

UNIVERSITÀ DEGLI STUDI DI SIENA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE E SCIENZE MATEMATICHE



UNIVERSITÀ
DI SIENA
1240

Theoretical Properties of Graph Neural Networks

Giuseppe Alessio D'Inverno

Ph.D Thesis in Information Engineering & Science

Supervisors

Prof. Maria Lucia Sampoli

Prof. Franco Scarselli

Prof. Monica Bianchini

Examination Committee

Prof. Pasquale Foggia

Prof. Marco Maggini

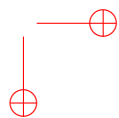
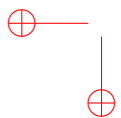
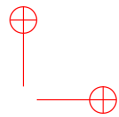
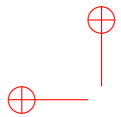
Prof. Hendrik Speleers

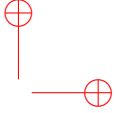
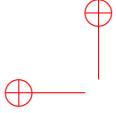
Thesis reviewers

Prof. Pasquale Foggia

Prof. Andrea Passerini

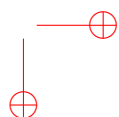
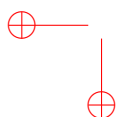
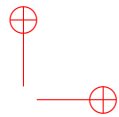
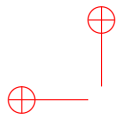
SIENA, 15/04/2024





Abstract

Graph Neural Networks (GNNs) have emerged in recent years as a powerful tool to learn tasks across a wide range of graph domains in a data-driven fashion; based on a message passing mechanism, GNNs have gained increasing popularity due to their intuitive formulation, closely linked with the Weisfeiler-Lehman (WL) test for graph isomorphism, to which they have proven equivalent. In this thesis, we provide a broad overview of two essential properties of GNNs by a theoretical point of view, namely, their approximation power and their generalization capabilities. We show that modern GNNs are universal approximators, given that they are made by a sufficient number of layers, which is tightly linked to the stable node coloring of the 1-WL test. GNNs are shown to be universal approximators also on more complex graph domains, like edge-attributed graphs and dynamic graphs. Generalization capabilities of GNNs are investigated by different perspectives. Bounds on the VC dimension of GNNs are provided with respect to the usual hyperparameters and with respect to the number of colors derived from the 1-WL test. GNNs ability to generalize to unseen data is also explored by a neurocognitive point of view, determining whether these models are able to learn the so-called identity effects.



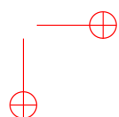
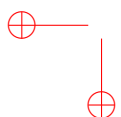
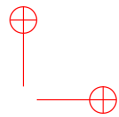
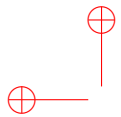
Contents

Acknowledgements	xiii
1 Introduction	1
1.1 Graph Neural Networks: A theoretical analysis	1
1.2 Thesis summary	2
1.2.1 Main contributions of the thesis	3
1.2.2 Structure of the thesis	5
I Background topics and literature review	7
2 Background Topics	9
2.1 Graph theory: Basic concepts	9
2.2 The color refinement algorithm and the Weisfeiler–Lehman test	10
2.3 Graph Neural Networks	11
3 Theory of Graph Neural Networks: A literature review	15
3.1 Neural networks as universal approximators	15
3.1.1 The approximation power of GNNs	15
3.2 Generalization capabilities of neural networks	17
3.2.1 GNN generalization properties	17
3.2.2 Neural networks and cognitive tasks: the <i>identity effect learning</i> case	17
II Approximation capabilities of Graph Neural Networks	19
4 Universality of GNNs for node-attributed graphs	21
4.1 Preliminaries	21
4.1.1 Unfolding trees and unfolding equivalence	21
4.1.2 The Weisfeiler–Lehman equivalence	22
4.2 Main Results	23
4.2.1 Unfolding and Weisfeiler–Lehman equivalence	23

4.2.2	Approximation capability	31
4.3	Experimental Validation	41
5	Universality of GNNs for SAUHG and Dynamic GNNs	45
5.1	Notation and Preliminaries	45
5.2	Weisfeiler-Lehman and Unfolding Trees	48
5.2.1	Equivalence for SAUHGs	49
5.2.2	Equivalence for Dynamic Graphs	51
5.3	Approximation Capability of GNNs for SAUHGs and DGNNs	53
5.3.1	GNNs for SAUHGs	53
5.3.2	GNNs for Dynamic Graphs	57
5.4	Experimental Validation	66
III	Generalization capabilities of Graph Neural Networks	69
6	VC dimension of message passing GNNs with Pfaffian activation functions	71
6.1	Notation and preliminaries	71
6.1.1	Results from the literature	72
6.2	Main Results	73
6.2.1	Main bounds	74
6.2.2	Computation of the main bounds for a specific GNN model	78
6.2.3	Bounds with 1-WL colors	80
6.3	Experimental validation	82
6.3.1	Experimental setting	82
6.3.2	Experimental results	83
7	Learning Identity Effects with GNNs	87
7.1	Rating impossibility for invariant learners	87
7.2	Theoretical analysis	88
7.2.1	What GNNs cannot learn: rating impossibility theorem	89
7.2.2	What GNNs can learn: identity effects on dicyclic graphs	94
7.3	Numerical results	98
7.3.1	Experimental Setup	98
7.3.2	Case study #1: two-letter words	99
7.3.3	Case study #2: dicyclic graphs	101
8	Other works	109
8.1	A topological description of loss surfaces based on Betti Numbers	109
8.2	Extension of Recurrent Kernels to different Reservoir Computing topologies	110
8.3	Splines Parameterization of Planar Domains by Physics-Informed Neural Networks	112
8.4	Visual Sequential Search Test Analysis: An Algorithmic Approach	113

Contents

9 Conclusions	115
Bibliography	119



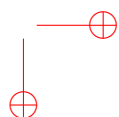
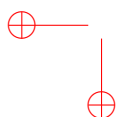
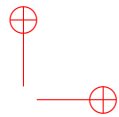
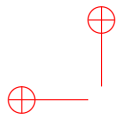
List of Figures

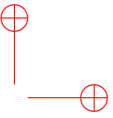
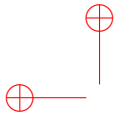
3.1	An example of a graph with some unfolding trees. The symbols outside the nodes represent features. The two nodes on the left part of the graph are equivalent and have equivalent unfolding trees.	16
4.1	A graphical representation of the relationship between the color refinement and the unfolding equivalence, applied on nodes 1 and 4 of the given graph.	26
4.2	(a) A regular graph where all nodes have the same features. All unfolding trees are equal. (b) The equivalence classes when only one node has different features. (c) The equivalence classes when all nodes have different features.	27
4.3	In (a) and (b), two graphs G, H are depicted that satisfy the lower bound of point (2) of of Theorem 4.9. Graphs in (a) and (b) are constructed by aggregating in a sequence two copies of the same subgraph (c); then, module (d) is added at the top of graph (a), while module (e) is added at the top of graph (b).	30
4.4	Structure of the proof of Theorem 4.13.	31
4.5	The ATTACH operator on trees.	33
4.6	Training accuracy on subsampled QM9 dataset, increasing number of WL colors (a), and increasing hidden layer size (b). The solid line represents the average over 15 runs, the shaded area represents the confidence interval.	42
5.1	Illustration of the statification of a dynamic graph. On the left, the temporal evolution of a graph, including non-existent nodes and edges (gray), is given, and on the right, the corresponding statified graph with the total amount of nodes and edges together with the concatenated attributes is shown.	47
5.2	Unfolding tree recursive construction	49
5.3	Structure of the proof of Theorem 5.21.	56
5.4	Structure of the proof of Theorem 5.29.	62
5.5	The four static graphs used as components to generate the synthetic dataset. Graphs a) and b) are equivalent under the static 1-WL test; same holds for c) and d).	67
5.6	Train accuracy over the epochs for a DGNN trained on the dataset containing dynamic graphs up to time length $T = 4$ (a) and $T = 5$ (b).	68

6.1	Summary of the parameters for each split of the ordered NCI1 dataset for the task E2	83
6.2	Results on the task E1 for GNNs with activation function arctan . Pictures (a) and (c) show the evolution of diff through the epochs, for different values of d , keeping fixed $L = 3$, and for different values of L , keeping fixed $d = 32$; Picture (b) shows how diff evolves as the hidden size increases, while Picture (d) shows how diff evolves as the number of layers increases.	84
6.3	Results on the task E1 for GNNs with activation function tanh . Pictures (a) and (c) show the evolution of diff through the epochs, for different values of d , keeping fixed $L = 3$, and for different values of L , keeping fixed $d = 32$; Picture (b) shows how diff evolves as the hidden size increases, while Picture (d) shows how diff evolves as the number of layers increases.	85
6.4	Results on the task E2 for GNNs with activation function tanh . Picture (a) shows the evolution of diff through the epochs, for different values of $\frac{V(G)}{C^k(G)}$, keeping fixed $L = 4$ and $d = 16$; Picture (b) shows how diff evolves as the ratio $\frac{V(G)}{C^k(G)}$ increases.	86
7.1	Graph modeling of a two-nodes graph: a vertex feature $\alpha(v) \in \mathbb{R}^q$ is attached to each node v of a two-node undirected graph, according to a given encoding \mathcal{E} . In this figure, \mathcal{E} is the one-hot encoding.	89
7.2	Graphical illustration of Lemma 7.5: a 6-cycle reaches a stable coloring in $\lfloor \frac{6}{2} \rfloor = 3$ steps with $\lfloor \frac{6}{2} \rfloor + 1 = 4$ colors. Numbers are used to identify nodes.	96
7.3	Stable 1-WL coloring for different types of dicyclic graphs: as stated in Theorem 7.6, 3-degree nodes have the same color in symmetric dicyclic graphs, and different color in the asymmetric ones.	97
7.4	Numerical results for the rating task on the two-letter words dataset using Gconv-glob with $L = 1, 2, 3$ layers. Rating should be equal to 1 if words are composed by identical letters, 0 otherwise. The distributed and Gaussian encodings, which deviate from the framework outlined in Theorem 7.2, exhibit superior performance compared to the other encodings. The other encodings makes the transformation matrix orthogonal and symmetric, being themselves orthogonal encodings.	100
7.5	Numerical results for the rating task on the two-letter words dataset using Gconv-diff with $L = 1, 2, 3$ layers. The same observations to those in Figure 7.4 can be made here as well.	101
7.6	Perfect classification of symmetric dicyclic graphs by n_{\max} iterations of the 1-WL test.	102
7.7	Graphical illustration of the extraction task. In this example, $n_{\max} = 6$ and $k = 5$	103
7.8	Extraction task performed by different GNN models, namely Gconv-glob (left) and Gconv-diff (right). We set $n_{\max} = 8$, $l = 8$ and, from top to bottom, $k = 7, 6, 5$	105
7.9	Graphical illustration of the extrapolation task. In this example, $n_{\max} = 5$ and $g = 2$	106
7.10	Extrapolation task performed by different GNN models, namely Gconv-glob (left) and Gconv-diff (right). We set $n_{\max} = 8$ and, from top to bottom, $(L, g) = (9, 1), (10, 2), (11, 3)$	107

List of Figures

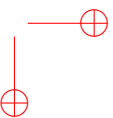
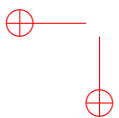
8.1	Recurrent Kernels associated with various Reservoir Computing topologies. RC and sparse RC converge to the same RK limit when the reservoir size $N \rightarrow \infty$. Leaky RC and Deep RC converge to their corresponding limits.	111
8.2	(Left) Error metric normalized between 0 and 1 as a function of sparsity for different reservoir sizes. (Right) Sparsity threshold above which the error metric is within 10% of the non-sparse limit. This gives an admissible sparsity level which decreases with the reservoir size.	111
8.3	Hourglass-shaped domain.	112
8.4	Flowchart of our method to compute the score of the performance in the Visual Sequential Search Task.	114

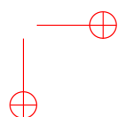
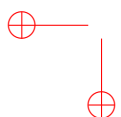
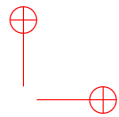
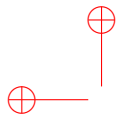




List of Tables

2.1 Basic notation	9
6.1 Upper bounds on VCdim of common architectures: p is the number of network parameters, N the number of nodes in the input graph or sequence, while \mathcal{C} is the maximum number of colors per graph.	77







Acknowledgements

At the end of a journey last 3 years, several are the people that I need to thank.

First of all, a huge thanks to my advisors Maria Lucia Sampoli, Franco Scarselli and Monica Bianchini. You taught me a lot, from how to do research to how to catch the once-in-a-lifetime opportunities, from how to give all my best to how to live the life a little lighter. You made me the researcher and in some way the man that I am now.

Thanks to the professors that hosted me in my visiting periods abroad, prof. Michael Unser in Lausanne, prof. Simone Brugiapaglia and prof. Mirco Ravanelli in Montréal. Thank you for the opportunity you gave me to expand my horizons and to work in such stimulating academic environments.

Thanks to my colleagues in Siena, that shared with me joys and pains of my academic career. Caterina and Veronica, you have been the greatest mates I could have ever wondered to have for this PhD. Paolo, Simone, Niccolò, Pietro, Barbara, thank you for sharing your knowledge of coding and not giving up on answering every doubt I had on Python programming. All the people in lab, even if we were noisy and the room was overcrowded, I enjoyed all of that, and I will surely miss it.

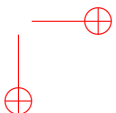
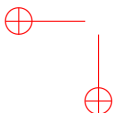
Thanks to my colleagues in Lausanne, especially thanks to Jonathan, who has revealed to be a brother in Christ, a violin mate and a good friend.

Thanks to my colleagues (met) in Montréal, Simone, Massimo and Salah. The time spent at MILA (and eating at my place) with you will always be among the good memories.

To my friends: you're too many, it's difficult to list you one by one, and I wouldn't want to forget anyone, because each of you supported me during these years and encouraged me, and that was gold to me. I'm grateful to all of you, wherever you are around the world.

My deepest gratitude goes to my parents: after all these years, I can understand how much you invested in my studies, and how much effort it required to you. I won't ever forget it. Thank you for your love and your patience, thank you for not ever giving up on my passions and indulging on them (with caution). Grazie.

To Giovanna, my sister: you're the example I've been following for my whole life, and during the PhD I made no exception in that. Thank you because you opened the way, even to ensure mom that we could easily live some months at the opposite corner of the



Earth. Thank you because you showed me how to balance work life and faith life. Ti voglio bene sorellina.

And on the top of all of it, my biggest thanks, my everlasting gratitude, goes to the One who paid the ransom and made me His: my dear Lord Jesus Christ. Dear Lord, I hope that this PhD can talk to everyone about how good You have been, how beautiful You are, and how You and You only are, and will be, the priority in my life. To you I dedicate every achievement You give me grace to achieve, and as long as I will breathe, I won't stop singing about your goodness.

"Come to me, all you who are weary and burdened, and I will give you rest. Take my yoke upon you and learn from me, for I am gentle and humble in." (Matthew 11:28-30)



Chapter 1

Introduction

Deep Learning is quickly becoming important in almost every application field. As a consequence, in recent years, the investigation of its properties by a rigorous theoretical analysis has become increasingly urgent. This thesis aims to shed light on the theoretical characteristics of Graph Neural Networks (GNNs) [1], a modern class of machine learning models designed for graph processing; specifically, this thesis focuses on two important theoretical aspects of GNNs, namely, universal approximation and generalization capabilities

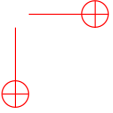
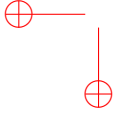
1.1 Graph Neural Networks: A theoretical analysis

Since their first introduction [2, 3], GNNs were proposed as a new learning paradigm to deal with structured data. Indeed, while standard neural networks require pre-processing of complex data, GNNs are able to process the information being aware of the graph structure; for this reason, GNNs have quickly become dominant in data-driven learning over several application scenarios such as network analysis [4], molecule prediction [5] and generation [6], text classification [7], and traffic forecasting [8].

GNNs are based on the so-called *message-passing mechanism* [9]. According to this paradigm, the neural architecture is implemented by an information flow through the graph. In particular, a vector of features is associated to every node in the graph. The algorithm computes a message for each edge, which is a function of the features of the edge end nodes. Each node aggregates the messages of its edges, using a permutation-invariant function. Then, each node updates its feature vector as a function of its attribute and aggregated message. Such a mechanism is present in almost all milestone GNN models, such as Graph Convolutional Networks (GCNs) [10], Graph Attention Networks (GATs) [11], GraphSAGE [12] and Graph Isomorphism Networks (GINs) [13].

Graph Neural Networks (GNNs) have been successfully implemented to solve a wide range of classification and regression tasks. Depending on the goal, a task can be defined as either graph-focused, node-focused, or edge-focused (also known as *link prediction*). In graph-focused tasks, the GNN returns an output for each input graph, in node and edge-focused tasks, an output is returned for each node or edge, respectively. Examples of GNN applications in graph-focused tasks are molecule property prediction [14, 9] and graph matching [15]. Effective node prediction has been performed by GNN models in weather forecasting [16] and power-grid analysis [17]. GNNs have also been applied in edge-focused tasks on citation networks [18] and recommendation systems [19].

As soon as GNN architectures began to achieve the state-of-the-art in many applica-



tions, the deep learning community began to focus on their theoretical properties. Initially much of the attention was paid to the expressive power of GNNs, that is, the power to distinguish classes of graphs compared to traditional graph-based algorithms. On this regard, the seminal works in [13] and [20] tightly linked the expressive power of GNNs with the so-called *1st order Weisfeiler–Lehman test* (1-WL test), an algorithm for testing if two graphs are *not* isomorphic. The WL test can distinguish large classes of graphs, but sometimes fails on very simple example graphs.

More recently, while the WL test, along with its higher-order extensions [20] has been widely exploited as a baseline for comparisons in designing new GNN architectures [21] [22] [23] [24], a lot of attention has been dedicated to two fundamental aspects, traditionally crucial in the field of deep learning: 1) the *approximation power* [25] [26], namely, the ability to approximate functions on graphs; 2) the *generalization capability* [27] [28] [29], namely, the ability of a neural architecture to performs well outside the training data. For instance, the *Vapnik Chervonenkis dimension* (VC dimension) can tell how much a model is prone to overfitting. On the other hand, the *universal approximation property* specifies that a model can approximate any function and, in theory, can be used to implement any application.

1.2 Thesis summary

This thesis is focused on the theoretical analysis of modern message-passing GNNs. The first main contribution is constituted by the analysis of the approximation capability of modern message-passing GNNs. Such GNNs are shown to be universal approximators on static graph domains modulo the *unfolding equivalence* relation, a concept tightly connected to the 1-WL test. Moreover, it is proved that two graphs are equivalent by comparing the collection of the *unfolding trees* originating from their nodes if and only if they are equivalent also based on the collection of the *colors* of the same nodes obtained by running the 1-WL test. This equivalence connects the literature results on the expressive power of GNNs with the results shown in this thesis on the approximation universality. Proving universal approximation ability of GNNs involve the design of an appropriate coding function that is able to collect the information of the graph through the local message-passing mechanism.

In a second contribution, such results are straightforwardly extended to GNNs designed for Static Attributed Undirected Homogeneous Graphs (SAUGHs), with node *and* edge attributes, and to GNNs designed to process *dynamic graphs*; the latter architectures are denoted as *Dynamic Graph Neural Networks* (DGNNs). This extension is carried out by expanding the paradigms of 1-WL test and unfolding trees to these graph domains, proving their correspondent equivalence, and proving the correspondent approximation universality results.

The third part of the thesis is devoted to the investigation of the generalization capabilities of modern message-passing GNNs. First, bounds on the VC dimension are provided for GNNs with *Pfaffian* activation functions. The class of Pfaffian functions, introduced in algebraic geometry, includes the elementary functions, and it is used in the deep learning context to analyze the topological properties of neural networks or

loss surfaces [30] [28]. Moreover, the generalization capabilities of GNNs are also studied through the lens of *identity effect learning*, namely, the capability of a learning architecture to determine whether a sample is made by two identical substructures or not. Identity effect learning is studied in GNNs under two different point of views: on one hand, GNNs are shown to be unable to learn identity effect when training exploits Stochastic Gradient Descent (SGD), over a set of graphs representing two-letter words; on the other hand, the potential of GNNs in learning identity effects is shown through the link with the 1-WL test, over graphs with more complex topologies. This analysis is particularly interesting as it bridges the theory of GNNs with their ability in *cognitive tasks*, essential in real-life applications.

The rest of this section, summarizes the main contributions of the thesis (Section [1.2.1]) and delineates its structure (Section [1.2.2]).

1.2.1 Main contributions of the thesis

The main contributions of the thesis are summarized in the following.

1. New results on the approximation capabilities of GNNs, described in publication P01:
 - We prove that, on connected graphs, modern GNNs, realizing node-focused functions, are capable of approximating, in probability and up to any precision, any measurable function that respects the 1-WL equivalence. Intuitively, this means that GNNs are a kind of universal approximators for functions on the nodes of the graph, modulo the limits enforced by the 1-WL test. Such a result describes the GNN capability for node classification/regression tasks.
 - The presented proof on the approximation capability of GNNs is the most general that we are aware of, since it holds for node-attributed graphs with real feature vectors and for a broad class of GNNs, which includes most of the current models. Moreover, it is assumed that the target function is measurable, which permits the approximation of discontinuous and more complex functions w.r.t. existing results, e.g. [31]. Finally, the proof is based on a technique that allows us to deduce information on the architecture of the GNN that can reach the desired approximation. Such an information cannot be derived with the Stone-Weierstrass theorem, previously used in other works, and includes, for instance, hints on the number of iterations, the number of layers, the dimension of hidden features, and the type of the network to be used to implement the aggregation function.
 - It is shown that, in order to reach any desired approximation accuracy, a single real hidden feature is sufficient, the aggregation network must contain at least one hidden layer, and the GNN must adopt at least $2N - 1$ iterations, namely the GNN must include $2N - 1$ layers, where N is the maximum number of nodes of any graph in the domain. The latter bound on GNN iterations/layers can be surprising because we may expect that N iterations are sufficient to diffuse the information on the whole graph. We will clarify that such a bound is due to the nature of node classification/regression tasks. Actually, N iterations

are sufficient for graph classification/regression tasks, but they are not enough for node-focused tasks, which are more expensive from a computational point of view.

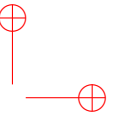
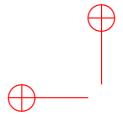
- A set of experiments has been carried out in order to show that GNNs, if their architectures are sufficiently general, can approximate any function, modulo the unfolding equivalence/1-WL test, up to a desired degree of precision, so as suggested by the proposed theoretical results.
2. New results on the approximation capabilities of GNNs over SAUGs and dynamic graphs, described in publication P02:
 - We present new versions of the 1-WL test and of the unfolding equivalence relation appropriate for dynamic graphs and SAUGs with node and edge attributes, and we show that they induce the same equivalences on nodes. Such a result makes it possible to use them interchangeably to study the expressiveness of GNNs.
 - We show that generic GNN models for dynamic graphs and SAUGs with node and edge attributes are capable of approximating, in probability and up to any precision, any measurable function on graphs that respects the 1-WL/unfolding equivalence.
 - The result on approximation capability holds for graphs with unconstrained real feature vectors and target functions. Thus, most of the domains used in practical applications are included. Moreover, our theory is based on space partitioning, which allows us to deduce information about the GNN architecture that can achieve the desired approximation.
 - Numerical experiments are presented over the domain of dynamic graphs to validate our theoretical findings.
 3. New bounds on the VC dimension of modern message-passing GNNs with Pfaffian activation functions, described in publication P03:
 - We provide upper bounds for message-passing GNNs with Pfaffian activation functions with respect to the main hyperparameters, such as the feature dimension, the hidden feature size, the number of message-passing layers implemented and the total number of nodes in the training domain. To demonstrate these results, we exploit the theory of Pfaffian functions and the characterization of the VC dimension of the model via topological analysis.
 - We also study the trend of the VC dimension w.r.t. the colors in the dataset obtained by running the WL test. The theoretical result suggests that the number of colors have an important effect on GNN generalization capability. On one hand, a large total number of colors in training set improves generalization, since it increases the examples available for learning; on the other hand, a large number of colors in each graph raises the VC dimension and therefore it increases the empirical risk value.

- Our theoretical findings are assessed by a preliminary experimental study; Specifically, we evaluate the gap between predictive performance on training data and that on unseen data.
4. Investigation of the generalization limits and capabilities of GNNs when learning identity effects, described in publication P04:
- GNNs are shown to be incapable of learning identity effects via SGD training under sufficient conditions determined by the existence of a suitable transformation τ of the input space; an application to the problem of classifying identical two-letter words is provided and supported by numerical experiments.
 - Conversely, GNNs are shown to be *capable* of learning identity effects in terms of binary classification of *dicyclic graphs*, i.e., graphs composed by two cycles of different or equal length; a numerical investigation of the gap between our theoretical results and practical performance of GNNs is provided.

1.2.2 Structure of the thesis

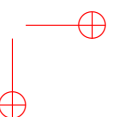
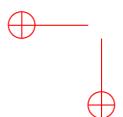
The thesis is organized to guide the reader through the analysis of the theoretical properties of GNNs, first with an overview on their universal approximation ability, then exploring their generalization capacity. In particular, background topics common to each main contribution are presented in Chapter 2, where basic concepts of graph theory, the GNN model in its more general form under the message-passing paradigm, along with an explanation of the 1-WL test, are collected. Chapter 3 provides a detailed review of the relevant literature in the field. Specifically, existing works on approximation universality for GNNs are discussed, and an overview on the literature concerning the analysis of the generalization capability of GNNs, under several point of views, is provided. In Chapter 4, a strict bijection is established between unfolded trees and the 1-WL test; GNNs are shown to be universal approximators on classes of equivalence determined by the 1-WL over the domain of static, node-attributed graphs. Additionally, a characterization of the number of layers required for node-focused tasks is provided. Chapter 5 extends the results presented in Chapter 4 to GNNs working over two larger graph domains, namely SAUHG and dynamic graphs. An appropriate extension of the theoretical concepts of unfolding trees and 1-WL tests is provided on these two graph domains. Chapter 6 introduces some bounds for the VC dimension of GNNs whenever the activation function is Pfaffian, a characterization that includes common activation functions such as arctangent, hyperbolic tangent, and sigmoid. These bounds are stated not only in terms of the common hyperparameters of a GNN, but also with respect to the number of colors deriving from applying the 1-WL test over the dataset. In Chapter 7, the generalization capability of GNNs is investigated under the lens of *identity effect learning*. Two case studies are investigated and experimental results are provided to validate the theoretical findings. Chapter 8 contains a collection of other works carried on during the PhD, not strictly related to the main contribution of this thesis. Contributions of this chapter include: a topological analysis of the loss surface of common neural networks based on Betti Numbers; an empirical analysis of the convergence of different Reservoir Computing (RC)

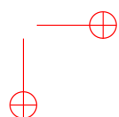
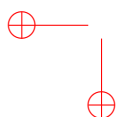
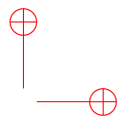
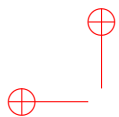
topologies, neural networks characterized by fixed random weights, to correspondent Recurrent Kernel architectures; a parameterization technique for planar domains that exploits Physics Informed Neural Networks, learning models based on the minimization of a Partial Differential Equation in the loss function; a development of an algorithm to analyse gaze movements in the so-called Visual Sequential Search Task to detect chronic or extrapyramidal diseases in patients. Finally, in Chapter [9](#), conclusions are drawn and perspectives for future works are discussed.



Part I

Background topics and literature review





Chapter 2

Background Topics

This section is devoted to introduce the notation used throughout the chapters and the main basic concepts necessary to understand the content of this thesis. If some basic concepts are specific to a single chapter, they will be introduced in the chapter itself. Basic notation is reported in Table 2.1 to help the reader throughout the thesis.

G	graph
V	set of nodes
E	set of edges
v	node
α_v	attribute for node v
q	dimension of input feature space
d	dimension of hidden feature space
o	dimension of output feature space
k	layer index
L	total number of layers
t	timestamp index
$c_v^{(k)}$	1-WL test coloring of node v at iteration k

Table 2.1: Basic notation

2.1 Graph theory: Basic concepts

An *unattributed graph* G can be defined as a pair (V, E) , where V is a set of *nodes* and $E \subseteq V \times V$ is a set of *edges* between nodes. The *number of nodes* of a graph G is denoted by $N := |V|$. Such a graph can be defined by its *adjacency matrix* $\mathbf{A} \in \{0, 1\}^{N \times N}$, where $\mathbf{A}_{ij} = 1$ if $e_{ij} = (i, j) \in E$, otherwise $\mathbf{A}_{ij} = 0$. The *neighborhood* of a node v is defined as $\text{ne}[v] = \{u \in V | (u, v) \in E\}$. A graph G is said to be *undirected* if its adjacency matrix is symmetric, *directed* otherwise. A graph can be said to be *node-attributed* or *labeled* if there exists a map $\alpha : V \rightarrow \mathbb{R}^q$, which assigns to every $v \in V$ a *node feature* (or *vertex feature* or *label*) vector $\alpha(v) \in \mathbb{R}^q$. Sometimes a label will be denoted as α_v for ease of reading. Therefore, a labeled graph G can be defined by a triplet $G = (V, E, \alpha)$. All node features of a labeled graph can be stacked in a *feature matrix* $\mathbf{L}_G \in \mathbb{R}^{N \times q}$. [We report here the definition of Graph Isomorphism, a concept that will be crucial for our analysis in the next chapters.](#)

Definition 2.1 (Graph Isomorphism). Let G_1 and G_2 be two static graphs, then $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are **isomorphic** to each other $G_1 \approx G_2$, if and only if there exists a bijective function $\phi : V_1 \rightarrow V_2$, with

1. $v_1 \in V_1 \Leftrightarrow \phi(v_1) \in V_2 \quad \forall v_1 \in V_1$,
2. $\{v_1, v_2\} \in E_1 \Leftrightarrow \{\phi(v_1), \phi(v_2)\} \in E_2 \quad \forall \{v_1, v_2\} \in E_1$

Graph isomorphism (GI) gained prominence in the graph theory community when it emerged as one of the few natural problems in the complexity class NP that could neither be classified as being hard (NP-complete) nor shown to be solvable with an efficient algorithm (that is, a polynomial-time algorithm) [32]. Indeed it lies in the class NP-Intermediate.

In Chapter 5, different domains of graphs, namely Static Attributed Undirected Homogeneous Graphs (SAUGHs) and dynamic graphs, will be introduced. Unless otherwise specified, we will always deal with finite labeled undirected graphs.

2.2 The color refinement algorithm and the Weisfeiler–Lehman test

The *first order Weisfeiler–Lehman test* (*1-WL test* in short) [33] is a method to test whether two graphs are isomorphic, based on a graph coloring algorithm, called *color refinement*. The coloring algorithm is applied in parallel on the two graphs. Each node keeps a state (or color) that gets refined in each iteration by aggregating information from its neighbors' state. The refinement stabilizes after a few iterations and it outputs a representation of the graph. Two graphs with different representations, i.e. with a different number of nodes for each color, are not isomorphic. Conversely, if the number of nodes sharing the same color matches for the two graphs, then the graphs are *possibly* isomorphic. Note that the test is not conclusive in the case of a positive answer, as the graphs may still be non-isomorphic. Actually, the algorithm just provides an approximate solution to the problem of graph isomorphism.

There exist different versions of the coloring algorithm: in this thesis, we adopt a coloring scheme in which also the node labels are considered. Since GNNs take into account both the structure and labels of graphs, it is useful to consider both these sources of information, in order to analyse the GNN expressive power. Such an approach has been used, for example, in [34]. More precisely, the coloring is carried out by an iterative algorithm which, at each iteration, computes a *node coloring* $c_i^{(k)} \in \Sigma$, being Σ a subset of the values representing the colors. The node colors are initialized on the basis of the node features and then updated using the coloring from the previous iteration. The algorithm is sketched in the following.

1. At iteration 0, we set

$$c_v^{(0)} = \text{HASH}_0(\alpha_v)$$

where $\text{HASH}_0 : \mathbb{R}^q \rightarrow \Sigma$ is a function that bijectively encodes real features using colors. In case of unattributed graphs, we assume $q = 1$ and $\alpha_v = 1, \forall v \in V$ and $\forall G = (V, E) \in \mathcal{G}$, being \mathcal{G} the entire set of graphs to be considered.

2. For $k \geq 0$, we set

$$c_v^{(k+1)} = \text{HASH}(c_v^{(k)}, \{\{c_u^{(k)} \mid u \in \text{ne}[v]\}\})$$

where $\text{HASH} : \Sigma \times \Sigma^* \rightarrow \Sigma$ is a function that bijectively maps the input pairs to a unique value in Σ . The notation $\{\{\cdot\}\}$ represent a *multiset*. Moreover, we assume that the same HASH function is used for all the iterations¹

In order to compare two graphs $G' = (V', E')$, $G'' = (V'', E'')$, the coloring refinement is applied in parallel on G' and G'' , and, at each step, the color profiles generated on each graph are compared, namely, $\{\{c_n^{(k)} \mid n \in V'\}\} = \{\{c_m^{(k)} \mid m \in V''\}\}$ is evaluated. If, at any iteration, the colors of the graphs are different, then the 1-WL test fails and we can conclude that the graphs are not isomorphic; otherwise, the test succeeds. Indeed, if the color profiles agree, G' and G'' may or may not be isomorphic.

In this thesis we use the color refinement also to compare nodes. Thus, given two nodes u, v , which in the most general case can belong to different graphs, we compare their colors at each iteration, i.e., $c_u^k = c_v^k$. If, at any iteration, the node colors are different, then the 1-WL node test fails, otherwise it succeeds. Notice that the color of a node n at iteration k depends on the sub-graph G_n^k , defined by the k -hop neighbourhood of n . Thus, intuitively, the 1-WL node test allows to check the isomorphism of the neighbourhoods of two nodes, $G_u^k \sim G_v^k$.

2.3 Graph Neural Networks

Graph Neural Networks adopt a local computational mechanism to process graphs. The information related to a node v is stored into a feature vector $\mathbf{h}_v \in \mathbb{R}^d$, which is updated recursively by combining the feature vectors of neighboring nodes. After k iterations, the feature vector \mathbf{h}_v^k is supposed to contain a representation of both the structural information and the node information within a k -hop neighborhood. Once processing is complete, the node feature vectors can be used to classify the nodes or the entire graph.

More rigorously, in the following Chapters we consider GNNs that use the following general updating scheme:

$$\mathbf{h}_v^{(k+1)} = \text{COMBINE}^{(k+1)}(\mathbf{h}_v^k, \text{AGGREGATE}^{(k)}(\{\{\mathbf{h}_u^k, u \in \text{ne}[v]\}\})) \quad (2.1)$$

where the node feature vectors are initialized with the node labels, i.e., $\mathbf{h}_v^0 = \alpha_v \in \mathbb{R}^q$ for each v . Here, differently from other approaches, we assume that labels can contain real numbers. Moreover, $\text{AGGREGATE}^{(k+1)}$ is a function which aggregates the neighboring node features obtained in the (k) -th iteration, and $\text{COMBINE}^{(k+1)}$ is a function that combines the aggregation of the neighborhood of a node with its feature at the (k) -th iteration. In graph classification/regression tasks, the GNN is provided with a final

¹In [35], it is assumed that the HASH functions are different at each step, so that the algorithm can reuse the same finite set of colors, e.g., denoted by the integer numbers 1 to N , where N is the number of nodes in the graph. This can be achieved by bijectively re-mapping the colors after each refinement step. The two algorithms are equivalent w.r.t. the goal of isomorphism testing. Here, we prefer to adopt a unique HASH function to simplify our discussion on the properties of the algorithm.

READOUT layer that produces the output $\mathbf{o} \in \mathbb{R}^o$, combining all the feature vectors at the last iteration L :

$$\mathbf{o} = \text{READOUT}(\{\mathbf{h}_v^L, v \in V\}) \quad (2.2)$$

whereas, in node classification/regression tasks, the READOUT layer produces an output for each node, based on its features:

$$\mathbf{o}_v = \text{READOUT}(\mathbf{h}_v^L) \quad (2.3)$$

In this thesis, we will focus on both graph and node classification/regression tasks.

The framework described by Eqs. (2.1)–(2.3) is commonly used to study theoretical properties of modern GNNs (see, e.g., [13]). The class of models included in such framework is rather wide and includes, for example, GraphSAGE [12], GCN [36], GAT [11], GIN [13], ID-GNN [24], and GSN [23].

It is worth mentioning that the the original GNN (OGNN) model [2] is not formally covered, both because in OGNNs the input of $\text{AGGREGATE}^{(k)}$ and $\text{COMBINE}^{(k)}$ contains the node labels α_v and possibly also the edge features, and because the node features are not initialized to α_v . Other models, such as MPNN [9], NN4G [3] and GN [37] are not included as well for similar reasons. Of course, Eq. (2.2) could easily be extended to include also OGNNs and the models mentioned above, but here we prefer not to complicate the proposed framework to keep the notation and proofs simple.

The updating scheme we choose as a reference for the analysis carried on in Chapter 6 and 7 follows [20].

The hidden feature vector $\mathbf{h}_v^{(k+1)} \in \mathbb{R}^d$ of a node v at the message-passing iteration $k+1$, for $k = 1, \dots, L-1$, is defined as

$$\mathbf{h}_v^{(k+1)} = \sigma(\mathbf{W}_{\text{comb}}^{(k+1)} \mathbf{h}_v^{(k)} + \mathbf{W}_{\text{agg}}^{(k+1)} \mathbf{h}_{\text{ne}[v]}^{(k)} + \mathbf{b}^{(k+1)}), \quad (2.4)$$

where $\mathbf{h}_{\text{ne}[v]}^{(k)} = \text{POOL}\{\{\mathbf{h}_u^{(k)} \mid u \in \text{ne}[v]\}\}$, $\sigma: \mathbb{R}^d \rightarrow \mathbb{R}^d$ is an element-wise activation function and POOL is the aggregating operator on the neighbor node's features. The aggregating operator can be defined as a non-learnable function, such as the sum, the mean or the minimum, across the hidden features of the neighbors. With respect to Eq. (2.1), we have that $\text{AGGREGATE}^{(k)}(\cdot) = \text{POOL}(\cdot)$, $\forall k = 1, \dots, L$, while $\text{COMBINE}^{(k+1)}(\mathbf{h}_v, \mathbf{h}_{\text{ne}[v]}) = \sigma(\mathbf{W}_{\text{comb}}^{(k+1)} \mathbf{h}_v + \mathbf{W}_{\text{agg}}^{(k+1)} \mathbf{h}_{\text{ne}[v]} + \mathbf{b}^{(k+1)})$.

In this case, the READOUT for graph classification tasks has been defined as

$$\text{READOUT}(\{\mathbf{h}_v^{(L)} \mid v \in V\}) := f\left(\sum_{v \in V} \mathbf{h}_v^{(L)} \mathbf{w} + \mathbf{b}\right) \quad (2.5)$$

For each node v , the hidden state is initialized as $\mathbf{h}_v^{(0)} = \alpha_v \in \mathbb{R}^q$

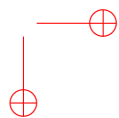
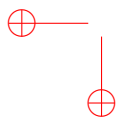
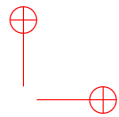
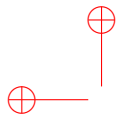
The learnable parameters of the GNN can be summarized as

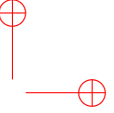
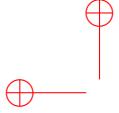
$$\Theta := (\mathbf{W}_{\text{comb}}^{(0)}, \mathbf{W}_{\text{agg}}^{(0)}, \mathbf{b}^{(0)}, \mathbf{W}_{\text{comb}}^{(1)}, \mathbf{W}_{\text{agg}}^{(1)}, \mathbf{b}^{(1)}, \dots, \mathbf{W}_{\text{comb}}^{(L)}, \mathbf{W}_{\text{agg}}^{(L)}, \mathbf{b}^{(L)}, \mathbf{w}, b),$$

with $\mathbf{W}_{\text{comb}}^{(0)}, \mathbf{W}_{\text{agg}}^{(0)} \in \mathbb{R}^{q \times d}$, $\mathbf{W}_{\text{comb}}^{(k)}, \mathbf{W}_{\text{agg}}^{(k)} \in \mathbb{R}^{d \times d}$, for $k = 1, \dots, L$, and $\mathbf{b}^{(k)} \in \mathbb{R}^d$, for $k = 0, \dots, L$; moreover, $\mathbf{w} \in \mathbb{R}^{d \times o}$ and $b \in \mathbb{R}^o$.

This model has been proven to match the expressive power of the Weisfeiler–Lehman test [20] and can, therefore, be considered a good representative model of the message-passing GNN class. Specifically, the characterization reported below states the equivalence, on a node coloring level, referring to the particular model defined in (2.4).

Theorem 2.2 (See [20, Thm 2]). *Let $G = (V, E, \alpha)$ be a graph with initial coloring $c^{(0)}(v) \in \mathbb{R}$ for each node $v \in V$ (so that $c^{(0)} \in \mathbb{R}^{|V(G)|}$). Then, for all $k \geq 0$, there exists a GNN of the form (2.4) such that the hidden feature vector $\mathbf{H}^{(k)} \in \mathbb{R}^{|V(G)|}$ produced by the GNN at layer k coincides with the color vector $\mathbf{C}^{(k)} \in \mathbb{R}^{|V(G)|}$ produced by the 1-WL test at iteration t , i.e., $\mathbf{C}^{(k)} \equiv \mathbf{H}^{(k)}$.*





Chapter 3

Theory of Graph Neural Networks: A literature review

In this chapter we provide an overview of the literature on the mathematical foundations of neural networks, more specifically GNNs. We will focus on works that have studied its approximation power and generalization capabilities.

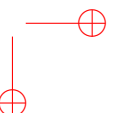
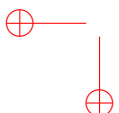
3.1 Neural networks as universal approximators

Since the very first introduction of neural networks, researchers have been focused on the evaluation of their approximation power. The seminal work of [38] and [39] showed that MLPs with one or more hidden layers are universal approximators on any measurable function and up to any degree of precision, provided that a sufficient number of neurons is chosen; in [40] some bounds are provided on the approximation error that depend on the number of neurons in the hidden layer. An overview on the results and techniques present in the literature can be found in [41], [42].

3.1.1 The approximation power of GNNs

In [1], the approximation capability of the original GNN model (OGNN), namely the first GNN model to be proposed, has been studied using the concept of unfolding trees and unfolding equivalence. The unfolding tree T_v , with root node v , is constructed by unrolling the graph starting from v (see Fig. 3.1). Intuitively, T_v exactly describes the information used by the GNN at node v and can be employed to study the expressive power of GNNs in node classification/regression tasks. The unfolding equivalence is, in turn, an equivalence relationship defined between nodes having the same unfolding tree. In [1], it was proved that OGNNs can approximate in probability, up to any degree of precision, any measurable function $\tau : (\mathcal{G} \times \mathcal{V}) \rightarrow \mathbb{R}^o$ — where \mathcal{V} is the set of all nodes of all graphs in \mathcal{G} —, that respects the unfolding equivalence, namely, that produces the same outputs on equivalent nodes. Currently, unfolding trees — also termed *computation graphs* [43] — are widely used to study the GNN expressiveness. Universal approximation results have been proved for Linear Graph Neural Networks [26, 44], Folklore Graph Neural Networks [45] and, more generally, for a large class of GNNs [13, 26] that includes most of the recent architectures, also considered in this chapter.

Despite many advances in research on approximation theory for GNNs, there are still open problems to be investigated. First of all, the most general results available on modern GNNs are based on the Stone–Weierstrass theorem and state that the functions which can



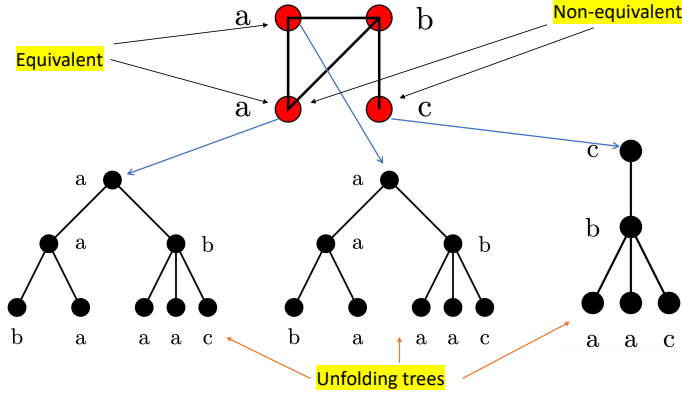


Figure 3.1: An example of a graph with some unfolding trees. The symbols outside the nodes represent features. The two nodes on the left part of the graph are equivalent and have equivalent unfolding trees.

be approximated by GNNs are dense in the invariant continuous function space, modulo the 1-WL test [26]. However, the Stone–Weierstrass theorem is existential in nature, so that, given a target function to be approximated, it does not allow to construct a GNN architecture that can reach the desired approximation — defining, for example, the number of its layers, and the feature dimension required to build the approximator. Moreover, the current results apply only to continuous functions on node/edge labels, which are defined on a compact subset of \mathbb{R}^q , a fact that may not hold in practical application domains, since, for instance, the function to be approximated may show step-wise behavior with respect to some inputs. Finally, all the results on the expressive capacity of modern GNN models are dedicated to graph classification/regression tasks, but node classification/regression problems are also widely present in practical applications and it is important to generalize the theoretical results on expressivity to them as well. In addition, it is useful to study the relationships between unfolding trees and the 1-WL test in this context. Indeed, it can be observed that the Weisfeiler–Lehman test assigns a color to all the nodes of a graph to make them distinguishable, and it can be naturally expected that the equivalence classes defined by the colors are related to those defined by the unfolding trees. In fact, it has been proved that the two mechanisms, colors or unfolding trees, produce the same profiles for graphs [46], namely the same number of nodes per equivalence class, but whether they produces exactly the same profiles with respect to single nodes, i.e., nodes get assigned the same equivalence class, is still an open problem. A formal and precise answer to this question will allow us to use the two frameworks in a targeted or exchangeable way in the context of node classification/regression tasks.

3.2 Generalization capabilities of neural networks

The study of the generalization capabilities of neural networks has always been crucial for the development of efficient learning algorithms. In recent decades, several complexity measures have been proposed to establish reliable generalization bounds, such as Vapnik–Chervonenkis (VC) dimension [47], Rademacher complexity [48, 49], and Betti numbers [30].

3.2.1 GNN generalization properties

Several approaches have been exploited in order to give some insights on the generalization abilities of GNNs. In [43], this issue has been investigated by providing new bounds on the *Rademacher complexity* in binary classification tasks; the study is carried out by focusing on the computation trees of the nodes, which are tightly linked to the 1-WL test [46, 50]. Similarly, in [51], the authors derive generalization bounds based on Transductive Rademacher Complexity, which differs from the standard Rademacher Complexity by taking into account also unobserved instances. In [52], it is proven that the stability and, consequently, the generalization capabilities of Graph Convolutional Networks (GCNs) depend on the largest eigenvalue of the convolutional filter; therefore, to ensure a better generalization ability, the eigenvalue should be independent of the graph size. Under the lens of the PAC-learnability framework, the generalization bounds presented in [43] have been improved in [53], showing a tighter dependency on the maximum node degree and the spectral norm for the weights. This result aligns with the findings in [52]. In [54], the authors provide sharper bounds on the GNNs robustness to noise by investigating the correlation between attention and generalization in GNNs: specifically, GCNs and Graph Isomorphism Networks (GINs) are considered. The results show a link between the trace of the Hessians of the weight matrices and the stability of GNNs. A correlation between attention and generalization in GCNs and GINs is empirically investigated in [27]. In [28], bounds on the VC dimension of the earliest GNN model with Pfaffian activation function are provided as well. Our work extends those results to generic GNNs of the form (2.1). The authors of [51] provide bounds on the VC dimension of GCNs for linear and ReLU activation functions.

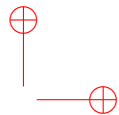
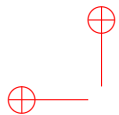
Our results are particularly related to the work in [55] where bounds for the VC dimension of modern GNNs are studied, when the activation function is a piecewise linear polynomial function. Bounds are derived also in terms of the number of colors computed by the 1-order WL test on the graph domain.

Nevertheless, aside from [28], all the aforementioned works focus solely on specific GNN models with piecewise polynomials activation functions, leaving out all GNN models with other common activation functions as arctangent, hyperbolic tangent or sigmoid.

3.2.2 Neural networks and cognitive tasks: the *identity effect learning case*

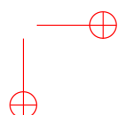
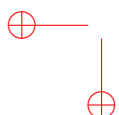
An alternative approach for assessing the generalization capabilities of neural networks is based on investigating their ability to learn specific *cognitive tasks* [56, 57, 58], which

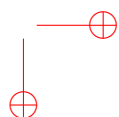
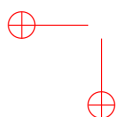
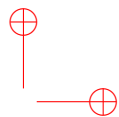
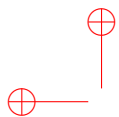
have long been of primary interest as neural networks were originally designed to emulate functional brain activities. Among the various cognitive tasks, the linguistic community has shown particular interest in the study of the so-called *identity effects*, i.e. the task of determining whether objects are made up of two identical components or not [59, 60]. To provide a simple but illustrative example, we can consider an experiment in which the words AA, BB, CC are assigned to the label “good”, while AB, BC, AC are labelled as “bad”. Now, imagine a scenario where a subject is presented with new test words, such as XX or XY. Thanks to the human ability of abstraction, the subject will be immediately able to classify the new words correctly, even though the letters X and Y were not part of the training set. The identity effect learning finds other examples in, for instance, *reduplication* (which happens when words are inflected by repeating all or a portion of the word) [61] or *contrastive reduplication* [62]. Besides their relevance in linguistics, the analysis of identity effects can serve as an intuitive and effective tool to evaluate the generalization capabilities of neural networks in a variety of specific tasks. These tasks encompass the identification of equal patterns in natural language processing [63] as well as molecule classification or regression [5]. In the context of molecule analysis, the exploitation of molecular symmetries as in the class of *bicyclic compounds* [64] plays a crucial role as it can be exploited to retrieve molecular orientations [65] or to determine properties of molecular positioning [66]. Furthermore, the existence of different symmetries in interacting molecules can lead to different reactions. Recently, it has been shown in [67] that Multilayer Perceptrons (MLPs) and Recurrent Neural Networks (RNNs) cannot learn identity effects via Stochastic Gradient Descent nor Adam, under certain conditions on the encoding utilized to represent the components of the objects.

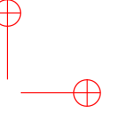
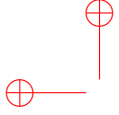


Part II

Approximation capabilities of Graph Neural Networks







Chapter 4

Universality of GNNs for node-attributed graphs

Approximation capabilities of GNNs are among the most essential theoretical properties to be analyzed in order to describe which kind of functions, and up to which degree of precision, these models are able to approximate, given a specific graph domain. In this chapter, we present an alternative approach to study the approximation capability of recent GNNs by the point of view of node-focused tasks. The chapter is organized as follows. Notation and basic concepts are introduced in Section 4.1, while Section 4.2 presents the main contribution of this Chapter. In Section 4.3, we present the experiments conducted to validate our theoretical results.

4.1 Preliminaries

In this section, we introduce the notation and the basic definitions required for the understanding of this chapter.

4.1.1 Unfolding trees and unfolding equivalence

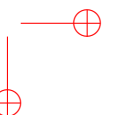
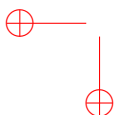
Unfolding trees [1] and *unfolding equivalence* are two concepts that have been introduced in [1] with the aim of capturing the expressive power of the OGNN model. Intuitively, an *unfolding tree* T_v^k is the tree obtained by unfolding the graph up to the depth k , using the node v as its root. Fig. 3.1 shows some examples of unfolding trees. In the following, a formal recursive definition is provided.

Definition 4.1. The unfolding tree T_v^k of a node v up to depth k is

$$T_v^k = \begin{cases} \text{Tree}(\alpha_v) & \text{if } k = 0 \\ \text{Tree}(\alpha_v, T_{\text{ne}[v]}^{k-1}) & \text{if } k > 0 \end{cases}$$

where $\text{Tree}(\alpha_v)$ is a tree constituted of a single node with label α_v and $\text{Tree}(\alpha_v, T_{\text{ne}[v]}^{k-1})$ is the tree with the root node labeled with α_v and having sub-trees $T_{\text{ne}[v]}^{k-1}$. The set $T_{\text{ne}[v]}^{k-1} = \{T_{u_1}^{k-1}, T_{u_2}^{k-1}, \dots\}$ collects all unfolding trees having depth $k-1$, with $u_i \in \text{ne}[v]$, $\forall i$. Moreover, the *unfolding tree of v* , $T_v = \lim_{k \rightarrow \infty} T_v^k$, is obtained by merging all unfolding trees T_v^k for any k .

¹Unfolding trees are also referred to as *computational graphs* [43] or *search trees* [34, 13].



Note that, since a GNN adopts a local computation framework, its knowledge about the graph is updated step by step, every time Eq. (2.1) is applied. Actually, at the first step, $k = 0$, the feature vectors \mathbf{h}_v^0 depends only on the local label, while, at step k , the GNN updates the feature vector \mathbf{h}_v^k incorporating the information related to the k -distant neighbourhood of v . Thus, intuitively, the unfolding tree T_v^k describes the information that is theoretically available to the GNN at node v and step k . Such an observation has been used in [1] to study the expressive power of the OGNN model and will be used also in this chapter for the same purpose.

In this context, two questions have been answered.

- (1) Can GNNs compute and store into the node features a coding of the unfolding trees, namely can GNNs store all the theoretically available information?
- (2) Since unfolding trees are different from the input graphs, how does this affect the GNN expressive power?

Regarding the first question, it has been shown that, indeed, both OGNNs and modern GNNs can compute and store in the node features a coding of the unfolding trees, provided that the appropriate network architectures are used in $\text{COMBINE}^{(k)}$ and $\text{AGGREGATE}^{(k)}$ [34, 1, 13]. Regarding question (2), we can easily argue that if two nodes have the same unfolding tree, then GNNs produce the same encoding on those nodes. This fact highlights an evident limitation of the expressive power of GNNs. The unfolding equivalence is a formal tool designed to capture such a limit: it is an equivalence relation that brings together nodes with the same unfolding tree, namely it groups nodes that cannot be distinguished by GNNs.

Definition 4.2. Two nodes v, u are said to be *unfolding equivalent*, $v \sim_{ue} u$, if $T_v = T_u$. Analogously, two graphs G_1, G_2 are said to be *unfolding equivalent*, $G_1 \sim_{ue} G_2$, if there exists a bijection between the nodes of the graphs that respects the partition induced by the unfolding equivalence on the nodes [2].

Since GNNs have to fulfill the unfolding equivalence, also the functions on graphs that they can realize share this limit. In our results on the approximation capability of GNNs, the focus is on functions that preserve the unfolding equivalence. Those functions are general enough except that they produce the same output on equivalent nodes.

4.1.2 The Weisfeiler–Lehman equivalence

In this chapter, we use the color refinement introduced in Chapter 2 also to compare nodes. Thus, given two nodes u, v , which, in the most general case, can belong to different graphs, we compare their colors at each iteration, i.e., we check if $c_v^k = c_u^k$. If, at any iteration, the node colors are different, then the 1-WL node test fails, otherwise it succeeds. Notice that the color of a node v at iteration k depends on the sub-graph G_v^k , defined by the k -hop neighbourhood of v . Thus, intuitively, the 1-WL node test allows to check the isomorphism of the neighbourhoods of two nodes, $G_v^k \sim G_u^k$.

²For the sake of simplicity, and with notation overloading, we adopt the same symbol \sim_{ue} both for the equivalence between graphs and between nodes.

With the mentioned algorithms, we can easily produce a definition of WL-equivalence for graphs and nodes.

Definition 4.3 (WL-equivalence). Two graphs, $G' = (V', E')$ and $G'' = (V'', E'')$, are said to be **WL-equivalent** if they have the same multisets of colors at each iteration of the color refinement algorithm, i.e., $\{\{c_n^{(k)} \mid n \in V'\}\} = \{\{c_m^{(k)} \mid m \in V''\}\}$ for any t . Analogously, two nodes, v and u , are said to be **WL-equivalent**, $v \sim_{WL} u$, if they have the same colors at each step of the color refinement algorithm, i.e., $c_v^{(k)} = c_u^{(k)}$ for any k .

It is interesting to observe that the color refinement procedure must be iterated until a difference in colors is detected between the compared items, either graphs or nodes, or until a maximum number of iterations is reached. It is well known that the color refinement of the common Weisfeiler–Lehman test, defined for graph comparison, can be halted when the node partition defined by colors become stable: if the two graphs share the colors when the stability is reached, then the equality will last forever. More precisely, let $\pi_k(G)$ be the partition of the nodes of G constructed by collecting in the same class the nodes that have the same color at iteration k . It is not difficult to prove that the partitions become finer at each iteration, $\pi_{k-1}(G) \succeq \pi_k(G)$, and that there exists an iteration L at which they become stable, $\pi_{L-1}(G) \equiv \pi_L(G)$. Moreover, it can be proved that $N - 1$, where N is the number of nodes in G , is both an upper and lower bound on the number L of iterations needed to achieve stability [68].

Note that the stability of the node partition does not imply that the colors do not change. Actually, if the colors are not reused, as in our definition, and at least an edge exists, new colors appear at each iteration. Intuitively, this happens because the use, at a node v , of a new color, which has not been considered in the past, causes the algorithm to create new colors for the neighbors of v as well: thus, new colors will be generated forever. This observation can be used to explain why the upper bound on the iterations of the color refining procedure is different in the case of node or graph equivalence. We will see that we must wait for $2N - 1$ iterations before halting the procedure in the former case, whereas, as mentioned above, $N - 1$ iterations are sufficient in the latter.

4.2 Main Results

In this section, the main results of the chapter are presented and discussed.

4.2.1 Unfolding and Weisfeiler–Lehman equivalence

The first proposed result regards the relationship between the unfolding and the Weisfeiler–Lehman equivalence. The following two theorems clarify that the two equivalence relations produce the same partitions of nodes and graphs. Moreover, the correspondence exists also between the intermediate equivalences defined by, respectively, the colors at each iteration of the WL algorithm and the unfolding trees having a corresponding depth. Formally, let us denote by \sim_{ue_k} the unfolding equivalences between nodes and graphs that are defined as in [4.2] but considering unfolding trees of depth k in place of infinite trees. Similarly, let

us denote by \sim_{WL_k} the WL-equivalences that are defined as in [4.3](#), where only the colors of the refinement procedure up to the k -th iteration are considered.

Since both unfolding equivalence and color equivalence have been described using a node-localized recursive definition, it is natural to investigate possible connections between these two relations. Indeed, in the following, we show that they are equivalent on a domain of graphs with node features, i.e. that they define the same relationship between nodes.

Theorem 4.4.

Let $G = (V, E, \alpha)$ be a graph and let $v, u \in V$, with features α_v, α_u . Then, $\forall k \in \mathbb{N}$

$$T_v^k = T_u^k \text{ iff } c_v^{(k)} = c_u^{(k)} \quad (4.1)$$

where $c_v^{(k)}$ and $c_u^{(k)}$ represent the node coloring of v and u at time k , respectively.

Proof. The proof is carried out by induction on t , which represents both the depth of the unfolding trees and the iteration step in the WL colouring.

For $k = 0$, $T_u^0 = \text{Tree}(\alpha_u) = \text{Tree}(\alpha_v) = T_v^0$ if and only if $\alpha_u = \alpha_v$ and $c_u^{(0)} = \text{HASH}_0(\alpha_u) = \text{HASH}_0(\alpha_v) = c_v^{(0)}$. Let us suppose that Eq. [\(4.1\)](#) holds for $k - 1$, and prove that it holds also for k .

(\rightarrow) Assuming that $T_u^k = T_v^k$, we have

$$T_u^{k-1} = T_v^{k-1} \quad (4.2)$$

and

$$\text{Tree}(\alpha_u, T_{\text{ne}[u]}) = \text{Tree}(\alpha_v, T_{\text{ne}[v]}) \quad (4.3)$$

By induction, Eq. [\(4.2\)](#) is true if and only if

$$c_u^{(k-1)} = c_v^{(k-1)} \quad (4.4)$$

Eq. [\(4.3\)](#) implies that $\alpha_u = \alpha_v$ and $T_{\text{ne}[u]} = T_{\text{ne}[v]}$, which means that an ordering on $\text{ne}[u]$ and $\text{ne}[v]$ exists s.t.

$$T_{\text{ne}_i(u)} = T_{\text{ne}_i(v)} \quad \forall i = 1, \dots, |\text{ne}[u]| \quad (4.5)$$

Hence, Eq. [\(4.5\)](#) holds if and only if an ordering on $\text{ne}[u]$ and $\text{ne}[v]$ exists s.t.

$$c_{\text{ne}(u)_i} = c_{\text{ne}(v)_i} \quad \forall i = 1, \dots, |\text{ne}[u]|$$

that is

$$\{\{c_m^{(k-1)} | m \in \text{ne}[u]\}\} = \{\{c_n^{(k-1)} | n \in \text{ne}[v]\}\} \quad (4.6)$$

Putting together Eqs. [\(4.4\)](#) and [\(4.6\)](#), we obtain:

$$\text{HASH}(c_u^{(k-1)}, \{\{c_m^{(k-1)} | m \in \text{ne}[u]\}\}) = \text{HASH}(c_v^{(k-1)}, \{\{c_n^{(k-1)} | n \in \text{ne}[v]\}\})$$

which implies that $c_u^{(k)} = c_v^{(k)}$.

(\leftarrow) The proof of the converse implication follows a similar reasoning, but some different steps are required in order to reconstruct the unfolding equivalence from the equivalence based on the 1-WL test.

Let us assume that $c_u^{(k)} = c_v^{(k)}$; by definition,

$$\text{HASH}(c_u^{(k-1)}, \{\{c_m^{(k-1)} \mid m \in \text{ne}[u]\}\}) = \text{HASH}(c_v^{(k-1)}, \{\{c_n^{(k-1)} \mid n \in \text{ne}[v]\}\}) \quad (4.7)$$

Being the HASH function bijective, Eq. (4.7) implies that:

$$c_u^{(k-1)} = c_v^{(k-1)} \quad (4.8)$$

and

$$\{\{c_m^{(k-1)} \mid m \in \text{ne}[u]\}\} = \{\{c_n^{(k-1)} \mid n \in \text{ne}[v]\}\} \quad (4.9)$$

Eq. (4.8) is true if and only if, by induction,

$$T_u = T_v \quad (4.10)$$

which implies

$$\alpha_u = \alpha_v \quad (4.11)$$

Moreover, Eq. (4.9) means that an ordering on $\text{ne}[u]$ and $\text{ne}[v]$ exists such that

$$c_{\text{ne}(u)_i} = c_{\text{ne}(v)_i} \quad \forall i = 1, \dots, |\text{ne}[u]| \quad (4.12)$$

Instead, by induction, Eq. (4.12) holds if and only if an ordering on $\text{ne}[u]$ and $\text{ne}[v]$ exists so as $T_{\text{ne}_i(u)} = T_{\text{ne}_i(v)}$, $\forall i = 1, \dots, |\text{ne}[u]|$, i.e.,

$$T_{\text{ne}[u]} = T_{\text{ne}[v]} \quad (4.13)$$

Finally, putting together Eqs. (4.11) and (4.13), we obtain

$$\text{Tree}(\alpha_u, T_{\text{ne}[u]}) = \text{Tree}(\alpha_v, T_{\text{ne}[v]})$$

that means $T_u = T_v$.

□

As a consequence, we can derive the following Corollary which is just a rephrasing of Theorem 4.4 in terms of the equivalence notation.

Corollary 4.5.

Let $G = (V, E, \alpha)$ be a labeled graph. Then, for each $v, u \in V$, $v \sim_{ue} u$ holds if and only if $v \sim_{WL} u$ holds. Moreover, for each integer $k \geq 0$, $v \sim_{ue_k} u$ if and only if $v \sim_{WL_k} u$. (The initialization of the colors is based on the initial labeling of the nodes.)

Moreover, again from Theorem 4.4 we can extend the results to the domain of graphs.

Corollary 4.6.

Let G_1, G_2 be two labeled graphs. Then, $G_1 \sim_{ue} G_2$ if and only if $G_1 \sim_{WL} G_2$. Moreover, for each integer $k \geq 0$, $G_1 \sim_{ue^k} G_2$ if and only if $G_1 \sim_{WL^k} G_2$.

Both the unfolding equivalence and the WL equivalence have been described using a recursive definition local to nodes. Figure 4.1 shows an example in which the unfolding trees and the colors of two nodes are iteratively computed: in the example, the colors of the nodes become different when also the unfolding trees differ.

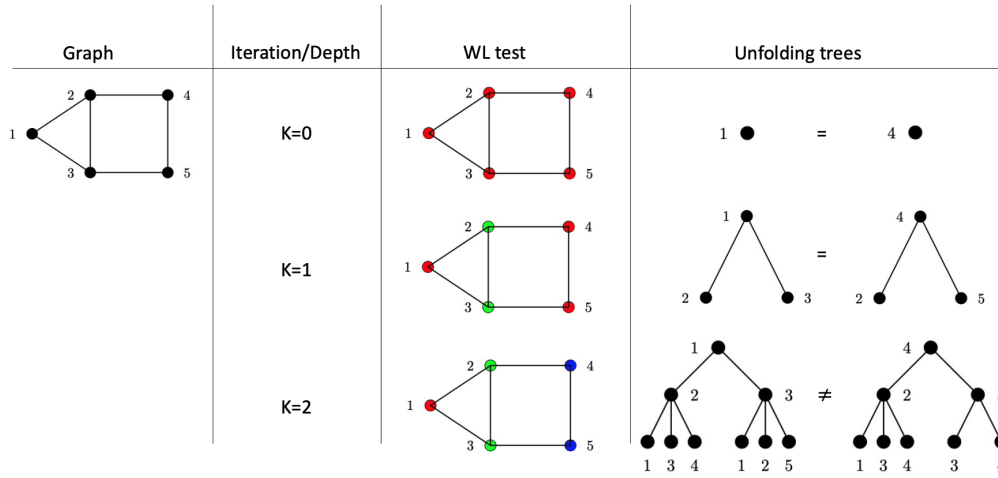


Figure 4.1: A graphical representation of the relationship between the color refinement and the unfolding equivalence, applied on nodes 1 and 4 of the given graph.

Indeed, the existence of a relationship between the two types of equivalence appears to be a natural consequence of their definition. In fact, it is sometimes assumed in the literature (f.i., in [44]) that the two tools can be used interchangeably but, as far as we know, there was no formal demonstration of their effective equivalence. More precisely, in [46, 69, 70], it has been proved that the 1-WL test and unfolding trees produce the same profile on graphs without attributes. Therefore, Corollary 4.6 is just an extension of those results to the case of graphs with attributes. Instead, Corollary 4.5, which focuses on nodes, is completely novel.

Corollaries 4.5 and 4.6 are interesting since they formally confirm that the two equivalences are exactly interchangeable and can be used together to study GNNs. While the Weisfeiler–Lehman test has been often adopted to analyse the expressive power of GNNs in terms of their capability of recognizing different graphs, the unfolding equivalence and, more precisely, unfolding trees, can provide a tool to understand the information that a GNN can use at each node to implement its function.

For example, it is well known that GNNs cannot distinguish regular graphs where nodes have the same features (see e.g. [34]). Of course, in this case, a GNN is not able to distinguish any node, since all the unfolding trees are equal (see Figure 4.2a). Actually, on the one hand, when a target node has different attributes with respect to the others, also the unfolding trees incorporate such a difference and the nodes at different distances

from this target node belong to different equivalence classes (see Figure 4.2b). On the other hand, if all the attributes are different, then each node belongs to a different class, since all unfolding trees are different (see Figure 4.2c).

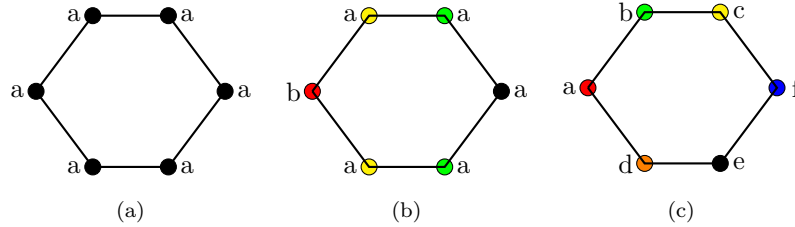


Figure 4.2: (a) A regular graph where all nodes have the same features. All unfolding trees are equal. (b) The equivalence classes when only one node has different features. (c) The equivalence classes when all nodes have different features.

We observe that, in principle, by adding random features to the node labels, we could make all the nodes distinguishable and improve the GNN expressive power. This fact was already mentioned for OGNs [1] and has been recently observed also for modern GNN models [71]. Obviously, this is true only in theory, as the introduction of random features usually produces overfitting. However, some particular tasks exist where random attributes help, for example, see [1] and [34].

A further important issue of our analysis regards how much deep the unfolding trees have to be, i.e., how many iterations of color refinement are needed, in order to make the equivalence stable. Actually, Corollaries 4.5 and 4.6 suggest that the unfolding and Weisfeiler–Lehman equivalences remain paired up to any depth/iteration k . Those equivalences naturally become finer and finer as the iterations proceed, i.e., $\sim_{ue_{k-1}} \succsim \sim_{ue_k}$ and $\sim_{WL_{k-1}} \succsim \sim_{WL_k}$, until L , when they become stable and equal to the corresponding infinite equivalences, namely $\sim_{ue_{k-1}} \equiv \sim_{ue_k} \equiv \sim_{ue}$ and $\sim_{WL_{k-1}} \equiv \sim_{WL_k} \equiv \sim_{WL}$. As already mentioned in Section 4.1, according to the literature [68], it is known that, for the WL equivalence on graphs, $N - 1$ is both an upper and lower bound on L , where N is the maximum number of nodes in the graphs.

Taking inspiration from the results in [68] about covering trees, we are now going to show that, for equivalences *on nodes*, the bounds are different and we must wait up to $2N - 1$ iterations, i.e., trees of depth $2N - 1$, until the equivalences become stable.

In order to prove this, we need first to present the concept of *universal covering*, first introduced in [46], which allows us to derive useful properties on the unfolding trees (see [46] for more details).

Let $G = (V, E)$. Given a graph $H = (V', E', \alpha)$, with α being the labeling function $\alpha : \mathcal{G} \rightarrow \mathbb{R}^q$ and a homomorphism ψ from H to G , if:

- ψ is a bijection from $\text{ne}(v)$ onto $\text{ne}(\psi(v))$
- $\alpha(v) = \alpha(\psi(v))$
- $\alpha(u) = \alpha(\psi(u)) \forall u \in \text{ne}(v)$

for all $v \in V'$, then ψ is called an *attributed covering map* and H is called a *covering graph*. Given a connected graph G and a vertex $x \in V$, let us define a graph $U_x(G)$ as follows. The vertex set of $U_x(G)$ consists of all non-backtracking walks in G starting at x , that is, of sequences (x_0, x_1, \dots, x_m) such that $x_0 = x$, x_i and x_{i+1} are adjacent, and $x_{i+1} \neq x_{i-1}$. Two such walks are *adjacent* in $U_x(G)$ if one of them extends the other by one component, that is, one is $(x_0, \dots, x_m, x_{m+1})$ and the other is (x_0, \dots, x_m) . $U_x(G)$ is a tree and γ_G defined as $\gamma_G(x_0, \dots, x_m, x_{m+1}) = x_{m+1}$ is a covering map from $U_x(G)$ to G . We call U an *attributed universal cover* of G if U covers any covering graph of G . Therefore, $U_x(G)$ is an attributed universal cover of G .

Given that we are dealing with attributed graphs, we will drop the "attributed" adjective from now on, to make the notation lighter.

First of all, we recall a result which is proved in [46], showing the bijective correspondence between universal coverings and colors up to a certain depth/iteration.

Lemma 4.7. [46] *Let U and W be universal covers of graphs G and H , respectively. Furthermore, let ψ be a covering map from U to G and ξ be a covering map from W to H . Let $x \in V(U)$ and $y \in V(W)$, and let $u = \psi(x)$ and $v = \xi(y)$. Then, for any k , $U_x^k \cong W_y^k$ if and only if $c_u^{(k)} = c_v^{(k)}$.*

We observe that we can always identify the node x from a covering W_x of a graph H with its mapping u via ψ ; i.e., $x = u$. This allows us to restate the previous bijective relationship as: $U_u^k \cong W_v^k$ if and only if $c_u^{(k)} = c_v^{(k)}$.

We will now bridge the concepts of universal coverings and unfolding trees, passing through the colour refinement algorithm.

Lemma 4.8. *Let G and H be connected graphs and x, y be nodes of G and H , respectively. Then $T_x^k \cong T_y^k$ if and only if $U_x^k \cong W_y^k$ for all k .*

Proof. Putting together Lemma 4.4 and 4.7, the thesis is obtained straightforwardly. \square

We are now able to state the following result.

Theorem 4.9. *The following statements hold for graphs with at most N nodes.*

1. *Let G and H be connected graphs and x, y be nodes of G and H , respectively. The infinite unfolding trees T_x, T_y are equal if and only if they are equal up to depth $2N - 1$, i.e., $T_x = T_y$ iff $T_x^{2N-1} = T_y^{2N-1}$.*
2. *For any N , there exist two graphs G and H with nodes x, y , respectively, such that the infinite unfolding trees T_x, T_y are different, but they are equal up to depth $2N - 16\sqrt{N}$, i.e., $T_x \neq T_y$ and $T_x^i = T_y^i$ for $i \leq 2N - 16\sqrt{N}$.*

Proof of Theorem 4.9. . The proof is based on the observation that in [46] points (1) and (2) are proved for universal covers, whereas our points refer to unfolding trees. Taking into account Lemma 4.8, we have that universal covers and unfolding trees produce the same isomorphism on nodes. Thus, the thesis follows immediately. \square

In order to get an intuitive explanation of why the bounds to achieve stability are different for graph and node equivalences, let us consider the case of two graphs G and H that are not equivalent, i.e., $G \not\equiv_{WL} H$ holds. Moreover, let us assume that the parallel application of the refinement algorithm detects the difference in colors at iteration \bar{L} , namely $G \not\equiv_{WL_{\bar{L}}} H$, for example because a new color is generated for graph G that is not present in H . At this iteration, the WL algorithm is halted, since we detected at least a node v in G that is different from all the nodes in H . However, if we continue the color refinement, the new color of u will generate other new colors, which are not present in H , also for the neighbors of v . After at most N iterations, the difference spreads throughout the graph, so that, finally, all nodes in G are different from those in H . This is intuitively correct, since all the nodes in G are connected to a node that does not exist in H . Therefore, we can observe that, while the first difference between the nodes of the two graphs arises after $N - 1$ iterations, the diffusion of such information to all the nodes takes additional N steps. Obviously, a similar conclusion can be derived also considering the unfolding equivalence and the depths of the unfolding trees.

An example that illustrates this situation is depicted in Figure 4.3. The two graphs in (a) and (b) have been proposed in 46 and satisfy the lower bound of point (2) of Theorem 4.9. In the example, we assume that all the nodes have the same attributes, even if, for the sake of clarity, they are displayed with different symbols in terms of their "role" in the coloring scheme. The graphs in (a) and (b) are constructed using copies of the subgraph modules in (c), (d) and (e), which are merged in a sequence; (a) and (b) are equal except at the top: in (a), at the end of the sequence, there is a copy of (d), while in (b) there is a copy of (e). It is worth noting that (a) and (b) *do not* satisfy the relation $2N - 16\sqrt{N} > N$; nevertheless, adding multiple times module (c) to the tail of both (a) and (b), we can find two graphs satisfying the requested relation. Indeed the interesting case happens when the sequence is long enough so that $2N - 16\sqrt{N} > N$ holds. In this case, we have the following situation: graphs (a) and (b) are distinguishable by the 1-WL test in less than N steps; nevertheless, a number of steps $k > 2N - 16\sqrt{N} > N$ is needed to distinguish the nodes v and u . Therefore, intuitively, the color refinement approach can recognize that (a) and (b) are not isomorphic, but the detection of their difference occurs only when the information about the asymmetry — which is on one side of the sequence — arrives to the other side of the sequence, where the different modules have been placed. After that, the difference of the two modules is detected and this information is propagated to the rest of the graphs in a number of iterations proportional to the length of the sequences to arrive back to nodes v and u .

In order to formally link the concept of unfolding trees to the computational capability of GNNs, let us now recall the definition of unfolding equivalence.

Definition 4.10. Let $\mathcal{D} = (\mathcal{G} \times \mathcal{V})$, where \mathcal{G} is a set of graphs and \mathcal{V} is a subset of their nodes. A function $\mathbf{f} : \mathcal{D} \rightarrow \mathbb{R}^o$ is said to preserve the unfolding equivalence on \mathcal{D} if $v \sim u$ implies $\mathbf{f}(G, v) = \mathbf{f}(G, u)$.

The class of functions that preserve the unfolding equivalence on \mathcal{D} will be denoted with $\mathcal{F}(\mathcal{D})$. A characterization of $\mathcal{F}(\mathcal{D})$ can be given taking into account 11 (Theorem 1) and Theorem 4.9.

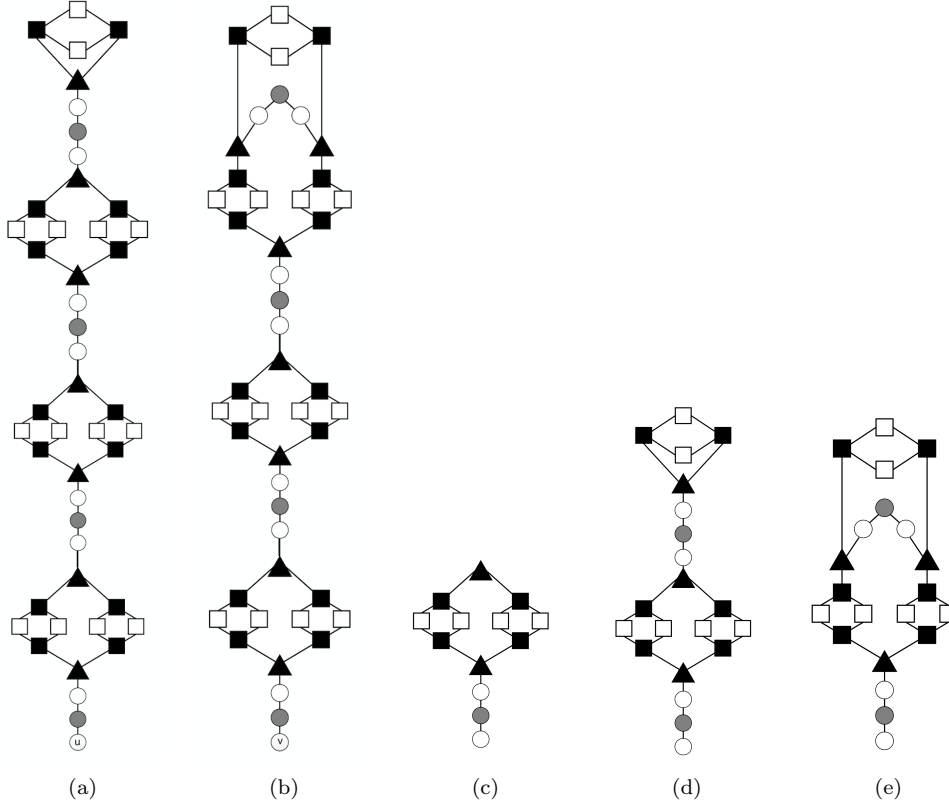


Figure 4.3: In (a) and (b), two graphs G, H are depicted that satisfy the lower bound of point (2) of Theorem 4.9. Graphs in (a) and (b) are constructed by aggregating in a sequence two copies of the same subgraph (c); then, module (d) is added at the top of graph (a), while module (e) is added at the top of graph (b).

Theorem 4.11 (Functions of unfolding trees). *A function \mathbf{f} belongs to $\mathcal{F}(\mathcal{D})$ if and only if there exists a function κ , defined on trees, such that $\mathbf{f}(G, v) = \kappa(T_v^{2N-1})$, for any node $v \in \mathcal{V}$.*

The above theorem represents an improvement of the results reported in [41]; indeed, considering the unfolding tree down to the depth $2N - 1$, we can provide the complete information on a graph to a function \mathbf{f} belonging to $\mathcal{F}(\mathcal{D})$.

Note that Theorem 4.11 suggests not only that the functions that compute the output on a node using unfolding trees preserve the unfolding equivalence, but also that the converse holds, namely all the functions that preserve the unfolding equivalence can be computed as functions of the unfolding trees. Since GNNs can implement only functions of the unfolding trees, we may expect that there is a tight relationship between what GNNs can learn and the class $\mathcal{F}(\mathcal{D})$. Actually, in [1], it has been shown that the OGNN model can approximate in probability, up to any degree of precision, any function in $\mathcal{F}(\mathcal{D})$ and a similar result will be derived for modern GNNs in this chapter.

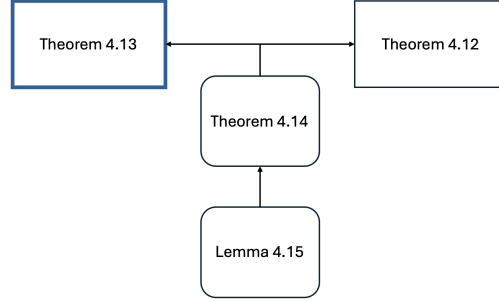


Figure 4.4: Structure of the proof of Theorem 4.13

4.2.2 Approximation capability

So far, we have considered what GNNs cannot do, since we have proved that they are unable to distinguish nodes that originate equal unfolding trees. Another obvious limit is that, at each node v , a GNN considers only the part of the graph that is reachable from v and cannot implement any function depending on the information inaccessible from that node. For this reason, for simplicity, in the following we assume that the graphs are connected.

In this section, we focus our attention on two questions that are related to each other, namely which functions can be approximated by GNNs and if there are any limitations other than that related to the unfolding equivalence. In order to address these questions, we consider the class of functions that preserve the unfolding equivalence (see Definition 4.10). Our aim is to prove that GNNs can approximate in probability, up to any precision, any function of this class, which means that GNNs are a sort of universal approximators on graphs, modulo the limitations due to the unfolding equivalence. This result is stated in Theorem 4.13.

The structure of the proof of Theorem 4.13 summarized in Figure 4.4 is the following:

- We first prove Theorem 4.12 which states that GNNs are deterministically universal approximators on a finite set of graph-node patterns;
- We prove then Theorem 4.14, with the help of the Lemma 4.15, to prove that Theorem 4.12 and Theorem 4.13 are equivalent. Therefore, Theorem 4.13 is straightforwardly obtained.

Let us start by proving Theorem 4.12.

Theorem 4.12. *For any finite set of patterns $\{(G_i, v_i) \mid G_i \in \mathcal{G}, v_i \in V_i, 1 \leq i \leq n\}$, with $N = \max_i |G_i|$ and with graphs having integer features, for any function $\tau : \mathcal{D} \rightarrow \mathbb{R}^o$, which preserves the unfolding equivalence, and for any real $\varepsilon > 0$, there exist continuously differentiable functions $\text{AGGREGATE}^{(k)}$, $\text{COMBINE}^{(k)}$, $\forall k \leq 2N - 1$, s.t.*

$$\mathbf{h}_v^k = \text{COMBINE}^{(k)}(\mathbf{h}_v^{k-1}, \text{AGGREGATE}^{(k)}\{\mathbf{h}_u^{k-1}, u \in \text{ne}[v]\})$$

and a function READOUT, with feature dimension $d = 1$, i.e., $\mathbf{h}_v^k \in \mathbb{R}$, so that the function φ (realized by the GNN), computed after $2N - 1$ steps, satisfies the condition

$$|\tau(G_i, v_i) - \varphi(G_i, v_i)| \leq \varepsilon \quad (4.14)$$

for any i , $1 \leq i \leq n$.

Proof. For the sake of simplicity, the theorem will be proved assuming $o = 1$, i.e. $\tau(G, v) \in \mathbb{R}$. However, the result can be easily extended to the general case when $\tau(G, v) \in \mathbb{R}^o$. Indeed, in this case, the GNN that satisfies the theorem can be defined by stacking o GNNs, each one approximating a component of $\tau(G, v)$.

According to Theorem 4.11, there exists a function κ s.t. $\tau(G, v) = \kappa(T_v)$. Therefore, an unfolding tree of depth $2N - 1$, where N is the maximum number of nodes in the graph domain, is enough to store the graph information, so that κ can be designed to satisfy $\tau(G, v) = \kappa(T_v) = \kappa(T_v^{2N-1})$; moreover, according to Theorem 4.9, the depth of the truncated unfolding tree is enough to respect the unfolding equivalence over all the nodes of every graph in the domain. Consequently, the main idea of the proof consists in designing a GNN that is able to encode the unfolding tree into the node features, i.e., for each node v , we want to have $\mathbf{h}_v = \nabla(T_v^{2N-1})$, where ∇ is an encoding function that maps trees into real numbers. More precisely, the encodings are constructed recursively by AGGREGATE^(k) and COMBINE^(k) functions using the neighbourhood information. After k steps, the node features contain the encoding of the unfolding tree $\nabla(T_v)$ of depth k . Then, after a number of steps L larger than $2N - 1$, the GNN, by the READOUT function, can produce the desired output $\kappa(T_v^L)$.

Accordingly, the theorem can be proved provided that we can implement the above mentioned procedure, which means that there exist appropriate functions ∇ , AGGREGATE^(k), COMBINE^(k) and READOUT. The existence of the READOUT function is obvious, since, given that unfolding trees can be encoded in node features, READOUT has just to decode the representation and compute the target output. Then, let us focus on the other functions. They will be defined in two steps. Initially, AGGREGATE^(k), COMBINE^(k), and READOUT will be defined without taking into account that they have to be continuously differentiable. Later, this farther constraint will be considered.

The coding function ∇

Let ∇ be a composition of any two injective functions μ_∇ and ν_∇ , $\mu_\nabla \circ \nu_\nabla$, with the properties described in the following.

- μ_∇ is an injective function from the domain of the unfolding trees \mathcal{T}^N , calculated on the nodes of the graph G_i , to the Cartesian product $\mathbb{N} \times \mathbb{N}^P \times \mathbb{Z}^{\text{len}(\alpha\mathbf{v})} = \mathbb{N}^{P+1} \times \mathbb{Z}^{\text{len}(\alpha\mathbf{v})}$, where N is the number of nodes of the graph and P is the maximum number of nodes a tree could have.

Intuitively, in the Cartesian product, \mathbb{N} represents the tree structure, \mathbb{N}^P denotes the node numbering, while, for each node, an integer vector $\in \mathbb{Z}^{\text{len}(\alpha\mathbf{v})}$ is used to encode the node features. Note that α exists and is injective, since the maximum information contained in an unfolding tree is given by the union of all its node features and all its structural information, which is exactly equal to the codomain size of α .

- ν_∇ is an injective function from $\mathbb{N}^{P+1} \times \mathbb{Z}^{\text{len}(\alpha v)}$ to \mathbb{R} , whose existence is guaranteed by the cardinality theory, since the two sets have the same cardinality.

Since μ_∇ and ν_∇ are injective, also the existence and the injectiveness of ∇ are ensured.

Functions $\text{AGGREGATE}^{(k)}$ and $\text{COMBINE}^{(k)}$
 Functions $\text{AGGREGATE}^{(k)}$ and $\text{COMBINE}^{(k)}$ must satisfy

$$\begin{aligned} \nabla(T_v) = \mathbf{h}_v &= \text{COMBINE}^{(k)}(\mathbf{h}_v, \text{AGGREGATE}^{(k)}\{\{\mathbf{h}_u, u \in \text{ne}[v]\}\}) \\ &= \text{COMBINE}^{(k)}(\nabla(T_v), \text{AGGREGATE}^{(k)}\{\{\nabla(T_u), u \in \text{ne}[v]\}\}) \end{aligned}$$

$\forall k \leq 2N - 1$, where $2N - 1$ is the number of nodes. In a simple solution, $\text{AGGREGATE}^{(k)}$ decodes the trees of the neighbour T_u of v and stores them into a data structure to be accessed by $\text{COMBINE}^{(k)}$. For example, the trees can be collected into the coding of a new tree, i.e., $\text{AGGREGATE}^{(k)}(\{\{\nabla(T_u)\}, u \in \text{ne}[v]\}) = \nabla(\bigcup_{u \in \text{ne}[v]} \nabla^{-1}(\nabla(T_u)))$, where $\bigcup_{u \in \text{ne}[v]}$ denotes an operator that constructs a tree, with a root having void features, from a set of sub-trees (see Figure 4.5). Then, $\text{COMBINE}^{(k)}$ assigns the correct features to the root by extracting them from T_v , i.e.,

$$\text{COMBINE}^{(k)}(\nabla(T_v), b) = \nabla(\text{ATTACH}(\nabla^{-1}(\nabla(T_v)), \nabla^{-1}(b)))$$

where ATTACH is an operator that construct a tree following the procedure depicted in Figure 4.5 and b is the result of the $\text{AGGREGATE}^{(k)}$ function.

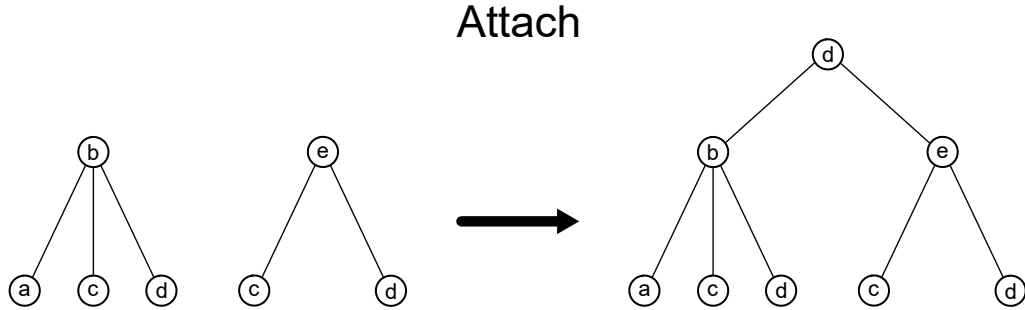


Figure 4.5: The ATTACH operator on trees.

Unfortunately, with this definition, $\text{AGGREGATE}^{(k)}$, $\text{COMBINE}^{(k)}$, and READOUT may not be differentiable. Nevertheless, Eq. (4.14) has to be satisfied only for a finite number of graphs, namely G_i . Thus, we can specify other functions $\overline{\text{AGGREGATE}}^{(k)}$, $\overline{\text{COMBINE}}^{(k)}$, and $\overline{\text{READOUT}}$, which produce exactly the same computations when they are applied on the graphs G_i , but that can be extended to the rest of their domain, so that they are continuously differentiable. Obviously, such an extension exists since those functions are only constrained to interpolate a finite number of points ³. \square

³It is worth noting that a similar extension can also be applied to the coding function ∇ and to the decoding function ∇^{-1} . In this case, the coding function is not injective on the whole domain, but only on the graphs mentioned in the theorem.

We can now state the main result of this chapter.

Theorem 4.13 (Approximation by GNNs). *Let \mathcal{G}_C be a domain containing connected graphs with at most N nodes. For any measurable function $\tau \in \mathcal{F}(\mathcal{D})$, $\mathcal{D} = (\mathcal{G}_C, \mathcal{V})$, preserving the unfolding equivalence, any norm $\|\cdot\|$ on \mathbb{R} , and any probability measure P on \mathcal{D} , there exists a GNN defined by the continuously differentiable functions $\text{COMBINE}^{(k)}$, $\text{AGGREGATE}^{(k)}$, $\forall k \leq 2N - 1$, and by the function READOUT , with feature dimension $d = 1$ (i.e. $\mathbf{h}_v^k \in \mathbb{R}$), such that function φ (realized by the GNN) computed after $2N - 1$ steps satisfies the condition*

$$P(\|\tau(G, v) - \varphi(G, v)\| \leq \epsilon) \geq 1 - \lambda$$

for any reals ϵ, λ , where $\epsilon > 0$, $0 < \lambda < 1$.

The proof is mainly given by proving that Theorem 4.13 and Theorem 4.12 are equivalent. Indeed, by adopting an argument similar to that proposed in [1], it can be shown that Theorem 4.12, where the domain contains a finite number of graphs and the features are integers, is equivalent to Theorem 4.13, which holds in probability for more generic domains of connected graphs.

Theorem 4.14. *Theorem 4.13 holds if and only if Theorem 4.12 holds.*

Proof. Although the proof is quite identical to that contained in [1], for the sake of completeness we report it here with our notation. We need first to recall the following results, whose proof is given in [1] (Lemma 1).

Lemma 4.15. *For any probability measure P on \mathcal{D} , and any reals λ, δ , where $0 < \lambda \leq 1$, $\delta \geq 0$, there exist a real $\bar{b} > 0$, which is independent of δ , a set $\bar{\mathcal{D}} \subseteq \mathcal{D}$, and a finite number of partitions $\bar{\mathcal{D}}_1, \dots, \bar{\mathcal{D}}_l$ of $\bar{\mathcal{D}}$, where $\bar{\mathcal{D}} = \mathcal{G}_i \times \{v_i\}$, with $\mathcal{G}_i \subseteq \mathcal{G}$ and $v_i \in \mathcal{V}_i$, such that:*

1. $P(\bar{\mathcal{D}}) \geq 1 - \lambda$ holds;
2. for each i , all the graphs in \mathcal{G}_i have the same structure, i.e., they differ only for the values of their labels;
3. for each set $\bar{\mathcal{D}}_i$, there exists a hypercube $\mathcal{H}_i \in \mathbb{R}^a$ such that $\alpha_G \in \mathcal{H}_i$ holds for any graph $G \in \mathcal{G}_i$, where α_G denotes the vector obtained by stacking all the feature vectors of G ;
4. for any two different sets $\mathcal{G}_i, \mathcal{G}_j$, $i \neq j$, their graphs have different structures or their hypercubes $\mathcal{H}_i, \mathcal{H}_j$ have a null intersection, i.e. $\mathcal{H}_i \cap \mathcal{H}_j = \emptyset$;
5. for each i and each pair of graphs $G_1, G_2 \in \mathcal{G}_i$, the inequality $\|\alpha_{G_1} - \alpha_{G_2}\|_\infty \leq \delta$ holds;
6. for each graph $G \in \bar{\mathcal{D}}$, the inequality $\|\alpha_G\|_\infty \leq \bar{b}$ holds.

Intuitively, this lemma suggests that a graph domain with continuous features can be partitioned into small subsets so that the features of the graphs are almost constant in

each partition. Moreover, a finite number of partitions is sufficient, in probability, to cover a large part of the domain.

Theorem 4.13 is more general than Theorem 4.12, which makes this implication straightforward. Instead, suppose that Theorem 4.12 holds and show that this implies Theorem 4.13. Let us apply Lemma 4.15 with values for P and λ equal to the corresponding values of Theorem 4.13, being δ any positive real number. Then, there is a real \bar{b} and $\bar{\mathcal{D}} \subset \mathcal{D}$ s.t. $P(\bar{\mathcal{D}}) > 1 - \lambda$. Let \mathcal{M} be the subset of \mathcal{D} that contains only the graphs G satisfying $\|\alpha_G\|_\infty \leq \bar{b}$. Note that, since \bar{b} is independent of δ , then $\bar{\mathcal{D}} \subset \mathcal{M}$ for any δ . Since τ is integrable, there exists a continuous function which approximates τ , in probability, up to any degree of precision. Thus, without loss of generality, we can assume that τ is equi-continuous on \mathcal{M} . By definition of equi-continuity, a real $\bar{\delta} > 0$ exists such that

$$|\tau(G_1, v) - \tau(G_2, v)| \leq \frac{\varepsilon}{2} \quad (4.15)$$

holds for any node v and for any pair of graphs G_1, G_2 having the same structure and satisfying $\|\alpha_{G_1} - \alpha_{G_2}\|_\infty \leq \bar{\delta}$.

Let us apply Lemma 4.15 again, where, now, the δ of the hypothesis is set to $\bar{\delta}$, i.e. $\delta = \bar{\delta}$. From then on, $\bar{\mathcal{D}} = \mathcal{G}_i \times \{v_i\}$, $1 \leq i \leq n$, represents the set obtained by the new application of Lemma 4.15 and $I_i^{\bar{b}, \bar{\eta}}$, $1 \leq i \leq 2d$, denote the corresponding intervals defined in the proof of the same lemma. Let $\theta : \mathbb{R} \rightarrow \mathbb{Z}$ be a function that encodes reals into integers as follows: for any i and any $z \in I_i^{\bar{b}, \bar{\eta}}$, $\theta(z) = i$. Thus, θ assigns to all the values of an interval $I_i^{\bar{b}, \bar{\eta}}$ the index i of the interval itself. Since the intervals do not overlap and are not contiguous, θ can be continuously extended to the entire \mathbb{R} . Moreover, θ can be extended also to vectors, being $\theta(\mathbf{Z})$ the vector of integers obtained by encoding all the components of \mathbf{Z} . Finally, let $\Theta : \mathcal{G} \rightarrow \mathcal{G}$ represent the function that transforms each graph by replacing all the feature labels with their coding, i.e. $\mathbf{L}_{\Theta(G)} = \theta(\mathbf{L}_G)$. Let $\bar{G}_1, \dots, \bar{G}_m$ be graphs, each one extracted from a different set \mathcal{G}_i . Note that, according to points 3., 4., 5. of Lemma 4.15, Θ produces an encoding of the sets \mathcal{G}_i . More precisely, for any two graphs G_1 and G_2 of $\bar{\mathcal{D}}$, we have $\Theta(G_1) = \Theta(G_2)$ if the graphs belong to the same set, i.e., $G_1, G_2 \in \mathcal{G}_i$, while $\Theta(G_1) \neq \Theta(G_2)$ otherwise. Thus, we can define a decoding function Γ s.t. $\Gamma(\Theta(\bar{G}_i), v_i) = (\bar{G}_i, v_i)$, $1 \leq i \leq n$.

Consider, now, the problem of approximating $\tau \circ \Gamma$ on the set $(\Theta(\bar{G}_1), v_1), \dots, (\Theta(\bar{G}_n), v_n)$. Theorem 4.12 can be applied to such a set, because it contains a finite number of graphs with integer labels. Therefore, there exists a GNN that implements a function $\bar{\varphi}$ s.t., for each i ,

$$|\tau(\Gamma(\Theta(\bar{G}_i), v_i)) - \bar{\varphi}(\Theta(\bar{G}_i), v_i)| \leq \frac{\varepsilon}{2} \quad (4.16)$$

However, this means that there is also another GNN that produces the same result operating on the original graphs G_i , namely a GNN for which

$$\varphi(G_i, v_i) = \bar{\varphi}(\Theta(\bar{G}_i), v_i) \quad (4.17)$$

holds. Actually, the graphs G_i and \bar{G}_i are equal except that the former has the coding of the feature labels attached to the nodes, while the latter contains the whole feature

labels. Thus, the GNN that operates on \bar{G}_i is that suggested by Theorem 4.12, except that $\overline{\text{AGGREGATE}}^{(0)}$ preliminarily creates a coding of $\theta(\alpha_v)$.

Putting together equality (4.17) with Eqs. (4.15) and (4.16), it immediately follows that, for any $(G, v) \in \bar{\mathcal{D}}_i$,

$$\begin{aligned} |\tau(G, v) - \varphi(G, v)| &= |\tau(G, v) - \tau(\bar{G}_i, v) + \tau(\bar{G}_i, v) - \varphi(G, v)| \\ &\leq |\tau(\bar{G}_i, v) - \varphi(G, v)| + \frac{\varepsilon}{2} \\ &= |\tau(\Gamma(\Theta(\bar{G}_i), v)) - \bar{\varphi}(\Theta(\bar{G}_i), v)| + \frac{\varepsilon}{2} \leq \varepsilon \end{aligned}$$

Thus, the GNN described by Eq. (4.17) satisfies $|\tau(G, v) - \varphi(G, v)| \leq \varepsilon$ in the restricted domain $\bar{\mathcal{D}}$. Since $P(\bar{\mathcal{D}}) \geq 1 - \lambda$, we have:

$$P(\|\tau(G, v) - \varphi(G, v)\| \leq \varepsilon) \geq 1 - \lambda$$

which proves the theorem. \square

Theorem 4.13 intuitively states that, given a function τ , there exists a GNN that can approximate it. $\text{COMBINE}^{(k)}$ and $\text{AGGREGATE}^{(k)}$ can be any continuously differentiable function, while no assumptions are made on READOUT . This situation does not correspond to practical cases, where the GNN adopts particular architectures and those functions are realized by neural networks or, more generally, parametric models — for example made of layers of sums, max, average, etc. Therefore, it is of fundamental interest to clarify whether the theorem still holds when the components $\text{COMBINE}^{(k)}$, $\text{AGGREGATE}^{(k)}$ and READOUT are parametric models.

Let us now study the case when the employed components are sufficiently general to be able to approximate any function. We call \mathcal{Q} this class of networks, which corresponds to GNN models with universal components. In order to simplify our discussion, we introduce the transition function $\mathbf{f}^{(k)}$ to indicate the stacking of the $\text{AGGREGATE}^{(k)}$ and $\text{COMBINE}^{(k)}$, i.e.,

$$\mathbf{f}^{(k)}(\mathbf{h}_v^k, \{\{\mathbf{h}_u^{k-1}, u \in \text{ne}[v]\}\}) = \text{COMBINE}^{(k)}(\mathbf{h}_v^{k-1}, \text{AGGREGATE}^{(k)}(\{\{\mathbf{h}_u^{k-1}, u \in \text{ne}[v]\}\})).$$

Then, we can formally define the class \mathcal{Q} .

Definition 4.16. A class \mathcal{Q} of GNN models is said to have *universal components* if, for any $\epsilon > 0$ and any continuous target functions $\overline{\text{COMBINE}}^{(k)}$, $\overline{\text{AGGREGATE}}^{(k)}$, $\overline{\text{READOUT}}$, there exists a GNN belonging to \mathcal{Q} , with functions $\text{COMBINE}_w^{(k)}$, $\text{AGGREGATE}_w^{(k)}$, READOUT_w and parameters w such that

$$\begin{aligned} \left\| \bar{\mathbf{f}}^{(k)}(\mathbf{h}, \{\{\mathbf{h}_1, \dots, \mathbf{h}_s\}\}) - \mathbf{f}_w^{(k)}(\mathbf{h}, \{\{\mathbf{h}_1, \dots, \mathbf{h}_s\}\}) \right\|_{\infty} &\leq \epsilon \\ \left\| \overline{\text{READOUT}}(\mathbf{q}) - \text{READOUT}_w(\mathbf{q}) \right\|_{\infty} &\leq \epsilon \end{aligned}$$

holds, for any input values $\mathbf{h}, \mathbf{h}_1, \dots, \mathbf{h}_s, \mathbf{q}$. Here, the transition functions $\bar{\mathbf{f}}^{(k)}$ and $\mathbf{f}_w^{(k)}$ are defined using the target functions $\overline{\text{COMBINE}}^{(k)}$, $\overline{\text{AGGREGATE}}^{(k)}$, and the GNN functions $\text{COMBINE}_w^{(k)}$, $\text{AGGREGATE}_w^{(k)}$, respectively, and $\|\cdot\|_{\infty}$ is the infinity norm.

The following result shows that Theorem 4.13 still holds even for GNNs with universal components.

Theorem 4.17 (Approximation by neural networks). *Let us assume that the hypotheses of Theorem 4.13 are fulfilled and \mathcal{Q} is a class of GNNs with universal components. Then, there exists a parameter set w and some functions $\text{COMBINE}_w^{(k)}$, $\text{AGGREGATE}_w^{(k)}$, READOUT_w , implemented by neural networks in \mathcal{Q} , such that the thesis of Theorem 4.13 holds.*

Proof. As in the proof of Theorem 4.13, without loss of generality, we will assume that the feature dimension is $d = 1$. First of all, note that Theorem 4.13 ensures that we can find $\overline{\text{COMBINE}}^{(k)}$, $\overline{\text{AGGREGATE}}^{(k)}$, $\forall k \leq L$, and $\overline{\text{READOUT}}$ so that, for the corresponding function $\bar{\varphi}$ implemented by the GNN,

$$P(\|\tau(G, v) - \bar{\varphi}(G, v)\| \leq \varepsilon/2) \geq 1 - \lambda \quad (4.18)$$

holds. Let us consider the corresponding transition function $\bar{\mathbf{f}}$, defined by

$$\bar{\mathbf{f}}^k(\mathbf{h}_v^{k-1}, \{\{\mathbf{h}_u^{k-1}, u \in \text{ne}[v]\}\}) = \overline{\text{COMBINE}}^{(k)}(\mathbf{h}_v^{k-1}, \overline{\text{AGGREGATE}}^{(k)}\{\{\mathbf{h}_u^{k-1}, u \in \text{ne}[v]\}\})$$

Since $\overline{\text{COMBINE}}^{(k)}$ and $\overline{\text{AGGREGATE}}^{(k)}$ are continuously differentiable, $\bar{\mathbf{f}}^k$ is continuously differentiable. Considering that the theorem has to hold only in probability, we can also assume that the domain is bounded, so that $\bar{\mathbf{f}}^k$ is bounded and has a bounded Jacobian. Let B be a bound on the Jacobian/derivative of $\bar{\mathbf{f}}^k$ for any k and any input. The same argument can also be applied to the function $\overline{\text{READOUT}}$, which is continuously differentiable w.r.t. its input and can be assumed to have a bounded Jacobian/derivative. Let us assume that B is also a bound for the Jacobian/derivative of $\overline{\text{READOUT}}$. Moreover, let $\text{COMBINE}_w^{(k)}$ and $\text{AGGREGATE}_w^{(k)}$ be functions implemented by universal neural network that approximate $\overline{\text{COMBINE}}^{(k)}$, $\overline{\text{AGGREGATE}}^{(k)}$, $\forall k \leq L$, respectively, and such that

$$\mathbf{f}_w^k(\mathbf{h}_v^{k-1}, \{\{\mathbf{h}_u^{k-1}, u \in \text{ne}[v]\}\}) = \text{COMBINE}_w^{(k)}(\mathbf{h}_v^{k-1}, \text{AGGREGATE}_w^{(k)}\{\{\mathbf{h}_u^{k-1}, u \in \text{ne}[v]\}\})$$

and let us assume that

$$\|\bar{\mathbf{f}}^k - \mathbf{f}_w^k\|_\infty \leq \eta \quad (4.19)$$

holds for every k and a $\eta > 0$. Let READOUT_w be the function implemented by a universal neural network that approximates $\overline{\text{READOUT}}$, so that

$$\|\overline{\text{READOUT}} - \text{READOUT}_w\|_\infty \leq \eta$$

In the following, it will be shown that, when η is sufficiently small, the GNN implemented by the approximating neural networks is sufficiently close to the GNN of Theorem 4.13 so that the thesis is proved.

Let $\bar{\mathbf{F}}^k$, \mathbf{F}_w^k be the global transition functions of the GNNs that are obtained by stacking all the $\bar{\mathbf{f}}^k$ and \mathbf{f}_w^k for all the nodes of the input graph. The node features are computed at each step by $\bar{\mathbf{H}}^k = \bar{\mathbf{F}}^k(\bar{\mathbf{H}}^{k-1})$, $\mathbf{H}^k = \mathbf{F}_w^k(\mathbf{H}^{k-1})$, where $\bar{\mathbf{H}}^k, \mathbf{H}^k$ denote the stacking of all the node features of the graph obtained by the two transition functions, respectively. Then,

$$\|\bar{\mathbf{H}}^1 - \mathbf{H}^1\|_\infty = \|\bar{\mathbf{F}}^1(\mathbf{H}^0) - \mathbf{F}_w^1(\mathbf{H}^0)\|_\infty \leq \eta(2N - 1) \quad (4.20)$$

where $N = |G|$ is number of nodes in the input graph. Moreover,

$$\begin{aligned} \|\bar{\mathbf{H}}^2 - \mathbf{H}^2\|_\infty &= \|\bar{\mathbf{F}}^2(\bar{\mathbf{H}}^1) - \mathbf{F}_w^2(\mathbf{H}^1)\|_\infty \\ &= \|\bar{\mathbf{F}}^2(\bar{\mathbf{H}}^1) - \bar{\mathbf{F}}^2(\mathbf{H}^1) + \bar{\mathbf{F}}^2(\mathbf{H}^1) - \mathbf{F}_w^2(\mathbf{H}^1)\|_\infty \\ &\leq \|\bar{\mathbf{F}}^2(\bar{\mathbf{H}}^1) - \bar{\mathbf{F}}^2(\mathbf{H}^1)\|_\infty + \|\bar{\mathbf{F}}^2(\mathbf{H}^1) - \mathbf{F}_w^2(\mathbf{H}^1)\|_\infty \\ &\leq \eta N B + \eta N = \eta N (B + 1). \end{aligned}$$

Here, $\|\bar{\mathbf{F}}^2(\bar{\mathbf{H}}^1) - \bar{\mathbf{F}}^2(\mathbf{H}^1)\|_\infty \leq \eta N B$ holds because of Eq. (4.20), which bounds the difference between $\bar{\mathbf{H}}^1$ and \mathbf{H}^1 , and due to the fact that the Jacobian/derivative of $\bar{\mathbf{F}}^2$ is bounded by B . Moreover, $\|\bar{\mathbf{F}}^2(\mathbf{H}^1) - \mathbf{F}_w^2(\mathbf{H}^1)\|_\infty \leq \eta N$ holds by Eq. (4.19).

The above reasoning can then be applied recursively to prove that

$$\|\bar{\mathbf{H}}^k - \mathbf{H}_w^k\|_\infty \leq \eta N \sum_{i=0}^{k-1} B^i$$

Since the output of the GNN is computed using the encoding at step L , we have

$$\|\bar{\varphi}(G, v) - \varphi_w(G, v)\|_\infty = \|\overline{\text{READOUT}}(\bar{\mathbf{H}}^L) - \text{READOUT}_w(\mathbf{H}^L)\|_\infty \leq \eta N + B(\eta N \sum_{i=0}^L B^i).$$

Finally, since we can consider the maximum number of nodes N as bounded⁴, then we can find a GNN based on neural networks so that η is small enough to achieve

$$\|\bar{\varphi}(G, v) - \varphi_w(G, v)\|_\infty \leq \epsilon/2$$

which, together with Eq. (4.18), produces the bound of Theorem 4.13. \square

Some topics related to the proof are discussed below, to better understand some properties of GNNs.

- In the proof of Theorem 4.13, we first define an encoding function ∇ (see the Appendix) that maps trees to real numbers. The functions $\text{COMBINE}^{(k)}$ and $\text{AGGREGATE}^{(k)}$ are designed so that, at each step, the node feature vector approximates a coding of the unfolding function $\mathbf{h}_v^k = \nabla(T_v^k)$. The function READOUT decodes the unfolding and produces the desired outputs.
- In the proof of Theorem 4.17, it is shown that Theorem 4.13 still holds even when the transition and READOUT functions are approximated. Thus, we can use any parametric model to implement those functions. We can expect that, also for the GNNs of Theorem 4.17, the transition function stores into the feature vector an approximate coding of the unfolding tree, while READOUT decodes such a coding and gives the desired outputs. Obviously, in a practical case, a GNN can store only useful information, required to produce the output, and not just all the informative content of the unfolding trees.

⁴For the sake of simplicity, we skip over a very formal proof of this claim. Intuitively, note that the theorem has to be proved and Lemma 4.15 clarifies that any graph domain can be covered with high probability by a finite number of structures, which obviously have a bounded number of nodes.

The following remarks may further help to understand our results.

- *GNNs with universal components.* Intuitively, the universality condition means that the architectures used to implement $\mathbf{f}_w^{(k)}$ and READOUT_w must be sufficiently general to be able to approximate any possible target function. From the theory of standard neural networks, those architectures must have at least two layers (one hidden and one output layer) [41]. Such a conclusion is similar to the one reported in [13], where a related result is described and where it is suggested that, in order to be able to implement the 1-WL test, the GNN must use a two layer transition function. Indeed, in this way, the GNN can implement an injective encoding of the input graph into the node features. Nonetheless, the proposed result is slightly different with respect to the one reported in [13] as, in theory, the encoding may fail to be injective, provided that the approximation remains sufficiently good in probability. However, the conclusion about the architecture still holds.

GNNs with transition functions $\mathbf{f}_w^{(k)}$ exploiting two layer architectures include Graph Isomorphism Networks (GINs) [13], which were claimed to realize an injective encoding. Similarly, also the OGNN model, for which a result similar to Theorem 4.13 was proved, adopts a two layer architecture for the transition function: in this case, $\text{AGGREGATE}_w^{(k)}$ consists of a MultiLayer Perceptron (MLP) with a hidden layer and $\text{COMBINE}_w^{(k)}$ was implemented by a sum. Similar results have been devised also in [26], where a different version of the $\text{COMBINE}_w^{(k)}$ function has been modeled as a sum of MLPs.

- *READOUT universality.* The condition on the universality of the READOUT function can be relaxed, provided that a higher dimension for the feature vector is used, namely $o > 1$. READOUT_w can indeed cooperate with the transition function in order to produce the output. In the limit case, the output can be completely prepared by the transition function and stored in some components of \mathbf{h}_v^L so that READOUT_w is just a projection function.
- *GNN architectures that are not universal approximators.* Most of GNN models, e.g. Graph Convolutional Neural Networks, GraphSAGE and so on, use a single layer architecture to implement the transition function. Thus, even if they do employ universal components, such as those specified by Definition 4.16, they have a limited computational power with respect to two layer architectures and this is supported by theoretical results. In [13], Lemma 7, it is shown that, if the transition function is made up by a single layer with ReLU activation functions, the encoding function cannot be injective. A similar result was obtained for linear recursive neural networks [5] in [73]. However, in general, it is not correct to assert that GNNs with single layer transition functions cannot be universal approximators for functions on graphs, as this property depends on the used GNN model and on other architectural/training details. For example, a GNN model with a single layer transition component can use several iterations of Eq. (2.1) to emulate a GNN with

⁵Recursive neural networks [72] are the ancestors of GNNs and assume that the input graph is directed, positional and acyclic.

a deeper transition component. In the former model, the node features emulate the transition network hidden layers and COMBINE must contain a self-loop, namely must have access to the previous features of each node.

- *Feature dimension.* Surprisingly, Theorems 4.13 and 4.17 suggest that a feature vector of dimension $d = 1$ is enough to establish the universal approximation capability of GNNs. It is obvious, however, that the dimension of the feature vector plays an important role in determining the complexity of the coding function for a given domain. We expect that the larger the dimension, the smaller the complexity of the coding. This complexity, in turn, affects the complexity of the transition function, the difficulty in learning such a function, the number of patterns required for training the GNN and so on.
- *Number of steps.* Theorems 4.13 and 4.17 suggest that $2N - 1$ steps are enough to approximate any function. Such a result is a consequence of Theorems 4.9 and 4.11. Intuitively, this bound can be explained reusing the discussion on Theorem 4.9. A GNN can employ up to $N - 1$ iterations/layers to diffuse all the information from one node to any other node with the message passing mechanism. After $N - 1$ iterations, the information stored in a node provides a sort of signature for that node, which may allow to distinguish some nodes from others. Yet, such a signature is not complete, since the first time a node “communicate” with another has no information about itself. Adding N iterations/layers allows nodes to communicate again and exchange their current signatures to produce more accurate signatures. It is worth noting that this reasoning provides also an intuitive explanation about why graph regression/classification tasks differ from node tasks. In graph tasks, the GNN uses a READOUT function that aggregates the features of all the nodes in the graph, and possibly can do the work required by the second diffusion phase. In node tasks, READOUT operates only on a single node, so that the second diffusion phase is mandatory.
- *The same COMBINE and AGGREGATE can be used for all the layers.* Even if, for clarity, in our theoretical analysis, we focus on the GNN model that is the most used and exploits different functions in each layer k , our proofs do not exploit such a characteristic. Therefore, all the results hold also for those GNNs — sometimes called *recursive* — using the same COMBINE and AGGREGATE functions on each layer.

Note that, throughout the chapter, we have used the idea that the unfolding tree represents the information available to a GNN to compute its output, and we have mentioned that a similar approach has been applied by other authors as well. From a formal point of view, Theorem 4.13 defines a method by which a GNN can actually encode an unfolding tree into the node features, so that it has been proved that all the information collected into the unfolding trees can be used by GNNs. However, also the reverse implication holds true, that is a GNN cannot encode more information into features than that contained into the unfolding trees. Indeed, this is a consequence of the fact that

GNNs have no greater discriminatory capability than the 1-WL test (see [20], Theorem 1). Therefore, the unfolding trees totally collect the information used by a GNN.

Consequently, we can provide an alternative way to describe the approximation ability of GNNs as a function of their unfolding trees.

Corollary 4.18. *The class of functions implemented by a GNN with universal components is dense in probability in the $\mathcal{F}(\mathcal{D})$ class of functions that preserve the unfolding equivalence in the domain \mathcal{D} of connected graphs.*

4.3 Experimental Validation

In this section, we support our theoretical findings with a set of experiments. For this purpose, we show that a GNN can approximate a function $F_{WL} : \mathcal{G} \rightarrow \mathbb{N}$ that models the 1-WL test. Indeed, the function F_{WL} assigns to each graph a target label that represents the class of equivalence of the 1-WL. For simplicity, we only focus on the ability of the GNN to approximate this function, so that only the training performance is considered, i.e., we do not investigate its generalization capability over a test set. Since the 1-WL test provides the finest partition of graphs reachable by a GNN, the mentioned task experimentally establishes the expressive power of GNNs.

Dataset The graph dataset used for the experiments is derived from the QM9 molecules' dataset [74, 75]. Specifically, the subset of molecules that compose our dataset are selected as follows:

- Homogeneous features are assigned to all nodes of all graphs in QM9, as we are interested in evaluating the approximation ability of GNNs based only on the graph topology;
- The 1-WL test is run all over the entire QM9 dataset for $k = 4$ iterations, and for each graph, the target is the corresponding 1-WL output, represented as a natural number;
- We select the color classes containing more than `thr` graphs, where `thr` is a fixed threshold.

For training purposes, the targets are normalized between 0 and 1 and spaced uniformly in the range $[0, 1]$. Therefore, the distance between each class label is $d = \frac{1}{\text{numclasses}-1}$. A graph G with target y_G will be said to be correctly classified if, given $\text{out} = \text{GNN}(G)$, we have $|\text{out} - y_G| < d/2$.

Experimental setup The GNN used in the experiments is the Graph Isomorphism Network (GIN) [13]. A GIN computes

$$\mathbf{h}_v^{(k)} = \text{MLP}\left((1 + \epsilon)\mathbf{h}_v^{(k-1)} + \sum_{u \in \text{ne}[v]} \mathbf{h}_u^{(k-1)}\right), \quad (4.21)$$

where the attention parameter ϵ is either a trainable parameter or a fixed scalar. In our setting, we fix $\epsilon = 0$. It has been proven that GINs can implement the 1-WL test and produce a different representation for each graph that can be distinguished in this way [13]. Thus, GINs, with an appropriate READOUT, can approximate any function on graphs preserving the unfolding equivalence. The MLP in a GIN layer has one hidden layer with h_{gin} neurons; the dimension of the GIN features is h_{gin} as well. The MLP in a GIN layer implements the hyperbolic tangent as activation function, and batch normalization. The GIN includes $L_{\text{max}} = k$ layers, so as k is the number of iterations performed by 1-WL to generate the targets. After the last GIN layer, the READOUT function is implemented performing a global aggregation, after which a linear layer $W_{\text{gin_out}}$ of size $h_{\text{gin}} \times 1$ is added; eventually, a sigmoid activation function is applied. The model is trained over 500 epochs using the Adam optimizer with an initial learning rate $\lambda = 10^{-3}$. We carried out the experiments as follows.

- In the first experimental setting, we evaluate the GNN performance for different values of the threshold thr , which affects the cardinality of the training set and its 1-WL color classes. The values of the threshold thr are taken in the integer interval $[30, 45]$, the hidden layer of the MLP has dimension $h_{\text{gin}} = 64$.
- In the second experimental setting, we evaluate the GNN expressive power varying both the number of neurons in the GIN MLP and the size of the hidden features, which, as specified above, are kept equal. In these experiments, the threshold is fixed as $\text{thr} = 35$, the hidden layer sizes h_{gin} are taken from the list $[4, 8, 16, 32, 64]$.

Each experiment is statistically evaluated over 15 runs. The overall training is performed on an Intel(R) Core(TM) i7-9800X processor running at 3.80GHz, using 31GB of RAM and a GeForce GTX 1080 Ti GPU unit [6].

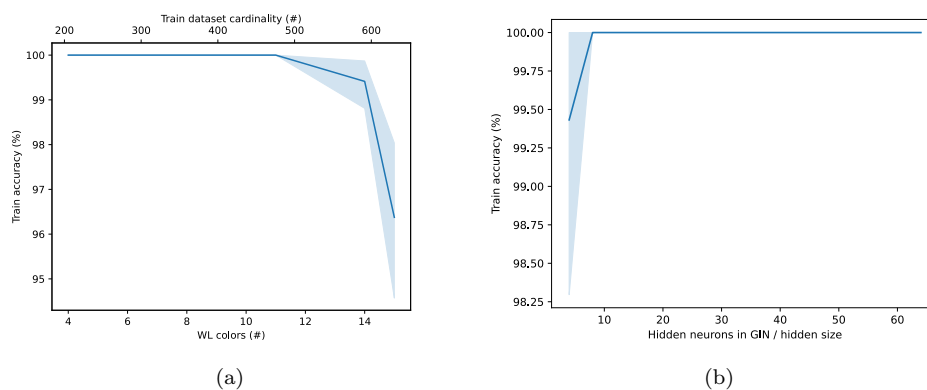
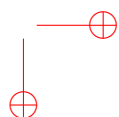
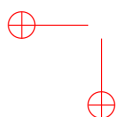
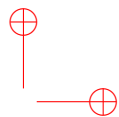
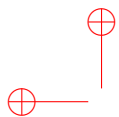


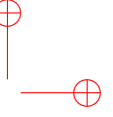
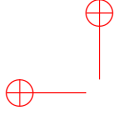
Figure 4.6: Training accuracy on subsampled QM9 dataset, increasing number of WL colors (a), and increasing hidden layer size (b). The solid line represents the average over 15 runs, the shaded area represents the confidence interval.

⁶Code available at <https://github.com/AleDinve/static-gnn>.

Results Our experimental results are summarized in Figure 4.6. Figure 4.6 (a) shows the evolution of the training accuracy for different numbers of WL colors; Figure 4.6 (b) displays the evolution of the training accuracy for increasing numbers of hidden neurons in the GIN MLP. In both experiments the average training accuracy is never less than 96%. Moreover, in at least one of the 15 runs per value, 100% training accuracy is reached. These results confirm the approximation power of GNNs equipped with sufficiently general components.

In the next chapter we will see how to extend the results presented in the current chapter on the approximation power of GNNs, suited for node-attributed graphs, to graph learning models that process different domains of graphs.





Chapter 5

Universality of GNNs for SAUHG and Dynamic GNNs

In this chapter, we propose a study on the expressive power of GNNs for two domains of particular interest, namely dynamic graphs and static attributed undirected homogeneous graphs (SAUHGs) with node and edge attributes. Dynamic graphs are interesting from a practical and a theoretical point of view and are used in several applications [76]. Moreover, dynamic GNN models are structurally different from GNNs for static graphs, and the results and methodology required to analyze their expressive power cannot directly be deduced from existing literature. On the other hand, SAUHGs with node and edge attributes are interesting because, as mentioned above, they act as a standard form for several other types of graphs that can all be transformed to SAUHGs [77].

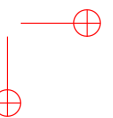
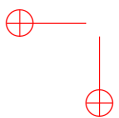
The chapter is organized as follows. In section 5.1, the specific notation used throughout the chapter is described, and the main definitions are introduced. In section 5.2, we introduce novel 1-WL and unfolding equivalences suitable for dynamic graphs and SAUHGs with node and edge attributes and we prove that those equivalences are equal. In section 5.3 the approximation theorems for GNNs on both graph types are presented. Finally, we support our theoretical findings by setting up synthetic experiments in 5.4.

5.1 Notation and Preliminaries

Before extending the work about the expressive power of GNNs to dynamic and edge-attributed graph domains, we introduce some mathematical notation and preliminary definitions. We remark that we limit our analysis to finite graphs.

Let us introduce static, node/edge attributed, undirected and homogeneous graphs, which will be denoted by SAUHGs. The reason for defining and using them comes from [77]. Here, it is shown that a large number of graph types can be bijectively transformed into SAUHGs. Therefore, SAUHGs can be used as a standard form for different types of graphs, including directed or undirected graphs, hypergraphs, multigraphs, heterogeneous or attributed graphs, and any composition of those. More details on the importance and the relevance in applications of SAUHGs are given in [77].

Definition 5.1 (Static Attributed Undirected Homogeneous Graphs). G' is called **static, node/edge attributed, undirected, homogeneous graph (SAUHG)** if $G' = (V', E', \alpha', \omega')$, with V' is a finite set of nodes, $E' \subset \{\{u, v\} \mid \forall u, v \in V'\}$ is a finite set of edges and node and edge attributes are determined by the mappings $\alpha' : V' \rightarrow A$, $\omega' : E' \rightarrow B$



that map into the arbitrary node attribute set A , and edge attribute set B . The domain of SAUHGs will be denoted as \mathcal{G}' .

Remark 5.2. (Attribute sets) In the above definition of the SAUHG, the node and edge attribute sets A and B can be arbitrary. However, without loss of generality, we can assume that the attribute sets are equal because they can be arbitrarily extended to $A' = A \cup B$. Additionally, any arbitrary attribute set A' can be embedded into the k -dimensional vector space \mathbb{R}^k . Since the attribute sets in general do not matter for the theories in this Chapter and to support a better readability, in what follows we consider $\alpha' : V' \rightarrow \mathbb{R}^q$, $\omega : E' \rightarrow \mathbb{R}^{q_e}$ for every SAUHG.

All the aforementioned graph types are static, but temporal changes play an essential role in learning on graphs representing real-world applications; thus, we give a definition of dynamic graphs consisting of a discrete-time representation.

Definition 5.3 (Dynamic Graph). Let $I = [0, \dots, l] \subsetneq \mathbb{N}_0$ be a set of timesteps. Then a **(discrete) dynamic graph** can be defined as a vector of static graph snapshots, i.e., $G = (G_t)_{t \in I}$, where $G_t = (V_t, E_t) \forall t \in I$. Furthermore,

$$\alpha_v(t) := \alpha(v, t), \quad v \in V_t \quad \text{and}$$

$$\omega_{\{u,v\}}(t) := \omega(\{u, v\}, t), \quad \{u, v\} \in E_t$$

where $\alpha : V \times I \rightarrow A$ and $\omega : E \times I \rightarrow B$ define the vector of **dynamic node/edge attributes**. Moreover, $V := \bigcup_{t \in I} V_t$ and $E := \bigcup_{t \in I} E_t$ are the total node and edge set of the dynamic graph. In particular, when a node v does not exist, its attributes and neighborhood are empty, respectively. Finally, let us define

$$\Omega_{\text{ne}[v]}(t) = \left(\omega_{\{v, x_1\}}(t), \dots, \omega_{\{v, x_{|\text{ne}[v](t)|}\}}(t) \right)_{t \in I}$$

to be the sequence of **dynamic edge attributes of the neighborhood** corresponding node at each timestep. Note that, as in Remark 5.2, in the sequel, we assume the attribute sets to be equal and corresponding to \mathbb{R}^q .

To prove the approximation theorems for SAUHGs and dynamic graphs, we need to specify the GNN architectures capable of handling those graph types. Given that a SAUHG acts as a standard form for all graph types, the ordinary GNN architecture will be extended to take also edge attributes into account. This can be done by analogously including the edge attributes in the first iteration to the processing of the node information in the general GNN framework as follows.

Definition 5.4 (SGNN). For a SAUHG $G' = (V', E', \alpha', \omega')$ let $u, v \in V'$ and $e = \{u, v\}$. The SGNN propagation scheme for iteration $k \geq 0$ is defined as

$$\mathbf{h}_v^{(k+1)} = \text{COMBINE}^{(k+1)} \left(\mathbf{h}_v^{(k)}, \text{AGGREGATE}^{(k+1)} \left(\{\{\mathbf{h}_u^{(k)}\}\}_{u \in \text{ne}[v]}, \{\{\omega_{\{u,v\}}\}\}_{u \in \text{ne}[v]}\right) \right)$$

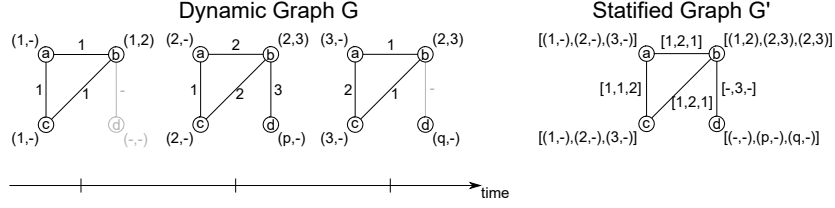


Figure 5.1: Illustration of the statification of a dynamic graph. On the left, the temporal evolution of a graph, including non-existent nodes and edges (gray), is given, and on the right, the corresponding statified graph with the total amount of nodes and edges together with the concatenated attributes is shown.

The output for a node-specific learning problem after the last iteration L respectively is given by

$$\mathbf{z}_v = \text{READOUT}(\mathbf{h}_v^L),$$

using a selected aggregation scheme and a suitable READOUTfunction, and the output for a graph-specific learning problem is determined by

$$\mathbf{z} = \text{READOUT}(\{\{\mathbf{h}_v^L \mid v \in V'\}\}).$$

For the dynamic case, we have chosen a widely used GNN model that is consistent with the theory we built. Based on [78], the discrete dynamic graph neural network (DGNN) uses a GNN to encode each graph snapshot. Here, the model is modified by using the previously defined SGNN in place of the standard one.

Definition 5.5 (Discrete DGNN). Given a discrete dynamic graph $G = (G_t)_{t \in I}$, a **discrete DGNN** using a continuously differentiable recursive function \mathbf{f} for temporal modelling can be expressed as:

$$\begin{aligned} \mathbf{h}_1(t), \dots, \mathbf{h}_n(t) &:= \text{SGNN}(G_t) \quad \forall t \geq 0 \\ \mathbf{q}_1(0), \dots, \mathbf{q}_n(0) &= \mathbf{h}_1(0), \dots, \mathbf{h}_n(0) := \text{SGNN}(G_0) \\ \mathbf{q}_v(t) &:= \mathbf{f}(\mathbf{q}_v(t-1), \mathbf{h}_v(t)) \quad \forall v \in V \end{aligned} \quad (5.1)$$

where $\mathbf{h}_v(t) \in \mathbb{R}^d$ is the hidden representation of node v at time t of dimension d and $\mathbf{q}_v(t) \in \mathbb{R}^d$ is an d -dimensional hidden representation of node v produced by \mathbf{f} , and $\mathbf{f} : \mathbb{R}^s \times \mathbb{R}^d \rightarrow \mathbb{R}^s$ is a neural architecture for temporal modeling (in the methods surveyed in [78], \mathbf{f} is almost always an RNN or an LSTM).

The stacked version of the discrete DGNN is then:

$$\begin{aligned} \mathbf{H}(t) &= \text{SGNN}(G_t) \\ \mathbf{Q}(0) &= \mathbf{H}(0) = \text{SGNN}(G_0) \\ \mathbf{Q}(t) &= \mathbf{F}(\mathbf{Q}(t-1), \mathbf{H}(t)) \end{aligned} \quad (5.2)$$

where $\mathbf{H}(t) \in \mathbb{R}^{N \times d}$, $\mathbf{Q}(t) \in \mathbb{R}^{N \times s}$, $\mathbf{F} : \mathbb{R}^{N \times s} \times \mathbb{R}^{N \times d} \rightarrow \mathbb{R}^{N \times s}$, being N the number of nodes, d and s the dimensions of the hidden representation of a node produced respectively

by the SGNN and by the \mathbf{f} . Applying \mathbf{F} corresponds to component-wise applying \mathbf{f} for each node [78].

To conclude, a function $\text{READOUT}_{\text{dyn}}$ will take as input $\mathbf{Q}(t)$ and gives a suitable output for the considered task, so that altogether the DGNN will be described as

$$\varphi(t, G, v) = \text{READOUT}_{\text{dyn}}(\mathbf{Q}(t)).$$

We remark that the DGNN is a Message-Passing model because the SGNN is one, by definition.

The GNNs expressivity of attributed and dynamic graphs is studied in terms of their capability to distinguish two non-isomorphic graphs.

Definition 5.6 (Graph Isomorphism for attributed and dynamic graphs). In case the two graphs are **attributed**, i.e., $G_1 = (V_1, E_1, \alpha_1, \omega_1)$ and $G_2 = (V_2, E_2, \alpha_2, \omega_2)$, then $G_1 \approx G_2$ if and only if additionally there exist bijective functions $\varphi_\alpha : A_1 \rightarrow A_2$ and $\varphi_\omega : B_1 \rightarrow B_2$ with images $A_i := \text{im}(\alpha_i)$ and $B_i := \text{im}(\omega_i)$, $i = 1, 2$.

1. $\varphi_\alpha(\alpha_1(v_1)) = \alpha_2(\phi(v_1)) \quad \forall v_1 \in V_1$,
2. $\varphi_\omega(\omega_1(\{u_1, v_1\})) = \omega_2(\{\phi(u_1), \phi(v_1)\}) \quad \forall \{u_1, v_1\} \in E_1$.

If the two graphs are **dynamic**, they are called to be isomorphic if and only if the static graph snapshots of each timestep are isomorphic.

As seen in Chapter 4, the expressive power of GNNs can also be approached from the point of view of their approximation capability. This property generally analyzes the capacity of different GNN models to approximate arbitrary functions [79].

Different universal approximation theorems can be defined depending on the model, the considered input data, and the sets of functions.

Since the results in Chapter 4 hold for undirected and node-attributed graphs only, we aim to extend the universal approximation theorem to GNNs working on SAUHGs (cf. Def. 5.1) and dynamic graphs (cf. Def. 5.3). For this purpose, in the next sections, we introduce a static attributed and a dynamic version of both the WL test and the unfolding trees to show that the graph equivalences regarding the attributed/dynamic WL test and attributed/dynamic unfolding trees are equivalent. With these notions, we define the set of functions that are attributed/dynamic unfolding tree preserving and reformulate the universal approximation theorem to the attributed and dynamic cases (cf. Theorem. 5.21 and Theorem. 5.29).

5.2 Weisfeiler-Lehman and Unfolding Trees

This section first proposes the extended versions of the unfolding trees (UT) and the Weisfeiler-Lehman (WL) test to SAUHGs and dynamic graphs. Having these two notions, it is possible to show the equivalence between the extended versions of UT-equivalence and the WL-equivalence for SAUHGs and dynamic graphs.

5.2.1 Equivalence for SAUHGs

The extended result on SAUHGs is formalized and proven in Thm. 5.12. The original WL test and unfolding tree notions defined in Chapters 2 and 4 cover all graph properties except edge attributes. Thus, the notions of unfolding trees and the WL test have to be extended to an attributed version.

Definition 5.7 (Attributed Unfolding Tree). The **attributed unfolding tree** T_v^k in graph $G' = (V', E', \alpha', \omega')$ of node $v \in V'$ **up to depth** $k \in \mathbb{N}_0$ is defined as

$$T_v^k = \begin{cases} \text{Tree}(\alpha'_v), & \text{if } k = 0 \\ \text{Tree}(\alpha'_v, \Omega'_{\text{ne}[v]}, T_{\text{ne}[v]}^{k-1}) & \text{if } k > 0, \end{cases}$$

where $\text{Tree}(\alpha'_v)$ is a tree constituted of node v with attribute α'_v . $\text{Tree}(\alpha'_v, \Omega'_{\text{ne}[v]}, T_{\text{ne}[v]}^{k-1})$ is the tree consisting of the root node v and subtrees $T_{\text{ne}[v]}^{k-1} = \{\{T_{u_1}^{k-1}, \dots, T_{u_{|\text{ne}[v]|}}^{k-1}\}\}$ of depth $k-1$, that are connected by the corresponding edge attributes $\Omega'_{\text{ne}[v]} = \{\{\omega'_{\{v, u_1\}}, \dots, \omega'_{\{v, u_{|\text{ne}[v]|}\}}\}\}$ of the neighbors of v .

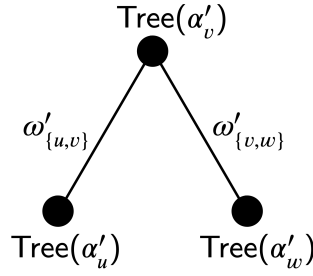


Figure 5.2: Unfolding tree recursive construction

Moreover, the **attributed unfolding tree** of v determined by $T_v = \lim_{k \rightarrow \infty} T_v^k$ is obtained by merging all unfolding trees T_v^k of any depth k .

Definition 5.8 (Attributed Unfolding Equivalence). Let $G_1 = (V_1, E_1, \alpha_1, \omega_1)$ and $G_2 = (V_2, E_2, \alpha_2, \omega_2)$ be two SAUHGs. Then G_1 and G_2 are **attributed unfolding tree equivalent**, noted by $G_1 \sim_{AUT} G_2$, if and only if $\{\{T_u \mid u \in V_1\}\} = \{\{T_v \mid v \in V_2\}\}$. Analogously, two nodes $u \in V_1, v \in V_2$ are unfolding tree equivalent, noted by $u \sim_{AUT} v$ if and only if $T_u = T_v$.

Using the definition of the attributed unfolding equivalence on graphs, the 1-WL test defined in Chapter 2 is extended to attributed graphs.

Definition 5.9 (Attributed 1-WL test). Let HASH be a bijective function that codes every possible node attribute with a color from a color set \mathcal{C} and $G' = (V', E', \alpha', \omega')$. The **attributed 1-WL (1-AWL) test** is defined recursively through the following.

- At iteration $i = 0$, the color is set to the hashed node attribute:

$$c_v^{(0)} = \text{HASH}(\alpha'_v)$$

- At iteration $i > 0$, the HASH function is extended to the edge weights:

$$c_v^{(k)} = \text{HASH}\left(c_v^{(k-1)}, \Omega'_{\text{ne}[v]}, c_{\text{ne}[v]}^{(k-1)}\right)$$

We can now define the attributed 1-WL equivalence on graphs, which is an extension of the 1-WL equivalence of graphs and nodes by using the attributed version of the 1-WL test (cf. Def. [5.9](#)).

Definition 5.10 (Attributed 1-WL equivalence). Two nodes u, v are attributed WL-equivalent, noted by $u \sim_{AWL} v$, if and only if $c_u = c_v$.

Analogously, let $G_1 = (V_1, E_1, \alpha_1, \omega_1)$ and $G_2 = (V_2, E_2, \alpha_2, \omega_2)$ be two SAUHGs. Then, $G_1 \sim_{AWL} G_2$, if and only if for all nodes $v_1 \in V_1$ there exists a corresponding node $v_2 \in V_2$ such that $v_1 \sim_{AWL} v_2$.

The above definitions along with Lemma. [5.11](#), help us to prove Theorem. [5.12](#) that poses the relation between the attributed unfolding equivalence and the attributed 1-WL test. In more detail, the Lemma states the equivalence between the attributed unfolding tree equivalence of nodes and the equality of their attributed unfolding trees up to a specific depth. In [\[1\]](#), it has been shown that the unfolding trees of infinite depth are not necessary to consider for this equivalence. Instead, the larger number of nodes of both graphs under consideration is sufficient for the depth of the unfolding trees, which is finite since the graphs are bounded. Hence the following result determines the equivalence between the attributed unfolding trees of two nodes and their colors resulting from the attributed 1-WL test.

Lemma 5.11. Consider $G' = (V', E', \alpha', \omega')$ as the SAUHG resulting from a transformation of an arbitrary static graph $G = (V, E, \alpha, \omega)$ with nodes $u, v \in V$ and corresponding attributes α_u, α_v . Then it holds

$$\forall k \in \mathbb{N}_0 : T_u^k = T_v^k \iff c_u^{(k)} = c_v^{(k)}.$$

Proof. The proof is carried out by induction on d , which represents both the depth of the unfolding trees and the iteration step in the WL coloring.

$k = 0$: It holds

$$\begin{aligned} T_u^0 &= \text{Tree}(\alpha'_u) = \text{Tree}(\alpha'_v) = T_v^0 \\ \iff \alpha'_u &= \alpha'_v \text{ and } c_u^{(0)} = \text{HASH}(\alpha'_u) = \text{HASH}(\alpha'_v) = c_v^{(0)}. \end{aligned}$$

$k > 0$: Suppose that Eq. [\(5.11\)](#) holds for $k - 1$, and prove that it holds also for k .

- By definition, $T_u^k = T_v^k$ is equivalent to

$$\begin{cases} T_u^{k-1} = T_v^{k-1} & \text{and} \\ \text{Tree}(\alpha'_u, \Omega'_{\text{ne}[u]}, T_{\text{ne}[u]}^{k-1}) = \text{Tree}(\alpha'_v, \Omega'_{\text{ne}[v]}, T_{\text{ne}[v]}^{k-1}). \end{cases} \quad (5.3)$$

- Applying the induction hypothesis, it holds that

$$T_u^{k-1} = T_v^{k-1} \iff c_u^{(k-1)} = c_v^{(k-1)}. \quad (5.4)$$

- Eq. (5.3) is equivalent to the following:

$$\alpha'_u = \alpha'_v, \quad \Omega'_{\text{ne}[u]} = \Omega'_{\text{ne}[v]} \quad \text{and} \quad T_{\text{ne}[u]}^{k-1} = T_{\text{ne}[v]}^{k-1}.$$

Given the definition of the unfolding trees and their construction, this is equivalent to

$$\begin{cases} \omega'_{\{u, u_i\}} = \omega'_{\{v, v_i\}} & \forall u_i \in \text{ne}_u, v_i \in \text{ne}_v \quad \text{and} \\ T_{u_i}^{k-1} = T_{v_i}^{k-1} & \forall u_i \in \text{ne}_u, v_i \in \text{ne}_v. \end{cases} \quad (5.5)$$

- By the induction hypothesis, Eq. (5.5) is equivalent to

$$c_{\text{ne}_u}^{(k-1)} = c_{\text{ne}_v}^{(k-1)}, \quad \text{i.e.,}$$

$$\{\{c_{u_i}^{(k-1)} \mid u_i \in \text{ne}[u]\}\} = \{\{c_{v_i}^{(k-1)} \mid v_i \in \text{ne}[v]\}\}.$$

- Putting together Eq. (5.4), (5.5), and the fact that the HASH function is bijective, we obtain:

$$\begin{aligned} & \text{HASH}\left(\left(c_u^{(k-1)}, \Omega'_{\text{ne}[u]}, \{\{c_{u_i}^{(k-1)} \mid u_i \in \text{ne}[u]\}\}\right)\right) \\ &= \text{HASH}\left(\left(c_v^{(k-1)}, \Omega'_{\text{ne}[v]}, \{\{c_{v_i}^{(k-1)} \mid v_i \in \text{ne}[v]\}\}\right)\right) \end{aligned}$$

which, by definition, is equivalent to $c_u^{(k)} = c_v^{(k)}$. □

Directly from Lem. 5.11, the equivalence of the attributed unfolding tree equivalence and the attributed 1-WL equivalence of two nodes belonging to the same graph can be formalized.

Theorem 5.12. *Consider G' as in Lem. 5.11. Then, it holds*

$$\forall u, v \in V' : u \sim_{AUT} v \iff u \sim_{AWL} v.$$

Proof. The proof follows from the proof of Lem. 5.11. □

5.2.2 Equivalence for Dynamic Graphs

Now the previously introduced concepts of unfolding tree and WL equivalences are extended to the dynamic case. Note that Lem. 5.11 and, therefore, Thm. 5.12 also hold in case G' is the SAUHG resulting from a transformation of a dynamic graph $G = (G_t)_{t \in I}$ to its static attributed version. However, the GNNs working on dynamic graphs usually use a significantly different architecture than those that work on static attributed graphs. Therefore, the derivation of the various equivalences on dynamic graphs separately is now presented.

First, dynamic unfolding trees are introduced as a sequence of unfolding trees for each graph snapshot respectively.

Definition 5.13 (Dynamic Unfolding Tree). Let $G = (G_t)_{t \in I}$ with $G_t = (V_t, E_t, \alpha_t, \omega_t)$ be a dynamic graph. The **dynamic unfolding tree** $T_v^k(t)$ at time $t \in I$ of node $v \in V$ up to depth $k \in \mathbb{N}_0$ is defined as

$$T_v^k(t) = \begin{cases} \text{Tree}(\alpha_v(t)), & \text{if } k = 0 \\ \text{Tree}(\alpha_v(t), \Omega_{ne_v(t)}, T_{ne_v(t)}^{k-1}(t)) & \text{if } k > 0, \end{cases}$$

where $\text{Tree}(\alpha_v(t))$ is a tree constituted of node v with attribute $\alpha_v(t)$. Furthermore, $\text{Tree}(\alpha_v(t), \Omega_{ne_v(t)}, T_{ne_v(t)}^{k-1}(t))$ is the tree with root node v with attribute $\alpha_v(t)$. Additionally, $T_{ne_v(t)}^{k-1}(t) = \{T_{u_1}^{k-1}(t), \dots, T_{u_{|ne_v(t)|}}^{k-1}(t)\}$ are corresponding subtrees with edge attributes $\Omega_{ne_v(t)}$. If the node v does not exist at time t , the corresponding tree is empty, there is no tree of depth $k > 0$ for this timestep and v does not occur in any neighborhood of other nodes.

We are now ready to introduce the equivalence of two dynamic graphs in function of their dynamic unfolding trees.

Definition 5.14. Two nodes $u, v \in V$ are said to be **dynamic unfolding equivalent** $u \sim_{DUT} v$ if $T_u(t) = T_v(t)$ for every timestep t . Analogously, two dynamic graphs G_1, G_2 are said to be **dynamic unfolding equivalent** $G_1 \sim_{DUT} G_2$, if there exists a bijection between the nodes of the graphs that respects the partition induced by the unfolding equivalence on the nodes.

Consistently, we introduce the 1-WL test for dynamic graphs, called Dynamic 1-WL test.

Definition 5.15 (Dynamic 1-WL test). Let $G = (G_t)_{t \in I}$ with $G_t = (V, E, \alpha_t, \omega_t)$ be a dynamic graph. Let HASH_t^0 be a bijective function encoding every node attribute of G_t with a color from a color set \mathcal{C} .

The **dynamic 1-WL test (1-DWL)** generates a vector of color sets one for each timestep $t \in I$ by:

- At iteration $k = 0$ the color is set to the hashed node attribute or a fixed color for non-existent nodes:

$$c_v^{(0)}(t) = \begin{cases} \text{HASH}_t^0(\alpha_v(t)) & \text{if } v \in V_t, \\ c^\perp & \text{otherwise.} \end{cases}$$

- Then, the aggregation mechanism is defined by the bijective function HASH_t for $k > 0$:

$$c_v^{(k)}(t) = \text{HASH}_t \left((c_v^{(k-1)}(t), \Omega_{ne[v](t)}, c_{ne[v](t)}^{(k-1)}(t)) \right)$$

Note that for $k > 0$, $c_v^{(k-1)}(t) = c^\perp$ holds for a non-existent node at time t . Further, the neighborset is empty so the other inputs of HASH_t are empty, and together with c^\perp it will always give the same color for non-existent nodes.

Definition 5.16 (Dynamic 1-WL equivalence). Two nodes $u, v \in V$ in a dynamic graph G are said to be **dynamic WL equivalent**, noted by $u \sim_{DWL} v$, if their colors resulting from the WL test are pairwise equal per timestep. Analogously, let G_1 and G_2 be dynamic graphs. Then $G_1 \sim_{DWL} G_2$, if and only if for all nodes $v_1 \in V_t^{(1)}$ there exists a corresponding node $v_2 \in V_t^{(2)}$ with $c_{v_1}(t) = c_{v_2}(t)$ for all $t \in I$.

As in Subsection 5.2.1 we now state the bijective correspondence between the Dynamic UT equivalence and the Dynamic WL equivalence.

Theorem 5.17 (Equivalence of Dynamic WL Equivalence and Dynamic UT Equivalence for nodes). *Let $G = (G_t)_{t \in I}$ be a dynamic graph and $u, v \in V$. Then, it holds*

$$u \sim_{DUT} v \iff u \sim_{DWL} v.$$

Proof. Two nodes are dynamic unfolding tree equivalent iff they are attributed unfolding tree equivalent at each timestep $u \sim_{AUT} v \quad \forall t \in I$ 5.14. Further, as consequence of Thm. 5.12, it holds that for all $t \in I$ the two nodes are attributed WL equivalent $u \sim_{AWL} v$ and thus, the two nodes are dynamic WL equivalent by Def. 5.16. In case of the non-existence of u at a certain timestep t , the Theorem still holds. \square

5.3 Approximation Capability of GNNs for SAUHGs and DGNNs

The results from Sec. 5.2.1 and Sec. 5.2.2 can be brought together in the formulation of a universal approximation theorem for GNNs working on SAUHGs and dynamic graphs and the set of functions that preserve the attributed or dynamic unfolding equivalence, respectively.

5.3.1 GNNs for SAUHGS

Since the goal is to show the attributed extension of the universal approximation theorem, it is necessary to define the corresponding family of attributed unfolding equivalence-preserving functions. A function preserves the attributed unfolding equivalence if the output of the function is equal when two nodes are attributed unfolding equivalent.

Definition 5.18. Let \mathcal{G}' be the domain of bounded SAUHGs, $G' = (V', E', \alpha', \omega') \in \mathcal{G}'$ a SAUHG and $u, v \in V'$ two nodes. Then a function $\mathbf{f} : \mathcal{G}' \rightarrow \mathbb{R}^o$ is said to **preserve the attributed unfolding equivalence** on \mathcal{G}' if

$$v \sim_{AUT} u \Rightarrow \mathbf{f}(G', v) = \mathbf{f}(G', u).$$

All functions that preserve the attributed unfolding equivalence are collected in the set $\mathcal{F}(\mathcal{G}')$.

Analogously to what stated in [1], there exists a relation between the unfolding equivalence preserving functions and the unfolding trees for attributed graphs.

Proposition 5.19 (Functions of attributed unfolding trees). *A function \mathbf{f} belongs to $\mathcal{F}(\mathcal{G}')$ if and only if there exists a function κ defined on trees such that for any graph $G' \in \mathcal{G}'$ it holds $\mathbf{f}(G', v) = \kappa(T_v)$, for any node $v \in G'$.*

Proof. The proof works analogously to the proof of the unattributed version presented in [1]. We show both equivalence directions:

- \Leftarrow If there exists a function κ on attributed unfolding trees such that $\mathbf{f}(G', v) = \kappa(T_v)$ for all $v \in G'$, then $u \sim_{AUT} v$ for $u, v \in G$ implies $\mathbf{f}(G', u) = \kappa(T_u) = \kappa(T_v) = \mathbf{f}(G', v)$.
- \Rightarrow If \mathbf{f} preserves the attributed unfolding equivalence, then a function κ on the attributed unfolding tree of an arbitrary node v can be defined as $\kappa(T_v) := \mathbf{f}(G', v)$. Then, if T_u and T_v are two attributed unfolding trees, $T_u = T_v$ implies $\mathbf{f}(G', u) = \mathbf{f}(G', v)$ and κ is uniquely defined. □

Finally, considering the properties of SAUHGs given in Sec. 5.2.1 we can state the universal approximation capability of the SGNNs on bounded SAUHGs. Since its proof proceeds analogously to the one in Chapter 4, what remains to prove is that a domain of SAUGH graphs with continuous attributes can be partitioned into small subsets, so that the attributes of the graphs are almost constant in each partition. Moreover, in probability, a finite number of partitions is sufficient to cover a large part of the domain. This can be summarized in the following Lemma.

Lemma 5.20. *For any probability measure P on \mathcal{D}' , and any reals λ, δ , where $\lambda > 0$, $\delta \geq 0$, there exists a real $\bar{b} > 0$, which is independent of δ , a set $\mathcal{D}' \subseteq \mathcal{D}'$, and a finite number of partitions $\mathcal{D}'_1, \dots, \mathcal{D}'_p$ of \mathcal{D}' , where $\mathcal{D}'_j = \mathcal{G}'_j \times \{v_j\}$, with $\mathcal{G}'_j \subseteq \mathcal{G}'$ and $v_j \in \mathcal{G}'_j$, such that:*

1. $P(\bar{\mathcal{D}}') \geq 1 - \lambda$ holds;
2. for each j , all the graphs in \mathcal{G}'_j have the same structure, i.e., they differ only in the values of their attributes;
3. for each set \mathcal{D}'_j , there exists a hypercube $\mathcal{H}_j \subset \mathbb{R}^{NM2k}$ such that $\gamma_G \in \mathcal{H}_j$ holds for any graph $G' \in \mathcal{G}'_j$ with $N = \max_{G' \in \mathcal{G}'} |V'|$ and $M = \max_{G' \in \mathcal{G}'} |E'|$. Here, $\gamma_{G'}$ denotes the vector obtained by concatenating all the attribute vectors of both nodes and edges of G' , namely $\gamma_{G'} = [\mathbf{A}_{G'} | \mathbf{\Omega}_{G'}]$, where $\mathbf{A}_{G'}$ is the concatenation of all the node attributes and $\mathbf{\Omega}_{G'}$ is the concatenation of all edge attributes;
4. for any two different sets $\mathcal{G}'_i, \mathcal{G}'_j$, $i \neq j$, their graphs have different structures, or their hypercubes $\mathcal{H}_i, \mathcal{H}_j$ are disjoint, i.e., $\mathcal{H}_i \cap \mathcal{H}_j = \emptyset$;
5. for each j and each pair of graphs $G_1, G_2 \in \mathcal{G}'_j$, the inequality $\|\gamma_{G_1} - \gamma_{G_2}\|_\infty \leq \delta$ holds;
6. for each graph $G' \in \bar{\mathcal{D}}'$, the inequality $\|\gamma_{G'}\|_\infty \leq \bar{b}$ holds.

Proof. The proof is similar to the one contained in [1]. The only remark needed here is that we can consider the whole concatenating of all attributes from both nodes and edges without loss of generality; indeed, if we were considering the node and the edge attributes separately, we would need conditions on the hypercubes, s.t.:

$$\begin{aligned} & \|\mathbf{A}_{G_1} - \mathbf{A}_{G_2}\|_\infty \leq \delta^A, \delta^A > 0, \\ \text{and} \quad & \|\mathbf{\Omega}_{G_1} - \mathbf{\Omega}_{G_2}\|_\infty \leq \delta^\Omega, \delta^\Omega > 0. \end{aligned}$$

Then we can stack those attribute vectors, as in the statement, s.t. :

$$\begin{aligned} \|\gamma_{G_1} - \gamma_{G_2}\|_\infty &= \|([\mathbf{A}_{G_1}|\mathbf{0}] + [\mathbf{0}|\boldsymbol{\Omega}_{G_1}]) - ([\mathbf{A}_{G_2}|\mathbf{0}] + [\mathbf{0}|\boldsymbol{\Omega}_{G_2}])\|_\infty \\ &\leq \|\mathbf{A}_{G_1} - \mathbf{A}_{G_2}\|_\infty + \|\boldsymbol{\Omega}_{G_1} - \boldsymbol{\Omega}_{G_2}\|_\infty \\ &\leq \delta^A + \delta^\Omega := \delta \end{aligned}$$

which allows us to exploit the same proof contained in [\[1\]](#). \square

We are now able to give an important result.

Theorem 5.21 (Universal Approximation Theorem by SGNN). *Let \mathcal{G}' be the domain of bounded SAUHGs with the maximal number of nodes $N = \max_{G' \in \mathcal{G}'} |G'|$. For any measurable function $\mathbf{f} \in \mathcal{F}(\mathcal{G}')$ preserving the attributed unfolding equivalence (cf. Def. [5.18](#)), any norm $\|\cdot\|$ on \mathbb{R} , any probability measure P on \mathcal{G}' , for any reals ϵ, λ where $\epsilon, \lambda > 0$, there exists a SGNN defined by the continuously differentiable functions $\text{COMBINE}^{(k)}$, $\text{AGGREGATE}^{(k)}$, at iteration $k \leq 2N - 1$, and by the function READOUT , with hidden dimension $d = 1$, i.e., $\mathbf{h}_v^k \in \mathbb{R} \forall v$, such that the function φ (realized by the GNN) computed after $2N - 1$ steps for all $G' \in \mathcal{G}'$ satisfies the condition*

$$P(\|\mathbf{f}(G', v) - \varphi(G', v)\| \leq \epsilon) \geq 1 - \lambda.$$

Following what done in Chapter [4](#), the proof is carried on by proving the equivalence of the above Theorem with the following one, where the domain contains a finite number of graphs and the attributes are integers.

Theorem 5.22. *For any finite set of p patterns*

$\{(G'_j, v) \mid G'_j \in \mathcal{G}', v \in V'_j, j \in [p]\}$, with the maximal number of nodes in the domain $N = \max_{G' \in \mathcal{G}'} |G'|$, for any function τ which preserves the attributed unfolding equivalence, and for any real $\epsilon > 0$, there exist continuously differentiable functions $\text{AGGREGATE}^{(k)}$, $\text{COMBINE}^{(k)}$, $\forall k \leq 2N - 1$, s.t.

$$\mathbf{h}_v^k = \text{COMBINE}^{(k)}\left(\mathbf{h}_v^{(k-1)}, \text{AGGREGATE}^{(k)}\left(\{\{\mathbf{h}_u^{k-1}\}_{u \in \text{ne}[v]}\}, \{\{\boldsymbol{\omega}_{\{u,v\}}\}_{u \in \text{ne}[v]}\}\right)\right)$$

and a function READOUT , with hidden dimension $d = 1$, i.e., $\mathbf{h}_v^k \in \mathbb{R}$, so that the function φ (realized by the SGNN), computed after $2N - 1$ steps, satisfies the condition

$$|\tau(G'_j, v) - \varphi(G'_j, v)| \leq \epsilon \quad \text{for any } v \in V'_j. \quad (5.6)$$

Proof. As for the proof of Theorem [4.12](#), the idea is designing a GNN that can approximate any function τ that preserves the attributed unfolding equivalence. According to Thm. [5.19](#) there exists a function κ , s.t.

$$\tau(G'_j, v) = \kappa(T_v).$$

Therefore, the GNN has to encode the attributed unfolding tree into the node attributes, i.e., for each node v , we want to have $\mathbf{h}_v = \nabla(T_v)$, where ∇ is an encoding function that maps attributed unfolding trees into real numbers. The existence and injectiveness of ∇ are ensured by construction. More precisely, the encodings are constructed recursively by the $\text{AGGREGATE}^{(k)}$ and the $\text{COMBINE}^{(k)}$ functions using the neighborhood information, i.e., the node and edge attributes.

Consequently, the theorem can be proven given that there exist appropriate functions ∇ , $\text{AGGREGATE}^{(k)}$, $\text{COMBINE}^{(k)}$ and READOUT .

For this purpose, the functions $\text{AGGREGATE}^{(k)}$ and $\text{COMBINE}^{(k)}$ must satisfy $\forall k \leq 2N - 1$:

$$\begin{aligned} \nabla(T_v^k) &= \mathbf{h}_v^k \\ &= \text{COMBINE}^{(k)} \left(\mathbf{h}_v^{(k-1)}, \text{AGGREGATE}^{(k)} \left(\{\{\mathbf{h}_u^{(k-1)}\}_{u \in \text{ne}[v]}\}, \{\{\boldsymbol{\omega}'_{\{u,v\}}\}_{u \in \text{ne}[v]}\} \right) \right) \\ &= \text{COMBINE}^{(k)} \left(\nabla(T_v^{k-1}), \text{AGGREGATE}^{(k)} \left(\{\{\nabla(T_u^{k-1})\}_{u \in \text{ne}[v]}\}, \Omega'_{\text{ne}[v]} \right) \right). \end{aligned}$$

In a simple solution, $\text{AGGREGATE}^{(k)}$ decodes the attributed trees of the neighbors u of v , T_u^{k-1} , and stores them into a data structure to be accessed by $\text{COMBINE}^{(k)}$. The detailed construction of the appropriate functions is given in Chapter 4. \square

Adopting an argument similar to that in Chapter 4 it is proven that the previous theorem is equivalent to Thm. 5.21.

Theorem 5.23. *Theorem 5.21 holds if and only if Theorem 5.22 holds.*

Figure 5.3 summarize the structure of the proof of Theorem 5.21.

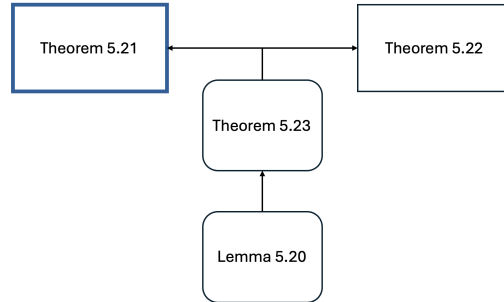


Figure 5.3: Structure of the proof of Theorem 5.21.

We want now to study the case when the employed components (COMBINE , AGGREGATE , READOUT) are sufficiently general to be able to approximate any function preserving the unfolding equivalence. We call this class of networks, \mathcal{Q}_S , *SGNN models with universal components*. To simplify our discussion, we introduce the transition function $\mathbf{f}^{(k)}$ to indicate the concatenation of the $\text{AGGREGATE}^{(k)}$ and $\text{COMBINE}^{(k)}$, i.e.,

$$\begin{aligned} \mathbf{f}^{(k)}(\mathbf{h}_v^k, \{\{\mathbf{h}_u^{(k-1)}\}_{u \in \text{ne}[v]}\}, \{\{\boldsymbol{\omega}_{\{u,v\}}\}_{u \in \text{ne}[v]}\}) = \\ \text{COMBINE}^{(k)} \left(\mathbf{h}_v^{(k-1)}, \text{AGGREGATE}^{(k)} \left(\{\{\mathbf{h}_u^{(k-1)}\}_{u \in \text{ne}[v]}\}, \{\{\boldsymbol{\omega}_{\{u,v\}}\}_{u \in \text{ne}[v]}\} \right) \right) \end{aligned}$$

Then, we can formally define the class \mathcal{Q}_S .

Definition 5.24. A class \mathcal{Q}_S of SGNN models is said to have *universal components* if, for any $\epsilon > 0$ and any continuous target functions $\overline{\text{COMBINE}}^{(k)}$, $\overline{\text{AGGREGATE}}^{(k)}$, $\overline{\text{READOUT}}$,

there exists an SGNN belonging to \mathcal{Q}_S , with functions $\text{COMBINE}_\theta^{(k)}$, $\text{AGGREGATE}_\theta^{(k)}$, READOUT_θ and parameters θ such that

$$\begin{aligned} \left\| \bar{\mathbf{f}}^{(k)}(\mathbf{h}, \{\mathbf{h}_1, \dots, \mathbf{h}_n\}) - \mathbf{f}_\theta^{(k)}(\mathbf{h}, \{\mathbf{h}_1, \dots, \mathbf{h}_n\}) \right\|_\infty &\leq \epsilon \\ \left\| \overline{\text{READOUT}}(\mathbf{q}) - \text{READOUT}_\theta(\mathbf{q}) \right\|_\infty &\leq \epsilon, \end{aligned}$$

holds, for any vectors $\mathbf{h}, \mathbf{h}_1, \dots, \mathbf{h}_n \in \mathbb{R}^d$, $\mathbf{q} \in \mathbb{R}^d$. The transition functions $\bar{\mathbf{f}}^{(k)}$ and $\mathbf{f}_\theta^{(k)}$ correspond to the target function and the SGNN, respectively.

The following result shows that Theorem 5.21 still holds even for SGNNs with universal components.

Theorem 5.25 (Approximation by Neural Networks). *Assuming that the hypotheses of Theorem 5.21 are fulfilled and \mathcal{Q}_S is a class of SGNNs with universal components. Then, there exists a parameter set θ and some functions $\text{COMBINE}_\theta^{(k)}$, $\text{AGGREGATE}_\theta^{(k)}$, READOUT_θ , implemented by Neural Networks in \mathcal{Q}_S , such that the thesis of Theorem 5.21 holds.*

Proof. The proof is identical to the one of the Theorem 4.17 contained in Chapter 4. \square

5.3.2 GNNs for Dynamic Graphs

Suitable functions that preserve the unfolding equivalence on dynamic graphs are dynamic systems. Before this statement is formalized and proven in Prop. 5.28, dynamic systems and their property to preserve the dynamic unfolding equivalence are defined in the following.

Definition 5.26 (Dynamic System). Let \mathcal{G} be a domain of dynamic graphs and let $V = \bigcup_t V_t$.

A **dynamic system** is defined as a function $\mathbf{dyn} : \mathcal{D} := I \times \mathcal{G} \times V \rightarrow \mathbb{R}^o$ formalized for $G = (G_t)_{t \in I} \in \mathcal{G}$, and $v \in V_t$ by

$$\mathbf{dyn}(t, G, v) := \mathbf{g}(x_v(t)). \quad (5.7)$$

Here, $\mathbf{g} : \mathbb{R}^s \rightarrow \mathbb{R}^o$ is an output function, and the *state function* $x_v(t)$ is determined by

$$x_v(t) = \begin{cases} \mathbf{a}(t, G, v) & \text{if } t = 0 \\ \mathbf{f}(x_v(t-1), \mathbf{a}(t-1, G, v)) & \text{if } t > 0, \end{cases}$$

for $v \in V_t$, where $\mathbf{a} : I \times \mathcal{G} \times V \rightarrow \mathbb{R}^s$ is a function that processes the graph snapshot at time t and provides an s -dimensional internal state representation for each node v . Finally, $\mathbf{f} : \mathbb{R}^s \times \mathbb{R}^s \rightarrow \mathbb{R}^s$ is a recursive function, that is called *state update function*.

Definition 5.27. A dynamic system $\mathbf{dyn}(\cdot, \cdot, \cdot)$ **preserves the dynamic unfolding tree equivalence** on \mathcal{G} if and only if for any input graph sequences $G_1, G_2 \in \mathcal{G}$, and two nodes $u, v \in V$ it holds

$$v \sim_{DUT} u \implies \mathbf{dyn}(t, G_1, v) = \mathbf{dyn}(t, G_2, u) \quad \forall t.$$

The class of dynamic systems that preserve the unfolding equivalence on \mathcal{D} will be denoted with $\mathcal{F}(\mathcal{D})$. A characterization of $\mathcal{F}(\mathcal{D})$ is given by the following result (following the work in [1]).

Proposition 5.28 (Functions of dynamic unfolding trees). *A dynamic system \mathbf{dyn} belongs to $\mathcal{F}(\mathcal{D})$ if and only if there exists a function κ defined on attributed trees such that for all $(t, G, v) \in \mathcal{D}$ it holds*

$$\mathbf{dyn}(t, G, v) = \kappa\left(\left(T_v(i)\right)_{i \in [t]}\right).$$

Proof. We show the proposition by proving both directions of the equivalence relation:

\Rightarrow : If there exists κ such that $\mathbf{dyn}(t, G, v) = \kappa\left(\left(T_v(i)\right)_{i \in [t]}\right)$ for all triplets $(t, G, v) \in \mathcal{D}$, then for any pair of nodes $u \in G_1, v \in G_2$ with $u \sim_{DUT} v$ it holds

$$\mathbf{dyn}(t, G_1, u) = \kappa\left(\left(T_u(i)\right)_{i \in [t]}\right) = \kappa\left(\left(T_v(i)\right)_{i \in [t]}\right) = \mathbf{dyn}(t, G_2, v).$$

\Leftarrow : On the other hand, if \mathbf{dyn} preserves the unfolding equivalence, then we can define κ as

$$\kappa\left(\left(T_v(i)\right)_{i \in [t]}\right) = \mathbf{dyn}(t, G, v).$$

Note that the above equality is a correct specification for a function. In fact, if

$$\kappa\left(\left(T_v(i)\right)_{i \in [t]}\right) = \kappa\left(\left(T_u(i)\right)_{i \in [t]}\right)$$

implies $\mathbf{dyn}(t, G, u) = \mathbf{dyn}(t, G, v)$, then κ is uniquely defined. □

Finally, the universal approximation of the Message-Passing GNN for dynamic graphs can be stated.

Theorem 5.29 (Universal Approximation Theorem by DGNN). *Let $G = (G_t)_{t \in I}$ be a discrete dynamic graph in the graph domain \mathcal{G} and $N = \max_{G \in \mathcal{G}} |G|$ be the maximal number of nodes in the domain. Let $\mathbf{dyn}(t, G, v) \in \mathcal{F}(\mathcal{D})$ be any measurable dynamical system preserving the unfolding equivalence, $\|\cdot\|$ be a norm on \mathbb{R} , P be any probability measure on \mathcal{D} and ϵ, λ be any real numbers where $\epsilon, \lambda > 0$. Then, there exists a DGNN composed by SGNNs with $2N - 1$ layers and hidden dimension $d = 1$, and Recurrent Neural Network with state dimension $s = 1$ such that the function φ realized by this model satisfies*

$$P(\|\mathbf{dyn}(t, G, v) - \varphi(t, G, v)\| \leq \epsilon) \geq 1 - \lambda \quad \forall t \in I.$$

To prove the theorem above, we need some preliminary results. Using the same argument used for SAUHGs in Theorem 5.21 we need, as a preliminary result, the extension of [1] (Lemma 1) to the domain of dynamic graphs \mathcal{D} , analogously to the extension to the domain of SAUHGs in Lemma 5.20.

Remark 5.30. Lemma 5.20 holds for the domain of dynamic graphs \mathcal{D} . Indeed, taking into account the argument in [77], one can establish a bijection between the domain of dynamic graphs and the domain of SAUHGs; on the latter, we can directly apply Lemma 5.20.

Thm. 5.29 is equivalent to the following, where the domain contains a finite number of elements in \mathcal{D} and the attributes are integers.

Theorem 5.31. *For any finite set of p patterns*

$$\{(t^{(j)}, G^{(j)}, v^{(j)}) \mid (t^{(j)}, G^{(j)}, v^{(j)}) \in \mathcal{D}, j \in [p]\}$$

with the maximal number of nodes $N = \max_{G \in \mathcal{G}} |G|$ and with graphs having integer features, for any measurable dynamical system preserving the unfolding equivalence, $\|\cdot\|$ be a norm on \mathbb{R} , P be any probability measure on \mathcal{D} and ϵ be any real number where $\epsilon > 0$. Then, there exists a DGNN as defined in Def. 5.5 such that the function φ (realized by this model) satisfies the condition

$$\|\mathbf{dyn}(t^{(j)}, G^{(j)}, v^{(j)}) - \varphi(t^{(j)}, G^{(j)}, v^{(j)})\| \leq \epsilon \quad (5.8)$$

$\forall j \in [p]$ where $t^{(j)} \in I$.

Proof. The proof involves assuming that the output dimension is $o = 1$, i.e., $\mathbf{dyn}(t, G, v) \in \mathbb{R}$, but the result can be extended to the general case with $o \in \mathbb{N}$ by concatenating the corresponding results. As a result of Thm. 5.28, there exists a function κ , s.t. $\mathbf{dyn}(t, G, v) = \mathbf{g}(x_v(t)) = \kappa((T_v(i))_{i \in [t]})$ where $T_v(i)$ is an attributed unfolding tree. Given N_t as the number of nodes of the graph at timestep t , in order to store the graph information, an attributed unfolding tree of depth $2N_t - 1$ is required for each node, in such a way that κ can satisfy

$$\mathbf{dyn}(t, G, v) = \kappa((T_v(i))_{i \in [t]}) = \kappa((T_v^{N_t}(i))_{i \in [t]}).$$

The required depth is a straight consequence of Theorem 4.9. The main idea behind the proof of Theorem 5.31 is to design a DGNN that can encode the sequence of attributed unfolding trees $(T_v(i))_{i \in [t]}$ into the node attributes at each timestep t , i.e., $\mathbf{q}_v(t) = \#_t((T_v(i))_{i \in [t]})$. This is achieved by using a coding function that maps sequences of $t + 1$ attributed trees into real numbers. To implement the encoding that could fit the definition of the DGNN, two coding functions are needed: the ∇ function, which encodes the attributed unfolding trees, and the family of coding functions $\#_t$. The composition of these functions is used to define the node's attributes, and the DGNN can produce the desired output by using this encoded information as follows:

$$\begin{aligned} \mathbf{q}_v(0) &= \mathbf{h}_v(0) = \#_0(\nabla^{-1}(\mathbf{h}_v(0))) \\ \mathbf{q}_v(t) &= \#_t(\text{APPEND}_t(\#_{t-1}^{-1}(\mathbf{q}_v(t-1)), \nabla^{-1}(\mathbf{h}_v(t)))) \end{aligned} \quad (5.9)$$

where the auxiliary function APPEND_t is defined similarly as in Chapter 4 and the ∇ , $\#_t$ coding functions are defined in the following.

APPEND_t

Let $\mathcal{T}^k(v)$ be the domain of the attributed unfolding trees with root v , up to a certain

depth d . The function

$\text{APPEND}_t : \{(T_v^k(i))_{i \in [t-1]}\} \cup \emptyset \times \mathcal{T}^k(v) \rightarrow \{(T_v^k(i))_{i \in [t]}\}$ is defined as follows:

$$\begin{aligned} \text{APPEND}_0(\emptyset, T_v^k(0)) &:= T_v^k(0) \\ \text{APPEND}_t((T_v^k(0), \dots, T_v^k(t-1)), T_v^k(t)) \\ &:= (T_v^k(0), \dots, T_v^k(t-1), T_v^k(t)) \end{aligned}$$

Intuitively, this function appends the unfolding tree snapshot of the node v at time t to the sequence of the unfolding trees of that node at the previous $t-1$ timesteps.

In the following, the coding functions are defined; their existence and injectiveness are provided by construction.

The ∇ Coding Function

Let $\nabla := \mu_\nabla \circ \nu_\nabla$ be a composition of any two injective functions μ_∇ and ν_∇ with the following properties:

- μ_∇ is an injective function from the domain of static unfolding trees, calculated on the nodes in the graph G_t , to the Cartesian product $\mathbb{N} \times \mathbb{N}^P \times \mathbb{Z}^A = \mathbb{N}^{P+1} \times \mathbb{Z}^A$, where P is the maximum number of nodes a tree could have.

Intuitively, in the Cartesian product, \mathbb{N} represents the tree structure, \mathbb{N}^P denotes the node numbering, while, for each node, an integer vector in \mathbb{Z}^A is used to encode the node attributes. Notice that μ_∇ exists and is injective since the maximal information contained in an unfolding tree is given by the union of all its node attributes and all its structural information, which just equals the dimension of the codomain of μ_∇ .

- ν_∇ is an injective function from $\mathbb{N}^{P+1} \times \mathbb{Z}^A$ to \mathbb{R} , whose existence is guaranteed by the cardinality theory, since the two sets have the same cardinality.

Since μ_{∇_t} and ν_{∇_t} are injective, also the existence and the injectiveness of ∇_t is ensured.

The $\#_t$ Coding Family

Similarly to ∇ , the functions $\#_t := \mu_{\#_t} \circ \nu_{\#_t}$ are composed by two functions $\mu_{\#_t}$ and $\nu_{\#_t}$ with the following properties:

- $\mu_{\#_t}$ is an injective function from the domain of the dynamic unfolding trees $\mathcal{T}_t^k(v) := \{(T_v^k(i))_{i \in [t]}\}$ to the Cartesian product $\mathbb{N}^t \times \mathbb{N}^{tP_t} \times \mathbb{Z}^{tA} = \mathbb{N}^{t(P_t+1)} \times \mathbb{Z}^{tA}$, where P_t is the maximum number of nodes a tree could have at time t .
- $\nu_{\#_t}$ is an injective function from $\mathbb{N}^{t(P_t+1)} \times \mathbb{Z}^{tA}$ to \mathbb{R} , whose existence is guaranteed by the cardinality theory, since the two sets have the same cardinality.

Since $\mu_{\#_t}$ and $\nu_{\#_t}$ are injective, also the existence and the injectiveness of $\#_t$ are ensured.

The recursive function \mathbf{f} , $\text{AGGREGATE}_t^{(k)}$, $\text{COMBINE}_t^{(k)}$

The recursive function \mathbf{f} has to satisfy

$$\mathbf{f}(\mathbf{q}_v(t-1), \mathbf{h}_v(t)) = \#_t((T_v(i))_{i \in [t]}) = \mathbf{q}_v(t),$$

where the $\mathbf{h}_v(t)$ is the hidden representation of node v at time t extracted from the t -th SGNN, i.e., $\mathbf{h}_v(t) = \text{SGNN}(G_t, v)$. In particular, at each iteration i , we have

$$\mathbf{h}_v^k(t) = \text{COMBINE}_t^{(k)} \left(\mathbf{h}_v^{k-1}(t), \text{AGGREGATE}_t^{(k)} \left(\{\{\mathbf{h}_u^{k-1}(t)\}_{u \in \text{ne}[v](t)}\}, \{\{\boldsymbol{\omega}_{\{u,v\}}(t)\}_{u \in \text{ne}[v](t)}\} \right) \right)$$

Further, the functions $\text{AGGREGATE}_t^{(k)}$ and $\text{COMBINE}_t^{(k)}$ – following the proof in [50] – must satisfy

$$\begin{aligned} \nabla(T_v^k(t)) &= \mathbf{h}_v^k(t) = \\ &\text{COMBINE}_t^{(k)} \left(\mathbf{h}_v^{k-1}(t), \text{AGGREGATE}_t^{(k)} \left(\{\{\mathbf{h}_u^{k-1}(t)\}_{u \in \text{ne}[v](t)}\}, \{\{\boldsymbol{\omega}_{\{u,v\}}(t)\}_{u \in \text{ne}[v](t)}\} \right) \right) \\ &= \text{COMBINE}_t^{(k)} \left(\nabla(T_v^{k-1}(t)), \text{AGGREGATE}_t^{(k)} \left(\{\{\nabla(T_u^{k-1}(t))\}_{u \in \text{ne}[v](t)}\} \right) \right) \end{aligned}$$

$\forall k \leq 2N - 1$ and $\forall t \in I$.

For example, the trees can be collected into the coding of a new tree, i.e.,

$$\text{AGGREGATE}_t^{(k)}(\nabla(T_u^{k-1}(t)), u \in \text{ne}[v](t)) = \nabla(\cup_{u \in \text{ne}[v](t)} \nabla^{-1}(\nabla(T_u^{k-1}(t)))) ,$$

where $\cup_{u \in \text{ne}[v](t)}$ denotes an operator that constructs a tree with a root having void attributes from a set of subtrees (see Fig. 4.5). Then, $\text{COMBINE}_t^{(k)}$ assigns the correct attributes to the root by extracting them from $T_v^{k-1}(t)$, i.e.,

$$\text{COMBINE}_t^{(k)}(\nabla(T_v^{k-1}(t)), b) = \nabla(\text{ATTACH}(\nabla^{-1}(\nabla(T_v^{k-1}(t))), \nabla^{-1}(b))),$$

where ATTACH is the operator defined as shown in Figure 4.5 in Chapter 4.

Now, notice that, with this definition, $\text{AGGREGATE}_t^{(k)}$, $\text{COMBINE}_t^{(k)}$, and $\text{READOUT}_{\text{dyn}}$ may not be differentiable. Nevertheless, Eq. (5.8) has to be satisfied only for a finite number of graphs, namely G_j . Thus, we can specify other functions $\overline{\text{AGGREGATE}}_t^{(k)}$, $\overline{\text{COMBINE}}_t^{(k)}$, and $\overline{\text{READOUT}}$, which produce exactly the same computations when they are applied on the graphs G_j , but that can be extended to the rest of their domain so that they are continuously differentiable. Obviously, such an extension exists since those functions are only constrained to interpolate a finite number of points [1].

The $\text{READOUT}_{\text{dyn}}$ function

Eventually, $\text{READOUT}_{\text{dyn}}$ must satisfy:

$$\boldsymbol{\kappa}(\cdot) := \text{READOUT}_{\text{dyn}}(\#_t(\cdot))$$

so that, ultimately,

$$\begin{aligned} \text{dyn}(t, G, v) &= \\ &\text{READOUT}_{\text{dyn}}(\#_t(\text{APPEND}_t(\#_{t-1}^{-1}(\mathbf{q}_v(t-1)), \nabla^{-1}(\mathbf{h}_v(t)))))) \end{aligned}$$

□

¹Notice that a similar extension can also be applied to the coding function ∇ and to the decoding function ∇^{-1} . In this case, the coding function is not injective on the whole domain, but only on the graphs mentioned in the theorem.

The equivalence between Theorem 5.29 and Theorem 5.31 is formally proven by the following Theorem.

Theorem 5.32. *Theorem 5.29 holds if and only if Theorem 5.31 holds.*

Proof. The proof is similar to the one contained in [1]. Nevertheless, we want to highlight that in this case, patterns are taken from $\mathcal{D} := I \times \mathcal{G} \times \mathcal{V}$, as we are proving it in the context of the dynamic graphs. \square

Figure 5.4 summarize the structure of the proof of Theorem 5.29.

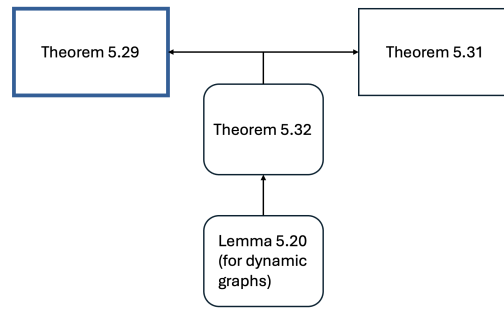


Figure 5.4: Structure of the proof of Theorem 5.29.

Theorem 5.29 intuitively states that, given a dynamical system \mathbf{dyn} , there is a DGNN that approximates it. The functions which the DGNN is a composition of (such as the dynamical function \mathbf{f} , $\text{COMBINE}^{(k)}$, $\text{AGGREGATE}^{(k)}$, etc.) are supposed to be continuously differentiable, but their analytical formulation is not defined. This situation does not correspond to practical cases where the DGNN adopts particular architectures, and those functions are Neural Networks, or more generally, parametric models – for example, made of layers of sum, max, average, etc. Thus, it is of fundamental interest to clarify whether the theorem still holds when the components of the DGNN are parametric models.

Definition 5.33. A class $\mathcal{Q}_{\mathcal{D}}$ of discrete DGNN models is said to have *universal components* if the employed SGNNs have universal components as defined in Def. 5.24 and the employed recurrent model is designed such that for any $\epsilon_1, \epsilon_2 > 0$ and any continuously differentiable target functions $\bar{\mathbf{f}}, \overline{\text{READOUT}}_{\mathbf{dyn}}$ there is a discrete DGNN in the class $\mathcal{Q}_{\mathcal{D}}$, with functions $\mathbf{f}_{\theta}, \text{READOUT}_{\mathbf{dyn},\theta}$ and parameters θ such that, for any input vectors $\mathbf{h} \in \mathbb{R}^d, \mathbf{q}, \mathbf{q}^* \in \mathbb{R}^s$, it holds

$$\begin{aligned} \|\bar{\mathbf{f}}(\mathbf{q}, \mathbf{h}) - \mathbf{f}_{\theta}(\mathbf{q}, \mathbf{h})\|_{\infty} &\leq \epsilon_1, \\ \|\overline{\text{READOUT}}_{\mathbf{dyn}}(\mathbf{q}^*) - \text{READOUT}_{\mathbf{dyn},\theta}(\mathbf{q}^*)\|_{\infty} &\leq \epsilon_2. \end{aligned}$$

Then we can show that Theorem 5.29 still holds even for discrete DGNNs with universal components.

Theorem 5.34 (Approximation by Neural Networks). *Assume that the hypotheses of Thm. 5.29 are fulfilled and $\mathcal{Q}_{\mathcal{D}}$ is a class of discrete DGNNs with universal components.*

Then, there exists a parameter set θ , and the functions $\bar{\mathbf{f}}$, $\overline{\text{READOUT}}_{\text{dyn}}$, implemented by Neural Networks in $\mathcal{Q}_{\mathcal{D}}$, such that Theorem 5.29 holds.

Proof. The idea of the proof follows from the same reasoning adopted for Theorem 4.17 in Chapter 4. Intuitively, since the discrete DGNN of Thm. 5.29 is implemented by continuously differentiable functions, its output depends continuously on the possible changes in the DGNN implementation: small changes in the function implementation cause small changes in the DGNN outputs. Therefore, the functions of the DGNN of Thm. 5.29 can be replaced by Neural Networks, provided that those networks are suitable approximators.

As in the proof of the dynamic version of the approximation theorem, without loss of generality, we will assume that the attribute dimension is $d = 1^2$.

First of all, note that Thm. 5.29 ensures that we can find continuously differentiable functions $\bar{\mathbf{f}}$, $\overline{\text{READOUT}}_{\text{dyn}}$ such that, for the corresponding function $\bar{\varphi}$ implemented by the DGNN it holds:

$$P(\|\text{dyn}(t, G, v) - \bar{\varphi}(t, G, v)\| \leq \frac{\epsilon}{2}) \geq 1 - \lambda \quad \forall t \in I, \epsilon, \lambda > 0. \quad (5.10)$$

Considering that the theorem has to hold only in probability, we can also assume that the domain is bounded to a finite set of patterns $\{(t^{(k)}, (G_t)_{t \in I}^{(k)}, v^{(k)}) \mid i = 1, \dots, p\}$ (as in Theorem 5.31). As a result, the functions $\bar{\mathbf{f}}$ and $\overline{\text{READOUT}}_{\text{dyn}}$ are bounded and have a bounded Jacobian. We can take the maximum of these Jacobians, which we will denote as B .

Moreover, let \mathbf{f}_{θ} , $\text{READOUT}_{\text{dyn}, \theta}$ be universal components for DGNN, as in Def. 5.33 that approximate $\bar{\mathbf{f}}$, $\overline{\text{READOUT}}_{\text{dyn}}$, respectively. Further, let $\epsilon_1, \epsilon_2, > 0$ be the corresponding approximation errors, i.e.,

$$\begin{aligned} \|\bar{\mathbf{f}}(\mathbf{q}, \mathbf{h}) - \mathbf{f}_{\theta}(\mathbf{q}, \mathbf{h})\|_{\infty} &\leq \epsilon_1, \text{ and} \\ \|\overline{\text{READOUT}}_{\text{dyn}}(\mathbf{Q}(t)) - \text{READOUT}_{\text{dyn}, \theta}(\mathbf{Q}(t))\|_{\infty} &\leq \epsilon_2 \end{aligned} \quad (5.11)$$

hold $\forall t \in I$.

Now, from the proof of Theorem 5.25 we know that

$$P(\|\overline{\text{SGNN}}_i(G, v) - \text{SGNN}_{\theta, i}(G, v)\| \leq \epsilon_s) \geq 1 - \lambda_i$$

for $i \in [t]$, $\epsilon_s > 0$ and for any norm. Then we can take every λ_i small enough, s.t.

$$\|\overline{\text{SGNN}}_i(G, v) - \text{SGNN}_{\theta, i}(G, v)\|_{\infty} \leq \epsilon_s$$

holds on a finite set of patterns large enough to include those ones of the i -th timestep of each patterns of dynamic graphs on which Eq. (5.10) holds.

Therefore, if we define $\bar{\mathbf{h}}(t) := \overline{\text{SGNN}}_i(G_t)$ and $\mathbf{h}_{\theta}(t) := \text{SGNN}_{\theta, i}(G_t)$ we have

$$\|\bar{\mathbf{h}}(t) - \mathbf{h}_{\theta}(t)\|_{\infty} = \|\overline{\text{SGNN}}_i(G_t) - \text{SGNN}_{\theta, i}(G_t)\|_{\infty} \leq \epsilon_s.$$

²A GNN can theoretically be modeled with multiple components by stacking Neural Networks for each dimension, respectively.

In addition, let $\bar{\mathbf{H}}(t)$ and $\mathbf{H}_\theta(t)$ be the internal representations produced by $\overline{\text{SGNN}}$ and SGNN_θ , stacked over all the nodes of the input graph. Then it holds

$$\|\bar{\mathbf{H}}(t) - \mathbf{H}_\theta(t)\|_\infty \leq N\epsilon_s \quad \forall t \in I, \quad (5.12)$$

where $N = \max_{G \in \mathcal{G}} |G|$ is the maximum number of nodes of the static graphs input in the bounded domain. Let again $\bar{\mathbf{Q}}(0) := \bar{\mathbf{H}}(0)$ and $\bar{\mathbf{Q}}(t) := \bar{F}(\bar{\mathbf{Q}}(t-1), \bar{\mathbf{H}}(t))$ be the stacking of the internal states produced by DGNN's internal recursive function $\bar{\mathbf{f}}$. Analogously, let $\mathbf{Q}_\theta(0) := \mathbf{H}_\theta(0)$ and $\mathbf{Q}_\theta(t) := \bar{F}_\theta(\mathbf{Q}_\theta(t-1), \mathbf{H}_\theta(t))$ be the output produced by the corresponding function of the parameterized DGNN.

Then it holds:

$$\|\bar{\mathbf{Q}}(0) - \mathbf{Q}_\theta(0)\|_\infty = \|\bar{\mathbf{H}}(0) - \mathbf{H}_\theta(0)\|_\infty \leq N\epsilon_s \quad (5.13)$$

and

$$\begin{aligned} \|\bar{\mathbf{f}}(\bar{\mathbf{Q}}(0), \cdot) - \bar{\mathbf{f}}(\mathbf{Q}_\theta(0), \cdot)\|_\infty &\leq B\|\bar{\mathbf{Q}}(0) - \mathbf{Q}_\theta(0)\|_\infty \\ \|\bar{\mathbf{f}}(\cdot, \bar{\mathbf{H}}(1)) - \bar{\mathbf{f}}(\cdot, \mathbf{H}_\theta(1))\|_\infty &\leq B\|\bar{\mathbf{H}}(1) - \mathbf{H}_\theta(1)\|_\infty \end{aligned}$$

for a bound B on the Jacobian of $\bar{\mathbf{f}}(\mathbf{q}, \mathbf{h}) \quad \forall t \in I$ and $\forall \mathbf{q}$, which, along with Eq. (5.12) and Eq. (5.13) gives

$$\begin{aligned} \|\bar{\mathbf{f}}(\bar{\mathbf{Q}}(0), \cdot) - \bar{\mathbf{f}}(\mathbf{Q}_\theta(0), \cdot)\|_\infty &\leq N\epsilon_s B \\ \|\bar{\mathbf{f}}(\cdot, \bar{\mathbf{H}}(1)) - \bar{\mathbf{f}}(\cdot, \mathbf{H}_\theta(1))\|_\infty &\leq N\epsilon_s B \end{aligned} \quad (5.14)$$

Therefore, we have that:

$t = 1$:

$$\begin{aligned} &\|\bar{\mathbf{Q}}(1) - \mathbf{Q}_\theta(1)\|_\infty \\ &= \|\bar{\mathbf{f}}(\bar{\mathbf{Q}}(0), \bar{\mathbf{H}}(1)) - \bar{\mathbf{f}}(\mathbf{Q}_\theta(0), \mathbf{H}_\theta(1))\|_\infty \\ &\stackrel{\text{add } 0}{=} \|\bar{\mathbf{f}}(\bar{\mathbf{Q}}(0), \bar{\mathbf{H}}(1)) - \bar{\mathbf{f}}(\mathbf{Q}_\theta(0), \bar{\mathbf{H}}(1)) \\ &\quad + \bar{\mathbf{f}}(\mathbf{Q}_\theta(0), \bar{\mathbf{H}}(1)) - \bar{\mathbf{f}}(\mathbf{Q}_\theta(0), \mathbf{H}_\theta(1)) \\ &\quad + \bar{\mathbf{f}}(\mathbf{Q}_\theta(0), \mathbf{H}_\theta(1)) - \bar{\mathbf{f}}(\mathbf{Q}_\theta(0), \mathbf{H}_\theta(1))\|_\infty \\ &\stackrel{\Delta\text{-ineq.}}{\leq} \|\bar{\mathbf{f}}(\bar{\mathbf{Q}}(0), \bar{\mathbf{H}}(1)) - \bar{\mathbf{f}}(\mathbf{Q}_\theta(0), \bar{\mathbf{H}}(1))\|_\infty \\ &\quad + \|\bar{\mathbf{f}}(\mathbf{Q}_\theta(0), \bar{\mathbf{H}}(1)) - \bar{\mathbf{f}}(\mathbf{Q}_\theta(0), \mathbf{H}_\theta(1))\|_\infty \\ &\quad + \|\bar{\mathbf{f}}(\mathbf{Q}_\theta(0), \mathbf{H}_\theta(1)) - \bar{\mathbf{f}}(\mathbf{Q}_\theta(0), \mathbf{H}_\theta(1))\|_\infty \\ &\stackrel{(5.14)}{\leq} 2N\epsilon_s B + N\epsilon_1 \\ &:= \lambda_1(\epsilon_s, \epsilon_1). \end{aligned}$$

$t > 0$: Analogously, it follows for $t > 1$ that

$$\begin{aligned}
& \|\bar{\mathbf{Q}}(t) - \mathbf{Q}_\theta(t)\|_\infty \\
&= \|\bar{\mathbf{f}}(\bar{\mathbf{Q}}(t-1), \bar{\mathbf{H}}(t)) - \mathbf{f}_\theta(\mathbf{Q}_\theta(t-1), \mathbf{H}_\theta(t))\|_\infty \\
&= \|\bar{\mathbf{f}}(\bar{\mathbf{Q}}(t-1), \bar{\mathbf{H}}(t)) - \bar{\mathbf{f}}(\mathbf{Q}_\theta(t-1), \bar{\mathbf{H}}(t)) \\
&\quad + \bar{\mathbf{f}}(\mathbf{Q}_\theta(t-1), \bar{\mathbf{H}}(t)) - \bar{\mathbf{f}}(\mathbf{Q}_\theta(t-1), \mathbf{H}_\theta(t)) \\
&\quad + \bar{\mathbf{f}}(\mathbf{Q}_\theta(t-1), \mathbf{H}_\theta(t)) - \mathbf{f}_\theta(\mathbf{Q}_\theta(t-1), \mathbf{H}_\theta(t))\|_\infty \\
&\leq \|\bar{\mathbf{f}}(\bar{\mathbf{Q}}(t-1), \bar{\mathbf{H}}(t)) - \bar{\mathbf{f}}(\mathbf{Q}_\theta(t-1), \bar{\mathbf{H}}(t))\|_\infty \\
&\quad + \|\bar{\mathbf{f}}(\mathbf{Q}_\theta(t-1), \bar{\mathbf{H}}(t)) - \bar{\mathbf{f}}(\mathbf{Q}_\theta(t-1), \mathbf{H}_\theta(t))\|_\infty \\
&\quad + \|\bar{\mathbf{f}}(\mathbf{Q}_\theta(t-1), \mathbf{H}_\theta(t)) - \mathbf{f}_\theta(\mathbf{Q}_\theta(t-1), \mathbf{H}_\theta(t))\|_\infty \\
&\leq N\lambda_0 B + N\epsilon_s B + N\epsilon_1 \\
&:= \lambda_1(\epsilon_s, \epsilon_1).
\end{aligned}$$

The above reasoning can then be applied recursively to prove that

$$\|\bar{\mathbf{Q}}(t) - \mathbf{Q}_\theta(t)\|_\infty \leq \lambda_t(\epsilon_s, \epsilon_1),$$

where $\lambda_t(\epsilon_s, \epsilon_1)$ could be found as little as possible, according to ϵ_s, ϵ_1 . Finally, let $\epsilon_2 > 0$, so that

$$\begin{aligned}
& \|\bar{\varphi}(t, G, v) - \varphi_\theta(t, G, v)\|_\infty \\
&= \|\overline{\text{READOUT}}_{\text{dyn}}(\bar{\mathbf{Q}}(t)) - \text{READOUT}_{\text{dyn}, \theta}(\mathbf{Q}_\theta(t))\|_\infty \\
&\leq \|\overline{\text{READOUT}}_{\text{dyn}}(\bar{\mathbf{Q}}(t)) - \overline{\text{READOUT}}_{\text{dyn}}(\mathbf{Q}_\theta(t))\|_\infty \\
&\quad + \|\overline{\text{READOUT}}_{\text{dyn}}(\mathbf{Q}_\theta(t)) - \text{READOUT}_{\text{dyn}, \theta}(\mathbf{Q}_\theta(t))\|_\infty \\
&\leq \lambda_t B + \epsilon_2 = \lambda(\epsilon_s, \epsilon_1, \epsilon_2).
\end{aligned}$$

Thus, we choose $\epsilon_s, \epsilon_1, \epsilon_2$, s.t. $\lambda \leq \frac{\epsilon}{2}$; going back in probability, we obtain

$$P(\|\bar{\varphi}(t, G, v) - \varphi_\theta(t, G, v)\| \leq \frac{\epsilon}{2}) \geq 1 - \lambda \quad \forall t \in I,$$

which, along with Eq. (5.10), proves the result. \square

Remark 5.35. The following remarks may further help to understand the results proven in the previous paragraphs:

- Thm. 5.21 suggests an alternative approach to process several graph domains with a universal SGNN model. Actually, almost all the graphs, including, e.g., hypergraphs, multigraphs, directed graphs, etc., can be transformed to SAUHG with node and edge attributes [77]. Then, we can use a universal GNN model on such a domain using sufficiently expressive AGGREGATE and COMBINE functions.
- Thms. 5.21 and 5.29 specify that the approximation is modulo unfolding equivalence, or, correspondingly, modulo WL equivalence. It can be observed that in the dynamic case, only a part of the architecture affects the equivalence. Actually, a dynamic

GNN contains two modules: the first one, an SGNN, produces an embedding of the input graph at each time instance; the second component contains a Recurrent Neural Network that processes the sequence of the embeddings. The dynamic unfolding equivalence is defined by sequences of unfolding trees, which are built independently for each node and time instance by the SGNN. Similarly, the dynamic WL equivalence is defined by sequences of colors defined independently at each time step. Intuitively, the Recurrent Neural Network does not affect the equivalence, since Recurrent Neural Networks can be universal approximators and implement any function of the sequence without introducing other constraints beyond those already introduced by the SGNN.

- Thm. 5.29 does not hold for any Dynamic GNN, as we take into account a discrete recurrent model working on graph snapshots (also known as Stacked DGNN). Nevertheless, several DGNNs of this kind are listed in [76], such as GCRN-M1 [80], RgCNN [81], PATCHY-SAN [82], DyGGNN [83], and others. Still, the approximation capability depends on the functions AGGREGATE and COMBINE designed for each GNN working on the single snapshot and the implemented Recurrent Neural Network. For example, the most general model, the original RNN, has been proven to be a universal approximator [84].

5.4 Experimental Validation

In this Section, we support our theoretical findings with an experimental setup. For this purpose, we show that a DGNN can approximate a function $F_{DWL} : \mathcal{G} \rightarrow \mathbb{N}$ that models the 1-DWL test. The function F_{DWL} assigns to each dynamic graph a target label that represents the class of equivalence of the 1-DWL. We focus on the ability of the DGNN to approximate this function, so only training performances are considered, i.e., we do not investigate the generalization capabilities over a test set. Since the 1-DWL test provides the finest partition of graphs reachable by a DGNN, the mentioned task experimentally evaluates the expressive power of DGNNs.

Dataset. The dataset consists of dynamic graphs, i.e., vectors of static graph snapshots of fixed length T . Each static snapshot is one of the graphs in Fig. 5.5. Since the dataset is composed of all the possible combinations of the four graphs, it contains 4^T dynamic graphs. Given that the graphs in Fig. 5.5 are pairwise 1-WL equivalent (a) is 1-WL equivalent to b) and c) is 1-WL equivalent to d)), the number of classes is 2^T , with $\frac{4^T}{2^T} = 2^T$ graphs in each class. For each dynamic graph, the target is the corresponding 1-DWL output, represented as a natural number. For training purposes, the targets are normalized between 0 and 1 and uniformly spaced in the interval $[0, 1]$. Therefore, the distance between each class label is $d = 1/2^T$. A dynamic graph G with target y_G will be said to be correctly classified if, given $\text{out} = \text{DGNN}(G)$, we have $|\text{out} - y_G| < d/2$.

Experimental setup. The Dynamic Graph Neural Network used in the experiments is composed of two modules: a Graph Isomorphism Network (GIN) [13] and a Recurrent

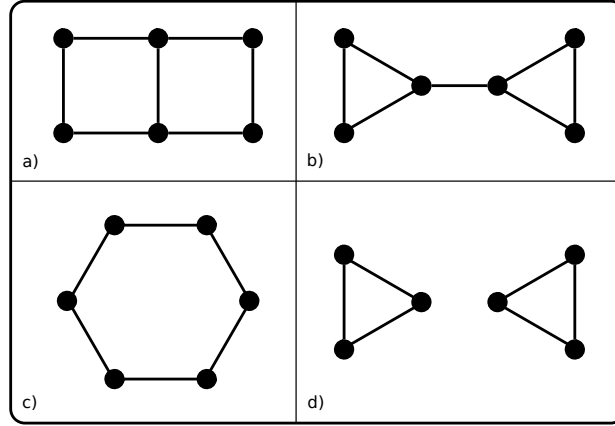


Figure 5.5: The four static graphs used as components to generate the synthetic dataset. Graphs a) and b) are equivalent under the static 1-WL test; same holds for c) and d).

Neural Network (RNN), which together implement the static GNN and the temporal network f of Eq. (5.1), respectively. Since it has been proven that the GIN is a universal architecture [13] and the RNNs are universal approximators for dynamical systems on vector sequences [84], the architecture used in the experiments fits the hypothesis of Thm. 5.29. Thus, it can approximate any dynamic system of the temporal graph domain. The MLP in the GIN has one hidden layer and the output layer of same size, which is $h_{\text{gin}} \in \{1, 4, 8\}$ with the hyperbolic tangent as activation function and batch normalization. The GIN includes 6 message passing layers³; for graph-focused tasks, it is sufficient to perform the message-passing convolution for several times equal to the maximum number of nodes over the graphs in the dataset domain. After the last GIN layer, a linear layer $W_{\text{gin_out}}$ of size $h_{\text{gin}} \times h_{\text{gin_out}}$ is applied, where $h_{\text{gin_out}}$ is fixed equal to 8. Furthermore, $h_{\text{rnn}} = 8$ is the size of the hidden state of the RNN. The model is trained over 300 epochs using the Adam optimizer with a learning rate $\lambda = 10^{-3}$. Each setting is therefore described by fixing the GIN hidden layer h_{gin} and the time length T of the samples in the synthetic dataset described above and is evaluated over 10 runs. The overall training is then performed on an Intel(R) Core(TM) i7-9800X processor running at 3.80GHz using 31GB of RAM and a GeForce GTX 1080 Ti GPU unit⁴.

Results. Our experimental results are summarized in Figure 5.6.

The evolution of the training accuracy over the epochs for different GIN hidden layer sizes (and consequently, for different hidden representation sizes) h_{gin} and for dynamic graphs up to time lengths of $T = 4$ and $T = 5$ is presented. All the architectures statistically reach 100% accuracy for experiments on both time lengths. Even with fixing $h_{\text{gin}} = 1$ this DGNN achieves perfect classification at a slower rate. It may appear surprising that, even with a hidden representation of size 1, the DGNN can well approximate the function

³As investigated in Subsection 5.35

⁴Code available at <https://github.com/AleDinve/dyn-gnn>.

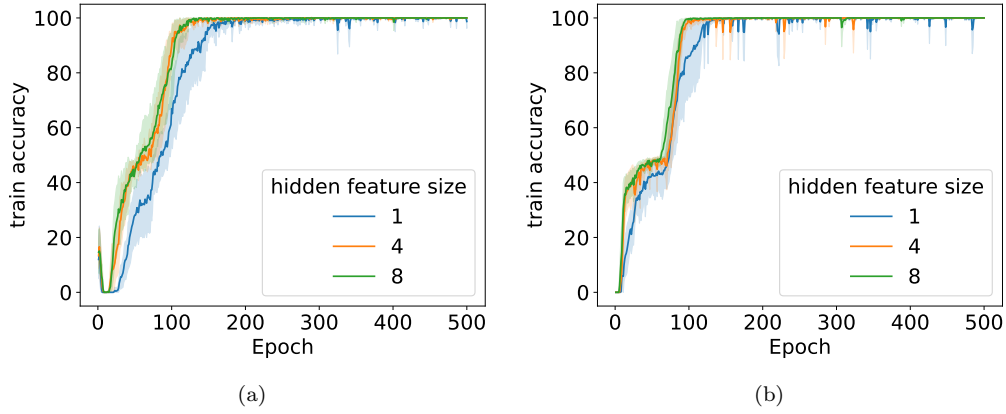
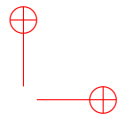
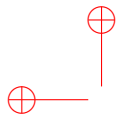


Figure 5.6: Train accuracy over the epochs for a DGNN trained on the dataset containing dynamic graphs up to time length $T = 4$ (a) and $T = 5$ (b).

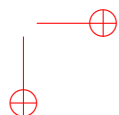
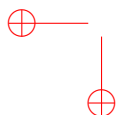
F_{DWL} . However, as we already pointed out in Section [5.3](#), the possibility of reaching the universal approximation with a feature of dimension 1 is confirmed by Thm. [5.29](#).

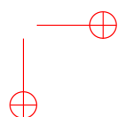
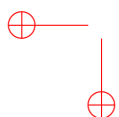
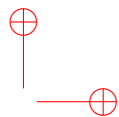
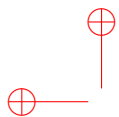
In the upcoming chapters, we will move our focus from the approximation power of GNNs to the theoretical analysis of their generalization capabilities, which is a fundamental aspect of every neural network model. In more detail, in the next chapter we will present new bounds on the VC dimension, a measure of complexity commonly used to assess the generalization capacity, for modern GNNs.

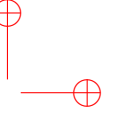
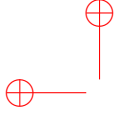


Part III

Generalization capabilities of Graph Neural Networks







Chapter 6

VC dimension of message passing GNNs with Pfaffian activation functions

Message passing GNNs with common activation functions such as hyperbolic tangent, sigmoid and arctangent still lack of a characterization in terms of VC dimension.

This chapter aims to close this gap, providing new bounds for modern message passing GNNs with Pfaffian activation functions. *Pfaffian functions* are a large class of differentiable maps, which includes activation functions like `tanh`, `logsig`, `atan`, and, more generally, most of the function used in Engineering having continuous derivatives up to any order.

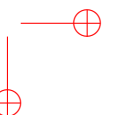
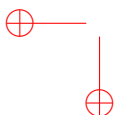
The contributions of this chapter are schematized below.

- We provide upper bounds for message passing GNNs with Pfaffian activation functions with respect to the main hyperparameters, such as the feature dimension, the hidden feature size, the number of message passing layers implemented and the total number of nodes in the training domain. To prove these results we exploit theoretical results in the literature that link the theory of Pfaffian functions and the characterization of the VC dimension of the model via topological analysis.
- We also study the trend of the VC dimension w.r.t. the colors in the dataset obtained by running the WL test. The theoretical result suggests that the number of colors have an important effect on the GNN generalization capability. On one hand, a large total number of colors in the training set improves generalization, since it increases the examples available for learning; on the other hand, a large number of colors in each graph raises the VC dimension and therefore it increases the empirical risk value.
- Our theoretical findings are experimentally assessed by a preliminary experimental study; specifically, we evaluate the gap between the predictive performance on the training and test data.

The chapter is organized as follows. In Section [6.1](#), we introduce the main concepts and the notation used. In Section [6.2](#), we state and discuss our main theoretical results. Then, in Section [6.3](#), we report the experimental validation of our achievements.

6.1 Notation and preliminaries

VC dimension — The Vapnik–Chervonenkis (VC) dimension is a measure of complexity of a hypothesis set, which can be used to bound the empirical error of machine learning



models. Formally, a binary classifier \mathcal{L} with parameters $\boldsymbol{\theta}$ is said to *shatter* a set of patterns $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ if, for any binary labeling of the examples $\{y_i\}_{i=1, \dots, n}$, $y_i \in \{0, 1\}$, there exists $\boldsymbol{\theta}$ s.t. the model \mathcal{L} correctly classifies all the patterns, i.e. $\sum_{i=1}^n |\mathcal{L}(\boldsymbol{\theta}, \mathbf{x}_i) - y_i| = 0$. The *VC dimension* of the model \mathcal{L} is the dimension of the largest set that \mathcal{L} can shatter.

The VC dimension has been linked with the generalization capability of machine learning models. Actually, given a training set and a test set for the classifier \mathcal{L} , whose patterns are i.i.d. samples extracted from the same distribution, the VC dimension allows to compute a bound on the difference between the training and test error. Formally, it has been proved [85] that, for any $\eta > 0$,

$$\Pr(\text{test error} \leq \text{training error} + \sqrt{\frac{1}{N} [\text{VCdim}(\log(\frac{2N}{\sqrt{\text{VCdim}}}) + 1) - \log(\frac{\eta}{4})]}) = 1 - \eta \quad (6.1)$$

holds, where N is the size of the training dataset and VCdim is the VC dimension of \mathcal{L} .

Pfaffian Functions Formally, a Pfaffian chain of order $\ell \geq 0$ and degree $\alpha \geq 1$ in an open domain $U \subseteq \mathbb{R}^n$ is a sequence of analytic functions $\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_\ell$ in U satisfying the differential equations

$$d\mathbf{f}_j(\mathbf{x}) = \sum_{1 \leq i \leq n} g_{ij}(\mathbf{x}, \mathbf{f}_1(\mathbf{x}), \dots, \mathbf{f}_j(\mathbf{x})) dx_i$$

for $1 \leq j \leq \ell$. Here, $g_{ij}(\mathbf{x}, y_1, \dots, y_j)$ are polynomials in $\mathbf{x} \in U$ and $y_1, \dots, y_j \in \mathbb{R}$ of degree not exceeding α . A function $f(\mathbf{x}) = P(\mathbf{x}, \mathbf{f}_1(\mathbf{x}), \dots, \mathbf{f}_\ell(\mathbf{x}))$, where $P(\mathbf{x}, y_1, \dots, y_\ell)$ is a polynomial of degree not exceeding β , is called a *Pfaffian function* of *format* $\text{format}(f) = (\alpha, \beta, \ell)$.

Pfaffian maps are a large class of functions that includes most of the functions with continuous derivatives used in practical applications [86]. Note that the arctangent atan , the logistic sigmoid logsig and the hyperbolic tangent tanh are Pfaffian functions, with *format* $\text{format}(\text{atan}) = (3, 1, 2)$, $\text{format}(\text{logsig}) = (2, 1, 1)$, and $\text{format}(\text{tanh}) = (2, 1, 1)$, respectively.

6.1.1 Results from the literature

In the following, we will recall some notation and some results from the literature that are crucial to the development of the proof of the main results of this chapter.

Let $\tau_1, \dots, \tau_{\bar{s}}$ be a set of C^∞ infinitely differentiable functions from $\mathbb{R}^{p+\gamma}$ to \mathbb{R} . Suppose that $\Phi(\mathbf{y}, \boldsymbol{\theta}), \mathbf{y} \in \mathbb{R}^\gamma, \boldsymbol{\theta} \in \mathbb{R}^p$ is a quantifier-free logical formula constructed using the logical "and" and atoms in the form of $\tau_i(\mathbf{y}, \boldsymbol{\theta}) = 0$. Note that, fixed $\boldsymbol{\theta}$, $\Phi(\cdot, \boldsymbol{\theta})$ takes as input a vector \mathbf{y} and returns a logical value, so that it can be used as a classifier with input \mathbf{y} and parameters $\boldsymbol{\theta}$. Later, we will see that $\tau_1, \dots, \tau_{\bar{s}}$ can be specified so that $\Phi(\cdot, \boldsymbol{\theta})$ defines the computation of a GNN. Moreover, its VC dimension can be easily defined. In fact, Φ is said to shatter a set $\mathcal{S} = \{\bar{\mathbf{y}}_1, \dots, \bar{\mathbf{y}}_r\}$ if, for any set of binary assignments $\delta = [\delta_1, \dots, \delta_r] \in \{0, 1\}^r$, there exist parameters $\boldsymbol{\theta}$ such that, for any i , $\Phi(\bar{\mathbf{y}}_i, \boldsymbol{\theta})$ is true if $\delta_i = 1$, and $\Phi(\bar{\mathbf{y}}_i, \boldsymbol{\theta})$ is false if $\delta_i = 0$. Then, the VC dimension of Φ is defined as the size

of the maximum set that Φ can shatter, i.e.,

$$\text{VCdim}(\Phi) = \max_{\mathcal{S} \text{ is shattered by } \Phi} |\mathcal{S}|$$

Interestingly, it can be shown that the VC dimension of $\Phi(\cdot, \boldsymbol{\theta})$ can be studied by analysing the topological properties of the inverse image of the components of $\Phi(\mathbf{y}, \cdot)$ for every input \mathbf{y} . More precisely, let $\bar{\mathbf{y}}_1, \dots, \bar{\mathbf{y}}_z$ be vectors in $\mathbb{R}^{\bar{\gamma}}$, and $T : \mathbb{R}^{\bar{p}} \rightarrow \mathbb{R}^{\bar{u}}$, $\bar{u} \leq \bar{p}$, be defined as

$$T(\bar{\boldsymbol{\theta}}) = [\bar{\tau}_1(\bar{\boldsymbol{\theta}}), \dots, \bar{\tau}_{\bar{u}}(\bar{\boldsymbol{\theta}})], \quad (6.2)$$

where $\bar{\tau}_1(\bar{\boldsymbol{\theta}}), \dots, \bar{\tau}_{\bar{u}}(\bar{\boldsymbol{\theta}})$ are functions of the form $\tau_i(\bar{\mathbf{y}}_j, \bar{\boldsymbol{\theta}})$, i.e., for each r , $1 \leq r \leq \bar{u}$, there exist integers i and j such that $\bar{\tau}_r(\bar{\boldsymbol{\theta}}) = \tau_i(\bar{\mathbf{y}}_j, \bar{\boldsymbol{\theta}})$. Let $[\epsilon_1, \dots, \epsilon_{\bar{u}}]$ be a regular value of T ¹ and assume that there exists a positive integer B that bounds the number of connected components of $T^{-1}(\epsilon_1, \dots, \epsilon_{\bar{u}})$ and does not depend on the chosen $\epsilon_1, \dots, \epsilon_{\bar{u}}$ and on the $\bar{\mathbf{y}}_j$ selected to be contained in the $\bar{\tau}_i$. Then, the following theorem holds ([87]).

Theorem 6.1 ([87]). *The VC dimension of Φ is bounded as follows:*

$$\text{VCdim}(\Phi) \leq 2 \log B + \bar{p}(16 + 2 \log \bar{s}).$$

The theorem provides a bound for the VC dimension that depends on the number \bar{p} of parameters, on the total number \bar{s} of functions τ_i , and on B , which may further depend on the number of parameters.

Then, a bound on B can be obtained based on the literature. The following result provides a bound for equations of Pfaffian functions.

Theorem 6.2 ([88]). *Consider a system of equations $\bar{q}_1(\boldsymbol{\theta}) = 0, \dots, \bar{q}_k(\boldsymbol{\theta}) = 0$, where \bar{q}_i , $1 \leq i \leq k$, are Pfaffian functions in a domain $\mathcal{P} \subseteq \mathbb{R}^{\bar{p}}$, having a common Pfaffian chain of length $\bar{\ell}$ and maximum degrees $(\bar{\alpha}, \bar{\beta})$. Then the number of connected components of the set $\{\boldsymbol{\theta} | \bar{q}_1(\boldsymbol{\theta}) = 0, \dots, \bar{q}_k(\boldsymbol{\theta}) = 0\}$ is bounded by*

$$2^{\frac{\bar{\ell}(\bar{\ell}-1)}{2}+1} (\bar{\alpha} + 2\bar{\beta} - 1)^{\bar{p}-1} ((2\bar{p} - 1)(\bar{\alpha} + \bar{\beta}) - 2\bar{p} + 2)^{\bar{\ell}}$$

6.2 Main Results

Our main result provides a bound on the VC dimension of GNNs in which COMBINE^(t), AGGREGATE^(t) and READOUT are Pfaffian functions. More precisely, we consider a slightly more general version of the GNN model in Eq. (2.1), where the updating scheme is

$$\mathbf{h}_v^{(t+1)} = \text{COMBINE}^{(t+1)}(\mathbf{h}_v^{(t)}, \text{AGGREGATE}^{(t+1)}(\{\{\mathbf{h}_u^{(t)} | u \in V\}, A_v)), \quad (6.3)$$

and A_v is the v -th column of the connectivity matrix, which represents the neighborhood of v . The advantage of the model in Eq. (6.3) is that it makes explicit the dependence of AGGREGATE^(t) on the graph connectivity. Actually, here we want to underline what the inputs of AGGREGATE^(t) are to clarify and make formally precise the assumptions that those functions are Pfaffian and have a given format.

¹We recall that $[\epsilon_1, \dots, \epsilon_{\bar{u}}]$ is a regular value of T if either $T^{-1}([\epsilon_1, \dots, \epsilon_{\bar{u}}]) = \emptyset$ or $T^{-1}([\epsilon_1, \dots, \epsilon_{\bar{u}}])$ is a $(\bar{p} - \bar{u})$ -dimensional C^∞ -submanifold of $\mathbb{R}^{\bar{p}}$.

6.2.1 Main bounds

Our main result, namely Theorem 6.4, provides a bound on the VC dimension of general GNNs with Pfaffian activation function and on GNNs with logistic sigmoid activation function w.r.t. the total number \bar{p} of parameters, the number of computation units H , the number of layers L , the feature dimension d , the maximum number N of nodes in a graph, and the attribute dimension q . Here, we assume that GNN computation units include the neurons computing the hidden features and the outputs. Thus, there is a computation unit for each component of a feature, each layer, each node of the input graph and a further computation unit for the READOUT.

The first step of the proof consists of defining a set of equations $\tau_i(\mathbf{y}, \boldsymbol{\theta}) = 0$ that specifies the computation of a GNN. Obviously, $\boldsymbol{\theta}$ includes just the GNN parameters, namely $\boldsymbol{\theta} = \Theta$. If COMBINE^(k) has p_{comb} parameters for $1 \leq k \leq L$, AGGREGATE^(k) has p_{agg} parameters for $1 \leq k \leq L$, and READOUT has p_{read} parameters, the total number of parameters is $L(p_{\text{comb}} + p_{\text{agg}}) + p_{\text{read}}$.

Moreover, intuitively, \mathbf{y} represents the input of a GNN, so that for a given graph $\mathbf{G} = (G, \mathbf{L})$ in \mathcal{G} each \mathbf{y} contains some vectorial representation of \mathbf{G} , namely the Nq graph attributes in \mathbf{L}_G , and a vectorial representation of the adjacency matrix, which requires $N(N-1)/2$ elements. Furthermore, to define the equations, we use the same trick as in [87] and we introduce new variables for each computation unit of the network. Those variables belongs to the input \mathbf{y} of τ . More precisely, we consider a vector of d variables $\mathbf{h}_v^{(k)}$ for each node v and for each layer k . Note that, since we may be interested in defining more computations of the GNN on more graphs at the same time, here v implicitly addresses a specific node of some graph in the domain. Finally, a variable READOUT for each graph contains just a single output of the GNN. In total, the dimension of \mathbf{y} is $\bar{p} = Nq + N(N-1)/2 + NdL + 1$.

Then, by (2.1), the computation of GNN is straightforwardly defined by the following set of $LNd + Nq + 1$ equations,

$$\mathbf{h}_v^{(0)} - \boldsymbol{\alpha}_v = 0 \quad (6.4)$$

$$\mathbf{h}_v^{(k)} - \text{COMBINE}^{(k+1)}(\mathbf{h}_v^{(k)}, \text{AGGREGATE}^{(k+1)}(\{\{\mathbf{h}_u^{(k)} \mid u \in \text{ne}(v)\}, \mathbf{A})) = 0 \quad (6.5)$$

$$\overline{\text{READOUT}} - \text{READOUT}(\{\{\mathbf{h}_v^{(L)} : v \in V\}) = 0 \quad (6.6)$$

where \mathbf{A} is the variable storing the adjacency matrix of the input graph. We can assume that \mathbf{A} is valid for every graph in the domain if the domain is made by finite graphs.

The following lemma specifies the format of Pfaffian functions involved in the above equations.

Lemma 6.3. *Let COMBINE^(t), AGGREGATE^(t) and READOUT be Pfaffian functions with format, respectively, $(\alpha_{\text{comb}}, \beta_{\text{comb}}, \ell_{\text{comb}})$, $(\alpha_{\text{agg}}, \beta_{\text{agg}}, \ell_{\text{agg}})$, $(\alpha_{\text{read}}, \beta_{\text{read}}, \ell_{\text{read}})$ w.r.t. the variables \mathbf{y} and $\boldsymbol{\theta}$ described above, then:*

1. the left part of Eq. (6.4) is a polynomial of degree 1;
2. the left part of Eq. (6.5) is a Pfaffian function with format $(\alpha_{\text{agg}} + \beta_{\text{agg}} - 1 + \alpha_{\text{comb}}\beta_{\text{agg}}, \beta_{\text{comb}}, \ell_{\text{comb}} + \ell_{\text{agg}})$;

3. the left part of Eq. (6.6) is a Pfaffian function with format $(\alpha_{\text{read}}, \beta_{\text{read}}, \ell_{\text{read}})$;
4. Eqs. (6.4)–(6.6) constitute a system of Pfaffian equations with a maximal format $(\alpha_{\text{system}}, \beta_{\text{system}}, \ell_{\text{system}})$, where $\alpha_{\text{system}} = \max\{\alpha_{\text{agg}} + \beta_{\text{agg}} - 1 + \alpha_{\text{comb}}\beta_{\text{agg}}, \alpha_{\text{read}}\}$, $\beta_{\text{system}} = \max\{\beta_{\text{comb}}, \beta_{\text{read}}\}$ and $\ell_{\text{system}} = \text{LN}d(\ell_{\text{comb}} + \ell_{\text{agg}}) + \ell_{\text{read}}$.

Proof. The first point is straightforwardly evident, while the third is true by definition. The second point can be derived by applying the composition lemma for Pfaffian functions [89], according to which, if two functions f and g have format $(\alpha_f, \beta_f, \ell_f)$ and $(\alpha_g, \beta_g, \ell_g)$, respectively, then their composition $f \circ g$ has format $(\alpha_g + \beta_g - 1 + \alpha_f\beta_g, \beta_f, \ell_f + \ell_g)$. Finally, the fourth point is obtained by taking the maximum of the format of the involved Pfaffian equations also observing that the common chain is the concatenation of the chains. \square

We can now state the main result of this chapter.

Theorem 6.4. *Let us consider the GNN model described by Eq. (6.3). If COMBINE^(t), AGGREGATE^(t) and READOUT are Pfaffian functions with format $(\alpha_{\text{comb}}, \beta_{\text{comb}}, \ell_{\text{comb}})$, $(\alpha_{\text{agg}}, \beta_{\text{agg}}, \ell_{\text{agg}})$, $(\alpha_{\text{read}}, \beta_{\text{read}}, \ell_{\text{read}})$, respectively, then the VC dimension satisfies*

$$\text{VCdim}(\text{GNN}) \leq 2 \log B + \bar{p}(16 + 2 \log \bar{s}) \quad (6.7)$$

where $B \leq 2^{\frac{\bar{\ell}(\bar{\ell}-1)}{2}+1}(\bar{\alpha} + 2\bar{\beta} - 1)^{\bar{p}-1}((2\bar{p} - 1)(\bar{\alpha} + \bar{\beta}) - 2\bar{p} + 2)^{\bar{\ell}}$, $\bar{\alpha} = \max\{\alpha_{\text{agg}} + \beta_{\text{agg}} - 1 + \alpha_{\text{comb}}\beta_{\text{agg}}, \alpha_{\text{read}}\}$, $\bar{\beta} = \max\{\beta_{\text{comb}}, \beta_{\text{read}}\}$, $\bar{p} = p_{\text{comb}}^{(0)} + p_{\text{agg}}^{(0)} + (L - 1)(p_{\text{comb}} + p_{\text{agg}}) + p_{\text{read}}$, $\bar{\ell} = \bar{p}H$, $H = \text{LN}d(\ell_{\text{comb}} + \ell_{\text{agg}}) + \ell_{\text{read}}$ and $\bar{s} = \text{LN}d + Nq + 1$ hold. By substituting the definitions in Eq. (6.7), we obtain

$$\begin{aligned} \text{VCdim}(\text{GNN}) &\leq \bar{p}^2(\text{LN}d(\ell_{\text{comb}} + \ell_{\text{agg}}) + \ell_{\text{read}})^2 \\ &\quad + 2\bar{p} \log(3\gamma) \\ &\quad + 2\bar{p} \log((4\gamma - 2)\bar{p} + 2 - 2\gamma) \\ &\quad + \bar{p}(16 + 2 \log(\text{LN}d + Nq + 1)) \end{aligned} \quad (6.8)$$

where $\bar{\alpha}, \bar{\beta} \leq \gamma$ for a constant $\gamma \in \mathbb{R}$.

Proof. Let \mathbf{T} be defined as in Eq. (6.2), where $\tau_i(\mathbf{y}, \boldsymbol{\theta}) = 0$ are the equations in (6.4), (6.5), (6.6). Combining Theorem 6.2 with the formats provided by point 3. of Lemma 6.3 for any input graph and any value of the variables \mathbf{y} , the number of connected components of \mathbf{T}^{-1} satisfies

$$B \leq 2^{\frac{\bar{\ell}(\bar{\ell}-1)}{2}+1}(\bar{\alpha} + 2\bar{\beta} - 1)^{\bar{p}-1}((2\bar{p} - 1)(\bar{\alpha} + \bar{\beta}) - 2\bar{p} + 2)^{\bar{\ell}}, \quad (6.9)$$

where $\bar{p} = p_{\text{comb}}^{(1)} + p_{\text{agg}}^{(1)} + (L - 1)(p_{\text{comb}} + p_{\text{agg}}) + p_{\text{read}}$, $\bar{\alpha} = \alpha_{\text{system}}$, $\bar{\beta} = \beta_{\text{system}}$, $\bar{\ell} = \bar{p}(\text{LN}d(\ell_{\text{comb}} + \ell_{\text{agg}}) + \ell_{\text{read}})$.

By Theorem 6.1, the VC dimension of the GNN described by Eqs. (6.4)–(6.6) is bounded by

$$\text{VCdim}(\text{GNN}) \leq 2 \log B + \bar{p}(16 + 2 \log \bar{s}), \quad (6.10)$$

where $\bar{s} = LNd + Nq + 1$. Thus, substituting Eq. (6.9) in Eq. (6.10), we have:

$$\begin{aligned} \text{VCdim}(\text{GNN}) &\leq 2 \log B + \bar{p}(16 + 2 \log \bar{s}) \\ &\leq 2 \log \left(2^{\frac{\bar{\ell}(\bar{\ell}-1)}{2}+1} (\bar{\alpha} + 2\bar{\beta} - 1)^{\bar{p}-1} ((2\bar{p} - 1)(\bar{\alpha} + \bar{\beta}) - 2\bar{p} + 2)^{\bar{\ell}} \right) + \\ &\quad + \bar{p}(16 + 2 \log \bar{s}) \\ &= \bar{\ell}(\bar{\ell}-1) + 2(\bar{p}-1) \log (\bar{\alpha} + 2\bar{\beta} - 1) + 2\bar{\ell} \log ((2\bar{p} - 1)(\bar{\alpha} + \bar{\beta}) - 2\bar{p} + 2) + \\ &\quad + \bar{p}(16 + 2 \log \bar{s}) + 2, \end{aligned}$$

obtaining Eq. (6.7).

If we denote $H = LNd((\ell_{\text{comb}} + \ell_{\text{agg}}) + \ell_{\text{read}})$, we have:

$$\begin{aligned} \text{VCdim}(\text{GNN}) &\leq \bar{p}H(\bar{p}H - 1) + 2(\bar{p} - 1) \log(\alpha_{\text{system}} + 2\beta_{\text{system}} - 1) \\ &\quad + 2\bar{p}H \log((2\bar{p} - 1)(\alpha_{\text{system}} + \beta_{\text{system}}) - 2\bar{p} + 2) \\ &\quad + \bar{p}(16 + 2 \log(\bar{s})) + 2 \\ &\leq \bar{p}^2 H^2 + 2\bar{p} \log(3\gamma) \\ &\quad + 2\bar{p}H \log((4\gamma - 2)\bar{p} + 2 - 2\gamma) \\ &\quad + \bar{p}(16 + 2 \log(\bar{s})) + 2. \end{aligned} \tag{6.11}$$

Then, by replacing \bar{p} , H and \bar{s} , and setting $\gamma = \max\{\bar{\alpha}, \bar{\beta}\}$, it follows that:

$$\begin{aligned} \text{VCdim}(\text{GNN}) &\leq (p_{\text{comb}^{(1)}} + p_{\text{agg}^{(1)}} + (L-1)(p_{\text{comb}} + p_{\text{agg}}) + p_{\text{read}})^2 (LNd(\ell_{\text{comb}} + \ell_{\text{agg}}) + \ell_{\text{read}})^2 \\ &\quad + 2(p_{\text{comb}^{(1)}} + p_{\text{agg}^{(1)}} + (L-1)(p_{\text{comb}} + p_{\text{agg}}) + p_{\text{read}}) \log(3\gamma) \\ &\quad + 2(p_{\text{comb}^{(1)}} + p_{\text{agg}^{(1)}} + (L-1)(p_{\text{comb}} + p_{\text{agg}}) + p_{\text{read}}) \cdot \\ &\quad \cdot \log((4\gamma - 2)(p_{\text{comb}^{(1)}} + p_{\text{agg}^{(1)}} + (L-1)(p_{\text{comb}} + p_{\text{agg}}) + p_{\text{read}}) + 2 - 2\gamma) \\ &\quad + (p_{\text{comb}^{(1)}} + p_{\text{agg}^{(1)}} + (L-1)(p_{\text{comb}} + p_{\text{agg}}) + p_{\text{read}})(16 + 2 \log(LNd + Nq + 1)), \end{aligned}$$

which leads to Eq. (6.8) as in the thesis. \square

We observe that the dominant term is $\bar{p}^2 H^2 = \bar{p}^2 (LNd(\ell_{\text{comb}} + \ell_{\text{agg}}) + \ell_{\text{read}})^2$. Thus, the theorem suggests that the VC dimension is $O(\bar{p}^2 L^2 N^2 d^2)$, w.r.t. the number of parameters \bar{p} of the GNN, the number of Layers L , the number N of graph nodes, and the feature dimension d . Note that those hyperparameters are related by constraints, which should be considered in order to understand how the VC dimension depends on the single hyperparameters. Thus, the VC dimension is at most $O(p^4)$ since, when the number of parameters p grows, also the number L of layers and/or the feature dimension d grow. Interestingly, such a result is similar to those already obtained for feedforward and recurrent neural networks with Pfaffian activation functions.

Table 6.1 compares our result with those available in the literature. Therefore, even if GNNs are more complex, the order of growth of the VC dimension, w.r.t. the parameters, is the same of those simple models. The following theorem clarifies how the VC dimension depends on each hyperparameter.

Theorem 6.5. *With respect to the Pfaffian functions $\text{COMBINE}^{(k)}$, $\text{AGGREGATE}^{(k)}$ and READOUT defined in Theorem 6.4, if $p_{\text{comb}}, p_{\text{agg}}, p_{\text{read}} \in \mathcal{O}(d)$, then the VC dimension of*

Activation function	Bound	References
Modern GNNs		
Piecewise polynomial	$O(p \log(\mathcal{C}p) + p \log(N))$	[55]
tanh, logsig or atan	$O(p^4 N^2)$	this work
tanh, logsig or atan	$O(p^4 \mathcal{C}^2)$	this work
Original GNNs [2]		
Polynomial	$O(p \log(N))$	[28]
Piecewise polynomial	$O(p^2 N \log(N))$	[28]
tanh, logsig or atan	$O(p^4 N^2)$	[28]
Positional RecNNs		
Polynomial	$O(pN)$	[90]
logsig	$O(p^4 N^2)$	[90]
Sequences (recurrent neural networks)		
Polynomial	$O(pN)$	[91]
Piecewise polynomial	$O(p^2 N)$	[91]
tanh or logsig	$O(p^4 N^2)$	[91]
Vectors (Static multilayer networks)		
Binary	$O(p \log p)$	[92, 93, 94]
Polynomial	$O(p \log p)$	[95]
Piecewise polynomial	$O(p^2)$	[95, 91]
tanh, logsig or atan	$O(p^4)$	[87]

Table 6.1: Upper bounds on VCdim of common architectures: p is the number of network parameters, N the number of nodes in the input graph or sequence, while \mathcal{C} is the maximum number of colors per graph.

a GNN defined as in Equation (2.1), w.r.t. \bar{p}, H, N, L, d, q satisfies

$$\text{VCdim}(\text{GNN}) \leq \mathcal{O}(\bar{p}^4)$$

$$\text{VCdim}(\text{GNN}) \leq \mathcal{O}(N^2)$$

$$\text{VCdim}(\text{GNN}) \leq \mathcal{O}(L^4)$$

$$\text{VCdim}(\text{GNN}) \leq \mathcal{O}(d^6)$$

$$\text{VCdim}(\text{GNN}) \leq \mathcal{O}(q^2)$$

Proof. Based on Theorem 6.4, we can derive the orders of growth of the VC dimension w.r.t. the dimension of the features, the number of layers and the graph dimension, obtaining Theorem 6.5. \square

The proof of Theorem 6.4 adopts the same reasoning used in [87], to derive a bound on the VC dimension of feedforward neural networks with Pfaffian activation functions, and in [28], to provide a bound for the first GNN model. Intuitively, the proof is based on the following steps: it is shown that the computation of GNNs on graphs can be represented by a set of equations defined by Pfaffian functions with format $(\bar{\alpha}, \bar{\beta}, \bar{\ell})$, where $\bar{\alpha}, \bar{\beta}, \bar{\ell}$

are those defined in the theorem; then, the bound is obtained exploiting a result in [87], that connects the VC dimension with the number of connected components in the inverse images of a system of Pfaffian equations, and a result in [88], that allows to estimate the required number of connected components. Note that our bound and other bounds obtained for networks with Pfaffian activation functions are larger than those for networks with simpler activations. As explained in [87] and [28], such a difference is likely to be due to the fact that tight bounds are more difficult to achieve with Pfaffian functions and also depends on the current general limits of mathematics in the field.

6.2.2 Computation of the main bounds for a specific GNN model

In this section, we derive the VC dimension bounds specifically for the architecture described by Eqs. (2.4)–(2.5).

Given an attributed graph domain \mathcal{G} , containing graphs up to N nodes and whose attributes have dimension q , let a GNN be described by the updating scheme defined by Equation (2.4). The hidden states are initialised as $\mathbf{h}_v^{(0)} = \boldsymbol{\alpha}_v$.

As in Section 6.2.1, the first step of the proof consists of defining a set of equations $\tau_i(\mathbf{y}, \boldsymbol{\theta}) = 0$ that specifies the computation of a GNN. Obviously, $\boldsymbol{\theta}$ includes just the GNN parameters, namely $\boldsymbol{\theta} = \Theta$. It can be easily calculated that the total number of parameters is $p = (2d + 1)(d(L - 1) + q + 1) - q$. Indeed, the parameters are:

- $\mathbf{W}_{\text{comb}}^{(1)}, \mathbf{W}_{\text{agg}}^{(1)} \in \mathbb{R}^{q \times d}, \mathbf{b}^{(1)} \in \mathbb{R}^{1 \times d}$, so that we have $2dq + d$ parameters;
- $\mathbf{W}_{\text{comb}}^{(k)}, \mathbf{W}_{\text{agg}}^{(k)} \in \mathbb{R}^{d \times d}, \mathbf{b}^{(k)} \in \mathbb{R}^{1 \times d}$ for $k = 2, \dots, L$, so that we have $(2d^2 + d)(L - 1)$ parameters;
- $\mathbf{w} \in \mathbb{R}^{d \times 1}, b \in \mathbb{R}$, so that we have $d + 1$ parameters (we consider $o = 1$ for ease of computation).

Summing up, we have $2dq + d + (2d^2 + d)(L - 1) + d + 1 = (2d + 1)(d(L - 1) + q + 1) - q$ parameters. Moreover, intuitively \mathbf{y} represents the input of a GNN, so that for a given graph $\mathbf{G} = (G, \mathbf{L})$ in \mathcal{G} each \mathbf{y} contains some vectorial representation of \mathbf{G} , namely the Nq graph attributes in \mathbf{L}_G and a vectorial representation of the adjacency matrix, which requires $N(N - 1)/2$ elements.

To define the equations, we use the same trick as in [87] and we introduce new variables for each computation unit of the network. Those variables belong to the input \mathbf{y} of τ . More precisely, we consider a vector of d variables $\mathbf{h}_v^{(k)}$ for each node v and for each layer t . Note that v implicitly addresses a specific node of some graph in the domain, since we may be interested in defining more computations of the GNN on more graphs at the same time. Finally, a variable READOUT for each graph contains just a single output of the GNN. In total, the dimension of \mathbf{y} is $Nq + N(N - 1)/2 + NdL + 1$.

Then, by Equation (2.4), the computation of the GNN is straightforwardly defined by

the following set of $LNd + Nq + 1$ equations,

$$\mathbf{h}_v^{(0)} - \boldsymbol{\alpha}_v = 0 \quad (6.12)$$

$$\mathbf{h}_v^{(k)} - \sigma\left(\mathbf{h}_v^{(k-1)}\mathbf{W}_{\text{comb}}^{(k)} + \sum_u \mathbf{h}_u^{(k-1)}\mathbf{W}_{\text{agg}}^{(k)}m_{v,u} + \mathbf{b}^{(k)}\right) = 0 \quad (6.13)$$

$$\text{READOUT} - \sigma\left(\sum_{v \in V} \mathbf{h}_v^{(L)}\mathbf{W} + b\right) = 0 \quad (6.14)$$

where $m_{v,u}$ is a binary value, which is 1 when v and u are connected and 0, otherwise.

The following lemma specifies the format of Pfaffian functions involved in the above equations.

Lemma 6.6. *Let σ be a Pfaffian function in \mathbf{x} with format $(\alpha_\sigma, \beta_\sigma, \ell_\sigma)$, then w.r.t. the variables \mathbf{y} and \mathbf{w} described above,*

1. the left part of Equation (6.12) is a polynomial of degree 1;
2. the left part of Equation (6.13) is a Pfaffian function having format $(2 + 3\alpha_\sigma, \beta_\sigma, \ell_\sigma)$;
3. the left part of Equation (6.14) is a Pfaffian function having format $(1 + 2\alpha_\sigma, \beta_\sigma, \ell_\sigma)$;
4. Equations (6.12)–(6.14) constitute a system of Pfaffian equations with format $(2 + 3\alpha_\sigma, \beta_\sigma, H\ell_\sigma)$, where the shared chain is obtained by concatenating the chains of $H = LNd + 1$ equations in (6.13), (6.14) that include an activation function.

Proof. The first point is straightforward. The second and third points can be derived by applying the composition lemma for Pfaffian functions. Actually, the formula inside σ in Equation (6.13) is a polynomial of degree 3, due to the factors $\mathbf{h}_u^{(k-1)}\mathbf{W}_{\text{agg}}^{(k)}m_{v,u}$, while formula inside σ in Equation (6.14) is a polynomial of degree 2, due to the factors $\mathbf{h}_v^{(L)}\mathbf{W}$. Moreover, polynomials are Pfaffian functions with null chains, with α equal to 0 and β equal to their degrees. Thus, the functions inside the σ in Equations (6.13) and (6.14) have format $(0, 3, 0)$ and $(0, 2, 0)$, respectively. Then, the thesis follows by the composition lemma in [89], according to which if two functions f and g have format $(\alpha_f, \beta_f, \ell_f)$ and $(\alpha_g, \beta_g, \ell_g)$, respectively, then their composition $f \circ g$ has format $(\alpha_g + \beta_g - 1 + \alpha_f\beta_g, \beta_f, \ell_f + \ell_g)$. Finally, the fourth point is a consequence of the fact that the equations are independent and the chains can be concatenated. The length of the chain is an obvious consequence of the fact that there are $H = LNd + 1$ equations using σ . The degree is obtained copying the largest degree of a Pfaffian function, which is the one in Equation (6.13). \square

Now we can state the main results on the VC dimension of the GNN model described by Eqs. (2.4), (2.5).

Theorem 6.7. *Let us consider the GNN model described by Eqs. (2.4), (2.5). If σ is a Pfaffian function in \mathbf{x} with format $(\alpha_\sigma, \beta_\sigma, \ell_\sigma)$, then the VC dimension satisfies*

$$\text{VCdim}(\text{GNN}) \leq 2\log B + \bar{p}(16 + 2\log \bar{s})$$

where $B \leq 2^{\frac{\bar{\ell}(\bar{\ell}-1)}{2}+1}(\bar{\alpha} + 2\bar{\beta} - 1)^{\bar{p}-1}((2\bar{p} - 1)(\bar{\alpha} + \bar{\beta}) - 2\bar{p} + 2)^{\bar{\ell}}$, $\bar{\alpha} = 2 + 3\alpha_\sigma$, $\bar{\beta} = \beta_\sigma$, $\bar{\ell} = \bar{p}H\ell_\sigma$, and $\bar{s} = LNd + Nq + 1$ holds.

If σ is the logistic sigmoid activation function, we have

$$\text{VCdim}(\text{GNN}) \leq \bar{p}^2 H^2 + 2\bar{p} \log(9) + 2\bar{p}H \log(16\bar{p}) + \bar{p}(16 + 2\log(\bar{s}))$$

Proof. As in [6.2.1](#), it is enough to combine Lemma [6.6](#), Theorem [6.2](#) and Theorem [6.1](#) to obtain the bounds stated in the thesis. The bounds on the VC dimension of the specific GNN with `logsig` as activation function is easily found since the format of `logsig` is (2,1,1):

$$\begin{aligned} \text{VCdim}(\text{GNN}) &\leq ((2d+1)(d(L-1)+q+1)-q)^2(LNd+1)^2 \\ &\quad + 2((2d+1)(d(L-1)+q+1)-q) \log(9) \\ &\quad + 2((2d+1)(d(L-1)+q+1)-q) \log(16((2d+1)(d(L-1)+q+1)-q)) \\ &\quad + ((2d+1)(d(L-1)+q+1)-q)(16+2\log(LNd+Nq+1)). \end{aligned}$$

□

The following theorem shows that the obtained bounds are perfectly coherent with the ones achieved in the general case.

Theorem 6.8. *The VC dimension of a GNN defined as in Equation [\(2.1\)](#), w.r.t. \bar{p}, H, N, L, d, q satisfies*

$$\begin{aligned} \text{VCdim}(\text{GNN}) &\leq \mathcal{O}(\bar{p}^2 H^2) \\ \text{VCdim}(\text{GNN}) &\leq \mathcal{O}(N^2) \\ \text{VCdim}(\text{GNN}) &\leq \mathcal{O}(L^4) \\ \text{VCdim}(\text{GNN}) &\leq \mathcal{O}(d^6) \\ \text{VCdim}(\text{GNN}) &\leq \mathcal{O}(q^2) \end{aligned}$$

Proof. Based on Theorem [6.7](#), we can derive the orders of growth of the VC dimension w.r.t. the dimension of the features, the number of the layers and the graph dimension, obtaining Theorem [6.8](#). □

6.2.3 Bounds with 1-WL colors

The developed theory is easily adapted to the case when nodes can be grouped according to their colors defined by the Weisfeiler–Lehman algorithm. Since a group of nodes with the same color are computationally equivalent, then the corresponding equations can be merged. Formally, for a given graph G , let $C_1(G) = \sum_{k=1}^L C^k(G)$ be the number of colors generated by 1-WL, where $C^k(G)$ is the number of colors at step $k > 0$. Moreover, let us assume that $C^k(G)$ is bounded in the domain, namely there is C_1 such that $C_1(G) \leq C_1$ for all the graphs G in the domain. Similarly, let $C_0(G)$ be the number of colors of the graph generated by 1-WL at the initial step and assume that there exists C_0 such that $C_0(G) \leq C_0$ for all the graphs of the domain. The following theorem provides a bound on the VC dimension w.r.t. the numbers of colors.

Theorem 6.9. *Assume a subset $\mathcal{S} \subseteq \mathcal{G}$ and consider a GNN using the logistic sigmoid `logsig` as the activation function. The VC dimension of the GNN satisfies*

$$\begin{aligned} \text{VCdim}(\text{GNN}(C_1)) &\leq \mathcal{O}(C_1^2) \\ \text{VCdim}(\text{GNN}(C_0)) &\leq \mathcal{O}(\log(C_0)). \end{aligned}$$

Proof. Let us call Basic GNN (BGNN) the model of Eqs. (2.4)–(2.5). The proof is based on the introduction of an extended version, which we call EGNN, that can simulate the BGNN. Due to this capability, the EGNN can shatter any set that is shattered by the BGNN so that its VC dimension is greater or equal to the VC dimension of the BGNN. The proof will follow by bounding the VC dimension of the former model. More precisely, the EGNN exploits the same aggregation mechanism of the BGNN to compute the features, which is described by Eq. (2.4). On the other hand, the READOUT function is defined as

$$\text{READOUT}\left(\{\mathbf{h}_v^{(L)} \mid v \in V\}\right) := f\left(\sum_{v \in V} \mathbf{w} \mathbf{h}_v^{(L)} c_v + b\right), \quad (6.15)$$

where c_v are additional real inputs used to weight each node feature in the READOUT function. The simulation is based on the following steps.

- 1) Each input graph G of the BGNN is transformed to another graph G' , where all the nodes having the same 1-WL color are merged into a single node and the edges are merged consequently;
- 2) The EGNN is applied to G' and each c_v is set equal to the number of nodes that have been merged to obtain node v .

Note that a GNN cannot distinguish nodes with the same color as the computation is the same on all these nodes. Thus, the BGNN and the EGNN produce the same features on nodes sharing color. As a consequence, also the READOUTs of the two models have the same output, when the c_v are equal to the number of nodes within each color cluster.

Given these assumptions, the number of equations describing the Pfaffian variety associated to the EGNN is reduced to $s_c = C_1 d + C_0 q + 1$, which can be used in place of \bar{s} in Theorem 6.1. Moreover, also the chains of the Pfaffian functions in merged equations can be merged and we have that H can be replaced by $H_c = C_1 d + 1$. Finally, the length of the chain ℓ of Theorem 6.2 is replaced by $\bar{\ell}_c = \bar{p} H_c \ell_\sigma$.

With the above changes, we can replace the variables in Eq. (6.11) as in 6.2.1, obtaining:

$$\begin{aligned} \text{VCdim}(\text{GNN}) &\leq \bar{p} H_c (\bar{p} H_c - 1) + 2\bar{p} \log(9) \\ &\quad + 2\bar{p} H_c \log(16\bar{p} - 7) \\ &\quad + \bar{p}(16 + 2 \log(\bar{s}_c)) + 2 \\ &\leq \bar{p}^2 (C_1 d + 1)^2 + 2\bar{p} \log(9) \\ &\quad + 2\bar{p}(C_1 d + 1) \log(16\bar{p} - 7) \\ &\quad + \bar{p}(16 + 2 \log(C_1 d + C_0 q + 1)). \end{aligned}$$

and the thesis holds. \square

The obtained result suggests that the VC dimension depends quadratically on the total number of node colors and logarithmically on the initial number of colors. Actually, a GNN processes all the nodes of a graph at the same time and the GNN architecture is similar to a feedforward network where some computation units are replicated at each

node. Thus, the complexity of the GNN grows with the number of nodes and this explains the dependence of the VC dimension on the number of nodes (see Theorem 6.8). On the other hand, nodes with the same colors cannot be distinguished by the GNN: this means that, in theory, we can use the same computation units for a group of nodes sharing the color. Therefore, to get tighter bounds on the VC dimension, we can consider the number of colors in place of the number of nodes. Finally, it is worth mentioning that if the number of nodes/colors for each graph increases, the VC dimension of the GNN also increases and, as a consequence, decreases its generalization capability, which also depends on the number of patterns in the training set (see Eqs. 6.1). In GNNs and graph focused tasks, each graph of the training set is a pattern, but graphs with the same set of colors have to be considered duplicates; in node focused tasks, the nodes are patterns and the nodes with the same color are duplicates. Thus, the generalization performance of a GNN improves when the total number of graph/node colors in the training set increases, in both graph and node focused tasks.

6.3 Experimental validation

In this section, we present an experimental validation of our theoretical results. We will show how the VC dimension of GNNs evolves along with the variation of the parameters respecting the bounds found in Theorems 6.8 and 6.9. In our experimental framework we will exploit the GNN convolution described in Eqs. 2.4 - 2.5.

6.3.1 Experimental setting

We design two experimental frameworks to assess the validity, respectively, of Theorems 6.8 and 6.9. For both frameworks, we train a Graph Neural Network made by message passing layers defined as in Equation 2.4, where the activation function f is either `arctan` or `tanh`; the final READOUT layer is an affine layer $\mathbf{W}_{\text{out}} \in \mathbb{R}^{d \times o}$, after which a `logsig` activation function is applied; here o denotes the dimension of the output. The model is trained via Adam optimizer with an initial learning rate $\lambda = 10^{-3}$. The hidden feature size is denoted by d and the number of layers by L .

- E1:** We measure the evolution of the difference between training and validation accuracy $\text{diff} = \text{train_acc} - \text{test_acc}$ through the training epochs, over the NCI1 dataset taken from the TUDataset repository [96]. The choice of the datasets has been driven by their binary classification nature. On the first part of the experiment, we fix the hidden feature size to $d = 32$ and let the number of layers vary in the range $L \in [2, 3, 4, 5, 6]$, to measure how diff evolves. On the second part, we fix the number of layers to $L = 3$ and let the hidden feature size d vary in $[8, 16, 32, 64, 128]$, to perform the same task. We train the model for 500 epochs in each run, with the batch size fixed at 32.
- E2:** We measure the evolution of the difference between training and validation accuracy $\text{diff} = \text{train_acc} - \text{test_acc}$ through the training epochs over the dataset NCI1, whose graphs are increasingly ordered according to the ratio $\frac{|V(G)|}{C^k(G)}$, splitted in 4 different

groups. The intuition here is that, being the number of nodes of the graphs bounded, splitting the ordered dataset should provide datasets in which the total number of colors is progressively increasing. Hidden size is fixed as $d = 16$, the number of

	Split 1	Split 2	Split 3	Split 4
# Nodes	27667	30591	31763	32673
# Colors	26243	26569	24489	16348
$\min_G \frac{\#Nodes(G)}{\#Colors(G)}$	1.000	1.105	1.208	1.437
$\max_G \frac{\#Nodes(G)}{\#Colors(G)}$	1.105	1.208	1.437	8

Figure 6.1: Summary of the parameters for each split of the ordered NCI1 dataset for the task **E2**.

layers is fixed as $L = 4$, the batch size is fixed equal to 32. We report in Table 6.1 the reference values for each split. We train the model for 2000 epochs (the number of epochs is greater with respect to task **E1** because this task presented greater instability during the training optimization)

Each experiment is statistically evaluated over 10 runs. The overall training is performed on an Intel(R) Core(TM) i7-9800X processor running at 3.80GHz, using 31GB of RAM and a GeForce GTX 1080 Ti GPU unit. The code developed to run the experiments exploits the Python package `PytorchGeometric`; the code can be found at <https://github.com/AleDinve/vc-dim-gnn>.

6.3.2 Experimental results

Task E1 Numerical results for the NCI1 dataset are reported in Figures 6.2-6.3. The behaviour of the evolution of diff proves to be consistent with the bounds provided by Theorem 6.8 with respect to increasing the hidden dimension or the number of layers. Although it is hard to establish a precise function that links the VC dimension to diff , given also the complex nature of the theoretical framework as the one of the Pfaffian functions, we can partially rely on Equation (6.1) (which is valid for large sample sets) to argue that our bounds are verified by this experimental setting.

Task E2 Here, numerical results for the NCI1 dataset are reported in Figure 6.4. Similar observations as for the experimental setting **E1** can be drawn here: indeed, the evolution of diff in our experiment is consistent with the bounds presented in Theorem 6.9 as the ratio between colors and nodes increases.

In the next chapter, we will examine the generalization capabilities of GNNs through the lens of cognitive tasks. In particular, we will assess the ability of GNNs to learn the so-called *identity effects*.

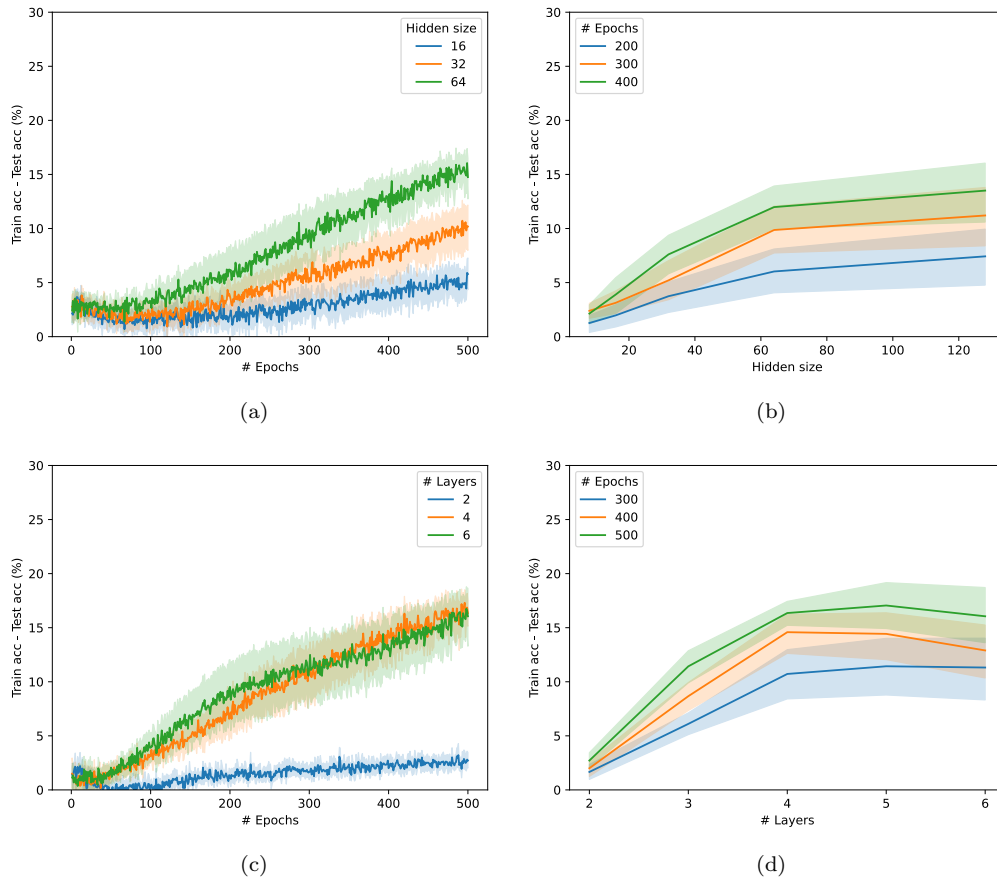


Figure 6.2: Results on the task **E1** for GNNs with activation function \arctan . Pictures (a) and (c) show the evolution of diff through the epochs, for different values of d , keeping fixed $L = 3$, and for different values of L , keeping fixed $d = 32$; Picture (b) shows how diff evolves as the hidden size increases, while Picture (d) shows how diff evolves as the number of layers increases.

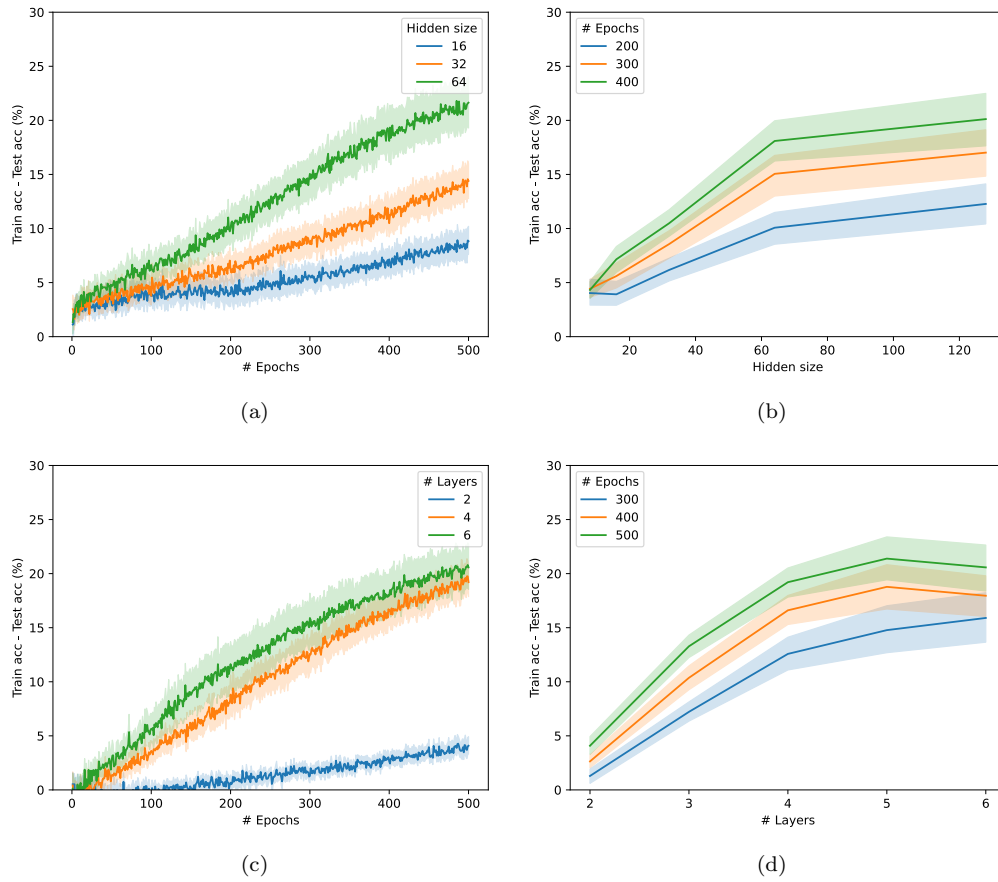


Figure 6.3: Results on the task **E1** for GNNs with activation function `tan``h`. Pictures (a) and (c) show the evolution of `diff` through the epochs, for different values of d , keeping fixed $L = 3$, and for different values of L , keeping fixed $d = 32$; Picture (b) shows how `diff` evolves as the hidden size increases, while Picture (d) shows how `diff` evolves as the number of layers increases.

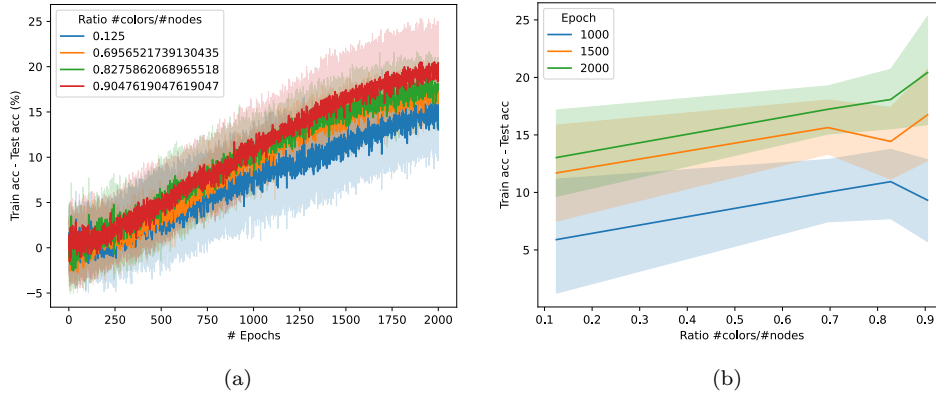


Figure 6.4: Results on the task **E2** for GNNs with activation function \tanh . Picture (a) shows the evolution of diff through the epochs, for different values of $\frac{V(G)}{C^k(G)}$, keeping fixed $L = 4$ and $d = 16$; Picture (b) shows how diff evolves as the ratio $\frac{V(G)}{C^k(G)}$ increases.



Chapter 7

Learning Identity Effects with GNNs

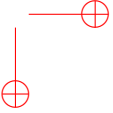
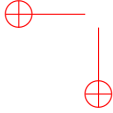
This chapter examines the generalization limits and capabilities of GNNs when learning identity effects, a task that determines whether an object is made up of two identical components or not. More precisely, we consider two tasks. In the former, we deal with graphs constituted by two nodes connected by an edge: the goal is to distinguish whether the nodes have the same attributes. In the latter, the graphs are made by two distinct cycles of nodes connected by an edge: the objective is to distinguish whether the two cycles have the same length or not. Those two tasks intuitively model two basic cases of learning identity effects with graphs. The former is about the recognition of identities related to node attributes, whereas the latter is about the recognition of identities about graph structure. We will see that a GNN can solve the latter task, but not the former.

The chapter is structured as follows. In Section [7.1](#) we review theory of impossibility theorems for invariant learners, namely some literature results that will be used to derive the main results. In Section [7.2](#) we present and prove our main theoretical results. Section [7.3](#) shows numerical experiments conducted to validate our findings.

7.1 Rating impossibility for invariant learners

Let us recall the theory of rating impossibility [\[67\]](#), which will be applied to the case of identity effect learning. In general, we assume to train a machine learning model (ML model) to perform a rating assignment task, which consists of giving a rating r to a pattern \mathbf{x} . Let \mathcal{I} be the set of all possible inputs \mathbf{x} . Our ML model is trained on a dataset $D \subseteq \mathcal{I} \times \mathbb{R}$ consisting of a finite set of input-rating pairs (\mathbf{x}, r) . The model is trained via a suitable optimization method Θ , such as Stochastic Gradient Descent (SGD) or Adaptive Moment Estimation (Adam) [\[97\]](#), which, for any given training dataset D , outputs the optimized set of parameters $\Theta = \Theta(D) \in \mathbb{R}^p$ of a model $\varphi = \varphi(\Theta, \cdot)$. The rating prediction on a novel input $\mathbf{x} \in \mathcal{I}$ is then given by $r = \varphi(\Theta, \mathbf{x})$. Formally, after training, the ML model implements a map $\mathcal{L} : \mathcal{D} \times \mathcal{I} \rightarrow \mathbb{R}$ such that $\mathcal{L}(D, \mathbf{x}) = \varphi(\Theta(D), \mathbf{x})$ holds.

Given the stochastic nature of neural network training, we adopt a nondeterministic point of view. Hence we require the notion of *equality in distribution*. Two random variables X, Y taking values in \mathbb{R}^q are said to be *equal in distribution* (denoted by $X \stackrel{d}{=} Y$) if $\mathbb{P}(X \leq \mathbf{x}) = \mathbb{P}(Y \leq \mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^q$, where inequalities hold componentwise. Rating impossibility happens when $\mathcal{L}(D, \mathbf{x}_1) \stackrel{d}{=} \mathcal{L}(D, \mathbf{x}_2)$ for two inputs $\mathbf{x}_1 \neq \mathbf{x}_2$ drawn from $\mathcal{I} \setminus D$. Sufficient conditions for rating impossibility are identified by the following theorem from [\[67\]](#), which involves the existence of an auxiliary transformation τ of the inputs.



Theorem 7.1 (Rating impossibility for invariant learners, [67, Thm 1]). *Consider a dataset $D \subseteq \mathcal{I} \times \mathbb{R}$ and a transformation $\tau : \mathcal{I} \rightarrow \mathcal{I}$ such that*

$$(i) \quad \tau(D) \stackrel{d}{=} D \text{ (invariance of the data)} \quad \square^1$$

Then, for any learning algorithm $\mathcal{L} : \mathcal{D} \times \mathcal{I} \rightarrow \mathbb{R}$ and any input $\mathbf{x} \in \mathcal{I}$ such that

$$(ii) \quad \mathcal{L}(\tau(D), \tau(\mathbf{x})) \stackrel{d}{=} \mathcal{L}(D, \mathbf{x}) \text{ (invariance of the algorithm),}$$

we have $\mathcal{L}(D, \tau(\mathbf{x})) \stackrel{d}{=} \mathcal{L}(D, \mathbf{x})$.

This theorem states that under the invariance of the data and of the algorithm, the learner cannot assign different ratings to an input \mathbf{x} and its transformed version $\tau(\mathbf{x})$. This leads to rating impossibility when $\tau(\mathbf{x}) \neq \mathbf{x}$ and $\mathbf{x}, \tau(\mathbf{x}) \in \mathcal{I} \setminus D$.

We conclude by recalling some basic notions on SGD training. Given a dataset D , we aim to find parameters Θ that minimize an objective function of the form

$$F(\Theta) = \mathbb{L}(\{\varphi(\Theta, \mathbf{x}), r\} : (\mathbf{x}, r) \in D), \quad \Theta \in \mathbb{R}^p,$$

where \mathbb{L} is a (possibly regularized) loss function. We assume F to be differentiable over \mathbb{R}^p in order for its gradients to be well defined. Given a collection of subsets $(D_i)_{i=0}^{k-1}$ with $D_i \subseteq D$ (usually referred to as training batches, which can be either deterministically or randomly generated), we define F_{D_i} as the function F where the loss is evaluated only on data in D_i . In SGD-based training, we randomly initialize Θ_0 and iteratively compute

$$\Theta_{i+1} = \Theta_i - \eta_i \frac{\partial F_{D_i}}{\partial \Theta}(\Theta_i), \quad (7.1)$$

for $i = 0, 1, \dots, k-1$, where the sequence of step sizes $(\eta_i)_{i=0}^{k-1}$ is assumed to be either deterministic or random and independent of $(D_i)_{i=0}^{k-1}$. Note that, being Θ_i a random vector for each i , the output of the learning algorithm $\mathcal{L}(D, \mathbf{x}) = \varphi(\Theta_i, \mathbf{x})$ is a random variable.

7.2 Theoretical analysis

In this Section, we present our results. More specifically, in Section 7.2.1 we focus on the study of learning identity effects for a two-letter word dataset. In this case, the dataset is constituted of graphs with two nodes, which are labeled with an alphabet letter and linked by an edge. The goal is to recognize whether the two letters of the nodes are equal or not. We establish a rating impossibility theorem for GNNs on such a dataset under certain technical assumptions related to the invariance of the node attributes. In Section 7.2.2, we study the GNN capability in recognizing graph connectivity and we consider dicyclic graphs, which are constituted by two cycles connected by an edge. It is proved that symmetric dicyclic graphs can be distinguished from asymmetric ones by the 1-WL test, and consequently by GNNs.

¹Here, by abuse of notation, $\tau(D) := \{\tau(\mathbf{x}), r\} : (\mathbf{x}, r) \in D$ holds.

7.2.1 What GNNs cannot learn: rating impossibility theorem

Let us assume that the input space is in the form $\mathcal{I} = \mathbb{R}^q \times \mathbb{R}^q$ and the learning algorithm

$$\mathcal{L}(D, \mathbf{x}) = f(\mathbf{B}, \mathbf{G}\mathbf{u} + \mathbf{H}\mathbf{v}, \mathbf{H}\mathbf{u} + \mathbf{G}\mathbf{v}), \quad \forall \mathbf{x} = (\mathbf{u}, \mathbf{v}) \in \mathcal{I}, \quad (7.2)$$

where $\Theta = (\mathbf{B}, \mathbf{G}, \mathbf{H})$ are the trained parameters defined as

$$\begin{aligned} \mathbf{G} &= \mathbf{W}_{\text{comb}}^{(1)}, & \mathbf{H} &= \mathbf{W}_{\text{agg}}^{(1)}, \\ \mathbf{B} &= \left(\mathbf{b}^{(1)}, \mathbf{W}_{\text{comb}}^{(2)}, \mathbf{W}_{\text{agg}}^{(2)}, \mathbf{b}^{(2)}, \dots, \mathbf{W}_{\text{comb}}^{(N)}, \mathbf{W}_{\text{agg}}^{(N)}, \mathbf{b}^{(N)} \right). \end{aligned}$$

and the \mathbf{W}, \mathbf{b} are those defined in (2.4). Here, f denotes a parametric ML model in the parameters $\Theta = (\mathbf{B}, \mathbf{G}, \mathbf{H})$ and the input $\mathbf{x} = (\mathbf{u}, \mathbf{v})$. With respect to the function φ above defined, we have that $\varphi((\mathbf{B}, \mathbf{G}, \mathbf{H}), (\mathbf{u}, \mathbf{v})) = f(\mathbf{B}, \mathbf{G}\mathbf{u} + \mathbf{H}\mathbf{v}, \mathbf{H}\mathbf{u} + \mathbf{G}\mathbf{v})$. The learner defined by equation (7.2) mimics, in this specific setting, the behaviour of several GNN architectures, GCN included, over graphs composed by two nodes (see Figure 7.1). In fact, when the graph is composed by only two nodes, the convolution ends up being a weighted sum of the hidden states of the two nodes, i.e., $\mathbf{h}_{\text{ne}}^{(k)} = \mathbf{h}_u^{(k)}$ and

$$\mathbf{h}_v^{(k+1)} = \sigma(\mathbf{W}_{\text{comb}}^{(k+1)} \mathbf{h}_v^{(k)} + \mathbf{W}_{\text{agg}}^{(k+1)} \mathbf{h}_u^{(k)} + \mathbf{b}^{(k+1)}).$$

This property will have practical relevance in Theorem 7.4 and its experimental realization in Section 7.3.2.

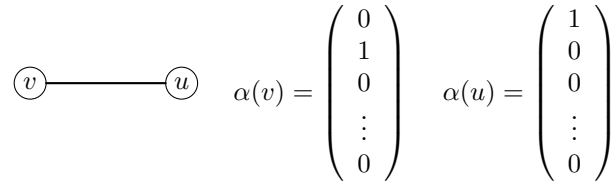


Figure 7.1: Graph modeling of a two-nodes graph: a vertex feature $\alpha(v) \in \mathbb{R}^q$ is attached to each node v of a two-node undirected graph, according to a given encoding \mathcal{E} . In this figure, \mathcal{E} is the one-hot encoding.

In the following result we identify sufficient conditions on the dataset D and the training procedure able to guarantee invariance of GNN-type models of the form (7.2) trained via SGD to a suitable class of transformations τ .

Theorem 7.2 (Invariance of GNN-type models trained via SGD). *Assume the input space is in the form of $\mathcal{I} = \mathbb{R}^q \times \mathbb{R}^q$. Let $\tau : \mathcal{I} \rightarrow \mathcal{I}$ be a linear transformation defined by $\tau(\mathbf{x}) = (\mathbf{u}, \tau_2(\mathbf{v}))$ for any $\mathbf{x} = (\mathbf{u}, \mathbf{v}) \in \mathcal{I}$, where $\tau_2 : \mathbb{R}^q \rightarrow \mathbb{R}^q$ is linear. Moreover, assume that*

- the matrix $\mathbf{T}_2 \in \mathbb{R}^{q \times q}$ associated with the transformation τ_2 is orthogonal and symmetric;
- the dataset $D = \{((\mathbf{u}_i, \mathbf{v}_i), r_i)\}_{i=1}^n$ is invariant under the transformation $\tau_2 \otimes \tau_2$, i.e.,

$$(\mathbf{u}_i, \mathbf{v}_i) = (\tau_2(\mathbf{u}_i), \tau_2(\mathbf{v}_i)), \quad \forall i = 1, \dots, n. \quad (7.3)$$

Suppose k iterations of SGD as defined in (7.1) are used to determine parameters $\Theta_k = (\mathbf{B}_k, \mathbf{G}_k, \mathbf{H}_k)$ with objective function

$$F(\Theta) = \sum_{i=1}^n \ell(f(\mathbf{B}, \mathbf{G}\mathbf{u}_i + \mathbf{H}\mathbf{v}_i, \mathbf{H}\mathbf{u}_i + \mathbf{G}\mathbf{v}_i), r_i) + \lambda \mathcal{R}(\mathbf{B}),$$

where $\Theta = (\mathbf{B}, \mathbf{G}, \mathbf{H})$ are the parameters, $\lambda \geq 0$ is a real meta-parameter, ℓ , f and \mathcal{R} are real-valued functions, representing the loss, the model and a regularization term, respectively, and defined so that F is differentiable. Suppose the random initialization of the parameters \mathbf{B} , \mathbf{G} and \mathbf{H} to be independent and that the distributions of \mathbf{G}_0 and \mathbf{H}_0 are invariant with respect to right-multiplication by \mathbf{T}_2 . Then, the learner L defined by $\mathcal{L}(D, \mathbf{x}) = f(\mathbf{B}_k, \mathbf{G}_k \mathbf{u} + \mathbf{H}_k \mathbf{v}, \mathbf{H}_k \mathbf{u} + \mathbf{G}_k \mathbf{v})$, for $\mathbf{x} = (\mathbf{u}, \mathbf{v})$, satisfies $\mathcal{L}(D, \mathbf{x}) \stackrel{d}{=} \mathcal{L}(\tau(D), \tau(\mathbf{x}))$.

Proof. Given a batch $D_i \subseteq D$, define $J_i := \{j \in \{1, \dots, n\} : ((\mathbf{u}_j, \mathbf{v}_j), r_j) \in D_i\}$ and

$$F_{D_i}(\Theta) = \sum_{j \in J_i} \ell(f(\mathbf{B}, \mathbf{G}\mathbf{v}_j + \mathbf{H}\mathbf{u}_j, \mathbf{H}\mathbf{v}_j + \mathbf{G}\mathbf{u}_j), r_j) + \lambda \mathcal{R}(\mathbf{B}).$$

Moreover, consider an auxiliary objective function, defined by

$$\begin{aligned} \tilde{F}_{D_i}(\mathbf{B}, \mathbf{G}_1, \mathbf{H}_1, \mathbf{H}_2, \mathbf{G}_2) = \\ \sum_{j \in J_i} \ell(f(\mathbf{B}, \mathbf{G}_1 \mathbf{v}_j + \mathbf{H}_1 \mathbf{u}_j, \mathbf{H}_2 \mathbf{v}_j + \mathbf{G}_2 \mathbf{u}_j), r_j) + \lambda \mathcal{R}(\mathbf{B}). \end{aligned}$$

Observe that $F_{D_i}(\Theta) = \tilde{F}_{D_i}(\mathbf{B}, \mathbf{G}, \mathbf{H}, \mathbf{H}, \mathbf{G})$. Moreover,

$$\frac{\partial F_{D_i}}{\partial \mathbf{B}}(\Theta) = \frac{\partial \tilde{F}_{D_i}}{\partial \mathbf{B}}(\Theta) \quad (7.4)$$

$$\frac{\partial F_{D_i}}{\partial \mathbf{G}}(\Theta) = \frac{\partial \tilde{F}_{D_i}}{\partial \mathbf{G}_1}(\Theta) + \frac{\partial \tilde{F}_{D_i}}{\partial \mathbf{G}_2}(\Theta) \quad (7.5)$$

$$\frac{\partial F_{D_i}}{\partial \mathbf{H}}(\Theta) = \frac{\partial \tilde{F}_{D_i}}{\partial \mathbf{H}_1}(\Theta) + \frac{\partial \tilde{F}_{D_i}}{\partial \mathbf{H}_2}(\Theta) \quad (7.6)$$

Moreover, replacing D_i with its transformed version $\tau(D_i) = \{((\mathbf{u}_j, \tau(\mathbf{v}_j)), r_j)\}_{j \in D_i}$, we see that $F_{\tau(D_i)}(\Theta) = \tilde{F}_{D_i}(\mathbf{B}, \mathbf{G}, \mathbf{H}\mathbf{T}_2, \mathbf{H}, \mathbf{G}\mathbf{T}_2)$. This leads to

$$\frac{\partial F_{\tau(D_i)}}{\partial \mathbf{B}}(\Theta) = \frac{\partial \tilde{F}_{D_i}}{\partial \mathbf{B}}(\mathbf{B}, \mathbf{G}, \mathbf{H}\mathbf{T}_2, \mathbf{H}, \mathbf{G}\mathbf{T}_2) \quad (7.7)$$

$$\begin{aligned} \frac{\partial F_{\tau(D_i)}}{\partial \mathbf{G}}(\Theta) = \frac{\partial \tilde{F}_{D_i}}{\partial \mathbf{G}_1}(\mathbf{B}, \mathbf{G}, \mathbf{H}\mathbf{T}_2, \mathbf{H}, \mathbf{G}\mathbf{T}_2) \\ + \frac{\partial \tilde{F}_{D_i}}{\partial \mathbf{G}_2}(\mathbf{B}, \mathbf{G}, \mathbf{H}\mathbf{T}_2, \mathbf{H}, \mathbf{G}\mathbf{T}_2) \mathbf{T}_2^T \end{aligned} \quad (7.8)$$

$$\begin{aligned} \frac{\partial F_{\tau(D_i)}}{\partial \mathbf{H}}(\Theta) = \frac{\partial \tilde{F}_{D_i}}{\partial \mathbf{H}_1}(\mathbf{B}, \mathbf{G}, \mathbf{H}\mathbf{T}_2, \mathbf{H}, \mathbf{G}\mathbf{T}_2) \mathbf{T}_2^T \\ + \frac{\partial \tilde{F}_{D_i}}{\partial \mathbf{H}_2}(\mathbf{B}, \mathbf{G}, \mathbf{H}\mathbf{T}_2, \mathbf{H}, \mathbf{G}\mathbf{T}_2). \end{aligned} \quad (7.9)$$

Now, denoting $\ell = \ell(f, r)$ and $f = f(\mathbf{B}, \mathbf{u}, \mathbf{v})$, we have

$$\begin{aligned} \frac{\partial \tilde{F}_{D_i}}{\partial \mathbf{G}_1} &= \sum_{j \in D_i} \frac{\partial \ell}{\partial f} \frac{\partial f}{\partial \mathbf{u}} \mathbf{v}_j^T, & \frac{\partial \tilde{F}_{D_i}}{\partial \mathbf{H}_1} &= \sum_{j \in D_i} \frac{\partial \ell}{\partial f} \frac{\partial f}{\partial \mathbf{u}} \mathbf{u}_j^T, \\ \frac{\partial \tilde{F}_{D_i}}{\partial \mathbf{H}_2} &= \sum_{j \in D_i} \frac{\partial \ell}{\partial f} \frac{\partial f}{\partial \mathbf{v}} \mathbf{v}_j^T, & \frac{\partial \tilde{F}_{D_i}}{\partial \mathbf{G}_2} &= \sum_{j \in D_i} \frac{\partial \ell}{\partial f} \frac{\partial f}{\partial \mathbf{v}} \mathbf{u}_j^T. \end{aligned}$$

In addition, thanks to assumption (7.3), we have $\mathbf{u}_j^T \mathbf{T}_2^T = \mathbf{u}_j^T$ and $\mathbf{v}_j^T \mathbf{T}_2^T = \mathbf{v}_j^T$ for all $j \in J_i$. Thus, we obtain

$$\frac{\partial \tilde{F}_D}{\partial \mathbf{G}_1} \mathbf{T}_2^T = \frac{\partial \tilde{F}_D}{\partial \mathbf{G}_1}, \quad \frac{\partial \tilde{F}_D}{\partial \mathbf{H}_1} \mathbf{T}_2^T = \frac{\partial \tilde{F}_D}{\partial \mathbf{H}_1}, \quad (7.10)$$

$$\frac{\partial \tilde{F}_D}{\partial \mathbf{H}_2} \mathbf{T}_2^T = \frac{\partial \tilde{F}_D}{\partial \mathbf{H}_2}, \quad \frac{\partial \tilde{F}_D}{\partial \mathbf{G}_2} \mathbf{T}_2^T = \frac{\partial \tilde{F}_D}{\partial \mathbf{G}_2}. \quad (7.11)$$

Now, let $(\mathbf{B}'_0, \mathbf{G}'_0, \mathbf{H}'_0) \stackrel{d}{=} (\mathbf{B}_0, \mathbf{G}_0, \mathbf{H}_0)$ and let $(\mathbf{B}'_i, \mathbf{G}'_i, \mathbf{H}'_i)$ for $i = 1, \dots, k$ be the sequence generated by SGD, applied to the transformed data $\tau(D)$. By assumption, we have $\mathbf{B}'_0 \stackrel{d}{=} \mathbf{B}_0$, $\mathbf{G}_0 \stackrel{d}{=} \mathbf{G}'_0 \stackrel{d}{=} \mathbf{G}'_0 \mathbf{T}_2$ and $\mathbf{H}_0 \stackrel{d}{=} \mathbf{H}'_0 \stackrel{d}{=} \mathbf{H}'_0 \mathbf{T}_2$. We now show by induction that $\mathbf{B}'_i \stackrel{d}{=} \mathbf{B}_i$, $\mathbf{G}_i \stackrel{d}{=} \mathbf{G}'_i \stackrel{d}{=} \mathbf{G}'_i \mathbf{T}_2$ and $\mathbf{H}_i \stackrel{d}{=} \mathbf{H}'_i \stackrel{d}{=} \mathbf{H}'_i \mathbf{T}_2$ for all indices $i = 1, \dots, k$. Using equations (7.4) and (7.7) and the inductive hypothesis, we have

$$\begin{aligned} \mathbf{B}'_{i+1} &= \mathbf{B}'_i - \eta_i \frac{\partial F_{\tau(D_i)}}{\partial \mathbf{B}}(\mathbf{B}'_i, \mathbf{G}'_i, \mathbf{H}'_i) \\ &= \mathbf{B}'_i - \eta_i \frac{\partial \tilde{F}_{D_i}}{\partial \mathbf{B}}(\mathbf{B}'_i, \mathbf{G}'_i, \mathbf{H}'_i \mathbf{T}_2, \mathbf{H}'_i, \mathbf{G}'_i \mathbf{T}_2) \\ &\stackrel{d}{=} \mathbf{B}_i - \eta_i \frac{\partial \tilde{F}_{D_i}}{\partial \mathbf{B}}(\mathbf{B}_i, \mathbf{G}_i, \mathbf{H}_i, \mathbf{H}_i, \mathbf{G}_i) \\ &= \mathbf{B}_i - \eta_i \frac{\partial F_{\tau(D_i)}}{\partial \mathbf{B}}(\mathbf{B}_i, \mathbf{G}_i, \mathbf{H}_i) = \mathbf{B}_{i+1}. \end{aligned}$$

Similarly, using equations (7.5), (7.8) and (7.11) and the inductive hypothesis, we see that

$$\begin{aligned}
\mathbf{G}'_{i+1} &= \mathbf{G}'_i - \eta_i \frac{\partial F_{\tau(D_i)}}{\partial \mathbf{G}}(\mathbf{B}'_i, \mathbf{G}'_i, \mathbf{H}'_i) \\
&= \mathbf{G}'_i - \eta_i \left(\frac{\partial \tilde{F}_{D_i}}{\partial I}(\mathbf{B}'_i, \mathbf{G}'_i, \mathbf{H}'_i \mathbf{T}_2, \mathbf{H}'_i, \mathbf{G}'_i \mathbf{T}_2) \right. \\
&\quad \left. + \frac{\partial \tilde{F}_{D_i}}{\partial L}(\mathbf{B}'_i, \mathbf{G}'_i, \mathbf{H}'_i \mathbf{T}_2, \mathbf{H}'_i, \mathbf{G}'_i \mathbf{T}_2) \mathbf{T}_2^T \right) \\
&= \mathbf{G}'_i - \eta_i \left(\frac{\partial \tilde{F}_{D_i}}{\partial \mathbf{G}_1}(\mathbf{B}'_i, \mathbf{G}'_i, \mathbf{H}'_i \mathbf{T}_2, \mathbf{H}'_i, \mathbf{G}'_i \mathbf{T}_2) \right. \\
&\quad \left. + \frac{\partial \tilde{F}_{D_i}}{\partial \mathbf{G}_2}(\mathbf{B}'_i, \mathbf{G}'_i, \mathbf{H}'_i \mathbf{T}_2, \mathbf{H}'_i, \mathbf{G}'_i \mathbf{T}_2) \right) \\
&\stackrel{d}{=} \mathbf{G}_i - \eta_i \left(\frac{\partial \tilde{F}_{D_i}}{\partial \mathbf{G}_1}(\mathbf{B}_i, \mathbf{G}_i, \mathbf{H}_i, \mathbf{H}_i, \mathbf{G}_i) \right. \\
&\quad \left. + \frac{\partial \tilde{F}_{D_i}}{\partial \mathbf{G}_2}(\mathbf{B}_i, \mathbf{G}_i, \mathbf{H}_i, \mathbf{H}_i, \mathbf{G}_i) \right) \\
&= \mathbf{G}_i - \eta_i \frac{\partial F_{D_i}}{\partial \mathbf{G}}(\mathbf{B}_i, \mathbf{G}_i, \mathbf{H}_i) = \mathbf{G}_{i+1}.
\end{aligned}$$

One proceeds analogously for \mathbf{H}'_{i+1} using equations (7.6), (7.9) and (7.10). Similarly, one also sees that $\mathbf{G}'_{i+1} \mathbf{T}_2 \stackrel{d}{=} \mathbf{G}_{i+1}$ and $\mathbf{H}'_{i+1} \mathbf{T}_2 \stackrel{d}{=} \mathbf{H}_{i+1}$ combining the previous equations with symmetry and orthogonality of \mathbf{T}_2 .

In summary, we have

$$\begin{aligned}
\mathcal{L}(D, \mathbf{x}) &= f(\mathbf{B}_k, \mathbf{G}_k \mathbf{u} + \mathbf{H}_k \mathbf{v}, \mathbf{H}_k \mathbf{u} + \mathbf{G}_k \mathbf{v}) \\
&\stackrel{d}{=} f(\mathbf{B}'_k, \mathbf{G}'_k u + \mathbf{H}'_k v, \mathbf{H}'_k u + \mathbf{G}'_k v) \\
&\stackrel{d}{=} f(\mathbf{B}'_k, \mathbf{G}'_k u + \mathbf{H}'_k \mathbf{T}_2 v, \mathbf{H}'_k u + \mathbf{G}'_k \mathbf{T}_2 v) \\
&= \mathcal{L}(\tau(D), \tau(\mathbf{x})),
\end{aligned}$$

which concludes the proof. \square

Remark 7.3 (On the assumptions of Theorem 7.2). At first glance, the assumptions of Theorem 7.2 might seem quite restrictive, especially the assumption about the invariance of the distributions of \mathbf{G}_0 and \mathbf{H}_0 with respect to right-multiplication by the symmetric orthogonal matrix \mathbf{T}_2 . Yet, this hypothesis holds, e.g., when the entries of \mathbf{G}_0 and \mathbf{H}_0 are independently and identically distributed according to a centered normal distribution thanks to the rotational invariance of isotropic random Gaussian vectors (see, e.g., [98, Proposition 3.3.2]). This is the case in common initialization strategies such as Xavier initialization [99]. In addition, numerical results presented in Section 7.3 suggest that rating impossibility might hold in more general settings, such as when the model f includes ReLU activations (hence, when F has points of nondifferentiability) or for models trained via Adam as opposed to SGD.

Application to identity effects

As a practical application of Theorem 7.2 to identity effects, we consider the problem of classifying identical two-letter words of the English alphabet $\mathcal{A} := \{A, B, \dots, Z\}$, following 67. Consider a training set D formed by two-letter words that do not contain Y nor Z . Words are assigned the label 1 if they are composed by identical letters and 0 otherwise. Our goal is to verify whether a learning algorithm is capable of generalizing this pattern correctly to words containing the letters Y or Z . The transformation τ of Theorem 7.2 is defined by

$$\tau(xY) = xZ, \quad \tau(xZ) = xY, \quad \text{and} \quad \tau(xy) = xy, \quad (7.12)$$

for all letters $x, y \in \mathcal{A}$, with $y \neq Y, Z$. Note that this transformation is of the form $\tau = I \otimes \tau_2$, where I is the identity map. Hence, it fits the setting of Theorem 7.2. Moreover, since D does not contain Y nor Z letters, $\tau(D) = D$. Hence, condition (i) of Theorem 7.1 is satisfied.

In order to represent letters as vectors of \mathbb{R}^q , we need to use a suitable encoding. Its choice is crucial to determine the properties of the transformation matrix \mathbf{T}_2 associated with τ_2 , needed to apply Theorem 7.2. Formally, an encoding of an alphabet \mathcal{A} is a set of vectors $\mathcal{E} \subseteq \mathbb{R}^d$, of the same cardinality of \mathcal{A} , to which letters can be associated with. In our case, $|\mathcal{A}| = 26 = |\mathcal{E}|$. We say that an encoding is orthogonal if it is an orthonormal set of \mathbb{R}^d . For example, the popular one-hot encoding $\mathcal{E} = \{\mathbf{e}_i\}_{i=1}^{26} \subseteq \mathbb{R}^{26}$, i.e., the canonical basis of \mathbb{R}^{26} , is an orthogonal encoding.

In this setting, every word is modeled as a graph defined by two nodes connected by a single unweighted and undirected edge. Each node v is labeled with a node feature $\alpha(v) \in \mathbb{R}^q$, corresponding to a letter's encoding. For instance, Figure 7.1 represents a two-letter word with one-hot encoding.

Theorem 7.4 (Inability of GNNs to classify identical two-letter words outside the training set). *Let $\mathcal{E} \subseteq \mathbb{R}^{26}$ be an orthogonal encoding of the English alphabet \mathcal{A} and let L be a learner obtained by training a GNN of the form (2.4) via SGD to classify identical two-letter words. Assume that words in the training set D do not contain the letter Y nor Z . Then, L assigns the same rating (in distribution) to any word of the form xy where $y \in \{Y, Z\}$, i.e., $\mathcal{L}(D, xY) \stackrel{d}{=} \mathcal{L}(D, xZ)$ for any $x \in \mathcal{A}$. Hence, it is unable to generalize to identity effect outside the training set.*

Proof. As discussed above, the transformation τ defined by (7.12) is of the form $\tau = I \otimes \tau_2$. Moreover, the matrix associated with the linear transformation τ_2 is of the form $\mathbf{T}_2 = \mathbf{B}^{-1}\mathbf{P}\mathbf{B}$, where \mathbf{B} is the change-of-basis matrix from the orthonormal basis associated with the encoding \mathcal{E} to the canonical basis of \mathbb{R}^{26} (in particular, \mathbf{B} is orthogonal and $\mathbf{B}^{-1} = \mathbf{B}^T$) and \mathbf{P} is a permutation matrix that switches the last two entries of a vector, i.e., using block-matrix notation,

$$\mathbf{P} = \left[\begin{array}{c|cc} \mathbf{I} & & \mathbf{0} \\ \hline & 0 & 1 \\ \mathbf{0} & 1 & 0 \end{array} \right], \quad \mathbf{I} \in \mathbb{R}^{24 \times 24}.$$

Hence, \mathbf{T}_2 is orthogonal and symmetric, and therefore fits the framework of the Theorem 7.2

On the other hand, as discussed in Section 7.2.1, every GNN of the form (2.4) is a model of the form (7.2). Thus, Theorem 7.2 yields $\mathcal{L}(D, xy) \stackrel{d}{=} \mathcal{L}(\tau(D), \tau(xy))$, for all letters $x, y \in \mathcal{A}$. In particular, $\mathcal{L}(D, xY) \stackrel{d}{=} \mathcal{L}(\tau(D), xZ)$, which corresponds to condition (ii) of Theorem 7.1. Recalling that $\tau(D) = D$, also condition (i) holds. Hence, we can apply Theorem 7.1 and conclude the proof. \square

7.2.2 What GNNs can learn: identity effects on dicyclic graphs

We now analyze the expressivity of GNNs to learn identity effects related to the *topology* of the graphs in the dataset. This novel setting requires to design *ex novo* the formulation of our problem. In fact, we are not focusing on the feature matrix X_G of a graph anymore, but on its adjacency matrix A , which contains all the topological information. Here we focus on a particular class of graphs, which we call dicyclic graphs. A dicyclic graph is a graph composed by an m -cycle and an n -cycle, linked by a single edge. Since a dicyclic graph is uniquely determined by the length of the two cycles, we can identify it with the equivalence class $[m, n]$ over the set of pairs (a, b) , $a, b \in \mathbb{N}$, defined as $[m, n] := \{(m, n), (n, m)\}$. A dicyclic graph $[m, n]$ is symmetric if $m = n$ and asymmetric otherwise.

In this section we provide an analysis of the expressive power of GNNs when learning identity effects on dicyclic graphs (i.e., classifying whether a dicyclic graph is symmetric or not). We start by proving a lemma that shows how information propagates through the nodes of a cycle, during the 1-WL test iterations, when one of the nodes has a different initial color with respect to all the other nodes.

Lemma 7.5 (1-WL test on m -cycles). *Consider an m -cycle in which the vertices are numbered from 0 to $m - 1$ clockwise, an initial coloring $c^{(0)} = [0, 1, \dots, 1]^T \in \mathbb{N}^m$ (vector indexing begins from 0, and the vector is meant to be circular, i.e., $c^{(0)}(m) = c^{(0)}(0)$), and define the function HASH as*

$$\begin{cases} \text{HASH}(0, \{j, h\}) = 0 \\ \text{HASH}(i, \{j, h\}) = i & \text{if } j \neq h, i < \lfloor \frac{m}{2} \rfloor \\ \text{HASH}(i, \{j, h\}) = i + 1 & \text{if } j = h, i < \lfloor \frac{m}{2} \rfloor \\ \text{HASH}(\lfloor \frac{m}{2} \rfloor, \{j, h\}) = \lfloor \frac{m}{2} \rfloor \end{cases},$$

with $j, h \leq \lfloor \frac{m}{2} \rfloor$. Then, HASH is an injective coloring over the m -cycle at each iteration k of the 1-WL test. This gives, at each iteration $0 \leq k < L = \lfloor \frac{m}{2} \rfloor$, the coloring

$$\begin{cases} c^{(k)}(i) = i & \text{if } 0 \leq i \leq k \\ c^{(k)}(i) = k + 1 & \text{if } k < i < m - k \\ c^{(k)}(i) = m - i & \text{if } m - k \leq i < m \end{cases}, \quad (7.13)$$

and the 1-WL test terminates after $L = \lfloor \frac{m}{2} \rfloor$ iterations (i.e., $c^{(L)} = c^{(L-1)}$), giving $\lfloor \frac{m}{2} \rfloor + 1$ colors.

Proof. We prove the lemma by induction on k .

Case $t = 1$ We start with $c^{(0)}(0) = 0$ and $c^{(0)}(i) = 1$, for $i = 1, \dots, m-1$. We only have three hashing cases:

- $\text{HASH}(0, \{\{1, 1\}\}) = 0$, the color assigned to node 0;
- $\text{HASH}(1, \{\{0, 1\}\}) = 1$, the color assigned to nodes 1 and $m-1$;
- $\text{HASH}(1, \{\{1, 1\}\}) = 2$, the color assigned to all nodes $1 < i < m-1$.

This shows that $c^{(1)}$ satisfies [\(7.13\)](#) and that HASH is injective at iteration $k = 1$. Hence, the claim is true for $k = 1$.

Inductive step $k \rightarrow k+1$ Assume that the inductive hypothesis is true for step t . Hence, our coloring is of the form [\(7.13\)](#) and that HASH is injective at iteration k . This means that for $0 < i \leq k$ we have $c^{(k)}(i-1) < c^{(k)}(i) < c^{(k)}(i+1)$ and for $m-k \leq i < m-1$ we have $c^{(k)}(i+1) < c^{(k)}(i) < c^{(k)}(i-1)$; thus, for $0 < i \leq k$ or $m-k-1 \leq i < m-1$, we see that

$$c^{(k+1)}(i) = \text{HASH}(c^{(k)}(i), \{\{c^{(k)}(i-1), c^{(k)}(i+1)\}\}) = i.$$

For $i = k+1$ we have $c^{(k)}(i-1) < c^{(k)}(i) = c^{(k)}(i+1)$ and for $i = m-k-2$ we have $c^{(k)}(i+1) < c^{(k)}(i) = c^{(k)}(i-1)$; therefore, for $i = k+1$ and $i = m-k-2$, we also have

$$c^{(k+1)}(i) = \text{HASH}(c^{(k)}(i), \{\{c^{(k)}(i-1), c^{(k)}(i+1)\}\}) = i.$$

For all the remaining indices $k+1 < i < m-k-2$, we have $c^{(k)}(i-1) = c^{(k)}(i) = c^{(k)}(i+1)$, so

$$\begin{aligned} c^{(k+1)}(i) &= \text{HASH}(c^{(k)}(i), \{\{c^{(k)}(i-1), c^{(k)}(i+1)\}\}) \\ &= (k+1) + 1 = k+2. \end{aligned}$$

The HASH function is still injective, as for $0 < i \leq k+1$ we have $c^{(k)}(i-1) < c^{(k)}(i) < c^{(k)}(i+1)$, for $m-k-1 \leq i < m-1$ we have $c^{(k)}(i+1) < c^{(k)}(i) < c^{(k)}(i-1)$, and for $k+1 < i < m-k-1$ it holds $\text{HASH}(c^{(k)}(i), \{\{c^{(k)}(i-1), c^{(k)}(i+1)\}\}) = \text{HASH}(k+1, \{\{k+1, k+1\}\}) = k+2$. Therefore, we have

$$\begin{cases} c^{(k+1)}(i) = i & \text{if } 0 \leq i \leq k+1 \\ c^{(k+1)}(i) = k+2 & \text{if } k+1 < i < m-k-1. \\ c^{(k+1)}(i) = m-i & \text{if } m-k-1 \leq i < m \end{cases}$$

Termination of the 1-WL test At iteration $\lfloor \frac{m}{2} \rfloor - 1$ we have

$$\begin{cases} c^{(\lfloor \frac{m}{2} \rfloor - 1)}(i) = i & \text{if } 0 \leq i \leq \lfloor \frac{m}{2} \rfloor - 1 \\ c^{(\lfloor \frac{m}{2} \rfloor - 1)}(i) = \lfloor \frac{m}{2} \rfloor & \text{if } i = \lfloor \frac{m}{2} \rfloor \text{ or } i = \lceil \frac{m}{2} \rceil. \\ c^{(\lfloor \frac{m}{2} \rfloor - 1)}(i) = m-i & \text{if } \lceil \frac{m}{2} \rceil + 1 \leq i < m \end{cases}$$

This concludes the proof. \square

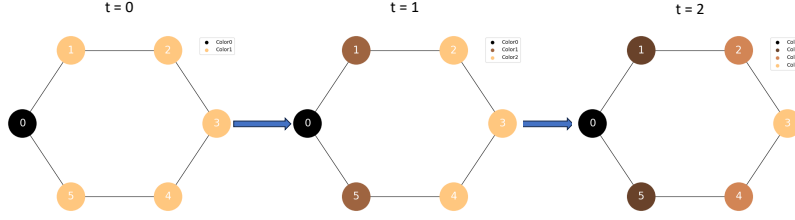


Figure 7.2: Graphical illustration of Lemma 7.5: a 6-cycle reaches a stable coloring in $\lfloor \frac{6}{2} \rfloor = 3$ steps with $\lfloor \frac{6}{2} \rfloor + 1 = 4$ colors. Numbers are used to identify nodes.

A graphical representation of Lemma 7.5 can be found in Figure 7.2. We observe that the specific node indexing of Lemma 7.5 was adopted just to ease computations; nevertheless, it is possible to construct a HASH function for other choices of node indexing. This is due to the fact that the mapping depends only on the topological structure in each node's neighborhood. This lemma represents the core of next theorem's proof, which establishes the ability of the 1-WL test to classify dicyclic graphs with identical cycles. Intuitively, if we have a dicyclic graph where node colors are uniformly initialized, one step of 1-WL test yields a coloring depending entirely on the number of neighbours for each node. In a dicyclic graph $[m, n]$ we always have $m + n - 2$ nodes of degree two and 2 nodes of degree three, so $c^{(1)}(i) = 1$ for all 2-degree nodes i , and $c^{(1)}(j) = 0$ for the two 3-degree nodes j . Hence, each cycle of the dicyclic graph satisfies the initial coloring hypothesis of Lemma 7.5.

Theorem 7.6 (1-WL test on dicyclic graphs). *The 1-WL test gives the same color to the 3-degree nodes of a uniformly colored dicyclic graph $[m, n]$ (i.e., $c^{(0)} = 0 \in \mathbb{N}^{m+n}$) if and only if $m = n$. Therefore, the 1-WL test can classify symmetric dicyclic graphs.*

Proof. After one iteration on the 1-WL test, regardless of the symmetry of the dicyclic graph, we obtain a coloring in which only 3-degree nodes have a different color, whose value we set to 0. We can therefore split the coloring vector $c^{(1)} \in \mathbb{N}^{m+n}$ in two subvectors, namely, $c^{(1)} = [(c_1^{(1)})^L, (c_2^{(1)})^L]^T$ corresponding to each cycle, respectively, and where $c_1^{(1)}(0)$ and $c_2^{(1)}(0)$ correspond to the 3-degree nodes. We treat the symmetric and the asymmetric cases separately.

The symmetric case We let $c_1^{(0)} = c_2^{(0)} = c_0^{(0)}$, with $c_0^{(0)} = [0, 1, \dots, 1]$. In this case, we run the 1-WL test in parallel on both vectors $c_1^{(k)}$ and $c_2^{(k)}$, where the HASH function in Lemma 7.5 is extended on the 3-degree nodes as $\text{HASH}(0, \{0, j, h\}) = 0$. Therefore, for each $k \geq 0$,

$$c_0^{(k+1)}(0) = \text{HASH}(c_0^{(k)}(0), \{c_0^{(k)}(0), c_0^{(k)}(1), c_0^{(k)}(m-1)\}) = 0.$$

Thanks to Lemma 7.5 we obtain $c_1^{(\lfloor \frac{m}{2} \rfloor)} = c_2^{(\lfloor \frac{m}{2} \rfloor)}$, which is a stable coloring for the whole graph, as the color partition is not refined anymore.

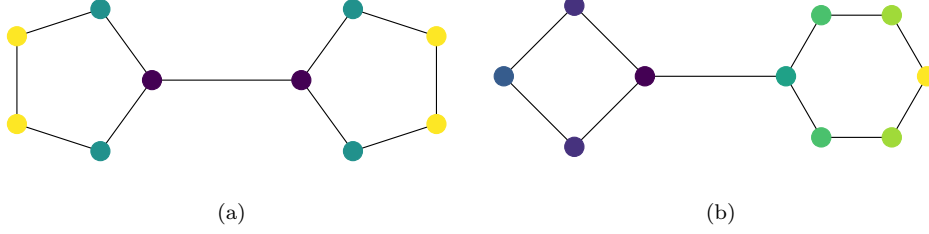


Figure 7.3: Stable 1-WL coloring for different types of dicyclic graphs: as stated in Theorem 7.6, 3-degree nodes have the same color in symmetric dicyclic graphs, and different color in the asymmetric ones.

The asymmetric case Without loss of generality, we can assume $m = \text{length}(c_1^{(k)}) \neq \text{length}(c_2^{(k)}) = m + \delta$ for some $\delta \in \mathbb{N}$, $\delta > 0$. We also assume for now that m is odd (the case m even will be briefly discussed later). We extend the HASH function from Lemma 7.5 to colors $j, h > \lfloor \frac{m}{2} \rfloor$. For $j > \lfloor \frac{m}{2} \rfloor$ or $k > \lfloor \frac{m}{2} \rfloor$ we define

$$\begin{cases} \text{HASH}(0, \{j, h\}) = \infty \\ \text{HASH}(i, \{j, h\}) = \lfloor \frac{m}{2} \rfloor + i & \text{if } j \neq k, i \leq \lfloor \frac{m}{2} \rfloor \\ \text{HASH}(i, \{j, h\}) = \lfloor \frac{m}{2} \rfloor + i + 1 & \text{if } j = k, i \leq \lfloor \frac{m}{2} \rfloor \end{cases}$$

Running in parallel the 1-WL test on the two cycles, computing the coloring vectors $c_1^{(\lfloor \frac{m}{2} \rfloor + 1)}$ and $c_2^{(\lfloor \frac{m}{2} \rfloor + 1)}$ up to iteration $\lfloor \frac{m}{2} \rfloor + 1$, for $i = \lfloor \frac{m}{2} \rfloor + 1$ we have $c_2(i) = \lfloor \frac{m}{2} \rfloor + 1$. Therefore, given the extension of the HASH function just provided, this new color starts to backpropagate on the indices $i < \lfloor \frac{m}{2} \rfloor + 1$, $i > m - h - \lfloor \frac{m}{2} \rfloor - 1$ until it reaches the index 0. As a consequence, it exists an iteration index L such that $c_2^{(L)}(0) = \text{HASH}(0, \{j, k^*\})$ with $k^* > \lfloor \frac{m}{2} \rfloor$ and, finally, $c_2^{(L)}(0) = \infty$, giving $c_1^{(L)}(0) \neq c_2^{(L)}(0)$, as claimed.

The case in which m is even works analogously, but we have to modify the HASH function in a different way to preserve injectivity. In particular, for $j, h \leq m/2$, we define

$$\begin{cases} \text{HASH}(i, \{j, h\}) = \frac{m}{2} & \text{if } j = k, i = \frac{m}{2} \\ \text{HASH}(i, \{j, h\}) = \frac{m}{2} + 1 & \text{if } j \neq k, i = \frac{m}{2} \end{cases}$$

This concludes the proof. \square

Theorem 7.6 establishes in a deterministic way the power of the 1-WL test in terms of distinguishing between symmetric and asymmetric dicyclic graphs, given a sufficient number of iterations directly linked with the maximum cycle length in the considered domain. Examples of 1-WL stable colorings on dicyclic graphs are presented in Figure 7.3.

Employing well-known results in the literature concerning the expressive power of GNNs (see [20, 13] and in particular Theorem 2.2), we can prove the main result of this subsection on the classification power of GNNs on the domain of dicyclic graphs.

Corollary 7.7 (GNNs can classify symmetric dicyclic graphs). *There exist a GNN of the form (2.4) and a READOUT function able to classify symmetric dicyclic graphs.*

Proof. Let $[m, n]$ be a dicyclic graph and $c^{(L)}$ be the stable coloring of $[m, n]$ produced by the 1-WL test with initial uniform coloring. By Theorem 7.6 the graph can be correctly classified by the 1-WL test, i.e., by its stable coloring. Using Theorem 2.2, a GNN f_{Θ} exists such that f_{Θ} can learn the stable coloring for each input graph for each iteration step t . Let $c^{(L)}$ be the stable coloring computed by a GNN for a dicyclic graph $[m, n]$. Let (u, v) be the 3-degree nodes of the dicyclic graph. Then, the READOUT can be modeled as

$$\text{READOUT}(c^{(L)}) = \begin{cases} 1 & \text{if } c^{(L)}(u) = c^{(L)}(v) \\ 0 & \text{otherwise} \end{cases}.$$

With such a READOUT, the GNN assigns the correct rating to the dicyclic graph (i.e., 1 if the graph is symmetric, 0 otherwise). \square

Remark 7.8 (The gap between theory and practice in Corollary 7.7). Corollary 7.7 shows that GNNs are powerful enough to match the 1-WL test’s expressive power for the classification of symmetric dicyclic graphs (as established by Theorem 7.6). However, it is worth underlining that this result only proves the *existence* of a GNN model able to perform this task. In contrast to the results presented in Section 7.2.1, this corollary does not mention any training procedure. Nevertheless, the numerical experiments in Section 7.3.3 show that GNNs able to classify symmetric dicyclic graphs *can* be trained in practice, albeit achieving generalization outside the training set is not straightforward and depends on the GNN architecture.

7.3 Numerical results

This section presents the results of experimental tasks designed to validate our theorems. We analyze the consistency between theoretical and numerical findings, highlighting the significance of specific hypotheses, and addressing potential limitations of the theoretical results.

7.3.1 Experimental Setup

We take in account two different models for our analysis:

- The Global Additive Pooling GNN (*Gconv-glob*) applies a sum pooling at the end of the message-passing convolutional layers [12]. In the case of the 2-letter words setting, the resulting vector $\mathbf{h}_{\text{glob}} \in \mathbb{R}^h$ undergoes processing by a linear layer, while in the dicyclic graphs setting, an MLP is employed. A sigmoid activation function is applied at the end.
- The Difference GNN (*Gconv-diff*), takes the difference between the hidden states of the two nodes in the graph (in the 2-letter words setting) or the difference between the hidden states of the 3-degree nodes (in the dicyclic graphs setting) after the message-passing convolutional layers. The resulting vector $\mathbf{h}_{\text{diff}} \in \mathbb{R}^h$ is then fed into a final linear layer, followed by the application of a sigmoid activation function.

The choice of the last READOUT part is driven by empirical observation on their effectiveness on the two different tasks.

Training is performed on an Intel(R) Core(TM) i7-9800X processor running at 3.80GHz using 31GB of RAM along with a GeForce GTX 1080 Ti GPU unit².

7.3.2 Case study #1: two-letter words

To validate Theorem 7.2 we consider a classification task using the two-letter word identity effect problem described in Section 7.2.1, following the experimental setup presented in 67.

Task and datasets

In accordance with the setting of Section 7.2.1, each word is represented as a graph consisting of two nodes connected by a single unweighted and undirected edge (see Figure 7.1). Each node is assigned a node feature $\mathbf{x} \in \mathbb{R}^{26}$, corresponding to a letter's encoding.

The training set D_{train} includes all two-letter words composed of any English alphabet letters except Y and Z. The test set D_{test} is a set of two-letter words where at least one of the letters is chosen from Y, Z. Specifically, we consider $D_{\text{test}} = \{\text{YY}, \text{ZZ}, \text{YZ}, \text{ZT}, \text{EY}, \text{SZ}\}$.

Vertex feature encodings

In our experiments, we consider four different encodings of the English alphabet, following the framework outlined in Section 7.2.1. Each encoding consists of a set of vectors drawn from \mathbb{R}^{26} .

- *One-hot encoding*: This encoding assigns a vector from the canonical basis to each letter: A is encoded as \mathbf{e}_1 , B as \mathbf{e}_2 , ..., and Z as \mathbf{e}_{26} .
- *Haar encoding*: This encoding assigns to each letter the columns of a 26×26 orthogonal matrix drawn from the orthogonal group $O(26)$ using the Haar distribution 100.
- *Distributed encoding*: This encoding assigns a random combination of 26 bits to each letter. In this binary encoding, only j bits are set to 1, while the remaining $26 - j$ bits are set to 0. In our experiments, we set $j = 6$.
- *Gaussian encoding*: This encoding assigns samples from the multivariate normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$, where $\mathbf{0} \in \mathbb{R}^n$ and $\mathbf{I} \in \mathbb{R}^{n \times n}$. In our experiments, we set $n = 16$.

Observe that only the one-hot and the Haar encodings are orthogonal (see Section 7.2.1) and hence satisfy the assumption of Theorem 7.4. On the other hand, the distributed and the Gaussian encodings do not fall within the setting of Theorem 7.4.

²Code available at https://github.com/AleDinve/gnn_identity_effects.git

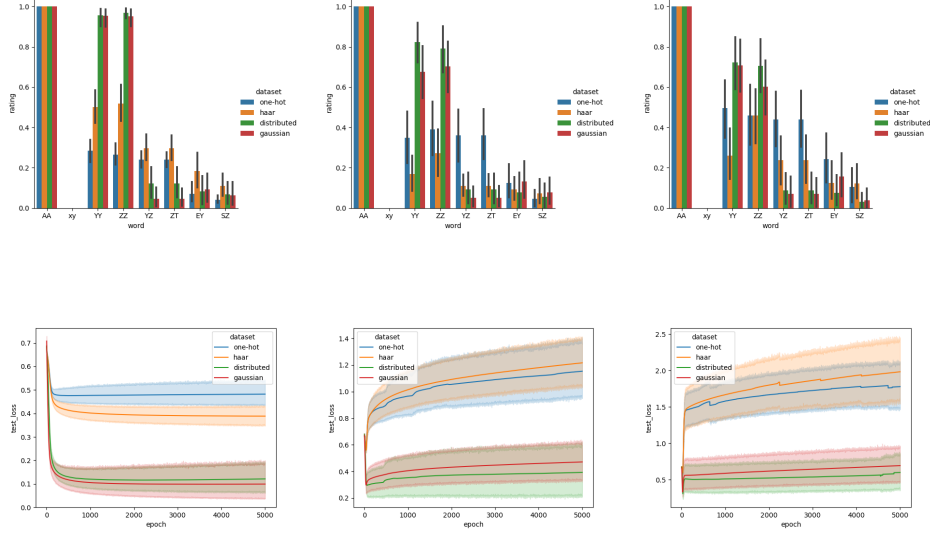


Figure 7.4: Numerical results for the rating task on the two-letter words dataset using Gconv-glob with $L = 1, 2, 3$ layers. Rating should be equal to 1 if words are composed by identical letters, 0 otherwise. The distributed and Gaussian encodings, which deviate from the framework outlined in Theorem 7.2, exhibit superior performance compared to the other encodings. The other encodings makes the transformation matrix orthogonal and symmetric, being themselves orthogonal encodings.

We run 40 trials for each model (i.e., Gconv-glob or Gconv-diff, defined in Section 7.3.1) with l layers (ranging from 1 to 3). In each trial, a different training set is randomly generated. The models are trained for 5000 epochs using the Adam optimizer with a learning rate of $\lambda = 0.0025$, while minimizing the binary cross-entropy loss. The hidden state dimension is set to $d = 64$, and Rectified Linear Units (ReLUs) are used as activation functions.

The numerical results are shown in Figures 7.4 7.5, where we propose two different types of plots:

- On the top row, we compare the ratings obtained using the four adopted encodings. The first two words, AA and a randomly generated word with nonidentical letters, denoted xy, are selected from the training set to showcase the training accuracy. The remaining words are taken from D_{test} , allowing assessment of the generalization capabilities of the encoding scheme outside the training test. The bars represent the mean across trials, while the segments at the center of each bar represent the standard deviation.
- On the bottom row, we show loss functions with respect to the test set over the training epochs for each encoding. The lines represent the average, while the shaded areas represents the standard deviation.

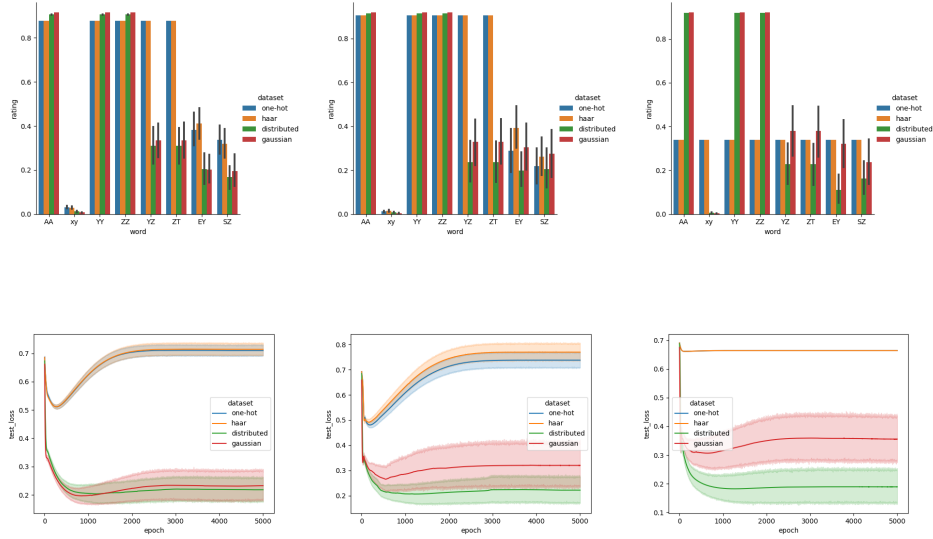


Figure 7.5: Numerical results for the rating task on the two-letter words dataset using Gconv-diff with $L = 1, 2, 3$ layers. The same observations to those in Figure 7.4 can be made here as well.

Our numerical findings indicate that the rating impossibility theorem holds true for the one-hot encoding and the Haar encoding. However, notable differences in behavior emerge for the other two encodings. The 6-bit distributed encoding exhibits superior performance across all experiments, demonstrating higher rating accuracy and better loss convergence. The Gaussian encoding yields slightly inferior results, yet still showcases some generalization capability. It is important to note that despite variations in experimental settings such as architecture and optimizer (specifically, the use of ReLU activations and the Adam optimizer), the divergent behavior among the considered encodings remains consistent. This highlights the critical role of the transformation matrix \mathbf{T}_2 within the hypothesis outlined in Theorem 7.4. It is interesting to notice that increasing the number of layers contributes to the so-called *oversmoothing effect* [101, 102]: many message passing iterations tend to homogenize information across the nodes, generating highly similar features.

7.3.3 Case study #2: dicyclic graphs

We now consider the problem of classifying unlabeled symmetric dicyclic graphs, introduced in Section 7.2.2. In Corollary 7.7 we proved the existence of GNNs able to classify symmetric dicyclic graphs. In this section, we assess whether such GNNs can be computed via training (see also Remark 7.8). With this aim, we consider two experimental settings based on different choices of training and test set: an *extraction task* and an *extrapolation task*, summarized in Figures 7.8 and 7.10, respectively, and described in detail below. Each task involves running 25 trials for the Gconv-glob and Gconv-diff models defined in Section

7.3.1 The number of layers in each model is determined based on the specific task.

The models are trained over 5000 epochs using a learning rate of $\lambda = 0.001$. We employ the Adam optimizer, minimizing the binary crossentropy, and incorporate the AMSGrad fixer [103] to enhance training stability due to the large number of layers. Labels are all initialized uniformly as $h_v^{(0)} = 1$ for each node in each graph. The hidden state dimension is set to $d = 100$, and ReLU activation functions are utilized.

The results presented in Figures 7.6, 7.8, and 7.10 should be interpreted as follows: each circle represents a dicyclic graph $[m, n]$; the color of the circle corresponds to the rating, while the circle's radius represents the standard deviation.

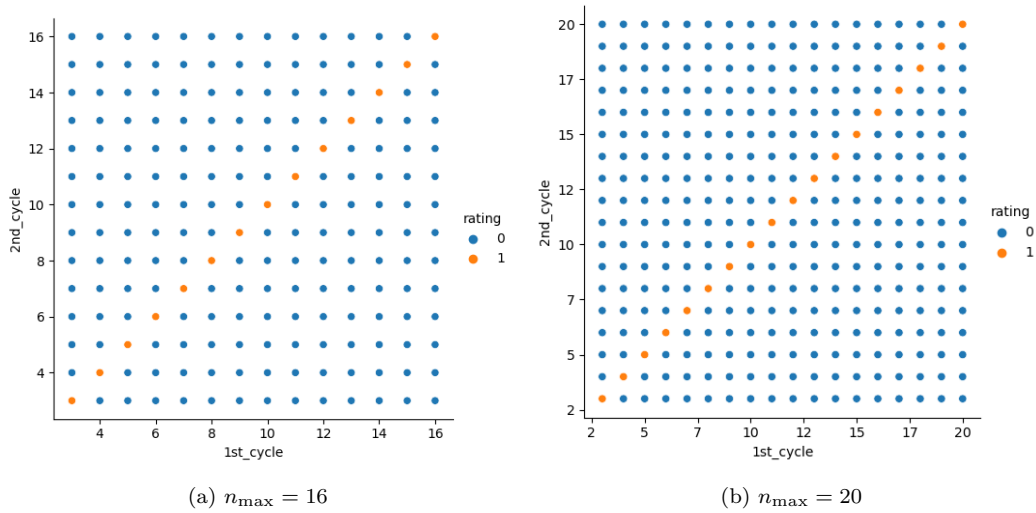


Figure 7.6: Perfect classification of symmetric dicyclic graphs by n_{\max} iterations of the 1-WL test.

1-WL test performance

In Theorem 7.6 we showed that the 1-WL test can classify symmetric dicyclic graphs. This holds true regardless of the length of the longer cycle, provided that a sufficient number of iterations is performed. The results in Figure 7.6 show that the 1-WL test achieves indeed perfect classification accuracy in n_{\max} iterations, where n_{\max} is the maximum length of a cycle in the dataset, in accordance with Theorem 7.6.

Extraction task

In this task, we evaluate the capability of GNNs to generalize to unseen data, specifically when the minimum length of cycles in the test dataset is smaller than the maximum length of those in the training dataset. More specifically, the training set D_{train} consists of pairs $[m, n]$ where $3 \leq m, n \leq n_{\max}$ and $m, n \neq k$ with $3 \leq k \leq n_{\max}$, while the test set D_{test} comprises pairs $[k, a]$ with $3 \leq a \leq n_{\max}$. Figure 7.7 illustrates this setting.

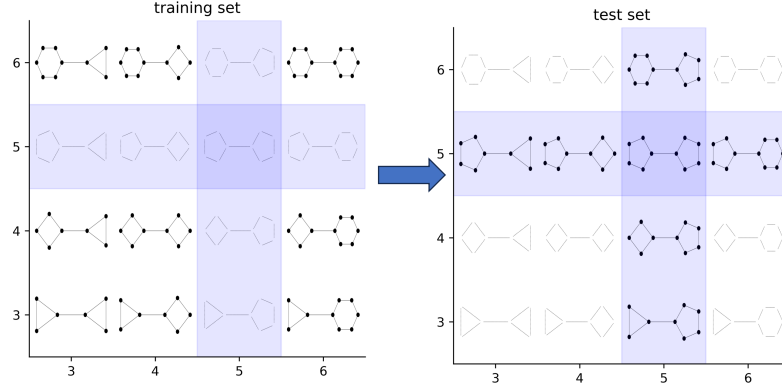


Figure 7.7: Graphical illustration of the extraction task. In this example, $n_{\max} = 6$ and $k = 5$.

In our experiments, we set $n_{\max} = 8$ and consider k values of 7, 6, and 5. In this setting, $|D_{\text{test}}| = (8 - 2) \cdot 2 - 1 = 11$ and $|D_{\text{train}}| = (8 - 2)^2 - |D_{\text{test}}| = 25$. The number of GNN layers is $L = n_{\max}$. The numerical results are presented in Figure 7.8. We observe that the Gconv-diff model achieves perfect performance in our experiments (standard deviation values are not reported because they are too low), showing consistence with the theoretical setting. On the other hand, the Gconv-glob model demonstrates good, but not perfect, performance on the test set. A critical point in our numerical examples seems to be $k = 5$, which falls in the middle range between the minimum and maximum cycle lengths in the training set (3 and 8, respectively). This particular value is closer to the minimum length, indicating a relatively unbalanced scenario.

Overall, the different performance of Gconv-diff and Gconv-glob on the extraction task shows that, despite the theoretical existence result proved in Corollary 7.7, the choice of architecture is crucial for achieving successful generalization.

Extrapolation task

In this task, we assess GNNs' ability to generalize to unseen data with cycle lengths exceeding the maximum length in the training dataset. Specifically, the training set D_{train} comprises pairs $[m, n]$ where $3 \leq m, n \leq n_{\max}$, while the test set D_{test} consists of pairs $[n_{\max} + k, n']$ with $0 < k \leq g$ and $3 \leq n' \leq n_{\max} + g$. Figure 7.9 illustrates the extrapolation task.

In our experiments, we set $n_{\max} = 8$ and consider values of g as 1, 2, and 3. The number of GNN layers is $L = n_{\max} + g$. Therefore, $|D_{\text{train}}| = (8 - 2)^2 = 36$, $|D_{\text{test}, g=1}| = (9 - 2) \cdot 2 - 1 = 13$, $|D_{\text{test}, g=2}| = (10 - 2) \cdot 4 - 4 = 28$ and $|D_{\text{test}, g=3}| = (11 - 2) \cdot 6 - 9 = 45$. Numerical results are presented in Figure 7.10. In the extraction task, both models achieved perfect training accuracy. Conversely, in the extrapolation task, the Gconv-glob model struggles to classify the training set accurately, especially when the number of layers is equal to 9. This behavior may be attributed to the homogeneous nature of sum pooling at the end of the message passing, as it does not take into account the role of 3-degree nodes (which play a key role in our theory, as illustrated by Theorem 7.6) and

Corollary [7.7](#)).

On the other hand, the Gconv-diff model consistently achieves perfect training accuracy over the training set and achieves perfect generalization for $g = 1$, showing once again the importance of architecture choice in practice. However, when $g \geq 2$ there is a noticeable region of misclassification for pairs $[m, n]$ where $m, n \geq n_{\max}$. This behavior could be explained by the limited capacity of the hidden states, but the optimization process might also play a significant role. Moreover, for $g \geq 2$ the numerical results of the extrapolation task resemble the rating impossibility phenomenon observed in the two-letter words framework. However, it is important to note that, at least for the Gconv-diff model, we observe significantly different ratings between graphs $[m, n_{\max} + g]$ where $m < n_{\max}$ and graphs $[n_{\max} + i, n_{\max} + j]$ with $i, j > 0$. In contrast, in the two-letter words framework ratings typically do not exhibit such a consistent and distinguishable pattern.

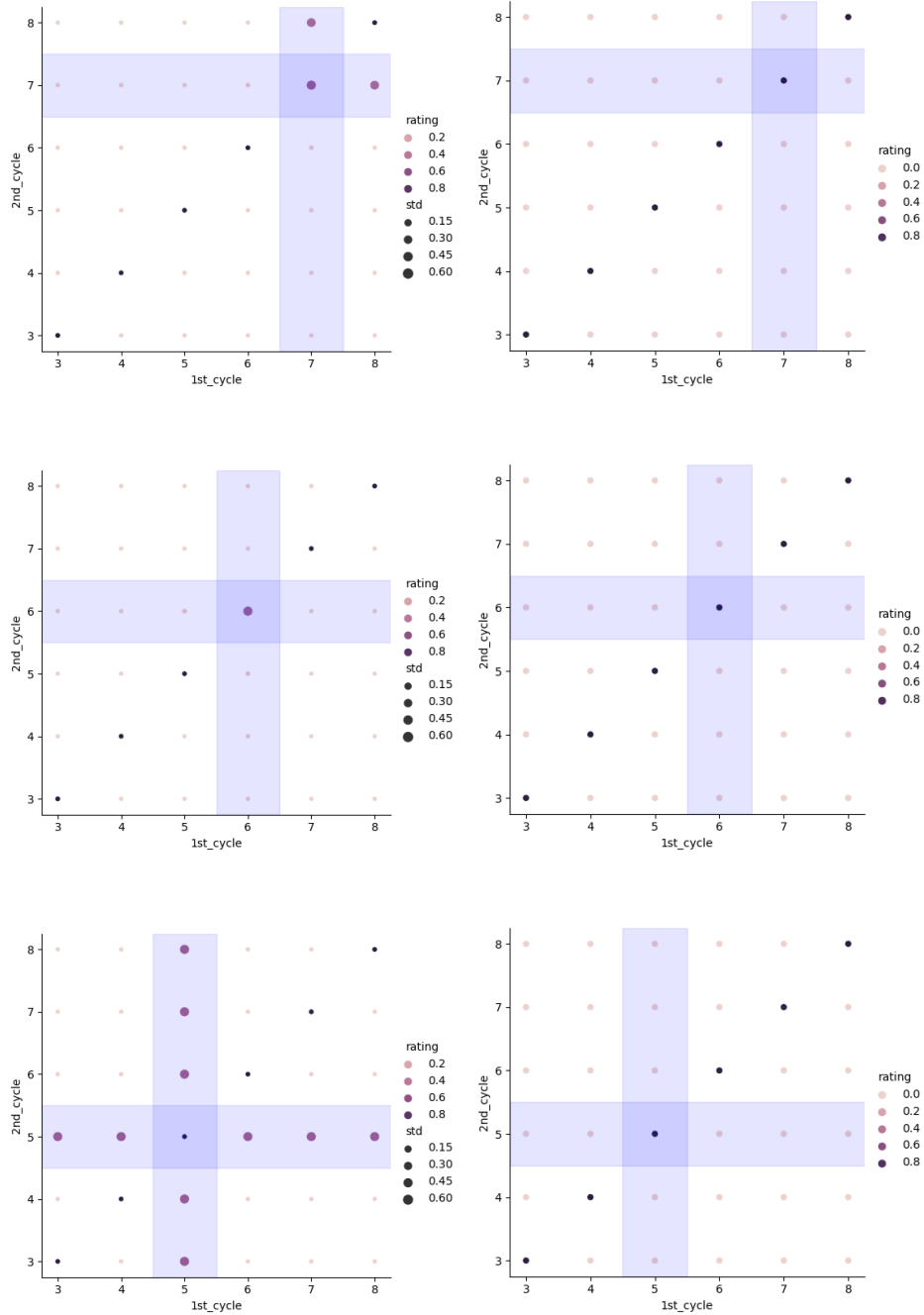


Figure 7.8: Extraction task performed by different GNN models, namely Gconv-glob (left) and Gconv-diff (right). We set $n_{\max} = 8$, $l = 8$ and, from top to bottom, $k = 7, 6, 5$.

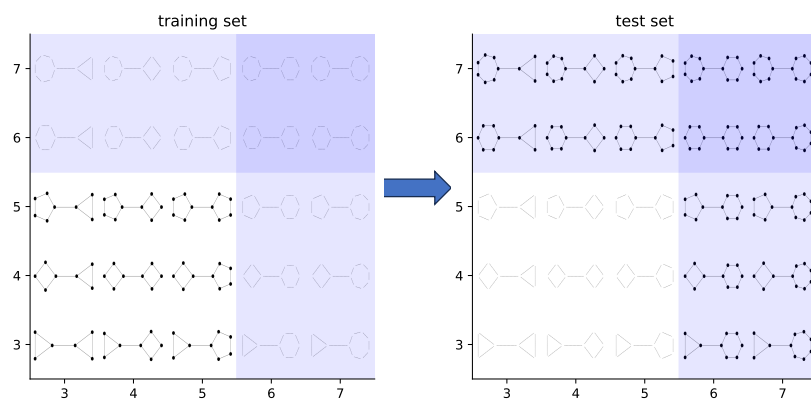


Figure 7.9: Graphical illustration of the extrapolation task. In this example, $n_{\max} = 5$ and $g = 2$.

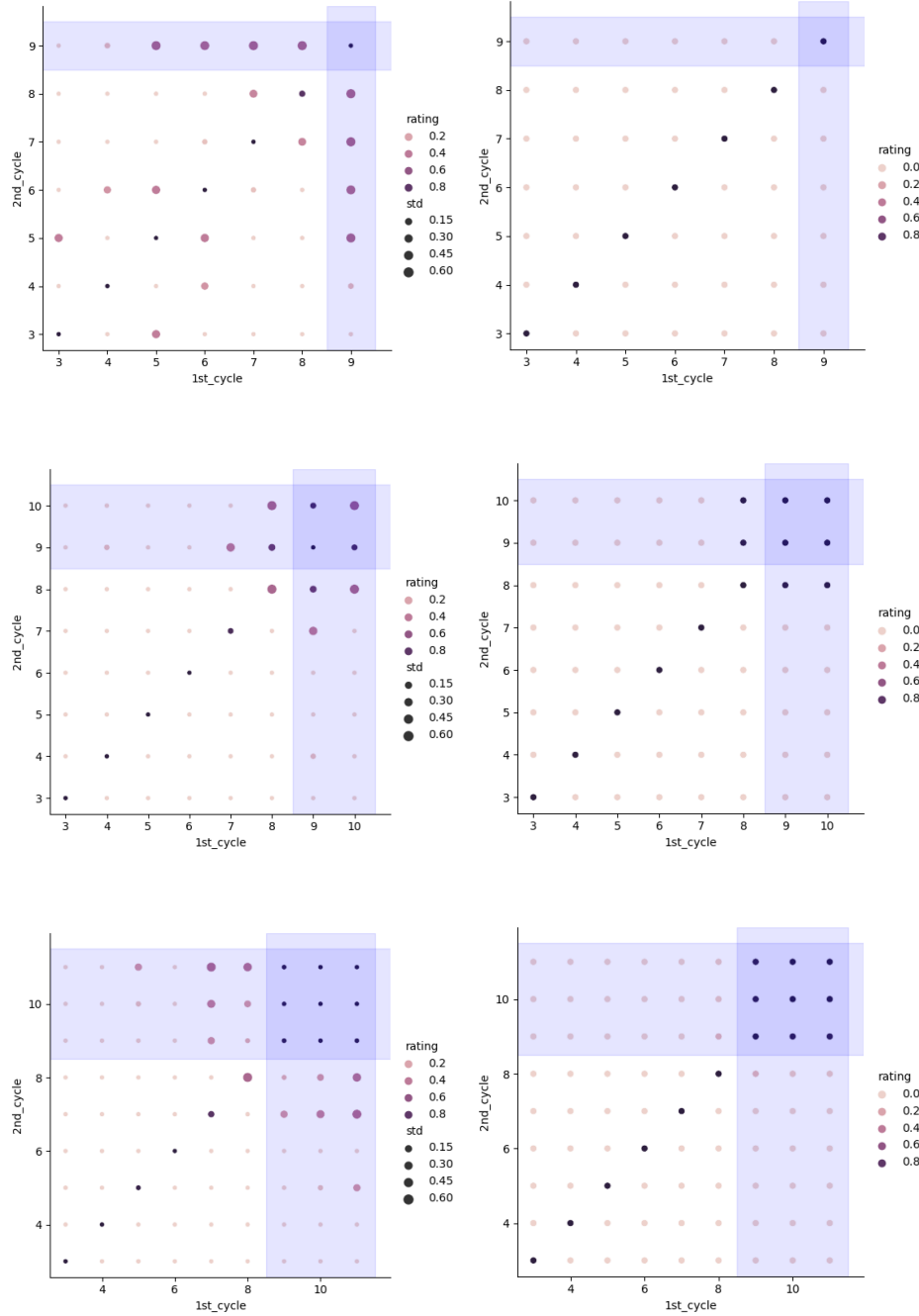
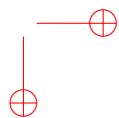
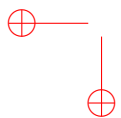
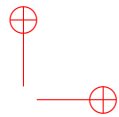
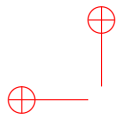


Figure 7.10: Extrapolation task performed by different GNN models, namely Gconv-glob (left) and Gconv-diff (right). We set $n_{\max} = 8$ and, from top to bottom, $(L, g) = (9, 1), (10, 2), (11, 3)$.





Chapter 8

Other works

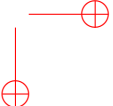
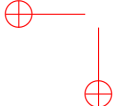
The chapter is dedicated to the discussion of works carried out during the PhD and not strictly related to the main topics of this thesis. In more detail Section [8.1](#) study some bounds on the topological complexity of common loss functions in terms of Betti numbers's characterization, analysing the effect of skip connections and ℓ_2 regularization on the topological complexity [P05]. In Section [8.2](#) we describe derivation of the Recurrent Kernel limit of different Reservoir Computing topologies, showing that different topologies can lead to the same asymptotic limit [P06]. The content of publication [P08] is reported in Section [8.3](#), where we describe how Physics Informed Neural Networks (PINNs) can be profitably used to construct a parameterization of a planar domain by only knowing its boundary representation. Finally, we describe in Section [8.4](#) a new method for the analysis of the Visual Sequential Search Test (VSST), a neurocognitive task commonly used in clinical settings as a diagnostic tool for the evaluation of frontal functions, based on the episode matching algorithm [P07].

8.1 A topological description of loss surfaces based on Betti Numbers

In setting up the training procedure for learning models, the characterization of the loss function to be minimized is a crucial aspect, as the whole training efficiency relies on its shape, which in turn depends on the network architecture. Several works have already dealt with the analysis of the surface of the loss function, identifying conditions for the presence (or absence) of spurious valleys in a theoretical [\[104\]](#) or empirical-driven way [\[105\]](#), pointing out the role of saddle points in slowing down the learning [\[106\]](#), and giving hints on the topological structure of the loss for networks with different types of activation functions [\[107, 108\]](#).

The contribution of publication [P05] aims to give a characterization of the complexity of loss functions based on a topological argumentation. More precisely, given a layered neural network \mathcal{N} and a loss function $\mathcal{L}_{\mathcal{N}}$ computed on some training data, we will measure the complexity of $\mathcal{L}_{\mathcal{N}}$ by the topological complexity (w.r.t. the set of parameter θ) of the set $S_{\mathcal{N}} = \{\theta | \mathcal{L}_{\mathcal{N}}(\theta) \leq z\}$. Such an approach is natural, since $S_{\mathcal{N}}$, observed at each level z , provides the form of the loss function: for example, if $\mathcal{L}_{\mathcal{N}}$ has k isolated minima, then $S_{\mathcal{N}}$ has k disconnected regions for some small z .

In our investigation, we determined that when a network employs a Pfaffian non-linearity, both the Mean Square Error (MSE) and Binary Cross Entropy (BCE) loss functions can be represented as Pfaffian functions. Subsequently, we analyzed the respective



Pfaffian chains obtained in each case. In more detail, we examined the differences in the complexity and performance of the Pfaffian chains resulting from the use of the two loss functions.

When studying the complexity of the loss landscape, a super-exponential dependency on the network parameters has been found; interestingly, a qualitative difference can be highlighted between the shallow and the deep case, as we focus on the impact of the number of neurons h . Indeed, as the number of layers starts to increase, the superexponential dependency involves a term h^2 , and not h anymore. This result is aligned with the general intuition and previous works in literature [30]. In any case, the asymptotic analysis shows that the sum of Betti numbers has an exponential dependency on the square of the number of samples m . We also derived the characterization of the topological complexity for loss functions with an additional ℓ_2 regularization term; from our analysis point of view, it seems that the presence of a regularization term is not implied in the design of the loss landscape, pointing out to a different role of the regularization itself in the network training, e.g. the optimization process. Moreover, in our study, we demonstrated that incorporating skip connections (as in ResNets [109]) into the network does not affect the Betti numbers' bounds.

8.2 Extension of Recurrent Kernels to different Reservoir Computing topologies

Reservoir Computing is a machine learning technique used for training Recurrent Neural Networks, which fixes the internal weights of the network and trains only a linear layer, resulting in faster training times [110]. Its simplicity and effectiveness have made it a popular choice for various tasks [111]. Additionally, the random connections within Reservoir Computing networks make them a useful framework for comparison with biological neural networks [112].

In the last years, researchers have proposed several methods to optimize and enhance the performance and efficiency of Reservoir Computing. The availability of these diverse Reservoir Computing variants provides flexibility in selecting the most appropriate configuration for a given task.

Increasing the number of neurons in a Reservoir Computing network leads to the convergence of its behavior to a recurrent kernel, as discussed in [113]. In machine learning, kernel methods are commonly employed to train linear models on non-linear data by calculating scalar products between input points in a dual space. Recurrent Kernels are a variant in which these scalar products are dynamically updated over time based on changes in the input data. As kernel methods require the calculation of scalar products between all pairs of input points, recurrent kernels offer an interesting alternative to Reservoir Computing when the number of data points is limited. Additionally, recurrent kernels have been useful for theoretical studies, such as stability analysis in Reservoir Computing, as they provide a deterministic limit with analytical expressions [114]. Prior research on Recurrent Kernels has been mainly limited to vanilla Reservoir Computing and structured transforms. In publication [P06], we have derived the Recurrent Kernel

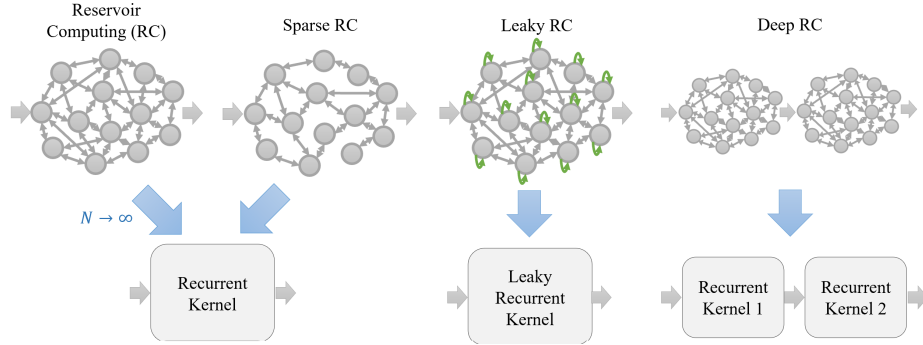


Figure 8.1: Recurrent Kernels associated with various Reservoir Computing topologies. RC and sparse RC converge to the same RK limit when the reservoir size $N \rightarrow \infty$. Leaky RC and Deep RC converge to their corresponding limits.

limit of different reservoir topologies. We have shown that different topologies can lead to the same asymptotic limit.

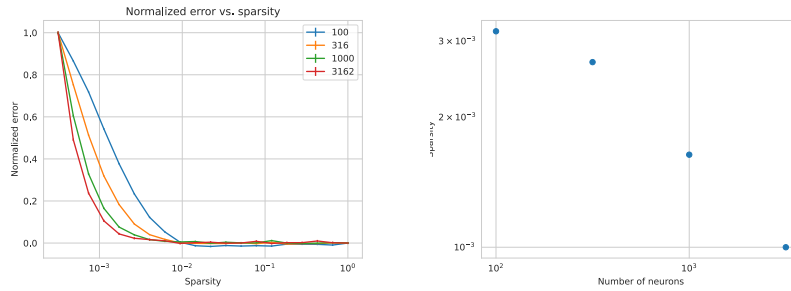


Figure 8.2: (Left) Error metric normalized between 0 and 1 as a function of sparsity for different reservoir sizes. (Right) Sparsity threshold above which the error metric is within 10% of the non-sparse limit. This gives an admissible sparsity level which decreases with the reservoir size.

More specifically, the presence of sparsity does not affect convergence at all, which justifies the sparse initialization of reservoir weights to speed up computation. Convergence has been studied numerically and validated for a wide range of parameters, especially for bounded activation functions.

Finally, we have derived how Recurrent Kernels extend to Deep Reservoir Computing, and how it sheds new insight on how to set the consecutive reservoir sizes. In a nutshell, a good rule of thumb to choose the reservoir sizes in Deep Reservoir Computing is to choose them all equal. First reservoirs can be chosen slightly (around 5%) larger than the last ones to decrease further the distance with the asymptotic limit performance.

8.3 Splines Parameterization of Planar Domains by Physics-Informed Neural Networks

The generation of structured grids on bounded domains is a crucial issue in the development of numerical models for solving differential problems. In particular, the representation of the given computational domain through a regular parameterization allows us to define a univalent mapping which can be computed as the solution of an elliptic problem, equipped with suitable Dirichlet boundary conditions.

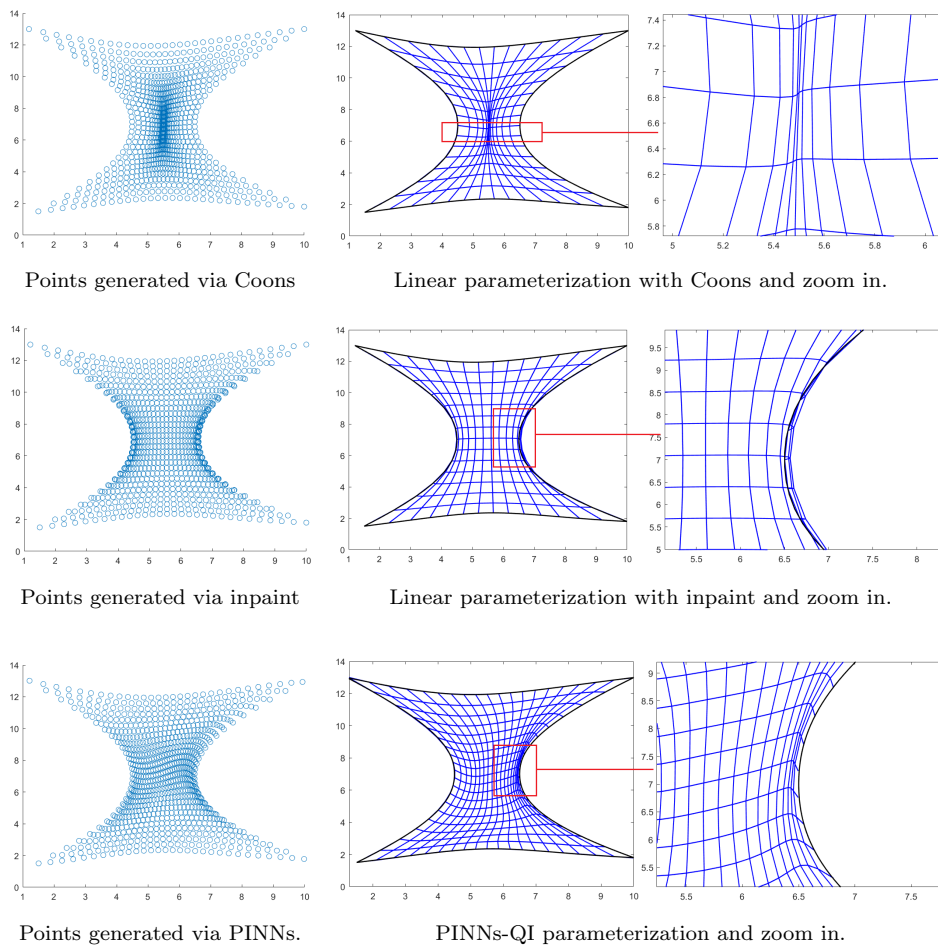


Figure 8.3: Hourglass-shaped domain.

In the last two decades machine learning and deep learning techniques have started to play an active role in the setting up of new methods for the numerical solution of PDEs, [\[115\]](#), [\[116\]](#), [\[117\]](#). In particular, Physics Informed Neural Networks (PINNs) [\[118\]](#), [\[119\]](#), [\[120\]](#) have emerged as an intuitive and efficient deep learning framework to solve PDEs, carrying

on the training of a neural network by minimizing the loss functional which incorporates the PDE itself, *informing* the neural network about the physical problem to be solved. In the paper, we exploit the PINN model in order to solve the PDE associated to the differential problem of the parameterization on both convex and non-convex planar domains, for which the describing PDE is known. The main contribution of this work consisted in introducing a novel algorithm to compute a single patch planar domain parameterizations. In particular, the discrete description of the computational domain was achieved by using PINNs; then, the final parameterization was obtained by means of *quasi-interpolation*(QI): a local approach to construct approximants to given functions or data with full approximation order. The considered QI operator provides a spline parameterization, i.e., a continuous description, of the desired smoothness. Figure [8.3](#) shows a visual comparison between classical methods used in literature and our method.

8.4 Visual Sequential Search Test Analysis: An Algorithmic Approach

The Trail Making Test (TMT) is a popular neuropsychological test, commonly used in clinical settings as a diagnostic tool for the evaluation of some frontal functions. While classical TMT requires an individual to draw lines sequentially connecting an assigned sequence of letters and/or numbers (the ROIs) with a pencil or mouse, the same task can be performed by using the eye-tracking technology and asking the subject to fixate the sequence of ROIs in the prescribed order [\[121\]](#). Eye-tracking studies have proved their efficacy in the diagnosis of many common neurological pathologies, such as Parkinson's disease, brain trauma and neglect phenomena.

The Visual Sequential Search Test (VSST) is an eye-tracking modified version of TMT which evaluates high order cognitive functions. Visual search can be quantified in terms of the analysis of the scan-path, which is a sequence of saccades and fixations. Thus, the identification of precise scores of the VSST may provide a measure of the subject's visual spatial ability and high order mental activity. The VSST is a repeated search task, in which patients are asked to connect by gaze a logical sequence of numbers and letters. In publication [\[P07\]](#) we present an algorithmic approach to the analysis of the VSST based on the episode matching method. The data set included two groups of patients, one with Parkinson's disease, and another with chronic pain syndrome, along with a control group. First, we pre-processed the data recording the fixation sequence as a series of symbols (possible repeated) representing the fixated locations. Since the observed sequences (scan-paths) have a length quite different from each other, a global alignment is not suitable to evidence their similarity (if any) [\[122\]](#). Therefore, we proposed to compare the expected scan-path with the observed scan-path using dot-plots. This provided a visual and hence a qualitative comparison between them but did not permit to evaluate it quantitatively. Then, we used the episode matching method, traditionally used in bioinformatics applications, to assign a score to a set of patients, under a specific VSST task to perform. The proposed score was validated by comparing the performance of the three different groups: the group of patients with extrapyramidal disease, the second

one of patients suffering from chronic pain syndrome and the control group. Our results, as expected, confirmed the worst performance of extrapyramidal patients than the chronic pain and control groups, in general. In particular, the medians of the three classes were significantly different from each other, so suggesting that our method can be employed as a measure of the performance in the VSST. The method we proposed is illustrated in the flowchart of Figure 8.4

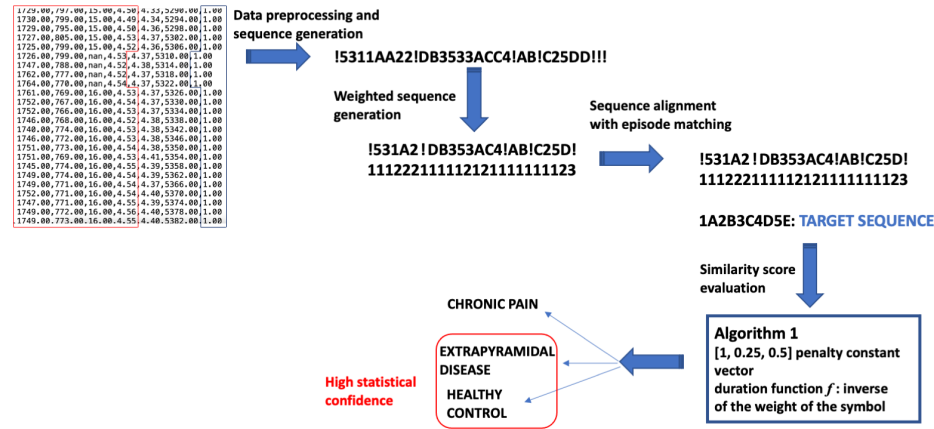


Figure 8.4: Flowchart of our method to compute the score of the performance in the Visual Sequential Search Task.

Chapter 9

Conclusions

This thesis has been dedicated to giving a comprehensive theoretical overview of modern GNNs, under the lens of two crucial aspects such as their approximation power and their generalization capabilities. In Chapter 4, GNNs have been proven to be universal approximators over the domain of undirected node-attributed graphs, modulo the unfolding equivalence. A bound on the required number of layers and a sufficient condition for universality has been provided. This result has been extended in Chapter 5 to GNN models that act over the domains of SAUHG and dynamic graphs similarly.

As future work, extending all our results for graphs in continuous-time representation would be interesting. One difficulty in this context is deciding whether two continuous-time dynamic graphs are called WL equivalent since there are many possibilities for dealing with the given timestamps. The investigation of the equivalence of dynamic graphs requires determining the handling of dynamic graphs that are equal in their structure but differ in their temporal occurrence, i.e., dependent on the commitment of the WL equivalence or the unfolding tree equivalence, it is required to decide whether the concepts need to be **time-invariant**. For time-invariant equivalence, the following concepts hold as they are. In the case in which two graphs with the same structure should be distinguished when they appear at different times, the node and edge attributes can be extended by an additional dimension carrying the exact timestamp. Thereby, the unfolding trees of two (structural) equal nodes would be different, having different timestamps in their attributes. Then, all dynamic graphs $G^{(j)} \in \mathcal{G}$ are defined over the same time interval I . This assumption can be made without loss of generality, since the set of timestamps of $G^{(j)}$ noted by $I_{G^{(j)}}$ can be padded by including missing timestamps t_q and $G^{(j)}$ can be padded by empty graphs $G_q^{(j)}$ where $V_q^{(j)} = \emptyset$, $E_q^{(j)} = \emptyset$, $\alpha_q(\emptyset) = \emptyset$, $\omega_q(\emptyset) = \emptyset$.

Furthermore, this thesis considers extensions of the usual 1-WL test and the commonly known unfolding trees. Further future work could be to investigate extensions, for example, the n -dim attributed/dynamic WL test or other versions of unfolding trees, covering GNN models not considered by the frameworks used in this thesis. These extensions might result in a more exemplary classification of the expressive power of different GNN architectures.

Moreover, the results shown in Chapters 4 and 5 mainly focus on the expressive power of GNNs. However, GNNs with the same expressive power may differ for other fundamental properties, e.g., the computational and memory complexity and the generalization capability. Understanding how the architecture of AGGREGATE⁽ⁱ⁾, COMBINE⁽ⁱ⁾, and READOUT impact those properties is of fundamental importance for practical applications of GNNs.

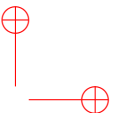
In Chapter 6 we derived new bounds for the VC dimension of modern message passing GNNs with Pfaffian activation functions, closing the gap left in the literature with

respect to the set of common used activation functions; moreover, we exhibit a preliminar experimental validation to partially show consistency between theory and practice.

Several directions of improvement can be provided: first of all, our analysis lacks of *lower bounds*, which could give a more precise intuition of the degradation of generalization capabilities for GNNs within the architectural framework here analyzed. In addition, providing a map between the VC dimension and the difference between the training and test accuracy would be much more informative; we could establish a quantitative measure with respect to the number of parameters that could allow us to better explain the experimental performance. Finally, it would be interesting to extend the analysis on the VC dimension to other GNN paradigms, such as Graph Transformers [123] and Graph Diffusion Models [124].

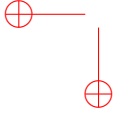
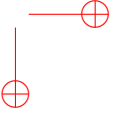
In Chapter 7 we extensively investigate the generalization capabilities of GNNs when learning identity effects through a combination of theoretical and experimental analysis. From the theoretical perspective, we established that GNNs, under mild assumptions, cannot learn identity effects when orthogonal encodings are used in a specific two-letter word classification task. On the positive side, we showed the existence of GNNs able to successfully learn identity effects on dicyclic graphs, thanks to the expressive power of the Weisfeiler–Lehman test. The experimental results strongly support these theoretical findings and provide valuable insights into the problem. In the case of two-letter words, our experiments highlight the key influence of encoding orthogonality on misclassification behavior. Our experiments on dicyclic graphs demonstrate the importance of the correct architecture selection in order to achieve generalization.

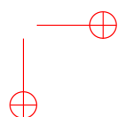
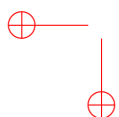
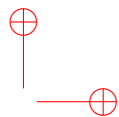
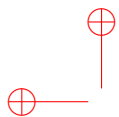
Several directions of future research naturally stem from our analysis. First, while Theorem 7.4 identifies sufficient conditions for rating impossibility, it is not known whether (any of) these conditions are also necessary. Moreover, numerical experiments on two-letter words show that generalization outside the training set is possible when using nonorthogonal encodings; justifying this phenomenon from a theoretical perspective is an open problem. On the other hand, our numerical experiments on dicyclic graphs show that achieving a good generalization depends on the choice of the architecture; this suggests that rating impossibility theorems might hold under suitable conditions on the GNN architecture in that setting. Another interesting open problem is the evaluation of the GNNs' expressive power on more complex graph domains. In particular, conducting extensive experiments on molecule analyses mentioned in Section 3.2.2, which naturally exhibit intricate structures, could provide valuable insights into modern chemistry and drug discovery applications.



Publications and Activities

Journal Articles

- P01 **G. A. D’Inverno**, M. Bianchini, M. L. Sampoli, and F. Scarselli. “On the approximation capability of GNNs in node classification/regression tasks”. *Soft Computing, Accepted*. arXiv:2106.08992 (2021).
- P02 S. Beddar-Wiesing, **G. A. D’Inverno**, C. Graziani, V. Lachi, A. Moallem-Oureh, F. Scarselli, and J. M. Thomas. “Weisfeiler–Lehman goes Dynamic: An Analysis of the Expressive Power of Graph Neural Networks for Attributed and Dynamic Graphs”. *Neural Networks*, 106213 (2024).
- P03 **G. A. D’Inverno**, S. Brugiapaglia, and M. Ravanelli. “Generalization Limits of Graph Neural Networks in Identity Effects Learning”. *Neural Networks: Graph Representation Learning special issue, submitted*. arXiv:2307.00134 (2023).
- P04 **G. A. D’Inverno**, M. Bianchini, and F. Scarselli. “VC dimension of Graph Neural Networks with Pfaffian activation functions”. *Neural Networks, submitted*. arXiv:2401.12362 (2024).
- P05 M. S. Bucarelli, **G. A. D’Inverno**, M. Bianchini, F. Scarselli and F. Silvestri. A topological description of loss surfaces based on Betti Numbers. *Neural Networks, First Revision*. arXiv:2401.03824 (2024).
- P06 **G. A. D’Inverno** and J. Dong. “Extension of Recurrent Kernels to different Reservoir Computing topologies”. *Neurocomputing, Submitted*.
- P07 **G. A. D’Inverno**, S. Brunetti, M. L. Sampoli, D. F. Muresanu, A. Rufa, and M. Bianchini. “Visual Sequential Search Test Analysis: An Algorithmic Approach”. In: *Mathematics* 9.22 (2021), 2952.
- P08 A. Falini, **G. A. D’Inverno**, M. L. Sampoli, and F. Mazzia. “Splines Parameterization of Planar Domains by Physics-Informed Neural Networks”. In: *Mathematics* 11.10 (2023), 2406.
- 
- 



Bibliography

- [1] F. Scarselli et al., “Computational Capabilities of Graph Neural Networks,” IEEE Transactions on Neural Networks, vol. 20, pp. 81–102, 2009.
- [2] F. Scarselli et al., “The Graph Neural Network Model,” IEEE Transactions on Neural Networks, vol. 20, pp. 61–80, 2009.
- [3] A. Micheli, “Neural network for graphs: A contextual constructive approach,” IEEE Transactions on Neural Networks, vol. 20, no. 3, pp. 498–511, 2009.
- [4] W. Fan, Y. Ma, Q. Li, J. Wang, G. Cai, J. Tang, and D. Yin, “A Graph Neural Network framework for social recommendations,” IEEE Transactions on Knowledge and Data Engineering, vol. 34, no. 5, pp. 2033–2047, 2020.
- [5] O. Wieder, S. Kohlbacher, M. Kuenemann, A. Garon, P. Ducrot, T. Seidel, and T. Langer, “A compact review of molecular property prediction with Graph Neural Networks,” Drug Discovery Today: Technologies, vol. 37, pp. 1–12, 2020.
- [6] P. Bongini, M. Bianchini, and F. Scarselli, “Molecular generative Graph Neural Networks for drug discovery,” Neurocomputing, vol. 450, pp. 242–252, 2021.
- [7] M. Malekzadeh, P. Hajibabae, M. Heidari, S. Zad, O. Uzuner, and J. H. Jones, “Review of Graph Neural Network in text classification,” in 2021 IEEE 12th annual ubiquitous computing, electronics & mobile communication conference (UEMCON). IEEE, 2021, pp. 0084–0091.
- [8] W. Jiang and J. Luo, “Graph Neural Network for traffic forecasting: A survey,” Expert Systems with Applications, p. 117921, 2022.
- [9] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Neural message passing for quantum chemistry,” ser. Proceedings of ICML 2017. PMLR, 2017, pp. 1263–1272.
- [10] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” arXiv preprint arXiv:1609.02907, 2016.
- [11] P. Veličković et al., “Graph attention networks,” ser. Proceedings of ICLR 2018, 2018.
- [12] W. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” ser. Advances in Neural Information Processing Systems, 2017.
- [13] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, “How Powerful are Graph Neural Networks?” ser. Proceedings of the ICLR 2018, 2018.

- [14] F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi, "Hierarchical representation learning in graph neural networks with node decimation pooling," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 5, pp. 2195–2207, 2020.
- [15] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli, "Graph matching networks for learning the similarity of graph structured objects," in *International conference on machine learning*. PMLR, 2019, pp. 3835–3845.
- [16] A. M. Karimi, Y. Wu, M. Koyuturk, and R. H. French, "Spatiotemporal graph neural network for performance prediction of photovoltaic power systems," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 17, 2021, pp. 15 323–15 330.
- [17] B. Donon, B. Donnot, I. Guyon, and A. Marot, "Graph neural solver for power systems," in *2019 international joint conference on neural networks (ijcnn)*. IEEE, 2019, pp. 1–8.
- [18] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," *Advances in neural information processing systems*, vol. 33, pp. 22 118–22 133, 2020.
- [19] X. Wang, X. He, Y. Cao, M. Liu, and T.-S. Chua, "Kgat: Knowledge graph attention network for recommendation," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 950–958.
- [20] 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 *AAAI Conference on Artificial Intelligence*, 2019, pp. 4602–4609.
- [21] 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*, vol. 34, pp. 2625–2640, 2021.
- [22] 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*, vol. 34, pp. 2625–2640, 2021.
- [23] G. Bouritsas, F. Frasca, S. Zafeiriou, and M. M. Bronstein, "Improving graph neural network expressivity via subgraph isomorphism counting," *arXiv preprint arXiv:2006.09252*, 2020.
- [24] J. You, J. Gomes-Selman, R. Ying, and J. Leskovec, "Identity-Aware Graph Neural Networks," ser. *Proceedings of the Conference on Artificial Intelligence (AAAI 21)*, 2021.
- [25] A. Loukas, "What graph neural networks cannot learn: depth vs width," *arXiv preprint arXiv:1907.03199*, 2019.
- [26] W. Azizian and M. Lelarge, "Expressive power of invariant and equivariant graph neural networks," *arXiv preprint arXiv:2006.15646*, 2020.
- [27] B. Knyazev, G. W. Taylor, and M. Amer, "Understanding attention and generalization in graph neural networks," *Advances in neural information processing systems*, vol. 32, 2019.
- [28] F. Scarselli, A. C. Tsoi, and M. Hagenbuchner, "The Vapnik–Chervonenkis dimension of graph and recursive neural networks," *Neural Networks*, vol. 108, pp. 248–259, 2018.
- [29] X. Zhou and H. Wang, "The generalization error of graph convolutional networks may enlarge with more layers," *Neurocomputing*, vol. 424, pp. 97–106, 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0925231220317367>
- [30] M. Bianchini and F. Scarselli, "On the complexity of neural network classifiers: A comparison between shallow and deep architectures," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 8, pp. 1553–1565, 2014.

- [31] S. Jegelka, “Theory of Graph Neural Networks: Representation and Learning,” arXiv preprint arXiv:2204.07697, 2022.
- [32] M. Grohe and P. Schweitzer, “The Graph Isomorphism Problem,” Communications of the ACM, vol. 63, no. 11, pp. 128–134, 2020.
- [33] A. A. Lehman and B. Weisfeiler, “A reduction of a graph to a canonical form and an algebra arising during this reduction,” Nauchno-Technicheskaya Informatsiya, vol. 2, no. 9, pp. 12–16, 1968.
- [34] R. Sato, “A survey on the expressive power of graph neural networks,” arXiv preprint arXiv:2003.04078, 2020.
- [35] S. Kiefer, “Power and limits of the weisfeiler–lehman algorithm,” Ph.D. dissertation, Dissertation, RWTH Aachen University, 2020.
- [36] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” ser. Proceedings of ICLR 2017, 2017.
- [37] P. Battaglia et al., “Relational inductive biases, deep learning, and graph networks,” arXiv preprint arXiv:1806.01261, 2018.
- [38] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” Neural networks, vol. 2, no. 5, pp. 359–366, 1989.
- [39] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” Mathematics of control, signals and systems, vol. 2, no. 4, pp. 303–314, 1989.
- [40] A. R. Barron, “Universal approximation bounds for superpositions of a sigmoidal function,” IEEE Transactions on Information theory, vol. 39, no. 3, pp. 930–945, 1993.
- [41] F. Scarselli and A. Chung Tsoi, “Universal Approximation Using Feedforward Neural Networks: A Survey of Some Existing Methods, and Some New Results,” Neural Networks, vol. 11, no. 1, pp. 15–37, 1998.
- [42] J. Berner, P. Grohs, G. Kutyniok, and P. Petersen, “The modern mathematics of deep learning,” Mathematical Aspects of Deep Learning, p. 1, 2022.
- [43] V. Garg, S. Jegelka, and T. Jaakkola, “Generalization and representational limits of graph neural networks,” ser. Proceedings of ICML 2020. PMLR, 2020, pp. 3419–3430.
- [44] H. Maron, H. Ben-Hamu, N. Shamir, and Y. Lipman, “Invariant and equivariant graph networks,” arXiv preprint arXiv:1812.09902, 2018.
- [45] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman, “Provably powerful graph networks,” Advances in neural information processing systems, vol. 32, 2019.
- [46] A. Krebs and O. Verbitsky, “Universal covers, color refinement, and two-variable counting logic: Lower bounds for the depth,” ser. Proceedings of the 30th Annual ACM/IEEE Symposium on Logic in Computer Science. IEEE, 2015, pp. 689–700.
- [47] V. Vapnik, E. Levin, and Y. Le Cun, “Measuring the VC-dimension of a learning machine,” Neural Computation, vol. 6, no. 5, pp. 851–876, 1994.
- [48] P. L. Bartlett and S. Mendelson, “Rademacher and Gaussian complexities: Risk bounds and structural results,” Journal of Machine Learning Research, vol. 3, no. Nov, pp. 463–482, 2002.
- [49] N. Golowich, A. Rakhlin, and O. Shamir, “Size-independent sample complexity of neural networks,” in Conference on Learning Theory. Proceedings of Machine Learning Research, 2018, pp. 297–299.

- [50] G. A. D’Inverno, M. Bianchini, M. L. Sampoli, and F. Scarselli, “On the approximation capability of GNNs in node classification/regression tasks,” arXiv preprint arXiv:2106.08992, 2021.
- [51] P. Esser, L. Chennuru Vankadara, and D. Ghoshdastidar, “Learning theory can (sometimes) explain generalisation in graph neural networks,” Advances in Neural Information Processing Systems, vol. 34, pp. 27 043–27 056, 2021.
- [52] S. Verma and Z.-L. Zhang, “Stability and generalization of graph convolutional neural networks,” in Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 1539–1548.
- [53] R. Liao, R. Urtasun, and R. Zemel, “A pac-bayesian approach to generalization bounds for Graph Neural Networks,” arXiv preprint arXiv:2012.07690, 2020.
- [54] H. Ju, D. Li, A. Sharma, and H. R. Zhang, “Generalization in Graph Neural Networks: Improved PAC-Bayesian bounds on graph diffusion,” in International Conference on Artificial Intelligence and Statistics. PMLR, 2023, pp. 6314–6341.
- [55] C. Morris, F. Geerts, J. Tönshoff, and M. Grohe, “WL meet VC,” arXiv preprint arXiv:2301.11039, 2023.
- [56] G. F. Marcus, S. Vijayan, S. Bandi Rao, and P. M. Vishton, “Rule learning by seven-month-old infants,” Science, vol. 283, no. 5398, pp. 77–80, 1999.
- [57] G. F. Marcus, The Algebraic Mind: Integrating Connectionism and Cognitive Science. MIT Press, 2003.
- [58] L. E. Suárez, B. A. Richards, G. Lajoie, and B. Misisic, “Learning function from structure in neuromorphic networks,” Nature Machine Intelligence, vol. 3, no. 9, pp. 771–786, 2021.
- [59] L. Benua, “Identity effects in morphological truncation,” in Papers in optimality theory, J. Beckman, S. Urbanczyk, and L. Walsh, Eds. GLSA (Graduate Linguistic Student Association), Dept. of Linguistics, University of Massachusetts, Amherst, MA, 1995, pp. 77–136.
- [60] G. Gallagher, “Learning the identity effect as an artificial language: bias and generalisation,” Phonology, vol. 30, no. 2, pp. 253–295, 2013.
- [61] L. Paschen, “Trigger poverty and reduplicative identity in lakota,” Natural Language & Linguistic Theory, pp. 1–37, 2021.
- [62] J. Ghomeshi, R. Jackendoff, N. Rosen, and K. Russell, “Contrastive focus reduplication in English (the salad-salad paper),” Natural Language & Linguistic Theory, vol. 22, pp. 307–357, 2004.
- [63] Q. Wu and Q. Wang, “Natural language processing based detection of duplicate defect patterns,” in 2010 IEEE 34th Annual Computer Software and Applications Conference Workshops. IEEE, 2010, pp. 220–225.
- [64] J. F. Liebman and A. Greenberg, “A survey of strained organic molecules,” Chemical Reviews, vol. 76, no. 3, pp. 311–365, 1976.
- [65] P. R. Bunker and P. Jensen, Molecular Symmetry and Spectroscopy. NRC research press, 2006, vol. 46853.
- [66] C. Pettinari and C. Santini, “IR and Raman Spectroscopies of Inorganic, Coordination and Organometallic Compounds,” in Encyclopedia of Spectroscopy and Spectrometry, 3rd ed., J. Lindon, G. Tranter, and D. W. Koppenaal, Eds. Oxford: Academic Press, 2017, pp. 347–358.

- [67] S. Brugiapaglia, M. Liu, and P. Tupper, “Invariance, encodings, and generalization: learning identity effects with neural networks,” Neural Computation, vol. 34, no. 8, pp. 1756–1789, 2022.
- [68] S. Kiefer and B. D. McKay, “The Iteration Number of Colour Refinement,” ser. Proceedings of the 47th International Colloquium on Automata, Languages, and Programming (ICALP 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
- [69] D. Angluin, “Local and global properties in networks of processors (extended abstract),” ser. Proceedings of the 12th Annual ACM Symposium on Theory of Computing. New York, NY, USA: Association for Computing Machinery, 1980, p. 82–93.
- [70] H. Dell, M. Grohe, and G. Rattan, “Lovász meets Weisfeiler and Leman,” arXiv preprint arXiv:1802.08876, 2018.
- [71] R. Sato, M. Yamada, and H. Kashima, “Random Features Strengthen Graph Neural Networks,” ser. Proceedings of SDM21, 2021.
- [72] A. Sperduti and A. Starita, “Supervised neural networks for the classification of structures,” IEEE Transactions on Neural Networks, vol. 8, no. 3, pp. 714–735, 1997.
- [73] M. Bianchini and M. Gori, “Theoretical properties of recursive neural networks with linear neurons,” IEEE Transactions on Neural Networks, vol. 12, no. 5, pp. 953–967, 2001.
- [74] L. Ruddigkeit, R. Van Deursen, L. C. Blum, and J.-L. Reymond, “Enumeration of 166 billion organic small molecules in the chemical universe database gdb-17,” Journal of chemical information and modeling, vol. 52, no. 11, pp. 2864–2875, 2012.
- [75] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. Von Lilienfeld, “Quantum chemistry structures and properties of 134 kilo molecules,” Scientific data, vol. 1, no. 1, pp. 1–7, 2014.
- [76] S. M. Kazemi, R. Goel, K. Jain, I. Kobyzev, A. Sethi, P. Forsyth, and P. Poupart, “Representation Learning for Dynamic Graphs: A Survey,” J. Mach. Learn. Res., vol. 21, pp. 70:1–70:73, 2020. [Online]. Available: <http://jmlr.org/papers/v21/19-447.html>
- [77] J. M. Thomas, S. Beddar-Wiesing, and A. Moallem-Oureh, “Graph Type Expressivity and Transformations,” arXiv:2109.10708, 09 2021.
- [78] J. Skardinga, B. Gabrys, and K. Musial, “Foundations and Modelling of Dynamic Networks Using Dynamic Graph Neural Networks: A Survey,” IEEE Access, 2021.
- [79] M. M. Bronstein, J. Bruna, T. Cohen, and P. Velickovic, “Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges,” CoRR, vol. abs/2104.13478, 2021. [Online]. Available: <https://arxiv.org/abs/2104.13478>
- [80] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, “Structured sequence modeling with graph convolutional recurrent networks,” in International conference on neural information processing. Springer, 2018, pp. 362–373.
- [81] A. Narayan and P. H. Roe, “Learning graph dynamics using deep neural networks,” IFAC-PapersOnLine, vol. 51, no. 2, pp. 433–438, 2018.
- [82] M. Niepert, M. Ahmed, and K. Kutzkov, “Learning convolutional neural networks for graphs,” in International conference on machine learning. PMLR, 2016, pp. 2014–2023.
- [83] A. Taheri, K. Gimpel, and T. Berger-Wolf, “Learning to represent the evolution of dynamic graphs with recurrent models,” in Companion proceedings of the 2019 world wide web conference, 2019, pp. 301–307.

- [84] B. Hammer, "On the approximation capability of recurrent neural networks," Neurocomputing, vol. 31, no. 1-4, pp. 107–123, 2000.
- [85] V. Vapnik and A. Y. Chervonenkis, "On the uniform convergence of relative frequencies of events to their probabilities," Theory of Probability & Its Applications, vol. 16, no. 2, pp. 264–280, 1971.
- [86] A. G. Khovanski, Fewnomials. American Mathematical Soc., 1991, vol. 88.
- [87] M. Karpinski and A. Macintyre, "Polynomial bounds for VC dimension of sigmoidal and general pfaffian neural networks," J. Comput. Syst. Sci., vol. 54, no. 1, pp. 169–176, 1997. [Online]. Available: <https://doi.org/10.1006/jcss.1997.1477>
- [88] A. Gabrielov and N. Vorobjov, "Complexity of computations with pfaffian and noetherian functions," Normal forms, bifurcations and finiteness problems in differential equations, vol. 137, pp. 211–250, 2004.
- [89] A. Gabrielov and N. Vorobjov, "Complexity of stratification of semi-pfaffian sets," Discrete & computational geometry, vol. 14, no. 1, pp. 71–91, 1995.
- [90] B. Hammer, "On the generalization ability of recurrent networks," in Artificial Neural Networks—ICANN 2001: International Conference Vienna, Austria, August 21–25, 2001 Proceedings 11. Springer, 2001, pp. 731–736.
- [91] P. Koiran and E. D. Sontag, "Neural networks with quadratic vc dimension," journal of computer and system sciences, vol. 54, no. 1, pp. 190–198, 1997.
- [92] E. Baum and D. Haussler, "What size net gives valid generalization?" Advances in neural information processing systems, vol. 1, 1988.
- [93] W. Maass, "Neural nets with superlinear vc-dimension," Neural Computation, vol. 6, no. 5, pp. 877–884, 1994.
- [94] A. Sakurai, "On the vc-dimension of depth four threshold circuits and the complexity of boolean-valued functions," Theoretical computer science, vol. 137, no. 1, pp. 109–127, 1995.
- [95] P. Goldberg and M. Jerrum, "Bounding the vapnik-chervonenkis dimension of concept classes parameterized by real numbers," in Proceedings of the sixth annual conference on Computational learning theory, 1993, pp. 361–369.
- [96] 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 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020), 2020. [Online]. Available: www.graphlearning.io
- [97] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.
- [98] R. Vershynin, High-dimensional probability: An introduction with applications in data science. Cambridge University Press, 2018, vol. 47.
- [99] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in Proceedings of the 13th International Conference on Artificial Intelligence and Statistics. JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.
- [100] F. Mezzadri, "How to generate random matrices from the classical compact groups," Notices of the American Mathematical Society, vol. 54, no. 5, pp. 592–604, 2007.
- [101] K. Oono and T. Suzuki, "Graph neural networks exponentially lose expressive power for node classification," arXiv preprint arXiv:1905.10947, 2019.

- [102] C. Cai and Y. Wang, “A note on over-smoothing for graph neural networks,” arXiv preprint arXiv:2006.13318, 2020.
- [103] S. J. Reddi, S. Kale, and S. Kumar, “On the convergence of Adam and beyond,” arXiv preprint arXiv:1904.09237, 2019.
- [104] L. Venturi, A. S. Bandeira, and J. Bruna, “Spurious valleys in two-layer neural network optimization landscapes,” arXiv preprint arXiv:1802.06384, 2018.
- [105] I. Safran and O. Shamir, “Spurious local minima are common in two-layer relu neural networks,” in International Conference on Machine Learning. PMLR, 2018, pp. 4433–4441.
- [106] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization,” Advances in neural information processing systems, vol. 27, 2014.
- [107] C. D. Freeman and J. Bruna, “Topology and geometry of half-rectified network optimization,” arXiv preprint arXiv:1611.01540, 2016.
- [108] Q. Nguyen and M. Hein, “The loss surface of deep and wide neural networks,” in International conference on machine learning. PMLR, 2017, pp. 2603–2612.
- [109] 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.
- [110] H. Jaeger, “The “echo state” approach to analysing and training recurrent neural networks—with an erratum note,” Bonn, Germany: German National Research Center for Information Technology GMD Technical Report, vol. 148, no. 34, p. 13, 2001.
- [111] M. Lukoševičius and H. Jaeger, “Reservoir computing approaches to recurrent neural network training,” Computer Science Review, vol. 3, no. 3, pp. 127–149, 2009.
- [112] F. Damicelli, C. C. Hilgetag, and A. Goulas, “Brain connectivity meets reservoir computing,” PLoS Computational Biology, vol. 18, no. 11, p. e1010639, 2022.
- [113] J. Dong, R. Ohana, M. Rafayelyan, and F. Krzakala, “Reservoir computing meets recurrent kernels and structured transforms,” Advances in Neural Information Processing Systems, vol. 33, pp. 16 785–16 796, 2020.
- [114] J. Dong, E. Börve, M. Rafayelyan, and M. Unser, “Asymptotic stability in reservoir computing,” in 2022 International Joint Conference on Neural Networks (IJCNN). IEEE, 2022, pp. 01–08.
- [115] W. Cai, X. Li, and L. Liu, “A phase shift deep neural network for high frequency approximation and wave problems,” SIAM Journal on Scientific Computing, vol. 42, no. 5, pp. A3285–A3312, 2020.
- [116] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, “Physics-informed machine learning,” Nature Reviews Physics, vol. 3, no. 6, pp. 422–440, 2021.
- [117] M. Burger, L. Ruthotto, S. Osher et al., “Connections between deep learning and partial differential equations,” European Journal of Applied Mathematics, vol. 32, no. 3, pp. 395–396, 2021.
- [118] M. Raissi, P. Perdikaris, and G. E. Karniadakis, “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations,” Journal of Computational physics, vol. 378, pp. 686–707, 2019.

- [119] J. Yu, L. Lu, X. Meng, and G. E. Karniadakis, “Gradient-enhanced physics-informed neural networks for forward and inverse pde problems,” Computer Methods in Applied Mechanics and Engineering, vol. 393, p. 114823, 2022.
- [120] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli, “Scientific machine learning through physics-informed neural networks: where we are and what’s next,” Journal of Scientific Computing, vol. 92, no. 3, p. 88, 2022.
- [121] G. Veneri, E. Pretegianni, F. Rosini, P. Federighi, A. Federico, and A. Rufa, “Evaluating the human ongoing visual search performance by eye tracking application and sequencing tests,” Computer methods and programs in biomedicine, vol. 107, no. 3, pp. 468–477, 2012.
- [122] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison, Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids. Cambridge University Press, 1998.
- [123] S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim, “Graph transformer networks,” Advances in neural information processing systems, vol. 32, 2019.
- [124] M. Zhang, M. Qamar, T. Kang, Y. Jung, C. Zhang, S.-H. Bae, and C. Zhang, “A survey on graph diffusion models: Generative ai in science for molecule, protein and material,” arXiv preprint arXiv:2304.01565, 2023.