Extensions and Applications of Graph Neural Networks

Guillaume Lachaud

▶ **To cite this version:**

Guillaume Lachaud. Extensions and Applications of Graph Neural Networks. Artificial Intelligence [cs.AI]. Sorbonne Université, 2023. English. NNT : 2023SORUS434 . tel-04383168

HAL Id: tel-04383168
https://theses.hal.science/tel-04383168

Submitted on 9 Jan 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Sorbonne Université

École doctorale n°130: *Informatique, Télécommunications et Électronique*

Laboratoire d'Informatique, Signal et Image, Electronique et Télécommunications

Extensions and Applications of Graph Neural Networks

THÈSE

présentée par

Guillaume Lachaud

soutenance prévue le 28 septembre 2023

pour obtenir le grade de docteur de Sorbonne Université

discipline: Informatique

Thèse dirigée par:
M^me^ Maria Trocan — Professeure, ISEP

Thèse encadrée par:
M^me^ Patricia Conde-Cespedes — Professeure associée, ISEP

Rapporteurs:
M. Dan Istrate — Professeur (HDR), UTC
M. Behçet Uğur Töreyin — Professeur, Informatics Institute (ITU), İstanbul Teknik Üniversitesi

---

Membres du jury:
M^me^ Anissa Mokraoui — Professeure, L2TI, Université Sorbonne Paris Nord, Présidente du jury
M. Christophe Marsala — Professeur, LIP6, Sorbonne Université

Abstract

Graphs are used everywhere to represent interactions between entities, whether physical such as atoms, molecules or people, or more abstract such as cities, friendships, ideas, etc. Amongst all the methods of machine learning that can be used, the recent advances in deep learning have made graph neural networks the de facto standard for graph representation learning. This thesis can be divided in two parts. First, we review the theoretical underpinnings of the most powerful graph neural networks. Second, we explore the challenges faced by the existing models when training on real world graph data.

The powerfulness of a graph neural network is defined in terms of its expressiveness, i.e., its ability to distinguish non isomorphic graphs; or, in an equivalent manner, its ability to approximate permutation invariant and equivariant functions. We distinguish two broad families of the most powerful models. We summarise the mathematical properties as well as the advantages and disadvantages of these models in practical situations.

Apart from the choice of the architecture, the quality of the graph data plays a crucial role in the ability to learn useful representations. Several challenges are faced by graph neural networks given the intrinsic nature of graph data. In contrast to typical machine learning methods that deal with tabular data, graph neural networks need to consider not only the features of the nodes but also the interconnectedness between them. Due to the connections between nodes, training neural networks on graphs can be done in two settings: in transductive learning, the model can have access to the test features in the training phase; in the inductive setting, the test data remains unseen. We study the differences in terms of performance between inductive and transductive learning for the node classification task. Additionally, the features that are fed to a model can be noisy or even missing. In this thesis we evaluate these challenges on real world datasets, and we propose a novel architecture to perform missing data imputation on graphs.

Finally, while graphs can be the natural way to describe interactions, other types of data can benefit from being converted into graphs. In this thesis, we perform preliminary work on how to extract the most important parts of skin lesion images that could be used to create graphs and learn hidden relations in the data.

Résumé

Les graphes sont utilisés partout pour représenter les interactions, qu'elles soient physiques comme les atomes, les molécules ou les humains, ou plus abstraites comme les villes, les amitiés, les idées, etc. Parmi toutes les méthodes d'apprentissage automatique qui peuvent être utilisées, les dernières avancées en apprentissage profond font des réseaux de neurones de graphes la référence de l'apprentissage de représentation des graphes. Cette thèse se divise en deux parties. Dans un premier temps, nous faisons un état de l'art des fondations mathématiques des réseaux de neurones de graphes les plus puissants. Dans un second temps, nous explorons les défis auxquels sont confrontés ces modèles quand ils sont entraînés sur des jeux de données réels.

La puissance d'un réseau de neurones est définie par rapport à son expressivité, c'est-à-dire sa capacité à distinguer deux graphes non isomorphes ; ou, de manière équivalente, sa capacité à approximer les fonctions qui sont invariantes ou équivariantes par rapport aux permutations. Nous discernons deux grandes familles de modèles expressifs. Nous présentons leurs propriétés mathématiques ainsi que les avantages et les inconvénients de ces modèles lors d'applications pratiques.

En parallèle du choix de l'architecture, la qualité de la donnée joue un rôle crucial dans la capacité d'un modèle à apprendre des représentations utiles. Les réseaux de neurones de graphes sont confrontés à des problèmes spécifiques aux graphes. À l'inverse des modèles développés pour les données tabulaires, les réseaux de neurones de graphes doivent prendre en compte aussi bien les attributs des nœuds que leur interdépendance. À cause de ces liens, l'apprentissage d'un réseau de neurones sur des graphes peut se faire de deux manières : en apprentissage transductif, où le modèle a accès aux attributs des données de test pendant l'entraînement ; en apprentissage inductif, où les données de test restent cachées. Nous étudions les différences en termes de performance entre l'apprentissage transductif et inductif pour la classification de nœuds. De plus, les attributs des nœuds peuvent être bruités ou manquants. Dans cette thèse, nous évaluons ces défis sur des jeux de données réels, et nous proposons une nouvelle architecture de réseau de neurones de graphes pour imputer les attributs manquants des nœuds d'un graphe.

Enfin, si les graphes sont le moyen privilégié de décrire les interactions, d'autres types de données peuvent aussi bénéficier d'une conversion sous forme de graphes. Dans cette thèse, nous effectuons un travail préliminaire sur l'extraction des parties les plus importantes d'images de lésions de la peau. Ces patches pourraient être utilisés pour créer des graphes et découvrir des relations latentes dans la donnée.

Contents

List of Figures

List of Tables

Acknowledgements

This thesis would not have been possible without my advisors, Maria and Patricia, for whom I am particularly grateful. Thank you for the long journey, filled with arduous tasks but ultimately exceedingly rewarding: it opened up new paths for my life and career that I will gladly embark on. The growth I needed, you gave me, and for that I am eternally thankful!

A thesis cannot be done without advisors, and so too can it be carried out only with a jury. My thanks therefore go to Professors Dan Istrate and Behçet Uğur Toreyin for accepting to review my manuscript; and to Professors Anissa Mokraoui and Christophe Marsala for accepting to be part of the jury.

Although this adventure is essentially a solitary journey, it cannot be undertaken without companions you encounter on the road. Too many people deserve thanks that I will necessarily miss some; I apologise in advance and will still endeavour to perform this perilous exercice.

Je tiens évidemment à remercier tous les membres du personnel de l'ISEP, dont je connais certains depuis presque huit ans à présent. Ma gratitude va tout particulièrement pour Gilles, avec lequel nos conversations sont comme un bon vin qui vieillit bien, et pour lequel des nouvelles saveurs se découvrent avec le temps ; qui eut cru que nous nous embarquerions dans cette grande aventure qu'est la Roue du Temps ? Les sujets de discussions ne manquaient déjà pas, mais la thèse a été l'occasion d'en trouver bien d'autres, et je souhaite que l'après-thèse en fournisse encore davantage. Mes remerciements s'étendent bien sûr à Zakia, Yousra, Bahareh, Céline, Lionel, Madjid, Nouredine, Saad, Maurras, Christophe, Frédéric, Xun, Matthieu, Louis-Joseph, Karim, Iteb, Iyeb, Mohammed, Nicole, Carole, Samantha, Sébastien, Ziqi, Béatrice, Hang, Yue, Thomas, Ahmed, Sabrina, Nour, Edith, Valérie, Nicole, Emmanuelle, Florence, Lina, Wafa, Pierre, Jérémie, Hedi, Raja, Hervé, Jérôme, Henri, ainsi qu'à tous ceux que j'ai oubliés. Je remercie également Hélène avec laquelle j'espère des collaborations futures, et Joséphine qui est un beau rayon de soleil (je n'ai pas oublié les crêpes, promis !). Je pense aussi à tous les étudiants qui je l'espère m'ont fait devenir un meilleur enseignant.

En remerciant les membres de l'ISEP, je ne peux oublier ceux qui en contrôlent les portes (bien plus Saint-Pierre que Cerbère) : les agents de sécurité Jimmy et Romuald. Ce sont avec eux que les journées commencent et se terminent ; je peux dire sans hésitation que ces journées n'auraient pas été aussi bonnes sans eux.

My attention must now turn to the main companions of this thesis: my fellow PhD students. Without order and with a similar gratitude, I must thank them all: Mariam, Louis, Ayoub, Seoyoung, Fahim, Dalhatu, Shufan, Jade, Tatty, Eduardo, Idowu, Masoud, Hongxiu, Dayu, Yaya, Xia, Kevin, Xuanbang, Xiaodong, Lina, Stéphane, Souha, Tristan, Ana. I must also thank the interns that shared our lunches: Bastien, Lorenzo, Ferdinand, Ehsan, and Katty & Rohan. Abdul, Mr President, you have a bright future ahead, and I can't wait to be there with you to witness these beautiful days.

Je remercie aussi mes amis (je ne préfère pas commencer la liste, elle serait nécessairement incomplète... les aléas de la mémoire !), les clubs d'aïkido et de judo de la Celle Saint-Cloud, en particulier les maîtres Jean-Paul, Alexandre, Guy et Philippe qui me transmettent leur passion à chaque séance. Je remercie également

"C'est dans cette salle"... à tout honneur tout seigneur, ces remerciements seraient incomplets si j'oubliais ceux qui ont vécu au jour le jour avec moi, à savoir les locataires de la mythique L307 ! Arthur, 8 ans déjà, d'ingénieur à docteur, un beau chemin qui j'espère continuera dans les années à venir.

Viennent maintenant les trois Mousquetaires de cette salle, sans lesquelles cette thèse serait bien différente.
Naty, eres una persona maravillosa! Las palabras no pueden hacerte justicia. Te mereces lo mejor y lo tendrás. Un gran abrazo!
楠，你真行！我对你感激涕零。一帆风顺！
Abir, ma3ndich klame bach nchoukrek.

Enfin, je tiens tout particulièrement à remercier toute ma famille,
Mes parents, mon frère, ma grand-mère, mon papy,
Ceux qui sont là, ceux à venir, et ceux qui sont partis.

Chapter 1

Introduction

1.1 Motivation

Graphs are mathematical objects that capture relations between entities. They represent physical interactions, such as between protons and electrons at the microscopic scale, and between galaxies at the macroscopic scale. They also represent informational interactions. For instance, we encounter graphs everywhere throughout our day: when we interact with other people on social networks, browse content on streaming platforms, buy products using online retailers, or when we search for the best itineraries for travelling from one place to another. They are also used in science, where epidemiologists study the spread of the disease, chemists explore the properties of atoms and molecules, and programming code can be analyzed using its abstract syntax tree representation.

The manipulation of graph data can be divided into three types of tasks: node level tasks, such as node classification; edge level tasks, e.g., link prediction; and graph classification. Throughout the years, the treatment of graphs in different fields has generated a vast range of approaches. For instance, a widespread approach of learning with graphs is the computation of graph statistics, e.g., counting the number of times a pattern is appearing in a graph. Random walks, graph kernels, graph spectral theory, and the use of shallow embeddings, e.g., via the learning of an encoder and a decoder, are also active areas of research.

Most of these methods suffer from some common limitations. In particular, they do not scale efficiently to large scale graphs, because there is parameter sharing. Additionally, these methods usually do not exploit the features of the nodes and the edges. Also, these approaches are designed to learn embeddings of the nodes encountered during training. They do not generalize to unseen nodes. They are *transductive* rather than *inductive*.

One way to mitigate the scaling limitations is to use *deep learning*. Neural networks are models in which the parameters are stacked in layers. They allow the efficient learning of complex representations of input data using the automatic adjustment of parameters via loss optimization. Although the first models were theorized 70 years ago, the big breakthrough came with the advent of Graphical Processing Units (GPUs) and the creation of large scale datasets freely made available on the Web.

After the successes of deep learning in computer vision with Convolutional Neural Networks (CNNs) and in natural language processing with Recurrent Neural Networks (RNNs), new neural networks, Graph Neural Networks (GNNs), were proposed for learning with graphs. The first major architecture was the Graph Convolutional Network (GCN) in 2016 Kipf and Welling (2017). Most GNNs behave according to a *message-passing* framework: to update the representation of a node, information is gathered from its neighbours. Figure 1.1 shows how the stacking of layers influences the update of a node. With a model of two layers, the node we want to update (the blue node), has received information from almost all the nodes in the graph, e.g., from all the green nodes.

The equations representing message passing neural networks (MPNNs) are Equations 4.1 and 4.2

1.2 Challenges of graph neural networks

The structure of graph data adds a layer of complexity to the training of neural networks. Indeed, GNNs rely on two elements: information, contained in the features of the nodes, edges, etc.; and the flow of information, represented by the interconnection between the nodes, e.g., the structure of the graph. These aspects raise challenges unique to graphs. In this thesis we address three of these issues: the difference between transductive and inductive learning, multilabel classification with graphs, and noise in the nodes of the graph. Specifically, we deal with *node classification*.

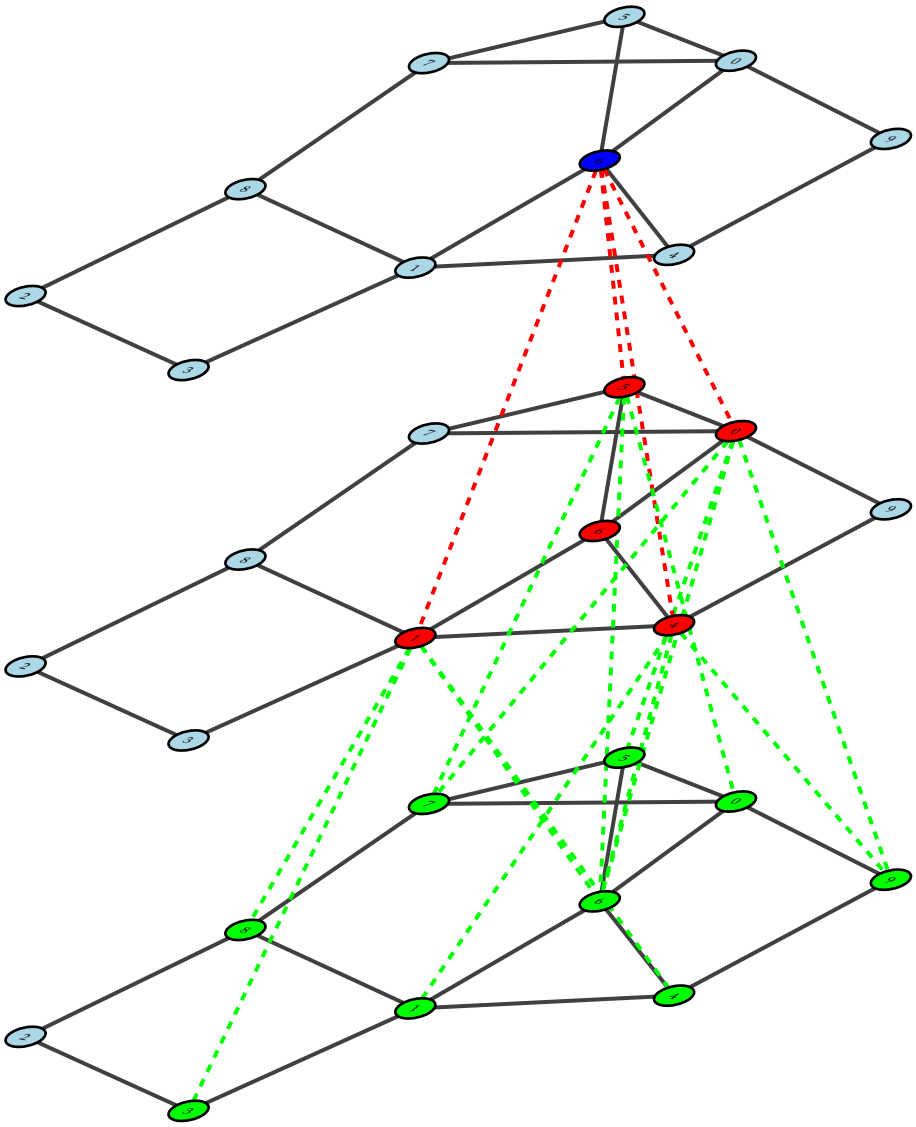
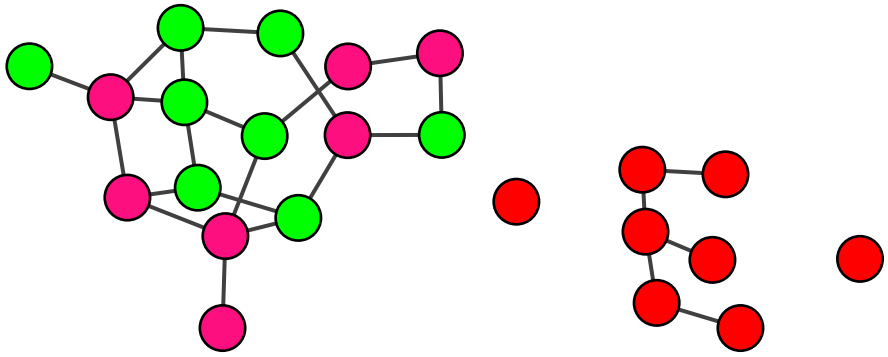


Figure 1.1: Transmission of information in a MPNN model with two layers. The node we update is the blue node at the top. The red nodes represent the nodes involved in the update of our node in the first layer of the model. The green nodes represent those involved in the update of our node after the second layer.

In node and edge level tasks, the way we generate train and test datasets results in two settings of learning: *transductive* learning and *inductive* learning. Most splits of a graph will create pairs of nodes linked by an edge where one node is in the training set and the other in the test set. In transductive learning, all the nodes are used during the training phase; the model has access to the unlabelled features of the test nodes. In inductive learning, only the subgraph induced by the train nodes is used for training. This is illustrated in Figure 1.2. The full graph is shown in Figure 1.2-(a), with the training nodes in green and the test nodes in pink. In transductive learning, the model has access to both the green and pink nodes at training time. In inductive learning, the model has access to only the green nodes, which are reproduced in Figure 1.2-(b) in red. Hence, the graph structure on which the model is trained in different in both settings.



(a) Transductive learning. The training nodes are in green, the test nodes in pink.

(b) Inductive learning. The training nodes are in red.

Figure 1.2: Difference between transductive and inductive learning. (a) the model has access to both the train and test nodes at training time. (b) the model has only access to the train nodes at training time.

In classification, a GNN predicts the label of a node by leveraging the information from the node and from its neighbours. Both the features of the node and the neighbours will push the model towards a label, but this label may be different in each case. The challenge is to determine which label to choose. This is illustrated in Figure 1.3. The features of the node we want to classify (in pink) are similar to those of a *cat*, but all the neighbours of the node (in green) are *dogs*. Is it a dog in cat's clothing having an identity crisis? Is it a cat living in a dog's neighbourhood? Which information is more valuable: the features or the neighbours?

Classification is also dependent on the quality of the data. In particular, GNNs

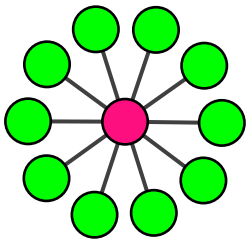


Figure 1.3: Node classification on a node where the features indicate a label (cat), and the neighbours indicate another label (dog).

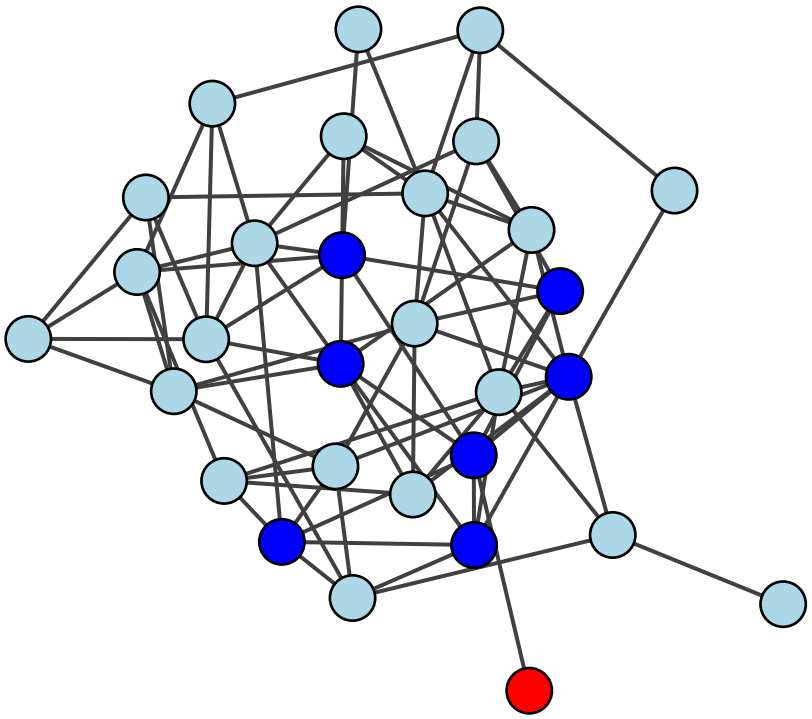
may be confronted with noisy data, i.e., the features of the nodes have noise. The message-passing nature of GNNs propagates the noise of a node to the rest of the graph, as illustrated by Figure 1.4. The influence of the noise differs according the location of the node in the graph, i.e., an affluent node will reach other nodes faster than a more isolated node, but the strength of the noise will be different.

Another way to improve the performance of GNNs is to tackle the problem of graph modelling, i.e., the construction of graph to represent interactions. Improving the quality of the graphs will improve the quality of the predictions as well.

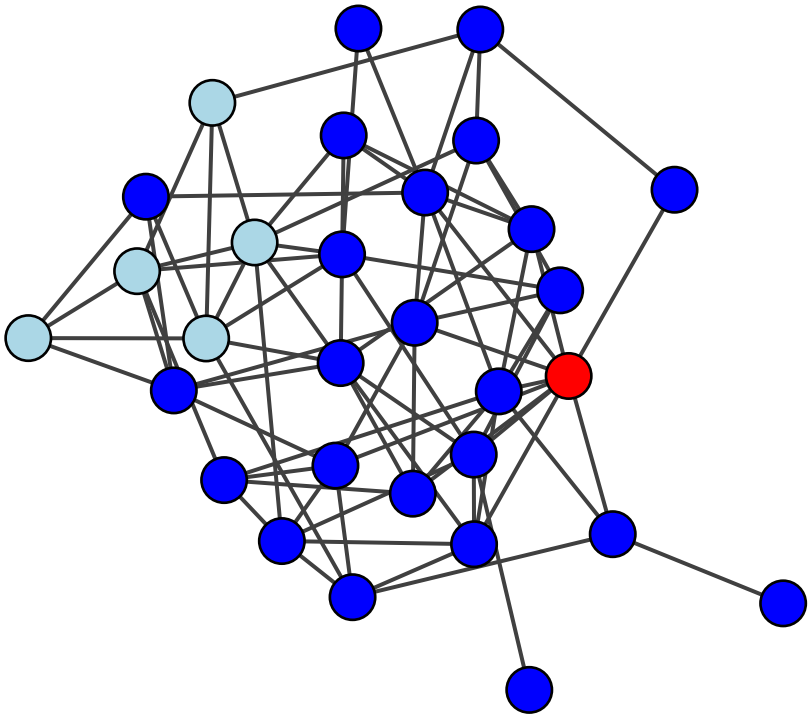
1.3 Contributions

Our contributions are the following:

- we present a survey of the most expressive GNNs, dividing the architectures into groups corresponding to the techniques used to increase expressiveness,
- we demonstrate the value of multilabel classification on a real world dataset. In particular, multilabel classification helps perform a refined diagnosis of the errors made by a GNNs;
- we show that different GNNs behave differently in terms of noise;
- we empirically show that GNNs are the most vulnerable to noise which targets the most isolated nodes;



(a) Small degree node



(b) High degree node

Figure 1.4: Noise propagation after two layers. The red node is the noisy node; the deep blue nodes are the nodes impacted by the noise. (a) the noisy node has the fewest neighbours in the graph. (b) the noisy node has the most neighbours in the graph.

- we introduce a new GNN architecture for missing data imputation;
- we provide preliminary results for graph creation in medical related data using extracted patches.

1.4 Thesis outline

We organize the thesis as follows:

- Chapter 2 explains the main concepts related to graphs and graph learning, as well as the main methods of graph learning before graph neural networks.
- Chapter 3 presents the main ideas behind deep learning and graph neural networks. It also presents the most expressive types of graph neural networks, where expressiveness is the ability to distinguish non isomorphic graphs.
- Chapter 4 presents the most commonly used graph neural networks in practice, as well as some of the most respected benchmarks to compare architectures.
- Chapter 5 first introduces the concept of transductive and inductive learning, then shows the results of multilabel classification on a real world academic citation network.
- Chapter 6 presents an analysis of the performance of several graph neural networks when noise is injected into the features. The chapter concludes with a novel graph neural network architecture for missing data imputation.
- Chapter 7 presents preliminary work for generating graph data out of medical images.
- Chapter 8 concludes the thesis and offers perspectives for future work.

Moreover, this thesis is divided into two parts. The first part contains Chapters 2 and 3 and concerns the representation of graphs and the theoretical aspect of

graph neural networks. The second part contains Chapters 4-7 and deals with practical applications of GNNs.

1.5 Publications

This thesis comprises the following articles that have been published internationally.

- Lachaud, G., Conde-Cespedes, P., and Trocan, M. (2022). Comparison between inductive and transductive learning in a real citation network using graph neural networks. In ASONAM 2022, 2022 (IEEE). 10.1109/ASONAM55673.2022.10068589.
- Lachaud, G., Conde-Cespedes, P., and Trocan, M. (2021). Entropy role on patch-based binary classification for skin melanoma. In ICCCI 2021. 10.1007/978-3-030-88113-9_26.
- Lachaud, G., Conde-Cespedes, P., and Trocan, M. (2022). Graph neural networks-based multilabel classification of citation network. In ACIIDS 2022. 10.1007/978-3-031-21967-2_11.
- Lachaud, G., Conde-Cespedes, P., and Trocan, M. (2022). Mathematical expressiveness of graph neural networks. Mathematics 10, 4770.
- Lachaud, G., Conde-Cespedes, P.C., and Trocan, M. (2022). Patch selection for melanoma classification. In ICCCI 2022. 10.1007/978-3-031-16014-1_13.
- Lachaud, G., Conde-Cespedes, P.C., and Trocan, M. (2023). Scalable Missing Data Imputation with Graph Neural Networks. In IWCIM 2023, IEEE ICASSP 2023 Satelite Workshop.

Part I

Graph Representation

Chapter 2

Introduction to graphs, and graph representation learning and signal processing

This chapter presents an overview of the mathematical concepts related to graphs and machine learning with graphs that will be used throughout the thesis. Specifically, Section 2.1 introduces the definition of graphs and related items, such as trees, directed acyclic graphs, etc. Section 2.2 defines the most common types of tasks done on graphs, e.g., node, edge, and graph level tasks. Section 2.3 concludes this chapter by mentioning some of fundamental techniques of machine learning with graphs.

Contents

2.1 Graphs

2.1.1 Mathematical representation

Mathematically, a graph G is represented by a pair (V, E) consisting of a set of nodes (vertices) V and a set of edges E. Each edge can be expressed as a pair (u, v) of nodes u and v in V. Figure 2.1 shows an example of a graph, where each type of element (node, edge) has been highlighted, e.g., nodes appear in blue while edges appear in red.

An edge (u, v) can be directed, in which case we refer to u as the source and v as the target. If a graph is undirected, then $(u, v) \in E \Rightarrow (v, u) \in E$. When edges have a weight, e.g. $e_{uv} \in \mathbb{R}^d$ for some dimension d, we say that the graph is weighted. More generally, each node may possess *features*. When the nodes or the edges have the features, we can speak of an *attributed graph*. Figure 2.2 shows an undirected, a directed, and a weighted graph sharing the same underlying graph.

The adjacency matrix associated with Figure 2.2(a) is

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Associated with each graph is an adjacency matrix A. If the graph is unweighted, then $A \in \{0, 1\}^{|V| \times |V|}$ where $A_{uv} = 1$ if and only if $(u, v) \in E$. If the graph is undirected, then A is symmetric. If the graph is weighted, the coefficient A_{uv} corresponds to the weight of e_{uv}.

Each node may possess *features*, which can take the form of scalar or vector values.

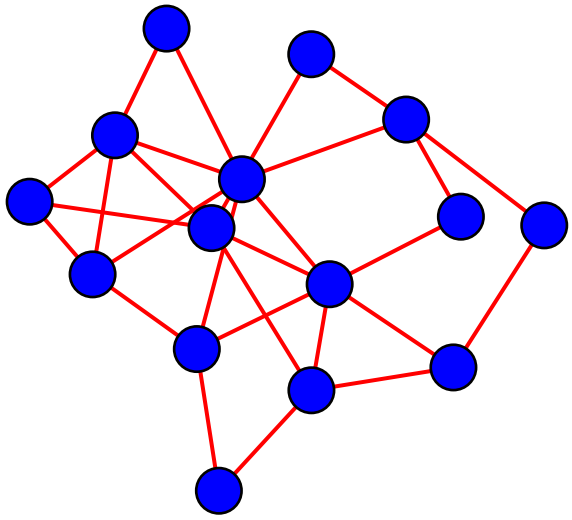
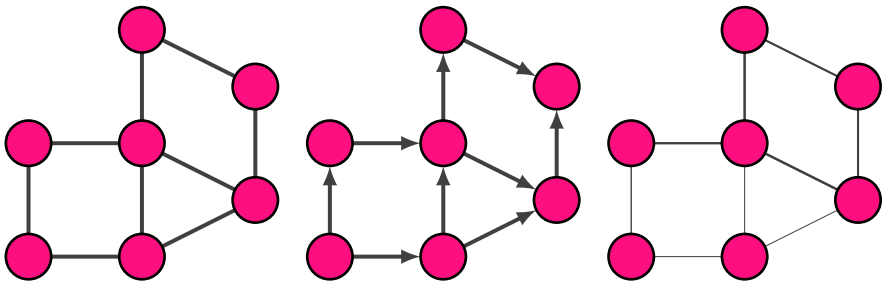


Figure 2.1: Example of a graph. Nodes are colored in blue and edges in red. The graph is the combination of the set of vertices and the set of edges.



(a) Undirected graph (b) Directed graph (c) Weighted graph

Figure 2.2: Examples of the main types of graphs.

The dimensionality can be viewed as the size of the embeddings. The *curse of dimensionality* in machine learning refers to the exponential growth of the number of samples required to generalize.

The features may take categorical, discrete or continuous values. The features of all the nodes are represented by a matrix X. If the features are continuous, $X \in \mathbb{R}^{n \times n_d}$, where n_d is the dimensionality of the nodes's features.

2.1.2 Important definitions

Most of these definitions can be found in graph theory textbooks. See for example Bondy and Murty (2008); Diestel (2012).

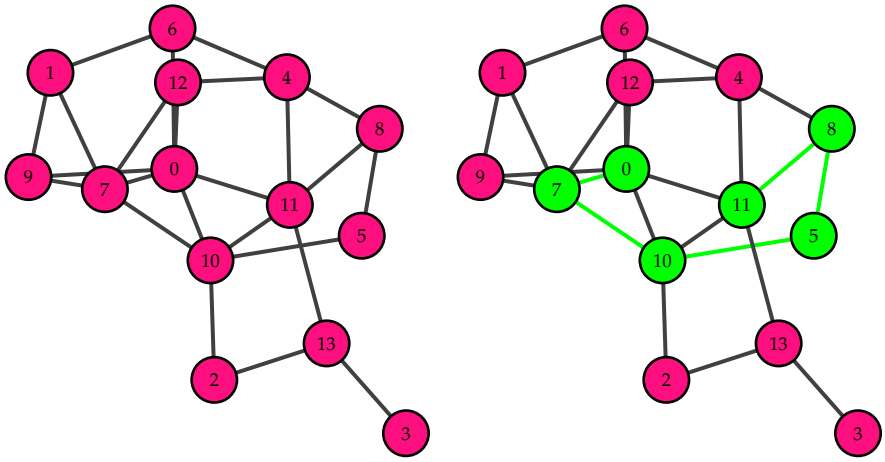
Note how a subgraph and an induced subgraph share the same nodes, but the induced subgraph can have more edges.

A *subgraph* $G^S = (V^S, E^S)$ of G is a graph where $V^S \subset V$ and $E^S \subset E$. An *induced subgraph* is a subgraph where $e = (u, v) \in (V^S)^2$ for all edges in E^S. In other words, an induced subgraph contains all the edges connecting the nodes in the subgraph. Figure 2.3 illustrates this point, by showing both a graph (pink), a subgraph (green) and the induced subgraph with the same nodes (green as well).

A graph is said to be *simple* if it does not contain any loops or parallel edges. A graph is *acyclic* if it does not contain any cycles. A *path* is a simple graph in which the vertices can be arranged in a linear sequence such that two vertices are neighbours if and only if they are consecutive in the sequence. Two nodes u and v are said to be *connected* if there exists a path from u to v. A graph is *connected* if every pair of vertices in the graph are connected.

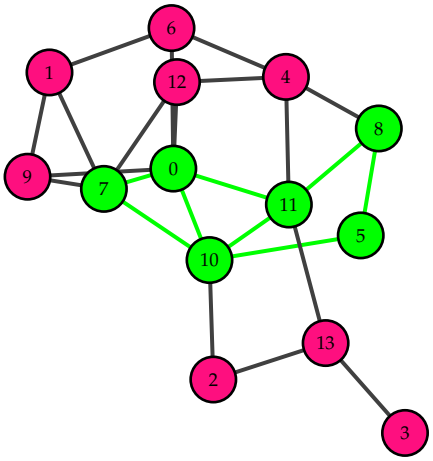
A *tree* is a connected graph in which every pair of vertices is connected by only one path. A *directed acyclic graph* (*DAG*) is a directed graph whose underlying graph is a tree.

In some cases, it is important to identify a starting point in a graph. A *rooted graph* G_v is a graph where v acts as the root node. Another term for rooted graph is *egonet*. Similarly, a rooted tree is a tree where one of the terminal nodes, e.g., nodes with a single neighbour, is taken as the root.



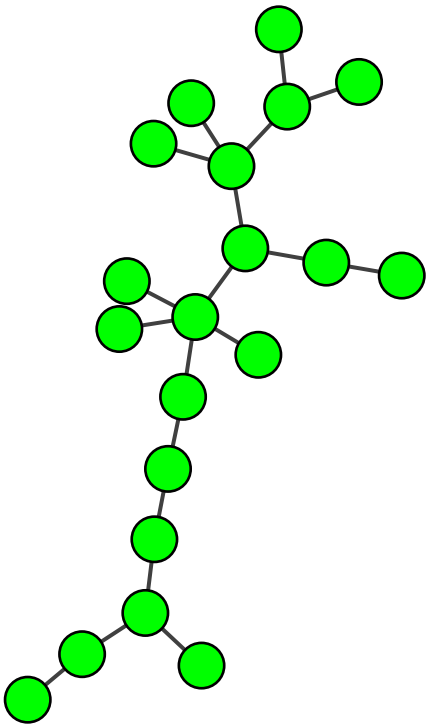
(a) Full graph

(b) Subgraph

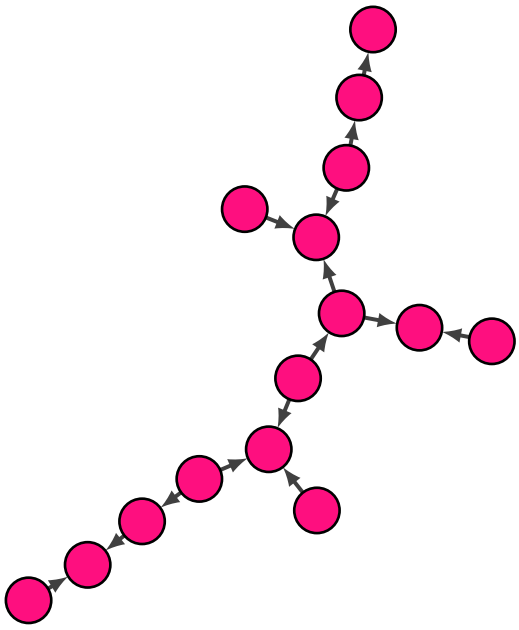


(c) Induced Subgraph

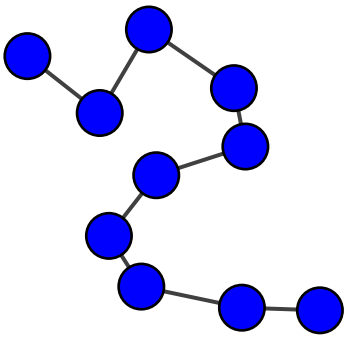
Figure 2.3: Example of a graph and its subgraphs. Note that (b) and (c) differ in the set of edges that are included in the subgraph.



(a) Tree



(b) Directed Acyclic Graph (DAG)



(c) Path

Figure 2.4: Examples of special graphs.

2.2 Learning with graphs

Tasks performed on graphs can be divided into three categories, depending on the type of object they target: nodes, edges, or graphs. The tasks performed on nodes and edges are usually supervised. They require training on a subset of the graph with labelled items, e.g, nodes or edges, and predict values on unlabelled items. For this reason, it is often called *semi-supervised* learning. Clustering , a central approach of unsupervised learning, is more often seen as a *community detection* task.

In this thesis we will not deal with unsupervised learning.

2.2.1 Node level tasks

Node-level tasks aim at predicting information about nodes. There are two ways this can be achieved: prediction on the node, e.g. classification or regression; or learning a node embedding that can be used in downstream applications. Given a node $u \in V$ and its features x_u, the task is to predict its label y_u.

Many examples of node classification appear in social networks. For example, given conversations on social media where some of the messages are labelled, we want to predict the content of the other messages. Another application is learning user behaviour to detect abnormal users, e.g., bots.

2.2.2 Edge level tasks

Edge-level tasks aim at predicting information concerning the relations between the nodes. In a social network, this can amount to finding which users are friends. For a large online retailer, the goal is to predict which products will be most commonly bought together. In biology, determining how two proteins will react together is achieved by doing link prediction on protein-protein interaction networks. This is useful for deciding if two drugs treatments can be taken simultaneously. More formally, given a subset of edges E_{train} of E, the goal is to predict the existence of the remaining edges, e.g. $E \setminus E_{train}$.

2.2.3 Graph level tasks

Graph-level tasks aim at prediction information concerning the entire graph. For example, given a molecule graph, we want to predict some of its properties, such as its toxicity, its solubility, etc. Another well known use case of graph level tasks is programming code analysis, in which we use the programme's representation, such as the AST (Abstract Syntax Tree), to determine its behaviour.

2.3 Machine learning with graphs

Prior to the advent of deep learning, graph-based learning relied on methods such as exploiting the graph statistics, random walks, spectral graph theory, and graph coloring. We present some of these methods, because they play a role in the design and the analysis of GNNs. For more detailed surveys, we refer to the following books Hamilton; Ma and Tang (2021).

2.3.1 Graph statistics and random walks

To gain information about a graph, we can compute features such as the node degree depicted in Equation 2.1, which indicates how many neighbours a node has. To refine this measure and take into account the centrality of the neighbours, we can compute the *eigenvector centrality*, represented in Equation 2.2. Taken over all the nodes, this equation can be rewritten in the form of Equation 2.3 where it becomes apparent that λ is an *eigenvalue* of the matrix A, and $\mathbf{e}$ an associated *eigenvector*. We can ensure the posivity of the eigenvector centrality because A is a real square matrix with positive entries; by virtue of Perron-Frobenius theorem Meyer and Stewart (2023), there is a unique real-valued eigenvalue of largest magnitude, and its eigenvector can be taken to be strictly positive.

A strictly positive vector must have only positive entries.

$$d_u = \sum_{v \in V} A_{uv} \tag{2.1}$$

$$e_u = \frac{1}{\lambda} \sum_{v \in V} A_{uv} e_v \tag{2.2}$$

$$\lambda \mathbf{e} = A\mathbf{e} \tag{2.3}$$

Beyond centrality, we can use other measures such as *clustering coefficient* which aims at quantifying how the nodes are clustered. This can be done at the global scale, e.g., as a measure of how the entire graph is clustered, or at the local scale. Equation 2.4 shows one version of a local clustering coefficient. This equation counts the number of triangles with u as one the vertices taken over all such possible triangles. More generally, this approach can be extended to other structures, such as stars or squares, etc.

The higher the clustering coefficient, the higher the nodes are clustered together.

This number represent the number of edges that exist between two distinct neighbours of u (the numerator in the equation). The size of the set of all possible triangles represents the number of pairs of distinct neighbourhoods we can form (the denominator in the equation.)

$$c_u = \frac{|(v_1, v_2) \in E : v_1, v_2 \in \mathcal{N}_u|}{\binom{d_u}{2}} \tag{2.4}$$

Instead of counting the exact number of paths or patterns in a graph, we can use *random walks* to get an approximation of the desired value. One of the biggest advantages of random walks is that it reduces the computational complexity required by exhaustive counts on a graph.

2.3.2 Graph spectral theory

In practice, it is common to use the *Laplacian* matrix L instead of the adjacency matrix. It is defined by Equation 2.5. The diagonal matrix D is the *degree matrix*. Each entry diagonal entry represents the degree of a vertix, i.e., D_{uu} is the degree of node u.

$$L = D - A \tag{2.5}$$

The Laplacian has many interesting properties. For instance, the multiplicity of the 0 is equal to the number of connected components of the associated graph. Instead of using the unnormalized version of the Laplacian, we can use the version, defined in Equation 2.6.

$$\tilde{L} = D^{-1/2} A D^{-1/2} \tag{2.6}$$

2.3.3 Graph isomorphism and the Weisfeiler-Leman algorithm

The problem of *graph isomorphism* can be formulated mathematically in the following way. Given two graphs G_1, G_2, they are said to be isomorphic if they have the same number of nodes and if there exists a permutation that maps each node of G_1 to a node in G_2, while preserving the structure, i.e., the edges.

One famous class of algorithms used for determining if two graphs are isomorphic is the Weisfeiler–Leman (WL) algorithm, also called 1-WL (Weisfeiler and Leman, 1968). It can be viewed as a *graph coloring* scheme. Let $c_i^{(t)}$ be the color of node i at step t. Then, the algorithm iterates according to Equation equation 2.7. Each node updates its color at step $t+1$ using its color and that of its neighbours at the previous step t. The mechanism used for the update is a *hash* function. A comprehensive review of the use of WL in machine learning is presented in (Morris et al., 2021).

$$c_v^{(t+1)} = HASH\left(\left(c_v^{(t)}, \left\{\left\{c_u^{(t)} \mid u \in \mathcal{N}_v\right\}\right\}\right)\right) \tag{2.7}$$

Chapter 3

Deep learning and graph neural networks

This chapter presents the main ideas behind deep learning models. It also introduces the concept of graph neural networks and offers an analysis of the expressiveness of GNNs. Expressiveness can be viewed from different points of views. It can be viewed as the ability to distinguish distinct graphs, e.g. differentiate between a toxic molecule and life-saving drug, between a pandemic outbreak and a seasonal flu, etc. Conversely, if two graphs are isomorphic, i.e., they are the same up to a permutation, expressiveness can be viewed in terms of the ability to assign the same result to the two graphs. More generally, since GNNs must be insensitive to the order of the nodes, we can measure how well they can approximate any permutation invariant functions.

This chapter is divided as follows. Section 3.1 describes deep learning and neural networks, while Section 3.2 focuses on graph neural networks in particular. Section 3.3 reviews the most expressive architectures of graph neural networks. Section 3.4 concludes the chapter.

This chapter, more specifically Section 3.3, was the subject of one publication Lachaud et al. (2022c).

Contents

3.1 Deep learning

3.1.1 Overview

Deep learning architectures are models that automatically learn representations of raw data. They consist in stacks of layers of parameters, interspersed with non-linear activation functions, as shown in Equation 3.1. Each layer ϕ_l, for $l = 1, \ldots, L$, can contain a non-linear activation function, like the ReLU or the sigmoid functions, and the symbol $\circ$ represents the composition operator. Layers

are divided in two categories: filters, which create an abstract representation of the input; and pooling layers, also known as coarsening or downsampling operators, which reduce the dimensions of the inputs as they go through the network. Figure 3.1 shows an example of a neural network. The inputs on the left are propagated throughout the layers. The main families of neural networks, such as multilayer perceptrons, convolutional neural networks (CNNs), recurrent neural networks (RNNs), graph neural networks (GNNs) are distinguished by the types of filters they use.

$$\Phi = \phi_L \circ \phi_{L-1} \circ \dots \phi_1 \tag{3.1}$$

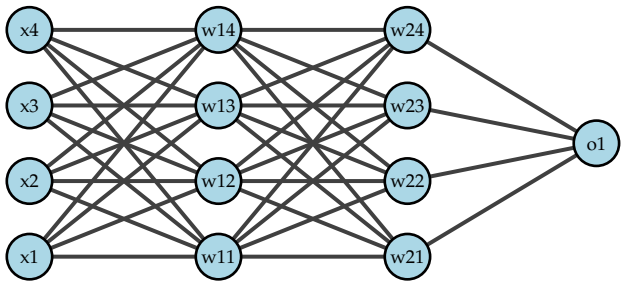


Figure 3.1: Example of a neural network. The x_i represent the inputs; o_1 is the output of the network.

Ingrained in most architectures is the idea of *shared weights*: the same filter is applied on different parts of the input. For instance, if a filter in a CNN acts as an edge detector, it will be applied throughout the image to detect all the edges Krizhevsky et al. (2017). Moreover, one of the reasons for the success of deep learning is the overparametrization of the networks: the models contain more parameters than the number of samples.

3.1.2 Forward and backward propagation

A neural network operates in two ways: a *forward pass* and a *backward pass*. During the forward pass, the model is fed inputs and produces outputs, such as classifying an image. The backward pass is used in the learning phase to adjust the parameters.

In practice, the forward pass is accomplished using matrices and tensors. This allows the models to be efficiently parallelized, meaning that several inputs can be processed at once, and the computations can sometimes be distributed upon several devices. Most deep learning libraries such as TensorFlow and PyTorch include native support for tensors.

In order to adjust the parameters, we use a *loss function* that computes how far our predictions are straying from the desired output. To decrease the loss, we want to move in the opposite direction of the gradient of the loss: this is the *gradient descent* algorithm. One of the key insights behind the success of deep learning is that the gradient can be *backpropagated* throughout the network using the *chain rule* of derivatives Lecun et al. (1998).

Because the computational graph forms a directed acyclic graph, the backpropagation of the gradients throughout the graph can be performed efficiently. When a neural network is created, a computational graph is created (specifically, a directed acyclic graph). Each element's gradient in the graph is already known. When the backward pass starts, the DAG is followed and the chain rule used to update the parameters.

3.1.3 Deep learning in practice

During the training of neural networks, there are a few reoccuring issues: the scalability of the models, exploding and vanishing gradients, overfitting. The depth of the networks, in particular creating very deep networks, is also an issue.

The speed gain obtained by performing computations on GPU instead of the CPU is such that most deep learning models are trained on GPU. Moreover, special processing units called TPU (Tensor Processing Units) have been designed for the whole purpose of neural network computation. When the training data exceeds the memory of the GPU, we have to resort to *mini-batch* learning, as opposed to *full-batch* learning. A fixed number of samples are randomly drawn from the training data. In this case, we talk about *stochastic gradient descent*, because the training produces an estimate of the gradient. In addition to meeting the memory constraints, mini-batch learning acts as a *regularizer* and helps reduce *overfitting* of the model to the training data.

Other means of regularization include the usual *weight penalization* schemes, i.e., imposing constraints such as l_1 or l_2 norms on the weights. An approach specific to deep learning is the use of *dropout*: during the training phase, neurons are randomly removed. This prevents a strong reliance on specific neurons.

Very deep networks, e.g., networks that have more than 100 layers, can be created using *skip-connections* introduced in He et al. (2016). Deep networks are faced with the problem of *vanishing* and *exploding* gradients. The deeper the network, the bigger the effect. To counteract this effect, one solution is to normalize the inputs and perform additional normalization at each layer. This solution has the added benefit of preventing the network from being too sensible to a single feature.

3.2 Graph neural networks

3.2.1 Overview

Graph neural networks first emerged in the context of extending recurrent neural networks to handle structured data (Sperduti and Starita, 1997). For more complete surveys of graph neural networks, see (Zhang et al., 2022; Wu et al., 2021).

The layers of a GNN are usually of two types (Ma and Tang, 2021): they can be *graph filters*, which operate on the nodes' hidden representations and produce new hidden representations. They behave according to Equation 3.2. The layers can also be *graph pooling layers*, in which case the graph is coarsened into a smaller graph. The pooling layers follow Equation 3.3.

$$H^{(l+1)} = \sigma_l \left(g_l(S, H^{(l)}) \right) \tag{3.2}$$

$$S^{(l+1)}, H^{(l+1)} = pool(S^{(l)}, H^{(l)}) \tag{3.3}$$

Here, g_l is a filter function that modifies the input signal $H^{(l)}$ while preserving the structure of the graph, represented by S; $pool$ is a function that reduces the node dimension of the graph, e.g., if $S^{(l)} \in \mathbb{R}^{n_l \times n_l}$, then $S^{(l+1)} \in \mathbb{R}^{n_{l+1} \times n_{l+1}}$

with $n_{l+1} < n_l$.

Most of the leading GNNs now follow a structure similar to the one introduced in (Gilmer et al., 2017): the hidden representation of a node is updated using the hidden representation of its neighbors. This framework, called the *Message-Passing Neural Network* (*MPNNs*) framework was also independently derived in Battaglia et al. (2018). MPNNs are sometimes called spatial GNNs because they rely on an aggregation scheme using the information coming from the neighbours of a node to update the node's representation. More formally, the MPNN graph filter takes the form of Equations 3.4 and 3.5.

$$m_v^{(l+1)} = AGGREGATE\left(\left\{\left\{h_u^{(l)} \mid u \in \mathcal{N}_v\right\}\right\}\right) \tag{3.4}$$

$$h_v^{(l+1)} = UPDATE\left(h_v^{(l)}, m_v^{(l+1)}\right) \tag{3.5}$$

$AGGREGATE$ is a function that maps the multiset of the neighbors' representations into a single vector, e.g., the *sum* operator; the $UPDATE$ operator can be a linear mapping of the concatenation of $h_v^{(l+1)}$ and $m_v^{(l+1)}$, e.g., $W \times \left[h_v^{(l+1)}, m_v^{(l+1)}\right]$ for some weight matrix W. $UPDATE$ can also be the sum of $h_v^{(l)}$ and $m_v^{(l+1)}$.

The other type of graph filters are often called spectral filters and rely on either the Laplacian L or the adjacency matrix A to update the representation of the nodes. These filters follow Equation 3.6 and are also commonly used in practice (Bruna et al., 2014; Defferrard et al., 2016; Wu et al., 2019). The function $p_l(S)$ is a polynomial of S, where $S = L$ or $S = A$, f_l can be a learnable function such as a neural network.

$$H^{(l+1)} = \sigma_l\left(p_l(S) f_l(H^{(l)})\right) \tag{3.6}$$

The distinction between spectral and spatial GNNs has more to do with the field of study from which the network is derived than in true differences between the architectures; indeed the authors in (Balcilar et al., 2021) show that both spectral and spatial GNNs can be expressed in terms of a more general framework. For more comprehensive reviews of all the GNN architectures, see (Wu et al., 2021; Zhang et al., 2022).

3.2.2 Permutation equivariance and invariance

Because the ordering in which the nodes are given is arbitrary, GNNs must be designed to make this order irrelevant (Bronstein et al., 2021). Mathematically, this translates to permutation invariance and equivariance: given a permutation matrix P, a function f is said to be *permutation invariant* if Equation 3.7 holds; likewise, f is said to be *permutation equivariant* if Equation 3.8 holds.

$$f(PXP^T) = f(X) \tag{3.7}$$

$$f(PXP^T) = Pf(X) \tag{3.8}$$

3.2.3 Expressiveness and Weisfeiler-Leman

Measuring the expressive power of GNNs serves two purposes: to find the type of tasks that GNNs can solve and the ones it cannot; and to compare architectures to find more expressive ones. Using GNNs instead of traditional neural networks such as Multi-Layer Perceptrons (MLPs), which have been adapted to handle graph structured data, is motivated by the fact that GNNs are exponentially more expressive. That is, increasing the number of layers of a GNN creates exponentially more equivalence classes of rooted graphs than it does for MLPs (Chen et al., 2021). Moreover, the depth and width of a GNN play a vital role in the expressiveness of the model. If the model is not wide enough or deep enough, there are some properties of a graph that it cannot capture, such as cycle detection, perfect coloring, and shortest path (Loukas, 2020).

When we analyze the expressiveness of a family of GNNs, the expressive power is usually represented in two different ways: the ability to distinguish non-isomorphic graphs, or the ability to approximate any permutation invariant function on graphs. The works of (Xu et al., 2019b; Morris et al., 2019) launched a vast area of research surrounding the expressiveness of GNNs, in terms of their limitations and the ways in which these limitations can be uplifted (Maron et al., 2019a).

While the Weisfeiler-Leman (see Section 2.3.3) is known to fail at distinguishing some graphs, it performs well in most cases (Cai et al., 1992). For this reason,

it is often used as the reference when determining the expressive power of a GNN. The two concurrent works (Xu et al., 2019b; Morris et al., 2019) proved that standard MPNNs, without node features, are at most as powerful as the 1-WL test. Additionally, (Xu et al., 2019b) proposed the Graph Isomorphism Network (GIN) and proved that the architecture is as powerful at the 1-WL test.

The WL tests can be extended to k-WL where $k \geq 2$ (Douglas, 2011). Instead of coloring a single node, we color tuples of size k. Let $v_i \in V^k$ be a k-tuple of $\mathcal{G}$, i.e. $v_i = (v_{i_1}, \ldots, v_{i_k})$ where $v_{i_j} \in V$ for $j \in [k]$. For the k-WL, we define the neighborhood of a tuple v_i to be

$$\mathcal{N}_j(v_i) = \left(\left\{(v_{i_1}, \ldots, v_{i_{j-1}}, u, v_{i_{j+1}}, v_{i_k}) \mid u \in V\right\}\right). \tag{3.9}$$

Similarly, for Folklore WL (k-FWL) (Douglas, 2011), a variant of the WL algorithms that uses a different update rule, the neighborhood of v_i is defined as

$$\mathcal{N}^F_u(v_i) = \left((u, v_{i_2}, \ldots, v_{i_k}), (v_{i_1}, u, \ldots, v_{i_k}), \ldots, (v_{i_1}, \ldots, v_{i_{k-1}}, u)\right). \tag{3.10}$$

Using these neighborhoods, the update rule for k-WL follows Equation 3.11, while k-FWL follows Equation 3.12.

$$c^{(t+1)}_{v_i} = HASH\left(\left(c^{(t)}_{v_i}, \left\{\left\{c^{(t)}_u \mid u \in \mathcal{N}_j(v_i), j \in [k]\right\}\right\}\right)\right) \tag{3.11}$$

$$c^{(t+1)}_{v_i} = HASH\left(\left(c^{(t)}_{v_i}, \left\{\left\{c^{(t)}_u \mid u \in \mathcal{N}^F_j(v_i), j \in [n]\right\}\right\}\right)\right) \tag{3.12}$$

3.3 Most expressive GNNs

In this section, we present an overview of the approaches used to improve the expressiveness of GNNs. We restrict ourselves to approaches that have mathematical theorems that prove that the architectures are indeed at least as expressive as the standard GNNs. We distinguish two main groups, which are represented in Figure 3.2. On the one hand, there are models that achieve the highest level of expressiveness by using higher order data, such as hypergraph data, at the cost of intensive computational requirements. On the other hand,

more recent models, while not as powerful as higher order methods, manage to be more expressive than standard GNNs while being computationally efficient, by using node identification or by incorporating graph substructure information in the model.

3.3.1 Higher Order Networks and Universal Approximation

Compared to a maximally expressive standard graph neural network such as the Graph Isomorphism Network (GIN) (Xu et al., 2019b), higher order networks gain expressiveness by incorporating knowledge about the hypergraph data, such as hyperedges between sets of nodes (Morris et al., 2019). Moreover, instead of building the layers in the network to be permutation invariant, non-invariant functions can be used then summed over all the set of permutations to produce permutation invariant functions (Murphy et al., 2019).

Building network layers that work on tuples of nodes instead of single nodes requires using tensors of higher dimension. Instead of having an input feature matrix $X \in \mathbb{R}^{n \times d}$ with an adjacency matrix $A \in \{0,1\}^{n \times n}$, a hyper-graph can be represented using a tensor $\mathsf{X} \in \mathbb{R}^{n^k \times d}$ (Maron et al., 2019b). In this manner, X_i represents the features of node i, $\mathsf{x}_{i,j}$ of edge (i,j), $\mathsf{X}_{i,j,l}$ of the hyper edge (i,j,l), and so on.

Because the ordering of the nodes is arbitrary, the GNN layers should be designed to be either permutation equivariant or invariant. A composition of an equivariant layer with an invariant layer leads to an invariant function. In (Maron et al., 2019b), the authors characterize all such types of linear layers. Namely, given a permutation matrix P and a function vec that vectorizes a tensor, a linear layer $L \in \mathbb{R}^{1 \times n^k}$ is invariant if and only if it follows Equation 3.13:

$$P^{\bigotimes k} \mathrm{vec}(L) = \mathrm{vec}(L) \tag{3.13}$$

where $\bigotimes$ is the Kronecker product. Similarly, a linear layer $L \in \mathbb{R}^{n^k \times n^k}$ is equivariant if and only if it follows Equation 3.14.

$$P^{\bigotimes 2k} \mathrm{vec}(L) = \mathrm{vec}(L) \tag{3.14}$$

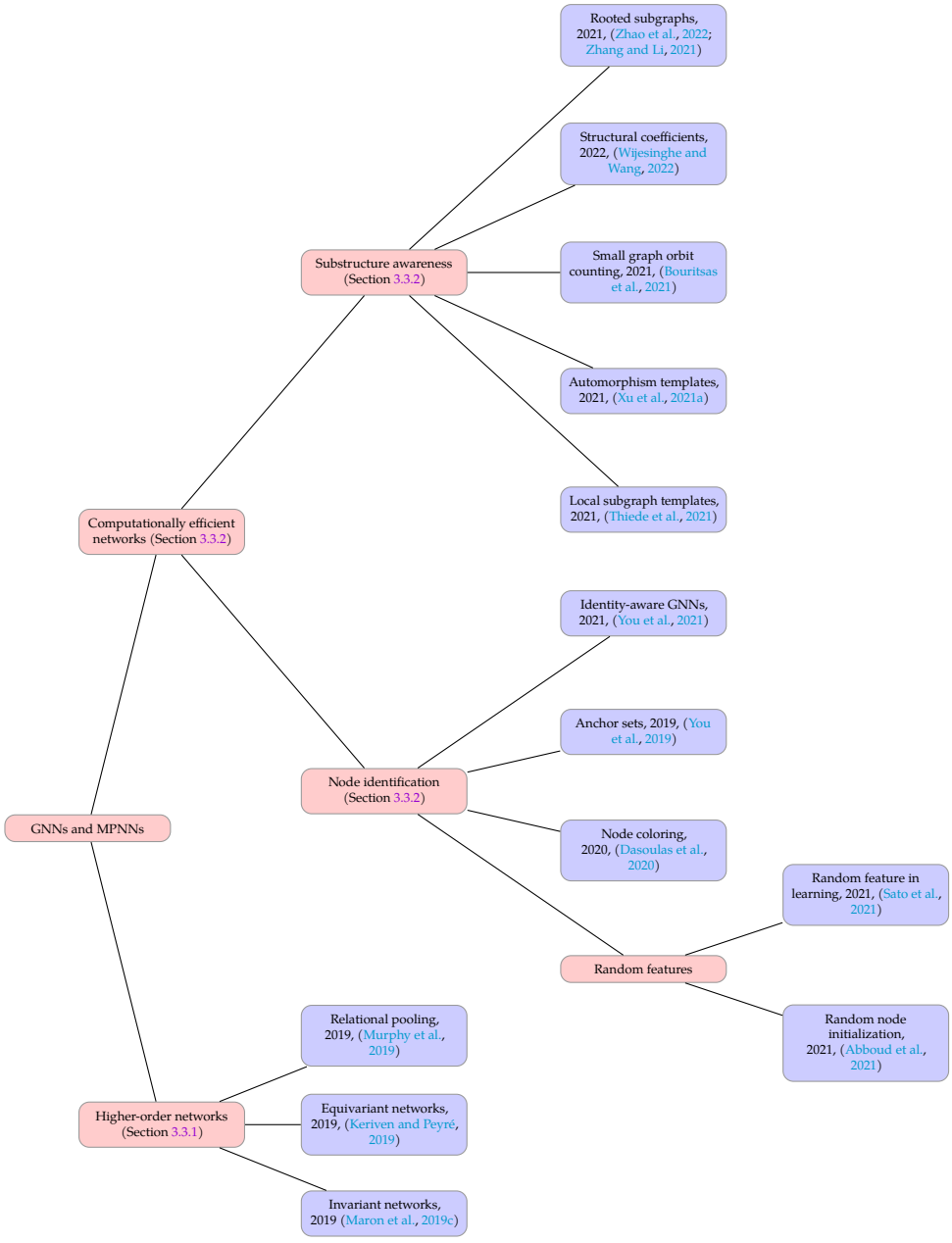


Figure 3.2: Overview of mathematically expressive GNNs. Red boxes refer to sections of the chapter; and blue boxes represent ideas introduced by specific papers.

The authors (Maron et al., 2019b) further provide a basis for the space of invariant and equivariant layers, alongside their dimension.

Instead of trying to directly create an invariant layer, one can use arbitrary functions and sum over all the permutations. This was first proposed in (Murphy et al., 2019). Given $X_{features,id}$ the feature matrix concatenated with a one-hot encoding of a position of the node, f a GNN, a Relational Pooling GNN (RP-GNN) layer follows Equation 3.15. π is a permutation of the nodes of $\mathcal{G}$. The one-hot encoding added to X is permuted with π while X remains fixed. This prevents the sum from reducing to a single element. Furthermore, selecting the permutations on which to perform the sum can eliminate the factorial complexity induced by all the permutations.

$$f_{RP}(G) = \frac{1}{|\mathcal{V}|!} \sum_{\pi \in \Pi_{|\mathcal{V}|}} f(\pi(A), X_{features,\pi(id)}) \tag{3.15}$$

Building equivariant and invariant layers raises a question: can the network approximate any invariant or equivariant function, i.e., can GNNs act as universal approximators? Provided that the tensors have a high enough dimension, (Maron et al., 2019c) proved that models based on the linear layers defined above can indeed approximate any invariant functions. Specifically, networks using layers that are equivariant or invariant for a group G can be expressed as in Equation 3.16:

$$\phi = m \circ h \circ \sigma_l(L_d) \circ \cdots \circ \sigma_1(L_1) \tag{3.16}$$

where m is a multi-layer perceptron that flattens the output, h is a function that is invariant for the group G, and L_i are the equivariant layers for the group G that follow Equation 3.14.

Similarly, in addition to providing a different proof of the result from (Maron et al., 2019c) regarding the universality of the linear layers following Equation 3.13 with respect to invariant functions, the authors of (Keriven and Peyré, 2019) show that, given a sufficient tensor size, equivariant networks can approximate any equivariant function.

In terms of graph isomorphism, k-order GNNs, that is, GNNs that use a k-order tensor as input such as hypergraphs, are more expressive than GNNs.

Specifically, since k-WL are known to be strictly more powerful than $(k-1)$-WL for $k \geq 2$, and k-WL and $(k-1)$-FWL have the same expressive power, adapting k-WL for GNNs leads to more expressive GNNs (Maron et al., 2019a; Morris et al., 2019). Thus, k-order GNNs are as powerful as k-WL (Maron et al., 2019a).

The problem with k-order GNNs is that they are computationally expensive. Universality results for computationally reasonable GNNs were first proven in (Azizian and Lelarge, 2021), additionally showing that Folklore GNNs are the most expressive for a given k.

To design GNNs that are more powerful than 1-WL while still being computationally efficient, we must exploit node properties such as their identification and the substructures that they belong to.

3.3.2 Computationally Efficient and Powerful Networks

Instead of exploiting higher order knowledge, GNNs can obtain higher expressiveness than the GIN architecture by adding information about the node being updated. This can take several forms: adding features to the nodes to make them identifiable (Dasoulas et al., 2020); using the structure of the local subgraphs to adapt the way information is transmitted (Thiede et al., 2021); using rooted graphs instead of trees centered around the nodes being updated.

Instead of using higher-order networks, finding more expressive architectures than standard GNNs can be done by looking at the failure cases of MPNNs. Most of the failures can be attributed to two related causes: the anonymity of nodes in the message passing stage, where it is impossible to keep track of where the information is coming from (Loukas, 2020); and the lack of substructure awareness from MPNNs (Chen et al., 2020).

Mitigating node anonymity is the subject of Section 3.3.2, while injecting substructure awareness into MPNNs is discussed in Section 3.3.2.

Node Identification

MPNNs cannot distinguish information coming from identical nodes (i.e., nodes with the same features and same degree). One way to remedy this problem is to color each identical node with a different color. This is the process introduced in (Dasoulas et al., 2020) with the k-CLIP (Colored Local Iterative Procedure) algorithm. All the nodes with identical attributes are mapped into groups $V_1, \ldots, V_K$. Within each group, each node is assigned a distinct color, by concatenating the color attribute, e.g., a one-hot encoding, with the features of the node. k different coloring $\mathcal{C}_k$ are chosen out of all the possible colorings. A standard MPNN is trained for each coloring, and the final output is given by Equation 3.17:

$$h_G = \psi\left(\max_{c \in \mathcal{C}_k} \sum_{v \in \mathcal{V}} h^c_{v,L}\right) \tag{3.17}$$

where h_G is the graph readout of the network, ψ is a learnable function, and $h^c_{v,L}$ is the hidden representation of node v at the last layer L of the network using the coloring c. Compared to a standard MPNN, the complexity of k-CLIP has an extra k factor that corresponds to the number of colorings used. CLIP can be used with all the $\prod_{k=1}^{K} |V_k|$ possible colorings, in which case it is named ∞_{CLIP}. While the ∞_{CLIP} is a universal approximator (Dasoulas et al., 2020), it suffers from an exponential growth with respect to the size of the $V_k, k \in [K]$. k-CLIP is a random algorithm, e.g., two different runs might lead to two different colorings. It is more efficient than ∞_{CLIP}, and universality results can still be obtained for its expectancy (Dasoulas et al., 2020).

When a GNN has a sufficient number of layers, a node is feeding back information to itself in its update: in an undirected graph, a node is a neighbor of its neighbor. For example, given the graph in Figure 3.3a and the trees of height 1 starting from each of its nodes in Figure 3.3b, B is a neighbor of A. With a standard MPNN, A will be used to update B, which in turn will be used to update A, with no indication of the provenance of the information. Therefore, as shown in (You et al., 2021), graphs with different structures can have the same GNN computational graph. To alleviate this problem, the authors propose to introduce identity-aware GNNs (ID-GNNs), where they use ego networks (rooted graphs) in which the root is colored and thus can be identified in the computational graph. Figure 3.3c shows the ego networks of height 1 for each of

the nodes in the graph of Figure 3.3a. An ID-GNN layer follows Equations 3.18 and 3.19, where $MESSAGE_0$ and $MESSAGE_1$ can be MLPs, attention mechanisms, etc. $MESSAGE_0$ is the message function for neighbors of the root node, while $MESSAGE_1$ is the message function to update the root node. This set of equations are similar to Equations 3.4 and 3.5, except that the root node of the ego graphs is identified and its message is used differently than for the rest of the nodes. There is almost no added complexity compared to a standard MPNN: by setting $MESSAGE_0 = MESSAGE_1$, we recover a standard MPNN. However, ID-GNNs can differentiate graphs that a Graph Isomorphism Network (Xu et al., 2019b) cannot, making them strictly more powerful than the 1-WL test (You et al., 2021).

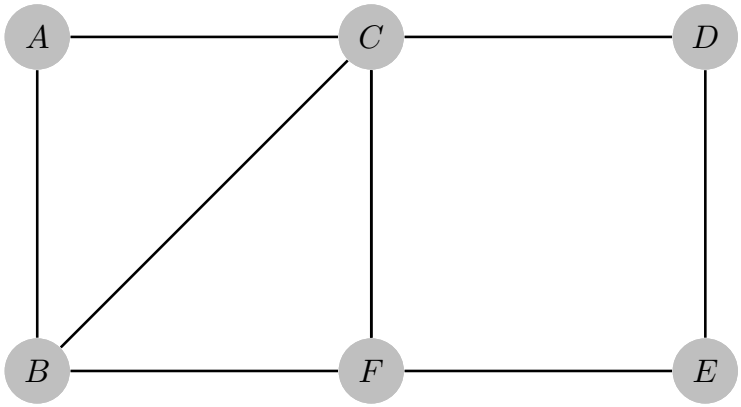
$$m_v^{(l+1)} = AGGREGATE\left(\left\{\left\{MESSAGE_0(h_u^{(l)}) \mid u \in \mathcal{N}_v\right\}\right\}\right) \tag{3.18}$$

$$h_v^{(l+1)} = UPDATE\left(MESSAGE_1(h_v^{(l)}), m_v^{(l+1)}\right) \tag{3.19}$$

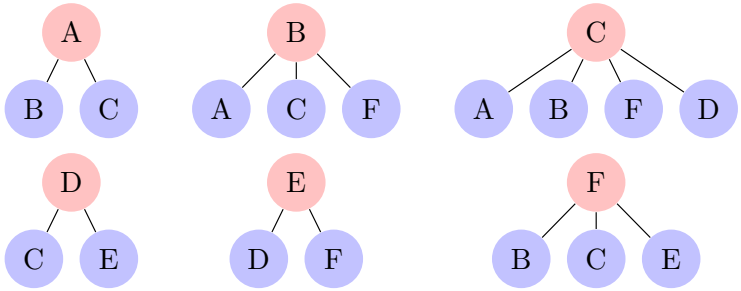
A simpler approach consists in simply assigning a random feature to each node at the beginning of the learning phase. Namely, the node feature matrix X is concatenated with a matrix $R \in \mathbb{R}^{N \times d_r}$ where R is a feature matrix that was sampled from a random distribution. rGIN, the architecture obtained by adding the random feature assignment to a standard GIN, can distinguish graphs that GIN cannot (Sato et al., 2021). The random features ensure that graphs that would lead to the same GNN computational graphs otherwise, will produce different graphs most of the time. Additionally, random node initialization allows MPNNs to become universal approximators without requiring higher order tensors (Abboud et al., 2021).

Instead of using the neighborhood of each node, one can use anchor sets, which consist of nodes sampled from the graphs. In (You et al., 2019), the authors propose the Position-aware Graph Neural Network. At each layer, k sets $S_i, i \in [k]$ of nodes are chosen. Then, for each node v, a message $m_{v,i}$ is computed between v and the nodes in S_i, as in Equation 3.20. Finally, the hidden representation is updated by aggregating the messages from all the anchor sets, as shown in Equation 3.21.

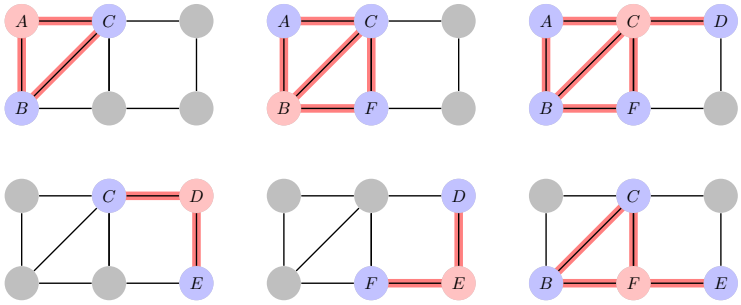
$$m_{v,i}^{(l)} = AGGREGATE\left(\left\{\left\{h_u^{(l)} \mid u \in S_i\right\}\right\}\right) \tag{3.20}$$



(a) A graph with 6 nodes. A, D, and E have degree 2, B and F have degree 3 and C has degree 4.



(b) Rooted trees with depth 1 for each node. If the nodes have identical features and are not identified, the computational tree of MPNN for A, D, and E are the same.



(c) Rooted subgraphs with depth 1 for each node. Compared with the routed trees, the A rooted graph is distinguishable from D.

Figure 3.3: Example of a graph with its rooted trees and rooted subgraphs. The rooted subgraphs incorporate structural information that is lost in rooted trees. MPNNs use rooted trees to update notes. If the features are identical, MPNNs with a single layer will treat nodes B and F as the same; if the network uses the rooted graph instead, it will distinguish B and F.

$$h_v^{(l+1)} = AGGREGATE\left(\left\{\left\{m_{v,i}^{(l)} \mid i \in [k]\right\}\right\}\right) \tag{3.21}$$

Removing the dependency on the neighborhood of the nodes changes the objective function of the network. Following (You et al., 2019), the representation learning objective of a GNN is written in Equation 3.22. ϕ represents the neural network parameterized by θ. Nodes u and v are sampled according to $\mathcal{V}_{train}$, the distribution of nodes in the training set. S_u is the q-hop neighborhood graph of u, parameterized by q, the maximum distance to u. S_u and S_v are sampled according to $p(\mathcal{V})$, the distribution of the set of nodes in the graph. d_z is a similarity metric, and d_y is a target similarity metric. By contrast, the learning objective of a P-GNN is written in Equation 3.23, where S is an anchor set, sampled from the distribution $p(\mathcal{V})$. P-GNN can share information across the whole graph using common anchors between nodes, while a standard GNN is restricted to the nodes in the neighborhood. This makes P-GNN able to approximately capture properties that GNNs cannot capture, such as the shortest paths in the network.

$$\min_\theta \mathbb{E}\left[\mathcal{L}(d_z(\phi_\theta(u, S_u), \phi_\theta(v, S_v)) - d_y(u, v))\right] \tag{3.22}$$

$$\min_\theta \mathbb{E}\left[\mathcal{L}(d_z(\phi_\theta(u, S), \phi_\theta(v, S)) - d_y(u, v))\right] \tag{3.23}$$

In place of identifying each node, the expressiveness of MPNNs can also be improved by making them aware of the substructures found in the graph.

Substructure Awareness

One area of interest is whether MPNNs can count substructures. That is, given a graph structure or pattern, can an MPNN count the number of times that such structure, up to isomorphisms, appears in the graph? The authors of (Chen et al., 2020) show that MPNNs cannot count patterns with three or more nodes. However, MPNNs can perform subgraph-count of star-shaped patterns. Looking at k-WL tests, the authors further show that finite k-WL cannot perform an induced-subgraph-count of patterns that have more than a given number of nodes.

MPNNs rely on a star-shaped aggregation pattern: they aggregate information coming from neighbors to a central node (see Equations 3.4 and 3.5). For

example, in Figure 3.3a, node C and E are treated in the same way to update node F. However, C and E do not have the same structural information, as C is part of a triangle with B and F, while E forms one of the endpoints of a path of length 2 with F. To capture this information, (Thiede et al., 2021) proposes the Autobahn architecture. Given a list of template graphs, at each layer, the network decomposes the graph into a collection of subgraphs that are isomorphic to the templates. A single neuron in the Autobahn is applied to subgraphs that match the list of templates. The activation from the different templates are combined using *narrowing* and *promotion* functions that allow us to, respectively, extend or decrease the number of nodes that an activation function takes. The authors (Thiede et al., 2021) argue that if the templates are carefully chosen, Autobahn can match the expressiveness of higher order networks.

In a similar work, the authors of (Xu et al., 2021a) propose the GRaph Automorphic Equivalence (GRAPE) network, which uses automorphism groups in a similar manner to Autobahn (Thiede et al., 2021). They focus on template search using genetic algorithms: templates are produced from a pool, and mutated via edge mutation or node mutation until a satisfactory template has been produced. The authors (Xu et al., 2021a) further show that GRAPE can distinguish certain graphs that MPNNs cannot distinguish.

By contrast, in (Bouritsas et al., 2021), the authors propose using a list of small connected graphs to directly add structural features to nodes and edges by counting the number of times a node (or an edge) acts as a member of an orbit of one the graphs. For example, in Figure 3.3c, node C acts as a point in a triangle for the A rooted graph, but as one end of a path of length 3 in the D rooted path. The structural features can be concatenated with the original features and a GNN is trained on the new features. The authors (Bouritsas et al., 2021) show that under certain conditions on the subgraph matching, this type of architecture, called Graph Substructure Networks (GSN), can be strictly more powerful than MPNNs and 1-WL.

On the topic of substructures, a new hierarchy of local isomorphisms between subgraphs is proposed in (Wijesinghe and Wang, 2022): subgraph isomorphism, overlap isomorphism and subtree isomorphism. Let S_u be the neighborhood subgraph of u. It is the subgraph induced by $\tilde{\mathcal{N}}_u = \mathcal{N}_u \cup u$. The overlap subgraph between two adjacent vertices u and v is defined by $S_{uv} = S_u \cap S_v$. Let $\mathcal{S} = \{S_v | v \in \mathcal{V}\}$ and $\mathcal{S}^* = \{S_{vu} | (v, u) \in \mathcal{E}\}$. *Structural coefficients* for nodes and

their neighbors can be obtained by defining a function $\omega \in \mathcal{S} \times \mathcal{S}^* \to \mathbb{R}$ that have the following properties: (1) *local closeness*: $\omega(S_v, S_{vu}) > \omega(S_v, S_{vu'})$ if S_{vu} and $S_{vu'}$ are complete graphs and S_{vu} has more vertices than $S_{vu'}$; (2) *local denseness*: $\omega(S_v, S_{vu}) > \omega(S_v, S_{vu'})$ if S_{vu} and $S_{uv'}$ have the same number of vertices, but S_{vu} has more edges; (3) *isomorphic invariant*: $\omega(S_v, S_{vu}) > \omega(S_v, S_{vu'})$ if S_{vu} and $S_{vu'}$ are isomorphic.

Based on these criteria, the authors in (Wijesinghe and Wang, 2022) propose the GraphSNN architecture. It follows Equations 3.24–3.26. Equation 3.27 is an example of a valid ω function, parameterized by $\lambda > 0$. $|\mathcal{V}_{vu}|$ is the number of vertices of $S_u v$, and $|\mathcal{E}_{vu}|$ its number of edges. Furthermore, the $\omega(S_v, S_{vu})$ can be normalized for the network operations. With this ω, GraphSNN is more powerful than 1-WL (Wijesinghe and Wang, 2022).

$$m_a^{(l)} = AGGREGATE_0\left(\left\{\left\{\left(\omega(S_v, S_{vu}), h_u^{(l)}\right) \mid u \in \mathcal{N}_v\right\}\right\}\right) \tag{3.24}$$

$$m_v^{(l)} = AGGREGATE_1\left(\{\{(\omega(S_v, S_{vu}) \mid u \in \mathcal{N}_v\}\}\right) h_v^{(l)} \tag{3.25}$$

$$h_v^{(l+1)} = UPDATE(m_a^{(l)}, m_v^{(l)}) \tag{3.26}$$

$$\omega(S_v, S_{vu}) = \frac{|\mathcal{E}_{vu}|}{|\mathcal{V}_{vu}| \cdot |\mathcal{V}_{vu} - 1|} |\mathcal{V}_{vu}|^{\lambda} \tag{3.27}$$

Alternatively, instead of relying on structural coefficients or graph templates, we can exploit the rooted subgraphs of each node. In this setting, we employ a *network uplifting* scheme where a base GNN is used to compute node representations, which are fed to an outer GNN. In (Zhao et al., 2022), the authors use rooted graphs to compute three types of information: the centroid encoding, which is obtained when the node is the root of the rooted graph; the subgraph encoding, which represents the information from the other nodes in the rooted graph; and the context encoding, which represents the information that the node carries in the other rooted graphs. Let $G^{(l)}[\mathcal{N}_k(v)]$ be the v rooted graph with height k, with the node representations from layer l. Let $GNN^{(l)} = POOL_{GNN^{(l)}}(EMB^{(l)}(i|G^{(l)}[\mathcal{N}_k(v)])|i \in \mathcal{N}_k(v))$ be the inner GNN, with $POOL_{GNN^{(l)}}$ the pooling operator after the last layer of the GNN, and $EMB^{(l)}(i|G^{(l)})$ the embeddings before the pooling operation of the GNN. Let $d_{u|v}^{(l)}$ be the encoding of distance from node u to v at layer l (Li et al., 2020). Let σ be the sigmoid function, $\odot$ the element-wise product. The GNN As Kernel

(GNN-AK) architecture follows Equations 3.28–3.31. The authors (Zhao et al., 2022) prove that this GNN-AK is strictly more powerful than 2-WL and not less powerful than 3-WL.

$$h_{v,subgraph}^{(l+1)} = POOL_{GNN}\left(\left\{\left(\sigma\left(d_{u|v}^{(l)}\right) \odot EMB\left(i \mid G^{(l)}[\mathcal{N}_k(v)]\right)\right), i \in \mathcal{N}_k(v)\right\}\right) \tag{3.28}$$

$$h_{v,centroid}^{(l+1)} = EMB\left(v \mid G^{(l+1)}[\mathcal{N}_k(v)]\right) \tag{3.29}$$

$$h_{v,context}^{(l+1)} = POOL_{CONTEXT}\left(\left\{\left(\sigma\left(d_{u|v}^{(l)}\right) \odot EMB\left(v \mid G^{(l)}[\mathcal{N}_k(u)]\right)\right), v \in \mathcal{N}_k(u)\right\}\right) \tag{3.30}$$

$$h_v^{(l+1)} = UPDATE\left(h_{v,subgraph}^{(l+1)}, h_{v,centroid}^{(l+1)}, h_{v,context}^{(l+1)}\right) \tag{3.31}$$

WL and MPNNs encode a rooted subtree for each node. This is illustrated in Figure 3.3a,b. The nodes in red in Figure 3.3b are updated using information from the nodes in blue. Information about edges between the nodes in blue is lost. This might prevent the network from learning useful information at the graph level. Instead, (Zhang and Li, 2021) proposes to learn subgraph representations centered around rooted graphs. Let G_v^h be be the rooted graph of v with height h, and $\mathcal{N}(v|G_w^h)$ the neighborhood of v within w's rooted subgraph. A layer of Nested Graph Neural Network (NGNN) follows Equations 3.32 and 3.33. In the last layer, the representation of node w is obtained by performing a readout, as shown in Equation 3.34 where L is the last layer and R_0 is the readout function. These new representations can be used as an input to a second GNN to perform graph-level tasks. These nested architectures are strictly more powerful than 1-WL (Zhang and Li, 2021).

$$m_{v,G_w^h}^{(l+1)} = AGGREGATE\left(\left\{\left\{h_{u,G_w^h}^{(l)} \mid u \in \mathcal{N}(v \mid G_w^h\right\}\right\}\right) \tag{3.32}$$

$$h_{v,G_w^h}^{(l)} = UPDATE\left(h_{v,G_w^h}^{(l)}, m_{v,G_w^h}^{(l+1)}\right) \tag{3.33}$$

$$h_w = R_0\left(h_{v,G_w^h}^{L} \mid v \in G_w^h\right) \tag{3.34}$$

While graphs may be difficult to distinguish, it is usually easier to distinguish subgraphs. With this in mind, (Bevilacqua et al., 2022) propose using bags of subgraphs that can then be readout, and on which a set operation can be performed. There can be different policies in selecting the subgraphs. These methods are more powerful than 1-WL.

3.4 Discussion

In this section, we summarise the results concerning the expressiveness of GNNs, and we provide outlooks for future works.

3.4.1 Summary

The early works regarding GNN expressiveness were focused on two goals: providing bounds for MPNNs, and overcoming those bounds. The works of (Xu et al., 2019b; Morris et al., 2019) established Weisfeiler–Leman tests as the standard of comparison when it comes to expressiveness of GNNs. The higher-order Weisfeiler–Leman tests provide a template for higher-order GNNs (Douglas, 2011).

Higher-order methods also arise when trying to design GNN layers that are universal approximators or invariant (Maron et al., 2019c) or equivariant functions (Keriven and Peyré, 2019). To this end, hypergraphs with tensors containing hyperedges data are required (Maron et al., 2019b).

With higher-order networks, the complexity grows exponentially with the number of nodes. This makes these networks computationally expensive, and alternatives must be crafted to handle large graphs. Approaches with low overhead compared to a standard MPNNs include some form of node identification. This can be achieved by adding colors to each node (Dasoulas et al., 2020), using rooted graphs (You et al., 2021), fixing sets of nodes that are used to update all the nodes in the graph (You et al., 2019), or by adding random features to the nodes (Abboud et al., 2021; Sato et al., 2021).

To retain part of the expressiveness of higher-order networks, subgraphs can be used as templates on which to perform the convolutions. This breaks the star-shaped pattern of MPNN convolutions and allows for other patterns such as triangles, circles, etc. A network with a good selection of templates can be viewed as a higher-order network where only a few substructures are used (Thiede et al., 2021). These templates can be searched via genetic algorithms (Xu et al., 2021a).

Substructures can also be used to add structural information to the nodes. This

can be achieved by counting the roles a node plays in different templates and adding this information to the features (Bouritsas et al., 2021). Alternatively, structural coefficients which take into account the shapes of neighborhoods can inject structural information into the network (Wijesinghe and Wang, 2022).

On top of the previous methods, GNNs can be used a building block for other GNNs. An inner GNN using rooted graphs can be used to produce hidden representations that are fed into an outer GNN (Zhang and Li, 2021). Moreover, each node can be updated using three types of information: the centroid information, when the node is the root of the rooted graph; the subgraph information, which is the information coming from its neighbors; and the context information, which is the information it contributes to rooted graphs where it is not the root (Zhao et al., 2022).

All the expressiveness results of the architectures discussed in this chapter are presented in Table 3.1. The table contains only results that have a mathematical proof in the associated paper. In some cases, this could mean that higher bounds can exist that have not been proven. Conversely, new architectures that are developed today could have a high expressiveness, but since no mathematical proof is provided, the evidence is only empirical.

3.4.2 Future work

To the best of our knowledge, there has yet been no papers investigating the expressive power of methods that combine the methods presented in Section 3.3.2. For example, can the gains made in mathematical expressiveness from adding node identifiers be combined with the ones made from using nested graph neural networks? Or do these gains have the same root, rendering the combination as powerful as either of the methods individually?

Another way to improve expressiveness is to extend GNNs to more complex structures such as cell complexes, of which graphs are special cases (Bodnar et al., 2021). Graphs can be seen as part of a bigger framework of geometric objects, which can be worked on using *geometric deep learning* (Bronstein et al., 2021). A possible line of work would be to explore other types of structures that share profound links with graphs, and create efficient architectures that can be adapted to graphs.

Table 3.1: Expressiveness of GNNs. Expressiveness is given with respect to how the authors proved the results. GIN corresponds to the most powerful standard MPNN.

Architecture	**Expressiveness**
GIN (Xu et al., 2019b)	1-WL
k-GNN (Morris et al., 2019)	(k-1)-WL
RP-GNN (Murphy et al., 2019)	strictly superior to GIN
PPGN (Maron et al., 2019a)	3-WL
∞_{CLIP} (Dasoulas et al., 2020)	universal approximator
ID-GNN (You et al., 2021)	>1-WL
rGIN (Sato et al., 2021)	>1-WL, universal approximator
PGNN (You et al., 2019)	greater than MPNN
Autobahn (Thiede et al., 2021)	depends on the templates, can achieve k-GNN performance
GRAPE (Xu et al., 2021a)	stricly more powerful than MPNN
GSN (Bouritsas et al., 2021)	more powerful than MPNN and 1-WL under conditions.
GraphSNN (Wijesinghe and Wang, 2022)	$> 1 - WL$
GNN-AK (Zhao et al., 2022)	>2-WL, $\geq$ 3-WL
NGNN (Zhang and Li, 2021)	$>$ 1-WL

Part II

Learning

Chapter 4

GNNs in practice

This chapter introduces the most frequently used GNNs in real world application, as well as the datasets on which our experiments are performed. The most used GNNs differ from the most expressive ones because of the computational cost one the most expressive GNNs is prohibitive when we want to perform classification on large graphs. Additionally, standard GNNs can outperform the most expressive architectures on real world datasets.

This chapter is divided as follows. Section 4.1 presents the most commonly used GNN architectures, as well as some of the tricks that can be used to improve their performance. Section 4.2 presents the dataset we are using in our experiments, discusses the splitting of data, the difference between transductive and inductive learning, and the choice of node features.

Contents

4.1 GNNs in practice

In the first part of this thesis, we focused on the theoretical background of deep learning and graph neural networks. In this part, we present more empirical results. In particular, this first chapter takes a look at empirically and historically important graph neural networks and datasets.

Some of the more important advances in the field of GNNs include the formalization of the generic architecture of a GNN and the exploration of various neighbourhood selection schemes. Furthermore, the development of new architectures has mostly relied on adapting successful techniques from other domains, such as attention mechanisms and knowledge distillation, or on exploiting unique properties of graph stuctured data, such as reversible GNNs or GNNs that leverage information from multihop neighbours. Another particularity of GNNs is the possibility to leverage the graph information in the design of the features. Besides, the dependencies between nodes play a role in how a graph is split between training and test data, which can affect model performance. In the rest of this section we present the main concepts that will be used throughout the paper.

4.1.1 Message Passing Neural Framework

As presented in the first part of this thesis, most GNNs follow the message passing neural network (MPNN) framework (see Section 3.2.1) introduced in Gilmer et al. (2017). The network aggregates information about the neighbors of a node to produce messages that are used to update the hidden representation of each node. More formally, let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ the set of edges. h_v^l represents the features of a node v at the layer l, and H^l the feature matrix for all the nodes. e_{vw} represents the edge from node v to node w. If X denotes the node features matrix, we have by convention $H^0 = X$. An update from layer l to layer $l+1$ takes the form

$$m_v^{l+1} = \sum_{w \in \mathcal{N}_v} M_l(h_v^l, h_w^l, e_{vw}) \tag{4.1}$$

$$h_v^{l+1} = U_l(h_v^l, m_v^{l+1}) \tag{4.2}$$

where M_l and U_l are respectively the message and update functions of layer l. The changes compared to Equations 3.4 and 3.5 from Section 3.2.1 are the addition of the e_{vw} in the message computation, and the use of the sum (Σ) operator, instead of the more generic AGGREGATE function. While the theoretical guarantees exposed in the first part of the thesis hold for any aggregate function, in practice the most commonly used function is the sum operator.

The three architectures, GCN, GAT, and GRAPHSAGE, that we describe next play an important role in the evolution of graph neural networks. They took the field away from signal processing and into the deep learning era; they added the paramount *attention* mechanism, and started the sampling thread that allowed the architectures to scale to large graphs with more than a billion of nodes.

4.1.2 Graph Convolutional Networks

Graph Convolutional Networks (GCNs) were introduced in Kipf and Welling (2017). They are one of the earliest forms of MPNNs. They extend the concept of spectral convolutions on graphs by approximating the convolution with Chebyshev polynomials Defferrard et al. (2016). A GCN layer follows Equation 4.3. σ represents an activation function, such as the ReLU function. $\tilde{D}$ is the nor-

malized degree matrix, $\tilde{A}$ the normalized adjacency matrix. W^l represents the parameters of the layer.

$$H^{l+1} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^l W^l\right) \tag{4.3}$$

A GCN is usually composed of several of these layers, with the activation function of the last layer being a softmax to output probabilities.

4.1.3 Graph Attention Networks and attention mechanisms

Attention mechanisms were introduced in Bahdanau et al. (2015) in the context of natural language processing, with the goal of learning the most important parts of a sentence, e.g., which part of the sentence contains the meaning. Inspired by their success, they were imported to GNNs with the introduction of the Graph Attention neTwork (GAT) in Velickovic et al. (2018) and remain part of most of the leading architectures Chien et al. (2021); Sun and Wu (2020); Sun et al. (2021). The update rule is presented in Equation 4.4.

$$h_u^{(l+1)} = \sigma\left(\sum_{v\in\mathcal{N}_u} \alpha_{uv}^{(l)} W^{(l)} h_v^{(l)}\right) \tag{4.4}$$

$\alpha_{uv}^{(l)}$ is the *normalized attention coefficient* at layer l of node v with respect to u, that is, it indicates how important the features of node v are to node u. The coefficients are retrieved by computing the attention coefficients then performing a softmax for normalization. More formally, with $e_{uv}^{(l)}$ the attention coefficients at layer l and a the attention mechanism, $\alpha_{uv}^{(l)}$ and $e_{uv}^{(l)}$ can be computed using Equations 4.5 and 4.6.

$$e_{uv}^{(l)} = a\left(W^{(l)}h_u^{(l)}, W^{(l)}h_v^{(l)}\right) \tag{4.5}$$

$$\alpha_{uv}^{(l)} = \frac{\exp(e_{uv}^{(l)})}{\sum_{w\in\mathcal{N}_u}\exp(e_{uw}^{(l)})} \tag{4.6}$$

The attention mechanism can be any function that takes as input two vectors, with the same dimension as the product $W^{(l)}h_u^{(l)}$ and outputs a real value. For example, a can be a feed-forward neural network.

4.1.4 GraphSAGE and neighbourhood selection

GraphSAGE (Graph SAmple and aggreGatE) is a type of GNN and MPNN introduced in Hamilton et al. (2017). It is designed with the goal of performing inductive learning, i.e. to generate node embeddings for unseen data. It follows Equations 4.7-4.9. N_v represents a sampling of $\mathcal{N}_v$. Usually, N_v produces a fixed-size set. CONCATENATE is the concatenation operator. Equation 4.7 can be generalized by using a different operator from the sum operator, provided that the operator is still permutation invariant, i.e. produces the same result regardless of the order of the nodes.

$$h_{\mathcal{N}_v}^{l+1} = \sigma\left(W^l\left(\frac{1}{\mid N_v \mid +1}\left(h_v^l + \sum_{v \in N_v} h_u^l\right)\right)\right) \tag{4.7}$$

$$h_v^{l+1} = \sigma\left(W^l \cdot \text{CONCATENATE}\left(h_v^l, h_{\mathcal{N}_v}^l\right)\right) \tag{4.8}$$

$$h_v^{l+1} = \frac{h_v^{l+1}}{\|h_v^{l+1}\|} \tag{4.9}$$

While the neighbourhood function can be defined arbitrarily, in practice we draw a uniform sample of fixed size from the neighbors of the node. This draw is performed for every layer. The choice of neighbourhoods for each node can affect the performance of the network. Classical graph neural networks such as Graph Convolutional Networks Kipf and Welling (2017) used the full neighbourhoods of each node. To be able to handle larger graphs, some approaches use the same sampling scheme as GraphSAGE while trying to preserve the graph structure as in Cluster-GCN Chiang et al. (2019).

4.1.5 GNN advanced tricks

In this section, we present some of the tricks that can be used to improve the performance of GNNs. These tricks can be applied to virtually all GNNs.

Reversible GNNs: the challenge of GPU size

GNNs are memory intensive, to the point that many models cannot be run with large datasets on standard GPUs. Most of the memory cost comes from having to store the features tensor of the whole graph at each layer. To solve this problem, *reversible* graph neural networks have been proposed in Liu et al. (2019); Li et al. (2021a). The idea is to propagate the information from layer to layer using a chain of operations that will be performed in an inverse way for the backpropagation. It increases the time complexity in order to decrease the space complexity.

Self-knowledge distillation

Another approach used for decreasing memory load is knowledge distillation Hinton et al. (2015). A large neural network is trained while smaller networks try to approximate its results. The large network acts as a teacher and the smaller networks as students. Using this principle, it was shown in Zhang et al. (2019) that an efficient way to implement knowledge distillation is to train the early layers of a network as the students and the whole network as the teacher.

Adaptive Graph Diffusion Networks (AGDN): the challenge of oversmoothing

In addition to memory constraints, adding layers to GNNs can lead to oversmoothing the features, where the information of a single node is drowned in the information coming from everywhere in the graph Li et al. (2019). One way to counteract this effect is to try and retrieve information at each layer from different *hop-neighbourhoods*. One example of such architecture was proposed in Sun and Wu (2020).

Nodes that are one-hop apart are the direct neighbours, those that are 2-hop apart are the neighbours of the neighbours, and so on.

4.2 Dataset

Most of our experiments in this thesis are performed on one dataset: the academic citation network that comprises all the papers in the arXiV Computer Science repository. In this section we provide a description of this dataset. We also discuss the question of splitting a graph dataset, the difference between transductive and inductive learning, and the choice of features.

4.2.1 Overview

ogbn-arxiv is a dataset from the Open Graph Benchmark (OGB) Hu et al. (2020). It is an academic citation network that contains all the papers in the arXiV Computer Science repository. It contains 169,343 nodes and 1,166,243 edges. Each node has a 128 feature vector. These features represent the average embeddings of the words in the title and the abstract. These embeddings are obtained using a word2vec model Mikolov et al. (2013). The features are normalized. The splitting of the dataset between train, validation and test is done using the year in which the articles were published: the papers up to 2017 are in the train set. Those published in 2018 are in the validation set. Papers from 2019 and 2020 constitute the test set.

The labels in the dataset represent the category of the paper, e.g. machine learning, computer vision, etc. Each paper is labelled by the authors and the arXiv moderators; these labels are assigned a number between 0 and 39, representing the 40 categories of the arXiv CS (Computer Science) repository.

Each category is denoted by two letters; their full name can be found at https://arxiv.org/corr/subjectclasses.

In our discussion of transductive and inductive learning, we also use the *ogbn-papers100M* and *ogbn-mag* datasets from OGB. We briefly describe them in the remainder of the section.

ogbn-papers100M is a dataset constructed in a similar way to *ogbn-arxiv*, except that it contains more than 111 million papers and that the classification task is performed on the subset of papers that corresponds to all the papers published on the arXiv website; it is not restricted to computer science and there are 172 subject areas.

ogbn-mag is a heterogeneous graph which contains about 736,000 papers, 1.1

million papers, 8,700 institutions and 60,000 fields of studies. The features of the nodes are constructed with the same approach that was used for creating the features of *ogbn-arxiv*. The task is to predict the venue of the paper.

4.2.2 Dataset split

A naive approach to splitting a graph between train, validation and test sets, is to randomly assign each node to one group. This is what was done in Yang et al. (2016) on graphs like CiteSeer, Cora and Pubmed Sen et al. (2008). Similarly, the authors of Chiang et al. (2019) follow a randomized split without validation set for training their model. By contrast, the authors of Hu et al. (2020), who incorporated the Amazon dataset in Chiang et al. (2019) as the *ogbn-products* in OGB, recommend using a split that relies on the sales ranking of the products: the most sold items are in the training set while the most rarely sold ones are in the test set. The authors further argue that this split matches the behaviour of real-world applications where, due to the cost of labelling data, the most important nodes are labelled first and a model is used to infer the labels of the less important nodes.

In (Hu et al., 2020), the authors proposed splitting the dataset with respect to the year of publication of the papers, on the basis that this reflects one of the real world applications of GNNs, which is to predict the category of new papers using only already published papers; furthermore, they argue it is a more challenging task than just randomly splitting between train, validation and test. Thus, the split is the following: the train set consists of all papers published before 2018; the validation set has all the papers published in 2018; and the test comprises all the papers from 2019 (inclusive) onwards.

4.2.3 Transductive and inductive learning

Due to the interconnectedness of nodes in graph data, it is not possible to create a series of independent examples to be fed to a neural network, as can be done in computer vision with images or in natural language processing with sequences of text input. Thus, there are two approaches for splitting a dataset between training and test data: the unlabeled features of the test nodes can be

used during the training phase of the model, or they can be ignored. The first approach corresponds to *transductive* learning, while the latter corresponds to *inductive* learning Yang et al. (2016). In transductive learning, the model tries to predict labels of nodes already seen in the training phase, while the model in inductive learning is tested on unseen nodes. Inductive learning is therefore better suited to improve the generalization power of a model.

The concepts of transductive and inductive learning are related to the choice of neighbourhood. In the transductive setting, the neighbours of the training nodes can belong to the validation and test sets; in this case, only their features are known at training time. In the inductive setting, the neighbours of the training nodes are restricted to other training nodes. The nature of the dataset, more specifically the way it evolves over time, can influence the preferred mode of learning, i.e., whether we should use transductive or inductive learning Yang et al. (2016). It is especially important in temporal graphs Rossi et al. (2020); Xu et al. (2020).

When training a GNN on a social citation networks from OGB, *ogbn-arxiv*, *ogbn-mag* and *ogbn-papers100M*, the recommended data split by the authors of Hu et al. (2020) is to put all the papers published before 2018 in the training set; those published in 2018 in the validation set and the rest in the test set. Table 4.1 shows the distribution of nodes and edges in *ogbn-arxiv*, *ogbn-mag* and *ogbn-papers100M*, as well as the number of edges that have a source and a target node in a different part of the data split, e.g., one node in the train set and the other in the test set. We see that in *ogbn-arxiv*, a model trained in a transductive setting can exploit about $200,000$ edges, of which only 40% are in the training set. The phenomenon is less important in *ogbn-mag* where 80% of the edges are in the training set. Considering only the edges between labeled nodes, which account for a small portion of all the edges, *ogbn-papers100M* has also 80% of its edges coming from the training set.

Citation networks change over time, as fields grow in importance while others dwindle. This is shown in Tables 4.2 and 4.3 which contain the most frequently occurring class per year in the *ogbn-arxiv* dataset. Some classes, such as the **cv** class (Computer Vision) were not present in the top 5 before suddenly reaching second then first position, as it happened between 2013 and 2015. These distribution changes emphasize the importance of inductive learning: GNNs need to be trained to be able to adapt to these changes.

Table 4.1: Distribution of Edges in Train, Validation and Test.

Graph	Detail	number of edges	edges overlap
arxiv	full graph	1,166,243	-
arxiv	self-loops, reverse edges	2,484,941	-
arxiv	train subgraph	829,007	-
arxiv	validation subgraph	86,671	-
arxiv	test subgraph	167,883	-
arxiv	train and validation	1,351,570	435,892
arxiv	train and test	1,710,834	713,944
arxiv	validation and test	506,098	251,544
mag	full graph (only citations)	5,416,271	-
mag	self-loops, reverse edges	11,568,931	-
mag	train subgraph	8,389,507	-
mag	validation subgraph	177,111	-
mag	test subgraph	123,583	-
mag	train and validation	10,149,426	1,582,808
mag	train and test	9,575,822	1,060,732
mag	validation and test	533,884	231,190
papers100M	full graph (only citations)	1,615,685,872	-
papers100M	self-loops, reverse edges	3,342,431,700	-
papers100M	train subgraph	21,315,319	-
papers100M	validation subgraph	330,045	-
papers100M	test subgraph	639,152	-
papers100M	train and validation	24,506,366	2,861,002
papers100M	train and test	25,144,635	3,190,164
papers100M	validation and test	1,583,153	613,956

4.2.4 Graph Information Aided Node feature exTraction (GIANT)

The choice of features associated with each node and edge can play a significant role in the efficiency of a classifier. Most methods of feature extraction using raw data, such as word2vec Mikolov et al. (2013) or BERT Devlin et al. (2019) do not leverage the graph topology when constructing the new features. The Graph Information Aided Node feature exTraction (GIANT) framework was proposed in Chien et al. (2021) to incorporate the graph topology in the feature extraction.

At the time of writing, the success of ChatGPT has initiated a new wave of

Table 4.2: Top 5 Classes (By Size) and Per Year in *obn-arxiv*. Only the 3 Most Prominent Classes Are Shown.

Year	Class 1	size	Class 2	Size	Class 3	Size
2019	lg	8690 (21.88%)	cv	8584 (21.62%)	cl	4075 (10.26%)
2018	cv	6846 (22.97%)	lg	4458 (14.96%)	cl	2849 (9.56%)
2017	cv	4326 (20.18%)	it	2597 (12.11%)	lg	2114 (9.86%)
2016	cv	2646 (16.19%)	it	2525 (15.45%)	lg	1374 (8.41%)
2015	it	2210 (18.36%)	cv	1453 (12.07%)	lg	1008 (8.38%)
2014	it	1755 (19.17%)	cv	705 (7.70%)	ds	631 (6.89%)
2013	it	1617 (19.88%)	ai	927 (11.40%)	lg	579 (7.12%))
2012	it	1316 (20.45%)	lg	680 (10.57%)	ai	547 (8.50%))
2011	it	1142 (25.80%)	ds	384 (8.67%)	lo	246 (5.56%)
2010	it	940 (26.37%)	ds	298 (8.36%)	lo	240 (6.73%)

Table 4.3: Top 5 Classes (By Size) and Per Year in *obn-arxiv*, continued. Only the 4th and 5th most prominent classes are shown.

Year	Class 4	Size	Class 5	Size
2019	it	2256 (5.68%)	ro	1602 (4.03%)
2018	it	2273 (7.63%)	ai	1232 (4.13%)
2017	cl	1753 (8.18%)	ai	933 (4.35%)
2016	cl	1185 (7.25%)	ds	850 (5.20%)
2015	ds	736 (6.12%)	si	532 (4.42%)
2014	lg	593 (6.48%)	ni	483 (5.28%)
2013	ds	552 (6.79%)	ni	441 (5.42%)
2012	ds	433 (6.73%)	ni	341 (5.30%)
2011	ni	223 (5.04%)	ai	217 (4.90%)
2010	ni	210 (5.89%)	ai	189 (5.30%)

Large-Language Model (LLM) extracted features GRA; He et al. (2023).

Chapter 5

Social networks and multilabel classification

This chapter deals with the difference between transductive and inductive learning, as well as the problem of multilabel classification. When a graph evolves over time, the distribution of the nodes changes. This distribution shift is reflected when the dataset is split into training and test sets. Training a GNN with (transductive) or without (inductive) access to the unlabelled test data leads to different performances from the models. Section 5.2 evaluates and compares state-of-the-art GNNs on node classification in an academic citation network.

In node classification, a GNN is using both the features of a node and its neighbours to determine its label. In some cases, this means that the features will push the model towards one label, while the neighbours will push it towards another. Furthermore, the notion of a correct label for a node can be blurred in settings where there are overlaps between the labels. In Section 5.3, we explore the problem of multilabel classification in an academic citation network. Section 5.4 concludes the chapter.

The results presented in Section 5.2 were published in Lachaud et al. (2022b), while the work in multilabel classification from Section 5.3 was presented in Lachaud et al. (2022a).

Contents

5.1 Introduction

A graph distribution can change over time. Attributes of users of social networks change. A hundred years ago, papers were written by individual people, there were fewer collaborations and the number of papers each paper cited was smaller. Contributions came from a few countries Dong et al. (2017). In a social network, an event can change the structure of the graph.

Another aspect of working with social graphs is that classification is not always perfect: while we can establish that a molecule is toxic or not, putting labels on people is more complicated, because the categories are not entirely distinct. Users can like several genres of music and have different groups of friends. Furthermore, a user can change over time: their taste of music can evolve, friendships can end while new ones are forged. Additionally, some categories can encompass or overlap with other categories.

In many cases, classification tasks require neural networks to produce multiple labels: images have several elements, sentences can be related to different topics, and nodes in a graph may be related to several classes via different neighbours. Multilabel classification for each type of data led to the development of specialized network architectures (Nam et al., 2014; Wang et al., 2016; Lanchantin et al., 2019).

In the context of a changing graph, the distribution shift may cause issues in the performance of GNNs in different settings: transductive and inductive learning. Evaluating GNNs in these two settings provides a better comparison ground between the architectures. Several explanations are possible as to why an architecture achieves the same performance in transductive and inductive learning: the train and test dataset may have the same distribution, or the architecture is failing to capture relevant information in the transductive setting.

Another point of interest is the nature of the shift: is it a *structural* shift, e.g., the way nodes are linked together has changed; or a *semantic* shift? An example of a structural shift is science moving towards collaborations and away from single author contributions. One example of a semantic shift is a field, such as computer vision, changing from handmade features to embeddings generated by deep learning approaches.

In this chapter, we analyze the behaviour of several GNNs on a graph that has a known distribution shift (see Section 4.2.3). We evaluate the GNNs in both transductive and inductive learning. We consider two sets of features for the nodes: the original ones, and improved ones proposed by some authors. We find that performance between transductive and inductive learning is similar. We then focus on the errors made by a GNN. We find that these errors are of three types. The first type of error is due to the the use of a common semantic field between two classes and the over-representation of one class compared to the other, which leads to misclassification of the small class in favour of the large one. The second type is due to a similar semantic field with multiple interactions between the classes. The third type is due to classes which sit at the intersection of other classes, e.g. robotics and human computer interaction.

A semantic field represents the set of words that are related to a topic. By analogy, the semantic field can describe the values that the features from the nodes of a given class take.

5.2 Transductive and inductive learning

In contrast to image and text data, graphs cannot easily be divided into multiple datasets, because nodes are interconnected with each other. In order to train a GNN, we must create a training set on which the model is trained; a validation set whose purpose is to find the best model; and a test set containing unseen data to see if the model is performing well.

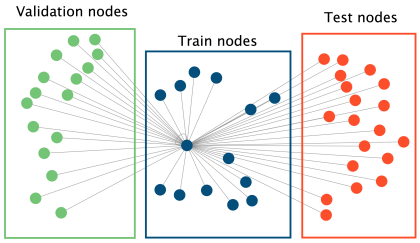


Figure 5.1: Node in the training set with neighbours in all the graph.

When the graph can dynamically evolve over time, as is the case for social networks or citation networks, the splitting is usually based on the temporality of the nodes: the earliest nodes are put in the training data and the most recent nodes form the test data. This is coherent with the goal of most of the applications using temporal graphs: being able to predict future nodes features, edges creation, etc., using a model trained on the existing data.

In transductive learning, the nodes in the training data might have access to the features of the nodes in the validation and test sets. Consider the node in Figure 5.1. For simplification purposes, edges between other nodes than the central nodes are removed. The edges are represented as undirected, and the self-loop has been added. Here the node has 48 neighbours (not counting itself) with the following distribution: 13 nodes are in the training set, 16 in the validation data and 19 in the test set. Each additional training node might contain other edges with nodes in the validation data and test data. After several layers of a GNN, and depending on the locality of the node in the graph, the node has received information about many of the nodes in the validation and test sets. In the inductive setting, the model is trained using only the subset of nodes in the training data; in our example, this represents the blue nodes. Since most of the GNN entries on the Open Graph Benchmark leaderboard train their models in a transductive setting, there is a spread of information from the validation and the test data to the training data.

The leaderboard is available online at this address: https://ogb.stanford.edu/docs/leader_nodeprop/.

In conjunction with exploiting the features of nodes in the test set, most methods use label information as a way to augment the features of the nodes: given a set of features X_{feats}, a one hot encoding vector is used to represent the label of the

node. The new features X_{onehot} are then concatenated with X_{feats} to produce the input features X. For the elements outside of the training set, the label is first left empty; i.e. all the entries in the one hot vector are set to zero. Then the vector is filled by applying a softmax to the output logits of the GNN. If there is information from the test set that is contained in the validation data, it means that the network will indirectly try to produce the best label features for the test nodes that are used in classifying the validation nodes. It is thus trying to fit the test data.

5.2.1 Experiments

We compare the performance of models trained in two different settings: transductive and inductive learning. In the first setting, the model has access to the training data and the unlabeled validation and test data. In the second setting, the model has only access to the training data. Each setting follows the same temporal split presented in Section 4.2. A practical way to remove the edges going from or to unlabeled nodes is to generate the subgraph of the network that contains only the nodes within the given set.

The architectures we compare are the best performing architectures on the *ogbn-arxiv* dataset. Specifically, we train Deep RevGAT Li et al. (2021a) and Adaptive Graph Diffusion Network (AGDN) Sun and Wu (2020) models. The Deep RevGAT architecture follows the message passing neural framework Gilmer et al. (2017). It is a GAT Velickovic et al. (2018) which has been converted to a reversible GNN Li et al. (2021a) to remove the memory constraints imposed by the storing of a feature matrix for each layer of a traditional GNN. We explore the importance of self-knowledge distillation Zhang et al. (2019) by training the Deep RevGAT with and without knowledge distillation. Additionally, we investigate the importance of feature selection, using either the original features, the features extracted with the GIANT framework, and the features with added label information.

All the experiments are performed with a 24 GB Nvidia RTX GPU. The code is written in Python, PyTorch and DGL (Deep Graph Library) Wang et al. (2019). The *ogbn-arxiv* dataset is taken from OGB Hu et al. (2020). Several Deep RevGAT are trained with a different number of layers. The AGDN model is trained on

the original features both with and without using a bag of tricks presented in Wang et al. (2021) that can improve performance.

Each model is trained for 2,000 epochs. At the end, the model weights saved are the ones which led to the best validation accuracy. Additionally, each model is trained for 10 runs in order to mitigate the fluctuation in accuracy over each run.

Table 5.1 shows the results of the training computations in a transductive setting. The difference between validation and test accuracy is also shown in the last column. Table 5.2 shows the results of the same models in the inductive setting.

Table 5.1: Validation and Test Accuracy for Transductive Learning. AGDN is trained with the original features.

Model	Validation accuracy	Test accuracy	Validation and test gap
RevGAT, teacher, 2 layers	76.99 ± 0.06	75.98 ± 0.12	1.01
RevGAT, KD 2 layers	77.13 ± 0.10	76.17 ± 0.15	0.96
RevGAT, 3 layers	77.13 ± 0.06	75.95 ± 0.11	1.18
RevGAT, KD 3 layers	76.93 ± 0.08	75.61 ± 0.15	1.32
RevGAT, 5 layers	77.11 ± 0.06	75.80 ± 0.12	1.31
RevGAT, KD, 5 layers	77.24 ± 0.09	75.97 ± 0.09	1.27
AGDN, original features	-	73.46 ± 0.17	-
AGDN with BoT	-	74.10 ± 0.15	-

Table 5.2: Validation and Test Accuracy for Inductive Learning. The Best Score is Highlighted in Bold. KD Stands for Knowledge Distillation.

Model	Validation accuracy	Test accuracy	Validation and test gap
RevGAT, 2 layers,	76.75 ± 0.08	75.91 ± 0.14	0.84
RevGAT, KD, 2 layers,	76.74 ± 0.05	75.81 ± 0.13	0.93
RevGAT, 3 layers,	76.78 ± 0.08	76.04 ± 0.10	0.74
RevGAT, KD, 3 layers,	76.78 ± 0.09	75.97 ± 0.09	0.81
RevGAT, 5 layers,	76.84 ± 0.05	76.00 ± 0.10	0.84
RevGAT, KD, 5 layers,	76.85 ± 0.06	$\mathbf{76.07 \pm 0.07}$	0.78
RevGAT(3), no label features,	76.54 ± 0.09	75.83 ± 0.10	0.71
RevGAT(3,KD), no label features,	76.63 ± 0.08	75.97 ± 0.06	0.66
AGDN, original features,	72.47 ± 0.12	73.20 ± 0.17	-0.73
AGDN with BoT,	73.62 ± 0.07	74.00 ± 0.07	-0.38
AGDN, GIANT features,	76.28 ± 0.17	75.38 ± 0.22	0.9

We see that in the transductive learning setting, the Deep RevGAT model achieves the best accuracy with the smallest number of layers. Using self-knowledge distillation with more layers actually worsens the performance. The increase in the difference between validation and test accuracy indicates that the model is trying to fit the validation data, and that the validation data and test data are not completely similar. This is expected because the addition of new papers to the citation network may change the structure of the graph, and the model might not be able to predict these unforeseen structures.

In the inductive setting, the performance of the model with self-knowledge distillation increases with the number of layers, while the gap between validation and test accuracy decreases. The model with 5 layers and knowledge distillation achieves the best test accuracy of all the models trained with inductive learning. The performance is close to the one from the top model in the transductive setting.

Models trained on the same dataset in transductive and in inductive learning usually achieve better test accuracy in transductive learning Xu et al. (2020). This is an expected behaviour because the transductive model already had access to test data in the training phase. The fact that the models achieve similar performances in transductive and inductive settings may be an indication that the models are not fully exploiting the information available during training, and that new architectures are needed to leverage it. Indeed, the fact that the AGDN model trained with the original features has a higher test accuracy than its validation accuracy suggests that the model is slightly underfitting the data, and that either the features must be changed, or the model needs to be modified to better exploit the data.

The use of features that exploit the graph topology, created with the GIANT framework, could partly explain why the gap between transductive and inductive learning is small: because the features are created using the entire graph, some of the test information is already embedded in the features of the training nodes. Since the AGDN model trained with the original features achieves comparable accuracy in both settings, the GIANT extracted features cannot account for the small gap.

In the inductive setting, directly encoding the label information as part of the nodes features using a one-hot encoding scheme does not seem to improve

performance. While the Deep RevGAT model with 3 layers trained with label features performs better than the model trained solely on the GIANT features, adding self-knowledge distillation reduces the difference.

The most likely reason why the models achieve comparable results in transductive and inductive settings is that each model uses some form of neighbourhood sampling in the training phase. This makes the models more robust and acts as a regularizer in preventing overfitting to the data. The importance of edges coming from validation and test data is reduced as they do not systematically appear in the sampled edges.

5.3 Error analysis and multilabel classification

The experiments in the remainder of the chapter will be performed in the transductive setting only. We now turn analyse the errors made by the GNNs. For these experiments, we use a GAT with the original features of the graph.

5.3.1 Single class classification

In this section, we use a single class classification approach. A deep analysis of the misclassifications will lead us to explore a multilabel approach in Section 5.3.2.

For all the experiments, we use a 24GB NVidia RTX GPU. The code is written in Python, Pytorch and PyTorch Geometric (Fey and Lenssen, 2019). We use the OGB (Open Graph Benchmark) (Hu et al., 2020) package to get the *ogbn-arxiv* dataset.

To analyze the misclassification errors made by the model, we need to go beyond the accuracy score and look at the confusion matrix on the test set, which will help us see where it fails to generalize. The value at row c_i and column c_j indicates the number of times the model has assigned the label c_j to a node from c_i, divided by the total number of nodes of category c_i in the test set. The rows have been normalized and each one adds up to 100. The categories were ordered in such a way that the small classes occupy the first columns while

Each value outside the diagonal is a mistake made by the model.

the middle of the matrix is for the most populated classes and the rest of the columns represent mostly middle-sized classes. A subset of the full confusion matrix is displayed in Figure 5.2 with the categories that are discussed in the rest of the paper.

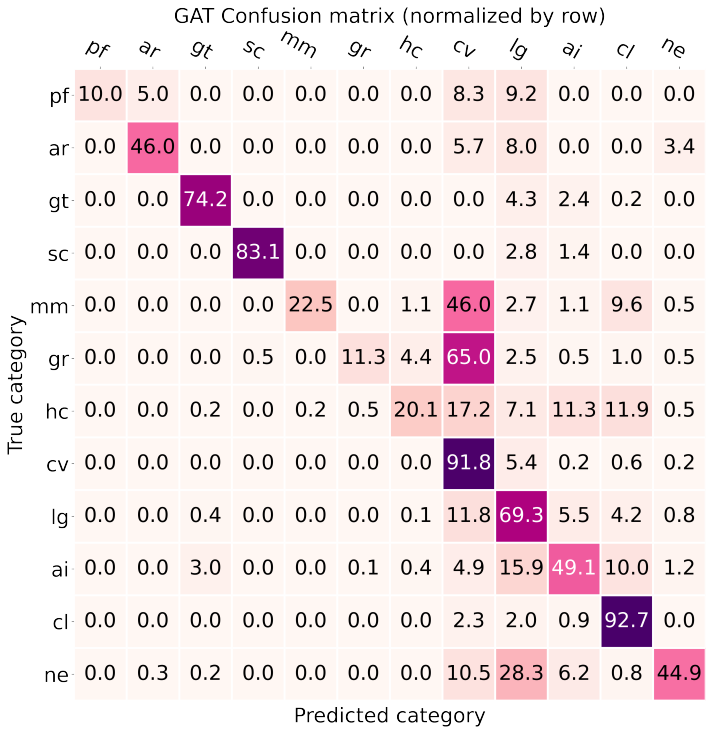


Figure 5.2: Subset of the confusion matrix.

We first observe that the size of the category in the training set, as represented by the "train size" column in Tables 5.4 and 5.5 is not a sufficient indicator of poor performance. The model achieves low accuracy on such categories as **ar** (Hardware Architecture) and **pf** (Performance) but successfully classifies nodes from **gt** (Computer Science and Game Theory) and **sc** (Symbolic Computing), despite the fact that these categories have approximately the same number of nodes in the training set. This suggests that some categories display more cohesiveness than others, and that the network is able to detect this pattern.

Still on the topic of categories with little representation, we see systematic misattribution for nodes in the **mm** (Multimedia) and **gr** (Graphics) categories, which are classified as **cv**. Considering that the three subjects likely share a

similar terminology, and that the initial features of the nodes were based on the words in the title and the abstract, there is little hope, without changing the features, to correctly predict these classes.

Next, we are faced with subject areas that are intrinsically interdisciplinary, which means they exploit ideas from other areas of research. The most eminent representative of these categories is **hc** (Human Computer Interaction). By design, HCI tends to capitalize on the advances in various fields, e.g. computer vision, natural language processing, and study the impact, positive or negative, they can have on users. In ogbn-arxiv, this will be reflected in two manners: **hc** nodes have neighbours that can belong to other classes, and two **hc** nodes can have vastly different features.

Finally, the error which is the key factor in driving down the accuracy is the confusion between categories within a group of similar categories. This is exemplified with the categories **cv** (Computer Vision), **lg** (Machine Learning), **ai** (Artificial Intelligence), **cl** (Computation and Language, mostly natural language processing) and **ne** (Neural and Evolutionary Computation). About 30% of **ai** nodes in the test set are incorrectly attributed to one of the aforementioned classes, while 20% of **lg** nodes and 35% of **ne** nodes are similarly misclassified. All these categories mutually fuel the research of the others. The two biggest reasons for the misclassification are a combination of two causes mentioned earlier: many nodes from these categories share a similar terminology, e.g. papers on neural networks have similar characteristics; and the nodes cite papers from all the areas in the group.

Considering the overlapping themes of some categories, as well as the interdisciplinary content of some papers, a multilabel classification approach is preferable to the single label classification task. Firstly, it allows a finer grained categorization of papers, distinguishing between papers in the robotics field that have a computer vision component with those that have a natural language processing component. Secondly, it helps concentrate on the bigger errors made by the neural networks: those in which the category is not in the top predictions.

5.3.2 Multilabel classification approach

Instead of focusing only on the top prediction of the model, we retrieve the three most likely predicted classes of our GAT model for each node in the test set. The set of estimated probabilities is usually obtained by applying a softmax activation function to the last layer of the neural network; in the case of multiclass classification, to make a prediction, we simply output the category which is associated with the highest probability. We compute the number of times the correct category is the prediction (accuracy, or top 1), as well as the number of times it appears in the two (top 2) or three (top 3) categories with the highest estimated probabilities. Overall, while the model achieves 72.4% accuracy, the right category is in the two highest predictions 87.3% of the time, a 15% increase. In the top 3, this number rises to 92.4%. Results for each category are presented in Tables 5.4 and 5.5, alongside the relative size of the category in the training set (given in percentage) and its population in the test set. The arXiv categories in bold are the ones discussed in the text. Additionally, a representation of the top 3 predictions for some nodes is presented in Figure 5.3.

Table 5.3: Top 3 score on training, validation and test

dataset	top 1	top 2	top 3
train	79.31	90.36	94.14
valid	73.62	87.76	92.73
test	72.27	87.25	92.36

We see that, within a group of non-mutually exclusive categories, there are some classes that attract most of the predictions, such as the **cv** and **cl** which are in the group of artificial intelligence related categories. These leads to poor accuracy scores for the **lg** and **ai** classes. However, when we look at the three highest estimated probabilities, the network gets most of the **lg** and **ai** samples right. For example, node 1 in Figure 5.3 belongs to the **lg** category, which is the second prediction of the model. Similarly, nodes 2, 3, 5 and 7 all belong to the **ai** category, which is the second or the third prediction from the model.

Additionally, the top three predictions are either related to the true category, or to the category of the neighbours. For example, node 1 has neighbours that belong to the **ai**, **lg**, **cv**, **cl** categories. This means that the model is properly learning from the information contained in the neighbours. Nodes with neighbours

Table 5.4: (Part 1) Top 3 category predicted by the GAT model. The train size represents the percentage of nodes in the training set that are from each category. The test column indicates the number of nodes from the test set that are in each category.

subject	top 1	top 2	top 3	train size (%)	test
cv	91.83	98.10	98.99	10.99	10477
lg	69.27	91.30	96.38	7.69	10740
it	90.56	96.28	97.54	17.91	2849
cl	92.74	97.06	98.27	4.77	4631
ai	49.14	71.13	82.68	5.70	1455
ds	69.87	86.85	92.43	5.97	1414
ni	55.12	84.32	91.12	4.46	1250
cr	67.04	82.18	87.91	3.15	1869
dc	52.73	75.12	83.07	3.23	1246
lo	67.94	90.18	94.13	3.96	733
ro	70.91	88.77	94.63	1.83	2066
si	68.40	82.61	88.18	3.14	1041
gt	74.16	87.40	91.55	2.76	627
sy	63.72	79.47	84.96	2.06	419
se	62.00	76.24	81.81	1.69	808
ir	46.52	77.47	90.13	1.48	892
cc	51.59	71.88	87.25	2.47	345
db	63.41	74.43	83.37	1.78	481
ne	44.90	64.97	82.96	1.42	628
os	8.33	25.00	50.00	0.08	36

Table 5.5: (Part 2) Top 3 category predicted by the GAT model. The train size represents the percentage of nodes in the training set that are from each category. The test column indicates the number of nodes from the test set that are in each category.

subject	top 1	top 2	top 3	train size (%)	test
cy	17.13	43.12	59.33	1.12	654
cg	75.72	85.62	90.42	1.64	313
dm	26.02	54.65	78.81	1.71	269
pl	47.41	74.09	81.87	1.39	386
hc	20.10	36.82	52.89	0.77	622
dl	76.17	81.78	85.05	1.21	214
fl	55.45	73.18	80.00	1.02	220
sd	77.47	89.89	94.95	0.50	475
ma	6.28	27.62	62.76	0.43	239
et	57.89	74.64	81.82	0.44	209
mm	22.46	52.94	66.31	0.42	187
sc	83.10	88.73	88.73	0.52	71
ce	28.36	44.78	52.99	0.42	134
na	33.33	55.56	70.37	0.48	54
gr	11.33	56.16	77.34	0.22	203
pf	10.00	34.17	52.50	0.26	120
ms	57.83	79.52	84.34	0.30	83
ar	45.98	65.52	74.71	0.27	87
oh	0.00	1.96	3.92	0.33	51
gl	0.00	0.00	0.00	0.02	5

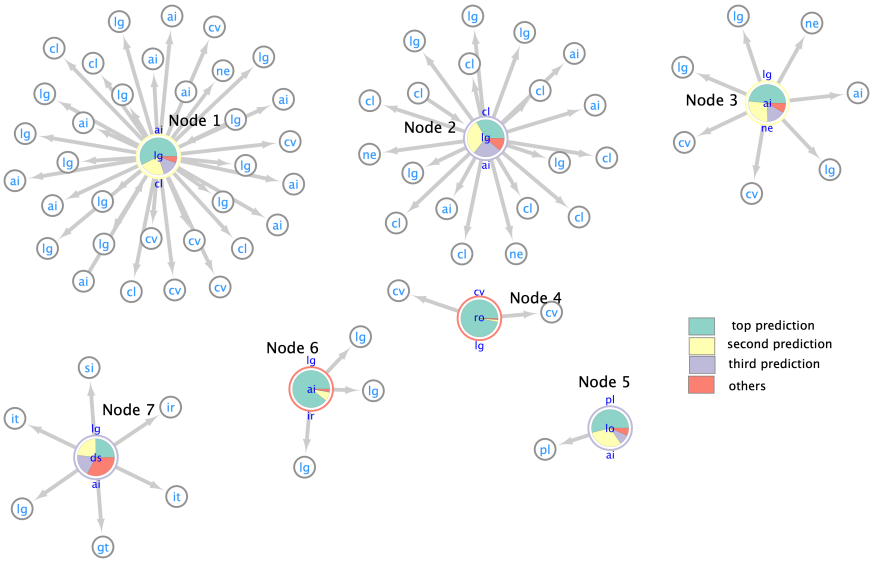


Figure 5.3: Top 3 predictions for a few nodes in the graph. The pie chart represents the probability assigned by the model to the the first three categories. For each node with a piechart, the label of the first prediction is the one on top, the second prediction the one in the middle and the third prediction the one at the bottom. The nodes without piecharts are the neighbours of the nodes on which we do the predictions, and have their true label written inside them.

from different categories than themselves will rarely be classified in the correct category, but the top predictions of the model will most often be related to the content of the paper. This suggests that focusing on a single category is not sufficient to properly classify a paper, and that a better way is to look at the first two or three predictions to get a meaningful categorization of the paper. Node 1 is a paper from the **ai** category, but it cites papers from the **cv** category; thus it is likely to contain a sizable amount of information related to **computer vision**, even if it is not the main theme of the paper.

We also observe that the challenges faced by interdisciplinary categories remain when we observe the top three predictions: the model correctly has **hc** in its top three predictions in only 53% of the cases. Node 4 and node 7 in Figure 5.3 illustrate the situation. Node 1 only has **lg** neighbours, while node 2 only has **cv** neighbours. Furthermore, **hc** is not in the first three predictions of the model.

5.4 Discussion

In this chapter we studied the difference between transductive and inductive learning for citation networks where the graphs have been converted to undirected graphs and the train, validation and test sets have been constructed using the temporal information of the nodes. We saw that the training datasets contain many edges from the validation and test sets: one fifth of the edges in the training set of *ogbn-mag* are from the validation or test set, while two thirds of the edges in *ogbn-arxiv* come from outside of the training set.

To analyze the importance of these edges, we trained state of the art GNNs in an inductive setting where the training set contains only edges from within the training set. We found that the networks achieved similar performances in either a transductive or inductive setting. This may be due to the neighbourhood sampling methods used by most GNNs that reduce overfitting to the data. This also suggests that current models can be improved, as models usually perform better in transductive settings.

Furthermore, we observed that the gap between validation and test accuracy was lower in the inductive setting than in the transductive setting. This is an indication that the models trained in the inductive setting were better at generalizing than their counterparts trained in the transductive setting. Moreover, this suggests that new architectures will likely be able to improve on the current results.

In this chapter, we also trained a GNN on the same dataset, reframing the problem as a multilabel classification problem where a node may belong to more than one category with a given probability. For instance, a paper in the robotics category might tackle a computer vision problem, while another one might deal with a natural language processing task. We found that considering the top three predicted classes, the real class was present in more than 92% of the cases. In addition, we observed that the categories in the top predictions are usually related to the true category, or to the category of the neighbours of the paper. These results validate the multilabel approach.

Some perspectives for future works include performing a similar analysis on bigger datasets to generalize our findings. The multilabel approach is likely to

extend to other domains, as objects in social networks or other real world data do not usually belong exclusively to one class. Furthermore, a different set of features can be explored to improve discrimination between classes.

Chapter 6

Noisy features and missing data imputation

This chapter deals with the noise in nodes features and its impact on GNN learning. Specifically, we deal with two types of noise: noisy features and missing features. Noise can arise from many situations, such as faulty sensors or misinformation. It can occur randomly or target specific nodes, e.g., target the most influent nodes or the most isolated ones. Section 6.1 present an overview of the problem of noise in graph data. Section 6.2 explores the impact of noise in the features on the training of state-of-the-art GNNs, in both transductive and inductive settings. Section 6.3 presents a novel GNN architecture to impute missing features. Section 6.4 concludes the chapter.

The work regarding noisy features in Section 6.2 is currently under review. The work regarding missing data imputation was published in Lachaud et al. (2023).

Contents

6.1 Noise in data

6.1.1 Noise in attributes

Most types of data contain some degree of noise. It can take many forms, such as blur in images, spelling or grammar mistakes in text; parts can be missing or distorted in audio and video recordings. The impact of noise on machine learning models varies with the severity of the noise Dodge and Karam (2016).

For instance, noise in the background of an image is less important than the background noise of an audio recording in a crowded train station. Additionally, noise can occur naturally or can be artificially injected to produce errors.

In graph structured data, the noise takes two forms: noise in the structure of the graph, and noise in the features and labels. Noise in the structure alters how nodes are connected together via the edges. Edges can be added or deleted. For example, in a social network where the nodes represent people and the edges indicate friendship, removing edges will hide the friendship between two people. In this way, a group of people forming a community might appear as disjoint groups of individuals.

Noise in the features covers the noise in the features of the nodes and the edges. As an illustration, in a products network, edges between two products indicate how many times they were bought together Chiang et al. (2019). Because the company selling the products may have different selling points, the information can take time to be aggregated. During this time, the weight of the edges does not represent the correct number of items sold together. In a different vein, social networks such as Facebook and Twitter, each record different information about their users, so each user of both networks may have different features on each platform. Classification using one or the other social network will produce different outcomes.

The importance of noise in graph structured data is twofold: noise in the structure impacts how the information propagates through the graph; and noise in the features changes the messages that are passed. Firstly, many of the machine learning approaches used in graph representation learning involve a flow of information through the neighborhood of the nodes. Indeed, in Graph Neural Networks (GNNs), most models follow a Message Passing Neural Network (MPNN) framework Gilmer et al. (2017). In this framework, a message is computed between a node and each of its neighbours, and these messages are aggregated to form the new node's representation.

Secondly, while the structure determines the nodes used to update the graph's representation, it is the features that create the representation. For instance, in a social network, wrong hobbies may be attributed to a user. When machine learning models try to predict what the hobbies of their friends are, they will use the wrong hobbies which will lead to classification errors. These errors will

spread to the friends of the friends, and might spread through the whole graph.

The impact of noise on the learning of GNNs can be amplified by the manner in which the learning is performed. There are two possible settings: transductive and inductive learning. In transductive learning, the GNN model has access to the validation and test features at training time; only the labels are hidden. In contrast, in inductive learning, the model is trained exclusively on the training nodes. These settings reflect different real life scenarios Yang et al. (2016). Transductive learning can be applied to the classification of products in an online retail store; the labelled nodes represent the bestselling items, and the goal is to predict the category of the rest of the items. In this case, the aim is to predict the category of known nodes. With respect to inductive learning, an example is the prediction of the behaviour of new users in a social network. The model must generalize to unseen nodes.

6.1.2 Missing data

Missing data is a problem that occurs across a wide range of domains. For example, in medical related domains, data can be missing as a result of a patient being treated by different providers, or resorting to multiple laboratories which do not centralize their data Wells et al. (2013). Incomplete data can have several causes: the information is not available, e.g. if a sensor is malfunctioning, it will stop recording measurements; the information is missing at random, i.e. no external cause can explain the incompleteness; or the information is missing due to an underlying reason, e.g. a person in a survey refuses to answer some questions Graham (2009).

Amongst its other successes, deep learning has helped promote several ways of performing feature imputation, i.e. filling the missing values based on the observed values. For example, auto-encoders use a two step process that first learns a compact encoding of the data, similar to compression, and tries to decode the compact representation so as to maximize the similarity with the input Vincent et al. (2008). In a different vein, Generative Adversarial Networks (GANs) combine two networks where one feeds into the other: the first network (generator) tries to generate artificial input while the other network (discriminator) learns to classify the data as artificial or real. In feature imputation, the

generator imputes the missing values while the discriminator tries to find which values where imputed and which ones were observed Yoon et al. (2018).

GRAPE You et al. (2020) achieved state-of-the-art performance on the problem of feature imputation by reframing the learning objective as a graph representation problem: the data is viewed as a bipartite graph with two types of nodes: the observations and the features. The edges between the nodes are the observed features, where the weight of the edge is the value of the feature. The feature imputation is done by training a Graph Neural Network (GNN) on the graph.

In this chapter, we address the issue of the scalability of GRAPE: we propose a mini-batch version that works on datasets which do not fit in GPUs for full batch training. We test our architecture on a dataset which cannot be used with the original GRAPE, due to memory issues. Additionally, we present simple preprocessing and post-processing steps that allow GRAPE to be applied to graph structured data, compared to only tabular data in the original paper You et al. (2020).

Traditional approaches to solve feature imputation tasks include methods such as the Expectation-Maximization algorithm Dempster et al. (1977), k-nearest neighbours (k-NN) Troyanskaya et al. (2001), and matrix completion Candès and Recht (2009). These approaches may suffer from scalability issues, such as using a k-NN on millions of observations; or do not generalize to unseen observations. For example, matrix completion requires retraining to fit the new data.

Many types of data can be seen as specific cases of graph data, e.g. an image is an euclidean grid, a sentence is a tree. Graphs and graph-based neural networks, Graph Neural Networks (GNNs), can be used to take into account the structural information of the data while imputing the missing features. Matrix completion approaches can be extended to graphs Kalofolias et al. (2014), while GNNs can be used to impute missing data by treating observations and features as part of a bipartite graph You et al. (2020). In graphs, missing features can be imputed by propagating the other features in the graph Rossi et al. (2022).

Deep learning approaches can face issues in terms of GPU or TPU memory when the data becomes too large. In this setting, several approaches can alleviate the problem: sampling the data to create mini-batches Chiang et al. (2019);

optimizing the architecture by removing less influential parameters Wu et al. (2019); or performing a trade-off in time and space complexity by using reversible operations that can be computed sequentially Li et al. (2021a).

6.1.3 GNN's treatment of noise

The problem of noise in graph data has been mostly studied from the point of view of noise or attacks in the structure of the graph, or in the labels.

With respect to the structure, a powerful way of disturbing the graph is to use a gradient-based attack, called *projected gradient descent* (PGD) Xu et al. (2019a), which tries to find the smallest perturbations that produce the greatest decrease in performance. To protect the GNN against such attacks, more sophisticated aggregation schemes than the mean operator, such as the Soft Medoid Geisler et al. (2020) or the Soft Median aggregators Geisler et al. (2021), must be devised.

The noise is often considered from the point of view of the downstream task, e.g. what matters is not how noisy the features are, it is how noisy labels are and how the metrics of the model are impacted by the noise. Li et al. (2021b) explore in great depth how neural networks react to noisy labels. Dai et al. (2021) propose an architecture that is robust against noisy labels in graphs.

Instead of relying on robust architectures against noise, a different approach consist in denoising the data. This can be achieved using the graph filtering properties of GNNs Rey et al. (2022). Another example is the use of graph convolutions to denoise mesh data Shen et al. (2022).

Apart from additive noise, missing data can contribute to the noise. In this case, the features can be reconstructed using graph neural networks. One approach is to create a bipartite graph consisting of observations and features; the edges between the nodes representing the value of the feature for a given observation. A GNN is then trained to perform edge weight prediction You et al. (2020). Alternatively, the features of the nodes can be propagated to the neighbours to fill in the missing values Rossi et al. (2022).

Recent studies have also shown that certain architectures of GNNs possess an internal *implicit denoising* mechanism Ma et al. (2021). This mechanism is studied

in more depth in Liu et al. (2022).

Parallel to the effort of denoising data, there are works which focus on creating better features that leverage the topological information available in graphs Chien et al. (2022).

6.2 Noisy features

Most Graph Neural Networks (GNNs) work by propagating information through the graph. They adhere to the *message passing neural network* (MPNNs) framework. What separates each MPNN architecture is the mechanism by which information is transferred. For instance, GCNs Kipf and Welling (2017) use an explicit weighting scheme based on the degree of the nodes, while GATs Velickovic et al. (2018) employ an implicit weighting scheme by assigning a different importance to each node via the attention mechanism.

Noise in features can take several forms, the most common of which is *additive noise*, e.g. the input can be expressed as the sum of a "clean" input and an extra term. For example, the noise component can be Gaussian noise. The noise can be entirely random, or follow an underlying pattern. For example, missing data in clinical studies is not always random, but may have a latent cause, such as the patient refusing to disclose personal information Graham (2009). Similarly, if the categories contain the values "wolf" and "dog", the noise may not entirely be random.

More formally, given X the feature matrix of a graph $\mathcal{G}$, we define X_{noisy} to be the feature matrix with additive noise. If we add random noise sampled from a normal distribution with probability p, X_{noisy} follows Equation 6.1. Here X_{uv} designates the feature v of node u.

$$(X_{\text{noisy}})_{uv} = \begin{cases} X_{uv} + \epsilon, \epsilon \sim \mathcal{N}(0,1) & \text{with probability } p \\ X_{uv} & \text{otherwise} \end{cases}. \tag{6.1}$$

By definition of a graph, the nodes are interconnected and are therefore not independently identically distributed. This means that the noise in one node

can influence the other nodes. This raises the question: where is a GNN most vulnerable to noisy data? Is it when the noise is random, or when the noise targets a specific set of nodes, e.g. the most influential or the most isolated nodes. If the noise targets the nodes with the highest degree, the noise will reach more nodes faster than if it targets the nodes with the smallest degree. However, GNNs will often assign less importance to the features of the most influential nodes, because the more some information is used, the less it has discriminating power.

To model the targeting of nodes according to the degree, we can define a probability p that a feature will contain noise, and some threshold t representing the minimal degree that a node must have to be corrupted. The noise targeting the high degree nodes can be expressed by Equation 6.2.

$$(X_{\text{noisy}})_{uv} = \begin{cases} X_{uv} + \epsilon, \epsilon \sim \mathcal{N}(0,1) & \text{with probability } p \text{ if } u.degree \geq t \\ X_{uv} & \text{otherwise} \end{cases} \tag{6.2}$$

Conversely, targeting the small degree nodes can be done using Equation 6.3, where t is the maximal value of the degrees of the target nodes.

$$(X_{\text{noisy}})_{uv} = \begin{cases} X_{uv} + \epsilon, \epsilon \sim \mathcal{N}(0,1) & \text{with probability } p \text{ if } u.degree \leq t \\ X_{uv} & \text{otherwise} \end{cases} \tag{6.3}$$

While Equations 6.2 and 6.3 look similar, they only overlap when there is no noise, or when all the nodes are noisy.

6.2.1 Problem Definition

In this section, we first present the dataset on which our experiments are made. The architectures we chose, GCN, GAT, and GraphSAGE, each represent fundamental ideas of GNNs:

- GCN belongs to the family of spectral GNNs which exploit properties of the Laplacian matrix of the graph Wang and Zhang (2022);

- GAT leverages the attention mechanism, which has achieved state-of-the-art results in many fields, e.g. with Transformers in NLP Devlin et al. (2019);

- GraphSAGE uses a sampling mechanism which becomes essential when we scale GNNs to larger graphs Chiang et al. (2019).

We train the models on different versions of the dataset which we modified by adding noise to the features. We investigate three types of noise:

- We add random noise to the graph, as per Equation 6.1;

- We add noise to nodes, according to Equation 6.2, starting with the nodes with the highest degree and proceeding to the nodes with the smallest degree;

- We add noise to nodes, according to Equation 6.3, starting with the nodes with the smallest degree and procedding to the nodes with the highest degree.

The experiments are performed in two settings:

- *Transductive learning*, where the model has access to the full graph at training time; only the validation and test labels are hidden.

- *Inductive learning*, where the model is trained on the subgraph of the whole graph that contains only nodes in the training data. In this setting, edges between train and test nodes do not appear at training time.

Each model is trained for 1,000 epochs. The validation accuracy is used to select the best model. We perform 5 runs to account for the randomness of the parameter initialization and the noise addition.

6.2.2 Random noise perturbation

In this section, we assume that the noise is distributed randomly, i.e. follows Equation 6.1.

Transductive setting

The results of training a GCN, a GAT and GraphSAGE in the transductive setting with a randomly added noise are shown in Figure 6.1.

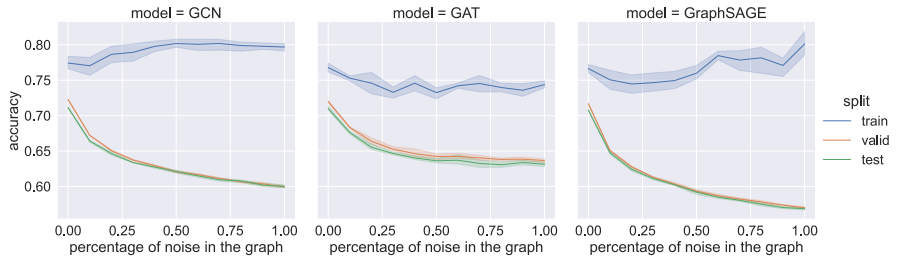


Figure 6.1: Training, validation and test accuracy of three models of GNNs in the transductive learning setting. The horizontal axis corresponds to the percentage of features to which noise has been added, i.e., p in Equation 6.1.

The sharpest decrease in accuracy occurs when $p \in [0, 0.2]$. When $p \geq 0.2$, the decrease follows a gentler slope. Because citation networks tend to be *scale-free* networks Barabási and Albert (1999), there are nodes in the networks, called *hubs*, from which we can reach almost all the nodes in the graph in a few hops. Formally speaking, the diameter of the graph, which is the greatest distance between two nodes, is small compared to the number of nodes. Thus, when $p = 0.2$, the noise has already spread throughout the graph.

In GCN and GraphSAGE, the generalization error between train and validation is increasing as p increases. This is due to the fact that these models do not possess any graph specific regularizing mechanism. Indeed, the weighting of the neighbours for the GCN in Equation 4.3 is proportional only to the degrees of the nodes. Similarly, in Equation 4.7, the regularizing mechanism of GraphSAGE is its sampling of neighbourhoods. In transductive learning, the effects of regularization by sampling are lessened by the fact that the model samples

from the whole graph.

Concerning the generalization error between validation and test, its existence in the first place is a result of a *distribution shift* that occurs between the different splits. This shift can manifest itself when the splitting of the dataset is not random. For instance, a good split for the Amazon dataset of products Chiang et al. (2019) is to split according to the number of sold items, because it is easier and more useful to label the bestselling items first and put them in the training set, and try to expand the labelling to the less popular items. Likewise, in citation or social networks, a challenging and natural split is to use the date information: the older nodes represent the training set while the test set comprises the most recent nodes Hu et al. (2020).

The distribution shift can have two causes: the structure of the graph can change, or the features can evolve. Firstly, the pattern in the edges between the nodes can change over time. For example, between 2006 and 2012, the Facebook graph changed drastically as the company expanded to other states and countries. The way people used the platform between these two dates evolved as well. Additionally, the structure can change as a result of periodical events. As an illustration, the subgraph of Twitter related to rugby will see yearly spikes of interest during the championship seasons, such as the Six Nations Tournament and the Champions Cup, and every four years when the World Cup takes place.

Secondly, the shift can also take place with respect to the features or the labels. For instance, what was considered rock music in the 1980s might not be labelled as such nowadays. Or the physical attributes of football players in the 1960s are very different from those of the current star players. In our case, the structural aspect of citation networks, and scientific collaborations at large, has been consistent for a long period of time. Indeed, the number of citations has followed a steady exponential growth for the last hundred years Dong et al. (2017). This means that the shift has occured with respect to the features and the labels.

Adding noise to the features undoes the distributional shift caused by the changing of features over time. When a certain level of noise is reached, the only remaining difference between nodes in the validation and the test set is structural. This explains the results of Figure 6.1, in which by the time all the nodes have been tampered with, there is no difference between the validation and test accuracy of the GCN and GraphSAGE models.

In the case of the GAT, going back to Equations 4.4-4.6, the α_{uv}^{l} performs an implicit assignment of different importances to the neighbours. This mechanism allows the model to downgrade the importance of nodes once their representation contains enough noise to negatively influence the learning process, e.g. when the nodes contribute to misclassifying other nodes. In short, the GAT performs regularization via attention.

Inductive setting

The results of training a GCN, a GAT and GraphSAGE in the inductive setting with a randomly added noise are shown in Figure 6.2.

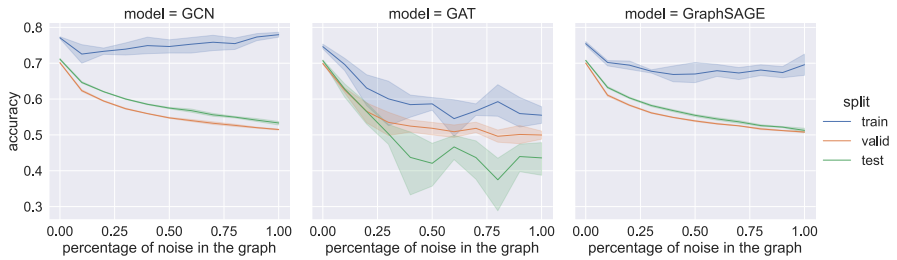


Figure 6.2: Training, validation and test accuracy of three models of GNNs in the inductive learning setting. The horizontal axis corresponds to the percentage of features to which noise has been added, e.g. p in Equation 6.1.

The first observation is that overall, the test accuracy is lower in the inductive learning setting than in the transductive learning setting. This is expected, because the inductive task is inherently a more challenging task than the transductive one Yang et al. (2016). Having access to the test data at training time allows the model to fit the structure of the graph. Moreover, if there is a change in the feature distribution between training and validation, the model can adapt to this change at training time in the transductive setting.

When $p = 1$, there is still a generalization error between the validation and test accuracy in the GCN model. The lower accuracy is explained by the fact that inductive learning is inherently a more challenging task than transductive learning Yang et al. (2016). The gap is the result of structural differences between the training graph and the full graph. For example, since nodes become popular

slowly over time, the structure may have evolved as the result of the formation of new hubs.

Conversely, the generalization error almost disappears in the case of GraphSAGE. This is the result of the sampling mechanism, which appears in Equation 4.7. The sampling acts as a regularizer, which also explains why the gap between the train and validation accuracy is smaller than with the GCN. Since the sampling prevents overfitting to the structure, the gap represents the information lost via noise. When $p \geq 0.4$, most of the information in the features has been drowned in the noise.

In contrast, GAT's performance present several peculiarities. Firstly, the model performs considerably worse than in the transductive setting. It also achieves a smaller test accuracy than its GCN and GraphSAGE counterparts, with the accuracy dropping below the 50% mark. Secondly, the model presents a wider range of outcomes for the different runs. This is an indication that the model is converging to different local minima at each run. Additionally, the low training accuracy indicates that the model performs excessive regularization.

This regularization is performed via the attention mechanism presented in Equations 4.4-4.6. When we consider the backpropagation phase Lecun et al. (1998) of a single node, we see that the noise will be amplified in two ways. In the first part, neighbours who share the same class as the node, but whose features have been tampered with so that they push the model to misclassify the node, will see their α_{uv}^{l} coefficient reduced, e.g. the model will attribute less importance to these nodes. In the second part, neighbours with a different class but whose features push the model to correctly classify the nodes, will have an increased α_{uv}^{l} coefficient. In other words, information will be silenced while noise will be amplified.

6.2.3 High degree node perturbation

In this section, we assume that the noise targets the high degree nodes first, according to Equation 6.2.

Transductive setting

The results of training a GCN, a GAT and GraphSAGE in the transductive setting with noise progressively added to the nodes, starting with the nodes with the highest degree, are shown in Figure 6.3. The horizontal axis represents the proportion of nodes that contain noise, e.g. the normalized t in Equation 6.2. Each color represents a different amount of noisy features per nodes, e.g. the p in Equation 6.2. We investigate three settings: minimal noise, with $p = 10\%$; medium noise, with $p = 50\%$; and high noise, with $p = 90\%$.

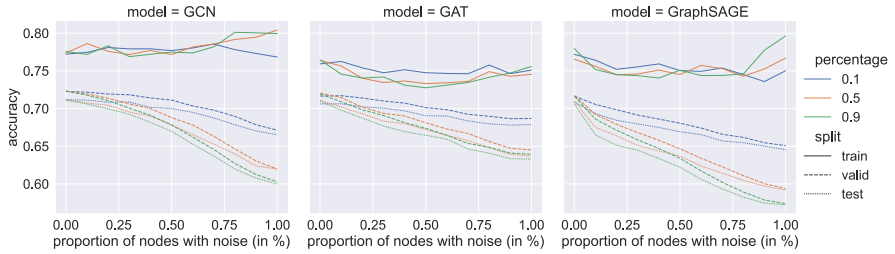


Figure 6.3: Training, validation and test accuracy of three models of GNNs in the transductive learning setting. The horizontal axis corresponds to the percentage of nodes to which noise has been added. The noise targets the nodes with the highest degree first then proceeds in a decreasing manner. The percentage indicates how many of the features of each node are tampered with, e.g. 0.1 means 10% of the features contain added noise.

For a given $t < 1$ and any p, the test accuracy of each model is higher than in the random noise setting.

Noise that spreads from an influential node will go through many nodes to reach the most isolated ones, by which time the effects of the noise will have been dampened. This phenomenon can be visualized with the following experiment, similar to the *small world* experiment de Sola Pool and Kochen (1978): the goal is to reach the best coverage of the U.S. population with the smallest group of people, in the sense that we want to select a group that minimizes the distance between any individual in the whole population and the group, e.g. each person is at least 5 hops away from the group.

If you select the members of the groups by taking the people with the highest degree, you will target people from the big cities. But you will miss the people

from isolated areas. Moreover, while some people will not be within reach of a member of the group, others will be within reach of several members. When, instead of selecting the people by their degree, you pick them randomly, you reduce the overlapping of the coverage and you increase the chance of selecting members that can reach the isolated nodes. Regarding our experiment, this means what when the noise is added to the high degree nodes, some nodes will not encounter any noise because they will be beyond reach of the L neighbourhood of the high degree nodes, where L is the number of layers of the GNN. Note that this also applies to GNNs with skip connections Xu et al. (2021b), although the depth the GNN then exceeds L.

Looking at GCN and GAT, we see a only a relatively small drop in performance until at least $t \geq 0.4$. This can be explained by going back to the equations of each model. In the GCN case, the node-wise update rule can be expressed by Equation 6.4. T is the transpose operator.

$$h_u^{l+1} = (W^l)^T \sum_{v \in \mathcal{N}_u \cup \{u\}} \frac{1}{\sqrt{\hat{d}_v \hat{d}_u}} h_u^l \tag{6.4}$$

The weighting of the nodes in the update rule depends doubly on the degree of the nodes: the degree of the target node, i.e. the node that is updated, and the degree of the source node, that is, the neighbour from which the edge starts. At a local level, considering only a single node update, which can be done by fixing the u, the nodes with the highest degree will always have lower importance than the other nodes. More formally, given a node u, two neighbours v_1 and v_2 such that $d_{v_1} > d_{v_2}$, we have $(1/d_{v_1})^{-1/2} < (1/d_{v_2})^{-1/2}$. Likewise, given two nodes u_1 and u_2 who share a common neighbour v, by the same reasoning we find that if $d_{u_1} > d_{u_2}$, u_2 will have a greater weight than u_1. This means that the GCN's explicit weighting scheme devalues the contributions from high degree nodes. This also means that targeting the high degree nodes is the least efficient way of propagating noise in a GCN model.

Regarding the GAT model, the unnormalized attention coefficient between two nodes u and v, given by Equation 4.5, is $a(W^l h_u^l, W^l h_v^l)$ where a is the attention mechanism. a can be any learnable function. For example, in the original paper Velickovic et al. (2018), it is given by Equation 6.5. Θ is the weight vector of the mechanism and $\|$ is the concatenation operator.

$$e_u^l v = \text{LeakyRELU}(\Theta^T [W^l h_u^l \| W^l h_v^l]) \tag{6.5}$$

The loss used in the training is the *cross-entropy* loss, given by Equation 6.6. K is the number of classes, e.g. 40 in our case. x is the features of the node, y its label, represented as a *one-hot encoding*; and $\hat{y}$ the prediction of the GNN. $\hat{y}$ is usually taken to be a probability distribution, i.e. $\sum_{i=0}^{K} \hat{y_i} = 1$. The predicted category is the one with the highest score.

$$l(x, y) = \sum_{i=0}^{K} -y_i \log(\hat{y}_i) \tag{6.6}$$

When the noise targets the high degree nodes, enough noise will push the model to misclassify nodes. In terms of loss, this means that the noise will increase the loss of the model. During the backpropagation stage, the weights will be modified using gradient descent to decrease the loss. Let u be a node and v_1 one its neighbours. If v_1 contains noise that results in decreasing the score of the prediction of the correct class of u, the loss increases. To reduce the loss, the model will change its weights, and will reduce the importance of v_1 in updating u, by lowering α_{uv_1}. Because the high degree nodes will be involved in many computations, the same reasoning shows that the overall importance of these nodes will decrease. In short, the attention mechanism will silence the noise.

In terms of the percentage of features altered, we observe two phenomena: when only a few features are modified, e.g. $p = 0.1$, the performance deteriorates only slightly; and whether we set $p = 0.5$ or $p = 0.9$, the performance is similar. The second point was already observed in Section 6.2.2 regarding random noise: beyond a certain amount of noise, the performance plateaus as the noise has spread through the whole graph and there is nothing else remaining to tamper with.

A very likely explanation for the first phenomenon is the fact that many types of GNN perform some form of *implicit denoising* during their training phase Ma et al. (2021). In particular, the authors of Ma et al. (2021) show that GCN and GAT can be seen as solving a specific form of denoising problem. This idea of denoising is further investigated in Liu et al. (2022), in which the authors

explore how denoising is impacted by the connectivity and the size of the graph, as well as by the architecture.

Inductive setting

The results of training a GCN, a GAT and GraphSAGE in the inductive setting with noise progressively added to the nodes, starting with the nodes with the highest degree, are shown in Figure 6.4.

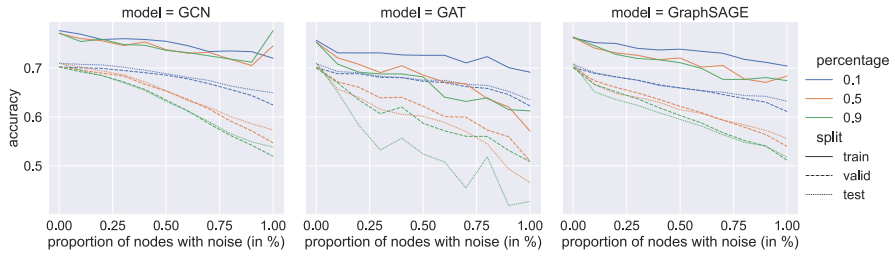


Figure 6.4: Training, validation and test accuracy of three models of GNNs in the inductive learning setting. The horizontal axis corresponds to the percentage of nodes to which noise has been added. The noise targets the nodes with the highest degree first then proceeds in a decreasing manner.

While GraphSAGE's performance in the transductive setting was the poorest of the three models, in the transductive setting it is on par with the other models. The reason for the early drop in accuracy in the transductive and inductive settings, e.g. when $p \in [0, 0.2]$, is likely the over-representation of high degree nodes in the sampling of the neighbours. When looking at the neighbourhood sampling of all the nodes, the nodes with high degree will appear multiple times. Thus, the sampling allows the noise to spread faster. As the noise fills the whole graph, the inductive mechanism of GraphSAGE starts to be more influencial in reducing the overall noise of the messages.

In contrast with the previous experiments, we see that the validation and test gap reaches a minimum around $p = 0.6$, and starts increasing again when $p > 0.6$. In all likelihood, this is due to the fact that apart from the papers who cite and are cited by few others, the newer papers tend to have less citations on average than the older ones, because there needs to be a certain period of time before

the paper starts getting cited. By targeting the nodes with the higher degree first, we are creating a distributional shift. Indeed, the models are trained on nodes with noise while the nodes in the validation and test set contain less noise in average. This leads to the models's generalizing capabilities to decrease.

6.2.4 Small degree node perturbation

In this section, we assume that the noise targets the small degree nodes first, according to Equation 6.3.

Transductive setting

The results of training a GCN, a GAT and GraphSAGE in the transductive setting with noise progressively added to the nodes, starting with the nodes with the smallest degree, are shown in Figure 6.5.

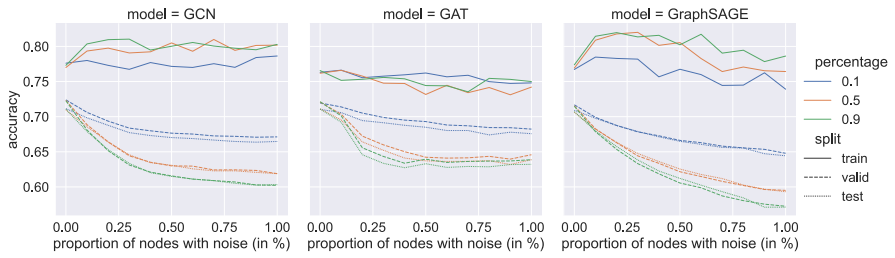


Figure 6.5: Training, validation and test accuracy of three models of GNNs in the transductive learning setting. The horizontal axis corresponds to the percentage of nodes to which noise has been added. The noise targets the nodes with the smallest degree first then proceeds in an ascending manner.

The model performance drops sharply when $t \in [0, 0.3]$, then it levels off. A similar reasoning from Section 6.2.3 about the degrees of the nodes shows that the GCN model will amplify the noise: the values of the nodes with the smallest degree are weighted with the largest coefficients. Likewise, the GAT will start by penalizing heavily the small degree nodes. Only when enough noise is present in the graph will the attention mechanism be able to rescale the importance of the noisy nodes.

Because GraphSAGE proceeds with a fixed neighbourhood sampling size, there is no neighbourhood sampling for the nodes with small degree: all their neighbours are considered. Therefore, GraphSAGE has no regularizing mechanism for small degree nodes. The lack of weighting, whether implicit or explicit, does not allow GraphSAGE to penalize the noisy nodes in any other way than by modifying the model's parameters, which results in low generalizing capabilities as the model is trying to fit against the noise.

We observe that, whether the noise is randomly distributed across the nodes or whether it targets high degree or small degree nodes, the presence of noise in the nodes's features is more important than the quantity of features modified. In other words, t is more important than p. Increasing t will decrease the accuracy more than if we increase p.

Inductive setting

The results of training a GCN, a GAT and GraphSAGE in the transductive setting with noise progressively added to the nodes, starting with the nodes with the smallest degree, are shown in Figure 6.6.

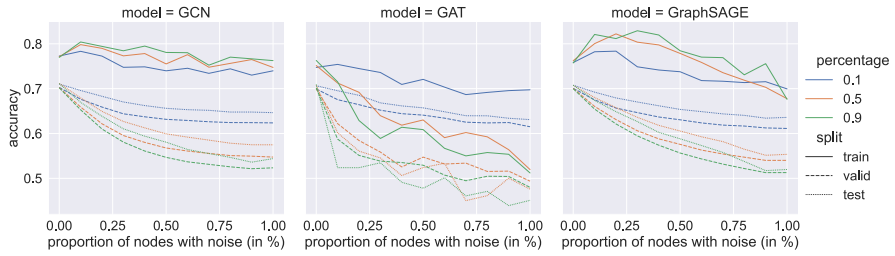


Figure 6.6: Training, validation and test accuracy of three models of GNNs in the inductive learning setting. The horizontal axis corresponds to the percentage of nodes to which noise has been added. The noise targets the nodes with the smallest degree first then proceeds in an ascending manner.

Except for the GAT, the validation accuracy is actually higher than the test accuracy. This indicates that the model is not learning properly. It is just fitting the noisy training data. The gap is consistent regardless of the amount of noise. This result hearkens back to the spread of fake news which can start more easily

as a rumour then spread to infect the whole population Shu et al. (2017). The graph can be more easily subverted by targeting the more isolated nodes which do not possess enough information about the rest of the graph to stave off the noise.

In whatever way the features are corrupted, the difference between transductive and inductive learning is amplified with the noise.

6.2.5 Extension to other types of architectures

In this section, we go beyond the GCN, GAT and GraphSAGE models to look at two other models: the Graph Isomorphism Network (GIN) Xu et al. (2019b), which is an MPNN model provably more powerful at discerning certain graph structures than GCN and GraphSAGE; and Simple Graph Convolution (SGC) Wu et al. (2019), a GNN which simplifies the computations performed in the GCN and contains only a single layer.

Graph Isomorphism Network

A single layer of the GIN model follows Equation 6.7. MLP^l is a multi-layer perceptron. $\epsilon^{(l)}$ is either a learnable parameter, or a fixed scalar value.

$$h_v^l = \mathrm{MLP}^l\left(\left(1+\epsilon^{(l)}\right)\cdot h_v^{l-1} + \sum_{u\in\mathcal{N}_v} h_u^{l-1}\right) \tag{6.7}$$

Compared with the GCN equation, the multi-layer perceptron adds more expressivity: it allows the model to adjust on a feature per feature basis, which results in more degrees of freedom. It can capture more patterns than the explicit weighting on the degrees of the nodes. The results of the training of the GIN with random noise in transductive and inductive learning are shown in Figure 6.7.

We see that the overall performance of the model, both in transductive and inductive learning, is far below that of the architectures studied above. This is due to the multilayer perceptron, or fully connected layer, that allows the model

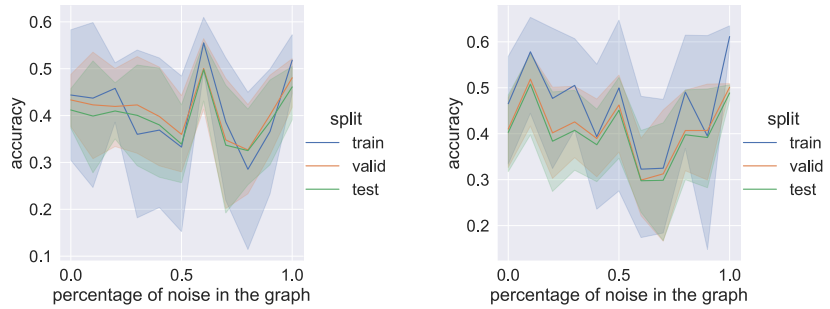


Figure 6.7: Training, validation and test accuracy of the GIN model in the transductive (left) and inductive (right) learning settings. The horizontal axis corresponds to p in Equation 6.1.

to weight each feature individually. This makes the model more sensitive to variations in the features. We also see a decrease followed by an increase in the accuracy when we increase p. This variation reflects the complexity of the task: when half the features are noisy, the model can learn to assume that half the inputs are noisy. When the ratio between clean and noisy features is different, the model might constantly oscillate to find a suitable value. This can be seen by the divergence in outcome which is less pronounced in the transductive case when $p = 0.6$.

Simple Graph Convolution

The SGC model was conceptualized as a simplification of the GCN. By writing $S = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$, the SGC model follows Equation 6.8. σ denotes an activation function, like the Softmax operator. K is a hyperparameter of the model. Θ represents the parameters of the model.

$$\hat{y} = \sigma(S^K X \Theta) \tag{6.8}$$

S^K gives the maximal depth of the neighbourhood that can be achieved. If S is the adjacency matrix, then S^K can represent K-hop neighbourhoods at most.

In comparison with the GCN model, the SGC first performs the matrix multiplication S^K, then combines the resulting power matrix with the parameters. The results of the training of the SGC with random noise in transductive and

inductive learning are shown in Figure 6.8.

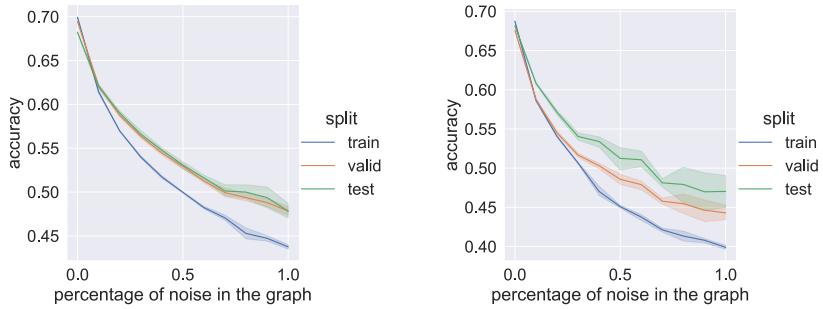


Figure 6.8: Training, validation and test accuracy of the SGC model in the transductive and inductive learning settings. The horizontal axis corresponds to the percentage of nodes to which noise has been added. The noise targets the nodes with the smallest degree first then proceeds in an ascending manner.

We observe that the performance decreases smoothly with the noise. This is expected because SGC is part of the architectures which exhibit implicit denoising behaviour Ma et al. (2021). However, because the model has a single layer, it is not possible to incrementally penalize the noise, which explains why the accuracy is lower than with the other models.

6.3 Missing data imputation

In this section, we extend the GRAPE architecture to the mini-batch setting that allows the model to work on datasets that do not fit in the GPU memory. We briefly describe the sampling approaches that are suitable for GRAPE learning. We also introduce lightweight preprocessing and post-processing steps that allow GRAPE to perform feature imputation on graph structured data.

6.3.1 GRAPE

GRAPE You et al. (2020) is a GNN architecture adapted from the GraphSAGE model Hamilton et al. (2017). The nodes used in GRAPE are of two types: observations and features. The graph is bipartite, where each edge has one end

in the set of observations, and the other end in the set of features. An edge e_{uv} between observation u and feature v means that observation u's value of feature v is $\mathbf{e_{uv}}$. At each layer of GRAPE, a message is computed for each node using its neighbours, and the message is combined with the node's representation at the previous layer to obtain the new representation.

More formally, let AGG_l denote the aggregation function of layer l, e.g. the mean or max function. Let CONCAT denote the function that concatenates tensors. Let $P^{(l)}$, $Q^{(l)}$ and $W^{(l)}$ be trainable weights for the layer l. Then the forward pass of the GRAPE model is described by Equations 6.9-6.11.

$$n_v^{(l)} = \text{AGG}_l \left(\sigma \left(P^{(l)} \cdot \text{CONCAT}(h_v^{(l-1)}, \mathbf{e}_{uv}^{(l-1)}) \right) \mid \forall u \in \mathcal{N}(v, \mathcal{E}) \right) \tag{6.9}$$

$$h_v^{(l)} = \sigma \left(Q^{(l)} \cdot \text{CONCAT} \left(h_v^{(l-1)}, n_v^{(l)} \right) \right) \tag{6.10}$$

$$\mathbf{e}_{uv}^{(l)} = \sigma \left(W^{(l)} \cdot \text{CONCAT} \left(\mathbf{e}_{uv}^{(l-1)}, h_u^{(l)}, h_v^{(l)} \right) \right) \tag{6.11}$$

To obtain the missing value of feature v of node u, a feedforward neural networks can be used on $h_u^{(L)}$ and $h_v^{(L)}$.

6.3.2 Scalable GRAPE

In order for GRAPE to work with graphs that do not fit in the GPU memory, we must introduce a sampling scheme that produces mini-batches on which the model can be trained. We present the modified version of GRAPE to scale for graphs that exceed the GPU memory. The algorithm appears in Algorithm 1, which is adapted from You et al. (2020). SAMPLE is a function that takes a graph as input and produces mini-batches according to a sampling strategy. $\mathcal{B}.nodes$ and $\mathcal{B}.edges$ represent respectively the set of nodes and of edges in the mini-batch; we have $\mathcal{B}.nodes \subseteq \mathcal{V}$ and $\mathcal{B}.edges \subseteq \mathcal{E}$.

Algorithm 1 Sample-based GRAPE forward propagation.

for $\mathcal{B} \in$ SAMPLE$(\mathcal{G})$ **do**
  for $v \in \mathcal{B}.nodes$ **do**
    for $l \in \{1, \dots, L\}$ **do**
      $n_v^{(l)} \leftarrow$
      $\text{AGG}_l \left(\sigma(P^{(l)} \cdot \text{CONCAT}(h_v^{(l-1)}, \mathbf{e}_{uv}^{(l-1)})) \right.$
      $\left. \mid \forall u \in \mathcal{N}(v, \mathcal{B}.edges) \right)$
      $h_v^{(l)} \leftarrow \sigma \left(Q^{(l)} \cdot \text{CONCAT} \left(h_v^{(l-1)}, n_v^{(l)} \right) \right)$
      $\mathbf{e}_{uv}^{(l)} = \sigma \left(W^{(l)} \cdot \text{CONCAT} \left(\mathbf{e}_{uv}^{(l-1)}, h_u^{(l)}, h_v^{(l)} \right) \right)$
    end for
  end for
end for

Node-wise sampling

Introduced in Hamilton et al. (2017), the node sampling strategy works by sampling a subset of the neighbours of each node and training the model on the induced subgraph. The sampling can be expanded beyond 1-hop neighbours by sampling the neighbourhood of the neighbours. The space complexity of the sampling strategy is $O(B^L)$, where B is the number of neighbours sampled, compared to $O(D^L)$ with D the average node degree for batch training using the full neighbourhood of each node.

There are several strategies that can be used to produce the mini-batches. A more complete survey of approaches used for large scale graph training can be found in Duan et al. (2022). Due to the nature of the graph, e.g. a bipartite graph, the use of cluster based sampling, such as ClusterGCN Chiang et al. (2019) that create clusters based on graph partitioning algorithms such as METIS Karypis and Kumar (1998), does not fit our approach.

6.3.3 GRAPE for graph data

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $X \in \mathbb{R}^{|\mathcal{V}| \times d}$, we can create an undirected bipartite graph $\mathcal{G}_{bp} = (\mathcal{V}_{bp}, \mathcal{E}_{bp})$ where $\mathcal{V}_{bp}$ contains two types of nodes: the observations and the features. $\mathcal{E}_{bp}$ contains the edges between the observations and the nodes, and the weight of the edges represent the value of the feature. We have

$|\mathcal{V}_{bp}| = |\mathcal{V}| + d$ and $|\mathcal{E}_{bp}| = |\mathcal{V}| \times d$. We also add a node identification attribute to the bipartite graph.

To apply GRAPE to graph structured data, we can use the following workflow, which is also demonstrated in Figure 6.9.

- a bipartite graph is created using the input graph;
- GRAPE is trained on the bipartite graph to predict the missing edges;
- the node identifier allows the mapping of the predicted edges to the features in the original graph.

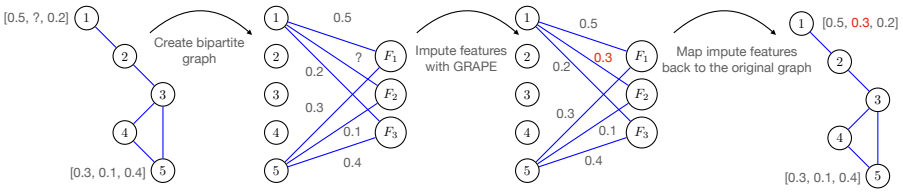


Figure 6.9: Example of the use of GRAPE on graph data. Edges and features are omitted for clarity. **Enlarge the figure**

6.3.4 Experiments

In this section, we present the experiments performed on a graph dataset whose associated bipartite graph does not fit in the GPU. We first investigate the training behaviour under different missing data regimes, then we perform an ablation study of the different parameters of the model.

6.3.5 Training behaviour

We train the mini-batch GRAPE model for 20 epochs. We use the *Mean Absolute Error* (MAE) as the loss function. The training and validation loss for different levels of missing data are shown in Figure 6.10.

We observe *overfitting* of the training data in the regime of extreme missing data, e.g. when we have less than 1% of the feature values. Conversely, the validation

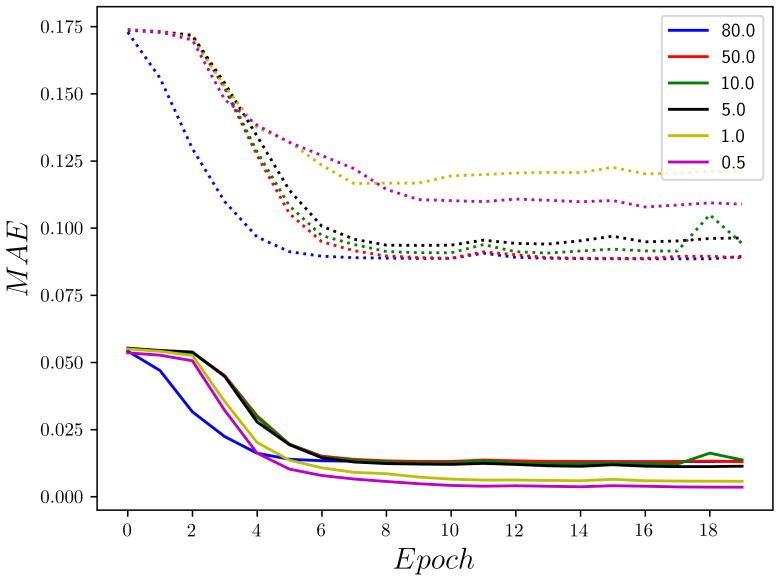


Figure 6.10: Train (line) and validation (dashed line) loss for different percentage of observed data. The lower the percentage, the higher the percentage of missing data. Under extreme scarcity, the model starts overfitting the training data.

loss starts increasing again after a certain number of epochs when there is a scarcity of observable features.

6.3.6 Ablation study

We evaluate the impact of the hyperparameters in the learning process. We test the influence of the model capacity, e.g. the hidden dimensions for the nodes and the edges. We also explore the influence of dropout. We investigate the importance of the number of layers in the model, as well as the batch size. The test MAE for the different runs appear in Table 6.1.

Table 6.1: Ablation study for the mini-batch version of GRAPE. The baseline model has batch size $b = 14000$, number of layers $l = 3$, $dropout = 0.0$ (no dropout), hidden dimension $n_l = 64$. The values in the table represent the test MAE. All the errors have to be multiplied by 10^{-2}.

Model	Test MAE for % of missing features					
	20	50	90	95	99	99.5
Baseline	8.82	8.83	9.40	9.76	12.5	13.5
$dropout = 0.2$	9.11	9.10	9.45	9.80	11.2	10.6
$l = 1$	8.87	8.85	9.00	9.20	11.2	11.9
$n_l = 16$	8.85	8.83	8.95	9.40	11.0	20.0
$b = 1000$	8.71	9.02	9.53	11.2	16.8	18.0

Reducing the model capacity, both in terms of the number of layers and the dimensions of the layers, improve performance, except in the case of extreme scarcity of observable data. This can be explained by the fact that a smaller model will overfit less, as it does not possess the capacity to interpolate the train data entirely.

Conversely, the fewer the observable features are, the more reducing the batch size will decrease the performance of the model. This is likely due to the fact that in a sparse feature regime, the model will fit entirely the training data of the batch, e.g. small training loss, but this will not generalize to unseen features.

A possible explanation for the lack of performance improvement when using dropout is that the mini-batch is already acting as a regularizer, so the effects of dropping edges is less significant than in a full batch setting. When the observable features are sparse, the effect of dropout is stronger as it helps alleviate the

overfitting of the model.

6.4 Discussion

In this chapter we investigate the sensitivity of different types of graph neural networks, especially message passing neural networks, to the noise in the features of the nodes. We explored different types of noise: random noise, where noise is added randomly to the features; noise which targets the high degree nodes; and noise which targets the small degree nodes. GNNs are trained both in the transductive setting and in the inductive learning, i.e. with and without access to the test data at training time. We find that weighting mechanisms, whether they are explicit, as in the GCN, or implicit, such as the attention mechanism in the GAT, help reduce the influence and the propagation of the noise.

We also found that some architectures present an implicit denoising mechanism that allows the model to maintain a certain level of accuracy under the presence of noise. Furthermore, architectures which lack this mechanism exhibit a more chaotic pattern of learning.

Out of the three types of noise, the noise targeting the small degree nodes first is the most challenging for the models. This reflects the fact that these nodes are more isolated; thus noise will have a relatively higher impact than it would for nodes with more neighbours.

We find that random noise in transductive learning leads to overfitting of the data. In inductive learning, the gap between training and validation accuracy is large, as well as the gap between validation and test accuracy.

In this chapter, we also proposed a new GNN architecture for missing data imputation. We extend the GRAPE architecture to handle mini-batches. We also add preprocessing and post-processing steps to perform feature imputation on graphs with GRAPE. Experiments show that mini-batch GRAPE performs well under different schemes of missing data, with performance decreasing when more than 99% of the features are missing. The ablation study shows that the model performs better with fewer layers and that the batch sampling acts as a good regularizer when there are enough observations.

Perspectives for future works regarding noisy features include analyzing similar types of perturbations for categorical features, and in temporal graphs. Regarding missing data imputation, perspectives include extending the model to incorporate the structural information of graphs when using GRAPE with graph data.

Chapter 7

Patch Extraction in Medical Imaging

This chapter deals with the problem of extracting patches in medical images. The goal is to find the patches with the most information for image classification with Convolutional Neural Networks (CNNs). In this chapter, we study the role of entropy and a spectral based similarity criterion on the training time of a neural network. We use the region of interest of the image to maximize the relevance of the patches for the classification task. The use of patches instead of the whole image allows us to study the influence of entropy at different scales. We evaluate two metrics: the time it takes for the CNN to converge, and the accuracy of the model.

Section 7.1 presents a brief overview of medical imaging, patch-based classification and information measures. Section 7.2 introduces the dataset and the two criteria we use: the ISIC dataset, entropy, and the Mean Exhaustive Minimum Distance criterion. Section 7.3 presents the results, in terms of training time and accuracy. Section 7.4 concludes this chapter.

The results in this chapter were published in Lachaud et al. (2021, 2022d).

Contents

7.1 Introduction

Convolutional Neural Networks (CNNs) have become one of the most effective machine learning solutions for computer vision problems such as classification, object detection, face recognition, etc. More specifically, CNNs are extensively used for medical image processing tasks. The main goal of this field is to extract relevant clinical information or knowledge from medical images. One can mention, for instance, computer-aided diagnosis of cancer using classification methods Anwar et al. (2018). Cancer is one of the leading causes of deaths worldwide Ritchie and Roser (2018). However, if the cancer is diagnosed early, chances of survival are far greater than for later stages Siegel et al. (2021). For this reason, there has been a lot of research focused on leveraging deep learning to improve cancer diagnosis and prognosis, especially in breast cancer Yala et al. (2019), skin cancer Esteva et al. (2017), and lung cancer Marentakis et al. (2021).

Early diagnosis is considered in terms of one of the four stages of cancer, not in terms of time since the onset of the disease. These two notions can coincide, but a cancer can also stay in stage I for decades. Stage IV, the final stage, is when the cancer has metastasized, i.e., has spread to other sites.

Medical images can be widely different depending on their source, such as CT (Computed Tomography) scans, MRI (Magnetic Resonance Imaging) images,

dermoscopy, etc. Typically, image classification tasks take as input the entire image. However, in some situations training an image patch, that is, a subset of the entire image, might be preferable. Not only is this less time consuming, but it can also improve the classifier performance in some particular situations. For instance, in Hou et al. (2016), the authors claimed that in cancer subtypes classification, the decision is mostly based on cellular-level visual features observed on image patch scale. Another example where patch based classification was preferred over pixel based classification is presented in Roy et al. (2019) where this approach was used for classification of breast histology. One can find other applications of patch-based classification in Rousseau et al. (2011) and Zhang et al. (2014).

A judicious choice of patches reduces the importance of noise and focuses on the most important parts of the image. Two approaches of selecting the patches used for classification are to score the patches individually based on a given metric, or to compare each patch with the other patches of an image and rank the similarity between the patches. In the first approach, the patches can be scored using entropy, while the second approach relies on a similarity measure between images.

On one hand, entropy is used in information theory as a way to quantify the level of information of an object. Higher entropy means that there is more information in the object. For instance, a random noise image has high entropy while a unicolored one has very low entropy. Entropy plays an important role in data compression where it provides the lower bound on the storage required to compress an object without loss of information Shannon (1948). Entropy can also be used for object reconstruction using the principle of maximum entropy, which aims at selecting the most uniform probability distribution amongst multiple candidate distributions. It can be used for image reconstruction where the candidates are the set of missing pixels Skilling and Bryan (1984). It applies to text data as well Nigam et al. (1999). Entropy can also be used in image texture analysis Zhu et al. (1997) and texture synthesis.

On the other hand, the Mean Exhaustive Minimum Distance (MEMD) is a criterion that was introduced in Havlíček and Haindl (2019) to compare two images by trying to find the best pairing of pixels from the first and the second image; the criterion score then indicates how similar the images are. A low score indicates that the images are similar, and a high score that the image are

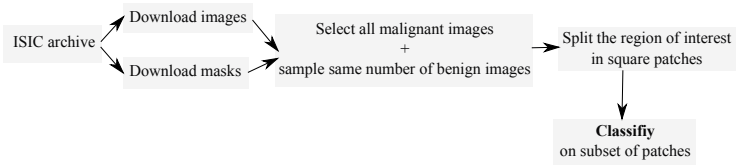


Figure 7.1: Data pre-processing workflow

different. This can be extended to patches, where we compare a single patch with several patches by averaging the scores.

The data is publicly available at https://www.isic-archive.com

The dataset we used in this study comes from the ISIC archive (International Skin Imaging Collaboration). Because of the lethality of melanoma cancer, the ISIC project was created to help improve skin cancer diagnosis via imaging data. They started an annual challenge in 2016 Gutman et al. (2016), and from 2019 onwards, the challenges have focused on dermoscopic image classification, with multiple diagnostic categories Rotemberg et al. (2021). The researchers who had the best results on the 2019 ISIC challenge Gessert et al. (2020a) studied patch-based classification on the HAM10000 dataset Tschandl et al. (2018) in Gessert et al. (2020b). They took multiple patches from each image and used an attention-based approach to combine the information from the patches and classify the images.

7.2 Proposed method

7.2.1 Dataset description and pre-processing

The ISIC archive database (see Rotemberg et al. (2021)) contains images of skin lesions which can be benign or malignant; other images can also have an unknown status. The image resolution varies across the datasets. The archive also has an API which can be used to get information about images or to retrieve lesion masks created by expert users. Our goal is to perform binary classification using patches of images. Our target variable has two labels indicating whether the lesion is *benign* or *malignant*.

The API is accessible here: https://isic-archive.com/api/v1

Originally, the ISIC challenge had more refined categories. Here we use only 2.

All the data pre-processing steps are described in Figure 7.1:

(a) Image (b) Mask

Figure 7.2: Example of a malignant skin lesion and its mask.

(a) 32×32 (b) 64×64 (c) 128×128 (d) 256×256

Figure 7.3: Example of patches of different size of the image from Figure 7.2.

Table 7.1: Number of patches for different patch sizes

patch size	number of patches
32×32	4,886,969
64×64	1,173,052
128×128	270,821
256×256	58,253

1. We download images from the ISIC archive, as well as the masks that are annotations from experts and indicate the lesion location.

2. We select all the malignant images with a mask. We sample the same number of benign images.

3. We create square patches of width 32, 64, 128 and 256. The patches are taken from the region of the interest of the image, as defined by the downloaded masks.

4. We compute the entropy of the patches, and use it to extract a subset of patches. This is explained in section 7.2.2.

5. We compute a spectral measure of similarity between a patch and all the patches of the same image; we use this measure to extract a subset of patches. The details are in section 7.2.3.

6. We train a classifier on all the datasets we have created in the two previous steps.

Table 7.2: Number of patches for each patch size

Patch size	**Number of patches**
32×32	$4,889,969$
64×64	$1,173,052$
128×128	$270,821$
256×256	$58,253$

We divide the images in three groups: 90% of the images are in the train set, with 20% of the train set reserved for validation; the remaining 10% constitute the test set.

7.2.2 Entropy

We are interested in the study of the behavior of the Shannon entropy Shannon (1948) of the images. The formula used for the calculation of entropy is the following:

$$H = -\sum_{k=0}^{M} p_k \log_2(p_k) \tag{7.1}$$

where M is the highest intensity of a pixel (in our case, 255), and p_k is the probability associated with the pixel intensity k in the grayscale image. In practice, the entropy is computed using histograms to estimate the probabilities. The entropy can take values between 0 and $\log_2(255) \approx 8$. Although the images in the dataset are in the RGB format, the entropy is computed on the grayscale version of the images. Our choice was motivated by the fact that there is no consensus on how to compute the entropy of an RGB image: Equation 7.1 does not have a canonical generalization to RGB images, while RGB conversion to grayscale is standardized in the ITU-R Recommendation BT.601-2.

Figure 7.4 shows the distribution of entropy amongst the patches for different patch sizes.

Table 7.3 shows the mean, standard deviation and some quantiles of entropy. We observe that, as the patch size grows, so does the entropy. This is expected because the more pixels we have, the more likely they are to have different intensities, which lead to a higher entropy. Also, the entropy for larger patch sizes is slightly more centered around the mean, which may be due to the fact that larger patch sizes will average some of the more extreme patches of smaller size. For example, instead of having multiple small patches of low and high entropy, a larger patch containing all the small patches will have a more average entropy.

We are interested in the impact of the entropy on the training of a classifier: whether it is faster to train on a dataset with low entropy than with a dataset with standard entropy; and whether a dataset with higher entropy is harder to train on. We split the created patches in three groups for each patch size :

- one containing the patches with entropy below the 15-th quantile, referred to as *low*.

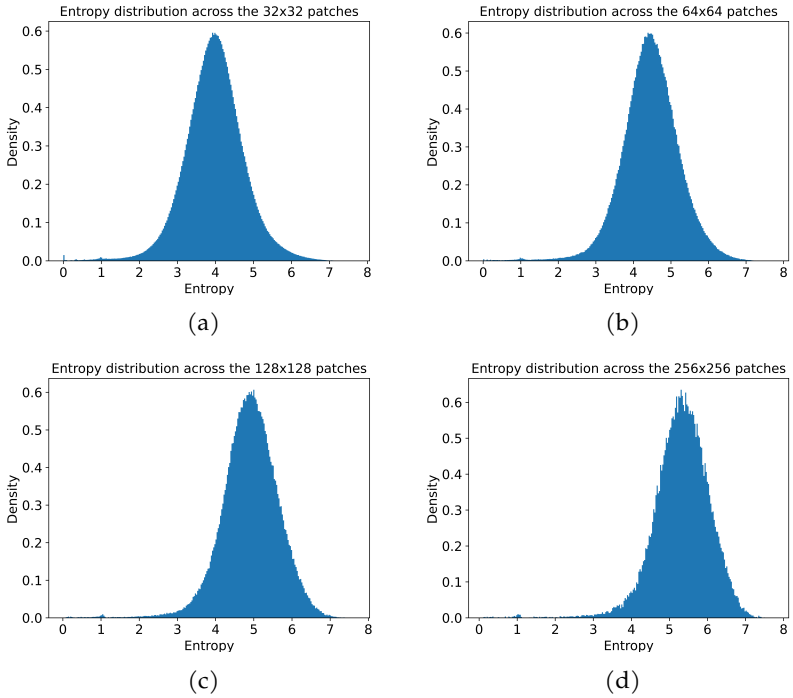


Figure 7.4: Distribution of patch entropy. (a)-(d) are taken for square patches of size 32, 64, 128 and 256 pixels.

Table 7.3: Entropy statistics

patch size	mean	standard deviation	quantile 15	42.5	57.5	85
32	3.974	0.779	3.247	3.85	4.104	4.71
64	4.456	0.765	3.75	4.335	4.588	5.191
128	4.903	0.747	4.223	4.795	5.047	5.633
256	5.319	0.735	4.66	5.229	5.475	6.029

- one with the patches entropy above the 85-th quantile, referred to as *high*.
- the last one with patches having entropy between the 42.5-th and 57.5-th quantiles, referred to as *intermediate*. Our choice for the quantile values is motivated by having the entropy be equally distant from the other groups, and keeping the same number of samples to make time comparisons meaningful.

We further extract two datasets for each patch size:

- a *low* dataset, whose patches are all the patches that rank below 30-th quantile of entropy with respect to the other patches of the same image.
- a *high* dataset, with entropy above the 70-th quantile.

7.2.3 Mean Exhaustive Minimum Distance (MEMD) criterion

The first methods of similarity measure usually consisted in computing certain features on a given image, such as the Haralick features Haralick et al. (1973), and then comparing the features obtained for different images. More recent techniques dealing with the structural similarity in textures have been proposed in Zujovic et al. (2009) and Qin and Yang (2004). Handling color or hyperspectral images is often done using histograms Yuan et al. (2015), but histograms require a large amount of data to get good estimates of the spectral distribution. A new criterion to evaluate the similarity of two images was proposed in Havlíček and Haindl (2019). This approach does not require histograms and generalizes to any number of channels.

Following the notation from Havlíček and Haindl (2021), let A and B be two images, which can have multiple channels. Let $M = \min(\#A, \#B)$, with $\#A$ and $\#B$ the number of pixels in A and B. Let $\langle A \rangle$ be the set of pairs of coordinates for the pixels of A, and U the unprocessed pairs of coordinates of pixels of B. Let ρ be the distance induced by a vector metric. $A_{i,j}$ denotes the pixel of A at coordinates (i, j). Similarly, $B_{k,l}$ is the pixel of B at coordinates (k, l). The MEMD criterion ζ is defined by Equation 2.

When writing $A_{i,j}$, the other dimensions representing the channels are implied.

$$\zeta(A,B) = \frac{1}{M} \sum_{(i,j)\in\langle A\rangle} \min_{(k,l)\in U} \{\rho(A_{i,j}, B_{k,l})\} \tag{7.2}$$

The lower the score is, the more similar images A and B are. Inversely, the higher the score, the higher the difference between the two images. The score can take values between 0 and 255. A score of 0 happens when we compare one image to itself; a score of 255 happens when we compare a white image with a black one.

To improve the computation time, Havlíček and Haindl (2021) suggested that the pixels of both the images be sorted with respect to the chosen norm. Finding the minimum distance between the pixels of the two images then comes down to choosing the closest unprocessed neighbour in the sorted array. In the special case where A and B are of the same size, we can simply match the first element of the sorted pixels of A with the first of element of the sorted pixels of B, and so on.

We compute the MEMD score of each patch with respect to all the other patches of the same image, and we average the scores. Figure 7.5 shows the distribution of the MEMD score at varying patch sizes. We observe two peaks. The first one on the left corresponds to the patches that are representative of the overall image, and the one on the right corresponds to the patches that are more unique. The reason why we only have two peaks is that the images of the lesion all share similar elements: a little bit of skin, the lesion, and some noise such as hair, a ruler, etc. The distinction between the lesion and the skin is quite drastic, meaning that few patches are going to be equally similar to skin and lesion. The variation in scores is in part due to the different number of patches per image. The more patches an image has, the less extreme the MEMD score of the patches will be. The patches with a score of 0 are from images that have only one patch. This happens for big patch sizes where the region of interest is too small to get more patches.

Similarly to what was done in Section 7.2.2, we create datasets using the same quantiles for the MEMD score.

Both the datasets created in the entropy section and in this section are taken by extracting 15% or 30% of the patches of an image. Thus, the datasets have 15% and 30% of the total number of patches in Table 7.2.

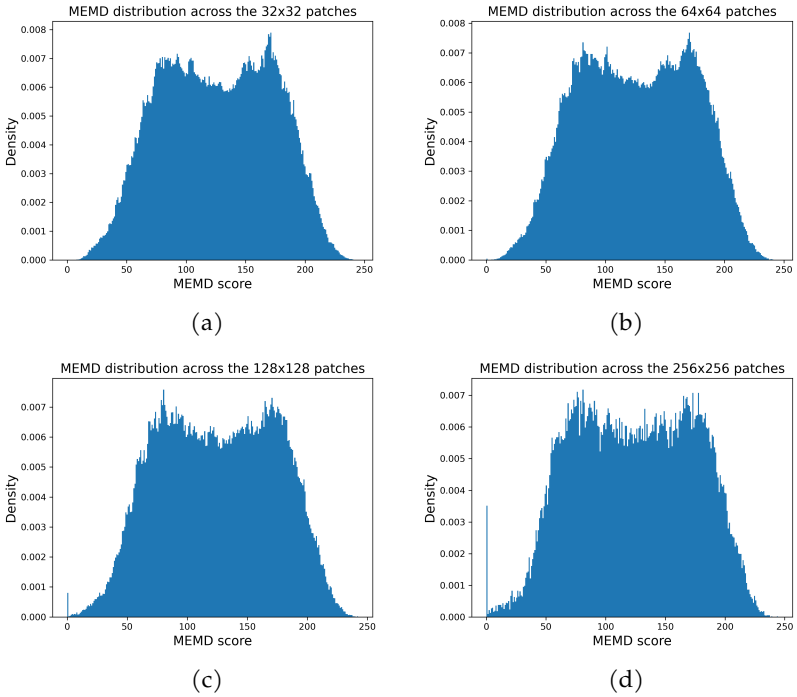


Figure 7.5: Distribution of MEMD score for patches of size (**a**) 32×32 (**b**) 64×64 (**c**) 128×128 and(**d**) 256×256

In the rest of the chapter, we use the max norm for ρ, i.e. $\rho : x \mapsto \|x\|_\infty = \max_i |x_i|$, where x_i are the coordinates of x. The distance induced by the max norm is $(x, y) \mapsto \|x - y\|_\infty$. Because the sorting of the pixels is done based on the norm of a single pixel, and the min is computed using the distance between two pixels, the optimization via sorting is not compatible with pixels with multiple channels. Indeed, let $p_1 = [135, 18, 89]$, $p_2 = [130, 16, 86]$ and $p_3 = [12, 134, 1]$. If we sort the pixels by the max norm, we get $P = [p_2, p_3, p_1]$. Selecting the closest matching pixel using the proposed method in Havlíček and Haindl (2021) would make us pair p_2 with p_3, which leads to $\zeta(p_2, p_3) = 118$. But p_2 and p_1 are clearly a better match, with $\zeta(p_2, p_1) = 5$. To alleviate the complications imposed by having multiple channels, we convert the images to grayscale before computing the MEMD score. Since grayscale images have only one channel, the optimization via sorting works.

The computation of the average MEMD of all the patches of an image has $O(m^2)$

with respect to m, the number of patches in the image. There is a trade-off between space and time complexity, where vectorizing part of the process using higher order tensors allows for faster computation but requires more space.

7.2.4 Network architecture and tuning parameters

Following Yilmaz and Trocan (2020) and Favole et al. (2020) who compared classifiers for the same task and dataset, we use a ResNet50 for the classification. ResNet50 He et al. (2016), is a 50-layer convolutional neural network, which contains *residual units* between convolutional blocks (stacks of convolutional layers) with identity mappings interspersed, to help propagate the gradient and mitigate the problem of vanishing and exploding gradients Glorot and Bengio (2010).

Though ResNets can be arbitrary deep, provided we have the computing resources to train the model, e.g. using 101 or 152 layers, we followed Yilmaz and Trocan (2020) and used the 50-layer version. Since we are interested in binary classification, e.g. whether the lesion is benign or malignant, we remove the last layer of the network, designed for multiclass classification, and replace it with a max pooling layer followed by a Dense layer with a *sigmoid* activation.

The optimizer used for the model is the Adam optimizer Kingma and Ba (2015) with a learning rate of 0.001. We use a *binary cross-entropy loss* for the training.

The model is trained for 10 epochs, with early stopping if the validation loss stops decreasing after 3 consecutive epochs.

Each dataset is split in the following way for training: 90% for training, of which 20% goes to validation, and 10% for testing.

7.3 Results

7.3.1 Training time

All the experiments were performed on a device with a 3.60 GHz Intel CPU, 32Gb of RAM and an NVidia Titan XP, running on Ubuntu. The code was written in Python and Tensorflow. The computation of the entropy was done using Pillow.

To account for the fact that a neural network may take more time to converge based on the random initialization of the parameters, we train 10 instances of a ResNet50 on each dataset. We display the 30-th quantile, the median and the 70-th quantile of the training time of the instances in Tables 7.4 and 7.5.

Table 7.4: Quantiles of training time for datasets of different entropy and patch size

		Quantile of training time (in seconds)		
patch size	entropy	30	50 (median)	70
32	high	1350.7	2013.2	2781.4
32	low	1534.9	2906.7	3078.5
64	high	291.0	382.9	441.9
64	low	290.6	338.3	414.2
128	high	155.0	204.6	220.0
128	low	204.8	255.0	255.4
256	high	142.4	152.2	189.7
256	low	189.6	226.4	226.5

Regarding the entropy datasets, we observe a tendency of faster convergence for datasets with higher entropy compared to datasets with lower entropy. Lower entropy means that the distribution of pixel intensity concentrates on fewer pixels than it does for higher entropy. This concentration makes for smoother textures, which might be harder for the classifier to learn. Higher entropy datasets have more salient features that more discernible and thus more easily learnable by the network.

As for the MEMD datasets, the dataset composed of patches with higher score tends to converge faster than the dataset with lower score. This might be ex-

Table 7.5: Quantiles of training time for datasets of varying MEMD score and patch size

		Quantile of training time (in seconds)		
patch_size	memd_score	30	50 (median)	70
32	high	3150.4	3254.8	3258.9
	low	3256.4	3260.0	3260.9
64	high	465.3	495.1	527.0
	low	564.4	691.9	986.3
128	high	241.5	281.9	387.9
	low	256.6	357.7	373.1
256	high	189.7	245.2	264.4
	low	215.4	226.7	275.2

plainable by the fact that a low MEMD score means a high similarity of the patch with the rest of the image, while a high score indicates a distinctive spectral texture compared with the other patches of the same image. Thus, the higher score patches capture the more unique features of the lesion, while the lower score patches are more representative of the overall texture of the lesion. The high representativeness of a patch might extend to patches of low score from another image, while the unique features are probably different between images. Therefore, the dataset with high score is richer in more unique patches, which provide more information than the similar patches contained in the lower score dataset. This, in turn, makes the network training converge faster for the dataset with higher score patches.

We see that the dataset with the highest entropy tends to be the fastest to converge. Since a higher entropy usually indicates that more information is present in the patch, we could expect the neural network to take longer to train. Conversely, a dataset with lower entropy would train faster because the patches would have less discriminating features, and the network would quickly classify them.

A possible explanation for this discrepancy is that patches with higher entropy might share a similar structure or have patterns not present for other patches, and thus are more recognizable by the network, while patches with lower entropy might have less salient features, which makes it harder for the classifier to classify them.

Concerning the training for the dataset with intermediate entropy, it seems to take longer to converge for smaller patch sizes compared to training on datasets with more extreme entropy, but reaches similar speeds in comparison with the other datasets when we increase the patch size. A reason for this could be that, for lower patch sizes, patches with average entropy might be more diverse than patches with lower or higher entropy, and the network will require more time to analyze the patterns. When the patches are bigger, a patch can be composed of smaller zones which vary greatly in entropy, but have an average entropy when we look at the entirety of the patch. Therefore, these patches would be easier to classify, which would lead to a faster training time.

Additionally, we investigate combining predictions from several patches of an image to classify the image. We train a Resnet for 10 epochs and choose the weights that result in the best validation loss. To classify an image from the test set, we individually classify its patches and aggregate the results. Let $\mathcal{P}_i$ be the set of patches from an image I_i, $|\mathcal{P}_i|$ the number of patches selected from the image, f be the classifier that maps a patch to 0 for a benign patch and 1 for a malignant one. The prediction $\hat{y}$ is given by the Equation 3.

$$\hat{y}_i = \begin{cases} 0 & \text{if } \left(\frac{1}{|\mathcal{P}|}\sum_{p\in\mathcal{P}_i} f(p)\right) < 0.5 \\ 1 & \text{otherwise} \end{cases} \tag{7.3}$$

7.3.2 Accuracy

These interpretations are borne out by the results of the experiments presented in Table 7.6. We observe that the classification accuracy is higher for datasets with high entropy or high MEMD than for datasets with low entropy or low MEMD. For the 128×128 patches, the accuracy does not improve when we select more patches: it stagnates around 50%. This indicates that this patch size is too small to properly discriminate the lesions. The problem is not about the number of patches but about the fact that small patches do not contain enough information to determine the status of the lesion. We believe that this situation holds also for even smaller patches, e.g. 32×32 or 64×64 patches. Conversely, for the case of 256×256 patches, we remark that using too few patches results in very low accuracy (around 25%); however, the accuracy increases considerably when we select more patches (30% of 15%), achieving 71% accuracy for patches

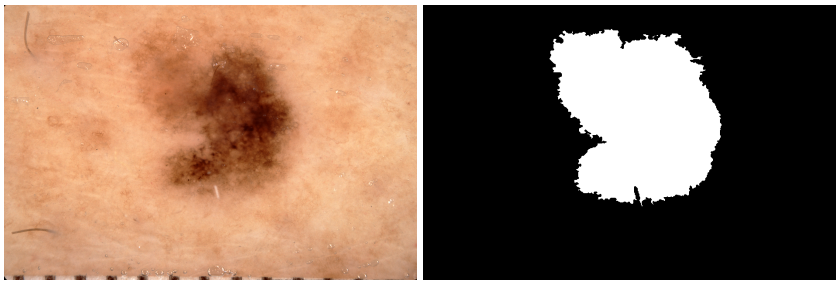
of high entropy. This accuracy is similar to the 74% accuracy obtained by the authors of Favole et al. (2020) when training on the whole region of interest with a ResNet50.

Table 7.6: Test accuracy (in percentage) for the different datasets. For a given patch size, the test images are the same for each method.

Dataset	low MEMD	high MEMD	low entropy	high entropy
128×128, 15% patches	46.7	50.5	46.2	52.7
128×128, 30% patches	43.9	51.6	39.6	52.7
256×256, 15% patches	27.2	26.3	25.1	32.0
256×256, 30% patches	45.5	57.2	52.7	**71.0**

The lower accuracy for the low MEMD and low entropy datasets, compared with the high MEMD and entropy datasets, suggests that it is not sufficient to select more patches to reach a higher level of accuracy; it is also important to select appropriate patches.

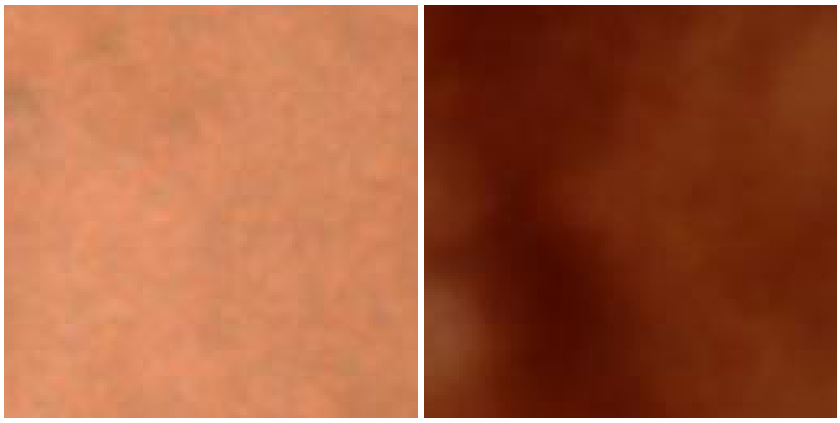
Figure 7.6 and Figure 7.7 illustrate the role of MEMD in patch selection. The patch on the left of Figure 7.7 is one of the patches with the lowest MEMD score for the image, while the patch on the right has one of the highest scores. Due to the fact that the masks cannot perfectly capture the lesion, there will always be some part of the skin that will be present in the mask. Since the skin has more uniform texture than the lesion, it is likely that patches of skin will have the lowest score.



(a) Malignant lesion (b) Mask of the lesion

Figure 7.6: Image of a mask and its lesion.

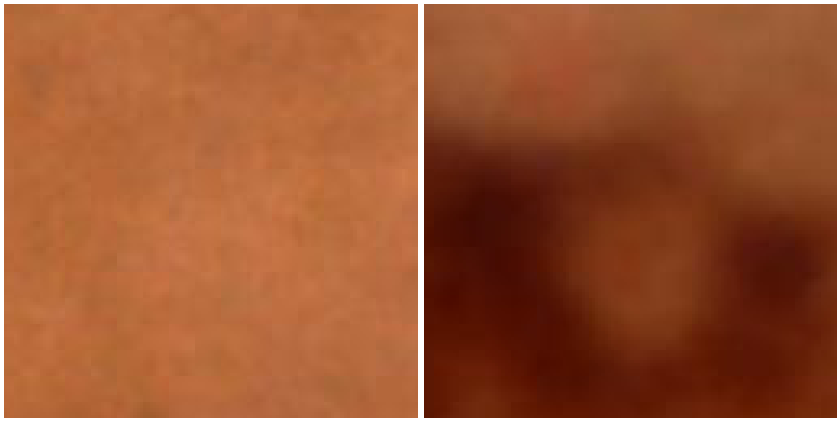
Similarly, Figure 7.8 shows, for the same image, patches that belong in the low entropy and the high entropy dataset. The patch on the left has a smoother



(a) Patch with low MEMD score (b) Patch with high MEMD score

Figure 7.7: Patches of low and high MEMD scores.

texture and the pixels have similar intensity. The patch on the right does not have the same regularity.



(a) Patch with low entropy score (b) Patch with high entropy score

Figure 7.8: Patches of low and high entropy

The datasets extracted using the entropy converge faster than the datasets extracted with the MEMD criterion. We hypothesize that a likely explanation is that patches extracted with the entropy share similar distributions of pixels, albeit sometimes shifted. The entropy quantifies the distribution of pixel intensity: the higher the entropy, the closer the pixel distribution will be to the uniform distribution. Thus, the patches from entropy extracted datasets are

similar across the images, and this similarity is learnable by the network. On the other hand, datasets extracted using the MEMD criterion do not provide any quantifiable information about the pixel distribution. Their score is only indicative of how representative the patch is with respect to the image. The network might thus be confronted with a wider variety of patches which leads to a longer training time.

7.4 Discussion

We examined the role of entropy and the MEMD criterion on the training time of a CNN for patch-based binary classification. The preprocessing is longer with the MEMD criterion because we have to compare patches two by two, whereas entropy requires a single computation per patch. We found that higher entropy leads to faster convergence than lower entropy; similarly, a higher MEMD score, which indicates that the patch does not resemble other patches from the same image, also leads to faster convergence. In terms of accuracy, the models trained on the higher entropy dataset or the higher MEMD are more performant than the models trained on the lower entropy or lower MEMD datasets. We also found that creating patch datasets using an absolute measure of information, such as entropy, makes the network train faster than when the datasets were created using a similarity measure. We also observed that patch size plays a significant role in the classifier accuracy, with small patches leading to poor results, regardless of the percentage of patches used.

Some perspectives to this work can be to explore the use of segmentation to obtain the regions of interest, increasing the number of images we can work with, and see if the results are comparable. Another possibility can be to analyze the effects resizing an image has on its entropy to quantify the loss of information, and the impact it can have on classification using resized images.

Additionally, these patches can be used as nodes in a graph, where the links can reflect the distance between the patches. A perspective would be to convert the patches into a graph and perform classsifiation using graph neural networks.

Chapter 8

Conclusion and perspectives

8.1 Conclusions

In this thesis, we proposed a grouping of the theoretically most expressive GNNs, and we empirically tackled the issues of multilabel classification, transductive and inductive learning, and learning with noise in the features. We also experimented with patch creation in medical images.

8.1.1 Contributions

In Chapter 3, we reviewed the most expressive GNNs. We find that they could be divided into different groups representing the techniques used for increasing their expressiveness compared to standard GNNs. Methods that rely on high-order tensors, e.g., tensors that represent hyper-edges or permutations, are the most expressive methods, but they are also the most expensive computationally. Methods relying on node identification and substructure awareness can reach similar levels of expressiveness while being computationally more efficient.

In Chapter 5, we showed the importance of choosing between transductive and inductive learning, and between single and multilabel classification. On the one hand, in the transductive setting, the leak of information from the test set to the

train set is detrimental to the quality of the evaluation of the models; besides, it can hide overfitting. On the other hand, many node classification tasks should be framed as multilabel tasks, in order to take into account the relations that might exist between the labels.

In Chapter 6, we investigated the performance of GNNs when we added noise to the features of the nodes. We empirically exhibited implicit noise reduction mechanisms that exist in some architectures. Furthermore, the noise that impacts the models the most is the noise that targets the nodes with the fewest neighbours. Regarding missing features, we proposed a novel GNN architecture that imputes the missing values.

In Chapter 7, we used two information measures, entropy and a spectral texture criterion, to extract patches out of images of skin lesions and performed classification using CNNs. We find that the choice of patches greatly influences the accuracy of the model.

8.2 Short Term Perspectives

We plan to continue the work related to the conversion of medical images into graphs, and the study of noise in dynamic networks.

Medical imaging and graphs: We empirically showed that we can extract patches from medical images using measures such as entropy or other criteria to perform classification. There are limitations to this approach. For instance, the size of the patch was fixed, while the images were of varying size. Moreover, there was no additional processing of the patches: the features used by the CNN were the pixels of the patch. We want to extend this approach in three ways:

- use a more suitable way of extracting regions of interest, e.g., segmentation using deep learning.
- design better features for the extracted regions, e.g., find low dimensional embeddings that capture the properties of the region, such as its shape, its texture, its volume, etc.
- create a graph where the nodes can be the extracted regions; the edges can

represent similarities between the nodes, such as their distance.

Then, a GNN can be trained on the generated graph. The goal would be then to expand the approach for node tracking in 3 dimensional images. For instance, 3D medical images can be viewed as stacks of 2D cuts. We can generate *inter-layer* edges which link nodes from different cuts. We can then train a GNN to perform node tracking, similar to particle tracking done with the Large Hadron Collider (LHC).

Dynamic graphs: In our experiments, we treated the academic citation network as a static graph; the use of graph neural networks on temporal or dynamic graphs is still rare. One of our goals will be to contribute to the main libraries for deep learning with graphs, e.g., PyTorch Geometric, to provide easier access to dynamic graphs and GNNs on dynamic graphs.

We then plan to extend our work from Chapter 6 to dynamic graphs. In particular, we would like to study the temporal impact of noise, e.g., find how a GNN reacts against strong ponctual noise or againt low intensity noise that spans across time. Furthermore, we would like to propose a GNN architecture for missing data imputation in dynamic graphs.

8.3 Long Term Perspectives

One of the recurring themes of this thesis is the importance of the features of nodes and of edges in graphs. Much of the research regarding the theoretical properties of graph neural networks has been done with the assumption that the nodes have no features, or only discrete features. The role of features in the learning of GNNs on different types of graphs, e.g., social networks, road networks, molecule graphs, is still largely unexplored. Features are seen as a means to an end; they are used insomuch as they improve the performance of a model. To make an analogy with images, it is as if the quality of the image is irrelevant, so long as we can classify the image properly. Designing methods to estimate the quality of the features would help curate graph datasets, which in turn would facilitate research on graphs.

One of the most important obstacles faced during the thesis was the issue of

scalability. This can manifest itself in two ways: either the graphs are too large to fit into the GPU memory, or the GNNs have too many parameters to fit into the memory. Expecting to solve the problem by increasing the GPU or TPU memory is pointless, because social graphs will increase likewise. While we used node sampling to address this issue in our architecture for missing data imputation, there are still other approaches to explore.

As mentioned in the short term perspectives, dynamic graphs, and especially deep learning on dynamic graphs, is still a vastly underexplored topic. One of the main reasons behind this is the lack of appropriate libraries. As the software matures, the tools to process dynamic graphs will improve, and there will be interesting problems to tackle. One particular topic, beyond the study of noise, is the explainability of dynamic graph neural networks. This will be all the more important as there will be three competing sources of information used by GNNs: temporal information, semantic information, and structural information. Furthermore, the study of dynamic graphs can provide insights into the emergence of distribution shifts in the data. For instance, in an academic citation network, the dominant topics change over the years; similarly, there are trends in social media which evolve over time.

Taking a step back, graph neural networks are just one among many approaches that can be used to solve graph related tasks. In the greater scheme of things, GNNs can be viewed from multiple lenses. For instance, graph filters are the fundamental tools behind graph signal processing Ortega et al. (2018). In addition, graph filters, and GNNs in general, are designed to be permutation invariant or equivariant. Preserving this symmetry is what defines GNNs in the same way that being translation invariant is at the core of CNNs. This idea of symmetry is at the core of geometric deep learning, where the purpose is to design neural networks that respect the symmetries of a given structure Bronstein et al. (2021). Exploring and bridging the different fields of studies related to graphs will help foster new ideas that will constitute the core of tomorrow's architectures.

One last axis of future research is to combine the aforementioned perspectives and apply them to medical data. This data takes several shapes, such as Electronic Health Records (EHRs), CT scans, MRI, fitness activity, etc. The connection and the curation of the components are still in their beginnings. There are privacy, political, economical and societal issues that need to be addressed

in order to move forward. One of the technical challenges is the creation of appropriate embeddings for all the types of data, so that they can used conjointly. Recent advances in Large Language Models and generative models look promising. Constructing a complex graph out of this data could help exhibit previously hidden patterns in the data.

Bibliography

GRAD/Train Your Own GNN Teacher: Graph-Aware Distillation on Textual Graphs.pdf at main · cmavro/GRAD. https://github.com/cmavro/GRAD/.

Ralph Abboud, İsmail İlkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 2112–2118. ijcai.org, 2021. doi: 10.24963/ijcai.2021/291.

Syed Muhammad Anwar, Muhammad Majid, Adnan Qayyum, Muhammad Awais, Majdi Alnowami, and Muhammad Khurram Khan. Medical Image Analysis using Convolutional Neural Networks: A Review. *Journal of Medical Systems*, 42(11):226, November 2018. ISSN 0148-5598, 1573-689X. doi: 10.1007/s10916-018-1088-1.

Waiss Azizian and Marc Lelarge. Expressive power of invariant and equivariant graph neural networks. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Muhammet Balcilar, Guillaume Renton, Pierre Héroux, Benoit Gaüzère, Sébastien Adam, and Paul Honeine. Analyzing the expressive power of graph neural networks in a spectral perspective. In *9th International Conference*

on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net, 2021.

Albert-László Barabási and Réka Albert. Emergence of Scaling in Random Networks. *Science (New York, N.Y.)*, 286(5439):509–512, October 1999. ISSN 0036-8075, 1095-9203. doi: 10.1126/science.286.5439.509.

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv:1806.01261 [cs, stat]*, October 2018.

Beatrice Bevilacqua, Fabrizio Frasca, Derek Lim, Balasubramaniam Srinivasan, Chen Cai, Gopinath Balamurugan, Michael M. Bronstein, and Haggai Maron. Equivariant subgraph aggregation networks. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

Cristian Bodnar, Fabrizio Frasca, Nina Otter, Yuguang Wang, Pietro Liò, Guido F. Montúfar, and Michael M. Bronstein. Weisfeiler and lehman go cellular: CW networks. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, Virtual*, pages 2625–2640, 2021.

John Adrian Bondy and Uppaluri Siva Ramachandra Murty. *Graph Theory*. Springer Publishing Company, Incorporated, 2008.

Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M. Bronstein. Improving Graph Neural Network Expressivity via Subgraph Isomorphism Counting, July 2021.

Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges. *arXiv:2104.13478 [cs, stat]*, May 2021.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, December 1992. ISSN 1439-6912. doi: 10.1007/BF01305232.

Emmanuel J. Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational Mathematics*, 9(6):717–772, 2009. doi: 10.1007/s10208-009-9045-5.

Lei Chen, Zhengdao Chen, and Joan Bruna. On graph neural networks versus graph-augmented MLPs. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. Can graph neural networks count substructures? In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, Virtual*, 2020.

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 257–266, July 2019. doi: 10.1145/3292500.3330925.

Eli Chien, Wei-Cheng Chang, Cho-Jui Hsieh, Hsiang-Fu Yu, Jiong Zhang, Olgica Milenkovic, and Inderjit S. Dhillon. Node Feature Extraction by Self-Supervised Multi-scale Neighborhood Prediction. *arXiv:2111.00064 [cs]*, October 2021.

Eli Chien, Wei-Cheng Chang, Cho-Jui Hsieh, Hsiang-Fu Yu, Jiong Zhang, Olgica Milenkovic, and Inderjit S. Dhillon. Node feature extraction by self-supervised multi-scale neighborhood prediction. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

Enyan Dai, Charu Aggarwal, and Suhang Wang. NRGNN: Learning a label noise resistant graph neural network on sparsely and noisily labeled graphs. In Feida Zhu, Beng Chin Ooi, and Chunyan Miao, editors, *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, pages 227–236. ACM, 2021. doi: 10.1145/3447548.3467364.

George Dasoulas, Ludovic Dos Santos, Kevin Scaman, and Aladin Virmaux. Coloring graph neural networks for node disambiguation. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 2126–2132. ijcai.org, 2020. doi: 10.24963/ijcai.2020/294.

Ithiel de Sola Pool and Manfred Kochen. Contacts and influence. *Social Networks*, 1(1):5–51, January 1978. ISSN 0378-8733. doi: 10.1016/0378-8733(78)90011-4.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 3844–3852, Red Hook, NY, USA, December 2016. Curran Associates Inc. ISBN 978-1-5108-3881-9.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data Via the EM Algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977. ISSN 2517-6161. doi: 10.1111/j.2517-6161.1977.tb01600.x.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-Training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019. doi: 10.18653/v1/n19-1423.

Reinhard Diestel. Graph Theory, volume 173 of. *Graduate texts in mathematics*, 593, 2012.

Samuel F. Dodge and Lina J. Karam. Understanding how image quality affects deep neural networks. In *Eighth International Conference on Quality of Multimedia*

Experience, QoMEX 2016, Lisbon, Portugal, June 6-8, 2016, pages 1–6. IEEE, 2016. doi: 10.1109/QoMEX.2016.7498955.

Yuxiao Dong, Hao Ma, Zhihong Shen, and Kuansan Wang. A century of science: Globalization of scientific collaborations, citations, and innovations. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*, pages 1437–1446. ACM, 2017. doi: 10.1145/3097983.3098016.

B. L. Douglas. The Weisfeiler-Lehman Method and Graph Isomorphism Testing, January 2011.

Keyu Duan, Zirui Liu, Peihao Wang, Wenqing Zheng, Kaixiong Zhou, Tianlong Chen, Xia Hu, and Zhangyang Wang. A Comprehensive Study on Large-Scale Graph Training: Benchmarking and Rethinking, October 2022.

Andre Esteva, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115–118, February 2017. ISSN 0028-0836, 1476-4687. doi: 10.1038/nature21056.

Florent Favole, Maria Trocan, and Ercüment Yilmaz. Melanoma Detection Using Deep Learning. In Ngoc Thanh Nguyen, Bao Hung Hoang, Cong Phap Huynh, Dosam Hwang, Bogdan Trawiński, and Gottfried Vossen, editors, *Computational Collective Intelligence*, volume 12496, pages 816–824. Springer International Publishing, Cham, 2020. ISBN 978-3-030-63006-5 978-3-030-63007-2. doi: 10.1007/978-3-030-63007-2_64.

Matthias Fey and Jan Eric Lenssen. Fast Graph Representation Learning with PyTorch Geometric. *arXiv:1903.02428 [cs, stat]*, April 2019.

Simon Geisler, Daniel Zügner, and Stephan Günnemann. Reliable graph neural networks via robust aggregation. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, Virtual*, 2020.

Simon Geisler, Tobias Schmidt, Hakan Sirin, Daniel Zügner, Aleksandar Bojchevski, and Stephan Günnemann. Robustness of graph neural networks at scale. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy

Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, Virtual*, pages 7637–7649, 2021.

Nils Gessert, Maximilian Nielsen, Mohsin Shaikh, René Werner, and Alexander Schlaefer. Skin lesion classification using ensembles of multi-resolution EfficientNets with meta data. *MethodsX*, 7:100864, 2020a. ISSN 22150161. doi: 10.1016/j.mex.2020.100864.

Nils Gessert, Thilo Sentker, Frederic Madesta, Rudiger Schmitz, Helge Kniep, Ivo Baltruschat, Rene Werner, and Alexander Schlaefer. Skin Lesion Classification Using CNNs With Patch-Based Attention and Diagnosis-Guided Loss Weighting. *IEEE Transactions on Biomedical Engineering*, 67(2):495–503, February 2020b. ISSN 0018-9294, 1558-2531. doi: 10.1109/TBME.2019.2915839.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR, 2017.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and D. Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, volume 9 of *JMLR Proceedings*, pages 249–256. JMLR.org, 2010.

John W Graham. Missing data analysis: Making it work in the real world. *Annual review of psychology*, 60:549–576, 2009.

David Gutman, Noel C. F. Codella, Emre Celebi, Brian Helba, Michael Marchetti, Nabin Mishra, and Allan Halpern. Skin Lesion Analysis toward Melanoma Detection: A Challenge at the International Symposium on Biomedical Imaging (ISBI) 2016, hosted by the International Skin Imaging Collaboration (ISIC). *arXiv:1605.01397 [cs]*, May 2016.

William L. Hamilton. Graph Representation Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14(3):1–159.

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*, NIPS'17, pages 1025–1035, Red Hook, NY, USA, December 2017. Curran Associates Inc. ISBN 978-1-5108-6096-4.

Robert M. Haralick, K. Shanmugam, and Its'Hak Dinstein. Textural Features for Image Classification. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-3(6):610–621, November 1973. ISSN 0018-9472, 2168-2909. doi: 10.1109/TSMC.1973.4309314.

Michal Havlíček and Michal Haindl. Texture spectral similarity criteria. *IET Image Processing*, 13(11):1998–2007, 2019. ISSN 1751-9667. doi: 10.1049/iet-ipr.2019.0250.

Michal Havlíček and Michal Haindl. Optimized Texture Spectral Similarity Criteria. In Krystian Wojtkiewicz, Jan Treur, Elias Pimenidis, and Marcin Maleszka, editors, *Advances in Computational Collective Intelligence*, volume 1463, pages 644–655. Springer International Publishing, Cham, 2021. ISBN 978-3-030-88112-2 978-3-030-88113-9. doi: 10.1007/978-3-030-88113-9_52.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, Las Vegas, NV, USA, 2016. IEEE. ISBN 978-1-4673-8851-1. doi: 10.1109/CVPR.2016.90.

Xiaoxin He, Xavier Bresson, Thomas Laurent, and Bryan Hooi. Explanations as Features: LLM-Based Features for Text-Attributed Graphs, May 2023.

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

L. Hou, D. Samaras, T. M. Kurc, Y. Gao, J. E. Davis, and J. H. Saltz. Patch-based convolutional neural network for whole slide tissue image classification. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2424–2433, 2016. doi: 10.1109/CVPR.2016.266.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural*

Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, Virtual, 2020.

Vassilis Kalofolias, Xavier Bresson, Michael M. Bronstein, and Pierre Vandergheynst. Matrix completion on graphs. *CoRR*, abs/1408.1717, 2014.

George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998. doi: 10.1137/S1064827595287997.

Nicolas Keriven and Gabriel Peyré. Universal invariant and equivariant graph neural networks. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 7090–7099, 2019.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6): 84–90, May 2017. ISSN 0001-0782, 1557-7317. doi: 10.1145/3065386.

Guillaume Lachaud, Patricia Conde-Cespedes, and Maria Trocan. Entropy role on patch-based binary classification for skin melanoma. In *International Conference on Computational Collective Intelligence*, pages 324–333. Springer, 2021.

Guillaume Lachaud, Patricia Conde Céspedes, and Maria Trocan. Graph neural networks-based multilabel classification of citation network. In Ngoc Thanh Nguyen, Tien Khoa Tran, Ualsher Tukeyev, Tzung-Pei Hong, Bogdan Trawinski, and Edward Szczerbicki, editors, *Intelligent Information and Database Systems - 14th Asian Conference, ACIIDS 2022, Ho Chi Minh City, Vietnam, November 28-30, 2022, Proceedings, Part II*, volume 13758 of *Lecture Notes in Computer Science*, pages 128–140. Springer, 2022a. doi: 10.1007/978-3-031-21967-2_11.

Guillaume Lachaud, Patricia Conde Céspedes, and Maria Trocan. Comparison between inductive and transductive learning in a real citation network using graph neural networks. In *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2022, Istanbul, Turkey, November 10-13, 2022*, pages 534–540. IEEE, 2022b. doi: 10.1109/ASONAM55673.2022.10068589.

Guillaume Lachaud, Patricia Conde-Cespedes, and Maria Trocan. Mathematical expressiveness of graph neural networks. *Mathematics*, 10(24):4770, 2022c.

Guillaume Lachaud, Patricia Conde Conde-Cespedes, and Maria Trocan. Patch selection for melanoma classification. In Ngoc Thanh Nguyen, Yannis Manolopoulos, Richard Chbeir, Adrianna Kozierkiewicz, and Bogdan Trawinski, editors, *Computational Collective Intelligence - 14th International Conference, ICCCI 2022, Hammamet, Tunisia, September 28-30, 2022, Proceedings*, volume 13501 of *Lecture Notes in Computer Science*, pages 148–159. Springer, 2022d. doi: 10.1007/978-3-031-16014-1_13.

Guillaume Lachaud, Patricia Conde-Cespedes, and Maria Trocan. Scalable Missing Data Imputation with Graph Neural Networks. In *IWCIM 2023, ICASSP 2023 Satelite Workshop*, 2023.

Jack Lanchantin, Arshdeep Sekhon, and Yanjun Qi. Neural message passing for multi-label classification. In Ulf Brefeld, Élisa Fromont, Andreas Hotho, Arno J. Knobbe, Marloes H. Maathuis, and Céline Robardet, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2019, Würzburg, Germany, September 16-20, 2019, Proceedings, Part II*, volume 11907 of *Lecture Notes in Computer Science*, pages 138–163. Springer, 2019. doi: 10.1007/978-3-030-46147-8_9.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998. ISSN 1558-2256. doi: 10.1109/5.726791.

Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. DeepGCNs: Can GCNs Go As Deep As CNNs? In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9266–9275, Seoul, Korea (South), October 2019. IEEE. ISBN 978-1-72814-803-8. doi: 10.1109/ICCV.2019.00936.

Guohao Li, Matthias Müller, Bernard Ghanem, and Vladlen Koltun. Training graph neural networks with 1000 layers. In Marina Meila and Tong Zhang,

editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 6437–6449. PMLR, 2021a.

Jingling Li, Mozhi Zhang, Keyulu Xu, John Dickerson, and Jimmy Ba. How does a neural network's architecture impact its robustness to noisy labels? In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, Virtual*, pages 9788–9803, 2021b.

Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. Distance encoding: Design provably more powerful neural networks for graph representation learning. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, Virtual*, 2020.

Jenny Liu, Aviral Kumar, Jimmy Ba, Jamie Kiros, and Kevin Swersky. Graph normalizing flows. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 13556–13566, 2019.

Songtao Liu, Zhitao Ying, Hanze Dong, Lu Lin, Jinghui Chen, and Dinghao Wu. How Powerful is Implicit Denoising in Graph Neural Networks. In *NeurIPS 2022 Workshop: New Frontiers in Graph Learning*, November 2022.

Andreas Loukas. What graph neural networks cannot learn: Depth vs width. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Yao Ma and Jiliang Tang. *Deep Learning on Graphs*. Cambridge University Press, 2021.

Yao Ma, Xiaorui Liu, Tong Zhao, Yozen Liu, Jiliang Tang, and Neil Shah. A unified view on graph neural networks as graph signal denoising. In Gianluca Demartini, Guido Zuccon, J. Shane Culpepper, Zi Huang, and Hanghang Tong, editors, *CIKM '21: The 30th ACM International Conference on Information*

and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021, pages 1202–1211. ACM, 2021. doi: 10.1145/3459637.3482225.

Panagiotis Marentakis, Pantelis Karaiskos, Vassilis Kouloulias, Nikolaos Kelekis, Stylianos Argentos, Nikolaos Oikonomopoulos, and Constantinos Loukas. Lung cancer histology classification from CT images based on radiomics and deep learning models. *Medical & Biological Engineering & Computing*, 59(1):215–226, January 2021. ISSN 0140-0118, 1741-0444. doi: 10.1007/s11517-020-02302-w.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 2153–2164, 2019a.

Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019b.

Haggai Maron, Ethan Fetaya, Nimrod Segol, and Yaron Lipman. On the universality of invariant networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 4363–4371. PMLR, 2019c.

Carl D Meyer and Ian Stewart. *Matrix Analysis and Applied Linear Algebra*. SIAM, 2023.

Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Yoshua Bengio and Yann LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.

Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:4602–4609, July 2019. ISSN 2374-3468, 2159-5399. doi: 10.1609/aaai.v33i01.33014602.

Christopher Morris, Yaron Lipman, Haggai Maron, Bastian Rieck, Nils M. Kriege, Martin Grohe, Matthias Fey, and Karsten Borgwardt. Weisfeiler and Leman go Machine Learning: The Story so far, December 2021.

Ryan L. Murphy, Balasubramaniam Srinivasan, Vinayak A. Rao, and Bruno Ribeiro. Relational pooling for graph representations. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 4663–4673. PMLR, 2019.

Jinseok Nam, Jungi Kim, Eneldo Loza Mencía, Iryna Gurevych, and Johannes Fürnkranz. Large-Scale Multi-label Text Classification — Revisiting Neural Networks. In Toon Calders, Floriana Esposito, Eyke Hüllermeier, and Rosa Meo, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 8725, pages 437–452. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. ISBN 978-3-662-44850-2 978-3-662-44851-9. doi: 10.1007/978-3-662-44851-9_28.

Kamal Nigam, John Lafferty, and Andrew McCallum. Using maximum entropy for text classification. In *IJCAI-99 Workshop on Machine Learning for Information Filtering*, volume 1, pages 61–67. Stockholom, Sweden, 1999.

Antonio Ortega, Pascal Frossard, Jelena Kovačević, José M. F. Moura, and Pierre Vandergheynst. Graph Signal Processing: Overview, Challenges and Applications, March 2018.

Xuejie Qin and Yee-Hong Yang. Similarity measure and learning with gray level aura matrices (GLAM) for texture image retrieval. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 1, pages I–I, June 2004. doi: 10.1109/CVPR.2004.1315050.

Samuel Rey, Santiago Segarra, Reinhard Heckel, and Antonio G. Marques. Untrained graph neural networks for denoising. *IEEE Trans. Signal Process.*, 70: 5708–5723, 2022. doi: 10.1109/TSP.2022.3223552.

Hannah Ritchie and Max Roser. Causes of death. *Our World in Data*, 2018.

Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal Graph Networks for Deep Learning on Dynamic Graphs. *arXiv:2006.10637 [cs, stat]*, October 2020.

Emanuele Rossi, Henry Kenlay, Maria I. Gorinova, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. On the unreasonable effectiveness of feature propagation in learning on graphs with missing node features. In Bastian Rieck and Razvan Pascanu, editors, *Learning on Graphs Conference, LoG 2022, 9-12 December 2022, Virtual Event*, volume 198 of *Proceedings of Machine Learning Research*, page 11. PMLR, 2022.

Veronica Rotemberg, Nicholas Kurtansky, Brigid Betz-Stablein, Liam Caffery, Emmanouil Chousakos, Noel Codella, Marc Combalia, Stephen Dusza, Pascale Guitera, David Gutman, Allan Halpern, Brian Helba, Harald Kittler, Kivanc Kose, Steve Langer, Konstantinos Lioprys, Josep Malvehy, Shenara Musthaq, Jabpani Nanda, Ofer Reiter, George Shih, Alexander Stratigos, Philipp Tschandl, Jochen Weber, and H. Peter Soyer. A patient-centric dataset of images and metadata for identifying melanomas using clinical context. *Scientific Data*, 8(1):34, 2021. ISSN 2052-4463. doi: 10.1038/s41597-021-00815-z.

François Rousseau, Piotr A. Habas, and Colin Studholme. A supervised patch-based approach for human brain labeling. *IEEE Trans. Medical Imaging*, 30 (10):1852–1862, 2011. doi: 10.1109/TMI.2011.2156806.

Kaushiki Roy, Debapriya Banik, Debotosh Bhattacharjee, and Mita Nasipuri. Patch-based system for Classification of Breast Histology images using deep learning. *Comput. Medical Imaging Graph.*, 71:90–103, 2019. doi: 10.1016/j.compmedimag.2018.11.003.

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random features strengthen graph neural networks. In Carlotta Demeniconi and Ian Davidson, editors, *Proceedings of the 2021 SIAM International Conference on Data Mining, SDM 2021, Virtual Event, April 29 - May 1, 2021*, pages 333–341. SIAM, 2021. doi: 10.1137/1.9781611976700.38.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective Classification in Network Data. *AI Magazine*, 29 (3):93, September 2008. ISSN 0738-4602, 0738-4602. doi: 10.1609/aimag.v29i3.2157.

Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948. doi: 10.1002/j.1538-7305.1948.tb01338.x.

Yuefan Shen, Hongbo Fu, Zhongshuo Du, Xiang Chen, Evgeny Burnaev, Denis Zorin, Kun Zhou, and Youyi Zheng. GCN-Denoiser: Mesh denoising with graph convolutional networks. *ACM Trans. Graph.*, 41(1):8:1–8:14, 2022. doi: 10.1145/3480168.

Kai Shu, Amy Sliva, Suhang Wang, Jiliang Tang, and Huan Liu. Fake news detection on social media: A data mining perspective. *SIGKDD Explor.*, 19(1): 22–36, 2017. doi: 10.1145/3137597.3137600.

Rebecca L. Siegel, Kimberly D. Miller, Hannah E. Fuchs, and Ahmedin Jemal. Cancer Statistics, 2021. *CA: A Cancer Journal for Clinicians*, 71(1):7–33, January 2021. ISSN 0007-9235, 1542-4863. doi: 10.3322/caac.21654.

John Skilling and RK Bryan. Maximum entropy image reconstruction-general algorithm. *Monthly notices of the royal astronomical society*, 211:111, 1984.

A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, May 1997. ISSN 1045-9227, 1941-0093. doi: 10.1109/72.572108.

Chuxiong Sun and Guoshi Wu. Adaptive Graph Diffusion Networks with Hop-Wise Attention. *arXiv:2012.15024* [*cs*], December 2020.

Chuxiong Sun, Hongming Gu, and Jie Hu. Scalable and Adaptive Graph Neural Networks with Self-Label-Enhanced training. *arXiv:2104.09376* [*cs*], July 2021.

Erik H. Thiede, Wenda Zhou, and Risi Kondor. Autobahn: Automorphism-based graph neural nets. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, Virtual*, pages 29922–29934, 2021.

Olga G. Troyanskaya, Michael N. Cantor, Gavin Sherlock, Patrick O. Brown, Trevor Hastie, Robert Tibshirani, David Botstein, and Russ B. Altman. Missing value estimation methods for DNA microarrays. *Bioinform.*, 17(6):520–525, 2001. doi: 10.1093/bioinformatics/17.6.520.

Philipp Tschandl, Cliff Rosendahl, and Harald Kittler. The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. *Scientific Data*, 5(1):180161, December 2018. ISSN 2052-4463. doi: 10.1038/sdata.2018.161.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In William W. Cohen, Andrew McCallum, and Sam T. Roweis, editors, *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, volume 307 of *ACM International Conference Proceeding Series*, pages 1096–1103. ACM, 2008. doi: 10.1145/1390156.1390294.

Jiang Wang, Yi Yang, Junhua Mao, Zhiheng Huang, Chang Huang, and Wei Xu. CNN-RNN: A Unified Framework for Multi-label Image Classification. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2285–2294, Las Vegas, NV, USA, June 2016. IEEE. ISBN 978-1-4673-8851-1. doi: 10.1109/CVPR.2016.251.

Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

Xiyuan Wang and Muhan Zhang. How powerful are spectral graph neural networks. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 23341–23362. PMLR, 2022.

Yangkun Wang, Jiarui Jin, Weinan Zhang, Yong Yu, Zheng Zhang, and David Wipf. Bag of Tricks for Node Classification with Graph Neural Networks. *arXiv:2103.13355 [cs]*, July 2021.

Boris Weisfeiler and Andrei Leman. The reduction of a graph to canonical form and the algebra which appears therein. *nti, Series*, 2(9):12–16, 1968.

Brian J Wells, Kevin M Chagin, Amy S Nowacki, and Michael W Kattan. Strategies for handling missing data in electronic health record derived data. *Egems*, 1(3), 2013.

Asiri Wijesinghe and Qing Wang. A new perspective on "How Graph Neural Networks Go Beyond Weisfeiler-Lehman?". In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6861–6871. PMLR, 2019.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Networks Learn. Syst.*, 32(1):4–24, 2021. doi: 10.1109/TNNLS.2020.2978386.

Da Xu, Chuanwei Ruan, Evren Körpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Fengli Xu, Quanming Yao, Pan Hui, and Yong Li. Automorphic equivalence-aware graph neural network. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, Virtual*, pages 15138–15150, 2021a.

Kaidi Xu, Hongge Chen, Sijia Liu, Pin-Yu Chen, Tsui-Wei Weng, Mingyi Hong, and Xue Lin. Topology attack and defense for graph neural networks: An optimization perspective. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 3961–3967. ijcai.org, 2019a. doi: 10.24963/ijcai.2019/550.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019b.

Keyulu Xu, Mozhi Zhang, Stefanie Jegelka, and Kenji Kawaguchi. Optimization of graph neural networks: Implicit acceleration by skip connections and more depth. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 11592–11602. PMLR, 2021b.

Adam Yala, Constance Lehman, Tal Schuster, Tally Portnoi, and Regina Barzilay. A Deep Learning Mammography-Based Model for Improved Breast Cancer Risk Prediction. *Radiology*, 292(1):60–66, July 2019. ISSN 0033-8419, 1527-1315. doi: 10.1148/radiol.2019182716.

Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 40–48. JMLR.org, 2016.

Ercument Yilmaz and Maria Trocan. Benign and Malignant Skin Lesion Classification Comparison for Three Deep-Learning Architectures. In Ngoc Thanh Nguyen, Kietikul Jearanaitanakij, Ali Selamat, Bogdan Trawiński, and Suphamit Chittayasothorn, editors, *Intelligent Information and Database Systems*, volume 12033, pages 514–524. Springer International Publishing, Cham, 2020. ISBN 978-3-030-41963-9 978-3-030-41964-6. doi: 10.1007/978-3-030-41964-6_44.

Jinsung Yoon, James Jordon, and Mihaela van der Schaar. GAIN: Missing data imputation using generative adversarial nets. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 5675–5684. PMLR, 2018.

Jiaxuan You, Rex Ying, and Jure Leskovec. Position-aware graph neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 7134–7143. PMLR, 2019.

Jiaxuan You, Xiaobai Ma, Daisy Yi Ding, Mykel J. Kochenderfer, and Jure Leskovec. Handling missing data with graph representation learning. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, Virtual*, 2020.

Jiaxuan You, Jonathan Michael Gomes Selman, Rex Ying, and Jure Leskovec. Identity-aware graph neural networks. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, the Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 10737–10745. AAAI Press, 2021.

Jiangye Yuan, Deliang Wang, and Anil M. Cheriyadat. Factorization-Based Texture Segmentation. *IEEE Transactions on Image Processing*, 24(11):3488–3497, November 2015. ISSN 1941-0042. doi: 10.1109/TIP.2015.2446948.

Fan Zhang, Yang Song, Weidong Cai, Min-Zhao Lee, Yun Zhou, Heng Huang, Shimin Shan, Michael J. Fulham, and David Dagan Feng. Lung nodule classification with multilevel patch-based context analysis. *IEEE Transactions on Biomedical Engineering*, 61(4):1155–1166, 2014. doi: 10.1109/TBME.2013.2295593.

Linfeng Zhang, Jiebo Song, Anni Gao, Jingwei Chen, Chenglong Bao, and Kaisheng Ma. Be Your Own Teacher: Improve the Performance of Convolutional Neural Networks via Self Distillation. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3712–3721, Seoul, Korea (South), October 2019. IEEE. ISBN 978-1-72814-803-8. doi: 10.1109/ICCV.2019.00381.

Muhan Zhang and Pan Li. Nested graph neural networks. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: An-*

nual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, Virtual, pages 15734–15747, 2021.

Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *IEEE Trans. Knowl. Data Eng.*, 34(1):249–270, 2022. doi: 10.1109/TKDE.2020.2981333.

Lingxiao Zhao, Wei Jin, Leman Akoglu, and Neil Shah. From stars to subgraphs: Uplifting any GNN with local structure awareness. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

Song Chun Zhu, Ying Nian Wu, and David Mumford. Minimax entropy principle and its application to texture modeling. *Neural Computation*, 9(8):1627–1660, 1997. doi: 10.1162/neco.1997.9.8.1627.

Jana Zujovic, Thrasyvoulos N. Pappas, and David L. Neuhoff. Structural similarity metrics for texture analysis and retrieval. In *2009 16th IEEE International Conference on Image Processing (ICIP)*, pages 2225–2228, Cairo, Egypt, November 2009. IEEE. ISBN 978-1-4244-5653-6. doi: 10.1109/ICIP.2009.5413897.

Résumé en Français

1 Chapitre 1: Introduction

Les graphes sont les objets les plus simples permettant de décrire les relations entre plusieurs entités. Ils sont présent tout autour de nous, du microscopique avec les atomes constitués de protons, de neutrons et d'électrons interagissant entre eux, jusqu'au macroscopique avec les planètes qui orbitent autour du Soleil au sein d'une galaxie qui fait elle-même partie d'un ensemble encore plus grand.

Au-delà des relations les physiques, les graphes permettent de représenter des concepts plus abstraits. Chaque jour notre vie baigne dans les graphes, que cela soit lors d'interactions avec d'autres personnes sur les réseaux sociaux, lorsque nous surfons sur les plateformes de contenu vidéo, lorsque nous achetons des produits sur internet, ou bien lorsque nous utilisons le GPS pour nous rendre d'un point à un autre. En science, les graphes peuvent s'utiliser pour suivre l'évolution d'une épidémie au sein d'une population ou bien à l'échelle mondiale; le code des langages de programmation peut être analysé en utilisant sa structure d'arbre.

La manipulation de ces graphes recquiert des outils adaptés et performants. Au contraire des approches traditionnelles d'apprentissage machine qui reposent sur des fondations et des modèles statistiques robustes, l'apprentissage profond (*deep learning*) est une approche orientée sur la donnée. Les premiers réseaux de neurones sont apparus il y a 70 ans, mais c'est avec l'apparition des processeurs graphiques (GPU) et l'accumulation de données sur internet que leur essor a vraiment eu lieu.

De manière similaire aux réseaux de neurones convolutionnels (CNNs) qui opèrent sur des images, et les réseaux de neurones récurrents (RNNs) qui opèrent principalement sur de la donnée textuelle, les réseaux de neurones de graphes (GNNs) manipulent des graphes. En quelques années, les GNNs se sont beaucoup développé, aussi bien avec la découverte de nouvelles architectures qu'avec l'établissement de fondements théoriques de plus en plus solides.

1.1 Les défis des réseaux de neurones de graphes

Parmi les défis que doivent relever les GNNs, nous en avons identifié trois spécifiques aux graphes qui seront traités dans la thèse : la différence entre l'apprentissage transductif et inductif, le choix de la classification multilabel, et la présence de bruit dans les attributs d'un graphe.

En tout premier lieu, l'apprentissage d'un GNN peut s'effectuer de deux manières : de manière transductive, c'est-à-dire que le GNN a accès aux données de test non labelisées pendant l'apprentissage; ou de manière inductive, où le GNN n'a accès qu'au sous-graphe formé par les nœuds de l'ensemble d'entraînement.

Une autre spécificité de l'apprentissage sur graphe est l'interdépendance entre les nœuds : ils ne peuvent pas être considérés comme indépendants et identiquement distribués. Lors de la classification, il faut prendre en compte les attributs des nœuds et son voisinage. Ces deux informations peuvent être en contradiction.

Au même titre que de bruit dans la structure ou dans les labels du graphe, la présence de bruit dans les attributs d'un nœud ou d'un lien ont des répercussions qui dépassent l'élément en question. L'interdépendance mentionnée plus haut fait que le bruit se propage dans le graphe. Le choix d'un GNN pour une tâche donnée doit donc prendre en considération la possibilité d'une présence de bruit dans la donnée.

1.2 Vers la création de graphes

Au-delà de l'apprentissage à proprement parlé sur les graphes, il y a la question de la construction des graphes. Si certains graphes émergent naturellement

comme la représentation la plus appropriée d'un phénomène, d'autres graphes peuvent être le fruit d'un long travail visant à décrire un système complexe. En ce sens, la création de graphes joue un rôle aussi important que l'apprentissage sur les graphes.

1.3 Contributions

Les contributions principales de cette thèse sont les suivantes :

- nous montrons les différences empiriques de performances de plusieurs modèles dans le cas de l'apprentissage transductif et inductif.
- nous montrons d'une part l'utilité de la classification multilabel pour l'analyse d'erreurs, et d'autre part sa pertinence pour la sélection d'architectures.
- nous faisons une étude empirique approfondie de l'influence du bruit dans les attributs sur l'apprentissage d'un GNN. En particulier, nous trouvons que les performances d'un modèle sont le plus dégradées lorsque le bruit cible les nœuds les plus isolés.
- nous proposons une nouvelle architecture de GNN qui permet l'imputation d'attributs manquants dans les données tabulaires ou les données de type graphe.
- nous présentons des résultats préliminaires sur la création de graphes à partir d'images médicales.

2 Chapitre 2: Introduction aux graphes, à l'apprentissage de représentations et au traitement du signal des graphes

Ce chapitre présente la représentation mathématique des graphes et les types de tâches qui sont effectués dessus, à savoir les tâches sur les nœuds comme la classification de nœuds, les tâches sur les liens comme la prédiction de liens, et les tâches sur les graphes.

Dans une deuxième partie, les principales méthodes d'apprentissage sur les graphes, à l'exception des réseaux de neurones de graphes, sont présentés. En outre, les méthodes les plus importantes sont:

- les méthodes statistiques, comme le calcul de statistiques élémentaires sur les graphes et les marches aléatoires.
- l'apprentissage de représentations de faible dimension, à l'aide d'encodeurs et décodeurs.
- le traitement de signal de graph (*graph signal processing*), qui traite les attributs d'un graphe comme un signal à transmettre.
- les algorithmes itératifs de coloration de graphes, avec en particulier l'algorithme de Weisfeiler-Leman.

3 Chapitre 3: Apprentissage profond et réseaux de neurones de graphes

L'apprentissage profond repose sur la création de réseaux de neurones composés de plusieurs couches contenant chacune des paramètres. Ces couches sont de deux natures : les filtres, qui visent à obtenir une représentation de plus en plus abstraite et complexe de la donnée initiale, et les couches de *pooling*, qui visent à réduire la granularité de la donnée. Les filtres sont souvent associés à des fonctions d'activation non-linéaires, telles que la sigmoïde ou la ReLU (Rectified Linear Unit).

Ces réseaux fonctionnent en deux étapes. Pour l'inférence, ou la prédiction, les entrées sont passées à travers le réseau, et la sortie représente la prédiction du réseau. Cela se nomme la *forward pass*. Pour l'apprentissage, une fonction de perte est utilisée. Son gradient est rétropropagé à travers le réseau, de sorte que les paramètres sont modifiés pour diminuer le coût du modèle par rapport à cette fonction de perte. L'obtention des dérivés des paramètres du modèle s'efffectue par une application successive du théorème de dérivation des fonctions composées. Cette étape s'appelle la *backward pass*.

Chaque type de réseau de neurones, e.g., les réseaux de neurones convolutionnels (CNNs), les réseaux de neurones récurrents (RNNs), ou les réseaux de neurones de graphes (GNNs), respectent un type de symétrie. Par exemple, les CNNs sont invariants par translation. En ce qui concerne les GNNs, comme l'ordre des nœuds est arbitraire, ils doivent être invariants aux permutations. La spécificité des GNNs repose sur la manière dont les filtres sont construits : ils peuvent suivre une approche dite spectrale, faisant appel à la matrice d'adjacence ou au Laplacien du graphe, ou une approche dite spatiale faisant appel au voisinage des nœuds. Bien que d'origine différentes, ces approches peuvent être regroupées.

Chaque GNN a un niveau d'expressivité correspodndant à sa capacité à distinguer deux graphes non isomorphes ou, de manière équivalente, à approximer toute fonction équivariante par permtuation. Au-delà d'une architecture de GNN standard, on peut construire des architectures plus expressives. Cette construction se fait généralement d'une des deux manières suivantes. La première manière consiste à exploiter de manière exhaustive l'information accessible, comme par exemple en sommant sur l'ensemble des permutations, ou bien en utilisant les hyper-liens qui relient plus de deux nœuds ensemble. La deuxième approche consiste à identifier les nœeuds lors de l'apprentissage, ou à encoder de l'information sur le voisinage de chaque nœud. La Figure 1 présente une synthèse des architectures les plus expressives.

4 Chapitre 4: Utilisation des GNNs en pratique

Bien que certaines architectures sont théoriquements les plus expressives, elles ne sont pas pour autant nécessairement les plus appropriées lors de l'apprentissage sur des jeux de données réels. Il y a plusieurs raisons à cela. D'une part, les résultats théoriques se focalisent souvent sur des graphes sans attributs, alors que la plupart des graphe réels ont des attributs. D'autre part, le temps d'exécution de ces méthodes est assez long. De plus, certains résultats empiriques montrent que des réseaux moins expressifs peuvent obtznir des meilleurs résultats.

Dans ce chapitre, nous présentons plusieurs de ces architectures, à savoir le GCN (Graph Convolutional Network), GAT (Graph Attention neTwork) et GraphSAGE (Graph Sample and AGregatE), ainsi que plusieurs des améliorations

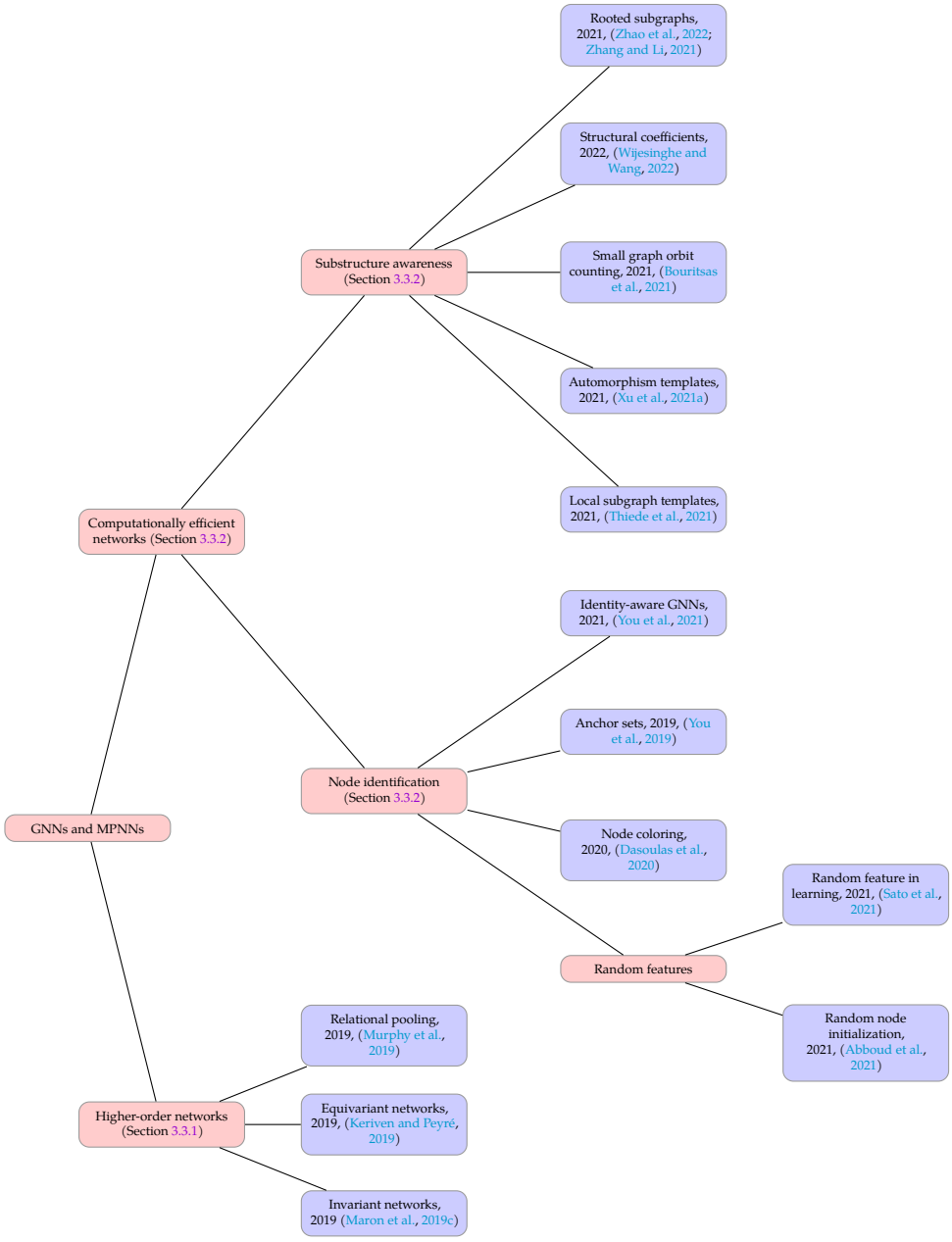


Figure 1: Vue d'ensemble des GNNs les plus expressifs.

qui peuvent être apportées à ces architectures. Dans une deuxième partie, nous présentons l'un des jeux de données sur lequel nous avons effectué la plupart de nos expériences, ainsi que les différentes manières de travailler dessus, tel l'apprentissage transductif et l'apprentissage inductif.

5 Chapitre 5: Apprentissage transductif et inductif, et classification multilabel

Deux problèmes qui se rencontrent lorsque l'on compare des architectures sur certains jeux de données sont la différence entre l'apprentissage transductif et l'apprentissage inductif, et la classification multilabel. Le premier problème a notamment lieu lorque l'on traite des graphes temporels en tant que graphes statiques. Selon la manière dont le graphe a été découpé en ensemble d'entraînement et ensemble de test, il peut y avoir une fuite d'information, dans l'apprentissage transductif, depuis l'ensemble de test vers l'ensemble d'entraînement (car les graphes sont souvent convertis en graphes non dirigés lors du prétraitement). Cela peut fausser l'évaluation des modèles.

Le second problème concerne la classification des nœuds. Dans certains cas, un nœud peut appartenir à plusieurs classes. L'évaluation d'un modèle pose alors la question suivante : est-ce l'architecture avec la plus grande précision, ou bien celle qui contient le plus souvent la classe correcte dans ces premières prédictions, qui est la meilleure ? Ce problème peut être vu autrement, du point de vue la corrélation entre les labels. En effet, il peut exister des liens causaux entre les labels. Deux classes peuvent avoir un champs sémantique (en ce qui concerne les attributs) similaires. Dans ce chapitre, nous traitons de ces problèmes sur un graphe de citation académique.

6 Chapitre 6: Attributs avec bruit et imputation d'attributs manquants

Alors que la qualité d'une image ou d'un texte est souvent observable par un être humain, celle d'un graphe est plus difficile à déterminer. Elle se décompose

en trois parties : la qualité de la structure du graphe, celle des attributs et celle des labels. Étant donné l'incertitude portant sur la qualité des attributs, il est important qu'un GNN soit résistant au bruit.

Dans ce chapitre, nous comparons les performances des GNNs les plus répandus face au bruit. En particulier, nous évaluons la précision face à plusieurs sortes de bruit : du bruit aléatoire, ou du bruit s'attaquant soit aux nœuds les plus influents, soit aux nœuds les plus isolés. Cela nous permet d'observer un mécanisme de réduction de bruit implicite pour certaines architectures. De plus, le bruit entraînant la plus grande perturbation est le bruit s'attaquant au nœuds les plus isolés.

Une autre source de bruit est la donnée manquante. Pour pallier ce manque, nous proposons une nouvelle architecture de GNN capable d'imputer les attributs manquants sur des données de type tabulaire ou de type graphe.

En amont de l'apprentissage sur les graphes se pose la question de la création d'un graphe. En prenant l'exemple d'une image ou d'un texte, il y a certains des éléments constituants qui sont plus importants que les autres. L'objectif est alors double : extraire les informations de l'objet afin d'obtenir la représentation la plus compacte, et maintenir la structure qui lie les éléments entre eux.

Ce chapitre présente un travail préliminaire effectué sur des lésions de la peau. L'objectif est d'extraire les patches d'une image à l'aide d'un critère tel que l'entropie, et d'évaluer les performances d'un modèle (ici, un CNN) sur ces patchs. Cette démarche peut permettre par la suite de créer des graphes à l'aide de ces patches et d'y ajouter d'autres informations médicales pour obtenir une donnée plus riche et atteindre des meilleures performances.

7 Chapitre 8: Conclusion et perspectives

7.1 Conclusions

Dans cette thèse, nous avons tout d'abord présenté la représentation théorique des réseaux de neurones de graphes, que cela soit dans leur forme générale ou dans les formes les plus puissantes en terme d'expressivité, ainsi que la manière

dans laquelle ils s'inscrivent dans la continuité des méthodes traditionnelles.

Nous avons montré l'importance du choix du type d'apprentissage, e.g., transductif ou inductif, classification simple ou multilabel. D'une part, la fuite d'information de l'ensemble de test vers l'ensemble d'entraînement dans l'apprentissage transductif nuit à la qualité de l'évaluation des modèles, en même temps qu'il peut masquer un éventuel surapprentissage. D'autre part, de nombreux problèmes de classification de nœuds devraient être posés comme des problèmes de classification multilabel, afin de prendre en compte les relations pouvant exister entre les labels.

La présence de bruit dans les attributs influence les performances des modèles et met en évidence le mécanisme implicite de réduction de bruit présent dans certains modèles. De plus, le bruit a un effet néfaste plus important lorsqu'il concerne les nœuds les plus isolés. En ce qui concerne les attributs manquants, nous avons montré, en proposant une nouvelle architecture, que les GNNs pouvaient être utilisés pour imputer les valeurs manquantes.

En dernier lieu, nous avons entamé une réflexion sur la création de graphes à partir de données provenant de l'imagerie médicale. Cette création doit se faire en respectant deux contraintes fondamentales : extraire la représentation la plus compacte qui préserve la structure de la donnée.

7.2 Perspectives

Les perspectives à court terme se concentrent sur deux axes principaux : poursuivre le travail sur la conversion d'images médicales en graphes, et l'étude de bruit dans les réseaux dynamiques.

En ce qui concerne les graphes d'images médicales, l'objectif est d'obtenir une représentation qui permette de dépasser les résultats actuels des réseaux convolutionnels, en exploitant les données structurelles des graphes. L'un des premiers points concerne la définition des attributs. Par exemple, pour une image des pores présents dans l'œil humain, si un pore est représenté par un nœud, quels attributs doit avoir ce nœud ? De plus, quels doivent être les liens entre les pores ? Faut-il que tous les nœuds soient reliés entre eux, ou bien seulement les voisins géographiques ?

Ce travail a pour but d'ouvrir la voie à la création de graphes issus de l'imagerie en 3D, où les liens entre les nœuds n'indiquent plus forcément un voisinage *intra-coupe*, mais peuvent aussi représenter un voisinage *inter-coupe*, i.e., entre deux pores de coupes différentes.

Le deuxième axe, concernant les graphes dynamiques, serait de prolonger les résultats présentés dans cette thèse, en particulier ceux du Chapitre 6. La question est d'étudier le rôle de bruit dans un graphe dynamique. Il ne s'agit plus seulement de s'attaquer aux nœuds isolés ou affluents, mais d'évaluer l'aspect temporel du bruit, par exemple étudier comment réagit un GNN face à du bruit ponctuel (d'un point de vue temporel) de forte intensité, ou bien face à du bruit continu de faible intensité.

De la même manière que nous avons proposé une architecture pour l'imputation de données manquantes pour les données tabulaires et les données de type graphe, l'objectif serait de concevoir une méthode permettant d'imputer les données manquantes dans un graphe dynamique.

Dans une perspective plus long terme, l'uns des objectifs de l'étude des réseaux dynamiques, du bruit et des données médicales serait la mise en place d'un système robuste pouvant servir d'aide au diagnostic pour les médecins ou la recherche médicale. L'étude du bruit, en parallèle de travaux sur l'explicabilité des GNNs, permettrait de fournir des résultats utilisables par les médecins. L'intégration de données médicales de plusieurs genres, comme les IRM, CT Scan, compte-rendus médicaux, etc., dans un unique graphe permettrait une vision d'ensemble plus élargie du suivi médical d'un patient, et de déceler des motifs cachés dans la donnée.