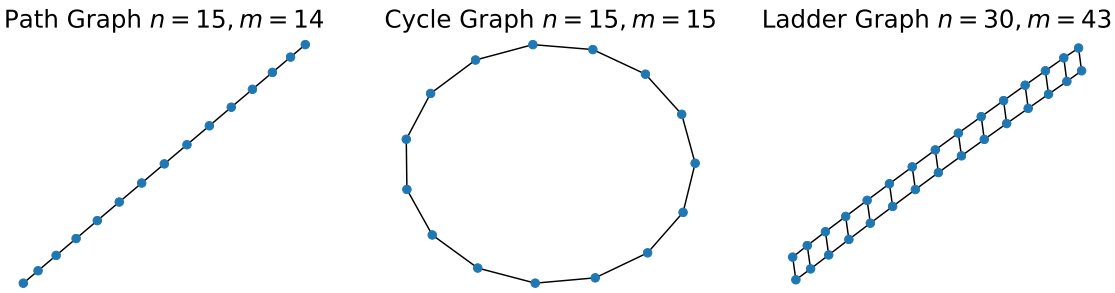

Figure 4.2: Illustration of synthetic graphs. n is the number of nodes; m is the number of edges.

4.4 EXPERIMENTS AND DISCUSSION

In this section, we evaluate the performance of **EvoNet** on synthetic and real-world datasets for predicting the evolution of graph topology, and we compare it against several baseline methods.

4.4.1 *Datasets*

4.4.1.1 *Synthetic Datasets*

The synthetic datasets consist of sequences of graphs where there is a specific pattern on how each graph emerges from the previous graph instance, i. e., adding/removing some graph structure at each time step. The patterns we experimented on are *paths*, *cycles* and *ladders*. An illustration of these patterns can be found in Figure 4.2.

The real-world datasets correspond to single graphs whose nodes incorporate temporal information, which We decompose into sequences of snapshots based on their timestamps. The size of the graphs in each sequence ranges from tens of nodes to several thousand of nodes. We summarise the statistics of these datasets in Table 4.1.

4.4.1.2 *Synthetic Datasets*

PATH GRAPH We denote a path graph of n nodes as P_n. It can be drawn such that all vertices and edges lie on a straight line, i. e., the path graph P_n is a tree with two nodes of degree 1, and the other $n - 2$ nodes of degree 2. We consider two scenarios. In both cases the initial graph in the sequence is P_3. In the *first* scenario, at each time step, we add one new node to the previous graph instance and we also add an edge between the new node and the last

node according to the previous ordering. The *second* scenario follows the same pattern, however, every three steps, instead of adding a new node, we remove the first node according to the previous ordering (along with its edge).

CYCLE GRAPH A cycle graph C_n is a graph on n nodes containing a single cycle through all the nodes. Note that if we add an edge between the first and the last node of P_n, we obtain C_n. Similar to the above case, we use C_3 as the initial graph in the sequence, and we again consider two scenarios. In the *first* scenario, at each time step, we increase the size of the cycle, e. g., we obtain C_{i+1} from C_i by inserting a new node between the first and the last node according to the previous ordering and connect them. In the *second* scenario, besides adding a new node at each time step, every 3 (or other value) steps, we also attach a special structure to the newly added node (e. g., K_3 or K_5[4]).

LADDER GRAPH The ladder graph L_n is a planar graph with $2n$ vertices and $3n - 2$ edges. It is the Cartesian product of two path graphs, as follows: $L_n = P_n \times P_2$. As the name indicates, the ladder graph L_n can be drawn as a ladder consisting of two rails and n rungs between them. We consider the following scenario: at each time step, we attach one rung (P_2) to the tail of the ladder. The two nodes of the rung are then connected to the two last nodes according to the previous ordering, respectively).

For all synthetic graphs, we set the attribute of each node equal to its degree, while we set the attribute of all edges to the same value (e. g., to 1).

4.4.1.3 *Real-World Datasets*

We also evaluate our model on six real-world datasets. They can be divided into three groups based on the nature of their sources.

BITCOIN TRANSACTION NETWORKS This group contains graphs derived from the Bitcoin transaction network, a who-trust-whom network of people who trade using Bitcoin (Kumar et al., 2018, 2016). Due to the anonymity of Bitcoin users, platforms seek to maintain a record of users' reputation in Bitcoin trades to avoid fraudulent transactions. The nodes of the network represent Bitcoin users, while an edge indicates that a trade has been executed between its two endpoint users. Each edge is annotated with an integer between -10 and 10, which indicates the rating of the one user given by the others. The datasets are collected separately from two platforms: **Bitcoin OTC** and **Bitcoin Alpha**. For

4 Complete graph with n nodes is denoted as K_n. e. g., K_3 is a triangle.

Table 4.1: Statistics of 6 real-world datasets.

	#Nodes	#Edges	% Pos.Edges	Timespan	
				Begin	End
BTC-OTC	5,881	35,592	89%	2010-11-08	2016-01-25
BTC-Alpha	3,783	24,186	93%	2010-11-08	2016-01-22
UCI-Forum	899	33,720	—	2004-05-15	2004-10-26
UCI-Message	1,899	59,835	—	2004-04-15	2004-10-26
EU-Core	986	332,334	—	1970-01-01	1972-03-14
DNC	1,891	39,264	—	2013-09-16	2016-05-25

all graphs in these two datasets, we set the attribute of each node equal to the average rating that the user has received from the rest of the community, and the attribute of each edge equal to the rating between its two endpoint users.

SOCIAL NETWORKS This group contains graphs generated from an online social network at the University of California, Irvine (Opsahl, 2013; Opsahl and Panzarasa, 2009). It has two datasets: one is derived from the private message exchange between users (**UCI-Message**); the other is based on the same user community, but focuses on their activity in the forum, i. e., public comment on a specific topic (**UCI-Forum**). The nodes of the networks represent users and the edges represent a message exchange or a shared interest (on a topic). All graphs in these two datasets are unweighted and unlabelled, thus we simply set the attribute of each node equal to its degree.

EMAIL EXCHANGE NETWORKS This group contains two datasets derived from different sources. The first is generated using email data from a large European research institution (Paranjape, Benson, and Leskovec, 2017), i. e., all incoming and outgoing email between members of the research institution. The second is collected from the 2016 Democratic National Committee (**DNC**) email leak (Rossi and Ahmed, 2015), where the links denote email exchanges between DNC members. Similar to social network datasets, the graphs in these two datasets are also unweighted and unlabelled. We treat them the same way.

4.4.2 *Baselines*

We compare **EvoNet** against several random graph generators, which are the traditional methods to study the topology evolution of temporal graphs, by proposing a driven mechanism behind the evolution. These models distinguish between each other by their rules to connect new emerged nodes with existing

ones. They usually have an initial graph, from which they gradually grow into the graph with expected size following the rule. To be precise, we have,

- Erdős-Rényi (ER) model proposed by Erdős and Rényi (1960). The topology of predicted graph at time step t is a binomial graph of size n_t whose edges are sampled from a binomial distribution $\mathrm{B}(\frac{n_t(n_t-1)}{2}, p)$, where p is a tunable parameter;

- Barabási–Albert (BA) model proposed by Albert and Barabási (2002). At each step, a new node k is added to the graph, and connect to every existing node i with a probability $\frac{d_i}{\sum_{j \in \mathbb{V} \setminus k} d_j}$. The topology of predicted graph at time step t is the result of such process;

- Powerlaw Clustering (Power) model proposed by Holme and Kim (2002). An improved version of Barabási–Albert model, where, after adding a node, an extra step is performed that every edge will be connected to its neighbours (forming a triangle) by a chance;

- Stochastic Block model (SBM) proposed by Airoldi et al. (2008). The topology of predicted graph at time step t is a Stochastic Block Model (SBM) graph of size n_t with 3 communities and fixed probabilities to connect within and between the communities;

- Kronecker Graph (Kron-fix, Kron-rand) model proposed by Leskovec et al. (2010). The topology of predicted graph at time step t is the Kronecker product $\mathcal{G}_t = \mathcal{G}_{t-1} \otimes \mathcal{G}_{\text{base}}$, i. e., the t^{th} Kronecker power of $\mathcal{G}_{\text{base}}$, which is the initial graph at time step 0. "Kron-fix" means that the initial graph is fixed and deterministic. "Kron-rand" corresponds to the stochastic Kronecker graph model, where the initial graph is a random graph parameterised by some tunable probabilities.

4.4.3 *Experimental Setup and Evaluation Metric*

In general, it is very challenging to measure the performance of a graph generative model since it requires comparing two graphs to each other, a long-existing problem in mathematics and computer science (Conte et al., 2004). We propose to use graph kernels to compare graphs to each other, and thus to evaluate the quality of the generated graphs.

Graph kernels are considered as one of the most effective tools for graph comparison (Nikolentzos, Siglidis, and Vazirgiannis, 2019). A graph kernel is a symmetric positive semi-definite function which takes two graphs as input, and

measures their similarity. In our experiments, we employ the *Weisfeiler-Lehman subtree kernel* which counts label-based subtree-patterns (Shervashidze et al., 2011). Note that we normalise the kernel values, so that the emerging values lie between 0 and 1.

As previously mentioned, each dataset corresponds to a sequence of graphs of length k. where each sequence represents the evolution of the topology of a single graph in k time steps. We use the first 80% of these graph instances for training and the rest of them serve as our test set. The window size w is set equal to 10, which means that we feed 10 consecutive graph instances to the model and predict the topology of the instance that directly follows the last of these 10 input instances. Each graph of the test set along with its corresponding predicted graph is then passed on to the *Weisfeiler-Lehman subtree kernel* which measures their similarity and thus the performance of the model.

The hyperparameters of **EvoNet** are chosen based on its performance on a validation set. The parameters of the random graph models are set under the principle that the generated graphs need to share similar properties with the ground-truth graphs. For instance, in the case of the Erdős-Rényi model, the probability of adding an edge between two nodes is set to some value such that the density of the generated graph is identical to that of the ground-truth graph. However, since the model should not have access to such information (e. g., density of the ground-truth graph), we use an MLP to predict this property based on past data (e. g., the number of nodes and edges of the previous graph instances). This is in par with how the proposed model computes the size of the graphs to be generated (i. e., using also an MLP).

4.4.4 *Experiment Analysis*

We next present the experimental results and compare the performance of **EvoNet** against that of the baselines.

SYNTHETIC DATASETS Figure 4.3 illustrates the experimental results on the synthetic datasets. Since the graph structures contained in the synthetic datasets are fairly simple, it is easy for the model to generate graphs very similar to the ground-truth graphs (normalised kernel values > 0.9). Hence, instead of reporting the kernel values, we compare the size of the predicted graphs against that of the ground-truth graphs. The figures visualise the increase of graph size on real sequence (orange) and predicted sequence (blue).

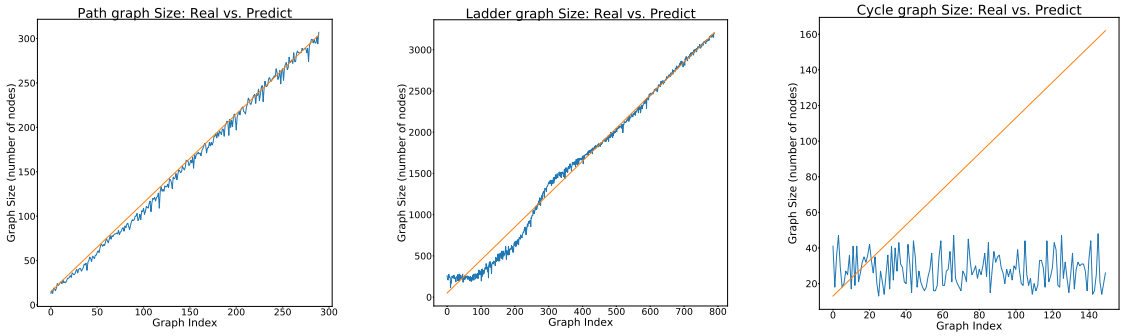

Figure 4.3: Comparison of graph size: predicted size (blue) vs. real size (orange). From **left** to **right**: Path graph, Ladder graph and Cycle graph.

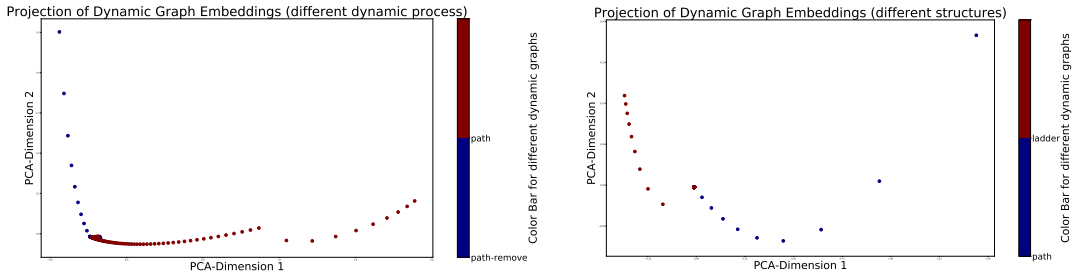

Figure 4.4: 2D projection of dynamic graph embeddings. **Left**: synthetic datasets following different dynamics. **Right**: synthetic datasets with different structures.

For path graphs, in spite of small variance, we have an accurate prediction on the graph size. For ladder graph, we observe a mismatch at the beginning of the sequence for small size graphs but then a coincidence of the two lines on large size graphs. This mismatch on small graphs may be due to a more complex structure in ladder graphs such as cycles, as supported by the results of cycle graph on the right figure, where we completely fail to predict the size of cycle graphs. In fact, we were not able to reconstruct the cycle structure in the prediction, with all the predicted graphs being path graphs. This failure could be related to the limitations of GNN model mentioned in Xu et al. (2018). Additional results of synthetic graphs, such as similarity histograms, size comparison for more complex dynamic scenario, and examples of generated graphs can be found in Appendix A.

DYNAMIC GRAPH EMBEDDING It is also important to check whether, in our *encode-decoder* framework, the learned code, which we refer to as *"dynamic graph embedding"*, is really meaningful, i. e., whether it is capable of capturing both structural feature of the graph class and temporal evolution of the series. We design two experiments to verify the effectiveness of our embedding, with the help of synthetic graphs.

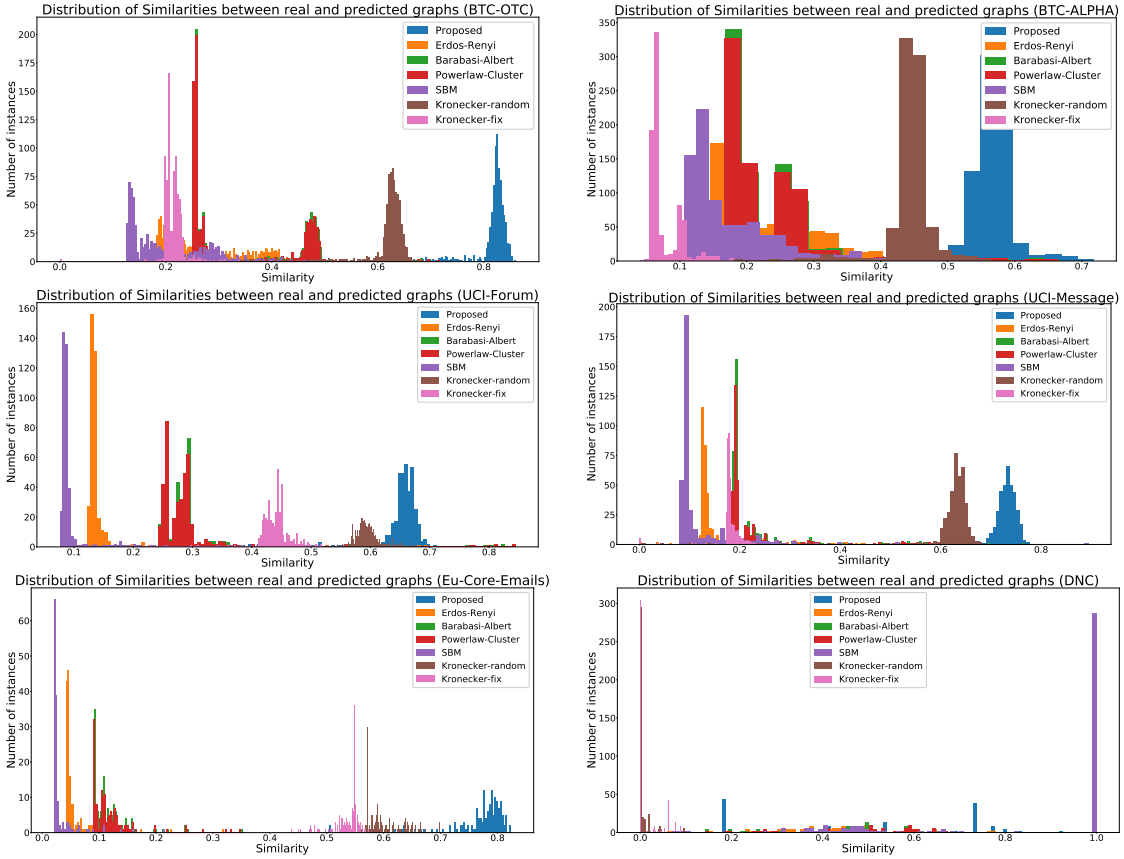

Figure 4.5: Similarity histograms on real-world datasets. Blue one is the result of **EvoNet**, which is compared against 6 random graph models. From **top** to **bottom**, from **left** to **right**: BTC-OTC, BTC-Alpha, UCI-Forum, UCI-Message, EU-Core-Emails and DNC datasets.

In the first experiment, we take as input two sequences of graphs belonging to the same class but following different evolution dynamics. Specifically, we took *path graph* and *path graph with removal*. In the second experiment, we control the evolution dynamic and vary the structures of graphs, where we use *path graph* and *ladder graph* following the same evolution of increasing size.

The dynamic graph embeddings of different datasets learned from these experiments are recorded and visualised in Figure 4.4. Each point represents the projections of embeddings of each graph in the sequence into a 2-dimensional space by Principle Component Analysis (PCA). As we can see from the figure, embeddings learned from different datasets, either with different dynamics or with different structure, are both well separated, which suggests that the embeddings are meaningful and can be use to predict the graph at the future time step. Those from the same dataset form special patterns such as a line in the space, which suggests a temporal dependency between these embeddings as they are learned from sequential data.

Table 4.2: Statistics on the similarity distribution of different models. The best results are set to **bold** and the second best ones are underlined.

Model \ Stat.	BTC-OTC		BTC-ALPHA		UCI-Forum	
	Mean	90%ile	Mean	90%ile	Mean	90%ile
ER	0.28	0.40	0.22	0.32	0.15	0.16
SBM	0.21	0.30	0.18	0.27	0.10	0.10
BA	0.35	0.48	0.23	0.28	0.29	0.31
Power	0.35	0.48	0.23	0.28	0.29	0.31
Kron-Rand	0.62	0.64	0.44	0.47	0.59	0.62
Kron-Fix	0.21	0.23	0.08	0.11	0.44	0.47
EvoNet	**0.82**	**0.84**	**0.55**	**0.59**	**0.64**	**0.68**

Model \ Stat.	UCI-Mesg		EU-Core		DNC	
	Mean	90%ile	Mean	90%ile	Mean	90%ile
ER	0.16	0.26	0.06	0.09	0.82	**1.00**
SBM	0.13	0.22	0.03	0.05	0.84	**1.00**
BA	0.23	0.33	0.12	0.16	**0.85**	**1.00**
Power	0.23	0.33	0.12	0.16	0.85	**1.00**
Kron-Rand	0.62	0.65	0.60	0.65	0.01	0.02
Kron-Fix	0.18	0.20	0.53	0.55	0.01	0.06
EvoNet	**0.71**	**0.75**	**0.76**	**0.81**	0.83	**1.00**

REAL-WORLD DATASETS Finally, we analyse the performance of our model on the six real-world datasets. We obtain the similarities between each pair of real and predicted graphs in the sequence and draw a histogram to illustrate the distribution of similarities. We show in Figure 4.5 the histogram plots of the six datasets. Among all the traditional random graph models, Kronecker graph model with learnable parameter ("Kron-Rand") performs the best. However, on every dataset (except DNC), our proposed method **EvoNet** (in blue) outperforms tremendously all the other methods. Statistics of the similarity distribution (e. g., average similarity, the value at 90^{th} percentile) of all datasets are summarised in Table 4.2, where we find that our proposed model has a consistent advantage over the traditional methods.

Overall, despite a failure in capturing some specific structures discovered in synthetic datasets, our experiments demonstrate the advantage of **EvoNet** over the traditional random graph models on predicting the evolution of dynamic graphs, especially for real world data with complex structures.

4.5 CHAPTER CONCLUSION

In this work, we propose **EvoNet**, a model that predicts the evolution of dynamic graphs, following an *encoder-decoder* framework. The proposed model consists of three components:

1. a graph neural network which transforms graphs to vectors,
2. a recurrent architecture which reads the input sequence of graph embeddings and predicts the embedding of the graph at the next time step,
3. a graph generation model which takes this embedding as input and predicts the topology of the graph.

We also propose an evaluation methodology for this task which capitalises on the well-established family of graph kernels. We apply the above methodology to demonstrate the predictive power of **EvoNet**. Experiments show that the proposed model outperforms traditional random graph methods on both synthetic and real-world datasets.

Improving the efficiency of the proposed model and its scalability on large graphs are potential directions for future work, as this model currently is not very computational friendly. This motivates us to simplify the *"encoder"* module in this framework, in particular, the MPNN, which we are going to present in the next chapter.

5

SIMPLIFIED GRAPH NEURAL NETWORKS

In this chapter, we focus on the sparsification of GNN models, in particular the MPNN, that we used in Chapter 4. Indeed, the high complexity and computational cost of the **EvoNet** model proposed in Chapter 4 motivates us to find a fast, light-weight and efficient alternative. Concurrently attracting a great amount of academic and industrial interest, the sparsification of neural network models become a promising direction to follow.

Thus, we conduct a structured study of the effect of sparsification on the trainable part of MPNNs known as the *Update* step. To this end, we design a series of models to successively sparsify the linear transform in the *Update* step. Specifically, we propose the *Expander* GNN model with a tuneable sparsification rate and the *Activation-Only* GNN, which has no linear transform in the *Update* step. In agreement with a growing trend in the literature the sparsification paradigm is changed by initialising sparse neural network architectures rather than expensively sparsifying already trained architectures.

Our novel benchmark models enable a better understanding of the influence of the *Update* step on model performance and outperform existing simplified benchmark models such as the Simple Graph Convolution (SGC) (Wu et al., 2019). The *Expander* GNNs, and in some cases the *Activation-Only* models, achieve performance on par with their vanilla counterparts on several downstream tasks, while containing significantly fewer trainable parameters. In experiments with matching parameter numbers our benchmark models outperform the state-of-the-art GNN models.

5.1 INTRODUCTION

Among various GNN models, MPNNs (Gilmer et al., 2017) and their variants are currently considered to be the dominating class. In MPNNs, the learning procedure can be separated into three major steps: *Aggregation*, *Update* and *Readout*, where *Aggregation* and *Update* are repeated iteratively so that each node's representation is updated recursively based on the transformed information aggregated over its neighbourhood.

There is thus a division of labour between the *Aggregation* and the *Update* step, where the *Aggregation* utilises local graph structure, while the *Update* step is only applied to single node representations at a time independent of the local graph structure. From this a natural question then arises:

> *What is the impact of the graph-agnostic Update step on the performance of GNNs?*

Since the *Update* step is the main source of model parameters in MPNNs, understanding its impact is fundamental in the design of parsimonious GNNs.

Wu et al. (2019) first challenged the role of the *Update* step by proposing the SGC model where they removed the non-linearities in the *Update* steps and collapsed the consecutive linear transforms into a single transform. Their experiments showed, surprisingly, that in some instances the *Update* step of GCN (Kipf and Welling, 2017) can be left out completely without the models' accuracy decreasing.

In the same spirit, we propose in this work to analyse the impact of the *Update* step and its sparsification in a systematic way. To this end, we propose two nested model classes, where the *Update* step is successively sparsified. In the first model class which we refer to as *Expander* GNN, the linear transform layers of the *Update* step are sparsified; while in the second model class, the linear transform layers are removed and only the activation functions remain in the model. We name the second model *Activation-Only* GNN and it contrasts the SGC where the activation functions where removed to merge the linear layers.

Inspired by the recent advances in the literature of sparse CNN architectures (Prabhu, Varma, and Namboodiri, 2018), we propose to utilise a random sparsification scheme, which is motivated by the study of expander graphs (hence the model's name).

Here the sparsification is performed at intialisation and accordingly saves the cost of more traditional methods, which often iteratively prune connections during training.

Through a series of empirical assessments on different graph learning tasks (graph and node classification as well as graph regression), we demonstrate that the *Update* step can be heavily simplified without inhibiting performance or relevant model expressivity. Our findings partly agree with the work in Wu et al. (2019), in that dense *Update* steps in GNN are expensive and often ineffectual. In contrast to their proposition, we find that there are many instances in which leaving the *Update* step out completely significantly harms performance. In these instances our *Activation-Only* model shows superior performance while matching the number of parameters and efficiency of the SGC.

Our contributions can be summarised as follows.

1. We explore the impact of the *Update* step and its sparsification in MPNNs through the newly proposed model class of *Expander* GNNs with tuneable density. We show empirically that *a sparse Update step matches the performance of the standard model architectures*.

2. As an extreme case of the *Expander* GNN, as well as an alternative to the SGC, we propose the *Activation-Only* GNNs that remove the linear transformation layer from the *Update* step and keep non-linearity in tact. We observe the *Activation-Only* models to exhibit comparable, sometimes significantly superior performance to the SGC while being equally time and memory efficient.

Both of our proposed model classes can be extrapolated without further efforts to a variety of models in the MPNN framework and hence provide practitioners with an array of efficient and often highly performant benchmark models.

The rest of this chapter is organised as follows. In Section 5.2, we provide an overview of the related work. Section 5.3 presents in detail our two proposed model classes. Section 5.4 discusses our experimental setting and empirical evaluation of the proposed models in a variety of downstream graph learning tasks.

5.2 RELATED WORK

In recent years the idea of *utilising expander graphs in the design of neural networks* is starting to be explored in the CNN literature. Most notably, Prabhu, Varma, and Namboodiri (2018) propose to replace linear fully connected layers in deep networks using an expander graph sampling mechanism and hence, propose a novel CNN architecture they call X-nets. The great innovation of this approach is that well-performing sparse neural network architectures are initialised rather than expensively calculated. Furthermore, they are shown to compare favourably in training speed, accuracy and performance trade-offs to several other state-of-the-art architectures. McDonald and Shokoufandeh (2019) and Kepner and Robinett (2019) build on the X-net design and propose alternative expander sampling mechanisms to extend the simplistic design chosen in the X-nets.

Independent of this literature branch, Bourely, Boueri, and Choromonski (2017) explore 6 different mechanisms to randomly sample expander graph layers. Across the literature the results based on expander graph layers are encouraging.

The *Sparisification and Pruning of neural networks* is a very active research topic (Blalock et al., 2020; Hoefler et al., 2021). In particular, a wealth of algorithms sparsifying neural network architectures at initialisation has recently been proposed (Lee, Ajanthan, and Torr, 2019; Tanaka et al., 2020; Wang, Zhang, and Grosse, 2020). While these algorithms make pruning decisions on a per-weight basis, Frankle et al. (2021) find that these algorithms produce equivalent results to a per-layer choice of a fraction of weights to prune, as is directly done in our chosen sparsification scheme. All of these research efforts are pruning CNNs, typically the VGG and ResNet architectures. *To the best of our knowledge, our work is the first investigating the potential of sparsifying the trainable parameters in GNNs.* This allows us to observe that conclusions drawn for CNNs do not directly carry over to GNNs. We observe that in the majority of cases pruning 90% of trainable weights at initialisation using our relatively simple sparsification scheme comes at no performance cost. The presence of the *Aggregation* step in the graph neural network architecture appears to significantly impact the response of the model to pruning.

Both Wu et al. (2019) and Salha, Hennequin, and Vazirgiannis (2019) observed that *simplifications in the Update step of the GCN model* is a promising area of research. Wu et al. (2019) proposed the SGC model, where simplification is achieved by removing the non-linear activation functions from the GCN model. This removal allows them to merge all linear transformations in the *Update* steps into a single linear transformation without sacrificing expressive power. Salha, Hennequin, and Vazirgiannis (2019) followed a similar rationale in their simplification of the graph autoencoder and variational graph autoencoder models. These works have had an immediate impact on the literature featuring as benchmark models and object of study in many recent papers: The idea of omitting the *Update* step guided Chen et al. (2020) in the design of simplified models and has found successful application in various areas where model complexity needs to be reduced (He et al., 2020; Waradpande, Kudenko, and Khosla, 2020) or very large graphs need to be processed (Salha, Hennequin, and Vazirgiannis, 2020). In our work we aim to extend these efforts by providing more simplified benchmark models for GNNs without a specific focus on the GCN.

5.3 INVESTIGATING THE ROLE OF THE UPDATE STEP

We present in this section the two proposed model classes, where we sparsify or remove the linear transform layer in the *Update* step of MPNNs, with the aim to

systematically analyse the impact of the *Update* step. We begin in subsection 5.3.1 by introducing the model structure of MPNN that we choose to work with. We then demonstrate how the *Expander* GNN and *Activation-Only* GNN are constructed in subsections 5.3.2 and 5.3.3, respectively.

5.3.1 *Message-Passing Neural Networks*

The general model structure of MPNNs has been introduced in subsection 2.4.1. Note that various choices of *Aggregation*, *Update* and *Readout* functions are proposed in the literature. To avoid shifting from the subject of this work, we stay with the simplest and most widely used function choices, such as sum, mean and max aggregators for the *Aggregation*, the MLP for the *Update* step and summation for the *Readout*. As an example to visualise our models in subsections 5.3.2 and 5.3.3 we use the following matrix representation of the GCN's model equation,

$$\boldsymbol{H}^{(L)} = \sigma\left(\hat{\boldsymbol{A}}\ldots\sigma\left(\hat{\boldsymbol{A}}\boldsymbol{H}^{(1)}\boldsymbol{W}^{(1)}\right)\ldots\boldsymbol{W}^{(L)}\right), \tag{5.1}$$

where σ denotes a nonlinear activation function, $\boldsymbol{W}^{(i)}$ contains the trainable weights of the linear transform in the *Update* step and $\hat{\boldsymbol{A}} = \tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}$ is the symmetric normalised adjacency matrix with $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ denoting the adjacency matrix with added self-loops and $\tilde{\boldsymbol{D}}$ being the corresponding degree matrix.

5.3.2 *Sparsifying the* Update *step: Expander GNN*

In this subsection we propose the *Expander*GNN model where sampled expander graphs are used to initialise sparse linear layers in the *Update* step. We begin by discussing how linear layers in a neural networks can be represented by bipartite graphs.

5.3.2.1 *Linear Layer as a graph*

The fully-connected linear transform layer in the MLP can be represented by a bipartite graph $\mathcal{B}(\mathbb{S}_1, \mathbb{S}_2, \mathbb{E})$, where $\mathbb{S}_1$ and $\mathbb{S}_2$ are two sets of nodes and $\mathbb{E}$ the set of edges that satisfy

$$\forall u \in \mathbb{S}_1, \forall v \in \mathbb{S}_2, \exists (u, v) \in \mathbb{E};$$
$$\forall u, v \in \mathbb{S}_1 (\text{resp.} \mathbb{S}_2), \nexists (u, v) \in \mathbb{E}.$$

The number of edges, i. e., parameters, is $|\mathbb{S}_1||\mathbb{S}_2|$ and the edges can be encoded in matrix form by $\boldsymbol{W} \in \mathbb{R}^{|\mathbb{S}_1|\times|\mathbb{S}_2|}$, the weight matrix in (5.1), that maps the input node features of dimension $|\mathbb{S}_1|$ to output node features of dimension $|\mathbb{S}_2|$.

5.3.2.2 *Expander Linear Layer*

Given the bipartite graph corresponding to a linear transform layer $\mathcal{B}(\mathbb{S}_1, \mathbb{S}_2, \mathbb{E})$, we follow the design of Prabhu, Varma, and Namboodiri (2018) to construct the sparsifier by sampling its subgraph of specific expander structure.

DEFINITION 5.1 (EXPANDER LINEAR LAYER). *Suppose $|\mathbb{S}_1| \leq |\mathbb{S}_2|$. For each vertex $u \in \mathbb{S}_1$, we uniformly sample d vertices $\{v_i^u\}_{i=1,\dots,d}$ from $\mathbb{S}_2$ to be connected to u. Then, the constructed graph $\mathcal{B}'(\mathbb{S}_1, \mathbb{S}_2, \mathbb{E}')$ is a subgraph of $\mathcal{B}$ with edge set $\mathbb{E}' = \{(u, v_i^u) : u \in \mathbb{S}_1, i \in \{1, \dots, d\}\}$. Else if $|\mathbb{S}_1| > |\mathbb{S}_2|$, we define the expander sparsifier with the roles of $\mathbb{S}_1$ and $\mathbb{S}_2$ reversed meaning that we sample nodes from $\mathbb{S}_1$. We call a linear layer with a computational graph $\mathcal{B}'(\mathbb{S}_1, \mathbb{S}_2, \mathbb{E}')$ an **expander linear layer**.*

The theoretical computational cost of an expander linear layer is then equal to $2nd \min(|\mathbb{S}_1|, |\mathbb{S}_2|)$ Floating Point Operations (FLOPs). The tunable parameter d can therefore lead to significant computational savings as the computational cost of a fully connected linear layer equals $2n|\mathbb{S}_1||\mathbb{S}_2|$ FLOPs.

We refer to the *density* of the expander linear layer as the ratio of the number of sampled connections to the number of connections in the complete bipartite graph. For example, the fully-connected layer has density 1. The sampling scheme in Definition 5.1 returns an expander linear layer of density $d / \max(|\mathbb{S}_1|, |\mathbb{S}_2|)$.

When we replace all linear layers in the *Update* steps of a GNN with expander linear layers constructed by the sampling scheme in Definition 5.1, we get the *Expander* GNN. An illustration can be found in Figure 5.1.

When compared to pruning algorithms which sparsify neural network layers by iteratively removing parameters according to certain metric during training, the expander sparsifiers have two advantages:

1. The expander design assures that paths exist between consecutive layers, avoiding the risk of *layer-collapse* that is common in many pruning algorithms, where the algorithm prunes all parameters (weights) in one layer and cuts down the flow between input and output (Tanaka et al., 2020).

2. The expander sparsifier removes parameters at initialisation and keeps the sparsified structures fixed during training, which avoids the expensive

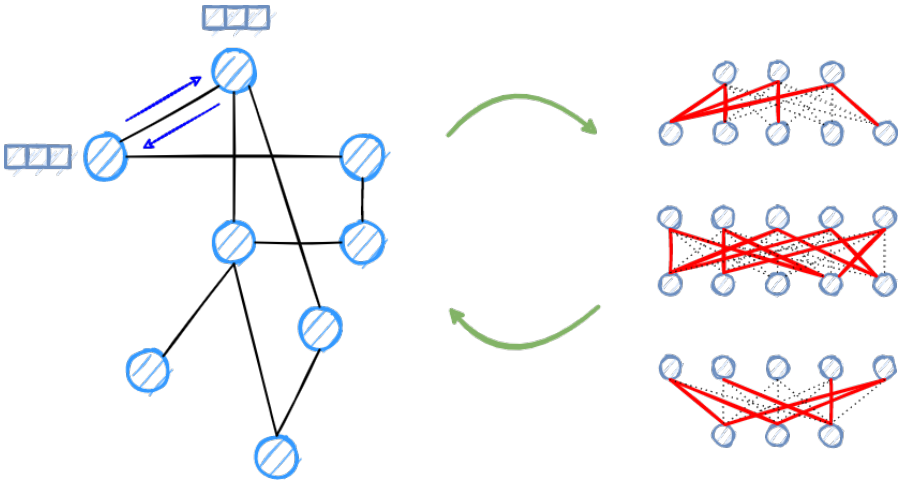

Figure 5.1: Illustration of the main computational steps in *Expander* GNNs. **(Left)** *Aggregation* or graph propagation step and **(Right)** *Update* step. The red lines in the *Update* step represent preserved connections in MLPs sampled as expander sparsifier structures. In the *Aggregation* step only a subset of the exchanged messages are illustrated.

computational cost stemming from adapting the neural network architecture during or after training and then retraining the network as is done in the majority of pruning algorithms (Frankle and Carbin, 2019; Han et al., 2015).

5.3.2.3 *Motivation of Expander Linear Layer*

The sampling scheme in Definition 5.1 samples bipartite graphs with good expansion properties, which are commonly discussed in the field of error correcting codes under the name "lossless expanders" (Hoory, Linial, and Widgerson, 2006, pp. 517-522). Expander graphs can be informally defined to be highly connected and sparse graphs (Lubotzky, 2012). They are successfully applied in communication networks where communication comes at a certain cost and is to be used such that messages are spread across the network efficiently (Lubotzky, 2012).

Equally, in a neural network each parameter (corresponding to an edge in the neural network architecture) incurs a computational cost and is placed to optimise the overall performance of the neural network architecture. Therefore, the use of expander graphs in the design of neural network architectures is conceptually well motivated.

In Bölcskei et al. (2019), the connectedness of a sparse neural network architecture was linked to the complexity of a given function class which can be approximated by sparse neural networks. Hence, utilising neural network

parameters to optimise the connectedness of the network maximises the expressivity of the neural network.

In Kepner and Robinett (2019) and Bourely, Boueri, and Choromonski (2017) the connectedness of the neural network architecture graph was linked – via the path-connectedness and the graph Laplacian eigenvalues – to the performance of neural network architectures. Therefore, for both the expressivity of the neural network and its performance, the connectedness, which is optimised in expander graphs, is a parameter of interest.

5.3.2.4 *Implementation of Expander Linear Layer*

The most straightforward way of implementing the expander linear layer is to store the weight matrix $\boldsymbol{W}$ as a sparse matrix. Sparse matrix multiplications can be accelerated on several processing units released in 2020 and 2021 such as the Sparse Linear Algebra Compute (SLAC) cores used in the Cerebras WSE-2 (Cerebras Systems, INC, 2021), the Intelligence Processing Unit (IPU) produced by Graphcore (Moor Insights and Strategy, 2020) and the NVIDIA A100 Tensor Core GPU (NVIDIA, 2020).

However, since we ran experiments on a NVIDIA RTX 2060 GPU, we use masks in our implementation, similar to those of several existing pruning algorithms, to achieve the sparsification. A mask $\boldsymbol{M} \in \{0,1\}^{|\mathbb{S}_1|\times|\mathbb{S}_2|}$ is of the same dimension as weight matrix and $\boldsymbol{M}_{u,v} = 1$ if and only if $(u,v) \in \mathbb{E}'$. The entrywise multiplication, denoted by $\odot$, is then applied to the mask and the weight matrix so that undesired parameters in the weight matrix are removed, i. e., (5.1) can be rewritten as,

$$\boldsymbol{H}^{(L)} = \sigma\left(\hat{\boldsymbol{A}}\ldots\sigma\left(\hat{\boldsymbol{A}}\boldsymbol{H}^{(1)}\boldsymbol{M}^{(1)}\odot\boldsymbol{W}^{(1)}\right)\ldots\boldsymbol{M}^{(L)}\odot\boldsymbol{W}^{(L)}\right). \tag{5.2}$$

5.3.3 *An Extreme Case: Activation-Only GNN*

In Gama, Ribeiro, and Bruna (2020) it is argued that the non-linearity present in GNNs, in form of the activation functions, has the effect of frequency mixing in the sense that "part of the energy associated with large eigenvalues" is brought "towards low eigenvalues where it can be discriminated by stable graph filters." The theoretical insight that activation functions help capture information stored in the high energy part of graph signals is strong motivation to consider an alternative simplification to the one made in the SGC.

In this alternative simplification, which we refer to as the *Activation-Only* GNN models, we remove linear transformations instead of activation functions

such that each message-passing step is immediately followed by a point-wise activation function. The resulting model can be seen as a natural extension of the *Expander* GNN, where the linear transformation of the *Update* step is completely forgone. Hence, in a *Activation-Only* GNN, (5.1) will be rewritten as,

$$\boldsymbol{H}^{(L)} = \sigma\left(\hat{\boldsymbol{A}} \ldots \sigma\left(\hat{\boldsymbol{A}}\boldsymbol{H}^{(1)}\right)\right). \tag{5.3}$$

This proposed simplification is applicable to a wide variety of GNN models as we will demonstrate in our extensive set of experiments in Section 5.4. For comparison we display the model equation of the SGC (Wu et al., 2019),

$$\boldsymbol{H}^{(L)} = \hat{\boldsymbol{A}}^{L}\boldsymbol{H}^{(1)}\Theta, \tag{5.4}$$

where $\Theta = \boldsymbol{W}^{(1)} \ldots \boldsymbol{W}^{(L)}$. Here the nonlinear activation functions have been removed and the linear transformations have been collapsed into a single linear transformation layer.

Interestingly, we observe that the repeated application of the symmetric matrix $\hat{\boldsymbol{A}}$ to the input data $\boldsymbol{X}$ is equivalent to an unnormalised version of the power method approximating the eigenvector corresponding to the largest eigenvalue of $\hat{\boldsymbol{A}}$. Hence, if sufficiently many layers L are used then inference is drawn in the SGC model simply on the basis of the first eigenvector of $\hat{\boldsymbol{A}}$.

5.4 EXPERIMENTS AND DISCUSSION

In order to study the influence of the *Update* step in GNNs, we proposed a series of models in Section 5.3, where its linear transform is gradually sparsified. By observing the trend of model performance change (on downstream tasks) with respect to the sparsity of the linear transform layer, we measure the impact of the *Update* step. In subsection 5.4.1 we provide an overview of our experimentation setup. Then, in subsections 5.4.2, 5.4.3 and 5.4.4, we observe the performance of the proposed benchmark models on the tasks of graph classification, graph regression and node classification, respectively. In subsection 5.4.5, we compare the performance of *Expander* GNNs and vanilla GNNs when they have equally many parameters and in subsection 5.4.6 we compare the convergence behaviour of the studied models.

Table 5.1: Model Equations of the Vanilla, *Expander* and *Activation-Only* GNN.

Model		Aggregation	Update	Remarks
GCN	Vanilla/*Expander*	$\boldsymbol{m}_i^{(l)} = \frac{1}{\sqrt{d_i}} \sum_{j \in \mathcal{N}(i)} \boldsymbol{h}_j^{(l)} \frac{1}{\sqrt{d_j}}$	$\boldsymbol{h}_i^{(l+1)} = \sigma(\boldsymbol{m}_i^{(l)} \boldsymbol{M}^{(l)} \odot \boldsymbol{W}^{(l)})$	1. **GCN:** d_i denotes the degree of node i. 2. **GIN:** ϵ is a learnable ratio added explicitly to the central node's own representation. 3. **PNA:** $\bigoplus$ corresponds to an operator formed by taking the tensor product of a vector containing three scalar functions and four aggregator functions, resulting in a tensor indexed by i, s, a, where the index i corresponds to the currently considered node, s corresponds to the scalar dimension and a indexes the aggregator dimension. For more details see Corso et al., 2020.
	Activation-Only		$\boldsymbol{h}_i^{(l+1)} = \sigma\left(\boldsymbol{m}_i^{(l)}\right)$	
GIN	Vanilla/*Expander*	$\boldsymbol{m}_i^{(l)} = (1+\epsilon)\boldsymbol{h}_i^{(l)} + \sum_{j \in \mathcal{N}(i)} \boldsymbol{h}_j^{(l)}$	$\boldsymbol{h}_i^{(l+1)} = \sigma(\boldsymbol{m}_i^{(l)} \boldsymbol{M}^{(l)} \odot \boldsymbol{W}^{(l)})$	
	Activation-Only		$\boldsymbol{h}_i^{(l+1)} = \sigma\left(\boldsymbol{m}_i^{(l)}\right)$	
GraphSage	Vanilla/*Expander*	$\boldsymbol{m}_i^{(l)} = \text{CONCAT}\left(\boldsymbol{h}_i^{(l)}, \text{MAX}_{j \in \mathcal{N}(i)} \sigma(\boldsymbol{h}_j^{(l)} \boldsymbol{M}_1^{(l)} \odot \boldsymbol{W}_1^{(l)})\right)$	$\boldsymbol{h}_i^{(l+1)} = \frac{\sigma(\boldsymbol{m}_i^{(l)} \boldsymbol{M}_2^{(l)} \odot \boldsymbol{W}_2^{(l)})}{\|\sigma(\boldsymbol{m}_i^{(l)} \boldsymbol{M}_2^{(l)} \odot \boldsymbol{W}_2^{(l)})\|_2}$	
	Activation-Only	$\boldsymbol{m}_i^{(l)} = \boldsymbol{h}_i^{(l)} + \text{MAX}_{j \in \mathcal{N}(i)} \sigma(\boldsymbol{h}_j^{(l)})$	$\boldsymbol{h}_i^{(l+1)} = \frac{\sigma(\boldsymbol{m}_i^{(l)})}{\|\sigma(\boldsymbol{m}_i^{(l)})\|_2}$	
PNA	Vanilla/*Expander*	$\boldsymbol{m}_i^{(l)} = \text{CONCAT}_{s,a}\left(\bigoplus_{j \in \mathcal{N}(i)} \boldsymbol{h}_j^{(l)}\right)$	$\boldsymbol{h}_i^{(l+1)} = \sigma(\boldsymbol{m}_i^{(l)} \boldsymbol{M}^{(l)} \odot \boldsymbol{W}^{(l)})$	
	Activation-Only	$\boldsymbol{m}_i^{(l)} = \frac{1}{12} \sum_{s,a} \left[\bigoplus_{j \in \mathcal{N}(i)} \boldsymbol{h}_j^{(l)}\right]_{i,s,a}$	$\boldsymbol{h}_i^{(l+1)} = \sigma\left(\boldsymbol{m}_i^{(l)}\right)$	

5.4.1 *General Settings and Baselines*

5.4.1.1 *Considered GNNs*

Throughout this section we refer to the standard, already published, architectures as "vanilla" architectures. We compare the performance of the vanilla GNN models, the *Expander* GNN models with different densities (10% and 50%), the *Activation-Only* GNN model (we report the best result obtained from the ReLU, PReLU and Tanh activation functions), as well as the SGC for the GCN models. To ensure that our inference is not specific to a certain GNN architecture only, we evaluate the performance across four representative GNN models of the literature state-of-the-art. The considered models are the GCN (Kipf and Welling, 2017), the GIN (Xu et al., 2018), the GraphSage (Hamilton, Ying, and Leskovec, 2017), and the PNA (Corso et al., 2020). The precise model equations of our proposed architectures applied to these GNNs can be found in Table 5.1.

5.4.1.2 *Datasets*

We experiment on eleven datasets from areas such as chemistry, social networks, computer vision and academic citation, for three major graph learning tasks.

For graph classification, we have four TU datasets (Kersting et al., 2016) which are either chemical or social network graphs, and two Image datasets (MNIST/CIFAR10) that are constructed from original images following the procedure in Knyazev, Taylor, and Amer (2019). To perform this conversion they first extract small regions of homogeneous intensity from the images, named "Superpixels" (Dwivedi et al., 2020), and construct a K-nearest neighbour graph from these superpixels. The technique we implemented to extract superpixels, the choice of K and distance kernel for constructing a nearest neighbour graph are the same as in Knyazev, Taylor, and Amer (2019) and Dwivedi et al. (2020).

For graph regression, we consider molecule graphs from the ZINC dataset (Irwin et al., 2012). And for node classification, we use four citation datasets (Hu et al., 2020; Sen et al., 2008; Wang et al., 2020), where the nodes are academic articles linked by citations. Details of the used datasets can be found in Table 2.1.

5.4.1.3 *Experimentation Details*

Since we aim to observe the performance of our benchmark models independent of the GNN choice, we use the model hyper-parameters found to yield a fair comparison of GNN models in Dwivedi et al. (2020).

Specifically, we follow the same training procedure, such as train/valid/test dataset splits, choice of optimiser, learning rate decay scheme, as well as the same hyper-parameters, such as initial learning rate, hidden feature dimensions and number of GNN layers. We also implement the same normalisation tricks such as adding batch normalisation after non-linearity of each *Update* step.

For the node classification task on citation datasets, we follow the settings from Wu et al. (2019). Our experiments found that the node classification task on citation graphs of small to medium size can be easily overfit and model performances heavily depend on the choice of hyper-parameters. Using the same parameters with Wu et al. (2019), such as learning rate, number of training epochs and number of GNN layers, helps us achieve similar results with the paper on the same model, which allows a fair comparison between the proposed *Activation-Only* models and the SGC. Our implementation is built upon the open source library Deep Graph Library (DGL) (Wang et al., 2019). The experiments are run on a Intel(R) Xeon(R) W-2123 processor with 64GB ram and a NVIDIA GeForce RTX 2080Ti GPU with 12GB ram.

5.4.1.4 *Loss functions*

After L message-passing iterations, we obtain

$$\boldsymbol{H}^{(L)} = \left[\boldsymbol{h}_1^{(L)}, \ldots, \boldsymbol{h}_n^{(L)}\right]^{\mathrm{T}} \in \mathbb{R}^{n \times p},$$

as the final node embedding, where we denote p as its feature dimension. Depending on the downstream task, we either keep working with $\boldsymbol{H}^{(L)}$ or construct a graph-level representation $\boldsymbol{g}$ from $\boldsymbol{H}^{(L)}$,

$$\boldsymbol{g} = \frac{1}{n} \sum_{i \in \mathbb{V}} \boldsymbol{h}_i^{(L)},$$

which we referred to as the *Readout* step in Section 2.4.1. $\boldsymbol{g}$ or $\boldsymbol{H}^{(L)}$ is then fed into a fully-connected network (MLP) to be transformed into the desired form of output for further assessment, e. g., a scalar value as a prediction score in graph regression. We denote this network as $f(\cdot)$, which, in our experiments, is fixed to be a three-layer MLP of the form

$$f(x) = \sigma(\sigma(x\boldsymbol{W}_1)\boldsymbol{W}_2)\boldsymbol{W}_3,$$

where $\boldsymbol{W}_1 \in \mathbb{R}^{p \times (p/2)}$, $\boldsymbol{W}_2 \in \mathbb{R}^{(p/2) \times (p/4)}$, $\boldsymbol{W}_3 \in \mathbb{R}^{(p/4) \times k}$ with k being the desired output dimension. The final output, either $f(\boldsymbol{g})$ or $f(\boldsymbol{H}^{(L)})$, is compared

to the ground-truth by a task-specific loss function. For graph classification and node classification, we choose cross-entropy loss and for graph regression, we use Mean Absolute Error (MAE).

5.4.2 *Graph Classification*

Table 5.2 shows the experiment results of the vanilla GCN/GIN models and their *Expander* and *Activation-Only* variants on the ENZYMES, DD, PROTEINS and IMDB-BINARY datasets for graph classification. The evaluation metric is classification accuracy, where the average accuracy, obtained from a 10-folder cross validation, is used. Note that we also report in Table 5.2 the number of parameters of the different models, normalised by the number of parameters in the vanilla model, e.g. 0.37 means that the number of parameters in *Activation-Only* model is 37% of the vanilla one.

One direct observation from Table 5.2 is that the *Expander* GCN models, even at 10% density, perform on par with the vanilla models. Surprisingly, the same is true for the *Activation-Only* model on the ENZYMES, DD and PROTEINS datasets. IMDB-BINARY is our only graph classification dataset where the node attributes are initialised to all be equal. This uninformative initialisation seems to lead to an increased performance if the linear *Update* step is present, visible in the performance gap of the *Activation-Only* models and the *Expander* GCN models. While for GIN model we still observe the *Activation-Only* model and to match the performance of the vanilla and *Expander* GIN. In most cases, the SGC performs worse than the *Activation-Only* GCN model.

Table 5.3 shows the graph classification results for the MNIST and CIFAR10 datasets. Due to the large computational cost implied by the size of these datasets we were unable to produce confidence intervals for the provided results. The GCN *Activation-Only* model outperforms the SGC by a larger margin than we observed on the TU datasets in Table 5.2. It seems that especially for these computer vision datasets the presence of activation functions in the GCN architecture has a large positive impact on model performance in the graph classification task. Additional experiments on the GIN architecture reaffirm the conclusions drawn on the GCN model on graph classification.

5.4.3 *Graph Regression*

In Table 5.4 the MAE of our studied and proposed models on the ZINC dataset for graph regression is displayed. Similar to the graph classification task, the

Table 5.2: 10-fold Cross Validation results (mean ± std) of the GCN/GIN on the graph classification task performed on the ENZYMES/DD/PROTEINS/IMDB-BINARY datasets. We set the best results to bold and underline the second best result. In addition, if the result of *Activation-Only* GCN is better than the SGC model, we put $*$ next to the result.

	ENZYMES		DD		Proteins		IMDB-BINARY	
GCN	ACC.	Params.	ACC.	Params.	ACC.	Params.	ACC.	Params.
Vanilla	$\underline{66.50 \pm 8.71}$	1.00	75.13 ± 3.44	1.00	$\mathbf{76.73 \pm 3.85}$	1.00	$\mathbf{72.70 \pm 5.68}$	1.00
SGC	63.67 ± 8.06	0.37	$\underline{75.90 \pm 3.93}$	0.43	67.65 ± 2.21	0.38	61.30 ± 3.61	0.35
Expander-50%	64.83 ± 8.64	0.57	74.53 ± 3.50	0.56	76.36 ± 3.43	0.57	$\underline{72.40 \pm 5.70}$	0.57
Expander-10%	66.33 ± 6.78	0.22	74.28 ± 2.52	0.21	$\underline{76.55 \pm 1.90}$	0.22	71.60 ± 5.50	0.22
Activation-Only	$\mathbf{66.67 \pm 6.71}*$	0.37	$\mathbf{76.57 \pm 5.20}*$	0.43	$75.92 \pm 2.88*$	0.38	$62.70 \pm 3.32*$	0.35
GIN	ACC.	Params.	ACC.	Params.	ACC.	Params.	ACC.	Params.
Vanilla	$\mathbf{67.67 \pm 7.68}$	1.00	68.76 ± 5.55	1.00	$\mathbf{72.51 \pm 2.39}$	1.00	69.00 ± 6.23	1.00
Expander-50%	$\underline{67.00 \pm 6.05}$	0.54	$\underline{70.03 \pm 4.20}$	0.51	70.08 ± 2.69	0.52	68.80 ± 5.90	0.52
Expander-10%	65.83 ± 7.75	0.16	68.59 ± 2.70	0.12	70.53 ± 3.96	0.13	$\mathbf{72.00 \pm 6.16}$	0.13
Activation-Only	62.83 ± 7.15	0.10	$\mathbf{72.49 \pm 4.30}$	0.19	$\underline{72.40 \pm 5.03}$	0.08	$\underline{69.50 \pm 3.88}$	0.03

Table 5.3: Results of the GCN/GIN on graph classification for the MNIST/CIFAR10 datasets. The format follows Table 5.2.

	MNIST		CIFAR10	
GCN	ACC.	Params.	ACC.	Params.
Vanilla	**90.77**	1.00	**52.04**	1.00
SGC	24.48	0.36	27.90	0.36
Expander-50%	90.75	0.57	50.69	0.57
Expander-10%	89.00	0.23	50.27	0.23
Activation-Only	83.84∗	0.36	48.31∗	0.36
GIN	ACC.	Params.	ACC.	Params.
Vanilla	90.33	1.00	**42.46**	1.00
Expander-50%	**92.31**	0.56	40.35	0.56
Expander-10%	88.73	0.20	35.93	0.20
Activation-Only	79.49	0.11	39.71	0.11

Expander GCN and *Expander* GraphSage models are on the same level with their corresponding vanilla models, regardless of their densities. The performance of the *Expander* GIN and *Expander* PNA models exhibits greater variance across the different densities, especially in the case of the PNA models the performance increases as the network gets denser indicating that the density of the *Update* step does positively contribute to the model performance of the PNA for the task of graph regression on the ZINC dataset. The *Activation-Only* models perform worse than their *Expander* counterparts on this task, again confirming the insight from the results of the *Expander* GNNs that the linear transform in the *Update* step does improve performance in this graph regression task. Again we see that *Activation-Only* GCN outperforms the SGC benchmark in this set of experiments.

Hence, for the task of graph regression we observe that both the linear transformation and non-linear activation function in the *Update* step have a positive impact on model performance. We might have been able to expect that the addition of the transformation performed in the *Update* step is of greater impact in a regression task, which is evaluated on a continuous scale, than in a classification task, where only a discrete label needs to be inferred.

Table 5.4: Results of the GCN/GIN/GraphSage/PNA on graph regression for the ZINC dataset. The format follows Table 5.2. The symbol ↓ highlights that smaller values correspond to better performance.

	GCN		GIN		GraphSage		PNA	
	MAE ↓	Params.	MAE ↓	Params.	MAE ↓	Params.	MAE ↓	Params.
Vanilla	**0.3823**	1.00	0.4939	1.00	**0.4530**	1.00	**0.3180**	1.00
SGC	0.6963	0.35	—	—	—	—	—	—
Expander-50%	0.3856	0.57	0.5274	0.51	0.4580	0.54	0.3380	0.51
Expander-10%	0.3958	0.22	**0.4888**	0.12	0.4720	0.17	0.3800	0.12
Activation-Only	0.5855∗	0.13	0.5220	0.01	0.4910	0.07	0.4490	0.02

Table 5.5: 10-fold Cross Validation results (mean ± std) of the GCN/GIN on node classification for the CORA/CiteSeer/PubMed/OGBN-Arxiv datasets. The format follows Table 5.2.

	Cora		CiteSeer		PubMed		OGBN-Arxiv	
GCN	ACC.	Params.	ACC.	Params.	ACC.	Params.	ACC.	Params.
Vanilla	80.54 ± 0.44	1.00	69.50 ± 0.19	1.00	79.04 ± 0.12	1.00	71.22 ± 0.76	1.00
SGC	80.40 ± 0.00	0.03	$\mathbf{72.70 \pm 0.00}$	0.02	78.90 ± 0.00	0.01	66.53 ± 0.07	0.05
Expander-50%	80.42 ± 0.28	0.50	69.43 ± 0.34	0.50	$\mathbf{79.34 \pm 0.28}$	0.50	$\mathbf{71.42 \pm 0.55}$	0.56
Expander-10%	$\mathbf{80.59 \pm 0.64}$	0.10	68.68 ± 0.73	0.10	78.95 ± 0.63	0.11	70.70 ± 0.42	0.19
Activation-Only	80.40 ± 0.00	0.03	$\mathbf{72.70 \pm 0.00}$	0.02	78.90 ± 0.00	0.01	68.29 ± 0.13	0.05
GIN	ACC.	Params.	ACC.	Params.	ACC.	Params.	ACC.	Params.
Vanilla	76.57 ± 1.36	1.00	$\mathbf{68.33 \pm 0.56}$	1.00	76.55 ± 0.84	1.00	$\mathbf{69.37 \pm 0.34}$	1.00
Expander-50%	77.06 ± 0.81	0.52	67.24 ± 0.49	0.51	76.06 ± 1.14	0.51	69.11 ± 0.86	0.59
Expander-10%	77.08 ± 0.96	0.13	64.83 ± 0.49	0.12	76.91 ± 0.51	0.12	68.78 ± 0.31	0.26
Activation-Only	$\mathbf{78.65 \pm 0.36}$	0.07	66.70 ± 0.65	0.06	$\mathbf{77.46 \pm 0.67}$	0.02	64.10 ± 0.32	0.08

5.4.4 *Node Classification*

Results from the node classification experiments on four citation graphs (CORA, CITESEER, PUBMED and ogbn-arxiv) can be found in Table 5.5. For medium-sized datasets such as CORA, CITESEER and PUBMED, we have the same observation as for the graph classification and graph regression tasks discussed in Sections 5.4.2 and 5.4.3, the *Expander* models, regardless of their sparsity, are performing on par with the vanilla ones. Only on the CITESEER dataset, we observe that the *Expander* GIN model with 10% density shows a small but non-negligible drop in performance compared to the vanilla model; while the 50% and 90% dense *Expander* models remain comparable to the vanilla one. The *Activation-Only* models also perform as well as or even better than (on CITESEER) the vanilla model. The performance of the GCN *Activation-Only* model and SGC is equally good across all three datasets.

These conclusions remain true for the large-scale dataset ogbn-arxiv with 169,343 nodes and 1,166,243 edges. The *ExpanderGCNs* are on par with the vanilla GCN while the *Activation-Only* model and SGC perform slightly worse. However, the training time of *Activation-Only* model and SGC is five times faster than that of the *Expander* and vanilla models. The *Activation-Only* model outperforms the SGC.

We observe that in the node classification task both the linear transformation and the non-linear activation function offer no benefit for the medium scale datasets. For the large-scale dataset we find that the linear transformation can be sparsified heavily without a loss in performance, but deleting it entirely does worsen model performance.

5.4.5 *Expander Sparsification*

DIFFERENT SAMPLERS In Figure 5.2(a) we compare the results from an *Expander* GCN to those achieved by a GCN model, where the expander linear layer, presented in Definition 5.1, is replaced by a deterministic sparsification construction from Bourely, Boueri, and Choromonski (2017) the "Regular Rotating Edge Construction (RREC)". Bourely, Boueri, and Choromonski (2017) observe that sparsifiers obtained from the RREC sampler have a significant lower algebraic connectivity than the Expander Linear Layer sampler which we chose to utilise in the *Expander* GNNs. We observe that *Expander* GCN outperforms the RREC sampled GCN for almost all parameter budgets. Therefore, we confirm that the

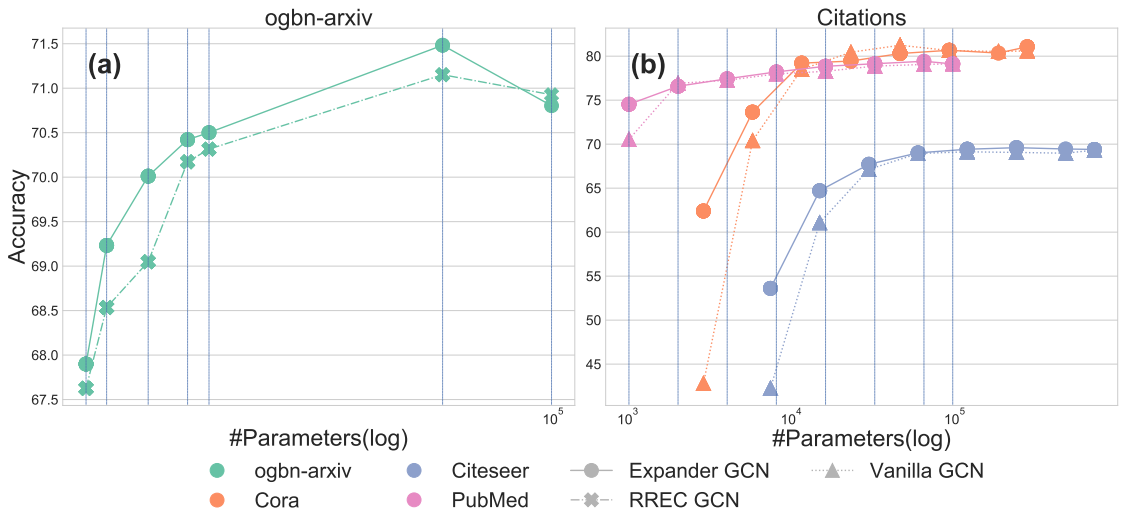

Figure 5.2: (a,b): Accuracy vs. Number of parameters plot on a logarithmic x-axis on (a) ogbn-arxiv for GCN models with different sparsifiers and (b) on CORA/CITESEER/PUBMED for vanilla and *Expander* GCN under the same parameter budget.

Expander Linear Layer is an appropriate choice for the *Expander* GNN model class.

SPARSE WIDE MODEL VS. DENSE NARROW MODEL The experiments in Sections 5.4.2, 5.4.3 and 5.4.4 show that the linear transform layer in the *Update* step of GNNs can often be sparsified to an arbitrary level without loss of performance. From this a natural question then arises:

> Will a shrunk model, i. e., a model with a smaller hidden dimension used in the *Update* step, matching the number of parameters of the sparsified *Expander* GNN, perform on par with its *Expander* GNN counterpart?

To study this question we compare the performance of vanilla GCNs to *Expander* GCNs with equally many parameters, but doubling the size of the hidden dimension of the vanilla GCN. Figure 5.2(b) shows the experiment results on the three citation datasets. We observe that for most parameter values the *Expander* GCN outperforms the vanilla GCN. This phenomenon becomes more evident when the number of parameter is small. In conclusion, it seems to be beneficial to choose a sparsified large model rather than a compact model with equally many parameters.

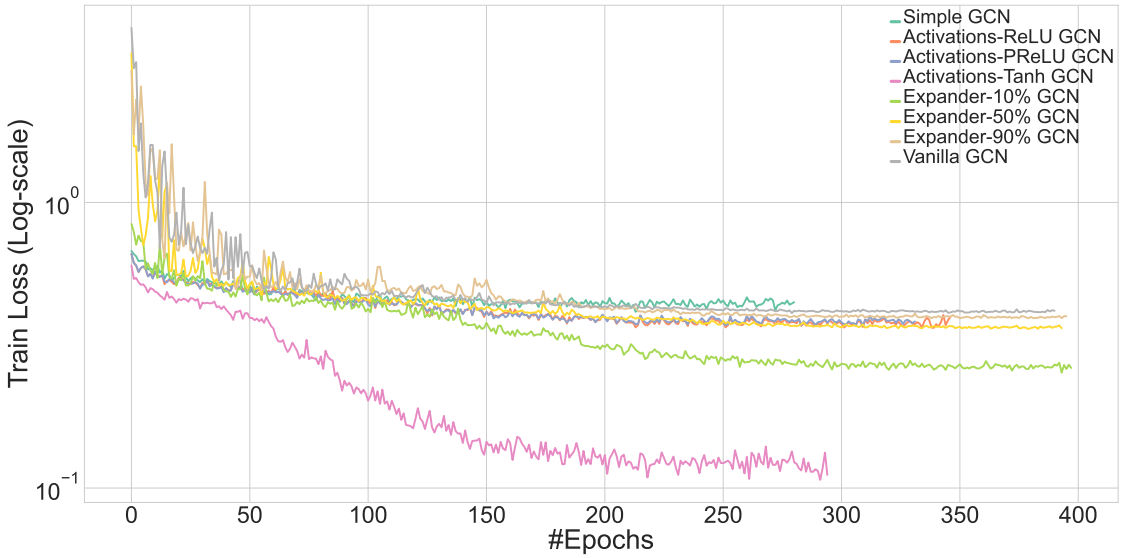

Figure 5.3: Training loss (cross-entropy) convergence behaviour of the different model types for the GCN used for graph classification on the PROTEINS dataset.

5.4.6 *Convergence Behaviour*

In Figure 5.3 we observe the training loss convergence of the vanilla GCN, *Expander* GCNs, *Activation-Only* GCNs and the SGC. When training these models we have implemented a learning rate decay scheme, where the training process is terminated if the learning rate drops below 10^{-6}. We are able to observe, that the number of epochs required for convergence is roughly equal for the *Activation-Only* GCN and SGC, as well as the *Expander* GCN and the vanilla GCN. For both pairwise comparisons we are able to observe, that our proposed models converge to a lower training loss than their counterparts.

5.5 CHAPTER CONCLUSION

With extensive experiments across different GNN models and graph learning tasks, we are able to confirm that the *Update* step can be sparsified heavily without a significant performance cost.

In fact for seven of the eleven tested datasets across a variety of tasks we found that the linear transform can be removed entirely without a loss in performance, i. e., the *Activation-Only* models performed on par with their vanilla counterparts, with around 70% reduction in number of parameters. The *Activation-Only* GCN model consistently outperformed the SGC model and especially in the computer vision datasets we witnessed that the activation functions seem to be crucial for good model performance accounting for an accuracy difference of up to 59%.

In Wu et al. (2019) it is hypothesised that "the non-linearity between GCN layers is not critical - but that the majority of the benefit arises from the local averaging." The above findings partially support this hypothesis as our results on the *Expander* GNNs also indicate that the *Update* step can be simplified significantly without a loss in performance. Contrary to Wu et al. (2019), we find that in many tasks and datasets, nonlinear activation functions result in a significant accuracy boost and it is in fact the linear transformation in the *Update* step that can be removed or heavily sparsified.

The *Activation-Only* GNN is an effective and simple benchmark model framework for any Message-Passing Neural Network. It enables practitioners to test whether they can cut the large amount of model parameters used in the linear transform of the *Update* steps. If the linear transform does contribute positively to the model's performance then the *Expander* GNNs provide a model class of tuneable sparsity which allows efficient parameter usage.

The finding in this work also raises a disturbing concern:

> Since the *Aggregation* step of graph structure plays a most important role in the MPNN, will it as well be more vulnerable to the graph structure noise?

We will investigate this issue in the next chapter on how will the disturbance of graph structure impact the learning process of the MPNN.

6

ROBUST GRAPH CONVOLUTIONAL NEURAL NETWORKS

In this chapter, we further investigate the division of labour between the *Aggregation* and the *Update* step in MPNNs and how one impacts the other. We learned from Chapter 5 that the *Aggregation* step which incorporates graph structure information plays a more important role than the *Update* step that utilise node feature information. *Does there exist an approach to enhance the node feature information in MPNNs?* This question is in particular useful under the noisy setting when the graph structure is perturbed, to which the MPNN architecture is *a priori* vulnerable.

Thus in this work, we study the robustness of MPNNs to perturbations of their input. We focus on the much used GCN and introduce the *random* GCN for which a random matrix theory analysis is possible. This analysis suggests that if the graph is sufficiently perturbed, or in the extreme case random, then the GCN fails to benefit from the node features. It is furthermore observed that enhancing the message passing step in GCNs by adding the node feature kernel to the adjacency matrix of the graph structure solves this problem. An empirical study of a GCN utilised for node classification on both synthetic and six real datasets further confirms the theoretical findings and demonstrates that perturbations of the graph structure can result in GCNs performing significantly worse than MLPs run on the node features alone. In practice, adding a node feature kernel to the message passing of perturbed graphs results in a significant improvement of the GCN's performance, thereby rendering it more robust to graph perturbations.

6.1 INTRODUCTION

With GNNs being a more and more impactful model type for the analysis of graph data, many architectures have been proposed in recent years, successively improving on weaknesses of previous architectures (e.g. Corso et al. (2020), Hamilton, Ying, and Leskovec (2017), and Xu et al. (2019)). A popular GNN architecture which has remained a benchmark throughout the past years, partly due to the simplicity of its model equation and partly due to its good performance is GCN (Kipf and Welling, 2017). The GCN is part of the MPNN model class, where the computations are split into a message passing step in which node features

are aggregated over neighbourhoods in the underlying graph structure and an update step in which node features are processed, most commonly by a MLP.

While much work is being done in the empirical exploration of GNNs, relatively fewer advances have been made in their theoretical analysis. A major advance in the theoretical line of research was the expressivity analysis of different message passing operators performed by Xu et al. (2019) and Morris et al. (2019). These analyses inspired many researchers to further investigate the expressivity of GNNs and resulted in a multitude of new architectures being proposed (Dasoulas et al., 2020; Maron et al., 2019). Another upcoming topic in the development of GNNs is their robustness to perturbations of the underlying graph structure (Sun et al., 2018; Zügner and Günnemann, 2019).

In this work, we introduce the *random* GCN, in which parameters of the *Update* step are randomly sampled from Gaussian distributions rather than trained as is commonly the case. The *random* GCN allows us to make use of several powerful random matrix theory tools to gain a theoretical understanding of the factors driving the inference obtained from the GCN model. Our most insightful hypothesis obtained in this way is that,

> *the message passing step dilutes (or in the extreme case completely ignores) information present in the node features if the underlying graph structure is noisy (or in the extreme case completely random).*

In our theoretical analysis we observe that if information of the node features is introduced to the message passing operation, then this loss of information is avoided. This leads us to hypothesise that the addition of the node feature kernel to the message passing operators in GNNs could render them more robust to noise or mispecification of the underlying graph structure.

In a second part of our presented work we test the hypotheses, obtained in our study of the *random* GCN, on the state-of-the-art GCN architecture applied to both synthetic and six real-world benchmark datasets. This allows us to empirically verify our theoretical insight, rendering the random features approach for theoretical analysis a promising avenue for further theoretical study of GNN architectures, and the inclusion of node feature information in the message passing step a valid method to increase the robustness of GNNs.

Our main findings may be summarised as:

1. We contribute both a theoretical and an empirical understanding of how graph and node feature information is processed by the GCN.

2. Most importantly, we find that the preservation of node feature information is entirely dependent on an informative underlying graph structure.

3. We furthermore, propose a novel GCN message passing scheme which results in more robust inference from a GCN to structural noise.

The remainder of this paper is organised as follows. In Section 6.2 we introduce related literature. In Section 6.3 we propose the *random* GCN and analyse it using tools from random matrix theory. The theoretical insight from Section 6.3 is then empirically verified in Section 6.4, where we confirm our hypotheses on the standard GCN on synthetic and six real benchmark datasets and observe the robust performance of the GCN when the node feature kernel matrix is added in the message passing step.

6.2 RELATED WORK

The robustness of GNNs to perturbations of their input is becoming a topic of increasing importance. There exists an extensive literature branch which studies *adversarial attack and defence strategies on graph data* in the context of GNNs summarised in Günnemann (2022), Zhou, Zheng, and Huang (2020) and Sun et al. (2018) with the latter pointing out directly the need for the development of more robust GNNs. In this work, we present one approach to robustifying the performance of GNNs to graph perturbations. In this literature the focus often lies on specific attack strategies perturbing the graph structure in order to alter the inference obtained from a GNN, most commonly the GCN, and defence strategies which aim to develop methodology which is robust to these attacks.

Recent advances in this literature include, Zügner and Günnemann (2019) proposing a meta learning approach to find optimal graph perturbations. Their perturbation mechanism is found to drastically decrease the global performance of GNNs to be in some cases worse than simple benchmarks such as logistic regression run on the node features only. Zügner and Günnemann (2020) propose an algorithm which certifies robustness of individual nodes for the GCN used for node classification under perturbations of the graph structure. In Geisler, Zügner, and Günnemann (2020) and Jin et al. (2021) the message passing operator in the GCN is replaced by the Soft Mediod function and the sum of several distance based adjacency matrices, respectively, with the aim of more robust GCN performance. Jin et al. (2020) propose to learn the graph structure jointly with the GNN parameters to robustify performance and also Entezari et al. (2020) propose to alter the graph structure by using a low rank approximation of the adjacency matrix. The works of Zhu et al. (2019) and Zhang and Zitnik (2020) are most closely related to our proposition of using a node feature kernel to reweight edges in Section 6.3.3 as they both propose to reweight edges based on

the node features. Our theoretical findings in Section 6.3 support this approach of more directly taking the node features into account in the aggregation scheme of GNNs to increasing their robustness.

This work distinguishes itself from adversarial attacks and defence literature fundamentally in that we study untargeted, random graph perturbations which arise as a result of mispecification of the data or uncertainty in the recording methods of the networks. For this kind of perturbation we are able to provide both theoretical (on a toy data example) and empirical understanding, which enables us to offer a distinction between the node feature data and the graph data in networks data sets and how these different information sources are processed by a GNN architecture.

Our work is also related to the literature studying the challenges that heterophilic graphs pose for GNNs (Pei et al., 2020; Zhu et al., 2021, 2020). This literature distinguishes homophilic and heterophilic graphs, in which edges in the graph predominantly connect nodes of equal and unequal classes, respectively. Both of these structures can be, from a theoretical standpoint, equally class-informative, it is only the structure of the class-information which varies. In our work here we consider an orthogonal problem, which is the situation of a diminishing class-structure in the graph, independent of its homo- or heterophilic nature, and the effect this diminishment has on the ability of GNNs to process the information contained in the node features.

6.3 ANALYSIS OF THE RANDOM GCN

In this section we present our theoretical analysis and main findings. Specifically, we consider a *random* GCN model[1], defined as

$$\boldsymbol{\Phi} = \sigma(\tilde{\boldsymbol{A}}\boldsymbol{X}\boldsymbol{W}), \tag{6.1}$$

where $\tilde{\boldsymbol{A}} \in \mathbb{R}^{n\times n}$ denotes the normalised adjacency operator encoding the graph structure (see (6.3) for its definition), $\boldsymbol{X} \in \mathbb{R}^{n\times p}$ corresponds to the node features, $\boldsymbol{W} \in \mathbb{R}^{p\times d}$ is a random matrix with $W_{ij} \sim \mathcal{N}(0,1)$ independent and identically distributed (i.i.d.) and σ is an activation function applied entry-wise.

1 In Section 6.4.2.1, we show that the performance of the large *random* GCN matches that of the vanilla GCN.

In particular, we will study the spectral behaviour of the *Gram matrix*[2] defined as

$$\boldsymbol{G} = \frac{1}{d}\boldsymbol{\Phi}\boldsymbol{\Phi}^\intercal = \frac{1}{d}\sigma(\tilde{\boldsymbol{A}}\boldsymbol{X}\boldsymbol{W})\sigma(\boldsymbol{W}^\intercal\boldsymbol{X}^\intercal\tilde{\boldsymbol{A}}^\intercal). \tag{6.2}$$

Before getting into the analysis of $\boldsymbol{G}$ we require assumptions on the node features and graph structure.

ASSUMPTION 1 (NODE FEATURES). *We suppose that $\boldsymbol{X}^\intercal = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n] \in \mathbb{R}^{p\times n}$, where $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ are independent node feature vectors, each being a sample from one of $k = 2$ distribution classes $\mathcal{C}_1$ and $\mathcal{C}_2$. We further assume that the node feature vectors $\boldsymbol{x}_i$ follow a Gaussian mixture model; Specifically, for $\boldsymbol{x}_i \in \mathcal{C}_a$, $\boldsymbol{x}_i = (-1)^a \frac{\boldsymbol{\mu}}{\sqrt{p}} + \boldsymbol{z}_i$ for some vector $\boldsymbol{\mu} \in \mathbb{R}^p$ and $\boldsymbol{z}_i \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_p/p)$.*

We stress that Assumption 1 can be relaxed to a larger class of random vectors $\boldsymbol{x} \in \mathcal{X}$, where $\mathcal{X}$ denotes any normed space, satisfying the concentration property,

$$\mathbb{P}(|\varphi(\boldsymbol{x}) - \mathbb{E}[\varphi(\boldsymbol{x})]| > t) \leq Ce^{-(t/\sigma)^q},$$

with $q \in \mathbb{R}^+$, for all 1-Lipschitz functions $\varphi : \mathcal{X} \to \mathbb{R}$.

Such vectors are called *random concentrated vectors* and have the particular property to be stable by Lipschitz transformations (Louart and Couillet, 2018). The simplest example of concentrated vectors is the standard Gaussian vector $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_p)$ (Ledoux, 2001). A more complicated class of examples arises from the fact that the concentration property is stable through Lipschitz maps:

REMARK 6.3.1. *if $\boldsymbol{z} \in \mathbb{R}^d$ is concentrated and $g : \mathbb{R}^d \to \mathbb{R}^p$ is 1-Lipschitz, then $g(\boldsymbol{z})$ is also concentrated.*

A large family of *generative models* falls under this more complicated class of examples, such as, the "fake" images generated by due to these images being constructed as Lipschitz transformations of random Gaussian vectors (Seddik et al., 2020).

Now we introduce the underlying model that defines the graph structure. We assume that the adjacency matrix $\boldsymbol{A}$ of the graph is generated by a SBM (Karrer and Newman, 2011).

ASSUMPTION 2 (GRAPH STRUCTURE). *We assume that the entries of $\boldsymbol{A}$ are independent (except for $A_{ii} = 1$ for all i) Bernoulli random variables with parameter*

2 $\boldsymbol{G}$ provides access to the internal functioning and performance evaluation of the *random* GCN.

$\pi_{ij} = q^2 C_{ab} \in (0,1)$ for $\boldsymbol{x}_i \in \mathcal{C}_a$ and $\boldsymbol{x}_j \in \mathcal{C}_b$. In particular, $q \in (0,1)$ represents the probability of an edge occurring between two nodes, while C_{ab} represents the probability of an edge arising between nodes in classes $\mathcal{C}_a$ and $\mathcal{C}_b$.

Note that self-loops are implicitly added in Assumption 2, where we assume $A_{ii} = 1$ for all i. Therefore, we consider that the normalised adjacency operator in (6.1) is defined as

$$\tilde{\boldsymbol{A}} = \frac{1}{\sqrt{n}} \left(\boldsymbol{A} - \boldsymbol{q}\boldsymbol{q}^\intercal \right), \tag{6.3}$$

where $\boldsymbol{q} = q\mathbf{1}_n$[3]. The centring with $\boldsymbol{q}\boldsymbol{q}^\intercal$ is necessary for the eigenvectors corresponding to the extremal eigenvalues of the operator to be class informative (Li, Han, and Wu, 2018), i.e., the centring operation removes the uninformative eigenvector corresponding to the largest eigenvalue of the adjacency matrix, simplifying the theoretical analysis. Specifically, for our analysis in the asymptotic regime where $n \to \infty$ (see Assumption 3 subsequently), the centring with $\boldsymbol{q}\boldsymbol{q}^\intercal$ and the normalisation by $\frac{1}{\sqrt{n}}$ are required so that $\tilde{\boldsymbol{A}}$ has a bounded spectral norm asymptotically. In practice, the centring by $\boldsymbol{q}\boldsymbol{q}^\intercal$ is not feasible as it results in a dense matrix. In our experiments in Section 6.4, we see this discrepancy to be of little consequence in practice.

REMARK 6.3.2. *Assumption 2 allows us to sample directed as well as undirected graphs. Often the spectral analysis of graphs needs to be restricted to undirected graphs, since the analysis of complex-valued spectra arising for directed graphs poses a significant challenge. We are able to include directed graphs since the Gram matrix, analysed in Section 6.3.1, and $\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^\intercal$, analysed in Section 6.3.2, have real spectra even if the underlying graph structure is directed.*

6.3.1 *Spectral Behaviour of the Gram Matrix*

Let $\tilde{\boldsymbol{X}}^\intercal = [\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_n] = \boldsymbol{X}^\intercal \tilde{\boldsymbol{A}} \in \mathbb{R}^{p \times n}$, the entries of the Gram matrix defined in (6.2) are given by

$$G_{ij} = \frac{1}{d} \sigma(\boldsymbol{W}^\intercal \tilde{\boldsymbol{x}}_i)^\intercal \sigma(\boldsymbol{W}^\intercal \tilde{\boldsymbol{x}}_j) = \frac{1}{d} \sum_{\ell=1}^{d} \sigma(\boldsymbol{w}_\ell^\intercal \tilde{\boldsymbol{x}}_i) \sigma(\boldsymbol{w}_\ell^\intercal \tilde{\boldsymbol{x}}_j),$$

3 The vectors $\boldsymbol{q}$ can be consistently estimated through the degree vector $\boldsymbol{d} = \boldsymbol{A}\mathbf{1}_n$ as $\boldsymbol{q} \approx \boldsymbol{d}/\sqrt{\boldsymbol{d}^\intercal \mathbf{1}_n}$.

where $\boldsymbol{w}_\ell^\intercal$ denotes the ℓ-th row of $\boldsymbol{W}^\intercal$. Since all the $\boldsymbol{w}_\ell$ follow the same distribution $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_p)$, taking the expectation over $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_p)$ (conditionally on $\boldsymbol{X}$ and $\boldsymbol{A}$) yields the average Gram matrix $\bar{\boldsymbol{G}}$ defined with entries

$$\bar{\boldsymbol{G}}_{ij} = \mathop{\mathrm{E}}_{\boldsymbol{w}|\boldsymbol{X},\boldsymbol{A}} \left[\sigma(\boldsymbol{w}^\intercal \tilde{\boldsymbol{x}}_i)\sigma(\boldsymbol{w}^\intercal \tilde{\boldsymbol{x}}_j)\right]. \tag{6.4}$$

In particular, in the large n, p, d limit, it has been shown in Louart, Liao, and Couillet (2018) that the spectrum (and largest eigenvectors) of $\boldsymbol{G}$ are fully described by $\bar{\boldsymbol{G}}$. Specifically, the *resolvent* of $\boldsymbol{G}$ defined as,

$$\boldsymbol{Q}(z) = (\boldsymbol{G} + z\boldsymbol{I}_n)^{-1}, \tag{6.5}$$

for $z \in \mathbb{C}_+$ (with $\Im(z) > 0$), has a *deterministic equivalent* $\bar{\boldsymbol{Q}}(z)$ (conditionally on $\boldsymbol{X}$ and $\boldsymbol{A}$).

DEFINITION 6.1 (DETERMINISTIC EQUIVALENT). *A squared deterministic matrix $\bar{\boldsymbol{Q}}(z)$ is said to be a deterministic equivalent for the symmetric random matrix $\boldsymbol{Q}(z)$ if, for all deterministic matrix $\boldsymbol{M} \in \mathbb{R}^{n\times n}$ and vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$ of bounded norms (spectral and Euclidean, respectively), we have, as $n \to \infty$, with some probability or almost surely*

$$\frac{1}{n}\operatorname{Tr}\left(\boldsymbol{M}(\boldsymbol{Q}(z) - \bar{\boldsymbol{Q}}(z))\right) \to 0, \quad \boldsymbol{u}^\intercal(\boldsymbol{Q}(z) - \bar{\boldsymbol{Q}}(z))\boldsymbol{v} \to 0,$$

which we will simply express using the notation $\boldsymbol{Q}(z) \leftrightarrow \bar{\boldsymbol{Q}}(z)$.

REMARK 6.3.3. *The notion of deterministic equivalent of a resolvent matrix $\boldsymbol{Q}$ is related to the existence of a non-asymptotic deterministic matrix having, in probability or almost surely, the same scalar observations as the random observations through $\boldsymbol{Q}$.*

REMARK 6.3.4. *Such a deterministic equivalent is a standard object within random matrix theory (Hachem, Loubaton, and Najim, 2007) since it allows us to characterise the behaviour of the eigenvalues of $\boldsymbol{G}$ as well as its largest (often informative) eigenvectors. Specifically, the spectral measure of $\boldsymbol{G}$,*

$$\mu_n = \frac{1}{n}\sum_{i=1}^{n} \delta_{\lambda_i(\boldsymbol{G})},$$

*where $\lambda_i(\boldsymbol{G})$ denotes the i^{th} **eigenvalue** of $\boldsymbol{G}$, is related to $\boldsymbol{Q}(z)$ through the Stieltjes transform*

$$q_n(z) = \int (t-z)^{-1}\mu_n(dt) = \frac{1}{n}\operatorname{Tr}(\boldsymbol{Q}(-z)).$$

*While the **eigenvector** $\hat{\boldsymbol{u}}_i \in \mathbb{R}^n$ corresponding to **eigenvalue** $\lambda_i(\boldsymbol{G})$ is related to $\boldsymbol{Q}(z)$ through the Cauchy-integral $\hat{\boldsymbol{u}}_i\hat{\boldsymbol{u}}_i^\intercal = \frac{-1}{2\pi i}\oint_{\Gamma_i}\boldsymbol{Q}(-z)dz$ where Γ_i is a positively oriented complex contour surrounding $\lambda_i(\boldsymbol{G})$.*

A large dimensional growth rate assumption provides the existence of $\bar{\boldsymbol{Q}}(z)$.

ASSUMPTION 3 (GROWTH RATE). *As $n \to \infty$,*

1. *$p/n \to c \in (0,\infty)$ and $d/n \to r \in (0,\infty)$;*
2. *$\limsup_n \|\tilde{\boldsymbol{X}}\| < \infty$[4] and $|\mathcal{C}_a|/n \to c_a \in (0,1)$;*
3. *σ is λ_σ-Lipschitz continuous with $\lambda_\sigma > 0$ constant.*

Under Assumption 3, we have from (Louart, Liao, and Couillet, 2018)

$$\boldsymbol{Q}(z) \leftrightarrow \bar{\boldsymbol{Q}}(z) = \left(\frac{\bar{\boldsymbol{G}}}{1+\delta_g(z)} + z\boldsymbol{I}_n\right)^{-1}, \tag{6.6}$$

where $\delta_g(z)$ is the unique positive solution to the fixed point equation $\delta_g(z) = \frac{1}{n}\operatorname{Tr}(\bar{\boldsymbol{G}}\bar{\boldsymbol{Q}}(z))$.

From (6.6), to describe the behaviour of $\boldsymbol{G}$ one needs to address the non-linearity σ in the matrix $\bar{\boldsymbol{G}}$, this is achieved by approximating $\bar{\boldsymbol{G}}$ by a more tractable form in the large n limit. An additional regularity condition on σ is needed which we formulate now.

ASSUMPTION 4 (REGULARITY OF σ). *Suppose that σ is twice differentiable with $\limsup_{n,x\in\mathbb{R}}|\sigma''(x)| < \infty$. Furthermore, for $\xi \sim \mathcal{N}(0,1)$ suppose $\mathrm{E}[\sigma(\xi)] = 0$ and $\mathrm{E}[\sigma^2(\xi)] = 1$.*

Denote the quantity $b_\sigma = \mathrm{E}[\sigma'(\xi)]$. Under Assumptions 3-4, from (Fan and Wang, 2020, Lemma F.1), the average Gram matrix $\bar{\boldsymbol{G}}$ can be approximated by the $n \times n$ matrix $\tilde{\boldsymbol{G}} = b_\sigma^2\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^\intercal + (1-b_\sigma^2)\boldsymbol{I}_n$, since almost surely as $n \to \infty$

$$\frac{1}{n}\|\bar{\boldsymbol{G}} - \tilde{\boldsymbol{G}}\|_F^2 \to 0. \tag{6.7}$$

This approximation ensures in particular that $\bar{\boldsymbol{G}}$ and $\tilde{\boldsymbol{G}}$ share the same spectrum.

4 This assumption holds if additional assumptions on the node feature mean vector $\boldsymbol{\mu}$ and the graph parameters C_{ab}, which shall be provided Assumption 5, are placed.

REMARK 6.3.5. *The approximation of $\bar{\boldsymbol{G}}$ by $\tilde{\boldsymbol{G}}$ in (6.7) is valid when the matrix $\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^\intercal$ is of bounded spectral norm. This will be ensured in Assumption 5 where additional assumptions are placed on our model parameters $\boldsymbol{\mu}$ and C_{ab}. Furthermore, since the node features $\boldsymbol{x}_i$ follow a Gaussian mixture model (as per Assumption 1), if $\tilde{\boldsymbol{A}}$ has a bounded spectral norm , then the matrix $\tilde{\boldsymbol{X}}$ falls under the setting of (Fan and Wang, 2020) in which the relation in (6.7) holds.*

Since the behaviour of the average Gram matrix $\bar{\boldsymbol{G}}$ reduces to the analysis of the spectral behaviour of the matrix $\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^\intercal$ as per the approximation in (6.7), we will analyse $\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^\intercal$ for the remainder of Section 6.3.

6.3.2 *Spectral Behaviour of $\tilde{X}\tilde{X}^\intercal$*

We first need further controls on the quantities $\boldsymbol{\mu}$ and C_{ab} as we describe in the following assumption.

ASSUMPTION 5. *As $n \to \infty$,*

1. $\limsup_n \|\boldsymbol{\mu}\| < \infty$;
2. $C_{aa} = 1 + \frac{\eta_a}{\sqrt{n}}$ *for* $a \in \{1,2\}$ *and* $C_{ab} = 1$ *for* $a \neq b \in \{1,2\}$, *where* $\eta_a = (-1)^a \eta$ *and* $\limsup_n \eta < \infty$.

REMARK 6.3.6. *Assumption 5.2 defines a dense graph such that the clustering with spectral methods is not asymptotically trivial. Real-World graphs are usually sparse and fall within our theoretical analysis by considering the entry-wise multiplication of the adjacency matrix $\boldsymbol{A}$ by a random binary mask as is done by Zarrouk et al. (2020). Furthermore without loss of generality, we have specified $\eta_a = (-1)^a \eta$ for clarity of exposition of our theoretical results (Theorem 6.3.7), which can be generalised to different choices of the inter-class similarities (choices of η_a).*

Our main result (Theorem 6.3.7) provides a deterministic equivalent for the resolvent of $\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^\intercal$ defined as

$$\boldsymbol{Q}_{\tilde{\boldsymbol{X}}}(z) = \left(\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^\intercal + z\boldsymbol{I}_n\right)^{-1}. \tag{6.8}$$

THEOREM 6.3.7. *Define the quantities* $\gamma_f = \|\boldsymbol{\mu}\|^2$, $\gamma_g = q^2\eta$, $\nu = q^2(1-q^2)$ *and the matrices* $\boldsymbol{U} = \begin{bmatrix} \bar{\boldsymbol{y}} & \boldsymbol{\phi} \end{bmatrix} \in \mathbb{R}^{n\times 2}$,

$$\boldsymbol{B} = \begin{bmatrix} \gamma_g^2(\frac{\gamma_f}{c}+1) & \gamma_g(\frac{\gamma_f}{c}+1) \\ \gamma_g(\frac{\gamma_f}{c}+1) & \frac{\gamma_f}{c} \end{bmatrix}, \quad \boldsymbol{T} = \begin{bmatrix} 1 & 0 \\ 0 & \nu \end{bmatrix},$$

where $\bar{\boldsymbol{y}} = \frac{\boldsymbol{y}}{\sqrt{n}}$ *(with* $\boldsymbol{y} \in \{-1,1\}^n$ *the vector of labels) and* $\boldsymbol{\phi} = \frac{1}{\sqrt{n}}\boldsymbol{N}\bar{\boldsymbol{y}}$ *with* $\boldsymbol{N} \in \mathbb{R}^{n\times n}$ *a random matrix having random i.i.d. entries with zero mean and variance* ν. *Under Assumptions 1, 2, 3 and 5, the resolvent* $\boldsymbol{Q}_{\tilde{\boldsymbol{X}}}(z)$ *has a deterministic*[5] *equivalent defined as*

$$\bar{\boldsymbol{Q}}_{\tilde{\boldsymbol{X}}}(z) = \zeta \cdot (1+\delta_1)\left(\boldsymbol{I}_n - \zeta \boldsymbol{U}\left[\boldsymbol{B}^{-1} + \zeta \boldsymbol{T}\right]^{-1}\boldsymbol{U}^\intercal\right),$$

where $\zeta = \frac{1+\delta_2}{\nu + z(1+\delta_1)(1+\delta_2)}$ *and* (δ_1, δ_2) *is the unique couple solution of the fixed point equations system*

$$\delta_1 = \frac{1}{c}\frac{\nu(1+\delta_1)}{\nu + z(1+\delta_1)(1+\delta_2)}, \; \delta_2 = \frac{\nu(1+\delta_2)}{\nu + z(1+\delta_1)(1+\delta_2)}.$$

Sketch of proof. The proof starts by determining a random equivalent of the adjacency matrix $\boldsymbol{A}$. Since A_{ij} is Bernoulli distributed (see Assumption 2) with parameter $q^2(1+(-1)^{k_i}\delta_{k_i=k_j}\eta/\sqrt{n})$ with $k_i \in \{1,2\}$ the class of node i, we may write $A_{ij} = q^2 + q^2(-1)^{k_i}\delta_{k_i=k_j}\eta/\sqrt{n} + N_{ij}$ where N_{ij} is a zero mean random variable with variance $\nu + \mathcal{O}(n^{-\frac{1}{2}})$. Hence, $\|\tilde{\boldsymbol{A}} - (q^2\eta\bar{\boldsymbol{y}}\bar{\boldsymbol{y}}^\intercal + \frac{1}{\sqrt{n}}\boldsymbol{N})\| \to 0$ as $n \to \infty$. Finally, exploiting standard random matrix theory tools from (Hachem, Loubaton, and Najim, 2007; Louart and Couillet, 2018) provides the deterministic equivalent $\bar{\boldsymbol{Q}}_{\tilde{\boldsymbol{X}}}(z)$. □

In essence, Theorem 6.3.7 shows that the deterministic equivalent $\bar{\boldsymbol{Q}}_{\tilde{\boldsymbol{X}}}(z)$ is composed of two main terms: a diagonal matrix

$$\zeta \cdot (1+\delta_1)\boldsymbol{I}_n,$$

5 The matrix $\bar{\boldsymbol{Q}}_{\tilde{\boldsymbol{X}}}(z)$ is not deterministic since it depends on the random vector $\boldsymbol{\phi}$. However, since we are interested in evaluating quantities of the forms $\frac{1}{n}\operatorname{Tr}(\boldsymbol{M}\bar{\boldsymbol{Q}}_{\tilde{\boldsymbol{X}}}(z))$ or $\boldsymbol{u}^\intercal\bar{\boldsymbol{Q}}_{\tilde{\boldsymbol{X}}}(z)\boldsymbol{v}$ for $\boldsymbol{M}$, $\boldsymbol{u}$ and $\boldsymbol{v}$ independent of $\boldsymbol{\phi}$, $\bar{\boldsymbol{Q}}_{\tilde{\boldsymbol{X}}}(z)$ has a deterministic behaviour asymptotically as $n \to \infty$.

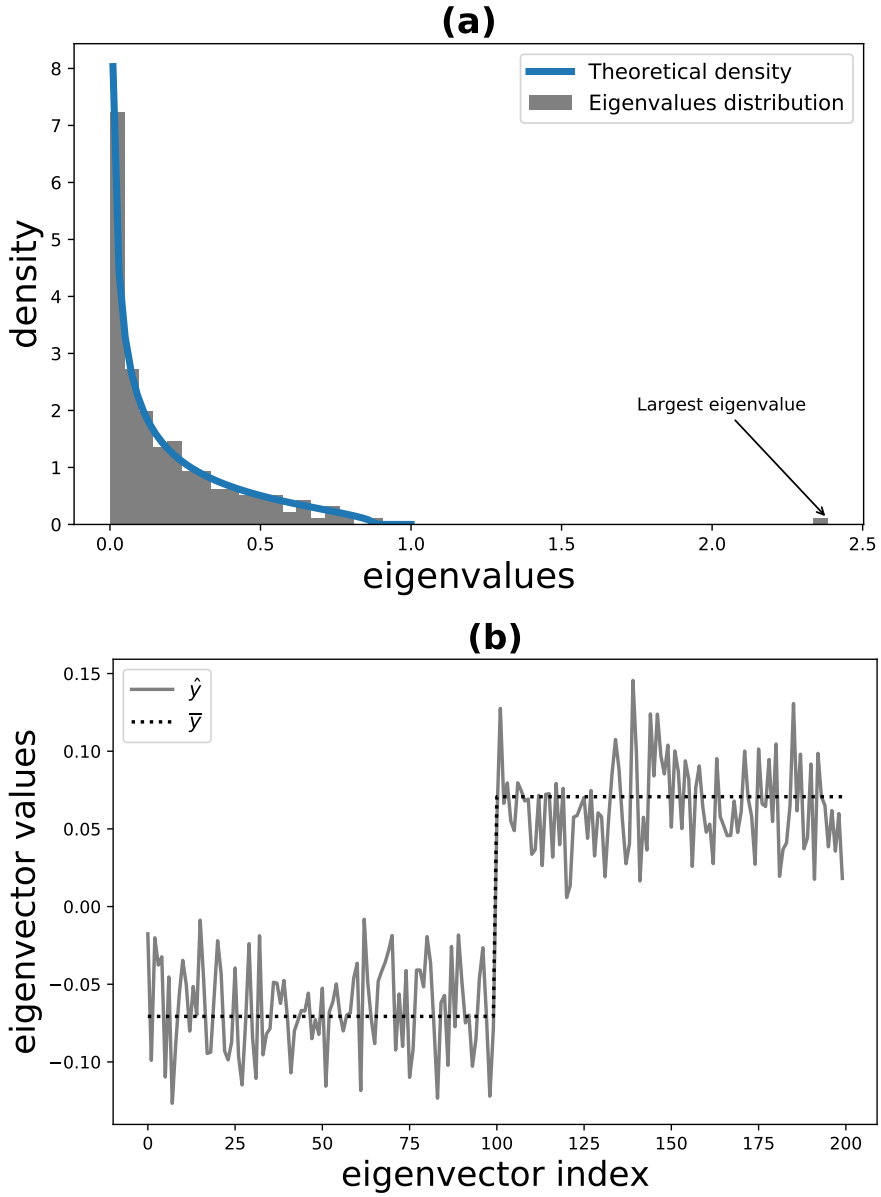

Figure 6.1: **(a)** Eigenvalues distribution of $\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^\intercal$ versus the theoretical density as per Theorem 6.3.7 (the theoretical density is obtained as $f(x) = \frac{1}{\pi}\lim_{\epsilon\to 0}\Im[q(x+i\epsilon)]$ where $q(z) = \frac{1}{n}\operatorname{Tr}(\bar{\boldsymbol{Q}}_{\tilde{\boldsymbol{X}}}(z))$). **(b)** Eigenvector of $\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^\intercal$ corresponding to its largest eigenvalue which correlates with $\bar{\boldsymbol{y}}$.

which describes the behaviour of the noise in the data model (both adjacency and node features), and an informative rank-2 matrix

$$\boldsymbol{U}\left[\boldsymbol{B}^{-1} + \zeta\boldsymbol{T}\right]^{-1}\boldsymbol{U}^\intercal,$$

which correlates with the vector of labels $\bar{\boldsymbol{y}}$ if the adjacency matrix and/or the node features are informative, i. e., values γ_g and γ_f, respectively, are sufficiently large.

An example with parameters $p = 1000$, $n = 200$, $q = 0.5$, $\eta = 4$ and $\boldsymbol{\mu} = [2, \mathbf{0}_{p-1}]^\intercal$ is illustrated in Figure 6.1. Figure 6.1(a) depicts the histogram of the eigenvalues of $\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^\intercal$ which converges to the limiting distribution described by Theorem 6.3.7, as well as (b) shows its dominant eigenvector which correlates with $\bar{\boldsymbol{y}}$.

Importantly, our analysis allows us to conclude that when the graph structure is completely noisy (i. e., $\eta = 0$), the dominant eigenvector of $\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^\intercal$ is no longer aligned with $\bar{\boldsymbol{y}}$ even if the node features are informative (i.e., γ_f large) as will be clarified in Corollary 6.3.8.

COROLLARY 6.3.8 (CASE $\eta = 0$). *Recall the notation and Assumptions of Theorem 6.3.7, for $\eta = 0$ (i.e., a non-informative graph structure), $\bar{\boldsymbol{Q}}_{\tilde{\boldsymbol{X}}}(z)$ takes the form*

$$\bar{\boldsymbol{Q}}_{\tilde{\boldsymbol{X}}}(z) = \zeta \cdot (1 + \delta_1) \left(\boldsymbol{I}_n - \frac{\zeta^2 \gamma_f}{c + \zeta \nu \gamma_f} \boldsymbol{\phi}\boldsymbol{\phi}^\intercal \right). \tag{6.9}$$

And, for $\hat{\boldsymbol{y}}$ the eigenvector of $\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^\intercal$ corresponding to its largest eigenvalue, $|\bar{\boldsymbol{y}}^\intercal \hat{\boldsymbol{y}}|^2 \xrightarrow[n\to\infty]{} 0$.

Sketch of proof. Expression (6.9) follows from Theorem 6.3.7 by simply taking the limit as $\eta \to 0$. The second part of the Corollary is obtained by computing

$$|\bar{\boldsymbol{y}}^\intercal \hat{\boldsymbol{y}}|^2 = \frac{-1}{2i\pi} \oint_\Gamma \bar{\boldsymbol{y}}^\intercal \boldsymbol{Q}_{\tilde{\boldsymbol{X}}}(-z) \bar{\boldsymbol{y}} dz,$$

where Γ is a small positively oriented complex contour surrounding the largest eigenvalue of $\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^\intercal$.

Hence, using $\bar{\boldsymbol{Q}}_{\tilde{\boldsymbol{X}}}(z)$ as a proxy allows us to state

$$|\bar{\boldsymbol{y}}^\intercal \hat{\boldsymbol{y}}|^2 + \frac{1}{2i\pi} \oint_\Gamma \bar{\boldsymbol{y}}^\intercal \bar{\boldsymbol{Q}}_{\tilde{\boldsymbol{X}}}(-z) \bar{\boldsymbol{y}} dz \to 0$$

almost surely as $n \to \infty$.

The final result is obtained by showing that $\bar{\boldsymbol{y}}^\intercal \boldsymbol{\phi}\boldsymbol{\phi}^\intercal \bar{\boldsymbol{y}}$ concentrates around its expectation with

$$\mathrm{E}\left[\bar{\boldsymbol{y}}^\intercal \boldsymbol{\phi}\boldsymbol{\phi}^\intercal \bar{\boldsymbol{y}}\right] = \frac{1}{n}\mathrm{Var}[\bar{\boldsymbol{y}}^\intercal \boldsymbol{N} \bar{\boldsymbol{y}}] = \frac{\nu}{n} \to 0,$$

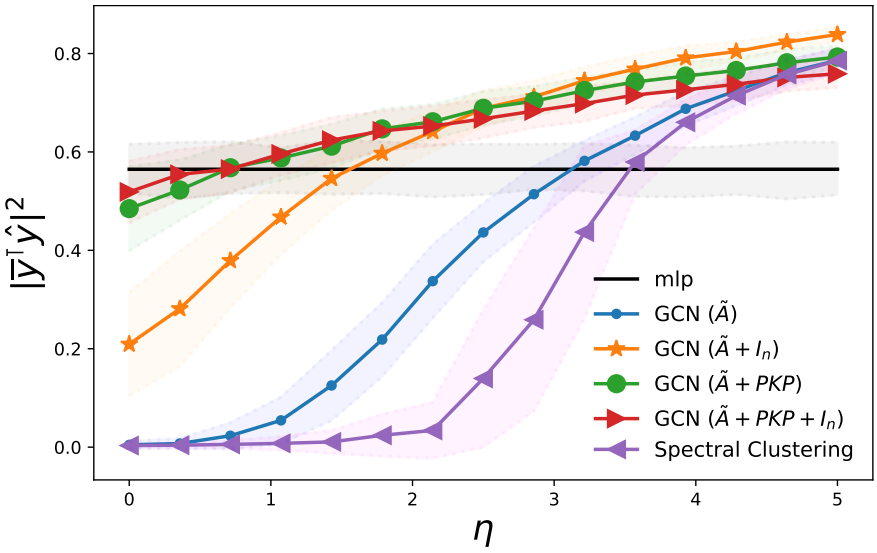

Figure 6.2: Alignment between the largest eigenvector of $\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^\intercal$ and the labels vector $\bar{\boldsymbol{y}}$ for different added node feature kernel message passing strategies in terms of η. The kernel matrix has entries $K_{ij} = \boldsymbol{x}_i^\intercal \boldsymbol{x}_j$, mean and std computed over 100 runs. ***The GCN with message passing operator $\tilde{\boldsymbol{A}} + PKP$ outperforms other models when the graph structure is noisy (i.e., low values of η).***

and by evaluating

$$\frac{1}{2i\pi} \oint_\Gamma \zeta(-z)(1 + \delta_1(-z))dz = 0.$$

□

The main conclusion from Corollary 6.3.8 is that when the graph structure is completely random (when $\eta = 0$, $\|\tilde{\boldsymbol{A}} - \frac{1}{\sqrt{n}}\boldsymbol{N}\| \to 0$), the largest eigenvector of $\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^\intercal$ (which is intuitively supposed to be informative) does not correlate with the labels vector $\bar{\boldsymbol{y}}$ independently of the information contained in the node features. To overcome this issue, we propose in the now following subsection 6.3.3 to utilise node feature kernels to ensure the preservation of node feature information.

6.3.3 *Message Passing through Node Feature Kernels*

As discussed in subsection 6.3.2, when the graph structure (in the extreme case) is completely random, the *random* GCN model fails to extract information from the node features. To make the message passing informative and thereby to robustify the GCN, we propose to consider the operator $\tilde{\boldsymbol{A}} + \bar{\boldsymbol{K}}$ instead of $\tilde{\boldsymbol{A}}$, where $\bar{\boldsymbol{K}}$ is a kernel matrix computed on the node features $\boldsymbol{X}$. Indeed, let $\boldsymbol{K}$ be a matrix with entries $K_{ij} = \kappa(\boldsymbol{x}_i^\intercal \boldsymbol{x}_j)$ for some smooth function $\kappa : \mathbb{R} \to \mathbb{R}$. Relying

on (El Karoui, 2010), the kernel matrix $\boldsymbol{K}$ can be approximated in spectral norm asymptotically as $n \to \infty$ by

$$\tilde{\boldsymbol{K}} = \kappa(0)\mathbf{1}_n\mathbf{1}_n^\intercal + \kappa'(0)\left(\frac{\gamma_f}{c}\bar{\boldsymbol{y}}\bar{\boldsymbol{y}}^\intercal + \boldsymbol{Z}\boldsymbol{Z}^\intercal\right) + \boldsymbol{\Delta}, \tag{6.10}$$

where $\boldsymbol{\Delta} = \frac{\kappa''(0)}{2p}\mathbf{1}_n\mathbf{1}_n^\intercal + (\kappa(1) - \kappa(0) - \frac{\gamma_f}{c}\kappa'(0))\boldsymbol{I}_n$. Hence, considering the matrix $\tilde{\boldsymbol{A}} + \boldsymbol{P}\boldsymbol{K}\boldsymbol{P}$, where $\boldsymbol{P} = \boldsymbol{I}_n - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^\intercal$ (the centring matrix), maintains the informative nature of the message passing step (through the term $\frac{\gamma_f}{c}\bar{\boldsymbol{y}}\bar{\boldsymbol{y}}^\intercal$) even in the case where the operator $\tilde{\boldsymbol{A}}$ is not informative. Intuitively, the addition of the node feature kernel can be interpreted as considering both the originally recorded graph and a node feature similarity graph in the message passing architecture. This addition gives GNNs the necessary expressive ability to preserve information present in the node features, which is lost in the case of uninformative or noisy graph structures.

Figure 6.2 shows the performance of different message passing strategies (compared to random MLP and spectral clustering; involving only node features or adjacency matrix, respectively) on an example with parameters $p = 500$, $n = 250$, $q = 0.4$, $\boldsymbol{\mu} = [1.7, \mathbf{0}_{p-1}]^\intercal$, which confirms the effectiveness of introducing a node feature kernel in the regime where the graph similarity is noisy (i.e., low values of η), a property which is also validated for practical GCNs as we will discuss in Section 6.4.

6.4 EXPERIMENTS AND DISCUSSION

In order to validate our theoretical findings in real-world scenarios, we experiment on the node classification task using GCNs on perturbed data. In subsection 6.4.2, we begin by justifying the use of the *random* GCN in the theoretical analysis. Then, we discuss results from experiments on both synthetic and real-world graphs involving a perturbation scheme on their edges. We show that the observed phenomena extend to deeper GCN models under structural perturbation. Importantly, we demonstrate that, under this setting, our proposed method is comparable with state-of-the-art GNN models also placing a particular emphasis on the node features and can be further improved when combined with other techniques. Finally, we examine the robustness of our proposed method under the setting where node features are also perturbed.

6.4.1 *Datasets and Implementation Details*

We work on synthetic SBM graphs aligned with Assumption 5, with the intra-community link probability being $q^2(1 + \frac{\eta}{\sqrt{n}})$ and the inter-community link probability being q^2. We vary the parameter η to generate SBM graphs with different types of community structure and keep other parameters fixed as $q = 0.5$, $p = 2500$, $n = 1600$, $\mu = (2, 0, \ldots, 0)$.

We furthermore work with six real-world datasets which often serve as node classification benchmarks. These are the three well-studied citation networks of Cora, CiteSeer and PubMed (Sen et al., 2008), an extended version of Cora (Bölcskei et al., 2019), called CoraFull, an Amazon co-purchase graph of Photo and a Co-author network from the authors of Computer Science (CS) (Shchur et al., 2018).

We follow the semi-supervised node classification setting proposed by Yang, Cohen, and Salakhutdinov (2016), i.e., we use their train/valid/test split for Cora/CiteSeer/PubMed, and we randomly sample 20 nodes from each class as training set, 500 nodes as validation set and another 1000 nodes as test set for CoraFull/Photo/CS. The statistics of these aformentioned datasets can be found in Table 2.1.

In line with our theoretical analysis, the main GCN architecture on which we experimented in this paper is a single-layer GCN (one iteration of message passing) stacked with a Multi-Layer Perception (MLP). The objective of the experiments is to validate our theoretical hypotheses and experiment with the robustness of GCN models under graph structure perturbation.

We also study empirically to what extent the validated hypotheses extrapolate to scenarios where deeper GCN architectures with multiple layers of graph propagation are used and/or node features are also perturbed. In comparison to the state-of-the-art models, in particular to those which also place particular emphasis on the node features, we demonstrate that our proposed method has superior or comparable performance and can be further improved when combined with other techniques.

All the experiments are performed using the Adam optimiser (Kingma and Ba, 2015) and the same set of hyper-parameters, with learning rate being 1e-2, number of epochs being 200 and hidden feature dimension being 128. We repeat each experiment 10 times and report the resulting means and standard deviations to accurately report the impact of random initialisation. Our implementation is built upon the open source library PyTorch Geometric (PyG) under MIT license (Fey and Lenssen, 2019). The experiments are run on a Intel(R) Xeon(R) W-2123

processor with 64GB ram and a NVIDIA GeForce RTX 2080Ti GPU with 12GB ram.

PERTURBATION SCHEME The studied perturbation scheme involves both *edge-deletion* noise, where a certain amount of existing edges are randomly sampled and removed from original graph, and *edge-insertion* noise, where we add a certain amount of connections sampled from the non-existing edges in the original graph. We consider scenarios where edges are removed or added or both. The ratio of edges changed, i. e., the perturbation ratio, is denoted by α for *edge-deletion* and β for *edge-insertion*.

A node feature kernel matrix is added to study its impact in practice, as shown in the following equation,

$$\boldsymbol{X}^{(i+1)} = \sigma\left((\epsilon\hat{\boldsymbol{A}} + (1-\epsilon)\mathcal{N}(\boldsymbol{K}))\boldsymbol{X}^{(i)}\boldsymbol{W}\right), \tag{6.11}$$

where $\hat{\boldsymbol{A}}$ is the GCN message passing operator of the perturbed graph, $\mathcal{N}(\boldsymbol{K}) = \boldsymbol{D}_K^{-1/2}\boldsymbol{K}\boldsymbol{D}_K^{-1/2}$ is the normalised kernel matrix built from node features ($\boldsymbol{D}_K = \text{diag}(\boldsymbol{K}\mathbf{1}_n)$). We are degree normalising the kernel to match the graph representation of the GCN message passing operator.

NODE FEATURE KERNELS As stated in Section 6.3.3, we use the kernel to introduce information from the node features to the message passing structure. The choices of qualified smooth kernel functions are many. In our experiment, we perform a proof of concept using the simple linear kernel, defined as the inner product between node features $K_{ij} = \boldsymbol{x}_i^\intercal \boldsymbol{x}_j$.

KERNEL SPARSIFICATION Using the full kernel matrix, where the kernel value is recorded between every pair of nodes, is both computationally costly and may incorporate redundant information. Therefore, we adopt a sparsification method using the adjacency matrix of the graph, with which (6.11) can be rewritten as,

$$\boldsymbol{X}^{(i+1)} = \sigma\left((\epsilon\hat{\boldsymbol{A}} + (1-\epsilon)\mathcal{N}(\boldsymbol{K}\circ\hat{\boldsymbol{A}}))\boldsymbol{X}^{(i)}\boldsymbol{W}\right), \tag{6.12}$$

where $\circ$ denotes Hadamard product. A consequence of this sparsification method is that the added computational cost stemming from the consideration of the node feature kernel is linear in the number of edges in the graph $|\mathbb{E}|$, where $\mathbb{E}$ denotes the graph's edge set, and the node feature dimension p, i.e., of order $\mathcal{O}(|\mathbb{E}|p)$. Our initial experiments sparsifying the kernel matrix by using a threshold below which all entries are set to zero resulted in worse performance

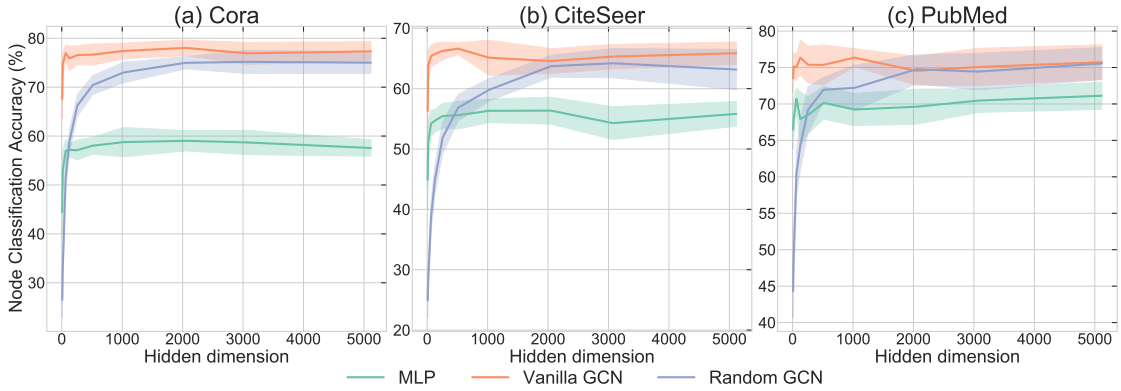

Figure 6.3: Performance change over the embedding dimension with different models: *random* GCN, vanilla GCN and MLP.

and introduced the threshold as an extra hyperparameter. Therefore, we chose to only pursue the sparsification scheme in (6.12). This preliminary observation could be a result of the particular node features that are recorded in the datasets.

6.4.2 *Experiment Analysis*

6.4.2.1 *Asymptotic Analysis of Random GCN*

To validate the practical applicability of the theoretical analysis in Section 6.3, we study the asymptotic behaviour of the *random* GCN, the vanilla GCN and a MLP baseline when the hidden dimension of node features grows. In Figure 6.3, we observe that with increasing hidden dimension, the performance of both the vanilla GCN and MLP remains stable, while the performance of *random* GCN converges to vanilla GCN's accuracy. Between hidden dimensions of 2000 and 3000 the performance of *random* GCN starts to match that of vanilla GCN. Hence, we have given an empirical indication of the conditions under which our theoretical model, the *random* GCN, and the vanilla GCN are equivalent.

6.4.2.2 *Robustness to Structural Noise: Synthetic SBM*

We first test the performance of the proposed model on synthetic SBM graphs. Three types of SBM graph are considered, which are distinguished by the parameter η. $\eta = 0$, $\eta = 4$ and $\eta = -4$ correspond to the cases where the synthetic graph has no, a *homophilic* and a *heterophilic* community structure, respectively. We experiment on perturbation scenarios with different *egde-deletion* and *edge-insertion* ratios. For each scenario, we record the performance of the vanilla GCN

Table 6.1: Performance of GCN with node-feature kernel under perturbation on synthetic SBM graphs. The best results are set to **bold** if their range of one standard deviation does not overlap with the standard deviation of their counterpart.

		SBM($q = 0.5, \eta = 0$)		SBM($q = 0.5, \eta = 4$)		SBM($q = 0.5, \eta = -4$)	
	(α, β)	GCN	GCN-k	GCN	GCN-k	GCN	GCN-k
Deletion	(0.0, 0.0)	50.53 ± 0.49	$\mathbf{66.36 \pm 0.81}$	$\mathbf{64.42 \pm 0.43}$	62.26 ± 1.04	$\mathbf{63.20 \pm 0.94}$	61.03 ± 1.08
	(0.2, 0.0)	51.03 ± 0.56	$\mathbf{65.44 \pm 1.07}$	58.63 ± 0.68	$\mathbf{71.57 \pm 1.42}$	$\mathbf{60.89 \pm 0.83}$	54.91 ± 1.00
	(0.5, 0.0)	49.29 ± 0.59	$\mathbf{64.14 \pm 1.01}$	60.76 ± 1.29	$\mathbf{68.80 \pm 2.04}$	58.41 ± 1.11	59.51 ± 2.47
Insertion	(0.0, 0.5)	50.57 ± 0.75	$\mathbf{68.57 \pm 1.25}$	60.49 ± 0.40	$\mathbf{68.20 \pm 1.38}$	58.82 ± 1.16	$\mathbf{63.54 \pm 0.97}$
	(0.0, 1.0)	49.19 ± 0.47	$\mathbf{59.31 \pm 0.58}$	53.67 ± 1.11	$\mathbf{66.57 \pm 1.73}$	54.87 ± 0.53	$\mathbf{60.84 \pm 0.75}$
Delet.+Insert.	(0.5, 0.5)	49.26 ± 0.59	$\mathbf{68.84 \pm 0.86}$	50.50 ± 0.37	$\mathbf{63.36 \pm 1.67}$	50.94 ± 0.86	$\mathbf{63.02 \pm 0.91}$
	(0.5, 1.0)	49.84 ± 0.69	$\mathbf{65.49 \pm 1.22}$	48.34 ± 0.22	$\mathbf{60.16 \pm 1.21}$	49.23 ± 0.45	$\mathbf{59.64 \pm 1.33}$

Table 6.2: Performance of MLP using only node features on six real-world datasets.

	Cora	CiteSeer	PubMed	CoraFull	Photo	CS
MLP	57.88 ± 0.58	57.23 ± 0.73	72.94 ± 0.32	41.56 ± 1.34	76.06 ± 0.95	87.97 ± 0.67

as well as its performance after adding the node-feature kernel, denoted by an appendage "-k"[6] in Table 6.1.

When a graph has no community structure, structural perturbation has little impact and adding node-feature information boosts the performance. When the graphs are *homophilic* and with a clear community structure, the impact from graph-structural noise become more visible. Adding a node-feature kernel significantly improves the model's robustness against *edge-deletion* and *edge-insertion* noise and their mix. The same conclusion can be drawn on *heterophilic* graphs perturbed by *edge-insertion* noise.

6.4.2.3 *Robustness to Structural Noise: Real-World datasets*

In this set of experiments, we study the change of model performance under the same perturbation setting on real-world datasets. Table 6.3 shows the results over different perturbation scenarios.

Naturally the performance decreases gradually when edges are removed or added, as we can see from each column. *Edge-insertion* noise seems to have a larger impact on the performance than the *edge-deletion* noise on these real-world datasets. When the noise ratio is large, the performance of vanilla GCN starts to be outperformed by the MLP that utilise only node feature information, whose results are shown in Table 6.2. But this influence can be largely compensated by adding the node feature kernel. Unlike results from synthetic SBM graphs, for *edge-deletion* noise, the addition of the kernel on real-world datasets seems to have almost no impact.

6.4.2.4 *Deeper GCN architecture and Benchmark Models*

The previous experiments are based on a single-layer GCN model. In practice, the best-performing GCN models on these datasets often contain several message-passing layers and therefore, we want to observe whether our theoretical results can be extrapolated to the multi-layer case. We build a 4-layer GCN model and repeat our experiments on the Cora and CS datasets.

The results are shown in Table 6.4. Results of the remaining datasets can be found in the Appendix B.4.2. Moreover, we also consider varying the weight

6 If not specified, the weight coefficient ϵ of the added kernel in graph propagation is set to 0.5.

Table 6.3: Performance of GCN with and without node-feature kernel under perturbation on six real-world datasets. The format follows Table 6.1. We add ↓ next to a value when it is smaller that of the MLP in Table 6.2, which indicates that the corresponding model performs worse than the MLP using only node features.

		Cora		CiteSeer		PubMed	
	(α, β)	GCN	GCN-k	GCN	GCN-k	GCN	GCN-k
Deletion	(0.0, 0.0)	**79.37 ± 0.65**	76.94 ± 0.35	67.45 ± 0.82	68.08 ± 0.91	76.04 ± 0.67	74.68 ± 0.76
	(0.2, 0.0)	76.15 ± 0.81	74.83 ± 1.24	66.79 ± 0.57	66.94 ± 0.82	**75.82 ± 0.99**	74.28 ± 0.39
	(0.5, 0.0)	72.49 ± 0.50	71.18 ± 1.00	63.53 ± 0.75	64.84 ± 1.14	73.95 ± 0.64	73.25 ± 0.75
Insertion	(0.0, 0.5)	68.57 ± 0.73	**73.10 ± 1.10**	59.85 ± 0.89	**66.11 ± 1.34**	64.18 ± 0.67 ↓	**72.38 ± 0.79** ↓
	(0.0, 1.0)	64.14 ± 1.02	**73.36 ± 0.98**	55.39 ± 0.93 ↓	**64.94 ± 0.77**	60.56 ± 0.80 ↓	**71.31 ± 0.51** ↓
Delet.+Insert.	(0.5, 0.5)	54.98 ± 1.13 ↓	**66.46 ± 1.03**	52.84 ± 0.68 ↓	**59.03 ± 1.04**	62.62 ± 0.72 ↓	**70.32 ± 0.82** ↓
	(0.5, 1.0)	48.09 ± 0.88 ↓	**62.52 ± 0.59**	42.28 ± 1.07 ↓	**58.07 ± 1.34**	53.25 ± 1.57 ↓	**69.65 ± 0.60** ↓

		CoraFull		Photo		CS	
	(α, β)	GCN	GCN-k	GCN	GCN-k	GCN	GCN-k
Deletion	(0.0, 0.0)	57.21 ± 0.84	56.88 ± 0.48	90.94 ± 0.49	90.09 ± 0.65	92.89 ± 0.41	92.63 ± 0.31
	(0.2, 0.0)	**57.25 ± 0.67**	55.56 ± 0.69	91.87 ± 0.40	92.19 ± 0.45	90.58 ± 0.48	90.89 ± 0.48
	(0.5, 0.0)	53.90 ± 0.70	54.62 ± 0.87	**91.10 ± 0.40**	87.97 ± 0.54	89.75 ± 0.60	**91.27 ± 0.67**
Insertion	(0.0, 0.5)	48.11 ± 0.89	**51.79 ± 0.65**	82.79 ± 1.43	84.18 ± 1.27	87.16 ± 0.65 ↓	**90.81 ± 0.70**
	(0.0, 1.0)	41.76 ± 1.03	**51.91 ± 1.00**	72.70 ± 6.40 ↓	**79.58 ± 1.80**	80.34 ± 0.80 ↓	**90.61 ± 0.37**
Delet.+Insert.	(0.5, 0.5)	34.70 ± 0.47 ↓	**46.50 ± 0.61**	69.70 ± 3.70 ↓	**74.65 ± 2.36** ↓	73.75 ± 0.98 ↓	**87.28 ± 0.72** ↓
	(0.5, 1.0)	27.50 ± 1.04 ↓	**43.04 ± 0.77**	61.13 ± 2.49 ↓	63.73 ± 5.04 ↓	66.26 ± 0.95 ↓	**87.51 ± 0.58** ↓

Table 6.4: Performance of GCN with and without node-feature kernel under perturbation on deep GCN models, compared with jump knowledge and GCNII. The format follows Table 6.1, where in addition we underline the second best result.

		Cora					
	(α, β)	GCN	GCN-k ($\epsilon = 0.5$)	GCN-k ($\epsilon = 0.2$)	GCN-jk	GCNII	GCN-k-jk
Deletion	(0.0, 0.0)	79.05 ± 1.36	79.67 ± 1.17	79.27 ± 1.50	80.39 ± 1.13	79.62 ± 1.24	79.74 ± 0.75
	(0.2, 0.0)	76.45 ± 1.39	77.73 ± 0.79	77.54 ± 1.63	77.84 ± 0.99	75.98 ± 0.87	77.62 ± 0.57
	(0.5, 0.0)	74.20 ± 1.15	72.22 ± 1.25	72.74 ± 1.23	74.97 ± 0.71	73.87 ± 0.91	74.16 ± 0.77
Insertion	(0.0, 0.5)	64.91 ± 1.87	71.18 ± 1.25	$\underline{73.20 \pm 0.84}$	73.05 ± 0.54	71.98 ± 1.39	$\mathbf{76.08 \pm 0.84}$
	(0.0, 1.0)	53.48 ± 3.49	63.20 ± 1.78	$\underline{71.24 \pm 1.04}$	65.86 ± 0.62	68.26 ± 1.15	$\mathbf{72.94 \pm 0.67}$
Delet.+Insert.	(0.5, 0.5)	47.40 ± 1.73	57.34 ± 1.53	60.77 ± 1.57	57.94 ± 0.92	$\underline{62.54 \pm 1.55}$	$\mathbf{67.11 \pm 0.93}$
	(0.5, 1.0)	35.27 ± 3.48	48.82 ± 1.03	$\underline{60.12 \pm 1.29}$	48.85 ± 1.06	59.82 ± 1.59	$\mathbf{62.67 \pm 1.15}$
		CS					
	(α, β)	GCN	GCN-k ($\epsilon = 0.5$)	GCN-k ($\epsilon = 0.2$)	GCN-jk	GCNII	GCN-k-jk
Deletion	(0.0, 0.0)	88.44 ± 0.84	89.81 ± 0.52	91.64 ± 0.39	90.19 ± 0.59	$\mathbf{92.13 \pm 0.39}$	$\underline{91.73 \pm 0.26}$
	(0.2, 0.0)	89.19 ± 0.57	88.41 ± 0.53	$\underline{91.68 \pm 0.55}$	91.04 ± 0.65	91.56 ± 0.53	$\mathbf{91.89 \pm 0.77}$
	(0.5, 0.0)	86.68 ± 0.57	86.17 ± 1.06	88.91 ± 0.62	88.44 ± 0.69	$\underline{90.01 \pm 0.69}$	$\mathbf{91.43 \pm 0.60}$
Insertion	(0.0, 0.5)	70.94 ± 2.59	84.36 ± 1.19	88.84 ± 0.57	87.37 ± 0.66	$\underline{90.36 \pm 0.58}$	$\mathbf{92.66 \pm 0.49}$
	(0.0, 1.0)	35.84 ± 6.91	81.06 ± 3.94	88.27 ± 0.92	81.70 ± 0.63	$\underline{89.33 \pm 1.02}$	$\mathbf{91.42 \pm 0.48}$
Delet.+Insert.	(0.5, 0.5)	45.08 ± 4.82	76.27 ± 1.08	82.23 ± 1.08	73.08 ± 1.07	$\mathbf{88.66 \pm 0.70}$	$\underline{87.53 \pm 0.85}$
	(0.5, 1.0)	18.16 ± 3.86	53.12 ± 6.21	80.80 ± 0.99	63.84 ± 0.95	$\mathbf{88.77 \pm 0.89}$	$\underline{87.89 \pm 0.37}$

Table 6.5: Performance of GCN with and without node-feature kernel under both graph-structural and node-feature perturbation on PubMed dataset. The format follows Table 6.1.

		$\gamma = 0.5$		$\gamma = 1.0$		$\gamma = 5.0$	
	(α, β)	GCN	GCN-k	GCN	GCN-k	GCN	GCN-k
Deletion	(0.0, 0.0)	72.78 ± 1.62	73.88 ± 2.04	68.40 ± 2.95	66.82 ± 1.81	42.43 ± 2.95	39.14 ± 2.60
	(0.2, 0.0)	70.11 ± 2.77	71.95 ± 1.70	68.28 ± 2.77	67.19 ± 1.62	40.23 ± 3.27	40.04 ± 2.43
	(0.5, 0.0)	71.61 ± 1.31	70.33 ± 2.38	63.81 ± 3.43	64.93 ± 2.14	40.41 ± 1.92	38.11 ± 1.99
Insertion	(0.0, 0.5)	62.93 ± 2.23	$\mathbf{70.52 \pm 2.08}$	59.85 ± 1.66	$\mathbf{64.77 \pm 2.37}$	40.00 ± 1.20	38.92 ± 1.71
	(0.0, 1.0)	58.23 ± 1.79	$\mathbf{68.14 \pm 1.67}$	55.37 ± 3.33	$\mathbf{63.35 \pm 2.30}$	38.25 ± 1.93	38.72 ± 1.93

coefficient of the added node-feature kernel in graph propagation and compare with two models specifically proposed for deep GNN architectures: Jumping Knowledge (JK, Xu et al. (2018)) and a GCN with residual connections and an identity mapping (GCNII, Chen et al. (2020)). Although not designed to tackle the graph-structural perturbation problem that we study in this work, the two models both utilise node feature information, which makes them reasonable competitors to our proposed method.

As we can see from Table 6.4, there is no fundamental change in the trends observed for the single-layer model. Adding node-feature kernel helps robustify model performance against perturbation when edges are inserted and/or removed. The improvement is even more significant when the weight coefficient ϵ on the kernel in graph propagation is increased. Additionally, our proposed method, with a high weight on the kernel, is comparable to GCNII model and in general performs better than the JK model. However, since JK can easily be combined with our method, the node-feature kernel and JK architecture yields the best-performing model as can be seen in the rightmost column of Table 6.4.

6.4.2.5 *Node feature noise*

We now study how node feature noise interacts with our proposed model. We perform the experiments with the same perturbation setting on the PubMed dataset. But add Gaussian noise $\mathcal{N}(\mathbf{0}, \gamma \operatorname{diag}(\sigma_i))$ to the node features where σ_i is the estimated variance of the i^{th} feature and γ is a scaling parameter. The results are recorded in Table 6.5. We observe that only when the node feature noise is five times larger than the graph structure noise ($\gamma = 5$), the addition of the node feature kernel stops to benefit the model performance. Our previous findings still hold if the node feature noise is reasonably small ($\gamma \leq 1$).

6.5 CHAPTER CONCLUSION

In this work, we have introduced , for the first time to the best of our knowledge, the *random* GCN, which we analysed theoretically using random matrix theory. Our analysis shed light on the way in which the GCN processes the different sources of information present in attributed graphs. In particular, this leads us to conclude that

> Perturbations of the graph structure strongly influence the performance of the GCN regardless of the information contained in the node features.

For SBM graphs the presence of community structure (and the degree to which this structure is present) is required (beneficial) for a message passing scheme which leads to eigenvectors of the message passing operator's Gram matrix that align with the node labels. These conclusions were confirmed in multiple experiments with the standard GCN architecture on synthetic and real-world datasets.

As expected from our theoretical analysis, we empirically observed graph structure perturbations to have a strong negative impact on the performance of standard GNN architectures, which drops rapidly below the performance of a MLP using only node features. And on both synthetic and real-world data we observe *the introduction of a node feature kernel to the GCN's message passing scheme to significantly improve the performance of the GCN in the presence of a noisy graph structure*. We believe that the introduced *random* GCN model is likely to be a suitable analysis framework for further GNN models, which combined with random matrix theory can allow us to gain further insight into the inference obtained from these models.

Part III

CONCLUSION

7

CONCLUSION

In this dissertation, we navigate the topic of graph representation learning from graph kernels to graph neural networks. Both methods are migrated from approaches in the classic machine learning domain, *kernel methods* and *neural networks*, respectively, to graph-structured data. They also inherit the advantages and drawbacks of their parents, with one being theoretically motivated, rigorous, effective with limited data but not very scalable. The other is end-to-end, powerful, scaling well to big data but lacking theoretical justification. With this in mind, our research focuses on tackling the drawbacks and developing more effective, efficient and robust methods for graph representation learning.

In the following sections, we summarise our main contributions reported in detail in the previous chapters and discuss future research directions that were not explored at the time of writing.

7.1 SUMMARY OF CONTRIBUTIONS

The contributions of the thesis can be summarised as follow:

VALID OPTIMAL ASSIGNMENT KERNEL We propose in Chapter 3 a novel kernel that finds an optimal assignment between the graph substructures and is guaranteed to be valid. We first represent the graphs as a set of node embeddings. These node embeddings are generated as the eigenvectors of the graph adjacency matrix, where structural information is naturally encoded. Assuming that all node embeddings from all graphs of the dataset reside in the same vector space, we then recursively partition this space by clustering methods and create a hierarchy corresponding to these hierarchical partitions, which includes a tree and a weight function defined on each cluster centroid that is associated with the compactness of the cluster. This hierarchy enables us to construct a histogram of each graph whose entry is the inner vertex of the tree. A histogram-intersection kernel is then applied to compare the similarity between the two graphs and assures the validity of the proposed kernel constructed following the procedure mentioned above. Our framework is flexible as we can switch between the node embedding methods as well as clustering algorithms. We

demonstrate that the proposed kernel is a competitive method to compare sets of vectors on graph classification and text categorisation tasks. We also show that the feature map of the vertices associated with the proposed kernel can be seen as a new and more expressive node embedding method compared with the original node embedding by experiments on the link prediction task.

PREDICTING TOPOLOGY EVOLUTION OF TEMPORAL GRAPHS Since its birth, GNNs have soon become the dominating force in the field of graph representation learning. In Chapter 4, we demonstrate the power of GNNs over traditional methods on a long-existing problem of predicting the topology evolution of dynamic graphs. We propose an *encoder-decoder* framework, as one may frequently see in classic machine learning. The ability of GNNs of transforming attributed graphs into vector representations makes them the perfect fit for the role of the *encoder* in this framework. The transformed embeddings are processed by an LSTM and output a prediction of the future embedding at the next time step. A *decoder* reads this embedding and generates the prediction of graph topology with the help of an autoregressive graph generation model. The proposed framework is flexible with the choice of GNNs, graph generation models, and the RNN that encodes temporal information, e. g., adding attention mechanism. These variants are not fully explored in our work. However, the primary choice, as we showed in Chapter 4, has already demonstrated a superior advantage over traditional methods such as random graph models in predicting the topology of future graphs that is the most similar to the reality (measured by the graph kernel), by experiments on both synthetic and real-world datasets.

ANALYSING THE ROLE OF THE UPDATE STEP IN MPNN We observe that the MPNN, the primary class of GNN, has two disjoint parts: one is the *Aggregation* step which utilises the graph structure information, and the other is the *Update* step which is graph-agnostic and only concerns node feature information. In Chapter 5, we study the role of the *Update* step in the MPNN framework. To this end, we monitor the performance change of the MPNN model while we sparsify the weight $\boldsymbol{W}$ of the *Update* step to different levels, even to an extreme case where the $\boldsymbol{W}$ corresponding to the linear transform is completely omitted. We denote the sparsified model as *Expander* GNN and the extreme model as *Activation-Only* GNN. Extensive experiments on graph classification, graph regression and node

classification over datasets across domains from biology to social networks have led us to two conclusions:

1. the *Aggregation* step plays a more critical role in GNN learning than the *Update* step as the *Update* step can be sparsified to an arbitrary level without undermining the model performance,
2. the non-linear activation function is a valuable component in MPNNs as removing the activation is consistently outperformed by keeping this component.

TOWARDS A MORE ROBUST GCN In Chapter 6, we further investigate the interaction between graph structure information and node feature information in the GCN model. We start from a simple synthetic example which is a 2-community SBM graph with node features being random vectors following 2 distinct multi-variate Gaussian distribution. We then propose a *random* GCN model, whose weight $\boldsymbol{W}$ in the *Update* step is a high-dimensional random Gaussian matrix fixed during training. Performance of this model approaches asymptotically that of a vanilla GCN, as the dimension of $\boldsymbol{W}$ is high enough ($\sim 10^3$ in practice). The assumption on the data and the *random* GCN allow us to introduce the tools from the random matrix theory developed in recent years to analyse classic machine learning. Theoretically, we analyse the spectral behaviour of the model after graph propagation and show that the noise in graph structure will heavily shadow the information from node features. If the graph structure is sufficiently perturbed, the node features cannot contribute to the model performance even if they are precise and informative. Inspired by the theoretical results, we propose adding a node feature kernel directly to the graph propagation step to robustify the model performance against structural perturbation. Experiments on node classification of both synthetic and realistic datasets demonstrate the effectiveness of the proposed node feature kernel in lessening the impact of graph structure noises.

7.2 FUTURE DIRECTIONS

We discuss in this section the future research interests and highlight two exciting directions that remain to be investigated.

SPARSIFYING THE AGGREGATION STEP IN MPNN Chapter 5 proposes sparsifying the linear transform in the *Update* step in MPNNs to reduce computa-

tional costs. However, the conclusion lead us to believe that sparsifying the *Aggregation* step could be more beneficial. A natural question then arise:

> Does there exist a sparse subgraph in the graph structure that reduces computational cost without losing performance?

This question is similar to the *lottery ticket hypothesis* in a classic machine learning setting (Frankle and Carbin, 2019). It is argued that there exists a sub-network in the original neural network, which can achieve comparable performance to the original one with a heavily reduced number of parameters. This sub-network is usually obtained by successively pruning the network weights during training.

There have also been efforts in the GNN area to reduce the computational cost of graph propagation. Most GNN models following the message-passing framework will face exponential growth of neighbourhoods and cause severe memory and time cost. Researchers tried to tackle this *neighbours explosion* by sampling strategies. Hamilton, Ying, and Leskovec (2017) first proposed randomly sampling a fixed number of neighbours for each node in graph propagation rather than using the whole neighbourhood, in order to make the GNN more scalable. Chen, Ma, and Xiao (2018) proposed to sample a set of nodes as a shared neighbourhood for all nodes at each propagation, by which they can limit the growth rate of the neighbourhood to linear. More recently Zeng et al. (2020) proposed to sample subgraphs as input to GNNs instead of the original graph. All these works speed up the training of GNNs and make GNNs available to large graphs.

Differing from the sampling strategy, we are more interested in finding a single, fixed subgraph that replaces the original graph without harming performance. The closest work to our knowledge might be Chen et al. (2021), where the authors performed the same pruning on the adjacency matrix during GNN training and identified at the end a *winning* subgraph that is sparse and high-performing.

Encouraged by the finding in Chen et al. (2021), we believe our work in Chapter 5 can be extended to the *Aggregation* step in MPNNs. Since we demonstrated that other than pruning during training, the *winning lottery ticket* in neural networks can also be identified by a pre-defined subgraph that exhibits good properties, e. g., *expander* graphs are sparse but highly-connected. The same logic may apply to GNNs as we can identify a subgraph which is sparse or with heavily reduced size but, under some measure, is not too *far away* from the original graph, such as subgraphs ob-

tained by K-core decomposition (Alvarez-Hamelin et al., 2005) or spectral approximation (Hermsdorff and Gunderson, 2019).

COMBINING KERNEL AND NEURAL NETWORK There is a growing interest in linking kernels and neural networks. The initial motivation is theoretical, as our sound understanding of kernels can provide some new perspectives for deep neural networks. For example, the learning dynamics of gradient descent for over-parameterised deep neural networks are shown to be governed by a particular kernel obtained at the initialisation (Bietti and Mairal, 2019; Jacot, Hongler, and Gabriel, 2018). However, researchers soon realised that there is also a benefit to combining these two methods in practice. Mairal (2016) propose combining kernel with CNNs. This new architecture can encode the invariance with the kernel and thus improve the performance over vanilla CNNs. Chen, Jacob, and Mairal (2019) proposed a combination of kernel methods and RNNs and showed its advantage over existing methods for modelling biological sequences.

Indeed, kernels and neural networks have different characteristics. The kernels are unsupervised, easy-to-regularise and decoupled from downstream tasks, while the neural networks are scalable and end-to-end trainable. Bridging the gap between the two methods seems beneficial to inherit both their advantages.

It is also an exciting direction for us, as working throughout this dissertation, we have already set foot on both fields of Graph Kernels and Graph Neural Networks. Several links have been drawn between both methods. Nikolentzos et al. (2018) proposed generating features for a graph by computing its similarity with a series of pre-defined indicator graphs using graph kernels. The features are then fed into a traditional neural network for downstream tasks. Du et al. (2019) presented a new class of graph kernel, which is equivalent to infinitely-wide GNNs initialised with random weights and trained with gradient descent, and thus enjoys the full expressive power of GNNs. Chen, Jacob, and Mairal (2020) and Nikolentzos and Vazirgiannis (2020) both proposed an end-to-end trainable kernel GNN by leveraging the specific type of kernels (random walk kernel for one and path kernel for the other) and their continuous relaxation or kernel approximation.

One possible extension of our work towards this direction might be to explore further the optimal assignment kernel framework proposed in Chapter 3. This framework has two essential steps: (1) generating a *bag of*

vectors for each graph, (2) constructing a valid kernel that can compute similarities/distances between sets. A new kernel class might be designed by leveraging the recent advance in optimal transport kernel (Mialon et al., 2021) and the expressive power of GNNs for node embeddings (Mrabah et al., 2021). Another possible direction is based on Nikolentzos et al. (2018), where the authors proposed obtaining graph feature maps using graph kernels with pre-defined indicator graphs. These graphs are fixed before training, but it will be interesting to update them according to the loss during training. As we do not want to limit the choices of graph kernels, unlike in Chen, Jacob, and Mairal (2020) and Nikolentzos and Vazirgiannis (2020), this problem is complex as graph kernels, in general, is not differentiable. In this case, gradient-free optimisation such as evolutionary algorithms or functional gradient descent (Shen et al., 2020) that restricts all operations in the Hilbert space instead of the graph space might be helpful.

7.3 EPILOGUE

The domain of graph representation learning has grown very fast in recent years. Exciting works come out every day and week. I am fortunate to spend my doctoral years during this era as I witnessed closely the emerge of Graph Neural Networks and the shift of paradigm since then in the graph representation learning area. Significant advances have been made, but many questions and challenges remain. I sincerely hope that the works described in this dissertation will help research move forward and shed some light on future studies towards more efficient, robust and versatile algorithms for graph-structured data.

Part IV

APPENDIX

A

APPENDIX TO CHAPTER 4

A.1 ADDITIONAL EXPERIMENTS

A.1.1 *Synthetic Datasets*

GRAPH SIZE COMPARISON We compare the size of generated graph with the real graph at the corresponding time step in Figure A.1. We show two scenario: Path graph with removal and Cycle graph with adding extra structure. Both show similar behaviour with the scenario considered in Figure 4.3.

SIMILARITY HISTOGRAM We record the similarity histogram of synthetic graphs in Figure A.2. As stated in Section 4.4, our proposed method is able to generate graphs very similar to the ground truth.

EXAMPLES OF THE PREDICTION We show some examples of the predicted graph and compare them with the ground truth. See Figure A.3, A.4, A.5, A.6, A.7, A.8, respectively for Path graphs, Ladder graphs with small size, Ladder graphs with large size, Cycle graphs, Path graphs with removal, Cycle graphs with adding extra structures. Our model is able to reconstruct some characteristic structures from the ground truth, but the exact size recovery is still difficult, in particular for the complex scenario.

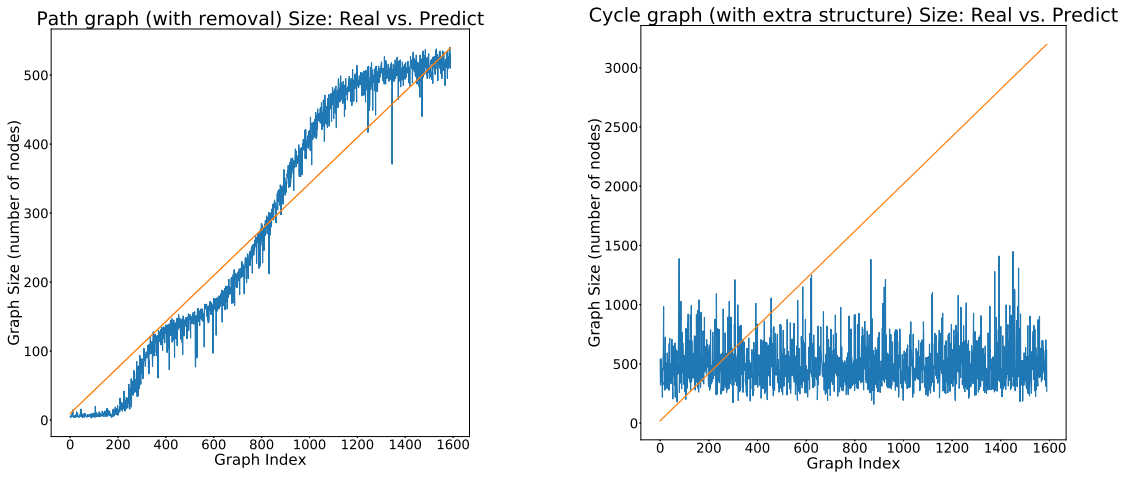

Figure A.1: Comparison of graph size: predicted size (blue) vs. real size (orange). **Left:** Path graphs with removal; **Right:** Cycle graphs with adding extra structures.

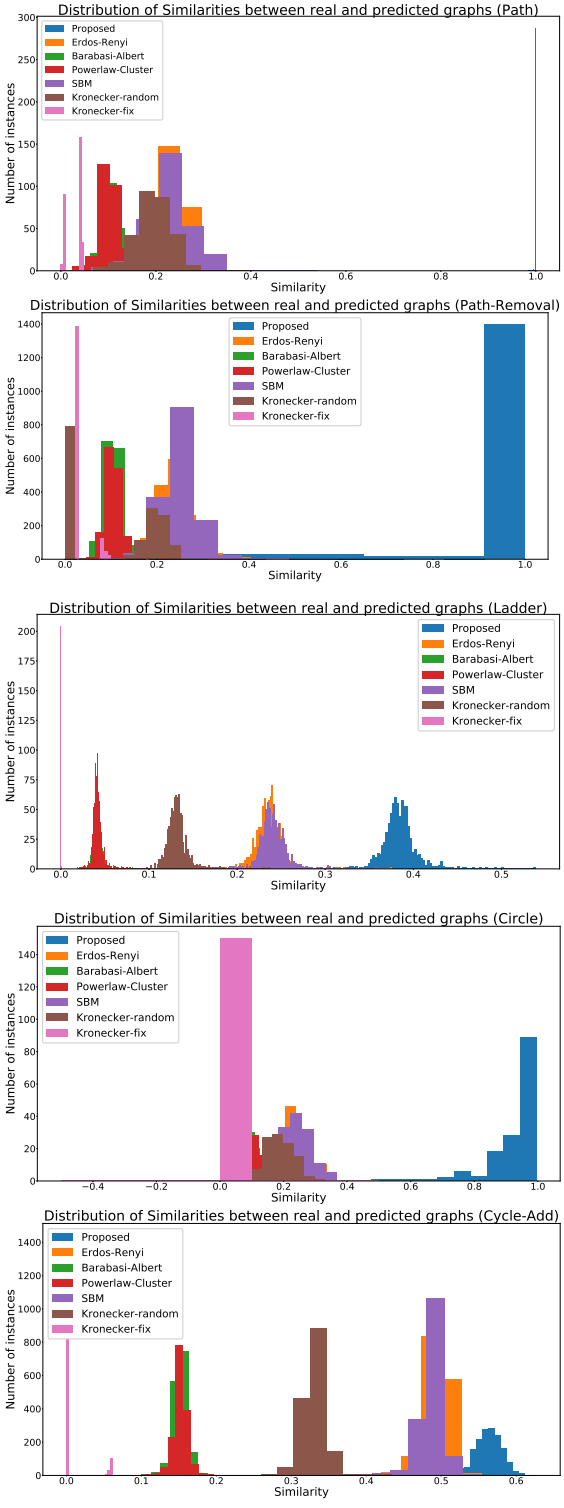

Figure A.2: Similarity histograms on synthetic datasets. Blue one is the result of **EvoNet**, which is compared against 6 random graph models. From **top** to **bottom**: Path graphs; Path graphs with removal; Ladder graphs; Cycle graphs; Cycle graphs with adding extra structures.

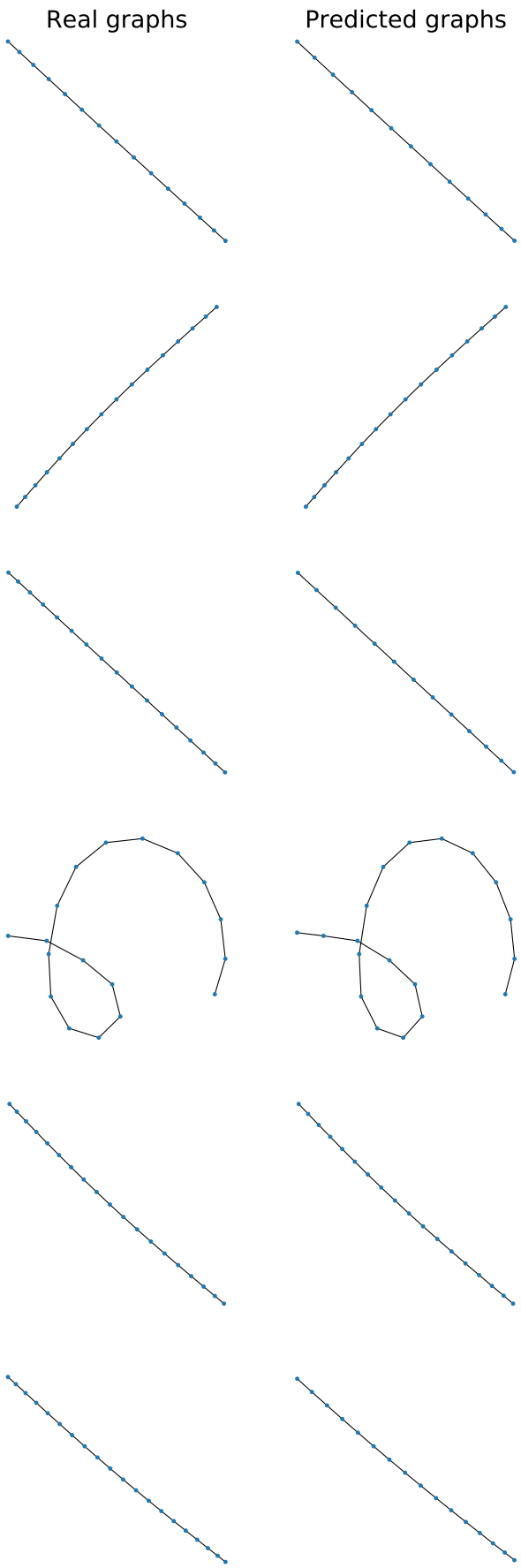

Figure A.3: Examples of predictions on Path graphs. **Left:** Real graphs; **Right:** Predicted graphs.

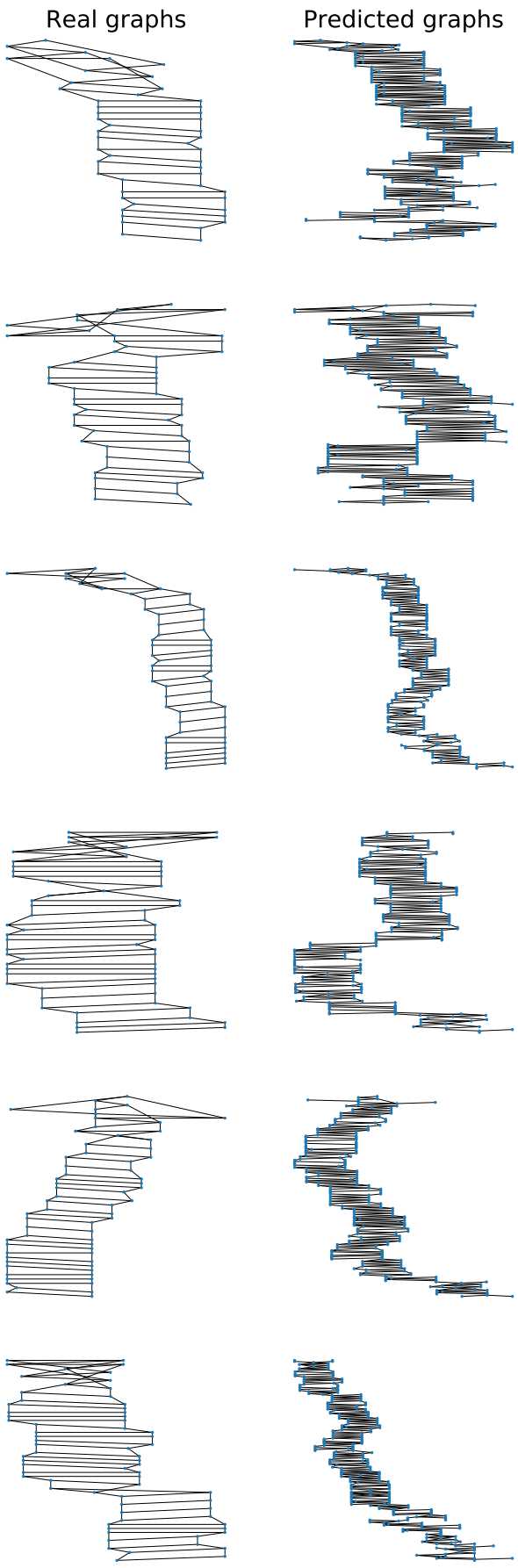

Figure A.4: Examples of predictions on *small sized* Ladder graphs. **Left:** Real graphs; **Right:** Predicted graphs.

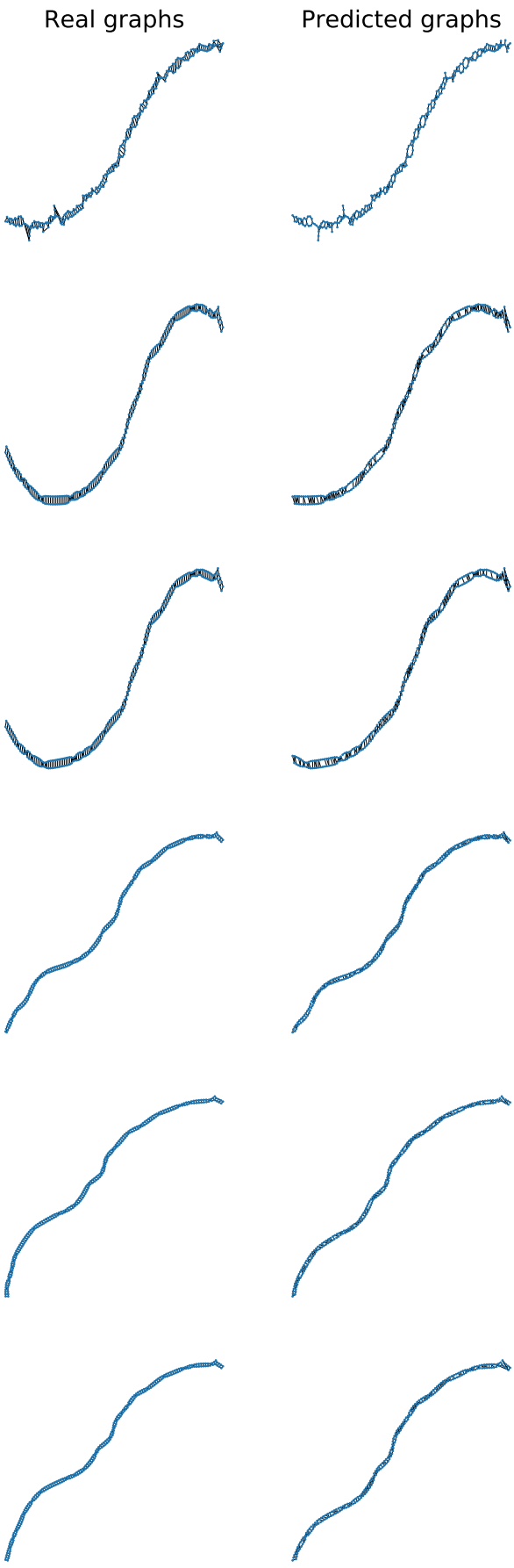

Figure A.5: Examples of predictions on *large sized* Ladder graphs. **Left:** Real graphs; **Right:** Predicted graphs.

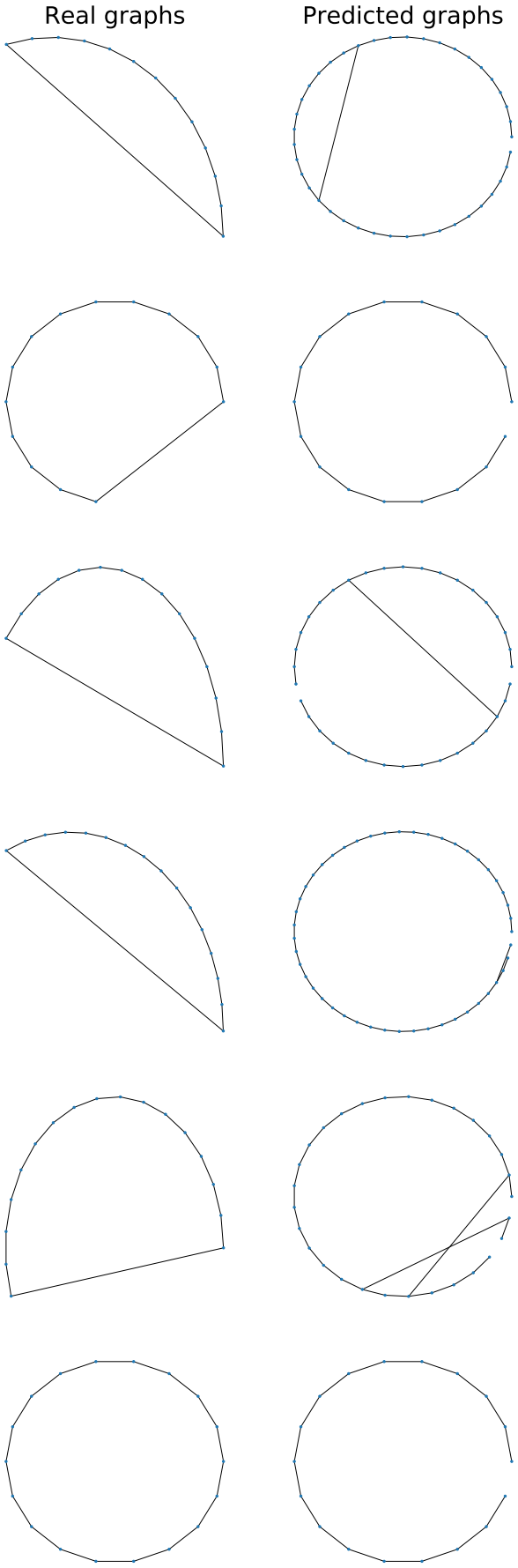

Figure A.6: Examples of predictions on Cycle graphs. **Left:** Real graphs; **Right:** Predicted graphs.

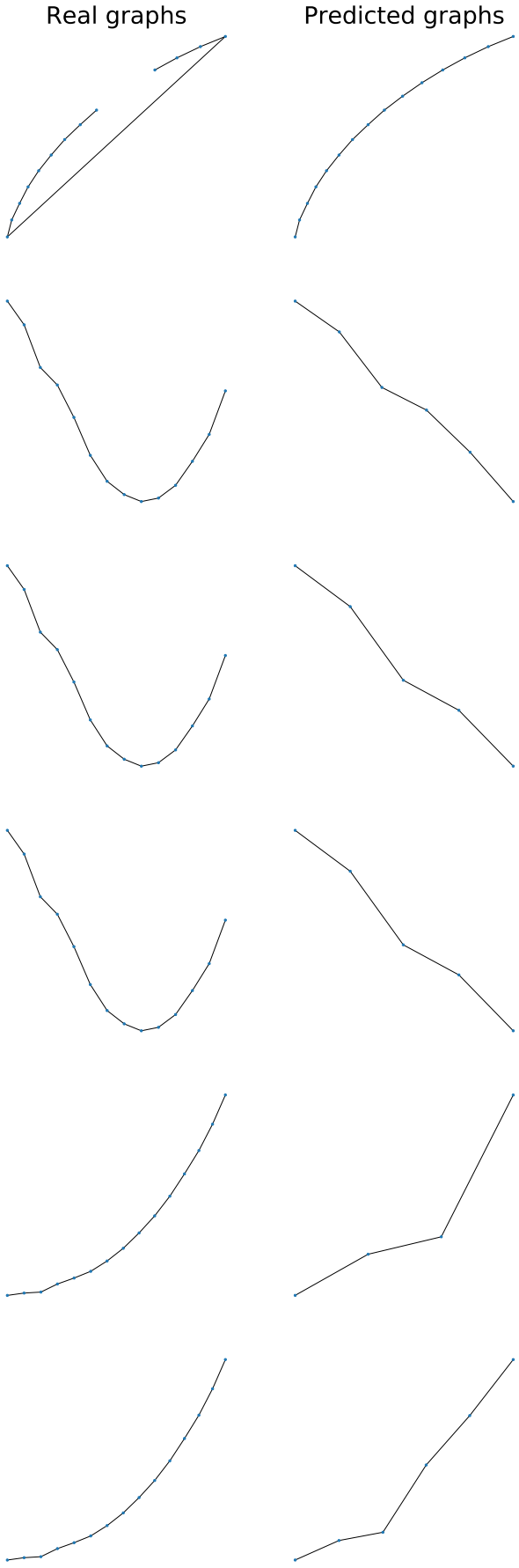

Figure A.7: Examples of predictions on Path graphs with removal. **Left:** Real graphs; **Right:** Predicted graphs.

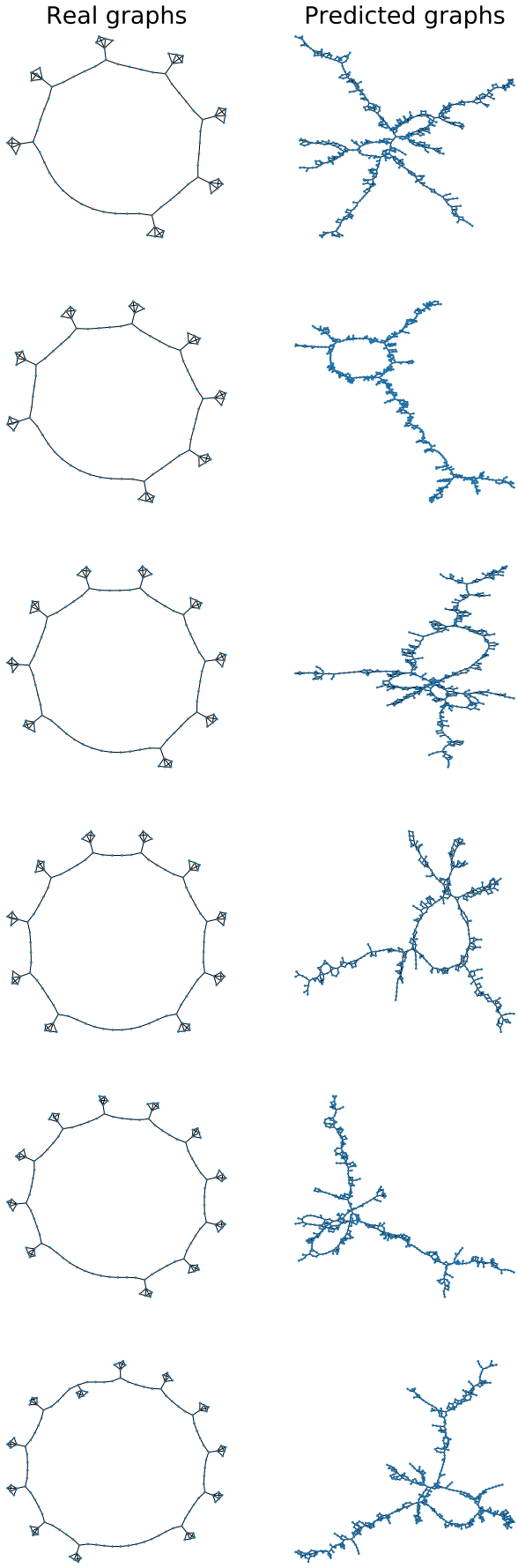

Figure A.8: Examples of predictions on Cycle datasets with adding extra structures. **Left:** Real graphs; **Right:** Predicted graphs.

B

APPENDIX TO CHAPTER 6

B.1 PRELIMINARIES OF RANDOM MATRIX THEORY

We begin recalling several random matrix theory tools that are needed to establish our main results. First, we recall a fundamental result (Theorem B.1.1) from Louart and Couillet, 2018 which provides a deterministic equivalent for the *resolvent* of a *sample covariance* matrix.

THEOREM B.1.1 (DETERMINISTIC EQUIVALENT FOR SAMPLE COVARIANCE MATRICES, LOUART AND COUILLET, 2018). *Let $\boldsymbol{M} \in \mathbb{R}^{n\times n}$ such that $\|\boldsymbol{M}\boldsymbol{M}^\intercal\| < \infty$[1] w.r.t. n and $\tilde{\boldsymbol{Z}} \in \mathbb{R}^{n\times p}$ some random matrix with i.i.d. entries having zero mean, unit variance and a finite forth order moment. In the limit $n \to \infty$ with $p/n \to c \in (0,\infty)$, the resolvent $\boldsymbol{Q}(z) = \left(\frac{1}{p}\boldsymbol{M}\tilde{\boldsymbol{Z}}\tilde{\boldsymbol{Z}}^\intercal\boldsymbol{M}^\intercal + z\boldsymbol{I}_n\right)^{-1}$ for $z \in \mathbb{C}$ with $\Im(z) > 0$, admits a deterministic equivalent $\bar{\boldsymbol{Q}}(z)$ defined as*

$$\bar{\boldsymbol{Q}}(z) = \left(\frac{\boldsymbol{M}\boldsymbol{M}^\intercal}{1+\delta(z)} + z\boldsymbol{I}_n\right)^{-1},$$

where $\delta(z)$ is the unique solution to the fixed point equation $\delta(z) = \frac{1}{p}\operatorname{Tr}\left(\boldsymbol{M}^\intercal\bar{\boldsymbol{Q}}(z)\boldsymbol{M}\right)$.

Proof. Denote $\boldsymbol{a}_i = \boldsymbol{M}\tilde{\boldsymbol{z}}_i$, hence

$$\boldsymbol{Q}(z) = \left(\frac{1}{p}\sum_{i=1}^{n}\boldsymbol{a}_i\boldsymbol{a}_i^\intercal + z\boldsymbol{I}_n\right)^{-1} = \boldsymbol{Q}_{-i} - \frac{\boldsymbol{Q}_{-i}\frac{1}{p}\boldsymbol{a}_i\boldsymbol{a}_i^\intercal\boldsymbol{Q}_{-i}}{1+\frac{1}{p}\boldsymbol{a}_i^\intercal\boldsymbol{Q}_{-i}\boldsymbol{a}_i},$$

where $\boldsymbol{Q}_{-i} = \left(\frac{1}{p}\sum_{j\neq i}^{n}\boldsymbol{a}_j\boldsymbol{a}_j^\intercal + z\boldsymbol{I}_n\right)^{-1}$, and we also have

$$\boldsymbol{Q}(z)\boldsymbol{a}_i = \frac{\boldsymbol{Q}_{-i}\boldsymbol{a}_i}{1+\frac{1}{p}\boldsymbol{a}_i^\intercal\boldsymbol{Q}_{-i}\boldsymbol{a}_i}.$$

1 $\|\boldsymbol{M}\boldsymbol{M}^\intercal\|$ remains constant as n goes to infinity.

A deterministic equivalent for $\boldsymbol{Q}(z)$ (which approximates $\mathbb{E}[\boldsymbol{Q}(z)]$) is of the form $\bar{\boldsymbol{Q}}(z) = (\boldsymbol{F} + z\boldsymbol{I}_n)^{-1}$ for some deterministic matrix $\boldsymbol{F}$, by computing the difference $\bar{\boldsymbol{Q}}(z) - \mathbb{E}[\boldsymbol{Q}(z)]$ using the above identities, we obtain

$$\bar{\boldsymbol{Q}}(z) - \mathbb{E}[\boldsymbol{Q}(z)] = \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}\left[\boldsymbol{Q}_{-i}\left(\frac{\boldsymbol{a}_i\boldsymbol{a}_i^\intercal}{1+\frac{1}{p}\boldsymbol{a}_i^\intercal\boldsymbol{Q}_{-i}\boldsymbol{a}_i} - \boldsymbol{F}\right)\bar{\boldsymbol{Q}}(z)\right] + \frac{1}{n^2}\sum_{i=1}^{n}\mathbb{E}\left[\boldsymbol{Q}_{-i}(z)\boldsymbol{a}_i\boldsymbol{a}_i^\intercal\boldsymbol{Q}_{-i}(z)\boldsymbol{F}\bar{\boldsymbol{Q}}(z)\right].$$

It can be shown that the matrix $\frac{1}{n^2}\sum_{i=1}^{n}\mathbb{E}\left[\boldsymbol{Q}_{-i}(z)\boldsymbol{a}_i\boldsymbol{a}_i^\intercal\boldsymbol{Q}_{-i}(z)\boldsymbol{F}\bar{\boldsymbol{Q}}(z)\right]$ has a vanishing operator norm as $n \to \infty$. Therefore, $\boldsymbol{F}$ can be taken as

$$\boldsymbol{F} = \frac{\mathbb{E}\left[\boldsymbol{a}_i\boldsymbol{a}_i^\intercal\right]}{1+\mathbb{E}\left[\frac{1}{p}\boldsymbol{a}_i^\intercal\boldsymbol{Q}_{-i}\boldsymbol{a}_i\right]} = \frac{\boldsymbol{M}\boldsymbol{M}^\intercal}{1+\frac{1}{p}\operatorname{Tr}(\boldsymbol{M}^\intercal\mathbb{E}\left[\boldsymbol{Q}_{-i}\right]\boldsymbol{M})}$$

Which provides the defined deterministic equivalent in Theorem B.1.1. □

Another useful result which is known as the perturbation lemma (Silverstein and Bai, 1995) is also needed here.

LEMMA B.1.2 (PERTURBATION LEMMA, SILVERSTEIN AND BAI, 1995). *Let $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n\times n}$ some symmetric matrices, $\boldsymbol{u} \in \mathbb{R}^n$, $\gamma \in \mathbb{R}$ and $z \in \mathbb{C}$ with $\Im(z) > 0$, then*

$$\left|\operatorname{Tr}\left(\boldsymbol{A}(\boldsymbol{B}+\gamma\boldsymbol{u}\boldsymbol{u}^\intercal+z\boldsymbol{I}_n)^{-1}\right) - \operatorname{Tr}\left(\boldsymbol{A}(\boldsymbol{B}+z\boldsymbol{I}_n)^{-1}\right)\right| \leq \frac{\|\boldsymbol{A}\|}{|\Im(z)|}.$$

In particular, for $\boldsymbol{A} = \frac{1}{n}\boldsymbol{I}_n$, we have $\frac{1}{n}\operatorname{Tr}(\boldsymbol{B}+\gamma\boldsymbol{u}\boldsymbol{u}^\intercal+z\boldsymbol{I}_n)^{-1} = \frac{1}{n}\operatorname{Tr}(\boldsymbol{B}+z\boldsymbol{I}_n)^{-1} + \mathcal{O}(n^{-1})$, which shows that the spectral measure of $\boldsymbol{B}+\gamma\boldsymbol{u}\boldsymbol{u}^\intercal$ is asymptotically close to that of $\boldsymbol{B}$ in the large n limit.

Finally, we will need the Woodbury matrix identity from the following Lemma.

LEMMA B.1.3 (WOODBURY IDENTITY). *Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{B} \in \mathbb{R}^{k\times k}$ invertible and $\boldsymbol{U} \in \mathbb{R}^{n\times k}$, then*

$$(\boldsymbol{A}+\boldsymbol{U}\boldsymbol{B}\boldsymbol{U}^\intercal)^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{U}\left(\boldsymbol{B}^{-1}+\boldsymbol{U}^\intercal\boldsymbol{A}^{-1}\boldsymbol{U}\right)^{-1}\boldsymbol{U}^\intercal\boldsymbol{A}^{-1}.$$

B.2 PROOF OF THEOREM 3.4

The proof starts by establishing a random equivalent for the normalised Adjacency operator given by $\tilde{\boldsymbol{A}} = \frac{1}{\sqrt{n}}(\boldsymbol{A} - \boldsymbol{q}\boldsymbol{q}^\mathsf{T})$. By Assumptions 2, 3 and 5, we have straightforwardly that, almost surely

$$\left\| \tilde{\boldsymbol{A}} - \left(q^2 \eta \bar{\boldsymbol{y}} \bar{\boldsymbol{y}}^\mathsf{T} + \frac{1}{\sqrt{n}} \boldsymbol{N} \right) \right\| \to 0, \tag{B.1}$$

where $\boldsymbol{N}$ is a random matrix with i.i.d. entries of zero mean and variance $v = q^2(1-q^2)$. Besides, since $\mathbb{E}[\boldsymbol{X}\boldsymbol{X}^\mathsf{T}] = \frac{\|\boldsymbol{\mu}\|^2}{c} \bar{\boldsymbol{y}} \bar{\boldsymbol{y}}^\mathsf{T} + \boldsymbol{I}_n$, letting $\gamma_f = \|\boldsymbol{\mu}\|^2$ and $\gamma_g = q^2\eta$, we consider the equivalent multiplicative model for $\boldsymbol{Y}$ defined as

$$\boldsymbol{Y} = \left(\gamma_g \bar{\boldsymbol{y}} \bar{\boldsymbol{y}}^\mathsf{T} + \frac{1}{\sqrt{n}} \boldsymbol{N} \right) \left(\frac{\gamma_f}{c} \bar{\boldsymbol{y}} \bar{\boldsymbol{y}}^\mathsf{T} + \boldsymbol{I}_n \right)^{\frac{1}{2}} \boldsymbol{Z},$$

where $\boldsymbol{Z}$ is random matrix with i.i.d. entries of zero mean and variance $\frac{1}{p}$. Conditionally on $\boldsymbol{N}$, applying Theorem B.1.1 for $\boldsymbol{M} = \left(\gamma_g \bar{\boldsymbol{y}} \bar{\boldsymbol{y}}^\mathsf{T} + \frac{1}{\sqrt{n}} \boldsymbol{N} \right) \left(\frac{\gamma_f}{c} \bar{\boldsymbol{y}} \bar{\boldsymbol{y}}^\mathsf{T} + \boldsymbol{I}_n \right)^{\frac{1}{2}}$, a deterministic equivalent of $\boldsymbol{Q}_{\boldsymbol{Y}}(z) = (\boldsymbol{Y}\boldsymbol{Y}^\mathsf{T} + z\boldsymbol{I}_n)^{-1}$ is given by

$$\bar{\boldsymbol{Q}}_{\boldsymbol{Y}|\boldsymbol{N}}(z) = \left(\frac{\left(\gamma_g \bar{\boldsymbol{y}} \bar{\boldsymbol{y}}^\mathsf{T} + \frac{1}{\sqrt{n}} \boldsymbol{N} \right) \left(\frac{\gamma_f}{c} \bar{\boldsymbol{y}} \bar{\boldsymbol{y}}^\mathsf{T} + \boldsymbol{I}_n \right) \left(\gamma_g \bar{\boldsymbol{y}} \bar{\boldsymbol{y}}^\mathsf{T} + \frac{1}{\sqrt{n}} \boldsymbol{N} \right)}{1 + \delta_1(z)} + z\boldsymbol{I}_n \right)^{-1} \tag{B.2}$$

$$= \left(\frac{\boldsymbol{U}\boldsymbol{B}\boldsymbol{U}^\mathsf{T} + \frac{1}{n}\boldsymbol{N}\boldsymbol{N}^\mathsf{T}}{1 + \delta_1(z)} + z\boldsymbol{I}_n \right)^{-1} \tag{B.3}$$

where

$$\boldsymbol{U} = [\bar{\boldsymbol{y}}, \boldsymbol{\phi}] \in \mathbb{R}^{n \times 2}, \quad \boldsymbol{B} = \begin{bmatrix} \gamma_2^2 \left(\frac{\gamma_1}{c} + 1 \right) & \left(\frac{\gamma_1}{c} + 1 \right) \gamma_2 \\ \left(\frac{\gamma_1}{c} + 1 \right) \gamma_2 & \frac{\gamma_1}{c} \end{bmatrix} \tag{B.4}$$

with $\boldsymbol{\phi} = \frac{1}{\sqrt{n}} \boldsymbol{N} \bar{\boldsymbol{y}}$, $\bar{\boldsymbol{y}} = \boldsymbol{y}/\sqrt{n}$ and $\delta_1(z) = \frac{1}{p} \operatorname{Tr}\left(\left(\boldsymbol{U}\boldsymbol{B}\boldsymbol{U}^\mathsf{T} + \frac{1}{n}\boldsymbol{N}\boldsymbol{N}^\mathsf{T} \right) \bar{\boldsymbol{Q}}_{\boldsymbol{Y}|\boldsymbol{N}}(z) \right)$.

Applying Lemma B.1.2, $\delta_1(z)$ is simply the solution to $\delta_1(z) = \frac{1}{p} \operatorname{Tr}\left(\frac{1}{n}\boldsymbol{N}\boldsymbol{N}^\mathsf{T} \bar{\boldsymbol{Q}}_{\boldsymbol{Y}|\boldsymbol{N}}(z) \right)$. Moreover, defining the matrix $\bar{\boldsymbol{Q}}^{\boldsymbol{B}}_{\boldsymbol{Y}|\boldsymbol{N}}(z) = \left(\frac{\frac{1}{n}\boldsymbol{N}\boldsymbol{N}^\mathsf{T}}{1+\delta_1(z)} + z\boldsymbol{I}_n \right)^{-1}$, we have by Lemma B.1.3

$$\bar{\boldsymbol{Q}}_{\boldsymbol{Y}|\boldsymbol{N}}(z) = \bar{\boldsymbol{Q}}^{\boldsymbol{B}}_{\boldsymbol{Y}|\boldsymbol{N}}(z) - \bar{\boldsymbol{Q}}^{\boldsymbol{B}}_{\boldsymbol{Y}|\boldsymbol{N}}(z) \boldsymbol{U} \left((1 + \delta_1(z)) \boldsymbol{B}^{-1} + \boldsymbol{U}^\mathsf{T} \bar{\boldsymbol{Q}}^{\boldsymbol{B}}_{\boldsymbol{Y}|\boldsymbol{N}}(z) \boldsymbol{U} \right)^{-1} \boldsymbol{U}^\mathsf{T} \bar{\boldsymbol{Q}}^{\boldsymbol{B}}_{\boldsymbol{Y}|\boldsymbol{N}}(z). \tag{B.5}$$

Again by Theorem B.1.1, a deterministic equivalent of $\bar{\boldsymbol{Q}}_{\boldsymbol{Y}|\boldsymbol{N}}^{-\boldsymbol{B}}(z)$ is given by

$$\bar{\boldsymbol{Q}}_{\boldsymbol{Y}}^{-\boldsymbol{B}}(z) = \left(\frac{\nu \boldsymbol{I}_n}{(1+\delta_1(z))(1+\delta_2(z))} + z\boldsymbol{I}_n \right)^{-1} = \frac{(1+\delta_1(z))(1+\delta_2(z))}{\nu + z(1+\delta_1(z))(1+\delta_2(z))} \boldsymbol{I}_n \tag{B.6}$$

where $\delta_2(z)$ is the unique solution to $\delta_2(z) = \frac{1}{n} \operatorname{Tr}\left(\frac{\nu \boldsymbol{I}_n}{(1+\delta_1(z))} \bar{\boldsymbol{Q}}_{\boldsymbol{Y}}^{-\boldsymbol{B}}(z) \right) = \frac{\nu(1+\delta_2(z))}{\nu + z(1+\delta_1(z))(1+\delta_2(z))}$, and similarly $\delta_1(z)$ satisfies $\delta_1(z) = \frac{1}{c} \frac{\nu(1+\delta_1(z))}{\nu + z(1+\delta_1(z))(1+\delta_2(z))}$. Therefore, replacing $\bar{\boldsymbol{Q}}_{\boldsymbol{Y}|\boldsymbol{N}}^{-\boldsymbol{B}}(z)$ in (B.5) by its deterministic equivalent $\bar{\boldsymbol{Q}}_{\boldsymbol{Y}}^{-\boldsymbol{B}}(z)$ and since $\boldsymbol{U}^\intercal \boldsymbol{U} \to \boldsymbol{T}$ almost surely, provides the final result of Theorem 3.4.

B.3 PROOF OF COROLLARY 3.5

Following the same procedure as in Section B.2, when $\eta = 0$, a deterministic equivalent for $\boldsymbol{Q}_{\boldsymbol{Y}}(z)$ takes the form

$$\bar{\boldsymbol{Q}}_{\boldsymbol{Y}}(z) = \zeta(z)(1+\delta_1(z)) \left(\boldsymbol{I}_n - \frac{\zeta^2(z)\gamma_f}{c + \nu\gamma_f \zeta(z)} \boldsymbol{\phi}\boldsymbol{\phi}^\intercal \right). \tag{B.7}$$

By definition of the deterministic equivalent, we have almost surely

$$|\bar{\boldsymbol{y}}^\intercal \hat{\boldsymbol{y}}|^2 = \frac{-1}{2\pi i} \oint_\Gamma \bar{\boldsymbol{y}}^\intercal \boldsymbol{Q}_{\boldsymbol{Y}}(-z) \bar{\boldsymbol{y}} dz \to_{n\to\infty} \frac{-1}{2\pi i} \oint_\Gamma \bar{\boldsymbol{y}}^\intercal \bar{\boldsymbol{Q}}_{\boldsymbol{Y}}(-z) \bar{\boldsymbol{y}} dz. \tag{B.8}$$

Hence, we need to evaluate the Cauchy-integral $\frac{-1}{2\pi i} \oint_\Gamma \bar{\boldsymbol{y}}^\intercal \bar{\boldsymbol{Q}}_{\boldsymbol{Y}}(-z) \bar{\boldsymbol{y}} dz$. In particular, the quadratic form $\bar{\boldsymbol{y}}^\intercal \bar{\boldsymbol{Q}}_{\boldsymbol{Y}}(z) \bar{\boldsymbol{y}}$ evaluates as

$$\bar{\boldsymbol{y}}^\intercal \bar{\boldsymbol{Q}}_{\boldsymbol{Y}}(z) \bar{\boldsymbol{y}} = \zeta(z)(1+\delta_1(z)) \left(1 - \frac{\zeta^2(z)\gamma_f}{c + \nu\gamma_f \zeta(z)} \bar{\boldsymbol{y}}^\intercal \boldsymbol{\phi}\boldsymbol{\phi}^\intercal \bar{\boldsymbol{y}} \right) \to_{n\to\infty} \zeta(z)(1+\delta_1(z)).$$

Indeed, since the mapping $\boldsymbol{X} \mapsto \frac{1}{\sqrt{n}} \bar{\boldsymbol{y}}^\intercal \boldsymbol{X} \bar{\boldsymbol{y}}$ is $\frac{1}{\sqrt{n}}$-Lipschitz transformation w.r.t. the Frobenius norm $\|\cdot\|_F$, then we have the concentration inequality, for all $t \geq 0$

$$\mathbb{P}\left(\left| \frac{1}{\sqrt{n}} \bar{\boldsymbol{y}}^\intercal \boldsymbol{N} \bar{\boldsymbol{y}} - \mathbb{E}\left[\frac{1}{\sqrt{n}} \bar{\boldsymbol{y}}^\intercal \boldsymbol{N} \bar{\boldsymbol{y}} \right] \right| > t \right) \leq C e^{-(\sqrt{n}t/\nu)^2}, \tag{B.9}$$

Table B.1: Performance of the GCN with and without node-feature kernel under perturbation on six real-world datasets with **multiple train/valid/test splits**. The format follows Table 6.1.

		Cora		CiteSeer		PubMed	
	(α, β)	GCN	GCN-k	GCN	GCN-k	GCN	GCN-k
	(0.0, 0.0)	76.25 ± 2.32	75.48 ± 1.87	66.31 ± 2.04	66.53 ± 2.10	74.80 ± 2.07	76.93 ± 2.20
Deletion	(0.2, 0.0)	74.82 ± 1.68	73.26 ± 1.88	64.96 ± 1.57	64.27 ± 2.34	75.14 ± 2.06	74.21 ± 2.84
	(0.5, 0.0)	70.78 ± 1.53	70.80 ± 1.83	63.14 ± 1.49	62.74 ± 1.80	74.17 ± 1.75	74.11 ± 2.06
Insertion	(0.0, 0.5)	67.01 ± 2.11	$\mathbf{71.96 \pm 1.79}$	57.35 ± 2.02	$\mathbf{62.19 \pm 1.68}$	63.70 ± 2.95	$\mathbf{71.62 \pm 2.01}$
	(0.0, 1.0)	58.92 ± 2.29	$\mathbf{68.50 \pm 1.45}$	50.53 ± 2.13	$\mathbf{59.60 \pm 2.04}$	57.07 ± 1.94	$\mathbf{69.56 \pm 1.92}$
		CoraFull		Photo		CS	
	(α, β)	GCN	GCN-k	GCN	GCN-k	GCN	GCN-k
	(0.0, 0.0)	58.42 ± 2.12	57.64 ± 1.45	91.48 ± 0.94	91.05 ± 1.28	92.09 ± 0.98	92.62 ± 0.92
Deletion	(0.2, 0.0)	56.36 ± 1.70	57.18 ± 1.68	90.51 ± 1.29	91.30 ± 1.22	91.37 ± 1.26	91.79 ± 0.91
	(0.5, 0.0)	54.58 ± 1.33	53.44 ± 1.47	90.07 ± 1.63	90.16 ± 1.10	89.78 ± 1.04	91.08 ± 1.27
Insertion	(0.0, 0.5)	48.53 ± 1.58	$\mathbf{53.55 \pm 1.44}$	81.77 ± 2.31	83.52 ± 1.43	88.19 ± 0.96	$\mathbf{91.25 \pm 1.00}$
	(0.0, 1.0)	43.18 ± 1.73	$\mathbf{50.77 \pm 1.89}$	75.21 ± 4.30	79.02 ± 3.04	83.83 ± 1.60	$\mathbf{90.32 \pm 0.89}$

for some constant $C \geq 0$ independent of n. In particular, since $\mathbb{E}\left[\frac{1}{\sqrt{n}}\bar{\boldsymbol{y}}^\intercal \boldsymbol{N}\bar{\boldsymbol{y}}\right] = 0$, we have

$$\mathbb{P}\left(\left|\left(\frac{1}{\sqrt{n}}\bar{\boldsymbol{y}}^\intercal \boldsymbol{N}\bar{\boldsymbol{y}}\right)^2 - \mathbb{E}\left[\left(\frac{1}{\sqrt{n}}\bar{\boldsymbol{y}}^\intercal \boldsymbol{N}\bar{\boldsymbol{y}}\right)^2\right]\right| > t\right) \leq C\, e^{-\frac{\sqrt{n}\, t}{2\nu}}, \tag{B.10}$$

which shows that $\bar{\boldsymbol{y}}^\intercal \boldsymbol{\phi}\boldsymbol{\phi}^\intercal \bar{\boldsymbol{y}} = \left(\frac{1}{\sqrt{n}}\bar{\boldsymbol{y}}^\intercal \boldsymbol{N}\bar{\boldsymbol{y}}\right)^2$ concentrates around its mean value, with

$$\mathbb{E}\left[\left(\frac{1}{\sqrt{n}}\bar{\boldsymbol{y}}^\intercal \boldsymbol{N}\bar{\boldsymbol{y}}\right)^2\right] = \frac{1}{n}\mathrm{Var}\left[\bar{\boldsymbol{y}}^\intercal \boldsymbol{N}\bar{\boldsymbol{y}}\right] = \frac{1}{n}\sum_{i,j=1}^{n} \bar{y}_i^2 \bar{y}_j^2 \mathrm{Var}[N_{ij}] = \frac{\|\bar{\boldsymbol{y}}\|^4 \nu}{n} \to 0.$$

Therefore, $\bar{\boldsymbol{y}}^\intercal \boldsymbol{\phi}\boldsymbol{\phi}^\intercal \bar{\boldsymbol{y}} \to 0$ almost surely as $n \to \infty$. The final step consists in evaluating the integral $\frac{1}{2\pi i}\oint_\Gamma \zeta(-z)(1+\delta_1(-z))dz = 0$ since the function $z \mapsto \zeta(-z)(1+\delta_1(-z))$ does not have singularities on the contour Γ. Indeed, this integral corresponds to the only noise case from the data model (i.e., $\boldsymbol{Y} = \frac{1}{\sqrt{n}}\boldsymbol{N}\boldsymbol{Z}$).

B.4 ADDITIONAL EXPERIMENTS

B.4.1 *Multiple Splits*

Shchur et al. (2018) argue that different train/valid/test splits of datasets may have a non-negligible impact on the performance of GNN models for the node classification task. To investigate the influence of different splits on our hypothesis, we construct train/valid/test split for each dataset following Yang, Cohen, and Salakhutdinov (2016) over 10 random seeds. As each experiment is repeated 10 times, a total of 100 results is obtained for a specific setting, i.e., specific α, β or ϵ. Similar to the results of one split, we report the mean and standard deviations of the 100 results in Table B.1. Although the standard deviation increases since we introduce more variation in the multi-split setting, the general trend remains the same, as shown in Table B.1. Our conclusion drawn in the main paper still holds: *perturbations of the graph structure strongly influence the performance of the GCN and adding a node feature kernel can robustify the GCN against such perturbations*.

B.4.2 *Models beyond GCN*

We also study the behaviour of our proposed method in a more general MPNN setting beyond GCN. We choose three characteristic models, which are GIN (Xu et al., 2019), GraphSage (Hamilton, Ying, and Leskovec, 2017) and Graph Attention Network (GAT) (Velickovic et al., 2018) models, and observe their performance under graph structural perturbation with node feature kernel (our proposed method) on three citation datasets. The models are also implemented as a single graph-propagation layer followed by a MLP readout, as was the case for the GCN. Results are gathered in Table B.2. Similar to the results of the GCN, we can observe from Table B.2 that on every model and every dataset, adding a node-feature kernel in the graph propagation helps to robustify the model performance against structural noises, in particular when edges are added. This empirical evidence demonstrates the versatility of our proposed method for a general MPNN model.

B.4.3 *Deeper GCN architecture and Benchmark Models*

In this set of experiments, we observe to what extent the conclusions drawn in our theoretical analysis carry over to deeper GCN architectures, and how our

Table B.2: Performance of the GIN/GraphSage/GAT with and without node-feature kernel under perturbation on three citation datasets. The format follows Table 6.1.

		Cora					
	(α, β)	GIN	GIN-k	Sage	Sage-k	GAT	GAT-k
Deletion	(0.0, 0.0)	78.56 ± 1.12	78.59 ± 0.8	75.94 ± 0.25	$\mathbf{77.12 \pm 0.65}$	78.20 ± 0.54	78.55 ± 0.69
	(0.2, 0.0)	76.61 ± 0.58	76.73 ± 1.79	73.87 ± 0.92	75.29 ± 1.25	75.54 ± 0.77	$\mathbf{77.01 \pm 0.30}$
	(0.5, 0.0)	71.31 ± 1.06	70.57 ± 0.63	67.35 ± 0.92	$\mathbf{71.98 \pm 0.99}$	71.25 ± 0.89	72.37 ± 0.74
Insertion	(0.0, 0.5)	71.08 ± 1.08	72.19 ± 0.87	70.24 ± 1.01	71.42 ± 1.23	67.45 ± 1.21	$\mathbf{72.13 \pm 0.53}$
	(0.0, 1.0)	66.21 ± 1.52	67.96 ± 0.67	66.0 ± 1.23	$\mathbf{70.57 \pm 0.95}$	62.14 ± 1.46	$\mathbf{67.68 \pm 0.89}$
Delet.+Insert.	(0.5, 0.5)	56.82 ± 1.35	$\mathbf{62.41 \pm 1.07}$	61.86 ± 1.32	63.61 ± 0.94	53.82 ± 0.86	$\mathbf{61.04 \pm 1.24}$
	(0.5, 1.0)	51.20 ± 2.04	$\mathbf{59.28 \pm 0.97}$	60.56 ± 0.81	$\mathbf{64.13 \pm 0.52}$	46.97 ± 1.15	$\mathbf{52.13 \pm 1.36}$
		CiteSeer					
	(α, β)	GIN	GIN-k	Sage	Sage-k	GAT	GAT-k
Deletion	(0.0, 0.0)	66.24 ± 0.89	66.96 ± 0.86	67.74 ± 0.85	68.97 ± 0.60	68.19 ± 0.92	67.82 ± 0.87
	(0.2, 0.0)	62.24 ± 1.19	$\mathbf{66.70 \pm 1.44}$	65.97 ± 1.06	66.22 ± 0.78	66.31 ± 0.93	66.90 ± 0.94
	(0.5, 0.0)	62.01 ± 1.01	62.30 ± 1.24	63.24 ± 0.59	64.97 ± 1.03	63.61 ± 1.03	65.43 ± 0.83
Insertion	(0.0, 0.5)	58.90 ± 1.31	$\mathbf{64.75 \pm 1.49}$	64.71 ± 0.94	66.21 ± 0.72	59.44 ± 1.34	$\mathbf{62.61 \pm 1.30}$
	(0.0, 1.0)	54.61 ± 1.28	$\mathbf{59.25 \pm 0.99}$	62.08 ± 0.99	63.04 ± 1.07	50.34 ± 0.99	$\mathbf{58.46 \pm 0.98}$
Delet.+Insert.	(0.5, 0.5)	50.46 ± 2.02	$\mathbf{57.90 \pm 1.51}$	57.88 ± 1.35	$\mathbf{63.56 \pm 0.67}$	50.75 ± 1.21	$\mathbf{55.81 \pm 1.04}$
	(0.5, 1.0)	43.98 ± 1.55	$\mathbf{48.40 \pm 1.47}$	58.51 ± 1.13	59.52 ± 1.07	43.56 ± 1.19	$\mathbf{50.17 \pm 1.35}$
		PubMed					
	(α, β)	GIN	GIN-k	Sage	Sage-k	GAT	GAT-k
Deletion	(0.0, 0.0)	77.13 ± 0.36	77.02 ± 0.58	76.44 ± 0.59	77.09 ± 0.64	76.08 ± 0.81	76.63 ± 0.47
	(0.2, 0.0)	75.96 ± 0.47	75.45 ± 0.44	75.38 ± 0.34	75.83 ± 0.83	75.93 ± 0.46	76.17 ± 0.55
	(0.5, 0.0)	74.29 ± 0.79	$\mathbf{76.09 \pm 0.45}$	72.10 ± 1.60	$\mathbf{76.24 \pm 0.49}$	73.46 ± 0.45	$\mathbf{76.05 \pm 0.51}$
Insertion	(0.0, 0.5)	71.85 ± 0.79	73.63 ± 1.10	73.48 ± 0.48	74.18 ± 0.43	67.42 ± 0.63	$\mathbf{69.80 \pm 0.69}$
	(0.0, 1.0)	64.27 ± 1.60	$\mathbf{69.61 \pm 1.30}$	74.37 ± 0.72	74.44 ± 0.32	64.73 ± 0.98	65.38 ± 0.73
Delet.+Insert.	(0.5, 0.5)	64.65 ± 1.07	$\mathbf{68.24 \pm 0.61}$	72.59 ± 0.61	$\mathbf{74.87 \pm 0.37}$	62.66 ± 0.81	$\mathbf{66.51 \pm 0.90}$
	(0.5, 1.0)	59.12 ± 1.19	$\mathbf{63.20 \pm 1.41}$	73.77 ± 0.37	72.64 ± 0.45	58.58 ± 0.80	$\mathbf{63.60 \pm 1.27}$

proposed model performs against similar methods in the deeper architecture setting. We add three extra message passing layer to the previous single-layer model and repeat our experiments on this deeper model as well as two state-of-the-art methods, the Jumping Knowledge (JK) and the GCN with residual connections and an identity mapping (GCNII), which are specifically designed for deeper models and take also advantage of the node feature information. Part of the results have already been shown in Section 4.1 of the main paper. In Table B.3 we show the results of the remaining four datasets, which agree with the trends observed in the main paper.

Table B.3: Performance of the GCN with and without node-feature kernel under perturbation on deep GCN models, compared with jump knowledge and GCNII. The format follows Table 6.1, where in addition we underline the second best result.

		CiteSeer					
	(α, β)	GCN	GCN-k ($\epsilon = 0.5$)	GCN-k ($\epsilon = 0.2$)	GCN-jk	GCNII	GCN-k-jk
Deletion	(0.0, 0.0)	66.72 ± 1.93	66.76 ± 0.78	67.75 ± 1.39	68.70 ± 1.06	67.59 ± 0.81	69.10 ± 0.92
	(0.2, 0.0)	66.57 ± 1.66	66.27 ± 1.34	66.91 ± 1.63	67.21 ± 0.41	66.13 ± 1.31	67.06 ± 0.97
	(0.5, 0.0)	62.68 ± 1.39	$\underline{64.52 \pm 2.03}$	62.72 ± 1.65	$\mathbf{65.49 \pm 0.94}$	61.91 ± 1.52	63.82 ± 2.01
Insertion	(0.0, 0.5)	55.76 ± 1.54	57.8 ± 0.99	$\underline{62.44 \pm 0.94}$	61.17 ± 0.85	60.52 ± 1.82	$\mathbf{63.15 \pm 1.39}$
	(0.0, 1.0)	45.14 ± 1.89	48.39 ± 1.88	56.69 ± 1.91	53.26 ± 1.44	$\underline{57.41 \pm 1.67}$	$\mathbf{62.83 \pm 1.06}$
Delet.+Insert.	(0.5, 0.5)	39.68 ± 1.96	48.26 ± 1.91	51.10 ± 1.26	49.50 ± 0.86	$\underline{53.19 \pm 1.58}$	$\mathbf{58.12 \pm 1.02}$
	(0.5, 1.0)	33.19 ± 2.1	40.5 ± 2.51	47.24 ± 2.41	44.8 ± 0.93	$\underline{50.25 \pm 1.2}$	$\mathbf{55.3 \pm 0.92}$
		PubMed					
	(α, β)	GCN	GCN-k ($\epsilon = 0.5$)	GCN-k ($\epsilon = 0.2$)	GCN-jk	GCNII	GCN-k-jk
Deletion	(0.0, 0.0)	76.50 ± 0.71	77.82 ± 0.86	77.80 ± 0.92	77.36 ± 0.74	78.14 ± 0.87	77.56 ± 1.62
	(0.2, 0.0)	74.80 ± 1.29	74.91 ± 1.33	$\mathbf{77.01 \pm 0.74}$	75.37 ± 0.94	75.83 ± 0.81	$\underline{76.67 \pm 1.20}$
	(0.5, 0.0)	73.92 ± 0.64	74.53 ± 0.97	$\mathbf{76.19 \pm 0.73}$	73.00 ± 0.93	75.00 ± 0.65	$\underline{75.60 \pm 1.09}$
Insertion	(0.0, 0.5)	57.89 ± 8.91	71.41 ± 0.91	$\mathbf{75.23 \pm 0.94}$	70.62 ± 0.97	72.01 ± 1.00	$\underline{73.71 \pm 1.10}$
	(0.0, 1.0)	46.98 ± 2.87	68.29 ± 7.93	72.42 ± 1.11	65.01 ± 1.70	$\underline{72.86 \pm 0.75}$	$\mathbf{73.20 \pm 0.89}$
Delet.+Insert.	(0.5, 0.5)	62.35 ± 1.01	65.19 ± 1.78	$\mathbf{70.79 \pm 1.08}$	65.07 ± 0.83	69.06 ± 0.67	$\underline{66.28 \pm 1.37}$
	(0.5, 1.0)	44.52 ± 0.88	55.64 ± 4.64	66.16 ± 1.57	65.20 ± 0.74	$\mathbf{70.22 \pm 0.65}$	$\underline{66.29 \pm 0.70}$
		CoraFull					
	(α, β)	GCN	GCN-k ($\epsilon = 0.5$)	GCN-k ($\epsilon = 0.2$)	GCN-jk	GCNII	GCN-k-jk
Deletion	(0.0, 0.0)	49.33 ± 0.61	55.27 ± 0.97	52.38 ± 1.62	$\mathbf{58.56 \pm 1.20}$	$\underline{56.83 \pm 0.67}$	56.10 ± 0.89
	(0.2, 0.0)	48.74 ± 1.34	50.71 ± 0.90	52.08 ± 1.26	$\mathbf{57.09 \pm 1.16}$	52.87 ± 1.06	$\underline{53.61 \pm 0.91}$
	(0.5, 0.0)	46.80 ± 1.35	45.33 ± 1.37	46.71 ± 1.12	51.42 ± 1.48	$\underline{51.57 \pm 1.03}$	$\mathbf{51.76 \pm 1.32}$
Insertion	(0.0, 0.5)	17.53 ± 3.15	24.48 ± 3.19	33.3 ± 2.78	49.47 ± 1.56	$\underline{51.19 \pm 0.94}$	$\mathbf{53.67 \pm 1.19}$
	(0.0, 1.0)	3.52 ± 0.76	9.60 ± 2.62	19.59 ± 2.42	42.07 ± 0.92	$\underline{48.79 \pm 1.28}$	$\mathbf{52.00 \pm 0.87}$
Delet.+Insert.	(0.5, 0.5)	7.10 ± 1.60	12.98 ± 1.56	22.06 ± 1.27	38.74 ± 0.86	$\underline{41.27 \pm 1.31}$	$\mathbf{45.74 \pm 1.22}$
	(0.5, 1.0)	3.69 ± 0.49	3.98 ± 1.15	5.33 ± 1.39	26.71 ± 0.77	$\underline{42.11 \pm 1.48}$	$\mathbf{43.23 \pm 1.33}$
		Photo					
	(α, β)	GCN	GCN-k ($\epsilon = 0.5$)	GCN-k ($\epsilon = 0.2$)	GCN-jk	GCNII	GCN-k-jk
Deletion	(0.0, 0.0)	87.67 ± 1.59	89.59 ± 0.63	88.64 ± 1.19	91.82 ± 0.58	$\underline{92.05 \pm 0.88}$	$\mathbf{92.42 \pm 0.58}$
	(0.2, 0.0)	90.35 ± 0.41	88.22 ± 0.77	85.66 ± 0.62	$\underline{91.86 \pm 0.70}$	$\mathbf{92.58 \pm 0.57}$	90.85 ± 0.59
	(0.5, 0.0)	88.04 ± 1.02	88.81 ± 0.33	87.23 ± 1.20	$\underline{90.44 \pm 1.07}$	87.65 ± 1.85	$\mathbf{91.45 \pm 0.52}$
Insertion	(0.0, 0.5)	35.12 ± 6.56	42.44 ± 11.1	47.5 ± 14.67	$\underline{83.34 \pm 2.74}$	$\mathbf{87.54 \pm 1.40}$	83.04 ± 4.58
	(0.0, 1.0)	27.75 ± 3.39	29.6 ± 3.22	24.05 ± 2.56	77.54 ± 3.44	$\mathbf{85.68 \pm 1.94}$	$\underline{81.89 \pm 3.24}$
Delet.+Insert.	(0.5, 0.5)	31.27 ± 4.58	27.65 ± 6.01	29.32 ± 4.17	$\underline{75.38 \pm 5.41}$	$\mathbf{87.32 \pm 1.11}$	73.57 ± 5.92
	(0.5, 1.0)	28.39 ± 2.12	24.89 ± 1.09	25.41 ± 4.25	62.97 ± 6.77	$\mathbf{81.10 \pm 1.22}$	$\underline{68.29 \pm 3.92}$

B.4.4 *Node Feature Noise*

We now provide further experiment results in Figure B.1 studying how node feature noise interacts with our proposed model. We observe that only when the node feature noise is five times larger than the graph structure noise, it

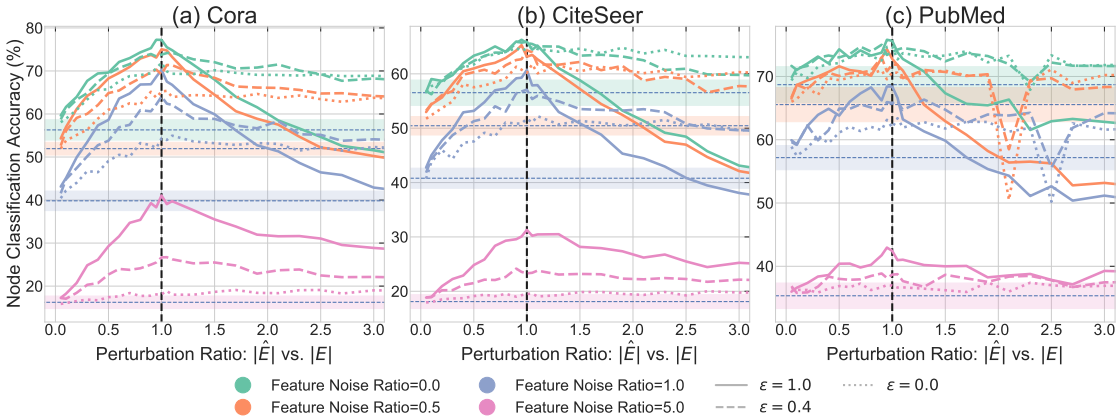

Figure B.1: Experiment Results with the co-appearance of graph structural noise and node feature noise on three citation datasets. The vertical line at a rate of 1 represents the performance on the unperturbed graph. On its left is the *edge deletion* case, with rate less than 1, where the most perturbed case corresponds to rate 0; on the right is the *edge insertion* case, where the perturbations grow with the rate. Different colours represent the extent of node feature perturbation.

starts to overshadow the benefit of adding the node feature kernel, as we can see from the pink curves, with-kernel (dashed or dotted line) performs worse than without-kernel (solid line). Otherwise, the performance of our kernel is robust and matches the observed trend repeatedly demonstrated in the previous experiments.

BIBLIOGRAPHY

Abadi, Martín et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: https://www.tensorflow.org/ (cit. on p. 22).

Achanta, Radhakrishna, Appu Shaji, Kevin Smith, Aurélien Lucchi, Pascal Fua, and Sabine Süsstrunk (2012). "SLIC Superpixels Compared to State-of-the-Art Superpixel Methods." In: *IEEE Trans. Pattern Anal. Mach. Intell.* 34.11, pp. 2274–2282. DOI: 10.1109/TPAMI.2012.120 (cit. on p. 1).

Airoldi, Edoardo M., David M. Blei, Stephen E. Fienberg, and Eric P. Xing (2008). "Mixed Membership Stochastic Blockmodels." In: *J. Mach. Learn. Res.* 9, pp. 1981–2014. URL: https://dl.acm.org/citation.cfm?id=1442798 (cit. on p. 66).

Albert, Réka and Albert-László Barabási (2002). "Statistical mechanics of complex networks." English. In: *Reviews of Modern Physics* 74.1, pp. 47–97. ISSN: 0034-6861. DOI: 10.1103/RevModPhys.74.47 (cit. on pp. 56, 66).

Alvarez-Hamelin, J. Ignacio, Luca Dall'Asta, Alain Barrat, and Alessandro Vespignani (2005). "k-core decomposition: a tool for the visualization of large scale networks." In: *CoRR* abs/cs/0504107. arXiv: cs/0504107. URL: http://arxiv.org/abs/cs/0504107 (cit. on p. 121).

Aronszajn, N. (1950). "Theory of Reproducing Kernels." In: *Transactions of the American Mathematical Society* 68.3, pp. 337–404. ISSN: 00029947. URL: http://www.jstor.org/stable/1990404 (cit. on p. 14).

Balcilar, Muhammet, Guillaume Renton, Pierre Héroux, Benoit Gaüzère, Sébastien Adam, and Paul Honeine (2021). "Analyzing the Expressive Power of Graph Neural Networks in a Spectral Perspective." In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. URL: https://openreview.net/forum?id=-qh0M9XWxnv (cit. on p. 21).

Barla, Annalisa, Francesca Odone, and Alessandro Verri (2003). "Histogram intersection kernel for image classification." In: *Proceedings of the 2003 International Conference on Image Processing, ICIP 2003, Barcelona, Catalonia, Spain, September 14-18, 2003*. IEEE, pp. 513–516. DOI: 10.1109/ICIP.2003.1247294 (cit. on p. 42).

Bietti, Alberto and Julien Mairal (2019). "On the Inductive Bias of Neural Tangent Kernels." In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, pp. 12873–12884. URL: https://proceedings.neurips.cc/paper/2019/hash/c4ef9c39b300931b69a36fb3dbb8d60e-Abstract.html (cit. on p. 121).

Blalock, Davis W., Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John V. Guttag (2020). "What is the State of Neural Network Pruning?" In: *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*. Ed. by Inderjit S. Dhillon, Dimitris S. Papailiopoulos, and Vivienne Sze. mlsys.org. URL: https://proceedings.mlsys.org/book/296.pdf (cit. on p. 75).

Bojchevski, Aleksandar, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann (2018). "NetGAN: Generating Graphs via Random Walks." In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by Jennifer G. Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 609–618. URL: http://proceedings.mlr.press/v80/bojchevski18a.html (cit. on p. 55).

Bölcskei, Helmut, Philipp Grohs, Gitta Kutyniok, and Philipp Petersen (2019). "Optimal Approximation with Sparsely Connected Deep Neural Networks." In: *SIAM J. Math. Data Sci.* 1.1, pp. 8–45. DOI: 10.1137/18M118709X (cit. on pp. 78, 107).

Borgwardt, Karsten M. and Hans-Peter Kriegel (2005). "Shortest-Path Kernels on Graphs." In: *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005), 27-30 November 2005, Houston, Texas, USA*. IEEE Computer Society, pp. 74–81. DOI: 10.1109/ICDM.2005.132 (cit. on pp. 15, 28, 45).

Borgwardt, Karsten M., Cheng Soon Ong, Stefan Schönauer, S. V. N. Vishwanathan, Alexander J. Smola, and Hans-Peter Kriegel (2005). "Protein function prediction via graph kernels." In: *Proceedings Thirteenth International Conference on Intelligent Systems for Molecular Biology 2005, Detroit, MI, USA, 25-29 June 2005*, pp. 47–56. DOI: 10.1093/bioinformatics/bti1007 (cit. on pp. 1, 23).

Bourely, Alfred, John Patrick Boueri, and Krzysztof Choromonski (2017). "Sparse Neural Networks Topologies." In: *CoRR* abs/1706.05683. arXiv: 1706.05683 (cit. on pp. 74, 79, 89).

Bruna, Joan, Wojciech Zaremba, Arthur Szlam, and Yann LeCun (2014). "Spectral Networks and Locally Connected Networks on Graphs." In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. URL: http://arxiv.org/abs/1312.6203 (cit. on p. 3).

Buchanan, Mark (2003). *Nexus: small worlds and the groundbreaking science of networks*. W. W. Norton & Company. ISBN: 978-0393324426. DOI: 10.5860/choice.40-3362. URL: https://books.google.fr/books?id=vdB83tHSZooC (cit. on p. 1).

Cerebras Systems, INC (2021). *Cerebras White Paper 3: Cerebras Systems: Achieving Industry Best AI Performance Through A Systems Approach*. https://cerebras.net/wp-content/uploads/2021/04/Cerebras-CS-2-Whitepaper.pdf. accessed May 2021 (cit. on p. 79).

Chang, Chih-Chung and Chih-Jen Lin (2011). "LIBSVM: A library for support vector machines." In: *ACM Trans. Intell. Syst. Technol.* 2.3, 27:1–27:27. DOI: 10.1145/1961189.1961199 (cit. on p. 45).

Chen, Dexiong, Laurent Jacob, and Julien Mairal (2019). "Recurrent Kernel Networks." In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, pp. 13431–13442. URL: https://proceedings.neurips.cc/paper/2019/hash/d60743aab4b625940d39b3b51c3c6a78-Abstract.html (cit. on p. 121).

Chen, Dexiong, Laurent Jacob, and Julien Mairal (2020). "Convolutional Kernel Networks for Graph-Structured Data." In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Vol. 119. Proceedings of Machine Learning Research. PMLR, pp. 1576–1586. URL: http://proceedings.mlr.press/v119/chen20h.html (cit. on pp. 121, 122).

Chen, Jie, Tengfei Ma, and Cao Xiao (2018). "FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling." In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=rytstxWAW (cit. on p. 120).

Chen, Ming, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li (2020). "Simple and Deep Graph Convolutional Networks." In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Vol. 119. Proceedings of Machine Learning Research. PMLR, pp. 1725–1735. URL: http://proceedings.mlr.press/v119/chen20v.html (cit. on pp. 75, 114).

Chen, Tianlong, Yongduo Sui, Xuxi Chen, Aston Zhang, and Zhangyang Wang (2021). "A Unified Lottery Ticket Hypothesis for Graph Neural Networks." In: *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, pp. 1695–1706. URL: http://proceedings.mlr.press/v139/chen21p.html (cit. on p. 120).

Chen, Tianqi, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang (2015). "MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems." In: *CoRR* abs/1512.01274. arXiv: 1512.01274. URL: http://arxiv.org/abs/1512.01274 (cit. on p. 22).

Chen, Zhengdao, Lisha Li, and Joan Bruna (2019). "Supervised Community Detection with Line Graph Neural Networks." In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. URL: https://openreview.net/forum?id=H1g0Z3A9Fm (cit. on p. 54).

Cho, Kyunghyun, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation." In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. Ed. by Alessandro Moschitti, Bo Pang, and Walter Daelemans. ACL, pp. 1724–1734. DOI: 10.3115/v1/d14-1179 (cit. on p. 59).

Conte, Donatello, Pasquale Foggia, Carlo Sansone, and Mario Vento (2004). "Thirty Years of Graph Matching in Pattern Recognition." In: *Int. J. Pattern Recognit. Artif. Intell.* 18.3, pp. 265–298. DOI: 10.1142/S0218001404003228 (cit. on p. 66).

Corso, Gabriele, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Velickovic (2020). "Principal Neighbourhood Aggregation for Graph Nets." In: *Advances in Neural Information Processing Systems 33: Annual Conference on*

Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual. Ed. by Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin. URL: https://proceedings.neurips.cc/paper/2020/hash/99cad265a1768cc2dd013f0e740300ae-Abstract.html (cit. on pp. 20, 81, 82, 93).

Csardi, Gabor and Tamas Nepusz (2006). "The igraph software package for complex network research." In: *InterJournal* Complex Systems, p. 1695. URL: http://igraph.sf.net (cit. on p. 22).

Dasoulas, George, Ludovic Dos Santos, Kevin Scaman, and Aladin Virmaux (2020). "Coloring Graph Neural Networks for Node Disambiguation." In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*. Ed. by Christian Bessiere. ijcai.org, pp. 2126–2132. DOI: 10.24963/ijcai.2020/294 (cit. on p. 94).

Dhillon, Inderjit S. and Dharmendra S. Modha (2001). "Concept Decompositions for Large Sparse Text Data Using Clustering." In: *Mach. Learn.* 42.1/2, pp. 143–175. DOI: 10.1023/A:1007612920971 (cit. on p. 34).

Donnat, Claire, Marinka Zitnik, David Hallac, and Jure Leskovec (2018). "Learning Structural Node Embeddings via Diffusion Wavelets." In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*. Ed. by Yike Guo and Faisal Farooq. ACM, pp. 1320–1329. DOI: 10.1145/3219819.3220025 (cit. on pp. 31, 48).

Du, Simon S., Kangcheng Hou, Ruslan Salakhutdinov, Barnabás Póczos, Ruosong Wang, and Keyulu Xu (2019). "Graph Neural Tangent Kernel: Fusing Graph Neural Networks with Graph Kernels." In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, pp. 5724–5734. URL: https://proceedings.neurips.cc/paper/2019/hash/663fd3c5144fd10bd5ca6611a9a5b92d-Abstract.html (cit. on p. 121).

Duvenaud, David, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams (2015). "Convolutional Networks on Graphs for Learning Molecular Fingerprints." In: *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*. Ed. by Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi

Sugiyama, and Roman Garnett, pp. 2224–2232. URL: https://proceedings.neurips.cc/paper/2015/hash/f9be311e65d81a9ad8150a60844bb94c-Abstract.html (cit. on pp. 3, 17).

Dwivedi, Vijay Prakash, Chaitanya K. Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson (2020). "Benchmarking Graph Neural Networks." In: *CoRR* abs/2003.00982. arXiv: 2003.00982 (cit. on p. 82).

El Karoui, Noureddine (2010). "The spectrum of kernel random matrices." English. In: *The Annals of Statistics* 38.1, pp. 1–50. ISSN: 0090-5364. DOI: 10.1214/08-AOS648 (cit. on p. 106).

Entezari, Negin, Saba A Al-Sayouri, Amirali Darvishzadeh, and Evangelos E Papalexakis (2020). "All you need is low (rank) defending against adversarial attacks on graphs." In: *Proceedings of the 13th International Conference on Web Search and Data Mining*, pp. 169–177 (cit. on p. 95).

Erdős, Pál and Alfréd Rényi (1960). "On the evolution of random graphs." English. In: *Publications of the Mathematical Institute of the Hungarian Academy of Sciences, Series A* 5, pp. 17–61 (cit. on pp. 55, 56, 66).

Fan, Wenqi, Yao Ma, Qing Li, Yuan He, Yihong Eric Zhao, Jiliang Tang, and Dawei Yin (2019). "Graph Neural Networks for Social Recommendation." In: *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*. Ed. by Ling Liu, Ryen W. White, Amin Mantrach, Fabrizio Silvestri, Julian J. McAuley, Ricardo Baeza-Yates, and Leila Zia. ACM, pp. 417–426. DOI: 10.1145/3308558.3313488 (cit. on p. 1).

Fan, Zhou and Zhichao Wang (2020). "Spectra of the Conjugate Kernel and Neural Tangent Kernel for linear-width neural networks." In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin. URL: https://proceedings.neurips.cc/paper/2020/hash/572201a4497b0b9f02d4f279b09ec30d-Abstract.html (cit. on pp. 100, 101).

Fey, Matthias and Jan Eric Lenssen (2019). "Fast Graph Representation Learning with PyTorch Geometric." In: *CoRR* abs/1903.02428. arXiv: 1903.02428 (cit. on pp. 22, 107).

Frankle, Jonathan and Michael Carbin (2019). "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks." In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

OpenReview.net. URL: https://openreview.net/forum?id=rJl-b3RcF7 (cit. on pp. 78, 120).

Frankle, Jonathan, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin (2021). "Pruning Neural Networks at Initialization: Why Are We Missing the Mark?" In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. URL: https://openreview.net/forum?id=Ig-VyQc-MLK (cit. on p. 75).

Fröhlich, Holger, Jörg K. Wegner, Florian Sieker, and Andreas Zell (2005). "Optimal assignment kernels for attributed molecular graphs." In: *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*. Ed. by Luc De Raedt and Stefan Wrobel. Vol. 119. ACM International Conference Proceeding Series. ACM, pp. 225–232. DOI: 10.1145/1102351.1102380 (cit. on p. 28).

Gama, Fernando, Alejandro Ribeiro, and Joan Bruna (2020). "Stability of Graph Neural Networks to Relative Perturbations." In: *2020 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2020, Barcelona, Spain, May 4-8, 2020*. IEEE, pp. 9070–9074. DOI: 10.1109/ICASSP40776.2020.9054341 (cit. on p. 79).

Gascon, Hugo, Fabian Yamaguchi, Daniel Arp, and Konrad Rieck (2013). "Structural detection of android malware using embedded call graphs." In: *AISec'13, Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security, Co-located with CCS 2013, Berlin, Germany, November 4, 2013*. Ed. by Ahmad-Reza Sadeghi, Blaine Nelson, Christos Dimitrakakis, and Elaine Shi. ACM, pp. 45–54. DOI: 10.1145/2517312.2517315 (cit. on p. 27).

Geisler, Simon, Daniel Zügner, and Stephan Günnemann (2020). "Reliable graph neural networks via robust aggregation." In: *Advances in Neural Information Processing Systems* 33, pp. 13272–13284 (cit. on p. 95).

Gilmer, Justin, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl (2017). "Neural Message Passing for Quantum Chemistry." In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, pp. 1263–1272. URL: http://proceedings.mlr.press/v70/gilmer17a.html (cit. on pp. 54, 72).

Goodfellow, Ian (2016). *Deep learning*. Ed. by Yoshua Bengio and Aaron Courville. Adaptive computation and machine learning. Weitere Infos unter http://www.deeplearningbo

Cambridge, Massachusetts: The MIT Press. xxii, 775 Seiten. ISBN: 9780262035613 (cit. on p. 2).

Goyal, Palash, Nitin Kamra, Xinran He, and Yan Liu (2018). "DynGEM: Deep Embedding Method for Dynamic Graphs." In: *CoRR* abs/1805.11273. arXiv: 1805.11273 (cit. on pp. 54–56).

Greene, Derek and Padraig Cunningham (2006). "Practical solutions to the problem of diagonal dominance in kernel document clustering." In: *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*. Ed. by William W. Cohen and Andrew W. Moore. Vol. 148. ACM International Conference Proceeding Series. ACM, pp. 377–384. DOI: 10.1145/1143844.1143892 (cit. on p. 50).

Griffa, Alessandra, Benjamin Ricaud, Kirell Benzi, Xavier Bresson, Alessandro Daducci, Pierre Vandergheynst, Jean-Philippe Thiran, and Patric Hagmann (2017). "Transient networks of spatio-temporal connectivity map communication pathways in brain functional systems." In: *NeuroImage* 155, pp. 490–502. DOI: 10.1016/j.neuroimage.2017.04.015 (cit. on p. 17).

Grover, Aditya and Jure Leskovec (2016). "node2vec: Scalable Feature Learning for Networks." In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*. Ed. by Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi. ACM, pp. 855–864. DOI: 10.1145/2939672.2939754 (cit. on pp. 31, 43, 44, 48).

Grover, Aditya, Aaron Zweig, and Stefano Ermon (2019). "Graphite: Iterative Generative Modeling of Graphs." In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 2434–2444. URL: http://proceedings.mlr.press/v97/grover19a.html (cit. on p. 55).

Günnemann, Stephan (2022). "Graph Neural Networks: Adversarial Robustness." In: *Graph Neural Networks: Foundations, Frontiers, and Applications*. Ed. by Lingfei Wu, Peng Cui, Jian Pei, and Liang Zhao. Springer. Chap. 8, pp. 149–176 (cit. on p. 95).

Hachem, Walid, Philippe Loubaton, and Jamal Najim (2007). "Deterministic equivalents for certain functionals of large random matrices." English. In: *The Annals of Applied Probability* 17.3, pp. 875–930. ISSN: 1050-5164. DOI: 10.1214/105051606000000925 (cit. on pp. 99, 102).

Hagberg, Aric A., Daniel A. Schult, and Pieter J. Swart (2008). "Exploring Network Structure, Dynamics, and Function using NetworkX." In: *Proceedings of the 7th Python in Science Conference*. Ed. by Gaël Varoquaux, Travis Vaught, and Jarrod Millman. Pasadena, CA USA, pp. 11 –15 (cit. on p. 22).

Hamilton, William L., Zhitao Ying, and Jure Leskovec (2017). "Inductive Representation Learning on Large Graphs." In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, pp. 1024–1034. URL: https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7ebea9-Abstract.html (cit. on pp. 20, 82, 93, 120, 138).

Han, Song, Jeff Pool, John Tran, and William J. Dally (2015). "Learning both Weights and Connections for Efficient Neural Network." In: *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*. Ed. by Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, pp. 1135–1143. URL: https://proceedings.neurips.cc/paper/2015/hash/ae0eb3eed39d2bcef4622b2499a05fe6-Abstract.html (cit. on p. 78).

Harris, Charles R. et al. (Sept. 2020). "Array programming with NumPy." In: *Nature* 585.7825, pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2 (cit. on p. 22).

Haussler, David (1999). *Convolution Kernels on Discrete Structures*. Tech. rep. Department of Computer Science, University of California at Santa Cruz. DOI: 10.1.1.110.638 (cit. on pp. 2, 28).

He, Xiangnan, Kuan Deng, Xiang Wang, Yan Li, Yong-Dong Zhang, and Meng Wang (2020). "LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation." In: *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*. Ed. by Jimmy Huang, Yi Chang, Xueqi Cheng, Jaap Kamps, Vanessa Murdock, Ji-Rong Wen, and Yiqun Liu. ACM, pp. 639–648. DOI: 10.1145/3397271.3401063 (cit. on p. 75).

Henaff, Mikael, Joan Bruna, and Yann LeCun (2015). "Deep Convolutional Networks on Graph-Structured Data." In: *CoRR* abs/1506.05163. arXiv: 1506.05163. URL: http://arxiv.org/abs/1506.05163 (cit. on p. 3).

Hermansson, Linus, Tommi Kerola, Fredrik Johansson, Vinay Jethava, and Devdatt P. Dubhashi (2013). "Entity disambiguation in anonymized graphs using graph kernels." In: *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*. Ed. by Qi He, Arun Iyengar, Wolfgang Nejdl, Jian Pei, and Rajeev Rastogi. ACM, pp. 1037–1046. DOI: 10.1145/2505515.2505565 (cit. on p. 27).

Hermsdorff, Gecia Bravo and Lee M. Gunderson (2019). "A Unifying Framework for Spectrum-Preserving Graph Sparsification and Coarsening." In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, pp. 7734–7745. URL: https://proceedings.neurips.cc/paper/2019/hash/cd474f6341aeffd65f93084d0dae3453-Abstract.html (cit. on p. 121).

Hoefler, Torsten, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste (2021). "Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks." In: *CoRR* abs/2102.00554. arXiv: 2102.00554 (cit. on p. 75).

Holme, Petter and Beom Jun Kim (2002). "Growing scale-free networks with tunable clustering." In: *Phys. Rev. E* 65. ISSN: 1063-651X. DOI: 10.1103/physreve.65.026107 (cit. on p. 66).

Hoory, Shlomo, Nathan Linial, and Avi Widgerson (2006). "Expander graphs and their applications." English. In: *Bulletin of the American Mathematical Society. New Series* 43.4, pp. 439–561. ISSN: 0273-0979. DOI: 10.1090/S0273-0979-06-01126-8 (cit. on pp. 11, 78).

Hu, Weihua, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec (2020). "Open Graph Benchmark: Datasets for Machine Learning on Graphs." In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin. URL: https://proceedings.neurips.cc/paper/2020/hash/fb60d411a5c5b72b2e7d3527cfc84fd0-Abstract.html (cit. on pp. 23, 82).

Hunter, J. D. (2007). "Matplotlib: A 2D graphics environment." In: *Computing in Science & Engineering* 9.3, pp. 90–95. DOI: 10.1109/MCSE.2007.55 (cit. on p. 22).

Irwin, John J., Teague Sterling, Michael M. Mysinger, Erin S. Bolstad, and Ryan G. Coleman (2012). "ZINC: A Free Tool to Discover Chemistry for Biology." In: *J. Chem. Inf. Model.* 52.7, pp. 1757–1768. DOI: 10.1021/ci3001277 (cit. on p. 82).

Ismagilov, R. S. (1997). "Ultrametric spaces and related Hilbert spaces." English. In: *Mathematical Notes* 62.2, pp. 186–197. ISSN: 0001-4346. DOI: 10.1007/BF02355907 (cit. on p. 33).

Jacot, Arthur, Clément Hongler, and Franck Gabriel (2018). "Neural Tangent Kernel: Convergence and Generalization in Neural Networks." In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, pp. 8580–8589. URL: https://proceedings.neurips.cc/paper/2018/hash/5a4be1fa34e62bb8a6ec6b91d2462f5a-Abstract.html (cit. on p. 121).

Jin, Ming, Heng Chang, Wenwu Zhu, and Somayeh Sojoudi (2021). "Power up! Robust Graph Convolutional Network via Graph Powering." In: *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021,Virtual Event, February 2-9, 2021*. AAAI Press, pp. 8004–8012. URL: https://ojs.aaai.org/index.php/AAAI/article/view/16976 (cit. on p. 95).

Jin, Wei, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang (2020). "Graph structure learning for robust graph neural networks." In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 66–74 (cit. on p. 95).

Karrer, Brian and M. E. J. Newman (Jan. 2011). "Stochastic blockmodels and community structure in networks." In: *Physical Review E* 83.1, 016107, p. 016107. DOI: 10.1103/PhysRevE.83.016107. arXiv: 1008.3926 [physics.soc-ph]. URL: https://ui.adsabs.harvard.edu/abs/2011PhRvE..83a6107K (cit. on p. 97).

Kepner, Jeremy and Ryan A. Robinett (2019). "RadiX-Net: Structured Sparse Matrices for Deep Neural Networks." In: *IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2019, Rio de Janeiro, Brazil, May 20-24, 2019*. IEEE, pp. 268–274. DOI: 10.1109/IPDPSW.2019.00051 (cit. on pp. 74, 79).

Kersting, Kristian, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann (2016). *Benchmark Data Sets for Graph Kernels*. http://graphkernels.cs.tu-dortmund.de. URL: http://graphkernels.cs.tu-dortmund.de (cit. on p. 82).

Kingma, Diederik P. and Jimmy Ba (2015). "Adam: A Method for Stochastic Optimization." In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. arXiv: 1412.6980 (cit. on p. 107).

Kipf, Thomas N. and Max Welling (2017). "Semi-Supervised Classification with Graph Convolutional Networks." In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=SJU4ayYgl (cit. on pp. 3, 19, 73, 82, 93).

Knyazev, Boris, Graham W. Taylor, and Mohamed R. Amer (2019). "Understanding Attention and Generalization in Graph Neural Networks." In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, pp. 4204–4214. URL: https://proceedings.neurips.cc/paper/2019/hash/4c5bcfec8584af0d967f1ab10179ca4b-Abstract.html (cit. on pp. 1, 23, 82).

Kondor, Risi and John D. Lafferty (2002). "Diffusion Kernels on Graphs and Other Discrete Input Spaces." In: *Machine Learning, Proceedings of the Nineteenth International Conference (ICML 2002), University of New South Wales, Sydney, Australia, July 8-12, 2002*. Ed. by Claude Sammut and Achim G. Hoffmann. Morgan Kaufmann, pp. 315–322 (cit. on p. 2).

Kondor, Risi, Hy Truong Son, Horace Pan, Brandon M. Anderson, and Shubhendu Trivedi (2018). "Covariant Compositional Networks For Learning Graphs." In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=SkIv3MAUf (cit. on p. 27).

Kriege, Nils M., Pierre-Louis Giscard, and Richard C. Wilson (2016). "On Valid Optimal Assignment Kernels and Applications to Graph Classification." In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. Ed. by Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, pp. 1615–1623. URL: https://proceedings.neurips.cc/paper/2016/hash/0efe32849d230d7f53049ddc4a4b0c60-Abstract.html (cit. on pp. 28, 29, 31, 32, 41, 45).

Kriege, Nils M. and Petra Mutzel (2012). "Subgraph Matching Kernels for Attributed Graphs." In: *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. icml.cc / Omnipress. URL: http://icml.cc/2012/papers/542.pdf (cit. on p. 1).

Kumar, Srijan, Bryan Hooi, Disha Makhija, Mohit Kumar, Christos Faloutsos, and V. S. Subrahmanian (2018). "REV2: Fraudulent User Prediction in Rating Platforms." In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM 2018, Marina Del Rey, CA, USA, February 5-9, 2018*. Ed. by Yi Chang, Chengxiang Zhai, Yan Liu, and Yoelle Maarek. ACM, pp. 333–341. DOI: 10.1145/3159652.3159729 (cit. on p. 64).

Kumar, Srijan, Francesca Spezzano, V. S. Subrahmanian, and Christos Faloutsos (2016). "Edge Weight Prediction in Weighted Signed Networks." In: *IEEE 16th International Conference on Data Mining, ICDM 2016, December 12-15, 2016, Barcelona, Spain*. Ed. by Francesco Bonchi, Josep Domingo-Ferrer, Ricardo Baeza-Yates, Zhi-Hua Zhou, and Xindong Wu. IEEE Computer Society, pp. 221–230. DOI: 10.1109/ICDM.2016.0033 (cit. on p. 64).

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning." In: *Nature* 521, pp. 436–444. ISSN: 0028-0836. DOI: 10.1038/nature14539 (cit. on p. 1).

Ledoux, Michel (2001). *The concentration of measure phenomenon*. Mathematical surveys and monographs v. 89. Includes bibliographical references (p. 171-179) and index. Providence, R.I: American Mathematical Society. x, 181 p. ISBN: 0821828649 (cit. on p. 97).

Lee, Namhoon, Thalaiyasingam Ajanthan, and Philip H. S. Torr (2019). "Snip: single-Shot Network Pruning based on Connection sensitivity." In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. URL: https://openreview.net/forum?id=B1VZqjAcYX (cit. on p. 75).

Leskovec, Jure, Deepayan Chakrabarti, Jon M. Kleinberg, Christos Faloutsos, and Zoubin Ghahramani (2010). "Kronecker Graphs: An Approach to Modeling Networks." In: *J. Mach. Learn. Res.* 11, pp. 985–1042. URL: https://dl.acm.org/citation.cfm?id=1756039 (cit. on pp. 5, 56, 66).

Leskovec, Jure, Jon M. Kleinberg, and Christos Faloutsos (2005). "Graphs over time: densification laws, shrinking diameters and possible explanations." In: *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, Illinois, USA, August 21-24, 2005*. Ed. by

Robert Grossman, Roberto J. Bayardo, and Kristin P. Bennett. ACM, pp. 177–187. DOI: 10.1145/1081870.1081893 (cit. on p. 5).

Leskovec, Jure, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney (2009). "Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters." In: *Internet Math.* 6.1, pp. 29–123. DOI: 10.1080/15427951.2009.10129177 (cit. on p. 1).

Leskovec, Jure, Anand Rajaraman, and Jeffrey D. Ullman (2014). *Mining of Massive Datasets, 2nd Ed*. Cambridge University Press. ISBN: 978-1107077232. URL: http://www.mmds.org/ (cit. on p. 51).

Li, Cheng, Xiaoxiao Guo, and Qiaozhu Mei (2016). "DeepGraph: Graph Structure Predicts Network Growth." In: *CoRR* abs/1610.06251. arXiv: 1610.06251 (cit. on pp. 54, 55).

Li, Qimai, Zhichao Han, and Xiao-Ming Wu (2018). "Deeper Insights Into Graph Convolutional Networks for Semi-Supervised Learning." In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*. Ed. by Sheila A. McIlraith and Kilian Q. Weinberger. AAAI Press, pp. 3538–3545. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16098 (cit. on p. 98).

Li, Xin and Dan Roth (2002). "Learning Question Classifiers." In: *19th International Conference on Computational Linguistics, COLING 2002, Howard International House and Academia Sinica, Taipei, Taiwan, August 24 - September 1, 2002*. URL: https://aclanthology.org/C02-1150/ (cit. on p. 50).

Li, Yujia, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel (2016). "Gated Graph Sequence Neural Networks." In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. arXiv: 1511.05493 (cit. on p. 60).

Louart, Cosme and Romain Couillet (May 2018). "Concentration of Measure and Large Random Matrices with an application to Sample Covariance Matrices." In: arXiv: 1805.08295 [math.PR] (cit. on pp. 97, 102, 133).

Louart, Cosme, Zhenyu Liao, and Romain Couillet (2018). "A random matrix approach to neural networks." English. In: *The Annals of Applied Probability* 28.2, pp. 1190–1248. ISSN: 1050-5164. DOI: 10.1214/17-AAP1328 (cit. on pp. 99, 100).

Lubotzky, Alexander (2012). "Expander graphs in pure and applied mathematics." English. In: *Bulletin of the American Mathematical Society. New Series* 49.1,

pp. 113–162. ISSN: 0273-0979. DOI: 10.1090/S0273-0979-2011-01359-3 (cit. on pp. 11, 78).

Lü, Linyuan and Tao Zhou (Mar. 2011). "Link prediction in complex networks: A survey." In: *Physica A: Statistical Mechanics and its Applications* 390. DOI: 10.1016/j.physa.2010.11.027 (cit. on p. 1).

Mairal, Julien (2016). "End-to-End Kernel Learning with Supervised Convolutional Kernel Networks." In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. Ed. by Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, pp. 1399–1407. URL: https://proceedings.neurips.cc/paper/2016/hash/fc8001f834f6a5f0561080d134d53d29-Abstract.html (cit. on p. 121).

Mallat, Stéphane (2009). *A Wavelet Tour of Signal Processing - The Sparse Way*. 3rd. Academic Press. ISBN: 978-0-12-374370-1. URL: https://www.elsevier.com/books/a-wavelet-tour-of-signal-processing/mallat/978-0-12-374370-1 (cit. on p. 21).

Manessi, Franco, Alessandro Rozza, and Mario Manzo (2020). "Dynamic graph convolutional networks." In: *Pattern Recognit.* 97. DOI: 10.1016/j.patcog.2019.107000 (cit. on p. 56).

Maron, Haggai, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman (2019). "Provably Powerful Graph Networks." In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, pp. 2153–2164. URL: https://proceedings.neurips.cc/paper/2019/hash/bb04af0f7ecaee4aae62035497da1387-Abstract.html (cit. on p. 94).

McDonald, Andrew W. E. and Ali Shokoufandeh (2019). "Sparse Super-Regular Networks." In: *18th IEEE International Conference On Machine Learning And Applications, ICMLA 2019, Boca Raton, FL, USA, December 16-19, 2019*. Ed. by M. Arif Wani, Taghi M. Khoshgoftaar, Dingding Wang, Huanjing Wang, and Naeem Seliya. IEEE, pp. 1764–1770. DOI: 10.1109/ICMLA.2019.00286 (cit. on p. 74).

Meng, Changping, S. Chandra Mouli, Bruno Ribeiro, and Jennifer Neville (2018). "Subgraph Pattern Neural Networks for High-Order Graph Evolution Prediction." In: *Proceedings of the Thirty-Second AAAI Conference on Artificial In-*

telligence, (AAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018. Ed. by Sheila A. McIlraith and Kilian Q. Weinberger. AAAI Press, pp. 3778–3787. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16941 (cit. on p. 55).

Mialon, Grégoire, Dexiong Chen, Alexandre d'Aspremont, and Julien Mairal (2021). "A Trainable Optimal Transport Embedding for Feature Aggregation and its Relationship to Attention." In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. URL: https://openreview.net/forum?id=ZK6vTvb84s (cit. on p. 122).

Mikolov, Tomás, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean (2013). "Distributed Representations of Words and Phrases and their Compositionality." In: *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*. Ed. by Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, pp. 3111–3119. URL: https://proceedings.neurips.cc/paper/2013/hash/9aa42b31882ec039965f3c4923ce901b-Abstract.html (cit. on p. 50).

Monti, Federico, Fabrizio Frasca, Davide Eynard, Damon Mannion, and Michael M. Bronstein (2019). "Fake News Detection on Social Media using Geometric Deep Learning." In: *CoRR* abs/1902.06673. arXiv: 1902.06673 (cit. on p. 17).

Moor Insights and Strategy (2020). *Graphcore White Paper: The Graphcore Second Generation IPU*. https://www.graphcore.ai/hubfs/MK2-%20The%20Graphcore%202nd%20Ge accessed May 2021. URL: https://www.graphcore.ai/hubfs/MK2-\%The\%Graphcore\%2nd\%Generation\%IPU\%Final\%v7.14.2020.pdf?hsLang=en (cit. on p. 79).

Morris, Christopher, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann (2020). "TUDataset: A collection of benchmark datasets for learning with graphs." In: *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*. arXiv: 2007.08663. URL: www.graphlearning.io (cit. on p. 22).

Morris, Christopher, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe (2019). "Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks." In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu,*

Hawaii, USA, January 27 - February 1, 2019. AAAI Press, pp. 4602–4609. DOI: 10.1609/aaai.v33i01.33014602 (cit. on pp. 54, 94).

Mrabah, Nairouz, Mohamed Bouguessa, Mohamed Fawzi Touati, and Riadh Ksantini (2021). "Rethinking Graph Auto-Encoder Models for Attributed Graph Clustering." In: *CoRR* abs/2107.08562. arXiv: 2107.08562. URL: https://arxiv.org/abs/2107.08562 (cit. on p. 122).

NT, Hoang and Takanori Maehara (2019). "Revisiting Graph Neural Networks: All We Have is Low-Pass Filters." In: *CoRR* abs/1905.09550. arXiv: 1905.09550. URL: http://arxiv.org/abs/1905.09550 (cit. on p. 21).

NVIDIA (2020). *NVIDIA White Paper: NVIDIA A100 Tensor Core GPU Architecture*. https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf. accessed May 2021. URL: https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf (cit. on p. 79).

Nguyen, Giang Hoang, John Boaz Lee, Ryan A. Rossi, Nesreen K. Ahmed, Eunyee Koh, and Sungchul Kim (2018). "Continuous-Time Dynamic Network Embeddings." In: *Companion of the The Web Conference 2018 on The Web Conference 2018, WWW 2018, Lyon , France, April 23-27, 2018*. Ed. by Pierre-Antoine Champin, Fabien Gandon, Mounia Lalmas, and Panagiotis G. Ipeirotis. ACM, pp. 969–976. DOI: 10.1145/3184558.3191526 (cit. on pp. 54, 56).

Niepert, Mathias, Mohamed Ahmed, and Konstantin Kutzkov (2016). "Learning Convolutional Neural Networks for Graphs." In: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*. Ed. by Maria-Florina Balcan and Kilian Q. Weinberger. Vol. 48. JMLR Workshop and Conference Proceedings. JMLR.org, pp. 2014–2023. URL: http://proceedings.mlr.press/v48/niepert16.html (cit. on p. 27).

Nikolentzos, Giannis, Polykarpos Meladianos, Antoine Jean-Pierre Tixier, Konstantinos Skianis, and Michalis Vazirgiannis (2018). "Kernel Graph Convolutional Neural Networks." In: *Artificial Neural Networks and Machine Learning - ICANN 2018 - 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part I*. Ed. by Vera Kurková, Yannis Manolopoulos, Barbara Hammer, Lazaros S. Iliadis, and Ilias Maglogiannis. Vol. 11139. Lecture Notes in Computer Science. Springer, pp. 22–32. DOI: 10.1007/978-3-030-01418-6_3. URL: https://doi.org/10.1007/978-3-030-01418-6_3 (cit. on pp. 121, 122).

Nikolentzos, Giannis, Polykarpos Meladianos, and Michalis Vazirgiannis (2017). "Matching Node Embeddings for Graph Similarity." In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*. Ed. by Satinder P. Singh and Shaul Markovitch. AAAI Press, pp. 2429–2435. URL: http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14494 (cit. on pp. 28, 36, 45).

Nikolentzos, Giannis, Giannis Siglidis, and Michalis Vazirgiannis (2019). "Graph Kernels: A Survey." In: *CoRR* abs/1904.12218. arXiv: 1904.12218 (cit. on pp. 15, 66).

Nikolentzos, Giannis and Michalis Vazirgiannis (2020). "Random Walk Graph Neural Networks." In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin. URL: https://proceedings.neurips.cc/paper/2020/hash/ba95d78a7c942571185308775a97a3a0-Abstract.html (cit. on pp. 121, 122).

Opsahl, Tore (2013). "Triadic closure in two-mode networks: Redefining the global and local clustering coefficients." In: *Soc. Networks* 35.2, pp. 159–167. DOI: 10.1016/j.socnet.2011.07.001 (cit. on p. 65).

Opsahl, Tore and Pietro Panzarasa (2009). "Clustering in weighted networks." In: *Soc. Networks* 31.2, pp. 155–163. DOI: 10.1016/j.socnet.2009.02.002 (cit. on p. 65).

Page, Lawrence, Sergey Brin, Rajeev Motwani, and Terry Winograd (1999). *The pagerank citation ranking: bringing order to the web*. Tech. rep. Stanford InfoLab. DOI: 10.1.1.31.1768 (cit. on p. 2).

Pang, Bo and Lillian Lee (2004). "A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts." In: *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics, 21-26 July, 2004, Barcelona, Spain*. Ed. by Donia Scott, Walter Daelemans, and Marilyn A. Walker. ACL, pp. 271–278. DOI: 10.3115/1218955.1218990. URL: https://aclanthology.org/P04-1035/ (cit. on p. 50).

Paranjape, Ashwin, Austin R. Benson, and Jure Leskovec (2017). "Motifs in Temporal Networks." In: *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, WSDM 2017, Cambridge, United Kingdom, February 6-10, 2017*. Ed. by Maarten de Rijke, Milad Shokouhi, Andrew Tomkins, and Min Zhang. ACM, pp. 601–610. DOI: 10.1145/3018661.3018731 (cit. on p. 65).

Pareja, Aldo, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B. Schardl, and Charles E. Leiserson (2020). "EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs." In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, pp. 5363–5370. URL: https://aaai.org/ojs/index.php/AAAI/article/view/5984 (cit. on pp. 55, 56).

Paszke, Adam et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, pp. 8024–8035. URL: https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html (cit. on p. 22).

Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12, pp. 2825–2830 (cit. on pp. 22, 46).

Pei, Hongbin, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang (2020). "Geom-gcn: Geometric graph convolutional networks." In: *International Conference on Learning Representations (ICLR)* (cit. on p. 96).

Prabhu, Ameya, Girish Varma, and Anoop M. Namboodiri (2018). "Deep Expander Networks: Efficient Deep Networks from Graph Theory." In: *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XIII*. Ed. by Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss. Vol. 11217. Lecture Notes in Computer Science. Springer, pp. 20–36. DOI: 10.1007/978-3-030-01261-8_2 (cit. on pp. 73, 74, 77).

Qiu, Jiezhong, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang (2018). "Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and node2vec." In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM 2018, Marina Del Rey, CA, USA, February 5-9, 2018*. Ed. by Yi Chang, Chengxiang Zhai, Yan Liu, and Yoelle Maarek. ACM, pp. 459–467. DOI: 10.1145/3159652.3159706 (cit. on p. 30).

Rahman, Mahmudur, Mansurul Alam Bhuiyan, Mahmuda Rahman, and Mohammad Al Hasan (2014). "GUISE: a uniform sampler for constructing fre-

quency histogram of graphlets." In: *Knowl. Inf. Syst.* 38.3, pp. 511–536. DOI: 10.1007/s10115-013-0673-3 (cit. on p. 10).

Ramon, Jan and Thomas Gärtner (2003). "Expressivity versus Efficiency of Graph Kernels." In: *Proceedings of the First International Workshop on Mining Graphs, Trees and Sequences*, pp. 65–74. DOI: 10.1.1.132.692 (cit. on p. 28).

Ribeiro, Leonardo Filipe Rodrigues, Pedro H. P. Saverese, and Daniel R. Figueiredo (2017). "*struc2vec*: Learning Node Representations from Structural Identity." In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*. ACM, pp. 385–394. DOI: 10.1145/3097983.3098061 (cit. on p. 31).

Rossi, Ryan A. and Nesreen K. Ahmed (2015). "The Network Data Repository with Interactive Graph Analytics and Visualization." In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*. Ed. by Blai Bonet and Sven Koenig. AAAI Press, pp. 4292–4293. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9553 (cit. on p. 65).

Rousseau, François and Michalis Vazirgiannis (2013). "Graph-of-word and TW-IDF: new approach to ad hoc IR." In: *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*. Ed. by Qi He, Arun Iyengar, Wolfgang Nejdl, Jian Pei, and Rajeev Rastogi. ACM, pp. 59–68. DOI: 10.1145/2505515.2505671 (cit. on pp. 1, 2).

Salha, Guillaume, Romain Hennequin, and Michalis Vazirgiannis (2019). "Keep It Simple: Graph Autoencoders Without Graph Convolutional Networks." In: *CoRR* abs/1910.00942. arXiv: 1910.00942 (cit. on p. 75).

Salha, Guillaume, Romain Hennequin, and Michalis Vazirgiannis (2020). "Simple and Effective Graph Autoencoders with One-Hop Linear Models." In: *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2020, Ghent, Belgium, September 14-18, 2020, Proceedings, Part I*. Ed. by Frank Hutter, Kristian Kersting, Jefrey Lijffijt, and Isabel Valera. Vol. 12457. Lecture Notes in Computer Science. Springer, pp. 319–334. DOI: 10.1007/978-3-030-67658-2_19 (cit. on p. 75).

Sanders, Niek J. (2011). *Twitter Sentiment Corpus*. Sanders Analytics LLC (cit. on p. 50).

Schölkopf, Bernhard, Koji Tsuda, and Jean-Philippe Vert (2004). *Kernel Methods in Computational Biology*. MIT press (cit. on p. 27).

Seddik, Mohamed El Amine, Cosme Louart, Mohamed Tamaazousti, and Romain Couillet (2020). "Random Matrix Theory Proves that Deep Learning Representations of GAN-data Behave as Gaussian Mixtures." In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Vol. 119. Proceedings of Machine Learning Research. PMLR, pp. 8573–8582. URL: http://proceedings.mlr.press/v119/seddik20a.html (cit. on p. 97).

Sen, Prithviraj, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad (2008). "Collective Classification in Network Data." In: *AI Mag.* 29.3, pp. 93–106. DOI: 10.1609/aimag.v29i3.2157 (cit. on pp. 82, 107).

Seo, Youngjoo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson (2018). "Structured Sequence Modeling with Graph Convolutional Recurrent Networks." In: *Neural Information Processing - 25th International Conference, ICONIP 2018, Siem Reap, Cambodia, December 13-16, 2018, Proceedings, Part I*. Ed. by Long Cheng, Andrew Chi-Sing Leung, and Seiichi Ozawa. Vol. 11301. Lecture Notes in Computer Science. Springer, pp. 362–373. DOI: 10.1007/978-3-030-04167-0_33 (cit. on pp. 55, 56).

Shchur, Oleksandr, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann (2018). "Pitfalls of Graph Neural Network Evaluation." In: *CoRR* abs/1811.05868. arXiv: 1811.05868 (cit. on pp. 23, 107, 138).

Shen, Zebang, Zhenfu Wang, Alejandro Ribeiro, and Hamed Hassani (2020). "Sinkhorn Barycenter via Functional Gradient Descent." In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin. URL: https://proceedings.neurips.cc/paper/2020/hash/0a93091da5efb0d9d5649e7f6b2ad9d7-Abstract.html (cit. on p. 122).

Shervashidze, Nino, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt (2011). "Weisfeiler-Lehman Graph Kernels." In: *J. Mach. Learn. Res.* 12, pp. 2539–2561. URL: http://dl.acm.org/citation.cfm?id=2078187 (cit. on pp. 16, 45, 55, 67).

Shervashidze, Nino, S. V. N. Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten M. Borgwardt (2009). "Efficient graphlet kernels for large graph comparison." In: *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, AISTATS 2009, Clearwater Beach, Florida, USA, April 16-18, 2009*. Ed. by David A. Van Dyk and Max Welling. Vol. 5. JMLR Pro-

ceedings. JMLR.org, pp. 488–495. URL: http://proceedings.mlr.press/v5/shervashidze09a.html (cit. on pp. 16, 28, 45).

Silverstein, Jack W. and Z. D. Bai (1995). "On the empirical distribution of eigenvalues of a class of large dimensional random matrices." English. In: *Journal of Multivariate Analysis* 54.2, pp. 175–192. ISSN: 0047-259X. DOI: 10.1006/jmva.1995.1051 (cit. on p. 134).

Sun, Lichao, Ji Wang, Philip S. Yu, and Bo Li (2018). "Adversarial Attack and Defense on Graph Data: A Survey." In: *CoRR* abs/1812.10528. arXiv: 1812.10528 (cit. on pp. 94, 95).

Swain, Michael J. and Dana H. Ballard (1991). "Color indexing." In: *Int. J. Comput. Vis.* 7.1, pp. 11–32. DOI: 10.1007/BF00130487 (cit. on p. 42).

Tanaka, Hidenori, Daniel Kunin, Daniel L. Yamins, and Surya Ganguli (2020). "Pruning neural networks without any data by iteratively conserving synaptic flow." In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin. URL: https://proceedings.neurips.cc/paper/2020/hash/46a4378f835dc8040c8057beb6a2da52-Abstract.html (cit. on pp. 75, 77).

Tang, Jie, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su (2008). "ArnetMiner: extraction and mining of academic social networks." In: *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008*. Ed. by Ying Li, Bing Liu, and Sunita Sarawagi. ACM, pp. 990–998. DOI: 10.1145/1401890.1402008 (cit. on p. 1).

Velickovic, Petar, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio (2018). "Graph Attention Networks." In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=rJXMpikCZ (cit. on p. 138).

Vert, Jean-Philippe (2008). "The optimal assignment kernel is not positive definite." In: *CoRR* abs/0801.4061. arXiv: 0801.4061 (cit. on pp. 17, 28).

Vinyals, Oriol, Samy Bengio, and Manjunath Kudlur (2016). "Order Matters: Sequence to sequence for sets." In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track*

Proceedings. Ed. by Yoshua Bengio and Yann LeCun. arXiv: 1511.06391 (cit. on p. 60).

Wang, Chaoqi, Guodong Zhang, and Roger B. Grosse (2020). "Picking Winning Tickets Before Training by Preserving Gradient Flow." In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. URL: https://openreview.net/forum?id=SkgsACVKPH (cit. on p. 75).

Wang, Kuansan, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia (2020). "Microsoft Academic Graph: When experts are not enough." In: *Quant. Sci. Stud.* 1.1, pp. 396–413. DOI: 10.1162/qss_a_00021 (cit. on p. 82).

Wang, Minjie et al. (2019). "Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs." In: *CoRR* abs/1909.01315. arXiv: 1909.01315 (cit. on pp. 22, 83).

Wang, Ziqi, Yuwei Tan, and Ming Zhang (2010). "Graph-Based Recommendation on Social Networks." In: *Advances in Web Technologies and Applications, Proceedings of the 12th Asia-Pacific Web Conference, APWeb 2010, Busan, Korea, 6-8 April 2010*. Ed. by Wook-Shin Han, Divesh Srivastava, Ge Yu, Hwanjo Yu, and Zi Helen Huang. IEEE Computer Society, pp. 116–122. DOI: 10.1109/APWeb.2010.60 (cit. on p. 1).

Waradpande, Vikram, Daniel Kudenko, and Megha Khosla (2020). "Deep Reinforcement Learning with Graph-based State Representations." In: *CoRR* abs/2004.13965. arXiv: 2004.13965 (cit. on p. 75).

Watts, Duncan J. and Steven H. Strogatz (1998). "Collective dynamics of 'small-world' networks." In: *Nature* 393, pp. 440–442. ISSN: 0028-0836. DOI: 10.1038/30918 (cit. on p. 1).

Wu, Felix, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger (2019). "Simplifying Graph Convolutional Networks." In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 6861–6871. URL: http://proceedings.mlr.press/v97/wu19e.html (cit. on pp. 72, 73, 75, 80, 83, 92).

Wu, Zonghan, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu (2021). "A Comprehensive Survey on Graph Neural Networks."

In: *IEEE Trans. Neural Networks Learn. Syst.* 32.1, pp. 4–24. DOI: 10.1109/TNNLS.2020.2978386 (cit. on p. 54).

Xu, Keyulu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka (2019). "How Powerful are Graph Neural Networks?" In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. URL: https://openreview.net/forum?id=ryGs6iA5Km (cit. on pp. 19, 93, 94, 138).

Xu, Keyulu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka (2018). "Representation Learning on Graphs with Jumping Knowledge Networks." In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by Jennifer G. Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 5449–5458. URL: http://proceedings.mlr.press/v80/xu18c.html (cit. on pp. 68, 82, 114).

Yanardag, Pinar and S. V. N. Vishwanathan (2015). "Deep Graph Kernels." In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*. Ed. by Longbing Cao, Chengqi Zhang, Thorsten Joachims, Geoffrey I. Webb, Dragos D. Margineantu, and Graham Williams. ACM, pp. 1365–1374. DOI: 10.1145/2783258.2783417 (cit. on p. 28).

Yang, Zhilin, William W. Cohen, and Ruslan Salakhutdinov (2016). "Revisiting Semi-Supervised Learning with Graph Embeddings." In: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*. Ed. by Maria-Florina Balcan and Kilian Q. Weinberger. Vol. 48. JMLR Workshop and Conference Proceedings. JMLR.org, pp. 40–48. URL: http://proceedings.mlr.press/v48/yanga16.html (cit. on pp. 23, 107, 138).

Yao, Liang, Chengsheng Mao, and Yuan Luo (2019). "Graph Convolutional Networks for Text Classification." In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, pp. 7370–7377. DOI: 10.1609/aaai.v33i01.33017370 (cit. on p. 17).

You, Jiaxuan, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec (2018). "GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models." In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by Jennifer G. Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning

Research. PMLR, pp. 5694–5703. URL: http://proceedings.mlr.press/v80/you18a.html (cit. on pp. 55, 61, 62).

Zarrouk, Tayeb, Romain Couillet, Florent Chatelain, and Nicolas Le Bihan (2020). "Performance-Complexity Trade-Off in Large Dimensional Statistics." In: *30th IEEE International Workshop on Machine Learning for Signal Processing, MLSP 2020, Espoo, Finland, September 21-24, 2020*. IEEE, pp. 1–6. DOI: 10.1109/MLSP49062.2020.9231568 (cit. on p. 101).

Zeng, Hanqing, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor K. Prasanna (2020). "GraphSAINT: Graph Sampling Based Inductive Learning Method." In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. URL: https://openreview.net/forum?id=BJe8pkHFwS (cit. on p. 120).

Zhang, Muhan and Yixin Chen (2017). "Weisfeiler-Lehman Neural Machine for Link Prediction." In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*. ACM, pp. 575–583. DOI: 10.1145/3097983.3097996 (cit. on pp. 46, 54).

Zhang, Muhan, Zhicheng Cui, Marion Neumann, and Yixin Chen (2018). "An End-to-End Deep Learning Architecture for Graph Classification." In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*. Ed. by Sheila A. McIlraith and Kilian Q. Weinberger. AAAI Press, pp. 4438–4445. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17146 (cit. on p. 27).

Zhang, Xiang and Marinka Zitnik (2020). "Gnnguard: Defending graph neural networks against adversarial attacks." In: *Advances in Neural Information Processing Systems* 33, pp. 9263–9275 (cit. on p. 95).

Zhou, Jie, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun (2020). "Graph neural networks: A review of methods and applications." In: *AI Open* 1, pp. 57–81. DOI: 10.1016/j.aiopen.2021.01.001 (cit. on pp. 21, 54).

Zhou, Yu, Haixia Zheng, and Xin Huang (2020). "Graph Neural Networks: Taxonomy, Advances and Trends." In: *CoRR* abs/2012.08752. arXiv: 2012.08752 (cit. on pp. 54, 95).

Zhu, Dingyuan, Ziwei Zhang, Peng Cui, and Wenwu Zhu (2019). "Robust graph convolutional networks against adversarial attacks." In: *Proceedings of the 25th*

ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1399–1407 (cit. on p. 95).

Zhu, Jiong, Ryan A Rossi, Anup Rao, Tung Mai, Nedim Lipka, Nesreen K Ahmed, and Danai Koutra (2021). "Graph neural networks with heterophily." In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 11168–11176 (cit. on p. 96).

Zhu, Jiong, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra (2020). "Beyond homophily in graph neural networks: Current limitations and effective designs." In: *Advances in Neural Information Processing Systems* 33, pp. 7793–7804 (cit. on p. 96).

Zügner, Daniel and Stephan Günnemann (2019). "Adversarial Attacks on Graph Neural Networks via Meta Learning." In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. URL: https://openreview.net/forum?id=Bylnx209YX (cit. on pp. 94, 95).

Zügner, Daniel and Stephan Günnemann (2020). "Certifiable Robustness of Graph Convolutional Networks under Structure Perturbations." In: *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*. Ed. by Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash. ACM, pp. 1656–1665. DOI: 10.1145/3394486.3403217 (cit. on p. 95).

ECOLE
DOCTORALE

Titre : Apprentissage de la Représentation des Graphes : des Noyaux aux Réseaux Neuronaux

Mots clés : Noyau de Graphe, Réseaux Neuronaux de Graphes, Apprentissage de la Représentation, Graphe Dynamique, Sparsification des Réseaux Neuronaux, Matrice Aléatoire

Résumé : Les graphes sont omniprésents en tant que structure de données, de par leur capacité à modéliser des informations relationnelles entre objets. Ainsi, la capture d'informations à partir de données structurées en graphes, soit l'exploration de graphes ou l'apprentissage automatique des graphes, est un sujet important. Dans cette thèse, nous présentons une série de contributions de recherche sur les sujets de l'apprentissage automatique des graphes en utilisant des méthodes à noyau et le modèle émergent des réseaux neuronaux de graphes (RNGs). Dans la première partie, nous présentons un nouveau noyau de graphe qui calcule la similarité entre deux graphes en calculant une affectation optimale de leurs plongements de nœuds dans le même espace vectoriel. En utilisant un algorithme de clustering, nous construisons une hiérarchie des sommets dans cet espace, à travers laquelle le noyau proposé peut trouver une correspondance optimale des sommets qui maximise la similarité globale. L'efficacité de ce noyau est démontrée empiriquement sur plusieurs datasets de classification de graphes. La deuxième partie de cette thèse est consacrée aux RNGs. Nous appliquons d'abord cette avancée récente au domaine des réseaux temporels, en proposant un modèle séquentiel pour prédire l'évolution des graphes dynamiques. Plus précisément, nous utilisons les RNGs et une architecture récurrente pour encoder la séquence de graphes en évolution dans un espace latent, puis nous employons un modèle génératif pour prédire la topologie du graphe à l'étape suivante et construire l'instance de graphe correspondant à cette topologie à partir de l'espace latent. Les travaux suivants se rapprochent des principes fondamentaux des RNGs en abordant leurs limites en termes de coût de calcul et de robustesse au bruit structurel. Dans le premier cas, nous démontrons que les réseaux de neurones de passage de messages, un type courant de RNG, peuvent être considérablement simplifiés en sparifiant ou, dans certains cas, en omettant leurs étapes de mise à jour. Pour le second, nous proposons de robustifier les RNGs avec un noyau de caractéristiques de nœuds pour contrer les perturbations potentielles sur les structures de graphes. Tous les modèles proposés sont évalués empiriquement sur des tâches correspondantes et se montrent compétitifs par rapport aux méthodes de référence.

Title : Graph Representation Learning: from Kernel to Neural Networks

Keywords : Graph Kernel, Graph Neural Networks, Representation Learning, Dynamic Graph, Neural Network Sparsification, Random Matrix

Abstract : Graphs are ubiquitous as they can naturally represent most real-world data. Thus, capturing information from graph-structured data, i.e., graph mining and machine learning for graphs, has long become and remains an important topic. In this thesis, we introduce a series of research contributions on subjects of machine learning for graphs using kernel methods and the emerging graph neural networks (GNNs) framework. In the first part, we present a novel graph kernel that computes the similarity between two graphs by computing an optimal assignment of their node embeddings in the same vector space. Using a clustering algorithm, we construct a hierarchy of the vertices in this space, through which the proposed kernel can find an optimal matching of the vertices that maximizes the overall similarity. The efficiency of this kernel is demonstrated empirically on several graph classification datasets. The second part of this presentation is devoted to GNNs. We first apply this recent advancement to the field of temporal networks, proposing a sequential framework to predict the evolution of dynamic graphs. Specifically, we use GNNs and a recurrent architecture to encode the sequence of evolving graphs into a latent space and then employ a generative model to predict the topology of the graph at the next step and construct the graph instance corresponding to that topology back from the latent space. The following work moves closer to the fundamentals of GNNs by addressing their limitations in terms of computational cost and robustness against structural noise. For the first one, we demonstrate that message-passing neural networks, a common type of GNNs, can be substantially simplified by sparsifying, or in some cases omitting their update steps. For the second, we propose to robustify GNNs with a node feature kernel to counter potential perturbations on their graph structures. All proposed models are evaluated empirically on corresponding tasks and are proven to be competitive with state-of-the-art methods.

Institut Polytechnique de Paris
91120 Palaiseau, France