Towards Expressive Graph Neural Networks : Theory, Algorithms, and Applications

Georgios Dasoulas

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2022IPPAX020

Thèse de doctorat

ÉCOLE POLYTECHNIQUE

IP PARIS

Towards Expressive Graph Neural Networks: Theory, Algorithms and Applications

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

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

Thèse présentée et soutenue à Palaiseau, le 21 mars 2022, par

GEORGIOS DASOULAS

Composition du Jury :

Franco Scarselli Professeur associé, SAILab, Department of Information Engineering and Mathematics, University of Siena	Rapporteur
Marc Lelarge Professeur associé, DYOGENE team, INRIA - École normale supérieure	Rapporteur
Maks Ovsjanikov Professeur, GeoViC Group, Laboratoire d'Informatique de l'Ecole Polytechnique, L'Institut Polytechnique de Paris	Président du jury
Marinka Zitnik Professeure assistante, Zitnik Lab, Department of Bioinformatics, Harvard University	Examinatrice
Eftratios Gallopoulos Professeur, HPCLab, Computer Engineering and Informatics Department (CEID), University of Patras	Examinateur
Michalis Vazirgiannis Professeur, DaSciM group, Laboratoire d'Informatique de l'Ecole Polytechnique, L'Institut Polytechnique de Paris	Directeur de thèse
Aladin Virmaux Chercheur, Noah's Ark Lab, Huawei Technologies France	Co-directeur de thèse

Abstract

Towards Expressive Graph Neural Networks: Theory, Algorithms and Applications

As the technological evolution of machine learning is accelerating nowadays, data plays a vital role in building intelligent models, being able to simulate phenomena, predict values and make decisions. In an increasing number of applications, data take the form of networks. The inherent graph structure of network data motivated the evolution of the graph representation learning field. Its scope includes generating meaningful representations for graphs and their components, i.e., the nodes and the edges. The research on graph representation learning was accelerated with the success of message passing frameworks applied on graphs, namely the Graph Neural Networks. Learning informative and expressive representations on graphs plays a critical role in a wide range of real-world applications, from telecommunication and social networks, urban design, chemistry, and biology. In this thesis, we study various aspects from which Graph Neural Networks can be more expressive, and we propose novel approaches to improve their performance in standard graph learning tasks. The main branches of the present thesis include: the universality of graph representations, the increase of the receptive field of graph neural networks, the design of stable deeper graph learning models, and alternatives to the standard message-passing framework. Performing both theoretical and experimental studies, we show how the proposed approaches can become valuable and efficient tools for designing more powerful graph learning models.

In the first part of the thesis, we study the quality of graph representations as a function of their discrimination power, i.e., how easily we can differentiate graphs that are not isomorphic. Firstly, we show that standard message-passing schemes are not universal due to the inability of simple aggregators to separate nodes with ambiguities (similar attribute vectors and neighborhood structures). Based on the found limitations, we propose a simple coloring scheme that can provide universal representations with theoretical guarantees and experimental validations of the performance superiority. Secondly, moving beyond the standard message-passing paradigm, we propose an approach for treating a corpus of graphs as a whole instead of examining graph pairs. To do so, we learn a soft permutation matrix for each graph, and we project all graphs in a common vector space, achieving a solid performance on graph classification tasks.

In the second part of the thesis, our primary focus is concentrated around the receptive field of the graph neural networks, i.e., how much

information a node has in order to update its representation. To begin with, we study the spectral properties of standard operators that encode adjacency information, namely the graph shift operators. We propose a novel parametric family of operators that can adapt throughout training and provide a flexible framework for data-dependent neighborhood representations. We show that the incorporation of this approach has a substantial impact on both node classification and graph classification tasks. Next, we study how considering the k-hop neighborhood information for a node representation can output more powerful graph neural network models. The resulted models are proven capable of identifying structural properties, such as connectivity and triangle-freeness.

In the third part of the thesis, we address the problem of long-range interactions, where nodes that lie in distant parts of the graph can affect each other. In such a problem, we either need the design of deeper models or the reformulation of how proximity is defined in the graph. Firstly, we study the design of deeper attention models, focusing on graph attention. We calibrate the gradient flow of the model by introducing a novel normalization that enforces Lipschitz continuity. Next, we propose a data augmentation method for enriching the node attributes with information that encloses structural information based on local entropy measures.

Résumé en Français

Vers des Graph Neural Networks: théorie, algorithmes et applications

L'évolution de l'apprentissage automatique s'accélérant, les donnés jouent un role de plus en plus important dans la construction de modèles intélligents, capables de simuler des phénomènes, de prédire des résultats complexes et de prendre des décisions. Dans un nombre sans cesse croissant d'applications, les données sont structurées et peuvent être vues comme des graphes. L'exploitation de cette structure est le coeur du domaine de l'apprentissage de représentations de graphes, qui consiste à calculer des représentations suffisamment expressives des graphes et de ses composants, c'est-à-dire les nœuds et les arêtes. Récemment, la domaine de l'apprentissage de représentations de graphes a été accéléré par le succès des algorithmes du type «message passing» (passation de messages) appliqués aux graphes, à savoir les «Graphe Neural Network» (réseaux de neurones sur les graphes). L'apprentissage de représentations informatives et expressives sur les graphes joue un rôle critique dans un large éventail d'applications du monde réel, depuis les télécommunications et les réseaux sociaux jusqu'à la conception urbaine, la chimie et la biologie. Dans cette thèse, nous étudions les différents aspects à partir desquels les réseaux neuronaux graphiques peuvent être plus expressifs, et nous proposons de nouvelles approches pour améliorer leurs performances dans les tâches standard d'apprentissages. Les principaux axes de la présente thèse sont : l'universalité des représentations de graphes, l'augmentation du champ réceptif des réseaux de neurones sur les graphes, la conception de modèles d'apprentissage de graphes stables et profonds et enfin les alternatives au cadre standard des algorithmes par passation de messages. En réalisant des études théoriques et expérimentales, nous montrons comment les approches proposées peuvent devenir des outils utiles et efficaces pour concevoir des modèles d'apprentissage de graphes plus expressifs et plus puissants.

Dans la première partie de la thèse, nous étudions la qualité des représentations de graphes en fonction de leur pouvoir de discrimination, c'est-à-dire la capacité à différencier des graphes qui ne sont pas isomorphes. Tout d'abord, nous montrons que les schémas standards de passation de messages ne sont pas universels, en raison de l'incapacité des agrégateurs simples à séparer les nœuds présentant des ambiguïtés (vecteurs d'attributs et structures de voisinage similaires). Sur la base des limitations constatées, nous proposons un schéma de coloration, qui bien que simple peut fournir des représentations universelles avec des garanties théoriques. Nous validons expérimentalement notre approche ainsi que

la supériorité des performances que nous obtenons. Puis, au-delà du paradigme standard de passation de messages, nous proposons une approche pour traiter un corpus de graphes comme un tout, au lieu d'examiner localement l'intégralité des paires de graphes. Pour ce faire, nous apprenons une matrice de permutation relaxée pour chaque graphe et nous projetons tous les graphes dans un espace vectoriel commun, ce qui permet d'obtenir de solides performances dans les tâches de classification de graphes.

Dans la deuxième partie de la thèse, nous nous concentrons sur le champ réceptif des réseaux neuronaux de graphes, c'est-à-dire sur la quantité d'informations dont dispose un nœud pour mettre à jour sa représentation. Pour commencer, nous étudions les propriétés spectrales d'opérateurs standards, qui encodent l'information d'adjacence, à savoir les opérateurs de déplacements. Nous proposons une nouvelle famille paramétrique d'opérateurs qui peuvent s'adapter tout au long de l'apprentissage afin de fournir un cadre flexible pour les représentations de voisinage dépendant des données. Nous montrons que l'incorporation de cette approche a un fort impact sur les tâches de classification des nœuds et des graphes. Ensuite, nous étudions comment la prise en compte des informations de voisinage k-hop pour les représentations de nœuds peut produire des modèles de réseaux neuronaux de graphes plus expressifs. Les modèles obtenus s'avèrent capables d'identifier des propriétés structurelles, telles que la connectivité et l'absence de triangle.

Dans la troisième partie de la thèse, nous abordons le problème des interactions à longue distance, où des nœuds situés dans des parties éloignées du graphe peuvent s'influencer mutuellement. Dans ce type de problème, nous avons besoin soit de concevoir des modèles plus profonds, soit de reformuler la manière dont la proximité est définie dans le graphe. Tout d'abord, nous étudions la conception de modèles d'attention plus profonds, en nous concentrant sur les modèles d'Attention de graphes. Nous calibrons le flux de gradient du modèle en introduisant une nouvelle normalisation qui force le modèles à être Lipschitz. Enfin, nous proposons une méthode d'augmentation des données pour enrichir les attributs des nœuds avec des informations qui contiennent des informations structurelles, basées sur des mesures d'entropie locale.

Acknowledgements

The present manuscript reflects the end of my doctoral work and 3 (and a half) amazing years spent with great mentors, collaborators, friends and family. All of these people have a special place in my heart and they deserve a special place at this manuscript too.

First and foremost, I would like to thank my academic supervisor Professor Michalis Vazirgiannis from École Polytechnique for his constant guidance and mentorship throughout my doctoral journey. I owe an equal gratitude to my industrial advisors Aladin Virmaux and Kevin Scaman from Huawei Technologies France, without whom most of my work wouldn't exist. Our conversations, their instructions and the many hours that we spent above a piece of paper or a code are invaluable to me. I would, also, like to thank Balázs Kégl and the whole team in Noah's Ark Lab for our great collaboration and their warm interest on doing research.

I am, also, very grateful and honored with the contribution of the members from my Ph.D. defense committee: the two reviewers Prof. Franco Scarselli (from University of Siena) and Prof. Marc Lelarge (from École Normale Superiore), the president of the jury Prof. Maks Ovsjanikov (from École Polytechnique), as well as Prof. Marinka Zitnik (from Harvard University) and Prof. Efstratios Gallopoulos (from University of Patras). Their constructive comments have been very important for my work.

Moreover, I would like to give a big thanks to all of my collaborators, Johannes Lutzeyer, Giannis Nikolentzos, Ludovic dos Santos, Guillaume Salha-Galvan, Michalis Chatzianastasis, Stratis Limnios, Dimitris Thilikos, Giorgos Siolas. It was very fortunate for me to have a chance working with you and I hope that I can continue to do with most of you. Except from the collaborations, I would like to thank all of my academic friends I met these years, George, Dimitris, Kyriakos, Ilias, George (x2), Zacharias, Manolis, Sissy, Katerina, Roberto and Maria with whom I spent great time.

Accounting the important people of my academic and personal life, I cannot forget my great friends from Greece. My beloved friends from my hometown Giannena, Michalis, Panos, Giorgos, Eleni and Stavros, whom although I see them just a few times per year, they are always in my mind. Also, my friends from my college years in Athens, Argyris, Christos, Yannis, Nikos, Vasilis, Aspa and Sotiria with whom I spent some of my most amazing years so far. And of course, I thank Elli, Lara and Nicole for our great time and their support all of this time.

Most importantly, I dedicate all of my work to my beloved Tina. You were always next to me, helping me, guiding me in both personal and academic decisions and cheering me up all of these years, full with patience, love and good will. The end or even the beginning of my doctoral journey would not be possible without you. Thank you for all you have done.

Last but not least, I owe my greatest gratitude for my family. My parents Rena and Triantafyllos, my sister Elena and my grandparents for their unconditional love and the immeasurable support they gave me. I owe to them not only this work, but everything I achieved in my entire life.

Thank you everyone that has been next to me until today and continues to do so.

George Dasoulas,
Paris, Spring 2022

Contents

List of Figures

List of Tables

List of Abbreviations

Agg	**A**ggregator
CLIP	**C**olored **L**ocal **I**terative **P**rocedure
GAE	**G**raph **A**uto-**E**ncoder
GAT	**G**raph **At**tention-Network
GCN	**G**raph **C**onvolutional **N**etwork
GGNN	**G**ated **G**raph **N**eural **N**etwork
GNN	**G**raph **N**eural **N**etwork
GPT	**G**enerative **P**re-**T**rained Model
GSO	**G**raph **S**hift **O**perator
Lip	**Lip**schitz
MLP	**M**ulti-**L**ayer **P**erceptron
MPNN	**M**essage **P**assing **N**eural **N**etwork
NN	**N**eural **N**etwork
PGSO	**P**arametrised **G**raph **S**hift **O**perator
VNE	**V**on **N**eumann **E**ntropy
VNGE	**V**on **N**eumann **G**raph **E**ntropy

Chapter 1

Introduction

The world as we know it is a product of changes. Molecular, historical, environmental, planetary transitions from one phase to another trigger a set of consequences. All of these changes are characterised by a common phenomenon, the *interactions* of *entities*, creating biomedical, communication or physical *networks*. A mathematical framework that accurately describes a network is given by the field of graph theory, where *graphs* are defined as structures that include sets of objects as the interactions in-between.

1.1 Graphs are Everywhere

Contrary to the set formulation, graphs serve as a flexible framework to encode interactions between elements of the same set. They also provide a well-established mathematical formalization for such information. Beginning in 1736, Leonard Euler initially set the foundations of graph theory for the sake of an urgent real-life transportation problem [24]. Since then, graphs have been a significant subject of mathematics research, as the number of applications was constantly increasing. Today, we can meet graphs nearly everywhere from chemistry and biology to social networks and politics [80, 135, 228], from physics to computer science [207, 231].

1.2 Machine Learning on Graphs

As the data saved and processed daily become larger and more complex, their network representations move beyond the standard formulation of a graph by incorporating contextual information upon the nodes or the edges (more technical information will be provided in Section 2.1). Such a combination of contextual and structural information allow for more complex problems to be defined, such as the *prediction* of node, edge or graph labels, the *classification* of a node or a graph into a specified class or the *clustering* of points inside a graph. Machine learning models appear to be an appropriate class of solvers for such problems.

Representation Learning In general, Machine Learning (ML) can be summarized by the process of learning from data in order to solve particular tasks. However, the success of an ML model is tightly dependent on the way that the data are represented, that is, the *features* we assume as input of the model. Until recently, most of the effective ways to extract valuable information from data for efficient ML models were based on manual feature engineering [254, 202]. Due to the high variance of the model performance and the unsafe human factor dependence (as humans annotate and form the data given to a model) of the ML models, the approach of automatic feature extraction has been developed, creating the idea for *representations* of the data. That is a way to project the unprocessed (or lightly processed) data into a space that can provide rich information to the user-end task. The process of generating such projections (or embeddings or representations) is called *representation learning*.

Representation learning has emerged as a prominent toolkit for graph-structured data due to the wide variety of tasks and forms that networks could express. The first, more traditional, approaches combined node-level statistics, such as node degrees, clustering coefficients, and centralities, to extract node-level features [106]. Graph kernel methods [223, 168] have been quite successful for generating graph-level features, where the objective is the label extraction for a whole graph instead of a node or edge individually.

The idea of considering neural networks as models over graphs has been firstly introduced in Sperduti and Starita [213]. Later on, Gori, Monfardini, and Scarselli [83] and Scarselli et al. [200] made the first steps towards establishing a rigid framework of the so-called Graph Neural Networks. The core idea of the utilization of deep learning to the field of graphs lies in a simple iterative procedure: A node learns a representation as a result of the information propagation from its neighborhood.

1.3 Real-World Applications

Graph representation learning has been a topic of accelerating research due to its broad applicability in various fields. Making inferences on graph-structured data was proven successful in complex topics from technological, societal, biomedical, and other areas. Next, we present a brief list of such applications, where GNNs appear to be a dominant approach.

1.3.1 Bioinformatics

Biomedical research is one of the most fruitful areas for graph learning models [135, 251]. Their applications relate to drug discovery, disease

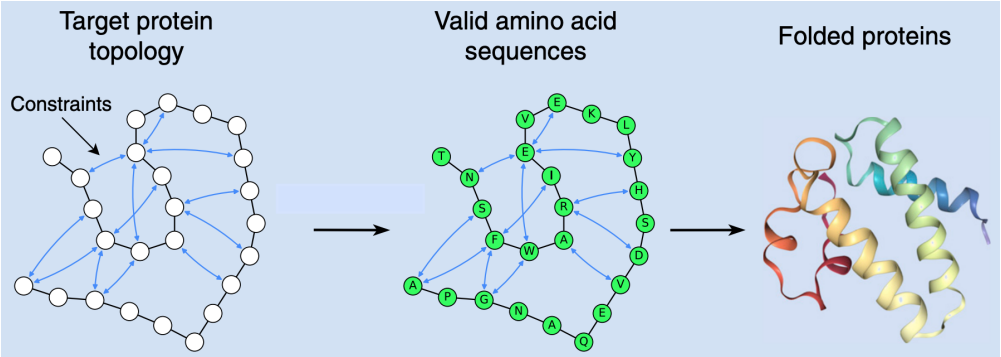


FIGURE 1.1: Visualization of protein sequence modeling into graph topology and inference of protein folding. The figure is based on the work of [214].

prediction and reasoning, and medical imaging. Regarding the drug discovery, GNNs have been utilized for the learning of protein structure and functionality [104, 81, 214], the prediction of the interactions between proteins [142, 242] and the prediction of molecular properties [64, 112]. In particular, a whole protein structure database has been recently constructed, based on AlphaFold [110], a prediction system that is built upon graph-learning blocks and models protein structures as attributed graphs. In Figure 1.1, a graph representation of protein sequences is presented. For the disease prediction and reasoning, research has been done for the association of diseases with RNA and the prediction of interactions among cells [171, 185, 188]. Finally, in the area of medical imaging, GNNs have been applied to the segmentation of images [148] and graphical representation of brain networks [126].

1.3.2 Social networks

Graph representation learning approaches have also been widely used in social network analysis [175, 86]. One of the most usual tasks in social networks is the prediction of interactions between two or multiple nodes, and we can meet this objective very often in recommender systems, where the relationship encoding between entities is crucial [231]. Graph learning models, such as Graph Auto-encoders [119] have been proven a prominent approach for this task. Moreover, considering networks of research collaboration as social networks, citation graphs [153, 79] are common use-cases, where graph learning models have been utilized. In such scenarios, the task is usually the node classification, where papers or authors' topics or other information needs to be predicted.

1.3.3 Physics

Recently, graph neural network models exhibited an impressive performance in tasks related to physics simulations. In a wide range of science and engineering areas, where the behavior of solids, fluids, or

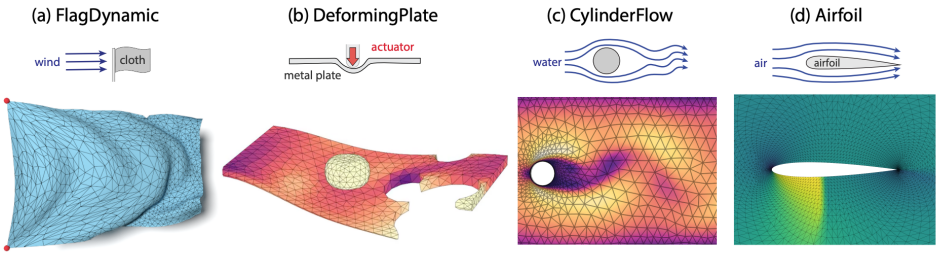


FIGURE 1.2: Graph representations can be utilized for the modelling of dynamics for different physical systems, from structural mechanics to fluid dynamics, such as a) flag waving, b) plate deformation, c) cylinder flow and d) air flow around an airwing. The mesh plots are taken from Pfaff et al. [177]

deformable materials is studied, mesh-based simulations have been a dominant approach for many years [207]. The core idea of mesh generation and modeling is the discretization of an object and its transformation into an interactions network of atoms or molecules. Machine learning methods have been proven successful in learning adaptive meshes with regards to the needs of each task and solver [194, 177]. The interplay between graph neural networks and mesh generation is visualized in Figure 1.2 for the modeling of complex physical systems dynamics.

1.3.4 Transportation Systems

Another aspect on which GNNs appear to be successful is the intelligent transportation systems, and, more specifically, the modeling of traffic networks [109]. Problems such as traffic flow, traffic speed, traffic accidents, and parking availability are a few that have been studied in the related literature [234]. One recent real-world application was the incorporation of a GNN model in a popular web mapping platform for the more accurate prediction of the estimated time of arrival [61]. In Figure 1.3, the approach of representing neighboring regions to adjacent nodes is visualized.

1.3.5 Communication Networks

In the field of telecommunications, the network topology can appear in abundant problems. Recently, there have been many approaches that are modeling this network topology through graph-based deep learning methods [108]. For the problem of optimal power allocation in wireless networks, various methods are based on graph convolutional networks [66, 44]. Moreover, regarding the paradigm of 5G network slicing, where the physical network infrastructure is divided into virtualized independent networks, architectures based on Graph Attention Networks have exhibited competitive results [203, 226].

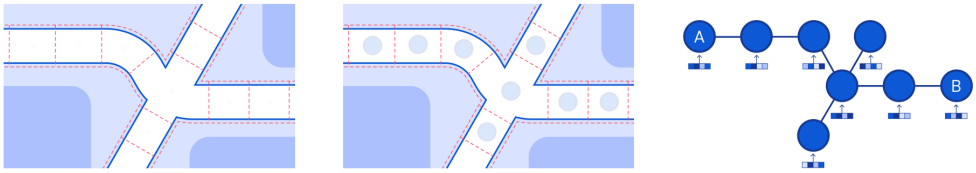


FIGURE 1.3: Graph representation of a traffic network for the prediction of the estimated time of arrival. Neighboring regions in a road network map appear as adjacent nodes in the graph-level representation. The visualization is taken from Derrow-Pinion et al. [61].

1.3.6 Neural Networks modeling

Considering a general neural network as a directed bipartite graph (and acyclical), we can define its architecture under the graph representation learning framework. Taking this into account, we can explore more possibilities of model designs, such as the case of *random wirings* [235], but, also, make inferences over the optimal configuration of model parameters [140]. The idea of the neural network representation as a graph has been recently studied in the field of Neural Architecture Search and AutoML, where the objective is to explore optimal architectures for given tasks in an automated manner [37, 130, 250, 134].

1.4 Contributions

In the context of the present thesis, our main focus is to build powerful graph neural networks that are able to provide rich and expressive representations for standard and more complex tasks. These tasks can be related to the discrimination of non-isomorphic graphs, the need for larger receptive fields, and the detection of long-range interactions in a graph. In most cases, our pipeline of work includes the introduction of an efficient approach, along with a theoretical validation and experimental evaluation. Next, we present more precisely the contributions of the present thesis, which are also visualized in Figure 1.4.

1.4.1 Discrimination Power

Universality One of the main questions that relate to learning graph representations is whether we can learn an embedding that can approximate any filter applied on a graph. In this manuscript, we study the problem of universality on graph representations, and we prove that specific topological criteria suffice to provide models that are universal approximators [56]. Our introduced models, CLIP and $k-$CLIP, can provide state-of-the-art results for real-world graph classification tasks but also succeed in synthetic graphs that express the need for a high discrimination power. Indeed, we show that CLIP achieves to

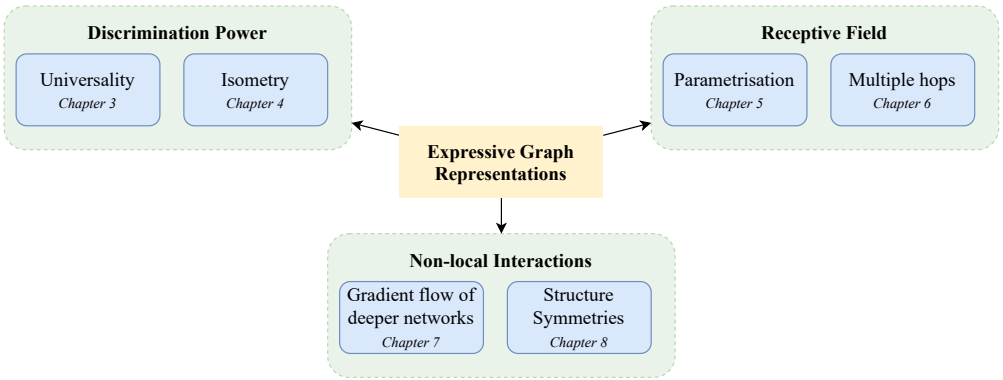


FIGURE 1.4: A visual overview of the present thesis' contributions with the corresponding chapters.

discriminate graphs with respect to the graph properties of connectivity, triangle-freeness, and bipartiteness.

Isometry Usually, by discrimination power, we refer to the ability of the representations to distinguish pairs of graphs that are either non-isomorphic or have different attribute information. Nevertheless, how can we compare the difference of the representations with the structural difference of the corresponding graphs? We show that standard GNNs are not able to preserve isometry, and we propose a novel approach that embeds a corpus of graphs into a common vector space [167]. To do so, the models learn a series of soft permutations, imposing a soft ordering over the graph nodes. We observe that our model can achieve competitive results both in graph classification and graph regression tasks.

1.4.2 Receptive Field

In convolutional networks, the receptive field is defined by the filter size of a layer, showing the depth and extent of input information that a neuron receives. Correspondingly, on graph neural networks, we can see the receptive field as an indicator of how large neighborhoods we take into account for the representation of a node.

Parametric Operators The neighborhood information of the graph nodes are encoded through the message passing operators that we utilize, such as the adjacency matrix or the Laplacian. For different tasks and models, different operators have been proposed. Within this scope, we propose a novel parametric family of operators that can adapt according to the given task and input information, namely the Parametric Graph Shift Operators (PGSO) [54]. We show that PGSO retains important spectral properties of standard operators, and we focus on its positive impact on a wide variety of graph learning tasks.

Multiple hops Standard GNNs are based on 1-hop aggregation schemes, where nodes that are not connected take into account one another implicitly through the utilization of more depths. We show that this design assumption limits the model expressivity in tasks, where the objective is the detection of a graph structural property, such as connectivity [166]. We propose $k-$hop graph neural networks that update the node-level representations based on an iterative scheme over $k-$hop neighborhoods, overcoming the aforementioned limitation.

1.4.3 Beyond Local Interactions

In various real-world applications, we assume that the considered graph structure respects the assortativity property, i.e., nodes that are similar (e.g., have similar node attributes) prefer to attach to each other. However, there are cases where either structural symmetries can affect the behavior of distant nodes (e.g., in molecular networks [152]) or the presence of structural noise can remove connections in a graph.

Stability and model depth Firstly, we study the efficient design of graph attention models that are able to detect long-range interactions [55]. We propose a novel normalization that enforces Lipschitz continuity and enables the construction of stable deep attention models with a significant impact on node classification tasks.

Structural symmetries Moreover, we propose an approach for augmenting node attribute vectors with structural information, based on entropy measures, that are indicators of structural symmetries [57]. We show that we can achieve similar performance to standard message passing neural networks through such augmentation and simple neural network models over the node attributes.

1.5 Thesis Structure

The present manuscript is organized as follows:

- Chapter 2 begins with a description of the main message passing framework and taxonomy of the current state-of-the-art in the field of graph representation learning. Then, it focuses on the expressivity aspects of GNNs.
- Chapter 3 and Chapter 4 present the works related to the discrimination power of GNNs. Specifically, Chapter 3 introduces CLIP, a model that provides universal representations, and Chapter 4 introduces $\pi-$GNN, the approach that learns soft permutations over graphs.
- Chapter 5 and Chapter 6 lie in the scope of the receptive field. Chapter 5 introduces PGSO for learning parametrized graph shift

operators, and Chapter 6 focuses on the iterative processing of multiple hop neighborhood information at a single layer.

- Chapter 7 and Chapter 8 address questions that are beyond the scope of local interactions. Chapter 7 presents LipschitzNorm, the normalization for stable deeper attention models, and Chapter 8 studies VNEstruct, the attribute augmentation method for structural symmetries.
- Chapter 9 discusses the conclusions of the present thesis and future directions that can be further examined.

List of Publications

For the sake of completeness, we summarize the list of research works, that have been performed in the context of the doctoral thesis. Works **1-7** resulted in publication, either in conference proceedings or in journal, while works **8**, **9** and **10** are currently under review. Works **2**, **6**, **8** and **9** are not discussed further in this manuscript, as they can be considered as out of the scope of this doctoral thesis' overall message. The rest of the works are presented and discussed throughout the present manuscript.

- **Conference Proceedings**

  1. **George Dasoulas**, Kevin Scaman, Aladin Virmaux. Lipschitz Normalization for self-attention layers with application to graph neural networks. In International Conference on Machine Learning (ICML), 2021.

  2. Michalis Chatzianastasis, **George Dasoulas**, Georgios Siolas, Michalis Vazirgiannis. Graph-based neural architecture search with operation embeddings. In ICCV Workshop on Neural Architectures: Past, Present and Future, 2021.

  3. **George Dasoulas**, Johannes Lutzeyer, Michalis Vazirgiannis. Learning parametrised graph shift operators. In International Conference on Learning Representations (ICLR), 2021.

  4. **George Dasoulas**, Giannis Nikolentzos, Kevin Scaman, Aladin Virmaux, Michalis Vazirgiannis. Ego-based entropy measures for structural representations. In International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2021.

  5. **George Dasoulas**, Ludovic Dos Santos, Kevin Scaman, Aladin Virmaux. Coloring graph neural networks for node disambiguation. In International Joint Conference on Aritificial Intelligence (IJCAI), 2020.

  6. Stratis Limnios, **George Dasoulas**, Dimitrios Thilikos, Michalis Vazirgiannis. Hcore-Init: Neural Network Initialization based on Graph Degeneracy. In International Conference on Pattern Recognition (ICPR), 2020.

- **Journal Publications**

    7. Giannis Nikolentzos, **George Dasoulas**, Michalis Vazirgiannis. K-hop graph neural networks. In Neural Networks Journal, Elsevier, 2020.

- **Under Review**

    8. Guillaume Salha-Galvan, Johannes Lutzeyer, **George Dasoulas**, Michalis Vazirgiannis. Modularity-aware Graph Autoencoders for Joint Community Detection and Link Prediction. TBA, 2022.

    9. Michalis Chatzianastasis, Johannes Lutzeyer, **George Dasoulas**, Michalis Vazirgiannis. Graph Ordering Attention Networks. TBA, 2021.

    10. Giannis Nikolentzos, **George Dasoulas**, Michalis Vazirgiannis. Permute me Softly. Learning Soft Permutations for Graph Representations. In CoRR, 2021.

Chapter 2

Preliminaries

Before proceeding with the main contributions, we present the essential elements that are appropriate for the definition of graph learning models and reveal a few aspects of how the expressivity of GNNs can be significant.

2.1 Notation

Firstly, we describe the notations that we will follow for the rest of the present manuscript. Specifically, we define what a graph is, how we can encode it and what tasks we usually meet in graph representation learning. We also present a categorization of the graph learning models, depending on how the information is propagated in a graph.

2.1.1 Graphs, tuples, and sets

We let a graph $\mathcal{G}$ be defined as a tuple of two sets, a set of nodes V and a set of edges $\mathcal{E}$: $\mathcal{G} = (V, \mathcal{E})$. Let $|V| = n$ be the number of nodes or *size* of graph $\mathcal{G}$ and $|gE| = m$ be the number of edges. Also, $\mathcal{E} = \{(u_i, u_j)|u_i \in V,\ u_j \in V,\ \text{there is an edge from } u_i \text{ to } u_j\}$. Given a node u, its $1-$hop neighborhood $\mathcal{N}_u$ is defined as the set of nodes that are edgepoints of edges attached to node u: $\mathcal{N}_u = \{v : (u, v) \in \mathcal{E} \text{ or } (u, v) \in \mathcal{E}\}$. The neighborhood definition is crucial for the mechanics graph learning models, because it formalizes the *paths* through the information propagates. In Figure 2.1 we show a simple example of the $1-$hop neighborhood. Nodes h, f, g are considered $2-$hop neighbors, as starting from node a we need two stops (or *hops*) to reach out these nodes.

Simple Graphs For the sake of simplicity, at many graph learning models, we assume that there are no *self-connections*, i.e edges from and to the same node, there is at most one edge between nodes and that for every edge $(u_i, u_j) \in gE$, there exists, also, the $(u_j, u_i) \in \mathcal{E}$, in other words the graph is *undirected*. In many cases throughout the present manuscript, these assumptions are not satisfied and explicitly mentioned.

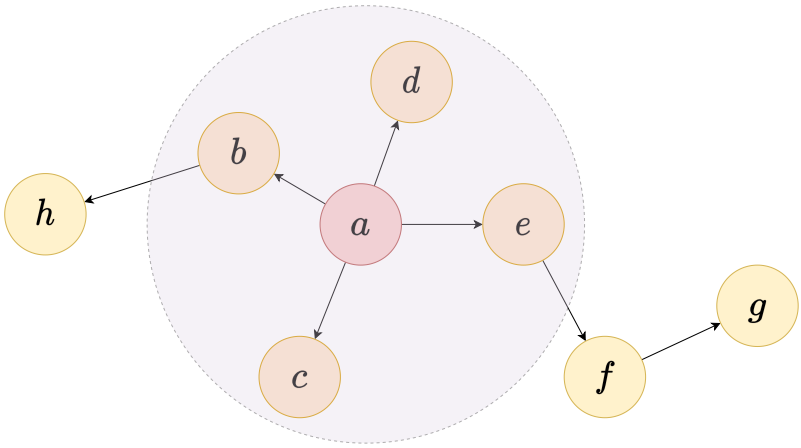


FIGURE 2.1: Example of neighborhood in a graph. The neighborhood of node a equals $\mathcal{N}_a = \{b, c, d, e\}$. Nodes h, f, g are considered as 2−hop neighbors of node a.

Attributes and Labels Many times, modeling a real-world network with a graph is not sufficient due to the contextual information that the network components carry. For example, in a social network, where the nodes refer to users and edges refer to their connections, typical information that needs to be processed can be posts, comments, or reactions of a user. For the encoding of such information, we define the *attribute matrices* $X_v \in \mathbb{R}^{n \times d_v}$ and $X_e \in \mathbb{R}^{m \times d_e}$, where d_v, d_e are the dimensionalities of the node attribute and edge attribute vectors respectively. Depending on the user-end task, graph might contain, also, node-level, edge-level or graph-level *label* information, that is vectors $Y_v \in \mathbb{N}^n$, $Y_e \in \mathbb{N}^m$, $Y_{\mathcal{G}} \in \mathbb{N}$. A typical example of an attributed graph is visualized in Figure 2.2, where we have node attribute vectors and node labels (they are encoded as the colors upon the attribute vectors).

2.1.2 Edge Encoding

The simplest encoding of an edge existence in a graph is the binary one. This leads to the definition of the *adjacency matrix* $A \in \{0, 1\}^{n \times n}$, where $A_{ij} = 1$ if and only if $(i, j) \in \mathcal{E}$. The degrees of $\mathcal{G}$ can now be represented by $D = \text{Diag}(A\mathbf{1}_n)$, where $\mathbf{1}_n$ is a vector of all ones of size n. In the case of an undirected graph, A is a symmetric matrix. Given A, D, we can define the Laplacian matrix $L = D - A$ and its normalization variants: the *symmetric normalized* Laplacian $L_{sym} = I_n - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ and the *random-walk normalized* Laplacian $L_{rw} = I_n - D^{-1} A$. Since L_{sym} is a real symmetric (and, also, positive semi-definite), we can decompose it into its eigenvectors matrix $U \in \mathbb{R}^{n \times n}$ and its eigenvalues matrix $\Lambda = \text{Diag}(\lambda_1, ..., \lambda_n)$, where $\lambda_1, ..., \lambda_n$ are the ordered eigenvalues of L_{sym}:

$$L_{sym} = U \Lambda U^{\top}. \tag{2.1}$$

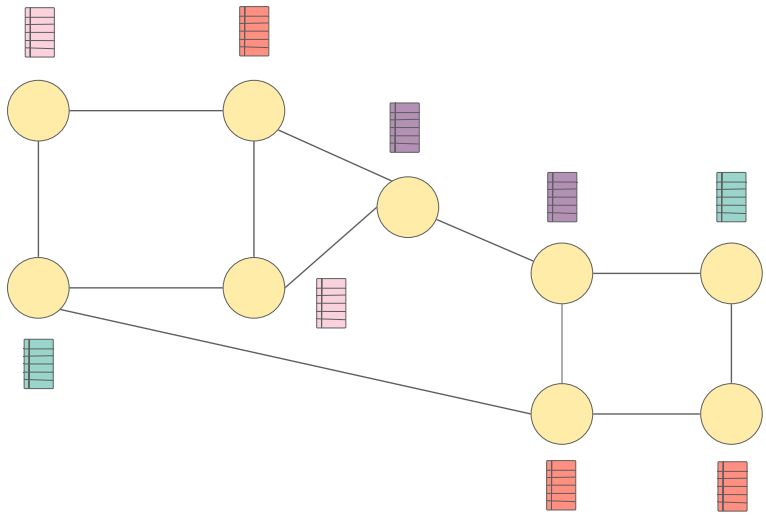


FIGURE 2.2: Example of an attributed graph. Each node has an attribute vector and a label. The label is denoted by the color of the attribute vector.

Given this decomposition of the adjacency information, we can formalize the information propagation in a spectral framework, using the graph convolution, as we will present in Section 2.2.2.

Invariance and equivariance A crucial structural characteristic of graphs is there is no assumption over the *order* of their nodes. That means any operation applied on a graph should not depend on or assume any particular node order. This property is called *permutation invariance* because regardless of any *permutation* of the nodes, the functions that act on the graph should remain unchanged. More formally, let a $n \times n$ matrix P, where each row and each column contain exactly one nonzero element that is equal to one. P is called a permutation matrix because if P is applied to another $n \times n$ matrix (e.g, adjacency A or the Laplacian L) it will permute the rows and the columns (and, consequently, the node labels of a graph). Also, a function f is called permutation invariant if $f(PX) = f(X)$ for all permutation matrices P.

Although the property of *permutation invariance* is desired in the global level of a graph (e.g., aggregating the information from the whole node-set), the majority of graph learning models consist of learning node-level representations, i.e., given a model $\mathcal{M}$ we learn representations $H = \mathcal{M}(X)$, where each row corresponds to the information of a node. As the authors in [31] describe, the rows of H should be aligned with the rows of X and, so, a permutation matrix P acting on X should act similarly, also, on H. This observation creates the need for another property, namely the *permutation equivariance*, that is: $f(PX) = Pf(X)$. In the standard forms of Graph Neural Networks, the node representation is based on neighborhood aggregation functions, and, in order to provide valid

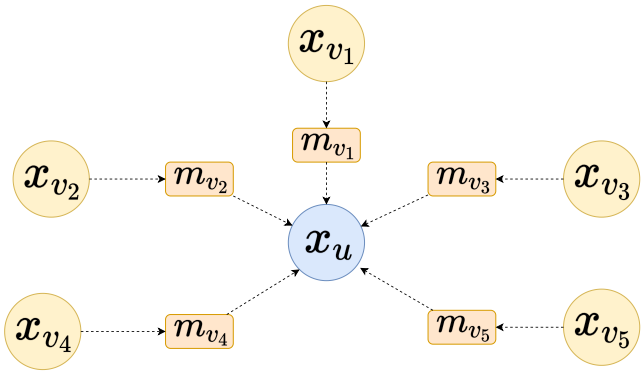


FIGURE 2.3: Message passing aspect of Graph Neural Networks

representations, these aggregation functions need to be permutation equivariant [150, 149].

2.2 Taxonomy of Graph Neural Networks

In the literature, there is a huge variety of formulations that define graph neural network operators based on either iterative or spectral terms [233, 92]. In every case, the core computational step of a GNN is the aggregation of the neighborhood information and its incorporation into a gradient-based optimization process [200]. In an effort to a summarizing framework, Bronstein et al. [31] suggest a categorisation of almost every previously suggested model on three categories: the **message passing**, the **attentional** and the **convolutional** aspect. Next, we give a description of each category with examples of representative models.

2.2.1 Message Passing aspect

According to the message-passing aspect, the information propagation among neighbors corresponds to the computation of arbitrary vectors across edges, called *messages*, as computed below:

$$h_u = \phi(x_u, \mathbf{Agg}(\{\psi(x_u, x_v) \mid v \in \mathcal{N}\})), \tag{2.2}$$

where ϕ, ψ are learnable functions such as MLPs. Specifically, ψ can be seen as a *message function* that describes the vector information sent from node u to node v. Moreover, **Agg** is an aggregation operator that can be chosen among standard operators, i.e., the summation, the average, the weighted average, the max-pooling. The **Agg** operator acts on the *messages* that are sent from a node u to its neighborhood $\mathcal{N}_u$. These types of models are also called *Message Passing Neural Networks* (MPNNs). A visualization of the message passing framework is depicted in Figure 2.3.

Typical examples of this class of models are:

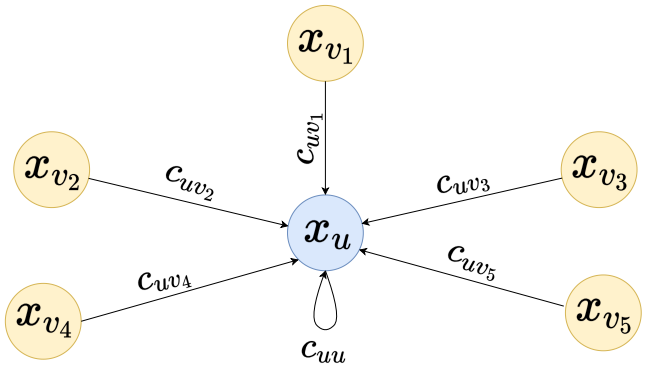


FIGURE 2.4: Convolutional aspect of Graph Neural Networks

- Graph Isomorphism Network [238], where the authors suggest the utilization of summation as the **Agg** operator.
- GraphSAGE [90], in which the **Agg** is chosen to be an average, while ψ is simply an identity function.

2.2.2 Convolutional aspect

According to the convolutional aspect, the neighborhood information is propagated in a direct way, assuming a fixed relation between a node u and its neighbor v. Specifically, the node representation is updated according to:

$$h_u = \phi(x_u, \mathbf{Agg}(\{c_{uv}\psi(x_v) \mid v \in \mathcal{N}\})), \tag{2.3}$$

where ϕ, ψ are learnable functions such as MLPs. Similarly to the message-passing aspect, **Agg** can be chosen among standard operators, i.e., the summation, the average, the weighted average, the max-pooling. We can have a visual description of the node representation computations is shown in Figure 2.4.

Representative models of this aspect are:

- Graph Convolutional Network [118]. It is one of the first works that have proposed graph convolutional models for the task of semi-supervised node classification.
- Simplified Graph Convolution [230], where in contrast to the GCN model, the non-linearities are removed after the aggregation step in order to allow the efficient design of deeper convolutional models.

Graph Convolutions The term *convolutional networks* derives from the relationship of the aforementioned type of networks with notions from graph signal processing [170]. In parallel with the research and design of propagation models on graphs, there was a need for a mathematical formulation of such models from a spectral aspect. Into this context, we

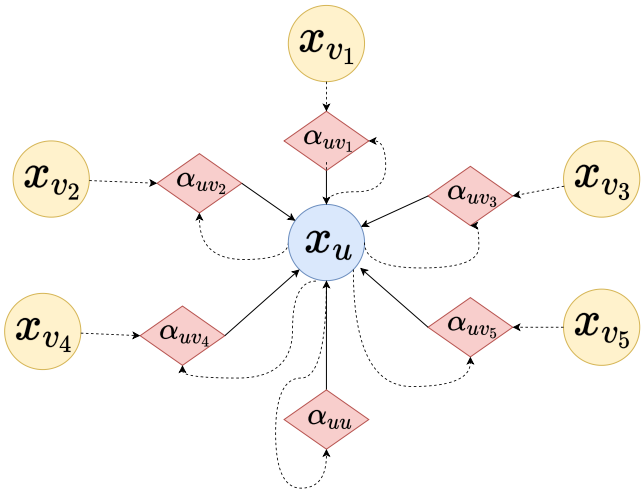


FIGURE 2.5: Attentional aspect of Graph Neural Networks

assume the contextual information of the nodes of a graph as a *graph signal* $x \in \mathbb{R}^n$ (for the sake of simplicity, we assume a scalar signal for each node). Let, also, a *graph filter* $g_\phi \in \mathbb{R}^n$ that is parametrised by ϕ. Then, a graph convolution can be considered as the multiplication in the Fourier domain of the signal x with the filter g_ϕ:

$$g_\phi * x = U g_\phi U^\top x, \tag{2.4}$$

where $U \in \mathbb{R}^{n \times n}$ denotes the matrix of eigenvectors of the normalized Laplacian L_{sym}. Given Equation 2.4, we can see $U^\top x$ as the graph Fourier transform of the input signal x and correspondingly $U\tilde{x}$ as the inverse graph Fourier transform, where $\tilde{x}$ is the result of the graph Fourier transform. Simplifying the graph convolution of Equation 2.4 into an aggregation scheme, we can obtain the update formula of Equation 2.3, where the choice of g_ϕ affects the values of ϕ, ψ and **Agg** functions.

2.2.3 Attentional aspect

In this formulation, the interactions between the nodes are expressed in an implicit manner. Instead of defining fixed relational vectors between nodes or assuming messages that have to be sent, in the *attentional* aspect, we define a self-attention layer that computes the importance coefficients of each edge. Specifically, the node representation is updated as below:

$$h_u = \phi(x_u, \mathbf{Agg}(\{\alpha(x_u, x_v)\psi(x_v) \mid v \in \mathcal{N}\})), \tag{2.5}$$

where $\alpha(\cdot, \cdot)$ describes the self-attention layer. In contrast to the convolutional aspect, the importance vectors are now feature-dependent. In Figure 2.5, we can see the considered interactions in this class of models.

The notation of *graph attention* has been firstly introduced in Veličković et al. [220]. The authors, based on previously introduced works on *self-attention* for sequence-based tasks [219], suggest a similar strategy for

the information exchange in a neighborhood.

2.3 Learning Tasks over Graphs

In learning scenarios over graphs, the main objective is the label prediction of either nodes, edges, or a whole graph. Depending on the label type, we can group the models and tasks to the following categories:

1. **Node-level tasks**: This class of tasks require node-level representations ($H \in \mathbb{R}^{n \times d_v}$). We can meet node-level representations in problems where the objective is a node-level prediction task, that is either *node classification* or *node regression*. In such a scenario, given a subset of labeled nodes, the model makes predictions over the subset of unlabeled nodes. For example, in citation networks as Cora and CiteSeer [153, 78], the nodes are scientific papers, and their labels are their corresponding topics. Node classification/regression are semi-supervised learning tasks [118, 220, 90], where the node-set V is split into the unlabeled and the labeled node sets: $V = V_u \cap V_l$. The objective is to predict the labels of the nodes that belong to V_u, based on the information that they receive from their neighborhoods. It is worth noting that the neighborhood of an unlabeled node can contain both unlabeled and labeled nodes, explaining the semi-supervised learning setup.

2. **Edge-level tasks**: Here, the domain of the required representations spans the space of V^2, which describes the possible pairs among graph nodes ($H \in \mathbb{R}^{m \times d_e}$). Similarly to node classification/regression, in the edge-level prediction, we can meet the task of *edge classification/regression*, where the goal is to predict the label of an edge in a semi-supervised manner [116]. A task that also requires edge-level representations and it has been very popular in recommendation systems is the *link prediction*, where the objective is the prediction of an edge existence between pairs of nodes [248, 253]. The majority of the works that have been successful in the task of the link prediction are based on Graph Auto-encoders (GAE) [119, 93, 193, 192]. GAEs can be considered as an extension of standard GNNs, as they consist of two parts: an *encoder* module, which could be a GNN itself for encoding the original graph representation, and a *decoder* module, that tries to reconstruct the adjacency information of the original graph.

3. **Graph-level tasks**: Finally, there is the case where a single representation for a whole graph ($H \in \mathbb{R}^{d_g}$) is required. We can meet this type of representations in graph-level tasks, such as *graph classification* and *graph regression*, where the goal is the prediction of a graph label. Real-world use-cases of graph classification and regression often come from bioinformatics datasets, where provided a graph structure of a molecule or a protein, the prediction of a chemical or

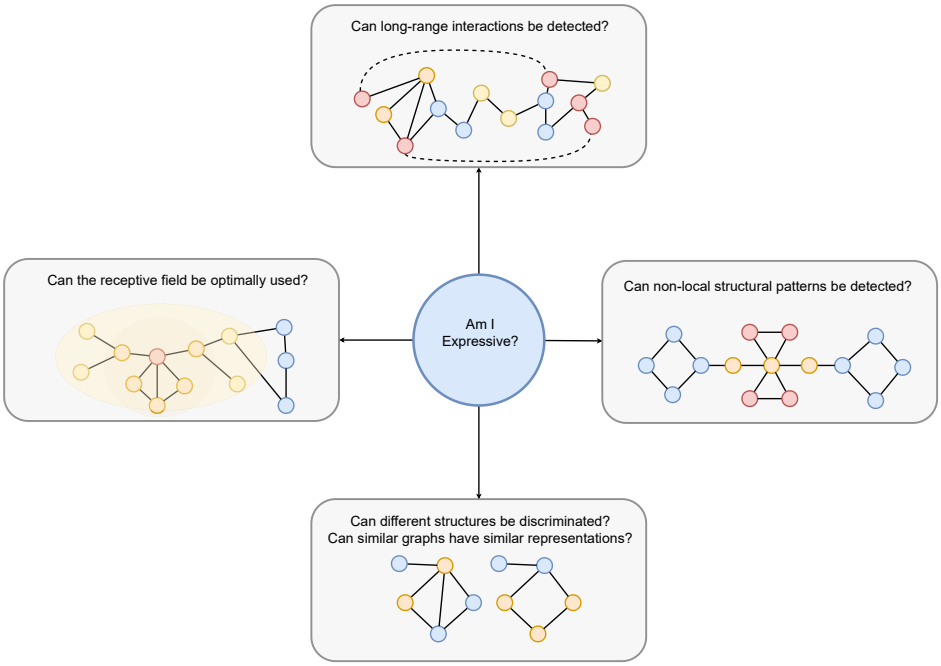


FIGURE 2.6: Key questions for determining whether a node/graph representation is expressive, including the receptive field, the long-range interactions, the non-assortativity and the universality.

molecular property is needed [80, 160, 102, 249]. In these scenarios, a collection of graphs $C_{\mathcal{G}} = \{G_1, ..., G_p\}$ is given that is partitioned into two sets, the labeled and the unlabeled ones, and the objective is to predict the labels of the unlabeled graphs.

2.4 Expressivity in GNNs

One of the main motivations of learning representations for graphs is the projection of the graph connectivity from a non-Euclidean to a Euclidean space, namely the $d-$dimensional real space $\mathbb{R}^d$. Among the research objectives in this field, the ability of the representations to maintain the most information from the initial space seems to dominate the interest of the research community. This ability is characterized by the term *expressivity* or *expressive power* [238], and it becomes more and more important due to the impact on user-end tasks, i.e., the accuracy of graph learning models. The basic criterion for characterizing expressive representations is their power to discriminate different graph structures [149, 9, 149, 163, 199].

In the recent past, the interest in learning rich and informative representations has been expanded in broader areas due to the arising of complex graph structures. These areas include:

a. the design of models that preserve the *isometry*, i.e., the difference of the output embeddings should be bounded by the difference of the input representations [98, 225].

b. the study of the receptive field of GNNs, i.e., the amount of information that a node can process from its direct (1-hop) or indirect (>1-hop) neighborhood [161, 1, 143].

c. the ability to capture *long-range dependencies* in graphs. In cases of networks, where distant nodes can interact with each other or in cases of structural noise, where edges appear as non-existent, graph learning models should be able to memorize information from larger neighborhoods. A lot of recent works focus on building deeper GNN architectures [136, 133, 132] so that implicit information from larger depths is propagated [152, 120]. Such a condition requires the reconsideration of the standard message-passing scheme so that we can capture structural symmetries regardless of the node proximity.

The four different aspects can be summarized into a graph of key questions as visualized in Figure 2.6. In this thesis, we consider all of these aspects under the umbrella of expressivity for GNNs, because they all help towards rich and expressive graph representations for different network scenarios.

Part I

Discrimination Power

Chapter 3

Universal Approximation on Graphs

3.1 Introduction

Learning good representations is seen by many machine learning researchers as the main reason behind the tremendous successes of the field in recent years [21]. In image analysis [124], natural language processing [219] or reinforcement learning [158], groundbreaking results rely on efficient and flexible deep learning architectures that are capable of transforming a complex input into a simple vector while retaining most of its valuable features. The *universal approximation theorem* [51, 101, 100, 178] provides a theoretical framework to analyze the expressive power of such architectures by proving that, under mild hypotheses, multi-layer perceptrons (MLPs) can uniformly approximate any continuous function on a compact set. This result provided a first theoretical justification of the strong approximation capabilities of neural networks, and was the starting point of more refined analyses providing valuable insights into the generalization capabilities of these architectures [19, 77, 198, 18].

Despite a large literature and state-of-the-art performance on benchmark graph classification datasets, graph neural networks need a similar theoretical foundation [238]. The universality for these architectures is either hinted at via equivalence with approximate graph isomorphism tests (k-WL tests in [238, 149]) or proved under restrictive assumptions (finite node attribute space in [164]). We introduce Colored Local Iterative Procedure (CLIP), which tackles the limitations of current GNNs under the message passing aspect, presented in Section 2.2.1. This type of GNNs is also called Message Passing Neural Networks (MPNNs). We show, both theoretically and experimentally, that adding a simple coloring scheme can improve the flexibility and power of these graph representations. More specifically, the contributions of this chapter are: 1) we provide a precise mathematical definition for universal graph representations, 2) we present a general mechanism to design universal neural networks using separability, 3) we propose a novel node coloring scheme

leading to CLIP, the first provably universal extension of MPNNs, 4) we show that CLIP achieves state-of-the-art results on benchmark datasets while significantly outperforming traditional MPNNs as well as recent methods on graph property testing.

The rest of this chapter is organized as follows: Section 3.2 gives an overview of the literature related to the universality arguments and connections with graph representation learning. Section 3.3 provides a precise definition for universal representations, as well as a generic method to design them using *separable* neural networks. In Section 3.4, we show that most state-of-the-art representations are not sufficiently expressive to be universal. Then, using the analysis of Section 3.3, Section 3.5 provides CLIP, a provably universal extension of MPNNs. Finally, Section 3.7 shows that CLIP achieves state-of-the-art accuracies on benchmark graph classification tasks, as well as outperforming its competitors on graph property testing problems.

3.2 Advances on universality and graph representation learning

As we have seen in Section 1.2, the first works investigating the use of neural networks for graphs used recurrent neural networks to represent directed acyclic graphs [213, 71]. More generic graph neural networks were later introduced by [83, 200] without showing that universality conditions can apply. Although a lot of spectral-based (the convolutional aspect in Section 2.2.2) and attention-based (the attentional aspect in Section 2.2.3) have been introduced [34, 94, 60, 118], here we focus in the message passing-based models (MPNNs), that assume an aggregation of neighborhood information through a local iterative process. This category contains most state-of-the-art graph representation methods such as [64, 86, 130, 244, 221, 72], DeepWalk [175], graphSAGE [90] or GIN [238].

Recently, Xu et al. [238] showed that MPNNs were, at most, as expressive as the Weisfeiler-Lehman (WL) test for graph isomorphism [229]. This surprising result led to several works proposing MPNN extensions to improve their expressivity, and ultimately tend towards *universality* [149, 150, 164, 42]. However, these graph representations are either as powerful as the k-WL test [149], or provide universal graph representations under the restrictive assumption of finite node attribute space [164]. Other recent approaches [150, 9] imply higher orders of tensors in the size of the considered graphs. Some more powerful GNNs are studied and benchmarked on real classical datasets and on graph property testing [123, 164, 42]: a set of problems that classical MPNNs cannot handle. Our work thus provides a more general and powerful result of universality, matching the original definition of [51] for MLPs.

3.3 Universal representations via separability

In this section we present the theoretical tools used to design our universal graph representation. More specifically, we show that *separable* representations are sufficiently flexible to capture all relevant information about a given object, and may be extended into universal representations.

3.3.1 Notations and basic assumptions

Let $\mathcal{X}, \mathcal{Y}$ be two topological spaces, then $\mathcal{F}(\mathcal{X}, \mathcal{Y})$ (resp. $\mathcal{C}(\mathcal{X}, \mathcal{Y})$) denotes the space of all functions (resp. continuous functions) from $\mathcal{X}$ to $\mathcal{Y}$. Moreover, for any group G acting on a set $\mathcal{X}$, $\mathcal{X}/G$ denotes the set of orbits of $\mathcal{X}$ under the action of G. Finally, $\|\cdot\|$ is a norm on $\mathbb{R}^d$, and $\mathcal{P}_n$ is the set of all permutation matrices of size n. In what follows, we assume that all the considered topological spaces are *Hausdorff* (see e.g. [29] for an in-depth review): each pair of distinct points can be separated by two disjoint open sets. This assumption is rather weak (e.g. all metric spaces are Hausdorff) and is verified by most topological spaces commonly encountered in the field of machine learning.

3.3.2 Universal representations

Let $\mathcal{X}$ be a set of objects (e.g. vectors, images, graphs, or temporal data) to be used as input information for a machine learning task (e.g. classification, regression or clustering). In what follows, we denote as *vector representation* of $\mathcal{X}$ a function $f : \mathcal{X} \to \mathbb{R}^d$ that maps each element $x \in \mathcal{X}$ to a d-dimensional vector $f(x) \in \mathbb{R}^d$. A standard setting for supervised representation learning is to define a class of vector representations $\mathfrak{F}_d \subset \mathcal{F}(\mathcal{X}, \mathbb{R}^d)$ (e.g. convolutional neural networks for images) and use the target values (e.g. image classes) to learn a *good* vector representation in light of the supervised learning task (i.e. one vector representation $f \in \mathfrak{F}_d$ that leads to a good accuracy on the learning task). In order to present more general results, we will consider neural network architectures that can output vectors of any size, i.e. $\mathfrak{F} \subset \cup_{d\in\mathbb{N}^*}\mathcal{F}(\mathcal{X}, \mathbb{R}^d)$, and will denote $\mathfrak{F}_d = \mathfrak{F} \cap \mathcal{F}(\mathcal{X}, \mathbb{R}^d)$ the set of d-dimensional vector representations of $\mathfrak{F}$. A natural characteristic to ask from the class $\mathfrak{F}$ is to be generic enough to approximate any vector representation, a notion that we will denote as *universal representation* [101].

Definition 1. A class of vector representations $\mathfrak{F} \subset \cup_{d\in\mathbb{N}^*}\mathcal{F}(\mathcal{X}, \mathbb{R}^d)$ is called a *universal representation* of $\mathcal{X}$ if for any compact subset $K \subset \mathcal{X}$ and $d \in \mathbb{N}^*$, $\mathcal{F}$ is uniformly dense in $\mathcal{C}(K, \mathbb{R}^d)$.

In other words, $\mathfrak{F}$ is a universal representation of a normed space $\mathcal{X}$ if and only if, for any continuous function $\phi : \mathcal{X} \to \mathbb{R}^d$, any compact

$K \subset \mathcal{X}$ and any $\varepsilon > 0$, there exists $f \in \mathfrak{F}$ such that

$$\forall x \in K, \ \|\phi(x) - f(x)\| \leqslant \varepsilon \,. \tag{3.1}$$

One of the most fundamental theorems of neural network theory states that one hidden layer MLPs are universal representations of the m-dimensional vector space $\mathbb{R}^m$.

Theorem 1 ([178, Theorem 3.1]). *Let $\varphi : \mathbb{R} \to \mathbb{R}$ be a continuous non polynomial activation function. For any compact $K \subset \mathbb{R}^m$ and $d \in \mathbb{N}^*$, two layers neural networks with activation φ are uniformly dense in the set $\mathcal{C}(K, \mathbb{R}^d)$.*

However, for graphs and structured objects, universal representations are hard to obtain due to their complex structure and invariance to a group of transformations (e.g. permutations of the node labels). We show in this chapter that a key topological property, *separability*, may lead to universal representations of those structures.

3.3.3 Separability is (almost) all you need

Loosely speaking, universal representations can approximate any vector-valued function. It is thus natural to require that these representations are *expressive* enough to separate each pair of dissimilar elements of $\mathcal{X}$.

Definition 2 (Separability). A set of functions $\mathfrak{F} \subset \mathcal{F}(\mathcal{X}, \mathcal{Y})$ is said to *separate* points of $\mathcal{X}$ if for every pair of distinct points x and y, there exists $f \in \mathfrak{F}$ such that $f(x) \neq f(y)$.

For a class of vector representations $\mathfrak{F} \subset \cup_{d \in \mathbb{N}^*} \mathcal{F}(\mathcal{X}, \mathbb{R}^d)$, we will say that $\mathfrak{F}$ is *separable* if its 1-dimensional representations $\mathfrak{F}_1$ separates points of $\mathcal{X}$. Separability is rather weak, as we only require the existence of different outputs for every pair of inputs. Unsurprisingly, we now show that it is a necessary condition for universality.

Proposition 1. *Let $\mathfrak{F}$ be a universal representation of $\mathcal{X}$, then $\mathfrak{F}_1$ separates points of $\mathcal{X}$.*

Proof of Proposition 1. Assume that there exists $x, y \in \mathcal{X}$ s.t. $\forall f \in \mathfrak{F}_1$, $f(x) = f(y)$. Then $K = \{x, y\}$ is a compact subset of $\mathcal{X}$ and let $\phi \in \mathcal{C}(K, \mathbb{R})$ be such that $\phi(x) = 1$ and $\phi(y) = 0$. Thus, for all $f \in \mathfrak{F}_1$, $\max_{z \in \{x,y\}} \|\phi(z) - f(z)\| \geqslant 1/2$ which contradicts universality (see Definition 1). □

While separability is necessary for universal representations, it is also key to designing neural network architectures that can be extended into universal representations. More specifically, under technical assumptions, separable representations can be composed with a universal representation of $\mathbb{R}^d$ (such as MLPs) to become universal.

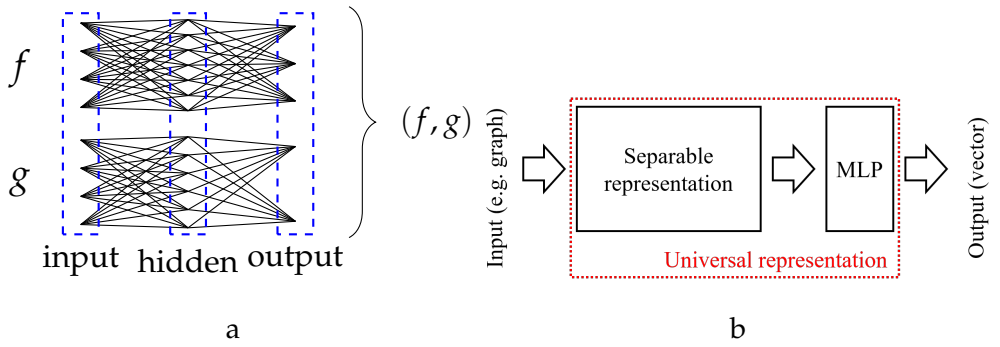


FIGURE 3.1: In Figure a, the concatenation of two MLPs f and g is visualized. In Figure b, universal representations can easily be created by combining a separable representation with an MLP.

Theorem 2. *For all $d \geqslant 0$, let $\mathcal{M}_d$ be a universal approximation of $\mathbb{R}^d$. Let $\mathfrak{F}$ be a class of vector representations of $\mathcal{X}$ such that:*

1. ***Continuity:*** *every $f \in \mathfrak{F}$ is continuous,*
2. ***Stability by concatenation:*** *for all $f, g \in \mathfrak{F}$, $x \mapsto (f(x), g(x)) \in \mathfrak{F}$,*
3. ***Separability:*** *$\mathfrak{F}_1$ separates points of $\mathcal{X}$.*

Then $\{\psi \circ f \ : \ \exists d \geqslant 1 \text{ s.t. } \psi \in \mathcal{M}_d, f \in \mathfrak{F}\}$ is a universal representation of $\mathcal{X}$.

Stability by concatenation is verified by most neural networks architectures, as illustrated for MLPs in Figure 3.1. The proof of Theorem 2 relies on the Stone-Weierstrass theorem (see e.g. [190, Theorem 7.32]) whose assumptions are continuity, separability, and the fact that the class of functions is an algebra. Fortunately, composing a separable and concatenable representation with a universal representation automatically leads to an algebra, and thus the applicability of the Stone-Weierstrass theorem and the desired result. Next, a complete proof of Theorem 2 is provided. First, a definition of the Stone-Weierstrass theorem is given.

Theorem 3 (Stone-Weierstrass)**.** *Let $\mathcal{A}$ be an algebra of real functions on a compact Hausdorff set K. If $\mathcal{A}$ separates points of K and contains a non-zero constant function, then $\mathcal{A}$ is uniformly dense in $\mathcal{C}(K, \mathbb{R})$.*

We verify that under the assumptions of Theorem 2 the Stone-Weierstrass theorem applies. In this setting, we first prove the theorem for $m = 1$ and use induction for the general case.

Let $K \subset \mathcal{X}$ be a compact subset of $\mathcal{X}$. We will denote

$$\mathcal{A}_0 = \{\psi \circ f \ : \ \exists d \geqslant 1 \text{ s.t. } \psi \in \mathcal{C}(\mathbb{R}^d, \mathbb{R}), f \in \mathfrak{F}\},$$

and will proceed in two steps: we first show that $\mathcal{A}_0$ is uniformly dense in $\mathcal{C}(K, \mathbb{R})$, then that $\mathcal{A}$ is dense in $\mathcal{A}_0$, hence proving Theorem 2.

Lemma 1. *$\mathcal{A}_0$ is a subalgebra of $\mathcal{C}(K, \mathbb{R})$.*

Proof. The subset $\mathcal{A}_0$ contains zero and all constants. Let $f, g \in \mathcal{A}_0$ so that

$$f(x) = \psi_f \circ \varphi_f(x), \qquad g(x) = \psi_g \circ \varphi_g(x),$$

with $\psi_f : \mathbb{R}^{d_f} \to \mathbb{R}$ and $\psi_g : \mathbb{R}^{d_g} \to \mathbb{R}$. Consider $\psi : \mathbb{R}^{d_f+d_g} \to \mathbb{R}$ such that $\psi(a, b) = \psi_f(a) + \psi_g(b)$. We define $\varphi(x) = (\varphi_f(x), \varphi_g(x)) \in \mathbb{R}^{d_f+d_g}$ and by assumption $\varphi \in \mathfrak{F}$. We have

$$\begin{aligned}(f + g)(x) &= \psi(\varphi_f(x), \varphi_g(x)) \\ &= \psi \circ \varphi(x)\end{aligned}$$

so that $f + g \in \mathcal{A}_0$ and we conclude that $\mathcal{A}_0$ is a vectorial subspace of $\mathcal{C}(K, \mathbb{R})$. We proceed similarly for the product in order to finish the proof of the lemma. □

Because $\mathfrak{F}_1$ separates the points of $\mathcal{X}$ by assumption, $\mathcal{A}_0$ also separates the points of $\mathcal{X}$. Indeed, let $x \neq y$ two distinct points of X so that $\exists f \in \mathfrak{F}$ such that $f(x) \neq f(y)$. There exists $g \in \mathcal{C}(\mathbb{R}^d, \mathbb{R})$ such that $g(f(x)) \neq g(f(y))$. From Theorem 3 we deduce that $\mathcal{A}_0$ is uniformly dense in $\mathcal{C}(K, \mathbb{R})$ for all compact subsets $K \subset \mathcal{X}$.

Finally we state that:

Lemma 2. *For any compact subset $K \subset \mathcal{X}$, $\mathcal{A}$ is uniformly dense in $\mathcal{A}_0$.*

Proof. Let $\epsilon > 0$ and $h = \psi_0 \circ f \in \mathcal{A}_0$ with $f \in \mathfrak{F}$ and $\psi_0 \in \mathcal{C}(\mathbb{R}^d, \mathbb{R})$. Thanks to the continuity of f, the image $\tilde{K} = f(K)$ is a compact of $\mathbb{R}^d$. By Theorem 1 there exists an MLP ψ such that $\|\psi - \psi_0\|_{\tilde{K},\infty} \leqslant \epsilon$. We have $\psi \circ f \in \mathcal{A}$ and $\|\psi_0 \circ f - \psi \circ f\|_{K,\infty} \leqslant \epsilon$ which concludes the proof. □

This last lemma completes the proof in the case $m = 1$. For $m \geqslant 2$ consider $\mathcal{A}_0 = \{\psi \circ f \ : \ \exists d \geqslant 1 \text{ s.t. } \psi \in \mathcal{C}(\mathbb{R}^d, \mathbb{R}^m), f \in \mathfrak{F}\}$ and proceed in a similar manner than Lemma 2 by decomposing $\psi \in \mathcal{C}(\mathbb{R}^d, \mathbb{R}^m)$ as

$$\psi(x) = \begin{pmatrix} \psi_1(x) \\ \psi_2(x) \\ \vdots \\ \psi_m(x) \end{pmatrix},$$

and applying Lemma 1 for each coefficient function $\psi_i \in \mathcal{C}(\mathbb{R}^d, \mathbb{R})$.

Since MLPs are universal representations of $\mathbb{R}^d$, Theorem 2 implies a convenient way to design universal representations of more complex object spaces: create a separable representation and compose it with a simple MLP (see Figure 3.1).

Corollary 1. *A continuous, concatenable and separable representation of $\mathcal{X}$ composed with an MLP is universal.*

Note that many neural networks of the deep learning literature have this two steps structure, including classical image CNNs such as AlexNet [124] or Inception [216]. Here, we use Corollary 1 to design universal graph and neighborhood representations, although the method is much more generic and may be applied to other objects.

3.4 Limitations of existing representations

In this section, we first provide a proper definition for graphs with node attributes under the formulation of Hausdorff spaces, needed for the proofs of universality, and then we show that message passing neural networks are not sufficiently expressive to be universal.

3.4.1 Graphs with node attributes

Consider a dataset of n interacting objects (e.g. users of a social network) in which each object $i \in [\![1, n]\!]$ has a vector attribute $v_i \in \mathbb{R}^m$ and is a node in an undirected graph G with adjacency matrix $A \in \mathbb{R}^{n \times n}$.

Definition 3. The space of graphs of size n with m-dimensional node attributes is the quotient space

$$\mathbf{Graph}_{m,n} = \left\{(v, A) \in \mathbb{R}^{n \times m} \times \mathbb{R}^{n \times n}\right\} / \mathcal{P}_n \,, \tag{3.2}$$

where A is the adjacency matrix of the graph, v contains the m-dimensional representation of each node in the graph and the set of permutations matrices $\mathcal{P}_n$ is acting on (v, A) by

$$\forall P \in \mathcal{P}_n, \quad P \cdot (v, A) = (Pv, PAP^\top) \,. \tag{3.3}$$

Moreover, we limit ourselves to graphs of maximum size $n_{\max}$, where $n_{\max}$ is a large integer. This allows us to consider functions on graphs of different sizes without obtaining infinite dimensional spaces and infinitely complex functions that would be impossible to learn via a finite number of samples. We thus define $\mathbf{Graph}_m = \bigcup_{n \leqslant n_{\max}} \mathbf{Graph}_{m,n}$. More details on the technical topological aspects of the definition are available in Appendix B.1, as well as a proof that $\mathbf{Graph}_m$ is Hausdorff.

3.4.2 Message passing neural networks

A common method for designing graph representations is to rely on local iterative procedures. Following the notations of Xu et al. [238], a *message passing neural network* (MPNN) [80] is made of three consecutive phases that will create intermediate node representations $x_{i,t}$ for each node $i \in [\![1, n]\!]$ and a final graph representation x_G as described by the following procedure: 1) **Initialization:** All node representations are initialized with their node attributes $x_{i,0} = v_i$. 2) **Aggregation and combination:** T

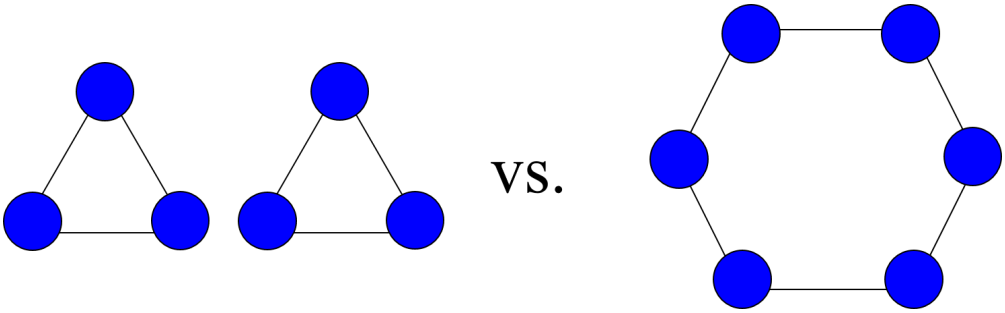


FIGURE 3.2: With identical node attributes, an hexagone and two triangles are indistinguishable using MPNNs.

local iterative steps are performed in order to capture larger and larger structural characteristics of the graph. 3) **Readout:** This step combines all final node representations into a single graph representation: $x_G = \text{READOUT}(\{x_{i,T}\}_{i \in [\![1,n]\!]})$, where READOUT is permutation invariant.

Unfortunately, MPNNs are not expressive enough to separate all graphs [238] and hence will not lead to universal representations. While these representations can be as expressive as the Weisfeiler-Lehman algorithm [229], they are not sufficiently expressive to construct isomorphism tests. For example, MPNNs cannot distinguish k-regular graph without or with identical node attributes. In order to prove this, we now show that k-regular graphs of size n cannot be separated using MPNNs (see Figure 3.2 for a simple example of such graphs).

Lemma 3. *Let $\mathfrak{F}$ be an MPNN and G_1, G_2 two k-regular graphs of size n and identical node attributes $v \in \mathbb{R}^d$, then $\forall f \in \mathfrak{F}$, $f(G_1) = f(G_2)$.*

Proof. Since all nodes have the same attribute, the initial representation of all nodes $v_i = v$ is identical. After each local aggregation, all node representations are equal to $x_{i,t+1} = f_t(f_{t-1}(...f_2(f_1(v))...)$, where $f_t(x) = \text{AGG\&COMB}^{(t)}(x, \{x, ..., x\})$. As a result, both representations are equal and $f(G_1) = f(G_2)$. □

Using Proposition 1, this directly implies that MPNNs are not universal, as they do not separate regular graphs.

Proposition 2. *MPNNs are not universal.*

Proof. Figure 3.2 shows two simple graphs that are undistinguishable using MPNNs. More specifically, each neighborhood is the same, and thus all aggregation steps will output the same value. In other words, MPNNs do not separate the two graphs of Figure 3.2, and are thus not universal using Proposition 1. □

Proof. The proof consists in showing that two very simple graphs are not separable using any MPNN. Let G_1 be an undirected hexagon, G_2

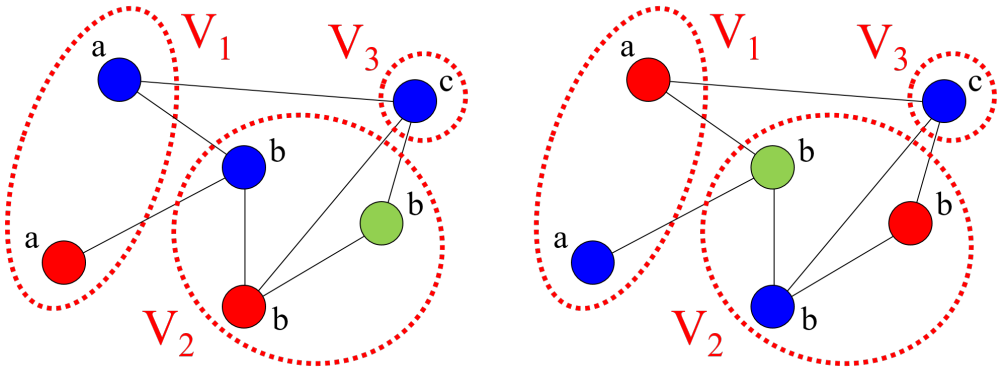


FIGURE 3.3: Example of two valid colorings of the same attributed graph. Note that each V_k contains nodes with identical attributes.

two undirected triangles, and $v_1 = v_2 = 0$ (see Figure 3.2. Then, after each local aggregation, all node representations are equal to

$$x_{i,t+1} = f_t(f_{t-1}(...f_2(f_1(0))...)\,, \tag{3.4}$$

where $f(x) = \text{AGG\&COMB}^{(t)}(x, \{x, x\})$. As a result, both representations are equal $x_{G_1} = x_{G_2}$ and MPNNs are not separable. □

3.5 Extending MPNNs using a simple coloring scheme

In this section, we present Colored Local Iterative Procedure (CLIP), an extension of MPNNs using colors to differentiate identical node attributes, that is able to capture more complex structural graph characteristics than traditional MPNNs. This is proved theoretically through a universal approximation theorem in Section 3.6.1 and experimentally in Section 3.7. CLIP is based on three consecutive steps: 1) graphs are colored with several different colorings, 2) a neighborhood aggregation scheme provides a vector representation for each colored graph, 3) all vector representations are combined to provide a final output vector. We now provide more information on the coloring scheme.

3.5.1 Colors to differentiate nodes

In order to distinguish non-isomorphic graphs, our approach consists in coloring nodes of the graph with identical attributes. This idea is inspired by classical graph isomorphism algorithms that use colors to distinguish nodes [154], and may be viewed as an extension of one-hot encodings used for graphs without node attributes [238].

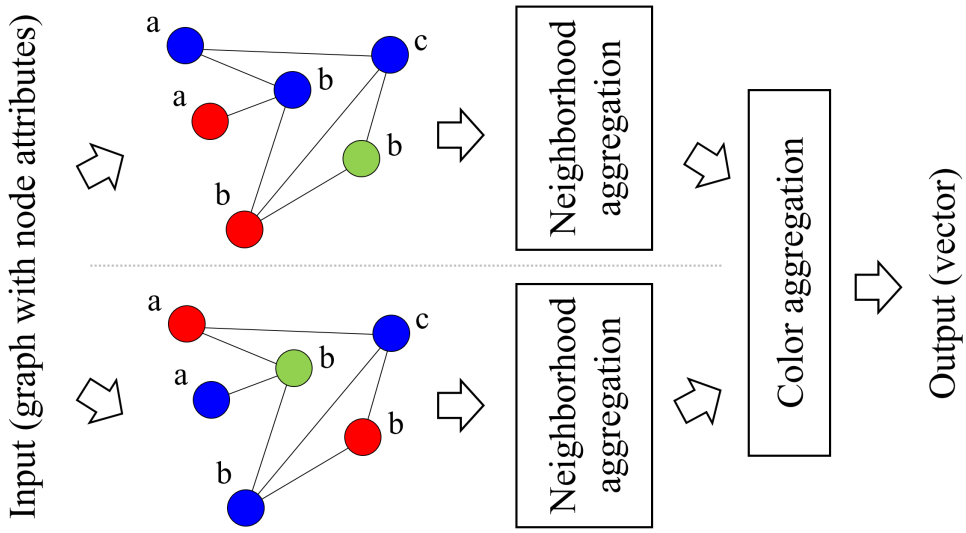


FIGURE 3.4: Schematic representation of CLIP.

For any $k \in \mathbb{N}$, let C_k be a finite set of k colors. These colors may be represented as one-hot encodings (C_k is the natural basis of $\mathbb{R}^k$) or more generally any finite set of k elements. At initialization, we first partition the nodes into groups of identical attributes $V_1, ..., V_K \subset [\![1, n]\!]$. Then, for a subset V_k of size $|V_k|$, we give to each of its nodes a distinct color from C_k (hence a subset of size $|V_k|$). For example, Figure 3.3 shows two colorings of the same graph, which is decomposed in three groups V_1, V_2 and V_3 containing nodes with attributes a, b and c respectively. Since V_1 contains only two nodes, a coloring of the graph will attribute two colors ($(1, 0)$ and $(0, 1)$, depicted as *blue* and *red*) to these nodes. More precisely, the set of colorings $\mathcal{C}(v, A)$ of a graph $G = (v, A)$ are defined as

$$\mathcal{C}(v, A) = \left\{ (c_1, ..., c_n) \ : \ \forall k \in [\![1, K]\!], (c_i)_{i \in V_k} \text{ is a permutation of } C_{|V_k|} \right\}. \tag{3.5}$$

3.5.2 The CLIP algorithm

In the CLIP algorithm, we add a coloring scheme to an MPNN in order to distinguish identical node attributes. This is achieved by modifying the initialization and readout phases of MPNNs as follows.

1. **Colored initialization:** We first select a set $\mathcal{C}_k \subseteq \mathcal{C}(v, A)$ of k distinct colorings uniformly at random (see Equation (3.5)). Then, for each coloring $c \in \mathcal{C}_k$, node representations are initialized with their node attributes concatenated with their color: $x^c_{i,0} = (v_i, c_i)$.

2. **Aggregation and combination:** This step is performed for all colorings $c \in \mathcal{C}_k$ using a universal set representation as the aggregation function: $x^c_{i,t+1} = \psi^{(t)}\left(x^c_{i,t}, \sum_{j \in \mathcal{N}_i} \varphi^{(t)}(x^c_{j,t})\right)$, where ψ and φ are MLPs with continuous non-polynomial activation functions and

$\psi(x, y)$ denotes the result of ψ applied to the concatenation of x and y. The aggregation scheme we propose is closely related to DeepSet [247], and a direct application of Corollary 1 proves the universality of our architecture.

3. **Colored readout:** This step performs a maximum overall possible colorings in order to obtain a final *coloring-independent* graph representation. In order to keep the stability by concatenation, the maximum is taken coefficient-wise

$$x_G = \psi\left(\max_{c \in \mathcal{C}_k} \sum_{i=1}^{n} x_{i,T}^c\right), \tag{3.6}$$

where ψ is an MLP with continuous non polynomial activation functions.

We treat k as a hyper-parameter of the algorithm and call k-CLIP (resp. ∞-CLIP) the algorithm using k colorings (resp. all colorings, i.e. $k = |\mathcal{C}(v, A)|$). Note that, while our focus is graphs with node attributes, the approach used for CLIP is easily extendable to similar data structures such as directed or weighted graphs with node attributes, graphs with node labels, graphs with edge attributes or graphs with additional attributes at the graph level. A visualization of CLIP is presented in Figure 3.4.

3.6 Universality of the node aggregation scheme

We now provide more details on the aggregation and combination scheme of CLIP and show that a simple application of Corollary 1 is sufficient to prove its universality for node neighborhoods. Each local aggregation step takes as input a couple $(x_i, \{x_j\}_{j \in \mathcal{N}_i})$ where $x_i \in \mathbb{R}^m$ is the representation of node i, and $\{x_j\}_{j \in \mathcal{N}_i}$ is the set of vector representations of the neighbors of node i. In the following, we show how to use Corollary 1 to design universal representations for node neighborhoods.

Definition 4. The set of node neighborhoods for m-dimensional node attributes is defined as

$$\textbf{Neighborhood}_m = \mathbb{R}^m \times \bigcup_{n \leqslant n_{\max}} \left(\mathbb{R}^{n \times m} / \mathcal{P}_n\right), \tag{3.7}$$

where the set of permutation matrices $\mathcal{P}_n$ is acting on $\mathbb{R}^{n \times m}$ by $P \cdot v = Pv$.

The main difficulty in designing universal neighborhood representations is that the node neighborhoods of Definition 4 are permutation invariant w.r.t. neighboring node attributes, and hence require permutation invariant representations. The graph neural network literature already contains several deep learning architectures for permutation invariant sets [88, 181, 247, 238], among which PointNet and DeepSet have the

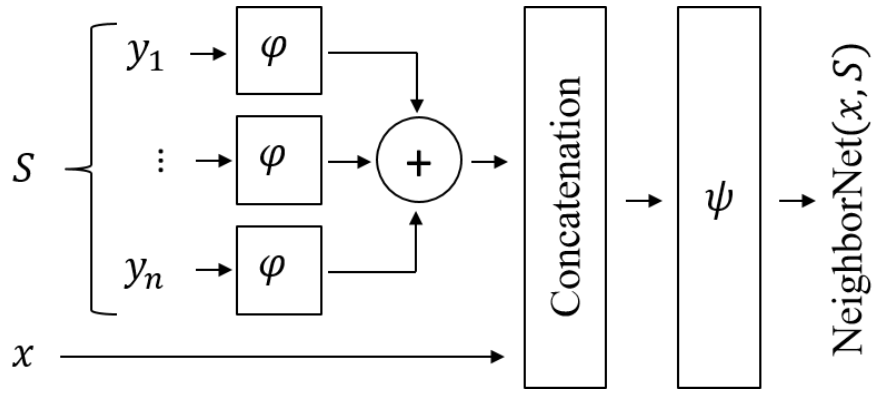


FIGURE 3.5: Diagram for NODEAGGREGATION (here denoted as NeighborNet), where ϕ and ψ are MLPs.

notable advantage of being provably universal for sets. Following Corollary 1, we compose a separable permutation invariant network with an MLP that will aggregate both information from the node itself and its neighborhood. While our final architecture is similar to Deepset [247], this section emphasizes that the general universality theorems of Section 3.3 are easily applicable in many settings including permutation invariant networks. The permutation invariant set representation used for the aggregation step of CLIP is as follows:

$$\text{NODEAGGREGATION}(x, S) = \psi\left(x, \sum_{y \in S} \varphi(y)\right), \tag{3.8}$$

where ψ and φ are MLPs with continuous non-polynomial activation functions and $\psi(x, y)$ denotes the result of the MLP ψ applied to the concatenation of x and y (see Figure 3.5).

Theorem 4. *The set representation described in Equation (3.8) is a universal representation of* $\textbf{Neighborhood}_m$.

Proof. By construction, NODEAGGREGATION is a continuous and concatenable representation. Moreover, its final stage is an MLP, and we thus only have to prove separability in order to use Corollary 1 and prove universality. Let $(x^1, S^1), (x^2, S^2) \in \textbf{Neighborhood}_m$ and suppose that $(x^1, S^1) \neq (x^2, S^2)$. First, if $x^1 \neq x^2$, the final MLP ψ can separate x^1 and x^2. Otherwise, $S^1 \neq S^2$, and let us assume that $S^1 \backslash S^2 \neq \varnothing$ (otherwise $S^2 \backslash S^1 \neq \varnothing$ and the argument is identical). Since MLPs are universal representations of $\mathbb{R}^m$, there exists an MLP φ such that, $\forall s \in S^1 \cup S^2$,

$$\begin{aligned} \varphi(s) &\geqslant 1 \quad \text{if } s \in S^1 \backslash S^2\,, \\ |\varphi(s)| &\leqslant \varepsilon \quad \text{otherwise}\,, \end{aligned}$$

Taking $\psi(x, y) = y$ and $\varepsilon = 1/3 \max\{|S^1|, |S^2|\}$, we have

$$\begin{aligned} \textsc{NodeAggregation}(x^1, S^1) &\geqslant 2/3\,, \\ \textsc{NodeAggregation}(x^2, S^2) &\leqslant 1/3\,, \end{aligned}$$

which proves separability and, using Corollary 1, the universality of the representation. □

3.6.1 Universality of CLIP

As the colorings are chosen at random, the CLIP representation is itself random as soon as $k < |\mathcal{C}(v, A)|$, and the number of colorings k will impact the variance of the representation. However, the representations provided by ∞-CLIP are deterministic and permutation invariant, as MPNNs are permutation invariant. The separability is less trivial and is ensured by the coloring scheme.

Theorem 5. *The ∞-CLIP algorithm with one local iteration ($T = 1$) is a universal representation of the space $\mathbf{Graph}_m$ of graphs with node attributes.*

In order to prove Theorem 5, we will be based on Corollary 1. Specifically, we need to prove the three conditions of continuity, concatenability and separability of the representations, in order to provide universality. In short, we will proceed in the proof by fixing a coloring on one graph and identifying all nodes and edges of the second graph using the fact that all pairs (v_i, c_i) are dissimilar in cases of different graphs. Next, the detailed proof of Theorem 5 is provided.

Proof of Theorem 5. The first direct observation is that CLIP provides continuous and concatenable representations. This is true, due to the constitution of the MLPs that are used in the model, that are both continuous and concatenable. So, we need to focus on the separability condition, and more specifically, whether the final output representation from CLIP can be different for any pair of different graphs. Proceeding with investigating the separability condition, we firstly take into account that the node aggregation step (denoted NODEAGGREGATION below) is a universal set representation, it is capable of approximating any continuous function. We will thus first replace this function by a continuous function g, and then show that the result still holds for $\textsc{NodeAggregation}^{(1)}$ by a simple density argument. We remind here for the sake of clarity that:

$$\textsc{NodeAggregation}(x, S) = \psi\left(x, \sum_{y \in S} \varphi(y)\right),$$

thus finding proper instances of ϕ, ψ will help us towards the proof. Let $G^1 = (v^1, A^1)$ and $G^2 = (v^2, A^2)$ be two distinct graphs of respective

sizes n_1 and n_2 (up to a permutation). Our goal is to find appropriate NODEAGGREGATION$(\cdot,\cdot)$ functions, that will provide different results for the two graphs. We can identify two cases:

- If $n^1 \neq n^2$, then for the choice of NODEAGGREGATION parameters we select: $\psi(x) = x$ and $\phi(x) = 1$. Thus, not caring about the disambiguation scheme in this, the final graph representations after the application of the readout function will be dependent on the number of nodes n^1 and n^2 respectively, yielding $x_{G^1} \neq x_{G^2}$.
- If $n^1 = n^2$, we will assume a fixed coloring of graph G^1, let the c^1. Let, also, $V = \{v_i^k\}_{i\in[\![1,n^1]\!],k\in\{1,2\}}$ be the set of node attributes of G^1 and G^2. Then, again for the choice a proper NODEAGGREGATION, we select: $\psi(x) = x$ and ϕ be a continuous function such that, $\forall x \in V$ and $S \subset V$,

$$\phi(x,S) = \sum_{i=1}^{n^1} \mathbb{1}\{x = (v_i^1, c_i^1)\} \prod_{j\neq i} \mathbb{1}\left\{A_{ij}^1 = \mathbb{1}\{(v_j^1, c_j^1) \in S\}\right\}. \quad (3.9)$$

  This choice of NODEAGGREGATION parameters results in counting the matching neighborhoods. Then, x_G counts the maximum number of matching neighborhoods for the best coloring, and we have $x_{G^1} = n^1$ and $x_{G^2} \leqslant n^1 - 1$. Finally, taking $\varepsilon < 1/2n^1$ in the definition of universal representation leads to the desired result, as then, using an ε-approximation of ϕ as NODEAGGREGATION$^{(1)}$, we have $x_{G^1} > n^1 - 1/2 > x_{G^2}$.

In both cases, the existence of such $g \in \mathcal{C}(\mathbb{R}^m, \mathbb{R})$ is assured by Urysohn's lemma (see e.g. [190, lemma 2.12]). Thus, we found proper choices of g function that separates any pair of different graphs, leading to the conclusion of the proof. □

Similar to the case of MLPs, only one local iteration is necessary to ensure the universality of the representation. This rather counter-intuitive result is due to the fact that all nodes can be identified by their color, and the readout function can aggregate all the structural information in a complex and non-trivial way. This means that any function over graphs could be approximated by the present model, even if a function (e.g. graph diameter) would require normally deeper models than a single iteration. Such a claim falls into the trap that we need to take into account the whole permutation set of colorings and choosing the best out of all colorings, yielding to an overrview of the whole graph structure in a single layer. As for MLPs, we should expect poor generalization capabilities for CLIP with only one local iteration, and deeper networks may allow for more complex representations and better generalization. This point is addressed in the experiments of Section 3.7. Moreover, ∞-CLIP may be slow in practice due to a large number of colorings, and reducing k will speed up the computation. Fortunately, while k-CLIP

provides a random representation, a similar universality theorem still holds even for $k = 1$.

Theorem 6. *The 1-CLIP algorithm with one local iteration ($T = 1$) is a random representation whose expectation is a universal representation of the space $\mathbf{Graph}_m$ of graphs with node attributes.*

The proof of Theorem 6 relies on using ∞-CLIP on the augmented node attribute vectors $v'_i = (v_i, c_i)$. As all node attributes are, by design, different, the maximum overall colorings in Equation (3.6) disappears and, for any possible coloring, 1-CLIP returns an ε-approximation of the target function.

Proof of Theorem 6. Consider a continuous function $\psi : \mathbf{Graph}_m \to \mathbb{R}^d$ and a compact $K' \subset \mathbf{Graph}_m$. Let extend K' with $K = K' \times [0,1]^{n_{\max}}$ and we define $\phi : \mathbf{Graph}_{m+n_{\max}} \to \mathbb{R}^d$ with $\phi((v,c),A) = \psi(v,A)$ for all $c \in \mathcal{C}(v,A)$. Since ∞-CLIP is universal there exists $f \in \infty$-CLIP such that, for all $((v,c),A) \in K$,

$$\|\phi((v,c),A) - f((v,c),A)\| \leqslant \varepsilon\,, \tag{3.10}$$

hence

$$\|\psi(v,A) - f((v,c),A)\| \leqslant \varepsilon\,. \tag{3.11}$$

Moreover, observe that for any coloring $c \in \mathcal{C}(v,A)$, ∞-CLIP and 1-CLIP applied to $((v,c),A)$ returns the same result, as all node attributes are dissimilar (by definition of the colorings) and $\mathcal{C}((v,c),A) = \varnothing$. Finally, 1-CLIP applied to (v,A) is equivalent to applying 1-CLIP to $((v,C),A)$ where C is a random coloring in $\mathcal{C}(v,A)$, and Equation (3.11) thus implies that any random sample of 1-CLIP is within an ε error of the target function ψ. As a result, its expectation is also within an ε error of the target function ψ, which proves the universality of the expectation of 1-CLIP. □

Remark 1. *Note that the variance of the representation may be reduced by averaging over multiple samples. Moreover, the proof of Theorem 6 shows that the variance can be reduced to an arbitrary precision given enough training epochs, although this may lead to very large training times in practice. This is aligned with the intuition that using a larger number k of colorings should approach the case of $\infty-CLIP$*

Remark 2. *In the scope of the proposed model, we considered discrete values of colors as identifiers of node labels, based on their node attributes. A further extension would be to consider continuous values of the colors. Such an assumption requires though vigilance over the ordering of the nodes, because a continuous relaxation would mean that there is a distance over the different node labels. Such an assumption is avoided in the discrete disambiguators, as we can use one-hot encoding for the identifiers.*

Remark 3. *In the case of k-CLIP, the random choice of colorings makes the model lose its permutation invariance property, as we do not take into account the whole permutation set. That is quite natural, since imposing identifiers into the node labels requires considering all the permutations of the identifiers. However, as we will see in the experiments, such a relaxation can show a solid behavior in graph classification tasks.*

3.6.2 Computational complexity

As the local iterative steps are performed T times on each node, and the complexity of the aggregation depends on the number of neighbors of the considered node, the complexity is proportional to the number of edges of the graph E and the number of steps T. Moreover, CLIP performs this iterative aggregation for each coloring, and its complexity is also proportional to the number of chosen colorings $k = |\mathcal{C}_k|$. Hence the complexity of the algorithm is in $O(kET)$.

Note that the number of all possible colorings for a given graph depends exponentially in the size of the groups $V_1, ..., V_K$,

$$|\mathcal{C}(v, A)| = \prod_{k=1}^{K} |V_k|!\,, \tag{3.12}$$

and thus ∞-CLIP is practical only when most node attributes are dissimilar. This worst-case exponential dependency in the number of nodes can hardly be avoided for universal representations. Indeed, a universal graph representation should also be able to solve the graph isomorphism problem. Despite the existence of polynomial-time algorithms for a broad class of graphs [145, 25], graph isomorphism is still quasi-polynomial in general [11]. As a result, creating a universal graph representation with polynomial complexity for all possible graphs and functions to approximate is highly unlikely, as it would also induce a graph isomorphism test of polynomial complexity and thus solve a very hard and long-standing open problem of theoretical computer science.

3.7 Experiments

In this section we empirically show the practical efficiency of CLIP and its relaxation. We run two sets of experiments to compare CLIP w.r.t. state-of-the-art methods in supervised learning settings: i) on five real-world graph classification datasets and ii) on four synthetic datasets to distinguish structural graph properties and isomorphism. Both experiments follow the same experimental protocol as described in [238]: 10-fold cross-validation with grid search hyper-parameter optimization.

3.7.1 Classical benchmark datasets

We performed experiments on five benchmark datasets extracted from standard social networks (IMDBb and IMDBm) and bioinformatics databases (MUTAG, PROTEINS, and PTC). All dataset characteristics (e.g. size, classes) are available in Appendix A.1. Following standard practices for graph classification on these datasets, we use one-hot encodings of node degrees as node attributes for IMDBb and IMDBm [238], and perform single-label multi-class classification on all datasets. We compared CLIP with six state-of-the-art baseline algorithms: 1) **WL:** Weisfeiler-Lehman subtree kernel [205], 2) **AWL:** Anonymous Walk Embeddings [105], 3) **DCNN:** Diffusion-convolutional neural networks [7], 4) **PS:** PATCHY-SAN [165], 5) **DGCNN:** Deep Graph CNN [249] and 6) **GIN:** Graph Isomorphism Network [238]. WL and AWL are representative of unsupervised methods coupled with an SVM classifier, while DCNN, PS, DGCNN, and GIN are four deep learning architectures. As the same experimental protocol as that of [238] was used, we present their reported results on Table 3.1.

Experimentation protocol We optimized the CLIP hyperparameters by grid search according to 10-fold cross-validated accuracy means. We used 2-layer MLPs, an initial learning rate of 0.001 and decreased the learning rate by 0.5 every 50 epochs for all possible settings. For all datasets the hyperparameters we tested were the number of hidden units within $\{32, 64\}$, the number of colorings $c \in \{1, 2, 4, 8\}$, the number of MPNN layers within $\{1, 3, 5\}$, the batch size within $\{32, 64\}$, and the number of epochs, that means, we select a single epoch with the best cross-validation accuracy averaged over the 10 folds. Note that standard deviations are fairly high for all models due to the small size of these classic datasets.

TABLE 3.1: Classification accuracies of the compared methods on benchmark datasets. The best performer w.r.t. the mean is highlighted with an asterisk. We perform an unpaired t-test with asymptotic significance of 0.1 w.r.t. the best performer and highlight with boldface the ones for which the difference is not statistically significant. 0-CLIP is the CLIP architecture without any colorings.

Dataset	PTC	IMDBb	IMDBm	PROTEINS	MUTAG
WL	59.9±4.3	**73.8±3.9**	**50.9±3.8**	**75.0±3.1**	90.4±5.7
DCNN	56.6	49.1	33.5	61.3	67.0
PS	60.0±4.8	71.0±2.2	45.2±2.8	**75.9±2.8**	**92.6±4.2**
DGCNN	58.6	70.0	47.8	**75.5**	85.8
AWL	-	**74.5±5.9**	**51.5±3.6**	-	87.9±9.8
GIN	**64.6±7.0**	**75.1±5.1**	**52.3±2.8**	**76.2±2.8**	89.4±5.6
0-CLIP	**65.9±4.0**	**75.4±2.0**	**52.5±2.6***	**77.0±3.2**	90.0±5.1
CLIP	**67.9±7.1***	**76.0±2.7***	**52.5±3.0***	**77.1±4.4***	**93.9±4.0***

TABLE 3.2: Ablation study: classification accuracies of k-CLIP on benchmark datasets w.r.t k.

Dataset	PTC	IMDBb	IMDBm	PROTEINS	MUTAG
0-CLIP	65.9±4.0	75.4±2.0	52.5±2.6	77.0±3.2	90.0±5.1
1-CLIP	65.3±12.8	75.2±3.9	52.2±4.0	75.1±4.5	91.1±7.0
4-CLIP	65.9±5.7	75.8±5.0	51.8±2.9	77.1±4.4	92.2±7.0
8-CLIP	67.9±7.1	75.7±3.8	52.5±3.0	76.8±4.8	93.9±4.1
16-CLIP	66.5±5.4	76.0±2.7	52.5±4.5	76.6±2.8	91.7±6.0

As Table 3.1 shows, CLIP can achieve state-of-the-art performance on the five benchmark datasets. Moreover, CLIP is consistent across all datasets, while all other competitors have at least one weak performance. This is a good indicator of the robustness of the method to multiple classification tasks and dataset types. Finally, the addition of colors does not improve the accuracy for these graph classification tasks, except on the MUTAG dataset. This may come from the small dataset sizes (leading to high variances) or an inherent difficulty of these classification tasks and contrasts with the clear improvements of the method for property testing (see Section 3.7.2).

CLIP performances w.r.t. the number of colorings k

Table 3.2 summarizes the performances of CLIP while increasing the number of colorings k. Overall we can see a small increase in performances and a reduction of the variances when k is increasing. Nevertheless, we should not jump to any conclusions since none of the models are statistically significantly better than the others.

We note that on the IMDBb and PROTEINS datasets, the difference between using or not a coloring scheme does not have a big impact on the performances. However, adding colors increases the performance of the algorithm on three out of five real-world datasets. The property testing section (Section 3.7.2) shows empirically that the color scheme improves the expressiveness of CLIP.

Empirical result In three out of five datasets, none of the recent state-of-the-art algorithms have statistically significantly better results than older methods (e.g. WL). We argue that, considering the high variances of all classification algorithms on classical graph datasets, graph property testing may be better suited to measure the expressiveness of graph representation learning algorithms in practice.

3.7.2 Graph property testing

We now investigate the ability of CLIP to identify structural graph properties, a task which was previously used to evaluate the expressivity of graph kernels and on which the Weisfeiler-Lehman subtree kernel has been shown to fail for bounded-degree graphs [123]. The performance of our algorithm is evaluated for the binary classification of four different structural properties: 1) connectivity, 2) bipartiteness, 3) triangle-freeness, 4) circular skip links [164] (the precise definitions of these properties are given below) against three competitors: a) GIN, a powerful and very efficient MPNN variant [238], b) Ring-GNN, a permutation invariant network that uses the ring of matrix addition and multiplication [42], c) RP-GIN, the Graph Isomorphism Network combined with Relational Pooling, as described by [164], which is able to distinguish certain cases of non-isomorphic regular graphs.

Our goal is to show that CLIP is able to distinguish basic graph properties, where classical MPNN cannot. We consider a binary classification task and we construct *balanced* synthetic datasets for each of the examined graph properties. The 20-node graphs are generated using Erdös-Rényi model [67] (and its bipartite version for the bipartiteness) with different probabilities p for edge creation. All nodes share the same (scalar) attribute. We thus have uninformative feature vectors. In particular, we generate datasets for different classical tasks [123]: 1) connectivity, 2) bipartiteness, 3) triangle-freeness, and 4) circular skip links [164]. Next, we present the generating protocol of the synthetic datasets and the experimentation setup that was followed.

Synthetic datasets:
In every case of the synthetic datasets, we follow the same pattern: we generate a set of random graphs using Erdös-Rényi model, which contains a specific graph property and belongs to the same class, and by proper edge addition, we remove this property, thus creating the second class of graphs. In this way, we assure that we do not change different structural characteristics other than the examined graph property.

1. **Connectivity dataset:** this dataset consists of 1000 (20-node) graphs with 500 positive samples and 500 negative ones. The positive samples correspond to disconnected graphs with two 10-node connected components selected among randomly generated graphs with an Erdös-Rényi model probability of $p = 0.5$. We constructed negative samples by adding to positive samples a random edge between the two connected components.

2. **Bipartiteness dataset:** this dataset consists of 1000 (20-node) graphs with 500 positive samples and 500 negative ones. The positive samples correspond to bipartite graphs generated with an Erdös-Rényi (bipartite) model probability of $p = 0.5$. For the negative samples (non-bipartite graphs) we chose the positive samples, and for each of them we added an edge between randomly

selected nodes from the same partition in order to form odd cycles [1].

3. **Triangle-freeness dataset:** this dataset consists of 1000 (20-node) graphs with 500 positive samples and 500 negative ones. The positive samples correspond to triangle-free graphs selected among randomly generated graphs with an Erdös-Rényi model probability of $p = 0.1$. We constructed negative samples by randomly adding new edges to positive samples until it creates at least one triangle.

4. **Circular skip links:** this dataset consists of 150 graphs of 41 nodes as described in [164, 42]. The Circular Skip Links graphs are undirected regular graphs with node degree 4. We denote a Circular skip link graph by $G_{n,k}$ an undirected graph of n nodes, where $(i, j) \in E$ holds if and only if $|i - j| \equiv 1$ or $k(\mod n)$ This is a 10-class multiclass classification task whose objective is to classify each graph according to its isomorphism class.

Experimentation protocol: We evaluate the different configurations of CLIP and its competitors GIN and RP-GIN based on their hyper-parameters. For the architecture implementation of the GIN, we followed the best performing architecture, presented in [238]. In particular, we used the summation as the aggregation operator, MLPs as the combination level for the node embedding generation and the sum operator for the readout function along with its refined version of concatenated graph representations across all iterations/layers of GIN, as described in [238].
In all the tested configurations for CLIP and its competitors (GIN, RP-GIN) we fixed the number of layers of the MLPs and the learning rate: we chose 2-layer MLPs and we used the Adam optimizer with initial learning rate of 0.001 along with a scheduler decaying the learning rate by 0.5 every 50 epochs. Concerning the other hyper-parameters, we optimized: the number of hidden units within $\{16, 32, 64\}$ (except for the CSL task where we only use 16 hidden units to be fair w.r.t. RP-GIN and Ring-GNN benchmarks), the number of MPNN layers within $\{1, 2, 3, 5\}$, the batch size within $\{32, 64\}$, and ran the model over 400 epochs. Regarding the RP-GIN architecture [164] we optimized the one-hot encoding dimension of the first update within $\{5, 10, 15, 20, 25, 30\}$ and the number of inference permutations within $\{1, 5, 16\}$. Regarding the CLIP algorithm, we optimized the number of colorings $c \in \{1, 2, 4, 8, 16\}$. We then performed 10-fold cross-validation with early stopping for the hyper-parameter optimization and we reported the best 10-fold cross-validated mean accuracy with its associated standard deviation.

Table 3.3 shows that CLIP is able to capture the structural information of connectivity, bipartiteness, triangle-freeness and circular skip links, while MPNN variants fail to identify these graph properties. Furthermore, we observe that CLIP outperforms RP-GIN, that was shown to

[1] Having an odd cycle in a graph makes the graph non-bipartite.

TABLE 3.3: Classification accuracies of the synthetic datasets. k-RP-GIN refers to a relational pooling averaged over k random permutations. We report Ring-GNN results from [42].

Property	Connectivity	Bipartiteness	Triangle-freeness	Circular skip links		
	mean ± std	mean ± std	mean ± std	mean ± std	max	min
GIN	55.2 ± 4.4	53.1 ±4.7	50.7±6.1	10.0 ± 0.0	10.0	10.0
Ring-GNN	-	-	-	(?) ± 15.7	80.0	10.0
1-RP-GIN	66.1±5.2	66.0±5.1	63.0±3.6	20.0 ± 7.0	28.6	10.0
16-RP-GIN	83.3±7.9	64.9±4.1	65.7±3.3	37.6 ± 12.9	53.3	10.0
0-CLIP	56.5 ± 4.0	55.4 ± 5.7	59.6 ± 3.8	10.0 ± 0.0	10.0	10.0
1-CLIP	73.3 ± 2.2	63.3 ±1.9	63.5 ±7.3	61.9 ±11.9	80.7	36.7
16-CLIP	**99.7 ± 0.5**	**99.2 ± 0.9**	**94.2±3.4**	**90.8 ± 6.8**	98.7	76.0

provide very expressive representations for regular graphs [164], even with a high number of permutations (the equivalent of colors in their method is set to $k = 16$).

Moreover, both for k-RP-GIN and k-CLIP, the increase of permutations and colorings respectively lead to higher accuracies. In particular, CLIP can capture almost perfectly the different graph properties with as little as $k = 16$ colorings. Finally, note that using one-hot encodings of (randomly chosen) node labels (which is equivalent to 1-CLIP as graphs have no node attributes in this experiment) provides a notable improvement over GIN, although far below the near-perfect accuracy of 16-CLIP.

3.7.3 3-regular graphs

Aligned with our theoretical observations in Section 3.4.2, we consider now the synthetic task of predicting structural properties in k-regular graphs. In particular, for this experiment we assume $k = 3$ and the task of detecting triangles, similar to the triangle-freeness property in Section 3.7.2. According to our theoretical analysis in Section 3.4.2, standard Message Passing Neural Networks are not capable of distinguishing nodes in 3-regular graphs, and thus should necessarily have a classification accuracy of 50% for any well-balanced binary classification task on these graph datasets.

For the visualization of such a claim, we constructed a synthetic dataset composed of 200 random 3-regular graphs, half of which contain at least one triangle (positive class). On this dataset, we performed a 10-fold cross-validation with c-CLIP and we show the results in Figure 3.6. For this toy example we fixed all hyperparameters but the number of colorings of the relaxation $c \in \{1, 16, 32, 128\}$ and did not change the learning rate during the learning phase. While MPNNs fail at learning this task (see Lemma 3), Figure 3.6 shows that CLIP successfully identifies triangles, up to a perfect 100% test accuracy with 32 colorings in

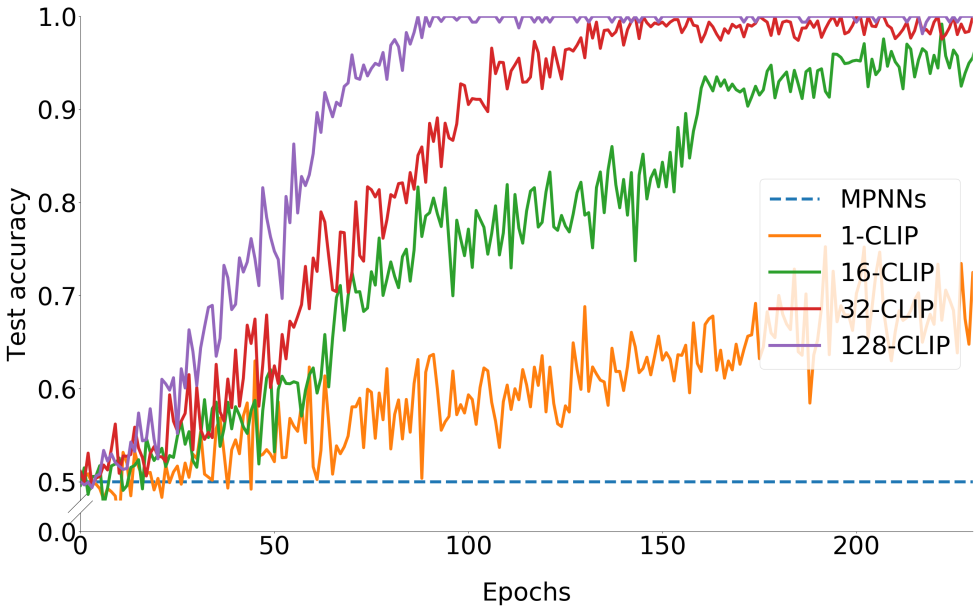


FIGURE 3.6: Classification accuracy for triangle detection in 3-regular graphs w.r.t. number of epochs for CLIP with increasing number of colorings. All MPNNs obtain a 50% accuracy at this task.

less than 150 epochs. This result shows that CLIP is capable of capturing structural information of 3-regular graphs that was not captured by other state-of-the-art algorithms.

Note that, since the produced dataset is balanced, the considered baselines all achieve an accuracy of 50% (see Lemma. This means that all of the baselines were not able to capture the triangle existence, an empirical result that is aligned to 3).

Intuitively one might assume that 1-CLIP should be able to achieve 100% accuracy. Indeed one can rethink the initial dataset as a much bigger one considering as new elements all the possible colorings of the initial one. In this case, the model should be able to detect perfectly the triangles but will need much more epochs to go trough this augmented dataset. That might be the reason why the learning slope w.r.t. the number of epochs is lower. Moreover, the model is able to detect triangles even if the number of colorings per graph ($c = 32$) is extremely small compared to the total number of possible colorings ($20! \approx 2.4 \cdot 10^{18}$). This empirical result illustrates the effectiveness of the proposed relaxation $k-$CLIP in practical scenarios.

3.8 Conclusion

In this chapter, we showed that a simple coloring scheme can improve the expressive power of MPNNs. Using such a coloring scheme, we extended MPNNs to create CLIP, the first universal graph representation.

Universality was proven using the concept of separable neural networks, and our experiments showed that CLIP is state-of-the-art on both graph classification datasets and property testing tasks.

The coloring scheme is especially well suited to hard classification tasks that require complex structural information to learn. The framework is general and simple enough to extend to other data structures such as directed, weighted or labeled graphs. Future work includes more detailed and quantitative approximation results depending on the parameters of the architecture, such as the number of colors k, or a number of hops of the iterative neighborhood aggregation.

Chapter 4

Learning Soft Permutations for Graph Representations

4.1 Introduction

In the previous chapter, we dealt with the problem of graph classification and how we can build graph representations that can discriminate non-isomorphic graphs using separability criteria. In real-world scenarios, such as chemoinformatics, bioinformatics, and social networks, this problem reflects in cases where we have a series of graphs on which we would like to assign labels. For instance, in chemistry, molecules can be modeled as graphs where vertices and edges represent atoms and chemical bonds, respectively. A social network is usually represented as a graph where users are mapped to vertices and edges capture friendship relationships. In such cases, we are not interested only in finding representations that should be different for dissimilar graphs, but, also, should preserve some form of *isometry*, i.e., they should reveal information about "how much different" the graphs are.

Due to the recent growth in the amount of produced graph-structured data, the field of graph representation learning has attracted much attention in the past years with applications ranging from drug design [112] to learning the dynamics of physical systems [177], where the aforementioned isometric property is often crucial. Among the different algorithms proposed in the field of graph representation learning, message passing GNNs (or MPNNs as they have been defined in Chapter 3) have recently shown significant success in solving real-world learning problems on graphs. However, for the problem of graph classification, MPNNs take into account each graph independently, omitting information that can be obtained from treating a corpus of graphs as a whole. With the exception of a few architectures [165, 149, 150, 169, 217], all the remaining models belong to the family of standard MPNNs. For a certain number of iterations, these models update the representation of each vertex by aggregating information from its neighborhood. In fact, there is a close connection between this update procedure and the

Weisfeiler-Leman test of isomorphism [229] and, based on these observations, the expressive power of MPNNs has been studied, similarly to Chapter 3 [238, 161, 159].

The need for representations that can capture information of some form of distance between the graphs requires the development of novel architectures, which will allow researchers in the field to break away from the message passing schemes. This chapter takes a step in this direction. Specifically, we first highlight some general limitations of approaches that project graphs into vector spaces. We consider a well-established distance measure for graphs, and we show that in the general case, there is no inner product space such that the distance induced by the norm of the space matches exactly the considered distance function. We also show that, for specific classes of graphs, such representations can be generated by imposing an ordering on the vertices of each graph. The above result motivates the design of a novel neural network model, so-called π-GNN, which learns a "soft" permutation (i.e. doubly stochastic) matrix for each graph and thus projects all graphs into a common vector space. The learned matrices impose a "soft" ordering on the vertices of the graphs, and the adjacency matrices are mapped into vectors based on this ordering. These vectors can then be fed into fully-connected or convolutional layers to deal with supervised learning tasks. To make the model more efficient in terms of running time and memory, we further relax the doubly stochastic matrices to row stochastic matrices. We compare the performance of the proposed model to well-established neural architectures on several benchmark datasets for graph classification and graph regression. Results show that the proposed model matches or outperforms competing methods. Our main contributions are summarized as follows:

- We demonstrate that graph embedding algorithms such as GNNs, cannot isometrically embed a metric space whose metric corresponds to a widely accepted distance function for graphs into a vector space. This is, however, possible for specific classes of graphs.
- We propose a novel neural network model, π-GNN, which learns a "soft" permutation matrix for each graph and uses this matrix to project the graph into a vector space. The set of permutation matrices induces an alignment of the input graphs, and thus graphs are mapped into a common space.
- We evaluate the proposed model on several graph classification and graph regression datasets where it achieves performance comparable and in some cases better than that of state-of-the-art GNNs.

The rest of this chapter is organized as follows. Section 4.2 provides an overview of the related work. Section 4.3 highlights the limitations of graph embedding approaches. Section 4.4 provides a detailed description of the proposed π-GNN model. Section 4.5 evaluates the proposed

model in graph classification and graph regression tasks. Finally, Section 4.6 concludes.

4.2 Standard MPNNs and Works Beyond Message Passing

The architectures that helped to make the field of graph representation learning active include modern variants of the old models [139] as well as approaches that generalized the convolution operator to graphs based on well-established graph signal processing concepts [34, 60, 118]. All of the standard MPNN architectures take into account a sample of graphs and perform a process of two phases, as we have seen in Chapter 3. First, a message-passing phase where vertices iteratively update their feature vectors by aggregating the feature vectors of their neighbors. Second, a readout phase where a permutation invariant function is employed to produce a feature vector for the entire graph.

Research has focused mainly on the message passing phase [249, 54], but also on the readout phase though in a smaller extent [238, 215, 244]. The family of MPNNs is closely related to the Weisfeiler-Lehman (WL) isomorphism test [229]. Specifically, these models generalize the relabeling procedure of the WL test to the case where vertices are annotated with continuous features. Standard MPNNs have been shown to be at most as powerful as the WL test in distinguishing non-isomorphic graphs [238, 161]. Some works aggregate other types of structures instead of neighbors such as small subgraphs [129] or paths [38]. Considerable efforts have also been devoted to building deeper MPNNs [133, 73] and more powerful models [161, 159]. There have also been made some efforts to develop GNN models that do not follow the design paradigm of MPNNs. For instance, Niepert, Ahmed, and Kutzkov [165] proposed a model that extracts neighborhood subgraphs for a subset of vertices, imposes an ordering on each subgraph's vertices, and applies a convolutional neural network on the emerging matrices [165]. Maron et al. [150] proposed k-order graph networks, a general form of neural networks that consist of permutation equivariant and invariant tensor operations. Instances of these models correspond to a composition of functions, typically a number of equivariant linear layers and an invariant linear layer which is followed by a multi-layer perceptron [149, 150]. Nikolentzos and Vazirgiannis proposed a model that generates graph representations by comparing the input graphs against a number of latent graphs using random walk kernels [169].

The works closest to the direction of this chapter are the ones reported in [13] and in [14]. In these works, the authors map input graphs into fixed-sized aligned grid structures. To achieve that, they align the vertices of each graph to a set of prototype representations. To obtain these prototype representations, the authors apply the k-means

algorithm to the vertices of all graphs, and each emerging centroid is considered as a prototype. They then compute for each input graph a binary correspondence matrix by assigning each vertex of the graph to its closest centroid. Based on this matrix, the authors produce a new graph (i.e. a new adjacency matrix and a new matrix of features), while they also impose an ordering on the vertices of the new graph using some centrality measure. Finally, these matrices are fed into an MPNN model, which is followed by a convolutional neural network to generate the output. Unfortunately, these models are not end-to-end trainable. Mapping input graphs into aligned grid structures involves non-differentiable operations and is thus applied as a preprocessing step. Furthermore, for large datasets, partitioning the vertices of all graphs into clusters using the k-means algorithm can be computationally expensive. On the other hand, the proposed model is better motivated and is end-to-end trainable since it employs a differentiable layer to compute a matrix of "soft" correspondences. Our work is also related to neural network models that learn to compare graphs to each other [15, 16, 227, 196].

4.3 Can We Generate Expressive Graph Representations?

As mentioned above, in several application areas, samples do not come in the form of fixed-sized vectors but in the form of graphs. For instance, in chemistry, molecules are typically represented as graphs, and in social network analysis, collaboration patterns are also mapped into graph structures. There exist several approaches that map graphs into vectors; however, in most cases, the emerging vectors could be of low quality, could be difficult to obtain, and may fail to capture the full complexity of the underlying graph objects. However, vector representations are very convenient since most well-established learning algorithms can operate on this type of data. These are developed in inner product spaces or normed spaces, in which the inner product or norm defines the corresponding metric. Ideally, we would like to project a collection of graphs into a Euclidean space such that some well-established notion of distance between graphs is captured as accurately as possible from the Euclidean norm of that space.

Before introducing the distance function, we present some key notation for graphs. Let $[n] = \{1, \dots, n\} \subset \mathbb{N}$ for $n \geqslant 1$. Let $G = (V, E)$ be an undirected graph, where V is the vertex set, and E is the edge set. We will denote by n the number of vertices and by m the number of edges. The adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is a symmetric matrix used to encode edge information in a graph. We say that two graphs G_1 and G_2 are isomorphic to each other, i.e. $G_1 \simeq G_2$, if there exists an

adjacency preserving bijection $\pi\colon V_1 \to V_2$, i.e. (u, v) is in E_1 if and only if $(\pi(u), \pi(v))$ is in E_2, call π an isomorphism from G_1 to G_2.

The distance function that we consider in this chapter is the *Frobenius distance* [85], one of the most well-studied distance measures for graphs. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs on n vertices with respective $n \times n$ adjacency matrices $\mathbf{A}_1$ and $\mathbf{A}_2$. The *Frobenius distance* is a function $d : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ where $\mathcal{G}$ is the space of graphs which quantifies the distance of two graphs and can be expressed as the following minimization problem:

$$d(G_1, G_2) = \min_{\mathbf{P} \in \Pi} ||\mathbf{A}_1 - \mathbf{P}\,\mathbf{A}_2\,\mathbf{P}^\top||_F\,, \tag{4.1}$$

where Π denotes the set of $n \times n$ permutation matrices, and $||\cdot||_F$ is the Frobenius matrix norm. For clarity of presentation, we assume n to be fixed (i.e. both graphs consist of n vertices). In order to apply the function to graphs of different cardinalities, one can append zero rows and columns to the adjacency matrix of the smaller graph to make its number of rows and columns equal to n. Therefore, the problem of graph comparison can be reformulated as the problem of minimizing the above function over the set of permutation matrices. A permutation matrix $\mathbf{P}$ gives rise to a bijection $\pi : V_1 \to V_2$. The function defined above seeks for a bijection such that the number of common edges $|\{(u, v) \in E_1 : \big(\pi(u), \pi(v)\big) \in E_2\}|$ is maximized. The above definition is symmetric in G_1 and G_2. The two graphs are isomorphic to each other if and only if there exists a permutation matrix $\mathbf{P}$ for which the above function is equal to 0. Unfortunately, the above distance function is not computable in polynomial time [85, 6]. Furthermore, computing the distance remains hard even if both input graphs are trees [85]. Its high computational cost prevents the above distance function from being used in practical scenarios. An interesting question that we answer next is how well graph embedding algorithms, i.e. approaches that map graphs into vectors, can embed the space of graphs equipped with the above function into a vector space. It turns out that for an arbitrary collection of graphs, isometrically embedding the aforementioned metric space into a vector space is not feasible.

Theorem 7. *Let $(\mathcal{G}, d)$ be a metric space where $\mathcal{G}$ is the space of graphs and d is the distance defined in Equation 4.1. The above metric space cannot be embedded in any Euclidean space.*

The proof of Theorem 7 can be found in Appendix B.2.

Unfortunately, most machine learning algorithms that operate on graphs map graphs explicitly or implicitly into vectors (e.g. the readout function of MPNNs). The above result suggests that these learning algorithms cannot generate maximally expressive graph representations, i.e. vector representations such that the Euclidean distances between the different

graphs are arbitrarily close (or equal) to those produced by the distance function defined in Equation 4.1.

The above result holds for the general case (i.e. arbitrary collections of graphs). If we restrict our set to contain instances of only specific classes of graphs, then, it might be possible to map graphs into vectors such that the pairwise Euclidean distances are equal to those that emerge from Equation 4.1. For instance, if our set consists only of paths or of cycles, we can indeed project them into a vector space such that the Euclidean norm induces the distance defined in Equation 4.1. To obtain such representations, the trick is to impose an ordering on the vertices of each graph such that those orderings are consistent across graphs. For example, in the case of the path graphs, we can impose the following ordering: the first vertex is one of the two terminal vertices. Let v denote that vertex. Then, the i-th vertex is uniquely defined and corresponds to the vertex u which satisifies $\text{sp}(v, u) = i - 1$ where $\text{sp}(\cdot, \cdot)$ denotes the shortest path distance between two vertices. Let n denote the number of vertices of the longest path in the input set of graphs. The adjacency matrix of each graph G_i is then expanded by zero-padding such that $\mathbf{A}_i \in \mathbb{R}^{n \times n}$. For any pair of graphs G_1, G_2, we then have:

$$\begin{aligned} d(G_1, G_2) = \min_{\mathbf{P} \in \Pi} ||\mathbf{A}_1 - \mathbf{P}\,\mathbf{A}_2\,\mathbf{P}^\top||_F &= ||\mathbf{A}_1 - \mathbf{I}_n\,\mathbf{A}_2\,\mathbf{I}_n^\top||_F \\ &= ||\mathbf{A}_1 - \mathbf{A}_2||_F\,, \end{aligned}$$

where $\mathbf{I}_n$ denotes the $n \times n$ identity matrix. Clearly, these graphs can be embedded into vectors in some Euclidean space as follows: $\mathbf{v}_i^{\text{adj}} = \text{vec}(\mathbf{A}_i)$ where vec denotes the vectorization operator which transforms a matrix into a vector by stacking the columns of the matrix one after another, and $\mathbf{v}_i^{\text{adj}} \in \mathbb{R}^{n^2}$. Then, the distance between two graphs is computed as:

$$d(G_1, G_2) = ||\mathbf{A}_1 - \mathbf{A}_2||_F = ||\mathbf{v}_1^{\text{adj}} - \mathbf{v}_2^{\text{adj}}||_2\,,$$

where $||\cdot||_2$ is the standard ℓ_2 norm of the input vector. The emerging vectors can thus be thought of as the representations of the path graphs in the Euclidean space $\mathbb{R}^{n^2}$. However, still, as shown next, it turns out that we cannot map these graphs into a vector space whose dimension is smaller than the cardinality of the set itself.

Proposition 3. *Let $\{P_2, P_3, \ldots, P_{n+1}\}$ be a collection of n path graphs, where P_i denotes the path graph consisting of i vertices. Let also $\mathbf{X} \in \mathbb{R}^{n \times (n+1)^2}$ be a matrix that contains the vector representations of the n graphs obtained as discussed above (i.e. the i-th row of matrix $\mathbf{X}$ contains the representation of the i-th graph). Then, the rank of the matrix $\mathbf{X}$ is equal to n.*

Proof. Without loss of generality, assume that the path graphs are sorted based on their order (i.e. number of vertices). Then, the i-th row of

matrix $\mathbf{X}$ contains the vector representation of graph P_{i+1}. Notice that given two consecutive path graphs P_i, P_j with $j - i = 1$, the first $(i+1)n^2$ elements of their representations are identical, while the last $(n-i-1)n^2$ and $(n-j-1)n^2 = (n-i-2)n^2$ of the two representations are equal to zero. Therefore, if we subtract the first vector from the second vector, we end up with a vector that has some nonzero elements in the position between $(n-i-1)n^2$ and $(n-i-2)n^2$. Then, we can start from the last row of $\mathbf{X}$ and subtract from each row the immediately preceding row. We obtain a set of $n-1$ orthogonal vectors, and therefore, the rank of the matrix $\mathbf{X}$ is equal to $n-1$. □

This is another negative result since it implies that even in cases where we can impose an ordering on the vertices of the graphs, which can be used to align the graphs, we cannot project them into a Euclidean space of fixed dimension and retain all pairwise distances.

4.4 π-Graph Neural Networks

Graph-level machine learning problems are usually associated with finite sets of graphs $\{G_1, \ldots, G_N\} \subset \mathcal{G}$. Given such a set of graphs, is it possible to align them such that we can then project them into a Euclidean space? The objective, in this case, would be to find a permutation matrix $\mathbf{P}_i^*$ for each graph G_i of the dataset where $i \in [N]$ such that the overall distance between graphs is minimized. This gives rise to the following problem:

$$\mathbf{P}_1^*, \ldots, \mathbf{P}_N^* = \underset{\mathbf{P}_1, \ldots, \mathbf{P}_N \in \Pi}{\arg\min} \sum_{i=1}^{N} \sum_{j=1}^{N} ||\mathbf{P}_i \, \mathbf{A}_i \, \mathbf{P}_i^\top - \mathbf{P}_j \, \mathbf{A}_j \, \mathbf{P}_j^\top||_F \,. \tag{4.2}$$

The above optimization problem imposes an ordering on the vertices of all graphs of the input set and thus each graph G_i can then be embedded into a common Euclidean space as $\mathbf{v}_i^{\text{adj}} = \text{vec}(\mathbf{P}_i^* \, \mathbf{A}_i \, \mathbf{P}_i^{*\top})$. Unfortunately, solving the above problem is hard. Note that the distance function defined in Equation 4.1 is a special case of the above problem when the number of samples is $N = 2$. Furthermore, it is clear that for each pair of graphs G_i, G_j with $i, j \in [N]$, the distance function of Equation 4.1 is a lower bound to the distances that emerge from the above permutation matrices. Thus, for any two graphs G_i, G_j, we have:

$$\begin{aligned} d(G_i, G_j) &= \min_{\mathbf{P} \in \Pi} ||\mathbf{A}_i - \mathbf{P} \, \mathbf{A}_j \, \mathbf{P}^\top||_F \\ &\leqslant ||\mathbf{P}_i^* \, \mathbf{A}_i \, \mathbf{P}_i^{*\top} - \mathbf{P}_j^* \, \mathbf{A}_j \, \mathbf{P}_j^{*\top}||_F \,. \end{aligned} \tag{4.3}$$

Given two graphs G_i, G_j, Equation 4.3 implies that the permutation matrices $\mathbf{P}_i^*, \mathbf{P}_j^*$ embed the adjacency matrices $\mathbf{A}_i, \mathbf{A}_j$ in a space at least as separable as the one provided from the Frobenius distances. In

Appendix B.3 we investigate an interesting connection between the objective in Equation 4.2 and a cost sharing game.

4.4.1 Learning soft permutations

Inspired by the above alignment problem, in this chapter, we propose a neural network model that performs an alignment of the input graphs and embeds them into a Euclidean space. Specifically, we propose a model that learns a unique permutation matrix for each graph G_i from the collection of input graphs $G_1, \ldots, G_N$. These permutation matrices enable the model to project graphs into a common vector space.

In fact, the problem of learning permutation matrices $\mathbf{P}_1, \ldots, \mathbf{P}_N \in \Pi$ has a combinatorial nature and is not feasible in practice. Therefore, in our formulation, we replace the space of permutations by the space of doubly stochastic matrices. Such approximate relaxations are common in graph matching algorithms [2, 107]. Let $\mathcal{D}$ denote the set of $n \times n$ doubly stochastic matrices, i.e. nonnegative matrices with row and column sums each equal to 1. The proposed model associates each graph G_i with a doubly stochastic matrix $\mathbf{D}_i \in \mathcal{D}$ and the vector representation of each graph is now given by $\mathbf{v}_i^{\text{adj}} = \text{vec}\big(\mathbf{D}_i \mathbf{A}_i \mathbf{D}_i^\top\big)$. Note that the price to be paid for the above relaxation is that the emerging graph representations are less expressive since two non-isomorphic graphs can be mapped to the same vector. In other words, there exist pairs of graphs G_i, G_j with $G_i \not\simeq G_j$ and doubly stochastic matrices $\mathbf{D}_i, \mathbf{D}_j \in \mathcal{D}$ such that $\mathbf{D}_i \mathbf{A}_i \mathbf{D}_i^\top = \mathbf{D}_j \mathbf{A}_j \mathbf{D}_j^\top$.

To compute these doubly stochastic matrices, we capitalize on ideas from optimal transport [176]. Specifically, we design a neural network that learns matrix $\mathbf{D}_i$ from two sets of feature vectors, one that contains some structural features of the vertices (and potentially their attributes) of a graph G_i and one trainable matrix that is randomly initialized. Let n denote the number of vertices of the largest graph in the input set of graphs. We first generate a matrix $\mathbf{Q}_i \in \mathbb{R}^{n \times d}$ for each graph G_i of the collection, which contains a number of local vertex features that are invariant to vertex renumbering (e.g. degree, number of triangles, etc.). Note that for a graph G_i consisting of n_i vertices, the last $n - n_i$ rows of matrix $\mathbf{Q}_i$ are initialized to the zero vector. Let also $\mathbf{W} \in \mathbb{R}^{n \times d}$ denote a matrix of trainable parameters. Note that we can think of matrix $\mathbf{W}$ as a matrix whose rows correspond to some latent vertices and which contain the features of those vertices. Then, the rows of matrix $\mathbf{Q}_i$ are compared against those of matrix $\mathbf{W}$ using some differentiable function f. In our experiments, we have defined f as the inner product between the two input vectors followed by the ReLU activation function. Thus we have that $\mathbf{S}_i = \text{ReLU}(\mathbf{Q}_i \mathbf{W}^\top) \in \mathbb{R}^{n \times n}$ where $\mathbf{S}_i$ is a matrix of scores or similarities between the vertices of a graph G_i and the model's trainable parameters.

Note that before computing matrix $\mathbf{S}_i$, we can first transform the vertex features using a fully-connected layer, i.e. $\tilde{\mathbf{Q}}_i = g(\mathbf{Q}_i \mathbf{H} + \mathbf{b})$ where $\mathbf{H} \in \mathbb{R}^{d \times \tilde{d}}$ and $\mathbf{b} \in \mathbb{R}^{\tilde{d}}$ is a weight matrix and bias vector, respectively, and g is a non-linear activation function. We can also use an MPNN architecture to produce new vertex representations, i.e. $\tilde{\mathbf{Q}}_i = \text{MPNN}(\mathbf{A}_i, \mathbf{Q}_i)$. In fact, the expressive power of the proposed model depends on how well those features capture the structural properties of vertices in the graph. Thus, highly expressive MPNNs could lead the proposed model into generating more expressive representations.

For a graph G_i, matrix $\mathbf{D}_i \in [0,1]^{n \times n}$ can then be obtained by solving the following problem:

$$\begin{aligned} &\max \sum_{j=1}^{n} \sum_{k=1}^{n} \mathbf{S}_i^{j,k} \mathbf{D}_i^{j,k} \\ &\text{s.t.} \\ &\mathbf{D}_i \, \mathbf{1}_n = \mathbf{1}_n \;\; \text{and} \;\; \mathbf{D}_i^\top \, \mathbf{1}_n = \mathbf{1}_n \,, \end{aligned} \tag{4.4}$$

where $\mathbf{1}_n$ is an n-element vector of ones, and $\mathbf{S}_i^{j,k}$ and $\mathbf{D}_i^{j,k}$ denote the element of the j-th row and k-th column of matrices $\mathbf{S}_i$ and $\mathbf{D}_i$. The emerging matrix $\mathbf{D}_i$ is a doubly stochastic matrix, while the above formulation is equivalent to solving a linear assignment problem. The solution of the above optimization problem corresponds to the optimal transport [176] between two discrete distributions with scores $\mathbf{S}_i$. Its entropy-regularized formulation naturally results in the desired soft assignment and can be efficiently solved on GPU with the Sinkhorn algorithm [49]. It is a differentiable version of the Hungarian algorithm [162], classically used for bipartite matching, that consists in iteratively normalizing $\exp(\mathbf{S}_i)$ along rows and columns, similar to row and column softmax.

By solving the above linear assignment problem, we obtain the doubly stochastic matrix $\mathbf{D}_i$ associated with graph G_i. Then, as described above we get $\mathbf{v}_i^{\text{adj}} = \text{vec}\big(\mathbf{D}_i \, \mathbf{A}_i \, \mathbf{D}_i^\top\big)$. This n^2-dimensional vector can be used as features for various machine learning tasks, e.g. graph regression or graph classification. For instance, for a graph classification problem with $|\mathcal{C}|$ classes, the output is computed as:

$$\mathbf{p}_i = \text{softmax}(\mathbf{W}^{(c)} \, \mathbf{v}_i^{\text{adj}} + \mathbf{b}^{(c)}) \,,$$

where $\mathbf{W}^{(c)} \in \mathbb{R}^{|\mathcal{C}| \times n^2}$ is a matrix of trainable parameters and $\mathbf{b}^{(c)} \in \mathbb{R}^{|\mathcal{C}|}$ is the bias term. We can even create a deeper architecture by adding more fully-connected layers. Since we have imposed some "soft" ordering on the vertices of each graph, we could also treat the matrix $\mathbf{D}_i \, \mathbf{A}_i \, \mathbf{D}_i^\top$ as an image and apply some standard convolution operation where filters of dimension $h \times n$ (with $h < n$) are applied to the representations of h vertices to produce new features. The filters are applied to each possible

sequence of vertices to produce feature maps of dimension $n - h + 1$. These feature maps can be fed to further convolution/pooling layers and finally to a fully-connected layer.

4.4.2 Vertex attributes

In the case of graphs that contain vertex attributes, the graphs' adjacency matrices do not incorporate the vertex information provided by the attributes. There is thus a need to take these attributes into account. Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ denote the matrix of node features where d is the feature dimensionality. The feature of a given node v_i corresponds to the i-th row of $\mathbf{X}$. To produce a representation of the graph that takes into account both the structure and the vertex attributes, we can first compute a second matrix of scores or similarities $\mathbf{S}_i^{att}$ between the attributes of the vertices of graph G_i and some trainable parameters $\mathbf{W}^{att} \in \mathbb{R}^{n \times d}$, as follows: $\mathbf{S}_i^{att} = \text{ReLU}(\mathbf{X}_i \, \mathbf{W}^{att\top}) \in \mathbb{R}^{n \times n}$. We can then solve a problem similar to that of Equation 4.4 and obtain matrix $\mathbf{D}_i^{att}$. Then, the "soft" permutation matrix can be obtained as follows:

$$\mathbf{D}_i = \alpha \mathbf{D}_i^{adj} + (1 - \alpha)\mathbf{D}_i^{att}\,, \tag{4.5}$$

where $\alpha \in [0, 1]$. Thus, each element of $\mathbf{D}_i$ is a convex combination of the elements of the two matrices $\mathbf{D}_i^{adj}$ and $\mathbf{D}_i^{att}$. We can then use the learned doubly stochastic matrices to also explicitly map the vertex attributes into a vector space. Therefore, for some graph G_i, we produce a second vector as follows:

$$\mathbf{v}_i^{\text{att}} = \text{vec}\big(\mathbf{D}_i \, \mathbf{X}_i\big)\,,$$

where $\mathbf{v}_i^{\text{att}} \in \mathbb{R}^{nd}$. If edge attributes are also available, the adjacency matrix $\mathbf{A}_i$ can be represented as a three-dimensional tensor (i.e. $\mathbf{A}_i \in \mathbb{R}^{n \times n \times d_e}$ where d_e is the dimension of edge attributes). We can then apply the "soft" permutation to this tensor and then map the emerging tensor into a vector.

4.4.3 Scaling to large graphs

A major limitation of the proposed model is that its complexity depends on the vertex cardinality of the largest graph contained in the input set. For instance, if there is a single very large graph (with cardinality n), and the number of vertices of the remaining graphs is much smaller than n, the model will learn $n \times n$ doubly stochastic matrices, and all the graphs will be mapped to vectors in $\mathbb{R}^{n^2}$ even though they could have been projected to a lower-dimensional space. This problem can be addressed by shrinking the adjacency matrix of the largest graph (e.g. by removing some vertices and their adjacent edges). However, this approach seems problematic since it results in the loss of information.

To deal with large graphs, we propose to reduce the number of rows of matrix $\mathbf{W}$ from n to some value $p < n$, i.e. $\mathbf{W} \in \mathbb{R}^{p \times d}$. The above will produce a rectangular (and not square) similarity matrix $\mathbf{S}_i \in \mathbb{R}^{n \times p}$ which will then lead to a rectangular doubly stochastic correspondence matrix that fulfills the following constraints (instead of the ones of Equation 4.4):

$$\mathbf{D}_i \, \mathbf{1}_p = \mathbf{1}_n \quad \text{and} \quad \mathbf{D}_i^\top \, \mathbf{1}_p = (n/p)\mathbf{1}_n \, . \tag{4.6}$$

To obtain matrix $\mathbf{D}_i$, the model applies the Sinkhorn algorithm. According to [49], for a general square cost matrix $\mathbf{S}$ of dimension $n \times n$, the worst-case complexity of solving an Optimal Transport problem is $\mathcal{O}(n^3 \log n)$. However, the employed Sinkhorn distances algorithm [49] exhibits empirically a quadratic $\mathcal{O}(n^2)$ complexity with respect to the dimension n of the cost matrix. In our case, complexity is even smaller since $\mathbf{S}$ is not a square matrix, but a rectangular matrix. Except for the application of the Sinkhorn algorithm, π-GNN also maps the adjacency matrix of G_i into a vector, by performing the following operation $\mathbf{D}_i \, \mathbf{A}_i \mathbf{D}_i^\top$ which requires $\mathcal{O}(n^2 p + p^2 n)$ time, and since $p \leqslant n$, this corresponds to $\mathcal{O}(n^2 p)$ time. Moreover, to project the features (if any) of G_i into a vector, the model performs the following matrix multiplication $\mathbf{D}_i \, \mathbf{X}_i$ which takes $\mathcal{O}(pnd)$ time where d is the dimension of the vertex representations. These operations can be efficiently performed on a GPU.

4.4.4 Dustbins

For some tasks, not all vertices of an input graph need to be taken into account. For instance, in some cases, just the existence of a single subgraph in a graph could be a good indicator of class membership. Furthermore, some graphs may consist of fewer than p vertices (i.e. the rows of matrix $\mathbf{W}$). To let the model suppress some vertices of the input graph and/or some latent vertices, we add to each set of vertices a dustbin so that unmatched vertices are explicitly assigned to it. This technique is common in graph matching and has been employed in other models [197]. We expand matrix $\mathbf{S}_i$ to obtain $\bar{\mathbf{S}}_i \in \mathbb{R}^{(n+1)\times(p+1)}$ by appending a new row and column, the vertex-to-bin, bin-to-vertex and bin-to-bin similarity scores, filled with a single learnable parameter $z \in \mathbb{R}$:

$$\bar{\mathbf{S}}_i^{j,p+1} = \bar{\mathbf{S}}_i^{n+1,k} = \bar{\mathbf{S}}_i^{n+1,p+1} = z \qquad \forall j \in [n+1], k \in [p+1] \, .$$

While vertices of the input graph (resp. latent vertices) will be assigned to a single latent vertex (resp. vertex of the input graph) or the dustbin, each dustbin has as many matches as there are vertices in the other set. We denote as $\mathbf{a} = [\mathbf{1}_n^\top \, p]^\top$ and $\mathbf{b} = [\mathbf{1}_p^\top \, n]^\top$ the number of expected matches for each vertex and dustbin in the two sets of vertices (i.e. input

TABLE 4.1: Summary of the synthetic dataset that we used in our experiments.

Synthetic Dataset	
Max # vertices	9
Min # vertices	2
Average # vertices	7.29
Max # edges	36
Min # edges	1
Average # edges	11.34
# graphs	191

vertices and latent vertices). The expanded matrix $\bar{\mathbf{D}}_i$ now has the constraints:

$$\bar{\mathbf{D}}_i \, \mathbf{1}_{p+1} = \mathbf{a} \quad \text{and} \quad \bar{\mathbf{D}}_i^\top \, \mathbf{1}_{n+1} = \mathbf{b} \, .$$

After the linear assignment problem has been solved, we can drop the dustbins and recover $\mathbf{D}_i$. Thus we can retain the first n rows and first k columns of matrix $\bar{\mathbf{D}}_i$.

4.5 Experimental Evaluation

In this section, we evaluate the performance of the proposed π-GNN model on a synthetic dataset, but also on real-world graph classification and graph regression datasets. We also evaluate the runtime performance and scalability of the proposed model.

4.5.1 Synthetic Dataset

To empirically verify that the proposed model can learn representations of high quality, we generated a dataset that consists of 191 small graphs, and for each pair of these graphs, we computed the Frobenius distance by solving the problem of Equation 4.1. Since the Frobenius distance function is intractable for large graphs, we generated graphs consisting of at most 9 vertices. Furthermore, each graph is connected and contains at least 1 edge. The dataset consists of different types of synthetic graphs. These include simple structures such as cycle graphs, path graphs, grid graphs, complete graphs, and star graphs, but also randomly-generated graphs such as Erdős-Rényi graphs, Barabási-Albert graphs and Watts-Strogatz graphs. Table 4.1 shows statistics of the synthetic dataset that we used in our experiments.

There are $^{191*192}/_{2} = 18,336$ pairs of graphs in total (including pairs consisting of a graph and itself). Based on this dataset, we generated a regression problem where the task is to predict the number of vertices of the input graph. In other words, the target of a graph G that consists of n vertices is $y = n$.

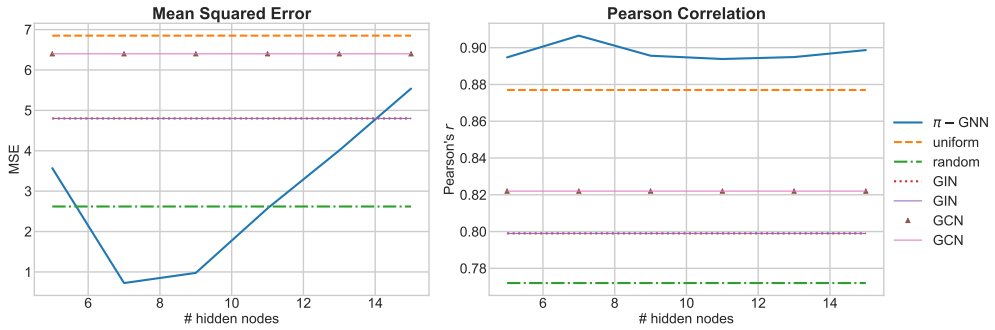


FIGURE 4.1: Mean Squared Error and Pearson Correlation of the Frobenius distances with respect to the number of latent nodes

We annotated the vertices of all graphs with two features (degree and number of triangles in which a vertex participates), we split the dataset into a training and a validation set, and we trained an instance of the proposed π-GNN model on the dataset. We stored the model that achieved the lowest validation loss in the disk, and we then retrieved it and performed a forward pass to obtain the representations $\mathbf{v}_i^{\text{adj}}$, $i \in \{1, \ldots, 191\}$ of all the graphs that are contained in the dataset.

Given these representations, we computed the distance between each pair of graphs G_i, G_j as $d(G_i, G_j) = ||\mathbf{v}_i^{\text{adj}} - \mathbf{v}_j^{\text{adj}}||_2$. We then compared these distances against the Frobenius distances between all pairs of graphs. We experimented with 6 different values for the number of latent vertices, i.e. $5, 7, 9, 11, 13$ and 15. Note that the largest graph in our dataset consists of 9 vertices. To assess how well the proposed model approximates the Frobenius distance function, we employed two evaluation metrics: the mean squared error (MSE) and the Pearson correlation coefficient. Figure 4.1 illustrates the achieved MSE values and correlation coefficients for the different number of latent vertices.

Meta Parameters Selection We compared the proposed model against the following two baselines: (1) *random*: this method randomly generates a permutation matrix for each graph, and the permutation matrix is applied to the graph's adjacency matrix. Then, the distance between two graphs is computed as the Frobenius norm of the difference of the emerging matrices. (2) *uniform*: this baseline generates for all graphs a uniform "soft" permutation matrix (i.e. all the elements are set equal to $1/9$). Then, once again, the distance between two graphs is computed as the Frobenius norm of the difference of the emerging matrices. Besides these two baselines, we also compare the proposed model against GCN [118] and GIN [238]. We treat the output of the readout function as the vector representation of a graph. Then, the distance between two graphs is defined as the Euclidean distance between the graphs' representations.

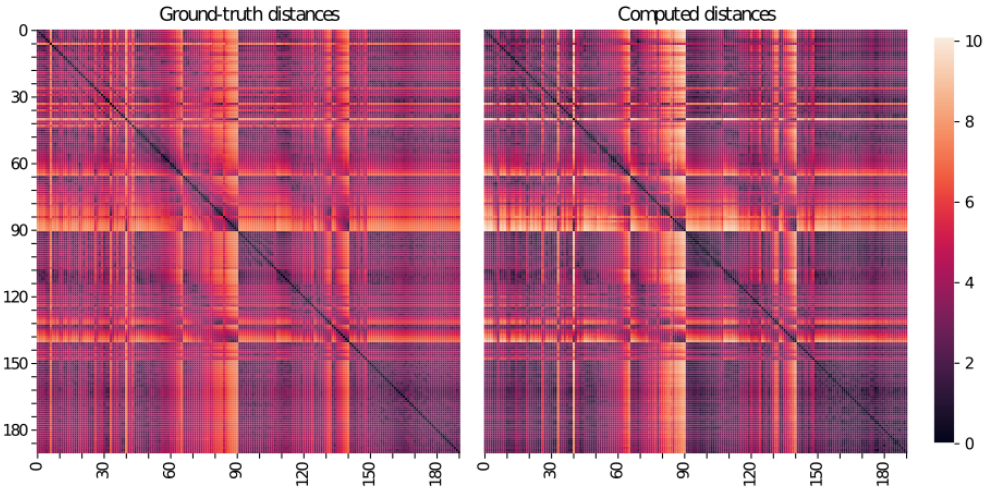


FIGURE 4.2: A heatmap of distances produced by the function of Equation 4.1 and by the proposed model.

We observe that for a number of latent vertices equal to 7 and 9, the representations produced by the proposed model achieve the lowest MSE values (less than 1), which demonstrate that they can learn high-quality representations that produce meaningful graph distances. On the other hand, most of the baselines fail to generate graph representations that can yield distances similar to those produced by the Frobenius distance function (MSE greater than 4 for most of them). Furthermore, for all considered number of latent vertices, the distances that emerge from the representations learned by the proposed model are very correlated with the ground-truth distances (correlation approximately equal to 0.9) and more correlated than the distances produced by any other method.

In Figure 4.2, we also provide a heatmap that illustrates the 191×191 matrix of Frobenius distances (left) where the element in the i-th row and the j-th column corresponds to the Frobenius distance between graphs G_i and G_j, and the matrix of distances produced by the proposed model (right). Clearly, the corresponding values in the two matrices are close to each other, which demonstrates that on datasets that contain small graphs, the proposed model can indeed accurately capture the distance between them. Due to the prohibitive computational complexity of Frobenius distance, we could not experimentally verify that this also holds in the case of larger graphs.

4.5.2 Real-World Datasets

Datasets. We evaluated the proposed model on the following well-established graph classification benchmark datasets: MUTAG, D&D, NCI1, PROTEINS, ENZYMES, IMDB-BINARY, IMDB-MULTI, REDDIT-BINARY, REDDIT-MULTI-5K, and COLLAB [160]. MUTAG contains 188 mutagenic aromatic and heteroaromatic nitro compounds, and the task is to predict whether or not each chemical compound has a mutagenic

effect on the Gram-negative bacterium *Salmonella typhimurium* [58]. D&D contains more than one thousand protein structures whose nodes correspond to amino acids and edges to amino acids that are less than 6 Ångstroms apart. The task is to predict if a protein is an enzyme or not [62]. NCI1 consists of several thousands of chemical compounds screened for activity against non-small cell lung cancer and ovarian cancer cell lines [224]. PROTEINS contains proteins, and again, the task is to classify proteins into enzymes, and non-enzymes [28].

ENZYMES contains 600 tertiary protein structures represented as graphs obtained from the BRENDA enzyme database, and the task is to assign the enzymes to their classes (Enzyme Commission top-level enzyme classes) [28]. IMDB-BINARY and IMDB-MULTI consist of graphs that correspond to movie collaboration networks. Each graph is the ego-network of an actor/actress, and the task is to predict which genre an ego-network belongs to [239]. REDDIT-BINARY and REDDIT-MULTI-5K contain graphs that model interactions between users of Reddit. Each graph represents an online discussion thread, and the task is to classify graphs into either communities or subreddits [239]. COLLAB is a scientific collaboration dataset that consists of the ego-networks of researchers from three subfields of Physics, and the task is to determine the subfield of Physics to which the ego-network of each researcher belongs [239].

We also assessed the proposed model's effectiveness on two graph classification datasets from the Open Graph Benchmark (OGB) [103], a collection of challenging large-scale datasets. Specifically, we experimented with two molecular property prediction datasets: ogbg-molhiv and ogbg-molpcba. Ogbg-molhiv is a collection of graphs, that represent molecules [232]. It contains $41,127$ graphs with an average number of 25.5 nodes per graph and an average number of 27.5 edges per graph. Nodes are atoms, and the edges correspond to chemical bonds between atoms. The graphs contain node features that are processed as in [103]. The evaluation metric is ROC-AUC. Ogbg-molpcba is another molecular property prediction dataset from [103]. It contains $437,929$ graphs with an average number of 26 nodes per graph and an average number of 28 edges per graph. Each graph corresponds to a molecule, where nodes are atoms, and edges show the chemical bonds. The node features are 9-dimensional, and the end task contains 128 sub-tasks. The evaluation metric is Average Precision due to the skewness of the class balance.

For the regression task, we conducted an experiment on the QM9 dataset [184]. The QM9 dataset contains approximately $134k$ organic molecules [184]. Each molecule consists of Hydrogen (H), Carbon (C), Oxygen (O), Nitrogen (N), and Flourine (F) atoms and contains up to 9 heavy (non Hydrogen) atoms. Furthermore, each molecule has 12 target properties to predict.

TABLE 4.2: Classification accuracy (± standard deviation) of the proposed model and the baselines on the 10 benchmark datasets. OOR means Out of Resources, either time (>72 hours for a single training) or GPU memory.

	MUTAG	**D&D**	**NCI1**	**PROTEINS**	**ENZYMES**
DGCNN	84.0 (± 6.7)	76.6 (± 4.3)	76.4 (± 1.7)	72.9 (± 3.5)	38.9 (± 5.7)
DiffPool	79.8 (± 7.1)	75.0 (± 3.5)	76.9 (± 1.9)	73.7 (± 3.5)	59.5 (± 5.6)
ECC	75.4 (± 6.2)	72.6 (± 4.1)	76.2 (± 1.4)	72.3 (± 3.4)	29.5 (± 8.2)
GIN	84.7 (± 6.7)	75.3 (± 2.9)	**80.0** (± 1.4)	73.3 (± 4.0)	**59.6** (± 4.5)
GraphSAGE	83.6 (± 9.6)	72.9 (± 2.0)	76.0 (± 1.8)	73.0 (± 4.5)	58.2 (± 6.0)
π-GNN	86.3 (± 6.2)	**77.6** (± 3.2)	76.9 (± 0.9)	72.2 (± 3.1)	57.5 (± 5.8)
π-GNN-d	**86.4** (± 5.6)	76.1 (± 4.1)	77.6 (± 1.5)	71.7 (± 2.6)	56.1 (± 5.0)

	IMDB-B	**IMDB-M**	**REDDIT-B**	**REDDIT-5K**	**COLLAB**
DGCNN	69.2 (± 3.0)	45.6 (± 3.4)	87.8 (± 2.5)	49.2 (± 1.2)	71.2 (± 1.9)
DiffPool	68.4 (± 3.3)	45.6 (± 3.4)	89.1 (± 1.6)	53.8 (± 1.4)	68.9 (± 2.0)
ECC	67.7 (± 2.8)	43.5 (± 3.1)	OOR	OOR	OOR
GIN	**71.2** (± 3.9)	48.5 (± 3.3)	89.9 (± 1.9)	**56.1** (± 1.7)	75.6 (± 2.3)
GraphSAGE	68.8 (± 4.5)	47.6 (± 3.5)	84.3 (± 1.9)	50.0 (± 1.3)	73.9 (± 1.7)
π-GNN	69.7 (± 3.8)	48.3 (± 3.7)	**90.0** (± 1.2)	53.2 (± 1.5)	73.1 (± 1.2)
π-GNN-d	70.8 (± 4.3)	**48.9** (± 3.5)	87.9 (± 1.8)	49.1 (± 2.7)	**75.7** (± 1.7)

Experimental Setup. In the first part of the graph classification experiments, we compare π-GNN against the following five MPNNs: (1) DGCNN [249], (2) DiffPool [244], (3) ECC [211], (4) GIN [238], and (5) GraphSAGE [90]. To evaluate the different methods, we employ the framework proposed in [68]. Therefore, we perform 10-fold cross-validation, and within each fold, a model is selected based on a 90%/10% split of the training set. Since we use the same splits as in [68], we provide the results reported in that paper for all the common datasets.

In the case of the OGB datasets, we compare π-GNN against the following models that have achieved top places on the OGB graph classification leaderboard: GCN [118], GIN [238], PNA [46], DGN [20], and PHC-GNN [127]. All these baselines belong to the family of MPNNs. Both datasets are already split into training, validation, and test sets, while all reported results are averaged over 10 runs.

In the graph regression task, we compare the proposed model against the following five models: (1) DTNN [232], (2) MPNN [232], (3) 1-2-GNN [161], (4) 1-2-3-GNN [161], and (5) PPGN [150]. The dataset is randomly split into 80% train, 10% validation, and 10% test. We trained a different network for each quantity. For the baselines, we use the results reported in the respective papers.

TABLE 4.3: Performance on the ogbg-molhiv and ogbg-molpcba datasets.

	ogbg-molhiv ROC-AUC	**ogbg-molpcba** Avg. Precision
DGN	79.70 ± 0.97	28.85 ± 0.30
PNA	79.02 ± 1.32	28.38 ± 0.35
PHC-GNN	79.34 ± 1.16	29.47 ± 0.26
GCN	76.06 ± 0.97	20.20 ± 0.24
GIN	75.58 ± 1.40	22.66 ± 0.28
π-GNN	79.12 ± 1.50	28.11 ± 0.32
π-GNN-d	79.09 ± 1.31	28.22 ± 0.41

Meta Parameters Selection For the proposed π-GNN, we provide results for two different instances: π-GNN, and π-GNN-d that correspond to models without and with dustbins, respectively. In all our experiments, we annotate each vertex with two structural features: (i) its degree and (ii) the number of triangles in which it participates. If vertices are already annotated with attributes, we set the value of α in Equation 4.5 to 0.5. In case vertices are annotated with discrete labels, we first map these labels to one-hot vectors.

For all standard graph classification datasets, we set the batch size to 64 and the number of epochs to 300. We use the Adam optimizer with a learning rate equal to 10^{-3}. Layer normalization [10] is applied on the $\mathbf{v}_i^{\text{adj}}$ and $\mathbf{v}_i^{\text{att}}$ vector representations of graphs, and the two outputs are fed to two separate two layer MLPs with hidden-dimension sizes of 256 and 128. The two emerging vectors are then concatenated and further fed to a final two-layer MLP.

Hyper-parameter Tuning The hyperparameters we tune for each dataset and model are the number of latent vertices $\in \{20, 30\}$, and the hidden-dimension size of the fully-connected layer we employ to transform the vertex features $\in \{32, 64\}$. For the experimentation on the OGB datasets, we set the batch size to 128 for ogbg-molhiv, and we choose the batch size from $\{128, 256\}$ for the ogbg-molpcba. Moreover, we choose the number of latent vertices from $\{20, 30, 40\}$, while we set the hidden-dimension size of the fully-connected layer that transforms the vertex features to 128. All the other experimental settings are the same as above. For the QM9 dataset, we set the batch size to 128, the number of latent vertices to 40, the hidden-dimension size of the vertex features to 128, and we use an adaptive learning rate decay based on validation performance. All the other experimental settings are the same as above.

Graph classification results. For the standard graph classification datasets, we report in Table 4.2 average prediction accuracies and standard deviations. We observe that the proposed model achieves the

TABLE 4.4: Mean absolute errors of the proposed model and the baselines on the QM9 dataset.

Target	Method						
	DTNN	**MPNN**	**1-2-GNN**	**1-2-3-GNN**	**PPGN**	**π-GNN**	**π-GNN-d**
μ	0.244	0.358	0.493	0.476	**0.0934**	0.536	0.538
α	0.95	0.89	**0.27**	**0.27**	0.318	0.374	0.372
$\varepsilon_{\text{HOMO}}$	0.00388	0.00541	0.00331	0.00337	**0.00174**	0.00394	0.00394
$\varepsilon_{\text{LUMO}}$	0.00512	0.00623	0.00350	0.00351	**0.0021**	0.00419	0.00421
$\Delta\varepsilon$	0.0112	0.0066	0.0047	0.0048	**0.0029**	0.0055	0.0055
$\langle R^2 \rangle$	17.0	28.5	21.5	22.9	**3.78**	26.34	26.16
ZPVE	0.00172	0.00216	**0.00018**	0.00019	0.000399	0.000235	0.000262
U_0	2.43	2.05	0.0357	0.0427	0.022	**0.0210**	**0.0210**
U	2.43	2.00	0.107	0.111	0.0504	0.0255	**0.0244**
H	2.43	2.02	0.070	0.0419	0.0294	0.0216	**0.0202**
G	2.43	2.02	0.140	0.0469	0.024	0.0211	**0.0203**
C_{v}	0.27	0.42	0.0989	**0.0944**	0.144	0.162	0.165

highest classification accuracy on 5 out of 10 datasets. Furthermore, it yields the second-best accuracy on 3 out of the remaining 5 datasets. GIN also achieves high performance since it outperforms all the other models on 4 datasets. On the PROTEINS dataset, π-GNN is the worst performing model, while on ENZYMES, it fails to achieve accuracy similar to that of GIN. The vertices of those two datasets are annotated with continuous attributes, and as discussed below, these attributes probably provide more information than the graph structure itself since we found that π-GNN can even outperform GIN on those two datasets if we take into account only the $\mathbf{v}_i^{\text{att}}$ vectors and ignore the ones that emerge from the adjacency matrices. Between the two variants of the proposed model, none of them consistently outperforms the other. Interestingly, it appears that their performance depends on the type of graphs and the associated task. For example, π-GNN-d outperforms π-GNN on the two IMDB datasets, while the latter outperforms the former on the two REDDIT datasets.

Table 4.3 summarizes the test scores of the proposed model and the baselines on the two OGB graph property prediction datasets. On both datasets, the two π-GNN instances achieve high levels of performance, while they also achieve very similar performance to each other. More specifically, on ogbg-molhiv, the two models outperform 3 out of the 5 baselines and achieve a test ROC-AUC score very close to that of the best performing model. On ogbg-molpcba, they beat 2 out of the 5 baselines, and again, they reach similar levels of performance to that of the model that performed the best.

Graph regression results. Table 4.4 illustrates mean absolute errors achieved by the different models on the QM9 dataset. On 4 out of 12 targets, both variants of the proposed model outperform the baselines providing evidence that π-GNN can also be very competitive in graph

TABLE 4.5: Classification accuracy (± standard deviation) of the proposed model and the baselines on the 5 chemo/bio-informatics datasets.

	MUTAG	D&D	NCI1	PROTEINS	ENZYMES
π-GNN	86.3 (± 6.2)	77.6 (± 3.2)	76.9 (± 0.9)	72.2 (± 3.1)	57.5 (± 5.8)
π-GNN ($\alpha = 1$)	88.1 (± 5.8)	76.1 (± 3.7)	77.0 (± 1.7)	72.5 (± 2.9)	54.3 (± 4.1)
π-GNN ($\alpha = 0$)	86.1 (± 6.3)	77.6 (± 3.2)	75.2 (± 1.7)	73.7 (± 2.5)	61.3 (± 5.9)
π-GNN-d	86.4 (± 5.6)	76.1 (± 4.1)	77.6 (± 1.5)	71.7 (± 2.6)	56.1 (± 5.0)
π-GNN-d ($\alpha = 1$)	88.3 (± 6.1)	75.0 (± 3.6)	77.0 (± 1.4)	71.0 (± 3.2)	55.8 (± 5.3)
π-GNN-d ($\alpha = 0$)	84.0 (± 8.1)	76.2 (± 3.9)	74.0 (± 1.4)	73.3 (± 3.5)	59.3 (± 5.7)

regression tasks. On most of the remaining targets, the proposed model yields mean absolute errors slightly smaller than those of the best-performing methods. However, on two targets (μ and $\langle R^2 \rangle$), it is largely outperformed by PPGN. Overall, even though the proposed model utilizes two simple structural features, in most cases, its performance is on par with very expressive models (e.g. 1-2-3-GNN, PPGN), which are equivalent to higher-dimensional variants of the WL algorithm in terms of distinguishing between non-isomorphic graphs.

4.5.3 Runtime Analysis

We next study how the empirical running time of the model varies with respect to the values of n (i.e. number of vertices of input graphs) and p (i.e. number of latent vertices). We generated a dataset consisting of 200 graphs. All graphs are instances of the Erdős-Rényi graph model and consist of n vertices, while they are divided into two classes (of equal size) where the graphs that belong to the first class are sparser than the ones that belong to the second. In the first experiment, we set $p = 10$, and we increase the number of vertices n of the 200 input graphs from 10 vertices to 100 vertices. In the second experiment, we set $n = 50$, and we increase the number of latent vertices p from 10 to 100. We set the batch size to 64 and train the model for 100 epochs. In both cases, we measure the average running time per epoch. The results are illustrated in Figure 4.3. The running time of the model seems to be independent of both the size of the input graphs and the number of latent vertices. We hypothesize that this has to do with the fact that the model mainly performs standard matrix operations that can be efficiently parallelized on a GPU.

4.5.4 Graph Structure vs. Vertex Attributes

In our experiments, when vertex attributes, or vertex labels are available, besides the representation $\mathbf{v}_i^{\text{adj}}$ that emerges from the adjacency matrix, we also generate a representation $\mathbf{v}_i^{\text{att}}$ that is associated with the matrix

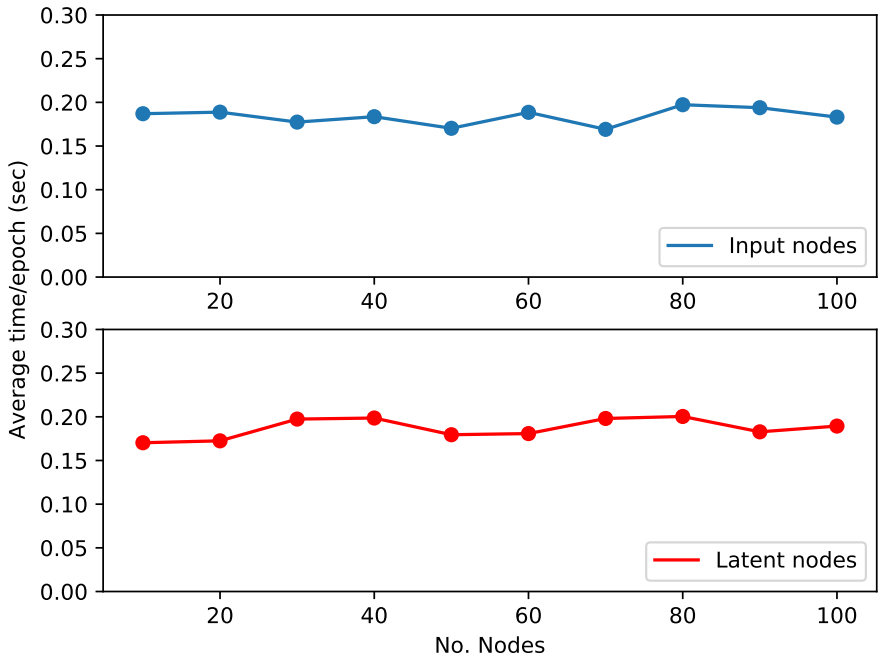


FIGURE 4.3: Average running time per epoch with respect to the number of vertices of the input graphs n (top), and to the number of latent vertices p (bottom).

of vertex features, and we combine the two representations to produce the output.

In this set of experiments, for the 5 datasets that contain either vertex labels or vertex attributes, we investigate whether utilizing solely $\mathbf{v}_i^{\text{adj}}$ or $\mathbf{v}_i^{\text{att}}$ can lead to some improvement in performance (i.e. $\alpha = 1$ in Equation 4.5 and ignore $\mathbf{v}_i^{\text{att}}$ or $\alpha = 0$ in Equation 4.5 and ignore $\mathbf{v}_i^{\text{adj}}$).

The results are reported in Table 4.5 and suggest that on some datasets, the adjacency matrices are more important than the vertex features, while on other datasets, the vertex features outweigh the adjacency matrices in terms of importance. For instance, on MUTAG, taking only the adjacency matrices into account yields an absolute improvement of at least 1.8% in accuracy over the standard π-GNN architecture, while on NCI1, the model's performance significantly drops if the adjacency matrices are ignored, and only the vertex features are taken into account. On the other hand, on the ENZYMES and PROTEINS datasets, using solely the $\mathbf{v}_i^{\text{att}}$ vector as a graph's representation results in a significant increase in performance (an absolute improvement in accuracy of at least 3.2% and 1.2%, respectively). Note that the vertices of these two datasets are annotated with continuous attributes, and these attributes might be better indicators of class membership than the graph structure itself.

4.6 Conclusion

In this chapter, we first identified some limitations of graph embedding approaches, and we then proposed π-GNN, a novel GNN architecture which learns a "soft" permutation matrix for each input graph and uses this matrix to map the graph into a vector. The ability of π-GNN to approximate the ground-truth graph distances was demonstrated through a synthetic experimental study. Finally, the proposed model was evaluated on several graph classification and graph regression tasks, where it performed on par with state-of-the-art models.

Part II

Receptive Field

Chapter 5

Learning Graph Shift Operators

5.1 Introduction

The topology of the observations plays a central role when performing machine learning tasks on graph structured data. A variety of supervised, semi-supervised or unsupervised graph learning algorithms employ different forms of operators that encode the topology of these observations. The most commonly used operators are either the adjacency matrix, the Laplacian matrix or their normalised variants. All of these matrices belong to a general set of linear operators, the *Graph Shift Operators (GSOs)* [195, 151].

Graph Neural Networks (GNNs) are representative cases of algorithms that use chosen GSOs to encode the graph structure, i.e., to encode neighbourhoods used in the aggregation operators. Several GNN variants [118, 90, 238] choose different variants of normalised adjacency matrices as GSOs. Interestingly, in a variety of tasks and datasets, the authors suggest the incorporation of explicit structural information of neighborhoods into the model, in order to provide a more expressive representation of the data topology and observe improved results [174, 248, 245]. In most of these approaches, the GSO is chosen without an analysis of the impact of this choice of representation. From this observation arise our two research questions.

Question 1: *Is there a single optimal representation to encode graph structures or is the optimal representation task- and data-dependent?*

On different tasks and datasets, the choice between the different representations encoded by the different graph shift operator matrices has shown to be a consequential decision. Due to the past successful approaches that use different GSOs for different tasks and datasets, it is natural to assume that there is no single optimal representation for all scenarios. Finding an optimal representation of network data could contribute positively to a range of learning tasks such as node and graph classification or community detection. Fundamental to this search is an answer to Question 1. In addition, we pose the following second research question.

Question 2: *Can we learn such an optimal representation to encode graph structure in a numerically stable and computationally efficient way?*

The utilisation of a GSO as a topology representation is currently a hand-engineered choice of normalised variants of the adjacency matrix. Thus, the learnable representation of node interactions is transferred into either convolutional filters [118, 90] or attention weights [220], keeping the used GSO constant. In this work, we suggest a parametrisation of the GSO. Specific parameter values in our proposed parametrised (and differentiable) GSO result in the most commonly used GSOs, namely the adjacency, unnormalised Laplacian and both normalised Laplacian matrices, and GNN aggregation functions, e.g., the averaging and summation message passing operations. The beauty of this innovation is that it can be seamlessly included in both message passing and convolutional GNN architectures. Optimising the operator parameters will allow us to find answers to our two research questions.

The remainder of this chapter is organised as follows. In Section 5.2 we give an overview of related work in the literature. Then in Section 5.3 we define our parametrised graph shift operator (PGSO) and discuss how it can be incorporated into many state-of-the-art GNN architectures. This is followed by a spectral analysis of our PGSO in Section 5.4, where we observe good numerical stability in practice. In Section 5.5 we analyse the performance of GNN architectures augmented by the PGSO in a node classification task on a set of stochastic blockmodel graphs with varying sparsity and on learning tasks performed on several real-world datasets.

5.2 Related Work

Graph shift operators emerge in different research fields in physics, network science, computer science and mathematics, taking usually the form of either graph Laplacian normalisations or variants of the adjacency matrix. An abundant number of machine learning applications exploit the expressivity of Laplacian operators, including unsupervised learning [147, 114], semi-supervised node classification on graph-structured data [118, 201] and supervised learning on computer vision tasks [36]. The majority of these works assumes a specified normalised version of the Laplacian that maintains the structural information of the problem and usually these versions differ depending on the dataset specifications and the end-user task. Recently, new findings on the impact of the chosen Laplacian representation have emerged that highlight the contribution of Laplacian regularisation [52, 191, 53]. The different GSO choices in different tasks indicate a data-dependent relation between the structure of the data and its optimal GSO representation. This observation motivates us to investigate how beneficial a well-chosen GSO can be for a learning task on structured data.

GNNs use a variety of GSOs to encode neighbourhood topologies, either normalizations of the adjacency matrix [238, 90] or normalisations of the graph Laplacian [118, 230]. Due to the efficiency and the predictive performance of GNNs, a research interest has recently emerged in their expressive power. One of the examined aspects is that of the equivalence of the GNNs expressive power with that of the Weisfeiler-Lehman graph isomorphism test [56, 150, 161, 238]. Another research direction is that of analysing the depth and the width of GNNs, moving one step forward to the design of deep GNNs [144, 136, 141, 3]. In this analysis, the authors study phenomena of Laplacian oversmoothing and combinatorial oversquashing, that harm the expressiveness of GNNs. In most of these approaches, however, the used GSO is fixed without a motivation of the choice. We hope that the parametrised GSO that is presented in this work can contribute positively to the expressivity analysis of GNNs.

Klicpera, Weißenberger, and Günnemann [121] demonstrate that varying the choice of the GSO in the message passing step of GNNs can lead to significant performance gains. In Klicpera, Weißenberger, and Günnemann [121] two fixed diffusion operators with a much larger receptive field than the 1-hop neighbourhood convolutions, are inserted into the architectures, leading to a significant improvement of the GNNs' performance. In our work here we replace the GSOs in GNN frameworks with the PGSO, which has a receptive field equal to the 1-hop neighbourhood of the nodes. We find that parameter values of our PGSO can be trained in a numerically stable fashion, which allows us to chose a parametric form unifying the most common GSOs and aggregation functions. As with standard GNN architectures the receptive field of the convolutions is increased in our architectures by stacking additional layers. Klicpera, Weißenberger, and Günnemann [121] increase the size of the receptive field and keep the neighbourhood representation fixed, while we keep the size of the receptive field fixed and learn the neighbourhood representation.

5.3 Parametrised Graph Shift Operators

We define notation and fundamental concepts in Section 5.3.1 and introduce our proposed parametrised graph shift operator $\gamma(A, \mathcal{S})$ in Section 5.3.2. In Section 5.3.3 we provide a detailed discussion of how $\gamma(A, \mathcal{S})$ can be applied to the message-passing operation in GNNs and we demonstrate use cases of the incorporation of $\gamma(A, \mathcal{S})$ in different GNN architectures.

5.3.1 Preliminaries

Let a graph G be a tuple, $G = (V, E)$, where V and E are the sets of nodes and edges and let $|V| = n$. We assume the graph G to be *attributed* with attribute matrix $X \in \mathbb{R}^{n \times d}$, where the i^{th} row of X contains the

d-dimensional attribute vector corresponding to node v_i. We denote the $n \times n$ identity matrix by I_n and the n-dimensional column vector of all ones by $\mathbf{1}_n$. Given the node and edge sets, one can define the *adjacency matrix*, denoted $A \in [0,1]^{n \times n}$, where $A_{ij} \neq 0$ if and only if $(i,j) \in E$, and the degree matrix of A as $D = \mathrm{Diag}(A\mathbf{1}_n)$.

Recently the notion of a GSO has been defined as a general family of operators which enable propagation of signals over graph structures [195, 208].

Definition 5. Graph Shift Operator A matrix $S \in \mathbb{R}^{n \times n}$ is called a *Graph Shift Operator* (GSO) if it satisfies $S_{ij} = 0$ for $i \neq j$ and $(i,j) \notin E$ [151, 74].

This general definition includes the adjacency and Laplacian matrices as instances of its class.

Remark 4. *According to Definition 5, the existence of an edge $(i,j) \in E$ does not imply a nonzero entry in the GSO, $S_{ij} \neq 0$. Hence, the correspondence between a GSO and a graph is not bijective in general.*

5.3.2 Parametrised GSO

We begin by defining our parametrised graph shift operator.

Definition 6. We define the *parametrised graph shift operator (PGSO)*, denoted by $\gamma(A, \mathcal{S})$, as

$$\gamma(A, \mathcal{S}) = m_1 D_a^{e_1} + m_2 D_a^{e_2} A_a D_a^{e_3} + m_3 I_n, \tag{5.1}$$

where $A_a = A + aI_n$ and $D_a = \mathrm{Diag}(A_a\mathbf{1}_n)$ is the degree matrix of A_a. We denote the parameter tuple corresponding the $\gamma(A, \mathcal{S})$ by $\mathcal{S} = (m_1, m_2, m_3, e_1, e_2, e_3, a)$ consisting of scalar multiplicative parameters m_1, m_2, m_3, scalar exponential parameters e_1, e_2, e_3 and an additive parameter a.

The main motivation of the parametrised form in Equation (5.1) is to span the space of commonly used GSOs and indeed we are able to generate a wide range of different graph shift operators by choosing different values for the parameter set $\mathcal{S}$. In Table 5.1, we give examples of parameter values in $\gamma(A, \mathcal{S})$ which result in the most commonly used GSOs and message-passing operators in GNNs. Unlike the GSO, the PGSO uniquely identifies the graph it corresponds to, i.e., the GSO and PGSO do not share the property discussed in Remark 4.

TABLE 5.1: Known Graph Shift Operators as parameter choices $\mathcal{S}$ in $\gamma(A, \mathcal{S})$.

$\mathcal{S}=(m_1, m_2, m_3, e_1, e_2, e_3, a)$	**Operator**	**Description**
$(0,1,0,0,0,0,0)$	A	Adjacency matrix and Summation Aggregation Operator of GNNs
$(1,-1,0,1,0,0,0)$	$D-A$	Unnormalised Laplacian matrix L
$(1,1,0,1,0,0,0)$	$D+A$	Signless Laplacian matrix Q [50]
$(0,-1,1,0,-1,0,0)$	$I_n - D^{-1}A$	Random-walk Normalised Laplacian L_{rw}
$(0,-1,1,0,-\frac{1}{2},-\frac{1}{2},0)$	$I_n - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$	Symmetric Normalised Laplacian L_{sym}
$(0,1,0,0,-\frac{1}{2},-\frac{1}{2},1)$	$D_1^{-\frac{1}{2}}A_1D_1^{-\frac{1}{2}}$	Normalised Adjacency matrix of GCNs [118]
$(0,1,0,0,-1,0,0)$	$D^{-1}A$	Mean Aggregation Operator of GNNs [238]

Although we base the definition of $\gamma(A, \mathcal{S})$ on the adjacency matrix, we can define the PGSO using other graph representation matrices, such as the non-backtracking operator B, $\gamma(B, \mathcal{S})$, [125, 26] or the diffusion matrix S, $\gamma(S, \mathcal{S})$ [121].

5.3.3 Suggested method: GNN-PGSO and GNN-mPGSO

Next, we formally discuss how $\gamma(A, \mathcal{S})$ is incorporated in GNN models. Let a GNN model be denoted by $\mathcal{M}(\phi(A), X)$, taking as input a non-parametrised function of the adjacency matrix $\phi(A) : [0, 1]^{n \times n} \to \mathbb{R}^{n \times n}$ and an attribute matrix (in case of an attributed graph) $X \in \mathbb{R}^{n \times d}$. Further, let K denote the number of aggregation layers that $\mathcal{M}$ consists of. The *Parametrised Graph Shift Operator* (PGSO) formulation transforms the GNN model $\mathcal{M}(\phi(A), X)$ into the *GNN-PGSO* model $\mathcal{M}'(\gamma(A, \mathcal{S}), X)$. Moreover, we define the *GNN-mPGSO* model $\mathcal{M}''(\gamma^{[K]}(A, \mathcal{S}^{[K]}), X)$, where $\gamma^{[K]}(A, \mathcal{S}^{[K]}) = [\gamma(A, \mathcal{S}^1), ..., \gamma(A, \mathcal{S}^K)]$, i.e., we assign each GNN layer a different parameter tuple $\mathcal{S}^l$ for $l \in \{1, \dots, K\}$.

PGSO and multiple layers Given the definitions of the GNN-PGSO and the GNN-mPGSO models, we note that the former model is making use of a single parameter set for ***all*** GNN layers, while the latter uses different parameters for each model layer. Thus, for the GNN-PGSO, the model *shares* the parameter *weights* throughout the model layers.

Message-passing steps and convolutions In a spectral-based GNN formulation [233], where the GSO is explicitly multiplied by the model parameters, it is straightforward to replace the GSO with $\gamma(A, \mathcal{S})$. However, with some further analysis $\gamma(A, \mathcal{S})$ can also be incorporated in spatial-based GNNs, where the node update equation is defined as a message-passing step. Here we illustrate the required analysis on the sum-based aggregation operator, where we sum the feature vectors $h_j \in \mathbb{R}^d$ in the neighborhood of a given node v_i, denoted $\mathcal{N}(v_i)$. The sum operator of the neighborhood representations can be reexpressed as: $\sum_{j:v_j \in \mathcal{N}(v_i)} h_j = \sum_{j=1}^{n} A_{ij} h_j$. Using this observation we can derive the application of $\gamma(A, \mathcal{S})$ in a message-passing step to be,

$$(\gamma(A, \mathcal{S})h)_i = m_1 \, (D_a)_i^{e_1} \, h_i + m_2 \sum_{j=1}^{n} (D_a)_i^{e_2} \, (A_a)_{ij} \, (D_a)_j^{e_3} \, h_j + m_3 h_i. \quad (5.2)$$

Examples The following examples highlight the usage of the $\gamma(A, \mathcal{S})$ operator:

1. In the standard GCN [118] the propagation rule of the node representation $H^{(l)} \in \mathbb{R}^{n \times d}$ in a computation layer l is,

$$H^{(l+1)} = \sigma\big(D_1^{-\frac{1}{2}} A_1 D_1^{-\frac{1}{2}} H^{(l)} W^{(l)}\big),$$

where $W^{(l)}$ is the trainable weight matrix and σ denotes a non-linear activation function. The GCN-PGSO and GCN-mPGSO models, respectively, perform the following propagation rules,

$$H^{(l+1)} = \sigma\big(\gamma(A, \mathcal{S})H^{(l)}W^{(l)}\big) \text{ and } H^{(l+1)} = \sigma\big(\gamma(A, \mathcal{S}^l)H^{(l)}W^{(l)}\big).$$

2. The Graph Isomorphism Network (GIN) [238] consists of the following propagation rule for a node representation $h_i^{(l)} \in \mathbb{R}^d$ of node v_i in the computation layer l,

$$h_i^{(l+1)} = \sigma\Big(h_i^{(l)}W^{(l)} + \sum_{j:v_j \in \mathcal{N}(v_i)} h_j^{(l)}W^{(l)}\Big).$$

Using the Equation (5.2) the propagation rule is transformed into the GIN-PGSO formulation as,

$$h_i^{(l+1)} = \sigma\Big(\big(m_1 \left(D_a\right)_i^{e_1} + m_3\big)h_i^{(l)}W^{(l)} + m_2 \sum_{j:v_j \in \mathcal{N}(v_i)} \epsilon_{ij} h_j^{(l)}W^{(l)}\Big),$$

where ϵ_{uv} are edge weights defined as $\epsilon_{ij} = \left(D_a\right)_i^{e_2} \left(D_a\right)_j^{e_3}$. Analogously, we can construct the GIN-mPGSO formulation by superscripting every parameter in $\mathcal{S}$ by (l).

Computational Cost Since in (5.1) the exponential parameters are applied only to diagonal matrices the PGSO and mPGSO are efficiently computable and optimisable. $\gamma(A, \mathcal{S})$ can be extended by using *vector* instead of scalar parameters. Although this extension leads to better expressivity, the computational cost is increased, as the number of parameters then depends on the size of the graph.

5.4 Spectral analysis of $\gamma(A, \mathcal{S})$

In this section we study spectral properties of $\gamma(A, \mathcal{S})$ in theoretical analysis in Section 5.4.1 and through empirical observation in Section 5.4.2. The obtained theoretical results provide a foundation for further analysis of methodology involving the PGSO and allow an efficient observation of spectral support bounds of commonly used GSOs, that are instances of $\gamma(A, \mathcal{S})$.

5.4.1 Theoretical Analysis

Here we investigate the spectral properties of our $\gamma(A, \mathcal{S})$. Throughout this section we assume that we work on undirected graphs. In Theorem 8 we show that $\gamma(A, \mathcal{S})$ has a real spectrum and eigenvectors independent of the parameters $\mathcal{S}$. In Theorem 9 we study the spectral support of $\gamma(A, \mathcal{S})$.

Theorem 8. *$\gamma(A, \mathcal{S})$ has real eigenvalues and a set of real eigenvectors.*

The proof of Theorem 8 follows directly from noticing that $\gamma(A, \mathcal{S})$, which is not symmetric in general, is similar to a symmetric matrix and therefore shares eigenvalues with this symmetric matrix [99, pp. 45, 60]; for the full proof the definition of similar matrices is fundamental.

Definition 7. Two matrices $\Phi, \Psi \in \mathbb{C}^{n \times n}$ are *similar* via a nonsingular similarity matrix $S \in \mathbb{C}^{n \times n}$ if

$$\Psi = S^{-1} \Phi S. \qquad \triangleleft$$

Next we note that the proposed operator $\gamma(A, \mathcal{S})$ is similar to the symmetric matrix $D_a^{-(e_2-e_3)/2} \gamma(A, \mathcal{S}) D_a^{(e_2-e_3)/2}$.

Real, symmetric matrices have a set of real eigenvalues and furthermore, possess a set of real eigenvectors. It is a known property of similar matrices that they share eigenvalues and that for a given eigenvector w of Φ, Sw is an eigenvector of Ψ [99, pp. 45, 60]. Therefore, by the similarity relationship to a real symmetric matrix via a real similarity matrix, $\gamma(A, \mathcal{S})$ has real eigenvalues and a set of real eigenvectors. $\square$

Being able to guarantee real eigenvalues and eigenvectors for all parameter values of $\gamma(A, \mathcal{S})$ enables practitioners to deploy our PGSO in spectral network analysis without having to replace elements of the algorithms which assume a real spectrum. As a result of Theorem 8 eigenvalue computations can be stabilised by working with the symmetric, similar PGSO used in the proof of Theorem 8.

Especially the theoretical analysis of the PGSO and algorithms involving the PGSO is aided by Theorem 8. There are several publications, where the complications and lack of results for complex valued graph spectra are discussed in the case of directed graphs [87, 33, 137]. To remain within the real domain independent of the parameter choice (for undirected graphs) enables analysts to access a wide variety of spectral results, which only hold for real symmetric matrices. An example of such a theorem is Cauchy's interlacing theorem [22, p. 709], which can be used to relate the adjacency matrix spectra of a graph and its subgraphs. Hall, Patel, and Stewart [89] and Porto and Allem [179] have been able to prove interlacing theorems for unnormalised, signless and normalised Laplacians. The fact that the PGSO has a real spectrum presents a first step to extend their work to potentially apply to the PGSO independent of the parameter values. However, this is only one example of a powerful theoretical result, which relies on a real spectrum and is accessible to our PGSO formulation due to the result shown in Theorem 8.

Now that the spectrum of $\gamma(A, \mathcal{S})$ has been shown to be real we study the spectral support of $\gamma(A, \mathcal{S})$. A common way to prove bounds on the spectral support of a matrix is via direct application of the Gershgorin Theorem [22, p. 293] as done in Theorem 9.

Theorem 9. *Let* $C_i = m_1(d_i + a)^{e_1} + m_2(d_i + a)^{e_2+e_3}a + m_3$ *and* $R_i = |m_2|(d_i + a)^{e_2+e_3}d_i$, *where* d_i *denotes the degree of node* v_i. *Furthermore, we denote eigenvalues of* $\gamma(A, \mathcal{S})$ *by* $\lambda_1 \leqslant \lambda_2 \leqslant \ldots \leqslant \lambda_n$. *Then, for all* $j \in \{1, \ldots, n\}$,

$$\lambda_j \in \left[\min_{i \in \{1,\ldots,n\}} (C_i - R_i), \max_{i \in \{1,\ldots,n\}} (C_i + R_i) \right]. \tag{5.3}$$

The proof of 9 follows. We will again be utilising the property that similar matrices share eigenvalues by considering the matrix $D_a^{e_3}\gamma(A, \mathcal{S})D_a^{-e_3}$, which is similar to $\gamma(A, \mathcal{S})$. The Gershgorin Theorem states that all eigenvalues of a matrix are contained in circles centred at the matrix's diagonal elements with radii equal to the corresponding off-diagonal row-sums of the matrix elements in absolute value [22, p. 293]. In Theorem 8 we showed that $\gamma(A, \mathcal{S})$ has a real spectrum and therefore the circles in the Gershgorin Theorem are in fact intervals on the real line in the case of $\gamma(A, \mathcal{S})$. The parametrised form of $D_a^{e_3}\gamma(A, \mathcal{S})D_a^{-e_3}$ is as follows,

$$D_a^{e_3}\gamma(A, \mathcal{S})D_a^{-e_3} = m_1 D_a^{e_1} + m_2 D_a^{e_2+e_3} A_a + m_3 I_n. \tag{5.4}$$

From 5.4 we can simply read off that the diagonal elements, denoted C_i, and off-diagonal row-sums of the matrix elements in absolute value, denoted R_i, of $D_a^{e_3}\gamma(A, \mathcal{S})D_a^{-e_3}$ take the following form,

$$\begin{aligned} R_i &= m_1(d_i + a)^{e_1} + m_2(d_i + a)^{e_2+e_3}a + m_3, \\ C_i &= |m_2|(d_i + a)^{e_2+e_3}d_i. \end{aligned}$$

Hence, via the similarity relationship of $D_a^{e_3}\gamma(A, \mathcal{S})D_a^{-e_3}$ and $\gamma(A, \mathcal{S})$ and a direct application of the Gershgorin Theorem applied to $D_a^{e_3}\gamma(A, \mathcal{S})D_a^{-e_3}$ we obtain the required result. □

Examples For the parametrisation of $\gamma(A, \mathcal{S})$ corresponding to the adjacency matrix, we obtain $C_i = 0$ and $R_i = d_i$. R_i is clearly maximised by the maximum degree and therefore, from (5.3) the spectral support of A is equal to $[-d_{\max}, d_{\max}]$, as required. Similarly, the spectral supports of L, L_{sym} and L_{rw}, can be deduced by plugging in the corresponding parameters into the result in (5.3). For the operator used in the message passing of the GCN [118] a Gershgorin bound such as the one in Theorem 9 has not been calculated yet as far as we are aware. We obtain $C_i = 1/(d_i + 1)$ and $R_i = d_i/(d_i + 1)$. Therefore, from (5.3) the spectral support of the Kipf and Welling operator is restricted to lie within $[-(d_{\max} - 1)/(d_{\max} + 1), 1]$, the lower bound of this interval tends to -1 as $d_{\max} \to \infty$. So only in the limit is their spectral support symmetric around 0.

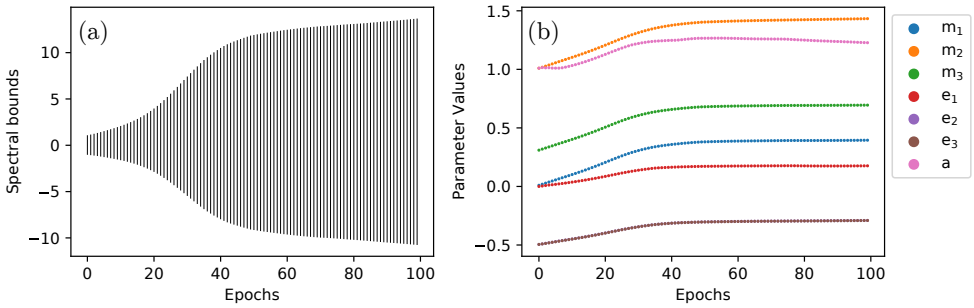


FIGURE 5.1: (a) bounds on the spectral support and (b) parameter values of $\gamma(A, \mathcal{S})$ plotted against the training epochs of a GCN-PGSO applied to a node classification task on the Cora graph. Note that the optimal values of e_2 and e_3 lie close together and hence appear as a single line in (b).

The bounds proven in Theorem 9 allow the observation of the spectral bounds for the many GSOs which can be obtained via specific parameter choices made in our PGSO formulation. In the context of GNNs such bounds are of value since they allow statements about numerical stability and vanishing/exploding gradients to be made. For example in Kipf and Welling [118] the observation that the spectrum of the symmetric normalised Laplacian is contained in the interval $[0, 2]$ motivated them to make use of the "renormalisation trick" to stabilise their computations. Since the PGSO is learned its spectral support varies throughout training. The bounds in Theorem 9 enable us to monitor bounds on the spectral support in a numerically efficient manner, avoiding the computation of eigenvalues at each iteration, as is showcased in the empirical observation in Section 5.4.2.

Since the majority of the commonly used graph shift operators correspond to specific parametrisations of $\gamma(A, \mathcal{S})$ the spectral properties of $\gamma(A, \mathcal{S})$ are general themselves, in the sense that we cannot establish spectral results for $\gamma(A, \mathcal{S})$, which have already been shown to not be present for one of its parametrisations. This generality and lack of specific spectral features, which hold for all parametrisations is precisely one of the strengths of utilising $\gamma(A, \mathcal{S})$, since it allows the graph shift operator to manifest different spectral properties in different tasks, when they are of benefit.

5.4.2 Empirical Observation

In this section we observe bounds on the spectral support, proven in Theorem 9 (Figure 5.1(a)) and optimal parameters (Figure 5.1(b)) of the PGSO incorporated in the GCN during training on a node-classification task on the Cora dataset. More details on the task will be provided in Section 5.5.3.

Surprisingly, the spectral support of the PGSO remains centered at 0 throughout training in Figure 5.1(a) without this being enforced by the design of the PGSO. Recall that the centre of the spectral support intervals has the following parametric form $C_i = m_1(d_i + a)^{e_1} + m_2(d_i + a)^{e_2+e_3}a + m_3$. It is nice to observe that this desirable property of the "renormalised" operator used by Kipf and Welling [118] is preserved throughout training. We also observe that the spectral bounds smoothly increase throughout training. The increasing support is a direct result of the learned PGSO parameters.

As we expected from our analysis of Figure 5.1(a), we observe the parameters of the PGSO to be smoothly varying throughout training in Figure 5.1(b). The parameters in Figure 5.1(b) can be seen to have been initialised at the values corresponding to the chosen GSO in the GCN and from there they smoothly vary towards new optimal values within the first 40 training epochs, which are then stable throughout the remainder of the training, ruling out exploding gradients. It is nice to note that in Section 5.5 Figure 5.6(b) we observe the accuracy of the GCN using the trained PGSO parameters, displayed in Figure 5.1(b), to slightly outperform the standard GCN.

5.5 Experiments

We are evaluating the performance of the PGSO for a synthetic dataset and 8 real-world datasets. In Section 5.5.1, we show the ability of $\gamma(A, \mathcal{S})$ to adapt to varying sparsity scenarios through a stochastic blockmodel study, in Section 5.5.2 we study the sensitivity of the GNN-PGSO's performance to different $\gamma(A, \mathcal{S})$ initialisations and in Section 5.5.3 we evaluate the contribution of PGSO and mPGSO to the GNN performance in real-world datasets.

5.5.1 Sparsity interpretation of $\gamma(A, \mathcal{S})$

The choice of a graph shift operator depends on the structural information of a dataset and on the end task. For example, in graph classification tasks the datasets usually contain small and sparse graphs [232], while in node classification tasks the graphs are larger and denser. In this section, we highlight the ability of $\gamma(A, \mathcal{S})$ to adapt to different sparsity regimes.

Stochastic Blockmodels In order to simulate graphs with varying sparsity levels, we utilise the parametrisation of stochastic blockmodel generation. Stochastic blockmodels (SBMs) are well-studied generative models for random graphs, that tend to contain community, i.e., block, structure [97, 111]. Let k be the number of communities, $\{C_1, ..., C_k\}$ be the k disjoint communities and p_{ij} be the probability of an edge occuring between a node $u \in C_i$ and a node $v \in C_j$. SBMs offer a flexible tool for random graph generation with specific properties, such as the

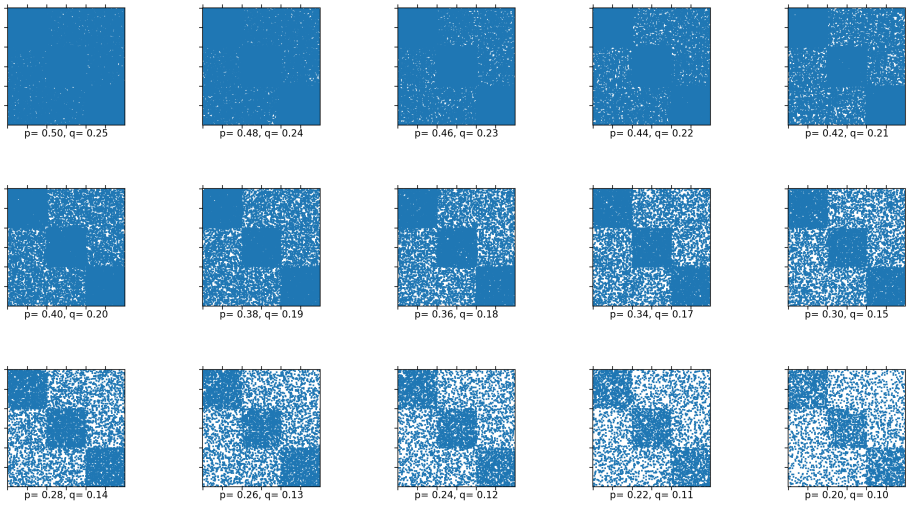


FIGURE 5.2: List of adjacency matrices generated by stochastic blockmodels for varying sparsity by decreasing the probability tuple (p, q). The format of this adjacency matrix visualisation is taken from [65]. The ordering of the node labels in Figure 5.2 corresponds to their block membership in order to highlight the block structure in the adjacency matrix plots.

detectability of the community structure [59] and the *sparsity* level of the graph by choosing appropriate edge probabilities p_{ij}. Here, we focus on a restricted stochastic blockmodel parametrisation where the probabilities of an edge occurring between nodes within the same community are all equal to p, i.e., $p_{ii} = p \quad \forall i \in \{1, ..., k\}$, and the probabilities of an edge between nodes in different communities are all equal to q, i.e., $p_{ij} = q \quad \forall i \neq j, \quad i, j \in \{1, ..., k\}$.

Dataset and experimentation setup In this experiment, we consider 15 p, q parameter combinations $(p, q) \in \{(0.5, 0.25), (0.48, 0.24), ..., (0.22, 0.11)\}$, where by decreasing the parameters p and q we increase the sparsity of the sampled networks. In Figure 5.2 we present a visualization of the generated graphs by displaying their corresponding adjacency matrices. For each parameter set we sample 25 graphs with 3 communities containing 200 nodes each. Figure 5.2 contains plots of adjacency matrices sampled from these 15 parameter combinations. The learning task performed by a GCN will be to assign community membership labels to each node. For all parameter choices

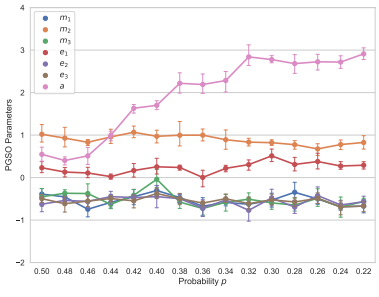


FIGURE 5.3: Mean and standard deviation of the PGSO parameters on SBM

the ratio p/q, i.e., the community detectability, is equal to 2, so that the theoretical difficulty the task remains constant across the different sparsity levels. As in Dwivedi et al. [65], all of the nodes have uninformative attributes, except for one node per community that carries its community membership as its node attribute. We set the train/validation/test splits equal to 80%, 10%, 10% of nodes, respectively, for each community. We use a 3-layer Graph Convolutional Network [118] with hidden size 64 and with our PGSO incorporated as the message passing operator and initialised to the GCN configuration. We, then, observe the learned parameter values of $\gamma(A, \mathcal{S})$ after 200 epochs.

Parameter Values Figure 5.3 shows the average optimal parameter values of $\gamma(A, \mathcal{S})$ found for each one of the 15 node classification datasets. We observe that all parameters remain close to constant as the sparsity of the SBM samples increases, except for the additive parameter a, which can be observed to clearly increase with increasing sparsity levels. The parameter a in the PGSO parametrisation plays a very similar role to the regularisation parameter of the normalised adjacency matrix, which is observed to significantly improve the performance of the spectral clustering algorithm in the task of community detection in the challenging sparse case in Dall'Amico, Couillet, and Tremblay [52] and Qin and Rohe [182]. It is very nice to see that the PGSO automatically varies this regularisation parameter, replicating the beneficial regularisation observed in the literature, without this behaviour being incentivised in any way by the model design or the parametrisation. In Figure 5.3 we can furthermore observe that the e_2 and e_3 values are closely aligned for all sparsity levels indicating that a symmetric normalisation of the adjacency matrix seems to be beneficial in this task.

5.5.2 Sensitivity Analysis of $\gamma(A, \mathcal{S})$ to different initialisations

In Section 5.5.1, we initialised a 3-layer GCN-PGSO model with the original GCN parameter configuration, as shown in Table 5.1. In this section we observe how sensitive the optimal parameters of $\gamma(A, \mathcal{S})$ and the final model accuracy are to different GSO initialisations.

Experimentation setup: We use the GCN-PGSO model, described in Section 5.3.3, for the node classification task on the *Cora* dataset [153]. We consider 5 different initialisations of $\gamma(A, \mathcal{S})$ that correspond to 1) the GCN operator $D_1^{-1/2} A_1 D_1^{-1/2}$, 2) the Adjacency matrix A, 3) the Random-Walk normalised Laplacian $L_{rw} = I - D^{-1}A$, 4) the Symmetric normalised Laplacian $L_{sym} = I - D^{-1/2}AD^{-1/2}$ and 5) a naive all-zeros initialisation, where all $\gamma(A, \mathcal{S})$ parameters are set to 0.

In Figure 5.4, we observe the parameter evolution of $\gamma(A, \mathcal{S})$ over 150 epochs for the different initialisations. The parameter values obtained

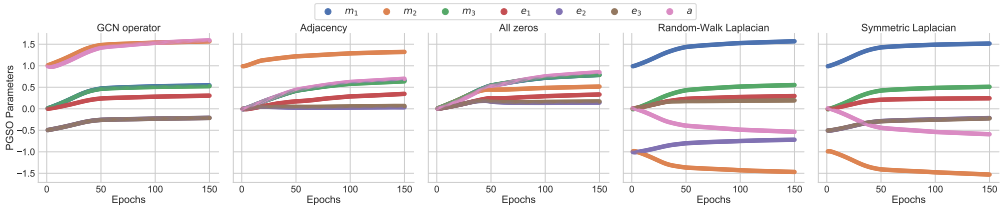


FIGURE 5.4: Parameter evolution of $\gamma(A, \mathcal{S})$ for 150 epochs, when applied on GCN-PGSO model for the node classification task on Cora.

from the GCN, Adjacency matrix and all-zeros initialisations exhibit a great amount of similarity; while the normalised Laplacian initilisations lead to similar optimal values for five of the seven parameters. For all initialisations we observe that parameters m_1, m_3, e_1, e_2, e_3 monotonically increase until they converge. Parameters m_2 and a initially increase for the GCN, Adjacency and all-zeros initialisations, while they initially decrease for the two normalised Laplacians. The achieved accuracy of the five initialisations is plotted in Figure 5.5, where it can be observed that the resulting accuracy from the two normalised Laplacian initialisations is slightly lower than the one achieved by the remaining three initialisations. Overall, the accuracy is not very sensitive to the different initialisations.

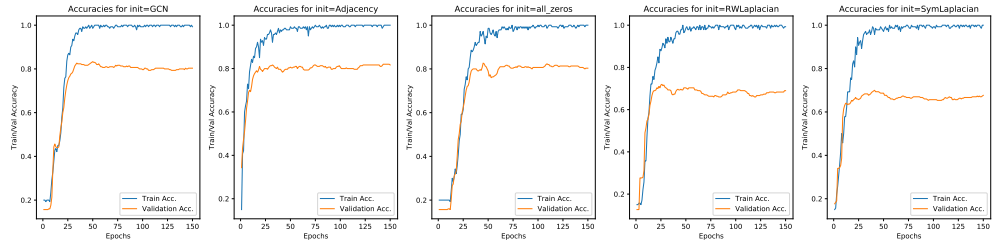


FIGURE 5.5: Train and Validation Accuracy on Cora using 5 different initialisation configurations for PGSO

5.5.3 Real-World scenarios

In this section, we evaluate the contribution of the parametrised GSO, when we apply it to a variety of graph learning tasks. In order to highlight the flexibility of $\gamma(A, \mathcal{S})$, we perform both node classification and graph classification tasks.

Datasets: For node classification, we have used the well-examined datasets *Cora* and *CiteSeer* [153, 79] and *ogbn-arxiv*, that is a citation network from a recently popular collection of graph benchmarks, the Open Graph Benchmark [102]. For graph classification, we have used the extensively-studied TU datasets *MUTAG*, *PTC-MR*, *IMDB-BINARY* and *IMDB-MULTI* [113] and *OGBG-MOLHIV* dataset from OGB [102].

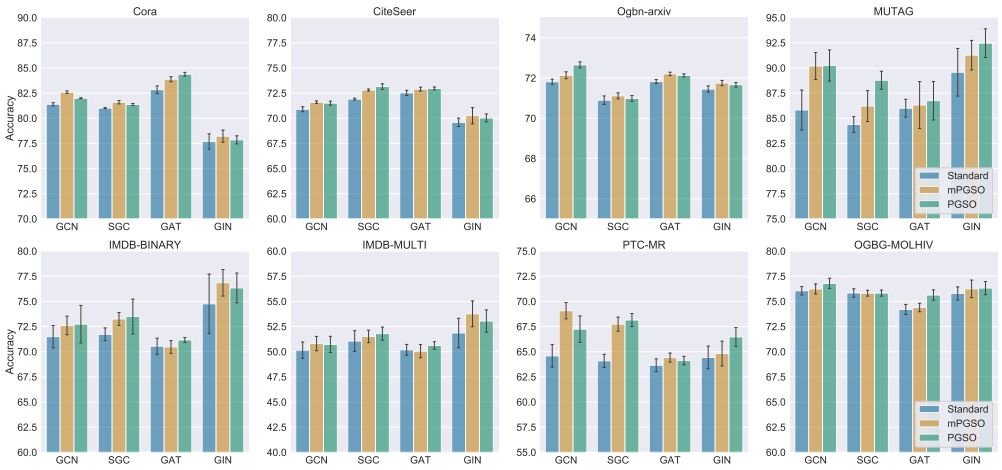


FIGURE 5.6: Classification accuracy results for both node and graph classification tasks (Validation ROC-AUC for OGBG-MOLHIV). Lower case letters denote a node classification task, while capital letters a graph classification task.

Details and statistics of the datasets can be found in Appendix A.1 and in Appendix A.2.

Experimentation Setup: For the node classification datasets Cora and CiteSeer, we performed cross validation with the train/validation/test splits being the same as in Kipf and Welling [118], while for Ogbn-arxiv, we used the same splitting method used in Hu et al. [102], according to the publication dates. For the TU datasets, we performed 10-fold cross validation with grid search hyper-parameter optimisation and for OGBG-MOLHIV, following Hu et al. [102] we used a *scaffold splitting* approach and measured the validation ROC-AUC. We compared the contribution of PGSO and mPGSO on 4 standard GNN baselines: 1) **GCN**: Graph Convolutional Network [118], 2) **GAT**: Graph Attention Network [220], 3) **SGC**: Simplified Graph Convolution [230] and 4) **GIN**: Graph Isomorphism Network [238]. A full description of the experimentation details for each task can be found in Appendix C.1.

In Figure 5.6, we show the contribution of the PGSO and the mPGSO methods, when applied to standard GNNs. For all datasets and GNN architectures, the inclusion of the PGSO or the mPGSO improves the model performance. In the graph classification tasks the performance improvement is higher than the node classification tasks. Specifically, on MUTAG, PTC-MR and IMDB-BINARY, we observe a significant improvement of the classification accuracy for all GNNs. In the comparison between PGSO and mPGSO we do not find a clear winner. For this reason, we prefer the PGSO variant, as it is more efficient and never harms the model's performance. In Appendix C.2 we study the convergence of optimal accuracy and loss values of the GNN-PGSO model in a node-classification and a graph classifcation dataset. In both

experiments we observe the PGSO model to converge slightly faster and to better accuracy and loss values than its conventional counterpart.

5.6 Conclusion

In this chapter, we proposed a parametrised graph shift operator (PGSO), that encodes graph structures and can be included in any graph representation learning scenario. Focusing on graph neural networks (GNNs), we demonstrate that the PGSO can be integrated in the GNN model training. We proved that the PGSO has real eigenvalues and a set of real eigenvectors and derived its spectral bounds, which when observed in practice show that our learned PGSO leads to numerical stable computations. A study on stochastic blockmodel graphs demonstrated the ability of the PGSO to automatically adapt to networks with varying sparsity, independently confirming the positive impact of GSO regularisation which was found in the literature. Experiments on 8 real-world datasets, where both node and graph classification was performed, demonstrate that the accuracy of a representative sample of the current state-of-the-art GNNs can be improved by the incorporation of our PGSO. In answer to our two research questions posed in Section 5.1, our experimental results have shown that the optimal representation of graph structures is task and data dependent. We have furthermore found that PGSO parameters can be incorporated in the training of GNNs and lead to numerically stable learning and message passing operators.

Chapter 6

Increasing the receptive field with multiple hops

6.1 Introduction

In Chapter 5, we introduced the PGSO as a way to parametrize graph shift operators, in order to incorporate the neighborhood encoding into the learning process. However, even with this type of parametric operators, the information that is propagated through a GNN layer corresponds to the direct 1-hop neighborhood. In the recent years, the graph-structured data that are subject of the machine learning focus are more complex and the detection of higher order patterns is necessary. For example, in social network analysis, one might be interested in predicting the interests of users represented by the nodes of a network [240]. In biology, an issue of high interest is the prediction of the functions of proteins modeled as graphs [28]. These applications typically involve tasks, where the required information can be acquired from larger than 1-hop neighborhoods.

The majority of standard GNN variants that address both node-related and graph-related tasks share the same basic ideas [83, 200, 138, 60, 112, 130, 90, 118, 220, 249, 72]. They are based on the aggregation of the representations from the direct 1-hop neighborhood. Thus, information from larger neighborhoods is implicitly expressed. Specifically, after k iterations of the message passing procedure, each node obtains a feature vector which captures the structural information within its k-hop neighborhood. These representations can be used as features for node-related tasks. For graph-related tasks, GNNs compute a feature vector for the entire graph using some permutation invariant readout function such as summing the feature vectors of all the nodes of the graph.

The studies that have made attempts to formally characterize the expressive power of GNNs [161, 238] take into account an aggregation from 1-hop neighborhoods for a single iteration. These studies have compared the expressiveness of GNNs with that of the WL test, and have shown that GNNs do not have more power in terms of distinguishing between non-isomorphic graphs than the WL algorithm. To make matters worse,

it was recently shown that the WL subtree kernel (which capitalizes on the WL test) has insufficient expressive power for identifying fundamental graph properties [123]. It remains though unclear how GNNs encode subgraph/graph information into their learned representations, and whether they can identify such properties, when we make the assumption that they update the node representations based on the interactions with the direct neighbors.

Present Work. In this chapter, we further analyze the representational power of GNNs, with respect to the neighborhood depth. Specifically, we study if GNNs that aggregate information explicitly from 1-hop neighborhoods can identify specific properties of graphs. We say that a GNN identifies a property if no two graphs are mapped to the same feature vector unless they both have or both do not have the property. We demonstrate that the standard GNN fails to identify fundamental graph properties such as connectivity, bipartiteness and triangle-freeness. We show that this limitation of GNNs stems from the myopic nature of the message-passing procedure which only considers the direct neighbors of each node. To account for that, we propose a novel architecture, called k-hop-GNNs, which takes into account not only the immediate neighbors of each node, but its whole k-hop neighborhood. By updating node features using not only the direct neighbors, but taking into account the entire k-hop neighborhood, we can capture structural information that is not visible when aggregating only the 1-hop neighborhood. The proposed model is strictly more powerful than the standard GNN architecture. Furthermore, in contrast to the GNN framework, the proposed architecture is capable of distinguishing global properties such as connectivity. We demonstrate the proposed architecture in a variety of node and graph classification tasks. The results show that the proposed k-hop-GNNs are able to consistently outperform traditional GNNs on most datasets. Our main contributions are summarized as follows:

- We show that standard GNNs with a 1-hop aggregator cannot identify essential graph properties such as connectivity, bipartiteness and triangle-freeness.
- We propose k-hop-GNNs, a novel architecture for performing machine learning on graphs which is more powerful than traditional GNNs.
- We evaluate the proposed architecture on several node classification and graph classification datasets, and achieve performance better or comparable to standard GNNs and to state-of-the-art algorithms.

The rest of this chapter is organized as follows. Section 6.2 presents the standard graph neural network model as a 1-hop aggregator procedure. Section 6.3 analyzes the expressive power of the graph neural network model highlighting its limitations. Section 6.4 presents the proposed

model for performing machine learning tasks on graph-structured data, and shows that it is theoretically more powerful than the standard graph neural network architecture. Section 6.5 evaluates the proposed architecture on several standard datasets. Finally, Section 6.6 concludes.

6.2 GNNs as 1-hop Aggregators

In order to highlight the limitations of the standard GNNs regarding the utilization of the direct neighborhood, we present a formulation of the message passing framework as a 1-hop aggregator process. For the sake of clarity, we redefine the notions of the k-hop neighborhood and the subgraph.

Let $G = (V, E)$ be a graph. Suppose each vertex $v \in V$ is annotated with a feature vector $h_v^{(0)} \in \mathbb{R}^d$. The neighborhood of radius k (or k-hop neighborhood) of a node $v \in V$ is the set of nodes at a distance less than or equal to k from v and is denoted by $\mathcal{N}_k(v)$. Given a set of nodes $S \subseteq V$, the subgraph induced by S is a graph that has S as its node set and it contains every edge of G whose endpoints are in S. The neighborhood subgraph of radius k of a node $v \in V$ is the subgraph induced by the neighborhood of radius k of v and v itself, and is denoted by G_v^k.

Suppose we have a GNN model that contains T neighborhood aggregation layers. In the t^{th} neighborhood aggregation layer ($t > 0$), the hidden state $h_v^{(t)}$ of a node v is updated as follows:

$$\begin{aligned} a_v^{(t)} &= \text{AGGREGATE}^{(t)}\left(\left\{h_u^{(t-1)} | u \in \mathcal{N}_1(v)\right\}\right) \\ h_v^{(t)} &= \text{MERGE}^{(t)}\left(h_v^{(t-1)}, a_v^{(t)}\right) \end{aligned} \tag{6.1}$$

By defining different $\text{AGGREGATE}^{(t)}$ and $\text{MERGE}^{(t)}$ functions, we obtain a different GNN variant. For the GNN to be end-to-end trainable, both functions need to be differentiable. Furthermore, since there is no natural ordering of the neighbors of a node, the $\text{AGGREGATE}^{(t)}$ function must be permutation invariant. There are numerous concrete implementations of the above GNN framework. Some of them integrate the $\text{AGGREGATE}^{(t)}$ and $\text{MERGE}^{(t)}$ steps into a single function [118, 249] as follows:

$$h_v^{(t)} = \frac{1}{|\mathcal{N}_1(v)| + 1} \sum_{u \in \mathcal{N}_1(v) \cup \{v\}} \text{MLP}^{(t)}\left(h_u^{(t-1)}\right)$$

where $\text{MLP}^{(t)}$ is a multi-layer perceptron of the t^{th} neighborhood aggregation layer. Note that the majority of the proposed models use 1-layer perceptrons instead of MLPs. Another widely-used GNN model

is implemented as follows [130]:

$$\begin{aligned} a_v^{(t)} &= \frac{1}{|\mathcal{N}_1(v)|} \sum_{u \in \mathcal{N}_1(v)} \text{MLP}_1^{(t)}\left(h_u^{(t-1)}\right) \\ h_v^{(t)} &= \text{MLP}_2^{(t)}\left(h_v^{(t-1)} + a_v^{(t)}\right) \end{aligned} \tag{6.2}$$

where again $\text{MLP}_1^{(t)}$ and $\text{MLP}_2^{(t)}$ are multi-layer perceptrons of the t^{th} neighborhood aggregation layer.

For node-level tasks, the node feature vectors $h_v^{(T)}$ of the final neighborhood aggregation layer are usually passed on to a fully-connected neural network. For graph-level tasks, GNNs apply a READOUT function to node representations generated by the final neighborhood aggregation layer to obtain a vector representation over the whole graph:

$$h_G = \text{READOUT}\left(\left\{h_v^{(T)} | v \in G\right\}\right) \tag{6.3}$$

Similarly to the $\text{AGGREGATE}^{(t)}$ function, the READOUT function is necessary to be differentiable and permutation invariant. A common READOUT function computes the mean of the representations of the nodes:

$$h_G = \frac{1}{|V|} \sum_{v \in V} h_v^{(T)}$$

However, there have also been proposed more sophisticated functions based on sorting [249], on concatenation across the iterations/layers [238] and on clustering [215, 244]. Based on the definitions of Equations 6.1, 6.2 and 6.3, we can formulate the limitations of the GNNs that derive from the assumption of the 1-hop neighborhood and build our suggested model.

6.3 Limitations of the Standard GNN Model

To gain theoretical understanding of the properties and weaknesses of GNNs, whose aggregators take into account the 1-hop neighborhood, we capitalize on concepts introduced by Goldreich in the context of property testing [82], and further refined by Kriege et al. for investigating the expressive power of graph kernels [123]. The graph property testing was examined, also, in Chapter 3 in the experimental level. Here, we are going to show how the connectivity, bipartiteness and triangle-freeness are theoretically difficult prediction targets for standard GNNs.

Let $\mathcal{G}_n$ be the set of graphs on n vertices, where $n \in \mathbb{N}$. A graph property is a set $\mathcal{P}$ of graphs that is closed under isomorphism. We denote the set of graphs in $\mathcal{P}$ on n vertices by $\mathcal{P}_n$. In this paper we study the following three fundamental graph properties: (1) connectivity, (2) bipartiteness, and (3) triangle-freeness. A graph is *connected* if there is a path from any

vertex to any other vertex in the graph. A graph $G = (V, E)$ is *bipartite* if its set of vertices V can be decomposed into two disjoint sets V_1 and V_2, i.e. $V = V_1 \cup V_2$, such that every edge $e \in E$ connects a vertex in V_1 to a vertex in V_2. Finally, a graph is *triangle-free* if it does not contain a triangle (i.e. a cycle of three vertices). Following the work from Kriege et al. [123], we say that a GNN can identify a property if no two graphs obtain the same representation unless they both have or both do not have the property.

Definition 8. Let $\mathcal{P}$ be a graph property. If for each $n \in \mathbb{N}$, a GNN produces different representations for every $G_1 \in \mathcal{P}_n$ and $G_2 \notin \mathcal{P}_n$, i.e. it holds that $h_{G_1} \neq h_{G_2}$, then we say that $\mathcal{P}$ can be identified by the GNN.

We next study if the standard GNN architecture can identify the above three graph properties. We assume that either all nodes or nodes with the same degree are annotated with the same feature vector. We first show that the standard GNN produces exactly the same representation for the nodes of all regular graphs of a specific degree in $\mathcal{G}_n$ for some $n \in \mathbb{N}$.

Lemma 4. *The standard GNN maps the nodes of two regular graphs of the same size and degree to the same feature vector.*

Proof. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two non-isomorphic regular graphs of the same degree with the same number of vertices. We show for an arbitrary iteration $t \geqslant 1$ and nodes $v_1 \in V_1$, $v_2 \in V_2$ that $h_{v_1}^{(t)} = h_{v_2}^{(t)}$. All nodes have the same initial representation, hence, in iteration 0, it holds that $h_{v_1}^{(0)} = h_{v_2}^{(0)}$. Assume for induction that $h_{v_1}^{(t-1)} = h_{v_2}^{(t-1)}$. Let $\mathcal{M}_{v_1} = \{h_{u_1}^{(t-1)} : u_1 \in \mathcal{N}_1(v_1)\}$ and $\mathcal{M}_{v_2} = \{h_{u_2}^{(t-1)} : u_2 \in \mathcal{N}_1(v_2)\}$ be the multisets of feature vectors of the neighbors of v_1 and v_2, respectively. By the induction hypothesis, we know that $\mathcal{M}_{v_1} = \mathcal{M}_{v_2}$ and that $h_{v_1}^{(t-1)} = h_{v_2}^{(t-1)}$ such that independent of the choice of the $\text{AGGREGATE}^{(t)}$ and $\text{MERGE}^{(t)}$ functions in Equation (6.1), we get $h_{v_1}^{(t)} = h_{v_2}^{(t)}$. This holds as the input to both functions $\text{AGGREGATE}^{(t)}$ and $\text{MERGE}^{(t)}$ is identical. This proves that $h_{v_1}^{(t)} = h_{v_2}^{(t)}$, and thereby the lemma. □

Note that the Lemma implies that two regular graphs with the same node degree and the same size also have the same graph representation because the READOUT function receives the same input. We next show that the GNN architecture cannot identify the three graph properties defined above since for each one of these properties, there exists one regular graph in $\mathcal{P}_n$ and another regular graph of the same degree in $\mathcal{G}_n \backslash \mathcal{P}_n$.

Theorem 10. *The standard GNN cannot identify connectivity, bipartiteness or triangle freeness.*

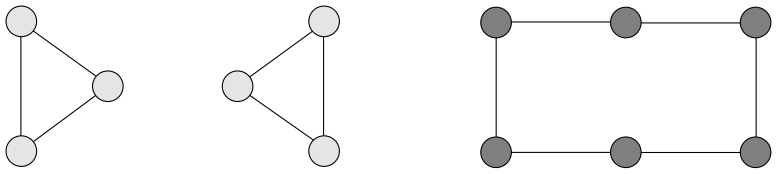


FIGURE 6.1: Two 2-regular graphs on 6 vertices. The two graphs serve as a counterexample for the proof of Theorem 10.

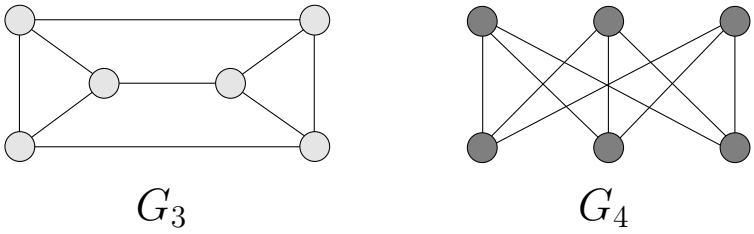


FIGURE 6.2: Two 3-regular graphs on 6 vertices. The two graphs serve as a counterexample for the proof of Theorem 10.

Proof. Consider a cycle with six vertices (graph G_2) and two triangles with three vertices (graph G_1) as illustrated in Figure 6.1. Both G_1 and G_2 are regular graphs of the same degree with the same number of vertices. Hence, according to Lemma 4, after T neighborhood aggregation steps, the nodes of both graphs have obtained identical representations, i.e. $h_v^{(T)} = h_u^{(T)}$, $\forall v, u \in V_1 \cup V_2$. Therefore, independent of the choice of the READOUT function in Equation 6.3, the two graphs will have identical representations, $h_{G_1} = h_{G_2}$, since the input to the READOUT function is identical. Clearly, G_1 is disconnected, while G_2 is connected. Hence, these two graphs correspond to a counterexample to the distinguishability of connectivity. Furthermore, consider the graphs G_3 and G_4 as illustrated in Figure 6.2. Note that G_3 contains triangles, but is not bipartite, whereas G_4 is bipartite, and triangle-free. Both G_3 and G_4 are regular graphs of the same degree with the same number of vertices. Therefore, they obtain identical representations $h_{G_3} = h_{G_4}$, and correspond thus to a counterexample to the distinguishability of the above two properties. □

We should note here that there exist complex network measures which have served as features for node classification tasks in previous studies and which can capture some of these graph properties [210, 35, 48].

6.4 k-hop Graph Neural Networks

In this section, we propose a generalization of GNNs, so-called k-hop Graph Neural Networks (k-hop GNNs). This new model consists of neighborhood aggregation layers that do not take into account only the direct neighbors of the nodes, but their entire k-hop neighborhood. Hence, instead of the neighborhood aggregation layer shown in Equation 6.1, the proposed model updates the hidden state $h_v^{(t)}$ of a node v as follows:

$$\begin{aligned} a_v^{(t)} &= \text{AGGREGATE}^{(t)}\left(\left\{h_u^{(t-1)} | u \in \mathcal{N}_k(v)\right\}\right) \\ h_v^{(t)} &= \text{MERGE}^{(t)}\left(h_v^{(t-1)}, a_v^{(t)}\right) \end{aligned} \tag{6.4}$$

We next present an instance of the proposed architecture which is strictly stronger than standard GNNs in terms of distinguishing non-isomorphic graphs, and is capable of identifying graph properties which are not captured by the standard GNN architecture.

6.4.1 Proposed Architecture

Let $G = (V, E)$ be a graph. In what follows, we will focus on a single node $v \in V$, and we will present how the representation of this node is updated during the neighborhood aggregation phase. Node v will also be referred as the root of the k-hop neighborhood subgraph G_v^k. For a given iteration/layer t, and a root node v, we define an *inner* representation x_u of each node $u \in \mathcal{N}_k(v)$ and we initialize it as $x_u = h_u^{(t-1)}$. We will next describe how the hidden state $h_v^{(t)}$ of the root v is computed. Let $\text{UPDATE}(w, S)$ denote a module which takes as input a node w and a set of nodes S, and is defined as follows:

$$\text{UPDATE}(w, S) = \text{MLP}\Big(\text{MLP}_1(x_w) + \sum_{u \in S} \text{MLP}_2(x_u)\Big)$$

where $\text{MLP}, \text{MLP}_1, \text{MLP}_2$ are multi-layer perceptrons and x_w, x_u are the *inner* representations of nodes w and u, respectively. The proposed approach uses a series of UPDATE modules to update the representations of the nodes that belong to the k-hop neighborhood of v, following a sequential procedure from the most distant ones to the direct neighbors of v. Although the neural network learns a new vector representation for some of the nodes $u \in \mathcal{N}_k(v)$, these feature vectors are only calculated in the context of updating the root node's representation. Hence, after computing the new representation $h_v^{(t)}$ of v, these representations are not useful any more. These *inner* representations should not be confused with the $h_u^{(t)}$ representation that the network learns for each of these nodes by taking into account their own k-hop neighborhoods.

Let $R_d(v)$ denote the set of nodes at distance (hop count) exactly $d > 0$ from v. Hence, $R_1(v) = \mathcal{N}_1(v)$ is the set of direct neighbors of v, while $R_d(v)$ denotes the ring of nodes at distance d, which we refer to as the nodes at level d. Note that the neighbors of a node $u \in R_d(v)$ belong to one of the next three sets: $R_{d-1}(v)$, $R_d(v)$ or $R_{d+1}(v)$. Specifically, u cannot be connected with nodes at levels $l > d+1$ because then these nodes would belong to level $d+1$ instead of l. Furthermore, u cannot be connected with nodes at levels $l < d-1$ because then u would belong to some level smaller than d. Given a node $u \in \mathcal{N}_k(v)$, the *inner* representation x_u of u is updated at most twice. The two updates aggregate information from the neighbors of u that are located at the immediately higher and at the same level of the neighborhood subgraph, respectively. Hence, the proposed model performs the following two types of updates of *inner* representations: (1) updates across rings of nodes, and (2) updates within a ring of nodes. We next present these two updates in detail:

- Updates ***across*** rings of nodes: Let $u \in R_d(v)$ be a node that belongs to the k-hop neighborhood of v and whose shortest path distance from v is equal to d. Let also $\mathcal{B} = \mathcal{N}_1(u) \cap R_{d+1}(v)$ denote the neighbors of u that belong to level $d+1$ of G_v^k. Note that $\mathcal{B}$ is empty if $k = d$ or if all the neighbors of u belong to levels $d-1$ and d of G_v^k. If $\mathcal{B}$ is not empty, the representation x_u of u is updated as follows:

  $$x_u = \text{UPDATE}^{(t)}_{d,across}\big(u, \mathcal{B}\big)$$

  Otherwise, if u has no neighbors at the next higher level (i.e. $\mathcal{N}_1(u) \cup R_{d+1}(v)$ is empty), then its representation is not updated.

- Updates ***within*** a ring of nodes: If u has one or more neighbors at the same level of G_v^k, and hence $\mathcal{D} = \mathcal{N}_1(u) \cap R_d(v)$ is not empty, the representation x_u of u is re-updated as follows:

  $$x_u = \text{UPDATE}^{(t)}_{d,within}\big(u, \mathcal{D}\big)$$

  Otherwise, if $\mathcal{D} = \varnothing$, its representation is not updated.

The proposed approach starts from the most distant nodes and follows a sequential procedure updating the feature vectors of nodes that are gradually closer to the root. The first type of update (across rings) precedes the second (within a ring). After all its direct neighbors $u \in \mathcal{N}_1(v)$ have been processed, the hidden state of the root node v is computed as follows:

$$h_v^{(t)} = \text{UPDATE}^{(t)}_{0,across}\big(v, \mathcal{N}_1(v)\big)$$

As mentioned above, although the model learns a new *inner* representation x_u for some of the nodes in $\mathcal{N}_k(v)$, these representations are only learned for the purpose of updating the root node's hidden state. Furthermore, it is clear that for a single neighborhood aggregation layer,

Algorithm 1: k-hop GNN

Input: Graph $G = (V, E)$, node features $\{h_v : v \in V\}$, number of neighborhood aggregation layers T, number of hops k

Output: Node features $\{h_v^{(T)} : v \in V\}$

for $t \in \{1, \ldots, T\}$ **do**
  for $v \in V$ **do**
    for $u \in R_k(v)$ **do**
      $\mathcal{D} \longleftarrow \mathcal{N}_1(u) \cap R_k(v)$
      $x_u \longleftarrow \text{UPDATE}^{(t)}_{k,within}(u, \mathcal{D})$
    end for
    for $i \in \{k-1, \ldots, 1\}$ **do**
      for $u \in R_i(v)$ **do**
        $\mathcal{B} \longleftarrow \mathcal{N}_1(u) \cap R_{i+1}(v)$
        $x_u \longleftarrow \text{UPDATE}^{(t)}_{i,across}(u, \mathcal{B})$
        $\mathcal{D} \longleftarrow \mathcal{N}_1(u) \cap R_i(v)$
        $x_u \longleftarrow \text{UPDATE}^{(t)}_{i,within}(u, \mathcal{D})$
      end for
    end for
    $h_v^{(t)} = \text{UPDATE}^{(t)}_{0,across}(v, \mathcal{N}_1(v))$
  end for
end for

the proposed model needs at most $2k$ UPDATE modules. As we will show next, the proposed model can capture the structural information within the root node's k-hop neighborhood even if it comprises of a single neighborhood aggregation layer. Hence, instead of using multiple neighborhood aggregation layers/iterations, it is more suitable to increase the value of k. The various steps of the proposed model are illustrated in Algorithm 1. We provide in the supplementary material a simple example that illustrates the update procedure that was presented above.

After T iterations (i.e. T neighborhood aggregation layers), the emerging node feature vectors $h_v^{(T)}$ can be used in any node-related task. For graph-level tasks, the proposed model can compute a vector representation over the whole graph by applying a READOUT function similar to the one shown in Equation 6.3.

6.4.2 Example

We next provide a simple example that illustrates the update procedure that is presented above. Specifically, Figure 6.3 shows the 2-hop neighborhood graph $G^2_{v_1}$ of a node $v_1 \in V$. As mentioned above, we first consider the most distant nodes (i.e. nodes v_4 and v_5 since $v_4, v_5 \in R_2(v_1)$). The representations of nodes v_4 and v_5 are not updated since these two nodes are at the frontier of $G^2_{v_1}$. Furthermore, there is no edge between the

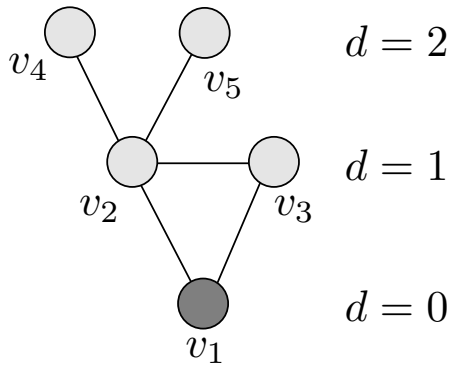


FIGURE 6.3: The 2-hop neighborhood graph $G^2_{v_1}$ of a node v_1 of graph G.

two nodes. However, these two nodes contribute to the update of the representation of node v_2. Specifically, the *inner* representation of node v_2 is updated as follows:

$$x_{v_2} = \text{UPDATE}^{(t)}_{1,across}\big(v_2, \{v_4, v_5\}\big)$$

Then, we update the *inner* representations of nodes whose shortest path distance from the root is 1 and which are connected with other nodes with the same distance from the root. There is one such pair of nodes (i.e. nodes v_2 and v_3) which are updated as:

$$x_{v_2} = \text{UPDATE}^{(t)}_{1,within}\big(v_2, \{v_3\}\big)$$
$$x_{v_3} = \text{UPDATE}^{(t)}_{1,within}\big(v_3, \{v_2\}\big)$$

Finally, we update the root by aggregating information from its direct neighbors:

$$h^{(t)}_{v_1} = \text{UPDATE}^{(t)}_{0,across}\big(v_1, \{v_2, v_3\}\big)$$

6.4.3 Expressive Power

We next study the identifiability of the proposed k-hop GNN. The following Theorem comprises the main results about graph properties that can be identified by the proposed model.

Theorem 11. *For the k-hop GNN, there exists a sequence of modules* $UPDATE^{(0)}_{0,across}$, $UPDATE^{(0)}_{1,within}$, $UPDATE^{(0)}_{1,across}$, ..., $UPDATE^{(T)}_{k-1,across}$, $UPDATE^{(T)}_{k,within}$ *such that*

1. *it can identify triangle-freeness for* $k \geqslant 1$
2. *connectivity for* $k > \delta_{min}$ *where* δ_{min} *is the minimum of the diameters of the connected components*

3. *bipartiteness for* $k \geqslant \frac{l-1}{2}$ *where* l *is the length of the smallest odd cycle in the graph (if any)*

Proof. We assume that the feature vectors of the nodes come from a countable set. This set may correspond to a subset of an uncountable set such as $\mathbb{R}^d$. Furthermore, the feature vectors of a set of nodes form a multiset (i.e. since some nodes may have identical feature vectors).

We will show that if a graph has one of the three considered properties (i.e. triangle-freeness, bipartiteness, and connectivity), some of its nodes can be mapped to different feature vectors compared to the nodes of a graph that does not have the property. Then, by applying an injective readout function, the two graphs can also be mapped to different feature vectors.

For simplicity of presentation, we will assume that the proposed model consists of a single neighborhood aggregation layer. The same results also hold for multiple neighborhood aggregation layers. We first show that the aggregation scheme that our model employs can represent universal functions over the pairs of a node and the multiset of its neighors. The following Lemma generalizes the setting in Xu et al. [238].

Lemma 5. *Assume* $\mathcal{X}$ *is countable, and let* $r \in \mathbb{N}$. *There exist functions* $f : \mathcal{X} \to \mathbb{R}^d$ *and* $f' : \mathcal{X} \to \mathbb{R}^d$, *such that* $h_i(c, X) = f(c) + \sum_{x \in X} f'(x)$ *is unique for each* $i \in \{0, \ldots, r\}$ *and each pair* (c, X), *where* $c \in \mathcal{X}$ *and* $X \subset \mathcal{X}$ *is a finite multiset. Moreover, any function* g_i *over such pairs can be decomposed as* $g_i(c, X) = \phi\big(f(c) + \sum_{x \in X} f'(x)\big)$ *for some function* ϕ.

The proof of Lemma 5 is in Appendix B.4.

In our setting, the UPDATE modules correspond to $g_i(c, X)$ functions. These modules use multi-layer perceptrons (MLPs) to model and learn f, f' and ϕ in the above Lemma, thanks to the universal approximation theorem [101]. Note that given two nodes v, v', if a node $u \in \mathcal{N}_k(v)$ obtains a representation that is never obtained by any node $u' \in \mathcal{N}_k(v')$, then based on the above Lemma, there exist UPDATE modules such that the root nodes v, v' are assigned different representations. Hence, for all three properties, it is sufficient to show that at some point of the algorithm, a node of the graph that satisfies the property can obtain a representation that is never obtained by any node of a graph that does not satisfy the property.

Triangle-freeness If a graph is not triangle-free, then there exist at least three nodes whose 1-hop neighborhoods contain a triangle. Let v be such a node. Then, clearly there are at least two nodes $u \in R_1(v)$ which are connected to each other by an edge. The representations of these nodes are updated as follows: $x_u = \text{UPDATE}^{(0)}_{1,within}\big(u, \mathcal{D}\big)$. On the other hand, no such update takes place in the case of triangle-free graphs since

$\mathcal{D} = \varnothing$. Hence, based on Lemma 5, the above $\text{UPDATE}^{(0)}_{1,within}$ module can generate different representations for the nodes that participate in a triangle from the representations of the nodes of the neighborhood subgraph of each node of a triangle-free graph.

Connectivity Let G be a disconnected graph and C its component which has the minimum diameter δ_{min}. Then, for an arbitrary node v of component C, it holds that $R_i(v) = \varnothing$ for all $i > \delta_{min}$. On the other hand, if the graph is connected, for some node v, it holds that $|R_i(v)| > 0$ for all $i \leqslant \delta$ where $\delta > \delta_{min}$ is the diameter of the connected graph. Hence, the representations of the nodes $u \in R_i(v)$ and $u' \in R_{i-1}(v)$ are updated as $x_u = \text{UPDATE}^{(0)}_{i,within}(u, \mathcal{D})$ and $x_{u'} = \text{UPDATE}^{(0)}_{i-1,across}(u', \mathcal{D}')$, respectively. Based on Lemma 5, the above two UPDATE modules can generate different representations for the nodes of a neighborhood subgraph of a disconnected graph compared to those of the nodes of the neighborhood subgraph of a connected graph.

Bipartiteness It is well-known that a graph is bipartite if and only if it does not contain an odd cycle. If G is bipartite and l is the length of the smallest odd cycle in G, then the k-hop neighborhood subgraphs ($k \geqslant \frac{l-1}{2}$) of more than one nodes contain a cycle of odd length. According to Lemma 6 (below), the k-hop neighborhood subgraph of a node v contains a cycle of odd length if and only if the shortest path lengths from two adjacent nodes $u, w \in \mathcal{N}_k(v)$ to v are identical. In other words, there exist two nodes both at the same level i of the k-hop neighborhood subgraph of node v that are connected to each other with an edge. During the process of updating the representation of the root v, the feature vectors of these nodes are also updated as follows: $x_u = \text{UPDATE}^{(0)}_{i,within}(u, \mathcal{D})$. This update does not take place in the case of a non-bipartite graph since $\mathcal{D} = \varnothing$ for all nodes of all neighborhood subgraphs. Based on Lemma 5, these nodes can obtain different representations from all the representations of the nodes of a neighborhood subgraph extracted from a bipartite graph.

Lemma 6. *Let G^k_v be the k-hop neighborhood subgraph of a node v. Then, G^k_v contains a cycle of odd length if and only if the shortest path lengths from two adjacent nodes $u, w \in \mathcal{N}_k(v)$ to v are identical.*

The proof of Lemma 6 is in Appendix B.5.

□

It should be mentioned that there have already been proposed GNNs that update the representations of the nodes based on their k-hop neighborhoods. Such GNNs employ polynomials of order k [60] or auto-regressive moving average (ARMA) filters [23] to approximate a transfer function that acts on the eigenvalues of the normalized Laplacian matrix,

and can be shown that they utilize information from the nodes' k-hop neighborhoods. However, in our experimental evaluation, these models fail to consistently capture the above three properties. Furthermore, other methods take into account higher-order neighborhoods by computing powers of the adjacency matrix [1].

6.4.4 Computational Complexity

The increase of expressiveness provided by the k-hop GNN model does not come without a price. Clearly the time complexity of the proposed model is higher than that of the standard GNN. The computational steps of the proposed method consist of two parts:

1. **Preprocessing step:** In this phase, the model extracts the neighborhood subgraphs of all nodes. For each node, the neighborhood subgraph can be extracted in linear time in the number of edges of the neighborhood. Hence, the complexity of the preprocessing step is $\mathcal{O}(nm)$ in the worst case (i.e. for a complete graph). For sparse graphs, it can become significantly lower, i.e. $\mathcal{O}(n\,\bar{d}^k)$ where $\bar{d}$ is the average degree of the nodes. We should note that this step is computed only once.
2. **Message passing procedure:** To compute the representation of each node, for each edge of its neighborhood subgraph, a message needs to be sent from some node to another node. Therefore, the complexity of one message passing iteration (i.e. one epoch in our implementation) is $\mathcal{O}(nm)$ in the worst case and $\mathcal{O}(n\,\bar{d}^k)$ for sparse graphs. Note however, that the proposed message passing layer allows a GPU-friendly implementation. Therefore, in practice, as verified by our experiments, the running time is not prohibitive.

To sum up, the total computational complexity of k-hop Graph Neural Network is $\mathcal{O}(nm)$, while for sparse graphs is $\mathcal{O}(n\,\bar{d}^k)$. On the other hand, the complexity of the standard GNN model is $\mathcal{O}(km)$ where k is the number of message passing iterations. For sparse graphs, and when the number of hops k is set to a small value, the complexity of the proposed model is comparable to the complexity of the standard GNNs.

6.5 Experimental Evaluation

In this Section, we evaluate the performance of the proposed k-hop GNN in two tasks: (1) node classification, (2) graph classification.

6.5.1 Node Classification

The main objective of node classification is to assign class labels to unlabeled nodes. We evaluate the proposed model on synthetic graphs with planted structural equivalences. Each node $v_i \in V$ has an associated

class label y_i and the goal is to learn a representation vector $h_{v_i}^{(T)}$ of v_i such that v_i's label can be predicted as $y_i = f(h_{v_i}^{(T)})$.

Synthetic Datasets

Datasets. To generate the graphs, we follow the same procedure as in [63]. Structurally equivalent nodes are assigned the same class labels.

All the generated graphs consist of a cycle of length 40 and some basic shapes ("house", "fan", "star") which are regularly placed along the cycle. In the "basic" setup, 10 instances of only one of the three types (randomly chosen with uniform probability) are placed along the cycle. In the "varied" setup, 10 instances of each one of the three shapes are randomly placed along the cycle. The use of multiple shapes increases the number and complexity of the structural role patterns, posing a challenge to the learning algorithms. To assess how the algorithms perform in noisy scenarios, we introduce two additional configurations ("basic perturbed" and "varied perturbed") where we add edges uniformly at random on the generated graphs. The number of edges that are added is equal to 10% of the edges of the graph. The shapes that are placed along the cycle graph in the different setups are illustrated in Table 6.1.

Baselines. We compare the proposed model against 3 recent state-of-the-art techniques for learning structural node representations: (1) RolX [95], (2) struc2vec [186], and (3) GraphWave [63]. Note that these 3 algorithms are unsupervised, in the sense that they only take a graph as input and not any class labels of the nodes. We also compare the k-hop GNN model against the standard GNN architecture as described in Equation 6.1, and the ChebNet [60] and ARMA [23] models which also aggregate information from the nodes' k-hop neighborhoods. The ChebNet and ARMA models were implemented using the Pytorch Geometric library [70].

Experimental setup. For each configuration, we generate 20 graphs using the procedure described above. For each graph, we perform 10-fold cross validation. We repeat the whole process 25 times. We measure the performance of the different algorithms using the following two evaluation metrics: (1) average accuracy and (2) average F1-score.

For the unsupervised algorithms, we learn an embedding for each node, and we predict the class label of each node in the test set using a 4-nearest neighbors classifier. For the proposed k-hop GNN model, the standard GNN model, ChebNet and ARMA, we train the models on the training set of each fold and use the models to classify the nodes of the test set.

For all the unsupervised algorithms, we use the default parameter values. Specifically, for struc2vec, we set the probability that the random walks stays in current layer to 0.3, the dimensionality of the embeddings to 128, the number of epochs to 5, the number of walks per node to 10, the walk length to 80 and the context window size to 10. Furthermore, we make

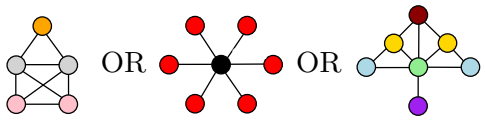
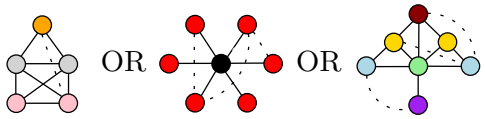
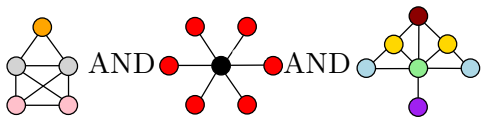
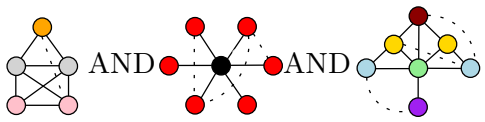
Configuration	Shapes placed along a cycle graph
basic	
basic perturbed	
varied	
varied perturbed	

TABLE 6.1: Example of synthetically generated structures for each configuration. The different colors denote structurally equivalent nodes. Dashed lines denote perturbed graphs (obtained by randomly adding edges).

use of all approximations OPT1, OPT2, and OPT3. For GraphWave, we use the multiscale version, set $d = 50$ and use evenly spaced sampling points t_i in range $[0, 100]$. Finally, for RolX, we did not use any approach for automatically detecting the number of different roles, but we directly provided the algorithm with the correct number of roles. As mentioned above, the representations learned by the unsupervised algorithms are fed into an MLP. We tune the number of hidden units and the dropout rate of the MLP. Specifically, we tune the number of hidden units from $\{8, 16, 32\}$ and the dropout rate from $\{0.0, 0.2\}$. We use 2 and 3 neighborhood aggregation layers for the standard GNN, and 1 layer for ChebNet, ARMA and the proposed 2-hop and 3-hop GNNs. For ChebNet, we use polynomials of order 2 and 3, and for ARMA, we set the number of stacks K to 2 and the depth T to 2 and 3. The neighborhood aggregation layers of the k-hop GNN models and of the standard GNN models consist of MLPs with 2 layers. Batch normalization is applied to the output of every neighborhood aggregation layer. The hidden-dimension size of the MLPs is chosen from $\{8, 16, 32\}$ and the dropout rate from $\{0.0, 0.2\}$. To train all neural networks, we use the Adam optimizer with initial

	Configuration							
	basic		basic perturbed		varied		varied perturbed	
	Accuracy	F1-score	Accuracy	F1-score	Accuracy	F1-score	Accuracy	F1-score
RolX	**1.000**	**1.000**	0.928	0.886	0.998	0.996	0.856	0.768
struc2vec	0.784	0.708	0.703	0.632	0.738	0.592	0.573	0.412
GraphWave	0.995	0.993	0.906	0.861	0.982	0.965	0.793	0.682
2-GNN	0.997	0.994	0.920	0.876	0.990	0.979	0.852	0.753
3-GNN	0.997	0.994	0.911	0.859	0.993	0.985	0.866	0.775
ChebNet (K=2)	0.988	0.979	0.866	0.787	0.852	0.732	0.624	0.471
ChebNet (K=3)	0.992	0.987	0.904	0.850	0.958	0.917	0.758	0.612
ARMA (T=2)	0.996	0.992	0.914	0.861	0.982	0.961	0.839	0.728
ARMA (T=3)	0.997	0.996	0.919	0.872	0.993	0.987	0.850	0.747
2-hop GNN	**1.000**	**1.000**	0.961	**0.934**	**0.999**	**0.999**	0.948	0.910
3-hop GNN	**1.000**	**1.000**	**0.962**	**0.934**	0.996	0.993	**0.952**	**0.916**

TABLE 6.2: Performance of the baselines and the proposed k-hop GNN models for learning structural embeddings averaged over 20 synthetically generated graphs for each configuration.

learning rate 0.01 and decay the learning rate by 0.5 every 50 epochs. We set the number of epochs to 200. We store the model that achieved the best validation accuracy into disk. At the end of training, the model is retrieved from the disk, and we use it to classify the test instances.

Results. Table 6.2 shows that the instances of the proposed k-hop GNN architecture outperform all the baselines in the node classification task. Interestingly, struc2vec is the worst-performing method in all configurations. Specifically, in the least challenging configuration ("basic"), the k-hop GNN models, and RolX perform the best. All these methods yield perfect performance, while the standard GNN models, the ChebNet and ARMA models, and GraphWave exhibit slightly worse performance. In the presence of noise ("basic perturbed" configuration), the performance of all methods degrades a lot. The 3-hop GNN model is the best-performing method followed by the 2-hop GNN model, RolX, and the 2-GNN model, in that order. The remaining models achieve slightly lower accuracies and F1-scores than the above 4 methods. In the "varied" configuration, the k-hop GNN models are once again the best-performing methods along with RolX. In the "varied perturbed" configuration, the 3-hop GNN model yields the best performance. The 2-hop GNN model achieves slightly worse performance, while the performance of the remaining methods is much lower. From the two noisy configurations, it is clear that the baseline methods are more prone to noise compared to the k-hop GNN model. Furthermore, with regards to the two instances of the k-hop GNN model, the 3-hop GNN model outperforms the 2-hop GNN model in two configurations (i.e. "basic perturbed" and "varied perturbed"), is outperformed by the 2-hop GNN model in one confuguration (i.e. "varied"), while the two models achieve the same performance in the "basic" configuration. Finally, the proposed k-hop GNN models outperform the two standard GNN models in all experiments, thereby validating our theoretical results. Overall, the

proposed k-hop GNN model is robust and achieves good performance, demonstrating that it can learn high-quality node representations.

Real-World Dataset

Datasets. We study the Enron dataset, an e-mail network encoding communication between employees in a company. We expect structural equivalences in job titles due to corporate organizational hierarchy. Nodes of the network represent Enron employees and edges correspond to e-mail communication between the employees. There are 143 nodes and 2,583 edges in the emerging network. An employee has one of seven functions in the company (e.g. CEO, manager, etc.). These functions provide ground-truth information about roles of the corresponding nodes in the network.

Baselines. We compare the proposed model against the 9 baseline algorithms which were presented above: (1) RolX, (2) struc2vec, (3) GraphWave, (4) 2-GNN, (5) 3-GNN, (6) ChebNet ($K = 2$), (7) ChebNet ($K = 3$), (8) ARMA ($T = 2$), and (9) ARMA ($T = 3$).

Experimental setup. We perform 10-fold cross validation, and repeat the whole process 20 times. For each algorithm, we report (1) its average accuracy, and (2) its average F1-score.

For the unsupervised algorithms, we first learn an embedding for each node, and then, for each fold, we use a 4-nearest neighbors classifier to predict the job titles of the nodes of the test set. For the proposed k-hop GNN model, the standard GNN model, ChebNet and ARMA, we train the models on the training set of each fold and use the models to classify the nodes of the test set.

For all algorithms, we set/optimize their hyperparameters as described in subsection 6.5.1 above.

Results. We can see in Table 6.3 that the supervised neural network models outperformed the unsupervised algorithms on this dataset. From the unsupervised algorithms, only struc2vec achieved performance comparable to that of the supervised models. In terms of accuracy, ARMA is the best-performing method. Both ARMA ($T = 2$) and ARMA ($T = 3$) outperform all the other methods on the Enron dataset. On the other hand, in terms of F1-score, 2-hop GNN achieves the best performance among the different methods, followed by 3-GNN and ARMA ($T = 2$). Surprisingly, the 3-hop GNN model performs much worse than the 2-hop GNN model. We hypothesize that this is related to the structure of the e-mail network.

	Accuracy	F1-score
RolX	0.264	0.154
struc2vec	0.323	0.190
GraphWave	0.257	0.149
2-GNN	0.357	0.183
3-GNN	0.366	0.195
ChebNet (K=2)	0.342	0.179
ChebNet (K=3)	0.360	0.191
ARMA (T=2)	0.374	0.192
ARMA (T=3)	**0.376**	0.190
2-hop GNN	0.366	**0.198**
3-hop GNN	0.327	0.171

TABLE 6.3: Performance of the baselines and the proposed k-hop GNN models for learning structural embeddings on the Enron dataset.

6.5.2 Graph Classification

We next apply the proposed model to the problem of graph classification., i.e. the supervised learning task of assigning a graph to a set of predefined categories. Specifically, given a set of graphs $\{G_1, \ldots, G_N\} \subseteq \mathcal{G}$ and their class labels $\{y_1, \ldots, y_N\}$, the goal is to learn a representation vector h_{G_i} such that the class label of every graph of the test set can be predicted as $y_i = f(h_{G_i})$. For this task, we are going to evaluate the performance of the proposed model in two different types of datasets: (1) synthetic datasets containing graphs that satisfy or do not satisfy the considered graph properties, and (2) standard widely-used datasets from real-world scenarios.

Synthetic Datasets

Datasets. In order to investigate if the proposed model can distinguish triangle-freeness, bipartiteness and connectivity, we created three synthetic datasets. Each one consists of 800 4-regular graphs of 60 nodes each and is assigned a class label which denotes whether it satisfies the corresponding property or not (i.e. binary classification task). All the nodes are assigned identical labels. Furthermore, all three datasets are balanced, i.e. half of the graphs (400 graphs) satisfy the examined graph property, while the rest of the graphs (400 graphs) do not satisfy it. Further details about the synthetic datasets are presented in the supplementary material.

Baselines. We compare our model against the standard GNN architecture of Equations 6.1 and 6.3, and against the ChebNet [60] and ARMA [23] models. Based on our theoretical results, we expect the standard GNN model to perform worse than the proposed model on these 3

	Connectivity	Bipartiteness	Triangle freeness
2-GNN	55.00 ± 5.30	53.78 ± 2.61	51.87 ± 7.43
3-GNN	56.20 ± 2.28	58.13 ± 2.10	55.90 ± 4.44
ChebNet (K=2)	56.37 ± 7.76	50.33 ± 1.20	53.12 ± 6.35
Chebnet (K=3)	57.62 ± 3.84	51.98 ± 3.56	54.75 ± 7.14
ARMA (T=2)	55.55 ± 5.59	54.50 ± 4.61	53.00 ± 3.18
ARMA (T=3)	55.63 ± 5.69	53.92 ± 3.22	54.25 ± 5.80
2-hop GNN	81.24 ± 5.22	84.69 ± 1.74	**84.06** ± 2.12
3-hop GNN	**94.77** ± 3.41	**91.12** ± 2.76	82.53 ± 5.33

TABLE 6.4: Average classification accuracy of the proposed k-hop GNN models and the baselines on the 3 synthetic datasets.

datasets. On the other hand, since the ChebNet and ARMA models aggregate information based on the k-hop neighborhood of each node, these models may be able to capture these properties.

Results. We report in Table 6.4 average prediction accuracies across the 10 folds. It is clear that the standard GNN architectures are unable to distinguish the 3 graph properties. Specifically, they all achieve an average accuracy slightly greater than 50% on all three datasets.The ChebNet and ARMA models, even though they aggregate information from each node's k-hop neighborhood, they are also unable to distinguish the 3 properties. On the other hand, the proposed k-hop GNN architectures achieved much higher average accuracies, indicating that the proposed architecture can distinguish the 3 properties in regular graphs. In the case of bipartiteness and connectivity, the 3-hop GNN model achieved very high accuracy. The performance of the 2-hop GNN model was slightly worse than that of the 3-hop GNN model. However, the former managed to better distinguish triangle-free graphs than the latter. Overall, we can conclude that in contrast to the GNN, ChebNet and ARMA architectures, the proposed model leads to more expressive node representations.

Real-World Datasets

We also evaluate the proposed k-hop GNN model on standard graph classification datasets derived from bio/chemoinformatics, and from social networks.

Datasets. We use the following 3 datasets from bioinformatics and chemoinformatics: (1) MUTAG, (2) PROTEINS, (3) NCI1. We also use the following 2 social interaction datasets: (1) IMDB-BINARY, (2)

IMDB-MULTI. In Appendix A.1, we provide details about the examined datasets [1].

Baselines. We compare our methods against four graph kernels: (1) the graphlet kernel (GK) [204], (2) the shortest-path kernel (SP) [27], (3) the Weisfeiler-Lehman subtree kernel (WL) [205], and (4) the Weisfeiler-Lehman Optimal Assignment kernel (WL-OA) [122]. The first three kernels are available in the GraKeL library [209], while for WL-OA we used the code provided by the authors. Besides graph kernels, we also compare the proposed model against the basic GNN architecture of Equations 6.1 and 6.3, against GS-SVM [75] which makes use of geometric scattering features, and against the following state-of-the-art deep learning architectures: (1) ChebNet [60], (2) ARMA [23], (3) PatchySan [165], (4) Deep Graph CNN (DGCNN) [249], (5) CapsGNN [236], and (6) 1-2-3-GNN [161]. For ChebNet and ARMA, we use the implementations contained in the PyTorch Geometric library [70]. For GS-SVM and the rest of the deep learning methods, we compare against the accuracies reported in the original papers.

Experimental Setup. We performed 10-fold cross-validation where 10% of the graphs of each training fold was used as a validation set. The whole process was repeated 10 times for each dataset and each approach.

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 $h = \{4, 5, 6, 7\}$, while the graphlet kernel that we implemented samples 500 graphlets of size up to 6 from each graph. For the proposed k-hop GNN models (2-hop GNN and 3-hop GNN), we used a single neighborhood aggregation layer, while for the standard GNN, we used 2 and 3 layers. The parameters of the neighborhood aggregation layers correspond to MLPs with 2 layers. Batch normalization is applied to the output of every neighborhood aggregation layer. The hidden-dimension size of the MLPs was chosen from $\{16, 32, 64\}$. To generate graph representations, we employed a readout function that sums the vector representations of the nodes. The generated graph representations are then fed into a two layer MLP, with a softmax output. We used the `ReLU` activation function, and we chose the batch size from $\{32, 64, 128\}$. We used the Adam optimizer with an initial learning rate of 10^{-3} and decay the learning rate by 0.5 every 50 epochs.

For ChebNet, ARMA and the proposed k-hop GNN models (2-hop GNN and 3-hop GNN), we used a single neighborhood aggregation layer, while for the standard GNN, we used 2 and 3 layers. The hidden-dimension size of these layers was chosen from $\{16, 32, 64\}$. For ChebNet, we used polynomials of order 2 and 3, and for ARMA, we set the number of stacks K to 2 and the depth T to 2 and 3. To generate graph

[1] The datasets, further references and statistics are available at `https://ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets`

	MUTAG	PROTEINS	NCI1	IMDB BINARY	IMDB MULTI	Average Rank
GK	69.97 (± 2.22)	71.23 (± 0.38)	65.47 (± 0.14)	60.33 (± 0.25)	36.53 (± 0.93)	16.8
SP	84.03 (± 1.49)	75.36 (± 0.61)	72.85 (± 0.24)	60.21 (± 0.58)	39.62 (± 0.57)	12.8
WL	83.63 (± 1.57)	73.12 (± 0.52)	84.42 (± 0.25)	73.36 (± 0.38)	**51.06** (± 0.47)	6.8
WL-OA	86.63 (± 1.49)	75.35 (± 0.45)	**85.74** (± 0.37)	73.61 (± 0.60)	50.48 (± 0.33)	3.0
GS-SVM	83.57 (± 6.75)	74.11 (± 4.02)	79.14 (± 1.28)	71.20 (± 3.25)	48.73 (± 2.32)	10.8
2-GNN	85.92 (± 2.19)	75.24 (± 0.45)	76.32 (± 0.41)	71.40 (± 0.74)	47.73 (± 0.86)	8.8
3-GNN	85.74 (± 1.48)	74.59 (± 0.71)	79.62 (± 0.45)	71.60 (± 0.84)	47.33 (± 1.01)	9.4
ChebyNet (K=2)	85.33 (± 1.42)	74.72 (± 0.97)	78.97 (± 0.35)	71.08 (± 0.51)	47.08 (± 0.60)	10.6
ChebyNet (K=3)	82.49 (± 1.52)	74.81 (± 0.82)	81.01 (± 0.39)	70.90 (± 0.73)	46.66 (± 0.59)	10.8
ARMA (T=2)	82.98 (± 1.90)	74.84 (± 0.59)	80.83 (± 0.42)	70.62 (± 0.95)	46.10 (± 0.82)	11.2
ARMA (T=3)	81.52 (± 1.22)	74.74 (± 0.67)	81.34 (± 0.38)	70.52 (± 0.71)	46.12 (± 0.98)	11.6
PatchySan ($k = 10$)	**88.95** (± 4.37)	75.00 (± 2.51)	76.34 (± 1.68)	71.00 (± 2.29)	45.23 (± 2.84)	9.6
DGCNN	85.83 (± 1.66)	75.54 (± 0.94)	74.44 (± 0.47)	70.03 (± 0.86)	47.83 (± 0.85)	9.6
CapsGNN	86.67 (± 6.88)	**76.28** (± 3.63)	78.35 (± 1.55)	73.10 (± 4.83)	50.27 (± 2.65)	5.0
1-2-3-GNN	86.1	75.5	76.2	**74.2**	49.5	6.0
2-hop GNN	87.93 (± 1.22)	75.03 (± 0.42)	79.31 (± 0.57)	73.33 (± 0.30)	49.79 (± 0.25)	5.4
3-hop GNN	87.56 (± 0.72)	75.28 (± 0.36)	80.61 (± 0.34)	-	-	4.8

TABLE 6.5: Average classification accuracy (± standard deviation) of the baselines and the proposed k-hop GNN models on the 5 graph classification benchmark datasets. The "Average Rank" column illustrates the average rank of each method. The lower the average rank, the better the overall performance of the method.

representations, we employed a readout function that sums the vector representations of the nodes. The generated graph representations are then fed into a two layer MLP, with a softmax output. We used the `ReLU` activation function, and we chose the batch size from $\{32, 64, 128\}$. We used the Adam optimizer with a learning rate of 10^{-2}, while we set the number of epochs to 100. We set the number of epochs to 500, and we select the epoch with the best validation accuracy.

Results. We report in Table 6.5 average prediction accuracies and standard deviations across the 10 repetitions. Note that the graphs contained in the IMDB-BINARY and IMDB-MULTI datasets correspond to the ego-networks of actors/actresses. The diameter of these graphs is at most equal to 2, and therefore, the 3-hop neighborhoods of the nodes are identical to their 2-hop neighborhoods. This is why we do not report the performance of the 3-hop GNN on these datasets.

In general, we observe that the variants of the proposed model achieve high levels of performance. Specifically, they achieve the second best performance on MUTAG, the fourth best performance on IMDB-BINARY and IMDB-MULTI, and the sixth best performance on the NCI1 and PROTEINS datasets. On most datasets, the proposed model yields only slightly worse accuracies compared to the best performing method, the WL-OA kernel. Interestingly, the two k-hop GNN models perform equally well in general. More specifically, the 3-hop GNN achieves slightly better accuracy than the 2-hop GNN on most datasets. However, the difference in performance is not very large. Furthermore, it should be mentioned that the proposed k-hop GNN models outperform the two

	MUTAG	PROTEINS	NCI1	IMDB BINARY	IMDB MULTI
2-GNN	0.01	0.08	0.29	0.07	0.10
3-GNN	0.02	0.12	0.38	0.09	0.13
2-hop GNN	0.03	0.18	0.59	0.18	0.22
3-hop GNN	0.04	0.27	0.82	0.18	0.22

TABLE 6.6: Average running time per epoch (in seconds) of the proposed k-hop GNN models and the standard GNN models on the 5 graph classification benchmark datasets.

	MUTAG	PROTEINS	NCI1	IMDB BINARY	IMDB MULTI
2-GNN	2.72	3.36	4.78	3.20	3.35
3-GNN	2.72	3.38	4.79	3.21	3.33
2-hop GNN	3.11	13.02	19.63	31.38	28.36
3-hop GNN	3.28	19.00	27.50	31.37	28.37

TABLE 6.7: Preprocessing time (in seconds) of the proposed k-hop GNN models and the standard GNN models on the 5 graph classification benchmark datasets.

standard GNN models on all datasets, demonstrating their superiority. Overall, the proposed architecture yields good performance, demonstrating that it can learn not only high-quality node representations, but also graph representations.

Runtime Analysis. We also compare the running time of the proposed model against that of the standard GNN model on the five real-world datasets. We report in Table 6.6 the average time per epoch, and in Table 6.7 the preprocessing time (both in seconds). The hyperparameters of all models are set to the same values. The obtained results are given below.

As expected, the standard GNN is faster than the proposed model both in terms of time per epoch and in terms of the preprocessing time. However, the running time of the proposed model is by no means prohibitive. In general, the average time per epoch of the proposed models is twice as high as that of the corresponding standard GNN models. Furthermore, we should note that the preprocessing time of the standard GNN is independent of the value of k, while for the proposed model, it increases as k increases (since larger neighborhoods need to be processed).

6.6 Conclusion

In this chapter, we analyzed the expressive power of GNNs with respect to their receptive field, showing that a wide class of GNN architectures

cannot identify fundamental properties of graphs, due to the limited information obtained from the 1-hop neighborhoods. We also proposed the k-hop GNN model which aggregates information from the nodes' k-hop neighborhoods, and is capable of identifying graph properties that are not captured by standard GNNs. We evaluated the proposed model on node and graph classification datasets, where it achieved results competitive with state-of-the-art algorithms.

Part III

Beyond local interactions

Chapter 7

Lipschitz Continuity of Graph Attention

7.1 Introduction

So far, we have studied approaches that can improve the discrimination power or better exploit the receptive field of GNNs. However, a very crucial success parameter of a neural network model is its ability to scale efficiently into larger depths, maintaining its predicting capabilities. In the case of graph neural networks (GNNs) [92], the model depth is directly related to the neighborhood size on which the model aggregates information. In such a case (visualized in Figure 7.1), shallow neural networks are fundamentally unable to capture long-range characteristics. However, many instances of real-world networks contain information that is shared through distant nodes (e.g., structural patterns [152] or structural noise presence). Thus, designing deep graph neural networks is a subject of extensive research [136, 133, 132, 144]. Recent studies [252, 136, 3] showed that deeper GNNs fail to their prediction tasks, while attention-based GNNs show a better behavior with respect to larger depths.

Over the last few years, attention models became extremely popular in a wide variety of deep learning applications. These architectures made their first appearance in natural language processing and neural machine translation [12, 76, 219], and gradually became state-of-the-art in multiple machine learning tasks, including sequential data learning [183, 146, 241], graph classification [220, 139] and computer vision [237]. Notably, Vaswani et al. [219] showed that efficient deep learning models could be created using attention layers only, leading to the *Transformer* architecture. Compared to convolutional or linear layers, attention layers have the advantage of allowing the selection of key features in the data while being amenable to backpropagation and gradient descent schemes.

Unfortunately, attention models tend to suffer from poor performance when their depth increases, and most applications have a relatively small number of layers (e.g. 6 for the Transformers in [219]). While depth is

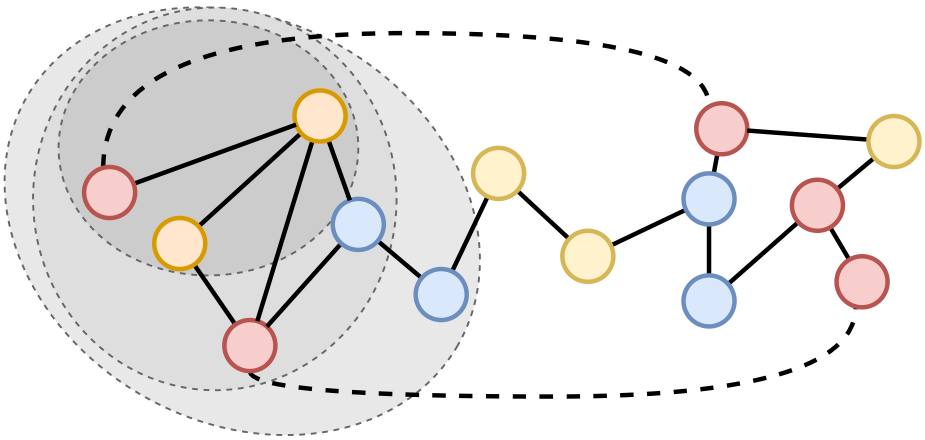


FIGURE 7.1: Long-range dependencies can occur in graphs. GNNs with local aggregation schemes need to be deep enough to capture such interaction. The grey leveled neighborhoods show the three consecutive layers of a GNN.

not necessarily synonymous with increased performance, deep architectures showed extremely good performance in many difficult tasks (e.g. image classification) that exhibit complex structural information [212]. Although there are cases of NLP models, such as GPT-3 [32], that can scale to very deep architectures (up to 96 layers), for graph attention models state-of-the-art architectures remain shallow and building deep architectures remains an open problem.

In this chapter, we show that enforcing Lipschitz continuity by normalizing the attention scores can significantly improve the performance of deep attention models. To do so, we present *LipschitzNorm*, a normalization scheme for self-attention layers that enforces Lipschitz continuity, and apply this normalization to attention-based GNNs, including graph attention networks (GAT) [220], and graph transformers (GT) [246, 206]. Moreover, we show that, without normalization, gradient explosion appears in these architectures due to a lack of Lipschitz continuity of the original attention mechanism [115]. Finally, we show that such a normalization allows to build deeper graph neural networks that show good performance for node label prediction tasks that exhibit long-range dependencies.

Gradient Explosion and Deeper GNNs As we can see from Figure 7.1, the long-range interactions create the need for deeper GNNs and as we will see in the next sections such a design requires a smooth gradient flow. However, as previous research showed, there are other problems

that can occur in long-range dependencies, such as: the *oversmoothing* phenomenon, a result of the multiple applications of Laplacian operators [252, 136] and the *oversquashing*, that happens due to the exponential increase of the neighborhood information creating a huge hypothesis space and thus a bottleneck in the node's representation [3]. In this chapter, we focus on how to mitigate the gradient explosion issue, without taking into account whether oversmoothing or oversquashing occur. However, we show empirically, that handling the gradient flow, the models can behave well in situations, where such issues appear, showing a possible interchanging relation between the different issues on deeper GNNs.

The remainder of this chapter is structured as follows: in Section 7.2, we provide an overview of the related work on attention mechanisms and graph learning models. Then, in Section 7.3, we provide precise definitions for Lipschitz continuity and attention models. In Section 7.4, we present our theoretical analysis and in Section 7.5 we introduce our normalization layer, called *LipschitzNorm*. Then, we empirically show in Section 7.6 the connection between Lipschitz continuity and gradient explosion during training. These are followed by the experimental evaluation in Section 7.7.

7.2 Related Work

Initially designed to extend the capabilities of recurrent neural networks [12], attention models rapidly became a highly efficient and versatile model for machine learning tasks in natural language processing [183, 241], computer vision [237] and recommender systems [243]. Recently, novel attention models have been introduced in graph-based systems showing state-of-the-art performance on graph classification [128], node classification [220, 206] and link prediction [253] tasks.

Attention and Lipschitz Continuity: Although the attention models gain more attraction, little progress has been made in the theoretical study of the attention. Pérez, Marinković, and Barceló [180] showed how attention-based models can be Turing complete and Cordonnier, Loukas, and Jaggi [45] studied the relationship of self-attention layers and the convolutional networks for image processing. One important direction that can help towards the expressivity of attention models is the analysis of Lipschitz continuity. Even though the computation of tight Lipschitz bounds of neural networks has been proven to be a hard task [222], a few approaches suggested Lipschitz-based normalization methods for neural networks [157, 84]. Kim, Papamakarios, and Mnih [115] showed that the standard dot-product self-attention is not Lipschitz continuous, proposing an alternative attention layer that satisfies the Lipschitz continuity. The latter work assumes that the input and output dimensions of the transformer are equal. Such an assumption is only

applicable to Transformer-based models, while for example in graph attention it does not hold, since the inputs are taken neighbor-wise and, thus, with variable length.

Attention and Graph Neural Networks: In this chapter, we study Lipschitz properties of the general form of self-attention from the optimization perspective. We propose a normalization that enforces the attention layer to be Lipschitz and prevents the model from gradient explosion phenomena. Graph Neural Networks (GNNs) is a class of models that suffer from gradient explosion and vanishing as the model depth increases and, thus enforcing the Lipschitz continuity of deep attention-based GNNs can enhance their expressivity. Due to the recent success of GNNs in various real-world applications, there is a growing interest in their expressive power, either investigating how GNNs can be universal approximators [238, 56, 149] or studying the impact of the depth and width of the models [136, 144]. The second aspect of the depth analysis still has a few unanswered questions, as the majority of the current state-of-the-art models employ shallow GNNs.

Depth in GNNs: Zhao and Akoglu [252] related the expressivity of graph convolutional networks with the *laplacian oversmoothing* effect and proposed a normalization layer as a way to alleviate it. More recently, Rong et al. [187] proposed an edge dropping framework on node classification tasks, in order to tackle over-fitting and over-smoothing phenomena and have shown empirically a constant improvement on the original datasets. Li et al. [133] and Li et al. [132] introduced frameworks of adaptive residual connections and generalized message-passing aggregators that allow for the training of very deep GCNs. Finally, Loukas [144] studied the effect of the depth and the width of a graph neural network model and Alon and Yahav [3] introduced the *oversquashing* phenomenon as a deterioration factor to the performance of the GNNs. However, to our knowledge, no study on the explicit relationship between the gradient explosion and the GNNs has yet been made.

7.3 Notations and Definitions

In this section, we recall the definitions of attention models as well as Lipschitz continuity. This notion will be central in our analysis and help us understand why gradient explosion appears when training attention models (see Section 7.6).

7.3.1 Basic Notations

For any matrix $M \in \mathbb{R}^{n\times m}$, we will denote as spectral norm $\|M\|_*$ its largest singular value, $(\infty,2)$-norm $\|M\|_{(\infty,2)} = \max_i(\sum_j M_{ij}^2)^{1/2}$, and

Frobenius norm $\|M\|_F = (\sum_{i,j} M_{ij}^2)^{1/2}$. Moreover, for $\mathbb{X}$ (resp. $\mathbb{Y}$) a vector space equipped with the norm $\|\cdot\|_{\mathbb{X}}$ (resp. $\|\cdot\|_{\mathbb{Y}}$), the operator norm of a linear operator $f : \mathbb{X} \to \mathbb{Y}$ will denote the quantity $|||f|||_{\mathbb{X},\mathbb{Y}} = \max_{x \in X} \|f(x)\|_{\mathbb{Y}} / \|x\|_{\mathbb{X}}$ and $|||f|||_{\mathbb{X}} = |||f|||_{\mathbb{X},\mathbb{X}}$. Finally, the (Fréchet) derivative of a function $f : \mathbb{X} \to \mathbb{Y}$ at $x \in \mathbb{X}$ will denote (when such a function exists) the linear function $\mathbf{D}f_x : \mathbb{X} \to \mathbb{Y}$ such that, $\forall h \in \mathbb{X}$, $f(x+h) - f(x) = \mathbf{D}f_x(h) + o(\|h\|)$.

7.3.2 Lipschitz Continuity

A function $f : \mathbb{X} \to \mathbb{Y}$ is said to be Lipschitz continuous if there exists a constant L such that, for any $x, y \in \mathbb{X}$, $\|f(x) - f(y)\|_{\mathbb{X}} \leqslant L\|x - y\|_{\mathbb{Y}}$. The Lipschitz constant $L_{\mathbb{X},\mathbb{Y}}(f)$ will denote the smallest of such constants. Moreover, a Lipschitz continuous function f is derivable almost everywhere and (see Federer [69, Thm 3.1.6])

$$L_{\mathbb{X},\mathbb{Y}}(f) = \sup_{X \in \mathbb{X}} |||\mathbf{D}f_X|||_{\mathbb{X},\mathbb{Y}} \,. \tag{7.1}$$

The Lipschitz constant controls the perturbation of the output given a bounded input perturbation, and is a direct extension of the gradient norm to the multi-dimensional case. Indeed, when f is scalar-valued and differentiable, we have $\mathbf{D}f_x(h) = \nabla f(x)^\top h$ and $|||\mathbf{D}f_x|||_F = \|\nabla f(x)\|_2$. In our analysis, we will only consider the Lipschitz constant of attention layers for the Frobenius norm (i.e. the L_2-norm of the *flattened* input and output matrices), and derive upper bounds from the previous formula (see Section 7.4).

7.3.3 Attention Models

An *attention layer* is a soft selection procedure that uses scores to choose which input vectors to focus on. Before presenting attention layers in their most general form, we first focus on the more simple case with a single vector output in order to provide more intuition to the reader.

Single Output Case: Let $x_1, \ldots, x_n \in \mathbb{R}^d$ be a set of input vectors, and $g : \mathbb{R}^d \to \mathbb{R}$ a score function. Each vector is assigned a score $g(x_i)$ that measures the impact of the input vector on the output through a *softmax* function:

$$\mathrm{Att}(x) = \sum_{i=1}^{n} \frac{e^{g(x_i)}}{\sum_{j=1}^{n} e^{g(x_j)}} x_i \,. \tag{7.2}$$

In most applications, the score function is linear $g(x) = q^\top x$ where $q \in \mathbb{R}^d$ is a query vector that indicates the direction favored by the attention model.

General Case: In many applications, the output is not a single vector, but a collection of vectors. We thus switch to a matrix notation in order to simplify the definitions. Let $X \in \mathbb{R}^{d\times n}$ be an input matrix whose rows are the input vectors $x_1, \dots, x_n \in \mathbb{R}^d$. A score function $g : \mathbb{R}^{d\times n} \to \mathbb{R}^{m\times n}$ takes the input matrix and returns scores for each output vector $i \in \{1, \dots, m\}$ and each input vector $j \in \{1, \dots, n\}$. This score is usually linear or quadratic ; however, we will see in Section 7.4 that such a generalisation allows to consider more advanced score functions, including overall normalization by a scalar. The probability weights are then computed using a (row-wise) softmax operator softmax : $\mathbb{R}^{m\times n} \to \mathbb{R}^{n\times m}$ taking as input a score matrix $M \in \mathbb{R}^{m\times n}$,

$$\text{softmax}(M)_{ij} = \frac{e^{M_{ij}}}{\sum_{k=1}^{n} e^{M_{ik}}}. \tag{7.3}$$

Note that all rows sum to one, and all coordinates are between 0 and 1. Each row can thus be interpreted as a probability distribution over the n input vectors. Finally, the overall attention module Att : $\mathbb{R}^{d\times n} \to \mathbb{R}^{d\times m}$ returns a matrix whose columns are weighted averages of the inputs:

$$\text{Att}(X) = X\,\text{softmax}(g(X))^\top. \tag{7.4}$$

Multi-Head Attention: In order to augment the power of attention models, a common trick consists in concatenating multiple independent attention models. These *multi-head* models can thus focus on multiple directions of the input space at the same time, and are generally more powerful in practice. A standard procedure consists in first projecting the input vectors into multiple low-dimensional spaces, and combining the results of all attention layers using a linear function. Let d_I (resp. d_O) be the input (resp. output) dimension, h the number of heads, and $W_1, \dots, W_h \in \mathbb{R}^{d_I\times d}$ and $W_O \in \mathbb{R}^{d_O\times dh}$ be $h+1$ matrices, then

$$\text{MultAtt}(X) = W_O\Big(\text{Att}(W_1X)\Big\|\dots\Big\|\text{Att}(W_hX)\Big), \tag{7.5}$$

where the $\|$ operator denotes row-wise concatenation. In this chapter, we will consider each attention head separately, using the fact that the Lipschitz constant of multi-head attention can be bounded by that of each attention head.

Theorem 12. *If each attention head is Lipschitz continuous, then multi-head attention as defined in Equation (7.5) is Lipschitz continuous and*

$$L_F(\text{MultAtt}) \leqslant L_F(\text{Att})\|W_O\|_*\sqrt{\sum_{k=1}^{h}\|W_k\|_*^2}. \tag{7.6}$$

Proof. As $\text{MultAtt}(X) = W_O \begin{pmatrix} \text{Att}(W_1 X) \\ \vdots \\ \text{Att}(W_h X) \end{pmatrix}$, we have, for any matrix $H \in \mathbb{R}^{d \times n}$,

$$\mathbf{D}\text{MultAtt}_X(H) = W_O \begin{pmatrix} \mathbf{D}\text{Att}_{W_1 X}(W_1 H) \\ \vdots \\ \mathbf{D}\text{Att}_{W_h X}(W_h H) \end{pmatrix}. \quad (7.7)$$

First, note that, for any matrices $A \in \mathbb{R}^{n \times m}$ and $B \in \mathbb{R}^{m \times l}$, we have $\|AB\|_F^2 = \sum_i \|AB_i\|_2^2 \leqslant \sum_i \|A\|_*^2 \|B_i\|_2^2 = \|A\|_*^2 \|B\|_F^2$ by definition of the spectral norm $\|A\|_*$. Hence, we have, for any matrices $X, H \in \mathbb{R}^{d \times n}$,

$$\begin{aligned}
\|\mathbf{D}\text{MultAtt}_X(H)\|_F^2 &\leqslant \|W_O\|_*^2 \left\| \begin{pmatrix} \mathbf{D}\text{Att}_{W_1 X}(W_1 H) \\ \vdots \\ \mathbf{D}\text{Att}_{W_h X}(W_h H) \end{pmatrix} \right\|_F^2 \\
&= \|W_O\|_*^2 \sum_{k=1}^{h} \|\mathbf{D}\text{Att}_{W_k X}(W_k H)\|_F^2 \\
&\leqslant \|W_O\|_*^2 \sum_{k=1}^{h} |||\mathbf{D}\text{Att}_{W_k X}|||_F^2 \|W_k H\|_F^2 \\
&\leqslant \|W_O\|_*^2 \sum_{k=1}^{h} L_F(\text{Att})^2 \|W_k\|_*^2 \|H\|_F^2,
\end{aligned}$$

which leads to the desired result, as: $L_F(f) = \max_X |||\mathbf{D}f_X|||_F = \max_{X,H} \|\mathbf{D}f_X(H)\|_F / \|H\|_F$. □

Transformer Case: For Transformers, $m = n$ and the input matrix is decomposed as $X = (Q||K||V)$, where $Q, K, V \in \mathbb{R}^{d \times n}$ represent, respectively, *queries*, *keys* and *values*. The attention model is then

$$\text{Att}(X) = V \text{softmax}\left(\frac{Q^\top K}{\sqrt{d}}\right)^\top. \quad (7.8)$$

Note that the softmax is not multiplied by the whole input vector X, but only the values V. This is equivalent to projecting the output vectors on a subspace, and thus does not lead to an increase in the Lipschitz constant.

7.4 The Lipschitz Constant of Attention

As their name suggest, the purpose of attention layers is to select a small number of input vectors (softmax probabilities tend to focus most of their mass on the largest score). Unfortunately, large scores also tend to create large gradients. In order to show this behavior, we first provide a computation of the norm of the derivative of attention layers.

Derivative of Attention Models: A direct computation using the definition of Equation (7.4) and the chain rule gives

$$\mathbf{D}\mathrm{Att}_X(H) = H\mathrm{softmax}(g(X))^\top + X\mathbf{D}\mathrm{softmax}_{g(X)}(\mathbf{D}g_X(H))^\top, \quad (7.9)$$

where $H \in \mathbb{R}^{d\times n}$ is an input perturbation. We handle both terms separately, leading to the following upper bound on the Lipschitz constant.

Lemma 7. *For any $X \in \mathbb{R}^{d\times n}$, the norm of the derivative of attention models (see Equation (7.4)) is upper bounded by:*

$$\|\mathbf{D}\mathrm{Att}_X\|_F \leqslant \|\mathrm{softmax}(g(X))\|_F + \sqrt{2}\|X^\top\|_{(\infty,2)}\|\mathbf{D}g_X\|_{F,(2,\infty)}. \quad (7.10)$$

Equation (7.10) shows that the Lipschitz constant is controlled by two terms: the first one is related to the *uniformity* of the softmax probabilities, while the second one is related to the size of the input and gradient of the score function. In what follows, we will examine these two terms and show that normalizing the scores by a well-chosen scalar allows to control both simultaneously.

Proof. Using the chain rule on the derivative of $\mathrm{Att}(X) = X\mathrm{softmax}(g(X))^\top$, we immediately obtain, for any $H \in \mathbb{R}^{d\times n}$,

$$\mathbf{D}\mathrm{Att}_X(H) = H\mathrm{softmax}(g(X))^\top + X\mathbf{D}\mathrm{softmax}_{g(X)}(\mathbf{D}g_X(H))^\top, \quad (7.11)$$

and thus

$$\|\mathbf{D}\mathrm{Att}_X(H)\|_F \leqslant \|H\mathrm{softmax}(g(X))^\top\|_F + \|X\mathbf{D}\mathrm{softmax}_{g(X)}(\mathbf{D}g_X(H))^\top\|_F. \quad (7.12)$$

First, we have $\|H\mathrm{softmax}(g(X))^\top\|_F \leqslant \|H\|_F\|\mathrm{softmax}(g(X))\|_F$ by multiplicativity of the Frobenius norm. The second term follows from the bound, for any matrices $A \in \mathbb{R}^{d\times n}$ and $B \in \mathbb{R}^{m\times n}$,

$$\|A\mathbf{D}_X\mathrm{softmax}(B)^\top\|_F \leqslant \sqrt{2}\|A^\top\|_{(\infty,2)}\|B\|_{(2,\infty)}, \quad (7.13)$$

that we will prove below. Assuming that Equation (7.13) holds, we have, for any $H \in \mathbb{R}^{d\times n}$,

$$\|\mathbf{D}\mathrm{Att}_X(H)\|_F \leqslant \left(\|\mathrm{softmax}(g(X))\|_F + \sqrt{2}\|X^\top\|_{(\infty,2)}\|\mathbf{D}g_X\|_{F,(2,\infty)}\right)\|H\|_F, \quad (7.14)$$

and the desired result. Equation (7.13) is proven as follows: the derivative of the softmax is given by

$$\mathbf{D}_X\mathrm{softmax}(B)_{ij} = \sum_k \mathrm{softmax}(X)_{ij}\mathrm{softmax}(X)_{ik}(B_{ij} - B_{ik}), \quad (7.15)$$

and thus

$$
\begin{aligned}
(A\mathbf{D}_X\mathrm{softmax}(B)^\top)_{ij} &= \sum_k A_{ik}\mathbf{D}_X\mathrm{softmax}(B)_{jk} \\
&= \sum_{k,l} A_{ik}\mathrm{softmax}(X)_{jk}\mathrm{softmax}(X)_{jl}(B_{jk}-B_{jl}) \\
&= \sum_k A_{ik}\mathrm{softmax}(X)_{jk}B_{jk} \\
&\qquad - \sum_l \left(\sum_k A_{ik}\mathrm{softmax}(X)_{jk}\right)\mathrm{softmax}(X)_{jl}B_{jl} \\
&= \sum_k A_{ik}\mathrm{softmax}(X)_{jk}B_{jk} \\
&\qquad - \sum_l (A\mathrm{softmax}(B)^\top)_{ij}\mathrm{softmax}(X)_{jl}B_{jl} \\
&= \sum_k \mathrm{softmax}(X)_{jk}B_{jk}\left(A_{ik}-(A\mathrm{softmax}(B)^\top)_{ij}\right) .
\end{aligned}
$$

Inserting this last equality within the Frobenius norm, we get

$$
\begin{aligned}
\|A(\mathbf{D}_X\mathrm{softmax}(B))^\top\|_F^2 &= \sum_{i,j}\left(\sum_k \mathrm{softmax}(X)_{jk}B_{jk}(A_{ik}-(A\mathrm{softmax}(B)^\top)_{ij})\right)^2 \\
&\leqslant \sum_{i,j,k}\mathrm{softmax}(X)_{jk}B_{jk}^2\left(A_{ik}-(A\mathrm{softmax}(B)^\top)_{ij}\right)^2 \\
&= \sum_{j,k}\mathrm{softmax}(X)_{jk}B_{jk}^2\|A_k^\top-(A\mathrm{softmax}(B)^\top)_j^\top\|_2^2 ,
\end{aligned}
$$

where the inequality comes from Jensen's inequality applied to the square function (i.e. $\mathbb{E}[Z]^2 \leqslant \mathbb{E}[Z^2]$ for any r.v. Z) and the fact that $\mathrm{softmax}(X)_j$ is a probability distribution. Finally, as $(A\mathrm{softmax}(B)^\top)_j^\top$ is a weighted average of the vectors $A_k^\top$, and is thus in their convex hull, we have $\|A_k^\top-(A\mathrm{softmax}(B)^\top)_j^\top\|_2^2 \leqslant 2\max_k\|A_k^\top\|_2^2$, and thus

$$
\begin{aligned}
\|A(\mathbf{D}_X\mathrm{softmax}(B))^\top\|_F^2 &\leqslant 2\sum_{j,k}\mathrm{softmax}(X)_{jk}B_{jk}^2\|A^\top\|_{(\infty,2)}^2 \\
&\leqslant 2\sum_j\|\mathrm{softmax}(X)_j^\top\|_1\|B_j^\top\|_\infty^2\|A^\top\|_{(\infty,2)}^2 \\
&= 2\|B\|_{(2,\infty)}^2\|A^\top\|_{(\infty,2)}^2 ,
\end{aligned}
$$

where the second inequality uses the Hölder inequality and the last line is due to $\|\mathrm{softmax}(X)_j^\top\|_1 = 1$. This finishes the proof and leads to the desired inequality. □

Uniformity of the Softmax Probabilities: The first term in Equation (7.10) is directly related to *how far* the softmax probabilities are from being uniform. More precisely, we have

$$\|\text{softmax}(g(X))\|_F = \sqrt{\frac{m + \sum_{i=1}^{m} d_{\chi^2}(S_i, U_n)}{n}}, \tag{7.16}$$

where S_i is the i-th row of $\text{softmax}(g(X))$, U_n is the uniform distribution over n elements, and $d_{\chi^2}(p, q) = \sum_i q_i(p_i/q_i - 1)^2$ is the χ^2-divergence between p and q [47]. Hence, if all attention heads have uniform probabilities, then $d_{\chi^2}(S_i, U_n) = 0$ and $\|\text{softmax}(g(X))\|_F = \sqrt{m/n}$. On the contrary, the distances are maximum when the whole mass of the probabilities is on one element, and in such a case $d_{\chi^2}(S_i, U_n) = n - 1$ and $\|\text{softmax}(g(X))\|_F = \sqrt{m}$.

Lemma 8. *For any $M \in \mathbb{R}^{m \times n}$, we have*

$$\sqrt{m/n} \leqslant \|\text{softmax}(M)\|_F \leqslant \sqrt{m}\,. \tag{7.17}$$

When $m \gg 1$ (e.g. $m = n$ for Transformers), this implies that the gradients of attention models can be large and lead to the explosive phenomena observed in Section 7.6. Fortunately, controlling the scale of the scores is sufficient to control the uniformity of the probabilities.

Lemma 9. *If all the scores are bounded by $\alpha \geqslant 0$, i.e. for all $i \in \{1, \dots, m\}$ and $j \in \{1, \dots, n\}$, $|g(x)_{ij}| \leqslant \alpha$, then*

$$\|\text{softmax}(g(X))\|_F \leqslant e^{\alpha}\sqrt{\frac{m}{n}}\,. \tag{7.18}$$

Hence, the first objective of the normalization is to scale the scores to avoid softmax probabilities to put their entire mass on a single vector.

Impact of a Scalar Normalization: Without any additional control, Lemma 7 does not prove the Lipschitz continuity of attention models, as the second term is proportional to the norm of the input matrix $\|X^\top\|_{(\infty,2)}$. For example, if the scores are linear, then their derivative is constant, and the second term in Equation (7.10) is not bounded. In order to address this issue, we propose to normalize the score function by a scalar function $c : \mathbb{R}^{d \times n} \to \mathbb{R}_+$:

$$g(X) = \frac{\tilde{g}(X)}{c(X)}, \tag{7.19}$$

where $\tilde{g}$ is the original score function, and g the normalized one. When $c(X)$ is chosen wisely, this simple normalization is sufficient to obtain a tight bound on the Lipschitz constant of the attention (see Section 7.5).

Theorem 13. *Let $\alpha \geqslant 0$. If, for all $X \in \mathbb{R}^{d\times n}$, we have*

(1) $\|\tilde{g}(X)\|_\infty \leqslant \alpha c(X)$,

(2) $\|X^\top\|_{(\infty,2)} \|\mathbf{D}\tilde{g}_X\|_{F,(2,\infty)} \leqslant \alpha c(X)$,

(3) $\|X^\top\|_{(\infty,2)} \|\mathbf{D}c_X\|_{F,1} \|\tilde{g}(X)\|_{(2,\infty)} \leqslant \alpha c(X)^2$,

then attention models (see Equation (7.4)) with score function $g(X) = \tilde{g}(X)/c(X)$ are Lipschitz continuous and

$$L_F(\mathrm{Att}) \leqslant e^\alpha \sqrt{\frac{m}{n}} + \alpha\sqrt{8}\,. \tag{7.20}$$

Proof. Using Lemma 1 and Lemma 3 and the assumptions (1), we have,

$$\begin{aligned}\|\mathbf{D}\mathrm{Att}_X\|_F &\leqslant \|\mathrm{softmax}(g(X))\|_F + \sqrt{2}\|X^\top\|_{(\infty,2)} \|\mathbf{D}g_X\|_{F,(2,\infty)} \\ &\leqslant e^\alpha \sqrt{\frac{m}{n}} + \sqrt{2}\|X^\top\|_{(\infty,2)} \|\mathbf{D}g_X\|_{F,(2,\infty)}\,,\end{aligned}$$

where the first inequality is due to Lemma 1 and the second inequality is due to Lemma 3 and assumption (1) (as then $\|g(X)\|_\infty \leqslant \alpha$). Moreover, the derivative of the score function $g(X) = \tilde{g}(X)/c(X)$ gives

$$\mathbf{D}g_X(H) = \frac{\mathbf{D}\tilde{g}_X(H)}{c(X)} - \frac{\mathbf{D}c_X(H)\tilde{g}(X)}{c(X)^2}\,, \tag{7.21}$$

and thus,

$$\|\mathbf{D}g_X\|_{F,(2,\infty)} \leqslant \frac{\|\mathbf{D}\tilde{g}_X\|_{F,(2,\infty)}}{c(X)} + \frac{\|\mathbf{D}c_X\|_{F,1}\|\tilde{g}(X)\|_{(2,\infty)}}{c(X)^2}\,. \tag{7.22}$$

Finally, using this equation and assumption (2) and (3), we have

$$\begin{aligned}\|\mathbf{D}\mathrm{Att}_X\|_F &\leqslant e^\alpha \sqrt{\frac{m}{n}} + \frac{\sqrt{2}\|X^\top\|_{(\infty,2)} \|\mathbf{D}\tilde{g}_X\|_{F,(2,\infty)}}{c(X)} + \\ &\qquad + \frac{\sqrt{2}\|X^\top\|_{(\infty,2)} \|\mathbf{D}c_X\|_{F,1}\|\tilde{g}(X)\|_{(2,\infty)}}{c(X)^2} \\ &\leqslant e^\alpha \sqrt{\frac{m}{n}} + \sqrt{2}\alpha + \sqrt{2}\alpha \\ &\leqslant e^\alpha \sqrt{\frac{m}{n}} + \alpha\sqrt{8}\,,\end{aligned}$$

and the desired result. □

Remark 5. *Note that Theorem 2 still holds if* $\mathrm{Att}(X) = h(X)\,\mathrm{softmax}(g(X))^\top$ *and* $L_F(h) \leqslant 1$ *(i.e. the function h is contractive). In such a case, the assumptions become:*

(1) $\|\tilde{g}(X)\|_\infty \leqslant \alpha c(X)$,

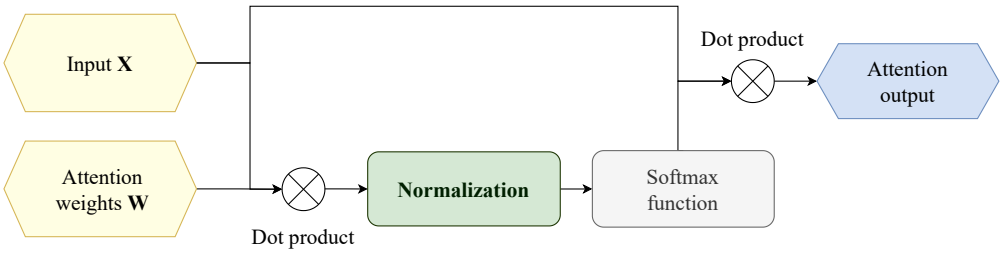


FIGURE 7.2: Pipeline of an attention mechanism along with the proposed normalization. For clarity, we assume a linear score function $g(x) = W^\top X$, expressed as a dot product operator.

(2) $\|h(X)^\top\|_{(\infty,2)} \|\mathbf{D}\tilde{g}_X\|_{F,(2,\infty)} \leqslant \alpha c(X)$,

(3) $\|h(X)^\top\|_{(\infty,2)} \|\mathbf{D}c_X\|_{F,1} \|\tilde{g}(X)\|_{(2,\infty)} \leqslant \alpha c(X)^2$.

First, note that α controls the scale of all the scores, as assumption (1) implies $\|g(X)\|_\infty \leqslant \alpha$. Thus, when the scores are allowed to reach values of order ≈ 1, and when $m \leqslant n$, Theorem 13 implies that the Lipschitz bound is also of order ≈ 1. The assumptions (1)-(3) of Theorem 13 are rather restrictive, and finding a proper normalization $c(X)$ in the general case is a difficult problem. However, we will see in the next section that a solution can be found in most practical cases of interest.

7.5 The *LipschitzNorm* normalization

In this section, we present our proposed normalization, the LipschitzNorm in three different settings: Lipschitz, linear and quadratic score functions. These settings cover most of the practical applications, including Transformers, GAT and GT models. All settings are particular instances of a common idea: impose assumptions (1)-(3) of Theorem 13 by dividing by the maximum of input values. A visualization of how LipschitzNorm is applied to an attention model with linear score function is shown in Figure 7.2.

Lipschitz scores: When the score function is Lipschitz, the assumptions in Theorem 13 can be met, for $\alpha \geqslant 0$, by

$$g(X) = \frac{\alpha\, \tilde{g}(X)}{\max\left\{\|\tilde{g}(X)\|_{(2,\infty)}, \|X^\top\|_{(\infty,2)} L_{F,(2,\infty)}(\tilde{g})\right\}}. \tag{7.23}$$

The denominator is composed of two terms: the first ensures that all the scores are bounded (by α), while the second ensures that the gradient of the normalized scores remains low compared to the scale of the input vector. Note that $\|X^\top\|_{(\infty,2)}$ is the maximum of the norm of input vectors.

Theorem 14. *If the score function $\tilde{g}$ is Lipschitz continuous, then the attention layer with score function as defined in Equation (7.23) is Lipschitz continuous*

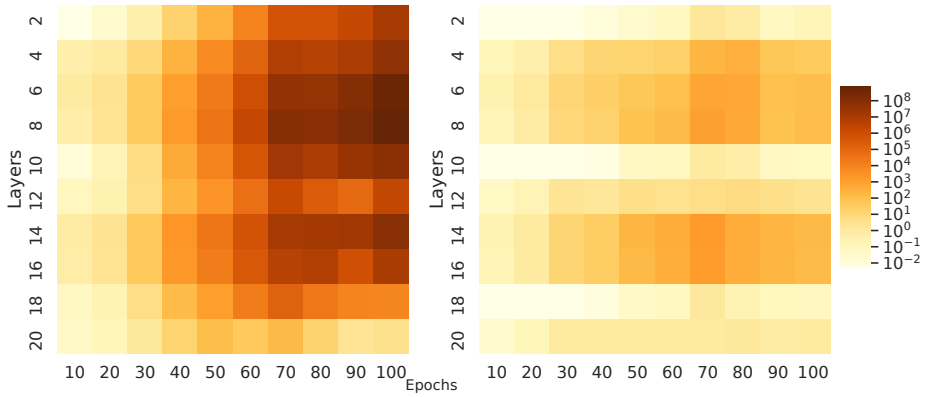


FIGURE 7.3: Gradient evolution of attention weights of a 20-layer GAT model for each layer throughout training. Each cell i, j represents the norm of the gradients of the attention weights in the i-th layer and trained until j-th epoch. The left heat map corresponds to the standard GAT without any normalization, where the phenomenon of **gradient explosion** occurs. The right heat map corresponds to the GAT model using LipschitzNorm. The proposed normalization restrains the attention weights from explosion.

and

$$L_F(\text{Att}) \leqslant e^{\alpha}\sqrt{\frac{m}{n}} + \alpha\sqrt{8}\,. \tag{7.24}$$

The proof can be found in Appendix B.5.1. Note that $\alpha = 0$ leads to a uniform distribution and gradient vanishing when $m \ll n$ (for example $m = 1$ when the output is a single vector). On the contrary, a large $\alpha \gg 1$ will lead to very large gradients that may destabilize training. In our experiments, we show that $\alpha = 1$ is a good trade-off that allows to create relatively peaked attention weights, while maintaining a low Lipschitz constant (see Section 7.7).

Linear Score Function: The initial definition of attention layers considers a linear score function $\tilde{g}(X) = Q^\top X$ for $Q \in \mathbb{R}^{d\times m}$. As this score function is Lipschitz continuous, Theorem 14 is directly applicable and leads to the following normalization, called LipschitzNorm,

$$g(X) = \frac{Q^\top X}{\|Q\|_F \|X^\top\|_{(\infty,2)}}\,. \tag{7.25}$$

Note that, contrary to Transformers, the query matrix Q is assumed to be a parameter of the model instead of an input.

Corollary 2. *The attention layer with score function as defined in Equation (7.25) is Lipschitz continuous and*

$$L_F(\text{Att}) \leqslant e^1 \sqrt{\frac{m}{n}} + \sqrt{8}\,. \tag{7.26}$$

Transformer Case: For Transformers, the query matrix Q is an input of the model, and the score function is thus quadratic. As quadratic functions are not Lipschitz, Theorem 14 is not applicable. Fortunately, we can adapt the same idea to this setting. As defined in Section 7.3.3, let $X = (Q||K||V)$ be a concatenation of queries, keys and values. Then, the assumptions in Theorem 13 are met by

$$g(X) = \frac{Q^\top K}{\max\{uv, uw, vw\}}\,, \tag{7.27}$$

where $u = \|Q\|_F$, $v = \|K^\top\|_{(\infty,2)}$, and $w = \|V^\top\|_{(\infty,2)}$. Compared to the linear case of Equation (7.25), we decompose the input matrix norm $\|X^\top\|_{(\infty,2)}$ into $\|K^\top\|_{(\infty,2)}$ and $\|V^\top\|_{(\infty,2)}$ and return the product between the maximum and second maximum of $\|Q\|_F$, $\|K^\top\|_{(\infty,2)}$, and $\|V^\top\|_{(\infty,2)}$.

Corollary 3. *The attention layer with score function as defined in Equation (7.27) is Lipschitz continuous and*

$$L_F(\text{Att}) \leqslant e^{\sqrt{3}} \sqrt{\frac{m}{n}} + 2\sqrt{6}\,. \tag{7.28}$$

Finally, as discussed in Section 7.3.3, we normalize multi-head attention by normalizing each attention head separately. Theorem 12 then directly implies the following bound on the Lipschitz constant of the whole multi-head attention layer:

$$L_F(\text{MultAtt}) \leqslant 11 \|W_O\|_* \sqrt{\sum_{k=1}^{h} \|W_k\|_*^2}\,, \tag{7.29}$$

where $W_1, \ldots, W_h$ (resp. W_O) represent the input (resp. output) projection matrices (see Equation (7.5)), and $m = n$.

Implementation details: Given an attention model $\mathcal{M}$ with score function g, we define $\mathcal{M}$-Lip as the updated attention model with the application of LipschitzNorm. This normalization requires three steps, that we now provide for the linear and Transformer settings:

1. **Frobenius norm of the queries**: First, we compute the Frobenius norm of Q: $u = \sqrt{\sum_i \|q_i\|_2^2}$.

2. **Input norms**: Then, we compute the maximum 2-norm of the input vectors $v = \max_i \|x_i\|_2$ (or $v = \max_i \|k_i\|_2$ and $w = \max_i \|v_i\|_2$ for Transformers).

3. **Scaling**: Finally, we divide the score function by the product uv (or $\max\{uv, uw, vw\}$ for Transformers).

Each attention head is treated separately, and thus all the norms and maximums are taken *per head*. Moreover, in the case of graph attention, the norms and maximums are computed *neighbor-wise*, i.e. for each node, we compute the maximum of the 2-norms of its neighbors.

Complexity: The overcost of the proposed method is based on the row-wise and column-wise norm computations. Given that n is the number of input vectors, d is the representation dimensionality and h the number of heads, the complexity of LipschitzNorm $\mathcal{O}(hnd)$. It remains negligible w.r.t. the overall cost of attention that is $\mathcal{O}(hnd^2)$.

7.6 Gradient Explosion and Vanishing

Similar to the deep neural networks, the design and efficient training of deep attention models has a tight connection with their Lipschitz continuity. In fact, given M Lipschitz continuous attention layers $\text{Att}_m(\cdot)$ with Lipschitz constants $l_m = L_F(\text{Att}_m)$, their composition $f = \text{Att}_1 \circ \text{Att}_2 \circ ... \circ \text{Att}_{m-1} \circ \text{Att}_m$ is Lipschitz continuous with Lipschitz constant upper-bounded by

$$L_F(f) \leqslant \prod_{m=1}^{M} L_F(\text{Att}_m). \tag{7.30}$$

Equation (7.30) implies that there is a multiplicative effect on the gradient flow of an M-layered attention model. Thus, enforcing the attention layer to be Lipschitz continuous with tight Lipschitz bounds can alleviate gradient explosion and allow for the design of deeper attention-based models.

Figure 7.3 (left picture) shows that gradient explosion occurs in a deep (20 layers) Graph Attention Network (GAT) [220] applied to a node classification task on Cora dataset [153]. Throughout 100 epochs of training, the gradients of the attention weights in each GAT layer exhibit a steep increase, reaching extremely large value of the order of 10^8. However, Figure 7.3 (right picture) shows that LipschitzNorm is able to prevent gradient explosion, and throughout training, the gradients of the attention weights remain stable. Figure 7.4 shows that gradient explosion also comes with poor performance, and even a total lack of improvement in training accuracy for a GAT model with 30 layers. Again, LipschitzNorm avoids this behavior and allows for proper training in all regimes, showing that enforcing the Lipschitz continuity of the attention layer can help towards the design of deeper architectures.

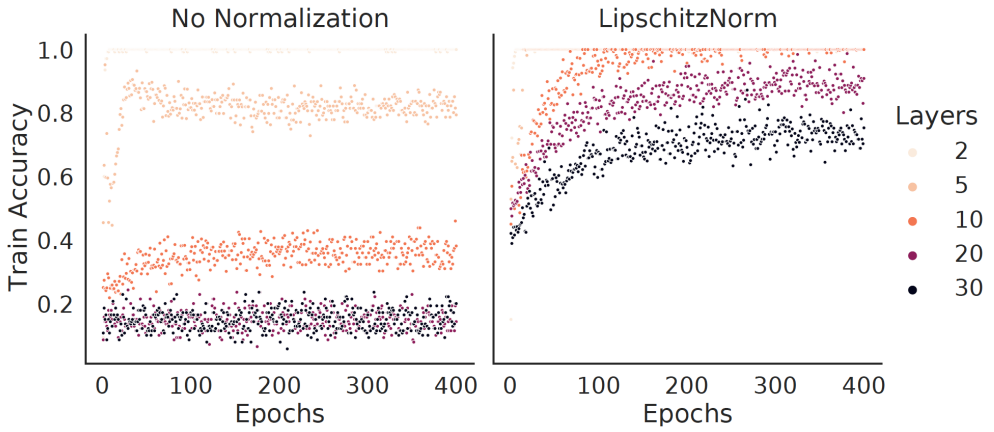


FIGURE 7.4: Convergence of train accuracy for a GAT model on node classification task using no normalization (left) and using LipschitzNorm (right).

7.7 Experimental Evaluation

We now examine the practical contribution of our normalization LipschitzNorm in real-world and synthetic benchmarks. In Section 7.7.1, we evaluate LipschitzNorm in real-world datasets that require the design of deeper GNN models. In Section 7.7.2, we perform a synthetic study of increasing data and model depth and in Section 7.7.3, we apply LipschitzNorm to attention-based GNNs of increasing model depth in real-world node classification tasks.

7.7.1 Node Classification with Missing Information

In most standard node classification benchmarks, the nodes present short-range dependencies, thus, making the fair evaluation of deeper models a difficult task. Towards a more solid comparison of deep GNNs, Zhao and Akoglu [252] presented a realistic framework that requires the design of deeper models by introducing an information noise of missing feature vectors. In particular, for a node classification task let a node attributed graph $\mathcal{D} = (V_u \cup V_l, E, X, U)$, where V_u, V_l are the node sets of the unlabeled and the labeled nodes respectively, E is the edge set, $X \in \mathbb{R}^{n \times d}$ is the node attribute matrix and $U \in \mathbb{N}^{n \times m}$ is the label matrix. For an unlabeled node subset $\mathcal{M} \subseteq V_u$ we remove its node attributes: $\{X_j | j \in \mathcal{M}\}$ and we call this framework *missing-vector setting* with fraction $p = \frac{|\mathcal{M}|}{|V_u|}$. This setting can represent cases of graph-based classification tasks with the cold-start phenomenon (i.e there is no history/feature information of the entities/nodes).

Dataset and Model Setup: We used three standard node classification datasets Cora, CiteSeer and PubMed [153, 78]. More details on the datasets can be found in Appendix A.2. The train/validation/test

TABLE 7.1: Classification accuracies for the missing-vector setting. In parentheses we denote the number of layers of the best model chosen for the highlighted accuracy. We denote by '-Pn' the application of PairNorm and by '-Lip' the application of the proposed LipschitzNorm.

	Cora		CiteSeer		Pubmed	
	0%	100%	0%	100%	0%	100%
GCN	82.5± 1.2 (2)	58.8 ± 3.5 (2)	**69.5 ± 2.1 (2)**	31.3 ± 2.7 (2)	77.9 ± 1.4 (2)	44.9 ± 4.4 (2)
GGNN	81.8 ± 2.0 (2)	68.2 ± 2.5 (6)	68.5 ± 1.9 (3)	40.5 ± 1.4 (5)	78.4 ± 2.1 (4)	56.6 ± 1.9 (4)
GAT	82.3 ± 2.3 (2)	65.3 ± 2.1 (4)	69.3 ± 1.6 (2)	42.8 ± 1.6 (4)	77.4 ± 0.5 (6)	63.1 ± 0.7 (4)
GAT-Pn	78.8 ± 0.6 (4)	73.8 ± 1.2 (12)	67.2 ± 0.8 (4)	**51.7 ± 1.1 (10)**	77.6 ± 1.6 (8)	70.4 ± 1.1 (12)
GAT-Lip	**83.1± 0.5 (5)**	**75.3± 0.9 (11)**	69.1 ± 1.5 (3)	50.9 ± 1.9 (9)	**78.9 ± 1.3 (5)**	**73.3 ± 1.4 (15)**

splits were the same as in [118]. Following Zhao and Akoglu [252], for each dataset we had 2 node feature setups: the 0% setup, where no attribute were removed and the 100% setup, where all attributes of the unlabeled nodes were removed. We experimented with 3 models: 1) **GCN**: Graph Convolutional Network [118], 2) **GGNN**: Gated Graph Neural Network [139] and 3) **GAT**: Graph Attention Network [220].

Model Selection: For all three GNN models, i.e GCN, GGNN, GAT and the normalization scenarios we performed cross-validation with predefined train/validation/test splits. For a fair comparison we used the same splits for all three datasets (Cora, CiteSeer and PubMed) as reported and used in [118].

Hyper-parameter tuning: We performed grid-search to tune the hyper parameters. The hyper-parameters that were tuned are the following:

- **Number of GNN layers**: For all models and datasets, we used l GNN layers where $l \in \{1, 2, 3, .., 20\}$.
- **Hidden units size**: The dimensionality of the hidden units in all models was in $\{8, 16, 32, 64, 128\}$.
- **Attention heads**: In the case of the GAT model, the attention heads that we used were in $\{1, 2, 4, 8\}$.
- **Dropout ratio**: The dropout ratio was set in $\{0, 0.5\}$.

Results: Table 7.1 shows the average classification accuracy achieved in the standard and the missing-vector setting. LipschitzNorm enables the training of deeper GAT layers, as in the missing-vector setting with $p = 100\%$, GAT-Lip (i.e. the GAT model with LipschitzNorm) achieves state-of-the-art classification accuracies in four out of the six setups. Moreover, it is noteworthy that GAT-Lip exhibits a solid performance for both the 0% and the 100% scenarios, outperforming PairNorm.

The need for a deeper GNN model is clear in Fig. 7.5. We visualize the performance of a GAT model with and without LipschitzNorm

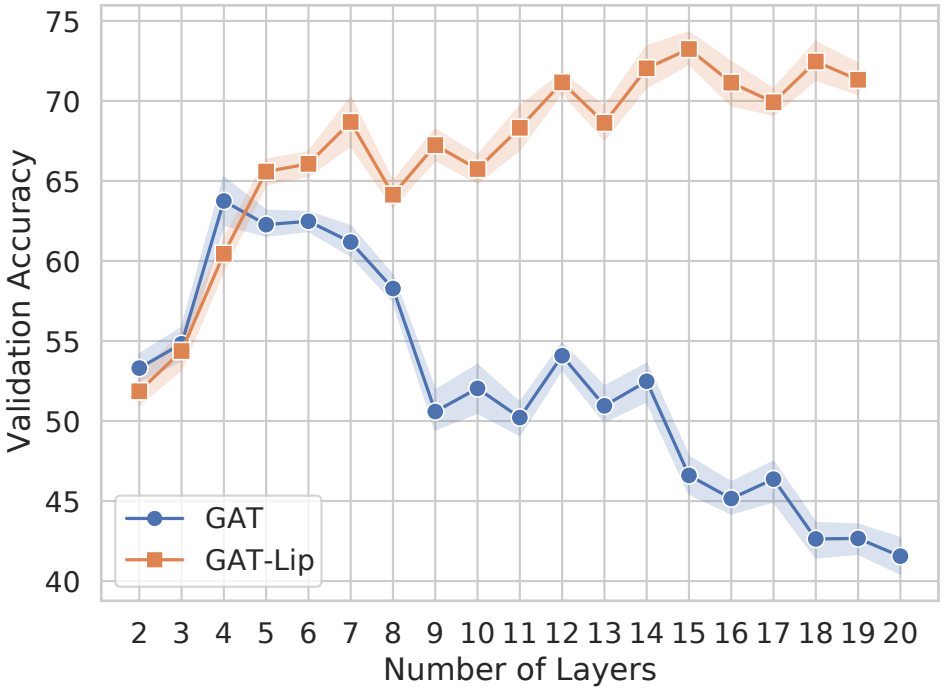


FIGURE 7.5: Classification accuracy of Graph Attention Network (GAT) with and without LipschitzNorm for the 100% setting of PubMed.

TABLE 7.2: Comparison of GAT-Lip, GCNII for Ogbn-arxiv in 0% setting and Cora/PubMed in 100%. Asterisk denotes the reported result in the public OGB leaderboard.

	Ogbn-arxiv (0%)	**Cora** (100%)	**PubMed** (100%)
GCNII	72.74 (-)*	74.9 $\pm$ 0.4 (14)	**73.9** $\pm$ **0.3** (16)
GAT-Lip	**74.62** $\pm$ **1.1 (8)**	**75.3**$\pm$ **0.9 (11)**	73.3 $\pm$ 1.4 (15)

in the 100% setting of the PubMed dataset. Specifically, the GAT-Lip model exhibits an increasing accuracy as the number of layers is higher, showing that a larger depth is required for the inference in the case of the missing feature information. Also, it is clear that LipschitzNorm has a crucial impact on the model training, as GAT without LipschitzNorm fails to learn the task.

Deep Attention vs Deep Convolution: LipschitzNorm is a normalization that can be included in any attention model to establish Lipschitz bounds and build deeper architectures. It is interesting to see how a deep attention-based graph model is compared to a deep convolution-based one. Thus, we compare the GAT-Lip model with the GCNII [39] in Cora, PubMed (100% setting) and in Ogbn-arxiv (0% setting) dataset [102].

Table 7.2 shows that GAT-Lip can outperform GCNII in the 0% setting of Ogbn-arxiv with a margin of $> 1.5\%$. Furthermore, on the 100% settings

of Cora and PubMed GAT-Lip and GCNII using a number of layers > 10 achieve similar accuracies without a clear lead.

7.7.2 Model Depth in Synthetic Trees

An intuitive way to show the ability of a deeper GNN model to capture long-range interactions is to generate synthetic graphs with nodes that are distant and have the same behavior. Thus, following Alon and Yahav [3], we create the TREES dataset. That is a set of directed trees (from the root to the leaves) of labeled nodes with increasing depth $d \in \{2, ..., 10\}$, where the leaves of the tree are colored *blue*, the root of the tree and the predecessors of the leaves are colored *green* and the rest of the nodes remain uncolored. The task is to predict the label of the tree's root green node, according to the label of the other green nodes. In other words, the label of the root node is affected by the information from the leaves.

TREES Dataset: The generated tree structure simulate the exponential growth of the receptive field of the nodes, so that the information passes between two distant nodes. For this goal, we created for every tree depth 5000 binary trees and we run each experiment 10 times. Following Alon and Yahav [3], we did not use explicitly extra blue neighbors, but, instead, we encoded their existence with 1-hot vectors of their cardinality as node attributes of the green nodes.

Model Setup: We compared the performance of Graph Convolutional Network [118], Graph Isomorphism Network [238], Gated Graph Neural Network [139] and Graph Attention Network [220]. Each model was implemented with $d + 1$ graph layers, where $d =$ tree depth and the hidden units size is set to 32. Moreover, we used either no normalization (None case in Figure 7.6), PairNorm [252], or our proposed LipschitzNorm.

Discussion: Figure 7.6 shows the train accuracy of the GNN models as the tree depth increases. Aligned with the previously found results, GAT and GGNN exhibit a better behavior with respect to the model depth. Moreover, the application of normalization methods has a significant impact on the performance of deeper models. GAT using the proposed normalization clearly outperforms the other architectures (for tree depth=10, GAT-Lip achieves 68.3% training accuracy, while GAT-Pn achieves 47.8% and all other variants achieve $< 25\%$), showcasing the contribution of LipschitzNorm to the design of deeper architectures.

7.7.3 Model Depth in Real-World Datasets

In this section, we measure the behavior of LipschitzNorm with respect to an increasing number of layers in real-world datasets. We apply the proposed normalization to two types of attention-based GNNs. We use

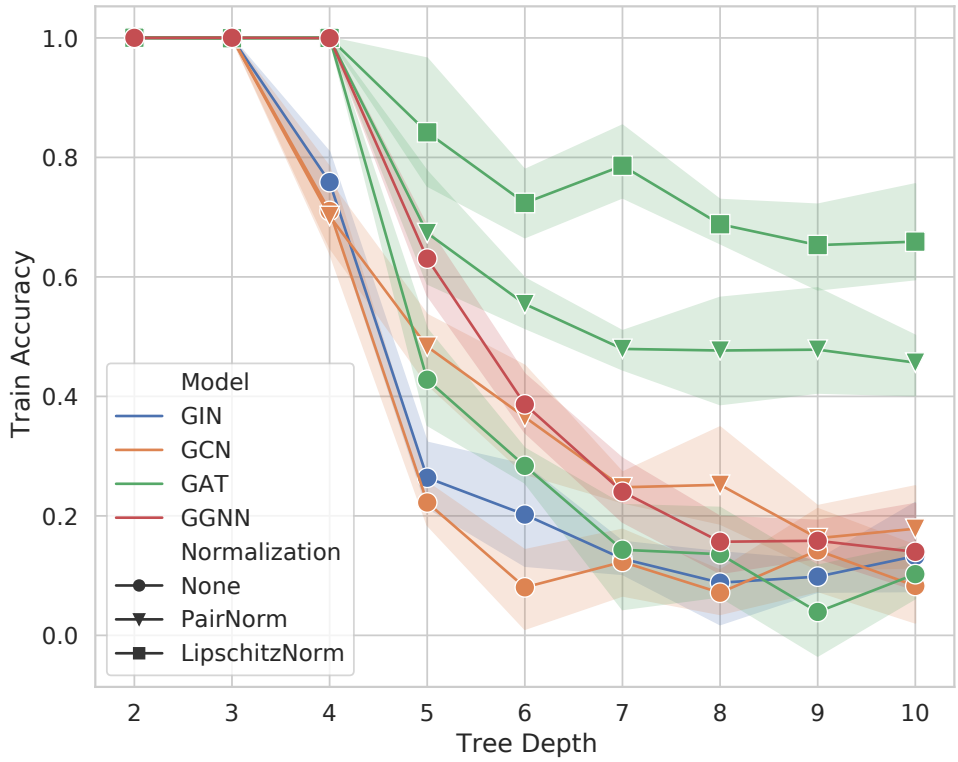


FIGURE 7.6: Train accuracies of four GNN models on the TREES dataset. We compare the model performance using: no normalization (circular dots), *PairNorm* (triangular dots) and the proposed *LipschitzNorm* (square dots).

the well-examined datasets *Cora* and *PubMed* [153] and two datasets from Open Graph Benchmark [102]: *Ogbn-arxiv* and *Ogbn-proteins*. Details and statistics of the datasets are provided in Appendix A.2.

Experimentation Setup: We used two attention-based graph neural networks, as described in Section 7.5: Graph Attention Network [220] and Graph Transformer [206]. For the two models, we compared the contribution of LipschitzNorm with two other normalization methods: the PairNorm [252] and the LayerNorm [10]. The number of attention layers was $l \in \{2, 5, 10, 15, 20, 25, 30\}$. We performed cross validation, where the train/validation/test splits in Cora and PubMed were the same as in Kipf and Welling [118], and for Ogbn-arxiv and Ogbn-proteins we used the same splitting methods as used in Hu et al. [102]. We used the Adam optimizer [117] with a weight decay $L = 5 * 10^{-4}$ and the initial learning rate was set in $\{0.1, 0.01, 0.005, 0.001\}$.

Model Selection: We performed for all models and datasets cross-validation with predefined train, validation and test splits and reported the best achieved validation accuracy. For Cora and PubMed, as in

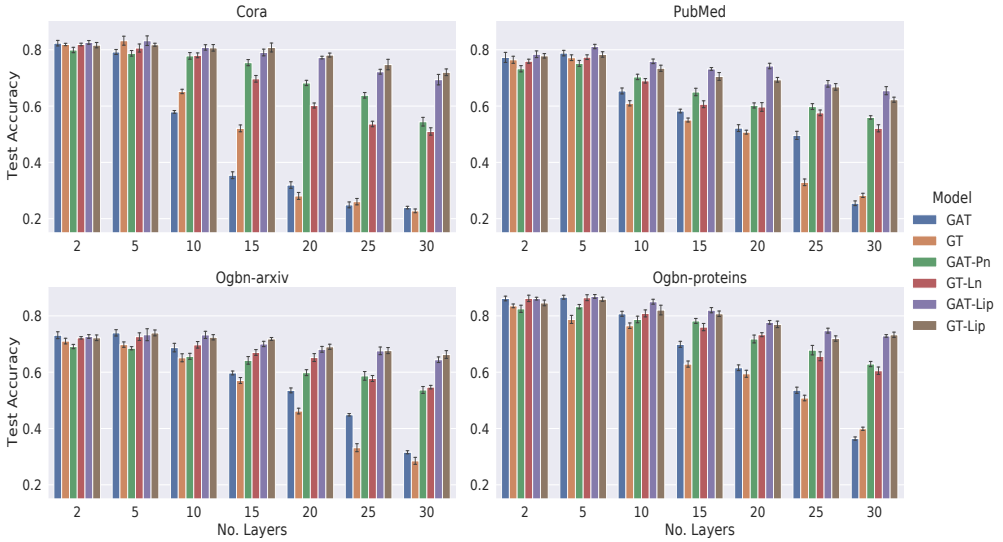


FIGURE 7.7: Test accuracies of a Graph Attention Network (GAT) and a Graph Transformer (GT). By '-Lip' we denote the application of *LipschitzNorm*, by '-Ln' the *LayerNorm* and by '-Pn' the *PairNorm*. In Ogbn-proteins dataset, the observed metric is ROC-AUC instead.

Section 7.1, we used the same splits as in [118]. For the other two datasets we have:

1. *Ogbn-arxiv*: We used the same splitting method as used in [102]. Specifically, the train split corresponds to the papers published until 2017, the validation split to the ones published in 2018 and the test split to the ones published in 2019. We used a full-batch training.

2. *Ogbn-proteins* For this dataset, we used, also, the same splitting method as in [102]. That is we split the nodes according to the node labels and in particular grouping according to the protein species. Similar to [206], we used neighbor sampling [90] as a sampling method, due to the size of the graph.

Model Depth: In order to examine the model behavior under the depth increase, for each architecture we used models consisting of l GNN layers, where $l \in \{2, 5, 10, 15, 20, 25, 30\}$. We run each experiment 5 times and we keep the configuration with the best average accuracy.

Hyper-parameter tuning: For each model depth and GNN model, we performed grid-search for hyper-parameter tuning. The hyper-parameters that were tuned are the following:

1. **Graph Attention Network** [220]: The dimensionality of the hidden units was set in $\{8, 16, 64, 128\}$. The number of attention heads was selected between $\{1, 2, 4, 8\}$ and we experimented over two

TABLE 7.3: Classification accuracies of deep GAT models (15 and 30 layers) with/without residual connections and with/without LipschitzNorm.

	Cora		**PubMed**	
Num. of layers	**15**	**30**	**15**	**30**
GAT	38.2 ± 1.7	29.5± 3.6	68.9± 1.5	28.2 ± 3.1
GAT-res	76.1 ± 1.2	63.5 ± 2.1	76.2 ± 1.1	63.8 ± 3.3
GAT-Lip	79.4 ± 0.7	**69.3 ± 4.1**	**76.4± 1.5**	67.2 ± 2.1
GAT-Lip-res	**80.2 ± 1.1**	**69.4 ± 2.8**	**77.3 ± 1.0**	**68.7 ± 1.8**

standard aggregators of the attention heads: a) *concatenation* and b) *averaging* of the attention heads. The dropout of the attention weights was set in $\{0, 0.2, 0.5\}$.

2. **Graph Transformer** from the UNIMP framework [206]: The hidden dimensionality was selected from $\{8, 16, 64, 128\}$ and the number of attentions heads from $\{1, 2, 4\}$. We tested *concatenation* and *averaging* of the attention heads and the dropout of the attention weights was set in $\{0, 0.5\}$.

Discussion: In Figure 7.7, we highlight the impact of LipschitzNorm on graph neural networks with respect to the model depth. For all four datasets LipschitzNorm enables both GAT and GT to learn and maintain information throughout layers. Even for a large number of layers ($l > 15$), where the models without any normalization fail to converge, the variants using the LipschitzNorm achieve comparable to the state-of-the-art results in the node classification tasks. More importantly, LipschitzNorm outperforms PairNorm and LayerNorm, as it enhances the performance of shallow architectures and maintains it to deeper architectures.

Residual Connections: Residual connections have been proven to be useful towards the design of deeper GNN models [133, 39]. Therefore, a comparison with LipschitzNorm and an evaluation how they can be combined is necessary. Table 7.3 suggests that residual connections with GAT layers enhance the performance of deep GNNs. However, LipschitzNorm significantly outperforms GAT-res in deep scenarios (+6% improvement on Cora with 30 layers). Moreover, combining LipschitzNorm with residual connections slightly improves the performance, showing the ability of our method to be smoothly incorporated in various models.

7.8 Conclusion

In this chapter, we introduced a novel normalization layer, called LipschitzNorm, for attention-based neural networks with a particular focus

on the design of attention-based GNNs. We proved that the application of LipschitzNorm enforces the Lipschitz continuity of self-attention layers. In an empirical study, we showed that Lipschitz continuous modules can prevent from gradient explosion phenomena and, thus, can improve the performance of deep attention models.

Focusing on Graph Neural Networks (GNNs), where designing deep models is still a challenging task, we applied LipschitzNorm to standard attention-based GNNs. We showed that LipschitzNorm allows to build deep GNN architectures with strong performance on node classification tasks that exhibit long-range interactions.

Chapter 8

Structural Symmetries in Graphs

8.1 Introduction

So far, we have studied models that learn representations over graphs, where the nodes are affected by their close neighbors. The property of node *homophily* or *assortativity* (i.e., similar nodes tend to interact with each other) is not always the case, as long-range dependencies, structural noise, or a misinformed network representation of the data can occur [152, 132, 133]. In Chapter 7, we investigated how attention-based GNNs can be extended into larger depths in order to capture more implicit information from larger neighborhoods through a normalization scheme.

Building deeper GNNs provides a solution for detecting information that is not expressed by close nodes. However, when the graph structure itself does not include such proximal relationships, deeper models do not seem to be an appropriate choice. However, some tasks require assigning similar representations to nodes that can be distant in the graph, but *structurally equivalent*, that is, nodes whose neighborhoods share similar structural characteristics. Figure 8.1 shows a visualization of the structural equivalence that can occur in a large network. For example, in chemistry, the properties of a molecule often depend on the interaction of the atoms at its opposite sides and their neighborhood topology [152].

These tasks require *structural representations*, i.e., embeddings that can identify structural properties of a node's neighborhood. There is a growing literature that addresses this problem through different approaches. RolX [96] extracts features for each node and performs non-negative matrix factorization to automatically discover node roles. Struc2vec [186] performs random walks on a constructed multi-layer graph to learn structural representations. GraphWave [63] and DRNE [218] employ diffusion wavelets and LSTM aggregation operators, respectively, to generate structural node embeddings. However, most of these approaches suffer from high time or space complexity.

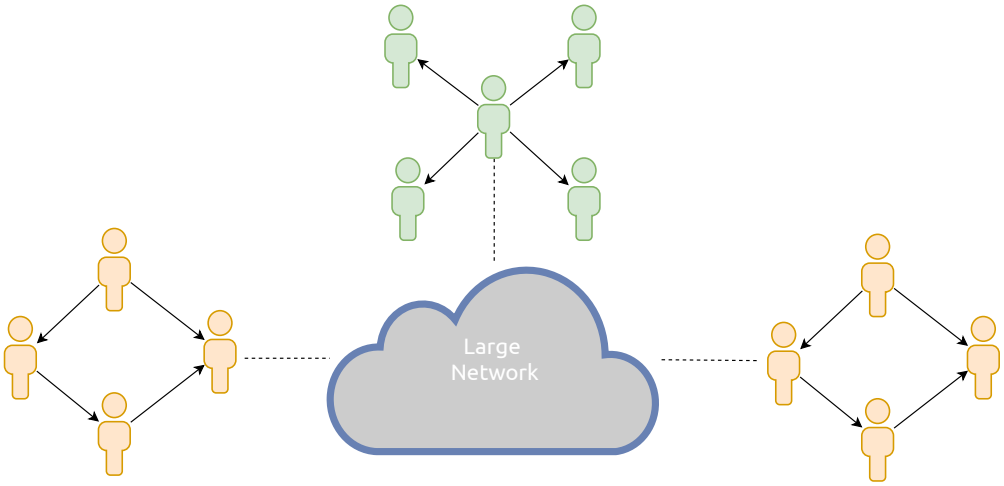


FIGURE 8.1: Nodes with similar roles can lie in distant parts of the same network.

In this chapter, we propose a novel and simple structural node representation algorithm, VNEstruct, that capitalizes on information-theoretic tools. The algorithm employs the Von Neumann entropy to construct node representations related to the structural identity of the neighborhood of each node. These representations capture the structural symmetries of the neighborhoods of the increasing radius of each node. We show empirically the ability of VNEstruct to identify structural roles and its robustness to graph perturbations through a node classification and node clustering study on highly symmetrical synthetic graphs. Moreover, we introduce a method of combining the generated representations by VNEstruct with the node attributes of a graph in order to avoid the incorporation of the graph topology in the optimization, contrary to the workflow of a GNN. Evaluated on real-world graph classification tasks, VNEstruct achieves state-of-the-art performance while maintaining a high efficiency compared to standard GNN models.

8.2 Structural Representations based on Von Neumann Entropy

Next, we present VNEstruct for generating structural node representations, employing the Von Neumann entropy, a model-agnostic measure that quantifies the structural complexity of a graph. The Von Neumann graph entropy (VNE) has been shown to have a linear correlation with other graph entropy measures [4]. Graph entropy methods have been recently proved successful for computing graph similarity [131].

8.2.1 Von Neumann Entropy on Graphs

In quantum mechanics, the state of a quantum mechanical system is described by a *density* matrix ρ, i.e., a positive semidefinite, hermitian

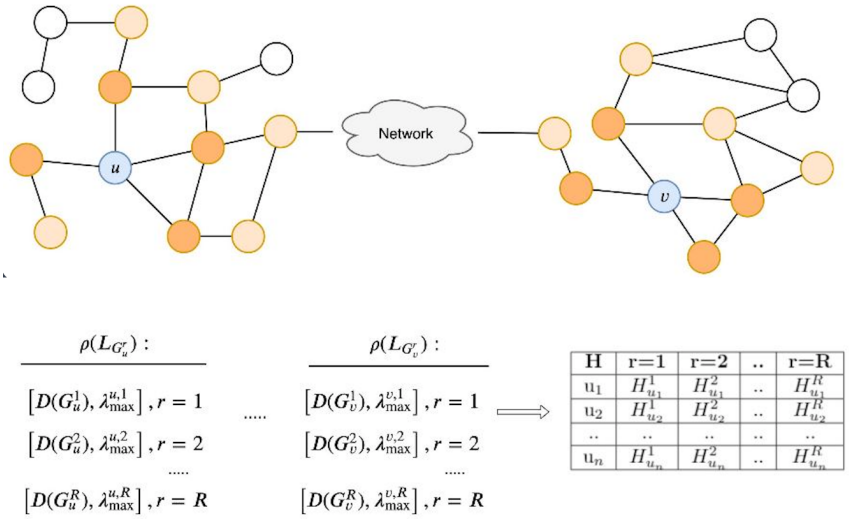


FIGURE 8.2: VNEstruct extracts ego-networks for a series of defined radii and computes the VNE for every radius. On the left, the two parts of the network may have a large distance through the network. The 1-hop ego-networks are highlighted with dark yellow, while the remaining nodes of the 2-hop ego-networks are highlighted with light yellow. The nodes u, v have structurally equivalent 1-hop neighborhoods G_u^1 and G_v^1, though their 2-hop neighborhoods G_u^2, G_v^2 do not.

matrix with unit trace [30]. The Von Neumann entropy of the quantum system is defined as:

$$H(\rho) = -\operatorname{Tr}(\rho \log \rho) = -\sum_{i=1}^{n} \lambda_i \log \lambda_i, \tag{8.1}$$

where $\operatorname{Tr}(\cdot)$ is the trace of a matrix, and λ_i's are the eigenvalues of ρ. Correspondingly, connecting it to graphs, given a graph $G = (V, E)$ and its Laplacian $L_G = D - A$, the VNE denoted by $H(G)$, is defined as in Equation 8.1, by replacing ρ with $\rho(L_G) = \frac{L_G}{\operatorname{Tr}(L_G)} = \frac{L_G}{2|E|}$ [30]. Note that $\lambda_i = \frac{1}{\operatorname{Tr}(L_G)} v_i$ where λ_i, v_i are the i-th eigenvalue of $\rho(L_G)$ and L_G, respectively. Therefore, $0 \leqslant \lambda_i \leqslant 1$ holds for all $i \in \{0, 1, \ldots, n\}$ [173]. This indicates that Equation 8.1 is equivalent to the Shannon entropy of the probability distribution $\{\lambda_i\}_{i=1}^n$. Hence, $H(G)$ serves as a skewness metric of the eigenvalue distribution, and it has been shown that it provides information on the structural complexity of a graph [173].

Efficient approximation scheme. The computation of VNE requires the eigenvalue decomposition of the density matrix which can be done in $\mathcal{O}(n^3)$ time. Recent works [40, 43] have proposed an efficient approximation of $H(G)$. Starting from Equation 8.1 and following [155], we obtain:

$$H(G) \approx \operatorname{Tr}\big(\rho(L_G)(I_n - \rho(L_G))\big) = Q, \tag{8.2}$$

where I_n is the $n \times n$ identity matrix, and

$$Q = \frac{\mathrm{Tr}(L_G)}{2m} - \frac{\mathrm{Tr}(L_G^2)}{4m^2} = 1 - \frac{1}{2m} - \frac{1}{4m^2}\sum_{i=1}^{n} d_i^2 \,, \tag{8.3}$$

where $m = |E|$ and d_i is the ith-node degree. Finally, as [40] suggests, we obtain a tighter approximation of $H(G)$:

$$\hat{H} = -Q \ln \lambda_{\max} \,, \tag{8.4}$$

where $\lambda_{\max}$ is the largest eigenvalue of $\rho(L(G))$. It can be shown that for any graph G, we have $H(G) \geqslant \hat{H}(G)$ where the equality holds if and only if $\lambda_{\max} = 1$ [43].

8.2.2 The VNEstruct Algorithm

Based on the VNE and its approximation, we introduce our proposed approach to construct structural representations. The VNEstruct algorithm extracts ego-networks of increasing radius and computes their VNE. Then, the representation of a node comprises of the Von Neumann entropies that emerged from the node's ego-networks. Therefore, the set of entropies of the ego-networks of a node serves as a "signature" of the structural identity of its neighborhood.

Let R be the maximum considered radius. For each $r \in \{1, .., R\}$ and each node $v \in V$, the algorithm extracts the r-hop neighborhood $G_v^r = (V', E')$, where $V' = \{u \in V | d(u, v) \leqslant r\}$ and $E' = \{(u, v) | u, v \in V', (u, v) \in E\}$. Next, $H(G_v^r)$ of the r-hop neighborhood of v is computed using Equation 8.4. Finally, the R entropies are arranged into a single vector $h_v \in \mathbb{R}^R$. As shown in Figure 8.2, VNEstruct identifies structural equivalences of nodes that are distant from each other. Specifically, nodes u and v share structurally identical 1-hop neighborhoods. Therefore, the entropies of their 1-hop neighborhoods are equal to each other.

Computational Complexity. The algorithm consists of: (1) the extraction of the ego-networks and (2) the computation of VNEs per subgraph. The first step is linear in the number of edges of the node's neighborhood. In the worst case, the complexity is $\mathcal{O}(nm)$, but for sparse graphs, the complexity is constant in practice. For the second step, following the approximation scheme in subsection 2.1, λ_{max} is computed through the power iteration method [156], which requires $\mathcal{O}(n + m)$ operations, as the Laplacian matrix has $n + m$ nonzero entries. Hence, the whole method exhibits linear complexity $\mathcal{O}(n + m)$, while for very sparse graphs, it becomes $\mathcal{O}(n)$.

Robustness over "small" perturbations. We will next show that utilizing the VNE, we can acquire robust structural representations over possible perturbations on the graph structure. Clearly, if two graphs are isomorphic to each other, then their entropies will be equal to each other. It is important, though, for structurally similar graphs to have

similar entropies, too. So, let $\rho, \rho' \in \mathbb{R}^{n \times n}$ be the density matrices of two graph laplacians $L_G, L_{G'}$, as described above. Let also $\rho = P\rho' P^\top + \epsilon$ where P is an $n \times n$ permutation matrix equal to $\arg\min_P ||\rho - P\rho' P^\top||_F$ and ϵ is an $n \times n$ symmetric matrix. If G, G' are nearly-isomorphic, then the Frobenius norm of ϵ is small. By applying the Fannes-Audenaert inequality [8], we have that:

$$|H(\tilde{\rho}) - H(\rho)| \leqslant \frac{1}{2} T \ln(n-1) + S(T),$$

where $T = ||\tilde{\rho} - \rho||_1$ is the trace distance between $\rho, \tilde{\rho}$ and $S(T) = -T \log T - (1-T)\log(1-T)$. However, $||\tilde{\rho} - \rho||_1 = \sum_i |\lambda_i^{\tilde{\rho}-\rho}| \leqslant n||\tilde{\rho} - \rho||_{op}$, where $||\cdot||_{op}$ is the operator norm. Therefore, $|H(\tilde{\rho}) - H(\rho)| \leqslant \frac{n}{2} ln(n-1)||\epsilon||_{op} + S(T)$, leading thus to a size-dependent upper bound of the difference between the entropies of structurally similar graphs.

8.2.3 Graph-level Representations

Next, we propose how the structural representations generated by VNEstruct can be combined with node attributes to perform graph classifications tasks. The majority of the state-of-the-art methods learn node representations using message-passing schemes [91, 238], where each node updates its representation according to its neighbors' representations, utilizing the graph structure information.

In this work, we do not use any message-passing scheme, and we ignore the graph structure. Instead, we augment the node attribute vectors of a graph with the structural representations generated by VNEstruct. Thus, information about the graph structure is implictly incorporated into the augmented node attributes. Given a matrix of node attributes $X \in \mathbb{R}^{n \times d}$, the approach performs the following steps:

- Computation of $H_v \in \mathbb{R}^{n \times R}$.
- Concatenation of node attribute vectors with structural node representations: $X' = [X||H] \in \mathbb{R}^{n \times (d+R)}$.
- Aggregation of node vectors X' into:
  $H_G = \psi(\sum_{v \in V_G} \phi(X'_v))$, where ϕ and ψ are MLPs.

This approach is on par with recent studies that propose to augment the node attributes with structural characteristics to avoid performing message-passing steps [41]. In comparison to a GNN, this procedure reduces the computational complexity of the training procedure since each graph is represented as a set of node representations.

8.3 Experiments

Next, we empirically show the robustness that VNEstruct exhibits to graph perturbations in subsection 3.1, and we evaluate its graph classification performance in subsection 3.2.

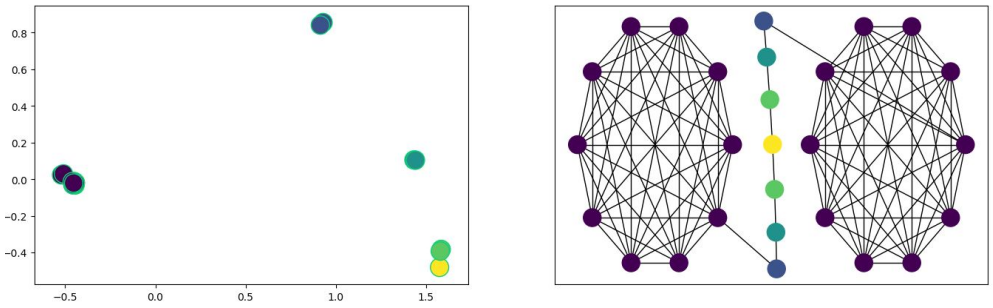


FIGURE 8.3: The barbell graph (right) and the 2-dimensional representations of its nodes produced by applying PCA to the VNEstruct embeddings (left).

8.3.1 Structural Role Identification

In order to evaluate the robustness of the structural representations generated by our method, we measure its performance on two cases of synthetic datasets, where the task of structural role identification is highlighted.

Toy example: Barbell Graph

This toy graph consists of two cliques of size 10 that are connected through a path of length 7. The graph is shown in Figure 8.3 (right). The different colors indicate the roles of the nodes in the graph. Figure 8.3 (left) illustrates the 2-dimensional representations of the 27 nodes of the graph. These representations were generated by the VNEstruct algorithm (we set $R = 3$ and then applied PCA to project them to the 2-dimensional space). We should mention that the proposed algorithm can identify the structural role of the nodes in the barbell graph and produce similar/identical embeddings for structurally similar/identical nodes.

Highly-symmetrical synthetic networks

The second case of the synthetic experiment consists of a collection of perturbed synthetic datasets, which were also introduced in Chapter 6, and it follows the experimental setup of Donnat et al. [63]. We perform both classification and clustering tasks with the same experimental setup as in [63].

Dataset setup. Similarly to Chapter 6, the generated synthetic datasets are identical to those used in [63]. They consist of basic symmetrical shapes, as shown in Table 8.1, that are regularly placed along a cycle of length 30. The *basic* setups use ten instances of only one of the shapes of Table 8.1 while the *varied* setups use ten instances of each shape, randomly placed along the cycle. The perturbed instances are formed by randomly rewiring edges. The colors in the shapes indicate the different classes.

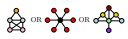
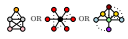
Configuration	Shapes	Algorithm	Homogeneity	Completeness	Silhouette	Accuracy	F1-score
Basic		DeepWalk	0.178	0.115	0.163	0.442	0.295
		RolX	**0.983**	**0.976**	0.846	**1.000**	**1.000**
		struc2vec	0.803	0.595	0.402	0.784	0.708
		GraphWave	0.868	0.797	0.730	0.995	0.993
		VNEstruct	**0.986**	**0.983**	**0.891**	0.920	0.901
Basic Perturbed		DeepWalk	0.172	0.124	0.171	0.488	0.327
		RolX	0.764	0.458	0.429	0.928	**0.886**
		struc2vec	0.625	0.543	0.424	0.703	0.632
		GraphWave	0.714	0.326	0.287	0.906	0.861
		VNEstruct	**0.882**	**0.701**	**0.478**	**0.940**	0.881
Varied		DeepWalk	0.327	0.220	0.216	0.329	0.139
		RolX	**0.984**	**0.939**	0.748	**0.998**	0.996
		struc2vec	0.805	0.626	0.422	0.738	0.592
		GraphWave	0.941	0.843	**0.756**	0.982	**0.965**
		VNEstruct	0.950	**0.945**	0.730	0.988	0.95
Varied Perturbed		DeepWalk	0.300	0.231	0.221	0.313	0.128
		RolX	0.682	0.239	0.062	0.856	0.768
		struc2vec	0.643	0.524	**0.433**	0.573	0.412
		GraphWave	0.670	0.198	0.005	0.793	0.682
		VNEstruct	**0.722**	**0.678**	0.399	**0.899**	**0.878**

TABLE 8.1: Performance of the baselines and the *VNEstruct* method for learning structural embeddings averaged over 20 synthetically generated graphs. Dashed lines denote perturbed graphs.

Evaluation. For the classification task, we measure the *accuracy* and the *F1-score*. For the clustering task, we report the 3 evaluation metrics that were also calculated in [63]: the *Homogeneity* evaluates the conditional entropy of the structural roles in the produced clustering result, the *Completeness* evaluates how many nodes with equivalent structural roles are assigned to the same cluster, and the *Silhouette* measures the intra-cluster distance vs. the inter-cluster distance.

In Table 8.1, VNEstruct outperforms the competitors on the perturbed instances of the synthetic graphs. On the basic and varied configurations, VNEstruct outperforms the competitors in the node clustering evaluation and achieves comparable performance with RolX in node classification. On the perturbed configurations, VNEstruct exhibits stronger performance than its competitors. The results in Table 8.1 suggest a comparison of VNEstruct, RolX, and GraphWave in noisy scenarios. This comparison is provided in Figure 8.4, where we report the performance with respect to the number of rewired edges (from 0 to 20). We see that VNEstruct is more robust than GraphWave and Rolx in the presence of noise.

8.3.2 Graph Classification

Next, we evaluate VNEstruct and the baselines in the task of graph classification. We compare our proposed algorithm against well-established message-passing algorithms for learning graph representations. Note that in contrast to most of the baselines, we pre-compute the entropy-based structural representations, and then we represent each graph as a set of vectors that encode structural characteristics. We used four common graph classification datasets (three from bioinformatics:

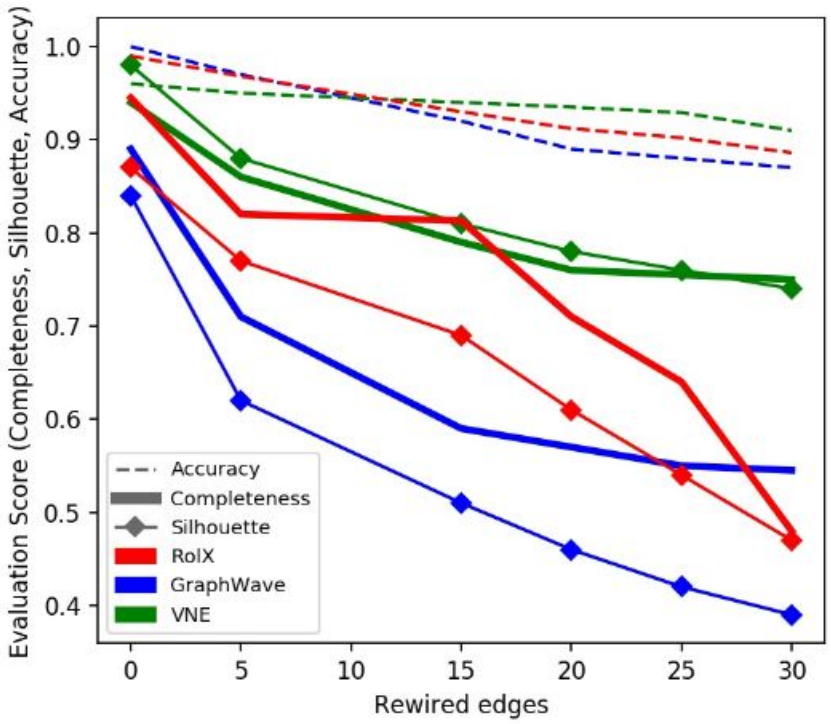


FIGURE 8.4: Classification and clustering performance of VNEstruct and the baselines with respect to noise.

MUTAG, PROTEINS, PTC-MR and one from social-networks: IMDB-BINARY [238]).

Baselines. The goal of the comparison is to show that by decomposing the graph structure and the attribute space, we can achieve comparable results to the state-of-the-art algorithms. Thus, we use as baselines graph neural network variants and specifically: DGCNN [249], Capsule GNN [236], GIN [238], GCN [118], GAT [220]. Moreover, GFN [41] augments the attributes with structural features and ignores the graph structure during the learning procedure.

Model setup. For the baselines, we followed the same experimental setup, as described in [41] and, thus, we report the achieved accuracies. For GAT, we used a summation operator as an aggregator of the node vectors into a graph-level representation. Regarding the VNEstruct, we performed 10-fold cross-validation with Adam optimizer, and a 0.3 learning rate decay every 50 epochs. In all experiments, we set the number of epochs to 300. We choose the radius of the ego-networks from $r \in \{1, 2, 3, 4\}$ and the number of hidden layers of the MLPs from $d \in \{8, 16, 32\}$.

Table 8.2 illustrates the average classification accuracies of the proposed approach and the baselines on the 5 graph classification datasets. Interestingly, the proposed approach achieves accuracies comparable to those of the state-of-the-art message-passing models. VNEstruct outperformed

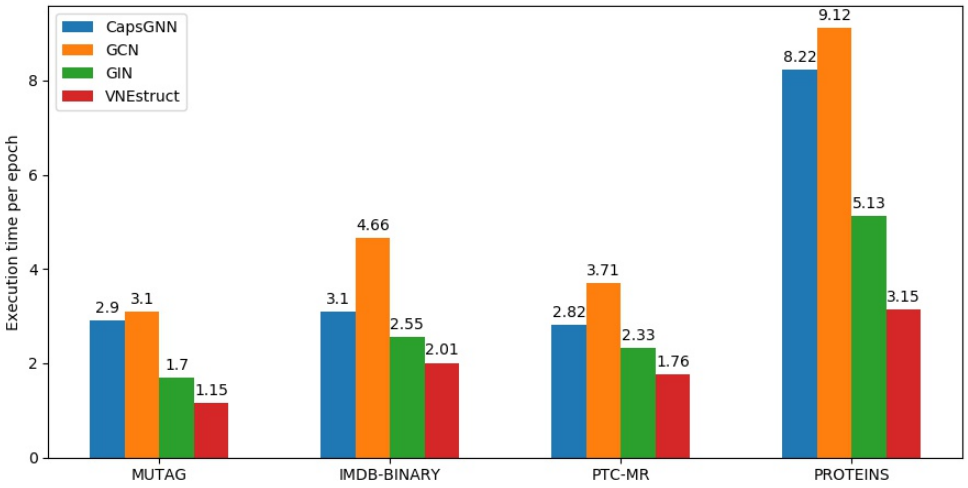


FIGURE 8.5: Training time per epoch (in sec) of VNEstruct and competitors for the graph classification tasks.

	MUTAG	IMDB-BINARY	PTC-MR	PROTEINS
DGCNN	85.83 ± 1.66	70.03 ± 0.86	58.62±2.34	75.54 ± 0.94
CapsGNN	86.67 ± 6.88	73.10 ± 4.83	-	76.28 ± 3.63
GAT	88.90 ± 3.21	75.39 ± 1.30	63.87 ± 5.31	76.1 ± 2.89
GIN	89.40 ± 5.60	75.10 ± 5.10	64.6 ± 7.03	76.20 ± 2.60
GCN	87.20 ± 5.11	73.30 ± 5.29	64.20 ± 4.30	75.65 ± 3.24
GFN	90.84 ± 7.22	73.00 ± 4.29	-	**77.44 ± 3.77**
VNEstruct	**91.08 ± 5.65**	**75.40 ± 3.33**	**65.39 ± 8.57**	77.41 ± 3.47

TABLE 8.2: Average classification accuracy (± standard deviation) of the baselines and the proposed VNEstruct.

all the baselines on 3 out of 4 datasets while achieving the second-best accuracy on the remaining dataset (i.e., PROTEINS).

Figure 8.5 illustrates the average training time per epoch of VNEstruct and the baselines that apply message-passing schemes. The proposed approach is generally more efficient than the baselines. Specifically, it is 0.31 times faster than GIN and 0.60 times faster than GCN on average. This improvement in efficiency is mainly because the graph structural features are computed in a preprocessing step and are then concatenated with the node attributes. However, the computational cost of the preprocessing step is negligible, as it is performed only once in the experimental setup. Furthermore, we should mention that due to the low dimensionality of the generated embeddings ($d \leqslant 4$), our method does not have any significant requirements in terms of memory.

8.4 Conclusion

In this chapter, we proposed VNEstruct to generate structural node representations based on the entropies of ego-networks. We showed the robustness of VNEstruct empirically under the presence of noise

in the graph. We also proposed an approach for performing graph classification that combines the representations of VNEstruct with the nodes' attributes, avoiding the computational cost of message passing schemes. The proposed approach exhibited a strong performance in real-world datasets, maintaining high efficiency.

Chapter 9

Conclusions and outlook

Graph neural networks have been a subject of accelerating research in the last years, mainly due to their broad applicability and astonishing results in diverse applications. Chemistry, bioinformatics, social networks, telecommunications, and meta-research are a few fields that have been affected by the emergence of graph representation learning. The ability of graph learning models to provide accurate predictions, build expressive representations and remain robust in deep scenarios is crucial.

9.1 Concluding remarks

The present thesis studied various conditions under which graph learning models can be expressive. Using both a theoretical study and an experimental evaluation, we focused on three different aspects: the *discrimination power* of graph representations, the *receptive field* of the used operators, and the *non-local interactions* that can occur in a network.

Discrimination power of graph representations

In Chapter 3 we studied *separability*, as the key topological criterion for building powerful graph neural networks. Moreover, we highlighted a set of graph classes (such as k-regular graphs) that standard message passing neural networks fail to express. Driven by the examined GNN limitations, we proposed CLIP, as an efficient approach, based on the Weisfeiler-Lehman test of graph isomorphism that can provide universal graph representations. Except for the theoretical guarantee of universality, CLIP was able to output state-of-the-art graph classification results in social and molecular networks.

In Chapter 4 investigating isometry properties of graph representation, we moved beyond the standard message-passing framework. In particular, we firstly showed that standard graph embedding models could not provide representations that preserve the isometry. We also suggested a new model, $\pi-$GNN, that learns soft permutation matrices in order to embed a corpus of graphs into a common space. $\pi-$GNN achieved

comparable to state-of-the-art results to both graph classification and graph regression tasks.

Receptive field of GNNs

In Chapter 5 we studied *graph shift operators* (GSOs). GSOs are matrices that most GNNs are using to encode the adjacency information, such as the adjacency matrix, the Laplacian matrix, or their normalized variants. Considering that different models and tasks require different operators, GNNs have limited expressivity over the neighborhood encoding. Studying the spectral properties of the graph shift operators, we introduced PGSO as a novel parametric family of operators that can adapt throughout training. PGSO showed great applicability and effectiveness over the training of numerous GNNs and had a critical impact on standard node classification and graph classification tasks.

In Chapter 6 we investigated whether standard aggregation schemes of GNNs can identify specific graph properties, such as connectivity, bipartiteness, and triangle-freeness. We showed that the latter failed to identify the wanted properties. We proposed $k-$hop graph neural networks as an alternative that considers the $k-$hop neighborhood at a single aggregation layer. Based on this scheme, our model was able to detect structural information that was not detectable in using only the $1-$hop neighborhoods. $k-$hop GNN achieved to distinguish graph properties that standard GNNs could not and demonstrated strong results in node and graph classification tasks.

Non-local interactions and neural networks

In Chapter 7 we studied the inability of standard GNN models to scale to deeper layers due to their local nature. In many networks, long-range interactions appear to be crucial, and, thus, the building of deeper architectures is needed. However, in graph neural networks, phenomena such as oversmoothing and oversquashing are limiting their ability to converge. Driven by these observations, we investigated how the gradient flow of graph learning models is affected by an increasing model depth. In particular, we developed a theoretical study of the Lipschitz constant of a specific class of GNNs, the attention-based ones. This study introduced a new type of normalization, namely LipschitzNorm, that allowed the design of deeper attention models and showed great success in real-world node classification problems.

In Chapter 8 we addressed questions in the area of structural node representations. The main difference between standard node representations and structural ones is that the latter depends on structural patterns and symmetries that can occur in a neighborhood. That being said, two nodes can have similar embeddings if they have a structurally equivalent neighborhood, regardless of their proximity. This type of interaction can be seen in real-world scenarios, such as molecular graphs, where the

structural symmetries play an essential role. We proposed VNEstruct, a simple structural node embedding algorithm that takes into account ego-based entropy measures of the neighborhoods, showing a solid performance on graph classification tasks.

9.2 Outlook and future directions

Next, we make a brief discussion on possible directions that the works of the present thesis can be extended.

Knowledge graphs and multi-modality Our arguments over the expressivity of graph neural networks make assumptions over the properties that the examined graphs satisfy. As discussed in Chapter 2, simple graphs are assumed, where a pair of edges can be connected by at most one link. However, in many cases of real-world networks, graphs present complex structures with multiple links per pair of nodes, forming sub-structures with various interaction types. This is the result of multi-modal learning scenarios, where different sources of information can be combined, forming network structures. *Hyper-networks* and *Knowledge graphs* are two approaches of formulating various types of graphs, where the edges correspond to relations between nodes and various relation types can occur in the same network [5, 17]. The extension of our work on the receptive field of graph neural networks over more complex forms of graphs can be studied. For example, the distance of two graphs can be defined using the permutation set of the adjacency matrix. Defining adjacency operators in hyper-networks seems a non-trivial task. Learning representations using a rich receptive field on such networks can offer new insights and unlock discoveries.

Non-locality and heterophily Most of the graphs modeling real-world problems are characterized by locality and homophily, which means the necessary amount of information for a node representation is concentrated in its direct neighborhood. Many standard inference methods are designed under these assumptions. There is limited research work in non-local or heterophilous networks, where nodes that affect each other do not interact immediately. Such cases of networks occur in molecular analyses and brain research, where structural symmetries can play a vital role [152]. Our works on structural node embeddings and building robust deeper graph learning models can be applied to such networks.

Dynamic Networks The majority of the current literature that studies graph representation learning models represents networks as static graphs, i.e., graphs that do not change over time. However, a robust and expressive framework for temporal graph learning models needs to be established to capture the dynamics and behavioral patterns of networks through time. Progress towards this direction has been made

by introducing models for discrete-time and continuous-time temporal graphs [189, 172]. Learning, however expressive representations in such dynamic scenarios, though, is far from optimal, and arguments over the discrimination power of dynamic graph learning models shall be expressed.

Applications Networks provide a flexible and formal framework to model complex interactions between components. A rapidly accelerating research focuses on extracting knowledge from graph structures that provide representations for diverse fields ranging from technological areas (e.g., communication networks, logistics) to socioeconomic areas, such as biomedical graphs, political networks, and social networks. Learning expressive representations can provide accurate predictions to such fields and, consequently, yield a high societal impact.

Bibliography

[1] Sami Abu-El-Haija et al. "MixHop: Higher-Order Graph Convolutional Architectures via Sparsified Neighborhood Mixing". In: *Proceedings of the 36th International Conference on Machine Learning*. 2019, pp. 21–29.

[2] Yonathan Aflalo, Alexander Bronstein, and Ron Kimmel. "On convex relaxation of graph isomorphism". In: *Proceedings of the National Academy of Sciences* 112.10 (2015), pp. 2942–2947.

[3] Uri Alon and Eran Yahav. "On the Bottleneck of Graph Neural Networks and its Practical Implications". In: *International Conference on Learning Representations*. 2021.

[4] Kartik Anand, Ginestra Bianconi, and Simone Severini. "Shannon and von Neumann entropy of random networks with heterogeneous expected degree". In: *Phys. Rev. E* 83 (3 2011), p. 036109.

[5] Siddhant Arora. "A Survey on Graph Neural Networks for Knowledge Graph Completion". In: *CoRR* abs/2007.12374 (2020). arXiv: 2007.12374. URL: https://arxiv.org/abs/2007.12374.

[6] Vikraman Arvind et al. "Approximate Graph Isomorphism". In: *International Symposium on Mathematical Foundations of Computer Science*. 2012, pp. 100–111.

[7] James Atwood and Don Towsley. "Diffusion-convolutional neural networks". In: *Advances in Neural Information Processing Systems*. 2016.

[8] Koenraad M R Audenaert. "A sharp continuity estimate for the von Neumann entropy". In: *Journal of Physics A: Mathematical and Theoretical* 40.28 (2007), 8127–8136.

[9] Waiss Azizian and marc lelarge. "Expressive Power of Invariant and Equivariant Graph Neural Networks". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=lxHgXYN4bwl.

[10] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. "Layer Normalization". In: *CoRR* (2016). arXiv: 1607.06450 [stat.ML].

[11] László Babai. "Graph isomorphism in quasipolynomial time". In: *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*. ACM. 2016, pp. 684–697.

[12] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *International Conference on Learning Representations*. 2015.

[13] Lu Bai et al. "Learning Aligned-Spatial Graph Convolutional-Networks for Graph Classification". In: *Proceedings of the Joint*

European Conference on Machine Learning and Knowledge Discovery in Databases. 2019, pp. 464–482.

[14] Lu Bai et al. "Learning Backtrackless Aligned-Spatial Graph Convolutional Networks for Graph Classification". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020).

[15] Yunsheng Bai et al. "Convolutional set matching for graph similarity". In: *arXiv preprint arXiv:1810.10866* (2018).

[16] Yunsheng Bai et al. "Simgnn: A neural network approach to fast graph similarity computation". In: *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. 2019, pp. 384–392.

[17] Ivana Balazevic, Carl Allen, and Timothy M. Hospedales. "Hypernetwork Knowledge Graph Embeddings". In: *CoRR* abs/1808.07018 (2018). arXiv: 1808.07018. URL: http://arxiv.org/abs/1808.07018.

[18] Peter Bartlett, Dave Helmbold, and Phil Long. "Gradient descent with identity initialization efficiently learns positive definite linear transformations". In: *International Conference on Machine Learning*. 2018, pp. 520–529.

[19] Eric B Baum and David Haussler. "What size net gives valid generalization?" In: *Advances in neural information processing systems*. 1989, pp. 81–90.

[20] Dominique Beaini et al. "Directional graph networks". In: *arXiv preprint arXiv:2010.02863* (2020).

[21] Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Representation learning: A review and new perspectives". In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1798–1828.

[22] Dennis S. Bernstein. *Matrix mathematics: Theory, facts, and formulas (second edition)*. Princeton, NJ:Princeton University Press, 2009.

[23] Filippo Maria Bianchi et al. "Graph Neural Networks with Convolutional ARMA Filters". In: *arXiv preprint arXiv:1901.01343* (2019).

[24] N. Biggs, E. K. Lloyd, and R. J. Wilson. *Graph Theory, 1736-1936*. USA: Clarendon Press, 1986. ISBN: 0198539169.

[25] Hans L Bodlaender. "Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees". In: *Journal of Algorithms* 11.4 (1990), pp. 631–643.

[26] Charles Bordenave, Marc Lelarge, and Laurent Massoulié. "Non-backtracking spectrum of random graphs: Community detection and non-regular Ramanujan graphs". In: *IEEE 56th Annual Symposium on Foundations of Computer Science*. IEEE. 2015, pp. 1347–1357.

[27] Karsten M Borgwardt and Hans-Peter Kriegel. "Shortest-path kernels on graphs". In: *Proceedings of the 5th International Conference on Data Mining*. 2005, pp. 74–81.

[28] Karsten M Borgwardt et al. "Protein function prediction via graph kernels". In: *Bioinformatics* 21.suppl_1 (2005), pp. i47–i56.

[29] N. Bourbaki. *General Topology: Chapters 1-4*. Addison-Wesley series in mathematics vol. 4. Springer, 1998. ISBN: 9783540642411.

[30] Samuel L. Braunstein, Sibasish Ghosh, and Simone Severini. "The Laplacian of a Graph as a Density Matrix: A Basic Combinatorial Approach to Separability of Mixed States". In: *Annals of Combinatorics* 10.3 (2006), pp. 291–317.

[31] Michael M. Bronstein et al. "Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges". In: *CoRR* abs/2104.13478 (2021). arXiv: 2104.13478. URL: https://arxiv.org/abs/2104.13478.

[32] Tom Brown et al. "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901.

[33] Richard A Brualdi. "Spectra of digraphs". In: *Linear Algebra and its Applications* (2010), pp. 2181–2213.

[34] Joan Bruna et al. "Spectral Networks and Locally connected networks on Graphs". In: *Proceedings of the International Conference on Learning Representations*. 2014.

[35] Murillo Guimarães Carneiro and Liang Zhao. "Organizational Data Classification Based on the Importance Concept of Complex Networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 29.8 (2017), pp. 3361–3373.

[36] Hong Chang and Dit-Yan Yeung. "Graph Laplacian Kernels for Object Classification from a Single Example". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2006, pp. 2011 –2016.

[37] Michail Chatzianastasis et al. "Operation Embeddings for Neural Architecture Search". In: *CoRR* abs/2105.04885 (2021). arXiv: 2105.04885. URL: https://arxiv.org/abs/2105.04885.

[38] Dexiong Chen, Laurent Jacob, and Julien Mairal. "Convolutional Kernel Networks for Graph-Structured Data". In: *Proceedings of the 37th International Conference on Machine Learning*. 2020, pp. 1576–1586.

[39] Ming Chen et al. "Simple and Deep Graph Convolutional Networks". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 1725–1735.

[40] Pin-Yu Chen et al. "Fast Incremental von Neumann Graph Entropy Computation: Theory, Algorithm, and Applications". In: ICML '19. 2019.

[41] Ting Chen, Song Bian, and Yizhou Sun. "Are Powerful Graph Neural Nets Necessary? A Dissection on Graph Classification".

In: *CoRR* (2019). arXiv: 1905.04579. URL: http://arxiv.org/abs/1905.04579.

[42] Zhengdao Chen et al. "On the equivalence between graph isomorphism testing and function approximation with GNNs". In: *NeurIPS 2019* abs/1905.12560 (2019). URL: http://arxiv.org/abs/1905.12560.

[43] Hayoung Choi et al. *Fast computation of von Neumann entropy for large-scale graphs via quadratic approximations*. 2018. arXiv: 1811.11087 [cs.IT].

[44] Arindam Chowdhury et al. "Efficient Power Allocation Using Graph Neural Networks and Deep Algorithm Unfolding". In: *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2021, pp. 4725–4729. DOI: 10.1109/ICASSP39728.2021.9415106.

[45] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. "On the Relationship between Self-Attention and Convolutional Layers". In: *International Conference on Learning Representations*. 2020.

[46] Gabriele Corso et al. "Principal Neighbourhood Aggregation for Graph Nets". In: *Advances in Neural Information Processing Systems*. Vol. 33. 2020.

[47] Imre Csiszár and Paul C. Shields. "Information Theory and Statistics: A Tutorial". In: *Commun. Inf. Theory* 1.4 (Dec. 2004), 417–528.

[48] Thiago Henrique Cupertino et al. "A Scheme for High Level Data Classification Using Random Walk and Network Measures". In: *Expert Systems with Applications* 92 (2018), pp. 289–303.

[49] Marco Cuturi. "Sinkhorn Distances: Lightspeed Computation of Optimal Transport". In: *Advances in Neural Information Processing Systems* 26 (2013), pp. 2292–2300.

[50] Dragoš Cvetković and Slobodan K. Simić. "Towards a spectral theory of graphs based on the signless Laplacian, II". In: *Linear Algebra and its Applications* (2010), pp. 2257 –2272.

[51] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[52] Lorenzo Dall'Amico, Romain Couillet, and Nicolas Tremblay. "Optimal Laplacian regularization for sparse spectral community detection". In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020, pp. 3237 –3241.

[53] Lorenzo Dall'Amico, Romain Couillet, and Nicolas Tremblay. "Optimized Deformed Laplacian for Spectrum-based Community Detection in Sparse Heterogeneous Graphs". In: *arXiv:1901.09715* (2019).

[54] George Dasoulas, Johannes F. Lutzeyer, and Michalis Vazirgiannis. "Learning Parametrised Graph Shift Operators". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=0OlrLvrsHwQ.

[55] George Dasoulas, Kevin Scaman, and Aladin Virmaux. "Lipschitz normalization for self-attention layers with application to graph neural networks". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 2456–2466. URL: https://proceedings.mlr.press/v139/dasoulas21a.html.

[56] George Dasoulas et al. "Coloring Graph Neural Networks for Node Disambiguation". In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*. Ed. by Christian Bessiere. Main track. International Joint Conferences on Artificial Intelligence Organization, July 2020, pp. 2126–2132. DOI: 10.24963/ijcai.2020/294. URL: https://doi.org/10.24963/ijcai.2020/294.

[57] George Dasoulas et al. "Ego-based Entropy Measures for Structural Representations". In: *CoRR* abs/2003.00553 (2020). arXiv: 2003.00553. URL: https://arxiv.org/abs/2003.00553.

[58] A. Debnath et al. "Structure-Activity Relationship of Mutagenic Aromatic and Heteroaromatic Nitro Compounds. Correlation with Molecular Orbital Energies and Hydrophobicity". In: *Journal of Medicinal Chemistry* 34.2 (1991), pp. 786–797.

[59] Aurelien Decelle et al. "Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications". In: *Physical Review E* (2011), p. 066106.

[60] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. "Convolutional neural networks on graphs with fast localized spectral filtering". In: *Advances in Neural Information Processing Systems*. 2016, pp. 3844–3852.

[61] Austin Derrow-Pinion et al. "ETA Prediction with Graph Neural Networks in Google Maps". In: *CoRR* abs/2108.11482 (2021). arXiv: 2108.11482. URL: https://arxiv.org/abs/2108.11482.

[62] P. Dobson and A. Doig. "Distinguishing Enzyme Structures from Non-enzymes Without Alignments". In: *Journal of Molecular Biology* 330.4 (2003), pp. 771–783.

[63] Claire Donnat et al. "Learning Structural Node Embeddings via Diffusion Wavelets". In: *Proceedings of the 24rd International Conference on Knowledge Discovery and Data Mining*. 2018, pp. 1320–1329.

[64] David K Duvenaud et al. "Convolutional networks on graphs for learning molecular fingerprints". In: *Advances in neural information processing systems*. 2015, pp. 2224–2232.

[65] Vijay Prakash Dwivedi et al. "Benchmarking Graph Neural Networks". In: *arXiv:2003.00982* (2020).

[66] Mark Eisen and Alejandro Ribeiro. "Large Scale Wireless Power Allocation with Graph Neural Networks". In: *2019 IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. 2019, pp. 1–5. DOI: 10.1109/SPAWC.2019.8815526.

[67] P. Erdös and A. Rényi. "On Random Graphs I". In: *Publicationes Mathematicae Debrecen* 6 (1959), p. 290.

[68] Federico Errica et al. "A fair comparison of graph neural networks for graph classification". In: *8th International Conference on Learning Representations*. 2020.

[69] Herbert Federer. *Geometric measure theory*. eng. Grundlehren der mathematischen Wissenschaften (Springer) ; 153. Berlin: Springer-Verlag Berlin Heidelberg, 1996.

[70] Matthias Fey and Jan Eric Lenssen. "Fast Graph Representation Learning with PyTorch Geometric". In: *arXiv preprint arXiv:1903.02428* (2019).

[71] Paolo Frasconi, Marco Gori, and Alessandro Sperduti. "A general framework for adaptive processing of data structures". In: *IEEE transactions on Neural Networks* 9.5 (1998), pp. 768–786.

[72] Alexis Galland and Marc Lelarge. "Invariant embedding for graph classification". In: *ICML 2019 Workshop on Learning and Reasoning with Graph-Structured Data*. Long Beach, United States, June 2019. URL: https://hal.archives-ouvertes.fr/hal-02947290.

[73] Claudio Gallicchio and Alessio Micheli. "Fast and Deep Graph Neural Networks". In: *Proceedings of the 34th AAAI Conference on Artificial Intelligence*. 2020, pp. 3898–3905.

[74] Fernando Gama, Alejandro Ribeiro, and Joan Bruna. "Stability of Graph Neural Networks to Relative Perturbations". In: *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE, 2020, pp. 9070 –9074.

[75] Feng Gao, Guy Wolf, and Matthew Hirn. "Geometric Scattering for Graph Data Analysis". In: *Proceedings of the 36th International Conference on Machine Learning*. 2019, pp. 2122–2131.

[76] Jonas Gehring et al. "A Convolutional Encoder Model for Neural Machine Translation". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2017, pp. 123–135.

[77] Stuart Geman, Elie Bienenstock, and René Doursat. "Neural networks and the bias/variance dilemma". In: *Neural computation* 4.1 (1992), pp. 1–58.

[78] C. L. Giles, K. D. Bollacker, and S. Lawrence. "CiteSeer: an automatic citation indexing system". English (US). In: *Proceedings of the ACM International Conference on Digital Libraries*. 1998, pp. 89–98.

[79] C. Lee Giles, Kurt D. Bollacker, and Steve Lawrence. "Citeseer: an automatic citation indexing system". In: *International Conference on Digital Libraries*. ACM Press, 1998, pp. 89 –98.

[80] Justin Gilmer et al. "Neural Message Passing for Quantum Chemistry". In: *Proceedings of the 34th International Conference on Machine Learning*. 2017, pp. 1263–1272.

[81] Vladimir Gligorijević et al. "Structure-based protein function prediction using graph convolutional networks". In: *Nature Communications* 12.1 (2021), p. 3168. ISSN: 2041-1723. DOI: 10.1038/s41467-021-23303-9. URL: https://doi.org/10.1038/s41467-021-23303-9.

[82] Oded Goldreich. *Introduction to property testing*. Cambridge University Press, 2017.

[83] Marco Gori, Gabriele Monfardini, and Franco Scarselli. "A New Model for Learning in Graph Domains". In: *Proceedings of the 2005 IEEE International Joint Conference on Neural Networks*. Vol. 2. 2005, pp. 729–734.

[84] Henry Gouk et al. "Regularisation of Neural Networks by Enforcing Lipschitz Continuity". English. In: *Machine Learning* (Dec. 2020).

[85] Martin Grohe, Gaurav Rattan, and Gerhard J Woeginger. "Graph Similarity and Approximate Isomorphism". In: *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science*. 2018.

[86] Aditya Grover and Jure Leskovec. "node2vec: Scalable feature learning for networks". In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2016, pp. 855–864.

[87] Krystal Guo and Bojan Mohar. "Hermitian adjacency matrix of digraphs and mixed graphs". In: *Journal of Graph Theory* (2017), pp. 217–248.

[88] Nicholas Guttenberg et al. "Permutation-equivariant neural networks applied to dynamics prediction". In: *arXiv preprint arXiv:1612.04530* (2016).

[89] Frank Hall, Kinnari Patel, and Michael Stewart. "Interlacing results on matrices associated with graphs". In: *Journal of Combinatorial Mathematics and Combinatorial Computing* (2009), pp. 113–127.

[90] William L. Hamilton, Rex Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Long Beach, California, USA, 2017, 1025–1035.

[91] William L. Hamilton, Rex Ying, and Jure Leskovec. "Representation Learning on Graphs: Methods and Applications". In: *CoRR* (2017).

[92] William L. Hamilton, Rex Ying, and Jure Leskovec. "Representation Learning on Graphs: Methods and Applications". In: *IEEE Data Eng. Bull.* 40.3 (2017), pp. 52–74.

[93] Arman Hasanzadeh et al. "Semi-Implicit Graph Variational Auto-Encoders". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/file/fd4771e85e1f916f239624486bff502d-Paper.pdf.

[94] Mikael Henaff, Joan Bruna, and Yann LeCun. "Deep convolutional networks on graph-structured data". In: *arXiv preprint arXiv:1506.05163* (2015).

[95] Keith Henderson et al. "RolX: Structural Role Extraction & Mining in Large Graphs". In: *Proceedings of the 18th International Conference on Knowledge Discovery and Data Mining*. 2012, pp. 1231–1239.

[96] Keith Henderson et al. "RolX: structural role extraction & mining in large graphs". In: *KDD*. 2012.

[97] Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. "Stochastic blockmodels: First steps". In: *Social networks* (1983), pp. 109 –137.

[98] Masanobu Horie et al. "Isometric Transformation Invariant and Equivariant Graph Convolutional Networks". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=FX0vR39SJ5q.

[99] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge, UK: Cambridge University Press, 1985.

[100] Kurt Hornik. "Approximation Capabilities of Multilayer Feedforward Networks". In: *Neural networks Journal* 4.2 (1991), pp. 251–257.

[101] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer Feedforward Networks are Universal Approximators". In: *Neural networks* 2.5 (1989), pp. 359–366.

[102] Weihua Hu et al. "Open Graph Benchmark: Datasets for Machine Learning on Graphs". In: *CoRR* (2020). arXiv: 2005.00687 [cs.LG].

[103] Weihua Hu et al. "Open Graph Benchmark: Datasets for Machine Learning on Graphs". In: *arXiv preprint arXiv:2005.00687* (2020).

[104] Vassilis N. Ioannidis, Antonio G. Marques, and Georgios B. Giannakis. "Graph Neural Networks for Predicting Protein Functions". In: *2019 IEEE 8th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*. 2019, pp. 221–225. DOI: 10.1109/CAMSAP45676.2019.9022646.

[105] Sergey Ivanov and Evgeny Burnaev. "Anonymous Walk Embeddings". In: *ICML* (2018).

[106] Matthew O. Jackson. *Social and Economic Networks*. USA: Princeton University Press, 2008. ISBN: 0691134405.

[107] Bo Jiang et al. "Graph Matching via Multiplicative Update Algorithm". In: *Advances in Neural Information Processing Systems*. 2017, pp. 3190–3198.

[108] Weiwei Jiang. "Graph-based Deep Learning for Communication Networks: A Survey". In: *CoRR* abs/2106.02533 (2021). arXiv: 2106.02533. URL: https://arxiv.org/abs/2106.02533.

[109] Weiwei Jiang and Jiayun Luo. "Graph Neural Network for Traffic Forecasting: A Survey". In: *CoRR* abs/2101.11174 (2021). arXiv: 2101.11174. URL: https://arxiv.org/abs/2101.11174.

[110] John Jumper et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596.7873 (2021), pp. 583–589. ISSN: 1476-4687. DOI: 10.1038/s41586-021-03819-2. URL: https://doi.org/10.1038/s41586-021-03819-2.

[111] Brian Karrer and Mark EJ Newman. "Stochastic blockmodels and community structure in networks". In: *Physical review E* (2011), p. 016107.

[112] Steven Kearnes et al. "Molecular graph convolutions: moving beyond fingerprints". In: *Journal of Computer-Aided Molecular Design* 30.8 (2016), 595–608. ISSN: 1573-4951. DOI: 10.1007/s10822-016-9938-8. URL: http://dx.doi.org/10.1007/s10822-016-9938-8.

[113] Kristian Kersting et al. *Benchmark Data Sets for Graph Kernels*. http://graphkernels.cs.tu-dortmund.de. 2016.

[114] Choongrak Kim et al. "A simple and exact Laplacian clustering of complex networking phenomena: Application to gene expression profiles". In: *Proceedings of the National Academy of Sciences* (2008), pp. 4083 –4087.

[115] Hyunjik Kim, George Papamakarios, and Andriy Mnih. "The Lipschitz Constant of Self-Attention". In: *CoRR* (2020). arXiv: 2006.04710 [stat.ML].

[116] Jongmin Kim et al. "Edge-Labeling Graph Neural Network for Few-Shot Learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[117] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations (ICLR)*. 2015.

[118] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *Proceedings of the 5th International Conference on Learning Representations* (ICLR). Palais des Congrès Neptune, Toulon, France, 2017.

[119] Thomas N. Kipf and Max Welling. "Variational Graph Auto-Encoders". In: *CoRR* (2016). arXiv: 1611.07308 [stat.ML].

[120] Johannes Klicpera, Florian Becker, and Stephan Günnemann. "GemNet: Universal Directional Graph Neural Networks for Molecules". In: *Advances in Neural Information Processing Systems*. Ed. by A. Beygelzimer et al. 2021. URL: https://openreview.net/forum?id=HS_sOaxS9K-.

[121] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. "Diffusion Improves Graph Learning". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 13354 –13366.

[122] Nils M Kriege, Pierre-Louis Giscard, and Richard Wilson. "On valid optimal assignment kernels and applications to graph classification". In: *Advances in Neural Information Processing Systems*. 2016, pp. 1623–1631.

[123] Nils M. Kriege et al. "A Property Testing Framework for the Theoretical Expressivity of Graph Kernels". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, July 2018, pp. 2348–2354.

[124] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. 2012, pp. 1097–1105.

[125] Florent Krzakala et al. "Spectral redemption in clustering sparse networks". In: *Proceedings of the National Academy of Sciences* (2013), pp. 20935 –20940.

[126] Sofia Ira Ktena et al. "Distance Metric Learning using Graph Convolutional Networks: Application to Functional Brain Networks". In: *CoRR* abs/1703.02161 (2017). arXiv: 1703.02161. URL: http://arxiv.org/abs/1703.02161.

[127] Tuan Le et al. "Parameterized Hypercomplex Graph Neural Networks for Graph Classification". In: *arXiv preprint arXiv:2103.16584* (2021).

[128] John Boaz Lee, Ryan Rossi, and Xiangnan Kong. "Graph Classification Using Structural Attention". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2018, 1666–1674.

[129] John Boaz Lee et al. "Graph Convolutional Networks with Motif-based Attention". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2019, pp. 499–508.

[130] Tao Lei et al. "Deriving Neural Architectures from Sequence and Graph Kernels". In: *Proceedings of the 34th International Conference on Machine Learning*. 2017, pp. 2024–2033.

[131] A. Li and Y. Pan. "Structural Information and Dynamical Complexity of Networks". In: *IEEE Transactions on Information Theory* 62.6 (2016), pp. 3290–3339. ISSN: 1557-9654. DOI: 10.1109/TIT.2016.2555904.

[132] Guohao Li et al. "DeeperGCN: All You Need to Train Deeper GCNs". In: *CoRR* (2020). arXiv: 2006.07739 [cs.LG].

[133] Guohao Li et al. "DeepGCNs: Can GCNs Go as Deep as CNNs?" In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 9267–9276.

[134] Jian Li et al. "Neural Architecture Optimization with Graph VAE". In: *CoRR* abs/2006.10310 (2020). arXiv: 2006.10310. URL: https://arxiv.org/abs/2006.10310.

[135] Michelle M. Li, Kexin Huang, and Marinka Zitnik. "Representation Learning for Networks in Biology and Medicine: Advancements, Challenges, and Opportunities". In: *CoRR* abs/2104.04883 (2021). arXiv: 2104.04883. URL: https://arxiv.org/abs/2104.04883.

[136] Qimai Li, Zhichao Han, and Xiao-Ming Wu. "Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[137] Yuemeng Li, Xintao Wu, and Aidong Lu. "Analysis of spectral space properties of directed graphs using matrix perturbation theory with application in graph partition". In: *IEEE International Conference on Data Mining*. IEEE. 2015, pp. 847–852.

[138] Yujia Li et al. "Gated Graph Sequence Neural Networks". In: *arXiv preprint arXiv:1511.05493* (2015).

[139] Yujia Li et al. "Gated Graph Sequence Neural Networks". In: *International Conference on Learning Representations*. 2016.

[140] Stratis Limnios et al. "Hcore-Init: Neural Network Initialization based on Graph Degeneracy". In: *2020 25th International Conference on Pattern Recognition (ICPR)*. 2021, pp. 5852–5858. DOI: 10.1109/ICPR48806.2021.9412940.

[141] Meng Liu, Hongyang Gao, and Shuiwang Ji. "Towards Deeper Graph Neural Networks". In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Association for Computing Machinery, 2020, pp. 338 –348.

[142] Xianggen Liu et al. "Deep geometric representations for modeling effects of mutations on protein-protein binding affinity". In: *PLOS Computational Biology* 17.8 (2021). Ed. by Roland L.Editor Dunbrack, e1009284. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1009284. URL: http://dx.doi.org/10.1371/journal.pcbi.1009284.

[143] Ziqi Liu et al. "GeniePath: Graph Neural Networks with Adaptive Receptive Paths". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01 (2019), pp. 4424–4431.

[144] Andreas Loukas. "What graph neural networks cannot learn: depth vs width". In: *International Conference on Learning Representations*. 2020.

[145] Eugene M. Luks. "Isomorphism of graphs of bounded valence can be tested in polynomial time". In: *Journal of Computer and System Sciences* 25.1 (1982), pp. 42 –65.

[146] Thang Luong, Hieu Pham, and Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Sept. 2015, pp. 1412–1421.

[147] Ulrike von Luxburg. "A Tutorial on Spectral Clustering". In: *Statistics and Computing* (2007), pp. 395 –416.

[148] Tengfei Ma et al. "Drug Similarity Integration Through Attentive Multi-view Graph Auto-Encoders". In: *CoRR* abs/1804.10850 (2018). arXiv: 1804.10850. URL: http://arxiv.org/abs/1804.10850.

[149] Haggai Maron et al. "Invariant and Equivariant Graph Networks". In: *International Conference on Learning Representations*. 2019.

[150] Haggai Maron et al. "Provably powerful graph networks". In: *Advances in Neural Information Processing Systems*. 2019, pp. 2156 –2167.

[151] Gonzalo Mateos et al. "Connecting the dots: Identifying network structure via graph signal processing". In: *IEEE Signal Processing Magazine* (2019), pp. 16 –43.

[152] Matthew K. Matlock et al. "Deep learning long-range information in undirected graphs with wave networks". In: *CoRR* abs/1810.12153 (2018). arXiv: 1810.12153.

[153] Andrew Kachites McCallum et al. "Automating the construction of internet portals with machine learning". In: *Information Retrieval* (2000), pp. 127 –163.

[154] Brendan D McKay. "Practical graph isomorphism". In: *Congressus Numerantium* 30 (1981), pp. 45–87.

[155] Giorgia Minello, Luca Rossi, and Andrea Torsello. "On the von Neumann entropy of graphs". In: *Journal of Complex Networks* (Nov. 2018). ISSN: 2051-1329.

[156] R. V. Mises and H. Pollaczek-Geiringer. "Praktische Verfahren der Gleichungsauflösung ." In: *ZAMM* 9.2 (1929), pp. 152–164.

[157] Takeru Miyato et al. "Spectral Normalization for Generative Adversarial Networks". In: *International Conference on Learning Representations*. 2018.

[158] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 00280836. URL: http://dx.doi.org/10.1038/nature14236.

[159] Christopher Morris, Gaurav Rattan, and Petra Mutzel. "Weisfeiler and Leman go sparse: Towards scalable higher-order graph embeddings". In: *Advances in Neural Information Processing Systems* (2020), pp. 21824–21840.

[160] Christopher Morris et al. "TUDataset: A collection of benchmark datasets for learningwith graphs". In: *arXiv preprint arXiv:2007.08663* (2020).

[161] Christopher Morris et al. "Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2019, pp. 4602–4609.

[162] James Munkres. "Algorithms for the assignment and transportation problems". In: *Journal of the society for industrial and applied mathematics* 5.1 (1957), pp. 32–38.

[163] Ryan Murphy et al. "Relational Pooling for Graph Representations". In: *Proceedings of the 36th International Conference on Machine Learning*. 2019, pp. 4663–4673.

[164] Ryan Murphy et al. "Relational Pooling for Graph Representations". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. Long

Beach, California, USA: PMLR, 2019, pp. 4663–4673. URL: http://proceedings.mlr.press/v97/murphy19a.html.

[165] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. "Learning Convolutional Neural Networks for Graphs". In: *Proceedings of the 33rd International Conference on Machine Learning*. 2016, pp. 2014–2023.

[166] Giannis Nikolentzos, George Dasoulas, and Michalis Vazirgiannis. "k-hop graph neural networks". In: *Neural Networks* 130 (2020), pp. 195–205. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2020.07.008. URL: https://www.sciencedirect.com/science/article/pii/S0893608020302495.

[167] Giannis Nikolentzos, George Dasoulas, and Michalis Vazirgiannis. *Permute Me Softly: Learning Soft Permutations for Graph Representations*. 2021. arXiv: 2110.01872 [stat.ML].

[168] Giannis Nikolentzos, Giannis Siglidis, and Michalis Vazirgiannis. "Graph Kernels: A Survey". In: *arXiv preprint arXiv:1904.12218* (2019).

[169] Giannis Nikolentzos and Michalis Vazirgiannis. "Random Walk Graph Neural Networks". In: *Advances in Neural Information Processing Systems*. 2020, pp. 16211–16222.

[170] Antonio Ortega et al. *Graph Signal Processing: Overview, Challenges and Applications*. 2018. arXiv: 1712.00468 [eess.SP].

[171] Xiaoyong Pan and Hong-Bin Shen. "Inferring Disease-Associated MicroRNAs Using Semi-supervised Multi-Label Graph Convolutional Networks". In: *iScience* 20 (2019), pp. 265–277. ISSN: 2589-0042. DOI: https://doi.org/10.1016/j.isci.2019.09.013. URL: https://www.sciencedirect.com/science/article/pii/S2589004219303517.

[172] Aldo Pareja et al. "EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs". In: *CoRR* abs/1902.10191 (2019). arXiv: 1902.10191. URL: http://arxiv.org/abs/1902.10191.

[173] Filippo Passerini and Simone Severini. "Quantifying Complexity in Networks: The von Neumann Entropy". In: *IJATS* 1 (2009), pp. 58–67.

[174] Hongbin Pei et al. "Geom-GCN: Geometric Graph Convolutional Networks". In: *8th International Conference on Learning Representations (ICLR)*. 2020.

[175] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. "Deepwalk: Online learning of social representations". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2014, pp. 701–710.

[176] Gabriel Peyré, Marco Cuturi, et al. "Computational Optimal Transport with Applications to Data Sciences". In: *Foundations and Trends® in Machine Learning* 11.5-6 (2019), pp. 355–607.

[177] Tobias Pfaff et al. "Learning Mesh-Based Simulation with Graph Networks". In: *9th International Conference on Learning Representations (ICLR)*. 2021.

[178] Allan Pinkus. "Approximation theory of the MLP model in neural networks". In: *Acta numerica* 8 (1999), pp. 143–195.

[179] Guilherme Porto and Luiz Emilio Allem. "Eigenvalue interlacing in graphs". In: *Proceeding Series of the Brazilian Society of Applied and Computational Mathematics* (2017), p. 010232.

[180] Jorge Pérez, Javier Marinković, and Pablo Barceló. "On the Turing Completeness of Modern Neural Network Architectures". In: *Proceedings of the 7th International Conference on Learning Representations (ICLR)*. 2019.

[181] Charles Ruizhongtai Qi et al. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". In: *CVPR* (2017). arXiv: 1612.00593.

[182] Tai Qin and Karl Rohe. "Regularized spectral clustering under the degree-corrected stochastic blockmodel". In: *Advances in neural information processing systems (NIPS)*. 2013, pp. 3120 –3128.

[183] Alec Radford et al. "Language Models are Unsupervised Multitask Learners". In: (2018).

[184] Raghunathan Ramakrishnan et al. "Quantum chemistry structures and properties of 134 kilo molecules". In: *Scientific Data* 1.1 (2014), pp. 1–7.

[185] Jiahua Rao et al. "Imputing single-cell RNA-seq data by combining graph convolution and autoencoder neural networks". In: *iScience Journal* 24.5 (2021), p. 102393. ISSN: 2589-0042. DOI: https://doi.org/10.1016/j.isci.2021.102393. URL: https://www.sciencedirect.com/science/article/pii/S2589004221003618.

[186] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. "struc2vec: Learning Node Representations from Structural Identity". In: *Proceedings of the 23rd International Conference on Knowledge Discovery and Data Mining*. 2017, pp. 385–394.

[187] Yu Rong et al. "The Truly Deep Graph Convolutional Networks for Node Classification". In: *CoRR* (2019). arXiv: 1907.10903.

[188] Emanuele Rossi et al. "ncRNA Classification with Graph Convolutional Networks". In: *CoRR* (2019). arXiv: 1905.06515 [q-bio.GN].

[189] Emanuele Rossi et al. "Temporal Graph Networks for Deep Learning on Dynamic Graphs". In: *CoRR* abs/2006.10637 (2020). arXiv: 2006.10637. URL: https://arxiv.org/abs/2006.10637.

[190] Walter Rudin. *Real and Complex Analysis, 3rd Ed.* New York, NY, USA: McGraw-Hill, Inc., 1987. ISBN: 0070542341.

[191] Alaa Saade, Florent Krzakala, and Lenka Zdeborová. "Spectral clustering of graphs with the bethe hessian". In: *Advances in Neural Information Processing Systems (NIPS)*. 2014, pp. 406 –414.

[192] Guillaume Salha, Romain Hennequin, and Michalis Vazirgiannis. "Keep It Simple: Graph Autoencoders Without Graph Convolutional Networks". In: *CoRR* abs/1910.00942 (2019). arXiv: 1910.00942. URL: http://arxiv.org/abs/1910.00942.

[193] Guillaume Salha, Romain Hennequin, and Michalis Vazirgiannis. "Simple and Effective Graph Autoencoders with One-Hop Linear Models". In: *CoRR* abs/2001.07614 (2020). arXiv: 2001.07614. URL: https://arxiv.org/abs/2001.07614.

[194] Alvaro Sanchez-Gonzalez et al. "Learning to Simulate Complex Physics with Graph Networks". In: *CoRR* abs/2002.09405 (2020). arXiv: 2002.09405. URL: https://arxiv.org/abs/2002.09405.

[195] Aliaksei Sandryhaila and José M. F. Moura. "Discrete Signal Processing on Graphs". In: *IEEE Transactions on Signal Processing* (2013), pp. 1644 –1656.

[196] Rodrigo Santa Cruz et al. "Visual Permutation Learning". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41.12 (2018), pp. 3100–3114.

[197] Paul-Edouard Sarlin et al. "SuperGlue: Learning Feature Matching with Graph Neural Networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 4938–4947.

[198] Andrew M Saxe, James L McClelland, and Surya Ganguli. "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks". In: *ICLR* (2014).

[199] Franco Scarselli et al. "Computational Capabilities of Graph Neural Networks". In: *IEEE Transactions on Neural Networks* 20.1 (2009), pp. 81–102. DOI: 10.1109/TNN.2008.2005141.

[200] Franco Scarselli et al. "The graph neural network model". In: *IEEE Transactions on Neural Networks* 20.1 (2009), pp. 61–80.

[201] Michael Schlichtkrull et al. "Modeling Relational Data with Graph Convolutional Networks". In: *The Semantic Web - 15th International Conference (ESWC)*. Springer/Verlag, 2018, pp. 593 –607.

[202] Sam Scott and Stan Matwin. "Feature Engineering for Text Classification". In: *Proceedings of the Sixteenth International Conference on Machine Learning*. ICML '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, 379–388. ISBN: 1558606122.

[203] Yan Shao et al. "Graph Attention Network-based DRL for Network Slicing Management in Dense Cellular Networks". In: *2021 IEEE Wireless Communications and Networking Conference (WCNC)*. 2021, pp. 1–6. DOI: 10.1109/WCNC49053.2021.9417321.

[204] Nino Shervashidze et al. "Efficient Graphlet Kernels for Large Graph Comparison". In: *Proceedings of the International Conference on Artificial Intelligence and Statistics*. 2009, pp. 488–495.

[205] Nino Shervashidze et al. "Weisfeiler-Lehman Graph Kernels". In: *The Journal of Machine Learning Research* 12 (2011), pp. 2539–2561.

[206] Yunsheng Shi et al. "Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification". In: *CoRR* (2020). arXiv: 2009.03509 [cs.LG].

[207] Jonathan Shlomi, Peter Battaglia, and Jean-Roch Vlimant. "Graph neural networks in particle physics". In: *Machine Learning: Science*

and Technology 2.2 (2021), p. 021001. ISSN: 2632-2153. DOI: 10.1088/2632-2153/abbf9a. URL: http://dx.doi.org/10.1088/2632-2153/abbf9a.

[208] D. I. Shuman et al. "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains". In: *IEEE Signal Processing Magazine* (2013), pp. 83 –98.

[209] Giannis Siglidis et al. "GraKeL: A Graph Kernel Library in Python". In: *arXiv preprint arXiv:1806.02193* (2018).

[210] Thiago Christiano Silva and Liang Zhao. "Network-Based High Level Data Classification". In: *IEEE Transactions on Neural Networks and Learning Systems* 23.6 (2012), pp. 954–970.

[211] Martin Simonovsky and Nikos Komodakis. "Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 3693–3702.

[212] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[213] Alessandro Sperduti and Antonina Starita. "Supervised neural networks for the classification of structures". In: *IEEE Transactions on Neural Networks* 8.3 (1997), pp. 714–735.

[214] Alexey Strokach et al. "Fast and Flexible Protein Design Using Deep Graph Neural Networks". In: *Cell Systems* 11.4 (2020), 402–411.e4. ISSN: 2405-4712. DOI: https://doi.org/10.1016/j.cels.2020.08.016. URL: https://www.sciencedirect.com/science/article/pii/S2405471220303276.

[215] Felipe Petroski Such et al. "Robust Spatial Filtering With Graph Convolutional Neural Networks". In: *IEEE Journal of Selected Topics in Signal Processing* 11.6 (2017), pp. 884–896.

[216] Christian Szegedy et al. "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[217] Jan Toenshoff et al. "Graph Learning with 1D Convolutions on Random Walks". In: *arXiv preprint arXiv:2102.08786* (2021).

[218] Ke Tu et al. "Deep Recursive Network Embedding with Regular Equivalence". In: KDD '18. 2018.

[219] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017, pp. 5998–6008.

[220] Petar Veličković et al. "Graph Attention Networks". In: *Proceedings of the 6th International Conference on Learning Representations (ICLR)*. 2018.

[221] Saurabh Verma and Zhi-Li Zhang. "Graph Capsule Convolutional Neural Networks". In: *ICLR* (2019).

[222] Aladin Virmaux and Kevin Scaman. "Lipschitz regularity of deep neural networks: analysis and efficient estimation". In: *Advances*

in Neural Information Processing Systems. Vol. 31. 2018, pp. 3835–3844.

[223] S. V. N. Vishwanathan et al. "Graph Kernels". In: *CoRR* abs/0807.0093 (2008). arXiv: 0807.0093. URL: http://arxiv.org/abs/0807.0093.

[224] N. Wale, I. Watson, and G. Karypis. "Comparison of descriptor spaces for chemical compound retrieval and classification". In: *Knowledge and Information Systems* 14.3 (2008), pp. 347–375.

[225] Matthew J. P. Walker et al. "Isometric Graph Neural Networks". In: *CoRR* abs/2006.09554 (2020). arXiv: 2006.09554. URL: https://arxiv.org/abs/2006.09554.

[226] Haozhe Wang et al. "A Graph Neural Network-Based Digital Twin for Network Slicing Management". In: *IEEE Transactions on Industrial Informatics* 18.2 (2022), pp. 1367–1376. DOI: 10.1109/TII.2020.3047843.

[227] Runzhong Wang, Junchi Yan, and Xiaokang Yang. "Learning Combinatorial Embedding Networks for Deep Graph Matching". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 3056–3065.

[228] Michael D. Ward, Katherine Stovel, and Audrey Sacks. "Network Analysis and Political Science". In: *Annual Review of Political Science* 14.1 (2011), pp. 245–264.

[229] Boris Weisfeiler and AA Lehman. "A reduction of a graph to a canonical form and an algebra arising during this reduction". In: *Nauchno-Technicheskaya Informatsia* 2.9 (1968), pp. 12–16.

[230] Felix Wu et al. "Simplifying Graph Convolutional Networks". In: *7th International Conference on Machine Learning (ICLR)*. 2019.

[231] Shiwen Wu et al. "Graph Neural Networks in Recommender Systems: A Survey". In: *CoRR* abs/2011.02260 (2020). arXiv: 2011.02260. URL: https://arxiv.org/abs/2011.02260.

[232] Zhenqin Wu et al. "MoleculeNet: a benchmark for molecular machine learning". In: *Chemical Science* (2018), pp. 513 –530.

[233] Zonghan Wu et al. "A comprehensive survey on graph neural networks". In: *IEEE Transactions on Neural Networks and Learning Systems* (2020), pp. 1 –21.

[234] Zonghan Wu et al. "Graph Wavenet for Deep Spatial-Temporal Graph Modeling". In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. IJCAI'19. Macao, China, 2019, 1907–1913.

[235] Saining Xie et al. "Exploring Randomly Wired Neural Networks for Image Recognition". In: *CoRR* abs/1904.01569 (2019). arXiv: 1904.01569. URL: http://arxiv.org/abs/1904.01569.

[236] Zhang Xinyi and Lihui Chen. "Capsule Graph Neural Network". In: *Proceedings of the 7th International Conference on Learning Representations*. 2019.

[237] Kelvin Xu et al. "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention". In: *Proceedings of the 32nd International Conference on Machine Learning*. Vol. 37. 2015, pp. 2048–2057.

[238] Keyulu Xu et al. "How Powerful are Graph Neural Networks?" In: *7th International Conference on Learning Representations (ICLR)*. 2019.

[239] P. Yanardag and S. Vishwanathan. "Deep Graph Kernels". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2015, pp. 1365–1374.

[240] Shuang-Hong Yang et al. "Like like alike - Joint Friendship and Interest Propagation in Social Networks". In: *Proceedings of the 20th International Conference on World Wide Web*. ACM. 2011, pp. 537–546.

[241] Zichao Yang et al. "Hierarchical Attention Networks for Document Classification". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. June 2016, pp. 1480–1489.

[242] Heng Yao, Jihong Guan, and Tianying Liu. "Denoising Protein–Protein interaction network via variational graph autoencoder for protein complex detection". In: *Journal of Bioinformatics and Computational Biology* 18.03 (2020). PMID: 32698725, p. 2040010. DOI: 10.1142/S0219720020400107. eprint: https://doi.org/10.1142/S0219720020400107. URL: https://doi.org/10.1142/S0219720020400107.

[243] Haochao Ying et al. "Sequential Recommender System based on Hierarchical Attention Networks". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. Main track. 2018, pp. 3926–3932.

[244] Zhitao Ying et al. "Hierarchical Graph Representation Learning with Differentiable Pooling". In: *Advances in Neural Information Processing Systems*. 2018, pp. 4801–4811.

[245] Jiaxuan You, Rex Ying, and Jure Leskovec. "Position-aware Graph Neural Networks". In: Proceedings of Machine Learning Research. 2019, pp. 7134 –7143.

[246] Seongjun Yun et al. "Graph Transformer Networks". In: *Advances in Neural Information Processing Systems*. Vol. 32. 2019, pp. 11983–11993.

[247] Manzil Zaheer et al. "Deep Sets". In: *Advances in Neural Information Processing Systems*. 2017.

[248] Muhan Zhang and Yixin Chen. "Link Prediction Based on Graph Neural Networks". In: *Advances in Neural Information Processing Systems 31*. 2018, pp. 5165–5175.

[249] Muhan Zhang et al. "An End-to-End Deep Learning Architecture for Graph Classification". In: *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*. 2018, pp. 4438–4445.

[250] Muhan Zhang et al. "D-VAE: A Variational Autoencoder for Directed Acyclic Graphs". In: *CoRR* abs/1904.11088 (2019). arXiv: 1904.11088. URL: http://arxiv.org/abs/1904.11088.

[251] Xiao-Meng Zhang et al. "Graph Neural Networks and Their Current Applications in Bioinformatics". In: *Frontiers in Genetics* 12 (2021), p. 1073. ISSN: 1664-8021. DOI: 10.3389/fgene.2021.690049. URL: https://www.frontiersin.org/article/10.3389/fgene.2021.690049.

[252] Lingxiao Zhao and Leman Akoglu. "PairNorm: Tackling Oversmoothing in GNNs". In: *International Conference on Learning Representations*. 2020.

[253] Zhou Zhao et al. "Link Prediction via Ranking Metric Dual-Level Attention Network Learning". In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. Main track. 2017, pp. 3525–3531.

[254] Alice Zheng and Amanda Casari. *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*. 1st. O'Reilly Media, Inc., 2018. ISBN: 1491953241.

Appendices

Appendix A

Datasets

A.1 Graph Classification Benchmarks

In this section, we present the details and statitics of the benchmark graph classification datasets. Table A.1 summarized the graph property characteristics of the datasets and, next, we provide complementary information on these datasets.

Social Network Datasets (IMDB-binary, IMDB-multi): These datasets refer to collaborations between actors/actresses, where each graph is an ego-graph of every actor and the edges occur when the connected nodes/actors are playing in the same movie. The task is to classify the genre of the movie that the graph derives from. IMDBb is a single-class classification dataset, while IMDBm is multi-class. For both social network datasets, we used one-hot encodings of node degrees as node attribute vectors.

Bio-informatics Datasets (MUTAG, PROTEINS, PTC-MR): MUTAG consists of mutagenic aromatic and heteroaromatic nitrocompounds with 7 discrete labels. PROTEINS consists of nodes, which correspond to secondary structureelements and the edges occur when the connected nodes are neighbors in the amino-acidsequence or in 3D space. It has 3 discrete labels. PTC consists of chemical compounds that reports the carcinogenicity for male and female rats and it has 19 discrete labels. For all bio-informatics datasets we used the node labels as node attribute vectors.

OGBG-MOLHIV: This dataset is a collection of graphs, that represent molecular networks [232]. Nodes are atoms and the edges correspond to chemical bonds between atoms. The graphs contain node features, that are processed as in Hu et al. [102].

A.2 Node Classification Benchmarks

In the next section, we similarly present the details of the benchmark node classification datasets. Table A.2 introduces the graph property

TABLE A.1: Graph Classification Datasets Statistics

Dataset	# graphs	# avg.$\|V\|$/graph	# avg.$\|E\|$/graph	# classes
MUTAG	188	17.93	19.81	2
PROTEINS	1113	39.06	72.65	2
PTC-MR	344	14.29	14.32	2
IMDB-BINARY	1000	19.77	96.52	2
IMDB-MULTI	1500	13.00	76.34	3
OGBG-MOLHIV	41,127	25.5	27.50	2

characteristics of the datasets and, next, we provide complementary information on these datasets.

Cora and CiteSeer are citation networks [118], where nodes correspond to documents and edges encode citation links. Both datasets contain node attributes, that are sparse bag-of-words representations of the documents.

Ogbn-arxiv is a citation network with directed edges, where each node corresponds to an arXiv paper and the edges denote citations from one paper to another [102]. The dataset contains node attributes, that are averaged word embeddings of the titles and the abstracts of dimensionality 128. The label of each node is the subject area of the paper and can take 40 values.

TABLE A.2: Node Clasification Datasets Statistics

Dataset	# graphs	# avg.$\|V\|$/graph	# avg.$\|E\|$/graph	# classes
Cora	1	2,708	5,429	2
CiteSeer	1	3,327	4,732	2
Ogbn-arxiv	1	169,343	1,166,243	40

Appendix B

Propositions and Proofs

B.1 Group action on Hausdorff spaces

In what follows, $\mathcal{X}$ is always a topological set and G a group of transformations acting on $\mathcal{X}$. The orbits of $\mathcal{X}$ under the action of G are the sets $Gx = \{g \cdot x : g \in G\}$. Moreover, we denote as $\mathcal{X}/G$ the quotient space of orbits, also defined by the equivalence relation: $x \sim y \iff \exists g \in G$ s.t. $x = g \cdot y$. As stated in Section 3.5, graphs with node attributes can be defined using invariance by permutation of the labels. We prove here that the resulting spaces are Hausdorff.

Definition 9 (Group invariance)**.** Let G a group, a function $f : \mathcal{X} \to \mathcal{Y}$ is *G-invariant* if

$$\forall x \in \mathcal{X}, \forall g \in G, f(x) = f(g \cdot x)\,. \tag{B.1}$$

Lemma 10 ([29, I, §8.3])**.** *Let $\mathcal{X}$ be a Hausdorff space and $\mathcal{R}$ an equivalence relation of $\mathcal{X}$. Then $\mathcal{X}/\mathcal{R}$ is Hausdorff if and only if any two distinct equivalence classes in $\mathcal{X}$ are contained in disjoints saturated open subsets of $\mathcal{X}$.*

Thanks to this lemma we prove the following proposition.

Proposition 4. *Let G a finite group acting on an Hausdorff space $\mathcal{X}$, then the orbit space $\mathcal{X}/G$ is Hausdorff.*

Proof. Let Gx and Gy two distinct classes with disjoint open neighbourhood U and V. By finiteness of G, the application $\pi : \mathcal{X} \to \mathcal{X}/G$ is open, hence the saturated sets $\tilde{U} = \pi^{-1}[\pi(U)]$ and $\tilde{V} = \pi^{-1}[\pi(V)]$ are open. Suppose that there exists $z \in \tilde{U} \cap \tilde{V}$, then $\pi(z) \in \pi(U) \cap \pi(V)$ and we finally get that $Gz \subset U \cap V = \varnothing$. Therefore $\tilde{U} \cap \tilde{V}$ is empty and $\mathcal{X}/G$ is Hausdorff by Lemma 10. □

Proposition 4 directly implies that the spaces $\mathbf{Graph}_m$ and $\mathbf{Neighborhood}_m$ are Hausdorff.

B.2 Proof of Theorem 7

For convenience we restate the Theorem.

Theorem 15. *Let $(\mathcal{G}, d)$ be a metric space where $\mathcal{G}$ is the space of graphs and d is the distance defined above. The above metric space cannot be embedded in any Euclidean space.*

Proof. The $N \times N$ Euclidean distance matrix $\mathbf{\Delta}^2$ contains pairwise distances between all N data points in a dataset. Suppose a dataset consists of the 5 graphs shown in Figure B.1 below.

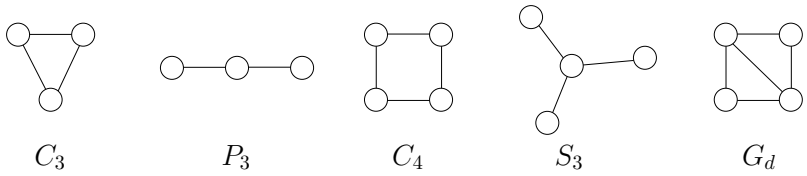


FIGURE B.1: A set of 5 graph which serve as a counterexample for the proof of Theorem 1.

Then, we obtain the following matrix of squared distances:

$$\mathbf{\Delta}^2 = \begin{bmatrix} d_{11}^2 & d_{12}^2 & d_{13}^2 & d_{14}^2 & d_{15}^2 \\ d_{21}^2 & d_{22}^2 & d_{23}^2 & d_{24}^2 & d_{25}^2 \\ d_{31}^2 & d_{32}^2 & d_{33}^2 & d_{34}^2 & d_{35}^2 \\ d_{41}^2 & d_{42}^2 & d_{43}^2 & d_{44}^2 & d_{45}^2 \\ d_{51}^2 & d_{52}^2 & d_{53}^2 & d_{54}^2 & d_{55}^2 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 6 & 4 & 4 \\ 2 & 0 & 4 & 2 & 6 \\ 6 & 4 & 0 & 6 & 2 \\ 4 & 2 & 6 & 0 & 4 \\ 4 & 6 & 2 & 4 & 0 \end{bmatrix}$$

The matrix of squared distances $\mathbf{\Delta}^2$ can be transformed into a matrix of similarities as follows:

$$\mathbf{K} = -\frac{1}{2}\mathbf{J}\,\mathbf{\Delta}^2\,\mathbf{J}$$

where $\mathbf{J}$ is the centering matrix $\mathbf{J} = \mathbf{I} - \frac{1}{5}\mathbf{1}\mathbf{1}^\top \in \mathbb{R}^{5\times 5}$ and $\mathbf{I}$ is the 5×5 identity matrix. Matrix $\mathbf{K}$ is not positive definite or positive semidefinite since it has a negative eigenvalue, i.e. $\min\{\lambda_1, \ldots, \lambda_5\} = -0.366$. Therefore, it cannot be decomposed as the following product:

$$\mathbf{K} = \mathbf{X}\,\mathbf{X}^\top$$

and it cannot be the Gram matrix of any vector representations of the graphs $\mathbf{x}_1, \ldots, \mathbf{x}_5$. Hence, the 5 graphs cannot be embedded in any Euclidean space. □

B.3 Cost Sharing Games

For convenience we restate the equation that provides the soft permutation matrices:

$$\mathbf{P}_1^*, \ldots, \mathbf{P}_N^* = \underset{\mathbf{P}_1,\ldots,\mathbf{P}_N \in \Pi}{\arg\min} \sum_{i=1}^{N}\sum_{j=1}^{N} ||\mathbf{P}_i\,\mathbf{A}_i\,\mathbf{P}_i^\top - \mathbf{P}_j\,\mathbf{A}_j\,\mathbf{P}_j^\top||_F \tag{B.2}$$

We show that there is a connection between optimal alignment of the graph corpus with congestion games. In particular, we transform the objective in Equation B.2 to a *cost sharing game*. Let we have N players, where their initial states are defined by the adjacency matrices $s_i :=$

$\mathbf{A}_i \, \forall[N]$. The strategy profile $\overrightarrow{\mathbf{P}} = [\mathbf{P}_1 \, ... \, \mathbf{P}_N]$ corresponds to the set of permutation matrices that apply to $\overrightarrow{\mathbf{A}} = [\mathbf{A}_1 \, ... \, \mathbf{A}_N]$. Given that the cost for each player is $c_i(\overrightarrow{\mathbf{P}}) = \sum_{j=1}^{N} ||\mathbf{P}_i \, \mathbf{A}_i \, \mathbf{P}_i^\top - \mathbf{P}_j \, \mathbf{A}_j \, \mathbf{P}_j^\top||_F$, the social cost is defined by $C(\overrightarrow{\mathbf{P}}) = \sum_{i=1}^{N} c_i(\overrightarrow{\mathbf{P}})$. Thus, the *social optimum* $(\mathbf{P}_1^*, ..., \mathbf{P}_N^*)$ is identical to the output of Equation B.2. This transformation can provide insights on the bounds of the distances of the transformed adjacencies provided by $\pi-$GNNs (socially optimal difference) with respect to the Frobenius distance (single optimal difference). We know that the Price of Anarchy (PoA) in a cost sharing game is bounded by the number of the graphs: PoA $\leqslant n$. Thus, having a NE strategy profile $\overrightarrow{\mathbf{P}}$ with a cost c_i, we have $c_i \leqslant n \cdot c_i^*$ for each player i with adjacency A_i.

B.4 Proof of Lemma 5

Lemma 11. *Assume $\mathcal{X}$ is countable, and let $r \in \mathbb{N}$. There exist functions $f : \mathcal{X} \to \mathbb{R}^d$ and $f' : \mathcal{X} \to \mathbb{R}^d$, such that $h_i(c, X) = f(c) + \sum_{x \in X} f'(x)$ is unique for each $i \in \{0, \ldots, r\}$ and each pair (c, X), where $c \in \mathcal{X}$ and $X \subset \mathcal{X}$ is a finite multiset. Moreover, any function g_i over such pairs can be decomposed as $g_i(c, X) = \phi\big(f(c) + \sum_{x \in X} f'(x)\big)$ for some function ϕ.*

Proof. We first show that there exists a mapping f' such that $\sum_{x \in X} f'(x)$ is unique for each finite multiset X. Because $\mathcal{X}$ is countable, there exists a mapping $Z : \mathcal{X} \to \mathbb{N}$ from $x \in \mathcal{X}$ to natural numbers. Because the multisets X are finite, there exists a number $N \in \mathbb{N}$ such that $|X| < N$ for all X. Then, an example of such f' is $f'(x) = N^{-Z(x)}$. The sum of the above function for all finite multisets X takes values less than 1, i.e. $\sum_{x \in X} f'(x) < 1$. Hence, if we also set $f(x) = Z(x) + (r - i)|\mathcal{X}|$, then it holds that $h_i(c, X) = f(c) + \sum_{x \in X} f'(x)$ is an injective function over pairs of elements and multisets, and is also unique for each $i \in \{0, \ldots, r\}$.

For any function g_i over the pairs (c, X), we can construct such ϕ for the desired decomposition by letting $g_i(c, X) = \phi\big(f(c) + \sum_{x \in X} f'(x)\big)$. Note that such ϕ is well-defined because $h_i(c, X) = f(c) + \sum_{x \in X} f'(x)$ is injective. □

B.5 Proof of Lemma 6

Lemma 12. *Let G_v^k be the k-hop neighborhood subgraph of a node v. Then, G_v^k contains a cycle of odd length if and only if the shortest path lengths from two adjacent nodes $u, w \in \mathcal{N}_k(v)$ to v are identical.*

Proof. Let u, w be two vertices such that $u, w \in \mathcal{N}_k(v)$. Assume that the shortest path lengths between each of these two vertices and the root v are identical and equal to $d \in \mathbb{N}$ such that $d \leqslant k$. If u and w are connected by an edge, then G_v^k contains a cycle of length $2d + 1$ which is clearly an odd number. This proves the first statement. For the second

statement, assume that G_v^k contains a cycle of odd length and there is no edge between two vertices whose shortest path lengths from the root v are identical. Then, from all the nodes of the cycle, there is a single node u such that the shortest path distance $d \in \mathbb{N}$ from the root v to u is maximum. Since this is a cycle, there are two paths from v to u of length d. Hence, the length of the cycle is $2d$ which is an even number, contradicting the assumption. □

B.5.1 Proof of Theorem 14

For the sake of reading convenience, we restate the theorem:

Theorem 16. *If the score function $\tilde{g}$ is Lipschitz continuous, then the attention layer with score function as defined in Equation (7.23) of the paper is Lipschitz continuous and*

$$L_F(\mathrm{Att}) \leqslant e^{\alpha}\sqrt{\frac{m}{n}} + \alpha\sqrt{8}\,. \tag{B.3}$$

Proof. First, as $c(X) = \max\left\{\|\tilde{g}(X)\|_{(2,\infty)}, \|X^\top\|_{(\infty,2)} L_{F,(2,\infty)}(\tilde{g})\right\}/\alpha$, we have $\alpha c(X) \geqslant \|\tilde{g}(X)\|_{(2,\infty)} \geqslant \|\tilde{g}(X)\|_\infty$ and assumption (1) of Theorem 2 is verified. Second, we have $\alpha c(X) \geqslant \|X^\top\|_{(\infty,2)} L_{F,(2,\infty)}(\tilde{g}) \geqslant \|X^\top\|_{(\infty,2)} \|\mathbf{D}\tilde{g}_X\|_{F,(2,\infty)}$ and assumption (2) of Theorem 2 is also verified. Finally, we have

$$\begin{aligned}
\alpha|\mathbf{D}c_X(H)| &\leqslant \max\left\{\left|\mathbf{D}\|\tilde{g}(\cdot)\|_{(2,\infty)}{}_X(H)\right|, \left|\mathbf{D}\|\cdot^\top\|_{(\infty,2)}{}_X(H)\right| L_{F,(2,\infty)}(\tilde{g})\right\} \\
&\leqslant \max\left\{\|\mathbf{D}\tilde{g}_X(H)\|_{(2,\infty)}, \|H^\top\|_{(\infty,2)} L_{F,(2,\infty)}(\tilde{g})\right\} \\
&\leqslant \max\left\{\|\mathbf{D}\tilde{g}_X\|_{F,(2,\infty)}\|H\|_F, \|H\|_F L_{F,(2,\infty)}(\tilde{g})\right\} \\
&\leqslant L_{F,(2,\infty)}(\tilde{g})\|H\|_F\,,
\end{aligned}$$

where the second inequality follows from the triangle inequality

$$\left|\|X+H\|_{(\infty,2)} - \|X\|_{(\infty,2)}\right| \leqslant \|H\|_{(\infty,2)},$$

implying that $|\mathbf{D}\|\cdot\|_{(\infty,2)}{}_X(H)| \leqslant \|H\|_{(\infty,2)}$. As a result, we have $\|X^\top\|_{(\infty,2)}\|\mathbf{D}c_X\|_{F,1}\|\tilde{g}(X)\|_{(2,\infty)} \leqslant \|X^\top\|_{(\infty,2)} L_{F,(2,\infty)}(\tilde{g})\|\tilde{g}(X)\|_{(2,\infty)}/\alpha \leqslant \alpha c(X)^2$ (using assumption (1) and (2)) and assumption (3) of Theorem 13 is also verified. We can thus apply Theorem 13 and obtain the desired result. □

B.5.2 Proof of Corollary 2

Next, we give the proof of Corollary 2 (Lipschitz continuity of attention with linear score functions).

Proof. First, note that replacing $L_{F,(2,\infty)}(\tilde{g})$ in Theorem 13 by any upper bound $M \geqslant L_{F,(2,\infty)}(\tilde{g})$ does not change the result and, as $L_{F,(2,\infty)}(\tilde{g}) = \|Q\|_*$ is hard to compute, we instead prefer the upper bound $\|Q\|_F \geqslant \|Q\|_*$ that is simple and fast to compute. As $\tilde{g}(X) = Q^\top X$ is Lipschitz, we can directly apply Theorem 14 with $\alpha = 1$ and $c(X) =$

$\max\left\{\|Q^\top X\|_{(2,\infty)}, \|X^\top\|_{(\infty,2)}\|Q\|_F\right\}$ to get the desired result. Moreover, the normalization simplifies to $c(X) = \|Q\|_F\|X^\top\|_{(\infty,2)}$, as $\|Q^\top X\|_{(2,\infty)} \leqslant \|Q\|_F\|X^\top\|_{(\infty,2)}$. □

B.5.3 Proof of Corollary 3

Next, we give the proof of Corollary 3 (Lipschitz continuity of attention with quadratic score functions).

Proof. Let $X = (Q||K||V)$ be a concatenation of queries, keys and values, and $\mathrm{Att}(X) = V\mathrm{softmax}\left(g(X)\right)^\top$. First, note that $\mathrm{Att}(X) = h(X)\,\mathrm{softmax}\left(g(X)\right)^\top$, where $h : X = (Q||K||V) \mapsto V$ is a projection. As projections are contractive, Remark 5 implies that Theorem 13 can be used in such a case if we replace $\|X^\top\|_{(\infty,2)}$ by $\|V^\top\|_{(\infty,2)}$ in assumptions (1)-(3). As proposed in Equation (7.27) of the paper, let $g(X) = \tilde{g}(X)/c(X)$ where $\tilde{g}(X) = Q^\top K$, $c(X) = \max\{uv, uw, vw\}$, $u = \|Q\|_F$, $v = \|K^\top\|_{(\infty,2)}$, and $w = \|V^\top\|_{(\infty,2)}$. Then, we have

$$\|Q^\top K\|_\infty \leqslant \|Q^\top K\|_{(2,\infty)} \leqslant \|Q\|_F\|K^\top\|_{(\infty,2)} = uv \leqslant c(X)\,,$$

and assumption (1) is verified (with $\alpha = 1$). Moreover, for any perturbation $H = (H_Q||H_K||H_V)$, where H_Q, H_K and H_V are the perturbations associated to, respectively, Q, K and V, we have

$$\begin{aligned}\|D\tilde{g}_X(H)\|_{(2,\infty)} &\leqslant \|Q^\top H_K\|_{(2,\infty)} + \|H_Q^\top K\|_{(2,\infty)}\\ &\leqslant \|Q\|_F\|H_K^\top\|_{(\infty,2)} + \|H_Q\|_F\|K^\top\|_{(\infty,2)}\\ &\leqslant u\|H_K\|_F + v\|H_Q\|_F\\ &\leqslant \sqrt{u^2+v^2}\|H\|_F\,,\end{aligned}$$

where the last inequality is due to the Cauchy-Schwarz inequality. Hence, we have

$$\|V^\top\|_{(\infty,2)}\|\mathbf{D}\tilde{g}_X\|_{F,(2,\infty)} \leqslant w\sqrt{u^2+v^2} \leqslant \sqrt{2}c(X)\,,$$

and assumption (2) is verified (with $\alpha = \sqrt{2}$). Finally, we have

$$\begin{aligned}|\mathbf{D}c_X(H)| &\leqslant \max\{v,w\}\|H_Q\|_F + \max\{u,w\}\|H_K^\top\|_{(\infty,2)} + \max\{u,v\}\|H_V^\top\|_{(\infty,2)}\\ &\leqslant \max\{v,w\}\|H_Q\|_F + \max\{u,w\}\|H_K\|_F + \max\{u,v\}\|H_V\|_F\\ &\leqslant \sqrt{\max\{v,w\}^2 + \max\{u,w\}^2 + \max\{u,v\}^2}\|H\|_F\,,\end{aligned}$$

where the last inequality is due to the Cauchy-Schwarz inequality, and thus

$$\begin{aligned}\|\tilde{g}(X)\|_{(2,\infty)}\|V^\top\|_{(\infty,2)}\|\mathbf{D}c_X\|_{F,1} &\leqslant uvw\sqrt{\max\{v,w\}^2 + \max\{u,w\}^2 + \max\{u,v\}^2}\\ &\leqslant \sqrt{3}uvw\max\{u,v,w\} \leqslant \sqrt{3}c(X)^2\,,\end{aligned}$$

and assumption (3) is verified (with $\alpha = \sqrt{3}$). Hence, Theorem 2 with $\alpha = \sqrt{3}$ is applicable and immediately provides the desired result. □

Appendix C

Experimental Details

C.1 Experimental Details for Real-World Benchmarks

In this section, we initially report the experimental setting that is common to all experiments and, then, separately provide the details which are specific to the different tasks (node classification and graph classification).

In our experiments, we use the Adam optimizer [117] with a weight decay on the parameters of $5 * 10^{-4}$ and an initial learning rate of 0.005 for the exponential parameters and an initial learning rate of 0.01 for all other model parameters. The differentiation between the two learning rates is crucial for the model training, as the fluctuation of the exponential parameters has a higher impact on the model behavior than the rest of the model parameters. We, also, used a learning rate scheduler that decayed both the learning rates by 0.5 every 50 epochs.

Node Classification Tasks We report below the experimental details used in the node classification datasets, regarding model selection, metrics and model design. The Cora and CiteSeer datasets are formulated as binary class classification problems, while the Ogbn-arxiv is formulated as a 40-class classification problem.

- **Model Selection**: We perform cross-validation for all GNN, GNN-PGSO and GNN-mPGSO models with predefined dataset splits. For Cora and CiteSeer, we use the same train/validation/test splits as in Kipf and Welling [118] for the sake of comparison with other methods that use the same splits. For Ogbn-arxiv, we use the splitting method, described in Hu et al. [102], that splits the data into the train split, containing all papers until 2017, the validation split, containing all papers published in 2018 and test split with the published papers since 2019.
- **Hyper-parameter Tuning**: We use *grid search* to tune the hyper-parameters of the models (dimensionality of hidden units, number of GNN layers, number of epochs and the batch size).

- **Evaluation Metric**: For all node classification datasets (Cora, CiteSeer, Ogbn-arxiv), we use as evaluation metric the validation accuracy.
- **Model Design**: The number of hidden units that was grid-searched was within $\{16, 32, 64\}$. The number of GNN layers that were used were within $\{1, 2, 3, 4\}$.
- **Experiment Design**: The number of epochs for the model training was $\{50, 100, 200\}$, batch size within $\{16, 32, 64\}$ and a dropout ratio of 0.5.

Graph Classification Tasks In the same fashion we report below the details of the experimentation setup of the graph classification tasks. The MUTAG, PTC-MR, IMDB-BINARY and OGBG-MOLHIV are formulated as binary class classification problems, while IMDB-MULTI is formulated as a 3-class classification problem.

- **Model Selection**: We perform 10-fold cross-validation for all GNN,GNN-PGSO, GNN-mPGSO models. For MUTAG, PTC-MR, IMDB-BINARY and IMDB-MULTI, we have used k random and stratified splits to provide balanced train/validation/test sets. For OGBG-MOLHIV we use the scaffold splitting as in Hu et al. [102], that seperates the graphs based on their 2D structural representations.
- **Hyper-parameter Tuning**: We use *grid search* to tune the hyper-parameters of the models (dimensionality of hidden units, number of GNN layers, number of epochs and the batch size).
- **Evaluation Metric**: For the MUTAG, PTC-MR, IMDB-BINARY, IMDB-MULTI datasets, we use as evaluation metric the standard classification accuracy, while for the OGBG-MOLHIV we use as Hu et al. [102] the validation ROC-AUC.
- **Model Design**: The number of hidden units that was grid-searched was within $\{16, 32, 64\}$. The number of GNN layers that were used were $\{1, 2, 3, 4, 5\}$.
- **Experiment Design**: The number of epochs for the model training was $\{100, 200, 300\}$, the batch size within $\{16, 32, 64\}$ and the dropout ratio of 0.5.

C.2 Convergence study

In this appendix, we empirically analyse the contribution of the PGSO in a node classification and a graph classification task regarding the accuracy and loss convergence. For the node classification task, we have used the *Cora* dataset and a GCN model. Both for the standard GCN and the GCN-PGSO models, we used the same experimentation

configuration for a fair comparison. For the graph classification task, we have used *PTC-MR* dataset and a GIN model. Both for the standard GIN and the GIN-PGSO models, we used the same configuration for a fair comparison. In Figures C.1 and C.2, we show the accuracy and loss convergence using the standard model and the PGSO model. As we can see, for both the node classification and graph classification task, the PGSO incorporation into the model has a positive impact on the achieved accuracy and the loss minimization throughout training. Specifically, we observe a slightly faster convergence of accuracy and loss in better values using the GCN-PGSO and GIN-PGSO models.

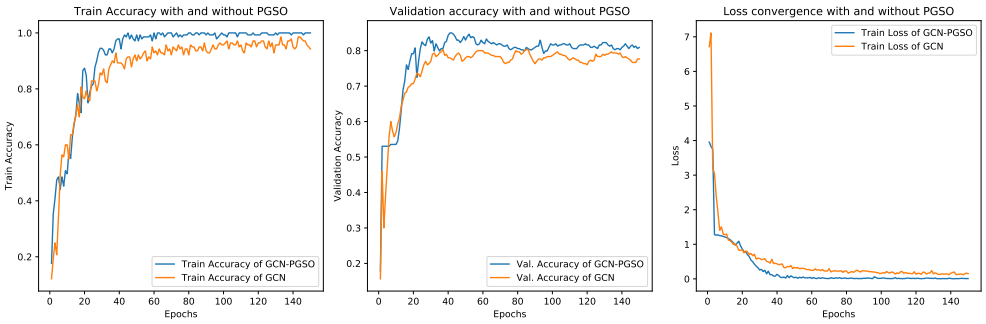


FIGURE C.1: Accuracy and Loss convergence for the node classification task on Cora

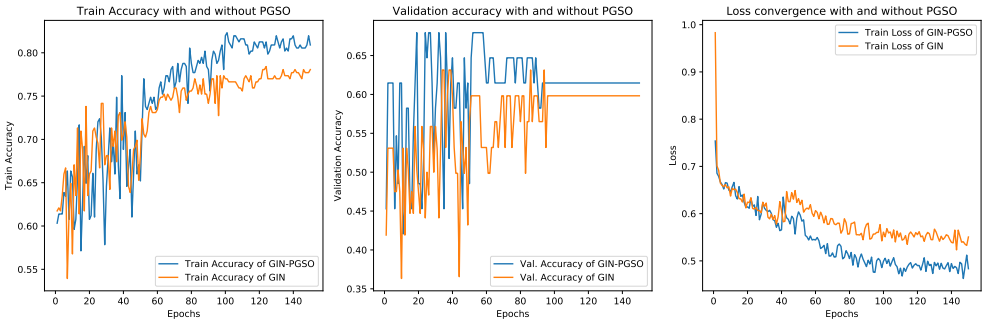


FIGURE C.2: Accuracy and Loss convergence for the graph classification task on PTC-MR

DOCTORAL SCHOOL

Titre : Vers des Graph Neural Networks expressives : théorie, algorithmes et applications

Mots clés : Apprentissage de représentations de graphes, Réseaux de neurones de graphes, Expressivité du modèle, Classification des nœuds, Classification de graphes, Représentations des nœuds

Résumé : L'évolution technologique de l'apprentissage automatique s'accélèrant, les données jouent un rôle de plus en plus important dans la construction de modèles intelligents, capables de simuler des phénomènes, de prédire des résultats et de prendre des décisions. Dans un nombre sans cesse croissant d'applications, les données sont structurées et peuvent être vues comme des graphes. L'exploitation de cette structure est le coeur du domaine de l'apprentissage de représentations de graphes, qui consiste à calculer des représentations suffisamment expressives des graphes et de ses composants, c'est-à-dire les nœuds et les arêtes. Récemment, la domaine de l'apprentissage de représentations de graphes a été accéléré par le succès des algorithmes du type ≪message passing≫ (passation de messages) appliqués aux graphes, à savoir les ≪Graphe Neural Network≫ (réseaux de neurones sur les graphes). L'apprentissage de représentations informatives et expressives sur les graphes joue un rôle critique dans un large éventail d'applications du monde réel, depuis les télécommunications et les réseaux sociaux jusqu'à la conception urbaine, la chimie et la biologie. Dans cette thèse, nous étudions les différents aspects à partir desquels les réseaux neuronaux graphiques peuvent être plus expressifs, et nous proposons de nouvelles approches pour améliorer leurs performances dans les tâches standard d'apprentissages. Les principaux axes de la présente thèse sont : l'universalité des représentations de graphes, l'augmentation du champ réceptif des réseaux de neurones sur les graphes, la conception de modèles d'apprentissage de graphes stables et profonds et enfin les alternatives au cadre standard des algorithmes par passation de messages. En réalisant des études théoriques et expérimentales, nous montrons comment les approches proposées peuvent devenir des outils utiles et efficaces pour concevoir des modèles d'apprentissage de graphes plus expressifs et plus puissants.

Dans la première partie de la thèse, nous étudions la qualité des représentations de graphes en fonction de leur pouvoir de discrimination. Cette partie se concentre sur l'approximation universelle et les propriétés d'isométrie des représentations graphiques.

Dans la deuxième partie de la thèse, notre objectif principal est concentré autour du champ réceptif des réseaux de neurones des graphes, c'est-à-dire la quantité d'informations dont dispose un nœud. Nous étudions des approches qui peuvent augmenter le champ réceptif, en paramétrant les encodages des graphes et en considérant des sous-graphes plus grands pour l'agrégation d'informations.

Dans la dernière partie de la thèse, nous étudions les interactions non locales dans les graphes. Cela inclut les dépendances à longue portée, où les nœuds qui se trouvent dans des parties éloignées du graphe peuvent s'affecter les uns les autres et les dépendances structurelles, où les modèles de voisinage symétriques sont cruciaux.

Title : Towards Expressive Graph Neural Networks: Theory, Algorithms and Applications

Keywords : Graph Representation Learning, Graph Neural Networks, Model Expressivity, Node Classification, Graph Classification, Node Representations

Abstract : As the technological evolution of machine learning is accelerating, data plays an important role in building intelligent models, being able to simulate phenomena, predict values and make decisions. In an increasing number of applications, data take the form of networks. The inherent graph structure of network data motivated the evolution of the graph representation learning. Its scope includes generating meaningful representations for graphs and their components. The research on graph representation learning was accelerated with the success of message passing frameworks applied on graphs, namely the Graph Neural Networks. Learning informative and expressive representations on graphs plays a critical role in a wide range of real-world applications, from telecommunication and social networks, urban design, chemistry, and biology. In this thesis, we study various aspects from which Graph Neural Networks can be more expressive, and we propose novel approaches to improve their performance in standard graph learning tasks. The main branches of the present thesis include the discrimination power of graph representations, the receptive field of the models and, the study of long-range interactions. Performing both theoretical and experimental studies, we show how the proposed approaches can become efficient tools for designing powerful graph learning models.

In the first part of the thesis, we study the quality of graph representations as a function of their discrimination power, i.e how easily we can differentiate graphs that are not isomorphic. This part is focused into universal approximation and isometry properties of graph representations.

In the second part of the thesis, our main focus is concentrated around the receptive field of the graph neural networks, i.e how much information a node has, in order to update its representation. We investigate approaches that can increase the receptive field, by parametrizing the encodings of the graph structures and by considering larger subgraphs for the information aggregation.

In the last part of the thesis, we study non-local interactions in graphs. This includes the long-range dependencies, where nodes that lie in distant parts of the graph can affect each other and structural dependencies, where symmetrical neighborhood patterns are crucial for the graph nodes.

Institut Polytechnique de Paris
91120 Palaiseau, France