Università degli Studi di Padova

Head Office: Università degli Studi di Padova
Department: Information Engineering

Ph.D. course in: Information Engineering
Curriculum: Information and Communication Technologies (ICT)
Series: XV, 2023

Improving the Effectiveness of Graph Neural Networks in Practical Scenarios

Thesis written with the financial contribution of
Ministero dell'Istruzione e del Merito - Miur.

Coordinator: Ch.mo Prof. Andrea Neviani
Supervisor: Ch.mo Prof. Fabio Vandin

Ph.D. student: Davide Buffelli

Abstract

In the past decade, deep learning has given new life to the field of artificial intelligence, providing many breakthroughs in areas like computer vision, natural language processing, audio, game-playing, and biology. The past few years have seen a particular interest in developing and applying deep learning models that can operate on graph-structured data, also called *graph neural networks (GNNs).* This is not surprising, as graphs are core data structures that are used to model relationships between different entities, which is a common scenario that appears in molecules, social networks, and physical interactions, just to cite a few.

GNNs have achieved many successes, and have been studied from both a theoretical, and a practical point of view. However, several questions remain unanswered. In this thesis, we focus on three open questions regarding *practical* aspects of GNNs, and we propose effective solutions for tackling them.

The first question concerns *global structural information*, i.e., information that depends on the global structure of the graph. This kind of information is particularly difficult to capture for GNNs, which are targeted towards local interactions. While global structural information has been previously overlooked, we show that it has an important impact on practical applications, and we propose a regularization strategy to provide this information to GNNs during training.

The second question concerns *size-generalization*, which is the ability of GNNs to generalize from small to large graphs. While GNNs are designed to operate on graphs of any size, it is observed that when trained on small graphs, they struggle at generalizing to large graphs. This is particularly problematic, as in certain domains, obtaining labels for large graphs is prohibitive. Furthermore, training on large graphs may require expensive computational resources. We propose a novel regularization strategy, that can be applied on any GNN, and that can improve size-generalization capabilities of up to 30%.

The third question focuses on *multi-task* settings. GNNs work by exchanging messages between nodes, and using learnable functions to produce node embeddings that encode structural and feature-related information. During training, GNNs tend to optimize the produced embeddings to the training loss, making it hard to reuse them effectively for different tasks. This requires the training of multiple models, and the use of different embeddings for different tasks. We propose a training strategy based on meta-learning that provides a single set of embeddings that can be used to perform multiple tasks while achieving performance comparable to those of single-task end-to-end trained models.

Acknowledgements

First of all, I would like to thank my supervisor, Fabio Vandin. I am extremely grateful for the freedom he has given me in choosing the research topics that would most interest me, and for being a great person that truly cares about others. Working with Fabio has taught me many skills that I will treasure for the rest of my career in research.

During my PhD I had the pleasure to share the office at the University of Padova with so many talented people: Andrea Tonon, Antonio Collesei, Dario Simionato, Diego Santoro, Ilie Sarpe, Leonardo Pellegrina, and Thach Văn Bùi. While I am grateful to every single one of them, a special thanks goes to Diego and Ilie. We started the PhD together, and we are now finishing it together. Sharing experiences, frustrations, and chats with you has been an important component of this journey.

During the PhD I had the immense luck of meeting many great researchers that have played a big role in my professional development.

While the global pandemic has created a lot of challenges, and forced everyone to work from home, I was very lucky to be able to do an internship at Samsung AI in Cambridge (UK). I would like to thank Efthymia Tsamoura, for supervising me during the internship, and for transmitting her dedication to research. While the pandemic forced everyone to work from home, and meeting people was not easy, I would still like to thank the very talented people I met at Samsung AI: Antoine Toisoul, Brais Martinez, Da Li, Jan Stuhmer, Juan-Manuel Perez-Rua, Shell Xu Hu, Timothy Hospedales, and Vladimir Pavlovic. Moving to Cambridge at the peak of the pandemic has been really challenging, and I would like to give a huge thanks to Alex Bailey, for being incredibly kind and for going the extra mile to help me in this process.

I am immensely grateful to Pietro Liò for hosting me in his group at the University of Cambridge. Pietro is one of the most kind, welcoming, and stimulating person, researcher, and professor I know, and I have grown a lot by working with him. I would also like to thank all the amazing people I have met there and that made the experience even better: Alexander Norcliffe, Antonio Longa, Arian Jamasb, Carlos Purves, Cătălina Cangea, Chaitanya Joshi, Charlie Harris, Charlotte Magister, Cristian Bodnar, Dobrik Georgiev, Emanuele Rossi, Gabriele Ciravegna, Gianluca Mittone, Iulia Duta, Luna Pianesi, Mateo Espinosa Zarlenga, Niccolò Pancino, Nikola Simidjievsky, Paul Scherer, Pietro Barbiero, Shyam Tailor, Simon Mathis, and Steve Azzolin.

I would also like to thank Bastian Rieck, for hosting me at the Helmholtz Institute, and for just being an awesome person. Bastian has a true passion and enthusiasm for research, and I hope to continue learning from him.

Right before the end of my PhD, I had the chance to intern at Meta AI in London. The experience here has been transformative, and I thoroughly enjoyed every second of it. It is quite impressive to see so much talent, knowledge, and great research, all in one place. Here I was lucky to be supervised by Vassilis Plachouras, which I would like to thank for giving me a lot of freedom and trust, and for always asking the right questions. I would also like thank my collaborators Ashish Gupta and Agnieszka Strzalka, for being always available, and open to share their knowledge. I also need to thank the whole team: Abdalgader Abubaker, Ciprian Pătru, Daniel Lobato Garcia, Ekaterina Trimbach, Konstantinos Papakonstantinou, Oleksandr Pryymak, Rishabh Agarwal, Takanori Maehara, Yatharth Saraf, and co-intern Chris Mingard (thank you for all the shared discussions and rants about politics). Everyone made me feel welcome from the very first moment, and working in this team has been an unforgettable and invaluable opportunity.

I want to thank my closest friends, that supported me in so many ways, and without whom I would not be the person I am today: Alberto Baci Ferrari, Alberto Fiocco, Andrea Daldosso, Carlo Alberto Campedelli, Enrico Giacomazzi, Giacomo Lucchi, Giacomo Luisi, Jacopo Montolli, Leonardo Cristini, Michele Scandola, Nicola Scardoni, Piergiorgio Rudella, Riccardo Bonfante, Riccardo Antonelli, Silvio Cardo, and Stefano Signorini. You probably did not even realise it, but your unconditional belief in me, and the many laughs, trips, plates of food, and (maybe some might say too many, but that is questionable) drinks, have been fundamental for overcoming the low moments, and for encouraging me to keep going forward.

I would also like thank my family, for always caring for me and supporting my decisions, and simply for being present when I needed it. Finally, the biggest thank you goes to Cloé, for always being there, at every step of this journey. You have helped me go through the difficult moments, and made the best moments even better. I could not have wished for a better person by my side.

Contents

1 Introduction 1

1.1 Thesis Contributions and Layout . . . 3

2 Preliminaries 5

2.1 Graphs . . . 5

2.1.1 The Weisfeiler-Leman Algorithm . . . 6

2.1.2 Random Walks . . . 8

2.1.3 Graph Coarsening . . . 9

2.2 Representation learning . . . 10

2.2.1 Graph Representation Learning . . . 11

2.3 Graph Neural Networks . . . 12

2.3.1 Examples of Graph Neural Networks . . . 14

2.3.2 Connection to the Weisfeiler-Leman Algorithm . . . 17

2.3.3 Oversmoothing & Oversquashing . . . 18

3 The Impact of Global Structural Information in Graph Neural Networks 19

3.1 Random Walks can empower Graph Neural Networks . . . 20

3.1.1 Empirical Analysis . . . 22

3.2 Injecting Global Information in MPNNs . . . 23

3.2.1 Types of Global Structural Information Injection . . . 24

3.2.2 Choice of Models . . . 26

3.2.3 Training & Implementation Details . . . 27

3.3 Evaluation of the Injection of Global Structural Information . . . 30

3.4 Practical Aspects . . . 33

3.4.1 RWRReg . . . 34

3.4.2 Fast Implementation of the Random Walk with Restart Regularization 37

3.5 Related Work . . . 37

3.6 Conclusions . . . 39

4 Improving Size-Generalization in Graph Neural Networks 41
4.1 Preliminaries . . . 43
4.1.1 Centered Kernel Alignment . . . 43
4.1.2 Central Moment Discrepancy . . . 43
4.2 Our Method . . . 44
4.2.1 Limitations . . . 45
4.3 Analysis of Node Embeddings . . . 46
4.4 Evaluation . . . 48
4.4.1 Results . . . 50
4.4.2 Ablation Study . . . 52
4.5 Related Work . . . 53
4.6 Conclusions . . . 56

5 Learning Multi-Task Representations 57
5.1 Preliminaries . . . 58
5.1.1 Multi-Task Learning . . . 59
5.1.2 Model-Agnostic Meta-Learning and ANIL . . . 59
5.2 SAME: Single-Task Adaptation for Multi-Task Embeddings . . . 60
5.2.1 Episode Design . . . 61
5.2.2 Model Architecture Design . . . 63
5.2.3 Meta-Training Design . . . 63
5.2.4 Connection between SAME and other Optimization-based Meta-Learning Methods . . . 65
5.3 Experiments . . . 66
5.3.1 Experimental Setting . . . 66
5.3.2 Results . . . 68
5.4 Related Work . . . 73
5.5 Conclusions . . . 74

6 Conclusion 75
6.1 Future Work . . . 75

Acronyms 77

Bibliography 79

I Introduction

While we are still far from the advent of intelligent machines that can rival those from the best sci-fi books and movies, the field of Artificial Intelligence (AI) is as active as ever, and new research progress emerges at unprecedented speed. One family of methods in particular deserves a special mention, and it is that of deep learning, which has played a central role in many of the biggest AI accomplishments of the past years: from beating the world champion at GO [194], to enabling autonomous driving, powering virtual assistants, and almost perfectly solving the protein folding problem [113].

Deep learning is a specific class of machine learning models, i.e., models that learn how to perform a given task by observing labelled examples[1]. The class of deep learning models is composed of *deep neural networks*, which are mathematical models loosely inspired by the brain.

While neural networks have a long history [93, 152, 180, 184], 2012 has been the breakthrough year for deep learning, with the introduction of the AlexNet model [128]. AlexNet showed that convolutional neural networks could perform image classification significantly better than any other machine learning method, and that GPUs can dramatically speed up the computations required for these models. The past ten years have only seen deep learning methods rise and become arguably the most popular techniques in the fields of machine learning and AI.

After the breakthrough in computer vision, deep learning methods have started taking over the natural language [12, 204], audio [94, 169], and reinforcement learning [158] domains. These successes have been enabled by a combination of factors including more efficient hardware (and better software support), and increased availability of data. Another important factor to mention regards the advancements in our understanding of how to identify the right *inductive biases* [157] for given tasks. The term "inductive bias" refers to the set of assumptions that are incorporated into the model (either with the choice of architecture, the training procedure, the training objective, or the training data), and that impact how the model generalizes to unseen data.

In fact, convolutional neural networks [72, 131] work particularly well on images because they share parameters across image locations, which makes them invariant to the position of the objects in the image, while recurrent neural networks [95, 184] work particularly well on sequential data,

[1]In contrast to the traditional computer science approach of writing a sequence of instructions outlining how to solve a given task.

as they share parameters across time-steps. However, these kind of models are not suitable for data represented with graphs, and manifolds, where there is no global system of coordinates, or shift-invariance, and, as a results, these domains remained untouched by the first wave of deep learning breakthroughs. Nevertheless, things rapidly changed in 2016, with the "official" birth of the *Geometric Deep Learning* [30] research field[2]. This new area of research focused on developing neural networks that could take as input data that lives in non-Euclidean spaces. One particular case that has received a lot of attention is that of neural networks that can process graph-structured data, also called *Graph Neural Networks (GNNs)*, which are, to this day, one of the most popular keywords in many top tier research venues.

The reason behind the large interest in these methods is that graph-structured data appears in a huge variety of domains. Graphs are used to represent social networks, physical interactions, molecules, transportation networks, meshes, transaction networks, and in general any kind of relational data. Unsurprisingly, GNNs have been applied with success in a large variety of domains (e.g., chemistry [73], biology [123], recommender systems [224], and transportation networks [58], just to cite a few).

At a high level, GNNs operate on the input graph by exchanging messages between connected nodes, while using neural networks to create and process these messages in order to obtain a vector representation (also called *embedding*) for each node that can help perform downstream tasks. A lot of work has been done to study the theoretical properties of GNNs (e.g., identifying the class of graphs that they are able to distinguish [216]), and improve practical aspects (e.g, scaling to billion nodes graphs [67]), but several questions remain open.

The work presented in this thesis aims at introducing new techniques to improve the effectiveness of GNNs in practical applications, and is driven by the following three questions.

1. *Is global structural information important for GNNs applications on graphs, and how can it be exploited by GNNs?* Contrary to other deep learning models, which obtain better performance with deeper architectures [92], GNNs seem to perform best when applied with a relatively small number of layers. This however limits the capability of GNNs of extracting information that depends on the global structure of the graph (more details on this are presented in Chapter 2.3.3).

2. *Can we improve the generalization properties of GNNs from small to large graphs?* In many domains, it is much easier to obtain labels for smaller graphs than it is for larger ones (e.g., in combinatorial optimization problems). Furthermore, training on smaller graphs requires less computational resources. While GNNs are designed to be able to operate on graphs

[2]There was already research in this area, but this was the moment in which the field got united under the umbrella term of "Geometric Deep Learning".

of any size, it is observed empirically that when trained on small graphs they struggle at generalizing to larger ones.

3. *Can we train GNNs to produce embeddings that are effective for multiple tasks?* GNNs tend to produce embeddings that are "optimized" for the task defined by the training loss. When there are multiple downstream tasks to perform, it is then needed to have multiple models (one for each task). However, in many scenarios it is desirable to have a single set of node embeddings which can be used to perform multiple downstream tasks of interest.

1.1 Thesis Contributions and Layout

This thesis starts in Chapter 2, with a background section presenting the main topics on graphs, graph representation learning, and GNNs, that are at the base of the work presented in the later chapters.

The main contributions of this thesis are three studies around practical shortcomings of GNNs, with the proposal of novel solutions for overcoming them. In more detail, we summarize our main contributions as follows.

- In Chapter 3 we tackle the question of whether global information (i.e., information that depends on the global structure of the graph) is needed in GNN applications. We empirically address this question by giving access to global information, such as the statistics of random walks, to several GNN models, and observing the impact it has on downstream performance. Our results show that global information can in fact provide significant benefits for common graph-related tasks. We further identify a novel regularization strategy based on random walks with restart that leads to an average accuracy improvement of more than 5% on all considered tasks, which include both inductive and transductive scenarios. The use of random walks with restart is further supported by a theoretical connection showing they can be used to speed-up the Weisfeiler-Leman algorithm.

- In Chapter 4 we propose a regularization strategy to improve the size-generalization capabilities of GNNs. We consider the scenario in which we only have access to the training data, and we propose a regularization strategy that can be applied to any GNN to improve its generalization capabilities from smaller to larger graphs without requiring access to the test data. Our regularization is based on the idea of simulating a shift in the size of the training graphs using coarsening techniques, and enforcing the model to be robust to such a shift. Experimental results on standard datasets show that popular GNN models, trained on the 50% smallest graphs in the dataset and tested on the 10% largest graphs, obtain performance improvements of up to 30% when trained with our regularization strategy.

- In Chapter 5 we introduce a method based on meta-learning that trains GNNs to produce node embeddings that can be used to perform multiple downstream tasks with performance comparable to those of separate single-task end-to-end trained models. In particular, we exploit the properties of optimization-based meta-learning to learn GNNs that can produce general node representations by learning parameters that can quickly (i.e., with a few steps of gradient descent) adapt to multiple tasks. This is in contrast to traditional multi-task learning approaches that instead try to directly learn to solve multiple tasks concurrently.

The contents of this thesis are based on the following publications completed during the course of my PhD:

- **Davide Buffelli**, Fabio Vandin, "The Impact of Global Structural Information in Graph Neural Networks Applications", *Data*, 2022 [36].

- **Davide Buffelli**, Fabio Vandin, "Graph Representation Learning for Multi-Task Settings: a Meta-Learning Approach", *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2022 [35]. An earlier version of this work was also presented at the *NeurIPS Workshop on Meta-Learning (MetaLearn) 2020* [33].

- **Davide Buffelli**, Pietro Liò, Fabio Vandin, "SizeShiftReg: a Regularization Method for Improving Size-Generalization in Graph Neural Networks", *Proceedings of the Thirty-sixth Conference on Neural Information Processing Systems (NeurIPS)*, 2022 [31].

Other works performed during my PhD studies that however are not part of this thesis are the following (* indicates equal contribution):

- **Davide Buffelli**, Fabio Vandin, "Attention-Based Deep Learning Framework for Human Activity Recognition with User Adaptation", *IEEE Sensors Journal*, 2021 [34].

- **Davide Buffelli***, Efthymia Tsamoura*, "Scalable Theory-Driven Regularization of Scene Graph Generation Models", (to appear in) *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence (AAAI)*, 2023 [32].

- Rishabh Jain, Gabriele Ciravegna, Pietro Barbiero, Francesco Giannini, **Davide Buffelli**, Pietro Liò, "Extending Logic Explained Networks to Text Classification", (to appear in) *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2022 [102].

2 Preliminaries

In this section we introduce the main concepts behind the work presented in this thesis, and we introduce the notation along the way.

Basic Notation. We use lowercase bold letters to indicate vectors $\mathbf{v} \in \mathbb{R}^d$, and uppercase bold letters to indicate matrices $\mathbf{M} \in \mathbb{R}^{n \times m}$. We use the subscript "$i$" to indicate the i-th element of a vector (v_i), while we refer to the element in the i-th row and j-column of a matrix with the subscript "i, j" ($M_{i,j}$). To indicate the i-th row of a matrix we use the subscript "$i, :$" ($\mathbf{M}_{i,:}$), while for the j-th column we use "$:, j$" ($\mathbf{M}_{:,j}$). We use $\mathbf{I}$ to indicate the *identity matrix*, i.e., the matrix where all entries in the main diagonal are equal to 1, and all other entries are zero. We usually do not specify the dimensionality of the identity matrix and imply it is the appropriate one for the context.

2.1 Graphs

Let $G = (V, E)$ be a graph, where V is the set of *nodes* with size n, and $E \subseteq V \times V$ is the set of *edges*. For a node $v \in V$, we use $\mathcal{N}_v$ to indicate the set of its *one-hop neighbours* (i.e., the nodes connected to v by an edge). The size of the neighbourhood of a node is called its *degree* (indicated with *degree*$(v) = |\mathcal{N}_v|$), and it is common to define the *degree matrix* as the diagonal matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$ with

$$\begin{cases} D_{i,i} = \textit{degree}(i) & \text{for } i = 1, \dots, n \\ D_{i,j} = 0 \quad \text{for } i, j = 1, \dots, n, i \neq j \end{cases} .$$

A *path* in a graph is a sequence of distinct nodes $v_1, v_2, \dots, v_k$ with $(v_i, v_{i+1}) \in E \; \forall i = 1, \dots, k-1$, and we say it has length $k-1$ if it traverses $k-1$ edges. A *walk* instead is a sequence of nodes $v_1, v_2, \dots, v_k$ with $(v_i, v_{i+1}) \in E \; \forall i = 1, \dots, k-1$ in which nodes (and edges) do not need to be distinct. We say two nodes u and v are at distance k in the graph if there is a path of length k that starts at u and ends at v and no path of length shorter than k exists between the two nodes. When we use the term "i-hop neighbourhood" of a node v, we refer to the set of nodes with distance at most i from v. An *attributed* graph $\hat{G} = (V, E, X)$, is a graph in which every node has an associated feature vector. In particular, X is a function that maps each node to its feature vector $X : V \rightarrow \mathbb{R}^d$. In this thesis we focus on attributed graphs in which the edges do not have a direction, as this is the most common scenario in the GNN literature. The presented methods can however be easily extended to graphs in which the attributes are also on the edges, or in which the edges are directed.

There are multiple ways to represent a graph in a format that allows efficient manipulations on a computer, each one with specific properties and applications [153]. We represent an attributed graph $\hat{G}$ with a tuple $(\mathbf{A}, \mathbf{X})$, where $\mathbf{A} \in \{0,1\}^{n \times n}$ is the adjacency matrix (i.e., the element in position i, j is equal to 1 if there is an edge between nodes i and j, and zero otherwise), and $\mathbf{X} \in \mathbb{R}^{n \times d}$ is the feature matrix, where the i-th row, $\mathbf{X}_{i,:}$, contains the d-dimensional feature vector for node i. An example illustration of the representation format of an attributed graph can be found in Figure 2.1.

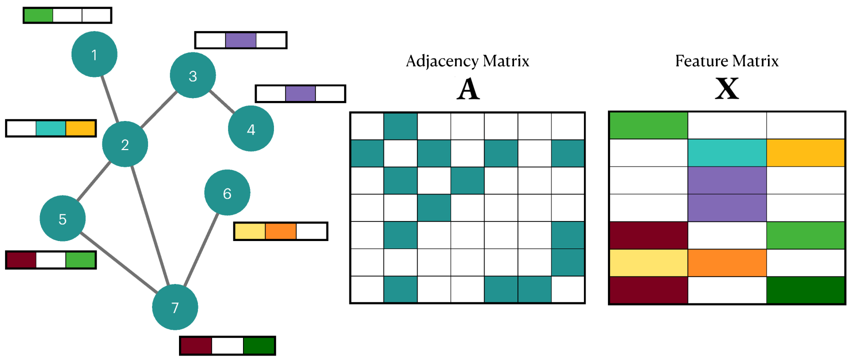


Figure 2.1: Illustration of an attributed graph and its adjacency and feature matrices. The first row of the adjacency matrix shows that node 1 is connected only to node 2, i.e., the colored entries indicate the elements of the matrix with value 1. The first row of the feature matrix contains the feature vector for node 1. In the feature matrix different colors represent different feature values.

2.1.1 The Weisfeiler-Leman Algorithm

Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are said to be *isomorphic* if there exists a bijective function $f : V_1 \rightarrow V_2$ such that $(u, v) \in E_1 \Leftrightarrow (f(u), f(v)) \in E_2 \forall u, v \in V_1$. An example of isomorphic graphs is shown in Figure 2.2. The problem of *graph isomorphism* consists of determining if two graphs are isomorphic, and there is no known solution in polynomial time or proof of NP-completeness.

The Weisfeiler-Leman algorithm [209], or 1-WL, is a heuristic for the problem of graph isomorphism. The algorithm is also known as *color refinement*, as, starting from an initial coloring of the nodes in a graph, it iteratively refines the coloring until convergence, and uses the final node coloring to compare graphs (graphs with different histograms of colors are non-isomorphic, while graphs with the same histogram of colors cannot be determined as isomorphic or not). The idea is

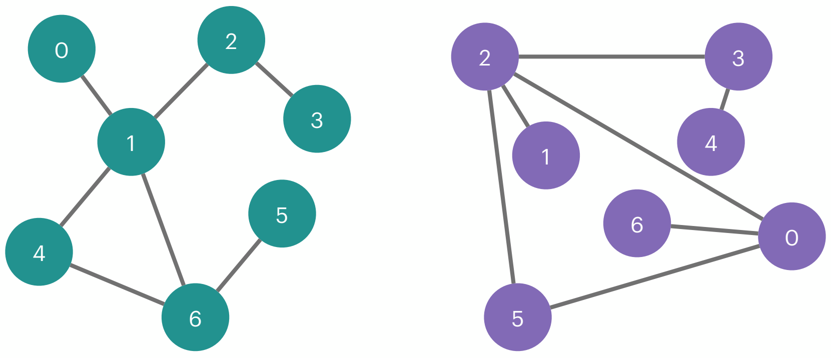


Figure 2.2: Example of two isomorphic graphs. The bijective function between the vertices of the two graphs that determines the isomorphism is $f(v) = (v+1) \bmod 7$, where v belongs to the graph on the left.

that at the i-th iteration, the color of a node is given by its color at iteration $i-1$, and the multiset of colors of its neighbours. As every node gets updated at every iteration, the color for a node at iteration i is influenced by the structure of its i-hop neighbourhood. In other words, two nodes have the same color at the end of the i-th iteration if and only if their i-hop neighbourhoods have the same structure (i.e., they are isomorphic).

More formally, following the terminology in Morris et al. [161] and Jegelka [103], a labelled graph is a graph $G = (V, E)$ coupled with a labeling function $l : V \to \Sigma$, where Σ is an *alphabet* (e.g., a sufficiently large subset of the natural numbers). Then, the 1-WL algorithm proceeds by iteratively refining the coloring $c^{(t)} : V \to \Sigma$ for the graph as follows:

$$c^{(0)}(v) = l(v) \tag{2.1}$$

$$c^{(t)}(v) = \text{HASH}\big(c^{(t-1)}(v), \{\{c^{(t-1)}(u) | u \in \mathcal{N}_v\}\}\big) \text{ for } t > 0 \tag{2.2}$$

where $\{\{\cdot\}\}$ indicates a multiset, and HASH is an ideal hashing function that assigns a different color to every neighbourhood (i.e., an injective function from the input pair to Σ). Given two graphs $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$, the algorithm compares their colorings $\{\{c^{(t)}(v) | v \in V_1\}\}$ and $\{\{c^{(t)}(v) | v \in V_2\}\}$ at every t. If the multisets differ at any iteration, then the graphs are certainly non-isomorphic, but if their coloring are the same at every iteration (the algorithms stops when the coloring remain stable from one iteration to the other, which happens in at most $\max\{|V_1|, |V_2|\}$ iterations), then it is not possible to conclude if the graphs are isomorphic or not. A graphical illustration of the steps of the algorithm is shown in Figure 2.3. An example of graphs that are not distinguishable by the 1-WL algorithm is the class of regular graphs with the same number of nodes.

Higher-order versions of the algorithm have been proposed to expand the class of distinguishable graphs [10, 39, 99], but it has been shown that the 1-WL is enough for most practical cases [11, 235].

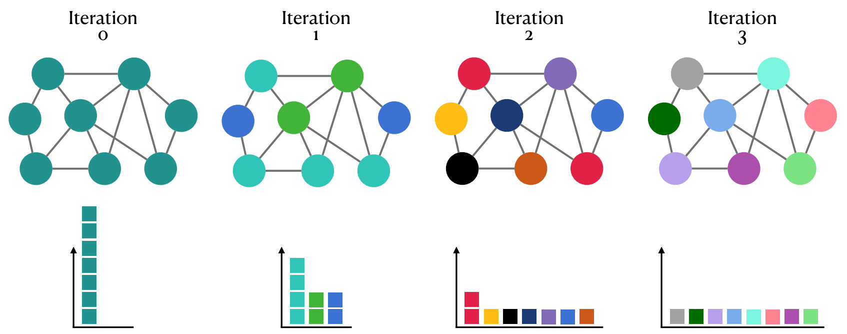


Figure 2.3: Graphical representation of the 1-WL algorithm. In this example there is a uniform initial coloring, so the first iteration distinguishes nodes with different degrees. If at iteration i two nodes have the same color, then their i-hop neighbourhoods have the same structure. After the third iteration the coloring remains the same. The histograms at every iteration can be compared to those of other graphs to determine if the graphs are not isomorphic.

2.1.2 Random Walks

A *random walk* on a graph is a stochastic process that starts at a given node, and randomly moves between adjacent nodes (i.e., nodes connected by an edge). As we consider unweighted graphs, the neighbour to move to at every iteration of the random walk is chosen uniformly.

Formally, if we have a graph $G = (V, E)$ with n nodes represented by an adjacency matrix $\mathbf{A}$, we then define the transition matrix $\mathbf{W} = \mathbf{D}^{-1}\mathbf{A}$, where $\mathbf{D}$ is the degree matrix. Intuitively, if the element in row i and column j of $\mathbf{W}$ is non-zero, then it indicates the probability that a random walker in node i moves to node j. If we let $\mathbf{p}^{(0)} \in \mathbb{R}^n, \sum_{i=1}^{n} p_i^{(0)} = 1$ be the vector where the i-th element indicates the probability of starting the random walk at node i, it is then the possible to obtain the probability that the random walk is in node u after k iterations by looking at the u-th element of $(\mathbf{W}^\top)^k\mathbf{p}^{(0)}$. In fact, the rule of the walk can be expressed as:

$$\mathbf{p}^{(i+1)} = \mathbf{W}^\top \mathbf{p}^{(i)} \quad \forall i > 0.$$

If the graph respects certain properties (it is irreducible and aperiodic), then the random walk converges to a stationary distribution $p_v^{(\infty)} = \frac{degree(v)}{2\cdot|E|}$. In other words, the probability of a random walk being at a certain node (after a sufficiently large number of steps) does not depend on the starting node, but on its degree. The stationary distribution can be obtained be computing the eigenvector of $\mathbf{W}$ with associated eigenvalue of 1.

Random walks on graphs are heavily studied [37, 74, 146, 166] and have many practical use cases [51, 70, 82, 186, 214].

Random Walks with Restart

A Random Walk with Restart (RWR) [170] is a random walk in which at every iteration there is a certain probability that the walk “jumps back” to the starting node. Formally, for a starting node v, a RWR is expressed with the following equation:

$$\mathbf{r}_{(v)}^{(i+1)} = (1-c)\mathbf{W}^{\top}\mathbf{r}_{(v)}^{(i)} + c\mathbf{e}_{(v)}$$

where $\mathbf{r}_{(v)}^{(i)}$ is a vector of size n where the j-th element indicates the probability that the RWR is in node j at iteration i, $\mathbf{e}_{(v)}$ is a vector where the v-th element is 1 and all the others are 0, c is the restart probability, and $\mathbf{W}$ is the transition matrix of the random walk. The restart probability c defines the probability that the walk “jumps” back to the starting node (a common value for c, used in many libraries, is 0.15). The *RWR vector* for node v is the steady state of the walk $\mathbf{r}_{(v)}^{(\infty)}$, and can be computed using the power iteration method. Over the years a large number of methods have been developed for its efficient and practical computation, or approximation, even for large scale graphs (e.g., see Lofgren [139] and Tong et al. [200]). Elements of $\mathbf{r}_{(v)}^{(\infty)}$ capture the relative relationships between nodes [200], and the RWR vectors capture the global structure of the graph [91, 106]. We will use the term *RWR matrix* to refer to the matrix $\mathbf{R} \in \mathbb{R}^{n\times n}$ in which the v-th row contains the RWR vector for node v, i.e., $\mathbf{R}_{v,:} = \mathbf{r}_{(v)}^{(\infty)}$.

2.1.3 Graph Coarsening

Given a graph $G = (V, E)$, a coarsened version $G^c = (V^c, E^c)$ is obtained by partitioning V and grouping together the nodes that belong to the same subset of the partition. More formally, given a partition $P = \{S_1, S_2, \dots, S_{n'}\}$ where $S_i \subseteq V$, $S_i \cap S_j = \emptyset$, $\bigcup_i S_i = V$, we associate a node s_i to each subset S_i and let

$$V^c = \{s_1, s_2, \dots, s_{n'}\}, \text{ and}$$
$$E^c = \{(s_i, s_j) \mid \text{there is a node in } S_i \text{ connected to at least one node in } S_j\}.$$

The goal of graph coarsening is to generate a coarsened version of a graph (i.e., $n' < n$) such that specific properties are preserved. For instance, Loukas and Vandergheynst [145] aim at preserving the principal eigenvalues and eigenspaces, while Jin et al. [107] aim at minimizing a distance function that measures the changes in the structure and connectivity that occur in the coarsening process.

Jin et al. [107] further prove that their algorithm bounds the spectral distance [111] between the original and a "lifted" version of the coarsened graph.

2.2 Representation learning

The way that data is presented to a machine learning algorithm is of great importance [62]. The way data is represented, or, in other words, the set of features used to describe the data, is called the *representation* of the data. Traditionally, the task of determining the representation to use for the data was done manually, taking advantage of human expertise and prior knowledge. For example, with regards to graph structured data, which is the focus of this work, traditional handcrafted features include the extraction of properties like node degrees, centrality measures [179], clustering coefficient [164], and motif counts [226].

In modern machine learning, and specially deep learning [81], this task is instead automated in the learning procedure. The term *Representation Learning* [20] refers exactly to the task of learning the best representation for the data, where "best" can have many meanings, but usually refers to the representation that can lead to the highest performance on the considered tasks (also referred to as *downstream* performance). The quality of a representation is then strictly related to the task(s) at hand. However, Bengio et al. [20] specify some general priors that should be behind a good representation, e.g.:

- Smoothness: similar representations should lead to similar downstream outputs.
- Multiple explanatory factors: if the data generating distribution is generated by different underlying factors, then the representation should allow to disentangle these factors of variation.
- Natural clustering: representations of objects from the same class should be close to each other.
- Sparsity: the representation for a given input should contain only the relevant information.

Popular traditional representation learning techniques include the Principal Component Analysis (PCA) [211], which has the goal of reducing the dimensionality of the data while preserving the informative content, and Independent Component Analysis (ICA) [50], which is a method for separating a multivariate signal into additive subcomponents. More recently, deep learning [81] has become the most popular method for representation learning, in both supervised and unsupervised settings.

2.2.1 Graph Representation Learning

In Graph Representation Learning (GRL) [86] the aim is to learn representations, which can be node-wise or graph-wise, that encode the structural and feature related information that is contained in a, possibly attributed, graph. These representations, which are also commonly called *embeddings*, can then be used to perform tasks such as graph classification (i.e., given a set of labels, assigning the correct label to each graph), node classification (i.e., given a set of labels, assigning the correct label to each node), link prediction (i.e., given an incomplete graph, identify the missing edges), etc. An example of node-level representation for a graph is shown in Figure 2.4. In a

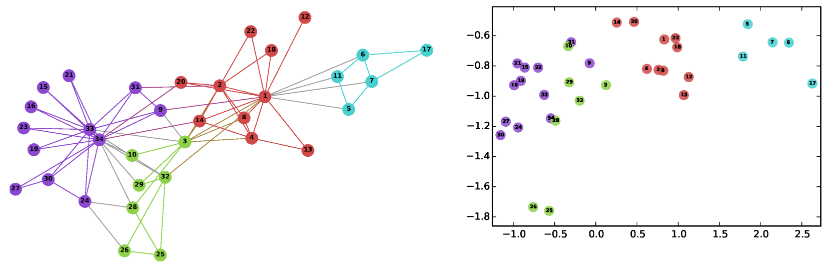


Figure 2.4: The graph on the left is Zachary's Karate network [227] and the two-dimensional embeddings on the right are obtained using DeepWalk [175]. The embeddings are such that nodes belonging to the same class (represented in the graph as nodes with the same color) are close to each other, encoding the different clusters in the graph. Figure taken from Perozzi et al. [175].

formal way, in GRL we want to design an *embedding function* f that maps an attributed graph $\hat{G}$ to either a d-dimensional vector $\boldsymbol{\zeta}$ (if we care about graph-level representations), or to a matrix $\mathbf{H} \in \mathbb{R}^{n \times d}$ where the i-th row contains the representation for the i-th node (if we want node-level representations). The dimension d of the embeddings is usually treated as a hyperparameter, but is generally much smaller than the number of nodes in the graph.

GRL presents several challenges with respect to other data modalities, such as images and text.

No fixed structure. Images and sequences have a fixed structure: every pixel is surrounded by exactly 8 pixels, and every element in a sequence has exactly one element before and one element after (the corner cases of the pixels on the border, and the elements at the beginning and end of a sequence can be easily dealt with padding [81]). This makes it easy to create local filters with parameters that can be shared across the input, which reduces the number of parameters of the model, leading to lower memory requirements and higher statistical efficiency [63]. In graphs the neighbourhoods of the nodes do not all have the same size, making it more challenging to share parameters.

No canonical ordering of the nodes. While in images there is a canonical orientation, and sequences by definition have an ordering of the elements, there is no ordering for the nodes in a graph. Consider our way of representing a graph through an adjacency matrix $\mathbf{A}$ and a feature matrix $\mathbf{X}$, as there is no canonical way of assigning an order to the nodes in the graph, multiple adjacency matrices can represent the same graph. More formally, given any permutation matrix[1] $\mathbf{P}$, the pair $(\mathbf{PAP}^\top, \mathbf{PX})$ is still representing the same graph. It is then desirable to have methods that are *equivariant* or *invariant* to the ordering of the nodes. An embedding function f is equivariant to the ordering of the nodes if $f(\mathbf{PAP}^\top, \mathbf{PX}) = \mathbf{P}f(\mathbf{A}, \mathbf{X})$ for all permutation matrices $\mathbf{P}$, while it is invariant to the ordering of the nodes if $f(\mathbf{PAP}^\top, \mathbf{PX}) = f(\mathbf{A}, \mathbf{X})$. Equivariant functions are used for node-level embeddings, while invariant functions are used for graph-level embeddings.

Traditional GRL approaches were based on dimensionality reduction techniques, in which the adjacency matrix is given as input and the target is to reduce its dimensionality while preserving some properties of the graph (e.g., pairwise distances between nodes) [17, 182, 199]. Successively, with the increasing popularity of deep learning, approaches based on neural networks have started taking the spotlight. In particular, taking inspiration from the area of natural language processing, where neural networks were used to produce word embeddings (e.g., see Mikolov et al. [155] and Pennington et al. [174]), new approaches based on random walks were proposed [82, 176]. These approaches would use random walks to extract sequences of nodes, and then treat these sequences as a corpus of phrases (where each node represents a word) and obtain embeddings using techniques from the natural language processing literature. While these approaches have been successful, they present some important limitations [86]: they do not leverage node features, which contain important amounts of information, and they are inherently *transductive*, meaning that they can only be used to generate embeddings for the nodes they are trained on. It is worth mentioning that Yang et al. [220] extended these random walk-based approaches to overcome these limitations, but the current state-of-the-art method for GRL is given by GNNs. GNNs overcome the limitations of prior methods by being able to easily leverage features (which could be attached to the nodes, edges, or could be global for the whole graph), and can be used in *inductive* settings, i.e., they can be used to generate embeddings for nodes/graphs that were not seen during training.

2.3 Graph Neural Networks

Graph Neural Networks (GNNs) are neural networks that operate on graph structured data. Since the first instances of GNNs have appeared [188, 195], the field has exploded in recent years with a plethora of different architectures (e.g., see Corso et al. [52], Kipf and Welling [121], Veličković et al. [205], and Xu et al. [216]) and applications (e.g., see Fung et al. [73], Klicpera et al. [123], Yasunaga

[1] A permutation matrix is a square matrix with exactly one entry with value 1 in each row and column, and 0 elsewhere.

et al. [222], and Ying et al. [224]). For a complete review of the field, we refer to the excellent surveys by Chami et al. [43] and Wu et al. [213], and books by Hamilton [86] and Wu et al. [212].

Most popular GNNs [43, 213] models follow the *message-passing* paradigm [80]. We will often use the terms Graph Neural Networks and Message-Passing Neural Networks (MPNNs) interchangeably. Let $\mathbf{H}^{(\ell)} \in \mathbb{R}^{n \times d'}$ be the matrix containing the node representations at layer ℓ (i.e., the representation for node i is contained in the i-th row $\mathbf{H}^{(\ell)}_{i,:}$), with $\mathbf{H}^{(0)} = \mathbf{X}$. Given an aggregation function Φ, which is permutation invariant, and a learnable update function Ψ (usually a neural network), a message passing layer updates the representation of every node v as follows:

$$\mathbf{m}^{(\ell)}_{(v)} = \Phi(\{\{\mathbf{H}^{(\ell)}_{u,:} \mid u \in \mathcal{N}_v\}\}) \tag{2.3}$$

$$\mathbf{H}^{(\ell+1)}_{v,:} = \Psi(\mathbf{H}^{(\ell)}_{v,:}, \mathbf{m}^{(\ell)}_{(v)}) \tag{2.4}$$

where $\mathbf{m}_{(v)}$ represents the aggregated message received by node v from its neighbours. Φ is called the *aggregation* function, while Ψ is usually referred to as the *update* function. After L message-passing layers, the final node embeddings $\mathbf{H}^{(L)}$ are used to perform a given task (e.g., they are the input to a network performing the task), and the whole system is trained end-to-end. When it is needed to have a graph-level representation, it is common to adopt a permutation invariant *readout* function Ω that aggregates all the node representations into a unique graph embedding: $\boldsymbol{\zeta} = \Omega(\{\{\mathbf{H}_{v,:} | v \in V\}\})$. A graphical representation of GNNs is shown in Figure 2.5.

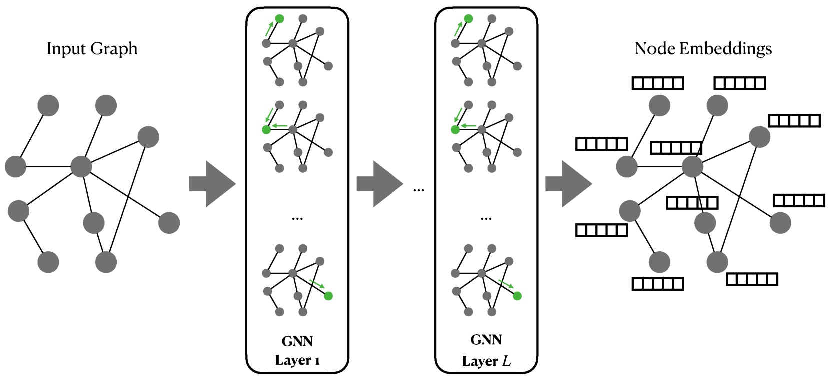


Figure 2.5: Graphical representation of a GNN. At every layer each node receives messages from its neighbours, aggregates them, and updates its representation. Figure replicated from Wu et al. [213].

Typical uses of GNNs for downstream tasks follow the same architectural pattern with an *encoder-decoder* structure [43, 87, 213]. The GNN is the encoder which produces node embeddings (low-dimensional vectors capturing relevant structural and feature-related information about each node), while the decoder uses the embeddings to carry out the desired downstream task. A scheme of the encoder-decoder architectural pattern is shown in Figure 2.6.

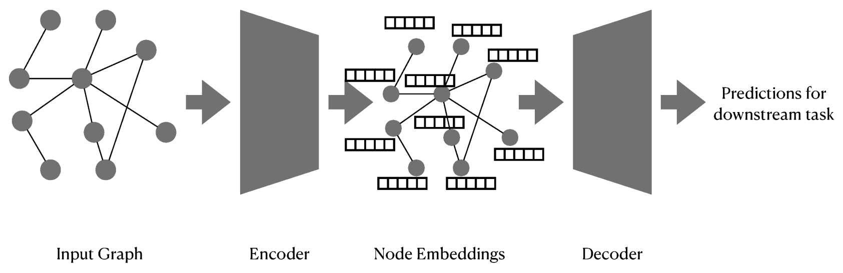


Figure 2.6: Graphical representation of the "encoder-decoder" architectural pattern. GNNs usually act as the encoder producing node embeddings, which are then used by a decoder to perform some downstream task.

2.3.1 Examples of Graph Neural Networks

While it would not be possible to provide a description of all the GNNs available in literature, we present below some of the most popular ones that can be found in later chapters of this thesis.

We use $\mathbf{W}$ to indicate the learnable *weight* matrix (i.e., the matrix with the parameters of the model), and $\mathbf{W}^{(\ell)}$ to indicate the weight matrix for layer ℓ. We use σ to indicate a generic non-liner function (typical choices in deep learning are the Rectifier Linear Unit and its variants, the Tanh, and the Sigmoid functions). We use CONCAT to refer to the function that concatenates two vectors or matrices.

Graph Convolutional Networks. The Graph Convolutional Network (GCN) [121] is one of the most popular GNN architectures. GCNs have a strong connection with spectral graph convolutions [56]. To introduce their formulation, let $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, and let $\tilde{\mathbf{D}}$ be a diagonal matrix with $\tilde{D}_{i,i} = \sum_j \tilde{A}_{i,j}$. The embedding for a node v at layer ℓ is then defined as follows:

$$\mathbf{H}^{(\ell)}_{v,:} = \sigma\left(\sum_{u \in \mathcal{N}_v} \frac{A_{v,u}}{\sqrt{\tilde{D}_{v,v}\tilde{D}_{u,u}}} \mathbf{H}^{(\ell-1)}_{u,:}\mathbf{W}^{(\ell)} + \frac{1}{\tilde{D}_{v,v}} \mathbf{H}^{(\ell-1)}_{v,:}\mathbf{W}^{(\ell)} \right) \tag{2.5}$$

In terms of the message-passing framework, we can then see that the "message" returned by the aggregation function is a weighted average of the representations of the neighbours (where the weight depends on the degree of the node), and the update function is the sum of the message and the previous representation of the node.

GraphSage. GraphSage [85] defines a general framework for applying MPNNs to large graphs. In particular the idea is to learn functions that operate on *sets* of neighbouring representations so that the model does not need access to the whole graph for computing the representation of a node. If we let $\tilde{\mathcal{N}}(v)$ be a function that returns a neighbourhood of v (and here neighbourhood can refer to any generic set of nodes "around" v), then we can express the GraphSage framework as follows:

$$\mathbf{m}^{(\ell)}_{(\tilde{\mathcal{N}}(v))} = \Phi\Big(\{\{\mathbf{H}^{(\ell-1)}_{u,:} | u \in \tilde{\mathcal{N}}(v)\}\}\Big) \tag{2.6}$$

$$\mathbf{H}^{(\ell)}_{v,:} = \sigma\Big(\mathbf{W}^{(\ell)} \cdot \text{CONCAT}(\mathbf{H}^{(\ell-1)}_{v,:}, \mathbf{m}^{(\ell)}_{(\tilde{\mathcal{N}}(v))})\Big) \tag{2.7}$$

Where Φ can be any permutation invariant function. Some examples of $\tilde{\mathcal{N}}(v)$ are: uniform samples from the k-hop neighbourhood, or frequently visited nodes from fixed length random walks starting at v.

Graph Attention Networks. The Graph Attention Network (GAT) [205] is a model designed to enable the possibility of giving different "importance" to different neighbours through an attention mechanism [12]. Given a, possibly learnable, *attention function* a, and projection weights $\mathbf{W}_{(a)}$, GAT performs the following computations:

$$e_{(v,u)} = \text{LeakyReLu}\Big(a\Big(\text{CONCAT}\Big(\mathbf{W}_{(a)}\mathbf{H}^{(\ell-1)}_{v,:}, \mathbf{W}_{(a)}\mathbf{H}^{(\ell-1)}_{u,:}\Big)\Big)\Big), \forall u \in \mathcal{N}_v \tag{2.8}$$

$$\alpha_{(v,u)} = \frac{\exp\Big(e_{(v,u)}\Big)}{\sum_{\forall k \in \mathcal{N}_v} \exp(e_{(v,k)})}, \forall u \in \mathcal{N}_v \tag{2.9}$$

$$\mathbf{H}^{(\ell)}_{v,:} = \sigma\left(\sum_{u \in \mathcal{N}_v} \alpha_{(v,u)} \mathbf{W}^{(\ell)} \mathbf{H}^{(\ell-1)}_{u,:}\right) \tag{2.10}$$

where $\text{LeakyReLu}(x) = \max(0, x) + \eta \min(0, x)$, with $\eta > 0$. The model can then be extended to use a multi-head attention mechanism, as introduced in Vaswani et al. [204]. The attention mechanism allows the model to give different weight α to different neighbours when computing the aggregation function.

Graph Isomorphism Network. The Graph Isomorphism Network (GIN) [216] was introduced to enhance the expressive power of GNNs (see Section 2.3.2). This is achieved through the use of learnable injective functions. GIN is formulated as follows:

$$\mathbf{H}^{(\ell)}_{v,:} = \text{MLP}^{(\ell)}\left((1 + \epsilon^{(\ell)})\mathbf{H}^{(\ell-1)}_{v,:} + \sum_{u \in \mathcal{N}_v} \mathbf{H}^{(\ell-1)}_{u,:}\right) \tag{2.11}$$

where $\epsilon^{(\ell)}$ can be a learnable parameter or a fixed scalar, and MLP indicates a multi-layer perceptron.

DiffPool. DiffPool [225] generates hierarchical representations of graphs, emulating the pooling layer of a Convolutional Neural Network [129, 191]. Each pooling layer produces a coarser representation of the graph that can ease the task of graph classification. DiffPool uses two GCNs at each layer. One that generates embeddings ($\text{GCN}^{(\ell)}_{embed}$), and another that generates a soft cluster assignment matrix ($\text{GCN}^{(\ell)}_{pool}$), that is then used to obtain the feature and adjacency matrices of the coarsened graph. The steps of DiffPool are provided below, where $n^{(\ell)}$ is a hyperparameter representing the number of clusters at layer l. At the last layer (L) everything is pooled into a single cluster/node to get the final graph embedding.

$$\mathbf{A}^{(0)}, \mathbf{X}^{(0)} = \mathbf{A}, \mathbf{X} \tag{2.12}$$

$$\dots$$

$$\mathbf{Z}^{(\ell)} = \text{GCN}^{(\ell)}_{embed}\left(\mathbf{X}^{(\ell-1)}, \mathbf{A}^{(\ell-1)}\right) \tag{2.13}$$

$$\mathbf{S}^{(\ell)} = \text{GCN}^{(\ell)}_{pool}\left(\mathbf{X}^{(\ell-1)}, \mathbf{A}^{(\ell-1)}\right) \tag{2.14}$$

$$\mathbf{X}^{(\ell)} = \mathbf{S}^{(\ell)\top}\mathbf{Z}^{(\ell)} \tag{2.15}$$

$$\mathbf{A}^{(\ell)} = \mathbf{S}^{(\ell)\top}\mathbf{A}^{(\ell)}\mathbf{S}^{(\ell)} \tag{2.16}$$

$$\dots$$

$$\mathbf{Z}^{(L)} = \text{GCN}^{(L)}_{embed}\left(\mathbf{X}^{(L-1)}, \mathbf{A}^{(L-1)}\right) \tag{2.17}$$

$$\mathbf{S}^{(L)} = \mathbf{1} \tag{2.18}$$

$$\zeta = \mathbf{S}^{(L)\top}\mathbf{Z}^{(L)} \tag{2.19}$$

Where we use $\mathbf{A}^{(\ell)}, \mathbf{X}^{(\ell)}$ to indicate the adjacency matrix and feature matrix for the pooled graph at layer ℓ. As it produces a final graph-level representation, GraphSage is used only for graph-level tasks.

k-GNN. The main idea behind k-GNNs [162] is to consider groups of k nodes, in order to perform massage passing between subgraph structures, rather than nodes. This should allow the network to access structural information that would not be available at node level. We present here the tractable version of the k-GNN algorithm, as opposed to the theoretically more powerful but intractable one. Let $[V]^k$ be the set of all possible k-elements subsets over V. Let $s = \{s_1, .., s_k\} \in [V]^k$, then a neighbourhood of s is defined as: $\mathcal{N}'(s) = \{t \in [V]^k || s \cap t| = k-1\}$. The *local* neighbourhood is then defined as $\mathcal{N}'_{(L)}(s) = \{t \in \mathcal{N}'_s | (v, w) \in$

E, for the unique $v \in s \;\; t$, and the unique $w \in t \;\; s\}$. The propagation function of each layer is then defined as, $\forall s \in [V]^k$,

$$f_k^{(\ell)}(s) = \begin{cases} f^{\text{iso}}(s) & \text{for } \ell = 0 \\ \sigma\Big(f_k^{(\ell-1)}(s) \cdot \mathbf{W}_{(1)}^{(\ell)} + \sum_{u \in \mathcal{N}'_{(L)}(s)} f_k^{(\ell-1)}(u) \cdot \mathbf{W}_{(2)}^{(\ell)}\Big) & \text{for } \ell = 1, \dots, L \end{cases} \tag{2.20}$$

Where $f^{\text{iso}}(s)$ is a function returning a one-hot encoding of the isomorphism type of s. The authors propose a hierarchical version that combines the information at different granularities, where the initial feature vector of s is given by a concatenation of the isomorphism type and the features learned by a $(k-1)$-GNN:

$$f_k^{(0)}(s) = \sigma\left(\text{CONCAT}\left(f^{\text{iso}}(s), \sum_{u \subset s} f_{k-1}^{(L_{k-1})}(u)\right) \cdot \mathbf{W}_{(k-1)}\right) \tag{2.21}$$

where L_{k-1} is the number of the last layer in the $(k-1)$-GNN.

2.3.2 Connection to the Weisfeiler-Leman Algorithm

There is a strong connection between the 1-WL algorithm and MPNNs, which is described in Theorem 2.1 (taken from Jegelka [103]):

Theorem 2.1 ([162, 216]). If for two graphs G_1, G_2 a message passing GNN f outputs $f(G_1) \neq f(G_2)$, then the 1-WL algorithm will determine that $G_1 \neq G_2$. A sufficient condition is that the aggregate function Φ, the update function Ψ, and readout function Ω, are injective multiset functions.

This connection, which was first noted by Kipf and Welling [121] and later expanded on by Xu et al. [216], indicates that some GNNs inherit the capabilities of the 1-WL algorithm in distinguishing non-isomorphic graphs, which also implies they inherit its shortcomings, e.g., on regular graphs [8, 39, 118]. Comparing against the 1-WL algorithm has become the "standard" way of measuring the *theoretical* expressiveness of GNNs. It is worth mentioning that while this connection gives interesting insights on the ability of different GNNs of distinguishing non-isomorphic graphs, it does not always reflect practical performance. In fact, the presence of attributes on the nodes of a graph can already make it possible to distinguish graphs that the 1-WL cannot distinguish [1]. Furthermore, while a GNN may have the theoretical ability of distinguishing all the graphs distinguished by the 1-WL, there are no guarantees that the learning process will actually reach a configuration of the parameters with this capability.

Over the past few years, many GNNs with higher distinguishing power than the 1-WL algorithm have been proposed, usually at the cost of higher computational complexity, e.g., see Bodnar et al. [24, 25], Cotta et al. [53], Maron et al. [150], Morris et al. [162], and Murphy et al. [163].

2.3.3 Oversmoothing & Oversquashing

Empirically, GNN architectures usually have a relatively small number of layers with respect to the architectures used in other domains. In fact, in many popular domains like computer vision, audio processing, and natural language processing, increasing the depth of the neural network has shown to provide great benefits (e.g., see He et al. [92], Kenton and Toutanova [116], and Oord et al. [168]). On the contrary, increasing the depth of GNNs often leads to much lower performance. There are in fact two phenomenons, known as *oversmoothing* and *oversquashing* that are to blame for this behavior.

Oversmoothing [136, 167] refers to the fact that the iterative message-passing mechanism at the base of most GNNs, in which every node aggregates the information from its neighbours, leads to all nodes having similar representations as the number of layers increases. Intuitively, if we take the example of an aggregation function that computes the average of the representation of the neighbours of a node, as the number of layers increases, the size of the neighbourhood that impacts the representation of a node becomes bigger, and aggregating many representations in a permutation invariant manner, pushes all nodes to a representation which is similar for all nodes. In fact, it has been proven that the representations converge to a value that carries limited information [167].

Oversquashing [5] instead refers to the distortion of messages coming from distant nodes. In fact, as the number of layers increases, the aggregated message needs to encode information from a usually exponentially increasing [45] number of nodes. Oversquashing causes the network to necessarily drop information with the increase in the number of layers.

Many works have been proposed to tackle these issues, e.g., see Bodnar et al. [23], Chen et al. [44], Topping et al. [201], and Zhou et al. [232], but it is still uncommon to encounter "deep" GNNs in practice, and whether deep GNNs are needed or not is still an object of debate [29].

3 The Impact of Global Structural Information in Graph Neural Networks

GNNs are usually deployed with a "small" number of layers. In fact, empirically, the best results are obtained when the message passing procedure is repeated a relatively small number of times (typical numbers are 2 to 5), as a higher number of layers leads to oversmoothing [136] and oversquashing [5] (introduced in Section 2.3.3). Consequently, *practical* GNNs are only leveraging the graph structure in the form of small neighbourhoods around each node. A direct repercussion of this limitation is that GNNs are not capable of accessing, or extracting, information that depends on the whole structure of the graph (e.g., random walk probabilities [151]).

In this chapter we are interested in studying whether global information (i.e., information that depends on the *whole* structure of the graph, and that cannot be recovered by just focusing on local neighbourhoods) is important for GNNs and their practical applications.

In fact, there is an ongoing debate in the GNN research community on whether it is needed to have "deep" GNNs [29], or, if most tasks of interest only require access to local neighbourhoods. We tackle this question directly at its root, and address the overlooked aspect of whether global structural information is useful for GNN models, by studying if global structural information is important in practical scenarios.

In more detail, we introduce three different ways to provide GNN models with global structural information, and study how they affect the performance of state-of-the-art MPNNs on common graph related tasks. The three strategies to include global structural information we consider are: (i) providing the model direct access to the adjacency matrix, (ii) providing the model direct access to RWR coefficients (introduced in Section 2.1.2), and (iii) combining (ii) with a regularization term which enforces the role of the information extracted by RWR. These methods are introduced to study the impact of global information, and are not meant to be used as practical strategies to improve the performance of GNNs. On the latter aspect, we show that the sole use of our regularization term provides significant gains in performance while being easily and efficiently applicable to any GNN model. The use of RWR is also supported by a theoretical contribution which proves they can increase the ability of GNNs in distinguishing non-isomorphic graphs.

Our Contribution. Previous studies on the capabilities and limitations of GNNs have focused on the relation between GNNs and the 1-WL algorithm [209] to study the *theoretical* expressiveness of these models (see Section 2.3.2), or on how to alleviate the oversmoothing and oversquashing issues (see Section 2.3.3). There are however no empirical studies on the practical impact of global information (i.e., information that depends on the whole structure of the graph) in MPNNs.

We assess whether providing global information regarding the whole graph structure has a significant impact on the performance of state-of-the-art MPNNs. In this regard, our contributions presented in this Chapter are threefold.

- We propose and formalize three different types of global structural information "injection". We test how the injection of global structural information impacts the performance of 6 GNN architectures (GCN [121], Graphsage [85], and GAT [205] for node-level tasks; GCN with global readout, DiffPool [225] and k-GNN [162] for graph-level tasks) on both transductive and inductive tasks. Results show that the injection of global structural information significantly impacts current state-of-the-art models on common graph-related tasks.
- As we discuss later in the paper, injecting global structural information can be impractical. We then identify a novel and practical regularization strategy, called RWRReg, based on RWR [170]. RWRReg maintains the permutation-invariance of GNN models, and leads to an average 5% increase in accuracy on both node classification, and graph classification.
- We introduce a theoretical result proving that the information extracted by RWR can "speed up" the 1-WL algorithm. In more detail we show that by constructing an initial coloring based on RWR probabilities, the 1-WL algorithm always terminates in one iteration. Given the known relationship between GNNs and the 1-WL algorithm (Section 2.3.2), this result shows that providing information obtained from RWR to GNN models can improve their *practical* ability of distinguishing non-isomorphic graphs.

3.1 Random Walks can empower Graph Neural Networks

We provide analytical evidence that RWR can significantly empower MPNNs by proving a connection with the 1-WL algorithm.

It is known that not all non-isomorphic graphs are distinguishable by the 1-WL algorithm, and that n iterations are enough to distinguish two graphs of n vertices which are distinguishable by the 1-WL algorithm. There is a tight connection between 1-WL and MPNNs (see Section 2.3.2). In particular, graphs that can be distinguished in k iterations by the 1-WL algorithm, can be distinguished by *certain* GNNs in k message passing iterations [162]. This implies that when using a GNN that can theoretically achieve the distinguishing power of the 1-WL algorithm, if such GNN is deployed with k' layers, it will not be able to distinguish graphs that are distinguishable by the 1-WL algorithm with $k'' > k'$ iterations.

Here, we prove that graphs that are distinguishable by 1-WL in k iterations have different feature representations extracted by RWR of length k, and hence if we use the RWR feature representations as initial coloring for the 1-WL algorithm, then the algorithm will always finish in one iteration. Given a graph $G = (V, E)$, we define its *k-step RWR representation* as $\{\{\mathrm{HASH}(\mathcal{R}^{(v)}), \forall v \in V\}\}$, where $\mathcal{R}^{(v)} = \{\{r^{(v)}_{u_1}, \ldots, r^{(v)}_{u_n}\}\}$, and each entry $r^{(v)}_{u_i}$ is the probability that a RWR of length k starting in v ends in $u_i \in V$.

Proposition 3.1. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two non-isomorphic graphs for which the 1-WL algorithm terminates with the correct answer after k iterations and starting from a uniform labelling (i.e., all nodes are initially assigned the same color). Then the k-step RWR representations of G_1 and G_2 are different.

Proof. Consider the 1-WL algorithm with a uniform initial labeling given by all 1's. It's easy to see that (i) after k iterations the label of a node v corresponds to the information regarding the degree distribution of the neighbourhood of distance $\leq k$ from v and (ii) in iteration $i \leq k$, the degrees of nodes at distance i from v are included in the label of v. In fact, after the first iteration, two nodes have the same color if they have the same degree, as the color of each node is given by the multiset of the colors of its neighbours (and we start with initial labeling given by all 1's). After the second color refinement iteration two nodes have the same color if they had the same color after the first iteration (i.e., have the same degree), and the multisets containing the colors (degrees) of their neighbours are the same. In general, after the k-th iteration, two nodes have the same color if they had the same color in iteration $k-1$, and the multiset containing the colors of the neighbours at distance k is the same for the two nodes. Hence, two nodes that have different colors after a certain iteration, will have different colors in all the successive iterations. Furthermore, the color after the k-th iteration depends on the color at the previous iteration (which "encodes" the distribution of degree of neighbours up to distance $k-1$ included), and the multiset of the degrees of neighbours at distance k.

Given two non-isomorphic graphs G_1 and G_2, if the 1-WL algorithm terminates with the correct answer starting from the all 1's labelling in k iterations, it means that there is no *matching* between vertices in V_1 and vertices in V_2 such that matched vertices have the same degree distribution for neighbourhoods at distance exactly k. Equivalently, any matching M that minimizes the number of matched vertices with different degree distribution has at least one such pair. Now consider one such matching M, and let $v \in V_1$ and $w \in V_2$ be vertices matched in M with different degree distributions for neighbourhoods at distance exactly k. Since v and w have different degree distributions at distance k, the number of choices for paths of length k starting from v and w must be different (since the number of choices for the k-th edge on the path is different). Therefore, there must be at least a node $u \in V_1$ and a node $z \in V_2$ that are matched by M but for which the number of paths of length k from v to u is different from the number of paths of length k from w to z. Since $r_u^{(v)}$ is proportional to the number of paths of length k from v to u, we have that $r_u^{(v)} \neq r_z^{(w)}$, that is $\mathcal{R}^{(v)} \neq \mathcal{R}^{(w)}$. Thus, the *$k$-step RWR representation* of G_1 and G_2 are different. $\square$

Since k iterations of the 1-WL algorithm are performed by MPNNs of depth k, but in practice MPNNs are limited to small depths, this result shows that RWR can empower MPNNs with relevant information that is discarded in practice.

3.1.1 Empirical Analysis

We further provide an empirical analysis of RWR and their capability of encapsulating global information. We consider the three node classification datasets (see Section 3.3), as this is the task which usually has the largest input graphs, and hence where this kind of information seems more relevant.

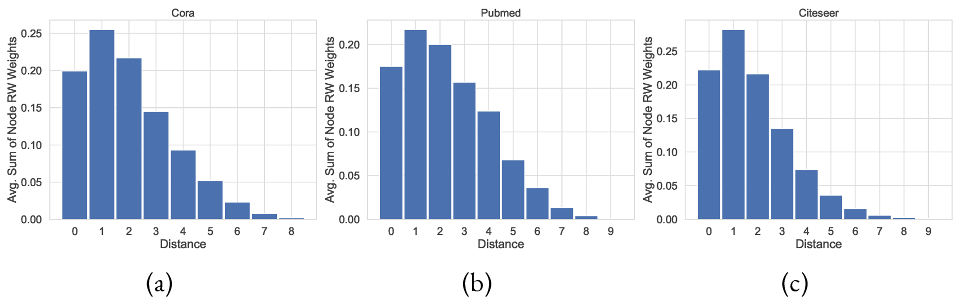


Figure 3.1: Average distribution of the RWR weights at different distances for the node classification datasets. Distance zero indicates the weight that a node assigns to itself.

We first consider the distribution of the RWR[1] weights at different distances from a given node. In particular, for each node, we take the sum of the weights assigned to the nodes at distance 1, the nodes at distance 2, and so on. We then take the average, over all nodes, of the sum of the RWR weights at each distance. We discard nodes that belong to connected components with diameter ≤ 4, and we only plot the values for the distances that have an average sum of weights higher than 0.001. Plots are shown in Figure 3.1. We notice that the RWR matrix $\mathbf{R}$ contains information that goes beyond the immediate neighbourhood of a node. In fact, we see that approximately

[1]We consider RWR, with a restart probability of 0.15, as done for the experimental evaluation of our proposed technique.

90% of the weights are contained within the 6-hop neighbourhood, with a significant portion that is not contained in the 2-hop neighbourhood usually accessed by MPNN models.

Next we analyse if RWR capture some non-trivial relationships between nodes. In particular, we investigate if there are nodes that are far from the starting node, but receive a higher weight than some closer nodes. To quantify this property we use the Kendall Tau-b[2] measure [115].

Table 3.1: Average and standard deviation, over all nodes, of Kendall Tau-b values measuring the non-trivial relationships between nodes captured by the RWR weights.

Dataset	**Average Kendall Tau-b**
Cora	0.729 ± 0.082
Pubmed	0.631 ± 0.057
Citeseer	0.722 ± 0.171

In more detail, for each node v we consider its row of the RWR matrix $\mathbf{R}_{v,:}$ where the i-th element is the weight that the RWR from node v has assigned to node i (i.e., $R_{v,i}$). We then define the vector $\mathbf{d}^{(v)} \in \mathbb{R}^n$ such that $d_j^{(v)} = \mathit{dist}(v, f_{sort\_weights}(j, \mathbf{R}_{v,:}))$, where *dist(x, y)* is the shortest path distance between node x and node y, and $f_{sort\_weights}(j, \mathbf{R}_{v,:})$ is the function returning the node with the j-th highest RWR weight in $\mathbf{R}_{v,:}$. Intuitively, if the RWR matrix isn't capable of capturing non-trivial relationship we would have that $\mathbf{d}^{(v)}$ is a sorted vector (with possible repetitions). By comparing $\mathbf{d}^{(v)}$ with its sorted version with the Kendall Tau-b rank, we obtain a value between 1 and -1 where 1 means that the two sequences are identical, and -1 means that one is the reverse of the other. Table 3.1 presents the results, averaged over all nodes, on the node classification datasets. These results show that while there is a strong relation between the information provided by RWR and the distance between nodes, there is information in the RWR that is not captured by shortest path distances.

As an example of the non-trivial relationships encoded by RWR, Figure 3.2 presents a $\mathbf{d}^{(v)}$ vector taken from a node in Cora. This vector obtains a Kendall Tau-b value of 0.591. We can observe that for distances greater than 1, we already have some non-trivial relationships. In fact, we observe some nodes at distance 3 that receive a larger weight than nodes at distance 2. There are many other interesting non-trivial relationships, for example we notice that some nodes at distance 7, and some at distance 11, obtain a higher weight than some nodes at distance 5.

3.2 Injecting Global Information in MPNNs

To test if MPNNs are missing on important information that is encoded in the structure of a graph, we inject global structural information into existing MPNN models, and test how the performance of these models changes in several graph-related tasks. Intuitively, based on a model's

[2]We use the Tau-b version because the elements in the sequences we analyze may not be all distinct.

$\mathbf{d}^{(1000)} =$

```
[1, 1, 1, 2, 2, 2, 2, 2, 3, 2, 3, 2, 2, 2, 2, 3, 2, 3, 2, 2, 3, 2, 2, 3, 2, 2, 2, 3, 2, 3, 3, 2, 2, 3, 3, 4, 3, 4,
3, 3, 3, 3, 3, 3, 4, 3, 3, 4, 3, 3, 4, 3, 3, 4, 3, 3, 3, 3, 4, 4, 3, 3, 3, 4, 3, 3, 3, 3, 4, 3, 4, 3, 3, 3, 4, 4, 3, 4, 4, 4, 4, 4,
4, 4, 3, 3, 3, 4, 4, 3, 3, 4, 3, 3, 4, 4, 3, 4, 3, 4, 3, 4, 4, 4, 4, 4, 5, 5, 3, 4, 4, 4, 4, 4, 4, 4, 4, 3, 4, 4, 5, 4, 4, 4, 6, 3,
4, 4, 5, 4, 4, 5, 4, 5, 4, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 5, 5, 4, 4, 4, 4, 5, 4, 4, 4, 4, 4, 5, 5, 5, 4, 5, 4, 5, 5, 4, 4, 4, 5, 4,
4, 4, 4, 5, 5, 4, 4, 4, 5, 4, 4, 5, 4, 4, 5, 4, 4, 5, 5, 5, 5, 5, 4, 5, 5, 4, 5, 5, 4, 5, 4, 4, 5, 4, 5, 5, 5, 5, 5, 4, 5, 4, 5, 5,
4, 4, 5, 4, 5, 5, 4, 5, 5, 5, 4, 4, 5, 5, 5, 4, 5, 4, 5, 5, 5, 5, 4, 5, 5, 5, 6, 5, 5, 5, 4, 5, 5, 5, 4, 4, 5, 5, 5, 4, 5, 5, 5, 4,
4, 5, 5, 5, 5, 4, 5, 5, 5, 4, 5, 4, 5, 4, 5, 4, 5, 4, 5, 5, 6, 4, 4, 4, 5, 5, 5, 4, 5, 4, 5, 5, 5, 6, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
5, 5, 5, 5, 5, 5, 5, 5, 6, 5, 4, 5, 5, 4, 6, 5, 4, 5, 5, 5, 4, 5, 5, 5, 6, 4, 4, 5, 5, 5, 4, 5, 5, 5, 5, 5, 5, 5, 4, 5, 6, 5, 5, 5,
6, 5, 4, 5, 5, 5, 4, 5, 4, 5, 4, 5, 4, 4, 5, 6, 6, 5, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 4, 5, 5, 6, 6, 5, 5, 4, 5, 5, 5, 5, 6, 5, 6, 5,
6, 5, 4, 5, 5, 5, 5, 5, 5, 6, 6, 5, 5, 5, 5, 6, 5, 5, 5, 5, 5, 6, 5, 5, 6, 6, 6, 6, 5, 5, 5, 5, 4, 6, 6, 5, 4, 4, 4, 4, 6, 5, 5, 5,
5, 5, 5, 5, 5, 6, 5, 5, 6, 5, 5, 5, 4, 6, 5, 5, 5, 5, 5, 6, 5, 5, 6, 6, 6, 5, 6, 6, 6, 5, 6, 5, 5, 5, 4, 5, 5, 6, 5, 5, 6, 4, 5, 5,
5, 6, 5, 6, 5, 4, 5, 6, 5, 4, 6, 5, 6, 5, 6, 4, 6, 5, 5, 5, 5, 6, 5, 6, 5, 5, 5, 6, 5, 6, 4, 6, 6, 5, 5, 6, 5, 5, 5, 5, 5, 5, 5, 5,
5, 5, 4, 6, 5, 5, 5, 6, 6, 6, 6, 6, 4, 6, 5, 6, 5, 5, 6, 5, 5, 6, 5, 6, 5, 5, 5, 5, 7, 5, 5, 5, 5, 5, 5, 4, 5, 6, 5, 5, 6, 4, 6, 5,
5, 5, 5, 5, 5, 6, 5, 6, 6, 5, 5, 6, 5, 5, 4, 6, 5, 5, 7, 5, 6, 6, 5, 5, 5, 6, 6, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 5, 6, 5, 5, 5, 5, 5,
5, 6, 5, 6, 6, 5, 6, 5, 6, 5, 6, 6, 5, 5, 5, 7, 5, 6, 5, 6, 5, 6, 6, 6, 5, 5, 5, 5, 5, 6, 4, 5, 6, 6, 6, 5, 5, 6, 5, 6, 5, 5, 5, 6,
5, 5, 6, 6, 4, 6, 5, 6, 5, 5, 6, 5, 5, 6, 5, 5, 6, 5, 6, 6, 4, 6, 6, 7, 5, 5, 5, 6, 6, 5, 6, 5, 6, 6, 5, 4, 5, 4, 6, 6, 6, 5, 5, 5,
6, 7, 6, 5, 4, 6, 5, 6, 6, 5, 4, 6, 5, 5, 5, 7, 5, 6, 5, 5, 4, 6, 5, 5, 6, 4, 5, 6, 5, 6, 5, 6, 6, 5, 5, 6, 7, 6, 6, 6, 6, 5, 6, 5,
5, 5, 5, 7, 5, 5, 5, 6, 5, 5, 5, 6, 6, 6, 5, 5, 6, 7, 5, 5, 6, 4, 6, 5, 6, 6, 4, 6, 6, 6, 6, 5, 6, 5, 6, 6, 5, 5, 6, 6, 6, 6, 5, 6,
7, 5, 5, 7, 6, 6, 6, 6, 5, 7, 6, 4, 5, 5, 6, 6, 5, 5, 5, 6, 6, 5, 5, 6, 7, 5, 5, 6, 5, 7, 5, 5, 6, 6, 7, 6, 6, 7, 6, 6, 6, 6, 6, 5,
5, 4, 5, 6, 5, 6, 7, 6, 6, 6, 6, 6, 6, 6, 4, 7, 6, 4, 4, 5, 6, 6, 6, 4, 6, 7, 6, 7, 6, 6, 5, 5, 4, 6, 6, 6, 5, 5, 6, 7, 6, 5, 4, 6,
7, 6, 4, 6, 5, 7, 6, 7, 5, 7, 6, 6, 4, 4, 5, 6, 4, 6, 6, 7, 6, 6, 6, 7, 5, 6, 6, 4, 6, 6, 6, 6, 5, 6, 5, 5, 7, 6, 7, 5, 7, 6, 6, 6,
5, 7, 6, 7, 6, 5, 6, 7, 6, 6, 5, 7, 7, 4, 6, 6, 5, 6, 4, 6, 6, 6, 5, 6, 6, 6, 6, 7, 4, 6, 5, 4, 6, 5, 6, 7, 7, 6, 6, 6, 4, 6, 6, 6,
6, 4, 4, 6, 4, 5, 4, 6, 7, 6, 6, 6, 6, 5, 6, 6, 6, 6, 5, 6, 5, 4, 4, 6, 6, 4, 5, 6, 7, 6, 6, 6, 5, 6, 4, 4, 5, 5, 4, 5, 7, 7, 4, 7,
6, 6, 7, 7, 7, 7, 5, 5, 6, 4, 5, 5, 6, 6, 6, 5, 6, 4, 7, 6, 6, 7, 6, 6, 6, 6, 4, 5, 6, 6, 7, 7, 4, 6, 6, 6, 6, 4, 4, 4, 5, 7, 5, 5,
6, 5, 5, 6, 7, 4, 6, 7, 6, 4, 6, 5, 5, 6, 5, 6, 6, 6, 6, 6, 5, 6, 6, 6, 5, 8, 6, 7, 6, 7, 6, 6, 6, 6, 6, 6, 6, 6, 6, 4, 6, 6, 6, 6,
6, 5, 6, 4, 5, 6, 6, 7, 4, 6, 6, 6, 6, 6, 8, 5, 7, 7, 4, 4, 6, 6, 6, 7, 6, 6, 6, 6, 6, 5, 6, 7, 6, 6, 4, 7, 5, 4, 6, 6, 4, 4, 4, 4,
7, 6, 6, 6, 6, 6, 6, 6, 7, 5, 7, 6, 4, 6, 4, 5, 6, 6, 7, 4, 4, 4, 6, 4, 7, 6, 7, 6, 5, 6, 7, 7, 4, 6, 7, 5, 4, 6, 6, 5, 6, 5, 6, 7,
4, 4, 4, 5, 6, 4, 7, 4, 4, 6, 6, 6, 6, 6, 7, 7, 6, 7, 5, 7, 6, 7, 6, 7, 6, 6, 4, 6, 6, 6, 6, 6, 6, 6, 8, 6, 5, 5, 7, 4, 5, 4, 6, 8,
8, 6, 7, 6, 7, 7, 4, 7, 6, 7, 7, 5, 7, 4, 6, 4, 6, 6, 6, 7, 7, 4, 7, 7, 7, 6, 6, 6, 6, 7, 6, 6, 6, 4, 6, 4, 7, 4, 6, 7, 6, 6, 5, 6,
4, 6, 4, 4, 4, 6, 7, 5, 4, 5, 6, 5, 4, 7, 4, 5, 7, 6, 6, 7, 6, 7, 6, 4, 4, 4, 7, 4, 4, 5, 7, 8, 4, 6, 5, 5, 6, 6, 4, 6, 6, 7, 6, 6,
7, 6, 7, 4, 7, 7, 5, 6, 6, 5, 4, 7, 6, 7, 7, 7, 6, 8, 7, 7, 6, 6, 5, 7, 6, 7, 7, 7, 8, 4, 7, 4, 7, 5, 7, 4, 6, 6, 5, 4, 4, 7, 4, 6,
4, 4, 4, 7, 7, 5, 5, 8, 6, 6, 5, 5, 6, 6, 6, 5, 6, 6, 7, 6, 6, 6, 5, 7, 7, 6, 5, 6, 7, 7, 4, 7, 6, 5, 4, 5, 6, 4, 6, 7, 8, 7, 7, 6,
6, 7, 6, 7, 7, 7, 5, 7, 7, 5, 6, 6, 6, 8, 6, 7, 6, 7, 8, 6, 6, 6, 5, 7, 6, 6, 7, 5, 6, 5, 6, 4, 5, 7, 4, 7, 6, 7, 6, 4, 7, 5, 7, 4,
7, 7, 7, 4, 7, 8, 6, 6, 5, 5, 6, 4, 7, 5, 6, 7, 6, 6, 6, 4, 7, 6, 6, 7, 7, 5, 7, 5, 7, 7, 6, 8, 6, 7, 6, 7, 5, 4, 6, 6, 6, 6, 7, 4,
6, 7, 6, 4, 9, 7, 6, 5, 6, 6, 4, 5, 6, 7, 5, 6, 6, 5, 6, 7, 6, 6, 8, 4, 8, 6, 9, 6, 6, 5, 6, 5, 6, 6, 6, 6, 6, 6, 7, 5, 6, 6, 7, 7,
5, 7, 8, 5, 6, 6, 7, 6, 4, 6, 6, 7, 7, 7, 7, 5, 7, 5, 4, 7, 5, 7, 7, 6, 6, 6, 7, 8, 7, 4, 10, 5, 7, 7, 6, 6, 8, 6, 6, 6, 7, 4, 7, 8,
5, 7, 7, 7, 7, 5, 5, 7, 5, 6, 6, 6, 5, 6, 5, 5, 5, 7, 5, 4, 5, 5, 6, 4, 6, 5, 6, 8, 4, 4, 6, 5, 5, 8, 7, 5, 7, 7, 7, 5, 8, 7, 6, 8,
5, 8, 7, 6, 7, 6, 7, 6, 8, 6, 8, 7, 7, 5, 6, 6, 6, 5, 7, 5, 5, 6, 5, 6, 7, 7, 7, 5, 7, 6, 7, 5, 6, 5, 6, 8, 7, 7, 6, 7, 5, 6, 6, 7,
6, 6, 7, 7, 7, 8, 5, 7, 8, 7, 5, 7, 6, 6, 7, 5, 5, 7, 7, 6, 6, 7, 5, 8, 6, 7, 8, 6, 8, 9, 7, 7, 5, 8, 8, 7, 7, 5, 5, 7, 5, 5, 9, 7,
6, 6, 7, 7, 6, 5, 8, 5, 10, 10, 7, 6, 8, 5, 6, 7, 6, 8, 5, 7, 6, 5, 7, 5, 5, 7, 7, 6, 5, 8, 6, 7, 5, 8, 8, 5, 5, 6, 7, 6, 6, 7, 5,
6, 8, 6, 7, 7, 5, 8, 9, 6, 7, 5, 7, 8, 6, 7, 5, 5, 7, 7, 5, 7, 7, 5, 7, 4, 6, 5, 7, 7, 7, 6, 7, 6, 7, 7, 7, 6, 8, 6, 5, 5, 6, 6, 6,
6, 8, 7, 7, 8, 5, 6, 8, 8, 8, 9, 5, 8, 8, 7, 8, 7, 5, 5, 6, 5, 7, 6, 8, 7, 8, 7, 9, 5, 7, 7, 5, 8, 5, 5, 7, 6, 6, 5, 9, 6, 7, 6, 6,
5, 7, 7, 6, 7, 8, 5, 7, 8, 7, 5, 7, 7, 8, 9, 7, 7, 5, 8, 7, 7, 8, 8, 5, 9, 5, 6, 6, 8, 7, 6, 10, 5, 8, 6, 8, 7, 7, 7, 6, 7, 6, 6, 5,
7, 6, 6, 5, 5, 7, 8, 8, 7, 5, 7, 7, 8, 6, 8, 8, 6, 7, 6, 6, 6, 6, 6, 8, 8, 8, 8, 6, 5, 11, 8, 7, 8, 8, 7, 8, 9, 7, 6, 6, 8, 8, 8, 9,
6, 7, 6, 5, 6, 5, 6, 8, 8, 6, 7, 7, 8, 8, 8, 7, 8, 8, 9, 6, 9, 8, 6, 7, 8, 7, 7, 5, 8, 7, 7, 7, 7, 10, 8, 7, 7, 9, 7, 8, 8, 8, 9, 8,
5, 7, 7, 7, 8, 8, 5, 7, 6, 7, 7, 7, 8, 6, 7, 7, 7, 5, 8, 7, 10, 8, 8, 8, 8, 8, 8, 8, 5, 7, 5, 11, 9, 6, 5, 6, 7, 8, 8, 8, 8, 7, 9,
8, 5, 8, 7, 7, 8, 8, 7, 6, 7, 8, 6, 6, 7, 7, 8, 6, 6, 5, 10, 6, 10, 8, 5, 9, 8, 5, 8, 6, 8, 8, 8, 6, 6, 8, 8, 9, 9, 7, 6, 7, 8, 8,
8, 9, 7, 10, 8, 5, 6, 8, 6, 8, 9, 9, 6, 6, 6, 7, 7, 5, 6, 5, 10, 8, 5, 9, 7, 9, 8, 9, 8, 8, 8, 7, 10, 10, 5, 6, 6, 8, 8, 8, 8, 5, 6,
8, 8, 8, 9, 5, 8, 8, 9, 8, 6, 8, 7, 11, 6, 8, 9, 11, 6, 8, 5, 8, 6, 12, 8, 5, 8, 7, 7, 6, 8, 9, 9, 6, 8, 6, 8, 9, 8, 7, 9, 8, 9, 9,
7, 11, 8, 9, 10, 10, 8, 10, 9, 8, 8, 9, 7, 5, 10, 9, 9, 8, 7, 8, 5, 6, 7, 8, 5, 6, 6, 10, 8, 8, 6, 9, 7, 11, 5, 8, 8, 7, 10, 8, 8,
8, 5, 9, 8, 6, 8, 8, 9, 8, 8, 8, 9, 8, 11, 8, 8, 11, 8, 6, 9, 9, 6, 8, 5, 8, 8, 8, 6, 8, 6, 7, 11, 6, 7, 7, 9, 6, 8, 8, 9, 6, 6, 9,
9, 11, 10, 9, 8, 9, 9, 10, 10, 10, 7, 9, 8, 8, 6, 7, 8, 9, 7, 6, 6, 8, 10, 9, 10, 6, 8, 9, 8, 10, 11, 9, 10, 10, 11, 6, 11, 11, 8,
9, 7, 7, 8, 8, 10, 8, 9, 10, 13, 9, 8, 9, 7, 9, 8, 11, 7, 9, 10, 9, 9, 12, 8, 8, 9, 9, 11, 9, 11, 9, 12, 10, 11, 11]
```

Figure 3.2: $\mathbf{d}^{(v)}$ vector for the 1000-th node in Cora.

performance when injected with different types of global structural information, we can understand if this additional knowledge can improve performance on the considered tasks. In the rest of this section we present the types of global structural information injection that we consider, and the models chosen for our experimental evaluation.

3.2.1 Types of Global Structural Information Injection

We consider three different types of global structural information injection, described below. The injection strategies presented in this section are not designed for *practical* use, as the scope of these strategies is to help us understand the importance of global structural information. At this point, our objective is to study the impact of global structural information that is not accessible to GNN models. We discuss scalability and practical aspects in Section 3.4.

Adjacency Matrix. We provide GNNs with direct access to the adjacency matrix by concatenating each node's adjacency matrix row to its feature vector. This explicitly empowers the GNN model with the connectivity of each node, and allows for higher level structural reasoning when considering a neighbourhood (the model will have access to the connectivity of the whole neighbourhood when aggregating messages from neighbouring nodes). While it might seem centered around local information, the row of the adjacency matrix for a specific node pinpoints the position of the node in the graph (i.e., it can be seen as a kind of positional encoding), which acts as a unique signature for the node, carrying some information that depends on the whole graph. Furthermore, during the message passing procedure, when a node aggregates information from its neighbours, it allows the network to get a more precise positioning of the node in the graph.

Random Walk with Restart (RWR) Matrix. We perform RWR [170] from each node v, thus obtaining a n-dimensional vector that gives a score of how much v is "related" to every other node in the graph. For every node, we concatenate its vector of RWR coefficients to its feature vector. The choice of RWR is motivated by their capability to capture the relevance between two nodes [200] and the global structure of a graph [91, 106], and by the possibility to modulate the exploration of long-range dependencies by changing the restart probability. Intuitively, if a RWR starting at node v is very likely to visit a node u (e.g., there are multiple paths that connect the two), then there will be a high score in the RWR vector for v at position u. This gives the GNN model higher level information about the global structure of the graph, and, again, it allows for high level reasoning on neighbourhood connectivity.

RWR Matrix + RWR Regularization. Together with the addition of the RWR score vector to the feature vector of each node, we also introduce a regularization term based on RWR that pushes nodes with mutually high RWR scores to have embeddings that are close to each other (independently of how far they are in the graph). We define the RWRReg ($\underline{\text{R}}$andom $\underline{\text{W}}$alk with $\underline{\text{R}}$estart $\underline{\text{Reg}}$ularization) loss as follows:

$$\mathcal{L}_{RWRReg} = \sum_{i,j \in V} R_{i,j} ||\mathbf{H}_{i,:} - \mathbf{H}_{j,:}||^2$$

where $\mathbf{H}$ is a matrix of size $n \times d$ containing d-dimensional node embeddings that are in between message-passing layers (see Section 3.2.3 for the exact point in which $\mathbf{H}$ is considered for each model). With this approach, the loss function used to train the model becomes: $\mathcal{L} = \mathcal{L}_{original} + \lambda \mathcal{L}_{RWRReg}$, where $\mathcal{L}_{original}$ is the original loss function for each model, and λ is a balancing term. In Section 3.4 we show how to compute the RWRReg term efficiently using GPUs. We expect this type of information injection to have the highest impact on performance of the models on downstream tasks.

3.2.2 Choice of Models

In order to test the effect of the different types of global structural information injection and to obtain results that are indicative of the whole class of MPNNs models, we conceptually identify four different categories of MPNNs from which we select representative models.

Simple Aggregation Models. Such models utilize a "simple" aggregation strategy, where each node receives messages (e.g., feature vectors) from its neighbours, aggregates them by assigning the same "importance" to each neighbour (e.g., by averaging their messages), and uses the aggregated messages to update its embedding vector. As a representative we choose GCN [121], one of the fundamental and widely used GNNs models. We also consider GraphSage [85], as it represents a different computation strategy where a set of neighbourhood aggregation functions are learned, and a sampling approach is used for defining fixed size neighbourhoods.

Attention Models. Several models have used an attention mechanism in a GNN scenario [132, 133, 205, 230]. These methods differ from the previous category as they use an attention mechanism to assign a different "weight", or "importance", to each neighbour. As a representative we focus on GAT [205], the first to present an attention mechanism over nodes for the aggregation phase, and one of the best performing models on several datasets.

Pooling Techniques. Pooling on graphs is a very challenging task, since it has to take into account the underlying graph structure. At a high level, pooling methods provide a coarsened version of the input graph by combining groups of nodes into clusters. Among the methods that have been proposed for differentiable pooling on graphs [40, 59, 76, 134, 225], we choose DiffPool [225] for its strong empirical results. Furthermore, it can learn to dynamically adjust the number of clusters (the number is a hyperparameter, but the network can learn to use fewer clusters if necessary).

Beyond WL. Morris et al. [162] prove that message-passing GNNs cannot be more powerful than the 1-WL algorithm, and propose k-GNNs, which rely on a *subgraph message-passing* mechanism and are proven to be as powerful as the k-WL algorithm. Another approach that goes beyond the WL algorithm was proposed by Murphy et al. [163]. Both models are computationally intractable in their initial theoretical formulation, so approximations are needed. As representative we choose k-GNNs, to test if subgraph message-passing is affected by additional global structural information.

3.2.3 Training & Implementation Details

All models are trained with early stopping on the validation set (stopping the training if the validation loss doesn't decrease for a certain amount of epochs), and unless explicitly specified, we use Cross Entropy as loss function for all the classification tasks.

For the task of graph classification we zero-pad the feature vectors of each node to make them all the same length when we inject structural information into the node feature vectors.

For the task of triangle counting we follow Knyazev et al. [126] and use the one-hot representation of node degrees as node feature vectors to impose some structural information in the network.

Computing Infrastructure. The experiments were run on a GPU cluster using 1 Nvidia 1080Ti, and on a CPU cluster (when the memory consumption was too big to fit in the GPUs) equipped with 8 CPUs 12-Core Intel Xeon Gold 5118 @2.30GHz, with 1.5Tb of RAM.

Implementation details

In the rest of this Section we go through each model used in our experiments, specifying architecture, hyperparameters, and the position of the node embeddings used for RWRReg.

GCN *(node classification).* We use a two layer architecture. The first layer outputs a 16-dimensional embedding vector for each node, and passes it through a ReLu activation, before applying dropout [196], with probability 0.5. The second layer outputs a c-dimensional embedding vector for each node, where c is the number of output classes and these vectors are passed through *Softmax* to get the output probabilities for each class. An additional L2-loss is added with a balancing term of 0.0005. The model is trained using the Adam optimizer [120] with a learning rate of 0.01.

We apply the RWRReg on the 16-dimensional node embeddings after the first layer.

GCN *(graph classification).* We first have two GCN layers, each one generating a 128-dimensional embedding vector for each node. Then we apply *max*-pooling on the features of the nodes and pass the pooled 128-dimensional vector to a two-layer feed-forward neural network with 256 neurons at the first layer and c at the last one, where c is the number of output classes. A ReLu activation is applied in between the two feed-forward layers, and *Softmax* is applied after the last layer. Dropout [196] is applied in between the last GCN layer and the feed-forward layer, and in between the feedforward layers (after ReLu), in both cases with probability of 0.1. The model is trained using the Adam optimizer [120] with a learning rate of 0.0005.

We apply the RWRReg on the 128-dimensional node embeddings after the last GCN layer.

GCN *(counting triangles)*. We first have three GCN layers, each one generating a 64-dimensional embedding vector for each node. Then we apply *max*-pooling on the features of the nodes and pass the pooled 64-dimensional vector to a one-layer feed-forward neural network with one neuron. Dropout [196] is applied in between the last GCN layer and the feed-forward layer with probability of 0.1. The model is trained by minimizing the mean squared error (MSE) and is optimized using the Adam optimizer [120] with a learning rate of 0.005.

We apply the RWRReg on the 64-dimensional node embeddings after the last GCN layer.

GraphSage. We use a two layer architecture. For Cora we sample 5 nodes per-neighbourhood at the first layer and 5 at the second, while on the other datasets we sample 10 nodes per-neighbourhood at the first layer and 25 at the second. Both layers are composed of *mean-aggregators* (i.e., we take the mean of the feature vectors of the nodes in the sampled neighbourhood) that output a 128-dimensional embedding vector per node. After the second layer these embeddings are multiplied by a learnable matrix with size $128 \times c$, where c is the number of output classes, giving thus a c-dimensional vector per-node. These vectors are passed through *Softmax* to get the output probabilities for each class. The model is optimized using Stochastic Gradient Descent with a learning rate of 0.7.

We apply the RWRReg on the 128-dimensional node embeddings after the second aggregation layer.

GAT. We use a two layer architecture. The first layer uses an 8-headed attention mechanism that outputs an 8-dimensional embedding vector per-node. LeakyReLu is set with slope $\alpha = 0.2$. Dropout [196] (with probability of 0.6) is applied after both layers. The second layer outputs a c-dimensional vector for each node, where c is the number of classes, and before passing each vector through *Softmax* to obtain the output predictions, the vectors are passed through an Elu activation [49]. An additional L2-loss is added with a balancing term of 0.0005. The model is optimized using Adam [120] with a learning rate of 0.005.

We apply the RWRReg on the 8-dimensional node embeddings after the first attention layer. A particular note needs to be made for the training of GATs: we found that naively implementing the RWRReg term on the node embeddings in between two layers brings to an exploding loss as the RWRReg term grows exponentially at each epoch. We believe this happens because the attention mechanism in GATs allows the network to infer that certain close nodes, even 1-hop neighbours, might not be important to a specific node and so they shouldn't be embedded close to each other. This clearly goes in contrast with the RWRReg loss, since 1-hop neighbours always have a high score. We solved this issue by using the attention weights to scale the RWR coefficients at each epoch (we make sure that gradients are not calculated for this operation as we only use them for scaling). This way the RWRReg penalties are in accordance with the attention mechanism, and are still encoding long-range dependencies.

DiffPool. We use a 1-pooling architecture. The initial node feature matrix is passed through two (one to obtain the assignment matrix and one for node embeddings) 3-layer GCN, where each layer outputs a 20-dimensional vector per-node. Pooling is then applied, where the number of clusters is set as 10% of the number of nodes in the graph, and then another 3-layer GCN is applied to the pooled node features. Batch normalization [100] is added in between every GCN layer. The final graph embedding is passed through a 2-layer MLP with a final *Softmax* activation. An additional L2-loss is added with a balancing term of 10^{-7}, together with two pooling-specific losses. The first enforces the intuition that nodes that are close to each other should be pooled together and is defined as: $\mathcal{L}_{LP} = \|\mathbf{A}^{(l)}, \mathbf{S}^{(l)^{\top}}\mathbf{S}^{(l)}\|_F$, where $\|\cdot\|_F$ is the Frobenius norm, and $\mathbf{S}^{(l)}$ is the assignment matrix at layer l. The second one encourages the cluster assignment to be close to a one-hot vector, and is defined as: $\mathcal{L}_E = \frac{1}{n}\sum_{i=1}^{n} H(\mathbf{S}_{i,:})$, where H is the entropy function. However, in the implementation available online, the authors do not make use of these additional losses. We follow the latter implementation. The model is optimized using Adam [120] with a learning rate of 0.001.

We apply the RWRReg on the 20-dimensional node embeddings after the first 3-layer GCN (before pooling). We tried applying it also after pooling on the coarsened graph, but the fact that this graph could change during training yields to poor results.

k-GNN. We use the hierarchical 1-2-3-GNN architecture (which is the one showing the highest empirical results). First a 1-GNN is applied to obtain node embeddings, then these embeddings are used as initial values for the 2 GNN (1-2-GNN). The embeddings of the 2-GNN are then used as initial values for the 3-GNN (1-2-3-GNN). The 1-GNN applies 3 graph convolutions, while 2-GNN and the 3-GNN apply 2 graph convolutions. Each convolution outputs a 64-dimensional vector and is followed by an Elu activation [49]. For each k, node features are then globally averaged and the final vectors are concatenated and passed through a three layer MLP. The first layer outputs a 64-dimensional vector, while the second outputs a 32-dimensional vector, and the third outputs a c-dimensional vector, where c is the number of output classes. To obtain the final output probabilities for each class, *log(Softmax)* is applied, and the negative log likelihood is used as loss function. After the first and the second MLP layers an Elu activation [49] is applied, furthermore, after the first MLP layer dropout [196] is applied with probability 0.5. The model is optimized using Adam [120] with a learning rate of 0.01, and a decaying learning rate schedule based on validation results (with minimum value of 10^{-5}).

Table 3.2: (Left) Node classification dataset statistics. (Right) Graph classification dataset statistics.

Dataset	Nodes	Edges	Classes	Features	Label Rate
Cora	2708	5429	7	1433	0.052
Pubmed	19717	44338	3	500	0.003
Citeseer	3327	4732	6	3703	0.036

Dataset	Graphs	Classes	Avg. # Nodes	Avg. # Edges
ENZYMES	600	6	32.63	62.14
D&D	1178	2	284.32	715.66
PROTEINS	1113	2	39.1	72.82
TRIANGLES	45000	10	20.85	32.74

We apply the RWRReg on the 64-dimensional node embeddings after the 1-GNN. We were not able to apply it also after the 2-GNN and the 3-GNN, as it would cause out-of-memory issues with our computing resources.

3.3 Evaluation of the Injection of Global Structural Information

We now present our framework for evaluating the effects of the injection of global structural information into GNNs, and the results of our experiments. We consider one *transductive* task (node classification) and two *inductive* tasks (graph classification, and triangle counting). We use each architecture for the task that better suits its design: GCN, GraphSage, and GAT for node classification, and DiffPool and k-GNN for graph classification. We also add an adapted version of GCN for graph classification, where we first obtain node-level embeddings, and then apply a readout function to combine node embeddings into a global graph embedding vector.

With regards to datasets, for node classification we considered the three most used benchmarking datasets in literature (at the time of writing the relative paper): Cora, Citeseer, and Pubmed [190]. Analogously, for graph classification we chose three frequently used datasets: ENZYMES, PROTEINS, and D&D [117]. Table 3.2 summarizes the datasets for node classification, graph classification, and triangle counting. The node classification datasets are available at `https://linqs.soe.ucsc.edu/data`, while the graph classification and the triangle counting at `https://chrsmrrs.github.io/datasets/`.

For all the considered models we take the hyperparameters from the implementations released by the authors. The only parameter tuned using the validation set is the balancing term λ when RWRReg is applied. We found that the RWRReg loss tends to be larger than the Cross Entropy loss for prediction, and the best values for λ lie in the range $[10^{-9}, 10^{-6}]$. For all the RWR-based techniques we used a restart probability of 0.15[3]. (The effects of different restart probabilities are explored in Section 3.4.)

[3]We use 0.15 as it is a common default value used in many papers and software libraries.

Table 3.3: Node classification accuracy results of different models with added Adjacency matrix features (AD), RWR features (RWR), and RWR features + RWR Regularization (RWR+RWRReg).

Model	**Structural Information**	**Dataset** Cora	Pubmed	Citeseer
GCN	none	0.799 ± 0.029	0.776 ± 0.022	0.663 ± 0.095
	AD	0.806 ± 0.035	0.779 ± 0.070	0.653 ± 0.104
	RWR	0.817 ± 0.025	0.782 ± 0.042	0.665 ± 0.098
	RWR+RWRReg	$\mathbf{0.842 \pm 0.026}$	$\mathbf{0.811 \pm 0.037}$	$\mathbf{0.690 \pm 0.102}$
GraphSage	none	0.806 ± 0.017	0.807 ± 0.016	0.681 ± 0.021
	AD	0.803 ± 0.014	0.803 ± 0.013	0.688 ± 0.020
	RWR	0.816 ± 0.014	0.807 ± 0.015	0.693 ± 0.019
	RWR+RWRReg	$\mathbf{0.837 \pm 0.015}$	$\mathbf{0.820 \pm 0.010}$	$\mathbf{0.728 \pm 0.020}$
GAT	none	0.815 ± 0.021	0.804 ± 0.011	0.664 ± 0.008
	AD	0.823 ± 0.019	0.796 ± 0.014	0.672 ± 0.017
	RWR	0.833 ± 0.020	0.811 ± 0.009	0.686 ± 0.009
	RWR+RWRReg	$\mathbf{0.848 \pm 0.019}$	$\mathbf{0.828 \pm 0.010}$	$\mathbf{0.701 \pm 0.011}$

Node Classification. For each dataset we follow the approach that has been widely adopted in literature: we take 20 labeled nodes per class as training set, 500 nodes as validation set, and 1000 nodes for testing. Most authors have used the train/validation/test split defined by Yang et al. [220]. Since we want to test the general effect of the injection of global structural information, we differ from this approach and we do not rely on a single split. We perform 100 runs, where at each run we randomly sample 20 nodes per class for training, 500 random nodes for validation, and 1000 random nodes for testing. We then report mean and standard deviation for the accuracy on the test set over these 100 runs.

Results are summarized in Table 3.3, where we observe that the simple addition of RWR features to the feature vector of each node is sufficient to give a performance gain (up to 2%). The RWRReg term then significantly increments the gain (up to **7.5%**). These results show that, perhaps surprisingly, even for the task of node classification global structural information is important.

Graph Classification. Following the approach from Ying et al. [225] and Morris et al. [162] we use 10-fold cross validation, and report mean and standard deviation of the accuracy on graph classification. Results are summarized in Table 3.4. The performance gains given by the injection of global structural information are even more apparent than for the node classification task.

Table 3.4: Graph classification accuracy results of different models with added Adjacency matrix features (AD), RWR features (RWR), and RWR features + RWR Regularization (RWR+RWRReg).

Model	**Structural Information**	**Dataset** ENZYMES	D&D	PROTEINS
GCN	none	0.570 ± 0.052	0.755 ± 0.028	0.740 ± 0.035
	AD	0.591 ± 0.076	0.779 ± 0.022	0.775 ± 0.042
	RWR	0.584 ± 0.055	0.775 ± 0.023	0.784 ± 0.034
	RWR+RWRReg	$\mathbf{0.616} \pm \mathbf{0.065}$	$\mathbf{0.790} \pm \mathbf{0.023}$	$\mathbf{0.795} \pm \mathbf{0.032}$
DiffPool	none	0.661 ± 0.031	0.793 ± 0.022	0.813 ± 0.017
	AD	0.711 ± 0.027	0.837 ± 0.020	0.821 ± 0.039
	RWR	0.687 ± 0.025	0.824 ± 0.028	0.783 ± 0.043
	RWR+RWRReg	$\mathbf{0.721} \pm \mathbf{0.039}$	$\mathbf{0.840} \pm \mathbf{0.024}$	$\mathbf{0.834} \pm \mathbf{0.038}$
k-GNN	none	0.515 ± 0.111	0.756 ± 0.021	0.763 ± 0.043
	AD	0.572 ± 0.063	0.778 ± 0.020	0.751 ± 0.034
	RWR	$\mathbf{0.573} \pm \mathbf{0.077}$	$\mathbf{0.794} \pm \mathbf{0.022}$	0.781 ± 0.028
	RWR+RWRReg	0.571 ± 0.080	0.786 ± 0.021	$\mathbf{0.785} \pm \mathbf{0.026}$

Intuitively, this is explained by the fact that the global structure of the nodes in a graph is important for distinguishing different graphs. Most notably, the addition of the adjacency features is sufficient to give a large performance boost (up to **11%**).

Surprisingly, models like DiffPool and k-GNN show an important difference in accuracy (up to **10%**) when there is injection of structural information, meaning that even the most advanced methods suffer from the inability to exploit global structural information.

Counting Triangles. The TRIANGLES dataset [126] is composed of randomly generated graphs, where the task is to count the number of triangles contained in each graph. This is a hard task for GNNs and, as in Knyazev et al. [126], we use node degrees as node features to impose some structural information in the network. The TRIANGLES dataset has a test set with 10'000 graphs, of which half are similar in size to the ones in the training and validation sets

Table 3.5: Mean Squared Error of GCN with different types of global structural information injection on the TRIANGLES dataset.

Model	**TRIANGLES Test Set** Global	Small	Large
GCN	2.290	1.311	3.608
GCN-AD	4.746	1.162	5.971
GCN-RWR	2.044	**1.101**	2.988
GCN-RWR+RWRReg	**2.029**	1.166	**2.893**

(4-25 nodes), and half are bigger (up to 100 nodes). This permits an evaluation of a model's capabilities generalization to graphs of unseen sizes.

For this regression task we use a three layer GCN, and we minimize the Mean Squared Error (MSE) loss. Table 3.5 presents MSE results on the test dataset as a whole and on the two splits separately. We see that the addition of RWR features and of RWRReg provides significant benefits (up to **19%** improvements), specially when the model has to generalize to graphs of unseen sizes, while the addition of adjacency features leads to overfitting. In fact, in Figure 3.3 we plot the evolution of the MSE on the training and test set over the training epochs. The GCN model injected with information from the adjacency matrix (indicated with GCN-AD) reaches the lowest error on the training set, while the highest on the test set, thus confirming its overfitting behaviour. We can observe that after 6 epochs, GCN-AD is already the model presenting the lowest training loss, and it remains so until the end. Furthermore we can notice how the test loss presents a growing trend, which is in contrast to the other models.

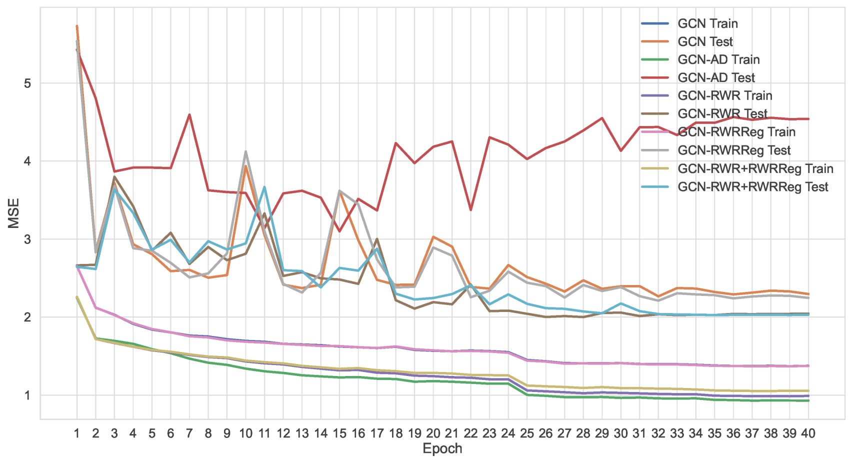


Figure 3.3: Training and test losses of GCN with different structural information injection on the triangle counting task.

3.4 Practical Aspects

From the results shown in Section 3.3, it would be tempting to propose the addition of adjacency matrix information or RWR information into node feature vectors as a strategy to improve the performance of GNN models. However, the benefits introduced by such a strategy come at a high cost: adding n features to each node increases the input size of $n \times n$ elements (which is prohibitive for large graphs). Furthermore, all the considered models have a weight matrix at each layer that depends on the feature dimension, which means we are also increasing the number of

Table 3.6: Results for the addition of *only* the RWRReg term to existing models on node classification (accuracy), graph classification (accuracy), and triangle counting (MSE– lower is better).

Model	**Regularization**	**Dataset**		
		Node Classification		
		Cora	Pubmed	Citeseer
GCN	none	0.799 ± 0.029	0.776 ± 0.022	0.663 ± 0.095
	RWRReg	$\mathbf{0.861} \pm \mathbf{0.025}$	$\mathbf{0.799} \pm \mathbf{0.034}$	$\mathbf{0.686} \pm \mathbf{0.096}$
GraphSage	none	0.806 ± 0.017	0.807 ± 0.016	0.681 ± 0.021
	RWRReg	$\mathbf{0.841} \pm \mathbf{0.016}$	$\mathbf{0.818} \pm \mathbf{0.017}$	$\mathbf{0.721} \pm \mathbf{0.021}$
GAT	none	0.815 ± 0.021	0.804 ± 0.011	0.664 ± 0.008
	RWRReg	$\mathbf{0.824} \pm \mathbf{0.022}$	$\mathbf{0.811} \pm \mathbf{0.013}$	$\mathbf{0.702} \pm \mathbf{0.013}$
		Graph Classification		
		ENZYMES	D&D	PROTEINS
GCN	none	0.570 ± 0.052	0.755 ± 0.028	0.740 ± 0.035
	RWRReg	$\mathbf{0.621} \pm \mathbf{0.041}$	$\mathbf{0.786} \pm \mathbf{0.024}$	$\mathbf{0.785} \pm \mathbf{0.036}$
DiffPool	none	0.661 ± 0.031	0.793 ± 0.022	0.813 ± 0.017
	RWRReg	$\mathbf{0.733} \pm \mathbf{0.032}$	$\mathbf{0.822} \pm \mathbf{0.025}$	$\mathbf{0.820} \pm \mathbf{0.038}$
k-GNN	none	0.515 ± 0.111	0.756 ± 0.021	0.763 ± 0.043
	RWRReg	$\mathbf{0.582} \pm \mathbf{0.075}$	$\mathbf{0.787} \pm \mathbf{0.022}$	$\mathbf{0.780} \pm \mathbf{0.028}$
		Triangles Test Set		
		Global	Small	Large
GCN	none	2.290	1.311	3.608
	RWRReg	**2.187**	**1.282**	**3.014**

parameters at the first layer by $n \times d^{(1)}$ (where $d^{(1)}$ is the dimension of the feature vector for each node after the first GNN layer). In this section we propose a practical way to take advantage of the injection of global structural information without increasing the number of parameters, and controlling the memory consumption during training.

3.4.1 RWRReg

From Section 3.3, the use of RWR coefficients as additional features coupled with the additional RWRReg term is the strategy that provides the highest performance improvement on all tasks. As discussed at the beginning of this section, the addition of RWR coefficients can be problematic, and hence we study the impact of using **only** the RWRReg term. We consider the same settings and tasks presented in Section 3.3, and results are shown in Table 3.6. The results show that the sole addition of the RWRReg term increases the performance of the considered models by more than **5%**. At the same time, RWRReg **(i)** does not increase the input size or the number of parameters, **(ii)** does not require additional operations at inference time, **(iii)** does not require additional supervision (it is in fact a *self-supervised* objective), **(iv)** maintains the permutation invariance of MPNN models, and **(v)** there is a vast literature on efficient methods for computing RWR, even for web-scale graphs (e.g., see Lofgren [139], Wang et al. [206], and Wei et al. [208]). Hence, the only downside of RWRReg is the storage of the RWR matrix during training on very large graphs. In the rest of this section we present a strategy to reduce the memory requirements for RWRReg, we explore how our regularization is impacted by different values of the restart parameter, and finally, we show how to compute RWRReg by using matrix multiplications to take advantage of GPU computing.

Sparsification of the RWR Matrix. To tackle the issue of storing in memory large RWR matrices, we explore how the sparsification of the RWR matrix affects the regularization of the model. In particular, we apply a *top-K* strategy: for each node, we only keep the K highest RWR weights. This approach can further take advantage of existing efficient methods to directly compute only the top-K RWR weights [140, 206, 207, 208]. As an example, TopPPR [208] provides guarantees on the precision of the returned scores, and requires only 15 seconds to retrieve the top-500 scores on a billion edge graph.

Figure 3.4 shows how different values of K impact performance on node classification (which usually is the task with the largest graphs). We can see that the addition of the RWRReg term is always beneficial. Furthermore, by taking the *top*-$\frac{n}{2}$, we can reduce the number of entries in the RWR matrix of $\frac{n^2}{2}$ elements, while still obtaining an average **3.2%** increment on the accuracy of the model. This strategy then allows the selection of the value of K that best suits the available memory, while still obtaining a high performing model (better than GCN without global structural information injection).

Impact of RWR Restart Probability. The use of RWR requires to set the restart probability parameter. We show how performance changes with different restart probabilities. Intuitively, higher restart probabilities might put more much focus on close nodes, as the random walker with frequently return to the starting node. On the other side, lower probabilities allow for more long-range exploration, but may get "trapped" into densely connected subgraphs. As such, we would expect lower probabilities to provide more information that is not already available to practical GNNs, and hence lead to higher performance. Figure 3.5 summarises how the accuracy

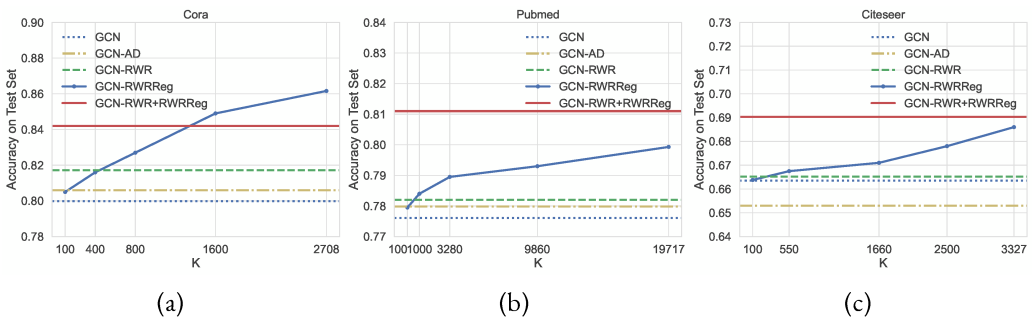


Figure 3.4: Performance of GCN on node classification for different values of K when trained with RWR-Reg with *Top-K* sparsification of the RWR matrix.

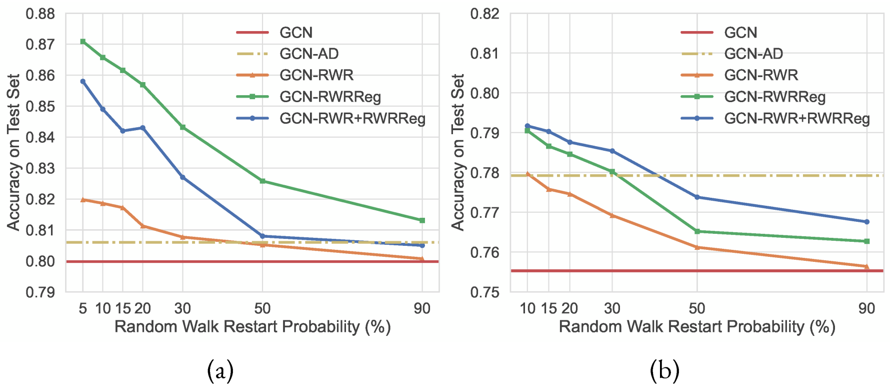


Figure 3.5: Accuracy on Cora (a), and on D&D (b), of GCN without and with the injection of structural information, and for different restart probabilities of RWR.

on node classification (side (a)) and graph classification (side (b)) changes with different restart probabilities[4]. In accordance to our intuition, higher restart probabilities focus on close nodes (and less on distant nodes), and produce lower accuracy. Furthermore, we notice how injecting RWR information is never detrimental to the performance of the model without any injection.

[4]We did not go below 5% for Cora, and 10% for D&D for stability reasons in the computation of the RWR coefficients.

3.4.2 Fast Implementation of the Random Walk with Restart Regularization

We can take advantage of the speed of GPUs by formulating the computation of the RWRReg loss through a product of matrices. Given the matrix of node embeddings **H** and the matrix with the RWR statistics **R**. We are interested in the following quantity

$$\mathcal{L}_{RWRReg} = \sum_{i,j} R_{i,j} ||\mathbf{H}_{i,:} - \mathbf{H}_{j,:}||^2$$

This is a summation over n^2 elements, which can be slow, specially for very large graphs. We then express the less as a product of matrices with the following procedure. Let us first define the matrices:

$$\hat{\mathbf{R}} = n \times n \text{ symmetric matrix with } \hat{\mathbf{R}}_{i,j} = \begin{cases} R_{i,j} + R_{j,i} & \text{for } i \neq j \\ R_{i,j} & \text{for } i = j \end{cases}$$
$$\hat{\mathbf{D}} = n \times n \text{ diagonal matrix with } \hat{D}_{i,i} = \sum_j \hat{R}_{i,j}$$
$$\boldsymbol{\Delta} = \hat{\mathbf{D}} - \hat{\mathbf{R}}$$

We can then express the loss as

$$\mathcal{L}_{RWRReg} = \sum_{i,j} R_{i,j} ||\mathbf{H}_{i,:} - \mathbf{H}_{j,:}||^2 = \sum_i \mathbf{H}_{:,i}^\top \boldsymbol{\Delta} \mathbf{H}_{:,i} = Tr(\mathbf{H}^\top \boldsymbol{\Delta} \mathbf{H})$$

Where $Tr(\cdot)$ is the trace of the matrix. Note that the matrix $\boldsymbol{\Delta}$ can be computed once, as a pre-processing step, and then used for computing the loss during training without the need of re-computing it.

3.5 Related Work

Random Walks with Restart and Graph Neural Networks. Several works have taken advantage of RWR in the context of MPNNs. Klicpera et al. [125] use RWR to create a new (weighted) adjacency matrix where message passing is performed. Li et al. [136] use random walks in a co-training scenario to add new nodes for the MPNNs' training set. Ying et al. [224] and Zhang et al. [229] use random walks to define aggregation neighbourhoods that are not confined to a fixed distance. Abu-El-Haija et al. [2] and Abu-El-Haija et al. [3] use powers of the adjacency matrix, which can be considered as random walk statistics, to define neighbourhoods of different scales. Zhuang and Ma [234] use random walks to define the positive pointwise mutual information matrix and then use it in place of the adjacency matrix in the MPNN formulation. Klicpera et al. [124] use a diffusion strategy based on RWR instead of aggregating information from neighbours. This last work has recently been extended by Bojchevski et al. [26] to scale to large graphs using RWRs to sample neighbourhoods. We remark how the aforementioned works focus on creating novel MPNN models, while we are interested in studying the impact of global structural information (which MPNNs do not have access to).

Regularization of Graph Neural Networks. Gao et al. [75], and Jiang and Lin [104] use regularization techniques to enforce that the embeddings of neighbouring nodes should be close to each other. The first uses Conditional Random Fields, while the second uses a regularization term based on the graph Laplacian. Both approaches only focus on 1-hop neighbours and do not take global information into account.

Theoretical Aspects of Graph Neural Networks. With regards to the study of the capabilities and weaknesses of GNNs, Li et al. [136] and Xu et al. [217] study the over-smoothing problem that appears in Deep-GCN architectures, while Xu et al. [216] and Morris et al. [162] characterize the relation to the Weisfeiler-Leman algorithm. Other works have expressed the similarity with distributed computing [144, 187], and the alignment with particular algorithmic structures [218]. These important contributions have advanced our understanding of the capabilities of GNNs, but they do not analyze or quantify the impact of *global* structural information.

Efficient computation of Random Walks with Restart. Our RWRReg term relies on the computation of the RWR coefficients for every node (for computing the loss function). When dealing with large graphs, there is a vast literature on fast approximations of RWR scores [6, 13, 139, 200, 206, 208].

Anonymous Random Walks. Recent work [154] has shown that *anonymous random walks* (i.e., random walks where the global identities of nodes are not known) of fixed length starting at node u are sufficient to reconstruct the local neighbourhood within a fixed distance of a node u [154]. Subsequently, anonymous random walks have been introduced in the context of learning graph representations [101]. Such results are complementary to ours, since they assume access to the distribution of *entire walks* of a given length, while our RWR representation only stores information on the probability of ending in a given node. In addition, such works do not provide a connection between RWR and 1-WL.

Overcoming the Oversquashing issue. Finally, we mention two works released during the writing of this thesis that propose strategies to overcome the oversquashing issue and allow information to be propagated more easily over the entirety of the graph [15, 55].

3.6 Conclusions

Whether *global* structural information (i.e., information that depends on the structure of the whole graph) is needed in GNNs for common tasks on graph-structured data is an open question. In this chapter we tackle this question directly at its root. In particular, we identify three strategies to inject *global* structural information into MPNN models, and we quantify their impact on popular downstream tasks. Our experiments show that the additional information significantly boosts the performance of all considered state-of-the-art models, highlighting and quantifying the importance that global structural information can have on common MPNN applications. We further discuss a novel practical regularization technique based on RWR, which leads to an average improvement of 5% on all models, and is supported by a novel connection between RWR and the 1-WL algorithm.

4 Improving Size-Generalization in Graph Neural Networks

GNNs are designed to be able to work on graphs of any size. However, empirical results show that they struggle at generalizing to graphs of sizes that differ from those in the training data [22, 79, 109, 223], and this is referred to as the problem of "size-generalization". Obtaining good size-generalization performance from smaller to larger graphs is crucial for several reasons. **First**, it is common for graphs in the same domain to vary in size. For example, social networks can range from tens to millions of nodes, and molecular graphs can go from few atoms, to large compounds with thousands of atoms. **Second**, in many scenarios, obtaining labels for smaller graphs is cheaper than for larger graphs. For example, recently deep learning is being used in combinatorial optimization problems [21], and GNNs are heavily used in this area as many problems can be formulated as graph classification instances [41, 78, 177]. Combinatorial optimization often deals with NP-hard problems, and obtaining ground truth labels can be extremely expensive for large scale graphs. **Third**, training on smaller graphs requires less computational resources.

Two approaches have been proposed in literature to tackle the poor size-generalization of GNNs. The first strategy requires an explicit definition of the generative process behind the data, which, in the case of Bevilacqua et al. [22], leads to a model requiring the computation of induced homomorphism densities over all connected k-vertex subgraphs in order to make a prediction. While this strategy shows great results on synthetic graphs produced by the assumed generative process, its benefits are reduced on real-world graphs, for which the underlying generative process is unknown. The second approach assumes access to graphs coming from different domains, or from the test distribution, and applies domain adaptation techniques to transfer knowledge across domains [60, 223]. These methods do not require ad-hoc models, but require knowledge about each graph's domain, or access to the test distribution, both of which are not always available or easily obtainable in advance.

In this chapter we consider the graph classification scenario in which we only have access to the training data (as in the first approach described above). However, in contrast to prior work, we do not try to improve the size-generalization capabilities (from smaller to larger graphs) of GNNs by handcrafting models on the base of specific knowledge or assumptions, but we study

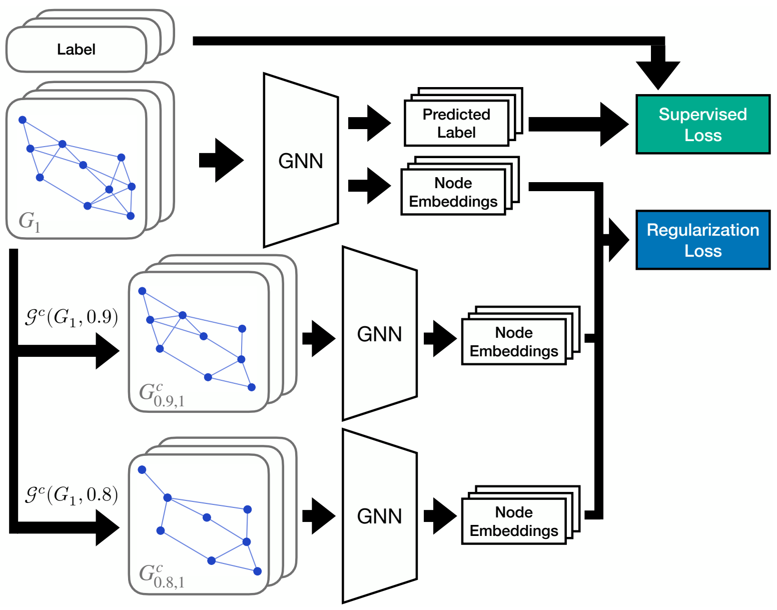


Figure 4.1: Overview of our method: given a training set of labelled graphs, we simulate a size shift by applying a coarsening function $\mathcal{G}^c$ and we regularize the model to be robust towards this shift. The regularization enforces the *distribution* of node embeddings generated by the model for the original graph and its coarsened versions to be similar.

a general form of regularization that can be applied to any GNN, and that aims at letting the model *learn* to be robust to size-shifts. The idea behind our regularization is to simulate a shift in the size of the training graphs, and to regularize the model to be robust towards this shift. The size-shift simulation is obtained using graph coarsening techniques, while the regularization is performed by minimizing the discrepancy between the distributions of the node embeddings generated by the GNN model for the original training graphs and their coarsened versions (an overview is shown in Figure 4.1).

In the experimental section we evaluate GNN models on popular benchmark datasets using a particular train/test split designed to test for size-generalization. In particular, following Bevilacqua et al. [22] and Yehudai et al. [223], we train on the 50% smallest graphs and test on the 10% largest. This split leads to an average size of the test graphs which is 3 to 9 times larger than the average size of the training graphs. Our results show that our regularization method applied to standard GNN models (GCN [121], PNA [52], GIN [216]) leads to an average performance improvement on the test set of up to 30% across datasets. Furthermore, these performance improvements show that standard GNN models trained with our regularization can achieve better size-generalization performance than more expensive models. The latter more expensive models were designed after making explicit assumptions on the generative process behind the graphs in the dataset [22], while our method relies on learning to be robust towards size-shifts.

Our contributions. Our contributions in this chapter are threefold.

- We propose a regularization strategy that can be applied to any GNN to improve its size-generalization capabilities on the task of graph classification.
- We analyze the impact that our strategy has on the embeddings generated by a GNN.
- We test our strategy on popular GNN models, and compare against previously proposed methods, showing that "standard" GNN models augmented with our regularization strategy achieve comparable or better size-generalization performance than more complex models designed after explicit assumptions on the generative process behind the data.

4.1 Preliminaries

We provide below a brief introduction to some techniques used in this chapter.

4.1.1 Centered Kernel Alignment

The Centered Kernel Alignment (CKA) [127] takes as input two matrices $\mathbf{A} \in \mathbb{R}^{m\times d'}, \mathbf{B} \in \mathbb{R}^{m\times d''}$ of representations and provides a value between 0 and 1 quantifying how aligned the representations are (allowing for $d' \neq d''$). CKA quantifies the similarity of representations learned by (possibly) different models and, for example, it gives us a way to study if different architectures are learning the same concepts, or if different layers are maintaining the same informative content.

4.1.2 Central Moment Discrepancy

The Central Moment Discrepancy (CMD) [228] is a metric used to measure the discrepancy between the distribution of high-dimensional random variables. Let p and q be two probability distributions with support in the interval $[a, b]^d$ and let c_k be the k-th order moment, then the CMD between p and q is defined as:

$$\mathrm{CMD}(p, q) = \frac{1}{|b-a|}\|\mathbb{E}(p) - \mathbb{E}(q)\|_2 + \sum_{k=2}^{\infty} \frac{1}{|b-a|^k}\|c_k(p) - c_k(q)\|_2. \tag{4.1}$$

At a practical level, we follow Zhu et al. [233], and limit the number of considered moments to 5. To align with the definition of CMD it is possible to treat node embeddings as realizations of *bounded* high dimensional distributions by applying bounded activation functions like tanh or sigmoid at the embeddings layer (as done in prior work [228, 233]). Nevertheless, in our experiments we notice good performance even when a ReLU activation is used.

4.2 Our Method

The high level idea behind our method is to “simulate” a size-shift in the training dataset, and to regularize the GNN to be robust to this shift (an overview is shown in Figure 4.1). While previous works rely on handcrafted models that incorporate domain-specific inductive biases and/or assumptions on the generative process, our method aims at *learning* to be robust towards size-shifts.

Simulating the shift. Let $\mathcal{G}^c : (G, r) \to G^c_r$ be a *coarsening function*, that takes as input a graph G with n nodes and a ratio $r \in (0, 1)$, and returns a coarsened version of the graph, G^c_r, which has $\lfloor r \times n \rfloor$ nodes. Our method is not bound to a specific coarsening function $\mathcal{G}^c$, any existing method can be used, e.g., the techniques by Jin et al. [107], Loukas [143], and Loukas and Vandergheynst [145]. Our method then proceeds in the following way: given a dataset of graphs $\mathcal{D} = \{G_1, G_2, \dots, G_\ell\}$, a coarsening function $\mathcal{G}^c$, and a set of k coarsening ratios $C = \{r_1, r_2, \dots, r_k\}$, we create a coarsened dataset $\mathcal{D}_{r_j}$ for each ratio by applying the coarsening function to each graph in the dataset: $\mathcal{D}_{r_j} = \{G^c_{r_j,1} = \mathcal{G}^c(G_1, r_j), G^c_{r_j,2} = \mathcal{G}^c(G_2, r_j), \cdots, G^c_{r_j,\ell} = \mathcal{G}^c(G_\ell, r_j)\}, \forall j = 1, 2, \dots, k$. When the graph is attributed, we obtain the features for each node of the coarsened version of the graph by aggregating the features of the corresponding nodes in the original graph using simple aggregations (e.g., mean, max, sum), as commonly done in readout functions for GNNs [213].

Regularizing the GNN. Given the graph dataset $\mathcal{D}$, its coarsened versions $\mathcal{D}_{r_1}, \mathcal{D}_{r_2}, \cdots, \mathcal{D}_{r_k}$, and a GNN f_θ, where θ are the parameters, we aim at regularizing the GNN so that the distribution of the node embeddings generated by the model for the graphs in the original dataset and their coarsened version is similar. In other words, for a given graph, we want the model to generate a *distribution* of node embeddings that is robust across different coarsened versions of the graph. In more detail, we regularize a GNN by minimizing the CMD [228] between the distribution of the node embeddings generated by the GNN for the original graph and the distribution of the node embeddings generated by the GNN for the coarsened version(s) of the graph. We choose CMD as it has proven to be successful and stable as a regularization term for non-linear models [142, 233]. Formally, let $\mathcal{L}$ be the supervised loss function that is used to train the model (e.g. cross-entropy for classification), and let λ be a term measuring the strength of the regularization, our optimization problem is

$$\operatorname*{argmin}_{\theta} \mathcal{L} + \lambda \mathcal{L}_{\text{size}}, \text{ where } \mathcal{L}_{\text{size}} = \sum_{j=1}^{k} \sum_{i=1}^{\ell} \text{CMD}(f_\theta(G_i), f_\theta(G^c_{r_j,i})) \tag{4.2}$$

Our method is model-agnostic, and can hence be applied to any GNN.

Pseudocode and practical aspect. Algorithm 1 presents the pseudocode for computing our regularization loss for a batch of graphs during training. The pseudocode is presented in an extended manner for clarity, but at a practical level, the coarsened versions of the training graphs are pre-computed before training, and the computation of the loss is done in a vectored manner so it is computed concurrently for all graphs in the batch in one pass (i.e., we do not iterate through the graphs in the batch). At a practical level, in our experiments we notice that training a model with our regularization introduces a 50% overhead in the running time for a training epoch w.r.t. training the same model without regularization.

Algorithm 1 Computing Regularization Loss for an Input Batch during Training

```
Require: Coarsening ratios $C$, coarsening function $\mathcal{G}^c$(graph, ratio)
Input: Batch $B$ of size $n_b$, GNN model $f_\theta$
  coarsened_batches ← {} {Create a new batch of coarsened graphs for each ratio}
  for $r$ in $C$ do
    Batch $B_r \leftarrow []$
    for $G$ in $B$ do
      $G_r^c \leftarrow \mathcal{G}^c(G, r)$
      $B_r$.add($G_r^c$)
    end for
    coarsened_batches ← coarsened_batches $\cup B_r$
  end for
  $\ell \leftarrow 0$ {Initialize loss}
  $\ell \leftarrow 0$ {Initialize loss}
  for $B_r$ in coarsened_batches do
    for $i$ in $\{1, 2, \dots, n_b\}$ do
      embs_og = $f_\theta(B[i])$ {Compute node embeddings for original graph}
      embs_coarse = $f_\theta(B_r[i])$ {Compute node embeddings for coarsened version of graph}
      $\ell \leftarrow \ell +$ CMD(embs_og, embs_coarse) {Compute CMD between node embeddings}
    end for
  end for
  Return $\ell$
```

4.2.1 Limitations

Similarly to previous works [22, 223] we are assuming that there are some properties that determine the label of a graph and that do not depend on the size of the graph. In a scenario in which small graphs do not carry information that is relevant for solving the task on larger graphs, we do not expect our regularization to provide substantial benefits, and the best option would be to include larger graphs in the training set. In our experiments, the ratio between the average size of the graphs in the test set and the average size of the graphs in the training set is between 3 and 9. While our method shows significant performance improvements in this setting, it may show lower performance improvements when this ratio reaches much higher values.

4.3 Analysis of Node Embeddings

Table 4.1: Average CKA values between the node embeddings generated by two models, one trained with and one without our regularization, across the graphs in a dataset and their coarsened versions.

Dataset	**NCI1**	**NCI109**	**PROTEINS**	**DD**
Original	0.43 ± 0.06	0.58 ± 0.10	0.45 ± 0.06	0.47 ± 0.01
Coarsened	0.12 ± 0.06	0.38 ± 0.13	0.34 ± 0.08	0.40 ± 0.01

Before evaluating how our regularization impacts the size-generalization performance of a model, we analyze the effects that our regularization has on the embeddings generated by the model. In order to do this, we consider two identical GIN [216] models and we train one with our regularization and one without. We then use these models to generate node embeddings and use the Central Kernel Alignment (CKA) [127] to study the generated representations, similarly to Joshi et al. [110]. CKA takes as input two matrices $A \in \mathbb{R}^{m \times d'}, B \in \mathbb{R}^{m \times d''}$ of representations and provides a value between 0 and 1 quantifying how aligned the representations are (allowing for $d' \neq d''$). CKA quantifies the similarity of representations learned by (possibly) different models and gives us a way to study the effects of our regularization. We report results averaged over 10 different random seeds.

First, we ask if a model trained with our regularization and a model trained without our regularization produce similar node embeddings. To compare the node embeddings *between* the two models we obtain a representation for each graph by concatenating the node embeddings for that graph. We then compute the CKA between the representations generated by the model trained with regularization and the representations from model trained without regularization. We do this for the original graphs and the coarsened versions of the graphs, and report the average CKA. Results shown in Table 4.1 highlight that there is low alignment between the embeddings generated by a model trained with regularization and one trained without regularization (the average CKA across datasets is 0.48 for the original graphs). This is even more apparent for the coarsened graphs, showing that our regularization is impacting the way a model generates embeddings, especially for size-shifted versions of the graphs. To analyze the trend more in detail, we also plot the CKA values across coarsening ratios in Figure 4.2 (a). Here we notice that the CKA between the embeddings generated by a model trained with our regularization and a model trained without,

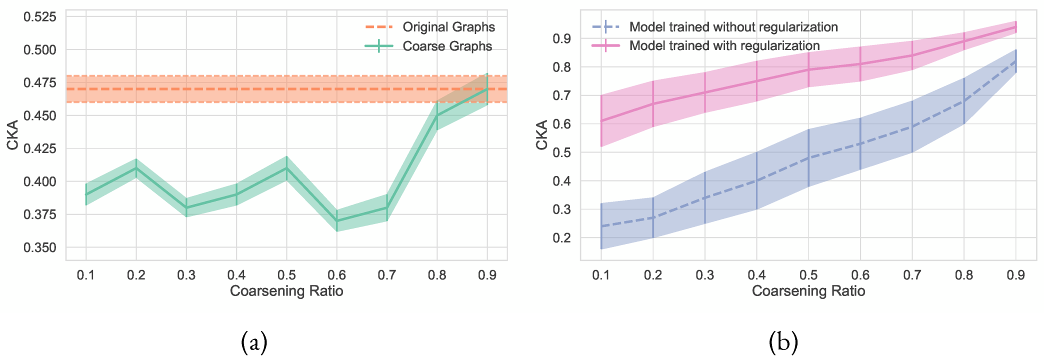
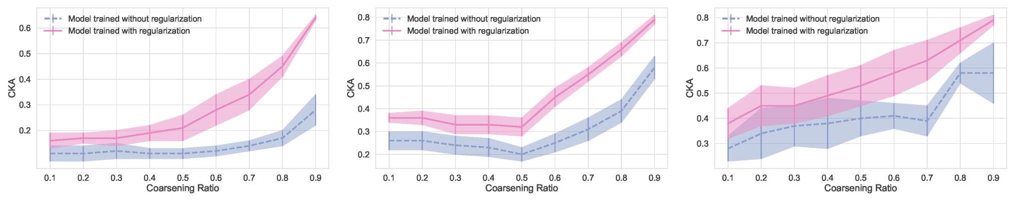


Figure 4.2: (a) Average CKA values between the node embeddings from two models, one trained with our regularization and one without, across the original graphs in the DD dataset and their coarsened versions for different ratios. The models produce representations which are not aligned (values are far from 1.0), specially for coarsened graphs. (b) Average CKA values between the node embeddings generated by the same model for the original graphs in DD and their coarsened versions. A model trained with our regularization produces representations of the coarsened graphs that are more aligned to the representations produced for the original graphs (i.e., it learns to be robust to size shifts).

decreases significantly with ratios lower than 0.8. This shows that, while the alignment between the representations learned by the models trained with and without our regularization is low even for a coarsening ratio of 0.9 (average CKA value is 0.47), it becomes 15% lower (on average) for ratios smaller than 0.7, indicating that when the size shift is strong, there is a stronger misalignment between models trained with and without regularization.

Second, we ask if a model trained with our regularization is in fact being more robust to size shifts with respect to a model trained without our regularization. To answer this question we analyze the alignment between the representations generated by the same model, trained with or without regularization, on original graphs and coarsened versions of the graphs. In Figure 4.2 (b) we plot the average CKA between the node embeddings for the original graphs and the node embeddings for the coarsened versions of the graphs, both generated by the same model, for different ratios for the DD dataset. The plot shows that a model trained with our regularization is learning to be more robust to size shifts, as its representation for coarsened versions of the graphs are much more aligned to the representations for the original graphs compared to a model trained without regularization. This is even more apparent for low coarsening ratios (i.e., when the shift is stronger), indicating that our regularization is in fact making the model more robust to size shifts. This trend is also confirmed for the other considered datasets, as shown in Figure 4.3. The reason why the CKA values for a model trained with and without regularization tend to become similar for small ratios (which is instead not apparent on the DD dataset shown in Figure 2 (b) in the main paper) is that the graphs in NCI1, NCI109, and PROTEINS are much smaller than the



(a) NCI1 (b) NCI109 (c) PROTEINS

Figure 4.3: Average CKA values between the node embeddings generated by the same model for the original graphs in (a) NCI1, (b) NCI109, (c) PROTEINS, and their coarsened versions. A model trained with our regularization produces representations of the coarsened graphs that are more aligned to the representations produced for the original graphs (i.e. it learns to be robust to size shifts).

graphs in DD (see Section 4.4), and hence graphs tend to become completely uninformative with ratios lower than 0.5.

4.4 Evaluation

We now present the framework used to test our proposed method, and compare against prior work.

Datasets. Following Bevilacqua et al. [22] and Yehudai et al. [223], we evaluate on datasets from the TUDataset library [160] with a train/test split explicitly designed to test for size-generalization. In particular, as introduced in Yehudai et al. [223], the training split contains the graphs with size smaller or equal than the 50-th percentile, and the test split contains graphs with a size larger or equal than the 90-th percentile. With this split, the average size of the test graphs is 3 to 9 times larger than the average size of the training graphs (in more detail, it is 3 for NCI1 and NCI109, 9 for PROTEINS, and 5 for DD). We use 10% of the training graphs as a validation set. Dataset statistics are shown in Table 4.2.

Models. As in Bevilacqua et al. [22], we consider three standard GNN models: GCN [121], GIN [216], and PNA [52], a more expressive GNN: RPGNN [163], and two graph kernels: the Graphlet Counting kernel (GC Kernel) [193], and the WL Kernel [192]. We further apply the Invariant Risk Minimization (IRM) [7] penalty to the standard GNN models, and report results for the E-Invariant models ($\Gamma_{\text{1-hot}}$, Γ_{GIN}, Γ_{RPGIN}) introduced in [22]. For IRM, we follow prior work [22], and artificially create two environments: one with graphs with size smaller than the median size of the graphs in the training set, and one with graphs with size larger than the median size in the training set.

Table 4.2: Dataset statistics; this table is taken from Bevilacqua et al. [22] and Yehudai et al. [223].

	NCI1			NCI109		
	all	Smallest 50%	Largest 10%	all	Smallest 50%	Largest 10%
Class A	49.95%	62.30%	19.17%	49.62%	62.04%	21.37%
Class B	50.04%	37.69%	80.82%	50.37%	37.95%	78.62%
Num. of graphs	4110	2157	412	4127	2079	421
Avg. graph size	29	20	61	29	20	61

	PROTEINS			DD		
	all	Smallest 50%	Largest 10%	all	Smallest 50%	Largest 10%
Class A	59.56%	41.97%	90.17%	58.65%	35.47%	79.66%
Class B	40.43%	58.02%	9.82%	41.34%	64.52%	20.33%
Num. of graphs	1113	567	112	1178	592	118
Avg. graph size	39	15	138	284	144	746

Computing Infrastructure. The experiments were run on a GPU cluster with 8 Nvidia 1080Ti, and on a CPU cluster equipped with 8 CPUs 12-Core Intel Xeon Gold 5118 @2.30GHz, with 1.5Tb of RAM.

Hyperparameters and Evaluation Protocol. For the standard GNN models we adopt the same hyperparameters of Bevilacqua et al. [22], which were obtained from a tuning procedure involving number of layers, learning rate, batch size, hidden layers dimension, and regularization terms. The GCN [121], GIN [216], and PNA [52] models used in our experiments have 3 graph convolution layers and a final multi-layer perceptron with a softmax activation function to obtain the predictions. Batch norm is used in between graph convolution layers, and ReLU is used as activation function. The networks are trained with a dropout of 0.3, and were tuned using the validation set. In particular the batch size was chosen between 64 and 128, the learning rate between 0.01, 0.005, and 0.001, and the network width between 32 and 64. All models are trained with early stopping (i.e., taking the weights at the epoch with best performance on the validation set). We do not modify the original hyperparameters when we apply our regularization. We tune the regularization coefficient λ and the coarse ratios C for GIN on the PROTEINS validation set (we find $\lambda = 0.1$ and $C = \{0.8, 0.9\}$ to be the best), and we apply these settings to *all models and datasets* when using our regularization, to show that our method can work without extensive (and expensive) hyperparameter tuning. We use the SGC coarsening algorithm [107] to obtain the coarsened versions of the graphs (chosen for its theoretical properties). The only hyperparameter that is tuned on a per-dataset basis is the aggregation strategy used to obtain the features for the nodes in the coarsened versions of the graphs (in particular we take the best performing between 'sum', 'max', and 'mean'). For all other models we use the hyperparameters introduced by their respective papers. Bevilacqua et al. [22] presented results by averaging over 10 runs (each time with

Table 4.3: Average Matthews Correlation Coefficient (MCC) for standard GNN models over the size-generalization test set. The models have been trained with (✓) and without (✗) our regularization method. The right-most column shows the average improvement brought by our regularization.

Dataset	**NCI1**		**NCI109**		**PROTEINS**		**DD**		Avg.
Reg.	✗	✓	✗	✓	✗	✓	✗	✓	Impr.
PNA	0.19 ± 0.08	0.22 ± 0.07	0.23 ± 0.07	0.24 ± 0.07	0.22 ± 0.12	0.33 ± 0.09	0.23 ± 0.09	0.27 ± 0.08	**+22%**
GCN	0.17 ± 0.06	0.25 ± 0.06	0.15 ± 0.06	0.19 ± 0.06	0.21 ± 0.10	0.29 ± 0.13	0.24 ± 0.07	0.26 ± 0.07	**+30%**
GIN	0.19 ± 0.06	0.23 ± 0.08	0.18 ± 0.05	0.20 ± 0.05	0.25 ± 0.07	0.36 ± 0.11	0.23 ± 0.09	0.25 ± 0.09	**+21%**

a different random seed), but given the high variance observed, we re-run the experiments for all the models and present the results averaged over 50 runs.

Tuning and Evaluation Procedure. To identify the values of λ and C to use for our method, we tried different values ($\lambda = \{1.0, 0.1, 0.01, 0.001\}$ and $C = \{(0.5), (0.8), (0.9), (0.5, 0.8), (0.5, 0.9), (0.8, 0.9)\}$) on the validation set using a GIN model on the PROTEINS dataset. We found that $\lambda = 0.1$ and $C = (0.8, 0.9)$ performed best on the validation set, and we then used those values on the validation set of other datasets and with other models. As the results on the validation sets were leading to better performance with respect to a model trained without regularization, we adopted the same values of λ and C for all datasets and models to show that our method can work without extensive (and expensive) hyperparameter tuning. It is possible that dataset-specific and model-specific tuning can lead to higher results.

We first obtained the results for Table 4.1, Figure 4.2, Table 4.3, and Table 4.4. The ablation is then used to understand the impact of the components of our method only after having evaluated it, as is the standard procedure for ablation studies.

4.4.1 Results

Table 4.3 shows how the three considered standard GNN models perform on the test set, containing the 10% largest graphs in the dataset, when trained with and without our regularization on the 50% smallest graphs in the dataset. As this split leads to an imbalanced dataset, we follow previous works and report the results in terms of Matthews correlation coefficient (MCC), which has been shown to be more reliable in imbalanced settings with respect to other common metrics [48]. MCC gives a value between -1 and 1, where -1 indicates perfect disagreement, 0 is the value for a random guesser, and 1 indicates perfect agreement between the predictions and the true labels. The results show that the use of the proposed regularization is always beneficial to the performance of the models, and that it leads to an average improvement across datasets of **21 to 30%**.

Table 4.4: Comparison of average Matthews Correlation Coefficient (MCC) over the size-generalization test data between the standard GNN models trained with our proposed regularization and previously proposed methods. Highlighted are the **first**, **second**, and **third** best performing models per dataset.

Dataset	**NCI1**	**NCI109**	**PROTEINS**	**DD**
PNA (reg) [ours]	0.22 ± 0.07	**0.24 ± 0.07**	**0.33 ± 0.09**	**0.27 ± 0.08**
GCN (reg) [ours]	**0.25 ± 0.06**	0.19 ± 0.06	0.29 ± 0.13	**0.26 ± 0.07**
GIN (reg) [ours]	0.23 ± 0.08	0.20 ± 0.05	**0.36 ± 0.11**	0.25 ± 0.09
PNA + IRM [7]	0.17 ± 0.07	0.20 ± 0.07	0.21 ± 0.12	0.24 ± 0.08
GCN + IRM [7]	0.22 ± 0.06	0.20 ± 0.06	0.23 ± 0.16	0.23 ± 0.08
GIN + IRM [7]	0.18 ± 0.06	0.15 ± 0.04	0.24 ± 0.08	0.21 ± 0.10
WL kernel [192]	**0.39 ± 0.00**	**0.21 ± 0.00**	0.00 ± 0.00	0.00 ± 0.00
GC kernel [193]	0.02 ± 0.00	0.01 ± 0.00	**0.29 ± 0.00**	0.00 ± 0.00
RPGIN [163]	0.18 ± 0.06	0.16 ± 0.04	0.22 ± 0.08	0.13 ± 0.04
$\Gamma_{\text{1-hot}}$ [22]	0.15 ± 0.05	**0.22 ± 0.06**	0.18 ± 0.08	0.22 ± 0.09
Γ_{GIN} [22]	0.24 ± 0.05	0.16 ± 0.07	0.28 ± 0.10	**0.27 ± 0.05**
Γ_{RPGIN} [22]	**0.26 ± 0.05**	0.19 ± 0.06	0.26 ± 0.07	0.20 ± 0.05

In Table 4.4, we compare the standard GNNs trained with our regularization strategy against more expressive models (RPGNN), graph kernels, models trained with the IRM strategy, and the E-invariant models. We notice that on 3 out of 4 datasets, a "standard" GNN trained with our regularization obtains the best performance on the test set composed of graphs with size larger than those present in the training set. Furthermore on all datasets there is at least one model trained with our regularization in the top-3 best performing models. We also confirm previous results [22, 60] showing that IRM is not effective in the graph domain, and that using theoretically more expressive models, like RPGNN, does not necessarily lead to good size-generalization performance. Graph kernels are highly dataset-dependent, and, while they can perform well for some datasets, in many cases they fail to perform better than a random classifier. Perhaps more surprisingly, on 3 out of 4 datasets there is at least one "standard" GNN trained with our regularization that performs comparably or better than the best E-Invariant model. In fact, E-invariant models are tied to the assumed causal model for the generation of graphs, which is not guaranteed to hold reliably for real-world datasets for which we do not have this kind of information. Furthermore, E-invariant models require the computation of induced homomorphism densities over all possible connected k-vertex subgraphs both at training time and at test time. Our method instead tries to *learn* to be robust to size-shifts, does not require any additional computation at inference time, and yet can lead even simple models like GCN to have size-generalization performance comparable to E-invariant models.

Finally, in addition to the benchmarks considered by Bevilacqua et al. [22], we also experiment on a dataset from a different domain. In particular, we consider a social network dataset obtained from Deezer [183]. We follow the same evaluation strategy as before: train on the 50% smallest and test on the 10% largest, and we apply our regularization with $\lambda = 0.1$ and $C = \{0.8, 0.9\}$ without any additional hyperparameter tuning. Results shown in Table 4.5 confirm the effectiveness of our method, with improvements of up to 20%.

Table 4.5: Average MCC results on the Deezer dataset for models trained with (✓) and without (✗) our regularization.

Dataset	**Deezer**	
Reg.	✗	✓
PNA	0.59 ± 0.06	0.64 ± 0.07
GCN	0.49 ± 0.10	0.59 ± 0.06
GIN	0.55 ± 0.08	0.61 ± 0.07

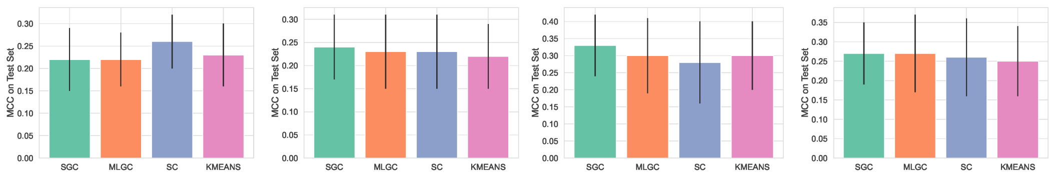
4.4.2 Ablation Study

In this section we study the contribution and importance of the different components of our regularization. We focus primarily on the PNA model because of its performance in the previous analysis, but similar conclusions are observed and reported also for GIN and GCN (in Table 4.8, Table 4.7, Table 4.10, and Table 4.9).

Changing Size of Coarsened Graphs. In Table 4.6, we train a PNA model (results for GCN and GIN can be found in Table 4.7, and Table 4.8) with our regularization strategy using different coarsening ratios C. We notice an overall trend in which the performance decreases as the coarsening ratio decreases. This follows intuitively as very low coarsening ratios may lead to uninformative graphs. Furthermore we notice that using all ratios (from 0.1 to 0.9) is usually not effective and setting

Table 4.6: Average MCC results on the size-generalization test set for a PNA model trained with our regularization strategy using different coarsening ratios.

Datasets	**NCI1**	**NCI109**	**PROTEINS**	**DD**
Ratio(s)				
0.1	0.07 ± 0.11	0.19 ± 0.08	0.12 ± 0.15	0.07 ± 0.14
0.2	0.11 ± 0.12	0.20 ± 0.08	0.18 ± 0.16	0.20 ± 0.11
0.3	0.15 ± 0.09	0.22 ± 0.07	0.17 ± 0.16	0.22 ± 0.10
0.4	0.18 ± 0.08	0.22 ± 0.08	0.23 ± 0.14	0.26 ± 0.11
0.5	0.22 ± 0.08	0.24 ± 0.07	0.29 ± 0.12	0.22 ± 0.09
0.6	0.23 ± 0.08	0.23 ± 0.07	0.26 ± 0.09	0.27 ± 0.10
0.7	0.22 ± 0.07	0.24 ± 0.06	0.30 ± 0.14	0.24 ± 0.06
0.8	0.24 ± 0.06	0.23 ± 0.07	0.32 ± 0.11	0.25 ± 0.09
0.9	0.19 ± 0.06	0.21 ± 0.08	0.28 ± 0.12	0.25 ± 0.10
$\{0.1, 0.9\}$	0.12 ± 0.10	0.20 ± 0.06	0.14 ± 0.16	0.14 ± 0.14
$\{0.5, 0.9\}$	0.23 ± 0.08	0.22 ± 0.07	0.32 ± 0.12	0.27 ± 0.08
$\{0.8, 0.9\}$	0.22 ± 0.07	0.24 ± 0.07	0.33 ± 0.09	0.27 ± 0.08
$\{0.3, 0.7\}$	0.22 ± 0.09	0.21 ± 0.08	0.24 ± 0.11	0.25 ± 0.09
ALL	0.18 ± 0.09	0.18 ± 0.09	0.17 ± 0.14	0.23 ± 0.10



(a) NCI1 (b) NCI109 (c) PROTEINS (d) DD

Figure 4.4: Average MCC on the test set of a PNA model trained with our regularization strategy using different coarsening methods: Spectral Graph Coarsening (SGC), Multi-level Graph Coarsening (MLGC), Spectral Clustering (SC), K-Means (KMEANS).

$C = \{0.8, 0.9\}$ seems a good default option, leading to the best performance on 3 out of 4 datasets.

Changing Coarsening Method. In Figure 4.4 we show how the performance of a PNA model change when it is regularized with our method using graphs coming from different coarsening functions. We consider the Spectral Graph Coarsening (SGC) and the Multi-level Graph Coarsening (MLGC) techniques introduced by Jin et al. [107], and two baselines, referred to as SC and KMEANS, based on standard spectral clustering [84] and K-Means clustering [138]. For the baselines we apply the clustering, merge the nodes belonging to the same cluster into a single super-node, and create an edge between two super-nodes if there is at least one edge in the original graph going from a node in one cluster to the other. For all coarsening methods we use the same coarsening ratios as above (i.e., $C = \{0.8, 0.9\}$), and we apply the same training procedure and regularization weight λ, keeping all hyperparameters equal across methods. We notice that our regularization strategy is quite robust towards the choice of the coarsening method, but in most cases specialised methods like SGC and MLGC provide the best results. Results for GCN and GIN are shown in Table 4.9, and Table 4.10. While the general trend seems to hold also for these models, for the NCI datasets, which are the ones with the smallest graphs, we observe a greater sensitivity to the choice of coarsening function.

4.5 Related Work

Size Generalization in Graph Neural Networks. Many works have noticed poor size-generalization in GNNs [22, 79, 109, 223]. For instance, Joshi et al. [109] consider the task of learning solutions for the traveling salesman problem, and show that GNNs struggle at generalizing to graphs larger than those appearing in the training set. Similarly, Gasteiger et al. [79] observe that GNNs applied to the molecular domain can be very sensible to several shifts in the data, including shifts in the size of the graphs. Some works have reported good size-generalization by creating

Table 4.7: Table shows the average Matthews correlation coefficient (MCC) for a GCN model trained with our proposed regularization with the Spectral Graph Coarsening (SGC) strategy applied with different coarsening ratios.

Datasets Ratio(s)	**NCI1**	**NCI109**	**PROTEINS**	**DD**
0.1	0.09 ± 0.12	0.23 ± 0.07	0.05 ± 0.14	0.15 ± 0.09
0.2	0.17 ± 0.11	0.23 ± 0.07	0.05 ± 0.16	0.20 ± 0.08
0.3	0.18 ± 0.10	0.23 ± 0.07	0.08 ± 0.18	0.23 ± 0.07
0.4	0.21 ± 0.07	0.24 ± 0.06	0.09 ± 0.17	0.24 ± 0.09
0.5	0.23 ± 0.07	0.22 ± 0.05	0.21 ± 0.14	0.27 ± 0.09
0.6	0.24 ± 0.05	0.21 ± 0.06	0.23 ± 0.17	0.25 ± 0.08
0.7	0.26 ± 0.06	0.21 ± 0.05	0.25 ± 0.16	0.24 ± 0.08
0.8	0.27 ± 0.06	0.20 ± 0.04	0.29 ± 0.15	0.27 ± 0.08
0.9	0.25 ± 0.07	0.19 ± 0.05	0.22 ± 0.14	0.25 ± 0.09
$\{0.1, 0.9\}$	0.15 ± 0.11	0.21 ± 0.06	0.09 ± 0.15	0.21 ± 0.09
$\{0.5, 0.9\}$	0.22 ± 0.05	0.22 ± 0.06	0.28 ± 0.12	0.25 ± 0.08
$\{0.8, 0.9\}$	0.25 ± 0.06	0.19 ± 0.06	0.29 ± 0.13	0.26 ± 0.07
$\{0.3, 0.7\}$	0.21 ± 0.07	0.22 ± 0.05	0.10 ± 0.14	0.23 ± 0.07
ALL	0.17 ± 0.08	0.23 ± 0.09	0.12 ± 0.16	0.21 ± 0.09

Table 4.8: Table shows the average Matthews correlation coefficient (MCC) for a GIN model trained with our proposed regularization with the Spectral Graph Coarsening (SGC) strategy applied with different coarsening ratios.

Datasets Ratio(s)	**NCI1**	**NCI109**	**PROTEINS**	**DD**
0.1	0.07 ± 0.11	0.04 ± 0.11	0.26 ± 0.12	0.22 ± 0.10
0.2	0.07 ± 0.11	0.07 ± 0.11	0.30 ± 0.10	0.24 ± 0.12
0.3	0.07 ± 0.11	0.10 ± 0.09	0.29 ± 0.15	0.25 ± 0.09
0.4	0.02 ± 0.13	0.09 ± 0.09	0.31 ± 0.15	0.24 ± 0.09
0.5	0.07 ± 0.11	0.10 ± 0.11	0.32 ± 0.12	0.25 ± 0.09
0.6	0.02 ± 0.11	0.08 ± 0.07	0.34 ± 0.12	0.26 ± 0.10
0.7	0.01 ± 0.09	0.09 ± 0.06	0.31 ± 0.14	0.25 ± 0.11
0.8	0.15 ± 0.10	0.15 ± 0.07	0.36 ± 0.11	0.27 ± 0.08
0.9	0.22 ± 0.08	0.19 ± 0.05	0.34 ± 0.10	0.25 ± 0.10
$\{0.1, 0.9\}$	0.09 ± 0.10	0.05 ± 0.09	0.32 ± 0.15	0.22 ± 0.10
$\{0.5, 0.9\}$	0.03 ± 0.10	0.09 ± 0.07	0.32 ± 0.13	0.23 ± 0.09
$\{0.8, 0.9\}$	0.23 ± 0.08	0.20 ± 0.05	0.36 ± 0.11	0.25 ± 0.09
$\{0.3, 0.7\}$	0.05 ± 0.12	0.08 ± 0.09	0.26 ± 0.15	0.24 ± 0.10
ALL	0.05 ± 0.12	0.11 ± 0.09	0.25 ± 0.16	0.23 ± 0.10

ad-hoc models that leverage the properties of the domain [16, 18, 71, 185, 202]. For example, Tsubaki and Mizoguchi [202] impose constraints dictated by the physical properties of their data, and report good size-generalization in predicting the energy of a molecule. While these works show that placing the right inductive biases can help the size-generalization capabilities of GNNs, the current literature is lacking a general method that can be applied on generic GNNs.

Bevilacqua et al. [22] tackle the issue of poor size-generalization by assuming a causal model describing the generative process for the graphs in the dataset, and designing a specific model which can be invariant to the size of the graphs obtained through this causal model. While this strategy shows great size-generalization capabilities on synthetic graphs generated according to their causal model, its benefits decrease when applied to real-world graphs where there are no guarantees that the causal model is correct. This method requires an ad-hoc model and additional computations at both train and test time, while our method can be applied on *any* GNN and does not require pre-computations at test time. Yehudai et al. [223] provide a theoretical and empirical analysis showing that, even for simple tasks, it is not trivial to obtain a GNN with good size-generalization properties. Yehudai et al. [223] consider a different scenario from ours, as, together with the labelled training graphs, they assume access to graphs from the test distribution, and treat the task as a domain-adaptation scenario.

Table 4.9: Table shows the average Matthews correlation coefficient (MCC) for a GCN model trained with our proposed regularization and different coarsening strategies: Spectral Clustering (SC), Spectral Graph Coarsening (SGC), Multilevel Graph Coarsening (MLGC), K-Means (KMEANS).

Datasets	NCI1	NCI109	PROTEINS	DD
GCN-SGC	0.25 ± 0.06	0.19 ± 0.06	0.29 ± 0.13	0.26 ± 0.07
GCN-MLGC	0.21 ± 0.07	0.21 ± 0.06	0.27 ± 0.14	0.25 ± 0.06
GCN-SC	0.28 ± 0.07	0.18 ± 0.06	0.24 ± 0.16	0.24 ± 0.07
GCN-KMEANS	0.28 ± 0.06	0.18 ± 0.05	0.25 ± 0.15	0.24 ± 0.07

Table 4.10: Table shows the average Matthews correlation coefficient (MCC) for a GIN model trained with our proposed regularization and different coarsening strategies: Spectral Clustering (SC), Spectral Graph Coarsening (SGC), Multilevel Graph Coarsening (MLGC), K-Means (KMEANS).

Datasets	NCI1	NCI109	PROTEINS	DD
GIN-SGC	0.23 ± 0.08	0.20 ± 0.05	0.36 ± 0.11	0.25 ± 0.09
GIN-MLGC	0.22 ± 0.07	0.19 ± 0.06	0.36 ± 0.10	0.25 ± 0.10
GIN-SC	0.08 ± 0.11	0.07 ± 0.08	0.32 ± 0.12	0.21 ± 0.10
GIN-KMEANS	0.21 ± 0.09	0.20 ± 0.06	0.35 ± 0.08	0.28 ± 0.09

Graph coarsening. While there is no consensus on what is the best metric to consider for coarsening a graph, many methods and metrics have been proposed in recent years [28, 64, 107, 143, 145]. There have also been some attempts at using GNNs for the task of graph coarsening [38, 147]. Orthogonally, graph coarsening has been used to reduce the computational resources needed for training GNNs on large graphs [98, 105]. For a detailed presentation of graph coarsening we refer to Chen et al. [46].

Invariant risk minimization, domain adaptation, and regularization with discrepancy measures. Our method is conceptually related to Invariant Risk Minimization (IRM) [7], which aims at learning representations that are invariant across training environments (where different training environments are intended as sets of data points collected under different conditions). IRM has been shown to have several shortcomings, specially when the data comes from a single environment (e.g. when there is access only to small graphs) [22, 181]. Our method does not require access to different training environments, which may not be easy to obtain in practical scenarios, and, as also observed in Bevilacqua et al. [22] and Ding et al. [60], our results show that IRM seems ineffective for improving size-generalization in GNNs.

Similarly to IRM, the field of domain adaptation [19] subsumes access to (unlabelled) external data, in addition to labelled training data, and aims at transferring knowledge between domains. Domain adaptation has been used in a large variety of fields, and many surveys are available [54, 83, 210]. Ding et al. [60] analyze how existing domain adaptation algorithms perform on graph data, and observe that these tend to not be effective, indicating that new methods specific for graph data are needed.

Discrepancy measures that have been used for regularizing deep learning models are the Maximum Mean Discrepancy (MMD) [65, 141, 142, 219] and the Central Moment Discrepancy

(CMD) [173, 228, 233], which has been shown to be empirically more effective. Finally, we mention the work by Zhu et al. [233] which, similarly to ours, uses CMD to obtain a GNN that is robust to biases in the sampling process that is used to select the nodes for the training set in node classification scenarios.

4.6 Conclusions

GNNs are heavily used models for the task of graph classification. In many domains it is typical for graphs to vary in size, and while GNNs are designed to be able to process graphs of any size, empirical results across literature have highlighted that GNNs struggle at generalizing to sizes unseen during training. In this Chapter we introduce a regularization strategy that can be applied on any GNN, and that improves size-generalization performance from smaller to larger graphs by up to 30%.

5 Learning Multi-Task Representations

GNNs are centered around the concept of *node representation learning*, and typical uses of GNNs for downstream tasks follow the *encoder-decoder* structure (see Section 2.3). The encoder produces node embeddings (low-dimensional vectors capturing relevant structural and feature-related information about each node), while the decoder uses the embeddings to carry out the desired downstream task. The model is then trained in an end-to-end manner, leading to highly specialized node embeddings. While this approach can achieve state-of-the-art performance, it also affects the generality and reusability of the embeddings. In fact, taking the node embeddings generated by an encoder trained for a given task, and using them to train a decoder for a different task leads to substantial performance loss (see Figure 5.1).

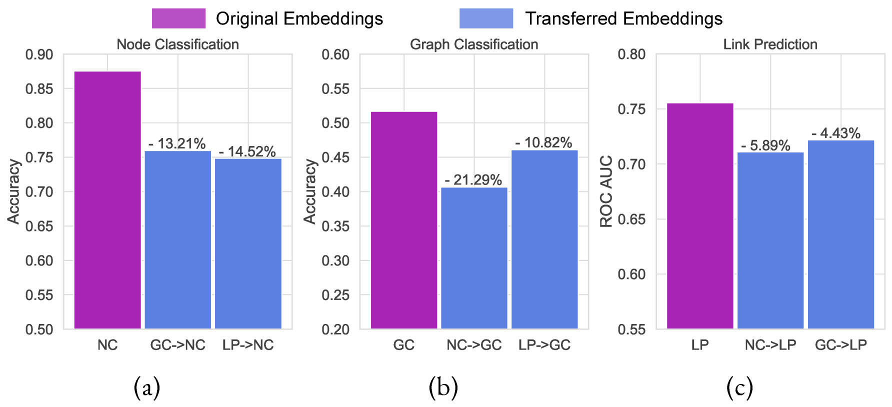


Figure 5.1: Performance drop when transferring node embeddings on the ENZYMES dataset to perform the tasks: (a) Node Classification (NC), (b) Graph Classification (GC), and (c) Link Prediction (LP). "x ->y" indicates that the embeddings obtained from a model trained on task x are used to train a network for task y.

The low transferability of node embeddings requires the use of one specialized encoder and one specialized decoder for each considered task. However, there are many practical scenarios in which *multiple* tasks must be performed on the same graph(s). For example, in a social network, both classification tasks (e.g., classify users as spammer or non-spammer) and link prediction tasks (e.g., suggest new connections between users) must be performed. While a trivial solution might be to deploy a specific model (generating specific node embeddings) for each task, this inevitably leads to significant overhead.

This chapter studies the problem of generating node embeddings that can be used for multiple tasks, which is a scenario with many practical applications that, nonetheless, has received little attention from the GNN community. In more detail, we propose a multi-task representation learning procedure, based on optimization-based meta-learning [68], that learns a GNN encoder producing node embeddings that generalize across multiple tasks. Our focus is on the most studied tasks in the GNN literature: graph classification, node classification, and link prediction.

The proposed meta-learning procedure is targeted towards multi-task representation learning and takes advantage of MAML [68] and ANIL [178] to reach a setting of the parameters where a few steps of gradient descent on a given task lead to good performance on that task. This procedure leads to an encoder-decoder model that can easily be adapted to perform each of the tasks *singularly*, and hence encourages the encoder to learn representations that can be reused across tasks. At the end of the training procedure, the decoder is discarded, and the encoder GNN is used to generate embeddings.

We summarize the contributions of this chapter as follows:

- We consider the under-studied problem of learning GNN models generating node representations that can be used to perform multiple tasks. In this regard, we design a meta-learning strategy for training GNN models with such capabilities.

- To the best of our knowledge, we are the first to propose a GNN model generating a *single* set of node embeddings that can be used to perform the three most common graph-related tasks (i.e., graph classification, node classification, and link prediction). In particular, the generated embeddings lead to comparable or even higher performance with respect to separate end-to-end trained single-task models.

- We show that the episodic training strategy at the base of our meta-learning procedure leads to a model generating node embeddings that are more effective for downstream tasks, even in single-task settings. This unexpected finding is of interest in itself, and may provide fruitful directions for future research.

5.1 Preliminaries

Below we provide a brief introduction to multi-task learning, and meta-learning, which are at the base of the topics presented in this chapter.

5.1.1 Multi-Task Learning

Multi-Task Learning (MTL) [42] is the area of machine learning which studies how to solve multiple learning tasks in parallel. The main idea is that several tasks have commonalities between them (e.g., learning to detect dogs and learning to detect cats), and so learning to solve them concurrently should not only be feasible, but may also lead to higher performance. Identifying tasks which "combine" well together, and making sure the that tasks do not interfere with each other during the learning procedure are however still open problems. A recent survey of the modern techniques for MTL with deep learning can be found in the survey by Vandenhende et al. [203].

Multi-Head Models

In deep learning, the standard approach for MTL is to employ a *multi-head* architecture (see Figure 5.2 (a)). A multi-head model is composed of a *backbone* and multiple *heads* (one for each task). The backbone is a neural network which processes the input to extract features which should be common across tasks. The features extracted by the backbone are then used by the heads (which are also neural networks) to perform the desired tasks (each head performs one task). The whole model is then trained end-to-end to minimize a combination of the single-task losses (e.g., the sum of the losses on each task). We refer to this strategy as the *classical* training procedure for multi-task models.

5.1.2 Model-Agnostic Meta-Learning and ANIL

MAML (Model-Agnostic Meta-Learning) is an optimization-based meta-learning strategy proposed by Finn et al. [68]. Let f_θ be a deep learning model, where θ represents its parameters. Let $p(\mathcal{E})$ be a distribution over episodes[1], with an episode $\mathcal{E}_i \sim p(\mathcal{E})$ being defined as a tuple containing a *loss function* $\mathcal{L}_{\mathcal{E}_i}(\cdot)$, a *support set* $\mathcal{S}_{\mathcal{E}_i}$, and a *target set* $\mathcal{T}_{\mathcal{E}_i}$: $\mathcal{E}_i = (\mathcal{L}_{\mathcal{E}_i}(\cdot), \mathcal{S}_{\mathcal{E}_i}, \mathcal{T}_{\mathcal{E}_i})$, where support and target sets are simply sets of labelled examples. MAML's goal is to find a value of θ that can quickly, i.e., in a few steps of gradient descent, be adapted to new episodes. This is done with a nested loop optimization procedure: an *inner loop* adapts the parameters to the support set of an episode by performing some steps of gradient descent, and an *outer loop* updates the initial parameters aiming at a setting that allows fast adaptation. Formally, by defining $\theta_i'(t)$ as the parameters after t adaptation steps on the support set of episode $\mathcal{E}_i$, we can express the computations in the inner loop as

$$\theta_i'(t) = \theta_i'(t-1) - \alpha \nabla_{\theta_i'(t-1)} \mathcal{L}_{\mathcal{E}_i}(f_{\theta_i'(t-1)}, \mathcal{S}_{\mathcal{E}_i}) \tag{5.1}$$

[1] The meta-learning literature usually derives episodes from *tasks* (i.e., tuples containing a dataset and a loss function). We focus on episodes to avoid using the term *task* for both a MTL task, and a meta-learning task.

where $\theta'_i(0) = \theta$, $\mathcal{L}(f_{\theta'_i(t-1)}, \mathcal{S}_{\mathcal{E}_i})$ indicates the loss over the support set $\mathcal{S}_{\mathcal{E}_i}$ for the model $f_{\theta'_i(t-1)}$ with parameters $\theta'_i(t-1)$, and α is the learning rate. The *meta-objective* that the outer loop tries to minimize is defined as $\mathcal{L}_{meta} = \sum_{\mathcal{E}_i \sim p(\mathcal{E})} \mathcal{L}_{\mathcal{E}_i}(f_{\theta'_i(t)}, \mathcal{T}_{\mathcal{E}_i})$, which leads to the following parameter update[2] performed in the outer loop:

$$\theta = \theta - \beta \nabla_\theta \mathcal{L}_{meta} = \theta - \beta \nabla_\theta \sum_{\mathcal{E}_i \sim p(\mathcal{E})} \mathcal{L}_{\mathcal{E}_i}(f_{\theta'_i(t)}, \mathcal{T}_{\mathcal{E}_i}). \tag{5.2}$$

Raghu et al. [178] showed that feature reuse is the dominant factor in MAML: in the adaptation loop, only the last layer(s) in the network are updated, while the first layer(s) remain almost unchanged. The authors then propose ANIL (Almost No Inner Loop) where they split the parameters in two sets: one that is used for adaptation in the inner loop, and one that is only updated in the outer loop. This simplification leads to computational improvements while maintaining performance.

5.2 SAME: Single-Task Adaptation for Multi-Task Embeddings

We design a meta-learning approach targeted towards representation learning, by building on three insights:

(i) **optimization-based meta-learning is implicitly learning robust representations.** The findings by Raghu et al. [178] suggest that, in a model trained with MAML, the first layers learn features that are reusable across episodes, while the last layers are set up for fast adaptation. MAML is then *implicitly* learning a model with two components: an *encoder* (the first layers), focusing on learning reusable representations that generalize across episodes, and a *decoder* (the last layers) that can be quickly adapted for different episodes.

(ii) **meta-learning episodes can be designed to encourage generalization.** By designing support and target sets to mimic the training and validation sets of a classical training procedure, then the meta-learning procedure is effectively optimizing for generalization.

(iii) **meta-learning can learn to quickly adapt to multiple tasks *singularly*, without having to learn to solve multiple tasks *concurrently*.** The meta-learning procedure can be designed so that, for each considered task, the inner loop adapts the parameters to a task-specific support set, and tests the adaptation on a task-specific target set. The outer loop then updates the parameters to allow this fast *multiple single-task adaptation*.

Based on ***(ii)*** and ***(iii)***, we design the meta-learning procedure such that the inner loop adapts to multiple tasks *singularly*, each time with the goal of *single-task generalization*. Using an encoder-

[2]We limit ourselves to one step of gradient descent for clarity, but any optimization strategy could be used.

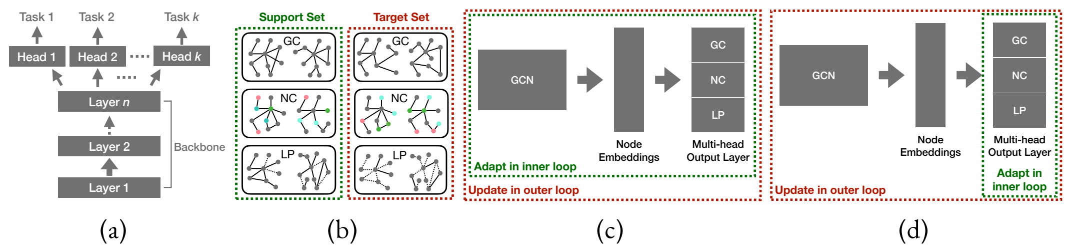


Figure 5.2: **Main ingredients of our meta-learning procedure SAME.** (a) Multi-head architecture. (b) Schematic representation of a *multi-task episode*. For each task, support and target set are designed to be as the training and validation sets for single-task training. (c-d) Overview of the parameter updates in SAME's meta-learning procedure. In the inner loop, the model is adapted *separately* to each task in the support set; the outer loop then tests the performance of each adaptation on the corresponding task in the target set, and updates the initial parameters of the network to allow it to rapidly be adapted to each task, by minimizing the meta-objective. In iSAME (c) all parameters are adapted in the inner loop. In eSAME (d) only the task-specific output layers are adapted in the inner loop. Both in iSAME and eSAME, after training the model, only the backbone GCN is kept, and used to generate embeddings.

decoder architecture, ***(i)*** suggests that this procedure leads to an encoder that learns features reusable across episodes. As, in each episode, the learner is adapting to multiple tasks, the encoder is learning features that generalize across multiple tasks. After training with our meta-learning strategy, the decoder is discarded, and only the encoder is kept and used to generate representations. Contrary to many applications of meta-learning, there is no adaptation performed at test time, as meta-learning is used only for training the model from which an encoder is extracted.

In the rest of this section, we formally present our meta-learning procedure for training multi-task graph representation learning models. There are three aspects that need to be defined: **(1) Episode Design:** how is an episode composed, **(2) Model Architecture Design:** what is the architecture of our model, **(3) Meta-Training Design:** how, and which, parameters are adapted/updated.

5.2.1 Episode Design

In our case, an episode becomes a *multi-task episode* (Figure 5.2 (b)). To formally introduce the concept, let us consider the case where the tasks are graph classification (GC), node classification

(NC), and link prediction (LP). We define a *multi-task episode* $\mathcal{E}_i^{(m)} \sim p(\mathcal{E}^{(m)})$ as a tuple $\mathcal{E}_i^{(m)} = (\mathcal{L}_{\mathcal{E}_i}^{(m)}, \mathcal{S}_{\mathcal{E}_i}^{(m)}, \mathcal{T}_{\mathcal{E}_i}^{(m)})$ where

$$\mathcal{L}_{\mathcal{E}_i}^{(m)} = \{\mathcal{L}_{\mathcal{E}_i}^{(\mathrm{GC})}, \mathcal{L}_{\mathcal{E}_i}^{(\mathrm{NC})}, \mathcal{L}_{\mathcal{E}_i}^{(\mathrm{LP})}\}, \tag{5.3}$$

$$\mathcal{S}_{\mathcal{E}_i}^{(m)} = \{\mathcal{S}_{\mathcal{E}_i}^{(\mathrm{GC})}, \mathcal{S}_{\mathcal{E}_i}^{(\mathrm{NC})}, \mathcal{S}_{\mathcal{E}_i}^{(\mathrm{LP})}\}, \tag{5.4}$$

$$\mathcal{T}_{\mathcal{E}_i}^{(m)} = \{\mathcal{T}_{\mathcal{E}_i}^{(\mathrm{GC})}, \mathcal{T}_{\mathcal{E}_i}^{(\mathrm{NC})}, \mathcal{T}_{\mathcal{E}_i}^{(\mathrm{LP})}\}. \tag{5.5}$$

The meta-objective $\mathcal{L}_{meta}^{(m)}$ of our method is then defined as:

$$\mathcal{L}_{meta}^{(m)} = \sum_{\mathcal{E}_i^{(m)} \sim p(\mathcal{E}^{(m)})} \lambda^{(GC)} \mathcal{L}_{\mathcal{E}_i}^{(\mathrm{GC})} + \lambda^{(NC)} \mathcal{L}_{\mathcal{E}_i}^{(\mathrm{NC})} + \lambda^{(LP)} \mathcal{L}_{\mathcal{E}_i}^{(\mathrm{LP})}. \tag{5.6}$$

where $\lambda^{(\cdot)}$ are balancing coefficients.

Support and target sets are set up to resemble training and validation sets. This way the outer loop's objective becomes to *maximize the performance on a validation set, given a training set*, hence encouraging generalization. In more detail, given a batch of graphs, we divide it in equally sized splits (one per task), and create support and target sets as follows:

Graph Classification: $\mathcal{S}_{\mathcal{E}_i}^{(\mathrm{GC})}$ and $\mathcal{T}_{\mathcal{E}_i}^{(\mathrm{GC})}$ contain labeled graphs, obtained with a random split.

Node Classification: $\mathcal{S}_{\mathcal{E}_i}^{(\mathrm{NC})}$ and $\mathcal{T}_{\mathcal{E}_i}^{(\mathrm{NC})}$ are composed of the same graphs, with different labelled nodes. We mimic the common semi-supervised setting [121] where feature vectors are available for all nodes, and only a small subset of nodes is labelled.

Link Prediction: $\mathcal{S}_{\mathcal{E}_i}^{(\mathrm{LP})}$ and $\mathcal{T}_{\mathcal{E}_i}^{(\mathrm{LP})}$ are composed of the same graphs, with different query edges. In every graph we randomly remove some edges, used as positive examples together with non-removed edges, and randomly sample pairs of non-adjacent nodes as negative examples.

Notice how we only need labels for *one* task for each graph. The full algorithm for the creation of *multi-task episodes* is provided below.

Episode Design Algorithm

Algorithm 2 contains the procedure for the creation of the episodes for our meta-learning procedures. The algorithm takes as input a batch of graphs (with graph labels, node labels, and node features) and the loss function balancing weights, and outputs a *multi-task episode*. We assume that each graph has a set of attributes that can be accessed with a *dot-notation* (like in most object-oriented programming languages).

Notice how the episodes are created so that **only one task is performed on each graph** (which implies that we only need labels for one task for each graph). This is important as in the inner loop of our meta-learning procedure, the learner adapts and tests the adapted parameters on one task at a time. The outer loop then updates the parameters, optimizing for a representation that leads to fast *single-task adaptation*. This procedure bypasses the problem of learning parameters that *directly* solve multiple tasks, which can be very challenging.

Another important aspect to notice is that the support and target sets are designed as if they were the training and validation splits for training a single-task model with the classical procedure. This way the meta-objective becomes to train a model that can generalize well.

5.2.2 Model Architecture Design

We use an encoder-decoder model with a multi-head architecture. The *backbone* (which represents the encoder) is composed of 3 GCN [121] layers with ReLU non-linearities and residual connections [92]. The decoder is composed of three *heads*. The node classification head is a single layer neural network with a *Softmax* activation that is shared across nodes and maps node embeddings to class predictions. In the graph classification head, first a single layer neural network (shared across nodes) performs a linear transformation (followed by a ReLU activation) of the node embeddings. The transformed node embeddings are then averaged and a final single layer neural network with *Softmax* activation outputs the class predictions. The link prediction head is composed of a single layer neural network with ReLU non-linearity that transforms node embeddings, and a single layer neural network that given concatenation of two embeddings outputs the probability of a link between them. We remark that after training the full model with the proposed meta-learning procedure, only the encoder is kept, and is used to generate node embeddings which can be fed to any machine learning model for downstream tasks.

5.2.3 Meta-Training Design

We first present the meta-learning training procedure, and successively describe which parameters are adapted/updated in the inner and outer loops.

Meta-Learning Training Procedure. The meta-learning procedure is designed such that the inner loop adaptation involves a *single task* at a time. Only the parameter update performed to minimize the meta-objective involves multiple tasks, but, crucially, it does not aim at a setting of parameters that can solve, or quickly adapt to, multiple tasks *concurrently*, but to a setting allowing **multiple fast single-task adaptation**.

The pseudocode of our procedure is in Algorithm 3. `init` is a method that initializes the weights of the GNN. `ADAPT` performs a few steps of gradient descent on a *task-specific* loss func-

Algorithm 2 Episode Design Algorithm

```
**Input:** Batch of $n$ graphs $\mathcal{B} = \{\mathcal{G}_1, .., \mathcal{G}_n\}$;   Loss weights $\lambda^{(GC)}, \lambda^{(NC)}, \lambda^{(LP)} \in [0, 1]$
**Output:** Episode $\mathcal{E}_i = (\mathcal{L}^{(m)}_{\mathcal{E}_i}, \mathcal{S}^{(m)}_{\mathcal{E}_i}, \mathcal{T}^{(m)}_{\mathcal{E}_i})$
$\mathcal{B}^{(GC)}, \mathcal{B}^{(NC)}, \mathcal{B}^{(LP)} \leftarrow$ equally divide the graphs in $\mathcal{B}$ in three sets

{Graph Classification}
$\mathcal{S}^{(\mathrm{GC})}_{\mathcal{E}_i}, \mathcal{T}^{(\mathrm{GC})}_{\mathcal{E}_i} \leftarrow$ randomly divide $\mathcal{B}^{(GC)}$ with a 60/40 split

{Node Classification}
for $\mathcal{G}_i$ in $\mathcal{B}^{(NC)}$ do
  num_labelled_nodes $\leftarrow \mathcal{G}_i$.num_nodes $\times 0.3$
  $\mathcal{N} \leftarrow$ divide nodes per class, then iteratively randomly sample one node per class without replacement and add it to $\mathcal{N}$ until $|\mathcal{N}| =$ num_labelled_nodes
  $\mathcal{G}'_i \leftarrow$ copy($\mathcal{G}_i$)
  $\mathcal{G}_i$.labelled_nodes $\leftarrow \mathcal{N}$;   $\mathcal{G}'_i$.labelled_nodes $\leftarrow \mathcal{G}_i$.nodes $\mathcal{N}$
  $\mathcal{S}^{(NC)}_{\mathcal{E}_i}$.add($\mathcal{G}_i$);   $\mathcal{T}^{(NC)}_{\mathcal{E}_i}$.add($\mathcal{G}'_i$)
end for

{Link Prediction}
for $\mathcal{G}_i$ in $\mathcal{B}^{(LP)}$ do
  $E^{(N)}_i \leftarrow$ randomly pick negative samples (edges that are not in the graph; possibly in the same number as the number of edges in the graph)
  $E^{1,(N)}_i, E^{2,(N)}_i \leftarrow$ divide $E^{(N)}_i$ with an 80/20 split
  $E^{(P)}_i \leftarrow$ randomly remove 20% of the edges in $\mathcal{G}_i$
  $\mathcal{G}'^{(1)}_i \leftarrow \mathcal{G}_i$ removed of $E^{(P)}_i$; $\mathcal{G}'^{(2)}_i \leftarrow$ copy($\mathcal{G}'^{(1)}_i$)
  $\mathcal{G}'^{(1)}_i$.positive_edges $\leftarrow \mathcal{G}'^{(1)}_i$.edges;   $\mathcal{G}'^{(2)}_i$.positive_edges $\leftarrow E^{(P)}_i$
  $\mathcal{G}'^{(1)}_i$.negative_edges $\leftarrow E^{1,(N)}_i$;   $\mathcal{G}'^{(2)}_i$.negative_edges $\leftarrow E^{2,(N)}_i$
  $\mathcal{S}^{(LP)}_{\mathcal{E}_i}$.add($\mathcal{G}'^{(1)}_i$);   $\mathcal{T}^{(LP)}_{\mathcal{E}_i}$.add($\mathcal{G}'^{(2)}_i$)
end for

$\mathcal{S}^{(m)}_{\mathcal{E}_i} \leftarrow \{\mathcal{S}^{(\mathrm{GC})}_{\mathcal{E}_i}, \mathcal{S}^{(\mathrm{NC})}_{\mathcal{E}_i}, \mathcal{S}^{(\mathrm{LP})}_{\mathcal{E}_i}\}$;   $\mathcal{T}^{(m)}_{\mathcal{E}_i} \leftarrow \{\mathcal{T}^{(\mathrm{GC})}_{\mathcal{E}_i}, \mathcal{T}^{(\mathrm{NC})}_{\mathcal{E}_i}, \mathcal{T}^{(\mathrm{LP})}_{\mathcal{E}_i}\}$
$\mathcal{L}^{(\mathrm{GC})}_{\mathcal{T}_i}, \mathcal{L}^{(\mathrm{NC})}_{\mathcal{T}_i} \leftarrow$ Cross-Entropy($\cdot$); ; $\mathcal{L}^{(\mathrm{LP})}_{\mathcal{T}_i} \leftarrow$ Bin. Cross-Entropy($\cdot$)
$\mathcal{L}^{(m)}_{\mathcal{E}_i} = \lambda^{(GC)}\mathcal{L}^{(\mathrm{GC})}_{\mathcal{T}_i} + \lambda^{(NC)}\mathcal{L}^{(\mathrm{NC})}_{\mathcal{T}_i} + \lambda^{(LP)}\mathcal{L}^{(\mathrm{LP})}_{\mathcal{T}_i}$
**Return** $\mathcal{E} = (\mathcal{L}^{(m)}_{\mathcal{E}_i}, \mathcal{S}^{(m)}_{\mathcal{E}_i}, \mathcal{T}^{(m)}_{\mathcal{E}_i})$
```

tion and support set (as in eq. 5.1), TEST computes the value of the meta-objective component on a *task-specific* loss function and target set for a model with parameters adapted on that task, and UPDATE optimizes the parameters θ by minimizing the meta-objective in eq. 5.6 (which is

Algorithm 3 Proposed (meta-learning based) procedure.

```
**Input:** Model $f_\theta$; Episodes $\mathcal{E} = \{\mathcal{E}_1, .., \mathcal{E}_n\}$; Coefficients $\lambda^{(GC)}, \lambda^{(NC)}, \lambda^{(LP)}$.
init($\theta$)
for $\mathcal{E}_i$ in $\mathcal{E}$ do
  o_loss $\leftarrow 0$
  for $\tau$ in (GC, NC, LP) do
    $\theta'^{(\tau)} \leftarrow \theta$
    $\theta'^{(\tau)} \leftarrow$ ADAPT($f_\theta, \mathcal{S}^{(\tau)}_{\mathcal{E}_i}, \mathcal{L}^{(\tau)}_{\mathcal{E}_i}$)
    o_loss $\leftarrow$ o_loss $+ \lambda^{(\tau)}$TEST($f_{\theta'^{(\tau)}}, \mathcal{T}^{(\tau)}_{\mathcal{E}_i}, \mathcal{L}^{(\tau)}_{\mathcal{E}_i}$)
  end for
  $\theta \leftarrow$ UPDATE($\theta$, o_loss, $\theta'^{(GC)}, \theta'^{(NC)}, \theta'^{(LP)}$)
end for
```

contained in `o_loss` in the pseudocode). Notice the multiple heads of the decoder are never used concurrently.

Parameter Update in Inner/Outer Loop. Let us partition the parameters of our model in four sets: $\theta = [\theta_{\text{GCN}}, \theta_{\text{NC}}, \theta_{\text{GC}}, \theta_{\text{LP}}]$ representing the parameters of the backbone (θ_{GCN}), node classification head (θ_{NC}), graph classification head (θ_{GC}), and link prediction head (θ_{LP}). We name our meta-learning strategy SAME (Single-Task Adaptation for Multi-Task Embeddings), and present two variants (Figure 5.2 c-d):

***Implicit* SAME (iSAME):** all the parameters θ are used for adaptation. This strategy makes use of the *implicit* feature-reuse factor of MAML, leading to parameters θ_{GCN} that are general across *multi-task episodes*.

***Explicit* SAME (eSAME):** only the head parameters $\theta_{\text{NC}}, \theta_{\text{GC}}, \theta_{\text{LP}}$ are used for adaptation (as done by ANIL). Contrary to iSAME, this strategy *explicitly* aims at learning the parameters θ_{GCN} to be general across *multi-task episodes* by only updating them in the outer loop.

The pseudocode in Algorithm 3 is the same for both iSAME and eSAME. The difference between the methods is in which subset of the parameters θ is updated by the `ADAPT` function. In iSAME the `ADAPT` function will update the head and the backbone parameters ($\theta_{\text{GCN}}, \theta_{\text{NC}}, \theta_{\text{GC}}, \theta_{\text{LP}}$), while for eSAME only the head parameters ($\theta_{\text{NC}}, \theta_{\text{GC}}, \theta_{\text{LP}}$) will be updated.

5.2.4 Connection between SAME and other Optimization-based Meta-Learning Methods

SAME is an instantiation of optimization-based meta-learning (in particular iSAME is an instantiation of MAML [68], while eSAME of ANIL [178]) specially designed for learning multi-task representations. In particular SAME employs the following design choices:

(1) In SAME each episode is composed of multiple tasks (i.e., downstream applications).

(2) In SAME each task (both in the inner and in the outer loop) can involve only a subset of the parameters of the model.

(3) In SAME's inner loop, *separate* adaptations are performed for each task in the episode. In the outer loop, the meta-objective defines how these multiple adaptations are combined for updating the initial representations of the parameters.

After training a model with SAME, an encoder is extracted and used to generate representations of the input that can then be fed to any machine learning model. As SAME is only used for training, no adaptation is performed at test time, and hence support and target sets are not required at test time.

5.3 Experiments

Our goal is to assess the quality of the representations learned by models trained with SAME, and to study the impact of SAME's underlying components. In more detail, we aim to answer the following questions:

Q1: *Do iSAME and eSAME lead to node embeddings that can be used to perform multiple downstream tasks with comparable (or better) performance than end-to-end single-task models?*

Q2: *Can node embeddings learned by a model trained with iSAME and eSAME be used for multiple tasks with comparable or better performance than classically trained (i.e., see Section 5.1.1) multi-task models?*

Q3: *Do iSAME and eSAME extract information that is not captured by the classical training procedure (i.e., see Section 5.1.1)?*

Q4 ***(Ablation Study)*****:** *What are the contributions of the different components of SAME's meta-learning procedure?*

Unless otherwise stated, accuracy (%) is used for NC and GC, while ROC AUC (%) is used for LP. (As a reminder, we use GC to refer to graph classification, NC for node classification, and LP for link prediction.)

5.3.1 Experimental Setting

Datasets. To perform multiple tasks, we consider datasets with graph labels, node attributes, and node labels from the TUDataset library [160]: ENZYMES [189], PROTEINS [61], DHFR and COX2 [198]. ENZYMES is a dataset of protein structures belonging to six classes. PROTEINS is a dataset of chemical compounds with two classes (enzyme and non-enzyme). DHFR, and COX2 are datasets of chemical inhibitors which can be active or inactive.

Experimental Setup. We perform a 10-fold cross validation, and average results across folds. To ensure a fair comparison, the same architecture is used for all training strategies. We set $\lambda^{(GC)} = \lambda^{(NC)} = \lambda^{(LP)} = 1$ as we noticed that weighting the losses did not provide significant benefits. Loss balancing techniques (e.g. *uncertainty weights* [114], and *gradnorm* [47]) were tested, both with SAME and with the classical training procedure, but they did not result effective. This is in accordance with recent works [130, 203] which observe that, when appropriate tuning is done, no method is significantly better than minimizing the sum of the task losses. As an example, a GNN model trained for GC and LP with the classical procedure and using GradNorm, achieves results on GC, NC, and LP, that are on average 0.5% higher than the same model trained without GradNorm. However, a GNN model trained for NC and LP with the classical procedure and using GradNorm achieves results that are 0.8% lower than the same model trained without GradNorm. A similar behaviour happens by applying Uncertainty Weights, and, when the improvements were positive, they would never be higher than 1.7%.

Implementation. We implement our models using PyTorch [171], PyTorch Geometric [66] and Torchmeta [57]. For all models the number and structure of the layers is as described in Section 4.2 of the paper, where we use 256-dimensional node embeddings at every layer.

At every cross-validation fold, 9 folds are used for training, and 1 for testing. Out of the 9 training folds, one is used as validation set. For each model we perform 100 iterations of hyperparameter optimization over the same search space (for shared parameters) using Ax [14].

Linear Model. The linear model trained on the embeddings produced by our proposed method is a standard linear SVM. In particular we use the implementation available in Scikit-learn [172] with default hyperparameters. For graph classification, we take the mean of the node embeddings as input. For link prediction we take the concatenation of the embeddings of two nodes. For node classification we keep the embeddings unaltered.

Deep Learning Baselines. We train the single task models for 1000 epochs, and the multi-task models for 5000 epochs, with early stopping on the validation set (for multi-task models we use the sum of the task validation losses or accuracies as metrics for early-stopping). Optimization is done using Adam [120]. For node classification and link prediction we found that normalizing the node embeddings to unit norm in between GCN layers helps performance.

Our Meta-Learning Procedure. We train the single task models for 5000 epochs, and the multi-task models for 15000 epochs, with early stopping on the validation set (for multi-task models we use the sum of the task validation losses or accuracies as metrics for early-stopping). Early stopping is very important in this case as it is the only way to check if the meta-learned model is overfitting the training data. The inner loop adaptation consists of 1 step of gradient descent. Optimization in the outer loop is done using Adam [120]. We found that normalizing the node embeddings to unit norm in between GCN layers helps performance.

Computing Infrastructure. The experiments were run on a Nvidia 1080Ti GPU, and on a CPU cluster equipped with 8 cpus 12-Core Intel Xeon Gold 5118 @2.30GHz, with 1.5Tb of RAM.

5.3.2 Results

Table 5.1: Results for a single-task model trained in a classical supervised manner, a fine-tuned model (trained on all, and fine-tuned on two tasks), and a **linear** classifier trained on node embeddings generated by a model trained with our strategies (iSAME, eSAME) in a multi-task setting.

Task			**Dataset**											
GC	NC	LP	ENZYMES			PROTEINS			DHFR			COX2		
			GC	NC	LP	GC	NC	LP	GC	NC	LP	GC	NC	LP
							Classical End-to-End Training							
✓			51.6			73.3			71.5			76.7		
	✓			87.5			72.3			97.3			96.4	
		✓			75.5			85.6			98.8			98.3
							Fine-Tuning							
✓	✓		48.3	85.3		73.6	72.0		66.4	92.4		80.0	92.3	
✓		✓	49.3		71.6	69.6		80.7	65.3		58.9	80.2		50.9
	✓	✓		87.7	73.9		80.4	81.5		80.7	56.6		87.4	52.3
							iSAME (ours)							
✓	✓		50.1	86.1		73.1	76.6		71.6	94.8		75.2	95.4	
✓		✓	50.7		83.1	73.4		85.2	71.6		99.2	77.5		98.9
	✓	✓		86.3	83.4		79.4	87.7		96.5	99.3		95.5	99.0
✓	✓	✓	50.0	86.5	82.3	71.4	76.6	87.3	71.2	95.5	99.5	75.4	95.2	99.2
							eSAME (ours)							
✓	✓		51.7	86.1		71.5	79.2		70.1	95.7		75.6	95.5	
✓		✓	51.9		80.1	71.7		85.4	70.1		99.1	77.5		98.8
	✓	✓		86.7	82.2		80.7	86.3		96.6	99.4		95.6	99.1
✓	✓	✓	51.5	86.3	81.1	71.3	79.6	86.8	70.2	95.3	99.5	77.7	95.7	98.8

Q1: We train a model with SAME, on all multi-task combinations, and use the embeddings produced by the learned encoder as the input for a **linear classifier**. We compare against models with the same task-specific architecture trained in a classical supervised manner on a single task, and with a fine-tuning baseline. The latter is a model that has been trained on all three tasks, and then fine-tuned on two specific tasks. The idea is that the initial training on all tasks should lead the model towards the extraction of features that it would otherwise not consider (by only seeing

Table 5.2: Δ_m (%) results for a classically trained multi-task model (Cl), a fine-tuned model (FT; trained on all three tasks and fine-tuned on two) and a **linear** classifier trained on the node embeddings generated by a model trained with our strategies (iSAME, eSAME) in a multi-task setting.

Task			**Model**	**Dataset**			
GC	NC	LP		ENZYMES	PROTEINS	DHFR	COX2
✓	✓		Cl	-0.1 ± 0.5	4.0 ± 1.0	-0.3 ± 0.2	0.5 ± 0.1
			FT	-4.5 ± 1.2	0.1 ± 0.5	-7.4 ± 1.4	0.1 ± 0.4
			iSAME	-2.3 ± 0.9	2.7 ± 1.5	-1.2 ± 0.4	-1.6 ± 0.2
			eSAME	-0.8 ± 0.8	3.2 ± 1.4	-1.8 ± 0.3	-1.2 ± 0.3
✓		✓	Cl	-25.3 ± 3.2	-5.3 ± 1.2	-28.3 ± 4.3	-21.4 ± 3.4
			FT	-5.1 ± 1.9	-5.4 ± 1.5	-24.5 ± 3.7	-22.6 ± 3.8
			iSAME	4.1 ± 0.5	-0.2 ± 0.9	0.2 ± 3.2	0.2 ± 0.5
			eSAME	3.2 ± 0.4	-1.2 ± 1.1	-0.7 ± 3.4	-0.8 ± 0.7
	✓	✓	Cl	7.2 ± 2.7	6.8 ± 0.9	-29.1 ± 7.7	-28.2 ± 4.5
			FT	-1.0 ± 0.3	3.1 ± 1.2	-28.9 ± 6.4	-28.3 ± 4.2
			iSAME	4.4 ± 1.1	6.1 ± 1.0	-0.1 ± 6.2	-0.6 ± 2.5
			eSAME	3.9 ± 1.3	6.1 ± 1.1	0.1 ± 6.4	-0.6 ± 2.6
✓	✓	✓	Cl	1.6 ± 1.3	2.9 ± 0.3	-18.9 ± 2.3	-16.9 ± 3.1
			iSAME	1.5 ± 1.0	2.2 ± 0.2	-0.5 ± 1.4	-0.9 ± 1.3
			eSAME	1.8 ± 0.9	2.8 ± 0.2	-1.0 ± 1.7	-0.4 ± 1.2

2 tasks). The fine-tuning process should then allow the model to use these features to target the specific tasks of interest. Results are shown in Table 5.1. The embeddings produced by the model learned with SAME in a multi-task setting achieve performance comparable to, and frequently even better than, end-to-end single-task models. In fact, the embeddings from SAME are never outperformed by more than 3%, and in 50% of the cases actually achieve higher performance. Moreover, the fine-tuning baseline is almost always outperformed by both single-task models, and our proposed methods. These results confirm that meta-learning is a powerful solution for multi-task representation learning on graphs.

Q2: We train the same multi-task model, both in the classical supervised manner (see Section 5.1.1), and with our proposed approaches, on all multi-task combinations. For our approaches, a **linear classifier** is then trained on top of the node embeddings produced by the learned encoder. We further consider the fine-tuning baseline introduced in **Q1**. The multi-task performance (Δ_m) metric [149] is used, defined as the average per-task drop with respect to the single-task baseline: $\Delta_m = \frac{1}{T}\sum_{i=1}^{T}\left(M_{(m,i)} - M_{(b,i)}\right)/M_{(b,i)}$, where $M_{(m,i)}$ is the value of the i-th task's metric for the multi-task model, and $M_{(b,i)}$ is the value for the baseline. Results are shown in Table 5.2. Multi-task models usually achieve lower performance than specialized single-task ones. Moreover, **linear** classifiers trained on the embeddings generated by a model trained with SAME are not only comparable, but in many cases significantly superior to classically trained multi-task models. In fact, a multi-task model trained in a classical manner is highly sensible to the tasks that are being learned (e.g. GC and LP negatively interfere with each other in every dataset), while our methods are much less sensible. For instance, the former has a worst-case average drop in performance of 29%, while our method has a worst-case average drop of less than 3%. Finally, the fine-tuning baseline generally performs worse than classically trained models, confirming that transferring knowledge in multi-task settings is not easy.

Q3: We train a multi-task model, and then train a new simple network (with the same architecture as the heads described in Section 5.2.2), which is refer to as *classifier*, on the embeddings generated by the multi-task model to perform a task that was not seen during training. We compare the performance of the classifier on the embeddings generated by a model trained in a classical manner, and with SAME. Intuitively, this tests gives us a way to analyse if the embeddings generated by a model trained with SAME contain "more information" than embeddings generated by a model trained in a classical manner. Results on the ENZYMES dataset are shown in Figure 5.3. Interestingly, the embeddings generated by a model trained with SAME lead to at least 10% higher performance. We observe an analogous trend on the other datasets, as shown in Table 5.3.

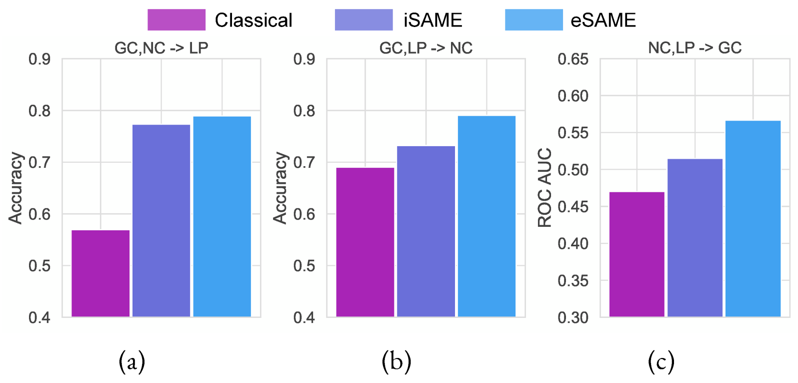


Figure 5.3: Results for model, trained on the embeddings generated by a multi-task model, performing a task that was not seen during training. "x, y->z" indicates that x, y are the tasks used for training the multi-task model, and z is the new task.

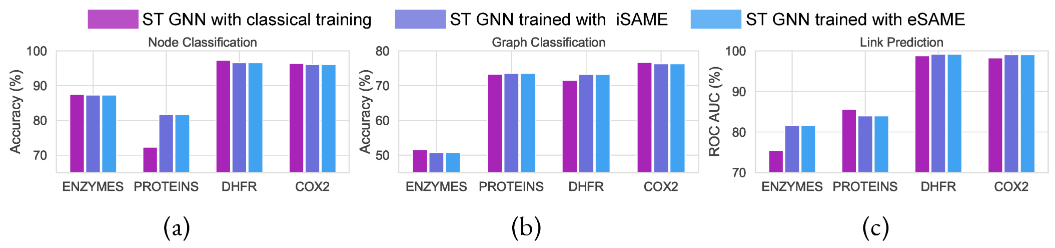


Figure 5.4: Results for a Single-Task GNN model (ST GNN) trained with the classical procedure, and a **linear** classifier trained on the embeddings generated by a model trained with an ablated "single-task" version of SAME.

Q4 *(Ablation Study)*: SAME's meta-learning procedure has two main ingredients:

Table 5.3: Results of a neural network trained on the embeddings generated by a multi-task model, to perform a task that was not seen during training by the multi-task model. “x,y ->z” indicates that the multi-task model was trained on tasks x and y, and the neural network is performing task z.

Task	**Model**	**Dataset**			
		ENZYMES	PROTEINS	DHFR	COX2
GC,NC ->LP	Cl	56.9 ± 3.9	54.4 ± 1.4	61.2 ± 2.2	59.8 ± 0.4
	iSAME	77.3 ± 4.5	88.5 ± 1.8	99.8 ± 1.8	97.1 ± 2.0
	eSAME	78.9 ± 2.8	89.1 ± 1.5	99.7 ± 2.2	95.8 ± 3.3
GC,LP ->NC	Cl	69.1 ± 1.2	57.3 ± 1.6	58.3 ± 9.3	68.9 ± 10.7
	iSAME	73.3 ± 2.1	59.2 ± 2.5	77.6 ± 1.6	78.1 ± 4.6
	eSAME	79.1 ± 1.7	64.7 ± 3.0	76.1 ± 2.7	76.9 ± 3.3
NC,LP ->GC	Cl	47.1 ± 2.4	75.3 ± 1.5	77.5 ± 3.1	79.9 ± 3.4
	iSAME	48.5 ± 5.5	76.1 ± 2.3	76.1 ± 3.7	79.7 ± 5.1
	eSAME	56.6 ± 3.1	74.6 ± 2.7	77.1 ± 3.6	79.3 ± 6.2

(1) the design of support and target sets, to encourage generalization by mimicking training and validation sets (see Section 5.2.1).

(2) the separate multiple single-task adaptations performed in the inner loop, which relieve the model from having to learn to solve all the tasks *concurrently* (Section 5.2.3).

To better understand the importance and contribution of each component we perform two experiments, for which results are presented below.

First, we isolate the contribution of ***(1)*** by applying iSAME and eSAME in a *single-task* setting (i.e., the same *single* task is performed in both inner and outer loops), with episodes following the generalization-encouraging design proposed in Section 5.2.1. Notice that this is like applying the original MAML and ANIL training procedures with our design of support and target sets. In this experiment, for every task, we train a **linear classifier** on top of the embeddings produced by a model trained with “single-task” iSAME and eSAME, and compare against a network with the same architecture trained in a classical end-to-end manner. Results are shown in Figure 5.4. For all three tasks, a **linear** classifier on the embeddings produced by a model trained with our methods achieves comparable, if not superior, performance to an end-to-end model. In fact, the linear classifier is never outperformed by more than 2%, and it can outperform the classical end-to-end model by up to 12%. We believe this unexpected outcome is particularly interesting, and hints that episodic training procedures can be used to learn better representations.

Second, we investigate the benefits of ***(2)*** by removing the separate multiple single-task adaptations of SAME and performing *all* tasks (i.e., GC, NC, and LP) concurrently both in the inner and outer loop. This leads to a simple *concurrent multi-task version* of the conventional training procedure of MAML and ANIL, but with our support and target set design. For this experiment, we evaluate the ablated versions of SAME on the same procedure of **Q2** and **Q3**, and compare

Table 5.4: Results for a single-task model trained in a classical supervised manner (Cl), and a **linear** classifier trained on the embeddings generated by a model trained with an ablated "single-task" version our meta-learning strategies ((a)iSAME, (a)eSAME).

Task	**Model**	**Dataset**			
		ENZYMES	PROTEINS	DHFR	COX2
NC	Cl	87.5 ± 1.9	72.3 ± 4.4	97.3 ± 0.2	96.4 ± 0.3
	(a)iSAME	87.3 ± 0.8	81.8 ± 1.6	96.6 ± 0.3	96.1 ± 0.4
	(a)eSAME	87.8 ± 0.7	82.4 ± 1.6	96.8 ± 0.2	96.5 ± 0.6
GC	Cl	51.6 ± 4.2	73.3 ± 3.6	71.5 ± 2.3	76.7 ± 4.7
	(a)iSAME	50.8 ± 2.9	73.5 ± 1.2	73.2 ± 3.2	76.3 ± 4.6
	(a)eSAME	52.1 ± 5.0	72.6 ± 1.6	71.6 ± 2.4	75.6 ± 4.1
LP	Cl	75.5 ± 3.0	85.6 ± 0.8	98.8 ± 0.7	98.3 ± 0.8
	(a)iSAME	81.7 ± 1.7	84.0 ± 1.1	99.2 ± 0.4	99.1 ± 0.5
	(a)eSAME	80.1 ± 3.4	84.1 ± 0.9	99.2 ± 0.3	99.2 ± 0.7

Table 5.5: Results of a neural network trained on the embeddings generated by a multi-task model, to perform a task that was not seen during training by the multi-task model. The multi-task model has been trained with an ablated version of iSAME and eSAME (which we refer to as (a)iSAME and (a)eSAME), where no single-task adaptation is performed, but a multi-task version of the traditional meta-learning procedure is applied. "x,y->z" indicates that the multi-task model was trained on tasks x and y, and the neural network is performing task z.

Task	**Model**	**Dataset**			
		ENZYMES	PROTEINS	DHFR	COX2
GC,NC ->LP	(a)iSAME	75.6 ± 3.3	88.3 ± 1.2	98.4 ± 0.9	95.1 ± 1.7
	(a)eSAME	79.4 ± 2.8	89.2 ± 1.6	97.4 ± 0.7	95.3 ± 1.4
GC,LP ->NC	(a)iSAME	71.8 ± 2.5	59.7 ± 3.2	76.4 ± 2.3	79.1 ± 2.3
	(a)eSAME	79.5 ± 1.6	63.8 ± 2.1	77.0 ± 2.1	78.9 ± 2.1
NC,LP ->GC	(a)iSAME	42.3 ± 5.5	75.8 ± 2.6	76.9 ± 4.4	78.3 ± 7.5
	(a)eSAME	53.6 ± 3.5	75.6 ± 2.1	77.3 ± 2.8	77.7 ± 5.2

against the results of iSAME and eSAME. The results from the ablated version are not significantly different from those of *non-ablated* iSAME and eSAME as shown in Table 5.4, Table 5.5, and Table 5.6.

From these experiments we draw two conclusions. ***(i)*** The *generalization-encouraging* design of support and target sets is what allows SAME to reach performance on multiple tasks that are comparable to specialised single-task models trained in a classical manner. ***(ii)*** The separate *multiple single-task adaptations* that are performed in the inner loop of iSAME and eSAME allow the models to reach the same performance of a version of SAME where all tasks are performed concurrently on all graphs, hence increasing the learning efficiency by not requiring labels for each task on every graph.

Table 5.6: Δ_m (%) results for a **linear** classifier trained on the node embeddings generated by a model trained with an ablated version of iSAME and eSAME (which we refer to as (a)iSAME and (a)eSAME), where no single-task adaptation is performed, but a multi-task version of the traditional meta-learning procedure is applied.

Task			**Model**	**Dataset**			
GC	NC	LP		ENZYMES	PROTEINS	DHFR	COX2
✓	✓		(a)iSAME	-3.2 ± 0.4	2.8 ± 1.3	-0.4 ± 0.3	-0.5 ± 0.2
			(a)eSAME	0.5 ± 1.1	1.7 ± 0.5	-0.5 ± 1.2	-1.5 ± 0.5
✓		✓	(a)iSAME	1.5 ± 2.9	-0.9 ± 1.1	0.1 ± 4.1	-0.2 ± 3.1
			(a)eSAME	3.0 ± 1.7	-2.5 ± 1.6	-0.4 ± 3.5	-0.4 ± 4.2
	✓	✓	(a)iSAME	4.3 ± 2.8	4.5 ± 1.1	-0.2 ± 6.2	-0.6 ± 4.2
			(a)eSAME	3.1 ± 0.4	5.0 ± 1.1	-0.1 ± 5.9	-0.5 ± 4.1
✓	✓	✓	(a)iSAME	1.0 ± 1.2	2.7 ± 0.3	-1.2 ± 2.4	-1.1 ± 2.9
			(a)eSAME	0.4 ± 1.2	1.1 ± 0.3	-1.3 ± 1.1	-0.9 ± 1.1

5.4 Related Work

MTL, and meta-learning are very active areas of research. We highlight works that are at the intersection of GNNs and these subjects, and point the interested reader to comprehensive reviews of each field. To the best of our knowledge there is no work using meta-learning to train a model for graph MTL, or proposing a GNN performing graph classification, node classification, and link prediction *concurrently*.

Multi-Task Learning and Graph Neural Networks. Works at the intersection of MTL and GNNs have mostly focused on multi-head architectures. These models are composed of a series of GNN layers followed by multiple heads (i.e., independent neural network layers) that perform the desired downstream tasks. In this category, Montanari et al. [159] propose a model for the prediction of physico-chemical properties. Holtz et al. [96] and Xie et al. [215] propose multi-task models for concurrently performing node and graph classification. Finally, Avelar et al. [9] introduce a multi-head GNN for learning multiple graph centrality measures, and Li and Ji [135] propose a MTL method for the extraction of multiple biomedical relations. Other related work includes Haonan et al. [89] which introduces a model that can be trained for several tasks singularly, hence, unlike the previously mentioned approaches and our proposed method, it can not perform multiple tasks concurrently.

Meta-Learning and Graph Neural Networks. Meta-Learning consists in *learning to learn*. Many methods have been proposed (see the review by Hospedales et al. [97]), specially in the area of *few-shot learning*. Garcia and Bruna [77] frame the few-shot learning problem with a partially observed graphical model and use GNNs as an inference algorithm. Liu et al. [137] use

GNNs to propagate messages between class prototypes and improve existing few-shot learning methods, while Suo et al. [197] use GNNs to introduce domain-knowledge in the form of graphs. There are also several works that use meta-learning to train GNNs in few-shot learning scenarios with applications to node classification [221, 231], edge labelling [119], link prediction [4, 27], and graph regression [165]. Finally, other combinations of meta-learning and GNNs involve adversarial attacks [236] and active learning [148].

5.5 Conclusions

This chapter introduces the use of meta-learning as a training strategy for graph representation learning in multi-task settings. We find that our method leads to models that produce "more general" node embeddings. In fact, our results show that the embeddings produced by a model trained with our technique can be used to perform graph classification, node classification, and link prediction, with comparable or better performance than separate single-task end-to-end supervised models. Furthermore, we find that the embeddings generated by a model trained with our procedure lead to higher performance on downstream tasks that were not seen during training, and that the episodic training procedure leads to better embeddings even in the single-task setting.

6 Conclusion

The aim of this thesis is to provide *practical* solutions that can help improving the performance of GNNs. In particular, the focus has been on three questions, introduced in Chapter 1, which cover topics regarding the impact of global structural information in GNNs applications, the difficulty in generalizing from small to large graphs, and the challenge of generating "general" embeddings that can be used effectively for multiple tasks. We summarize our contributions to the considered questions as follows.

1. *Is global structural information important for GNNs applications on graphs, and how can it be exploited by GNNs?* In Chapter 3 we analyzed the impact that information about the *global* structure of the graph can have on downstream node-level and graph-level tasks, and identified a strategy to "inject" some global information through a regularization loss.

2. *Can we improve the generalization properties of GNNs from small to large graphs?* In Chapter 4 we studied the problem of size-generalization, and proposed a technique for increasing the capabilities of GNNs in generalizing from small to large graphs.

3. *Can we train GNNs to produce embeddings that are effective for multiple tasks?* In Chapter 5 we observed how GNNs produce embeddings that are specific to the training task, but that are hardly reusable for other tasks, and we proposed a meta-learning based training strategy to obtain *multi-task* embeddings.

6.1 Future Work

There are many directions to explore in order to expand the knowledge on the problems considered in this thesis, as more research needs to be done before they can be considered fully solved. Below are some interesting directions for future work.

With regards to the topics on global structural information studied in Chapter 3, it can be interesting to see if more recent deep learning graph models based on the Transformer [204] architecture (see the survey by Min et al. [156]) are better equipped for capturing and processing global information.

Understanding the generalization capabilities of neural networks in general is still a very open research topic, so more theoretical work is needed for all models, not just GNNs. With regards to size generalization in GNNs, explored in Chapter 4, it could be interesting to explore how our method performs in scenarios where this kind of generalization is crucial, like combinatorial optimization (e.g., see Prates et al. [177]) for example. Furthermore, it would be interesting to see the impact of domain-specific coarsening techniques when applying our method.

On the multi-task embedding side, explored in Chapter 5, I hope the research in this thesis can encourage the community to explore this overlooked problem. I believe it would be interesting to propose an unsupervised version of SAME which could combine graph-level (e.g., see Ju et al. [112]), and node-level (e.g., see Hassani and Khasahmadi [90]) unsupervised training techniques, for more general embeddings. Furthermore, recent advancements in meta-learning (e.g., the work from Flennerhag et al. [69]) could be used to improve our method.

On a more general note, I believe the field of geometric deep learning has reached a good level of maturity, and these models are now ready to make important practical achievements. I am particularly excited to see the progress in the biological domain (e.g., [88, 108]). Finally, I believe the use of GNNs as an inductive bias for structurality and compositionality [122] will play an important role in deep learning applications that require reasoning capabilities.

Working on graph representation learning and GNNs has thought me a lot, and I am excited to see how the field evolves. I hope this thesis can be of use to practitioners, who are looking for effective ways to improve their models, and encourage researchers to continue pushing the limits of graph machine learning. I certainly aim at doing the same.

Acronyms

1-WL	Weisfeiler-Leman Algorithm
AI	Artificial Intelligence
CKA	Centered Kernel Alignment
CMD	Central Moment Discrepancy
CPU	Central Processing Unit
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GIN	Graph Isomorphism Network
GNN	Graph Neural Network
GPU	Graphics Processing Unit
GRL	Graph Representation Learning
MCC	Matthews Correlation Coefficient
MLP	Multi-Layer Perceptron
MMD	Maximum Mean Discrepancy
MPNN	Message-Passing Neural Network
MSE	Mean Squared Error
MTL	Multi-Task Learning
RWR	Random Walk with Restart
RWRReg	Random Walk with Restart Regularization
SAME	Single-Task Adaptation for Multi-Task Embeddings

Bibliography

1. R. Abboud, A. A. Ceylan, M. Grohe, and T. Lukasiewicz. “The Surprising Power of Graph Neural Networks with Random Node Initialization”. In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*. Ed. by Z.-H. Zhou. International Joint Conferences on Artificial Intelligence Organization, 2021, pp. 2112–2118 (cit. on p. 17).

2. S. Abu-El-Haija, A. Kapoor, B. Perozzi, and J. Lee. “N-GCN: Multi-scale Graph Convolution for Semi-supervised Node Classification”. In: *UAI*. 2018 (cit. on p. 37).

3. S. Abu-El-Haija, B. Perozzi, A. Kapoor, H. Harutyunyan, N. Alipourfard, K. Lerman, G. V. Steeg, and A. Galstyan. “MixHop: Higher-Order Graph Convolution Architectures via Sparsified Neighborhood Mixing”. In: *ICML*. 2019 (cit. on p. 37).

4. F. Alet, E. Weng, T. L. Perez, and L. Kaelbling. “Neural Relational Inference with Fast Modular Meta-learning”. In: *NeurIPS*. 2019 (cit. on p. 74).

5. U. Alon and E. Yahav. “On the Bottleneck of Graph Neural Networks and its Practical Implications”. In: *International Conference on Learning Representations*. 2020 (cit. on pp. 18, 19).

6. R. Andersen, F. Chung, and K. Lang. “Local Graph Partitioning using PageRank Vectors”. *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS’06)*, 2006. DOI: `10.1109/focs.2006.44`. URL: `http://dx.doi.org/10.1109/FOCS.2006.44` (cit. on p. 38).

7. M. Arjovsky, L. Bottou, I. Gulrajani, and D. Lopez-Paz. “Invariant risk minimization”. *arXiv preprint arXiv:1907.02893*, 2019 (cit. on pp. 48, 51, 55).

8. V. Arvind, J. Köbler, G. Rattan, and O. Verbitsky. “On the power of color refinement”. In: *International Symposium on Fundamentals of Computation Theory*. Springer. 2015, pp. 339–350 (cit. on p. 17).

9. P. Avelar, H. Lemos, M. Prates, and L. Lamb. “Multitask Learning on Graph Neural Networks: Learning Multiple Graph Centrality Measures with a Unified Network”. In: *ICANN Workshop and Special Sessions*. 2019 (cit. on p. 73).

10. L. Babai. “Graph Isomorphism in Quasipolynomial Time [Extended Abstract]”. In: *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*. STOC ’16. Association for Computing Machinery, Cambridge, MA, USA, 2016, 684â€“697. ISBN: 9781450341325. DOI: 10.1145/2897518.2897542. URL: https://doi.org/10.1145/2897518.2897542 (cit. on p. 8).

11. L. Babai and L. Kucera. “Canonical labelling of graphs in linear average time”. In: *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*. 1979, pp. 39–46. DOI: 10.1109/SFCS.1979.8 (cit. on p. 8).

12. D. Bahdanau, K. H. Cho, and Y. Bengio. “Neural machine translation by jointly learning to align and translate”. In: *3rd International Conference on Learning Representations, ICLR 2015*. 2015 (cit. on pp. 1, 15).

13. B. Bahmani, A. Chowdhury, and A. Goel. “Fast incremental and personalized PageRank”. *Proceedings of the VLDB Endowment* 4:3, 2010, pp. 173–184. ISSN: 2150-8097. DOI: 10.14778/1929861.1929864. URL: http://dx.doi.org/10.14778/1929861.1929864 (cit. on p. 38).

14. E. Bakshy, L. Dworkin, et al. “AE: A domain-agnostic platform for adaptive experimentation”. In: *NeurIPS MLSys Workshop*. 2018 (cit. on p. 67).

15. P. K. Banerjee, K. Karhadkar, Y. G. Wang, U. Alon, and G. Montúfar. “Oversquashing in GNNs through the lens of information contraction and graph expansion”. In: *2022 58th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. 2022, pp. 1–8. DOI: 10.1109/Allerton49937.2022.9929363 (cit. on p. 39).

16. P. Battaglia, R. Pascanu, M. Lai, D. Jimenez Rezende, et al. “Interaction networks for learning about objects, relations and physics”. *Advances in neural information processing systems* 29, 2016 (cit. on p. 54).

17. M. Belkin and P. Niyogi. “Laplacian eigenmaps and spectral techniques for embedding and clustering”. *Advances in neural information processing systems* 14, 2001 (cit. on p. 12).

18. I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio. “Neural combinatorial optimization with reinforcement learning”. *arXiv preprint arXiv:1611.09940*, 2016 (cit. on p. 54).

19. S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. W. Vaughan. “A theory of learning from different domains”. *Machine learning* 79:1, 2010, pp. 151–175 (cit. on p. 55).

20. Y. Bengio, A. Courville, and P. Vincent. “Representation learning: A review and new perspectives”. *IEEE transactions on pattern analysis and machine intelligence* 35:8, 2013, pp. 1798–1828 (cit. on p. 10).

21. Y. Bengio, A. Lodi, and A. Prouvost. "Machine learning for combinatorial optimization: A methodological tour d'horizon". *European Journal of Operational Research* 290:2, 2021, pp. 405–421. DOI: 10.1016/j.ejor.2020.07.063 (cit. on p. 41).

22. B. Bevilacqua, Y. Zhou, and B. Ribeiro. "Size-Invariant Graph Representations for Graph Classification Extrapolations". In: *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*. Ed. by M. Meila and T. Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 837–851. URL: http://proceedings.mlr.press/v139/bevilacqua21a.html (cit. on pp. 41, 42, 45, 48, 49, 51–55).

23. C. Bodnar, F. Di Giovanni, B. P. Chamberlain, P. Liò, and M. M. Bronstein. "Neural sheaf diffusion: A topological perspective on heterophily and oversmoothing in gnns". *arXiv preprint arXiv:2202.04579*, 2022 (cit. on p. 18).

24. C. Bodnar, F. Frasca, N. Otter, Y. Wang, P. Lio, G. F. Montufar, and M. Bronstein. "Weisfeiler and Lehman go cellular: CW networks". *Advances in Neural Information Processing Systems* 34, 2021, pp. 2625–2640 (cit. on p. 18).

25. C. Bodnar, F. Frasca, Y. Wang, N. Otter, G. F. Montufar, P. Lio, and M. Bronstein. "Weisfeiler and lehman go topological: Message passing simplicial networks". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 1026–1037 (cit. on p. 18).

26. A. Bojchevski, J. Klicpera, B. Perozzi, A. Kapoor, M. J. Blais, B. Rozemberczki, M. Lukasik, and S. Gunnemann. "Scaling Graph Neural Networks with Approximate PageRank". *ACM SIGKDD*, 2020 (cit. on p. 38).

27. A. J. Bose, A. Jain, P. Molino, and W. L. Hamilton. "Meta-Graph: Few Shot Link Prediction via Meta Learning". *arXiv*, 2019 (cit. on p. 74).

28. G. Bravo-Hermsdorff and L. M. Gunderson. "A Unifying Framework for Spectrum-Preserving Graph Sparsification and Coarsening". In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, 2019 (cit. on p. 55).

29. M. Bronstein. *Do we need deep graph neural networks?* 2020. URL: https://towardsdatascience.com/do-we-need-deep-graph-neural-networks-be62d3ec5c59 (visited on 11/17/2021) (cit. on pp. 18, 19).

30. M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. "Geometric Deep Learning: Going beyond Euclidean data". *IEEE Signal Processing Magazine* 34:4, 2017, pp. 18–42. DOI: 10.1109/MSP.2017.2693418 (cit. on p. 2).

31. D. Buffelli, P. Liò, and F. Vandin. "SizeShiftReg: a Regularization Method for Improving Size-Generalization in Graph Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho. 2022. URL: https://openreview.net/forum?id=wOI0AUAq9BR (cit. on p. 4).

32. D. Buffelli and E. Tsamoura. "Scalable Theory-Driven Regularization of Scene Graph Generation Models". In: *AAAI Conference on Artificial Intelligence*. 2023 (cit. on p. 4).

33. D. Buffelli and F. Vandin. "A meta-learning approach for graph representation learning in multi-task settings". *arXiv preprint arXiv:2012.06755*, 2020 (cit. on p. 4).

34. D. Buffelli and F. Vandin. "Attention-Based Deep Learning Framework for Human Activity Recognition With User Adaptation". *IEEE Sensors Journal* 21:12, 2021, pp. 13474–13483. DOI: 10.1109/JSEN.2021.3067690 (cit. on p. 4).

35. D. Buffelli and F. Vandin. "Graph Representation Learning for Multi-Task Settings: a Meta-Learning Approach". In: *2022 International Joint Conference on Neural Networks (IJCNN)*. 2022, pp. 1–8. DOI: 10.1109/IJCNN55064.2022.9892010 (cit. on p. 4).

36. D. Buffelli and F. Vandin. "The Impact of Global Structural Information in Graph Neural Networks Applications". *Data* 7:1, 2022. ISSN: 2306-5729. DOI: 10.3390/data7010010. URL: https://www.mdpi.com/2306-5729/7/1/10 (cit. on p. 4).

37. R. Burioni and D. Cassi. "Random walks on graphs: ideas, techniques and results". *Journal of Physics A: Mathematical and General* 38:8, 2005, R45–R78. DOI: 10.1088/0305-4470/38/8/r01. URL: https://doi.org/10.1088/0305-4470/38/8/r01 (cit. on p. 9).

38. C. Cai, D. Wang, and Y. Wang. "Graph Coarsening with Neural Networks". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=uxpzitPEooJ (cit. on p. 55).

39. J.-Y. Cai, M. Fürer, and N. Immerman. "An optimal lower bound on the number of variables for graph identification". *Combinatorica* 12:4, 1992, pp. 389–410 (cit. on pp. 8, 17).

40. C. Cangea, P. Veličković, N. Jovanović, T. Kipf, and P. Liò. "Towards Sparse Hierarchical Graph Classifiers". *NeurIPS Workshop on Relational Representation Learning*, 2018 (cit. on p. 26).

41. Q. Cappart, D. Chételat, E. B. Khalil, A. Lodi, C. Morris, and P. Veličković. "Combinatorial Optimization and Reasoning with Graph Neural Networks". In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, 2021. DOI: 10.24963/ijcai.2021/595 (cit. on p. 41).

42. R. Caruana. “Multitask learning”. *Machine learning* 28:1, 1997, pp. 41–75 (cit. on p. 59).

43. I. Chami, S. Abu-El-Haija, B. Perozzi, C. Ré, and K. Murphy. “Machine learning on graphs: A model and comprehensive taxonomy”. *arXiv preprint arXiv:2005.03675*, 2020 (cit. on p. 13).

44. D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun. “Measuring and relieving the over-smoothing problem for graph neural networks from the topological view”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04. 2020, pp. 3438–3445 (cit. on p. 18).

45. J. Chen, J. Zhu, and L. Song. “Stochastic Training of Graph Convolutional Networks with Variance Reduction”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 942–950 (cit. on p. 18).

46. J. Chen, Y. Saad, and Z. Zhang. “Graph coarsening: from scientific computing to machine learning”. *SeMA Journal* 79:1, 2022, pp. 187–223. DOI: 10.1007/s40324-021-00282-x (cit. on p. 55).

47. Z. Chen, V. Badrinarayanan, C.-Y. Lee, and A. Rabinovich. “GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks”. In: *ICML*. 2018 (cit. on p. 67).

48. D. Chicco and G. Jurman. “The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation”. *BMC Genomics* 21:1, 2020. DOI: 10.1186/s12864-019-6413-7 (cit. on p. 50).

49. D.-A. Clevert, T. Unterthiner, and S. Hochreiter. “Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)”. *ICLR*, 2016 (cit. on pp. 28, 29).

50. P. Comon. *Independent component analysis*. 1992 (cit. on p. 10).

51. D. Coppersmith, P. Doyle, P. Raghavan, and M. Snir. “Random walks on weighted graphs and applications to on-line algorithms”. *Journal of the ACM (JACM)* 40:3, 1993, pp. 421–453 (cit. on p. 9).

52. G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Veličković. “Principal Neighbourhood Aggregation for Graph Nets”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 13260–13271. URL: https://proceedings.neurips.cc/paper/2020/file/99cad265a1768cc2dd013f0e740300ae-Paper.pdf (cit. on pp. 12, 42, 48, 49).

53. L. Cotta, C. Morris, and B. Ribeiro. "Reconstruction for powerful graph representations". *Advances in Neural Information Processing Systems* 34, 2021, pp. 1713–1726 (cit. on p. 18).

54. G. Csurka. "A Comprehensive Survey on Domain Adaptation for Visual Applications". In: *Domain Adaptation in Computer Vision Applications*. Ed. by G. Csurka. Springer International Publishing, Cham, 2017, pp. 1–35. ISBN: 978-3-319-58347-1. DOI: 10.1007/978-3-319-58347-1_1. URL: https://doi.org/10.1007/978-3-319-58347-1_1 (cit. on p. 55).

55. A. Deac, M. Lackenby, and P. Veličković. "Expander Graph Propagation". In: *The First Learning on Graphs Conference*. 2022. URL: https://openreview.net/forum?id=IKevTLt3rT (cit. on p. 39).

56. M. Defferrard, X. Bresson, and P. Vandergheynst. "Convolutional neural networks on graphs with fast localized spectral filtering". *Advances in neural information processing systems* 29, 2016 (cit. on p. 14).

57. T. Deleu, T. Würfl, et al. "Torchmeta: A Meta-Learning library for PyTorch". *arXiv*, 2019 (cit. on p. 67).

58. A. Derrow-Pinion, J. She, D. Wong, O. Lange, T. Hester, L. Perez, M. Nunkesser, S. Lee, X. Guo, B. Wiltshire, P. W. Battaglia, V. Gupta, A. Li, Z. Xu, A. Sanchez-Gonzalez, Y. Li, and P. Velickovic. "ETA Prediction with Graph Neural Networks in Google Maps". In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. CIKM '21. Association for Computing Machinery, Virtual Event, Queensland, Australia, 2021, pp. 3767–3776. ISBN: 9781450384469. DOI: 10.1145/3459637.3481916. URL: https://doi.org/10.1145/3459637.3481916 (cit. on p. 2).

59. F. Diehl, T. Brunner, M. Truong Le, and A. Knoll. "Towards Graph Pooling by Edge Contraction". *ICML Workshop on Learning and Reasoning with Graph-Structured Data*, 2019 (cit. on p. 26).

60. M. Ding, K. Kong, J. Chen, J. Kirchenbauer, M. Goldblum, D. Wipf, F. Huang, and T. Goldstein. "A Closer Look at Distribution Shifts and Out-of-Distribution Generalization on Graphs". In: *NeurIPS 2021 Workshop on Distribution Shifts: Connecting Methods and Applications*. 2021. URL: https://openreview.net/forum?id=XvgPGWazqRH (cit. on pp. 41, 51, 55).

61. P. D. Dobson and A. J. Doig. "Distinguishing Enzyme Structures from Non-Enzymes Without Alignments". *Journal of Molecular Biology*, 2003 (cit. on p. 66).

62. G. Dong and H. Liu. *Feature engineering for machine learning and data analytics*. CRC Press, 2018 (cit. on p. 10).

63. S. S. Du, Y. Wang, X. Zhai, S. Balakrishnan, R. R. Salakhutdinov, and A. Singh. "How Many Samples are Needed to Estimate a Convolutional Neural Network?" In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper/2018/file/03c6b06952c750899bb03d998e631860-Paper.pdf (cit. on p. 11).

64. D. Durfee, Y. Gao, G. Goranci, and R. Peng. "Fully dynamic spectral vertex sparsifiers and applications". In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*. 2019, pp. 914–925 (cit. on p. 55).

65. G. K. Dziugaite, D. M. Roy, and Z. Ghahramani. "Training generative neural networks via maximum mean discrepancy optimization". In: *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*. 2015, pp. 258–267 (cit. on p. 55).

66. M. Fey and J. E. Lenssen. "Fast Graph Representation Learning with PyTorch Geometric". In: *ICLR RLGM Workshop*. 2019 (cit. on p. 67).

67. M. Fey, J. E. Lenssen, F. Weichert, and J. Leskovec. "GNNAutoScale: Scalable and Expressive Graph Neural Networks via Historical Embeddings". In: *International Conferences on Machine Learning (ICML)*. 2021 (cit. on p. 2).

68. C. Finn, P. Abbeel, and S. Levine. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *ICML*. 2017 (cit. on pp. 58, 59, 65).

69. S. Flennerhag, Y. Schroecker, T. Zahavy, H. van Hasselt, D. Silver, and S. Singh. "Bootstrapped Meta-Learning". In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=b-ny3x071E5 (cit. on p. 76).

70. F. Fouss, A. Pirotte, J.-m. Renders, and M. Saerens. "Random-Walk Computation of Similarities between Nodes of a Graph with Application to Collaborative Recommendation". *IEEE Transactions on Knowledge and Data Engineering* 19:3, 2007, pp. 355–369. DOI: 10.1109/TKDE.2007.46 (cit. on p. 9).

71. Z.-H. Fu, K.-B. Qiu, and H. Zha. "Generalize a Small Pre-trained Model to Arbitrarily Large TSP Instances". *Proceedings of the AAAI Conference on Artificial Intelligence* 35:8, 2021, pp. 7474–7482. URL: https://ojs.aaai.org/index.php/AAAI/article/view/16916 (cit. on p. 54).

72. K. Fukushima. “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”. *Biological Cybernetics* 36, 1980, pp. 193–202 (cit. on p. 1).

73. V. Fung, J. Zhang, E. Juarez, and B. G. Sumpter. “Benchmarking graph neural networks for materials chemistry”. *npj Computational Materials* 7:1, 2021, pp. 1–8 (cit. on pp. 2, 12).

74. F. GÃ¶bel and A. Jagers. “Random walks on graphs”. *Stochastic Processes and their Applications* 2:4, 1974, pp. 311–336. ISSN: 0304-4149. DOI: `https://doi.org/10.1016/0304-4149(74)90001-5`. URL: `https://www.sciencedirect.com/science/article/pii/0304414974900015` (cit. on p. 9).

75. H. Gao, J. Pei, and H. Huang. “Conditional Random Field Enhanced Graph Convolutional Neural Networks”. *ACM SIGKDD*, 2019. DOI: `10.1145/3292500.3330888`. URL: `http://dx.doi.org/10.1145/3292500.3330888` (cit. on p. 38).

76. H. Gao and S. Ji. *Graph U-Nets*. 2019. arXiv: `1905.05178 [cs.LG]` (cit. on p. 26).

77. V. Garcia and J. Bruna. “Few-Shot Learning with Graph Neural Networks”. In: *ICLR*. 2018 (cit. on p. 73).

78. M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi. “Exact Combinatorial Optimization with Graph Convolutional Neural Networks”. In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, 2019 (cit. on p. 41).

79. J. Gasteiger, M. Shuaibi, A. Sriram, S. Günnemann, Z. Ulissi, C. L. Zitnick, and A. Das. *How Do Graph Networks Generalize to Large and Diverse Molecular Systems?* 2022. DOI: `10.48550/ARXIV.2204.02782`. URL: `https://arxiv.org/abs/2204.02782` (cit. on pp. 41, 53).

80. J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. “Neural Message Passing for Quantum Chemistry”. In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML’17. JMLR.org, Sydney, NSW, Australia, 2017, pp. 1263–1272 (cit. on p. 13).

81. I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016 (cit. on pp. 10, 11).

82. A. Grover and J. Leskovec. “node2vec: Scalable feature learning for networks”. In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 2016, pp. 855–864 (cit. on pp. 9, 12).

83. H. Guan and M. Liu. “Domain adaptation for medical image analysis: a survey”. *IEEE Transactions on Biomedical Engineering*, 2021 (cit. on p. 55).

84. C. Guo, S. Zheng, Y. Xie, and W. Hao. “A survey on spectral clustering”. In: *World Automation Congress 2012*. 2012, pp. 53–56 (cit. on p. 53).

85. W. L. Hamilton, R. Ying, and J. Leskovec. “Inductive Representation Learning on Large Graphs”. In: *NeurIPS*. 2017 (cit. on pp. 14, 20, 26).

86. W. L. Hamilton. “Graph representation learning”. *Synthesis Lectures on Artifical Intelligence and Machine Learning* 14:3, 2020, pp. 1–159 (cit. on pp. 11–13).

87. W. L. Hamilton, R. Ying, and J. Leskovec. “Representation learning on graphs: Methods and applications”. *IEEE Data Engineering Bulletin*, 2017 (cit. on p. 13).

88. K. Han, B. Lakshminarayanan, and J. Z. Liu. “Reliable Graph Neural Networks for Drug Discovery Under Distributional Shift”. In: *NeurIPS 2021 Workshop on Distribution Shifts: Connecting Methods and Applications*. 2021. URL: `https://openreview.net/forum?id=311QRRkfrep` (cit. on p. 76).

89. L. Haonan, S. H. Huang, T. Ye, and G. Xiuyan. “Graph Star Net for Generalized Multi-Task Learning”. *arXiv*, 2019 (cit. on p. 73).

90. K. Hassani and A. H. Khasahmadi. “Contrastive Multi-View Representation Learning on Graphs”. In: *Proceedings of the 37th International Conference on Machine Learning*. ICML’20. JMLR.org, 2020 (cit. on p. 76).

91. J. He, M. Li, H.-J. Zhang, H. Tong, and C. Zhang. “Manifold-Ranking Based Image Retrieval”. In: *Proceedings of the 12th Annual ACM International Conference on Multimedia*. MULTIMEDIA ’04. Association for Computing Machinery, New York, NY, USA, 2004, 9â€“16. ISBN: 1581138938. DOI: `10.1145/1027527.1027531`. URL: `https://doi.org/10.1145/1027527.1027531` (cit. on pp. 9, 25).

92. K. He, X. Zhang, S. Ren, and J. Sun. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778 (cit. on pp. 2, 18, 63).

93. D. O. Hebb. *The Organization of Behavior*. Wiley, New York, 1949 (cit. on p. 1).

94. G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. “Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups”. *IEEE Signal Processing Magazine* 29:6, 2012, pp. 82–97. DOI: `10.1109/MSP.2012.2205597` (cit. on p. 1).

95. S. Hochreiter and J. Schmidhuber. “Long Short-Term Memory”. *Neural Comput.* 9:8, 1997, pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735 (cit. on p. 1).

96. C. Holtz, O. Atan, R. Carey, and T. Jain. “Multi-Task Learning on Graphs with Node and Graph Level Labels”. In: *NeurIPS Workshop on Graph Representation Learning*. 2019 (cit. on p. 73).

97. T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey. “Meta-Learning in Neural Networks: A Survey”. *arXiv*, 2020 (cit. on p. 73).

98. Z. Huang, S. Zhang, C. Xi, T. Liu, and M. Zhou. “Scaling Up Graph Neural Networks Via Graph Coarsening”. In: *In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD ’21)*. 2021 (cit. on p. 55).

99. N. Immerman and E. Lander. “Describing Graphs: A First-Order Approach to Graph Canonization”. In: *Complexity Theory Retrospective: In Honor of Juris Hartmanis on the Occasion of His Sixtieth Birthday, July 5, 1988*. Ed. by A. L. Selman. Springer New York, New York, NY, 1990, pp. 59–81. ISBN: 978-1-4612-4478-3. DOI: 10.1007/978-1-4612-4478-3_5. URL: https://doi.org/10.1007/978-1-4612-4478-3_5 (cit. on p. 8).

100. S. Ioffe and C. Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. *ICML*, 2015 (cit. on p. 29).

101. S. Ivanov and E. Burnaev. “Anonymous walk embeddings”. *arXiv*, 2018 (cit. on p. 38).

102. R. Jain, G. Ciravegna, P. Barbiero, F. Giannini, D. Buffelli, and P. Liò. “Extending Logic Explained Networks to Text Classification”. In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2022 (cit. on p. 4).

103. S. Jegelka. *Theory of Graph Neural Networks: Representation and Learning*. 2022. DOI: 10.48550/ARXIV.2204.07697. URL: https://arxiv.org/abs/2204.07697 (cit. on pp. 7, 17).

104. B. Jiang and D. Lin. “Graph Laplacian Regularized Graph Convolutional Networks for Semi-supervised Learning”. *arXiv* abs/1809.09839, 2018 (cit. on p. 38).

105. W. Jin, L. Zhao, S. Zhang, Y. Liu, J. Tang, and N. Shah. “Graph Condensation for Graph Neural Networks”. In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=WLEx3Jo4QaB (cit. on p. 55).

106. W. Jin, J. Jung, and U. Kang. “Supervised and extended restart in random walks for ranking and link prediction in networks”. *PLoS ONE* 14, 2019 (cit. on pp. 9, 25).

107. Y. Jin, A. Loukas, and J. JaJa. "Graph Coarsening with Preserved Spectral Properties". In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. Ed. by S. Chiappa and R. Calandra. Vol. 108. Proceedings of Machine Learning Research. PMLR, 2020, pp. 4452–4462. URL: https://proceedings.mlr.press/v108/jin20a.html (cit. on pp. 9, 10, 44, 49, 53, 55).

108. B. Jing, G. Corso, R. Barzilay, and T. S. Jaakkola. "Torsional Diffusion for Molecular Conformer Generation". In: *ICLR2022 Machine Learning for Drug Discovery*. 2022. URL: https://openreview.net/forum?id=D9IxPlXPJJS (cit. on p. 76).

109. C. K. Joshi, Q. Cappart, L.-M. Rousseau, and T. Laurent. "Learning TSP Requires Rethinking Generalization". In: *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*. Ed. by L. D. Michel. Vol. 210. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2021, 33:1–33:21. ISBN: 978-3-95977-211-2. DOI: 10.4230/LIPIcs.CP.2021.33. URL: https://drops.dagstuhl.de/opus/volltexte/2021/15324 (cit. on pp. 41, 53).

110. C. K. Joshi, F. Liu, X. Xun, J. Lin, and C.-S. Foo. "On Representation Knowledge Distillation for Graph Neural Networks". *arXiv preprint arXiv:2111.04964*, 2021 (cit. on p. 46).

111. I. Jovanović and Z. Stanić. "Spectral distances of graphs". *Linear Algebra and its Applications* 436:5, 2012, pp. 1425–1435. ISSN: 0024-3795. DOI: https://doi.org/10.1016/j.laa.2011.08.019. URL: https://www.sciencedirect.com/science/article/pii/S0024379511006021 (cit. on p. 10).

112. W. Ju, Y. Gu, X. Luo, Y. Wang, H. Yuan, H. Zhong, and M. Zhang. "Unsupervised graph-level representation learning with hierarchical contrasts". *Neural Networks* 158, 2023, pp. 359–368. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2022.11.019. URL: https://www.sciencedirect.com/science/article/pii/S0893608022004609 (cit. on p. 76).

113. J. M. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Zídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. A. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis. "Highly accurate protein structure prediction with AlphaFold". *Nature* 596, 2021, pp. 583–589 (cit. on p. 1).

114. A. Kendall, Y. Gal, and R. Cipolla. “Multi-task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics”. In: *CVPR*. 2018 (cit. on p. 67).

115. M. G. Kendall. “The Treatment of Ties in Ranking Problems”. *Biometrika* 33:3, 1945, pp. 239–251. ISSN: 00063444 (cit. on p. 23).

116. J. D. M.-W. C. Kenton and L. K. Toutanova. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of NAACL-HLT*. 2019, pp. 4171–4186 (cit. on p. 18).

117. K. Kersting, N. M. Kriege, C. Morris, P. Mutzel, and M. Neumann. *Benchmark Data Sets for Graph Kernels*. 2016 (cit. on p. 30).

118. S. Kiefer, P. Schweitzer, and E. Selman. “Graphs identified by logics with counting”. In: *International Symposium on Mathematical Foundations of Computer Science*. Springer. 2015, pp. 319–330 (cit. on p. 17).

119. J. Kim, T. Kim, S. Kim, and C. Yoo. “Edge-Labeling Graph Neural Network for Few-Shot Learning”. In: *CVPR*. 2019 (cit. on p. 74).

120. D. P. Kingma and J. Ba. “Adam: A Method for Stochastic Optimization”. *ICLR*, 2015 (cit. on pp. 27–29, 67, 68).

121. T. N. Kipf and M. Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *International Conference on Learning Representations (ICLR)*. 2017 (cit. on pp. 12, 14, 17, 20, 26, 42, 48, 49, 62, 63).

122. T. N. Kipf. “Deep learning with graph-structured representations”. In: *PhD Thesis*. University of Amsterdam, 2020 (cit. on p. 76).

123. J. Klicpera, F. Becker, and S. Günnemann. “GemNet: Universal Directional Graph Neural Networks for Molecules”. In: *Advances in Neural Information Processing Systems*. 2021 (cit. on pp. 2, 12).

124. J. Klicpera, A. Bojchevski, and S. Günnemann. “Predict then Propagate: Graph Neural Networks meet Personalized PageRank”. In: *ICLR*. 2019 (cit. on p. 38).

125. J. Klicpera, S. Weißenberger, and S. Günnemann. “Diffusion Improves Graph Learning”. In: *NeurIPS*. 2019 (cit. on p. 37).

126. B. Knyazev, G. Taylor, and M. Amer. “Understanding Attention in Graph Neural Networks”. In: *ICLR RLGM Workshop*. 2019 (cit. on pp. 27, 32).

127. S. Kornblith, M. Norouzi, H. Lee, and G. Hinton. “Similarity of neural network representations revisited”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 3519–3529 (cit. on pp. 43, 46).

128. A. Krizhevsky, I. Sutskever, and G. E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C. Burges, L. Bottou, and K. Weinberger. Vol. 25. Curran Associates, Inc., 2012. URL: `https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf` (cit. on p. 1).

129. A. Krizhevsky, I. Sutskever, and G. E. Hinton. “Imagenet classification with deep convolutional neural networks”. *Advances in neural information processing systems* 25, 2012 (cit. on p. 16).

130. V. Kurin, A. D. Palma, I. Kostrikov, S. Whiteson, and M. P. Kumar. *In Defense of the Unitary Scalarization for Deep Multi-Task Learning*. 2022. arXiv: `2201.04122 [cs.LG]` (cit. on p. 67).

131. Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. “Gradient-based learning applied to document recognition”. *Proceedings of the IEEE* 86:11, 1998, pp. 2278–2324. DOI: `10.1109/5.726791` (cit. on p. 1).

132. J. B. Lee, R. A. Rossi, S. Kim, N. K. Ahmed, and E. Koh. “Attention Models in Graphs: A Survey”. *arXiv* abs/1807.07984, 2018 (cit. on p. 26).

133. J. B. Lee, R. A. Rossi, and X. Kong. “Graph Classification using Structural Attention”. In: *ACM SIGKDD*. 2018 (cit. on p. 26).

134. J. Lee, I. Lee, and J. Kang. “Self-Attention Graph Pooling”. In: *ICML*. 2019 (cit. on p. 26).

135. D. Li and H. Ji. “Syntax-aware Multi-task Graph Convolutional Networks for Biomedical Relation Extraction”. In: *LOUHI*. 2019 (cit. on p. 73).

136. Q. Li, Z. Han, and X.-M. Wu. “Deeper insights into graph convolutional networks for semi-supervised learning”. In: *Thirty-Second AAAI conference on artificial intelligence*. 2018 (cit. on pp. 18, 19, 37, 38).

137. L. Liu, T. Zhou, G. Long, J. Jiang, and C. Zhang. “Learning to Propagate for Graph Meta-Learning”. In: *NeurIPS*. 2019 (cit. on p. 73).

138. S. Lloyd. “Least squares quantization in PCM”. *IEEE Transactions on Information Theory* 28:2, 1982, pp. 129–137. DOI: `10.1109/TIT.1982.1056489` (cit. on p. 53).

139. P. Lofgren. “Efficient Algorithms for Personalized PageRank”. *PhD Thesis, Stanford University*, 2015 (cit. on pp. 9, 35, 38).

140. P. Lofgren, S. Banerjee, and A. Goel. "Personalized PageRank Estimation and Search: A Bidirectional Approach". In: *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. WSDM '16. Association for Computing Machinery, San Francisco, California, USA, 2016, pp. 163–172. ISBN: 9781450337168. DOI: 10.1145/2835776.2835823. URL: https://doi.org/10.1145/2835776.2835823 (cit. on p. 35).

141. M. Long, Y. Cao, J. Wang, and M. Jordan. "Learning transferable features with deep adaptation networks". In: *International conference on machine learning*. PMLR. 2015, pp. 97–105 (cit. on p. 55).

142. M. Long, H. Zhu, J. Wang, and M. I. Jordan. "Deep Transfer Learning with Joint Adaptation Networks". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. JMLR.org, Sydney, NSW, Australia, 2017, pp. 2208–2217 (cit. on pp. 44, 55).

143. A. Loukas. "Graph Reduction with Spectral and Cut Guarantees". *Journal of Machine Learning Research* 20:116, 2019, pp. 1–42. URL: http://jmlr.org/papers/v20/18-680.html (cit. on pp. 44, 55).

144. A. Loukas. "What graph neural networks cannot learn: depth vs width". In: *ICLR*. 2020 (cit. on p. 38).

145. A. Loukas and P. Vandergheynst. "Spectrally Approximating Large Graphs with Smaller Graphs". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by J. Dy and A. Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 3237–3246. URL: https://proceedings.mlr.press/v80/loukas18a.html (cit. on pp. 9, 44, 55).

146. L. Lovász. "Random Walks on Graphs: A Survey". In: *Combinatorics, Paul Erdős is Eighty*. Ed. by D. Miklós, V. T. Sós, and T. Szőnyi. Vol. 2. János Bolyai Mathematical Society, Budapest, 1996, pp. 353–398 (cit. on p. 9).

147. T. Ma and J. Chen. "Unsupervised Learning of Graph Hierarchical Abstractions with Differentiable Coarsening and Optimal Transport". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 10. 2021, pp. 8856–8864 (cit. on p. 55).

148. K. Madhawa and T. Murata. "Active Learning on Graphs via Meta Learning". In: *ICML Workshop on Graph Representation Learning and Beyond, ICML*. 2020 (cit. on p. 74).

149. K.-K. Maninis, I. Radosavovic, and I. Kokkinos. "Attentive Single-Tasking of Multiple Tasks". In: *CVPR*. 2019 (cit. on p. 69).

150. H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman. “Provably powerful graph networks”. *Advances in neural information processing systems* 32, 2019 (cit. on p. 18).

151. N. Masuda, M. A. Porter, and R. Lambiotte. “Random walks and diffusion on networks”. *Physics Reports* 716-717, 2017. Random walks and diffusion on networks, pp. 1–58. ISSN: 0370-1573. DOI: https://doi.org/10.1016/j.physrep.2017.07.007. URL: https://www.sciencedirect.com/science/article/pii/S0370157317302946 (cit. on p. 19).

152. W. Mcculloch and W. Pitts. “A Logical Calculus of Ideas Immanent in Nervous Activity”. *Bulletin of Mathematical Biophysics* 5, 1943, pp. 127–147 (cit. on p. 1).

153. K. Mehlhorn. *Data structures and algorithms 2: graph algorithms and NP-completeness*. Vol. 2. Springer Science & Business Media, 2012 (cit. on p. 6).

154. S. Micali and Z. A. Zhu. “Reconstructing markov processes from independent and anonymous experiments”. *Discrete Applied Mathematics* 200, 2016, pp. 108–122 (cit. on p. 38).

155. T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. “Distributed representations of words and phrases and their compositionality”. *Advances in neural information processing systems* 26, 2013 (cit. on p. 12).

156. E. Min, R. Chen, Y. Bian, T. Xu, K. Zhao, W. Huang, P. Zhao, J. Huang, S. Ananiadou, and Y. Rong. *Transformer for Graphs: An Overview from Architecture Perspective*. 2022. DOI: 10.48550/ARXIV.2202.08455. URL: https://arxiv.org/abs/2202.08455 (cit. on p. 75).

157. T. M. Mitchell. *The Need for Biases in Learning Generalizations*. Technical report. New Brunswick, NJ: Rutgers University, 1980 (cit. on p. 1).

158. V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. “Human-level control through deep reinforcement learning”. *Nature* 518, 2015, pp. 529–533 (cit. on p. 1).

159. F. Montanari, L. Kuhnke, A. T. Laak, and D. A. Clevert. “Modeling Physico-Chemical ADMET Endpoints with Multitask Graph Convolutional Networks”. *Molecules*, 2019 (cit. on p. 73).

160. C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann. “TUDataset: A collection of benchmark datasets for learning with graphs”. In: *ICML Workshop on Graph Representation Learning and Beyond*. 2020 (cit. on pp. 48, 66).

161. C. Morris, Y. Lipman, H. Maron, B. Rieck, N. M. Kriege, M. Grohe, M. Fey, and K. Borgwardt. *Weisfeiler and Leman go Machine Learning: The Story so far*. 2021. DOI: 10.48550/ARXIV.2112.09992. URL: https://arxiv.org/abs/2112.09992 (cit. on p. 7).

162. C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. "Weisfeiler and leman go neural: Higher-order graph neural networks". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 4602–4609 (cit. on pp. 16–18, 20, 26, 31, 38).

163. R. Murphy, B. Srinivasan, V. Rao, and B. Ribeiro. "Relational pooling for graph representations". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 4663–4673 (cit. on pp. 18, 26, 48, 51).

164. M. Newman. *Networks*. Oxford university press, 2018 (cit. on p. 10).

165. C. Q. Nguyen, C. Kreatsoulas, and B. K. M. "Meta-Learning GNN Initializations for Low-Resource Molecular Property Prediction". In: *ICML Workshop on Graph Representation Learning and Beyond, ICML*. 2020 (cit. on p. 74).

166. R. I. Oliveira and Y. Peres. "Random walks on graphs: new bounds on hitting, meeting, coalescing and returning". In: *2019 Proceedings of the Meeting on Analytic Algorithmics and Combinatorics (ANALCO)*, pp. 119–126. DOI: 10.1137/1.9781611975505.13. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611975505.13. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611975505.13 (cit. on p. 9).

167. K. Oono and T. Suzuki. "Graph Neural Networks Exponentially Lose Expressive Power for Node Classification". In: *International Conference on Learning Representations*. 2019 (cit. on p. 18).

168. A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. "WaveNet: A Generative Model for Raw Audio". In: *9th ISCA Speech Synthesis Workshop*, pp. 125–125 (cit. on p. 18).

169. A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. *WaveNet: A Generative Model for Raw Audio*. 2016. DOI: 10.48550/ARXIV.1609.03499. URL: https://arxiv.org/abs/1609.03499 (cit. on p. 1).

170. L. Page, S. Brin, R. Motwani, and T. Winograd. "The PageRank citation ranking: Bringing order to the Web". In: *WWW*. 1998 (cit. on pp. 9, 20, 25).

171. A. Paszke, S. Gross, et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *NeurIPS*. 2019 (cit. on p. 67).

172. F. Pedregosa, G. Varoquaux, A. Gramfort, et al. "Scikit-learn: Machine Learning in Python". *Journal of Machine Learning Research*, 2011 (cit. on p. 67).

173. M. Peng, Q. Zhang, Y.-g. Jiang, and X.-J. Huang. "Cross-domain sentiment classification with target domain specific information". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2018, pp. 2505–2513 (cit. on p. 56).

174. J. Pennington, R. Socher, and C. D. Manning. "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543 (cit. on p. 12).

175. B. Perozzi, R. Al-Rfou, and S. Skiena. "DeepWalk: Online Learning of Social Representations". In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '14. ACM, New York, New York, USA, 2014, pp. 701–710. ISBN: 978-1-4503-2956-9. DOI: `10.1145/2623330.2623732`. URL: `http://doi.acm.org/10.1145/2623330.2623732` (cit. on p. 11).

176. B. Perozzi, R. Al-Rfou, and S. Skiena. "Deepwalk: Online learning of social representations". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014, pp. 701–710 (cit. on p. 12).

177. M. Prates, P. H. C. Avelar, H. Lemos, L. C. Lamb, and M. Y. Vardi. "Learning to Solve NP-Complete Problems: A Graph Neural Network for Decision TSP". *Proceedings of the AAAI Conference on Artificial Intelligence* 33, 2019, pp. 4731–4738. DOI: `10.1609/aaai.v33i01.33014731` (cit. on pp. 41, 76).

178. A. Raghu, M. Raghu, S. Bengio, and O. Vinyals. "Rapid learning or feature reuse? towards understanding the effectiveness of maml". In: *ICLR*. 2020 (cit. on pp. 58, 60, 65).

179. F. A. Rodrigues. "Network centrality: an introduction". In: *A mathematical modeling approach from nonlinear dynamics to complex systems*. Springer, 2019, pp. 177–196 (cit. on p. 10).

180. F. Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review* 65 6, 1958, pp. 386–408 (cit. on p. 1).

181. E. Rosenfeld, P. Ravikumar, and A. Risteski. "The Risks of Invariant Risk Minimization". In: *International Conference on Learning Representations*. Vol. 9. 2021 (cit. on p. 55).

182. S. T. Roweis and L. K. Saul. "Nonlinear dimensionality reduction by locally linear embedding". *science* 290:5500, 2000, pp. 2323–2326 (cit. on p. 12).

183. B. Rozemberczki, O. Kiss, and R. Sarkar. “Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs”. In: *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM ’20)*. ACM. 2020, pp. 3125–3132 (cit. on p. 52).

184. D. E. Rumelhart, G. E. Hinton, and R. J. Williams. “Learning representations by back-propagating errors”. *Nature* 323, 1986, pp. 533–536 (cit. on p. 1).

185. A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia. “Graph networks as learnable physics engines for inference and control”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 4470–4479 (cit. on p. 54).

186. P. Sarkar and A. W. Moore. “Random walks in social networks and their applications: a survey”. In: *Social Network Data Analytics*. Springer, 2011, pp. 43–77 (cit. on p. 9).

187. R. Sato, M. Yamada, and H. Kashima. “Approximation Ratios of Graph Neural Networks for Combinatorial Problems”. In: *NeurIPS*. 2019 (cit. on p. 38).

188. F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. “The Graph Neural Network Model”. *IEEE Transactions on Neural Networks* 20:1, 2009, pp. 61–80. DOI: 10.1109/TNN.2008.2005605 (cit. on p. 12).

189. I. Schomburg, A. Chang, C. Ebeling, M. Gremse, C. Heldt, G. Huhn, and D. Schomburg. “BRENDA, the enzyme database: updates and major new developments”. *Nucleic acids research*, 2004 (cit. on p. 66).

190. P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. “Collective Classification in Network Data”. *AI Magazine* 29:3, 2008, p. 93. ISSN: 0738-4602. DOI: 10.1609/aimag.v29i3.2157. URL: http://dx.doi.org/10.1609/aimag.v29i3.2157 (cit. on p. 30).

191. P. Sermanet, S. Chintala, and Y. LeCun. “Convolutional neural networks applied to house numbers digit classification”. In: *Proceedings of the 21st international conference on pattern recognition (ICPR2012)*. IEEE. 2012, pp. 3288–3291 (cit. on p. 16).

192. N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. “Weisfeiler-Lehman Graph Kernels”. *Journal of Machine Learning Research* 12:77, 2011, pp. 2539–2561. URL: http://jmlr.org/papers/v12/shervashidze11a.html (cit. on pp. 48, 51).

193. N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt. “Efficient graphlet kernels for large graph comparison”. In: *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*. Ed. by D. van Dyk and M. Welling. Vol. 5. Proceedings of Machine Learning Research. PMLR, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 2009, pp. 488–495. URL: `https://proceedings.mlr.press/v5/shervashidze09a.html` (cit. on pp. 48, 51).

194. D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. “Mastering the Game of Go with Deep Neural Networks and Tree Search”. *Nature* 529:7587, 2016, pp. 484–489. DOI: `10.1038/nature16961` (cit. on p. 1).

195. A. Sperduti and A. Starita. “Supervised neural networks for the classification of structures”. *IEEE Transactions on Neural Networks* 8:3, 1997, pp. 714–735. DOI: `10.1109/72.572108` (cit. on p. 12).

196. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. *Journal of Machine Learning Research* 15, 2014, pp. 1929–1958 (cit. on pp. 27–29).

197. Q. Suo, J. Chou, W. Zhong, and A. Zhang. “TAdaNet: Task-Adaptive Network for Graph-Enriched Meta-Learning”. In: *ACM SIGKDD*. 2020 (cit. on p. 74).

198. J. J. Sutherland, L. A. O’Brien, and D. F. Weaver. “Spline-Fitting with a Genetic Algorithm: A Method for Developing Classification Structure-Activity Relationships”. *Journal of Chemical Information and Computer Sciences*, 2003 (cit. on p. 67).

199. J. Tenenbaum, V. Silva, and J. Langford. “A Global Geometric Framework for Nonlinear Dimensionality Reduction”. *Science* 290, 2000, pp. 2319–2323 (cit. on p. 12).

200. H. Tong, C. Faloutsos, and J. Pan. “Fast Random Walk with Restart and Its Applications”. *ICDM*, 2006 (cit. on pp. 9, 25, 38).

201. J. Topping, F. Di Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein. “Understanding over-squashing and bottlenecks on graphs via curvature”. In: *International Conference on Learning Representations*. 2021 (cit. on p. 18).

202. M. Tsubaki and T. Mizoguchi. “On the equivalence of molecular graph convolution and molecular wave function with poor basis set”. *Advances in Neural Information Processing Systems* 33, 2020, pp. 1982–1993 (cit. on p. 54).

203. S. Vandenhende, S. Georgoulis, W. Van Gansbeke, M. Proesmans, D. Dai, and L. Van Gool. “Multi-Task Learning for Dense Prediction Tasks: A Survey”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021, pp. 1–1. DOI: 10.1109/TPAMI.2021.3054719 (cit. on pp. 59, 67).

204. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. “Attention is all you need”. *Advances in neural information processing systems* 30, 2017 (cit. on pp. 1, 15, 75).

205. P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. “Graph Attention Networks”. In: *International Conference on Learning Representations*. 2018 (cit. on pp. 12, 15, 20, 26).

206. S. Wang, R. Yang, R. Wang, X. Xiao, Z. Wei, W. Lin, Y. Yang, and N. Tang. “Efficient Algorithms for Approximate Single-Source Personalized PageRank Queries”. *ACM Transactions on Database Systems*, 2019. ISSN: 0362-5915. DOI: 10.1145/3360902. URL: http://dx.doi.org/10.1145/3360902 (cit. on pp. 35, 38).

207. S. Wang, R. Yang, X. Xiao, Z. Wei, and Y. Yang. “FORA: Simple and Effective Approximate Single-Source Personalized PageRank”. In: *SIGKDD 2017*. 2017, pp. 505–514 (cit. on p. 35).

208. Z. Wei, X. He, X. Xiao, S. Wang, S. Shang, and J. Wen. “TopPPR: Top-k Personalized PageRank Queries with Precision Guarantees on Large Graphs”. *SIGMOD*, 2018. DOI: 10.1145/3183713.3196920. URL: http://dx.doi.org/10.1145/3183713.3196920 (cit. on pp. 35, 38).

209. B. Weisfeiler and A. Leman. “A reduction of a graph to a canonical form and an algebra arising during this reduction”. In: *Nauchno-Technicheskaya Informatsia*. 1968 (cit. on pp. 6, 20).

210. G. Wilson and D. J. Cook. “A Survey of Unsupervised Deep Domain Adaptation”. *ACM Transactions on Intelligent Systems and Technology* 11:5, 2020, pp. 1–46. DOI: 10.1145/3400066 (cit. on p. 55).

211. S. Wold, K. Esbensen, and P. Geladi. “Principal component analysis”. *Chemometrics and intelligent laboratory systems* 2:1-3, 1987, pp. 37–52 (cit. on p. 10).

212. L. Wu, P. Cui, J. Pei, and L. Zhao. *Graph Neural Networks: Foundations, Frontiers, and Applications*. Springer Singapore, Singapore, 2022, p. 725 (cit. on p. 13).

213. Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. “A Comprehensive Survey on Graph Neural Networks”. *IEEE Transactions on Neural Networks and Learning Systems* 32:1, 2021, pp. 4–24. DOI: 10.1109/TNNLS.2020.2978386 (cit. on pp. 13, 44).

214. F. Xia, J. Liu, H. Nie, Y. Fu, L. Wan, and X. Kong. “Random Walks: A Review of Algorithms and Applications”. *IEEE Transactions on Emerging Topics in Computational Intelligence* 4:2, 2020, pp. 95–107. DOI: `10.1109/TETCI.2019.2952908` (cit. on p. 9).

215. Y. Xie, M. Gong, Y. Gao, A. K. Qin, and X. Fan. “A Multi-Task Representation Learning Architecture for Enhanced Graph Classification”. *Frontiers in Neuroscience*, 2020 (cit. on p. 73).

216. K. Xu, W. Hu, J. Leskovec, and S. Jegelka. “How Powerful are Graph Neural Networks?” In: *International Conference on Learning Representations*. 2019. URL: `https://openreview.net/forum?id=ryGs6iA5Km` (cit. on pp. 2, 12, 15, 17, 38, 42, 46, 48, 49).

217. K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka. “Representation Learning on Graphs with Jumping Knowledge Networks”. In: *ICML*. 2018 (cit. on p. 38).

218. K. Xu, J. Li, M. Zhang, S. S. Du, K.-i. Kawarabayashi, and S. Jegelka. “What Can Neural Networks Reason About?” In: *ICLR*. 2020 (cit. on p. 38).

219. H. Yan, Y. Ding, P. Li, Q. Wang, Y. Xu, and W. Zuo. “Mind the class weight bias: Weighted maximum mean discrepancy for unsupervised domain adaptation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2272–2281 (cit. on p. 55).

220. Z. Yang, W. Cohen, and R. Salakhudinov. “Revisiting semi-supervised learning with graph embeddings”. In: *International conference on machine learning*. PMLR. 2016, pp. 40–48 (cit. on pp. 12, 31).

221. H. Yao, C. Zhang, Y. Wei, M. Jiang, S. Wang, J. Huang, N. V. Chawla, and Z. Li. “Graph Few-shot Learning via Knowledge Transfer”. In: *AAAI*. 2020 (cit. on p. 74).

222. M. Yasunaga, H. Ren, A. Bosselut, P. Liang, and J. Leskovec. “QA-GNN: Reasoning with Language Models and Knowledge Graphs for Question Answering”. In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2021, pp. 535–546 (cit. on p. 12).

223. G. Yehudai, E. Fetaya, E. Meirom, G. Chechik, and H. Maron. “From Local Structures to Size Generalization in Graph Neural Networks”. In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by M. Meila and T. Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 11975–11986. URL: `https://proceedings.mlr.press/v139/yehudai21a.html` (cit. on pp. 41, 42, 45, 48, 49, 53, 54).

224. R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. “Graph convolutional neural networks for web-scale recommender systems”. In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2018, pp. 974–983 (cit. on pp. 2, 12, 13, 37).

225. Z. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec. “Hierarchical Graph Representation Learning with Differentiable Pooling”. In: *NeurIPS*. 2018 (cit. on pp. 16, 20, 26, 31).

226. S. Yu, Y. Feng, D. Zhang, H. D. Bedru, B. Xu, and F. Xia. “Motif discovery in networks: a survey”. *Computer Science Review* 37, 2020, p. 100267 (cit. on p. 10).

227. W. W. Zachary. “An Information Flow Model for Conflict and Fission in Small Groups”. *Journal of Anthropological Research* 33:4, 1977, pp. 452–473. ISSN: 00917710. URL: `http://www.jstor.org/stable/3629752` (visited on 07/28/2022) (cit. on p. 11).

228. W. Zellinger, B. A. Moser, T. Grubinger, E. Lughofer, T. Natschläger, and S. Saminger-Platz. “Robust unsupervised domain adaptation for neural networks via moment alignment”. *Information Sciences* 483, 2019, pp. 174–191. ISSN: 0020-0255. DOI: `https://doi.org/10.1016/j.ins.2019.01.025`. URL: `https://www.sciencedirect.com/science/article/pii/S0020025519300301` (cit. on pp. 43, 44, 56).

229. C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla. “Heterogeneous Graph Neural Network”. *ACM SIGKDD*, 2019. DOI: `10.1145/3292500.3330961`. URL: `http://dx.doi.org/10.1145/3292500.3330961` (cit. on p. 37).

230. J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D.-Y. Yeung. “GaAN: Gated Attention Networks for Learning on Large and Spatiotemporal Graphs”. In: *UAI*. 2018 (cit. on p. 26).

231. F. Zhou, C. Cao, K. Zhang, G. Trajcevski, T. Zhong, and J. Geng. “Meta-GNN: On Few-Shot Node Classification in Graph Meta-Learning”. In: *CIKM*. 2019 (cit. on p. 74).

232. K. Zhou, X. Huang, Y. Li, D. Zha, R. Chen, and X. Hu. “Towards deeper graph neural networks with differentiable group normalization”. *Advances in neural information processing systems* 33, 2020, pp. 4917–4928 (cit. on p. 18).

233. Q. Zhu, N. Ponomareva, J. Han, and B. Perozzi. “Shift-Robust GNNs: Overcoming the Limitations of Localized Graph Training data”. In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan. Vol. 34. Curran Associates, Inc., 2021, pp. 27965–27977. URL: `https://proceedings.neurips.cc/paper/2021/file/eb55e369affa90f77dd7dc9e2cd33b16-Paper.pdf` (cit. on pp. 43, 44, 56).

234. C. Zhuang and Q. Ma. “Dual Graph Convolutional Networks for Graph-Based Semi-Supervised Classification”. *WWW*, 2018. DOI: 10.1145/3178876.3186116. URL: http://dx.doi.org/10.1145/3178876.3186116 (cit. on p. 37).

235. M. Zopf. *1-WL Expressiveness Is (Almost) All You Need*. 2022. DOI: 10.48550/ARXIV.2202.10156. URL: https://arxiv.org/abs/2202.10156 (cit. on p. 8).

236. D. Zügner and S. Günnemann. “Adversarial Attacks on Graph Neural Networks via Meta Learning”. In: *ICLR*. 2019 (cit. on p. 74).