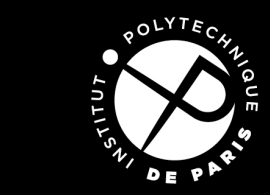


INSTITUT POLYTECHNIQUE DE PARIS

NNT : 20XXIPPAXXXX

Thèse de doctorat

Robust machine learning for Graphs/Networks

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom ParisTech

Thèse de doctorat de l'Université Internationale de Rabat

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat : Réseaux, Informations et Communications

Thèse présentée et soutenue à Rabat, le 02/02/2023, par

HAKIM HAFIDI

Composition du Jury :

Amal SEGHROUCHNI ELFALLAH Professeure, Sorbonne Université (LIP6)	Président
Pierre BORGNAT Directeur de recherche au CNRS, ENS Lyon (Laboratoire de Physique)	Rapporteur
Cédric RICHARD Professeur, Université Côte d'Azur (Laboratoire Lagrange)	Rapporteur
Mohammed BOULMALF Professeur, Université Internationale de Rabat (TICLab)	Examinateur
Karim BAINA Professeur, ENSIAS Université Mohammed V (Rabat IT Center)	Examinateur
Philippe CIBLAT Professeur, Télécom Paris (LTCI)	Directeur de thèse
Mounir Ghogho Professeur, Université Internationale de Rabat (TICLab)	Co-directeur de thèse

Contents

List of Figures iii

List of Tables v

Introduction 5

1 Background and related work 11
- 1.1 Notations and definitions . . . 12
  - 1.1.1 Graphs topology and the adjacency matrix . . . 12
  - 1.1.2 Attributed graphs . . . 13
- 1.2 The node classification problem . . . 14
- 1.3 Graph neural networks . . . 15

2 Contrastive learning for nodes' representation 19
- 2.1 Introduction . . . 20
- 2.2 Background . . . 22
  - 2.2.1 Problem formulation. . . . 22
  - 2.2.2 Graph Neural Networks (GNNs). . . . 22
  - 2.2.3 Contrastive learning . . . 23
  - 2.2.4 Simple Siamese neural networks for nodes representation . . . 23
- 2.3 Methodology of the proposed approach . . . 24
  - 2.3.1 GraphCL . . . 24
  - 2.3.2 Negative sampling strategies . . . 24
  - 2.3.3 Overview of GraphCL . . . 26
  - 2.3.4 Extension to inductive setup . . . 26
- 2.4 Experiments . . . 27
  - 2.4.1 Experimental setup . . . 27
  - 2.4.2 Results . . . 30
  - 2.4.3 Ablation study . . . 31
- 2.5 Discussion . . . 34
  - 2.5.1 Connection to mutual information . . . 34
  - 2.5.2 Understanding contrastive learning through alignment and uniformity on the hypersphere . . . 35
  - 2.5.3 Computational and model complexity . . . 37
- 2.6 Conclusion . . . 38

3 Graph assisted bayesian classifiers 41
- 3.1 Introduction . . . 42
- 3.2 New Graph-Assisted Bayesian Classifier . . . 44
  - 3.2.1 The Classifier derivation . . . 44
  - 3.2.2 Parameters' estimation . . . 50
  - 3.2.3 When does graph structure not help GAB . . . 51
- 3.3 Link to Graph Neural Networks . . . 53
  - 3.3.1 Recap of Graph Neural Networks . . . 53
  - 3.3.2 Relationship with a GNN based classifier . . . 54

3.4 Discussion on complexity issues . . . 56
3.5 Numerical Results . . . 59
3.5.1 Synthetic datasets . . . 59
3.5.2 Real datasets . . . 60
3.6 Conclusion . . . 62

4 Robustness to adversarial attacks **65**
4.1 Introduction . . . 66
4.2 Adversarial attacks . . . 67
4.2.1 Gradient-Based Attacks . . . 67
4.2.2 Optimization-Based Attacks . . . 68
4.2.3 Generative Adversarial Network (GAN)-Based Attacks . . . 68
4.3 Adversarial attacks on graphs . . . 69
4.3.1 Attacker's Capacity . . . 69
4.3.2 Methods of Attacking a Graph . . . 69
4.4 Measures to prevent adversarial attacks on graphs . . . 70
4.4.1 Adversarial training . . . 71
4.4.2 Data purification . . . 71
4.5 Robustness of the graph assisted Bayesian classifier . . . 71
4.5.1 The graph assisted Bayesian classifier as a belief propagation framework . . . 72
4.5.2 Node similarity as a simple defence mechanism for GAB . . . 74
4.6 Experiments . . . 74
4.6.1 Attack scenario . . . 74
4.6.2 Attack description . . . 75
4.6.3 Data description . . . 75
4.6.4 Results . . . 76
4.7 Conclusion . . . 76

Conclusion and perspectives **79**

A Appendix A **81**
A.1 Derivations for Eq. (3.3) . . . 81
A.2 Derivations for Eq. (3.12) . . . 82

Bibliography **89**

List of Figures

1.1 Example of d-hops neighborhoods of node u. The first set $\mathcal{N}_1(u) = \{a, b, c\}$ is the set of adjacent nodes to node u, the second comprises nodes of distance 2 from node u $\mathcal{N}_2(u) = \{d, e, f\}$, and the third set defines the 3-hops neighborhood $\mathcal{N}_3(u) = \{h, g\}$ 13

1.2 Example of an attributed graph, with node features and node labels encoded as colors of the nodes. The label space is defined as $\mathcal{Y} = \{"yellow", "green", "blue"\}$, uncolored nodes belong to the set of unlabelled nodes $\mathcal{V}_{UL} = \{c, g\}$. 14

1.3 Nodes with color indicate that the class is known; nodes with white indicate that a class must be predicted. 15

2.1 A high-level overview of our method for a subgraph around node u. $h_{u,1}$ and $h_{u,2}$ form a positive pair with a query $h_q = h_{u,1}$ and its corresponding key $h_q^+ = h_{u,2}$. 26

2.2 Classification accuracy on Cora dataset. Effect of the number of negative examples. 32

2.3 Classification accuracy on Cora dataset. Effect of the similarity between the current example and its corresponding negative samples. 33

2.4 A comparison of Siamese NN trained *with* vs *without* stop-gradient and batch normalization. Training loss across epochs. 35

2.5 A comparison of Siamese NN trained *with* vs *without* stop-gradient and batch normalization. Accuracy of a linear classifier trained on top of the representation on Cora dataset 36

2.6 Representations of Cora dataset nodes on $\mathbb{R}^2$ using encoders trained with a contrastive loss. Histograms of the cosine similarity between positive pairs. 37

2.7 Representations of Cora dataset nodes on $\mathbb{R}^2$ using encoders trained with a negative cosine similarity loss. Histograms of the cosine similarity between positive pairs. 38

2.8 Representations of Cora dataset nodes on $\mathbb{R}^2$ using encoders trained with a supervised cross entropy loss. Histograms of the cosine similarity between positive pairs. 39

2.9 Representations of Cora dataset nodes on $\mathbb{R}^2$ using encoders trained with a contrastive loss (**upper plot**), a negative cosine similarity loss (**middle plot)** and a supervised cross entropy loss (**bottom plot**). Feature distributions in $\mathbb{R}^2$ using Gaussian kernel density estimation. 40

3.1 GAB and GNN performance on synthetic datasets versus DoI. 60

3.2 GAB and GNN performance on Pubmed versus the size of the training set. 61

3.3 Accuracy performance with added noisy edges in Cora. 63

List of Tables

2.1 Description of datasets 28
2.2 Classification accuracy on transductive tasks and micro-averaged F1 score on inductive tasks 31
2.3 Classification accuracy on Cora dataset. Effect of the number of hops between the current example and its corresponding negative samples. 33
2.4 Classification accuracy on Cora dataset. Effect of the choice of the encoder 34

3.1 Intra- and inter-class connection probabilities and classification accuracies. 43
3.2 Number of parameters or weights to be tuned/learnt. 57
3.3 Number of flops for the training phase. 58
3.4 Number of flops for the inference phase. 59
3.5 Description of the real datasets. 60
3.6 30%-supervised node classification accuracy (%). 62

4.1 Statistics of datasets 75
4.2 GCN vs GAB : Effect of SPEIT attack [98] on the node classification accuracy (%) without using any defence mechanism 76
4.3 GCN vs GAB : Effect of SPEIT attack [98] on the node classification accuracy (%) using defence mechanisms 77

Abstract

Graphs are an effective tool for representing data produced by complex systems of interactions. They offer a unifying framework for aligning structured and unstructured data. They are characterized by a set of nodes, which represent the entities, and a set of links connecting them representing relationships between them. Nodes may be of different types and may further be associated with several features. And links may represent different relationships and may also be associated with different attributes or semantic content. The richness of these data as well as their appearance in several scientific disciplines and the proliferation of databases of this type have led to the emergence of several studies which aim to extend approaches from signal processing and deep learning to the graph setting. Recently, graph representation learning has attracted the attention of the scientific community as a way of analyzing graphs and helping to leverage the richness of information that resides in unstructured data. One of the major challenges facing graph representation learning is learning node embeddings which capture both node features and the graph structure. These representations can then be fed into downstream machine learning models. Several interesting applications and problems have been discussed in the literature; classification of graphs (eg. molecules), edge prediction (eg. Social network links), node regression and classification (eg. citation networks). Most successful approaches for graph representation learning are based on Graph Neural networks (GNN), a class of deep learning methods designed to perform inference on data described by graphs. A classic example is the Graph Convolutional Network (GCN). GCN filter can be seen as an aggregation operator, i.e., the representation of a node is obtained by averaging its intrinsic features with those of its neighbors. GCN was first designed for the task of graph-based semi-supervised learning which consists of classifying nodes in a graph where labels are only available for a small subset of nodes. The node classification task has then become a standard problem for studying the performance of GNN and several approaches have been proposed to build more sophisticated and expressive GNN. These models, however, suffer from several drawbacks. First, similarly to deep learning models on other modalities such as images and text, successful GNNs methods are trained in a supervised way and thus rely on quality labeled data to perform well. Secondly, their performance varies significantly across datasets and depends on the structure of the graph for node classification problems. Thirdly, it has been shown that they are vulnerable to adversarial attacks. The objective of this thesis is to address the above-mentioned problems by providing new techniques for the node representation and node classification problems. To address the first point, we use contrastive learning, a method that learns nodes representation in a self-supervised manner, which reduces the reliance on labeled data to obtain good representations. For the node classification problem, we propose a Bayesian based classifier that takes into account the structure of the graph. Lastly, we study the robustness of the proposed classifier to adversarial attacks and introduce a simple defence mechanisms against this type of attacks.

Résumé

Les graphes sont un outil efficace pour représenter les données produites par des systèmes d'interactions complexes. Ils offrent un cadre unifié pour aligner les données structurées et non structurées. Ils sont caractérisés par un ensemble de nœuds, qui représentent les entités, et un ensemble de liens les reliant, qui représentent les relations entre eux. Les nœuds peuvent être de différents types et peuvent en outre être associés à plusieurs caractéristiques. Les liens peuvent représenter différentes relations et peuvent également être associés à différents attributs ou contenus sémantiques. La richesse de ces données ainsi que leur apparition dans plusieurs disciplines scientifiques et la prolifération des bases de données de ce type ont conduit à l'émergence de plusieurs études qui visent à étendre les approches du traitement du signal et de l'apprentissage profond au cadre des graphes. Récemment, l'apprentissage de représentation de graphes a attiré l'attention de la communauté scientifique comme moyen d'analyser les graphes et d'aider à exploiter la richesse d'information qui réside dans les données non structurées. L'un des principaux défis de l'apprentissage de représentation des graphes est d'apprendre des représentations de nœuds qui capturent à la fois les caractéristiques des nœuds et la structure du graphe. Ces représentations peuvent ensuite être utilisées par des modèles d'apprentissage automatique pour résoudre différents problèmes. Plusieurs applications et problèmatiques intéressantes ont été abordés dans la littérature : la classification de graphes (par exemple, molécules), la prédiction de liens (par exemple, liens de réseaux sociaux), la régression ou la classification de nœuds (par exemple, des réseaux de citations). La plupart des approches efficaces pour l'apprentissage de représentation des graphes sont basées sur les réseaux de neurones pour graphes (GNN), une classe de méthodes d'apprentissage profond conçues pour effectuer des inférences sur des données décrites par des graphes. Un exemple classique est le réseau à convolution pour graphes (GCN). Le filtre GCN peut être considéré comme un opérateur d'agrégation, c'est-à-dire que la représentation d'un nœud est obtenue en faisant la moyenne de ses caractéristiques intrinsèques avec celles de ses voisins. Le GCN a d'abord été conçu pour la tâche d'apprentissage semi-supervisé, qui consiste à classifier les nœuds dans un graphe où les labels ne sont disponibles que pour un petit sous-ensemble de nœuds. La tâche de classification des nœuds est ensuite devenue un problème standard pour étudier les performances des GNN et plusieurs approches ont été proposées pour construire des GNN plus sophistiqués et plus expressifs. Ces modèles souffrent toutefois de plusieurs inconvénients. Tout d'abord, comme les modèles d'apprentissage profond pour d'autres modalités telles que les images et le texte, les méthodes GNN efficaces sont entraînées de manière supervisée et dépendent donc de données étiquetées de qualité pour être performantes. Deuxièmement, concernant les problèmes de classification des nœuds, les performances des GNN varient considérablement en fonctions des jeux de données et dépendent de la structure du graphe. Enfin, il a été démontré qu'ils sont vulnérables aux attaques adversariaux. L'objectif de cette thèse est de répondre aux problèmes mentionnés ci-dessus en fournissant de nouvelles techniques pour les problèmes de représentation et de classification des nœuds. Pour répondre au premier point, nous utilisons l'apprentissage contrastif, une méthode qui apprend la représentation des nœuds de manière auto-supervisée, ce qui réduit la dépendance aux données étiquetées pour obtenir de bonnes représentations. Pour le problème de la classification des nœuds, nous proposons un classificateur bayésien qui

prend en compte la structure du graphe. Enfin, nous étudions la robustesse du classificateur proposé aux attaques adverses et introduisons des mécanismes de défense simple contre ce type d'attaques.

Introduction

Context and objectives of the thesis

We live in a complex and interconnected environment. We can observe these interconnections in several phenomena, social networks, price fluctuations in the financial markets, road or air traffic as well as in brain activity or gene interaction. All these phenomena can be characterized by a set of entities that interact with each other which leads to a complex system. The deep understanding of this kind of systems requires to go beyond the study of the entities that compose them. Many studies have shown that graphs are an effective tool for representing relations between pairs of entities. It is through them that we are able to gain a deeper understanding of not only immediate but also high-order interactions, as well as how these interactions affect the underlying mechanisms that governs the studied phenomena. There has therefore been great interest in developing the theory and algorithms to handle graphs.

Graphs are characterized by a set of nodes, which represent the entities, and a set of links connecting them representing relationships between them. Nodes may be of different types, and may further be associated with several features. And links may represent different relationships and may also be associated with different attributes or semantic content.

Graphs have been studied through different and complementary prisms by several scientific communities. Mathematicians have been especially interested in developing what is now called classical graph theory which focuses on understanding and analyzing graph structure as well as solving combinatorial problem such as finding mappings, paths or flows. On the other hand, scientists from disciplines such as social science, biology and physics contributed to the development of network science, an academic filed that is defined as "the study of network representations of physical, biological, and social phenomena leading to predictive models of these phenomena." More recently researchers from signal processing and machine learning communities extended concepts from their respective fields to the graph setting.

In parallel to the interest of researchers in graphs and the development of a multitude of methods and techniques for graph analysis, the last few years have seen a rapid development of deep learning. A subfield of machine learning that is inspired by the brain structure and based on artificial neural networks. The proliferation of large databases and the acceleration of computing power have allowed these methods to achieve surprising success in various fields. These methods derive their success from their ability to automatically extract quality representations for different modalities, such as image, sound or text. Today we find direct applications of these methods in several technological tools that we use in our daily lives.

We have also witnessed in recent years the emergence and availability of large databases representing complex systems characterized by the interactions of millions or billions of components. Thanks to the internet revolution which offers the possibility to share and store huge amounts of data at low cost, we can now efficiently process and analyze these

kinds of databases. Large-scale projects have allowed scientists and the curious of all kinds to have access to quantities of quality data, which was impossible or even unimaginable a few decades ago. For example, web mapping informs us about the links or connections between web pages. Social networks offer social interaction data such as the list of friends or followers of each user of these platforms. In biology, researchers from around the world have collaborated to provide the scientific community with maps of protein-protein interactions.

The processing of these data can therefore be treated as a computational task on graphs in a wide range of important applications. For example, the recommendation of friends in social networks can be considered as a link prediction task, the detection of fraudulent accounts can be viewed as a node classification problem, while the prediction of the properties of chemicals can be considered as a graph classification task. Learning vector representations of either individual nodes or graphs is essential for facilitating these tasks. To achieve this, deep learning is a great prospect given its success in learning representations in images and texts. However, deep learning on graphs poses enormous challenges when compared to images and text. In graphs, nodes are unordered and can have a different number of neighbors. It is therefore necessary to design new deep learning models for graphs since traditional deep learning models cannot directly be applied.

One of the strengths of deep learning is its ability to automatically generate or extract meaningful representations. This is commonly called representation learning. In other words, we try to find a function that takes as input arbitrary objects and that has as a target a continuous space of fixed dimension. A good representation is one that keeps the essential features of the objects and reflects them in the geometric properties of the destination space. For example, images containing the same objects should have similar representations in the destination space. In the same way, words or sentences that have the same meaning should be represented by nearby vectors in the destination space.

Obtaining good representations is closely related to the relevance of a number of assumptions, the choice of the architecture, the training loss, and the optimization methods. These choices or assumptions are made to increase the ability of generalization of the learning algorithms, i.e., their effectiveness of accurately predicting outputs of previously unseen inputs. We refer to the set of assumptions as inductive or learning bias. For example, a powerful inductive bias of deep learning on images is the convolutional neural network architecture (CNN). In order to take use of the fact that image feature statistics are frequently translation invariant, CNNs use a particular architecture in which model parameters are connected or shared across image regions. The CNN model gains a beneficial inductive bias from this type of parameter sharing because its "filters" only need to learn from local features and can therefore generalize effectively to other areas in the image.

When it comes to graphs, finding representation of both the individual nodes and the complete graph is our main concern. One way to work with graphs is to ignore the graph structure and instead use a generic neural network model. For example, we could represent a graph as a list of its nodes and edges, and then use a standard neural network model, such as a recurrent neural network (RNN) or a convolutional neural network (CNN), to process the graph. However, this approach is likely to be suboptimal because it fails to use the graph structure as an inductive bias. For example, a social network contains information about the

relationships between people, but a generic neural network will not be able to exploit this information. To address this problem, a new family of neural network models have been developed, known as Graph Neural Networks (GNNs). GNNs use the graph structure as an inductive bias. In particular, they use the graph structure to define a computation graph, where the computation at each node depends on its neighbours in the graph. This allows them to capture the complex relationships between nodes in a graph. A node's embedding often takes into account data from its immediate neighbourhood, whereas graph embeddings are further calculated as some sort of aggregation (pooling) of node embeddings. This should be done using two distinct types of information. The first is the adjacency matrix of the graph, which define its structure. The second is the available information about the nodes and/or edges intrinsic features.

GNNs have then become the most successful approaches for graph representation learning. A classical example is the Graph Convolutional Network (GCN) [51]. The GCN filter can be seen as an aggregation operator, i.e. the representation of a node is obtained by averaging its intrinsic features with those of its first-order neighbors using the symmetrically normalized adjacency matrix. GCN was designed for the task of graph-based semi-supervised learning which consists of classifying nodes in a graph where labels are only available for a small subset of nodes. The node classification task has then become a standard problem for studying the performance of GNN and several approaches have been proposed to build more sophisticated and expressive GNN. As an example, the authors in [81, 77] introduced GAT and AGNN that use an attention mechanism to update the adjacency matrix by giving different weights to neighbors based on nodes' and edges' features. Other researchers explored higher-order information of the graph by repeatedly mixing feature representations of neighbors at various distances [2] or by modifying the propagation strategy of GCN by exploring its relation to the PageRank algorithm [53].

The success of deep learning has led to an increase in the use of machine learning models in security-critical fields. This has raised concerns about the security of machine learning models. One of the major concerns of machine learning models is their vulnerability to adversarial attacks. These are small perturbations to the input that can cause the model to produce erroneous outputs. These perturbations are usually imperceptible to humans. This is an important concern since the security of many systems such as autonomous cars, malware detection systems, and spam filters rely on the performance of machine learning models. In the past few years, the focus has shifted towards adversarial attacks on graph data. Adversarial attacks on graph data have been used to attack machine learning models in security-critical fields such as malware detection systems, spam filters, and recommendation systems. They have also been used to evaluate the robustness of machine learning models and to provide insights into the vulnerabilities of the models. These attacks can also be used to protect the privacy of users in social networks. Adversarial attacks on graphs are different to attacks on other types of data, because of the unique properties of graphs. The goal of adversarial attacks on graphs is to add or remove a limited number of edges or nodes to change the output of the model, usually the prediction of a node's label.

Our proposed work aim at improving three important aspects of the node classification task.

1. With the objective of reducing the reliance on human intervention for supervision

and leveraging the richness of the available unlabeled data, the first contribution is to develop a method capable of learning node representations in an unsupervised way. We proposed Graph Contrastive Learning (GraphCL), a general framework for learning node representations in a self supervised manner. GraphCL learns node embeddings by maximizing the similarity between the representations of two randomly perturbed versions of the intrinsic features and link structure of the same node's local subgraph. We use graph neural networks to produce two representations of the same node and leverage a contrastive learning loss to maximize agreement between them. In both transductive and inductive learning setups, we demonstrate that our approach significantly outperforms the state-of-the-art in unsupervised learning on a number of node classification benchmarks.

2. With the aim of improving the robustness of node classifiers to structural noise, we propose a graph-assisted Bayesian node classifier (GAB) which takes into account the degree of impurity of the graph, and show that it consistently outperforms GNN based classifiers on benchmark datasets, particularly when the degree of impurity is moderate to high.

3. With the objective of improving the robustness of node classifiers to adversarial attacks. We first study the robustness of our proposed classifier (GAB) to these attacks on graph structured data. We show that it is naturally significantly more robust then GNNs to this kind of attacks. We further suggest simple defence mechanisms and compare their performance with equivalent defences on GNNs.

Outline of the manuscript and contributions

There are four chapters in this thesis. Chapter 1 provides a brief summary of the background and related and prior work on graph-structured data analysis. Our original contributions are presented in Chapters 2, 3, and 4.

In chapter 1, we first introduce general notations and definitions about graph topology and attributed graphs. We then define the node classification problem on attributed graphs. Finally, we present related work on graph neural networks.

In chapter 2, we tackle the problem of learning nodes' representation in an unsupervised manner. We opt for constrastive learning, a self-supervised learning technique. We present the general framework of our approach along with multiple ways of generating negative sample for the contrastive learning framework. Lastly, we study the performance of the learnt representations on node classification problems. We present comparison of our approach with different baselines and state of the art methods on a number of benchmark datasets in both transductive and inductive learning settings.

In chapter 3, we focus on the node classification problem. We start from the observation that GNNs have performance that varies significantly with the datasets. We show how the degree of impurity of the graphs can explain these variations and introduce a novel Bayesian node classifier that takes into account the degree of impurity of graphs when classifying nodes. We present details of the derivations of our proposed classifier and comparison of

its performances with state of the art methods on multiple benchmark datasets. We show that it is more robust then GNN-based methods to structural noise.

In chapter 4, we study the robustness of the classifier we presented in chapter 3 to adversarial attacks. We first give detailed presentation of adversarial attacks and defences on graph-structured data. We then show that our classifier is more robust to these types of attacks then GNN based methods. Finally, we suggest simple defence mechanism to improve the robustness of our classifier.

Publications

Following is a list of publications resulting from the research conducted during the thesis.

- Hakim Hafidi, Mounir Ghogho, Philippe Ciblat, and Ananthram Swami. "Negative sampling strategies for contrastive self-supervised learning of graph representations". In: *Signal Processing* 190 (2022), p. 108310
- Hakim Hafidi, Mounir Ghogho, Philippe Ciblat, and Ananthram Swami. "Bayesian Node Classification for Noisy Graphs". In: *2021 IEEE Statistical Signal Processing Workshop (SSP)*. IEEE. 2021, pp. 246–250
- Hakim Hafidi, Mounir Ghogho, Philippe Ciblat, and Ananthram Swami. "Graph-Assisted Bayesian Node Classifiers". In: *IEEE Access* (2022 (under review))
- Hakim Hafidi, Mounir Ghogho, Philippe Ciblat, and Ananthram Swami. "On the Robustness of Bayesian Graph Classifiers to Adversarial Attacks". In: *IEEE Access* (2022 (To be submitted))

Chapter 1

Background and related work

1.1 Notations and definitions 12
1.1.1 Graphs topology and the adjacency matrix 12
1.1.2 Attributed graphs 13
1.2 The node classification problem 14
1.3 Graph neural networks 15

1.1 Notations and definitions

1.1.1 Graphs topology and the adjacency matrix

Definition 1 (*Graph*). A graph is a pair $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is a set of nodes (or vertices) and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges. Each edge $e \in \mathcal{E}$ is pair (u, v) with $u, v \in \mathcal{V}$. A graph can be weighted if there is a mapping $w : e \rightarrow w_e$ that assign a weight w_e to the corresponding edge e. Otherwise it is called an unweighted graph. A graph is undirected if $(u, v) \in \mathcal{E} \Leftrightarrow (v, u) \in \mathcal{E}$, i.e relationships between nodes are symmetric. In contrast, when relationships are asymmetric, i.e $(u, v) \in \mathcal{E}$ does not imply that $(v, u) \in \mathcal{E}$, the graph is said to be directed. In this thesis, unless explicitly mentioned, we consider graphs to be unweighted and undirected.

Definition 2 (*Adjacent nodes*). A node v is called adjacent to node u if there is an edge $(u, v) \in \mathcal{E}$.

Definition 3 (*Path*). A path is a sequence of nodes $P = \{u_1, \ldots, u_n\}$ such that u_i is adjacent to u_{i+1} for $1 \leq i \leq n$.

Definition 4 (*Distance*). The distance $\Delta(u, v)$ between nodes u and v is the number of edges in the shortest path connecting them.

Definition 5 (*Neighborhood*). The d-hops neighborhood $\mathcal{N}_d(u)$ of node u is the set of nodes of distance d from node u, i.e., $\mathcal{N}_d(u) = \{v \in \mathcal{V} | \Delta(u, v) = d\}$. When there is no room for confusion, we refer to the 1-hop neighborhood simply by $\mathcal{N}(u)$, the set of adjacent nodes to u.

The notion of neighborhood is essential to most graph learning techniques. It defines the primary mechanism by which information propagates between nodes. Figure 1.1 shows a simple example of d-hops neighborhoods of node u. The first set $\mathcal{N}_1(u) = \{a, b, c\}$ is the set of adjacent nodes to node u, the second comprises nodes of distance 2 from node u $\mathcal{N}_2(u) = \{d, e, f\}$, and the third set defines the 3-hops neighborhood $\mathcal{N}_3(u) = \{h, g\}$.

Definition 6 (*Node degree*). The degree d_u of a node is the number of elements in its 1-hop neighborhood.

Definition 7 (*Adjacency matrix*). An adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ represents the topological structure of the graph where $N = |\mathcal{V}|$ is the number of nodes in the graph. In the case of unweighted graphs, matrix entries are defined such that $A_{u,v} = 1$ if $(u, v) \in \mathcal{E}$ and $A_{u,v} = 0$ otherwise. It is a symmetric matrix in the case of undirected graphs.

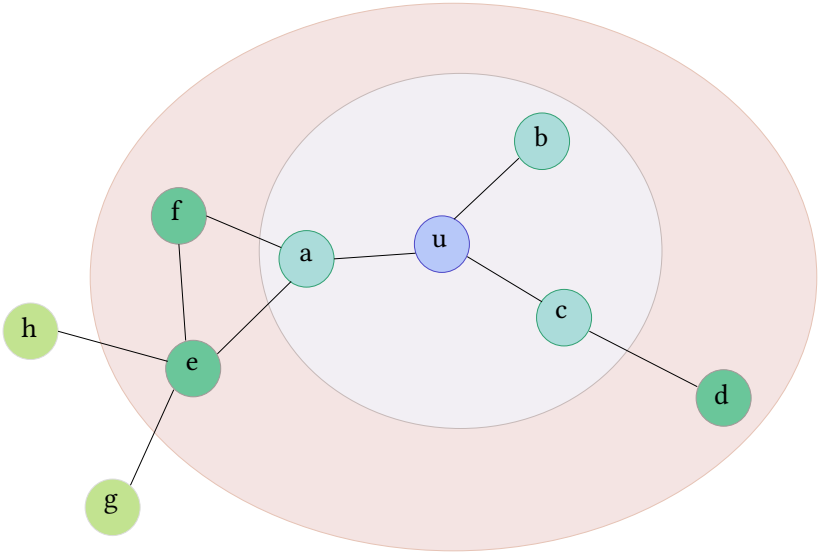


Figure 1.1: Example of d-hops neighborhoods of node u. The first set $\mathcal{N}_1(u) = \{a, b, c\}$ is the set of adjacent nodes to node u, the second comprises nodes of distance 2 from node u $\mathcal{N}_2(u) = \{d, e, f\}$, and the third set defines the 3-hops neighborhood $\mathcal{N}_3(u) = \{h, g\}$

1.1.2 Attributed graphs

Graphs, as defined above, encode structural information of the network of interest. Both the neighborhood sets and the adjacency matrix hold information about how the different components of our systems interact and relate to each other. While this kind of information is of upmost importance, it laks the contexual information that network components carry. For example in a citation network where nodes represent research papers and edges citations, textual information of the documents (the frequency of words that appear in the documents or their semantic representation in the form of text embeddings) would be very helpful in predicting the research field of the paper.

To encode such information, we represent each node $u \in \mathcal{V}$ by a feature vector $\boldsymbol{x}_u \in \mathbb{R}^{F \times 1}$ where F is the number of node's intrinsic features. The information of all nodes in the graph is either summarized in the form of a matrix $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N]^\top \in \mathbb{R}^{N \times F}$ or a set of feature vectors $\mathcal{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$.

Furthermore, depending on the application, we may have access to node-level or graph-level labels for a subset of nodes or graphs. Since the focus of this thesis is node-level problems,we define the label space $\mathcal{Y} = \{1, 2, \ldots, K\}$ as the set of possible categories or classes to which nodes may belong, where K is the number of classes. Therefore each node u belongs to a class $y_u \in \mathcal{Y}$. And the set on nodes in a graph $\mathcal{G}$ can be described as $\mathcal{V} = \mathcal{V}_L \cup \mathcal{V}_{UL}$ where $\mathcal{V}_L$ is the set of labelled nodes and $\mathcal{V}_{UL}$ the set of unlabelled nodes. Figure 1.2 illustrates an example of an attributed graph, with node features and node labels encoded as colors of the nodes. The label space is defined as $\mathcal{Y} = \{"yellow", "green", "blue"\}$, uncolored nodes belong to the set of unlabelled nodes $\mathcal{V}_{UL}$.

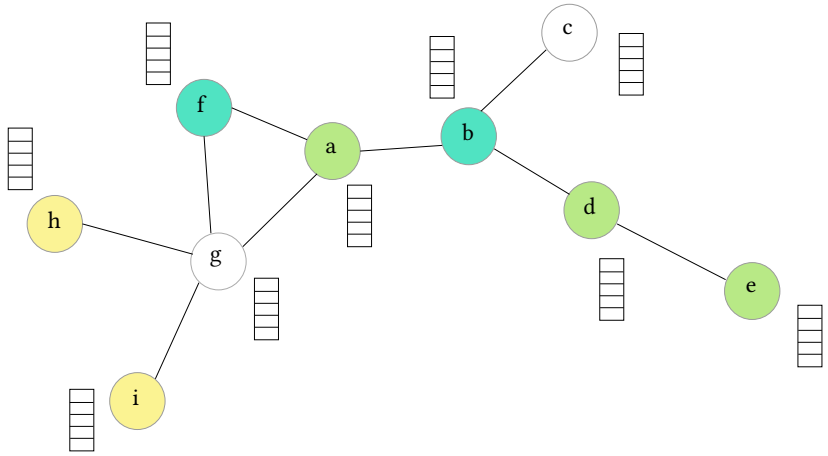


Figure 1.2: Example of an attributed graph, with node features and node labels encoded as colors of the nodes. The label space is defined as $\mathcal{Y} = \{"yellow", "green", "blue"\}$, uncolored nodes belong to the set of unlabelled nodes $\mathcal{V}_{UL} = \{c, g\}$.

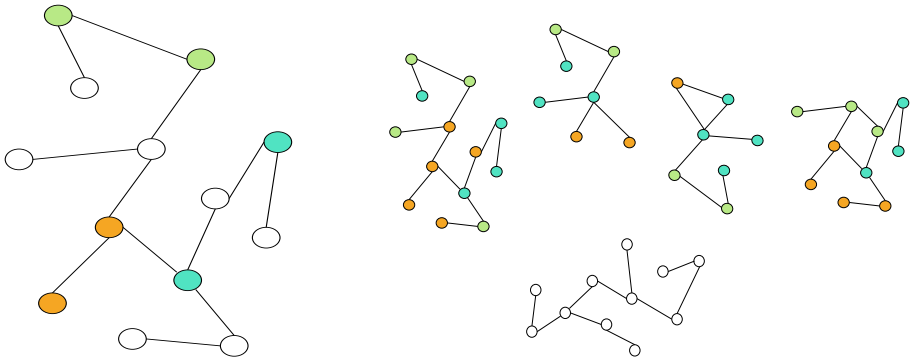
1.2 The node classification problem

As we have briefly mentioned in the introduction, multiple computational tasks are possible when dealing with graph data. One can be interested in classifying graphs as whole, classifying nodes or predicting the existence of edges for example. In this thesis we are specifically interested in the node classification task. We therefore introduce the latter problem in this section and formally describe it.

For node classification problems, we are interested in classifying nodes to different categories. We can therefore consider nodes to be the primary data of interest and edges as additional information to enhance the performance of our classifiers. In the sense that we can make decisions based only on the intrinsic attributes of nodes, and we may be able to make better decisions by considering the relationships between them.

It has been popular to categorize node classification problems by whether the model can generalize to unseen data. If so, the method operates in an *inductive* setting, otherwise it operates in a *transductive* setting. In the latter, the principal assumption is that we have access to features of all nodes of a given graph during the training time. We also have access to labels of a subset of nodes and the task consists of inferring labels of the remaining nodes of the graph. Whereas in the inductive learning setting, features and labels of nodes of a set of graphs are available. The goal is to learn rules that can generalize to nodes in previously unseen graphs (nodes whose features were not available during training).

We introduce here a more formal definition of the transductive setting of the node classification problem. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, its adjacency matrix $\boldsymbol{A}$, the feature matrix $\boldsymbol{X}$ and labels of a subset of nodes $\mathcal{V}_L$, the goal is to predict labels of the remaining nodes of the graph $\mathcal{V}_{UL}$.



(a) (b)

Figure 1.3: Nodes with color indicate that the class is known; nodes with white indicate that a class must be predicted.

1.3 Graph neural networks

In image processing, the concept of convolution is a strong basic element. The method involves moving a kernel matrix over the image to capture local patterns of interest inside the image. The idea of translational invariance, which states that a pattern is interesting regardless of where we discover it in the image, is encoded by this type of model. Additionally, it encodes a locality assumption: pixels that are close to one another will likely have stronger relationships than those that are far apart. This idea is at the heart of onvolutional neural networks (CNNs), which are the most commonly used neural network architecture for computer vision. [5] However, CNNs are limited to operating on data with a grid-like structure, such as images, videos, and text, which can be represented as a regular grid of pixels or characters. GNNs extend CNNs to data that can be represented as graphs, which are irregular structures that cannot be represented as regular grids. Hence, a GNN is a neural network that can operate on graph-structured data. The goal of GNNs is to learn a vector representation of the nodes, edges, or entire graph. This learned representation can then be used for downstream machine learning tasks, such as node classification, link prediction, and graph classification.

Historically, The first graph neural network was introduced by Gori et al. in 2005 [29]. In 2009, Micheli et al.[61] introduced the Neural Network for Graphs (NN4G) model, which is based on a constructive supervised learning algorithm. The NN4G model was used for node classification, graph classification, and graph regression. In 2013, Bruna et al. [6] proposed the spectral graph convolutional neural network (GCN) model. The spectral GCN model uses the Fourier basis of a graph to define a spectral convolution operation on the graph. In 2016, Defferrard et al.[18] proposed the Chebyshev spectral GCN model, which uses Chebyshev polynomials to approximate the spectral GCN model. This model can be used for node classification and graph classification. In 2017, Kipf and Welling [51] proposed the graph convolutional network (GCN) model, which is a simplified version of the Chebyshev spectral GCN model. The GCN model is a semi-supervised learning algorithm that can be used for node classification. In 2018, Hamilton et al. [38] proposed the GraphSAGE model, which is an inductive learning algorithm for GNNs. GraphSAGE learns a function

that generates embeddings for unseen nodes by sampling and aggregating features from the nodes' local neighborhoods. In 2019, Velickovic et al. [81] proposed the graph attention network (GAT) model, which uses self-attention to assign different weights to the neighbors of a node.

GNNs learn node representations by aggregating the features of neighboring nodes and edges. The output of the ℓ-th layer of these GNNs is generally expressed as:

$$\boldsymbol{h}_u^{(\ell)} = \sigma^{(\ell)}(\phi^{(\ell)}(\boldsymbol{h}_u^{(\ell-1)}, \{\boldsymbol{h}_v^{(\ell-1)} : v \in \mathcal{N}_1(u)\})) \tag{1.1}$$

where $\boldsymbol{h}_u^{(\ell)}$ is the feature vector of node u at the ℓ-th layer initialized by $\boldsymbol{h}_u^{(0)} = \boldsymbol{x}_u$ and $\mathcal{N}_1(u)$ is the set of first-order neighbors of node u. Different GNNs use different formulations for the non-linear function $\sigma^{(\ell)}$ (called activation function) and the linear function $\phi^{(\ell)}$ [38]. Note that a first-order GNN based classifier relies on one layer or equivalently considers only the 1-hop neighborhood in the graph.

Graph Convolutional Neural Network (GCN): The convolutional propagation rule used in GCN is defined as follows

$$\phi^{(\ell)} = (\boldsymbol{W}^{(\ell)})^\top \left(\frac{\boldsymbol{h}_u^{(\ell-1)}}{d_u + 1} + \sum_{v \in \mathcal{N}_1(u)} \frac{\boldsymbol{h}_v^{(\ell-1)}}{\sqrt{(d_u+1)(d_v+1)}} \right) \tag{1.2}$$

where

- $\boldsymbol{W}^{(\ell)}$ is a learnable weight matrix,
- d_u is the degree of node u.

The activation function (for any layer except the last one) is a rectified linear unit (ReLU). For the last layer, we consider the $\mathrm{softmax}$ which for each node u outputs the probability that node u belongs to class k. Then the node u is assigned to the class with the highest probability [51].

Graph convolution Operator Network (GON): In [3, 63], GON is defined as GCN where Eq. (3.20) is replaced with the following one

$$\phi_u^{(\ell)} = (\boldsymbol{W}_1^{(\ell)})^\top \boldsymbol{h}_u^{(\ell-1)} + (\boldsymbol{W}_2^{(\ell)})^\top (\sum_{v \in \mathcal{N}_1(u)} \boldsymbol{h}_v^{(\ell-1)}). \tag{1.3}$$

Unlike GCN, GON computes a transformation matrix of the central node that is different from the transformation of its neighbors.

Graph Isomorphism Network (GIN): In [89], GIN is defined as GCN or GON where Eqs.(3.20)-(3.21) are replaced with the following one

$$\phi_u^{(\ell)} = (\boldsymbol{W}^{(\ell)})^\top \left((1+\alpha)\boldsymbol{h}_u^{(\ell-1)} + \sum_{v \in \mathcal{N}_1(u)} \boldsymbol{h}_v^{(\ell-1)} \right). \tag{1.4}$$

where α is an positive hyper-parameter. GIN thus attributes a different learnable weight to the central node (through α) when combining information from its neighbors.

Graph Attention Network (GAT): In [81], GAT is defined as GCN, GON or GIN but with the following layer link

$$\phi_u^{(\ell)} = \sum_{v \in \mathcal{N}_1(u) \cup \{u\}} \alpha_{u,v}^{(\ell)} (\boldsymbol{W}^{(\ell)})^\top \boldsymbol{h}_v^{(\ell-1)}, \tag{1.5}$$

where $\alpha_{u,v}^{(\ell)}$ are normalized attention coefficients computed by an attention mechanism as follows:

$$\alpha_{u,v}^{(\ell)} = \frac{e^{\varsigma(\boldsymbol{w}^{(\ell)}[(\boldsymbol{W}^{(\ell)})^\top \boldsymbol{h}_u^{(\ell-1)} \| (\boldsymbol{W}^{(\ell)})^\top \boldsymbol{h}_v^{(\ell-1)}])}}{\sum_{k \in \mathcal{N}_1(u)} e^{\varsigma(\boldsymbol{w}^{(\ell)}[(\boldsymbol{W}^{(\ell)})^\top \boldsymbol{h}_u^{(\ell-1)} \| (\boldsymbol{W}^{(\ell)})^\top \boldsymbol{h}_k^{(\ell-1)}])}}. \tag{1.6}$$

with ς the leaky ReLu function, the weighting row vector $\boldsymbol{w}^{(\ell)} \in \mathbb{R}^{2H}$, where H is the size of the hidden layer and $\|$ corresponds to column concatenation.

Chapter 2

Contrastive learning for nodes' representation

2.1 Introduction . . . 20
2.2 Background . . . 22
2.2.1 Problem formulation. . . . 22
2.2.2 Graph Neural Networks (GNNs). . . . 22
2.2.3 Contrastive learning . . . 23
2.2.4 Simple Siamese neural networks for nodes representation 23
2.3 Methodology of the proposed approach . . . 24
2.3.1 GraphCL . . . 24
2.3.2 Negative sampling strategies . . . 24
2.3.3 Overview of GraphCL . . . 26
2.3.4 Extension to inductive setup . . . 26
2.4 Experiments . . . 27
2.4.1 Experimental setup . . . 27
2.4.2 Results . . . 30
2.4.3 Ablation study . . . 31
2.5 Discussion . . . 34
2.5.1 Connection to mutual information . . . 34
2.5.2 Understanding contrastive learning through alignment and uniformity on the hypersphere . . . 35
2.5.3 Computational and model complexity . . . 37
2.6 Conclusion . . . 38

2.1 Introduction

In many fields, the rapid increase in data volume and the complexity of its structure and representation make it difficult to exploit it effectively. Graphs offer a unified framework for aligning well-structured and unstructured data. However, graphs have long been poorly leveraged because of their complexity, and limited approaches relying on content associated with nodes and links. Recently, graph representation learning has attracted the attention of the scientific community as a way of analysing graphs and helping to exploit the richness of information that resides in poor-structured data. Graphs are characterized by a set of nodes, which represent the entities, and a set of links connecting them, representing relationships between the nodes. Nodes may be of different types, and may further be associated with several features. And links may represent different relationships and may also be associated with different attributes or semantic content. One of the major challenges facing graph representation learning is learning node embeddings which capture both node features and graph structure. These representations can then be fed into downstream machine learning models.

Most successful approaches for graph representation have been great efforts to generalize neural networks to graph data and fall under the umbrella of Graph Neural Networks (GNNs) or Deep Geometric Learning [3, 51, 5, 89]. These approaches have achieved remarkable results in a number of important tasks such as node classification [38, 9, 57] and link prediction [52, 95]. However, these methods are very reliant on human annotation and suffer from the necessity of some form of supervision. This requires high cost, expert knowledge in the domain and the use of annotated data, which is not often available. Hence, it is of importance to develop methods capable of learning representations in an unsupervised manner.

In order to compensate for the absence of labels or predefined tasks, some unsupervised methods have adopted the homophily hypothesis, which states that linked nodes should be nearby in the embedding space [42]. Inspired by the Skipgram algorithm for embedding words into a latent space, where adjacent vectors correspond to co-occurring words in a sentence [62], a majority of these methods use random walks to generate sentence-like sequences where co-occurring nodes are close to one another in the embedding space [69, 31] and can also be adapted to heterogeneous graphs [20, 93, 92]. Other methods, such as autoencoders, also employ the homophily hypothesis by reconstructing either the adjacency matrix or the neighborhood of a node [82, 52]. Despite their success in learning relatively powerful representations, relying on the homophily hypothesis may bias these methods towards emphasizing the direct proximity of nodes over topological information [82]. More recently, [80] proposed Deep Graph Infomax (DGI) that learns representations by training a discriminator to distinguish between representations of nodes that belong to the graph from nodes that belong to a corrupted graph. Leveraging recent advances in unsupervised visual representations [19], the success of DGI has been attributed to the maximization of mutual information between global and local parts of the input. This requires learning global representations of the entire graph which can be very costly and even intractable when dealing with large graphs.

To overcome the above-mentioned challenges, we here propose a contrastive frame-

work for self-supervised learning of nodes' representations, called GraphCL. We take inspiration from the success of contrastive losses in learning meaningful representations of images [13, 40] and develop a model that learns node embeddings by maximizing the similarity between the representations of two randomly perturbed versions (views) of the intrinsic features and link structure of the same node's local subgraph. The perturbation consists of randomly dropping from its L-hop subgraph, a subset of edges and nodes' intrinsic features. Other researchers have also used the contrastive loss to learn nodes or graph representations using different augmentation (perturbation) strategies. In [39], the authors used the diffusion matrix as a second view of the graph. In [90], the authors used four different strategies consisting of dropping nodes, perturbing edges, masking attributes or sampling subgraphs.

Contrastive learning is a special case of Siamese networks, which are weight-sharing neural networks applied to two or multiple inputs. Recent approaches use augmentations of the same data point as inputs and maximize the similarity between the learned representations of the two inputs. Maximizing the similarity between each pair of augmented data points in the dataset can lead to a trivial solution. Since we want representations of all pairs of augmented views to be equal, a possible solution is to map all nodes to a single point (representation). This is what we call a collapsing of representations to a single data point (i.e. if all representations are the same, then so are those of each pair of augmented views). Contrastive learning is one way of preventing this undesirable solution. It does so by contrasting between *positive* (similar) examples and *negative* (dissimilar) examples. The objective of the training phase is to map positive examples to nearby locations in the destination (representation) space while pushing away negative examples often by using *noise-contrastive estimation* [32]. One key component of contrastive learning frameworks is the choice of negative examples. The most common strategy is to uniformly sample from the training dataset using examples either from the current batch or from a memory bank. It has been shown that these approaches require large batches or memory banks to perform well for visual representation [13, 40]. To improve the performance and efficiency of contrastive frameworks, recent studies have proposed novel sampling strategies. Most strategies are based on the assumption that hard negative examples (i.e. examples that are hard to distinguish from a positive pair) are beneficial in learning more powerful representations. In [47], authors use hard negative mixing to synthesize new examples from the available hard negatives. In [86], the authors sample negatives from a *ring* around each positive (i.e. they sample negatives that are neither too close nor too far from the positive example).

More recent Siamese network architectures preventing representation collapsing still rely on the use of pairs of positive examples only. In [30], the authors experimentally show that the learned representations of their proposed framework do not collapse when using a momentum network even when not using negative examples. In [14], the authors avoid the collapsing phenomenon by simply using a stop-gradient strategy when directly maximizing the similarity between two augmented views. The stop-gradient strategy consists of considering the representation of one of the augmented views as a constant when updating the network parameters.

While these methods have shown surprisingly good results when applied to image

datasets, it is not clear whether they are easily generalizable to non-Euclidean data such as graphs. In this work, we introduce novel sampling strategies of negative examples based on the graph structure and show that our approach improves the performance of the learned representations on downstream classification tasks and outperforms existing methods. In addition, we conduct extensive experiments to study the different components of our Siamese network-based approach for learning nodes' representations which enable us to answer the following questions:

- Is a larger set of negative examples always useful in learning good representations?
- Does sampling hard negative examples improve the quality of the representation?
- Can we train a Siamese neural network to learn nodes' representations without using negative examples?

2.2 Background

2.2.1 Problem formulation.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph where $\mathcal{V}$ is a set of nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges. Each node $u \in \mathcal{V}$ is represented by a feature vector $x_u \in \mathbb{R}^P$. An adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ represents the topological structure of the graph where $N = |\mathcal{V}|$ is the number of nodes in the graph. Without loss of generality we assume the graphs to be unweighted i.e $\boldsymbol{A}_{u,v} = 1$ if $(u, v) \in \mathcal{E}$ and $\boldsymbol{A}_{u,v} = 0$ otherwise. We are also provided with a matrix $X \in \mathbb{R}^{N \times P}$ that summarizes the intrinsic feature vectors of all nodes.

Our objective is to learn an effective representation of nodes without human annotation. This will be done through the learning of a graph neural network encoder f that maps both node original feature and the graph structure to a higher level representation i.e. $f(X, A) = H^{(L)} \in \mathbb{R}^{N \times P'}$, where P' is the embedding size. The $u-$th row of $H^{(L)}$ corresponds to the embedding $h_u^{(L)}$ of node u. In the remainder of the chapter, h_u refers to the output of the GNN's last layer, i.e. $h_u = h_u^{(L)}$.

2.2.2 Graph Neural Networks (GNNs).

GNNs are a class of graph embedding architectures which use the graph structure in addition to node and edge features to generate a representation vector (i.e., embedding) for each node. Recent GNNs learn node representations by aggregating the features of neighboring nodes and edges. The output of the l-th layer of these GNNs is generally expressed as

$$h_u^{(l)} = COMBINE^{(l)}(h_u^{(l-1)}, AGGREGATE^{(l)}(\{(h_u^{(l-1)}, h_v^{(l-1)}) : v \in \mathcal{N}(u)\})), \quad (2.1)$$

where $h_u^{(l)}$ is the feature vector of node u at the l-th layer initialized by $h_u^{(0)} = x_u$ and $\mathcal{N}(u)$ is the set of first-order neighbors of node u. According to Eq. (2.1), $h_u^{(L)}$ corresponds to the

output of the last layer of the GNN, which involves the nodes of node u's L-hop subgraph. Different GNNs use different formulations of the COMBINE and AGGREGATE functions; the ones used in this work are described in subsection 2.4.1.

2.2.3 Contrastive learning

In this work, we consider the dictionary look up formulation of contrastive learning, which means that considering a query h_q, a corresponding positive pair h_q^+ and a set of negative examples Q_q^-, a contrastive loss is a function which has a low value when h_q is similar to h_q^+ and dissimilar to all elements of Q_q^-. A successful and widely used form of contrastive loss is defined as:

$$\mathcal{L}_{h_q,h_q^+,Q_q^-} = -\log \frac{\exp(h_q^\top h_q^+/\tau)}{\exp(h_q^\top h_q^+/\tau) + \sum_{h_n \in Q_q^-} \exp(h_q^\top h_n/\tau)}, \tag{2.2}$$

where τ is a temperature hyperparameter and h_q, h_q^+ and all h_n in Q_q^- are L_2 normalized feature vectors. The final loss is summed accross all queries q belonging to the dataset $\mathcal{D}$ and can be expressed as follows when scaled by the temperature τ [12]:

$$\mathcal{L} = -\frac{1}{\tau|\mathcal{D}|} \sum_{q \in \mathcal{D}} h_q^\top h_q^+ + \frac{1}{|\mathcal{D}|} \log \sum_{q \in \mathcal{D}} \left(\exp(h_q^\top h_q^+/\tau) + \sum_{h_n \in Q_q^-} \exp(h_q^\top h_n/\tau) \right), \tag{2.3}$$

where $|\mathcal{D}|$ is the number of elements in $\mathcal{D}$.

In Contrastive learning framework,, different negative sampling strategies (i.e., the way to build Q_q^-) may be employed to avoid collapsing of the contrastive loss optimization problem into a unique representation of all samples.

2.2.4 Simple Siamese neural networks for nodes representation

In [14], the authors argue that their approach can prevent collapsing when maximizing the similarities between the representations of two views of the same image without the use of negative examples. Their approach works by sampling two views of x_1 and x_2 of the same image x which they process using an encoder f and a multi-layer perceptron (MLP) prediction head g. Letting $p_1 = g(f(x_1))$, $p_2 = g(f(x_2))$, $h_1 = f(x_1)$ and $h_2 = f(x_2)$ denote respectively the outputs of the MLP prediction and the encoder, the objective is to minimize the symmetric negative cosine similarity loss which is defined as:

$$\mathcal{L} = \frac{1}{2}\mathrm{S}(p_1, \mathrm{stopgrad}(h_2)) + \frac{1}{2}\mathrm{S}(p_2, \mathrm{stopgrad}(h_1)), \tag{2.4}$$

where $\mathrm{S}(p_1, h_2) = -\frac{p_1}{\|p_1\|} \cdot \frac{h_2}{\|h_2\|}$ and the stopgrad operation consists of treating h_2, respectively h_1, as constant when updating the models' parameters.

2.3 Methodology of the proposed approach

2.3.1 GraphCL

GraphCL's objective is to learn node representations by maximizing the similarity between two embeddings of the same node. The two embeddings are obtained from applying a GNN encoder to two perturbed versions of the graph. This framework has three main components: a stochastic perturbation, a GNN based encoder and a contrastive loss function. We first introduce each of these components, and then give a high-level overview of the proposed method.

- **Stochastic perturbation.** We apply two stochastic perturbations to the graph which allow us to obtain two representations of the same node which we consider as positive examples. In this work, we consider simultaneous transformations of both node features and the connectivity of the graph. The graph structure is transformed by randomly dropping edges using samples from a Bernoulli distribution. For the node's original features, we apply a similar strategy by simply applying dropout to the input features;
- **Graph neural network encoder.** We apply a GNN based encoder that learns representations of all nodes in the graph. Our framework supports several choices of GNN architectures. Details about the choices of architectures are given in section 2.4.1.
- **Contrastive loss function.** We define a *pretext* prediction task that aims at identifying the corresponding positive example h_q^+ of a representation h_q given a set of generated examples, with h_q and h_q^+ being a positive pair of examples (i.e. obtained from the GNN representations of two transformations of the graph). As for the negative examples generation, details are provided in subsection 2.3.2.

2.3.2 Negative sampling strategies

Negative sampling has been shown to be a key ingredient for the success of contrastive learning frameworks. Different strategies have been proposed to build negatives examples for visual presentations [13, 40, 47, 86]. First, we investigate whether the conclusions that have been drawn from the most successful approaches of visual representations are still valid when applied to graphs. We hereafter introduce three negative sampling strategies: the two first are standard while the third one is new and well adapted to graph.

- **Random sampling.** This approach consists of considering the samples of the current (randomly generated) mini-batch as negatives. The problem with this approach is the number of negative examples is limited by the size of the mini-batch which is limited by the memory of the GPU. An alternative would be to randomly sample negatives from a memory bank that contains either representations of the whole training set or a queue with representations of the last few batches.

- **Feature-based sampling.** In [86], the authors propose to pick two percentiles ω_k and $\omega_l \in [0, 100]$ and considering h_{n_c} as a negative example for a representation of a query h_q if and only if $h_q^\top h_{n_c}$ is within the ω_k-th to the ω_l-th percentile of all $h_n \in Q_q^-$.This enables to build easily hard negative examples (i.e., negatives that are hard to distinguish from the current sample) which are beneficial in learning powerful representations as mentioned in [84, 91]. To adapt this method to the graph setting, instead of considering the similarities in the representation space, which requires using the encoder to learn the representations of all nodes in the graph, we simply consider the similarities of the nodes' original features. For each node u, we consider as negatives all nodes v whose original features are neither too close nor too far from those of node u (i.e. $x_u^\top x_v / \|x_u\| \|x_v\|$ is within the ω_k-th to the ω_l-th percentile of all nodes of the graph).

- **Graph-based sampling.** Using original feature similarities as a negative sampling strategy requires computing similarities between each pair of nodes in the graph then sorting them and fine tuning the model to select the best values for the percentiles ω_k and ω_l. To avoid this, we propose to make use of the graph structure information to select negatives. Instead of considering distances between the nodes' original features, we use the distance between the nodes on the graph. For each node u, we simply sample negatives from its l-th order neighbors.

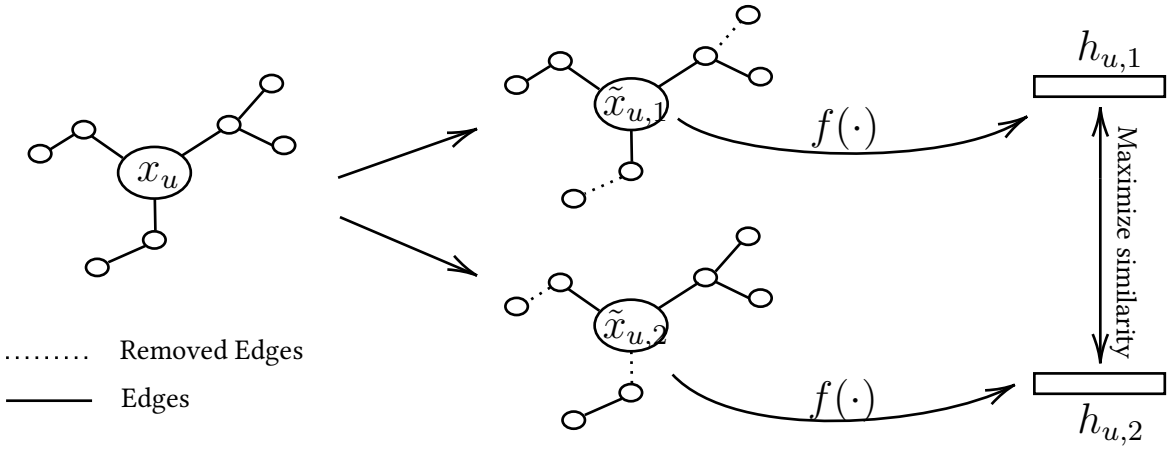


Figure 2.1: A high-level overview of our method for a subgraph around node u. $h_{u,1}$ and $h_{u,2}$ form a positive pair with a query $h_q = h_{u,1}$ and its corresponding key $h_q^+ = h_{u,2}$.

2.3.3 Overview of GraphCL

The training algorithm of GraphCL is summarized in the following steps:

1. Draw two stochastic perturbations t_1 and t_2 as defined in section 2.3.1 and illustrated in Figure 2.1. Apply them to nodes' original features and the graph structure:
   - $(\widetilde{X}_1, \widetilde{A}_1) \sim t_1(X, A)$
   - $(\widetilde{X}_2, \widetilde{A}_2) \sim t_2(X, A)$
2. Apply the encoder to both views of the graph:
   - $H_1^L = f(\widetilde{X}_1, \widetilde{A}_1)$
   - $H_2^L = f(\widetilde{X}_2, \widetilde{A}_2)$
3. Select negative examples as suggested in subsection 2.3.2.
4. Update parameters of the encoder f using the loss function defined in Eq. (2.3).

2.3.4 Extension to inductive setup

Unlike the transductive setup where we have access to the whole graph and features of all nodes of the graph during training time. In the inductive setup, the objective is to generate representations of nodes that were not used when training the model. These previously unseen nodes can be new nodes in an evolving graph such as a social network, we refer to this by the single graph inductive setup. These nodes can also come from previously unseen graphs which helps generalization across graphs with the same form of features, we refer to this by the multiple graph inductive setup.

GraphCL can be easily extended to both setups. The extension to the multiple graph setup is straightforward, as it consists of training the encoder on each of the available graphs for the training and using the learnt encoder to produce representations of nodes of the new graphs. For inductive learning on large graphs, we train the encoder by sampling minibatches of nodes. The training algorithm of GraphCL for the inductive setup is summarized in the following steps for each sampled minibatch $\mathcal{B}$:

1. For each node u in the minibatch we define (X_u, A_u) as the subgraph containing all nodes and edges that are at most L-hops from u in the graph and their corresponding features;
2. Draw two stochastic perturbations t_1 and t_2 as defined in section 2.3.1 and apply them to u's L-hop neighborhood subgraph:
   - $(\widetilde{X}_{u,1}, \widetilde{A}_{u,1}) \sim t_1(X_u, A_u)$
   - $(\widetilde{X}_{u,2}, \widetilde{A}_{u,2}) \sim t_2(X_u, A_u)$
3. Apply the encoder to the two representations of node u:
   - $h_{u,1} = f(\widetilde{X}_{u,1}, \widetilde{A}_{u,1})$
   - $h_{u,2} = f(\widetilde{X}_{u,2}, \widetilde{A}_{u,2})$
4. Update parameters of the encoder.

2.4 Experiments

We evaluate the effectiveness of GraphCL representations on both tranductive and inductive learning setups. The transductive learning setup consists of embedding nodes from a fixed graph (i.e. all node features and the entire graph structure are known during training time). On the other hand, the inductive learning setup consists of generating representation of unseen nodes or new graphs. Following common practice, we opt for a linear evaluation of the learned node representations. Specifically, we use these representations to train a logistic regression model to solve multiclass node classification tasks on five well-know benchmark datasets, three for the transductive learning setup and two for the inductive setup. We summarize the datasets and the baselines respectively in sections 2.4.1 and 2.4.1, provide model configuration and implementation details in section 2.4.1, and discuss the results in section 2.4.2.

2.4.1 Experimental setup

Datasets

For the transductive setting, we utilize Cora, Citeseer and Pubmed [73], three citation networks where nodes are bag of words representations of documents and edges correspond

Dataset	Task	Nodes	Edges	Features	Classes	Train/Val/Test Nodes
Cora	Transductive	2,708	5,429	1,433	7	140/500/1,000
Citeseer	Transductive	3,327	4,732	3,707	6	120/500/1,000
Pubmed	Transductive	19,717	44,338	500	3	60/500/1,000
ogbn-arxiv	Transductive	169,343	1,166,243	128	40	Time
Reddit	Inductive	231,443	11,606,919	602	41	151,708/23,699/55,334
PPI	Inductive	56,944 (24 graphs)	818,716	50	121 (multilabel)	44,906/6,154/5,524 (20/2/2 graphs)

Table 2.1: Description of datasets

to (undirected) citations. Each node belongs to one class. We also use ogbn-arxiv, which is another citation network, where nodes are computer science papers represented by a 128-dimensional feature vector obtained by averaging the embeddings of words in its title and abstract. Each node belongs to one of forty subject areas of arXiv CS papers [75].

On the other hand, a protein-protein interaction dataset (PPI) is used for the inductive setting on multiple graphs [101]. It consists of multiple graphs corresponding to different human tissues where node features are the positional gene sets, motif gene sets and immunological signatures. Each node has several labels among 121 labels from the gene ontology. For the inductive setting on large graphs, we use a Reddit dataset [38]. It represents a large social network where nodes correspond to Reddit posts (i.e. represented by their GloVe embedding [68]) and edges connecting two posts mean that the same user commented on them. Labels are the posts' *subreddit* and the objective is to predict the community structure of the social network.

Statistics of the datasets including data splits are given in table 3.5. For ogbn-arxiv dataset, we follow recommendations from the Open Graph Benchmark initiative and adopt a data split that is based on the publication dates of the papers [43]. More precisely, we train on papers published until 2017, validate on those published in 2018, and test on those published since 2019.

Baselines

For the transductive learning tasks, we use four unsupervised methods for comparison: Label Propagation (LP) [100], DeepWalk [69], Embedding Propagation (EP-B) [21], and Deep Graph Infomax (DGI) [80]. We also report the results of training logistic regression on the intrinsic input features only, and also on the concatenation of DeepWalk embeddings and the nodes' intrinsic features. Aside from unsupervised methods, we also compare our approach to strong supervised baselines, Graph Convolution Networks (GCN) [51].

For the inductive learning tasks, in addition to DeepWalk and DGI, we compare GraphCL with the unsupervised GraphSAGE methods [38]. We also provide results of two supervised approaches, FastGCN [10] and Gated Attention Networks (GaAN) [94].

Model configurations

Eq. (2.1) provides a general formulation of graph neural networks. Several architectures have been proposed for the choice of $AGGREGATE$ and $COMBINE$. In all our experiments the basic update rule is the mean pooling variant from [38].

$$h_u^{(l)} \leftarrow \left(W^{(l-1)}\right)^\top \cdot MEAN(\{h_u^{(l-1)}\} \cup \{h_v^{(l-1)}, \forall v \in \mathcal{N}(u)\}), \tag{2.5}$$

where the $MEAN$ operator is the element-wise mean of all vectors in $(\{h_u^{(l-1)}\} \cup \{h_v^{(l-1)}, \forall v \in \mathcal{N}(u)\})$, and $W^{(0)} \in \mathbb{R}^{P \times P'}$ and $W^{(l-1)} \in \mathbb{R}^{P' \times P'}$, for $l > 1$, are learnable linear transformations.

All GNN aggregation operations are computed in parallel resulting in a matrix representation as follows:

$$H^{(l)} = \hat{A} H^{(l-1)} W^{(l-1)} \tag{2.6}$$

where $H^{(l)} = [h_1^{(l)}, h_2^{(l)}, \ldots, h_N^{(l)}]^\top$ is the matrix of nodes' hidden feature vectors at the $l-$th layer and $\hat{A} = \check{D}^{-1}\check{A}$ is the normalized version of the adjacency matrix with added self-loop $\check{A} = A + I_N$ with $\check{D}$ being its diagonal degree matrix, i.e. $\check{D}_{ii} = \sum_j \check{A}_{ij}$. We also consider the symmetrically normalized version of the adjacency matrix where $\hat{A} = \check{D}^{-\frac{1}{2}}\check{A}\check{D}^{\frac{1}{2}}$. We refer to encoders using this variant by GCN when needed.

Transductive learning For Citeseer and Pubmed, we use a one layer GNN as defined in Eq. (2.6), the encoder is then simply expressed as:

$$f(X, A) = \hat{A} X W^{(0)} \tag{2.7}$$

For Cora, our encoder is a two-layer GNN:

$$f(X, A) = \hat{A}\sigma(\hat{A} X W^{(0)}) W^{(1)} \tag{2.8}$$

where σ is an exponential linear unit [15], and $f(X, A)$ is the concatenation of all nodes' embeddings. In each layer, we compute $P' = 512$ features resulting in a node embedding size of 512. For the larger ogbn-arxiv dataset, we use a three-layer GNN, and train the model by randomly sampling 1024 negative examples for each node.

Inductive learning For both inductive learning setups on large graphs and on multiple graphs , we use a three-layer mean-pooling encoder with residual units as follows:

$$H^{(1)} = \sigma(\hat{A} X W_1^{(0)} + X W_2^{(0)}) \tag{2.9}$$

$$H^{(2)} = \sigma(\hat{A} H^{(1)} W_1^{(1)} + H^{(1)} W_2^{(1)}) \tag{2.10}$$

$$f(X, A) = \hat{A} H^{(2)} W_1^{(2)} + H^{(2)} W_2^{(2)} \tag{2.11}$$

We set the hidden layers and the embedding size to $P' = 512$ and apply RELU as an activation function.

For the multiple-graph setting, we sample one graph at a time from the training set to train the contrastive loss function. For the single graph inductive setup, the scale of the dataset makes it impossible to fit into GPU memory. We therefore adopt the sub-sampling strategy of [38]. We first select a minibatch of nodes and construct for each of them their L-hop neighborhood subgraph by sampling a fixed size neighborhood. We sample 10 nodes in each of the three levels resulting in $1 + 10 + 100 + 1000 = 1111$ neighboring nodes .

We use Pytorch [66] and the Pytorch Geometric [24] libraries to implement all our experiments. We initialize all models using Glorot initialization [26] and trained them to minimize the contrastive loss provided in Eq. (2.3) using the Adam optimizer [50] with an initial learning rate of 0.001. We tune the weight decay in $\{0.001, 0.01, 0.05, 0.1, 0.15\}$. We further tune the temperature τ in the loss function in $\{0.1, 0.5, 0.8, 1.0\}$ and the number of epochs in $\{20, 50, 100, 150, 200\}$.

To define the stochastic perturbation, we tune the probability of dropping an edge in $[0.05, 0.75]$ and the probability of dropping node features in $[0.2, 0.8]$. GraphCL is found to be robust to different choices of the perturbation parameters. However, we found that applying high perturbations to node features (i.e. randomly dropping 50% to 70% of input features) and small perturbations of the graph structure (i.e. randomly dropping 10% to 20% of edges) results in stronger representations.

2.4.2 Results

We present the results of evaluating node representations using downstream multiclass node classification tasks in Table 2.2. We report average results over 50 runs of training followed by a logistic regression. Specifically, we use the mean classification accuracy on the test nodes for transductive tasks and the micro-averaged F1 score on the (unseen) test nodes for the inductive setting. We report the results of EP-B provided in [21] and [81], and also the results provided in [80]. To insure a fair comparison with the other methods, we report the results of the standard implementation of GraphCL which was described in the previous section. In particular we use a standard embedding size $P' = 512$. We refer to the results by **GraphCL** in table 2.2. We also report **GraphCL*** which refers to the results that were achieved using the best parameters including the best negative sampling strategy, choice of encoders and embedding size. For example, we notice a 1% gain on the classification accuracy of Cora when using a GCN encoder and sampling negative from the second order neighbors of the current example. Moreover, we notice a surprising 0.2 gain on the F1 score on PPI when increasing the embedding size to $P' = 2048$.

We see that the proposed GraphCL outperforms the previous state-of-the-art by achieving the best classification accuracy over the three transductive tasks and the best F1 score on inductive tasks. We note that, except for PPI dataset, GraphCL achieves competitive performance with strong supervised baselines without using label information. We assume that by maximizing agreement between representations that share the same information but have independent noise, GraphCL is able to learn representations that benefit from the richness of information in the graph which compensate for the information provided by the labels.

Transductive

Method	Cora	Citeseer	Pubmed	ogbn-arxiv
Raw features	$47.9 \pm 0.4\%$	$49.3 \pm 0.2\%$	$69.1 \pm 0.3\%$	$55.50 \pm 0.23\%$
DeepWalk [69]	67.2%	43.2%	65.3%	$70.07 \pm 0.13\%$
DeepWalk + features	$70.7 \pm 0.6\%$	$51.4 \pm 0.5\%$	$74.3 \pm 0.9\%$	_
EP-B [21]	$78.1 \pm 1.5\%$	$71.0 \pm 1.4\%$	$79.6 \pm 2.1\%$	$68 \pm 0.00\%$
DGI [80]	$82.3 \pm 0.6\%$	$71.8 \pm 0.7\%$	$76.8 \pm 0.6\%$	$70.18 \pm 0.12\%$
GraphCL	$\mathbf{83.6 \pm 0.5\%}$	$\mathbf{72.5 \pm 0.7\%}$	$\mathbf{79.8 \pm 0.5\%}$	$\mathbf{70.18 \pm 0.17\%}$
GraphCL*	$\mathbf{84.6 \pm 0.4\%}$	$\mathbf{73.1 \pm 0.6\%}$	$\mathbf{80.1 \pm 0.5\%}$	$\mathbf{71.38 \pm 0.13\%}$
GCN(supervised)[51]	81.5%	70.3%	79.0%	$71.74 \pm 0.002\%$

Inductive

	Method	Reddit	PPI
Unsupervised	Raw features	0.585	0.422
	GraphSage-GCN [38]	0.908	0.465
	GraphSage-mean [38]	0.897	0.486
	GraphSage-LSTM [38]	0.907	0.482
	GraphSage-pool [38]	0.892	0.502
	DGI [80]	0.940 ± 0.001	0.638 ± 0.002
	GraphCL	$\mathbf{0.951 \pm 0.01}$	$\mathbf{0.659 \pm 0.006}$
	GraphCL*	$\mathbf{0.960 \pm 0.01}$	$\mathbf{0.841 \pm 0.004}$
Supervised	FastGCN [10]	0.937	–
	GaAN [94]	0.958 ± 0.001	0.969 ± 0.002

Table 2.2: Classification accuracy on transductive tasks and micro-averaged F1 score on inductive tasks

2.4.3 Ablation study

We report on a study to understand the effects of different parameters. All experiments have been conducted using Cora dataset.

Effect of the number of negatives

Figure **??** shows the effect of the number of negatives on the accuracy of the downstream classification task. We find that training a contrastive loss with a small number of negatives leads to poor representations. However, our experiments show that at a certain threshold increasing the number of negatives does not improve the quality of the representations. Beyond that threshold the variations of the classification accuracy seem to be due to the

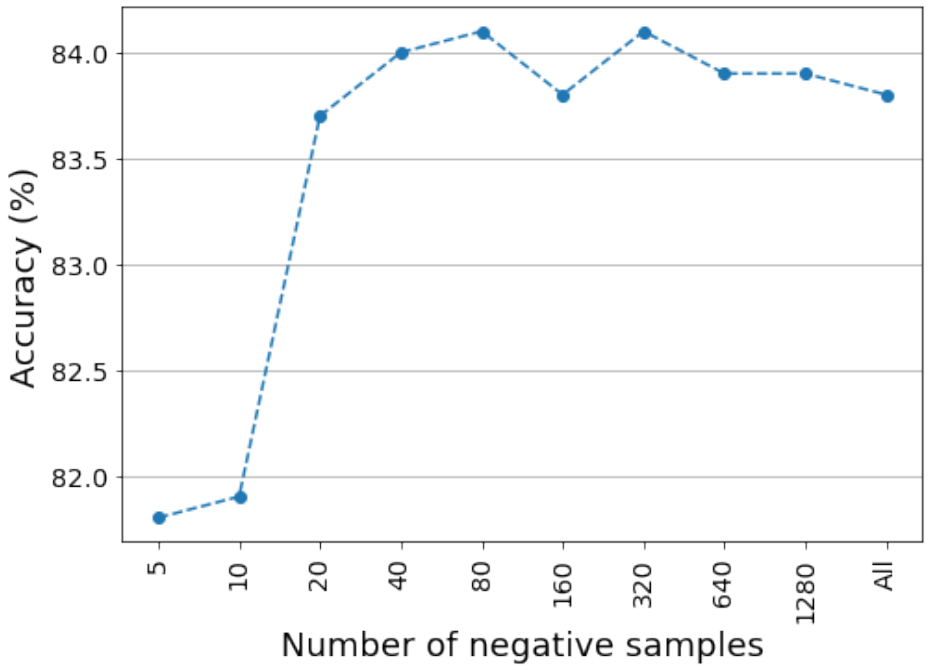


Figure 2.2: Classification accuracy on Cora dataset. Effect of the number of negative examples.

randomness of the training procedure only. Since using a large number of negatives slows down the training and requires more computing power, our findings suggest that one has to properly choose the number of negatives to optimize for both the quality of the representations and the training efficiency.

Effect of feature similarity based negative sampling strategies

We next analyse the effect of hard negative samples on the quality of the learned representations. We first implement the feature similarity based negative sampling strategy described in section 2.3.2. We select negatives from a *ring* around the current example. This is done by varying the values of the percentiles ω_l and ω_k. Figure **??** show the accuracy of a linear classifier trained on the learned representation while varying the distance of the *ring* from the current example. We select negatives from a ring of diameter 10% (i.e. $\omega_l - \omega_k = 10\%$). The results confirm our intuition that hard negatives improve the quality of the representations. We notice that selecting only negatives that are too far from the current example leads to poor representations. In fact, if all negatives are easy to distinguish from the current example, there is no reason for the encoder to learn higher level features that can help to distinguish between the corresponding positive example and all the negatives. On the other hand, selecting negatives from nodes that are very similar to the current example worsens the quality of the representations. This can be explained by the fact that negatives that are close to the current example are likely to belong to the same class and should rather be considered as positive examples. Training an encoder to push these examples away from the current example unsurprisingly leads to lower quality representations.

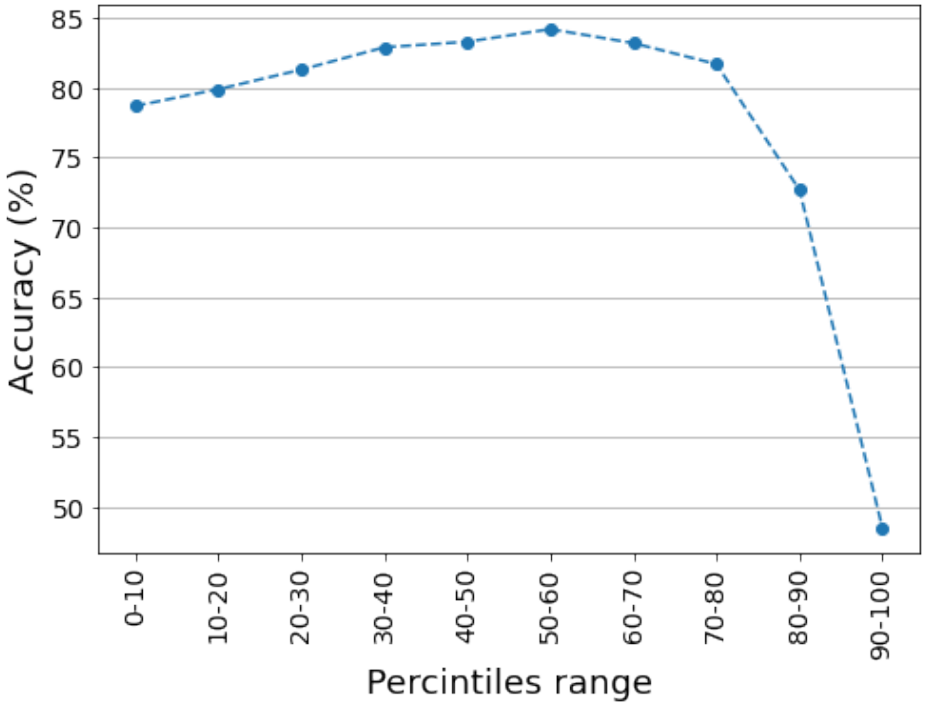


Figure 2.3: Classification accuracy on Cora dataset. Effect of the similarity between the current example and its corresponding negative samples.

Effect of graph based negative sampling strategies

l-th order neighbors	Accuracy
1	31.4 $\pm$ 1.2 %
2	**84.6 $\pm$ 0.4%**
3	80.8 $\pm$ 0.6 %

Table 2.3: Classification accuracy on Cora dataset. Effect of the number of hops between the current example and its corresponding negative samples.

Graphs provide additional information about the examples. We aim at taking advantage of the graph structure to sample negative examples. Table 2.3 shows the average accuracy of 50 runs of training to learn nodes' embeddings on top of which we apply a linear classifier. We sample negatives from the l-th order neighbors of the current example. Similarly to the results of the feature similarity based negative sampling strategy, we find that negatives that are at the right distance from the current example improve the quality of the learned representations. More specifically, we achieve the best performance when sampling negatives from the second order neighbors.

Encoder	Accuracy
MLP	66.1 %
Mean Pooling	83.6 %
GCN	**84.2 %**

Table 2.4: Classification accuracy on Cora dataset. Effect of the choice of the encoder

Training without negative samples

In the previous section, we have discussed the effect of negative sampling strategies on the quality of the learned representations by using them to linearly classify nodes on a multi-class classification downstream task. Here, we would like to see whether it is possible to learn meaningful representations by maximizing the similarities between the representations obtained from two views of the same graph without the use of any negative examples. To do so, we train a Siamese neural network to minimize the negative cosine similarity loss in Eq. (2.4). We implement the stop-gradient strategy described in section 2.2.4 and apply batch normalization on the hidden layer of the prediction MLP head (see section 2.2.4). Both stop-gradient and batch normalization have been reported to prevent the collapsing to a single representation when applied to Siamese neural networks for visual representations [14, 30]. Figures **??** and **??** respectively show the loss and accuracy of training a Siamese neural network without neither batch normalization nor stop-gradient referred to as *Siam*, with stop-gradient but without batch normalization referred to as *Siam+SG*, and with both stop-gradient and batch normalization referred to as *Siam+SG+BN*.

We observe that when training without stop-gradient and batch normalization, the loss function quickly converges to the minimum possible value -1. To verify that the cause is the collapsing to the single representation solution, we compute the standard deviation of all the representations which we found to be equal to zero for all features. We also notice that although adding stop-gradient and batch normalization prevent the collapsing to a single representation, the learned representations are still of low quality and perform much worse than the representations learned using negative samples.

2.5 Discussion

2.5.1 Connection to mutual information

The contrastive loss in Eq. (2.3) has been proposed as a lower bound estimator of the mutual information. A formal proof given by [78] shows that:

$$I(h_q, h_q^+) \geq \log(N) - \mathcal{L}, \tag{2.12}$$

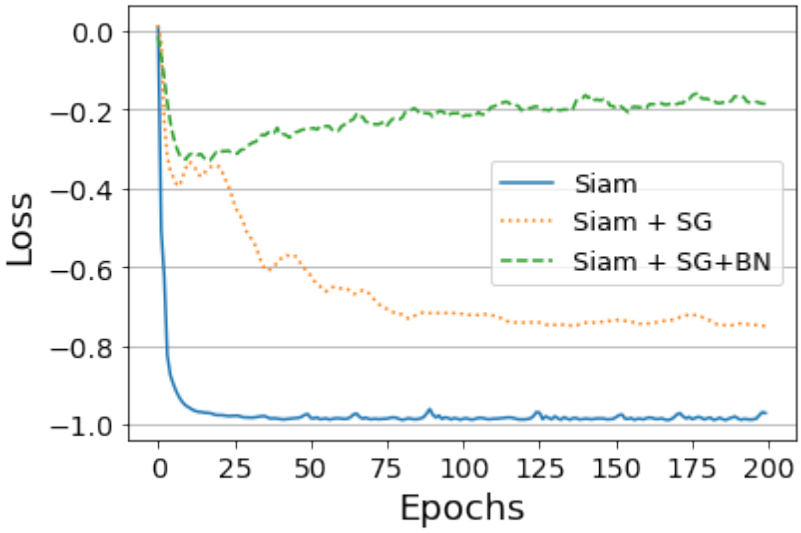


Figure 2.4: A comparison of Siamese NN trained *with* vs *without* stop-gradient and batch normalization. Training loss across epochs.

where N is the number of negative samples Q_q^- and $I(h_q, h_q^+)$ is the mutual information between h_q and h_q^+:

$$I(h_q, h_q^+) = \mathop{\mathbb{E}}_{(h_q, h_q^+) \sim p_{h_q, h_q^+}(\cdot)} \log \left[\frac{p(h_q, h_q^+)}{p(h_q) p(h_q^+)} \right] \tag{2.13}$$

where $p(h_q, h_q^+)$ is the joint distribution of h_q and h_q^+, and $p(h_q)$ and $p(h_q^+)$ are the corresponding marginals.

Therefore, given any N, minimizing the loss function $\mathcal{L}$ also maximizes the lower bound on the mutual information $I(h_q, h_q^+)$. We note however that it has been shown that the bound in Eq. (2.12) can be not tight . Our experiments suggest that contrastive methods' success highly depends on other parameter designs, and so cannot be solely attributed to the properties of the mutual information. This confirms the remark done in[59] where the bound in Eq. (2.12) was seen not to be tight. More precisely, results in Table 2.4.3 emphasize the impact of the choice of the encoder on the performance of the contrastive loss. The ablation study that we conducted also highlights the effect of the negative sampling strategy and the importance of hard negative examples for learning powerful representations.

2.5.2 Understanding contrastive learning through alignment and uniformity on the hypersphere

To better understand the behavior of GraphCL, we analyze it through the perspective of uniformity and alignment that has been introduced in [83]. The main idea behind the contrastive loss is to attracting positive pairs together in the representation space while pushing away the corresponding negative examples from the current sample. Eq. (2.3) actually encourages the learned representations to obey the following properties:

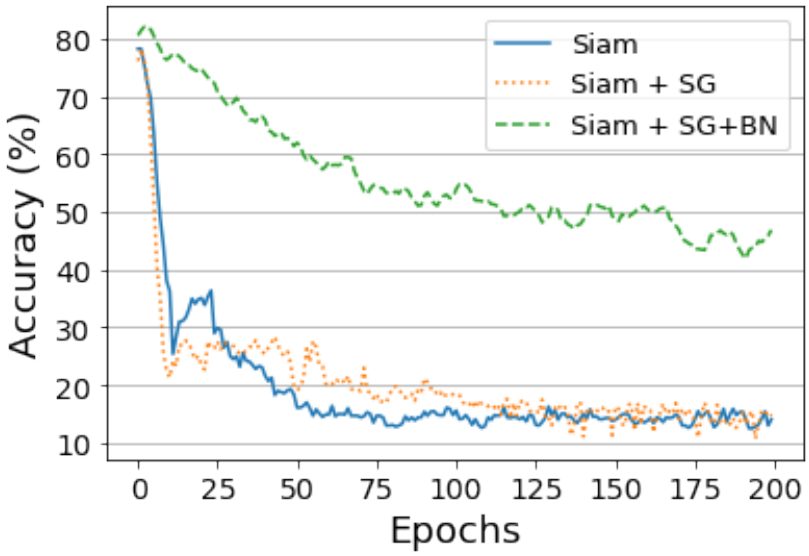


Figure 2.5: A comparison of Siamese NN trained *with* vs *without* stop-gradient and batch normalization. Accuracy of a linear classifier trained on top of the representation on Cora dataset

- **Alignment:** Representations of augmented views should be consistent and invariant to noise.
- **Uniformity:** The learned representations should match a prior distribution of high entropy (the uniform distribution over the hypersphere) to preserve as much information of the data as possible.

We visualize the learned representations of Cora dataset nodes in $\mathbb{R}^2$ (i.e $P' = 2$) to compare the behavior of the following methods:

- **GraphCL:** An encoder trained with the standard implementation of GraphCL as described above.
- **Siam+SG+BN:** A siamese neural network encoder trained with the negative cosine similarity loss using stop-gradient and batch normalization techniques.
- **Supervised GCN:** An encoder and a linear classifier trained jointly with a supervised cross entropy loss.

All encoders are 2-layers GCNs that map nodes to normalized feature vectors of dimension two. Figures 2.6 ,2.7 and 2.8 summarize the resulting distributions. GraphCL embeddings clearly display both properties. Positive pairs are more aligned than those learned using the negative cosine similarity and supervised loss with an average cosine similarity of 0.9 for GraphCL and 0.75 and 0.7 respectively for the other methods. Representations of GraphCL are also evenly distributed on the hypersphere and exhibit the most uniform distribution.

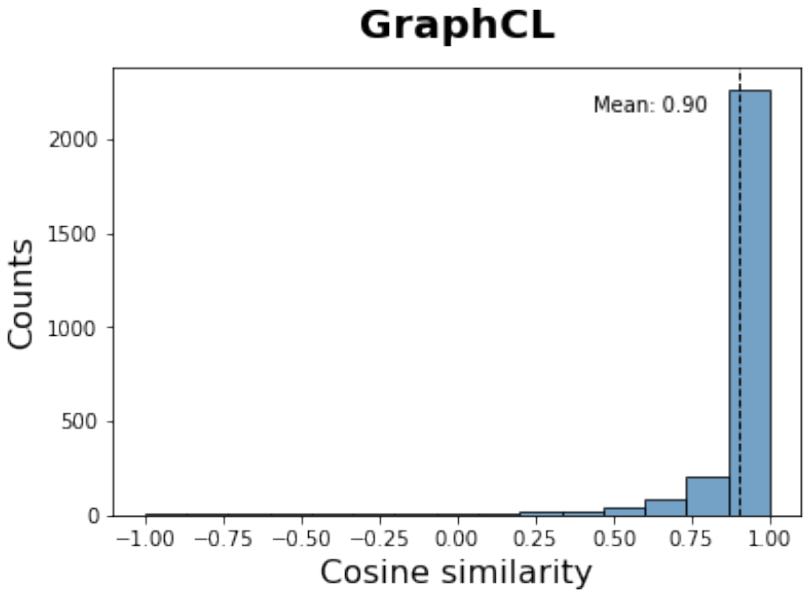


Figure 2.6: Representations of Cora dataset nodes on $\mathbb{R}^2$ using encoders trained with a contrastive loss. Histograms of the cosine similarity between positive pairs.

It has been shown in [83] that both the alignment and uniformity properties are important in learning highly transferable features to downstream tasks. This contributes to the the success of GraphCL and may explain its ability to outperform strong supervised baselines on nodes classification, especially on the transductive learning setup.

It is also worth noticing that although adding stop-gradient and batch normalization techniques to the training procedure of the negative cosine similarity loss (i.e. Siam+SG+BN in Figure **??**) prevent the collapsing to the single representation solution, the encoder fails to uniformly map the nodes' representations across the hypersphere. This explains its results on the classification downstream task (see Figure **??**).

2.5.3 Computational and model complexity

Last we discuss the computational and model complexity of GraphCL. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph and $N = |\mathcal{V}|$ the total number of nodes in the graph. Moreover, let L be the number of layers, M the minibatch size and R the number of neighbors being sampled for each node in the inductive setting. We assume for simplicity that the dimension of the nodes' hidden features is constant and denote it as P'. The computational complexity and space complexity of GraphCL depend on the choice of the encoder. We use the same encoder for the two branches (i.e. each of the subgraphs). For the transductive learning setup, the computational and space complexity are linear with respect to the number of nodes and are respectively $\mathcal{O}(LNP'^2)$ and $\mathcal{O}(LNP' + KP'^2)$. For the inductive learning, we use a sub-sampling strategy to load the graphs into memory; the computational complexity is then $\mathcal{O}(R^L NP'^2)$ and the space complexity is $\mathcal{O}(MR^L P' + LP'^2)$. The computational complexity is linear with respect to the number of nodes. Both the number of layers L and

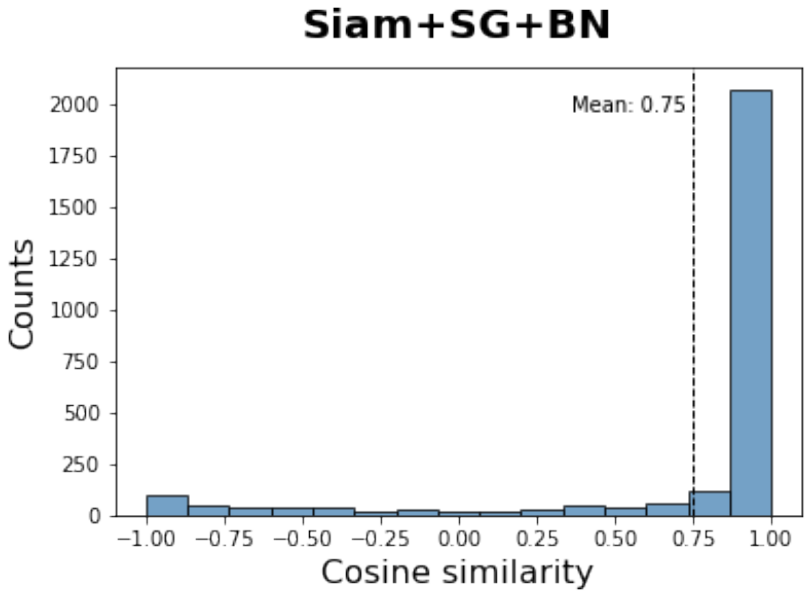


Figure 2.7: Representations of Cora dataset nodes on $\mathbb{R}^2$ using encoders trained with a negative cosine similarity loss. Histograms of the cosine similarity between positive pairs.

the number of sampled neighbors R are fixed and user-specified. The space complexity is linear with respect to the minibatch size M. The sampling strategy sacrifices time efficiency to save memory which is necessary for very large graphs.

2.6 Conclusion

We introduced GraphCL, a general framework for self-supervised learning of nodes' representations. The key idea of our approach is to maximize agreement between two representations of the same node. The representations are generated by injecting random perturbations to the graph structure and nodes' intrinsic features. We have conducted a number of experiments on both transductive and inductive learning tasks. Experimental results show that GraphCL outperforms state-of-the-art unsupervised baselines on nodes' classification tasks and is competitive with supervised baselines. We further investigated different negative sampling strategies including training with a similarity based loss without contrasting with negative samples and propose a graph based negative sampling strategy. In the future, we will investigate the potential of our approach in learning graphs' representations that are robust to adversarial attacks on the graph data and explore the reasons of the low quality of nodes' representations when training a siamese neural network without negative samples.

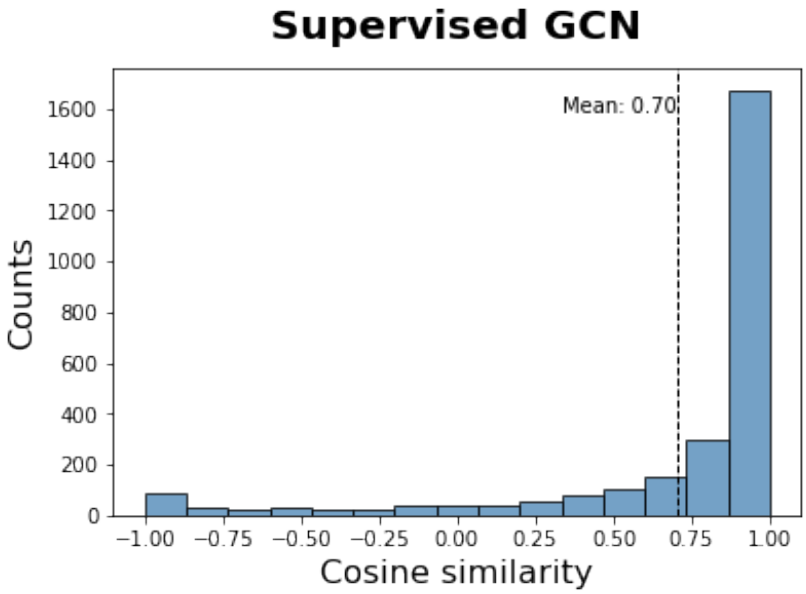


Figure 2.8: Representations of Cora dataset nodes on $\mathbb{R}^2$ using encoders trained with a supervised cross entropy loss. Histograms of the cosine similarity between positive pairs.

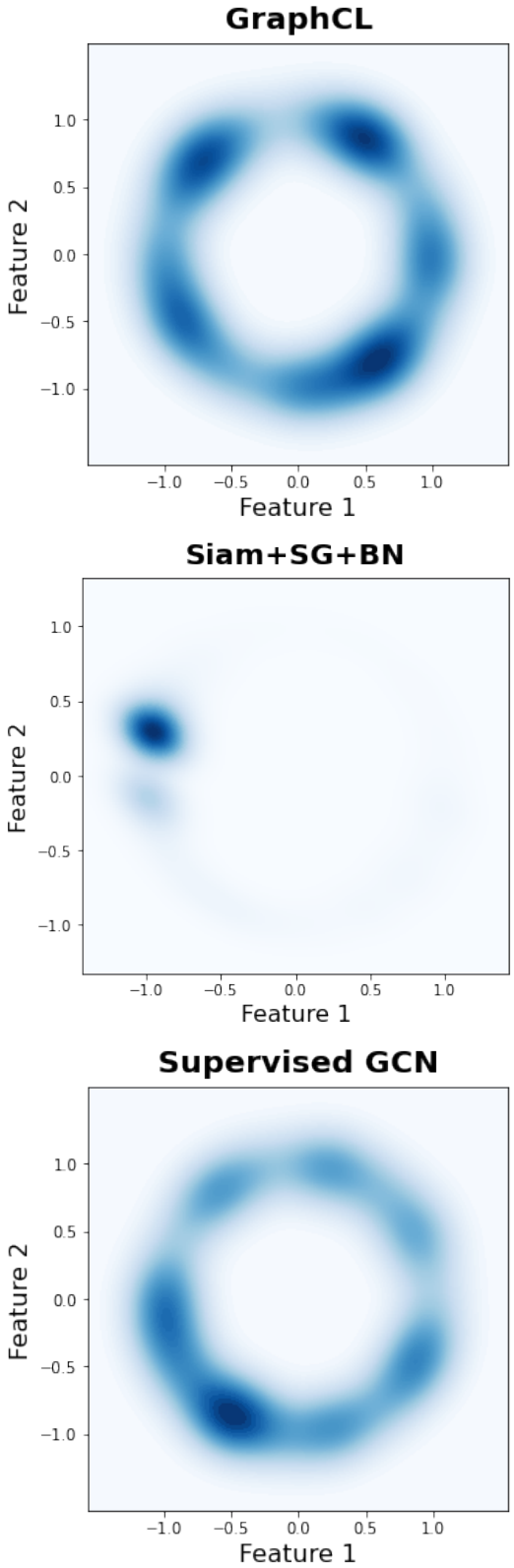


Figure 2.9: Representations of Cora dataset nodes on $\mathbb{R}^2$ using encoders trained with a contrastive loss (**upper plot**), a negative cosine similarity loss (**middle plot)** and a supervised cross entropy loss (**bottom plot**). Feature distributions in $\mathbb{R}^2$ using Gaussian kernel density estimation.

Chapter 3

Graph assisted bayesian classifiers

3.1 Introduction 42
3.2 New Graph-Assisted Bayesian Classifier 44
3.2.1 The Classifier derivation 44
3.2.2 Parameters' estimation 50
3.2.3 When does graph structure not help GAB 51
3.3 Link to Graph Neural Networks 53
3.3.1 Recap of Graph Neural Networks 53
3.3.2 Relationship with a GNN based classifier 54
3.4 Discussion on complexity issues 56
3.5 Numerical Results 59
3.5.1 Synthetic datasets 59
3.5.2 Real datasets 60
3.6 Conclusion 62

3.1 Introduction

Attributed graphs are useful tools for representing interactive phenomena such as social networks [4], financial market fluctuations [49], road or air traffic, human scene [72], brain activity [55], or gene interaction [54]. Graphs are described by *i)* a set of nodes associated with the entities (subscribers in social networks, planes in air traffic, etc.) and *ii)* a set of edges connecting them in order to represent their relationships. Graphs are said to be attributed when nodes and/or edges have assigned values, called *features*. While this type of data is rich for inference, it is not well suited to standard signal processing or machine learning techniques. The adaptation of these techniques to graph data has been the subject of numerous studies that aim to perform graph classification (e.g., molecules)[96] [25], edge prediction (e.g., social network links)[95], node regression or classification (e.g., citation networks) [35, 33, 36].

This works focuses on node classification, which is one of the most important tasks of machine learning on graphs. Its objective is to predict the class (called also *label*) of each unlabeled node of the graph by relying on both nodes' features and nodes' connections within the graph.

In most real world graphs, connections between nodes are far from arbitrary. In social networks for instance, people are more likely to connect with those who share similar characteristics or areas of interest [60]. A citation network has connections between articles if one cites another, so links between articles addressing the same research topic are more likely than links between those addressing different topics. [70]. As in social sciences literature, a *homophilic* graph can be described as one where similar nodes tend to be connected to each other [60]. Using this principle, one should make use of the topology of the graph to determine the class of a node, rather than relying solely on intrinsic features, as is commonly done with standard machine learning approaches.

When taking into account the graph topology in a classifier, the first question is: what kind of information to share over the graph? There are two approaches.

- Label Propagation (LP): only the labels (true or estimated) are propagated through the adjacent nodes in the graph to make a new decision. Several methods relying on voting have been developed to merge labels' information [100]. The labels at the initial step are either provided or estimated using the node's features only, i.e. ignoring the graph.
- Feature propagation (FP): the nodes' features are propagated through the adjacent nodes at each step. Typically, a weighted averaging of the current features of adjacent nodes (sometimes followed by a nonlinear function) is carried out for each node at each step.

In this work, we focus only on the FP approach. Graph neural networks (GNN) have recently been developed in order to adapt Neural Networks to attributed graphs. Typically, at each layer, they linearly combine the features of adjacent nodes and then apply an activation function. The training on labelled nodes consists of finding out the best weights

of each linear combination. In this work, we propose to derive in closed-form a classifier relying on (Bayesian) decision theory. As a result, we obtain an interpretable algorithm based on some parameters whose estimation step replaces the training phase of the GNN.

Before we go any further, let us briefly review the works on FP based on GNN. The most common way has been to extend the convolutional neural network (CNN) to the graph structure [41] by redefining the convolution operator on the graph domain. These CNN-based methods suffer from a high computational cost due to the necessity of performing an eigendecomposition of the graph Laplacian. To overcome this problem, different approaches avoid explicit computation of this eigendecomposition by using a polynomial expansion to represent the filters [18, 51]. More specifically, in [51], the authors consider a first-order polynomial approximation to build a neural network which they called Graph Convolutional Network (GCN) to do semi-supervised node classification. The GCN can also be seen as an aggregation operator, i.e. the representation of a node is obtained by averaging its intrinsic features with those of its first-order neighbors. The authors in [79, 77] respectively introduced Graph Attention Network (GAT) and Attention-based Graph Neural Network (AGNN) where different weights are assigned to neighbors based on nodes' and edges' features. Other researchers explored higher-order (or equivalently, multi-hop) information of the graph by repeatedly mixing features of neighbors at various distances [2] or by modifying the propagation strategy of GCN [53]. Although these approaches have achieved remarkable results on a number of benchmark datasets, we notice that their performance vary significantly across datasets. For instance, the gain compared to a simple logistic regression (i.e. no contribution from the neighbors) highly depends on the dataset.

We use the following example to explain the underlying reason for the performance variations over datasets. We define $\bar{p}$ as the average probability of intra-class connection (i.e., the probability that two nodes from the same class are connected), and $\bar{q}$ as the average probability of inter-class connection (i.e., the probability that two nodes from different classes are connected). The degree of *impurity* of the graph may be represented by the ratio $\bar{q}/\bar{p}$. Datasets with a degree of impurity less than 1 are called *assortative* [48] and correspond to graphs containing communities (nodes with similar features are connected to each other). In Table 3.1, we show the classification accuracies of a logistic regression classifier and a two-layer GCN for two widely-used datasets used for benchmarking GNN algorithms. We also display the estimated values for $\bar{p}$, $\bar{q}$ and the degree of impurity $\bar{q}/\bar{p}$. As expected, node classification using graph structure is easier with graphs offering low

Table 3.1: Intra- and inter-class connection probabilities and classification accuracies.

	Cora	**Citeseer**
Intra-class connectivity (p)	23×10^{-3}	12×10^{-3}
Inter-class connectivity (q)	5.5×10^{-3}	4.3×10^{-3}
Degree of Impurity (q/p)	0.23	0.36
Logistic Regression (LR)	56.0%	57.2%
Two-layer GCN [51]	81.5%	70.3%
Gain between GCN and LR	+45.5%	+22.9%

degree of impurity (like Cora). This may explain the performance variations over datasets.

In this work, a different point of view is taken by proposing a Bayesian classifier which does not rely on a neural network structure and is able to better adapt to the level of impurity than GNN-based classifiers.

Our approach to tackle the node classification problem is related to the so-called collective classification [8, 73] which refers to the classification of a set of connected nodes by using their intrinsic features and/or labels and their relative connections. Optimal collective classifications are carried out by maximizing the joint likelihood. However, optimal (and so exact) inference is an NP-hard issue in general, and is thus generally not well suited for the real-world networks. As a consequence, most collective classifiers rely on developing approximate inference [65, 8]. Some recent studies combine methods from collective classification with neural networks to ensure a better end-to-end learning, e.g. [45]. However, all the above-mentioned algorithms only make use of features of first-hop neighbors and thus rely on a propagation step to make use of higher-hop nodes' information in an iterative manner. We instead introduce a new classifier that directly takes into account higher-hop nodes. This has the additional advantage of being more interpretable.

Let us define some notations. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph where $\mathcal{V}$ is a set of nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges. Each node $u \in \mathcal{V}$ is represented by a feature vector $\boldsymbol{x}_u \in \mathbb{R}^{F\times 1}$ where F is the number of node's features. An adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N\times N}$ represents the topological structure of the graph where $N = |\mathcal{V}|$ is the number of nodes in the graph. Without loss of generality we assume the graph to be unweighted i.e $A_{u,v} = 1$ if $(u, v) \in \mathcal{E}$ and $A_{u,v} = 0$ otherwise. Let $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N]^\top$. Let y_i denote the label of the ith node and let K denote the number of classes, i.e. $y_i \in \{1, \ldots, K\}$.

The objective of node classification is to predict the class of all unlabeled nodes in the graph given the adjacency matrix $\boldsymbol{A}$, the feature matrix $\boldsymbol{X}$ and the set of available labels.

The chapter is organized as follows: in Section 3.2, we introduce our new graph-assisted Bayesian classifier (GAB). In Section 3.3, we compare our approach with GNN-based classifiers. In particular, we show our classifier has the shape of a GNN only in the noise-free case (i.e., $q = 0$) and under further conditions on the distribution of the features. In Section 3.4, complexity issues are discussed. In Section 3.5, numerical results are provided. Comparison with existing GNN-based methods are done on real datasets whose *degree of impurity* has been modified by injecting artificial noise (i.e., introducing fictitious edges between classes). We see that our proposed Bayesian classifier offers better performance than GNN-based classifiers. In Section **??**, concluding remarks are drawn.

3.2 New Graph-Assisted Bayesian Classifier

3.2.1 The Classifier derivation

In this section, we derive our new GAB classifier based on Maximum A Posteriori (MAP) principle. We develop this classifier for a node u, called *node of interest* in the rest of the chapter. Obviously, in practice, any node in the graph will be seen as a node of interest in a Round-Robin manner. We first consider the entire graph and then, in order to simplify the

derivations, we consider only the information provided by the hop distance between the node and a node of interest.

Let

- $\mathcal{V}_u$ be the set of nodes which will be involved in the classification of node u. Node u is not included in this set.
- $\mathcal{X}_u = \{\boldsymbol{x}_u\} \cup \{\boldsymbol{x}_v, v \in \mathcal{V}_u\}$ be the set of feature vectors of node u and its "helping" nodes.
- D_k be the distribution generating the feature vectors of nodes belonging to class k.

We do not assume that theses distributions are known. We instead either assume a shape for these distributions and then estimate their parameters, or approximate them using a neural network. The objective is to compute the posterior probability that a node u belongs to class k knowing information on the graph $\mathcal{I}_\mathcal{G}$ (typically its partial connectivity through the set $\mathcal{V}_u$), and $\mathcal{X}_u$. Consequently, the classifier makes the following decision

$$\hat{k}_u = \arg\max_k P_u(k).$$

where the posterior probability that needs to be computed is defined as:

$$P_u(k) = P(y_u = k|\mathcal{X}_u, \mathcal{I}_\mathcal{G}).$$

Using the Bayes' rule, we obtain

$$\begin{aligned} P_u(k) &= P(y_u = k|\mathcal{X}_u, \mathcal{I}_\mathcal{G}) \\ &= \frac{P(\mathcal{X}_u|y_u = k, \mathcal{I}_\mathcal{G})P(y_u = k|\mathcal{I}_\mathcal{G})}{P(\mathcal{X}_u|\mathcal{I}_\mathcal{G})} \\ &\propto P(\mathcal{X}_u|y_u = k, \mathcal{I}_\mathcal{G})P(y_u = k) \end{aligned} \tag{3.1}$$

since the denominator does not depend on k and the prior probability of an individual node u to belong to class k does not depend on the graph information. The above posterior probability can be rewritten as

$$P_u(k) \propto Q_u(k)\pi_k \tag{3.2}$$

with

$$\begin{cases} Q_u(k) &= P(\mathcal{X}_u|y_u = k, \mathcal{I}_\mathcal{G}), \\ \pi_k &= P(y_u = k). \end{cases}$$

Let V be the size of the set $\mathcal{V}_u$. Let $\{v_1, \cdots, v_V\}$ be the nodes of $\mathcal{V}_u$. In Appendix A.1, we show that

$$\begin{aligned} Q_u(k) &= D_k(\boldsymbol{x}_u) \sum_{k_{v_1}, \cdots, k_{v_V}=1}^{K} \prod_{\ell=1}^{V} D_{k_{v_\ell}}(\boldsymbol{x}_{v_\ell}) \\ &\times p(y_{v_1} = k_{v_1}, \cdots, y_{v_V} = k_{v_V}|y_u = k, \mathcal{I}_\mathcal{G}). \end{aligned} \tag{3.3}$$

Eq. (3.3) cannot be simplified further since the term $p(y_{v_1} = k_{v_1}, \cdots, y_{v_V} = k_{v_V}|y_u = k, \mathcal{I}_\mathcal{G})$ cannot be split into individual posterior probabilities. Indeed according to Example 1 below, one can see that in general $p(y_{v_1} = k_{v_1}, \cdots, y_{v_V} = k_{v_V}|y_u = k, \mathcal{I}_\mathcal{G}) \neq \prod_{\ell=1}^{V} p(y_{v_\ell} = k_{v_\ell}|y_u = k, \mathcal{I}_\mathcal{G})$.

Example 1 *Let us consider the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{A, B, C\}$ and $\mathcal{E} = \{(B, C)\}$. We set the number of classes to $K = 3$. We assume that only nodes belonging to the same class can be connected. We now compute $P(y_B = k, y_C = k'|y_A = 1, \mathcal{I}_\mathcal{G})$ in two different cases.*

In the first case, we consider that $\mathcal{I}_\mathcal{G}$ provides only information on the connections between the pairs (A, B) and (A, C). Consequently, we only know A is not connected to B and A is not connected to C. This leads to the following expression for $k \in \{2, 3\}$

$$P(y_B = k|y_A = 1, \mathcal{I}_\mathcal{G}) = P(y_C = k|y_A = 1, \mathcal{I}_\mathcal{G}) = 0.5, \tag{3.4}$$

and

$$P(y_B = k, y_C = k'|y_A = 1, \mathcal{I}_\mathcal{G}) = 0.25 \tag{3.5}$$

which is equal to $P(y_B = k|y_A = 1, \mathcal{I}_\mathcal{G}) \times P(y_C = k'|y_A = 1, \mathcal{I}_\mathcal{G})$.

In the second case, we consider that $\mathcal{I}_G$ provides information on all possible connections. Once again, for $k \in \{2, 3\}$

$$P(y_B = k|y_A = 1, \mathcal{I}_\mathcal{G}) = P(y_C = k|y_A = 1, \mathcal{I}_\mathcal{G}) = 0.5. \tag{3.6}$$

But we now have for $k \neq k' \in \{2, 3\}$

$$P(y_B = k, y_C = k'|y_A = 1, \mathcal{I}_\mathcal{G}) = 0, \tag{3.7}$$

which is different from $P(y_B = k|y_A = 1, \mathcal{I}_\mathcal{G}) \times P(y_C = k'|y_A = 1, \mathcal{I}_\mathcal{G})$. □

Consequently, in order to pursue analytical derivations offering practical algorithms, we make the following simplifying assumption:

$$\begin{aligned} & p(y_{v_1} = k_{v_1}, \cdots, y_{v_V} = k_{v_V}|y_u = k, \mathcal{I}_\mathcal{G}) \\ = & \prod_{\ell=1}^{V} p(y_{v_\ell} = k_{v_\ell}|y_u = k, \mathcal{I}_\mathcal{G}) \end{aligned} \tag{3.8}$$

which corresponds to assuming statistical independence between the classes of node u's neighboring nodes given the graph. As seen in Example 1, this independence generally does not hold true, but it is required to pursue closed-form derivations.

Using Eqs. (3.3) and (3.8), we obtain

$$\begin{aligned} Q_u(k) &= D_k(\boldsymbol{x}_u) \sum_{k_{v_1},\cdots,k_{v_V}=1}^{K} \prod_{\ell=1}^{V} D_{k_{v_\ell}}(\boldsymbol{x}_{v_\ell}) \\ &\times \prod_{\ell'=1}^{V} p(y_{v_{\ell'}} = k_{v_{\ell'}} | y_u = k, \mathcal{I}_\mathcal{G}) \\ &= D_k(\boldsymbol{x}_u) \sum_{k_{v_1},\cdots,k_{v_V}=1}^{K} \prod_{\ell=1}^{V} D_{k_{v_\ell}}(\boldsymbol{x}_{v_\ell}) \\ &\times p(y_{v_\ell} = k_{v_\ell} | y_u = k, \mathcal{I}_\mathcal{G}) \\ &= D_k(\boldsymbol{x}_u) \\ &\times \prod_{\ell=1}^{V} \left(\sum_{k'=1}^{K} p(y_{v_\ell} = k' | y_u = k, \mathcal{I}_\mathcal{G}) D_{k'}(\boldsymbol{x}_{v_\ell}) \right). \end{aligned}$$

Finally, we get

$$Q_u(k) = D_k(\boldsymbol{x}_u) \prod_{\ell=1}^{V} \left(\sum_{k'=1}^{K} r_{u,v_\ell}(k,k') D_{k'}(\boldsymbol{x}_{v_\ell}) \right) \tag{3.9}$$

with

$$r_{u,v}(k,k') = p(y_v = k' | y_u = k, \mathcal{I}_\mathcal{G}). \tag{3.10}$$

The term $r_{u,v}(k,k')$ is the probability to be in class k' for node v given that node u is in class k and that we have partial (or total) information on the graph $\mathcal{I}_\mathcal{G}$. Notice that the term $r_{u,v}(k,k')$ depends on the graph statistics as we will see in Eq. (3.13). In general, deriving a closed-form expression of $r_{u,v}(k,k')$ with respect to the graph statistics is very difficult due to combinatorial complexity.

Before going further, we make the following remark.

Remark 1 (How to take into account the labelled nodes?) *Some of the nodes in $\mathcal{V}_u$ may have already been labeled, so their classes are known. Eq.* (3.9) *should then be adapted to account for this knowledge. Let us consider that node v_1 is labelled and belongs to class k_1. The following term*

$$\sum_{k'=1}^{K} r_{u,v_1}(k,k') D_{k'}(\boldsymbol{x}_{v_1})$$

should be replaced with

$$\sum_{k'=1}^{K} r_{u,v_1}(k,k') \delta_{k',k_1} = r_{u,v_1}(k,k_1)$$

where δ is the so-called Kronecker index. Consequently, if $\mathcal{V}_u = \mathcal{L}_u \cup \mathcal{U}_u$ with $\mathcal{L}_u$ being the set

of labelled nodes and $\mathcal{U}_u$ being the set of unlabelled nodes, we have that

$$\begin{aligned} Q_u(k) &= D_k(\boldsymbol{x}_u) \prod_{v \in \mathcal{L}_u} r_{u,v}(k, k_v) \\ &\times \prod_{v \in \mathcal{U}_u} \left(\sum_{k'=1}^{K} r_{u,v}(k, k') D_{k'}(\boldsymbol{x}_v) \right) \end{aligned}$$

where k_v is the class of the labelled node v.

As an example, we consider $\mathcal{L}_u = \{v_1\}$ and $\mathcal{U}_u = \emptyset$, and obtain

$$Q_u(k) = D_k(\boldsymbol{x}_u).r_{u,v_1}(k, k_1),$$

which is the likelihood function for $\boldsymbol{x}_u$ assuming class k, corrected by a term depending on the probability of connection between class k and that of the labelled neighbor node.

In order to obtain a practical algorithm, $\mathcal{I}_\mathcal{G}$ for each node of interest, u, will be made to consist of only the distances between this node and other nodes of the graph, i.e.,

$$\mathcal{I}_\mathcal{G} = \{\text{distance of each node of } \mathcal{G} \text{ to root } u\}.$$

Therefore, we order $\mathcal{V}_u$ as follows

$$\mathcal{V}_u = \mathcal{N}_1(u) \cup \mathcal{N}_2(u) \cdots \cup \mathcal{N}_{\Delta_u}(u)$$

where $\mathcal{N}_d(u)$ is the set of d-hops neighbors of u and Δ_u is the maximum distance from node u; we have that $\Delta_u \leq V$. Hence, Eq. (3.9) can be rewritten as

$$\begin{aligned} Q_u(k) &= D_k(\boldsymbol{x}_u) \\ &\times \prod_{d=1}^{\Delta_u} \prod_{v \in \mathcal{N}_d(u)} \left(\sum_{k'=1}^{K} r_{u,v}(k, k') D_{k'}(\boldsymbol{x}_v) \right). \end{aligned}$$

The above expression will be next simplified further.

We recall that $r_{u,v}(k, k')$ is equal to $p(y_v = k' | y_u = k, \mathcal{I}_\mathcal{G})$. As $\mathcal{I}_\mathcal{G}$ now merely consists of the distances between each node and the node of interest, $r_{u,v}(k, k')$ no longer depends on the specific path(s) connecting u to v but only hop distance, between node v and node u. Moreover we assume that the probability of connection between two nodes only depends on the nodes' classes but not on the specific nodes. Consequently, $r_{u,v}(k, k')$ is replaced with $r^{(d)}(k, k')$.

To control the contributions of neighbors to the computation of the posterior probability, depending on their distance to node u, we propose to modify the expression of $Q_u(k)$ as follows

$$Q_u(k) = D_k(\boldsymbol{x}_u) \prod_{d=1}^{\Delta_u} \prod_{v \in \mathcal{N}_d(u)} \left(\sum_{k'=1}^{K} r^{(d)}(k, k')^{\gamma_d} D_{k'}(\boldsymbol{x}_v) \right). \tag{3.11}$$

where $\{\gamma_d\}_{d=1,\cdots,\Delta_u}$ are hyper-parameters to tune. Simulation results show that hyper-parameter γ_d decreases with d, as expected. Setting $\gamma_d = 1$ for all values of d implies

that all the nodes in $\mathcal{V}_u$ have the same weights and so contribute equally. This can be counter-productive especially when the degree of impurity is high. Indeed, information from distant nodes may not be reliable, mostly because of the simplifications made to $\mathcal{I}_\mathcal{G}$. The optimization of the hyperparameters should counter-act this kind of phenomenon.

The goal now is to derive $r^{(d)}(k, k')$. Let $\mathbf{R}^{(d)} \in \mathbb{R}^{K\times K}$ be the matrix whose (k, k')th element is $r^{(d)}(k, k')$. It is worth pointing out that, in general, $\mathbf{R}^{(1)}$ is not symmetric but is non-negative with the row-sums equal to 1 (while the column-sums are not necessary equal to 1). Consequently, $\mathbf{R}^{(1)}$ (shortened to $\mathbf{R}$) is a row-stochastic matrix. First of all, we have that

$$r^{(d)}(k, k') = P(y_v = k'|y_u = k, \mathcal{C}_d)$$

where $\mathcal{I}_\mathcal{G}$ has been replaced with $\mathcal{C}_d$, which is defined as the knowledge that both considered nodes are connected in d hops.

In Appendix A.2, we show that

$$\mathbf{R}^{(d)} = \mathbf{R}^d. \tag{3.12}$$

Since we are able to find $r^{(d)}(k, k')$ with respect to $r^{(1)}(k, k')$, we should now derive $r^{(1)}(k, k')$ in closed-form. We first define the following parameters:

- Let $p(k)$ denote the probability that any two randomly selected nodes belonging to the same class k are directly connected.
- Let $q(k, k')$ denote the probability that two randomly selected nodes not belonging to the same class are connected. We let $q(k, k) = p(k)$. We also assume symmetry, i.e. $q(k, k') = q(k', k)$.
- Let $\bar{p}$ define the average probability of connection between nodes of the the same class, i.e.

  $$\bar{p} = \frac{1}{K}\sum_{k=1}^{K} p(k).$$

  Similarly, let $\bar{q}$ define the average probability of connection between nodes not belonging to the same class, i.e.

  $$\bar{q} = \frac{1}{K(K-1)}\sum_{\substack{k,k'=1\\k\neq k'}}^{K} q(k, k').$$

- The degree of impurity (DoI), roughly evoked in Section 3.1, is defined as

  $$\text{DoI} = \frac{\bar{q}}{\bar{p}}.$$

This defines a general stochastic block model (SBM), a widely used random graph model for community detection and clustering [1, 67]. Note that we use SBM only for analytic

tractability, and that unlike the work on community detection, we are interested in node classification, given some labeled samples.

By using Bayes' rule, we have that

$$\begin{aligned} r^{(1)}(k,k') &= P(y_B = k' | y_A = k, \widetilde{\mathcal{I}}_{\mathcal{G}}) \\ &= \frac{P(\widetilde{\mathcal{I}}_{\mathcal{G}} | y_B = k', y_A = k) P(y_B = k' | y_A = k)}{P(\widetilde{\mathcal{I}}_{\mathcal{G}} | y_A = k)} \\ &= \frac{P(\widetilde{\mathcal{I}}_{\mathcal{G}} | y_B = k', y_A = k) P(y_B = k' | y_A = k)}{\sum_{k''=1}^{K} P(\widetilde{\mathcal{I}}_{\mathcal{G}} | y_A = k, y_B = k'') P(y_B = k'')} \\ &= \frac{q(k,k') P(y_B = k' | y_A = k)}{\sum_{k''=1}^{K} q(k,k'') P(y_B = k'')} \\ &= \frac{q(k,k') P(y_B = k')}{\sum_{k''=1}^{K} q(k,k'') P(y_B = k'')} \\ &= \frac{q(k,k') \pi_{k'}}{\sum_{k''=1}^{K} q(k,k'') \pi_{k''}}. \end{aligned} \tag{3.13}$$

As an example, if the probabilities of connection do not depend on the classes, i.e., $p = p(k)$ for any k and $q = q(k,k')$ for any $k \neq k'$, we obtain

$$r^{(1)}(k,k') = \begin{cases} \frac{p\pi_k}{p\pi_k + q \sum_{\substack{k''=1 \\ k'' \neq k}} \pi_{k''}} & k = k' \\ \frac{q\pi_{k'}}{p\pi_k + q \sum_{\substack{k''=1 \\ k'' \neq k}} \pi_{k''}} & k \neq k' \end{cases}.$$

Hence, according to Eqs. (3.2) and (3.11), we obtain

$$\begin{aligned} \hat{k}_u = \arg\max_k \; & \pi_k D_k(\boldsymbol{x}_u) \prod_{d=1}^{\Delta_u} \prod_{v \in \mathcal{N}_d(u)} \\ & \times \left(\sum_{k'=1}^{K} r^{(d)}(k,k')^{\gamma_d} D_{k'}(\boldsymbol{x}_v) \right) \end{aligned} \tag{3.14}$$

with $r^{(d)}(k,k')$ given by Eq. (3.12). We notice that the shape of the function to maximize is simple and corresponds to a product over all the considered nodes of a weighted sum of the possible distributions. The weights are perfectly characterized thanks to our derivations.

3.2.2 Parameters' estimation

In order to implement our node classifier, i.e. to compute Eq. (4.1), we need π_k, $p(k)$, $q(k,k')$, and the distributions $D_k(\cdot)$ for all k and all k'. Since these are unknown, they have to be estimated using the data. Here, in order to perform this estimation using simple algorithms, we consider only the available labeled nodes of the graph in the estimation. These algorithms are described below:

- Estimation of π_k: this is obtained by counting the number of labeled nodes belonging to class k divided by the total number of labeled nodes.
- Estimation of $p(k)$ and $q(k,k')$: we count all the pairs of labeled nodes belonging to class k and k'; then we count the number of these pairs that are connected in 1-hop. The estimate of $\hat{q}(k,k')$ is obtained by dividing the latter count by the former one. if these values do not depend on k and k', we also average over all the involved pairs (with $k \neq k'$ to obtain $q(k,k') = \bar{q}$ and when $k = k'$ to obtain $q(k,k) = \bar{p}$).
- Estimation of classes' statistics $D_k(\cdot)$: we assume a shape for the distribution , and this shape is dependent on a set of parameters. For instance, in the Cora dataset, the features are binary, so they are modeled as independent Bernoulli random variables; the probability associated with each feature is estimated using the proportion of non-zero elements of this feature in the labeled nodes, and Laplace smoothing. If the features are continuous-valued, we use the Gaussian distribution.

It is worth pointing out that a semi-supervised estimation approach may also be possible but this would require an iterative approach that cycles between parameter estimation and node classification. It is also worth noting that the estimation step plays the role of the learning phase in GNN-based classifiers or in the classifiers developed in [45].

The values of the hyperparameters γ_v are set to 1 by default. The optimization of these hyperparameters is addressed in Sections 3.4 and 3.5.
This optimization will be shown to improve classification performance. Indeed, these hyperparameters will adjust the degree to which neighbors of different orders should contribute to the classification of a node.

3.2.3 When does graph structure not help GAB

Thanks to Eq. (4.1), we will be able to characterize some conditions on the graph's parameters for which the graph through the proposed GAB helps each node to improve its classification performance. Reasoning by contradiction, one can see that the classifier does not take into account the neighbors if and only if (iff) the function

$$k \mapsto \prod_{v \in \mathcal{N}_d(u)} \left(\sum_{k'=1}^{K} r^{(d)}(k,k')^{\gamma_v} D_{k'}(\boldsymbol{x}_v) \right)$$

is independent of k for any d. Indeed, if true, the a posteriori distribution to maximize depends on k only through $D_k(\boldsymbol{x}_u)$.

The above-mentioned function leads to the same output regardless of k for any feature values iff the weights $r^{(d)}(k,k')$ are identical for any k. Therefore the condition for the neighbors to be useless is

$$r^{(d)}(k,k') = t^{(d)}(k'), \forall k. \tag{3.15}$$

According to Eq. (3.12), it is easy to prove that if it is satisfied for $d = 1$, then it remains true for any d. Therefore, we need to only focus on $d = 1$. According to Eqs. (3.13) and

(3.15), we obtain

$$\frac{q(k,k')\pi_{k'}}{\sum_{k''=1}^{K} q(k,k'')\pi_{k''}} = t^{(1)}(k'), \forall k. \tag{3.16}$$

We will now inspect some particular cases:

- In the case of constant intra-class and inter-class probabilities of connection, we obtain the following constraint for any pair (k,k') such that $k \neq k'$

$$\pi_{k'} + (\bar{q}/\bar{p}) \sum_{\substack{k''=1 \\ k'' \neq k'}}^{K} \pi_{k''} = (\bar{p}/\bar{q})\pi_k + \sum_{\substack{k''=1 \\ k'' \neq k}}^{K} \pi_{k''}.$$

  By setting $\nu = \sum_{k=1}^{K} \pi_k$, we have

$$\pi_{k'}(1 - \bar{q}/\bar{p}) + \pi_k(1 - \bar{p}/\bar{q}) = \nu(1 - \bar{q}/\bar{p})$$

  which implies that

$$\underline{\pi}_{k'} + \underline{\pi}_k \frac{1 - \bar{p}/\bar{q}}{1 - \bar{q}/\bar{p}} = 1 \tag{3.17}$$

  where $k \mapsto \underline{\pi}_k := \pi_k/\nu$ is a probability mass function. As $(1 - \bar{p}/\bar{q})/(1 - \bar{q}/\bar{p})$ is negative, this equation does not hold except if $\bar{p} = \bar{q}$. Consequently, the neighbors are not involved in the GAB classifier when the degree of impurity is 1 since there is no community structure.

- In the case of $K = 2$, Eq. (3.16) implies that

$$\frac{p(2)}{\bar{q}\pi_1 + p(2)\pi_2} = \frac{\bar{q}}{p(1)\pi_1 + q\pi_2}$$

  and

$$\frac{\bar{q}}{\bar{q}\pi_1 + p(2)\pi_2} = \frac{p(1)}{p(1)\pi_1 + \bar{q}\pi_2},$$

  which leads to

$$\bar{q} = \sqrt{p(1)p(2)}. \tag{3.18}$$

  In this setup, the neighbors are not involved in the GAB classifier when the inter-class probability of connection is the geometric mean of the intra-class probabilities of connection, and so not necessary when the degree of impurity (defined through the arithmetic mean) is equal to 1. Obviously, if $p(1) = p(2)$, we go back to the first item leading to $p = q$, i.e., a degree of impurity equal to 1. But when $p(1) \neq p(2)$, the degree of impurity leading to a graph-agnostic classifier is equal to the ratio between the geometric mean and the arithmetic mean of the intra-class probabilities of connection which is strictly smaller than 1 in general case.

3.3 Link to Graph Neural Networks

3.3.1 Recap of Graph Neural Networks

GNNs are a class of graph embedding architectures which use the graph structure in addition to node and edge features to generate a representation vector (i.e., embedding) for each node. GNNs learn node representations by aggregating the features of neighboring nodes and edges. The output of the ℓ-th layer of these GNNs is generally expressed as:

$$\boldsymbol{h}_u^{(\ell)} = \sigma^{(\ell)}(\phi^{(\ell)}(\boldsymbol{h}_u^{(\ell-1)}, \{\boldsymbol{h}_v^{(\ell-1)} : v \in \mathcal{N}_1(u)\})) \tag{3.19}$$

where $\boldsymbol{h}_u^{(\ell)}$ is the feature vector of node u at the ℓ-th layer initialized by $\boldsymbol{h}_u^{(0)} = \boldsymbol{x}_u$ and $\mathcal{N}_1(u)$ is the set of first-order neighbors of node u. Different GNNs use different formulations for the non-linear function $\sigma^{(\ell)}$ (called activation function) and the linear function $\phi^{(\ell)}$ [38]. Note that a first-order GNN based classifier relies on one layer or equivalently considers only the 1-hop neighborhood in the graph.

Graph Convolutional Neural Network (GCN): The convolutional propagation rule used in GCN is defined as follows

$$\phi^{(\ell)} = (\boldsymbol{W}^{(\ell)})^\top \left(\frac{\boldsymbol{h}_u^{(\ell-1)}}{d_u + 1} + \sum_{v \in \mathcal{N}_1(u)} \frac{\boldsymbol{h}_v^{(\ell-1)}}{\sqrt{(d_u+1)(d_v+1)}} \right) \tag{3.20}$$

where

- $\boldsymbol{W}^{(\ell)}$ is a learnable weight matrix,
- d_u is the degree of node u.

The activation function (for any layer except the last one) is a rectified linear unit (ReLU). For the last layer, we consider the $\mathrm{softmax}$ which for each node u outputs the probability that node u belongs to class k. Then the node u is assigned to the class with the highest probability [51].

Graph convolution Operator Network (GON): In [3, 63], GON is defined as GCN where Eq. (3.20) is replaced with the following one

$$\phi_u^{(\ell)} = (\boldsymbol{W}_1^{(\ell)})^\top \boldsymbol{h}_u^{(\ell-1)} + (\boldsymbol{W}_2^{(\ell)})^\top \Big(\sum_{v \in \mathcal{N}_1(u)} \boldsymbol{h}_v^{(\ell-1)} \Big). \tag{3.21}$$

Unlike GCN, GON computes a transformation matrix of the central node that is different from the transformation of its neighbors.

Graph Isomorphism Network (GIN): In [89], GIN is defined as GCN or GON where Eqs.(3.20)-(3.21) are replaced with the following one

$$\phi_u^{(\ell)} = (\boldsymbol{W}^{(\ell)})^\top \left((1+\alpha)\boldsymbol{h}_u^{(\ell-1)} + \sum_{v \in \mathcal{N}_1(u)} \boldsymbol{h}_v^{(\ell-1)} \right). \tag{3.22}$$

where α is a positive hyper-parameter. GIN thus attributes a different learnable weight to the central node (through α) when combining information from its neighbors.

Graph Attention Network (GAT): In [81], GAT is defined as GCN, GON or GIN but with the following layer link

$$\phi_u^{(\ell)} = \sum_{v \in \mathcal{N}_1(u) \cup \{u\}} \alpha_{u,v}^{(\ell)} (\boldsymbol{W}^{(\ell)})^\top \boldsymbol{h}_v^{(\ell-1)}, \tag{3.23}$$

where $\alpha_{u,v}^{(\ell)}$ are normalized attention coefficients computed by an attention mechanism as follows:

$$\alpha_{u,v}^{(\ell)} = \frac{e^{\varsigma(\boldsymbol{w}^{(\ell)}[(\boldsymbol{W}^{(\ell)})^\top \boldsymbol{h}_u^{(\ell-1)} \| (\boldsymbol{W}^{(\ell)})^\top \boldsymbol{h}_v^{(\ell-1)}])}}{\sum_{k \in \mathcal{N}_1(u)} e^{\varsigma(\boldsymbol{w}^{(\ell)}[(\boldsymbol{W}^{(\ell)})^\top \boldsymbol{h}_u^{(\ell-1)} \| (\boldsymbol{W}^{(\ell)})^\top \boldsymbol{h}_k^{(\ell-1)}])}}. \tag{3.24}$$

with ς the leaky ReLu function, the weighting row vector $\boldsymbol{w}^{(\ell)} \in \mathbb{R}^{2H}$, where H is the size of the hidden layer and $\|$ corresponds to column concatenation.

3.3.2 Relationship with a GNN based classifier

In this Section, we compare the shape of the proposed GAB and the GNN. In GNN, there is one activation function between each layer which implies that the multi-hop information has undergone several non-linear functions before arriving at the node of interest. In GAB, the multi-hop mixture is done prior to making the final decision and do not follow a successive concatenation of linear combination and activation function. All the operations are intermixed, therefore GNN and GAB are very different in terms of structure, except for the one-layer/one-hop case. We therefore focus here on the relation between the first-order GAB classifier and first-order GNN-based classifier. For doing that, we consider a binary classification problem (i.e $K = 2$). According to Eq. (4.1), we assign node u to class 1 if:

$$P_u(1) \geq P_u(2),$$

which implies that :

$$\frac{\pi_1 D_1(\boldsymbol{x}_u) \prod_{v \in \mathcal{N}_1(u)} (r(1,1) D_1(\boldsymbol{x}_v) + r(1,2) D_2(\boldsymbol{x}_v))}{\pi_2 D_2(\boldsymbol{x}_u) \prod_{v \in \mathcal{N}_1(u)} (r(2,1) D_1(\boldsymbol{x}_v) + r(2,2) D_2(\boldsymbol{x}_v))} > 1.$$

By setting

$$S(x) = \frac{D_1(x)}{D_2(x)}$$

and taking the log, we obtain the following test T (where $T > 0$ means "decide class 1"):

$$\begin{aligned} T &= \log\left(\frac{\pi_1}{\pi_2}\right) + \log\left(S(\boldsymbol{x}_u)\right) \\ &+ \sum_{v\in\mathcal{N}(u)} \log\left(\frac{r(1,2)+r(1,1)S(\boldsymbol{x}_v)}{r(2,2)+r(2,1)S(\boldsymbol{x}_v)}\right). \end{aligned}$$

As $q(1,2) = q(2,1)$, two classes in the system lead to $\bar{q} = q(1,2) = q(2,1)$. Consequently,

$$\frac{r(1,2)+r(1,1)S(\boldsymbol{x}_v)}{r(2,2)+r(2,1)S(\boldsymbol{x}_v)} = \frac{\frac{\bar{q}\pi_2+p(1)\pi_1 S(\boldsymbol{x}_v)}{p(1)\pi_1+\bar{q}\pi_2}}{\frac{p(2)\pi_2+\bar{q}\pi_1 S(\boldsymbol{x}_v)}{\bar{q}\pi_1+p(2)\pi_2}}.$$

Therefore, the test T can be split into three parts:

$$\begin{aligned} T &= \log\left(\frac{\pi_1}{\pi_2}\right) + \log\left(\frac{\bar{q}\pi_1+p(2)\pi_2}{p(1)\pi_1+q\pi_2}\right) \\ &+ \log\left(S(\boldsymbol{x}_u)\right) \\ &+ \sum_{v\in\mathcal{N}(u)} \log\left(\frac{\bar{q}\pi_2+p(1)\pi_1 S(\boldsymbol{x}_v)}{p(2)\pi_2+\bar{q}\pi_1 S(\boldsymbol{x}_v)}\right). \end{aligned}$$

The first part corresponds to a constant term and so is connected to the threshold. The second part is the contribution of the node of interest. The third part, which is the most interesting, corresponds to the contribution of the neighbors in the test. Clearly, in general, this term is not linear with respect to the nodes' features and so the test cannot be seen as a one-layer GNN.

If, in addition, $p(1) = p(2) = \bar{p}$, the test can be written easily with respect to the DoI as follows

$$\begin{aligned} T &= \log\left(\frac{\pi_1}{\pi_2}\right) + \log\left(\frac{\mathrm{DoI}.\pi_1+\pi_2}{\pi_1+\mathrm{DoI}.\pi_2}\right) \\ &+ \log\left(S(\boldsymbol{x}_u)\right) \\ &+ \sum_{v\in\mathcal{N}(u)} \log\left(\frac{\mathrm{DoI}.\pi_2+\pi_1 S(\boldsymbol{x}_v)}{\pi_2+\mathrm{DoI}.\pi_1 S(\boldsymbol{x}_v)}\right). \end{aligned}$$

For instance, we remark that if the DoI is much larger than the pdf ratio between the classes (which may be roughly related to the Kullback-Leibler divergence), then the third term is almost independent of the nodes's feature and the information provided by the graph is not used since it is not reliable.

Consider now that the graph is pure (i.e., $r(1,2) = r(2,1) = 0$ or equivalently $q = 0$), we obtain

$$\begin{aligned} T &= \log(\frac{\pi_1}{\pi_2}) + |\mathcal{N}(u)| \log\left(\frac{r(1,1)}{r(2,2)}\right) \\ &+ \sum_{v\in\mathcal{N}(u)\cup\{u\}} \log(S(\boldsymbol{x}_v)), \end{aligned} \tag{3.25}$$

where we consider that $p(1)$ may be different from $p(2)$. Once again the proposed classifier does not boil down to a one-layer GNN. Actually, it can be a GNN if the term $\log_2(S(\boldsymbol{x}_v))$ is a linear combination of $\boldsymbol{x}_v$. This can be achieved if the function $S(\boldsymbol{x}_v)$ is at least a power-function of $\boldsymbol{x}_v$, such as the Gaussian function or Binomial function.

Let us first consider the Gaussian case, i.e. $\boldsymbol{x}_v \sim \mathcal{N}(\boldsymbol{\mu}(v), \boldsymbol{\Sigma})$ where $\boldsymbol{\mu}(v)$ is either $\boldsymbol{\mu}_1$ (if class 1) or $\boldsymbol{\mu}_2$ (if class 2) and the correlation matrix is independent of the class (if not, a second-order polynomial occurs and the GAB is different from a GNN). According to Eq. (3.25), the first-order GAB test is equal to

$$T = \omega_0 + \left(\sum_{v \in \mathcal{N}(u) \cup \{u\}} \boldsymbol{\omega}_1^\top \boldsymbol{x}_v \right), \tag{3.26}$$

where

$$\begin{aligned} \omega_0 &= \log\left(\frac{\pi_2}{\pi_1}\right) \\ &+ (|\mathcal{N}(u)|+1)(\boldsymbol{\mu}_2^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1), \\ \boldsymbol{\omega}_1^\top &:= [\omega_{1,1}, \cdots, \omega_{1,F}] = (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^\top \boldsymbol{\Sigma}^{-1}. \end{aligned} \tag{3.27}$$

Consequently this test is a GNN-based test.

Let us now consider the Binomial case. We assume that features $x_{v,f}$ are independent binary random variables with probabilities $P(x_{v,f}=1|y_v=1) = \alpha_f^{(1)}$ and $P(x_{v,f}=1|y_v=2) = \alpha_f^{(2)}$. Eq. 3.25 can be written as Eq. (3.26) with

$$\begin{aligned} \omega_0 &= \log\left(\frac{\pi_2}{\pi_1}\right) + F\sum_{f=1}^{F} \log\left(\frac{1-\alpha_f^{(2)}}{1-\alpha_f^{(1)}}\right), \\ \omega_{1,f} &= -\log\left(\frac{\alpha_f^{(1)}}{\alpha_f^{(2)}}\frac{1-\alpha_f^{(2)}}{1-\alpha_f^{(1)}}\right). \end{aligned}$$

Consequently this test is a GNN-based test as well.

3.4 Discussion on complexity issues

In this section, we compare the different versions of the proposed GAB and the GAT in terms of parameters/weights to tune and the number of flops for doing this tuning during the training phase. We will consider only the two-hops case for the GAB and the two-layers case for the GAT.

First of all, we evaluate the number of parameters to be tuned. Let H and $d_{\max}$ be respectively the size of the hidden layer and the maximum degree of the considered graph.

For a GAB classifier, we need to estimate

- K^2 parameters for the transition matrix $\mathbf{R}$ through the terms $\{p(k)\}_k$ and $\{q(k,k')\}_{k \neq k'}$;

- the parameters of the class' distributions (obviously, this value depends on the shape of the assumed distributions):
  - KF parameters when each class of each feature is Bernoulli-distributed
  - $(KF+KF^2)$ parameters when each class is arbitrary Gaussian-distributed. The number of parameters can be reduced if the Gaussian distribution per class is structured. For instance, if the covariance is independent of the class, we only have $(KF+F^2)$ parameters. If in addition, the covariance matrix is diagonal or is assumed as a diagonal matrix for the sake of simplicity, we have $(KF+F)$ parameters;
  - K parameters for the priors $\{\pi_k\}_k$.

When Cora or PubMed dataset are considered (see Table 3.5 for more details), the distribution is assumed to be Bernoulli in our numerical evaluations (even if not), which implies that the total number of parameters to estimate, N_p is given by

$$N_p = K(K+F+1). \tag{3.28}$$

For GAT, we need to tune/learn the weights and the attention parameters of the neural network. As only two layers are considered we obtain FH weights for the first layer, HK weights for the second layer, and $2H$ (resp. $2F$) weights for the attention mechanism for the first layer (resp. the second one). As a consequence, the number of weights to tune, denoted by N_w is as follows:

$$N_w = FH + HK + 2(H+K). \tag{3.29}$$

According to the number of classes and features given in Table 3.5 and by assuming a hidden layer of size $H = 256$, we have the following values N_p and N_w for the Cora and PubMed dataset in Table 3.2.

Table 3.2: Number of parameters or weights to be tuned/learnt.

Dataset	N_p (GAB)	N_w (GAT)
Cora	10,087	369,066
PubMed	1,512	129,286

We observe that the number of parameters for GAB is much smaller (by more than an order of magnitude) than for GAT. In addition, the parameters in GAB are interpretable. Moreover, since for GAT, parameters are learnt using an iterative process, the computational complexity in terms of the number of epochs may significantly vary with the chosen optimization algorithm.

We here consider two variants for the GAB. The first one (denoted by GAB2) corresponds to the case where $\gamma_1 = \gamma_2 = 1$ while the second one (denoted by GAB2γ) is optimized with respect to the pair $\boldsymbol{\gamma} = (\gamma_1, \gamma_2)$. Let N_t, N_g, and N_v be the number of training

nodes, the number of tested pairs $\boldsymbol{\gamma}$, and the number of nodes for validation respectively. To estimate the N_p parameters during the training phase, we need N_pN_t flops since each parameter corresponds to counts or sums over each training node. To select the best pair of hyperparameters $\boldsymbol{\gamma}$, we compute our GAB on all validation and test nodes. A GAB evaluation leads to at most $K^2D_{\max}^2F$ flops by looking at Eq. (4.1). Indeed we consider that to compute each test for one class, we need $d_{\max}^2KF$ flops where F corresponds to the flops required to compute $D_k(\cdot)$ as it is the case for the Bernoulli case or the diagonal correlation matrix Gaussian case. Finally, we have that

$$N_{\text{flop}|\text{GAB2}\gamma} = N_pN_t + N_gN_vK^2d_{\max}^2F. \tag{3.30}$$

When we only consider GAB2, the second term in the Right Hand Side (RHS) has to be omitted. Note that the 2D grid for the hyperparameter $\boldsymbol{\gamma}$ is $[0:0.1:1]^2$ leading to $N_g = 121$.

For GAT2, we just consider the number of flops required for learning the weights (so the hyperparameters such as the learning rate are assumed to be obtained for free). To learn the weights, we apply a gradient-descent like algorithm with N_e epochs. Hereafter, we consider only the neural network's weights which are dominant. So the weights related to the attention mechanism are ignored.

We consider one epoch. For the first layer (resp. second layer), we have FH (resp. HK) weights to update and so FH (resp. HK) sums have to be computed once the gradient is known. For estimating the gradient, we average over N_t nodes, over the sum of the neighbors (at most $d_{\max}$) and a matrix computation of size HF (resp. KH) with the current feature of size F (resp. H). Consequently, we have

$$N_{\text{flop}|\text{GAT2}} = N_eFHN_td_{\max}FH + N_eHKN_td_{\max}HK. \tag{3.31}$$

In Table 3.3, we report the rough number of flops for Cora (with a supervision of 30% and 5%) and PubMed (with a supervision of 5%) with three classifiers. We set $N_e = 500$, $N_t = 140$ (Cora-5%) or $N_t = 1000$ (Cora-30% and PubMed-5%). Moreover $d_{\max} = 168$ for Cora and $d_{\max} = 171$ for PubMed.

Table 3.3: Number of flops for the training phase.

Dataset	**GAB2**	**GAB2γ**	**GAT2**
Cora-30%	1.01×10^7	1.99×10^{11}	1.14×10^{19}
Cora-5%	1.41×10^6	1.99×10^{11}	1.60×10^{18}
PubMed-5%	1.51×10^6	9.31×10^{10}	1.40×10^{18}

In the inference phase, applying our classifier (GAB2) leads to $K^2d_{\max}^2F$ flops (which corresponds to the number of flops in the second part of the RHS in Eq. (3.30) without N_g and N_v). For GAT2, we have $FHd_{\max}FH + HKd_{\max}HK$ flops (actually, we apply Eq. (3.31) by removing N_e and N_t). In Table 3.4, we report the rough number of flops during the inference phase for Cora and PubMed.

We remark that the numbers of flops for our GAB classifiers are much smaller (by many orders of magnitude) than for the GAT in both training and inference phases. Consequently,

Table 3.4: Number of flops for the inference phase.

Dataset	GAB2/GAB2γ	GAT2
Cora	2.3×10^9	1.60×10^{12}
PubMed	1.3×10^8	2.80×10^{11}

the structure imposed by the derivations of the GAB enable us to have interpretability and less complexity than the black box GAT.

3.5 Numerical Results

In this section, we conduct experiments with two kinds of datasets: the synthetic ones where we especially analyze the robustness to the degree of impurity; and some real benchmark ones where we compare our classifier to many other approaches based on GNN. The performance of our classifier (and the ordering with respect to standard GNN based approaches) depend on the dataset and the level of supervision.

3.5.1 Synthetic datasets

For the sake of simplicity, we consider two classes, i.e., $K = 2$. Each class is associated with a different statistical distribution of the node's feature vector $\boldsymbol{x}_v$. We assume here that the two distributions are multivariate Gaussian, i.e. $\boldsymbol{x}_v$ follows $\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ in class 1 and $\mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$ in class 2, with unknown mean and covariance. We assume that the features are uncorrelated in the two classes, i.e. $\boldsymbol{\Sigma}_1 = \text{diag}(\boldsymbol{\sigma}_1^2)$ and $\boldsymbol{\Sigma}_2 = \text{diag}(\boldsymbol{\sigma}_2^2)$, where $\boldsymbol{\sigma}_1^2$ and $\boldsymbol{\sigma}_2^2$ are $(F \times 1)$ vectors. To average over different configurations, different distributions are generated by drawing $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$ randomly from $\mathcal{U}([0,1]^F)$ and drawing $\boldsymbol{\sigma}_1$ and $\boldsymbol{\sigma}_2$ randomly from $\mathcal{U}([0.5, 3.5]^F)$. Moreover, the links are generated randomly using different values of the probability of intra-connectivity, $\bar{p}$, and the probability of inter-connectivity, $\bar{q}$ where $\bar{q} \leq \bar{p}$.

In each experimental setting, we evaluate the average node classification accuracy using a Monte Carlo simulation with $1,000$ runs. We set the number of nodes to $N = 5,000$ and the number of features to $F = 500$. For each run, we train each classifier on 500 (already-labeled) nodes and test its accuracy on $2,000$ nodes. We use the remaining nodes for validation.

In Fig. 3.1, we plot the classification accuracy versus the Degree of Impurity ($\bar{q}/\bar{p}$) for one-order GAB and one-layer GCN. We observe that the GAB is more robust to DoI than the GCN especially beyond a DoI of one half. As expected, the GAB is graph-agnostic when Eq. (3.18) is satisfied. For instance, when $p(1) = p(2) = 0.05$ then the agnosticity occurs at $\bar{q}/\bar{p} = 1$ while when $p(1) = 0.025$ and $p(2) = 0.075$, the agnosticity is reached at $\bar{q}/\bar{p} = 0.866$. We remark that GNN has similar behavior with respect to the usefulness of the information coming from the graph.

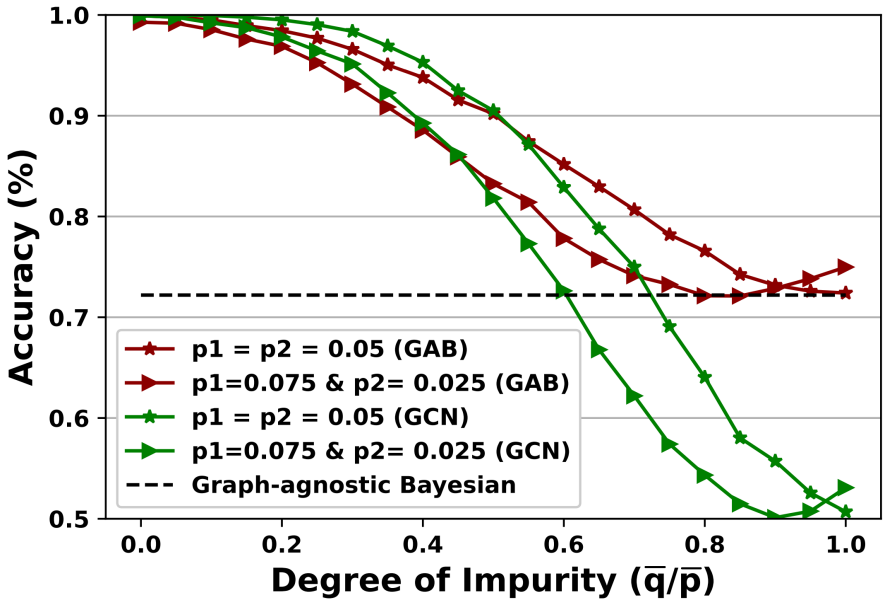


Figure 3.1: GAB and GNN performance on synthetic datasets versus DoI.

3.5.2 Real datasets

Unless otherwise stated, we use the attributed graphs described in Table 3.5, and we use the training/validation/testing split equal to 30%, 20%, 50%, respectively. To implement the GNN based classifiers, we use Pytorch where we initialize all the GNN weights by Glorot initialization, and we train them to minimize the cross entropy loss using the Adam optimizer with an initial learning rate of 0.005.

Table 3.5: Description of the real datasets.

Dataset	Classes	Nodes	Edges	Features
Cora[73]	7	2,708	5,429	1,433
CiteSeer[73]	6	3,327	4,732	3,707
PubMed[73]	3	19,717	44,338	500
CS[74]	15	18,333	163,788	6,805
Physics[74]	5	34,493	495,924	8,415
Sexual[64, 44]	2	1,888	2,096	20

In Fig. 3.3, we plot the accuracy versus the DoI: on the top, the GAB at several orders as well as the best combination of powers γ_d in Eq. (4.1). Here, the training phase enables us to estimate the parameters of the GAB except the γ_d. The powers γ_d are optimized with a grid search approach during the validation phase, on the middle panel, the GAT with several layers, and on the bottom panel the GCN at several layers. The dataset is here always Cora which implies that the distributions are Bernoulli. The x-axis starts with the real value of the DoI and we add links between previously unconnected nodes that belong to different classes with the goal of gradually varying the DoI to 1. The above mentioned classifiers are trained for each "impure" graph. We remark that the information on the graph is less accurate for

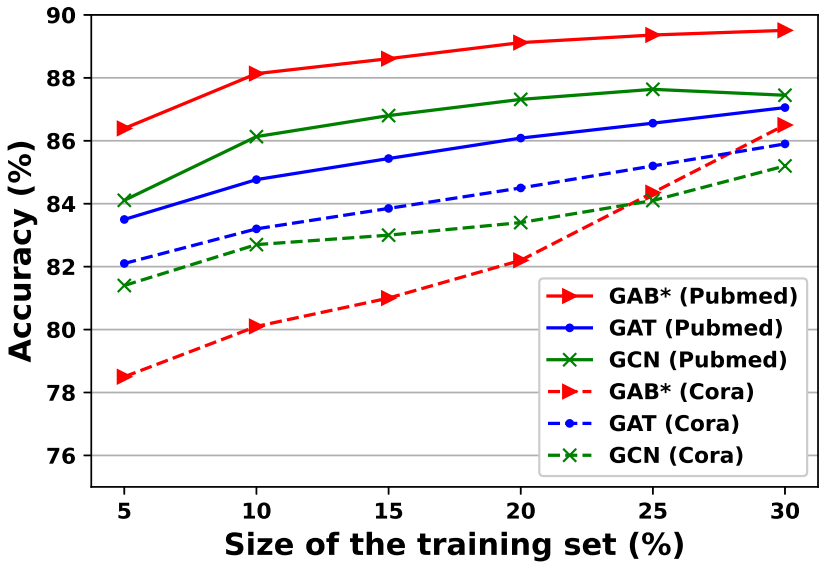


Figure 3.2: GAB and GNN performance on Pubmed versus the size of the training set.

any approach when we consider more hops; actually, the confidence on the "far" data is smaller. Nevertheless by averaging properly the hops as in GABγ, the performance are improved slightly. In any case, GABγ is better then the best other approaches since it enables to better adapt to the impurity. Moreover GABγ outperforms the graph-agnostic classifier; so it always manages to take benefit of the graph.

In Table 3.6, we compare our approaches GABγ and GAB$\star$ up to five hops (in GAB$\star$, we approximate the unknown distributions by two-layer NNs which are learnt during the training phase) with the methods presented and analyzed in [45]. We copy-paste the values given in [45] where the best number of layers and the best version of each approach have been selected. We select the best version of GAB in the sense that we optimized the hyperparameters $\boldsymbol{\gamma}$ by allowing at most the fifth-order case. In GABγ, the shape of the distributions is a priori given. For instance, for PubMed, we considered a Bernoulli one while it is inaccurate. We plot the accuracy rate and the ranking (in brackets) for a 30%-supervised graph. For computing the average accuracy and the ranking (for all the considered datasets) for our GAB approach, we select the best one. We remark that the performance of our approach is close to GBPN and GAT which are the best ones in the literature. When GABγ is bad, it corresponds to the case PubMed where the used distribution does not fit well with the true one. So improvement can be done by choosing a better approximation. In addition to the good performance of our proposed approach, we have interpretability of our algorithms since we wrote them in closed-form and we are able to understand the meaning of each element.

In Fig. 3.2, we plot the accuracy rate versus the training size (in percentage) for our approach (GAB$\star$) and the GCN and GAT with two-layers/hops. In solid line, we consider PubMed dataset and in dash line, we consider Cora dataset. We observe that for PubMed, our approach outperforms the state of the art regardless of the training size. Actually we should note than as PubMed is a large dataset, a small portion of training still leads to a

Table 3.6: 30%-supervised node classification accuracy (%).

Dataset	MLP	GCN	SAGE	GAT	GMNN	DeeperGCN	GBPN	GABγ	GAB$\star$
Cora	72.1(9)	87.1(3)	86.9(4)	87.1(2)	86.4(6)	**87.2(1)**	86.4(6)	86.9(4)	86.3(8)
CiteSeer	71.2(9)	73.5(5)	73.5(5)	73.1(7)	72.9(8)	73.9(4)	74.8(2)	**75.2(1)**	74.7(3)
PubMed	86.5(6)	87.1(5)	87.8(4)	88.1(3)	86.7(7)	84.7(9)	88.5(2)	86.4(8)	**89.5(1)**
CS	94.2(5)	93.2(9)	93.7(7)	94.0(6)	93.3(8)	94.9(3)	**95.5(1)**	94.5(4)	95.2(2)
Physics	95.8(9)	96.1(6)	96.3(5)	96.3(6)	96.1(8)	96.7(3)	**96.9(1)**	96.4(4)	96.7(2)
Sexual	74.5(8)	83.9(6)	93.3(5)	93.6(4)	77.0(7)	65.0(9)	**97.4(1)**	96.5(3)	97.1(2)
	84.4(7.6)	86.8(5.6)	88.9(5)	89.0(4.6)	85.2(7.3)	83.0(4.8)	89.9(2.2)	**90.1(2)**	

large amount of data to estimate the statistics required for GAB. In contrast, for Cora, our approach outperforms GCN and GAT only when the training is large enough (and as Cora is a small dataset, large enough means also when the training set in percentage is large enough).

According to all the previous experiments, we remark that our GAB approach is more robust to the degree of impurity and its performance is close to or better than the tested GNN approaches on a number of benchmark graphs.

3.6 Conclusion

We have proposed a new graph-assisted Bayesian-based node classifier. This classifier is able to take into account the degree of impurity of the graph. It is also shown to significantly outperform some GNN-based classifiers, in addition to providing more interpretability and requiring lower computational complexity (if the model of the nodes' distributions is well approximated in closed-form).

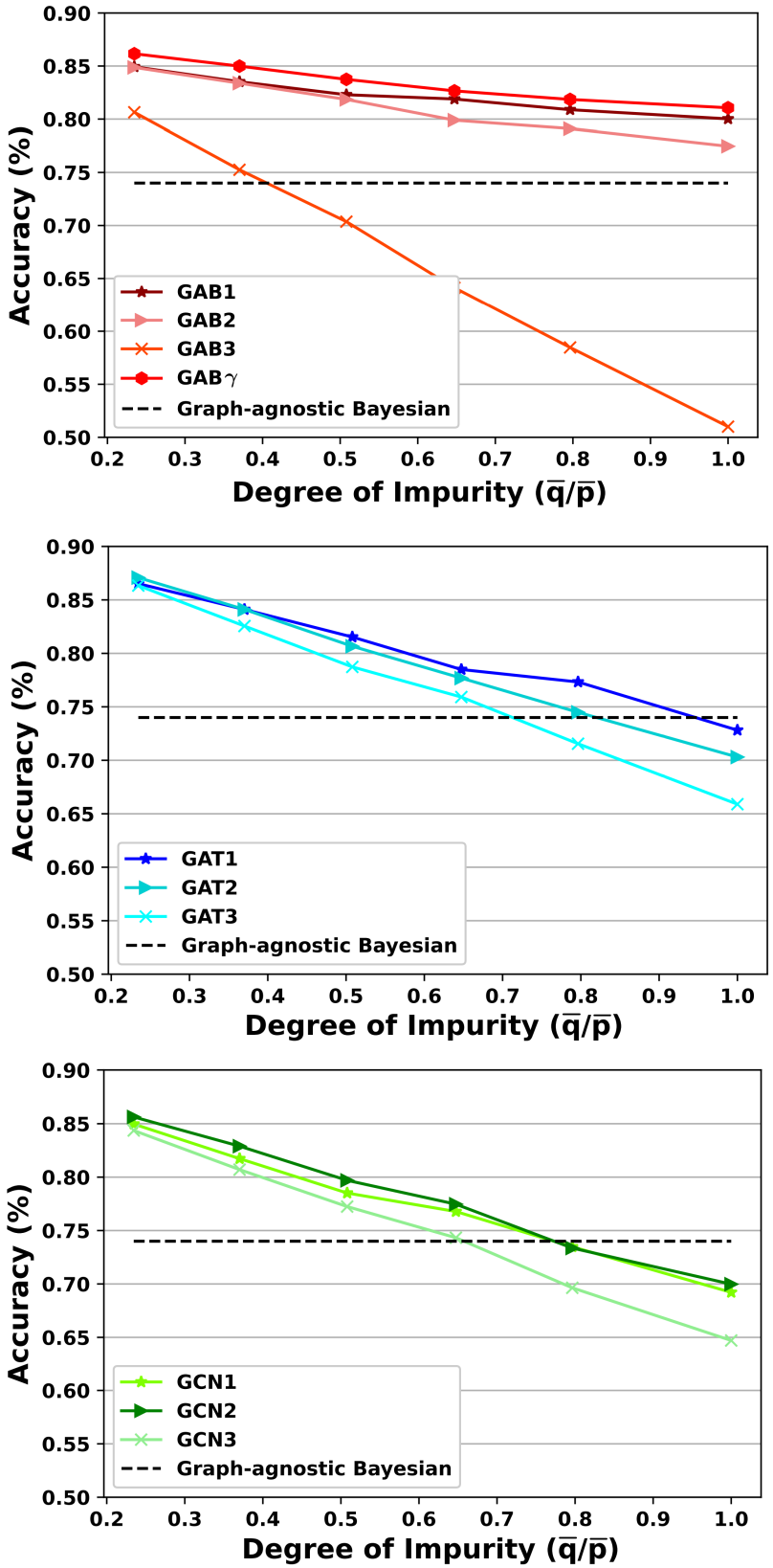


Figure 3.3: Accuracy performance with added noisy edges in Cora.

Chapter 4

Robustness to adversarial attacks

4.1 Introduction . . . 66
4.2 Adversarial attacks . . . 67
4.2.1 Gradient-Based Attacks . . . 67
4.2.2 Optimization-Based Attacks . . . 68
4.2.3 Generative Adversarial Network (GAN)-Based Attacks . . . 68
4.3 Adversarial attacks on graphs . . . 69
4.3.1 Attacker's Capacity . . . 69
4.3.2 Methods of Attacking a Graph . . . 69
4.4 Measures to prevent adversarial attacks on graphs . . . 70
4.4.1 Adversarial training . . . 71
4.4.2 Data purification . . . 71
4.5 Robustness of the graph assisted Bayesian classifier . . . 71
4.5.1 The graph assisted Bayesian classifier as a belief propagation framework . . . 72
4.5.2 Node similarity as a simple defence mechanism for GAB . . . 74
4.6 Experiments . . . 74
4.6.1 Attack scenario . . . 74
4.6.2 Attack description . . . 75
4.6.3 Data description . . . 75
4.6.4 Results . . . 76
4.7 Conclusion . . . 76

4.1 Introduction

As we have mentioned multiple times in the previous chapters, graphs can be used as an effective tool for representing many systems across multiple areas and disciplines. We have also seen that representation learning and more specifically graph neural networks have been used successfully to obtain powerful representations of either individual nodes on graphs or the entire graphs. Representations that can be fed into machine learning models to solve several downstream tasks. We recall that most graph neural networks take advantage of the message passing scheme which aim at gathering information from neighboring nodes to update each node's representation. This allows us to learn representations that capture both intrinsic nodes' features and the structural information of the graph. It also has been found that GNNs are useful in a variety of applications due to their strong representation learning capacity.

Despite their success, it was also shown that these models inherit drawbacks of classical deep learning models such as convolutional neural networks. More specifically their vulnerability to adversarial attacks. Which means that injecting small perturbations to either nodes' features or the structure of the graph significantly decreases the performance of these models. These vulnerabilities challenge the effectiveness of GNNs and make their deployment in critical applications a concern. On social networks, fraudulent or malicious accounts can link to normal users and thus fool fake account detection systems. They can then continue to operate normally on these platforms by spreading false information for example.

It is therefore necessary to carefully study the impact of this type of attacks on GNNs and investigate new techniques to increase the robustness of these models in order to promote the successful adoption of GNNs in a broader range of applications. This particular issue caught the attention of academics, who developed a variety of strategies and tactics to both deceive GNNs by carefully designing subtle alterations to the graph data and to protect them from these same attacks.

In the previous chapter we have introduced a new Bayesian based classifier for the problem of node classification. We have showed that its ability to take into account the degree of impurity of the graph makes it more suitable for graphs with weak community structure when compared to graph neural networks based approaches. In this chapter our primary objective is to study the robustness of the proposed classifier to adversarial attacks on graph structured data. We show that it is naturally significantly more robust than GNNs to this kind of attacks. We further suggest simple defence mechanisms and compare their performance with equivalent defences on GNNs.

The chapter is organized as follows. In section 4.2, we give an introduction to adversarial attacks and describe how they were first used and generated for images. In section 4.3, we focus on adversarial attacks for graph structured data by giving an introduction to the different types of attacks that were proposed in the literature. We then present some of the proposed defences against graph attacks in section 4.4. The two last section focus on the robustness of GAB to adversarial attacks. In section 4.5 we introduce a defence mechanism to improve the robustness of gab, while in section 4.6 we demonstrate how that mechanism

improves the robustness though a number of experiments.

4.2 Adversarial attacks

The term adversarial attack was first introduced in [17]. Adversarial attacks are small perturbations to the input that can cause a machine learning model to produce erroneous outputs. These perturbations are usually imperceptible to humans. The perturbations are generated in a way that maximizes the model's loss function. The field of adversarial attacks has grown significantly in the last few years.

Adversarial attacks can be targeted or untargeted. Targeted attacks aim to change the output of the model to a specific class. For example, a targeted attack on a malware detection system would be an attack that aims to change the classification of a malicious file to benign. On the other hand, untargeted attacks only aim to change the output of the model and do not have a specific target class.

Adversarial attacks can be generated in a white-box setting or a black-box setting. In the white-box setting, the adversary has access to the model's parameters and architecture. This is a common setting in the early works on adversarial attacks. The attacks are generated by using the gradient of the loss function with respect to the input. In the black-box setting, the adversary has no access to the model's parameters or architecture. In this setting, the adversary generates the attacks by repeatedly querying the model and observing its outputs.

Adversarial attacks can be generated using a variety of methods. In this section, we will discuss the different methods that have been used to generate adversarial attacks.

4.2.1 Gradient-Based Attacks

We will discuss the methods that use the gradient of the loss function with respect to the input to generate adversarial attacks. The gradient of the loss function can be used to identify the direction in which the input needs to be perturbed to maximize the loss function. The loss function can be chosen to achieve a specific goal such as a targeted attack.

One of the first works that used the gradient of the loss function to generate adversarial attacks was [76]. In this work, the authors used the box-constrained Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) optimization method to generate the attacks. The goal of the attack was to find the minimum perturbation that causes the model to misclassify the input. The L-BFGS method requires access to the model's parameters and architecture. Therefore, the attack is generated in a white-box setting.

Goodfellow et al. [28] proposed a fast method for generating adversarial attacks using the gradient of the loss function. This method, called the Fast Gradient Sign Method (FGSM), perturbs the input by a small value in the direction of the sign of the gradient of the loss function. The attack can be generated in a white-box setting or a black-box setting. In the white-box setting, the adversary has access to the model's parameters and architecture.

In this case, the gradient of the loss function can be calculated using backpropagation. In the black-box setting, the adversary has no access to the model's parameters or architecture. In this case, the adversary can estimate the gradient of the loss function by repeatedly querying the model and observing its outputs. The attack can be generated using any loss function that the adversary desires. For example, the adversary can choose the loss function to achieve a targeted attack or an untargeted attack.

Madry et al. [58] proposed an iterative version of the FGSM method. This method, called the Projected Gradient Descent (PGD) method, perturbs the input in multiple iterations by a small value in the direction of the sign of the gradient of the loss function. In each iteration, the perturbed input is projected onto the space of allowable perturbations. This method can also be used in a white-box setting or a black-box setting and can also be applied both the targeted and untargeted settings.

4.2.2 Optimization-Based Attacks

In this section, we will discuss the methods that use an optimization problem to generate adversarial attacks. The optimization problem can be used to generate a targeted attack or an untargeted attack.

Carlini and Wagner [7] proposed a method for generating adversarial attacks using an optimization problem. The goal of the optimization problem is to find the minimum perturbation that causes the model to misclassify the input. The optimization problem can be solved using the Adam [50] or the L-BFGS [56] optimization methods. The attack can be generated in a white-box setting or a black-box setting. In the white-box setting, the adversary has access to the model's parameters and architecture. In this case, the gradient of the loss function can be calculated using backpropagation. In the black-box setting, the adversary has no access to the model's parameters or architecture. In this case, the adversary can estimate the gradient of the loss function by repeatedly querying the model and observing its outputs.

4.2.3 Generative Adversarial Network (GAN)-Based Attacks

In this section, we will discuss the methods that use a generative adversarial network (GAN) to generate adversarial attacks. GANs were first introduced in [27]. They consist of two networks: a generator and a discriminator. The generator takes a random noise vector as input and generates an output. The discriminator takes the generator's output and tries to classify it as real or fake. The generator and the discriminator are trained simultaneously to compete with each other. GANs have been used to generate images that look realistic.

In [87], the authors proposed a method for generating adversarial attacks using a GAN. The GAN is trained to generate adversarial attacks. The generator takes the original input and generates an adversarial example. The discriminator takes the original input and the adversarial example and tries to classify them as real or fake. The generator and the discriminator are trained simultaneously to compete with each other. The generator can be

trained using any loss function that the adversary desires.

4.3 Adversarial attacks on graphs

The primary taxonomy of adversarial attacks on graph-structured data is briefly introduced in this section. Attack algorithms can be divided into many categories according to the objectives, capabilities, resources, and knowledge of the attackers. We make an effort to provide a concise explanation of the key aspect of graph adversarial attacks.

4.3.1 Attacker's Capacity

The attacker's capacity to introduce adversarial noise to the graph data defines a first categorization of adversarial attacks on graphs. Attacks can occur either at the training or the test phase. In the first scenario, **poisoning** attacks, also known as training-time attacks, aim to influence the performance of the targeted models by changing the training datasets. In other words, attackers are adding "poison" into data on which the models are trained leading to a decrease in their performances. In this case, after changing the training dataset, the target models' parameters are retrained. On the other hand, in **evasion** attacks, models are already trained on clean datasets and their parameters are fixed. Attacks take place only during the inference phase.

4.3.2 Methods of Attacking a Graph

The most common methods for attacking a graph are to either add or remove edges from the graph. This is because the edges in a graph represent relationships between nodes, and changing the edges can change the structure of the graph. In some cases, it is also possible to add or remove nodes from the graph or modify nodes' features.

Edge Modification

Most adversarial attacks on graphs focus on changing the edges in the graph. In these attacks, the attacker adds or removes edges from the graph. These attacks can be divided into two types: untargeted attacks, where the goal is to change the label of any node in the graph, and targeted attacks, where the goal is to change the label of a specific node. Untargeted attacks are the most common type of attack, and they are the type of attack that is most commonly used to evaluate an attack. The main goal of an untargeted attack is to change the label of as many nodes as possible. This can be achieved by adding or removing edges from the graph.

One of the first untargeted attacks on graphs was proposed by in [104]. In this attack, the attacker can either separately add or remove edges to the graph. The attacker uses a surrogate model to estimate the importance of each edge in the graph. The surrogate model

is a two-layer GCN, and it is trained to predict the labels of the nodes in the graph. The attacker then uses a greedy algorithm to add edges to the graph. At each iteration of the algorithm, the edge that maximizes the classification loss is selected. The algorithm stops when the maximum number of edges have been added. It is possible to both add and remove edges from the graph. The first attack of this type was proposed by Chen et al. [11]. In this attack, the attacker uses a surrogate model to estimate the importance of each edge in the graph. The surrogate model is a DeepWalk model, and it is also trained to predict the labels of the nodes in the graph.

Node Modification

Another type of attack is to add or remove nodes from the graph. This type of attack is less common, because it is less powerful since by modifying label of neighbors of the attacked node does not force automatically the change of the label of the considered node. But the main advantage of node modification attacks is that they are more difficult to detect, because the structure of the graph is not changed as much.

The first node modification attack was proposed by Dai et al. [16]. In this attack, the attacker can either add or remove nodes to the graph. The attacker uses a surrogate model to estimate the importance of each node in the graph. The surrogate model is a two-layer GCN, and it is trained to predict the labels of the nodes in the graph. The attacker then uses a greedy algorithm to add nodes to the graph. At each iteration of the algorithm, the attacker selects the node that maximizes the classification loss. The algorithm stops when the maximum number of nodes have been added.

Feature Modification

There are other types of attacks that are less common, but they have been proposed in recent years. These attacks include attacks that change the features of nodes. The first attack of this type was proposed by Chen et al. [11]. He uses DeepWalk model that is trained to predict the labels of the nodes in the graph as a surrogate model to estimate the importance of each feature. The attacker then uses a greedy algorithm to modify the features. At each iteration of the algorithm, the attacker selects the feature that maximizes the classification loss. The algorithm stops when the maximum number of features have been modified.

4.4 Measures to prevent adversarial attacks on graphs

We have demonstrated in earlier sections that subtle perturbations to graph data can easily mislead graph neural networks. It is incredibly challenging to use graph neural networks in safety-critical applications because of their vulnerability. Various countermeasure tactics have been put out to protect graph neural networks from these attacks. The approaches currently in use can be divided into different categories.

4.4.1 Adversarial training

A popular defense against adversarial attacks on image data is adversarial training [8]. In order for the trained model to successfully categorize upcoming adversarial cases, adversarial examples must be inserted into the training set. Similar to this, we can use this approach to counter graph adversarial attacks. It is suggested that during adversarial training, edges are dropped at random to provide perturbations on the adjacency matrix [16]. Even though such a straightforward technique only slightly increases classification accuracy (1% increase), it demonstrates some effectiveness with such inexpensive adversarial training. Additionally, projected gradient descent rather than randomly dropping edges is used to produce perturbations on the discrete input structure [88]. On the other hand, a dynamic regularization adversarial training technique is suggested to disturb the input features [23].

4.4.2 Data purification

Adversarial training techniques only focus on defending against evasion attacks, which focuses on attacks that happen during training. Instead, the majority of graph purification defence strategies concentrate on avoiding poisoning attacks. Purification approaches seek to clean the poisoned graph and create reliable graph neural network models based on it given that poisoning attacks introduce poisons into the training graph. Pre-processing [22] and graph learning [46] are two methods for achieving graph purification.

Pre-processing Pre-processing techniques initially clean up the data from the graph before training the GNN model on it. The GNN model is trained on an uninfected graph in this manner. Authors in [85] suggest a purification technique based on two conclusions made about the attacking techniques: Attackers typically favor adding edges over removing edges or changing characteristics, and they also frequently connect nodes with distinct features. They thus suggest a protection strategy that involves removing edges who connect nodes which have low Jaccard Similarity. The edge between these two nodes could be hostile because they are different from one another and it is unlikely that they are related in reality [71]. The experimental results show that the suggested defense strategy is effective and efficient.

Graph learning Pre-processing might not be the best option for cleaning the graph because it is not a part of the GNN training procedure and might unintentionally delete ordinary edges. Graph learning is an alternate purification technique that aims to eliminate adversarial patterns.

4.5 Robustness of the graph assisted Bayesian classifier

In this section, we are interested in the robustness of the graph assisted Bayesian (GAB) classifier that we introduced in chapter 3. In section 4.5.1, we provide an intuitive and

interpretable formulation of GAB. This formulation will allow us to introduce a simple defence mechanism that increases the robustness of GAB to adversarial attacks in section 4.5.

4.5.1 The graph assisted Bayesian classifier as a belief propagation framework

We recall that our graph assisted Bayesian classifier is defined as stated in Eq.3.14 as follows:

$$\begin{aligned}\hat{k}_u &= \arg\max_k \quad \pi_k D_k(\boldsymbol{x}_u) \prod_{d=1}^{\Delta_u} \prod_{v \in \mathcal{N}_d(u)} \\ &\qquad \times \left(\sum_{k'=1}^{K} r^{(d)}(k,k')^{\gamma_d} D_{k'}(\boldsymbol{x}_v) \right) \end{aligned} \tag{4.1}$$

In the rest of the chapter, we consider the first order classifier which only takes into account first-order neighbors of the nodes of interest. It can be expressed as follows:

$$\hat{k}_u = \arg\max_k P(y_u = k | \mathcal{X}_u, \mathcal{I}_\mathcal{G}), \tag{4.2}$$

where $\mathcal{I}_\mathcal{G}$ provides information only about adjacent nodes to node u and $\mathcal{X}_u = \{\boldsymbol{x}_u\} \cup \{\boldsymbol{x}_v, v \in \mathcal{N}_1(u)\}$ is the set of feature vectors of node u and its first order neighbors. The posterior probability of a node u belonging to class k knowing features of its first-order neighbors can then be expressed as:

$$P(y_u = k | \mathcal{X}_u, \mathcal{I}_\mathcal{G}) = \pi_k D_k(\boldsymbol{x}_u) \prod_{v \in \mathcal{N}_1(u)} \left(\sum_{k'=1}^{K} r(k,k') D_{k'}(\boldsymbol{x}_v) \right)$$

To solve the equation Eq.4.2, we have to compute the probability $P(y_u = k | \mathcal{X}_u, \mathcal{I}_\mathcal{G})$ for all $k \in \{1, \ldots, K\}$, then assign the class with the larger probability to the node. We can summarize the expression for all possible classes as follows:

$$\vec{\boldsymbol{p}}_u = \vec{\boldsymbol{\pi}} \odot \vec{\boldsymbol{b}}_u \odot \bigodot_{v \in \mathcal{N}_1(u)} \boldsymbol{R} \vec{\boldsymbol{b}}_v \tag{4.3}$$

where:

- The operator $\odot$ is the Hadamard element-wise product, i.e.

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_K \end{pmatrix} \odot \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_K \end{pmatrix} = \begin{pmatrix} x_1 y_1 \\ x_2 y_2 \\ \vdots \\ x_K y_K \end{pmatrix}$$

- The operator $\overset{\odot}{\prod}$ represents the Hadamard product of a sequence of vectors.
- $\vec{\boldsymbol{p}}_u = \begin{pmatrix} P(y_u = 1|\mathcal{X}_u, \mathcal{I}_\mathcal{G}) \\ P(y_u = 2|\mathcal{X}_u, \mathcal{I}_\mathcal{G}) \\ \vdots \\ P(y_u = K|\mathcal{X}_u, \mathcal{I}_\mathcal{G}) \end{pmatrix} \in \mathbb{R}^{K\times 1}$ where the k-th entry represents the probability of belonging to class k.
- $\vec{\boldsymbol{\pi}} = \begin{pmatrix} \pi_1 \\ \pi_2 \\ \vdots \\ \pi_K \end{pmatrix} \in \mathbb{R}^{K\times 1}$ where the k-th entry represents the prior probability of belonging to class k.
- $\vec{\boldsymbol{b}}_u = \begin{pmatrix} D_1(\boldsymbol{x}_u) \\ D_2(\boldsymbol{x}_u) \\ \vdots \\ D_K(\boldsymbol{x}_u) \end{pmatrix} \in \mathbb{R}^{K\times 1}$ where the k-th entry represents the prior probability of belonging to class k based only on intrinsic features of node u
- $\boldsymbol{R} \in \mathbb{R}^{K\times K}$ is the transition probability matrix whose entries $r(k, k')$ represent the probability that a node of class k is connected to a node of class k'.

Since or multiplication are element-wise, by taking the log of the previous equation, we obtain the following:

$$\log(\vec{\boldsymbol{p}}_u) = \log(\vec{\boldsymbol{\pi}}) + \log(\vec{\boldsymbol{b}}_u) + \sum_{v\in\mathcal{N}_1(u)} \log(\boldsymbol{R}\vec{\boldsymbol{b}}_v), \tag{4.4}$$

where $\log(\vec{x})$ is a vector whose i-th entry is $(\log(x_i)_i$.

This formulation allow us to make the following interpretation of the classifier. The left hand term represents our belief about the belonging of node u to each of one of the classes knowing its intrinsic features, its first-order neighbors and their corresponding features. Each entry of the vector represents a class. The higher the value of an entry, the more likely it is that a node belongs to the corresponding class. The right hand term is composed of three parts. The term involving π represents the likelihood of belonging to each of the classes based on the available labelled nodes in the training set. The second term can be interpreted as our initial belief when using only intrinsic features of node u. The third term make it possible to update our initial belief by taking into consideration the information from the neighboring nodes.

Unlike GNN-based methods who propagate features through the edges of the graph and update each node's features by taking into consideration the received features from the neighboring nodes before estimating the likelihood of belonging to each of the classes. In GAB, we first estimate the likelihood that each of the nodes belongs to each one of the classes, then we propagate this belief through the edges by taking into consideration the degree of impurity of the graph through the use of the transition matrix $\boldsymbol{R}$.

4.5.2 Node similarity as a simple defence mechanism for GAB

A part from the benefit of interpretability that the formulation of the previous section provides, it also allow us to introduce a simple defence mechanism against adversarial attacks. In the following, we motivate and describe our defence strategy.

As it was shown in multiple studies [71, 85], the majority of adversarial attacks on the graph structure, and especially for the node classification task, consists of adding malicious edges between unrelated nodes, i.e. nodes that belong to different classes and whose features are dissimilar. Adding these edges contribute to misleading the classifiers into assigning the wrong class to the attacked nodes. This is even more true for models that are based on the aggregation of features from neighbouring nodes such as GCN based classifiers.

As seen in section 4.4.2, data purification techniques consist of comparing adjacent nodes' similarity to remove suspicious links that connect dissimilar nodes. Although these kind of mechanisms were shown to be effective in defending against graph attacks, they might delete ordinary edges, which can have an opposite effect on the decision of the classifiers. We suggest instead to weight the contribution of the neighbors according to their similarities to the node of interest when using GAB. This can directly be applied to our classifier, by modifying Eq 4.4 as follows:

$$\log(\vec{\boldsymbol{p}}_u) = \log(\vec{\boldsymbol{\pi}}) + \log(\vec{\boldsymbol{b}}_u) + \sum_{v \in \mathcal{N}_1(u)} \frac{\mathrm{S}(u,v)}{\sum_{v' \in \mathcal{N}_1(u)} \mathrm{S}(u,v')} \log(\boldsymbol{R}\vec{\boldsymbol{b}}_v), \tag{4.5}$$

where $\mathrm{S}(u,v) = \frac{\boldsymbol{x}_u}{\|\boldsymbol{x}_u\|} \cdot \frac{\boldsymbol{x}_v}{\|\boldsymbol{x}_v\|}$ is the cosine similarity function that measures similarity between intrinsic features of nodes u and v. Note that the expression $\mathrm{S}(u,v)$ in the previous equation is general and can represent any similarity function depending on the nature of nodes' features.

4.6 Experiments

4.6.1 Attack scenario

It is essential to understand potential attackers' capabilities in order to evaluate adversarial robustness. In this experiment, we opt for a commonly used scenario to benchmark both adversarial attacks and countermeasures [97]. That is, attackers are aware of every node, edge, and label in the graph (aside from the labels of the test nodes), but they are oblivious of the target model or defense mechanism. The number of edges that they can perturb Δ_A is limited and is less then a ratio δ_e of all edges in the graph. During training, they are not allowed to make any changes to the original graph. Only a limited number of queries can be used to get predictions from the target model.

Dataset	Nodes	Edges	Features	Classes
Cora	2,680	5,148	302	7
Citeseer	3,191	4,172	768	6

Table 4.1: Statistics of datasets

4.6.2 Attack description

To compare the robustness of GAB with GNN based methods, we opt for the SPEIT attack [98] which can be described as follows. It is based on Enhanced Feature Gradiant Attack and Adversarial Adjacent Matrix Generation. It works by first constructing an attack matrix, then altering the features of the attack nodes by optimizing an attack loss. Many proposed attacks in prior research on graph adversarial attacks affect features and connections simultaneously. Finding the best solution, however, gets challenging as the search space grows. Because of this, the SPEIT attack modifies connections and features in turn. Instead of targeting all test nodes, it ignores test nodes that are already challenging to classify in favor of those that are likely to be accurately classified. Since, it does not have access to test nodes' labels, this strategy uses a number of models to predict the labels and look for their most shared predictions. The idea is that a node's similar classification across many models is most likely the result of its topology or feature properties. It is thus interesting to select these nodes and alter their features and/or their neighbourhood.

4.6.3 Data description

Datasets We use *"grb-cora"* and *"grb-citeseer"*, which are modified versions of *"cora"* and *"citeseer"* citation networks that were presented in previous chapters. This refined version [102] removes duplicated nodes and use pre-trained word embedding as initial node features instead of the binary dictionary of the original version. Table 4.1 shows statistics of datasets.

We also follow the data splitting scheme that was proposed in [97]. It is founded on the observation that was made in [103] which states that nodes with a small number of neighbors are more vulnerable to adversarial attacks. They thus suggest to divide datatsets into subsets with different degree distributions. Each of the sets provides a particular degree of difficulty, More precisely, we use four test sets named *"Easy"*, *"Medium"*, *"Hard"* and *"Full"*. The latter contains all test nodes.

	grb-cora				grb-citeseer			
	Easy	**Medium**	**Hard**	**Full**	**Easy**	**Medium**	**Hard**	**Full**
GCN	48.1	69.2	88.6	59.4	40.5	49.5	61.6	48.7
GAB⋆	65.3	77.4	89.2	73.7	56.4	62.7	74.6	64.5

Table 4.2: GCN vs GAB : Effect of SPEIT attack [98] on the node classification accuracy (%) without using any defence mechanism

4.6.4 Results

We conduct several experiments to compare the adversarial attack robustness of GNN-based models with that of GAB. We report the results in terms of accuracy obtained on several test sets with the different levels of difficulty described above.

Table 4.2 shows results of that comparison when applying the models without the use of any defence mechanism. The results demonstrate clearly that GAB is significantly more robust than GCN. This is due to the Bayesian nature of GAB that updates its belief on the class to which a node belongs when using features of its neighbors by taking into consideration the degree of impurity of the graph (see Eq. 4.5.1). When an adversarial edge is added to the graph, it usually connect the attacked node to a node from a different class, GAB is designed to give less importance to nodes that re more likely to belong to a different class. Which make it by design more robust than GCN. This was also shown in Figure 3.1 in the previous chapter. Although the purpose of chapter 3 was not to study the robustness to adversarial attacks, adding random edges between nodes from different classes can be interpreted as a simple untargeted attack. Results in Table 4.2 are consistent with those in Figure 3.1 although now we are using a more sophisticated attack than the random perturbation in the previous chapter.

On the other hand, table 4.3 shows accuracy on the node classification task of a "fortified" version of GCN called RobustGCN [99] and of the version of GAB that was defined in Eq. 4.5 which we refer to as RobustGAB⋆. Results hows improvement of both methods when compared models with no defences. RobustGAB⋆ is still slightly better then RobustGCN. It is worth noticing that the purpose of this work is to study the robustness of GAB to adversarial attacks. Thus, we have not conduct extensive experiment and comparison with all available defence mechanisms in the literature. A classical example is adversarial training which can be applied to any classifier since it modifies the training data and not the actual model, and it was shown to be effective in increasing the robustness of all models.

4.7 Conclusion

The robustness of node classifiers to adversarial attacks is a crucial concern for the use of these techniques in practical applications. The focus of this chapter is to study the robust-

	grb-cora				grb-citeseer			
	Easy	**Medium**	**Hard**	**Full**	**Easy**	**Medium**	**Hard**	**Full**
RobustGCN [99]	74.6	81.2	88.4	79.8	52.3	61.6	77.1	59.3
RobustGAB⋆	73.2	80.1	88.1	80.2	58.1	64.3	76.1	66.2

Table 4.3: GCN vs GAB : Effect of SPEIT attack [98] on the node classification accuracy (%) using defence mechanisms

ness of the Bayesian classifier we proposed. We show that our classifier can resist these attacks better than GNN-based methods. Then, based on the similarity between the intrinsic features of the nodes, we suggest novel defense methods to enhance its robustness.

Conclusion and perspectives

Conclusion

The research carried out for this thesis has made significant advancements in the field of graph learning. Our main goal was to contribute to the progress of graph learning techniques, specifically to the task of node classification. We achieved this by proposing novel graph-structured data processing techniques.

The first contribution is a method we called GraphCL. It is a general contrastive learning framework for learning nodes' representations in a self-supervised way. It learns node embeddings by maximizing the similarity between the representations of two randomly perturbed versions of the same graph. It starts by generating two representations of the same node using graph neural networks, then leverages a contrastive learning loss to optimize agreement between them.

The second contribution is related to the robustness of node classifiers to structural noise. Starting from the observation that GNNs perform differently based on the dataset, we examined the reason for this. We showed that these variations are related to the degree of impurity of the graph then proposed a novel Bayesian node classifier that takes into account the degree of impurity of graphs when classifying nodes.

The third contribution is about the robustness of node classifiers to adversarial attacks, a critical issue for the deployment of these methods in real world applications. We demonstrate that our classifier is more robust to these attacks than GNN-based techniques. Then, in order to increase its robustness, we propose novel defense mechanisms.

Perspectives

In future works, the following concerns deserve to be addressed.

1. In chapter 4, we studied the robustness of the Bayesian classifier that we proposed to adversarial attacks. We limited ourselves to first order classifiers. We had therefore started to analyze the robustness of higher order classifiers to this kind of attacks. The question of the robustness of prediction models being crucial for their deployment in critical applications, it is necessary for us to analyze more deeply the robustness of the models we propose. In the continuity of our work, we plan to pursue this analysis using other types of attacks. In particular the recently proposed injection methods which are more realistic because they are based on the injection of new nodes to graphs [103]. On a social network for example, it is easier to create a new malicious user and connect him to other users (by following other accounts for example), than to modify the properties or links of existing users.

2. Next, we would like to tackle the problem of link prediction. The most powerful

methods today are based on GNN. One of the approaches, for example, is to first use GNNs to obtain representations of nodes in a self supervised manner, and then estimate the probability of the existence of links between pairs of nodes based on their similarity. Another method consists in using GNNs to obtain a representation of the subgraph around the link we are interested in, and then use this representation to decide if the link exists or not [95]. Previous methods look for proprieties such as the presence or not of cycles, triangles or cliques in the subgraphs around the links to determine their probability of existence. We instead would like to take a different path by building on the Bayesian approach we proposed for node classification in chapter 3 to develop Bayesian models for link prediction.

3. Another research direction that interests us is the learning of representations in an unsupervised way. The framework we presented in chapter 2 is based on contrastive learning. This approach allowed us to obtain good quality node representations (i.e. representations that proved useful for the node classification task) and to minimize the need for manual annotation. However, this method requires the use of negative samples. Generating these examples and integrating them into the learning process can be very costly in terms of computation time and can even become unfeasible when dealing with very large graphs. Therefore, it is of great importance to improve the performance and efficiency of self-supervised learning methods to accelerate their deployment in real applications. This can be done either by optimizing the process of generating and using negative samples, or by developing other self-supervised approaches that do not rely on negative samples. We plan to explore both of these directions in the future.

Appendix A

A.1 Derivations for Eq. (3.3)

We first focus on the term $Q_u^{(1)}(k)$. We get

$$\begin{aligned} Q_u^{(1)}(k) &= P(\mathcal{X}_u|y_u = k, \mathcal{I}_\mathcal{G}) \\ &= \sum_{\{k_v\}_{v\in\mathcal{V}_u}} P(\mathcal{X}_u|y_u = k, \{y_v = k_v\}_{v\in\mathcal{V}_u}, \mathcal{I}_\mathcal{G}) \\ &\times \; p(\{y_v = k_v\}_{v\in\mathcal{V}_u}|y_u = k, \mathcal{I}_\mathcal{G}) \end{aligned}$$

As the classes of the neighbors are given, the information on the graph becomes redundant and so useless. Therefore $\mathcal{I}_\mathcal{G}$ can be removed from the first term.

$$\begin{aligned} Q_u^{(1)}(k) &= \sum_{\{k_v\}_{v\in\mathcal{V}_u}} P(\mathcal{X}_u|y_u = k, \{y_v = k_v\}_{v\in\mathcal{V}_u}) \\ &\times \; p(\{y_v = k_v\}_{v\in\mathcal{V}_u}|y_u = k, \mathcal{I}_\mathcal{G}) \end{aligned}$$

Given the classes, the samples of each node are run independently, so we get

$$\begin{aligned} Q_u^{(1)}(k) &= \sum_{\{k_v\}_{v\in\mathcal{V}_u}} \prod_{v'\in\mathcal{V}_u} P(\boldsymbol{x}_{v'}|y_u = k, \\ &\qquad \{y_v = k_v\}_{v\in\mathcal{V}_u}) \\ &\times \; p(\{y_v = k_v\}_{v\in\mathcal{V}_u}|y_u = k, \mathcal{I}_\mathcal{G}) \\ &= \sum_{\{k_v\}_{v\in\mathcal{V}_u}} P(\boldsymbol{x}_u|y_u = k) \\ &\times \; \prod_{v'\in\mathcal{V}_u} P(\boldsymbol{x}_{v'}|y_{v'} = k_{v'}) \\ &\times \; p(\{y_v = k_v\}_{v\in\mathcal{V}_u}|y_u = k, \mathcal{I}_\mathcal{G}) \\ &= D_k(\boldsymbol{x}_u) \sum_{\{k_v\}_{v\in\mathcal{V}_u}} \prod_{v'\in\mathcal{V}_u\setminus\{u\}} D_{k_{v'}}(\boldsymbol{x}_{v'}) \\ &\times \; p(\{y_v = k_v\}_{v\in\mathcal{V}_u}|y_u = k, \mathcal{I}_\mathcal{G}). \end{aligned}$$

which concludes the derivations by doing a re-ordering.

A.2 Derivations for Eq. (3.12)

- Let us start with $d = 2$:

$$\begin{aligned} r^{(2)}(k,k') &= P(y_v = k'|y_u = k, \mathcal{C}_2) \\ &= \sum_{k''=1}^{K} P(y_v = k'|y_u = k, y_w = k'', \mathcal{C}_2) \\ &\times \; P(y_w = k''|y_u = k, \mathcal{C}_1) \end{aligned}$$

  As the class of w is known, the information on the class of u becomes useless and only the fact that v and w are 1-hop neighbor remains also important. Therefore, we obtain

$$\begin{aligned} r^{(2)}(k,k') &= \sum_{k''=1}^{K} P(y_v = k'|y_w = k'', \mathcal{C}_1).r^{(1)}(k,k'') \\ &= \sum_{k''=1}^{K} r^{(1)}(k'',k').r^{(1)}(k,k''). \end{aligned} \tag{A.1}$$

  where w is the node connecting u and v. This node w exists since u and v are 2-hop connected. According to Eq. (A.1), we have

$$\mathbf{R}^{(2)} = \mathbf{R}^2.$$

- For any d, we have,

$$\begin{aligned} r^{(d)}(k,k') &= P(y_v = k'|y_u = k, \mathcal{C}_d) \\ &= \sum_{k''=1}^{K} P(y_v = k'|y_u = k, y_w = k'', \mathcal{C}_1) \\ &\times \; P(y_w = k''|y_u = k, \mathcal{C}_{d-1}) \\ &= \sum_{k''=1}^{K} P(y_v = k'|y_w = k'', \mathcal{C}_1).r^{(d-1)}(k,k'') \\ &= \sum_{k''=1}^{K} r^{(1)}(k'',k').r^{(d-1)}(k,k''). \end{aligned}$$

  Therefore

$$\mathbf{R}^{(d)} = \mathbf{R}^{(d-1)}\mathbf{R}.$$

- Finally, by induction, we conclude the derivations.

Bibliography

[1] Emmanuel Abbe. "Community detection and stochastic block models: recent developments". In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 6446–6531.

[2] Sami Abu-El-Haija et al. "MixHop: Higher-Order Graph Convolutional Architectures via Sparsified Neighborhood Mixing". In: *Proceedings of the 36th International Conference on Machine Learning*. 2019, pp. 21–29.

[3] James Atwood and Don Towsley. "Diffusion-convolutional neural networks". In: *Advances in Neural Information Processing Systems*. 2016, pp. 1993–2001.

[4] Stephen P Borgatti, Martin G Everett, and Jeffrey C Johnson. *Analyzing social networks*. Sage, 2018.

[5] Michael M Bronstein et al. "Geometric deep learning: going beyond Euclidean data". In: *IEEE Signal Processing Magazine* 34.4 (2017), pp. 18–42.

[6] Joan Bruna et al. "Spectral networks and locally connected networks on graphs". In: *arXiv preprint arXiv:1312.6203* (2013).

[7] Nicholas Carlini and David Wagner. "Towards evaluating the robustness of neural networks". In: *2017 ieee symposium on security and privacy (sp)*. Ieee. 2017, pp. 39–57.

[8] Soumen Chakrabarti, Byron Dom, and Piotr Indyk. "Enhanced hypertext categorization using hyperlinks". In: *Acm Sigmod Record* 27.2 (1998), pp. 307–318.

[9] Ines Chami et al. "Hyperbolic graph convolutional neural networks". In: *Advances in Neural Information Processing Systems*. 2019, pp. 4869–4880.

[10] Jie Chen, Tengfei Ma, and Cao Xiao. "FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling". In: *arXiv e-prints*, arXiv:1801.10247 (Jan. 2018), arXiv:1801.10247. arXiv: 1801.10247 [cs.LG].

[11] Jinyin Chen et al. "Link prediction adversarial attack". In: *arXiv preprint arXiv:1810.01110* (2018).

[12] Ting Chen and Lala Li. "Intriguing Properties of Contrastive Losses". In: *arXiv e-prints*, arXiv:2011.02803 (Nov. 2020), arXiv:2011.02803. arXiv: 2011.02803 [cs.LG].

[13] Ting Chen et al. "A Simple Framework for Contrastive Learning of Visual Representations". In: *arXiv e-prints*, arXiv:2002.05709 (Feb. 2020), arXiv:2002.05709. arXiv: 2002.05709 [cs.LG].

[14] Xinlei Chen and Kaiming He. "Exploring Simple Siamese Representation Learning". In: *arXiv e-prints*, arXiv:2011.10566 (Nov. 2020), arXiv:2011.10566. arXiv: 2011.10566 [cs.CV].

[15] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)". In: *arXiv e-prints*, arXiv:1511.07289 (Nov. 2015), arXiv:1511.07289. arXiv: 1511.07289 [cs.LG].

[16] Hanjun Dai et al. “Adversarial attack on graph structured data”. In: *International conference on machine learning*. PMLR. 2018, pp. 1115–1124.

[17] Nilesh Dalvi et al. “Adversarial classification”. In: *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2004, pp. 99–108.

[18] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. “Convolutional neural networks on graphs with fast localized spectral filtering”. In: *Advances in neural information processing systems*. 2016, pp. 3844–3852.

[19] R Devon Hjelm et al. “Learning deep representations by mutual information estimation and maximization”. In: *arXiv e-prints*, arXiv:1808.06670 (Aug. 2018), arXiv:1808.06670. arXiv: 1808.06670 [stat.ML].

[20] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. “metapath2vec: Scalable representation learning for heterogeneous networks”. In: *international conference on knowledge discovery and data mining*. 2017, pp. 135–144.

[21] Alberto Garcia Duran and Mathias Niepert. “Learning graph representations with embedding propagation”. In: *Advances in Neural information Processing Systems*. 2017, pp. 5119–5130.

[22] Negin Entezari et al. “All you need is low (rank) defending against adversarial attacks on graphs”. In: *Proceedings of the 13th International Conference on Web Search and Data Mining*. 2020, pp. 169–177.

[23] Fuli Feng et al. “Graph adversarial training: Dynamically regularizing based on graph structure”. In: *IEEE Transactions on Knowledge and Data Engineering* 33.6 (2019), pp. 2493–2504.

[24] Matthias Fey and Jan Eric Lenssen. “Fast Graph Representation Learning with PyTorch Geometric”. In: *arXiv e-prints*, arXiv:1903.02428 (Mar. 2019), arXiv:1903.02428. arXiv: 1903.02428 [cs.LG].

[25] Thomas Gaudelet et al. “Utilizing graph machine learning within drug discovery and development”. In: *Briefings in bioinformatics* 22.6 (2021), pp. 1–22.

[26] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *International Conference on Artificial Intelligence and Statistics*. 2010, pp. 249–256.

[27] Ian Goodfellow et al. “Generative adversarial networks”. In: *Communications of the ACM* 63.11 (2020), pp. 139–144.

[28] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and harnessing adversarial examples”. In: *arXiv preprint arXiv:1412.6572* (2014).

[29] Marco Gori, Gabriele Monfardini, and Franco Scarselli. “A new model for learning in graph domains”. In: *Proceedings. 2005 IEEE international joint conference on neural networks*. Vol. 2. 2005. 2005, pp. 729–734.

[30] Jean-Bastien Grill et al. “Bootstrap your own latent-a new approach to self-supervised learning”. In: *Advances in Neural Information Processing Systems*. Vol. 33. 2020.

[31] Aditya Grover and Jure Leskovec. “node2vec: Scalable feature learning for networks”. In: *International Conference on Knowledge Discovery and Data Mining*. 2016, pp. 855–864.

[32] Michael Gutmann and Aapo Hyvärinen. “Noise-contrastive estimation: A new estimation principle for unnormalized statistical models”. In: *Conference on Artificial Intelligence and Statistics*. 2010, pp. 297–304.

[33] Hakim Hafidi et al. “Bayesian Node Classification for Noisy Graphs”. In: *2021 IEEE Statistical Signal Processing Workshop (SSP)*. IEEE. 2021, pp. 246–250.

[34] Hakim Hafidi et al. “Graph-Assisted Bayesian Node Classifiers”. In: *IEEE Access* (2022 (under review)).

[35] Hakim Hafidi et al. “GraphCL: Contrastive Self-Supervised Learning of Graph Representations”. In: *arXiv e-prints*, arXiv:2007.08025 (July 2020), arXiv:2007.08025. arXiv: 2007.08025 [cs.LG].

[36] Hakim Hafidi et al. “Negative sampling strategies for contrastive self-supervised learning of graph representations”. In: *Signal Processing* 190 (2022), p. 108310.

[37] Hakim Hafidi et al. “On the Robustness of Bayesian Graph Classifiers to Adversarial Attacks”. In: *IEEE Access* (2022 (To be submitted)).

[38] Will Hamilton, Zhitao Ying, and Jure Leskovec. “Inductive representation learning on large graphs”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 1024–1034.

[39] Kaveh Hassani and Amir Hosein Khasahmadi. “Contrastive Multi-View Representation Learning on Graphs”. In: *arXiv e-prints*, arXiv:2006.05582 (June 2020), arXiv:2006.05582. arXiv: 2006.05582 [cs.LG].

[40] Kaiming He et al. “Momentum contrast for unsupervised visual representation learning”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 9729–9738.

[41] Mikael Henaff, Joan Bruna, and Yann LeCun. “Deep convolutional networks on graph-structured data”. In: *arXiv preprint arXiv:1506.05163* (2015).

[42] Peter D Hoff, Adrian E Raftery, and Mark S Handcock. “Latent space approaches to social network analysis”. In: *Journal of the American Statistical Association* 97.460 (2002), pp. 1090–1098.

[43] Weihua Hu et al. “Open graph benchmark: Datasets for machine learning on graphs”. In: *arXiv preprint arXiv:2005.00687* (2020).

[44] Junteng Jia and Austion R Benson. “Residual correlation in graph neural network regression”. In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2020, pp. 588–598.

[45] Junteng Jia et al. “Graph Belief Propagation Networks”. In: *arXiv preprint arXiv:2106.03033* (2021).

[46] Wei Jin et al. “Graph structure learning for robust graph neural networks”. In: *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 2020, pp. 66–74.

[47] Yannis Kalantidis et al. “Hard negative mixing for contrastive learning”. In: *Advances in Neural Information Processing Systems*. Vol. 33. 2020.

[48] Brian Karrer and Mark EJ Newman. “Stochastic blockmodels and community structure in networks”. In: *Physical review E* 83.1 (2011), p. 016107.

[49] Anish Khazane et al. “DeepTrax: Embedding Graphs of Financial Transactions”. In: *IEEE International Conference On Machine Learning And Applications (ICMLA)*. 2019, pp. 126–133.

[50] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *arXiv e-prints*, arXiv:1412.6980 (Dec. 2014), arXiv:1412.6980. arXiv: 1412.6980 [cs.LG].

[51] Thomas N. Kipf and Max Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *arXiv e-prints*, arXiv:1609.02907 (Sept. 2016), arXiv:1609.02907. arXiv: 1609.02907 [cs.LG].

[52] Thomas N. Kipf and Max Welling. “Variational Graph Auto-Encoders”. In: *arXiv e-prints*, arXiv:1611.07308 (Nov. 2016), arXiv:1611.07308. arXiv: 1611.07308 [stat.ML].

[53] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. “Predict then propagate: Graph neural networks meet personalized pagerank”. In: *arXiv preprint arXiv:1810.05997* (2018).

[54] Xin Li et al. “Gene function prediction with gene interaction networks: a context graph kernel approach”. In: *IEEE Transactions on Information Technology in Biomedicine* 14.1 (2009), pp. 119–128.

[55] Yanlin Li et al. “Supervised Graph Representation Learning for Modeling the Relationship between Structural and Functional Brain Connectivity”. In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, pp. 9065–9069.

[56] Dong C Liu and Jorge Nocedal. “On the limited memory BFGS method for large scale optimization”. In: *Mathematical programming* 45.1 (1989), pp. 503–528.

[57] Sitao Luan et al. “Break the Ceiling: Stronger Multi-scale Deep Graph Convolutional Networks”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 10943–10953.

[58] Aleksander Madry et al. “Towards deep learning models resistant to adversarial attacks”. In: *arXiv preprint arXiv:1706.06083* (2017).

[59] David McAllester and Karl Stratos. “Formal Limitations on the Measurement of Mutual Information”. In: *arXiv e-prints*, arXiv:1811.04251 (Nov. 2018), arXiv:1811.04251. arXiv: 1811.04251 [cs.IT].

[60] Miller McPherson, Lynn Smith-Lovin, and James M Cook. “Birds of a feather: Homophily in social networks”. In: *Annual review of sociology* 27.1 (2001), pp. 415–444.

[61] Alessio Micheli. “Neural network for graphs: A contextual constructive approach”. In: *IEEE Transactions on Neural Networks* 20.3 (2009), pp. 498–511.

[62] Tomas Mikolov et al. “Efficient Estimation of Word Representations in Vector Space”. In: *arXiv e-prints*, arXiv:1301.3781 (Jan. 2013), arXiv:1301.3781. arXiv: 1301.3781 [cs.CL].

[63] Christopher Morris et al. “Weisfeiler and Leman go neural: Higher-order graph neural networks”. In: *AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 4602–4609.

[64] Martina Morris. *HIV Transmission Network Metastudy Project: An Archive of Data From Eight Network Studies, 1988–2001*. 2011.

[65] Kevin Murphy, Yair Weiss, and Michael I Jordan. “Loopy belief propagation for approximate inference: An empirical study”. In: *arXiv preprint arXiv:1301.6725* (2013).

[66] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 8024–8035.

[67] Tiago P Peixoto. “Bayesian stochastic blockmodeling”. In: *Advances in network clustering and blockmodeling* (2019), pp. 289–332.

[68] Jeffrey Pennington, Richard Socher, and Christopher D Manning. “Glove: Global vectors for word representation”. In: *Conference on Empirical Methods in Natural Language Processing*. 2014, pp. 1532–1543.

[69] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. “Deepwalk: Online learning of social representations”. In: *International Conference on Knowledge Discovery and Data Mining*. 2014, pp. 701–710.

[70] Erhard Rahm and Andreas Thor. “Citation analysis of database publications”. In: *ACM Sigmod Record* 34.4 (2005), pp. 48–53.

[71] Alan Said, Ernesto W De Luca, and Sahin Albayrak. “How social relationships affect user similarities”. In: *Proceedings of the International Conference on Intelligent User Interfaces Workshop on Social Recommender Systems, Hong Kong*. 2010.

[72] Alvaro Sanchez-Gonzalez et al. “Graph networks as learnable physics engines for inference and control”. In: *International Conference on Machine Learning*. 2018, pp. 4470–4479.

[73] Prithviraj Sen et al. “Collective classification in network data”. In: *AI magazine* 29.3 (2008), pp. 93–93.

[74] Oleksandr Shchur et al. “Pitfalls of graph neural network evaluation”. In: *arXiv preprint arXiv:1811.05868* (2018).

[75] Arnab Sinha et al. “An overview of microsoft academic service (mas) and applications”. In: *Proceedings of the 24th international conference on world wide web*. 2015, pp. 243–246.

[76] Christian Szegedy et al. “Intriguing properties of neural networks”. In: *arXiv preprint arXiv:1312.6199* (2013).

[77] Kiran K Thekumparampil et al. “Attention-based graph neural network for semi-supervised learning”. In: *International Conference on Learning Representations* (2018).

[78] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. “Representation Learning with Contrastive Predictive Coding”. In: *arXiv e-prints*, arXiv:1807.03748 (July 2018), arXiv:1807.03748. arXiv: 1807.03748 [cs.LG].

[79] Petar Veličković et al. “Graph Attention Networks”. In: *International Conference on Learning Representations* (2018). URL: https://openreview.net/forum?id=rJXMpikCZ.

[80] Petar Veličković et al. “Deep Graph Infomax”. In: *arXiv e-prints*, arXiv:1809.10341 (Sept. 2018), arXiv:1809.10341. arXiv: 1809.10341 [stat.ML].

[81] Petar Veličković et al. “Graph Attention Networks”. In: *arXiv e-prints*, arXiv:1710.10903 (Oct. 2017), arXiv:1710.10903. arXiv: 1710.10903 [stat.ML].

[82] Daixin Wang, Peng Cui, and Wenwu Zhu. “Structural deep network embedding”. In: *International Conference on Knowledge Discovery and Data mining*. 2016, pp. 1225–1234.

[83] Tongzhou Wang and Phillip Isola. “Understanding Contrastive Representation Learning through Alignment and Uniformity on the Hypersphere”. In: *arXiv e-prints*, arXiv:2005.10242 (May 2020), arXiv:2005.10242. arXiv: 2005.10242 [cs.LG].

[84] Chao-Yuan Wu et al. “Sampling matters in deep embedding learning”. In: *IEEE International Conference on Computer Vision*. 2017, pp. 2840–2848.

[85] Huijun Wu et al. “Adversarial examples on graph data: Deep insights into attack and defense”. In: *arXiv preprint arXiv:1903.01610* (2019).

[86] Mike Wu et al. “Conditional Negative Sampling for Contrastive Learning of Visual Representations”. In: *arXiv e-prints*, arXiv:2010.02037 (Oct. 2020), arXiv:2010.02037. arXiv: 2010.02037 [cs.LG].

[87] Chaowei Xiao et al. “Generating adversarial examples with adversarial networks”. In: *arXiv preprint arXiv:1801.02610* (2018).

[88] Kaidi Xu et al. “Topology attack and defense for graph neural networks: An optimization perspective”. In: *arXiv preprint arXiv:1906.04214* (2019).

[89] Keyulu Xu et al. “How Powerful are Graph Neural Networks?” In: *arXiv e-prints*, arXiv:1810.00826 (Oct. 2018), arXiv:1810.00826. arXiv: 1810.00826 [cs.LG].

[90] Yuning You et al. “Graph Contrastive Learning with Augmentations”. In: *Advances in Neural Information Processing Systems*. Vol. 33. 2020.

[91] Yuhui Yuan, Kuiyuan Yang, and Chao Zhang. “Hard-aware deeply cascaded embedding”. In: *IEEE international conference on computer vision*. 2017, pp. 814–823.

[92] Chuxu Zhang, Ananthram Swami, and Nitesh V Chawla. “Shne: Representation learning for semantic-associated heterogeneous networks”. In: *International Conference on Web Search and Data Mining*. 2019, pp. 690–698.

[93] Chuxu Zhang et al. “Heterogeneous graph neural network”. In: *International Conference on Knowledge Discovery & Data Mining*. 2019, pp. 793–803.

[94] Jiani Zhang et al. “GaAN: Gated Attention Networks for Learning on Large and Spatiotemporal Graphs”. In: *arXiv e-prints*, arXiv:1803.07294 (Mar. 2018), arXiv:1803.07294. arXiv: 1803.07294 [cs.LG].

[95] Muhan Zhang and Yixin Chen. “Link prediction based on graph neural networks”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 5165–5175.

[96] Muhan Zhang et al. “An end-to-end deep learning architecture for graph classification”. In: *Thirty-second AAAI conference on artificial intelligence*. 2018.

[97] Qinkai Zheng et al. “Graph robustness benchmark: Benchmarking the adversarial robustness of graph machine learning”. In: *arXiv preprint arXiv:2111.04314* (2021).

[98] Qinkai Zheng et al. *KDD CUP 2020 ML Track 2 Adversarial Attacks and Defense on Academic Graph 1st Place Solution*. 2020. URL: https://github.com/Stanislas0/KDD_CUP_2020_MLTrack2_SPEIT.

[99] Dingyuan Zhu et al. “Robust graph convolutional networks against adversarial attacks”. In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 1399–1407.

[100] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. “Semi-supervised learning using gaussian fields and harmonic functions”. In: *International Conference on Machine Learning*. 2003, pp. 912–919.

[101] Marinka Zitnik and Jure Leskovec. “Predicting multicellular function through multilayer tissue networks”. In: *Bioinformatics* 33.14 (2017), pp. i190–i198.

[102] Xu Zou et al. “Dimensional reweighting graph convolutional networks”. In: *arXiv preprint arXiv:1907.02237* (2019).

[103] Xu Zou et al. “TDGIA: Effective injection attacks on graph neural networks”. In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2021, pp. 2461–2471.

[104] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. “Adversarial attacks on neural networks for graph data”. In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2018, pp. 2847–2856.