Graph Neural Networks

Operators and Architectures

Doctoral Dissertation submitted to the
Faculty of Informatics of the Università della Svizzera Italiana
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

presented by
Daniele Grattarola

under the supervision of
Prof. Cesare Alippi and Prof. Lorenzo Livi

December 2021

Dissertation Committee

Prof. Fabio Crestani	Università della Svizzera italiana, Switzerland
Prof. Luca Gambardella	Università della Svizzera italiana, Switzerland
Prof. Plamen Angelov	Lancaster University, United Kingdom
Prof. Christos Panayiotou	University of Cyprus, Cyprus
Prof. Alessandro Sperduti	Università di Padova, Italy

Dissertation accepted on 10 December 2021

Research Advisor

Prof. Cesare Alippi

Co-Advisor

Prof. Lorenzo Livi

PhD Program Director

The PhD program Director *pro tempore*

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Daniele Grattarola
Lugano, 10 December 2021

Cells interlinked within cells interlinked
within one stem.

Abstract

This thesis explores the field of graph neural networks, a class of deep learning models designed to learn representations of graphs.

We organise the work into two parts. In the first part, we focus on the essential building blocks of graph neural networks. We present three novel operators for learning graph representations: one graph convolutional layer and two methods for pooling. We put particular emphasis on the topic of pooling, introducing a universal and modular framework to describe pooling operators, a taxonomy to organise the literature, and a set of evaluation criteria to assess an operator's performance.

The second part focuses on specific graph neural network architectures and their applications to cutting-edge problems in dynamical systems and computational biology. We present three main contributions. First, we introduce an autoencoder architecture for learning graph representations in non-Euclidean spaces. We apply our model to the tasks of molecule generation and change detection in graph sequences. Second, we propose a graph neural network designed to be interpretable, specifically to solve the problem of seizure localisation in subjects with epilepsy. Finally, we discuss the design of autoregressive models for sequences of graphs.

Acknowledgements

My deepest gratitude goes to my advisor Cesare Alippi, who supported me throughout these four years and who has taught me many valuable lessons as a scientist and, more importantly, as a person. I also sincerely thank my co-advisor Lorenzo Livi for his invaluable guidance and help. I could not have found better mentors, and I am grateful for having been their student.

In my time at USI, I had the luck and the privilege of working with some of the most talented and brilliant people I know. I would like to thank my friends and colleagues Daniele Zambon, Pietro Verzelli, Andrea Cini, Alberto Gasparin, Elena Di Lascio, Lorenzo Ferretti, Ivan Marsica, Matteo Riva, and Arianna Blasi. I have learned a lot from them, and I owe them much of what I have accomplished.

A special thanks to Filippo Bianchi, with whom I have had an exceptionally productive collaboration over the years. He is a precious guide and friend, and I am grateful to him for the many days spent working together on our papers.

I want to thank, with all my heart, Dr. Taufik Valiante for welcoming me into the *Neuron To Brain* laboratory. I have found an important mentor in him, and I am thankful for the many exceptional conversations we had. During my time in Toronto, I have learned what it means to relentlessly dedicate oneself to science. I am honoured to have worked alongside Homeira Moradi, Anett Schumacher, Azadeh Naderian, Sara Mahallati, Victoria Barkley, David Groppe, Scott Rich, Gerard O'Leary, Kramay Patel, and all the people of N2B.

A sincere thank you also to the amazing people that I have met at USI, including Mariano Panta, Ioannis Mantas, Prof. Limongelli and his group, Vincent Herrmann, and Ivan Sekulic. For their precious support and patience, I thank the irreplaceable team of the Dean's office, Elisa Larghi, Jacinta Vigini, Nadia Ruggiero, Sabina Brambilla, and Nina Caggiano.

To all the people I have met during these years, too many to name them all: thank you for being part of my journey.

To Gaia and my family: I owe you everything.

Contents

Figures

Tables

Nomenclature

$\mathcal{G}$	Graph
$\mathcal{V}$	Node set
$\mathcal{E}$	Edge set
N	Number of nodes (cardinality of the node set)
(i, j)	Directed edge from node i to node j
$\mathbf{A}$	Adjacency matrix
$\mathbf{D}$	Degree matrix
$\mathbf{L}$	Laplacian
$\mathbf{L}_n$	Normalised Laplacian
$\mathcal{N}(i)$	Neighbourhood of node i
$\mathbf{X}$	Node attributes matrix
$\mathbf{x}_i$	Attribute of node i
D_n	Size of the node attributes
$\mathbf{E}$	Edge attributes tensor
$\mathbf{e}_{ij}$	Attribute of edge from node i to node j
D_e	Size of the edge attributes
$\mathbf{u}_k, \mathbf{v}_k$	Eigenvectors of a matrix
$\mathbf{U}$	Matrix with eigenvectors as columns
$\mathbf{\Lambda}$	Diagonal matrix of eigenvalues
MLP	Multi-layer perceptron
$\mathbf{A} \cdot \mathbf{B}$	Matrix multiplication
$\mathbf{a} \cdot \mathbf{b}$	Dot product

Chapter 1

Introduction

A fundamental trait of human intelligence is the ability to reason about complex systems in terms of their atomic components and the relations that exist among them. Knowledge about these relations can be exploited to perform cognitive tasks, like causal inference or hierarchical abstraction, and this way of reasoning is the cornerstone that allows for combinatorial generalisation—the ability to construct models of the world from a finite set of building blocks—to emerge. This idea is so essential that, throughout the fields of science, many social, biological, economic, and technological phenomena are described as systems of interacting entities, usually by representing them as graphs. It is therefore an interesting endeavour to design artificial intelligence (AI) systems that can naturally deal with a relational view of the world, *i.e.*, models with a *relational inductive bias* [14].

The idea of interpreting the world as a system with inherent structure, however, is in stark contrast to the current mainstream trends in AI, which are primarily centred on the paradigm of deep learning [115, 185] and of making little to no *a priori* assumptions about the data (the "hand-engineering" *vs.* "automatic feature extraction" dichotomy). Indeed, the enormous success and impact of deep learning across all areas of science—fuelled by an ever-increasing availability of data and computing resources—is an important signal that we cannot ignore in the development of general AI. However, the debate on the subject is lively and ongoing, and several limitations of deep learning-based systems have emerged, especially concerning their ability for combinatorial generalisation [138, 139].

The field of graph machine learning (GML) arises as a hybrid alternative to these two paradigms of AI, by combining the ability to learn *end-to-end* with an explicit assumption of structure in the data. The main focus of the field, and the topic of this thesis, is on *graph neural networks* (GNNs), a class of models de-

signed to learn representations of graph-structured data through the composition of differentiable functions, backpropagation, and gradient descent. In particular, GNN architectures are built from layers that extend and generalise the two core operations of convolutional neural networks, namely convolution and pooling, to deal with arbitrary graphs. By doing so, the structure underlying the data becomes an input of the model along with the data itself.

The research on GNNs dates back to the seminal works of Sperduti and Starita [198] and Gori et al. [72] between the late 1990s and early 2000s, although the field has recently undergone rapid growth with GNNs achieving state-of-the-art results on a broad spectrum of applications including physics [105, 41, 193], recommendation systems [226], natural language processing [225], chemistry and biology [54, 58, 60, 67, 62, 96], knowledge graphs [184, 10, 101], computer vision [147, 195, 167, 168], combinatorial optimisation and reasoning [33], reinforcement learning [242], computer networks [175], or even searching for anti-cancer foods [212], among many others (for more applications of GNNs, we address the reader to one of many surveys on the topic [230]).

However, despite this rapid growth and the remarkable results achieved by GNNs, the field of GML is still in its infancy and further research is needed to fully express the potential of such a groundbreaking class of AI techniques. In particular, many of the operators typically used to build GNNs are inadequate for solving more advanced tasks [156], due to their limited expressivity in representing graphs or convenient-but-limiting design choices. For example, we will see that the most common class of convolutional operators for GNNs, which are based on polynomial filters, can be significantly outperformed by more sophisticated kinds of layers (of which we propose one in Chapter 3). Also, some of the key components of GNNs—chiefly pooling operators—remain poorly understood and scattered throughout a chaotic literature on the subject. Here too, we will show that some of the pooling operators typically used in GNN architectures have significant design flaws which must be overcome (and we will propose two different approaches to do it in Chapter 4).

Additionally, while typical machine learning architectures are effective at tackling many application scenarios characterised by a regular lattice structure, their extension to the case of graphs with arbitrary topology remains an open area of research. As we argue in the next section, moving beyond the "lattice assumption" and towards a more general setting requires us to rethink the typical architectures of deep learning under this new light, while also unlocking new opportunities to solve interesting problems in the domain of graphs.

1.1 Contributions

In light of the context and limitations outlined above, this thesis collects the author's contributions to the field of GML along two main directions.

For the first direction, we focus on the design and analysis of new *operators* that GNNs use to transform the nodes and edges of a graph. For the second direction, we focus on the design of novel *architectures*, *i.e.*, particular arrangements of operators to solve a given task like classification, forecasting, or generation in the domain of graphs. Figure 1.1 shows a high-level schematic of the contributions of the thesis.

Operators Unlike conventional deep learning, the GNN literature is characterised by a rich variety of operators for convolution and pooling. This is due to the significantly more complex nature of graphs w.r.t. the typical lattice structure assumed by neural networks, which can be approached from different perspectives ranging from geometrical [27] to more computational methods [14, 65].

However, due to this complexity and variety in the field, a consensus on the general best practices and operators for GNNs has yet to be reached. In this regard, one objective of this research is to improve over the current state of the art by designing new general-purpose operators that can produce better representations of graphs. We also aim to study the existing building blocks of GNNs to understand their behaviour in practical settings, so that we can provide principled guidelines on how to best use them.

In pursuit of this research objective, we present novel convolutional and pooling operators for GNNs, which achieve state-of-the-art performance on many different classes of GML tasks. For convolution, we introduce a novel layer based on rational autoregressive moving-average (ARMA) graph filters and we show that it achieves better performance than the polynomial filters commonly used in GNNs [22]. For pooling, we present two techniques inspired by the complementary problems of finding the minimum and maximum cut on a graph, respectively: MinCut [20] and Node Decimation Pooling (NDP) [21]. MinCut is a pooling operator that learns how to reduce graphs by optimising, at the same time, the task loss of the problem at hand and an objective inspired by the minimum cut problem. NDP, on the other hand, is a node subsampling technique that uses as a sampling indicator the highest-frequency eigenvector of the graph Laplacian, in order to find a subsampling that is as regular as possible for a given graph.

We place particular emphasis on the topic of graph pooling because, as we will show, it is a subject that is often discussed in the literature (we identify

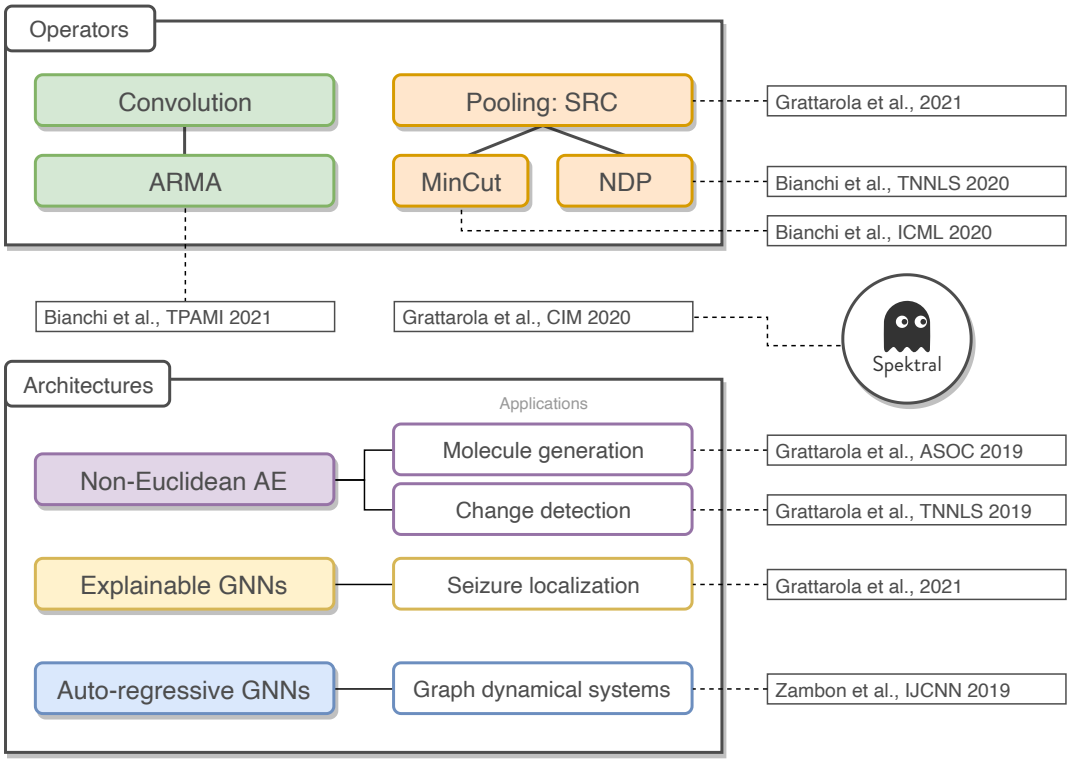


Figure 1.1. Schematic view of the contributions of the thesis. **Top**: operators. In green: the convolutional ARMA layer [22]. In orange: our unified framework for pooling operators (SRC) [79], and our two pooling operators, MinCut [20] and Node Decimation Pooling (NDP) [21]. **Bottom**: architectures. In purple: the constant-curvature manifold adversarial autoencoder (CCM-AAE) and its applications to molecule generation and change detection [75, 76]. In yellow: our explainable GNN based on the attention mechanism, designed to localise epileptic seizures in functional brain networks [77]. In blue: autoregressive GNNs to predict graph dynamical systems [247]. **Right**: Spektral, the GNN library developed within this research [74]

tens of different operators) while at the same time being poorly understood. In relation to this, one key contribution of this thesis is to analyse in-depth the literature on graph pooling and to propose a first unifying and modular framework to describe pooling operators, as well as proposing a taxonomy of pooling and a strategy for performance evaluation that goes beyond the typical benchmark-based approaches found in recent literature [79].

Architectures Since GML is a relatively recent topic in the AI landscape (or, at least, a topic that has started to draw significant attention only recently), the best design practices to solve real-world problems with GNNs are still a matter of ongoing research. An obvious starting point to approach this endeavour is to look at the neural network architectures typically used in deep learning literature and to adapt them to the more general case of graphs. While this may appear trivial at first, the assumption of lattice structure is so pervasive in deep learning (and, one could argue, generally in science) that removing it leads to many practical complications. Besides these issues, the domain of graphs enables new applications that are novel of their own, and require dedicated attention.

In this research, we have focused on three general-purpose GNN architectures whose extension to the more general case of graphs is an open research direction. In all our contributions, we develop novel architectures and apply them to specific application scenarios.

Non-Euclidean autoencoders We introduce an adversarial autoencoder for graphs that allows us to control the distribution of the embeddings and, crucially, the geometry of the embedding space [75]. In particular, we show that non-Euclidean manifolds with constant curvature are useful to represent graphs in many GML tasks. We apply this methodology to two key areas: 1) the generation of small molecules for *de novo* drug design [75], and 2) the detection of changes in temporal graph sequences, with a focus on computer vision and the detection of epileptic seizures [76].

Explainable GNNs We develop a methodology, based on the attention mechanism [210], which allows us to explain the predictions of a GNN [77]. Specifically, we focus on the task of localising the seizure onset zone in subjects with epilepsy, using functional networks to describe brain states. We use our methodology to identify the subject-specific areas of the brain involved in seizure generation. We show that our method strongly correlates with the analysis of profes-

sional electroencephalographers, with promising results for real-world deployment in epilepsy monitoring units.

Autoregressive GNNs We discuss the design of autoregressive (AR) models for sequences of graphs [247]. We begin by formalising the problem of autoregression in graph space. Then, we propose an AR GNN architecture for the prediction, which uses a GNN to embed the graph sequence in a vector space, a recurrent neural network to predict the next embedding, and a graph decoder to obtain the predicted graph.

1.2 Thesis outline

This thesis is structured into two parts, one for each principal direction of research.

In Part I, *Operators*, we discuss our contributions to the field of GML in terms of convolutional and pooling operators.

Chapter 2 introduces the notation and key concepts of graph theory that we use in the thesis, as well as presenting the reader with the relevant background on GNNs that will be required to understand all other chapters.

In Chapter 3, we present a novel graph convolutional operator based on ARMA graph filters.

In Chapter 4, we present a unifying framework and taxonomy of graph pooling, two novel pooling methods called MinCut and Node Decimation Pooling, and an extensive evaluation of several pooling methods from recent literature.

In Part II, *Architectures*, we discuss three main kinds of architectures and several related cutting-edge applications in dynamical systems and computational biology.

In Chapter 5, we present the constant-curvature manifold adversarial autoencoder (CCM-AAE) and apply it to the tasks of molecule generation for *de novo* drug design and change detection in sequences of graphs.

Chapter 6 introduces an explainable GNN based on the attention mechanism [210] and presents a case study on the topic of epileptic seizure localisation.

In Chapter 7, we discuss the topic of AR models in the domain of graphs, giving a formal characterisation of the problem and introducing a GNN for AR prediction.

Finally, Chapter 8 concludes the thesis and discusses future research directions.

The thesis is accompanied by an extensive appendix that reports all hyperparameters, a description of the datasets, the software and hardware used for the experiments, additional experiments and discussions, and proofs.

Complementary to our research, we have developed an open-source software library, called Spektral, for implementing GNNs in the TensorFlow ecosystem. Spektral has served as a platform for making the contributions presented in this thesis and, in general, the advances of recent GNN literature available to the larger scientific community. We present the library in Appendix G.

1.3 Publications

The work presented in this thesis is based on the following papers (listed in chronological order of publication):

- Daniele Grattarola, Daniele Zambon, Cesare Alippi, and Lorenzo Livi. Change detection in graph streams by learning graph embeddings on constant-curvature manifolds. *IEEE Transactions on Neural Networks and Learning Systems*, 2019 [76];

- Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Adversarial autoencoders with constant-curvature latent manifolds. *Applied Soft Computing*, 81:105511, 2019 [75];

- Daniele Zambon, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Autoregressive models for sequences of graphs. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019 [247];

- Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral clustering with graph neural networks for graph pooling. In *International Conference on Machine Learning (ICML)*, 2020 [20];

- Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Hierarchical representation learning in graph neural networks with node decimation pooling. *IEEE Transactions on Neural Networks and Learning Systems*, 2020 [21];

- Daniele Grattarola, Lorenzo Livi, Cesare Alippi, Richard Wennberg, and Taufik Valiante. Unsupervised seizure localisation with attention-based graph neural networks. *bioRxiv*, 2020 [77];

- Daniele Grattarola and Cesare Alippi. Graph neural networks in tensorflow and keras with spektral. *IEEE Computational Intelligence Magazine*, 2021 [74];
- Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Graph neural networks with convolutional arma filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021 [22];
- Daniele Grattarola, Daniele Zambon, Filippo Maria Bianchi, and Cesare Alippi. Understanding pooling in graph neural networks. *arXiv preprint arXiv:2110.05292*, 2021 [79].

The author has also contributed to the following publications:

- Benjamin Paassen, Daniele Grattarola, Daniele Zambon, Cesare Alippi, and Barbara Eva Hammer. Graph edit networks. In *International Conference on Learning Representations*, 2021 [161]
- Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Learning graph cellular automata. In *Neural Information Processing Systems*, 2021 [78]
- Jack B Maguire, Daniele Grattarola, Vikram Khipple Mulligan, Eugene Klyshko, and Hans Melo. Xenet: Using a new graph convolution to accelerate the timeline for protein design on quantum computers. *PLoS computational biology*, 17(9):e1009037, 2021 [135]

Part I

Operators

Chapter 2

Background

This chapter introduces the relevant background to understand the contributions of the thesis and frame them into the larger context of graph machine learning.

We begin by introducing the basic terminology of graph theory. Then, we give a brief introduction to the field of spectral graph theory up to the concepts of graph Fourier transform and graph convolution.

Finally, we present graph neural networks and their fundamental building blocks: convolutional layers, which learn a node-level representation of the graph; pooling layers, which reduce the size of the graph; readout layers, which map the graph to a global vector representation.

2.1 Definitions and notation

We begin this chapter by introducing the necessary concepts and terminology of graph theory that we will use throughout the thesis.

Graph A graph is a tuple $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, \dots, N\}$ is the node set of cardinality $|\mathcal{V}| = N$ and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the edge set.

Nodes We indicate nodes with their scalar indices, *e.g.*, node 1 or node i, although the numbering of the node set does not imply an order of the nodes, as $\mathcal{V}$ is usually an unordered set.

Edges We indicate edges as pairs of nodes (i, j). If an edge is *directed*, the order of the pair indicates the direction, *i.e.*, edge (i, j) goes *from* node i *to* node j. Otherwise, we consider edges as *undirected*, so that $(i, j) \in \mathcal{E} \iff (j, i) \in \mathcal{E}$.

Neighbours Given a node, it is often useful to consider the subset of nodes to which it is connected by an edge, which we call its *neighbours*. We can represent this idea with a *neighbourhood function* $\mathcal{N} : \mathcal{V} \to 2^{\mathcal{V}}$, where $2^{\mathcal{V}}$ indicates the power set of $\mathcal{V}$, which returns a subset of $\mathcal{V}$ such that $\mathcal{N}(i) = \{j \mid (j,i) \in \mathcal{E}\}$. For directed graphs, we can distinguish between outgoing and incoming neighbours, depending on whether we consider the target node i to be on the sending (left) or receiving (right) end of a directed edge. However, in this thesis, we only consider the case of incoming neighbours.

Attributes We also focus on the general family of *attributed graphs* in which each node and, optionally, each edge can have a vector attribute (or *feature*) associated with it. For example, in a social network, node attributes could describe information related to each individual (age, sex, job, height, *etc.*) while edge attributes could describe the relationship between individuals (friends, coworkers, year when they met, *etc.*). We use $\mathbf{x}_i \in \mathbb{R}^{D_n}$ to indicate the attribute associated with node i and, similarly, we use $\mathbf{e}_{ij} \in \mathbb{R}^{D_e}$ to indicate a vector attribute associated with edge (i,j).

Adjacency matrix We indicate with $\mathbf{A} \in \mathbb{R}^{N \times N}$ the adjacency matrix of the graph, for which $\mathbf{a}_{ij} \neq 0 \iff (i,j) \in \mathcal{E}$. The entries of the adjacency matrix can be binary to indicate the presence or absence of edges, or they can take any value to represent the connection weight between pairs of nodes. For undirected graphs, the adjacency matrix is symmetric, *i.e.*, $\mathbf{a}_{ij} = \mathbf{a}_{ji}$.

Attribute matrices We indicate with $\mathbf{X} \in \mathbb{R}^{N \times D_n}$ the matrix that represents all node attributes of a graph (in this case, an order of the nodes must be chosen, although the specific order is not important). The matrix of node attributes is also sometimes referred to as a *graph signal*, especially when drawing a parallel between classical signal processing and the theory of signal processing on graphs that we cover in later sections.

Similarly, although less common, we can represent edge attributes in a tensor $\mathbf{E} \in \mathbb{R}^{N \times N \times D_e}$ (in this case, by convention, if $\mathbf{a}_{ij} = 0$ then the edge attributes $\mathbf{e}_{ij,:} = [0, \dots, 0]$).

Degree matrix We indicate with $\mathbf{D} \in \mathbb{R}^{N \times N}$ the diagonal degree matrix, for which

$$\mathbf{d}_{ij} = \begin{cases} \sum_{j=0}^{N-1} \mathbf{a}_{ij} & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases} \tag{2.1}$$

Laplacian matrices Finally, we introduce the notion of Laplacian matrix which, as we will see in future sections, is directly related to the notion of Laplace operator in differential calculus. If a graph is undirected, *i.e.*, its adjacency matrix is symmetric, we define the *combinatorial Laplacian* matrix as $\mathbf{L} = \mathbf{D} - \mathbf{A}$ and the *symmetric normalised Laplacian* as $\mathbf{L}_n = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$. In the case of directed graphs, equivalent formulations of the combinatorial and normalised Laplacian exist, although we will not consider them here.

2.2 Spectral graph theory

A common approach to studying graphs and their topology relies on analysing the characteristic matrices we introduced in the previous section. Specifically, by studying the eigenvalue decomposition of matrices like the adjacency matrix or the Laplacian, which encode the topological structure of the graph, we can gain important knowledge about the graphs and their high-level properties. We refer to the process of analysing a graph through the spectral analysis of its characteristic matrices as *spectral graph theory*.

As we will see, some core concepts commonly found in calculus and machine learning can be seen as particular cases of a much more general theory that relies on graphs. This will allow us, in future sections, to define a notion of convolution on irregular geometrical domains, which we will then use to design convolutional neural networks for graphs.

In this section, we go over a brief introduction of the key concepts of graph spectral theory that we will use in the following chapters of the thesis.

2.2.1 Spectral analysis of the Laplacian

Many concepts of spectral graph theory are well-defined irrespective of the specific characteristic matrix that we use to describe the graph topology. In fact, most of the results that we will see regarding graph convolutional networks are well-defined for any choice of matrix, as long as the matrix has the same structure, or *sparsity pattern*, of the adjacency matrix. In particular, we refer to

the notion of *structure operators* to indicate the family of matrices $\mathbf{S}$ such that $\mathbf{s}_{ij} \neq 0 \iff \mathbf{a}_{ij} \neq 0, \forall i \neq j$, for which most of graph spectral theory can be derived.

However, for the scope of this thesis, it will be sufficient to cover a relatively small subset of graph spectral theory, *i.e.*, only the essential notions up to the idea of graph convolution. For this reason, throughout this section, we only focus on the Laplacian as a structure operator.

We start by noting that the Laplacian is a real symmetric positive-semidefinite matrix and, therefore, admits a full eigendecomposition with N non-negative real eigenvalues $0 \leq \lambda_0 \leq ... \leq \lambda_{N-1}$ and a set of N orthogonal eigenvectors $\mathbf{u}_0, ..., \mathbf{u}_{N-1}$. In particular, we can write the Laplacian as

$$\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{-1} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\top} \tag{2.2}$$

where $\mathbf{U} \in \mathbb{R}^{N \times N}$ is the orthogonal matrix having eigenvectors $\mathbf{u}_k$ as columns and $\mathbf{\Lambda} = \text{diag}(\lambda_0, \ldots, \lambda_{N-1})$ the diagonal matrix of eigenvalues. Note that each eigenvector $\mathbf{u}_k$ is an N-dimensional vector whose entries are associated with the nodes of the graph, in other words, a graph signal.

The eigenvectors and eigenvalues of the Laplacian are closely related to the notion of frequency. In particular, we can use a generalised notion of frequency called *local quadratic variation* to describe how much the elements of an eigenvector (and, in general, any graph signal) vary over all edges of the graph. The local quadratic variation of an eigenvector is given by:

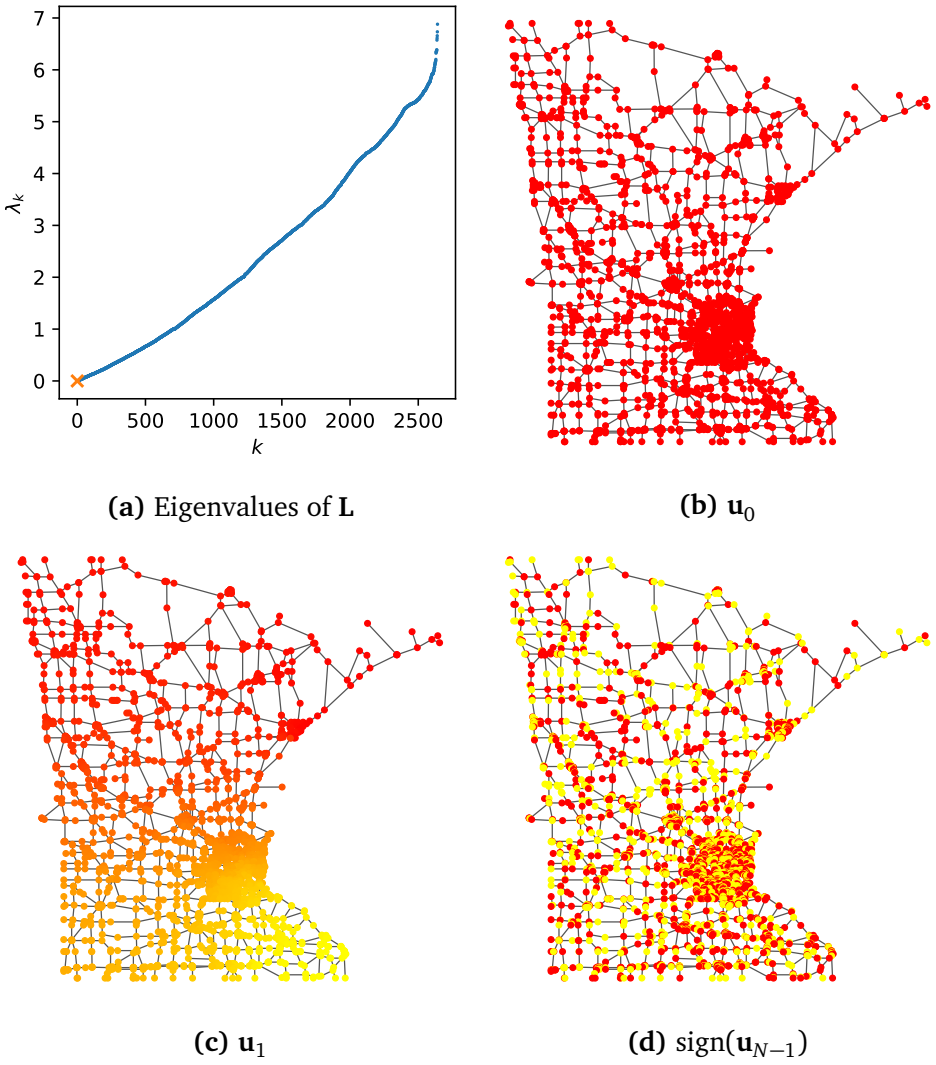
$$V_{\mathrm{L}}(\mathbf{u}_k) = \mathbf{u}_k^{\top}\mathbf{L}\mathbf{u}_k = \frac{1}{2}\sum_{(i,j)\in\mathcal{E}} \mathbf{A}_{ij}(\mathbf{u}_k[i] - \mathbf{u}_k[j])^2 = \lambda_k \left\|\mathbf{u}_k\right\|_2^2 = \lambda_k \tag{2.3}$$

As we see, the local quadratic variation of an eigenvector is its associated eigenvalue and, for this reason, the eigenvalues of the Laplacian are often referred to as the *frequencies* of the graph.

By further studying the spectrum of the graph Laplacian we can remark a few interesting aspects that are generally useful to analyse a graph.

Remark 1 The algebraic multiplicity of the null eigenvalues indicates the number of connected components of the graph, *i.e.*, the subgraphs for which any pair of nodes is connected by at least a path and whose nodes have no connections to the remaining node of the graph. Since a graph has at least one connected component, the smallest eigenvalue is always null, $\lambda_0 = 0$.

Remark 2 Eigenvectors associated with null eigenvalues are constant, *i.e.*, they have null local quadratic variation.



(a) Eigenvalues of $\mathbf{L}$

(b) $\mathbf{u}_0$

(c) $\mathbf{u}_1$

(d) sign($\mathbf{u}_{N-1}$)

Figure 2.1. Spectral analysis of the Minnesota road network. We plot the eigenvectors of the Laplacian as graph signals, so that the colour intensity of the i$^{\text{th}}$ node is given by $\mathbf{u}_k[i]$. (a) The eigenvalues of $\mathbf{L}$. Since the graph has only one component, there is only one null eigenvalue (marked "X"). (b) The eigenvector $\mathbf{u}_0$, associated with the null eigenvalue λ_0, is constant. (c) The eigenvector $\mathbf{u}_1$, associated with the smallest non-zero eigenvalue, changes slowly over the graph. (d) The eigenvector $\mathbf{u}_{N-1}$, associated with the largest eigenvalue, changes quickly over the graph (we plot the sign of $\mathbf{u}_{N-1}$ instead of the actual value, to accentuate the change).

Remark 3 The eigenvectors associated with the smallest and largest non-zero eigenvalues of the Laplacian are useful indicators to partition the graph (respectively, to find strongly connected communities and to find the maximum cut of the graph). We use this property of the Laplacian extensively in Chapter 4.

We summarise these remarks in Figure 2.1, using the Minnesota road network as an example.

2.2.2 Graph Fourier transform and graph convolution

In order to introduce graph convolutional networks, we cover here the essential notions that allow us to extend the idea of convolution to the domain of graphs. In particular, we introduce the idea of the *graph Fourier transform* (GFT) as a generalisation of the discrete Fourier transform, which we will then apply to define a generalised convolution in the domain of graphs.

To derive the GFT, we start from a keystone observation that allows us to understand much of the literature on graph neural networks: graph signals are a generalisation of conventional signals defined on regular domains, like time series and images.

Although this might be non-trivial to see at first, it can be helpful to think about images as an example. In images, each pixel is a vector describing the intensity of red, green, and blue light, similar to the node attributes in attributed graphs. Further, we can also easily see that the pixels themselves carry little information of their own and that the way in which pixels are arranged is what really gives meaning to an image. In a sense, it is as though pixels are arranged in a particular kind of graph, namely a regular *lattice graph*, which tells us which pairs of pixels should be close to each other in order to give meaning to the image. This applies trivially to the one-dimensional case of time series and can be generalised to any d-dimensional regular lattice (*e.g.*, the 4-dimensional space-time lattice).

From this observation, the quest of defining the convolution operation on graphs becomes a matter of translating the well-known tools of conventional signal processing to this general domain of graphs. In particular, our goal is to re-derive convolution in a way that does not assume a regular lattice structure of the data.

However, the typical intuition of convolution in the “time" domain (here, we generally talk about the *node domain*) as an operation based on shifting the signals cannot be easily applied here, since we lack a native notion of shift on the

graph due to the generally irregular structure and the fact that nodes might not have a given order (unlike in lattices, where the order of the nodes relative to their neighbours is given—*e.g.*, up, down, left, or right). Therefore, we have to look elsewhere, and in particular at the interpretation of convolution as a multiplication in the *frequency* domain.

Without loss of generality, let $f[n]$ and $g[n]$ be two discrete and finite one-dimensional signals over time, $n = 0, \ldots, N-1$, and let the Fourier transform of $f[n]$ be

$$\mathcal{F}\{f\}[k] = \hat{f}[k] = \sum_{n=0}^{N-1} f[n] e^{-i\frac{2\pi}{N}kn}, \tag{2.4}$$

while the inverse Fourier transform is

$$\mathcal{F}^{-1}\{\hat{f}\}[n] = f[n] = \frac{1}{N}\sum_{k=0}^{N-1} \hat{f}[k] e^{i\frac{2\pi}{N}kn}. \tag{2.5}$$

Now, let the discrete convolution between $f[n]$ and $g[n]$ be

$$(f \star g)[n] = \sum_{m=0}^{N-1} f[n-m] g[m]. \tag{2.6}$$

The convolution theorem states that

$$f \star g = \mathcal{F}^{-1}\left\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\right\}, \tag{2.7}$$

i.e., the effect of convolving the two signals can be obtained by transforming both signals to their frequency representation in the Fourier domain, then point-wise multiplying the two transformed signals, and finally re-transforming the result back to the original domain. Using this major result from signal processing, we can now introduce the GFT and the graph convolution.

As commented above, we can consider regular lattices as a particular kind of graph that bridges the conventional regular domains with the arbitrarily structured graphs we are interested in. Again, without loss of generality, here we look at one-dimensional lattices, also called *path graphs*.

The Laplacian of a path is given by:

$$\mathbf{L} = \begin{bmatrix} \mathbf{a}_{01} & -\mathbf{a}_{01} & & & & \\ & & \ldots & & & \\ \ldots & -\mathbf{a}_{i,i-1} & \mathbf{a}_{i,i-1} + \mathbf{a}_{i,i+1} & -\mathbf{a}_{i,i+1} & \ldots & \\ & & \ldots & & & \\ & & & -\mathbf{a}_{N-1,N-2} & \mathbf{a}_{N-1,N-2} & \end{bmatrix} \tag{2.8}$$

where the only non-zero entries are in the diagonal, subdiagonal and superdiagonal of the matrix.

The eigendecomposition of the Laplacian, in this case, can be obtained analytically as:

$$\mathbf{u}_k[n] = \begin{cases} 1, & \text{for } k = 0 \\ e^{i\pi(k+1)n/N}, & \text{for odd } k, k < N-1 \\ e^{-i\pi kn/N}, & \text{for even } k, k > 0 \\ cos(\pi n), & \text{for odd } k, k = N-1 \end{cases} \tag{2.9}$$

which are exactly the basis functions used to write the Fourier transform in Equation (2.4).

The reason for this parallel can be understood by considering that the Fourier transform was invented as a way of representing a signal in a different basis, namely a basis of orthogonal eigenfunctions of the Laplace operator. When moving from a continuous to a discrete domain, the eigenfunctions of the Laplace operator become the eigenvectors of the Laplacian operator which, as can be readily seen from the definition, has a similar meaning to its continuous counterpart (*i.e.*, it computes a local divergence of the signal). Given that the path graph is, as we saw, just a different way of representing the time domain, we recover the same analytical form of the Fourier eigenbasis in discrete time.

However, this result also tells us something important about the Fourier transform: it can be derived from the Laplacian of a graph, without assuming anything about its structure. If the graph represents a conventional regular domain, as we saw above for the path graph representing the temporal lattice, we recover known results from signal processing. However, the procedure is well defined for any arbitrary graph:

1. Compute the Laplacian $\mathbf{L}$;
2. Compute eigenvectors $\mathbf{u}_k$;
3. Compute the Fourier transform as

$$\hat{f}[k] = \sum_{n=0}^{N-1} f[n]\mathbf{u}_k[n]. \tag{2.10}$$

In short, we define the GFT of a graph signal as the representation of the signal in the basis of eigenvectors of the Laplacian. We can extend this principle to multi-dimensional graph signals (or, as we have called them, node attributes)

and write the GFT of Equation (2.10) in a compact matrix form. Given a graph signal $\mathbf{F} \in \mathbb{R}^{N \times D_n}$, we write its GFT as:

$$\hat{\mathbf{F}} = \mathbf{U}^\top \mathbf{F}. \tag{2.11}$$

The inverse GFT (iGFT), then, is simply:

$$\mathbf{F} = \mathbf{U}\hat{\mathbf{F}} \tag{2.12}$$

i.e., a map of the transformed signal back to its original basis. Note that Equations (2.11) and (2.12) are general versions of Equations (2.4) and (2.5).

Crucially, we note that the convolution theorem holds also for the GFT, so that we can write the convolution between two graph signals $\mathbf{F}, \mathbf{G} \in \mathbb{R}^{N \times D_n}$ as per Equation (2.7):

$$\mathbf{F} \star \mathbf{G} = \mathbf{U}\left(\mathbf{U}^\top \mathbf{F} \odot \mathbf{U}^\top \mathbf{G}\right), \tag{2.13}$$

where $\odot$ indicates element-wise multiplication.

From Equation (2.13), we can also define the concept of *graph filter*. We start by noting that conventional filtering is typically implemented as a convolution between the signal of interest and a filter function. Let $\mathbf{F}, \mathbf{G}$ in Equation (2.13) be, for simplicity of notation and without loss of generality, one-dimensional graph signals $\mathbf{f}, \mathbf{g} \in \mathbb{R}^N$ (*i.e.*, $D_n = 1$, omitted for simplicity). We can rewrite Equation (2.13) as:

$$\begin{aligned}
\mathbf{f} \star \mathbf{g} &= \mathbf{U}\left(\mathbf{U}^\top \mathbf{f} \odot \mathbf{U}^\top \mathbf{g}\right) && (2.14)\\
&= \mathbf{U}\left(\mathbf{U}^\top \mathbf{g} \odot \mathbf{U}^\top \mathbf{f}\right) && (2.15)\\
&= \mathbf{U}\left(\mathrm{diag}(\mathbf{U}^\top \mathbf{g})\mathbf{U}^\top \mathbf{f}\right) && (2.16)\\
&= \mathbf{U} \cdot \mathrm{diag}(\mathbf{U}^\top \mathbf{g}) \cdot \mathbf{U}^\top \mathbf{f}. && (2.17)
\end{aligned}$$

By comparing Equation (2.17) with the eigendecomposition of the Laplacian in Equation (2.2), *i.e.*, $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$, and noting that $\mathbf{\Lambda}$ and $\mathrm{diag}(\mathbf{U}^\top \mathbf{g})$ are both diagonal matrices, we can interpret the effect of a graph filtering operation as an action on the eigenvalues of the Laplacian:

$$g(\mathbf{\Lambda}) \coloneqq \mathrm{diag}(\mathbf{U}^\top \mathbf{g}). \tag{2.18}$$

It is also possible to write the action of the filter as an element-wise operation acting on the individual eigenvalues of the Laplacian, *i.e.*,

$$g(\lambda_k) = \sum_{n=0}^{N-1} \mathbf{u}_k[n]\mathbf{g}[n]. \tag{2.19}$$

We write the complete filtering operation as:

$$\mathbf{F} \star \mathbf{G} = \mathbf{U} g(\mathbf{\Lambda}) \mathbf{U}^\top \mathbf{F}, \tag{2.20}$$

Additionally, if $g(\lambda)$ is an analytic function, its action on the eigenvalues $\mathbf{\Lambda}$ can be equivalently written as an action on the entries of the Laplacian itself, *i.e.*,

$$\mathbf{g}(\mathbf{L}) = \mathbf{U} g(\mathbf{\Lambda}) \mathbf{U}^\top. \tag{2.21}$$

This latter formulation in the domain of nodes is much more efficient since it does not require computing the full eigendecomposition of $\mathbf{L}$ and the multiplication with $\mathbf{U}$. Putting everything together, for a graph with Laplacian $\mathbf{L}$, we write the filtering action with graph filter g on a graph signal $\mathbf{F}$ as:

$$\mathbf{F} \star \mathbf{G} = g(\mathbf{L}) \mathbf{F}. \tag{2.22}$$

This duality between node domain and frequency domain will be useful to design graph filters with a given response in the frequency domain while allowing us to apply them in the node domain efficiently.

In the following section, we show how to design graph filters that can be learned with stochastic gradient descent, leading to graph convolutional networks and the general family of graph neural networks.

2.3 Graph neural networks

Seminal research on Graph Neural Networks (GNNs) dates back to the works of Sperduti and Starita [198], Gori et al. [72] and Scarselli et al. [181], whose goal was to design a neural network that could process graph-structured data. In the following years, key developments in the field of graph signal processing [73, 173, 141, 159] led to the development of deep learning techniques for processing geometric data, under the umbrella term *geometric deep learning* popularised by the work of Bronstein et al. [27], which eventually led to modern GNN architectures.

In this section, we introduce GNNs and their fundamental building blocks.

We define a graph neural network as a sequence of differentiable operations that take as input a graph and return a vector representation of the graph itself or its nodes. In typical deep learning fashion, we group these operations into layers that perform one atomic transformation. Specifically, we distinguish among convolutional layers, pooling layers, and readout layers.

Convolutional layers are operators that compute a representation of the node features of a graph, with the general form:

$$f_{\text{conv}} : \mathcal{G} \mapsto \mathbf{X}' \in \mathbb{R}^{N \times D'_n}. \tag{2.23}$$

Such transformation can depend on the input graph's attributes (of both nodes and edges) and structure. It is also possible to define convolutional operators that transform the edge attributes of the input graph, although we do not cover this case here.

Pooling layers are operators that transform a graph with the generic goal of reducing the number of nodes:

$$f_{\text{pool}} : \mathcal{G} \mapsto \mathcal{G}', \tag{2.24}$$

where $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ and $|\mathcal{V}'| < |\mathcal{V}|$.[1]

Finally, readout layers compute a vector representation of the input graph and are similar, in spirit, to the flattening step that one applies after convolution in a typical CNN. A readout is a function:

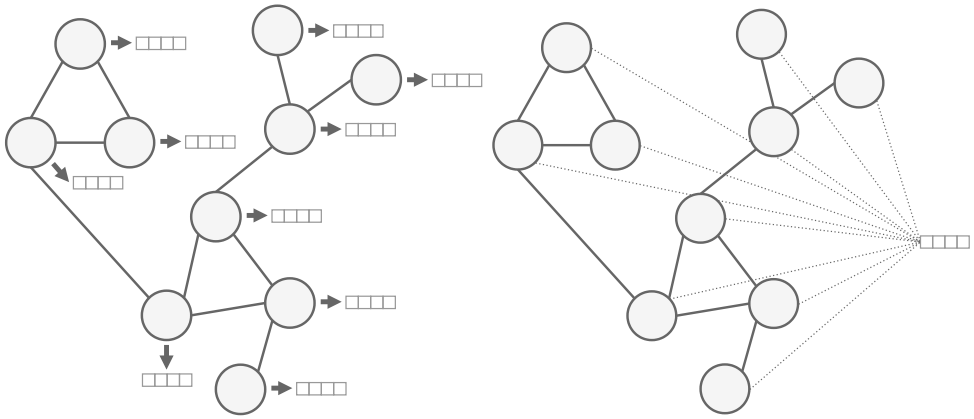
$$f_{\text{out}} : \mathcal{G} \mapsto \mathbf{z} \in \mathbb{R}^{D_{\text{out}}}. \tag{2.25}$$

The composition of such layers into a computational graph is usually referred to as a GNN architecture or, more simply, a GNN. Since GNNs are composed of differentiable operations, their parameters can be learned end-to-end with back-propagation and stochastic gradient descent, like any neural network. Different kinds of GNNs can be constructed to solve different tasks, with the principal difference being in the output. We distinguish two main cases.

In node-level learning, the output of the GNN is a representation of the nodes of the input graph. Examples of node-level learning include classifying the users of a social network, predicting traffic on different roads, or modelling the trajectory of particles under the effect of gravity. An example of node-level learning in classical deep learning is the task of image segmentation, in which each pixel is classified as belonging to a particular object or region. In this case, any pooling or readout operation must be reversed to ensure that the number of nodes is preserved, although typically this effect is obtained by not introducing pooling or readout layers at all.

Alternatively, in graph-level learning, the output of the GNN is a representation of the graph itself. This is more similar to how classical CNNs are used,

[1] In Chapter 4, we will comment on some particular cases in which pooling layers undesirably increase the number of nodes.



(a) Node-level: one output per node. **(b)** Graph-level: one output per graph.

Figure 2.2. Schematic representation of node-level and graph-level prediction. At the node level, the GNN outputs a prediction for each node. At the graph level, the GNN outputs a single prediction for the entire graph.

in that the entire graph is seen as the input of the network and the prediction represents a *global* description of the input. GNNs for graph-level learning can combine convolutional and pooling layers arbitrarily, like in CNNs, and the mapping of the graph to a vector representation is computed by a readout layer.

Another relevant use of GNNs is the prediction of edges, usually referred to as *link prediction*, although this can be framed as a particular case of node-level prediction (*i.e.*, predicting node representations that are similar for nodes that should be connected by an edge).

Figure 2.2 shows the difference between the two settings.

Similar to classical deep learning architectures, GNNs have non-linear activation functions, can be regularised with weight decay, dropout, and batch normalisation, and can be used to perform classification or regression tasks depending on the final activation function and loss used during training.

In the following sections, we introduce more in detail the principal works of GNN literature, with a focus on convolutional and pooling layers.

2.3.1 Convolutional operators

Convolutional layers are used to learn a representation of the nodes of a graph.

Traditionally, the literature distinguishes between two main approaches to implement convolutional layer, namely spectral and spatial, although the line between the two is often blurred and the distinction is primarily due to historical

reasons.

Spectral approaches are those methods based on the interpretation of convolution as a multiplication in the Fourier domain, which we have introduced in Section 2.2. Typically, spectral GNNs formulate a filtering operation in the graph Fourier domain using a filter $g_{\boldsymbol{\theta}}(\lambda)$ with learnable parameters $\boldsymbol{\theta}$. Then, the parameters of the filter are learned in a data-driven way, usually by gradient descent.

On the other hand, spatial approaches formulate convolution as an exchange of information between neighbours, and are usually referred to as *message-passing* layers. For each node i, a message-passing layer computes a vector $\mathbf{x}'_i \in \mathbb{R}^{D'_n}$ according to the following scheme:

$$\mathbf{x}'_i = \gamma\left(\mathbf{x}_i, \square_{j\in\mathcal{N}(i)}\, \phi\left(\mathbf{x}_i, \mathbf{x}_j, \mathbf{e}_{ji}\right)\right), \tag{2.26}$$

where ϕ is called *message* function, $\square$ indicates a permutation-invariant operation to *aggregate* the set of messages coming from the neighbourhood $\mathcal{N}(i)$ (*e.g.*, a sum $\sum$ or a product $\prod$), and γ is and *update* function that computes the final node representation.

Purely spectral approaches are seldom used in the literature and, although motivated by spectral graph theory, the most commonly used spectral layers can be seen as specific instances of the very general message-passing paradigm.

The first works to introduce neural networks capable of learning node representations were those of Sperduti and Starita [198], Gori et al. [72] and Scarselli et al. [181]. The methods proposed in these works implemented a message-passing scheme that was recurrently applied until converging to a fixed node representation. These papers introduced much of the keystone concepts of the field and are widely regarded as the first papers on GNNs, although the recurrent formulation was eventually discarded in favour of feed-forward models.

The first purely spectral approach to GNNs was that of Bruna et al. [29], who proposed to learn a spectral filter $g_{\boldsymbol{\theta}}(\lambda)$ (*cf.* Section 2.2.2) directly in the Fourier domain using cubic B-splines. The filter had the form $g_{\boldsymbol{\theta}}(\lambda) = \mathbf{B}\boldsymbol{\theta}$, where $\mathbf{B} \in \mathbb{R}^{N\times K}$ is a cubic B-spline basis and $\boldsymbol{\theta} \in \mathbb{R}^K$ is a vector of coefficients. This approach relied on the filtering formulation of Equation (2.20) and was computationally expensive due to the double product with the eigenvector basis $\mathbf{U}$. Also, the reliance on $\mathbf{U}$ meant that the filter was only valid for the specific graph associated with the corresponding Laplacian, and could be transferred easily to different graphs. Finally, the proposed filter was not *localised* in node space. A graph filter is said to be K-localised if the action of $g_{\boldsymbol{\theta}}(\mathbf{L})$ on a node depends only on its K-hop neighbours. However, the spectral filtering described above generally

resulted in the interaction of all nodes across the graph, contrary to the principle of local processing of conventional CNNs.

Since the inception of spectral GNNs, however, many approaches have been proposed to make the filtering operation learnable, efficient, and localised. The most common implementation of such filters is based on polynomials of the form:

$$g_{\boldsymbol{\theta}}(\mathbf{L}) = \sum_{k=0}^{K} \boldsymbol{\theta}_k \mathbf{S}^k, \tag{2.27}$$

where $\mathbf{S}$ is any structure operator (*cf.* Section 2.1) and $\boldsymbol{\theta} \in \mathbb{R}^{K+1}$ is a vector of learnable parameters. This formulation based on powers of $\mathbf{S}$ ensures that the filter:

1. Has a constant number of parameters that does not depend on the size of the graph;
2. Is exactly K-localised [47];
3. Does not rely on computing the GFT explicitly, which makes it efficient to apply and transferable to different graphs.

Although we have defined the GFT and graph convolution for a generic structure operator $\mathbf{S}$, the methods found in the literature are usually based on specific operators (typically the Laplacian or the adjacency matrix).

Defferrard et al. [47] introduced a GNN based on Chebyshev polynomials of a modified Laplacian matrix, which allowed them to compute the polynomial filter recursively with a cost of $O(K \cdot |\mathcal{E}|)$. We refer to this model as the Chebyshev GNN. The Chebyshev GNN transforms the node attributes $\mathbf{X} \in \mathbb{R}^{N \times D_n}$ into $\mathbf{X}' \in \mathbb{R}^{N \times D'_n}$ as:

$$\mathbf{X}' = \sum_{k=0}^{K-1} \mathbf{T}^{(k)} \mathbf{W}^{(k)} + \mathbf{b}^{(k)}, \tag{2.28}$$

where $\mathbf{W}^{(k)} \in \mathbb{R}^{D_n \times D'_n}$ and $\mathbf{b}^{(k)} \in \mathbb{R}^{D'_n}$ are learnable parameters and $\mathbf{T}^{(k)}$ indicates the k^{th} term of the Chebyshev polynomial of modified Laplacian $\bar{\mathbf{L}}$:

$$\mathbf{T}^{(0)} = \mathbf{X} \tag{2.29}$$

$$\mathbf{T}^{(1)} = \bar{\mathbf{L}}\mathbf{X} \tag{2.30}$$

$$\mathbf{T}^{(k \geq 2)} = 2 \cdot \bar{\mathbf{L}}\mathbf{T}^{(k-1)} - \mathbf{T}^{(k-2)}, \tag{2.31}$$

The modified Laplacian $\bar{\mathbf{L}}$ is given by:

$$\bar{\mathbf{L}} = \frac{2}{\lambda_{max}} \cdot (\mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}) - \mathbf{I} \tag{2.32}$$

which is a simple transformation of $\mathbf{L}$ to ensure that its spectrum lies in the interval $[-1, 1]$. Note that the computation of the Chebyshev polynomial of Equations (2.29)-(2.31) only requires one multiplication with $\bar{\mathbf{L}}$ at each step, which can be efficiently implemented as a sparse matrix multiplication with cost $O(|\mathcal{E}|)$.

In a later work, Kipf and Welling [106] proposed a simplification of the Chebyshev GNN based on a first-order polynomial of the normalised adjacency matrix. This formulation also replaced the 0-order term of the filter (*i.e.*, the identity matrix $\mathbf{L}^0 = \mathbf{I}$) by modifying the adjacency matrix to include self-loops, *i.e.*, making the diagonal of $\mathbf{A}$ non-zero, to improve the stability of the filter. Despite creating some confusion in the nomenclature, this latter model of GNN is called the Graph Convolutional Network (GCN). The GCN model computes the following transformation of the node attributes:

$$\mathbf{X}' = \bar{\mathbf{A}}\mathbf{X}\mathbf{W} + \mathbf{b} \tag{2.33}$$

where

$$\bar{\mathbf{A}} = \hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}, \tag{2.34}$$

$\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, and $\hat{\mathbf{D}}$ is the degree matrix of $\hat{\mathbf{A}}$.

We will discuss properties and limitations of these approaches in Chapter 3.

Besides operators that implement a spectral graph convolution, many other designs have been proposed to make GNNs more expressive, mostly based on the message-passing scheme of Equation (2.26). Here we focus only on a few important aspects and refer the reader to the numerous surveys on GNN literature for a detailed account of many GNN implementations (*e.g.*, [229]).

As previously noted, the key to the message-passing paradigm is the modular description of a layer's action in terms of message, aggregation, and update functions. For example, the GCN described above can also be seen as a particular kind of message-passing layer with a learnable linear projection as the message function and an aggregation function that computes a weighted average based on the specific reference operator.

In the following, we describe some of the principal message-passing layers that represent different kinds of computation.

Edge-specific messages with edge attributes First, from Equation (2.26) we see that message-passing layers can take into account edge attributes, making the message function ϕ edge-dependent. There are two main approaches to achieve this.

Simonovsky and Komodakis [195] proposed to compute the parameters of the message function ϕ as the output of a *kernel-generating network* (KGN):

$$f_{\text{KGN}} : \mathbf{e}_{ij} \mapsto \mathbf{W}_{(i,j)} \in \mathbb{R}^{D_n \times D'_n}. \tag{2.35}$$

The role of the KGN is to map the edge attributes to a matrix of parameters, which are then used to compute the messages from the neighbours. This ensures that each message depends on the specific edge attribute between a node and each neighbour. The overall formulation of the model, called Edge-Conditioned Convolution (ECC), is:

$$\mathbf{x}'_i = \mathbf{W}^\top_{(\text{root})}\mathbf{x}_i + \sum_{j \in \mathcal{N}(i)} f_{\text{FGN}}(\mathbf{e}_{j,i})^\top \mathbf{x}_j + \mathbf{b} \tag{2.36}$$

$$= \mathbf{W}^\top_{(\text{root})}\mathbf{x}_i + \sum_{j \in \mathcal{N}(i)} \mathbf{W}^\top_{(j,i)}\mathbf{x}_j + \mathbf{b} \tag{2.37}$$

where $\mathbf{W}_{(\text{root})} \in \mathbb{R}^{D_n \times D'_n}$ is a matrix of parameters for the root node (since, in general, we don't assume self-loops to be present but we still want to use a node's current attributes to compute its representation) and $\mathbf{b}$ is a bias vector of parameters.

An alternative approach to integrating edge attributes into the computed messages is to concatenate, or otherwise merge, $\mathbf{e}_{ji}$ to $\mathbf{x}_j$ and use the result to compute the messages. A model that follows this approach is the Crystal Convolution presented by Xie and Grossman [231].

Edge-specific messages without edge attributes In the absence of edge attributes, the message function can still be made edge-specific by computing messages as a function of both $\mathbf{x}_i$ and $\mathbf{x}_j$. This is the idea behind the very successful graph attention network (GAT) of Veličković et al. [211]. GAT works by computing a dynamic reference operator using the self-attention mechanism, so that the aggregation of messages is weighted by a score computed from the node attributes $\mathbf{x}_i$ and $\mathbf{x}_j$. At its core, GAT computes a simple convolution:

$$\mathbf{X}' = \bar{\mathbf{A}}\mathbf{X}\mathbf{W} + \mathbf{b} \tag{2.38}$$

where $\mathbf{W} \in \mathbb{R}^{D_n \times D'_n}$ and $\mathbf{b} \in \mathbb{R}^{D'_n}$ are parameters and

$$\bar{\mathbf{a}}_{ij} = \frac{\exp\left(\sigma\left(\boldsymbol{\theta}^\top[(\mathbf{X}\mathbf{W})_i \,\|\, (\mathbf{X}\mathbf{W})_j]\right)\right)}{\sum_{k \in \mathcal{N}(i)\cup\{i\}} \exp\left(\sigma\left(\boldsymbol{\theta}^\top[(\mathbf{X}\mathbf{W})_i \,\|\, (\mathbf{X}\mathbf{W})_k]\right)\right)}. \tag{2.39}$$

where $\boldsymbol{\theta} \in \mathbb{R}^{2D'_n}$ is a vector of parameters, $\|$ indicates concatenation, and σ is an activation function (in the original paper, a leaky rectified linear unit).

While GAT is certainly the leading approach for computing edge-specific messages without edge attributes, other possibilities exist like, for example, the EdgeConv layer of Wang et al. [215] which computes:

$$\mathbf{x}'_i = \sum_{j \in \mathcal{N}(i)} \text{MLP}\big(\mathbf{x}_i \| \mathbf{x}_j - \mathbf{x}_i\big). \tag{2.40}$$

We will not venture further into the subject since it is not crucial for the presentation of this thesis.

Graph Isomorphism Networks Finally, we mention a particularly important model called the Graph Isomorphism Network (GIN), which is motivated by theoretical results about the universality of neural networks to represent functions over sets. A GIN layer implements the following operation:

$$\mathbf{x}'_i = \text{MLP}\Big((1+\epsilon) \cdot \mathbf{x}_i + \sum_{j \in \mathcal{N}(i)} \mathbf{x}_j\Big) \tag{2.41}$$

where MLP indicates a multi-layer perceptron and ϵ is a parameter that can be set manually or learned (although, in practice, it is often set to 0). Assuming the universality of MLP, Equation (2.41) can be shown to be as powerful as the Weisfeiler-Lehman graph isomorphism test in distinguishing a given pair of graphs.

2.3.2 Pooling operators

Complementary to convolutional layers, pooling operators are a large and diverse class of techniques used both in GNNs and as stand-alone operators to reduce the number of nodes in a graph. Pooling layers bring the double benefit of reducing computational costs and making the graph representation more abstract, similar to the pooling layer in CNNs.

We generically define a pooling operation as any function that maps a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ to a graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ such that $|\mathcal{V}'| < |\mathcal{V}|$. The term *coarsening* is also used interchangeably with *pooling*. A formal definition of pooling operators is a contribution of this thesis and we will give it in Chapter 4. Here, we limit our exposition to a high-level description of the many diverse techniques for pooling that are found in the literature.

Among the early uses of pooling operators found in the seminal GNN literature, the Graclus algorithm [49] was used by Defferrard et al. [47] and later

adopted in other works on GNNs [149, 119, 57]. Graclus (shorthand for "graph clustering") is an algorithm that halves the size of the node set by iteratively collapsing a randomly selected node with its most strongly connected neighbour. The method can be shown to be equivalent to more expensive approaches based on the eigendecomposition of the adjacency matrix (we describe one such method in-depth in Chapter 4).

The literature about machine learning for point clouds (a field of computer vision that models three-dimensional objects as a collection of points in space) also introduced pooling techniques to generalise the traditional pooling layers of CNNs. Most notably, Simonovsky and Komodakis [195] adopt the VoxelGrid algorithm, in which a regular grid is overlaid on a point cloud and all the points in a voxel are summarised by their centroid. A similar voxel-oriented pooling is also proposed by Riegler et al. [174]. A different approach for pooling point clouds is proposed by Qi et al. [168], where the authors create a hierarchical representation by grouping together points around a given set of centroids, which are found with a *farthest point sampling* algorithm. We also mention the more recent work by Lei et al. [118] which proposes a pooling strategy based on *octree* partitioning [144].

Alternatively, many pooling techniques have been proposed based on different design principles and requirements. The graph-theoretical analysis proposed by Loukas [128], Hermsdorff and Gunderson [85], Cai et al. [31], aimed at coarsening graphs so that their spectrum is preserved in the pooled graph. In clique pooling [132], graphs are coarsened by aggregating maximal cliques. All nodes belonging to a clique are summarised by their maximum or average and become a new node in the coarsened graph. If a node in one clique shares an edge with a node in another, the nodes representing the cliques are connected in the new graph. Bacciu and Di Sotto [7] developed a method based on the non-negative matrix factorisation (NMF) of the adjacency matrix. After the decomposition, one of the two factors is used as a soft clustering matrix for coarsening the graph. In the EigenPooling approach by Ma et al. [133], a graph is first partitioned into subgraphs using spectral clustering, and each subgraph is mapped to a node in the pooled graph. Then, the eigenvectors of each subgraph's Laplacian are used to define a set of downsampling operators for the node attributes. In Laplacian pooling (LaPool) [155], nodes characterised by high local quadratic variation (*cf.* Equation (2.3)) between their features and those of their neighbours are selected as *leaders*, and the remaining nodes are assigned to one cluster using a sparse attention mechanism. When combined with simple low-pass graph filters like GCN, LaPool yields a band-pass filter that retains the medium frequencies of the graph signal. Xie et al. [232] propose a procedure that iteratively collapses nodes with

high similarity and similar neighbourhoods. This approach is similar to Graclus, although it accounts for second-order similarity relations between nodes.

The current trend (and state of the art) in graph pooling has seen the advent of learnable operators that, much like convolutional layers, can dynamically adapt to a particular task to compute optimal pooling.

The DiffPool operator [237] was among the first attempts to learn a pooling operator end-to-end. In DiffPool, a GNN is trained to compute a soft clustering matrix from the node features, which is then used to aggregate the nodes in each cluster. To ensure convergence to balanced clusters, two additional loss terms are minimised during training, namely a *link prediction* loss and an *entropy* loss. More recently, Bodnar et al. [23] proposed the Mapper-based PageRank (MPR), a pooling method based on the Mapper algorithm [197] using PageRank [163] as a *lens* function. Other approaches for computing learnable lens functions can be used, and the authors also show that their *Deep Graph Mapper* algorithm is a generalisation of those pooling methods based on a soft clustering, like DiffPool and LaPool.

However, despite being very effective in practice, methods based on learning a clustering operation have been criticised for their high memory cost [32]. To address these limitations, several works have proposed a family of sparse operators collectively known as *Top-K* methods. In these approaches, node features are projected to a scoring vector through a learnable transformation, and the scoring vector is then used to decide which nodes to keep in the coarsened graph. In their simplest formulation [88, 32], Top-K approaches are based on a linear projection to compute the scoring vector. Then, the highest K elements of the scoring vector identify K nodes to keep in the coarsened graph (hence the name Top-K). An alternative strategy is proposed by Knyazev et al. [108], where all nodes with a score above a given threshold are kept. The threshold can be either selected manually or learned end-to-end as a parameter. Finally, Lee et al. [117] propose to learn the scoring vector with a GCN [106], rather than a linear projection. Ranjan et al. [171] propose a more complex way of computing the scoring vector called Adaptive Structure Aware Pooling (ASAP). In ASAP, the graph is clustered using the *Master2Token* technique (also introduced in the paper), which for each node computes a cluster based on a self-attention mechanism. Then, for each cluster, a custom GNN layer (called *Local Extrema Convolution*) is used to compute the scoring vector and a Top-K selection is applied. In a similar spirit of Top-K, but with a focus on edges instead of nodes, Edge Contraction Pooling [51, 50] learns to compute a score for the incident edges of each node. Edges are then iteratively contracted (*i.e.*, their extremes are merged into a single node) according to the scores.

2.3.3 Readout operators

Several readout, or *global pooling*, methods to reduce graphs to a compact vector representation have also been proposed in the literature.

Typical operations to compute a graph readout include the sum, product, average or maximum of the node features, applied column-wise to the matrix of node attributes so that the typical structure of a readout function is:

$$f_{\text{out}} : \mathbb{R}^{N \times D_n} \rightarrow \mathbb{R}^{D_n} \tag{2.42}$$

We generally require the readout function to be invariant to node permutations, so that the representation of a graph does not depend on a specific order of the nodes.

Besides the simple approaches described above, however, more sophisticated techniques have been proposed to compute graph readouts. In one of the earliest works describing GNNs, Scarselli et al. [182] distinguished between *supervised* and *unsupervised* nodes, where the representation of supervised nodes learned by the GNN was used to predict global properties of the whole graph. We also mention SortPooling [248], which leverages the Weisfeiler-Lehman subtree kernel to sort node features in a sequence, and the global readout proposed in DEMO-Net [224], that groups nodes by degree and performs a concatenation of their sums. More recently, Navarin et al. [151] proposed a *universal* global pooling operator based on the results of Zaheer et al. [241].

Li et al. [124] proposed an attention-based readout where each node attribute is gated with a learnable attention vector before performing a simple global sum. We finally mention the readouts proposed in the works of Corcoran [39], Atwood and Towsley [6], Xu et al. [234] and Bai et al. [9].

Chapter 3

Convolution: GNNs with ARMA filters

This chapter presents the ARMA graph neural network [22], a graph convolutional layer based on a filter with a rational frequency response. The layer is inspired by the ARMA graph filter [92], which approximates the desired response by iterating a recursion. The ARMA GNN is designed to be more expressive than the typical convolutional layers, which are usually based on polynomial filters.

We show the advanced modelling ability of the ARMA GNN by studying its effect on real graph signals and comparing it to the effect of a popular polynomial GNN. We also run an extensive suite of benchmarks, achieving state-of-the-art performance in tasks of node classification, graph signal classification, graph classification and graph regression.

3.1 ARMA graph filters

The polynomial filters typically used in GNNs (*cf.* Section 2.3.1) have a finite impulse response

$$g_{\boldsymbol{\theta}}(\lambda) = \sum_{k=0}^{K} \boldsymbol{\theta}_k \lambda^k, \tag{3.1}$$

which computes a weighted moving average of graph signals on local K-hop neighbourhoods [206]. As a result, polynomial filters are more prone to overfitting the observed graph frequencies (*i.e.*, the eigenvalues of the Laplacian) when modelling high-order interactions [92, 206]. This hampers the GNN's generalisation capability, as it becomes sensitive to noise and small changes in the graph topology. Also, since polynomials are very smooth, they cannot model sharp

changes in the frequency response (*i.e.*, how the filter acts on the eigenvalues of the Laplacian, as per Equation (2.20)) and, after a few convolutions, the node features become too smoothed over the graph and the initial node information is lost [122].

To address the limitations of polynomial filters in GNNs, a more versatile class of filters is found in the family of *rational* filters, which can model a richer variety of frequency responses and can account for higher-order neighbourhoods than polynomial filters with the same number of parameters.

A rational graph filter of order K has the following form, in the node domain:

$$g(\mathbf{L}) = \left(\mathbf{I} + \sum_{k=1}^{K} q_k \mathbf{L}^k\right)^{-1} \left(\sum_{k=0}^{K-1} p_k \mathbf{L}^k\right) \tag{3.2}$$

where q_k and p_k are the filter's coefficients.

Different orders ($\leq K$) of the numerator and denominator in Equation (3.2) are trivially obtained by setting some coefficients to 0. Note that, by setting $q_k = 0, \forall k$, we recover a polynomial filter.

The expensive matrix inversion of Equation (3.2) makes rational filters unfeasible to use in practice, especially for GNNs. To address this issue, in [22], we propose a particular implementation of rational GNNs inspired by the autoregressive moving-average (ARMA) graph filters introduced by Isufi et al. [92], which implement a rational impulse response without explicitly computing the matrix inversion.

Specifically, the effect of a first-order ARMA filter (ARMA_1) is obtained by iterating, until convergence, the following recursion based on the potential kernel [92]:

$$\mathbf{X}^{(t+1)} = a\mathbf{M}\mathbf{X}^{(t)} + b\mathbf{X}, \tag{3.3}$$

where

$$\mathbf{M} = \frac{1}{2}(\lambda_{\max} - \lambda_{\min})\mathbf{I} - \mathbf{L}. \tag{3.4}$$

The recursion in Equation (3.3) is adopted in graph signal processing to apply a low-pass filter on a graph signal [129, 92], but it is also equivalent to the recurrent update used in Label Propagation [249] and Personalised Page Rank [163] to propagate information on a graph using a random walk with a restart probability.

Following the derivation of Isufi et al. [92, Theorems 1 and 2], we can analyse the frequency response of an ARMA_1 filter from the convergence of Equation (3.3):

$$\bar{\mathbf{X}} = \lim_{t\to\infty} \left[(a\mathbf{M})^t \mathbf{X}^{(0)} + b \sum_{i=0}^{t} (a\mathbf{M})^i \mathbf{X} \right]. \tag{3.5}$$

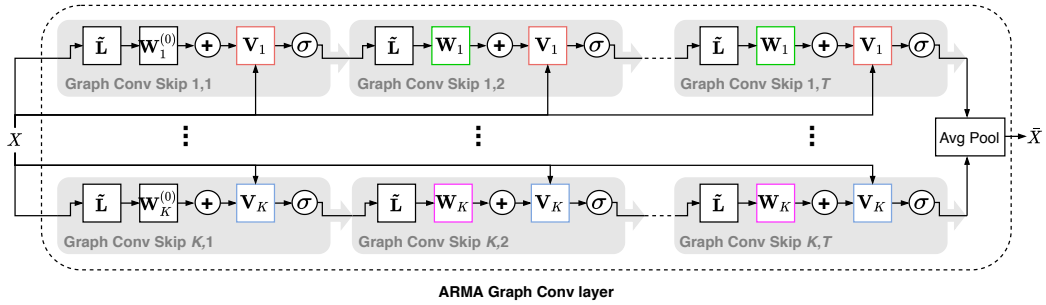


Figure 3.1. The ARMA convolutional layer.

The eigenvectors of $\mathbf{M}$ and $\mathbf{L}$ are the same, while the eigenvalues are related as follows: $\mu_m = (\lambda_{\max} - \lambda_{\min})/2 - \lambda_m$, where μ_m and λ_m represent the m^{th} eigenvalue of $\mathbf{M}$ and $\mathbf{L}$, respectively. Since $\mu_m \in [-1, 1]$, for $|a| < 1$ the first term of Equation (3.5), $(a\mathbf{M})^t$, goes to zero when $t \to \infty$, regardless of the initial point $\mathbf{X}^{(0)}$. The second term, $b\sum_{i=0}^{t}(a\mathbf{M})^i$, is a geometric series that converges to the matrix $b(\mathbf{I} - a\mathbf{M})^{-1}$, with eigenvalues $b/(1 - a\mu_m)$. It follows that the frequency response of the ARMA$_1$ filter is

$$g_{ARMA_1}(\mu_m) = \frac{b}{1 - a\mu_m}. \tag{3.6}$$

By summing K ARMA$_1$ filters, it is possible to recover the analytical form of the K^{th} order ARMA$_K$ filter. The resulting frequency response is:

$$g_{ARMA_K}(\mu_m) = \sum_{k=1}^{K} \frac{b_k}{1 - a_k\mu_m}. \tag{3.7}$$

3.2 ARMA$_1$ GNN filter

In typical graph signal processing approaches, the filter coefficients a and b in Equation (3.3) are identified with linear regression to reproduce the desired filter response, which must be provided *a priori* by the designer [92].

Here, instead, we consider a machine learning approach that does not require specifying the target response, but in which the parameters are learned end-to-end from the data by optimising a task-dependent loss function. Importantly, we also introduce non-linearities to enhance the representation capability of the model.

Specifically, we propose to approximate the effect of an ARMA$_1$ filter by iterating the following recursion:

$$\mathbf{X}^{(t+1)} = \sigma\left(\tilde{\mathbf{L}}\mathbf{X}^{(t)}\mathbf{W} + \mathbf{X}\mathbf{V}\right), \tag{3.8}$$

where $\mathbf{W}$ and $\mathbf{V}$ are learnable parameters matrices, and σ is a non-linearity. We refer to each step of Equation (3.8) as a Graph Convolutional Skip (GCS) step. The modified Laplacian matrix $\bar{\mathbf{L}}$ is obtained by setting $\lambda_{\min} = 0$ and $\lambda_{\max} = 2$ in Equation (3.4), which gives $\bar{\mathbf{L}} = \mathbf{I} - \mathbf{L}$. This is a reasonable simplification since the spectrum of $\mathbf{L}$ lies in $[0, 2]$ and the trainable parameters $\mathbf{W}$ and $\mathbf{V}$ can compensate for the small offset introduced.

Each step of Equation (3.8) is localised in the node space, as it performs a filtering operation that depends on local exchanges among neighbouring nodes and, through the *skip connection* term $\mathbf{X}^{(0)}\mathbf{V}$, also on the initial state $\mathbf{X}^{(0)}$. The computational complexity of each step is linear in the number of edges (both in time and space) since Equation (3.8) can be efficiently implemented as a sparse multiplication between $\bar{\mathbf{L}}$ and $\mathbf{X}^{(t)}$.

The neural network formulation of an ARMA$_1$ filter, which we refer to as the ARMA$_1$ *GNN* filter, is obtained by iterating Equation (3.8) until convergence, similar to the original recursive formulation of the ARMA$_1$ filter. The convergence of the iteration is guaranteed by Theorem 1.

Theorem 1. *It is sufficient that* $\|\mathbf{W}\|_2 < 1$ *and that* $\sigma(\cdot)$ *is a non-expansive map for Equation* (3.8) *to converge to a unique fixed point, regardless of the initial state* $\mathbf{X}^{(0)}$.

Proof. Let $\mathbf{X}_a^{(0)}$ and $\mathbf{X}_b^{(0)}$ be two different initial states and $\|\mathbf{W}\|_2 < 1$. After applying Equation (3.8) for $t+1$ steps, we obtain states $\mathbf{X}_a^{(t+1)}$ and $\mathbf{X}_b^{(t+1)}$. If the non-linearity $\sigma(\cdot)$ is a non-expansive map, such as the ReLU function, the following inequality holds:

$$\left\|\mathbf{X}_a^{(t+1)} - \mathbf{X}_b^{(t+1)}\right\|_2 = \tag{3.9}$$

$$= \left\|\sigma\left(\bar{\mathbf{L}}\mathbf{X}_a^{(t)}\mathbf{W} + \mathbf{X}\mathbf{V}\right) - \sigma\left(\bar{\mathbf{L}}\mathbf{X}_b^{(t)}\mathbf{W} + \mathbf{X}\mathbf{V}\right)\right\|_2 \leq \tag{3.10}$$

$$\leq \left\|\bar{\mathbf{L}}\mathbf{X}_a^{(t)}\mathbf{W} + \mathbf{X}\mathbf{V} - \bar{\mathbf{L}}\mathbf{X}_b^{(t)}\mathbf{W} - \mathbf{X}\mathbf{V}\right\|_2 = \tag{3.11}$$

$$= \left\|\bar{\mathbf{L}}\mathbf{X}_a^{(t)}\mathbf{W} - \bar{\mathbf{L}}\mathbf{X}_b^{(t)}\mathbf{W}\right\|_2 \leq \tag{3.12}$$

$$\leq \left\|\bar{\mathbf{L}}\right\|_2 \|\mathbf{W}\|_2 \left\|\mathbf{X}_a^{(t)} - \mathbf{X}_b^{(t)}\right\|_2. \tag{3.13}$$

If the non-linearity $\sigma(\cdot)$ is also a squashing function (*e.g.*, sigmoid or hyperbolic tangent), then the first inequality in (3.13) is strict.

Since the largest singular value of $\bar{\mathbf{L}}$ is ≤ 1 by definition, it follows that $\left\|\bar{\mathbf{L}}\right\|_2 \|\mathbf{W}\|_2 < 1$ and, therefore, (3.13) implies that Equation (3.8) is a contraction

mapping. The convergence to a unique fixed point and, thus, the inconsequentiality of the initial state follow by the Banach fixed-point theorem [68]. □

From Theorem 1 it follows that it is possible to choose an arbitrary $\epsilon > 0$ for which

$$\exists T_\epsilon < \infty \text{ s.t. } \left\| \mathbf{X}^{(t+1)} - \mathbf{X}^{(t)} \right\|_2 \leq \epsilon, \forall t \geq T_\epsilon. \tag{3.14}$$

Therefore, we can easily implement a stopping criterion for the iteration, which is met in finite time.

Then, by using backpropagation through time (BPTT), we can train the learnable parameters using conventional gradient descent.

3.3 ARMA$_K$ GNN filter

Similar to the original ARMA$_K$ filter, the output of the ARMA$_K$ GNN filter is obtained by combining the outputs of K ARMA$_1$ GNN filters, which we refer to as *stacks*:

$$\bar{\mathbf{X}} = \frac{1}{K} \sum_{k=1}^{K} \bar{\mathbf{X}}_{(k)}, \tag{3.15}$$

where $\bar{\mathbf{X}}_{(k)}$ is the output of the k^{th} stack at convergence of Equation (3.8):

$$\mathbf{X}_{(k)}^{(t+1)} = \sigma\left(\bar{\mathbf{L}}\mathbf{X}_{(k)}^{(t)}\mathbf{W}_{(k)} + \mathbf{X}_{(k)}\mathbf{V}_{(k)}\right) \tag{3.16}$$

$$\bar{\mathbf{X}}_{(k)} = \lim_{t\to\infty} \mathbf{X}_{(k)}^{(t)} \tag{3.17}$$

However, each k^{th} stack may require a different and possibly high number of iterations T_k to converge, depending on the initial state $\mathbf{X}_{(k)}^{(0)}$ and the random initialisation of the parameters $\mathbf{W}_{(k)}, \mathbf{V}_{(k)}$. This makes the implementation of the ARMA$_K$ GNN filter cumbersome because the computational graph is dynamic and changes every time the weight matrices are updated with gradient descent during training. Moreover, to train the parameters with BPTT, the neural network must be unfolded many times if T_k is large, introducing a high computational cost and leading to the vanishing gradient issue.

One solution is to follow the approach of *reservoir computing*, where the weight matrices of each stack are randomly initialised and left untrained [130, 63]. We note that the random weights initialisation guarantees that the K filters implement different filtering operations. To compensate for the lack of training, high-dimensional features are exploited to generate rich latent representations that disentangle the factors of variations in the data [205]. However, randomised

architectures with high-dimensional feature spaces are memory inefficient and computationally expensive at inference time.

A second approach, which we consider here, is to drop the requirement of convergence altogether and fix the number of iterations at a constant value $T_k = T, \forall k$. Even when T is small, we expect the GNN to learn a large variety of node representations thanks to the non-linearity and the trainable parameters [169]. In this way, we obtain a GNN that is easy to implement, fast to train and evaluate, and not affected by stability issues. Also, the constraint $\|\mathbf{W}\|_2 < 1$ of Theorem 1 can be relaxed by adding to the loss function an L_2 weight decay regularisation term.

Due to the limited number of iterations, however, the initial state $\mathbf{X}^{(0)}_{(k)}$ now influences the final representation $\mathbf{X}^{(T)}_{(k)}$ of each ARMA_1 filter, and so it must be selected more carefully. A natural choice is to initialise the state with $\mathbf{X}^{(0)}_{(k)} = \mathbf{1} \in \mathbb{R}^{M \times F}$ or with a linear transformation of the node features of the graph, *i.e.*, $\mathbf{X}^{(0)}_{(k)} = \mathbf{X}\mathbf{W}^{(0)}_{(k)}$, where $\mathbf{W}^{(0)}_{(k)} \in \mathbb{R}^{D_n \times D'_n}$. In our experiments, we adopted the latter initialisation so that the representation of the layer depends on the node features, like in conventional GNNs.

To encourage each filter to learn a frequency response different from the others, we apply stochastic dropout to the skip connections at every forward pass. This leads to learning a heterogeneous set of features that, when combined to form the output of the ARMA_K layer, yield powerful and expressive node representations.

Figure 3.1 shows the overall structure of the layer.

3.4 Properties and relationship with other approaches

Contrarily to GNNs that compute a purely spectral convolution through the GFT [29], ARMA GNNs do not explicitly depend on the eigendecomposition of $\mathbf{L}$, making them robust to perturbations in the underlying graph structure. For this reason, as formally proven for generic rational filters by Levie et al. [120], the proposed ARMA filters are transferable, *i.e.*, they can be applied to graphs with different topology not seen during training.

We note that the recursive formulation with shared parameters in each stack endows the GNN with a strong regularisation that helps prevent overfitting and greatly reduces the model complexity, in terms of the number of trainable parameters. In this regard, the ARMA GNN has similarities with the recurrent neural networks with residual connections used to process sequential data [227]. Indeed, the ARMA layer can naturally deal with a time-varying topology and

graph signals by replacing the constant term $\mathbf{X}^{(0)}$ in Equation (3.8) with a time-dependent input.

We also note that, since the ARMA_1 stacks are independent of each other, the computation of an ARMA_K layer can be effectively distributed across K parallel processing units.

Finally, we discuss the relationship between the proposed ARMA GNN and CayleyNet, a GNN architecture presented by Levie et al. [119] that also approximates the effect of a rational filter. Specifically, the filtering operation of a Cayley polynomial in the node space is:

$$\bar{\mathbf{X}} = w_0\mathbf{X} + 2\text{Re}\left\{\sum_{k=1}^{K} w_k(\mathbf{L}+i\mathbf{I})^k(\mathbf{L}-i\mathbf{I})^{-k}\right\}\mathbf{X}, \tag{3.18}$$

where w_k are the coefficients of the filter.

To approximate the matrix inversion in Equation (3.18) with a sequence of differentiable operations, CayleyNet adopts a fixed number T of Jacobi iterations. In practice, the Jacobi iterations approximate each term $(\mathbf{L}+i\mathbf{I})(\mathbf{L}-i\mathbf{I})^{-1}$ as a polynomial of order T with fixed coefficients. Therefore, the resulting filtering operation performed by a CayleyNet has the form:

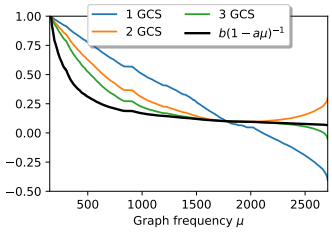
$$\bar{\mathbf{X}} \approx \sigma\left(w_0\mathbf{X} + 2\text{Re}\left\{\sum_{k=1}^{K} w_k\left(\sum_{t=1}^{T}\bar{\mathbf{L}}^t\right)^k\right\}\mathbf{X}\right), \tag{3.19}$$

where $\bar{\mathbf{L}}$ is an operator with the same sparsity pattern of $\mathbf{L}$. We note that Equations (3.18) and (3.19) slightly simplify the original formulation presented in the paper, but allow us to understand what type of operation is performed by the CayleyNet. Specifically, we see that Equation (3.19) implements a polynomial filter of order KT.

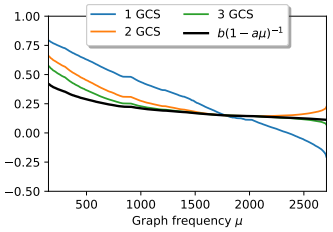
For this reason, CayleyNet shares strong similarities with the Chebyshev GNN in Equation (2.28), as it uses a (high-order) polynomial to propagate the node features on the graph for KT hops before applying the non-linearity. On the other hand, each of the K stacks in the ARMA layer propagates the current node representations $\mathbf{X}_{(k)}^{(t)}$ only for 1 hop and combines them with the initial state $\mathbf{X}_{(k)}^{(0)}$ before applying the non-linearity.

3.5 Spectral analysis of ARMA GNNs

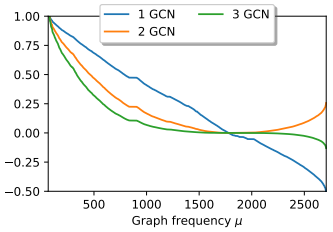
In this section, we show how the proposed ARMA layer can implement filtering operations with different frequency responses. The filter response of the ARMA



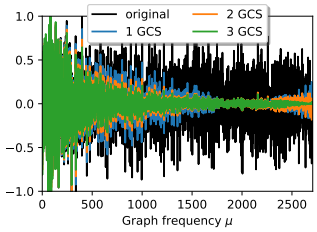
(a) $\tilde{h}$ in the 1st ARMA_1 GNN filter



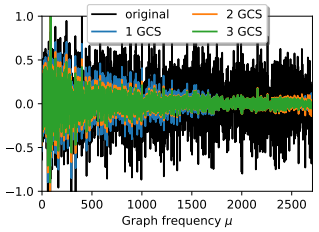
(b) $\tilde{h}$ in the 2nd ARMA_1 GNN filter



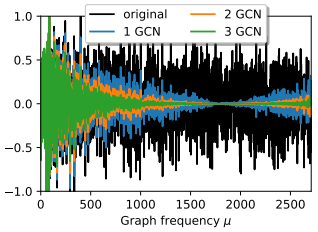
(c) $\tilde{h}$ in a GCN stack



(d) Comp. of $\mathbf{X}$ and $\bar{\mathbf{X}}$ in the 1st stack



(e) Comp. of $\mathbf{X}$ and $\bar{\mathbf{X}}$ in the 2nd stack



(f) Comp. of $\mathbf{X}$ and $\bar{\mathbf{X}}$ in a GCN stack

Figure 3.2. In (a, b), the empirical filter responses of two ARMA_1 GNN filters for $T = 1, 2, 3$; the black lines indicate the analytical response of an ARMA_1 filter with similar parameters. In (c), the empirical response of a GCN with $T = 1, 2, 3$ layers. In (d, e), the original components of the input graph signal $\mathbf{X}$ (in black), and the components of the graph signal $\bar{\mathbf{X}}$ processed by two ARMA_1 GNN filters for $T = 1, 2, 3$ (in colour). In (f), the components of $\bar{\mathbf{X}}$ processed by a GCN with $T = 1, 2, 3$ layers.

filter derived in Section 3.1 cannot be exploited to analyse our GNN formulation, due to the presence of non-linearities. Therefore, we first recall that a filter changes the components of a graph signal $\mathbf{X}$ on the eigenbasis induced by $\mathbf{L}$ (*cf.* Section 2.2.2). Let $\bar{\mathbf{X}}$ indicate a filtered graph signal so that:

$$\bar{\mathbf{X}} = \mathbf{U} g(\mathbf{\Lambda}) \mathbf{U}^\top \mathbf{X}. \tag{3.20}$$

By left-multiplying $\mathbf{U}^\top$ in Equation (3.20) we obtain

$$\mathbf{U}^\top \bar{\mathbf{X}} = g(\mathbf{\Lambda}) \mathbf{U}^\top \mathbf{X}. \tag{3.21}$$

We see that the term $\mathbf{U}^\top \bar{\mathbf{X}}$ defines how the original components $\mathbf{U}^\top \mathbf{X}$ are changed by the GNN. Therefore, we can compute numerically the unknown filter response of the ARMA layer as the ratio between $\mathbf{U}^\top \bar{\mathbf{X}}$ and $\mathbf{U}^\top \mathbf{X}$. We define the *empirical filter response* $\tilde{g}(m)$ at frequency λ_m as:

$$\tilde{g}(m) = \frac{D_n}{D'_n} \frac{\sum_{d=1}^{D'_n} \mathbf{u}_m \bar{\mathbf{X}}_{:,d}}{\sum_{d=1}^{D_n} \mathbf{u}_m \mathbf{X}_{:,d}}, \tag{3.22}$$

where $\mathbf{X}_{:,d}, \bar{\mathbf{X}}_{:,d}$ indicate the d^{th} columns of $\mathbf{X}$ and $\bar{\mathbf{X}}$.

The empirical filter response allows us to analyse the filtering implemented by an ARMA layer. We start by comparing the frequency response $g_{ARMA_1}(\mu_m)$ of an ARMA_1 filter given by Equation (3.6) with the empirical response $\tilde{g}_k(m)$ of the k^{th} stack in an ARMA_K GNN filter. To facilitate the interpretation of the results, we set the number of output features of the GNN filter to $D'_n = 1$ by letting $\mathbf{W} = a$ and $\mathbf{V} = b\mathbf{1}_{D_n}$ in Equation (3.8). Note that we are keeping the notation consistent with Equation (3.3), where a and b are the parameters of the ARMA_1 filter. In the following, we consider the graph and the node features from the Cora citation network.

Figures 3.2(a, b) show the empirical responses $\tilde{g}_1(m)$ and $\tilde{g}_2(m)$ of two different ARMA_1 GNN filters, when varying the number of propagation steps T (indicated as "t GCS" in the figures, where GCS refers to Graph Convolutional Skip step of Equation (3.8)). As T increases, $\tilde{g}_1(m)$ and $\tilde{g}_2(m)$ become more similar to the analytical responses of the ARMA_1 filters, depicted as a black line in the two figures. This supports our claim that $\tilde{g}(m)$ can approximate the unknown response of the GNN filtering operation.

Figure 3.2(d, e) show how the two filters modify the components of $\mathbf{X}$ on the Fourier basis. In particular, we depict in black the components $\mathbf{u}_m^\top \mathbf{X}$, $m = 1, \dots, M$. In colours, we depict the components $\mathbf{u}_m^\top \bar{\mathbf{X}}$, which show how much the filters affect the components associated with each frequency. The responses and

the signal components in Figures 3.2(a) and 3.2(d) are obtained for $a = 0.99$ and $b = 0.1$, while in Figures 3.2(b) and 3.2(e) for $a = 0.7$ and $b = 0.15$.

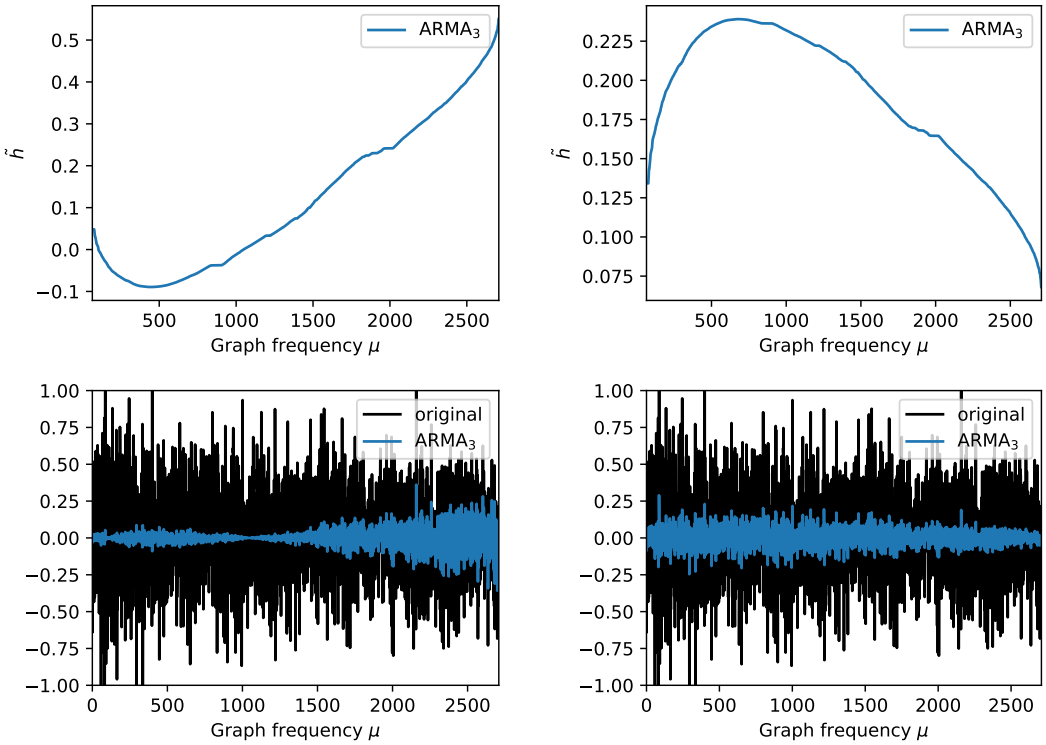
In Figure 3.2(c), we show the empirical response resulting from a stack of first-order polynomial GCNs [106]. As also highlighted in recent work [223, 134], the filtering obtained by stacking one or more of such GCNs has the undesired effect of symmetrically amplifying the lowest and also the highest frequencies of the spectrum. This is due to their filter response, which is $(1-\lambda)^T$ in the linear case and can assume negative values when T is odd. The effect is mitigated by summing $\gamma \mathbf{I}_N$ to the adjacency matrix, which adds self-loops with weight γ and shrinks the spectral domain of the graph filter. For high values of γ, GCN acts more as a low-pass filter that prevents high-frequency oscillations. This is due to the self-loops that limit the spread of information across the graph and the communication between neighbours. However, even after adding $\gamma \mathbf{I}_N$, GCN cuts almost completely the medium frequencies and amplifies the higher ones, as shown in Figure 3.2(f).

A stack of GCNs lacks flexibility in implementing different filtering operations, as the only degree of freedom to modify a GCN's response consists of manually tuning the hyperparameter γ to shrink the spectrum. On the other hand, different ARMA_1 GNN filters can generate heterogeneous responses, depending on the value of the trainable parameters in each filter. This is what provides powerful modelling capability to the proposed ARMA GNN, which can learn a large variety of filter responses that selectively shrink or amplify the Fourier components of the graph by combining K ARMA_1 layers.

Similarly to an ARMA_1 filter, each ARMA_1 GNN filter behaves as a low-pass filter that gradually dampens the Fourier components as their frequency increases. However, we recall that high-pass and band-pass filters can be obtained as a linear combination of low-pass filters [158]. To show this behaviour in practice, in Figure 3.3 we report the empirical filter responses and modified Fourier components obtained with two different ARMA_K GNN filters, for $K = 3$.

3.6 Experiments with ARMA GNNs

In this section we report some experimental results to compare the performance of GNN architectures based on our ARMA layers on four kinds of downstream tasks: node classification, graph signal classification, graph classification, and graph regression. Our experiments focus on comparing the proposed ARMA layer with GNN layers based on polynomial filters, namely Chebyshev GNNs [47] and GCN [106], and CayleyNet [119] that, like ARMA, is based on rational spec-



(a) High-pass filter **(b)** Band-pass filter

Figure 3.3. While each ARMA_1 GNN filter behaves as a low-pass filter, an ARMA layer with $K = 3$ can implement filters of different shapes. The ARMA layer in (a) implements a high-pass filtering operation that dampens low frequencies. The ARMA layer in (b) implements a band-pass filtering operation that mostly allows medium frequencies.

tral filters. As additional baselines, we also include Graph Attention Networks (GAT) [211], GraphSAGE [81], and Graph Isomorphism Networks (GIN) [234]. The comparison with these methods helps frame the proposed ARMA GNN within the current state of the art.

To ensure a fair and meaningful evaluation, we compare the performance obtained with a fixed GNN architecture, where we only change the graph convolutional layers. In particular, we fixed the GNN capacity (number of hidden units), used the same splits in each dataset, and the same training and evaluation procedures. Finally, in all experiments, we used the same order K for polynomial and rational filters, or a stack of K layers for GCN, GAT, GIN, and GraphSAGE layers. The details of every dataset considered in the experiments and the optimal hyperparameters for each model are deferred to Appendix A.

Node classification First, we consider transductive node classification on three citation networks: Cora, Citeseer, and Pubmed. The input is a single graph described by an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$, the node features $\mathbf{X} \in \mathbb{R}^{N \times D_n}$, and the labels $\mathbf{y}_l \in \mathbb{R}^{N_l}$ of a subset of nodes $\mathcal{V}_l \subset \mathcal{V}$. The targets are the labels $\mathbf{y}_u \in \mathbb{R}^{N_u}$ of the unlabelled nodes $\mathcal{V} \setminus \mathcal{V}_l$. The node features are sparse bag-of-words vectors representing text documents. The binary undirected edges in $\mathbf{A}$ indicate citation links between documents. The models are trained using 20 labels per document class and the performance is evaluated as classification accuracy on $\mathbf{y}_u$.

Secondly, we perform inductive node classification on the protein-protein interaction (PPI) network dataset. The dataset consists of 20 graphs used for training, 2 for validation, and 2 for testing. Contrarily to the transductive setting, the testing graphs (and the associated node features) are not observed during training. Additionally, each node can belong to more than one class (multi-label classification).

For the citation networks datasets, we exploit high dropout rates and L_2-norm regularisation to prevent overfitting. Table 3.1 reports the classification accuracy obtained by a GNN equipped with different graph convolutional layers.

Transductive node classification is a semi-supervised task that demands a simple model with strong regularisation to avoid overfitting the few labels available. This is the key to GCN's success compared to more complex filters, such as the Chebyshev GNN. Thanks to its flexible formulation, the proposed ARMA layer can implement the right degree of complexity and performs well on each task. On the other hand, since the PPI dataset is larger and more labels are available during training, less regularisation is required and the more complex models are advantaged. This is reflected by the better performance achieved by Chebyshev

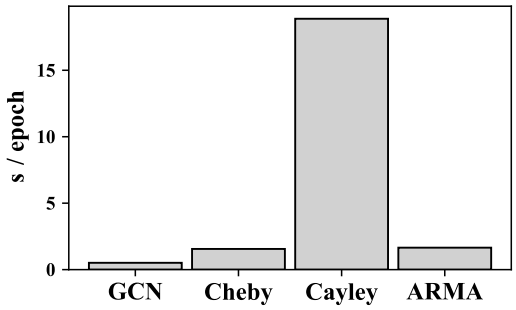


Figure 3.4. Training times on the PPI dataset.

filters and CayleyNet, compared to GCN. On PPI, ARMA significantly outperforms every other model, due to its powerful modelling capability that allows learning filter responses with different shapes. Since each layer in GAT, GraphSAGE, and GIN combines the features of a node only with those from its 1st order neighbourhood, similarly to a GCN, these architectures need to stack more layers to reach higher-order neighbourhoods and suffer from the same oversmoothing issue.

We note that the optimal depth T of the ARMA layer reported in Table A.2 is low in every dataset. We argue that a reason is the small average shortest path in the graphs (see Table A.1). Indeed, most nodes in the graphs can be reached with only a few propagation steps, which is not surprising since many real networks are small-world [218].

Figure 3.4 shows the training times of the GNN model configured with GCN, Chebyshev, CayleyNet, and ARMA layers. The ARMA layer exploits sparse operations that are linear in the number of nodes in $\mathbf{L}$ and can be trained in a time comparable to a Chebyshev filter. On the other hand, CayleyNet is slower than other methods, due to the complex formulation based on the Jacobi iterations that results in a high order polynomial.

Graph signal classification In this task, D different graph signals $\mathbf{X}_d \in \mathbb{R}^{N \times D_n}, d = 1, \dots, D$, defined on the same graph with adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$, must be mapped to labels $y_1, \dots, y_D$. We perform these experiments following the same setting of Defferrard et al. [47] for the MNIST and 20news datasets.

For the MNIST dataset, an 8-NN graph is defined on the 784 pixels of the MNIST images. To determine if node j belongs to the neighbourhood $\mathcal{N}(i)$ of node i, we compute the Euclidean distance between the 2D coordinates of pixels i and j and connect each node to its eight closest neighbours. Each graph signal is a flattened image $\mathbf{X}_d \in \mathbb{R}^{784 \times 1}$.

Table 3.1. Node classification accuracy.

Method	Cora	Citeseer	Pubmed	PPI
GAT	83.1 ±0.6	70.9 ±0.6	78.5 ±0.3	81.3 ±0.1
GraphSAGE	73.7 ±1.8	65.9 ±0.9	78.5 ±0.6	70.0 ±0.0
GIN	75.1 ±1.7	63.1 ±2.0	77.1 ±0.7	78.1 ±2.6
GCN	81.5 ±0.4	70.1 ±0.7	**79.0 ±0.5**	80.8 ±0.1
Chebyshev	79.5 ±1.2	70.1 ±0.8	74.4 ±1.1	86.4 ±0.1
CayleyNet	81.2 ±1.2	67.1 ±2.4	75.6 ±3.6	84.9 ±1.2
ARMA	**83.4 ±0.6**	**72.5 ±0.4**	78.9 ±0.3	**90.5 ±0.3**

Table 3.2. Graph signal classification accuracy.

GNN layer	MNIST	20news
GCN	98.48 ± 0.2	65.45 ± 0.2
Chebyshev	99.14 ± 0.1	68.24 ± 0.2
CayleyNet	99.18 ± 0.1	68.84 ± 0.3
ARMA	**99.20 ± 0.1**	**70.02 ± 0.1**

Table 3.2 reports the results obtained by using GCN, Chebyshev, CayleyNet, or ARMA. The results are averaged over ten runs and show that ARMA achieves a slightly higher accuracy compared to Chebyshev and CayleyNet, while the performance of GCN is significantly lower. Similarly to the PPI experiment, the larger amount of data allows more powerful architectures to achieve better performance compared to GCN.

The 20news dataset consists of 18,846 documents divided into 20 classes. Each graph signal is a document represented by a bag-of-words of the 10000 most frequent words in the corpus, embedded via Word2vec [146]. The underlying graph of 10000 nodes is a 16-NN graph similar to the MNIST dataset, with the difference that the node neighbourhoods are computed from the Euclidean distance between the embeddings vectors rather than the pixel coordinates. Again, we follow the same experimental setup as Defferrard et al. [47]. The classification accuracy reported in Table 3.2 shows that ARMA significantly outperforms every other model also on this dataset.

For this experiment, we used a particular configuration of the ARMA layer with $K = 1$ and $T = 1$ (see Table A.8), which is equivalent to a GCN with a skip connection. The skip connection allows weighting differently the contribution

of the original node features, compared to the features of the neighbours. It is important to note that, contrary to other downstream tasks, the 20news graph is generated from the similarity of word embeddings. Such an artificial graph always links an embedding vector to its first 16 neighbours. We argue that, for some words, the links might be not very relevant and using a skip connection allows weighting them less.

Similarly to the node classification datasets, the average shortest path in the 20news graph is low (see Table A.7). On the other hand, the MNIST graph has a much larger diameter, due to its regular structure with very localised connectivity. This could explain why the optimal depth T of the ARMA layer is larger for MNIST than for any other task (see Table A.8), as several steps are necessary to mix the node features on the graph.

Graph classification In this task, the d^{th} datum is a graph represented by a pair $\{\mathbf{A}_d, \mathbf{X}_d\}, d = 1, \dots D$, where $\mathbf{A}_d \in \mathbb{R}^{N_d \times N_d}$ is an adjacency matrix with N_i nodes, and $\mathbf{X}_d \in \mathbb{R}^{N_i \times D_n}$ are the node features. Each sample must be classified with a label y_i. We test the models on five different datasets. We use node degree, clustering coefficients, and node labels as additional node features. We compute the model performance with nested 10-fold cross-validation, using 10% of the training set in each fold for early stopping. Table 3.3 reports the average accuracy and also includes the results obtained by using GAT, GraphSAGE, and GIN as convolutional layers. The GNN equipped with the proposed ARMA layer achieves the highest mean accuracy compared to the polynomial filters (Chebyshev and GCN). Compared to CayleyNet, which is also based on a rational filter implementation, ARMA achieves not only a higher mean accuracy but also a lower standard deviation. These empirical results indicate that our implementation is robust and confirm the transferability of the proposed ARMA layer, discussed in Section 3.5.

Graph regression This task is similar to graph classification, with the difference that the target output y_i is now a real value, rather than a discrete class label. We consider the QM9 chemical database [170], which contains more than 130,000 molecular graphs. The nodes represent heavy atoms and the undirected edges the atomic bonds between them. Nodes have discrete attributes indicating one of four possible elements. The regression task consists of predicting a given chemical property of a molecule given its graph representation. As for graph classification, we evaluate the performance on the 80-10-10 train-validation-test splits of the nested 10-folds. We report in Table 3.4 the mean squared error

Table 3.3. Graph classification accuracy.

Method	Enzymes	Proteins	D&D	MUTAG	BHard
GAT	51.7±4.3	72.3±3.1	70.9±4.0	87.3±5.3	30.1±0.7
GraphSAGE	60.3±7.1	70.2±3.9	73.6±4.1	85.7±4.7	71.8±1.0
GIN	45.7±7.7	71.4±4.5	71.2±5.4	86.3±9.1	72.1±1.1
GCN	53.0±5.3	71.0±2.7	74.7±3.8	85.7±6.6	71.9±1.2
Chebyshev	57.9±2.6	72.1±3.5	73.7±3.7	82.6±5.2	71.3±1.2
CayleyNet	43.1±10.7	65.6±5.7	70.3±11.6	87.8±10.0	70.7±2.4
ARMA	**60.6**±7.2	**73.7**±3.4	**77.6**±2.7	**91.5**±4.2	**74.1**±0.5

Table 3.4. Graph regression mean squared error.

Property	GCN	Chebyshev	CayleyNet	ARMA
mu	0.445±0.007	0.433±0.003	0.442±0.009	**0.394**±0.005
alpha	0.141±0.016	0.171±0.008	0.118±0.005	**0.098**±0.005
HOMO	0.371±0.030	0.391±0.012	0.336±0.007	**0.326**±0.010
LUMO	0.584±0.051	0.528±0.005	0.679±0.148	**0.508**±0.011
gap	0.650±0.070	0.565±0.015	0.758±0.106	**0.552**±0.013
R2	0.132±0.005	0.294±0.022	0.185±0.043	**0.119**±0.019
ZPVE	0.349±0.022	0.358±0.001	0.555±0.174	**0.338**±0.001
U0_atom	0.064±0.003	0.126±0.017	1.493±1.414	**0.053**±0.004
Cv	0.192±0.012	0.215±0.010	0.184±0.009	**0.163**±0.007

(MSE) averaged over ten independent runs, relative to the prediction of 9 molecular properties. We see that each model achieves a very low standard deviation. One reason is the large amount of training data, which allows the GNN to learn a configuration that generalises well. Contrarily to the previous tasks, here there is no clear winner among GCN, Chebyshev, and CayleyNet, since each of them performs better than the others on some tasks. On the other hand, ARMA always achieves the lowest MSE in predicting each molecular property.

Chapter 4

Pooling

While the techniques for graph convolution have been largely studied and works like those of Gilmer et al. [65] and Battaglia et al. [14] have introduced general frameworks to unify the existing literature, less attention has been devoted to pooling layers. Only a few recent works have attempted to systematically analyse the effect of pooling in GNNs [108, 145] and, notably, a unifying formulation of pooling operators is still missing in GNN literature. In order to address these issues, this chapter contains several contributions to the subject of graph pooling.

In Section 4.1, we introduce a universal and modular framework to describe pooling operators [79]. We propose that, despite the diversity of operators found in the literature, all methods can be expressed as the composition of three functions: selection, reduction and connection. From this idea, we also derive a taxonomy of pooling operators based on four main axes: dense or sparse, trainable or non-trainable, fixed or adaptive, and hierarchical or global.

In Sections 4.2 and 4.3, we propose two new hierarchical methods for pooling graphs. The first, called MinCut [20], is a trainable, dense, and fixed operator designed to coarsen a graph according to its minimum cut partition. The second,

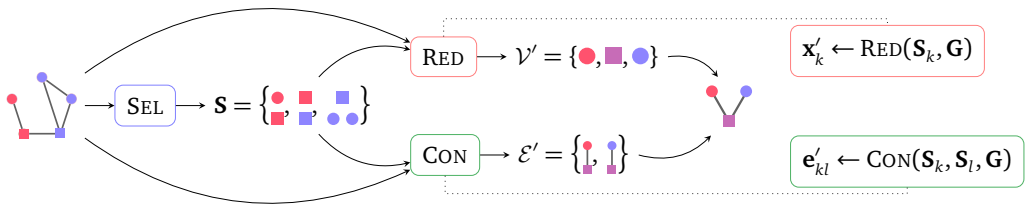


Figure 4.1. Schematic view of the SRC framework.

called Node Decimation Pooling [21], is a non-trainable, sparse, and adaptive method that performs a regular subsampling of the nodes. The method is based on using the highest-frequency eigenvector of the graph Laplacian as a sampling indicator, and is equivalent to finding the maximum cut of the graph.

Finally, in Section 4.4, we propose three ways of evaluating the performance of a graph pooling operator, moving beyond the simple benchmark-oriented approaches often found in the literature. We design experiments to test whether an operator is able to preserve 1) the node attributes, 2) the structure, and 3) the task-specific information. We compare different methods across the taxonomy and discuss guidelines and best practices for choosing an operator in practice.

4.1 Select, reduce, connect

Let a graph pooling operator be loosely defined as any function POOL that maps a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ to a new pooled graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$, with the generic goal of reducing the number of nodes from $N = |\mathcal{V}|$ to $K = |\mathcal{V}'| < N$. To understand and study graph pooling methods, it is useful to isolate the main operations that all methods must perform, regardless of their specific implementation. We identify three such operations: selection, reduction and connection (SRC); see Figure 4.1. With selection, the operator computes K subsets of nodes, each associated with one node of the output $\mathcal{G}'$; we refer to them as *supernodes*. With reduction, the operator aggregates the node attributes in each supernode to obtain the node attributes of $\mathcal{G}'$. Finally, the connection step computes edges among the K reduced nodes.

The SRC operations allow us to easily describe pooling methods, as done in Table 4.1. Accordingly, we define a pooling operator as any function

$$\text{POOL} : \mathcal{G} \mapsto \mathcal{G}' = (\mathcal{V}', \mathcal{E}') \tag{4.1}$$

written as the composition of

$$\underbrace{\mathcal{S} = \{\mathcal{S}_k\}_{k=1:K} = \text{SEL}(\mathcal{G});}_{\text{Selection}} \quad \underbrace{\mathcal{V}' = \{\text{RED}(\mathcal{G}, \mathcal{S}_k)\}_{k=1:K};}_{\text{Reduction}} \quad \underbrace{\mathcal{E}' = \{\text{CON}(\mathcal{G}, \mathcal{S}_k, \mathcal{S}_l)\}_{k,l=1:K}.}_{\text{Connection}} \tag{4.2}$$

We explain the meaning of each operation in the following paragraphs.

Select The selection function SEL maps the nodes of the input graph to the nodes of the pooled one. The role of SEL is crucial as it determines the number

of nodes in the output graph and what information from the input will be carried over to the output. A selection consists of assigning the N input nodes to K sets $\mathcal{S}_1, \ldots, \mathcal{S}_K \subseteq \mathcal{X}$, called *supernodes*: $\textsc{Sel} : \mathcal{G} \mapsto \mathcal{S} = \{\mathcal{S}_1, \ldots, \mathcal{S}_K\}$. Each supernode is a set

$$\mathcal{S}_k = \{(i, \mathbf{s}_i) \mid i \in \mathcal{V}, \mathbf{s}_i \in \mathbb{R}, \mathbf{s}_i > 0\}, \tag{4.3}$$

whose element $(i, \mathbf{s}_i)$ indicates that node i of the input graph is mapped to the k^{th} node of the pooled graph. The value $\mathbf{s}_i$ is a membership score of node i w.r.t. supernode k, *i.e.*, how much node i contributes to $\mathcal{S}_k$. In general, a node can be assigned to zero, one, or multiple supernodes, with different scores.

Reduce The reduction function computes the node attributes of graph $\mathcal{G}'$ by aggregating the node attributes of $\mathcal{G}$ selected in each supernode $\mathcal{S}_k$. A reduction consists of applying a function $\textsc{Red}$ to each supernode $\mathcal{S}_k$ to produce the k^{th} node attribute $\mathbf{x}'_k$ of $\mathcal{G}'$: $\textsc{Red} : \mathcal{G}, \mathcal{S}_k \mapsto \mathbf{x}'_k \in \mathbb{R}^{D_n}$.

Connect The connection function determines, for each pair of supernodes $\mathcal{S}_k, \mathcal{S}_l$, the presence or absence of an edge between the corresponding nodes k and l in the pooled graph. The function also computes the attributes to be assigned to new edges and reads: $\textsc{Con} : \mathcal{G}, \mathcal{S}_k, \mathcal{S}_l \mapsto \mathbf{e}'_{kl} \in \mathbb{R}^{D_e}$. We assume that the space of edge attributes contains a null attribute encoding the absence of an edge and that the edge set of a graph only contains non-null edges.

The reason why both $\textsc{Red}$ and $\textsc{Con}$ are defined as functions of graph $\mathcal{G}$ is that their output can depend on the full topology of the input graph in non-trivial ways. For example, pooling methods based on graph spectral theory often connect the nodes of $\mathcal{G}'$ based on the spectrum of the original graph $\mathcal{G}$. However, we notice that several pooling operators implement both $\textsc{Red}$ and $\textsc{Con}$ as functions of the supernodes only.

Given our definition of pooling operators as a combination of the SRC functions, we show in Table 4.1 how to express several pooling methods proposed in recent literature under the SRC formalism. We observe that the selection is commonly computed as a matrix $\mathbf{S} \in \mathbb{R}^{N \times K}$, where $\mathbf{S}_{ik}$ indicates the membership score of node i to supernode k, and $\mathbf{S}_{ik} = 0$ means that node i is not assigned to supernode k.

4.1.1 SRC as embedding operations

Since the three SRC functions are essentially node- and graph-embedding operations, we can rely on well-established theory (*e.g.*, see [180]) to study the

Table 4.1. Pooling methods in the SRC framework. GNN indicates a stack of one or more message-passing layers, MLP is a multi-layer perceptron, $\mathbf{L}$ is the normalised graph Laplacian, β is a regularisation vector (see [155]), $\mathbf{D}$ is the degree matrix, $\mathbf{u}_{max}$ is the eigenvector of the Laplacian associated with the largest eigenvalue, $\mathbf{i}$ is a vector of indices, $\mathbf{A}_{\mathbf{i},\mathbf{i}}$ selects the rows and columns of $\mathbf{A}$ according to $\mathbf{i}$.

Method	**Select**	**Reduce**	**Connect**
DiffPool [237]	$\mathbf{S} = \text{GNN}_1(\mathbf{A}, \mathbf{X})$ (w/ auxiliary loss)	$\mathbf{X}' = \mathbf{S}^\top \cdot \text{GNN}_2(\mathbf{A}, \mathbf{X})$	$\mathbf{A}' = \mathbf{S}^\top \mathbf{A}\mathbf{S}$
MinCut [20]	$\mathbf{S} = \text{MLP}(\mathbf{X})$ (w/ auxiliary loss)	$\mathbf{X}' = \mathbf{S}^\top \mathbf{X}$	$\mathbf{A}' = \mathbf{S}^\top \mathbf{A}\mathbf{S}$
NMF [7]	Factorise: $\mathbf{A} = \mathbf{W}\mathbf{H} \rightarrow \mathbf{S} = \mathbf{H}^\top$	$\mathbf{X}' = \mathbf{S}^\top \mathbf{X}$	$\mathbf{A}' = \mathbf{S}^\top \mathbf{A}\mathbf{S}$
LaPool [155]	$\begin{cases} \mathbf{V} = \lVert \mathbf{L}\mathbf{X} \rVert_d ; \\ \mathbf{i} = \{i \mid \mathbf{V}_i > \mathbf{V}_j, \forall j \in \mathcal{N}(i)\} \\ \mathbf{S} = \text{SparseMax}\left(\beta \frac{\mathbf{X}\mathbf{X}_\mathbf{i}^\top}{\lVert \mathbf{X} \rVert \lVert \mathbf{X}_\mathbf{i} \rVert}\right) \end{cases}$	$\mathbf{X}' = \mathbf{S}^\top \mathbf{X}$	$\mathbf{A}' = \mathbf{S}^\top \mathbf{A}\mathbf{S}$
Graclus [49]	$\mathcal{S}_k = \left\{ \mathbf{x}_i, \mathbf{x}_j \mid \text{ArgMax}_j \left(\frac{\mathbf{A}_{ij}}{\mathbf{D}_{ii}} + \frac{\mathbf{A}_{ij}}{\mathbf{D}_{jj}} \right) \right\}$	$\mathbf{X}' = \mathbf{S}^\top \mathbf{X}$	METIS [97]
NDP [21]	$\mathbf{i} = \{i \mid \mathbf{u}_{max,i} > 0\}$	$\mathbf{X}' = \mathbf{X}_\mathbf{i}$	Kron r. [53]
Top-K [88]	$\mathbf{y} = \frac{\mathbf{X}\mathbf{p}}{\lVert \mathbf{p} \rVert}$; $\mathbf{i} = \text{top}_K(\mathbf{y})$	$\mathbf{X}' = (\mathbf{X} \odot \sigma(\mathbf{y}))_\mathbf{i}$;	$\mathbf{A}' = \mathbf{A}_{\mathbf{i},\mathbf{i}}$
SAGPool [117]	$\mathbf{y} = \text{GNN}(\mathbf{A}, \mathbf{X})$; $\mathbf{i} = \text{top}_K(\mathbf{y})$	$\mathbf{X}' = (\mathbf{X} \odot \sigma(\mathbf{y}))_\mathbf{i}$;	$\mathbf{A}' = \mathbf{A}_{\mathbf{i},\mathbf{i}}$

expressive power of pooling operators when formulated under the SRC framework.

Consider a graph space defined on compact node and edge attribute sets $\mathbb{R}^{D_n}, \mathbb{R}^{D_e}$, and let $K(\mathcal{G})$ represent the number of nodes of $\mathcal{G}' = \textsc{Pool}(\mathcal{G})$, where $K(\mathcal{G}) \leq \bar{K}$ for all $\mathcal{G}$ and for some finite $\bar{K} \in \mathbb{N}$, $\bar{K} > 0$. By representing the output of the selection function as a matrix $\mathbf{S} \in \mathbb{R}^{N \times \bar{K}}$, we can then interpret Sel as permutation-equivariant node embedding operation $\mathbf{x}_i \mapsto \mathbf{S}_{i,:}$, from the space of node attributes to the space of supernodes assignments $\mathbb{R}^{\bar{K}}$ where we assumed, without loss of generality, that $\mathbf{S}_{ik} = 0$ for all $k > K(\mathcal{G})$ (this is necessary to ensure that any number of nodes $K(\mathcal{G})$ can be computed by Pool). Now, let $\mathcal{G}_{\mathcal{S}_k}$ indicate an augmentation of $\mathcal{G}$ such that its node features are $\mathbf{X}_{\mathcal{S}_k} = \mathbf{X} \| \mathbf{S}_{:,k}$, where $\|$ indicates concatenation, and similarly $\mathcal{G}_{(\mathcal{S}_k, \mathcal{S}_l)}$ is defined by $\mathbf{X}_{(\mathcal{S}_k, \mathcal{S}_l)} = \mathbf{X} \| \mathbf{S}_{:,k} \| \mathbf{S}_{:,l}$. It is immediate to see that the augmented graphs are an equivalent way of representing the inputs of Red and Con, which can then be seen as graph-embedding operations of the form:

$$\textsc{Red} : \mathcal{G}_{\mathcal{S}_k} \mapsto \mathbf{x}'_k; \;\; \textsc{Con} : \mathcal{G}_{(\mathcal{S}_k, \mathcal{S}_l)} \mapsto \mathbf{e}'_{kl}. \tag{4.4}$$

An interesting consequence of this interpretation of SRC is that by implementing Sel as a universal equivariant network [98], and Red and Con as universal invariant ones [140], the resulting operator is a universal approximator for any arbitrary choice of pooling with continuous Sel, Red and Con.

4.1.2 Taxonomy of graph pooling

The SRC framework is a general template to describe pooling operators and it allows us to characterise the different families of pooling methods found in the literature. We propose the following taxonomy of pooling operators based on four distinguishing characteristics and we show in Table 4.2 how existing pooling methods fit into this taxonomy.

Trainability A primary distinction among pooling operators is whether Sel, Red, and Con are learned end-to-end as part of the overall GNN architecture. In this case, we say that a method is *trainable*, *i.e.*, the operator has parameters which are learned by optimising a task-driven loss function, while in all other cases we say that methods are *non-trainable*. This distinction is important because, while non-trainable methods are often used as stand-alone algorithms for graph coarsening, trainable methods were specifically designed for GNNs and are a novel research topic of their own.

Generally, non-trainable methods are useful when there is strong prior information about the desired behaviour of pooling (*e.g.*, preserving connectivity [49] or filtering out some particular graph frequencies [155]). These prior assumptions are usually grounded on graph-theoretical properties and are useful when few data are available, since they do not increase the overall number of parameters and introduce no additional optimisation objectives when training the GNN. A well-known example of non-trainable pooling is the conventional grid pooling of CNNs, which pools spatially localised groups of pixels. On the other hand, trainable methods are more flexible and make fewer assumptions about the desired result. Therefore, they are useful in problems where the best pooling strategy is not known *a priori*. However, note that it is possible to integrate priors about the desired pooling behaviour also in trainable methods. These additional assumptions usually act as a regularisation.

Density of the supernodes A second axis of the taxonomy is concerned with the size of supernodes and the consequent cost of computing the selection function. We define the *density* of a pooling operator as the expected value $\mathbb{E}\left[|\mathcal{S}_k|/N\right]$ of the ratio between the cardinality of a supernode $\mathcal{S}_k$ and the number of nodes in $\mathcal{G}$. We say that a method is *dense* if SEL generates supernodes $\mathcal{S}_k$ whose cardinality is $\Theta(N)$, and *sparse* if supernodes have constant cardinality $O(1)$.[1] Figure 4.2a shows an example of sparse and dense selection.

This distinction is key, since sparse methods require much less computational resources, especially memory, which is a significant bottleneck even in modern GPUs. This makes them scale better to large graphs. However, as we will show in Section 4.4, sparse selection is a harder operation to learn than dense selection and may result in unexpected behaviours.

Adaptability of K It is also possible to distinguish pooling methods according to the number of nodes K of the pooled graph. If K is constant and independent from the input graph size, we say that a pooling method is *fixed*. In this case, K is a hyperparameter of the pooling operator and the output graph will always have K nodes. For example, K can be the number of output features of a neural network used to compute cluster assignments [20, 237]. On the other hand, if the number of supernodes is a function $K(\mathcal{G})$ of the input graph we say that the method is *adaptive*. In many cases, $K(\mathcal{G})$ is a function of N (*e.g.*, the ratio

[1] Intermediate situations—*e.g.*, $\Theta(\log N)$—are also possible, although here we focus on the two limit cases of $\Theta(N)$ and $O(1)$.

Table 4.2. Taxonomy of pooling operators. Methods are divided in trainable or non-trainable (**T / nT**), dense or sparse (**D / S**), fixed or adaptive (**F / A**), and hierarchical or global (**H / G**).

Method	T	nT	D	S	F	A	H	G
DiffPool [237], MinCut [20]	✓		✓		✓		✓	
Top-K methods [88, 32, 117, 108, 171], Edge Contract. [50]	✓			✓		✓	✓	
Coates and Ng [37], Voxelization-based [195, 174, 168, 118]		✓	✓			✓	✓	
NMF [7], EigenPooling [133], LaPool [155], Clique [132]		✓	✓			✓	✓	
Xie *et al.* [232], MPR [23]		✓	✓			✓	✓	
Graclus [49], NDP [21], Pooling in CNNs		✓		✓		✓	✓	
[124, 213, 151]	✓		✓		✓			✓
[248, 224, 39, 6, 234, 9]		✓	✓		✓			✓
Scarselli *et al.* [182]		✓		✓	✓			✓

$N/2$), but $K(\mathcal{G})$ could also depend on the input graph in a more complex way (*e.g.*, [155, 108]).

Adaptive pooling methods can compute graphs that have a size proportional to that of the input. On the other hand, all the coarsened graphs generated by fixed methods will have the same size. This can lead to situations where $K > N$ for some graphs, causing them to be upscaled by pooling, rather than coarsened. Figure 4.2b compares fixed and adaptive pooling and shows an example (2^{nd} row) where fixed pooling upscales the graph. For data with a wide or skewed distribution of the number of nodes, the values commonly chosen for K in fixed methods, like the average number of nodes in the training set, may cause small graphs to be upscaled and big graphs to be excessively shrunk. Therefore, if the relative graph size is important for solving a particular task, adaptive methods should be preferred.

Hierarchy A distinction often found in the literature is that between "regular" and global pooling, which is evident to the point where global pooling is usually referred to as a separate operation called readout (*cf.* Section 2.3.3). Here we show that this distinction can be formalised with the SRC framework. Specifically, a graph readout is a pooling method that reduces a graph to a single node discarding all topological information, *i.e.*, it is fixed with $K = 1$. Also, the connection function is a constant map to the empty set. On the other hand, we indicate all other methods as *hierarchical* pooling operators.

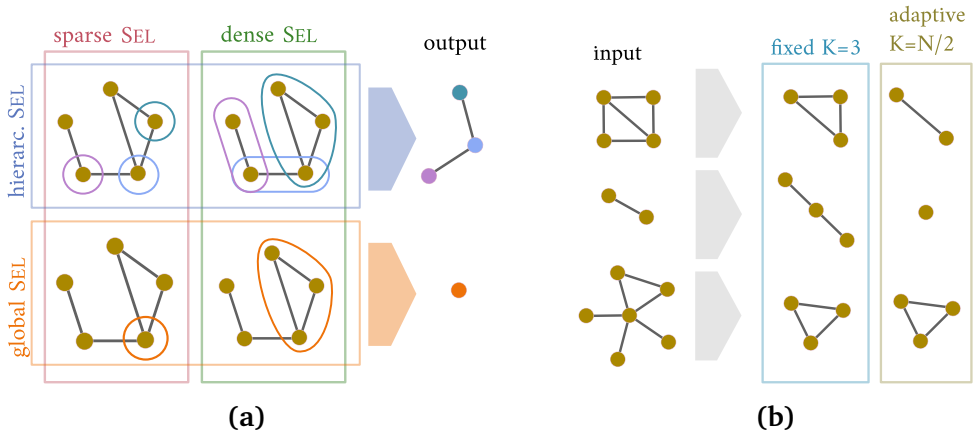


Figure 4.2. **(a)** Sparse supernodes have a constant cardinality ($|\mathcal{S}_k| = 1$) while dense supernodes scale with the size of the graph. Hierarchical methods reduce the graph gradually, while global methods always return one node. **(b)** Fixed methods return the same number of nodes ($K = 3$) while adaptive methods return graphs of size proportional to the input.

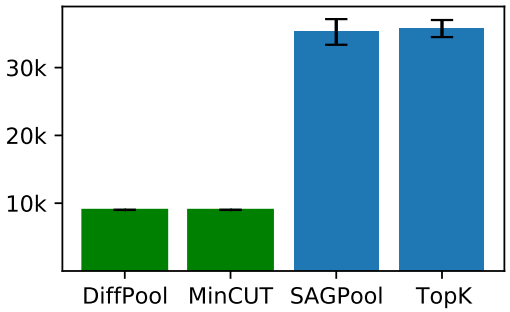


Figure 4.3. Maximum number of nodes that can be processed by dense (in green) and sparse (in blue) methods.

Discussion The main differences among pooling methods are in the selection function, while much less variety is found in the reduction and connection functions. A majority of methods (*e.g.*, [195, 168, 7, 132, 133, 155]—*cf.* Table 4.2) have adaptive K, with a dense and non-trainable selection. Adaptive methods are the most commonly found in the literature, although fixed pooling operators are currently the state of the art [237, 20]. We also note that, to the best of our knowledge, there are no pooling operators that are trainable, dense, and adaptive, which could be an interesting research topic in the near future.

Considering density and adaptability, we see that the memory cost of a pooling operator can range from $O(1)$ (sparse and fixed) to $O(N^2)$ (dense and adaptive). This is especially relevant for trainable methods, which usually need to fit into memory-bound computational units like GPUs and TPUs. Figure 4.3 shows the maximum number of nodes that can be processed by some representative pooling methods, without causing a GPU-out-of-memory exception (we provide details in Appendix C). As expected, sparse methods can pool graphs up to four times bigger than dense ones.

Having introduced a general framework for describing pooling operators, in the following sections we will present our further contributions to the topic of pooling operators for GNNs, namely two pooling techniques called *MinCut* and *Node Decimation*.

4.2 MinCut pooling

The contents of this section were adapted from "Spectral clustering with graph neural networks for graph pooling," Bianchi et al. [20].

Spectral clustering (SC) is a well-known clustering technique that uses the low-frequency eigenvectors of the Laplacian to find strongly connected communities on a graph. SC has been used to perform pooling in GNNs [29, 47] although 1) the approaches based on this technique cannot explicitly account for the node attributes and, 2) the eigendecomposition of the Laplacian is a non-differentiable and expensive operation, making it unpractical to use in large or complex architectures.

In this section, we present a clustering technique that addresses these limitations of SC. Specifically, we formulate a continuous relaxation of the normalised minimum cut problem and train a GNN to compute cluster assignments by optimising this objective. Our approach learns the solution found by SC while also

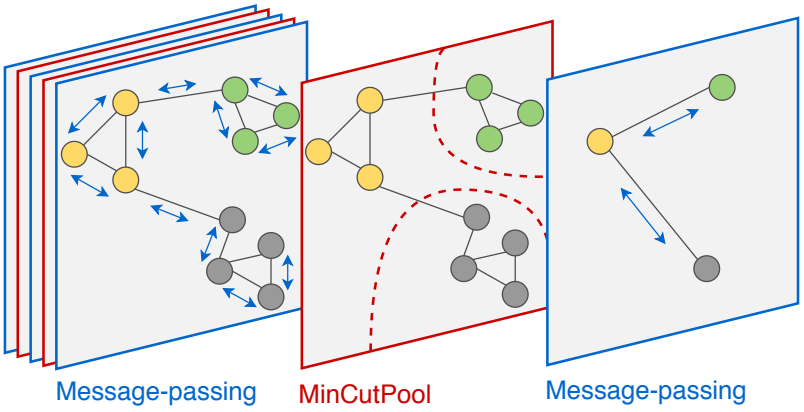


Figure 4.4. A deep GNN architecture alternating message-passing and the MinCut layer.

accounting explicitly for the node features to identify clusters. At the same time, our GNN-based implementation is differentiable and does not require computing the expensive eigendecomposition of the Laplacian, since it exploits spatially localised graph convolutions that are fast to compute.

Using the proposed clustering method to compute the selection function (*cf.* Section 4.1), we derive a trainable pooling operator called *MinCut*, which combines the flexibility of trainable operators with the inductive biases of non-trainable methods (in this case, SC). The parameters in a MinCut layer are learned by minimising the minimum cut objective, which can be jointly optimised alongside a task-specific loss. In the latter case, the minimum cut loss acts as a regularisation term, which prevents unwanted solutions, and the GNN can find the optimal trade-off between task-specific and clustering objectives. Because they are fully differentiable, MinCut layers can be stacked at different depths of a GNN to obtain a hierarchical representation and the overall architecture can be trained end-to-end (Figure 4.4).

In the following sections, we first introduce the minimum cut problem and its relation to SC, and then we present the proposed MinCut layer. We conclude this section with some experiments designed to test the performance of MinCut on some clustering tasks. We will show additional experimental analysis with the MinCut layer in Section 4.4.

4.2.1 Minimum cut and spectral clustering

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with adjacency matrix $\mathbf{A}$, the *K-way normalised minimum cut* problem is the task of partitioning $\mathcal{V}$ in K disjoint subsets of nodes

$\mathcal{V}_k$, $k = 1,\dots,K$, by removing the minimum volume of edges. The problem is equivalent to maximising

$$\frac{1}{K}\sum_{k=1}^{K}\frac{\sum_{i,j\in\mathcal{V}_k}\mathbf{a}_{i,j}}{\sum_{i\in\mathcal{V}_k,j\in\mathcal{V}\setminus\mathcal{V}_k}\mathbf{a}_{i,j}}, \tag{4.5}$$

where the numerator counts the edge volume within each cluster, and the denominator counts the edges between the nodes in a cluster and the rest of the graph [192]. Let $\mathbf{C}\in\{0,1\}^{N\times K}$ be a *cluster assignment matrix*, so that $\mathbf{c}_{ij}=1$ if node i belongs to cluster j, and 0 otherwise. The problem can be expressed as [48]:

$$\begin{aligned}&\text{maximise } \frac{1}{K}\sum_{k=1}^{K}\frac{\mathbf{C}_{:,k}^{\top}\mathbf{A}\mathbf{C}_{:,k}}{\mathbf{C}_{:,k}^{\top}\mathbf{D}\mathbf{C}_{:,k}},\\ &\text{s.t. } \mathbf{C}\in\{0,1\}^{N\times K},\ \mathbf{C}\mathbf{1}_K=\mathbf{1}_N\end{aligned} \tag{4.6}$$

where $\mathbf{C}_{:,k}$ is the k^{th} column of $\mathbf{C}$. Since problem (4.6) is NP-hard, it is recast in a relaxed continuous formulation that can be solved in polynomial time and guarantees a near-optimal solution [201]:

$$\begin{aligned}&\underset{\mathbf{Q}\in\mathbb{R}^{N\times K}}{\text{ArgMax}}\ \frac{1}{K}\sum_{k=1}^{K}\mathbf{Q}_k^{\top}\mathbf{A}\mathbf{Q}_k\\ &\text{s.t. } \mathbf{Q}=\mathbf{D}^{\frac{1}{2}}\mathbf{C}(\mathbf{C}^{\top}\mathbf{D}\mathbf{C})^{-\frac{1}{2}},\ \mathbf{Q}^{\top}\mathbf{Q}=\mathbf{I}_K.\end{aligned} \tag{4.7}$$

While problem (4.7) is still non-convex, there exists an optimal solution $\mathbf{Q}^*=\mathbf{U}_K\mathbf{O}$, where $\mathbf{U}_K\in\mathbb{R}^{N\times K}$ contains the eigenvectors of $\mathbf{A}$ corresponding to the K smallest eigenvalues, and $\mathbf{O}\in\mathbb{R}^{K\times K}$ is an orthogonal transformation [90].

SC obtains the cluster assignments by applying K-means clustering to the rows of $\mathbf{Q}^*$ [214]. One of the main limitations of SC lies in the computation of the spectrum of $\mathbf{A}$ to obtain $\mathbf{Q}^*$, which has a memory complexity of $O(N^2)$ and a computational complexity of $O(N^3)$. This prevents its applicability to large datasets.

To deal with the scalability issues of SC, the constrained optimisation in (4.7) can be solved by gradient descent algorithms that refine the solution by iterating operations whose individual complexity is $O(N^2)$, or even $O(N)$ [82]. These algorithms search the solution on the manifold induced by the orthogonality constraint on the columns of $\mathbf{Q}$, by performing gradient updates along the geodesics [220, 38]. Alternative approaches rely on QR factorisation to constrain the space of feasible solutions [42], and alleviate the cost $O(N^3)$ of the factorisation by ensuring that orthogonality holds only on one mini-batch at a time [189]. Dhillon et al.

[49] discuss the equivalence between graph clustering objectives and the kernel k-means algorithm, and their Graclus algorithm is a popular model-free method for hierarchical pooling in GNNs [47].

To learn a model that finds an approximate SC solution also for out-of-sample graphs, several works propose to use neural networks. Tian et al. [204] train an autoencoder to map the i^{th} row of the Laplacian to the i^{th} components of the first K eigenvectors. Yi et al. [236] define an orthogonality constraint to learn spectral embeddings as a volumetric reparametrisation of a precomputed Laplacian eigenbasis. Finally, Shaham et al. [189] propose a loss function to cluster generic data and process out-of-sample data at inference time. While these approaches learn to embed data in the Laplacian eigenspace of the given graph, they rely on non-differentiable operations to compute the cluster assignments and, therefore, are not suitable to perform pooling in a GNN trained end-to-end.

4.2.2 Spectral clustering with GNNs

We propose a GNN-based approach that addresses the aforementioned limitations of SC algorithms. Our method clusters the nodes according to the graph topology, *i.e.*, by assuming that nodes in the same cluster should be strongly connected, and to the node features, *i.e.*, by assuming that nodes in the same cluster should have similar features. Our method assumes that node features represent a good initialisation for computing the cluster assignments. This is a realistic assumption due to the homophily property of many real-world networks [143]. Additionally, even in disassortative networks (*i.e.*, networks where dissimilar nodes are likely to be connected [153]), the features of nodes in strongly connected communities tend to become similar due to the smoothing effect of convolution.

Let $\mathbf{X}'$ be the matrix of node representations yielded by one or more convolutional layers. We compute a cluster assignment of the nodes using a multi-layer perceptron (MLP) with softmax activation in the output layer, which maps each node feature $\mathbf{x}'_i$ into the i^{th} row of a soft cluster assignment matrix $\mathbf{S}$:

$$\begin{aligned} \mathbf{X}' &= \text{GNN}(\mathbf{X}, \mathbf{A}; \boldsymbol{\Theta}_{\text{GNN}}) \\ \mathbf{S} &= \text{MLP}(\mathbf{X}'; \boldsymbol{\Theta}_{\text{MLP}}), \end{aligned} \tag{4.8}$$

where $\boldsymbol{\Theta}_{\text{GNN}}$ and $\boldsymbol{\Theta}_{\text{MLP}}$ are trainable parameters. The softmax activation of the MLP guarantees that $\mathbf{s}_{ij} \in [0, 1]$ and enforces the constraints $\mathbf{S}\mathbf{1}_K = \mathbf{1}_N$ inherited from the optimisation problem in (4.6). We note that it is possible to add a temperature parameter to the softmax activation of the MLP to control how much $\mathbf{S}_{i,:}$ should be close to a one-hot vector, *i.e.*, the level of fuzziness in the cluster assignments.

The parameters $\mathbf{\Theta}_{\text{GNN}}$ and $\mathbf{\Theta}_{\text{MLP}}$ are jointly optimised by minimising an unsupervised loss function $\mathcal{L}_u$ composed of two terms, which approximates the relaxed formulation of the minimum cut problem:

$$\mathcal{L}_u = \mathcal{L}_c + \mathcal{L}_o = \underbrace{-\frac{Tr(\mathbf{S}^\top \mathbf{A}\mathbf{S})}{Tr(\mathbf{S}^\top \mathbf{D}\mathbf{S})}}_{\mathcal{L}_c} + \underbrace{\left\| \frac{\mathbf{S}^\top \mathbf{S}}{\|\mathbf{S}^\top \mathbf{S}\|_F} - \frac{\mathbf{I}_K}{\sqrt{K}} \right\|_F}_{\mathcal{L}_o}, \tag{4.9}$$

where $\|\cdot\|_F$ indicates the Frobenius norm and $\mathbf{D}$ is the degree matrix of $\mathbf{A}$.

The *cut loss* $\mathcal{L}_c$ evaluates the cut given by the soft cluster assignment $\mathbf{S}$ and is bounded by $-1 \leq \mathcal{L}_c \leq 0$. Minimising $\mathcal{L}_c$ encourages strongly connected nodes to be clustered together, since the inner product $\langle \mathbf{S}_{i,:}, \mathbf{S}_{j,:} \rangle$ increases when $\mathbf{a}_{i,j}$ is large. $\mathcal{L}_c$ has a single maximum, reached when the numerator $Tr(\mathbf{S}^\top \mathbf{A}\mathbf{S}) = \frac{1}{K}\sum_{k=1}^{K} \mathbf{S}_{:,k}^\top \mathbf{A}\mathbf{S}_{:,k} = 0$. This occurs if, for each pair of connected nodes (*i.e.*, $\mathbf{a}_{i,j} > 0$), the cluster assignments are orthogonal (*i.e.*, $\langle \mathbf{S}_{i,:}, \mathbf{S}_{j,:} \rangle = 0$). $\mathcal{L}_c$ reaches its minimum, -1, when $Tr(\mathbf{S}^\top \mathbf{A}\mathbf{S}) = Tr(\mathbf{S}^\top \mathbf{D}\mathbf{S})$. This occurs when, in a graph with K disconnected components, the cluster assignments are equal for all the nodes in the same component and orthogonal to the cluster assignments of nodes in different components. However, $\mathcal{L}_c$ is a non-convex function and its minimisation can lead to unwanted solutions. For example, given a connected graph, a trivial—yet optimal—solution is the one that assigns all nodes to the same cluster. As a consequence of the continuous relaxation, another bad minimum occurs when the cluster assignments are all uniform, that is, all nodes are equally assigned to all clusters. This problem is exacerbated by the convolutions, whose smoothing effect makes the node features more uniform.

To penalise the bad minima of $\mathcal{L}_c$, the *orthogonality loss* term $\mathcal{L}_o$ encourages the cluster assignments to be orthogonal and the clusters to be of similar size. Since the two matrices in $\mathcal{L}_o$ have unit norm, it is easy to see that $0 \leq \mathcal{L}_o \leq 2$. Therefore, $\mathcal{L}_o$ is commensurable to $\mathcal{L}_c$ and the two terms can be summed without rescaling them (see Figure 4.8 for an example). $\mathbf{I}_K$ can be interpreted as a rescaled clustering matrix $\mathbf{I}_K = \hat{\mathbf{S}}^\top \hat{\mathbf{S}}$, where $\hat{\mathbf{S}}$ assigns exactly N/K points to each cluster. The value of the Frobenius norm between clustering matrices is not biased by differences in the size of the clusters [114] and, thus, can be used to optimise intra-cluster variance.

While traditional SC requires computing the spectral decomposition for every new sample, here the cluster assignments are computed by a neural network that learns a mapping from the node feature space to the clusters assignment space. Since the neural network parameters are independent of the graph size, and since the convolutions in the GNN are localised in the node space and independent from the spectrum of the Laplacian, the proposed clustering approach generalises

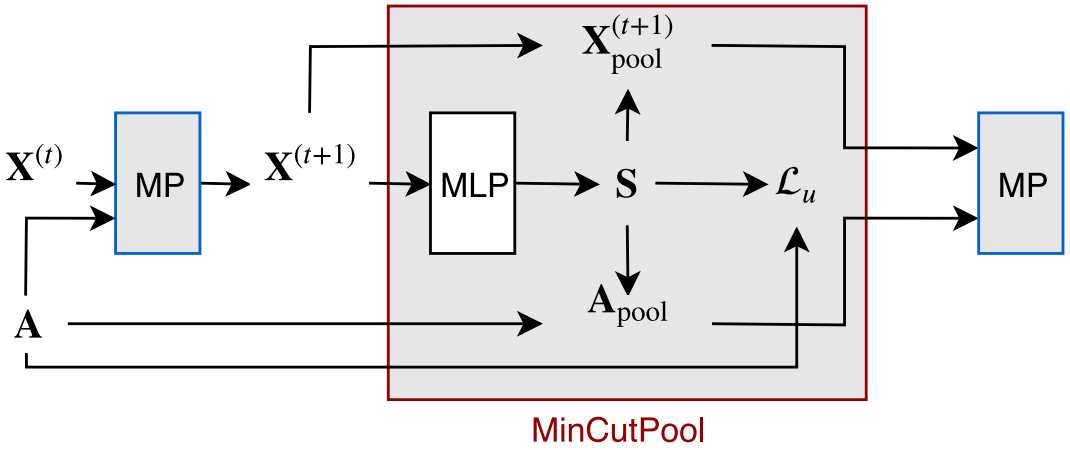


Figure 4.5. Schema of the MinCut layer.

to unseen graphs at inference time. This also gives the opportunity of training our network on small graphs and then using it to cluster larger ones.

4.2.3 Pooling and graph coarsening

The methodology proposed in Section 4.2.2 is a general technique that can be used to solve clustering tasks on any data represented by graphs. In this section, we show how to use it to perform pooling in GNNs and we introduce the MinCut layer, which uses the cluster assignment matrix $\mathbf{S}$ in Equation (4.8) as a selection operator (*cf.* Section 4.1). Figure 4.5 depicts a scheme of the MinCut layer.

The reduction and connection are computed, respectively, as:

$$\mathbf{X}' = \mathbf{S}^\top \mathbf{X} \tag{4.10}$$

$$\mathbf{A}' = \mathbf{S}^\top \mathbf{A}\mathbf{S}, \tag{4.11}$$

i.e., the reduction is a feature-wise weighted sum of the node attributes in each supernode and the connection is a weighted sum of the edges connecting two supernodes. The implementation of MinCut in the SRC framework is summarised in Table 4.1.

Note that $\mathbf{A}'$ is the matrix whose trace is maximised in $\mathcal{L}_c$ (Equation (4.9)). Therefore, $\mathbf{A}'$ will be a diagonal-dominant matrix that describes a graph with self-loops much stronger than any other connection. Since very strong self-loops hamper the propagation across adjacent nodes in the convolutions following the pooling layer, we compute the new adjacency matrix $\mathbf{A}'$ by zeroing the diagonal

and applying degree normalisation

$$\hat{\mathbf{A}} \leftarrow \mathbf{A}' - \mathbf{I}_K \odot \mathbf{A}'; \tag{4.12}$$

$$\mathbf{A}' \leftarrow \hat{\mathbf{D}}^{-\frac{1}{2}} \hat{\mathbf{A}} \hat{\mathbf{D}}^{-\frac{1}{2}}. \tag{4.13}$$

where $\hat{\mathbf{D}}$ is the degree matrix of $\hat{\mathbf{A}}$.

Because our GNN-based implementation of SC is fully differentiable, MinCut layers can be used to build deep GNNs that hierarchically coarsen the graph representation. The parameters of each MinCut layer can then be learned end-to-end, by jointly optimising $\mathcal{L}_u$ along with any supervised loss for a particular downstream task. Contrarily to SC methods that search for feasible solutions only within the space of orthogonal matrices, $\mathcal{L}_o$ only introduces a soft constraint that can be partially violated during the learning procedure. This allows the GNN to find the best trade-off between $\mathcal{L}_u$ and the supervised loss and makes it possible to handle graphs with intrinsically imbalanced clusters. Since $\mathcal{L}_c$ is non-convex, the violation of the orthogonality constraint could compromise the convergence to the global optimum of the minimum cut objective. However, we note that:

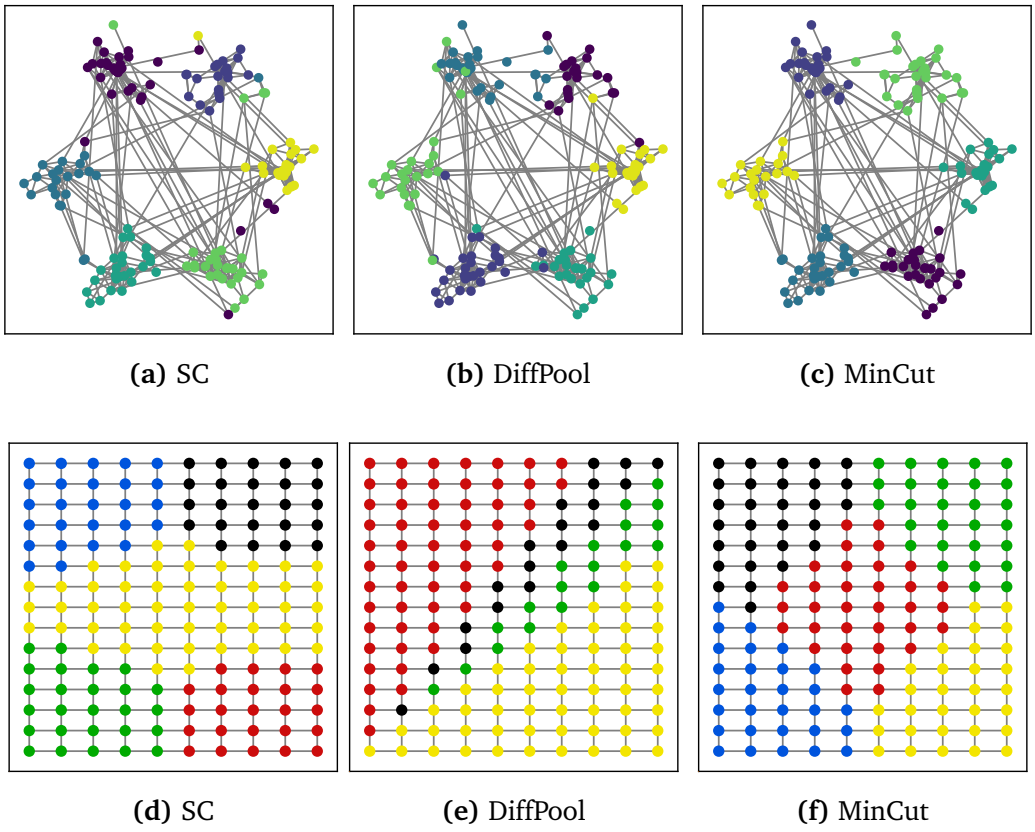
1. Since MinCut computes the supernode assignments from node features that become similar due to convolutional operations, the supernodes are likely to contain nodes that are both strongly connected and with similar features, reducing the risk of finding an unwanted solution;

2. The bad minima of $\mathcal{L}_c$ lead to discarding most of the information from the input graph and, therefore, optimising the task-specific loss encourages the GNN to avoid them;

3. Since the minimum cut objective acts mostly as a regularisation term, a solution that is sub-optimal for (4.7) could instead be preferable for the supervised downstream task;

To support these comments, we show in Section 4.2.4 that MinCut never yields unwanted solutions in practice and consistently achieves good performance on a variety of tasks.

Computational complexity The space complexity of the MinCut layer is $O(NK)$, as it depends on the dimension of the cluster assignment matrix $\mathbf{S} \in \mathbb{R}^{N \times K}$. The computational complexity is dominated by the numerator in the term $\mathcal{L}_c$, and is $O(N^2K + NK^2) = O(NK(K+N))$. Since $\mathbf{A}$ is usually sparse, we can exploit operations for sparse tensors and reduce the complexity of the first matrix multiplication to $O(|\mathcal{E}|K)$. Since the sparse multiplication yields a dense matrix, the second multiplication still costs $O(NK^2)$ and the total cost is $O(K(|\mathcal{E}| + NK))$.



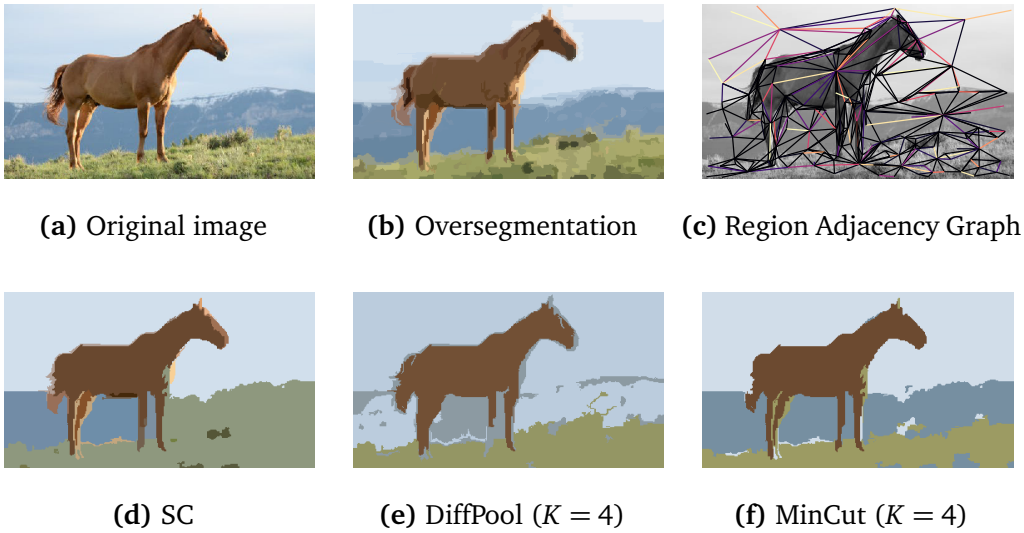
(a) SC **(b)** DiffPool **(c)** MinCut

(d) SC **(e)** DiffPool **(f)** MinCut

Figure 4.6. Node clustering on a community network (K=6) and on a grid graph (K=5).

4.2.4 Experiments with MinCut

In this section, we evaluate the performance of MinCut as a clustering technique (we perform a more in-depth analysis of pooling operators for GNNs in Section 4.4). We implement a one-layer GNN followed by a single-layer MLP to compute $\mathbf{S}$, and train the overall architecture by minimising $\mathcal{L}_u$. For comparison, we configure a similar architecture based on DiffPool where we optimise the auxiliary DiffPool losses (*cf.* Section 2.3.2 and Table 4.1) without any additional supervised loss. We also consider the clusters found by the conventional non-trainable SC. In the results, our approach is always indicated as MinCut for simplicity, although this experiment only focuses on the clustering results (*i.e.*, the selection) and does not involve the reduction and connection steps. A similar consideration holds for DiffPool.



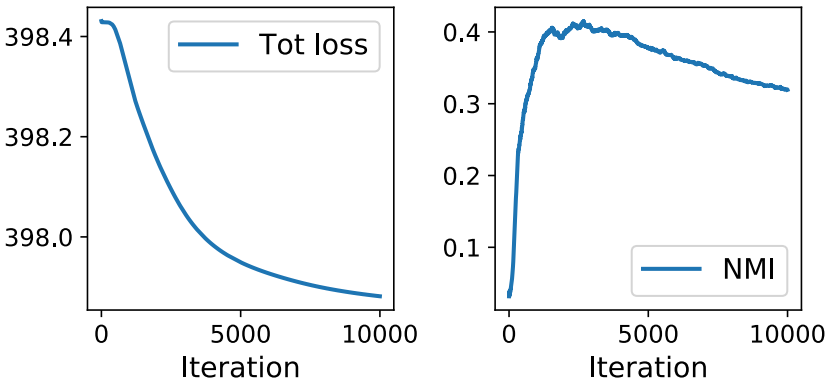
(a) Original image **(b)** Oversegmentation **(c)** Region Adjacency Graph

(d) SC **(e)** DiffPool ($K = 4$) **(f)** MinCut ($K = 4$)

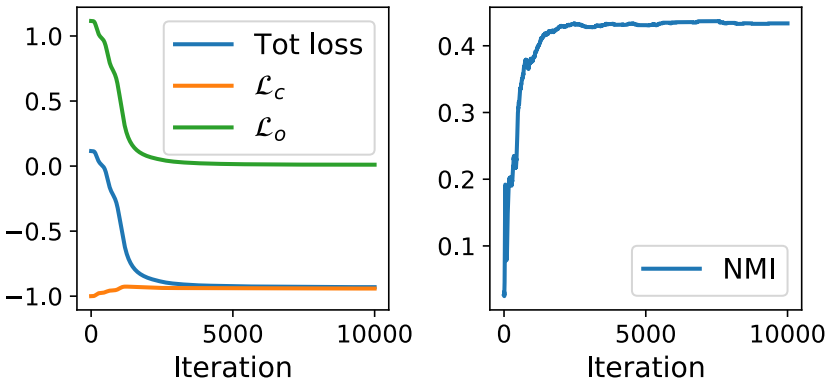
Figure 4.7. Image segmentation by clustering the nodes of the Region Adjacency Graph.

Clustering on synthetic networks We consider two simple graphs: the first is a network with six communities and the second is a regular grid. The adjacency matrix $\mathbf{A}$ is binary and the features $\mathbf{X}$ are the 2-dimensional node coordinates. Figure 4.6 depicts the node partitions generated by SC (a, d), DiffPool (b, e), and MinCut (c, f). MinCut generates accurate and balanced partitions, demonstrating that the cluster assignment matrix $\mathbf{S}$ is well-formed. In particular, we note that, because it uses node features, MinCut computes a different clustering than SC and achieves an overall better performance. On the other hand, DiffPool assigns some nodes to the wrong community in the first example and produces an unbalanced partition of the grid.

Image segmentation Given an image, we build a region adjacency graph [207] using as nodes the regions generated by an over-segmentation procedure [56]. The SC technique used in this example is the recursive normalised cut [192], which recursively clusters the nodes until convergence. For MinCut and DiffPool, node features consist of the average and total colour in each over-segmented region. We set the number of desired clusters to $K = 4$. The results in Figure 4.7 show that MinCut yields a precise and intuitive segmentation. On the other hand, SC and DiffPool aggregate wrong regions and, also, SC finds too many segments.



(a) DiffPool



(b) MinCut

Figure 4.8. Unsupervised losses and NMI of DiffPool and MinCut on Cora.

Table 4.3. NMI and CS obtained by clustering the nodes on citation networks over 10 different runs. The number of clusters K is equal to the number of node classes.

Dataset	K	Spectral clustering		DiffPool		MinCut	
		NMI	CS	NMI	CS	NMI	CS
Cora	7	0.025 ± 0.014	0.126 ± 0.042	0.315 ± 0.005	0.309 ± 0.005	**0.404** ± 0.018	**0.392** ± 0.018
Citeseer	6	0.014 ± 0.003	0.033 ± 0.000	0.139 ± 0.016	0.153 ± 0.020	**0.287** ± 0.047	**0.283** ± 0.046
Pubmed	3	0.182 ± 0.000	**0.261** ± 0.000	0.079 ± 0.001	0.085 ± 0.001	**0.200** ± 0.020	0.197 ± 0.019

Clustering on citation networks We cluster the nodes of three citation networks: Cora, Citeseer, and Pubmed. The nodes are documents represented by sparse bag-of-words feature vectors and the binary undirected edges indicate citation links between documents. Each node is labelled with the document class, which we use as ground truth for the clusters. To evaluate the partitions generated by each method, we check the agreement between the cluster assignments and the true labels. Table 4.3 reports the Completeness Score

$$\text{CS}(\tilde{\mathbf{y}}, \mathbf{y}) = 1 - \frac{H(\tilde{\mathbf{y}}|\mathbf{y})}{H(\tilde{\mathbf{y}})} \tag{4.14}$$

and the Normalised Mutual Information

$$\text{NMI}(\tilde{\mathbf{y}}, \mathbf{y}) = \frac{H(\tilde{\mathbf{y}}) - H(\tilde{\mathbf{y}}|\mathbf{y})}{\sqrt{H(\tilde{\mathbf{y}}) - H(\mathbf{y})}}, \tag{4.15}$$

where $\mathbf{y}$ indicates the true labels, $\tilde{\mathbf{y}}$ are the predicted cluster assignments, and $H(\cdot)$ is the entropy.

Once again, our GNN architecture performs better than SC, which does not account explicitly for the node features when generating the clusters. MinCut also outperforms DiffPool since the minimisation of the unsupervised loss in DiffPool fails to converge to a good solution. A pathological behaviour in DiffPool is revealed by Figure 4.8, which depicts the evolution of the NMI scores as the unsupervised losses in DiffPool and MinCut are minimised in training (note how the NMI of DiffPool eventually decreases). From Figure 4.8, it is also possible to see the interaction between the cut loss and the orthogonality loss in our approach. In particular, the cut loss does not converge to its minimum ($\mathcal{L}_c = -1$), corresponding to one of the unwanted solutions discussed in Section 4.2.2. Instead, MinCut learns the optimal trade-off between $\mathcal{L}_c$ and $\mathcal{L}_o$ and achieves a better and more stable clustering performance than DiffPool.

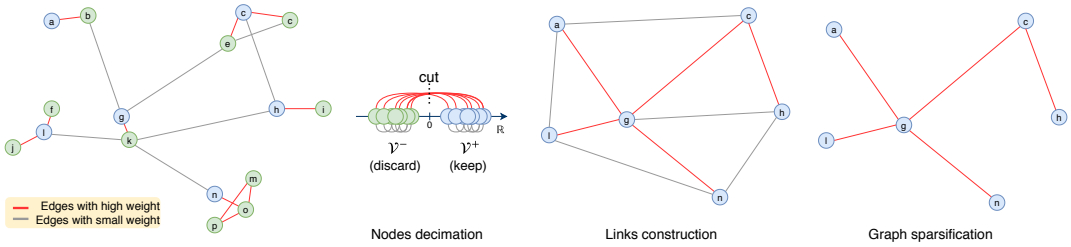


Figure 4.9. Schematic view of node decimation pooling. First, the nodes are partitioned into two sets according to a maximum cut objective and then they are decimated by dropping one of the two sets ($\mathcal{V}^-$). Then, a coarsened Laplacian is built by connecting the remaining nodes with a graph reduction procedure. Finally, the edges with low weights in the new adjacency matrix are dropped to make the resulting graph more sparse.

4.3 Node decimation pooling

The contents of this section were adapted from "Hierarchical representation learning in graph neural networks with node decimation pooling," Bianchi et al. [21].

As an alternative approach to the MinCut operator, in this section we present a non-trainable, sparse and adaptive pooling method called *Node Decimation Pooling* (NDP).

The idea behind the method is to subsample the nodes of a graph in a way that is as regular as possible, *i.e.*, so that as few adjacent nodes as possible are kept in the reduced graph. The rationale is that connected nodes exchange information during convolutional operations and, as a result, their features become similar and redundant. Therefore, subsampling is an effective way to reduce this redundancy.

The proposed NDP operator consists of a sparse selection based on the idea of *node decimation* followed by a two-step connection:

1. Connect the selected nodes with a *link construction* procedure;

2. Sparsify the adjacency matrix resulting from the coarsened Laplacian, so that only strong connections are kept, *i.e.*, those edges whose weight is associated to an entry of the adjacency matrix above a given threshold ϵ.

We propose a simple and efficient spectral algorithm that partitions the graph nodes in two sets by optimising a *maximum cut* objective. From this partition, we identify which nodes to keep and which to discard. Then, to preserve the original

topological structure of the graph, the two-step connection function that connects the subsampled nodes (which likely will have become disconnected after the subsampling) in a way that resembles the original graph's topology. The three main steps (decimation, link construction, and sparsification) are depicted in Figure 4.9.

Note that, under the SRC framework, the sparse selection assigns each node in the chosen partition to a supernode. In other words, each supernode contains exactly one node and there are as many supernodes as nodes in the partition. Consequently, the reduction function in this case simply returns the single node in each supernode. This is a general pattern that applies to every pooling method based on subsampling, which is the case here.

The proposed method is completely unsupervised and the coarsened graphs are pre-computed before training the GNN. In the following sections, we describe the three main steps of NDP and analyse the relation of NDP with the maximum cut problem.

4.3.1 Node decimation with maximum cut spectral partitioning

The proposed node decimation procedure is the closest possible approximation of a regular subsampling in the case of an irregular graph and identifies a partition of the nodes so that each subset contains approximately half of the nodes and the nodes in each subset are as little connected as possible.

The partition of the vertices, *i.e.*, the cut, that maximises the volume of edges whose endpoints are on opposite sides of the partition is the solution of the maximum cut problem [164]. The maximum cut objective is expressed by the integer quadratic problem

$$\max_{\mathbf{z}} \sum_{i,j\in\mathcal{V}} \mathbf{a}_{ij}(1-z_i z_j) \text{ s.t. } z_i \in \{-1,1\}, \tag{4.16}$$

where $\mathbf{z}$ is the vector containing the optimisation variables z_i for $i = 1,\dots,N$ indicating to which side of the bi-partition the node i is assigned to. Problem (4.16) is NP-hard and heuristics must be considered to solve it. The heuristic that gives the best-known maximum cut approximation in polynomial time is the Goemans-Williamson algorithm, which is based on the Semi-Definite Programming (SDP) relaxation [69]. Solving SDP is cumbersome and requires specific optimisation programs that scale poorly on large graphs. Therefore, we propose a simple algorithm based on graph spectral theory.

First, we rewrite the objective function (4.16) as a quadratic form of the graph Laplacian:

$$\begin{aligned}
&\sum_{i,j} \mathbf{a}_{ij}(1 - z_i z_j) = \sum_{i,j} \mathbf{a}_{ij} \left(\frac{z_i^2 + z_j^2}{2} - z_i z_j \right) \\
&= \frac{1}{2} \sum_i \left[\sum_j \mathbf{a}_{ij} \right] z_i^2 + \frac{1}{2} \sum_j \left[\sum_i \mathbf{a}_{ij} \right] z_j^2 - \sum_{i,j} \mathbf{a}_{ij} z_i z_j \\
&= \frac{1}{2} \sum_i \mathbf{d}_{ii} z_i^2 + \frac{1}{2} \sum_j \mathbf{d}_{jj} z_j^2 - \mathbf{z}^\top \mathbf{A} \mathbf{z} \\
&= \mathbf{z}^\top \mathbf{D} \mathbf{z} - \mathbf{z}^\top \mathbf{A} \mathbf{z} = \mathbf{z}^\top \mathbf{L} \mathbf{z}
\end{aligned} \tag{4.17}$$

where $\mathbf{D}$ is the degree matrix (*cf.* Section 2.1).

Then, we consider a continuous relaxation of the integer problem (4.16) by letting the discrete partition vector $\mathbf{z}$ assume continuous values, contained in a vector $\mathbf{c}$:

$$\max_{\mathbf{c}} \ \mathbf{c}^\top \mathbf{L} \mathbf{c}, \ \text{ s.t. } \ \mathbf{c} \in \mathbb{R}^N \ \text{ and } \ \|\mathbf{c}\|^2 = 1. \tag{4.18}$$

Equation (4.18) can be solved by considering the Lagrangian $\mathbf{c}^\top \mathbf{L} \mathbf{c} + \lambda \mathbf{c}^\top \mathbf{c}$ to find the maximum of $\mathbf{c}^\top \mathbf{L} \mathbf{c}$ under constraint $\|\mathbf{c}\|^2 = 1$. By setting the gradient of the Lagrangian to zero, we recover the eigenvalue equation $\mathbf{L}\mathbf{c} + \lambda \mathbf{c} = 0$. All the eigenvalues of $\mathbf{L}$ are non-negative and, by restricting the space of feasible solutions to vectors of unit norm, the trivial solution $\mathbf{c}^* = \infty$ is excluded. In particular, if $\|\mathbf{c}\|^2 = 1$, $\mathbf{c}^\top \mathbf{L} \mathbf{c}$ is a Rayleigh quotient and reaches its maximum $\lambda_{\max}$ (the largest eigenvalue of $\mathbf{L}$) when $\mathbf{c}^*$ corresponds to $\mathbf{u}_{\max}$, the eigenvector associated with $\lambda_{\max}$ (*cf.* Equation (2.3)).

Since the components of $\mathbf{u}_{\max}$ are real, we apply a rounding procedure to find a discrete solution. Specifically, we are looking for a partition $\mathbf{z}^* \in \mathcal{Z}$, where $\mathcal{Z} = \{-1, 1\}^N$ is the set of all feasible cuts so that $\mathbf{z}^*$ is the closest to $\mathbf{c}^*$. This amounts to solving the problem:

$$\mathbf{z}^* = \operatorname*{ArgMin}_{\mathbf{z} \in \mathcal{Z}} \|\mathbf{c}^* - \mathbf{z}\|_2, \tag{4.19}$$

with the optimum given by

$$z_i^* = \begin{cases} 1, & c_i^* \geq 0, \\ -1, & c_i^* < 0. \end{cases} \tag{4.20}$$

By means of the rounding procedure in Equation (4.20), the nodes in $\mathcal{V}$ are partitioned in two sets, $\mathcal{V}^+$ and $\mathcal{V}^- = \mathcal{V} \setminus \mathcal{V}^+$, such that

$$\mathcal{V}^+ = \{ i \in \mathcal{V} \mid \mathbf{u}_{\max}[i] \geq 0 \}. \tag{4.21}$$

In the NDP algorithm we always drop the nodes in $\mathcal{V}^-$, *i.e.*, the nodes associated with a negative value in $\mathbf{u}_{\max}$. However, it would be equivalent to drop the nodes in $\mathcal{V}^+$. The node decimation procedure offers two important advantages:

1. Since $|\mathcal{V}^+| \approx |\mathcal{V}^-|$, it removes approximately half of the nodes when applied;

2. The eigenvector $\mathbf{u}_{\max}$ can be quickly computed with the power method [19].

There exists an analogy between the proposed spectral algorithm for partitioning the graph nodes and spectral clustering [214]. However, as we saw in Section 4.2.1, spectral clustering solves the minimum cut problem [192], which is somehow orthogonal to the maximum cut problem considered here. In particular, spectral clustering identifies $K \geq 2$ clusters of densely connected nodes by cutting the smallest volume of edges in the graph, while our algorithm cuts the largest volume of edges yielding two sets of nodes, $\mathcal{V}^+$ and $\mathcal{V}^-$, that cover the original graph in a similar way. Moreover, spectral clustering partitions the nodes in K clusters based on the values of the eigenvectors associated with the $K \geq 1$ smallest eigenvalues, while our algorithm partitions the nodes in two sets based only on the last eigenvector $\mathbf{u}_{\max}$.

4.3.2 Link construction on the coarsened graph

After dropping nodes in $\mathcal{V}^-$ and all their incident edges, the resulting graph is likely to be disconnected. Therefore, we use a *link construction* procedure to obtain a connected graph supported by the nodes in $\mathcal{V}^+$. Specifically, we adopt the Kron reduction [53] to generate a new Laplacian $\mathbf{L}'$, which is computed as the Schur complement of $\mathbf{L}$ w.r.t. the nodes in $\mathcal{V}^-$. In detail, the reduced Laplacian $\mathbf{L}'$ is

$$\mathbf{L}' = \mathbf{L} \setminus \mathbf{L}_{\mathcal{V}^-,\mathcal{V}^-} = \mathbf{L}_{\mathcal{V}^+,\mathcal{V}^+} - \mathbf{L}_{\mathcal{V}^+,\mathcal{V}^-}\mathbf{L}_{\mathcal{V}^-,\mathcal{V}^-}^{-1}\mathbf{L}_{\mathcal{V}^-,\mathcal{V}^+} \tag{4.22}$$

where $\mathbf{L}_{\mathcal{V}^+,\mathcal{V}^-}$ identifies a sub-matrix of $\mathbf{L}$ with rows corresponding to the nodes in $\mathcal{V}^+$ and columns corresponding to the nodes in $\mathcal{V}^-$. All other matrices are constructed in a similar way.

It is possible to show that $\mathbf{L}_{\mathcal{V}^-,\mathcal{V}^-}$ is always invertible if the associated adjacency matrix $\mathbf{A}$ is irreducible. We note that $\mathbf{A}$ is irreducible when the graph is not disconnected (*i.e.*, has a single component), a property that holds when the algebraic multiplicity of the eigenvalue $\lambda_{\min} = 0$ is 1.

Let us consider the case where $\mathbf{A}$ has no self-loops ($\mathbf{a}_{ii} = 0, \forall i$). The Laplacian is by definition a weakly diagonally dominant matrix, since $\mathbf{L}_{ii} = \sum_{j=1, j\neq i}^{N} |\mathbf{L}_{ij}|$ for all $i \in \mathcal{V}$. If $\mathbf{A}$ is irreducible, then $\mathbf{L}$ is also irreducible. This implies that the strict

inequality $\mathbf{L}_{ii} > \sum_{j=1, j\neq i}^{n} |\mathbf{L}_{ij}|$ holds for at least one vertex $i \in \mathcal{V}^{-}$. It follows that the Kron-reduced Laplacian $\mathbf{L}_{\mathcal{V}^{-},\mathcal{V}^{-}}$ is also irreducible, diagonally dominant, and has at least one row with a strictly positive row sum. Hence, $\mathbf{L}_{\mathcal{V}^{-},\mathcal{V}^{-}}$ is invertible, as proven by Horn and Johnson [89, Corollary 6.2.27]. When $\mathbf{A}$ contains self-loops, the existence of the inverse of $\mathbf{L}_{\mathcal{V}^{-},\mathcal{V}^{-}}$ is still guaranteed through a small workaround, which is discussed in Appendix B.1. Finally, if the graph is disconnected then $\mathbf{A}$ is reducible (*i.e.*, it can be expressed in an upper-triangular block form by simultaneous row/column permutations); in this case, the Kron reduction can be computed using the generalised inverse $\mathbf{L}^{\dagger}_{\mathcal{V}^{-},\mathcal{V}^{-}}$ and the solution corresponds to a generalised Schur complement of $\mathbf{L}$.

$\mathbf{L}'$ in Equation (4.22) is a well-defined Laplacian where two nodes are connected if and only if there is a path between them in $\mathbf{L}$ (connectivity preservation property). Also, $\mathbf{L}'$ does not introduce self-loops and guarantees the preservation of resistance distance [194]. Finally, Kron reduction guarantees spectral interlacing between the original Laplacian $\mathbf{L} \in \mathbb{R}^{N\times N}$ and the new coarsened one $\mathbf{L}' \in \mathbb{R}^{K\times K}$, with $K \leq N$. Specifically, we have $\lambda_i \geq \lambda'_i \geq \lambda_{N-K+i}$, $\forall i = 1, \dots, K$, where λ_i and λ'_i are the eigenvalues of $\mathbf{L}$ and $\mathbf{L}'$, respectively.

The adjacency matrix of the new coarsened graph is recovered from the coarsened Laplacian:

$$\mathbf{A}' = \left(-\mathbf{L}' + \operatorname{diag}(\{ \sum_{j=1, j\neq i}^{K} \mathbf{L}'_{ij} \}_{i=1}^{K}) \right). \tag{4.23}$$

We note that $\mathbf{A}'$ does not contain self-loops; we refer to Appendix B.1 for a discussion on how to handle the case with self-loops in the original adjacency matrix.

To obtain a multi-resolution view of a graph, we can compute a *pyramid* of coarsened Laplacians by recursively applying node decimation followed by Kron reduction. At the end of the procedure, we derive the corresponding pyramid of adjacency matrices $\mathcal{A} = \{\mathbf{A}^{(0)}, \mathbf{A}^{(1)}, \dots, \mathbf{A}^{(l)}, \dots\}$ from the associated coarsened Laplacians.

4.3.3 Graph sparsification

Due to the connectivity preservation property, by repeatedly applying Kron reduction the graph eventually becomes fully connected. This implies a high computational burden in deeper layers of a GNN, since the complexity of convolutional operations depends on the number of edges.

To address this issue, it is possible to apply the spectral sparsification algorithm proposed by Batson et al. [13] to obtain a sparser graph. However, we

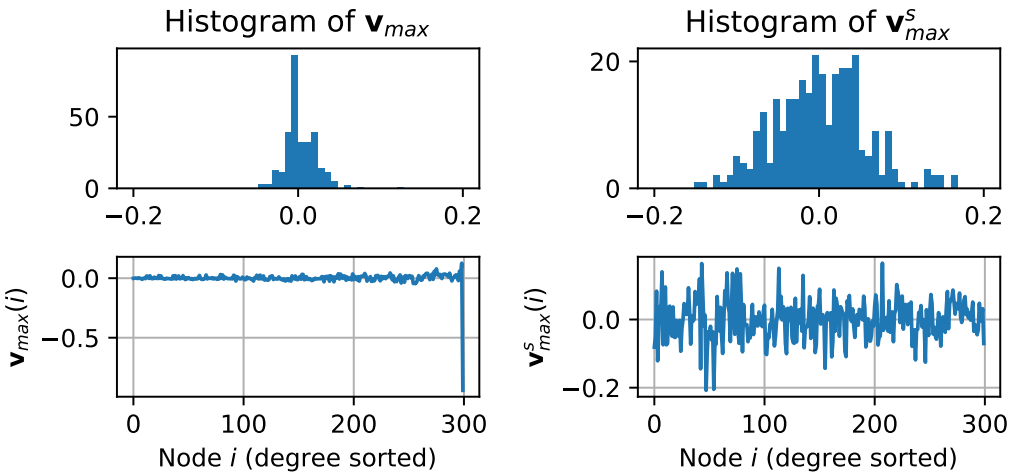


Figure 4.10. (Left) distribution and values assumed by $\mathbf{u}_{\max}$. (Right) distribution and values assumed by $\mathbf{v}_{\max}$. The entries of the eigenvectors are sorted by node degree. A Stochastic Block Model graph was used in this example.

found that this procedure leads to numerical instability and poor convergence during the learning phase. Therefore, to limit the number of edges with non-zero weights we propose a sparsification procedure that removes from the adjacency matrix of the coarsened graph the edges whose weight is below a small user-defined threshold ϵ:

$$\mathbf{a}_{ij}^{(l)} \leftarrow \begin{cases} 0, & \text{if } |\mathbf{a}_{ij}^{(l)}| \leq \epsilon \\ \mathbf{a}_{ij}^{(l)}, & \text{otherwise.} \end{cases} \tag{4.24}$$

4.3.4 Analysis and implementation details

Numerical precision in eigendecomposition The entries of $\mathbf{u}_{\max}$ associated with low-degree nodes usually have values so small that their signs may be flipped due to numerical errors. To address this issue, we instead consider the symmetric normalised Laplacian $\mathbf{L}_n = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$ (*cf.* Section 2.1) and consider its eigendecomposition with eigenvectors $\mathbf{v}_k$ and eigenvalues μ_k. The partition obtained by using $\mathbf{v}_{\max}$, *i.e.*, the eigenvector $\mathbf{L}_n$ associated with the maximum eigenvalue $\mu_{\max}$, is analytically the same to the one obtained in Equation (4.21) with $\mathbf{u}_{\max}$. Indeed, since $\mathbf{v}_{\max} = \mathbf{D}^{-1/2}\mathbf{u}_{\max}$, the values of the two eigenvectors are rescaled by positive numbers and, therefore, the sign of their components is the same. However, a positive effect of the degree normalisation is that the values

in $\mathbf{v}_{\max}$ associated with low-degree nodes are amplified.

Figure 4.10 compares the values in the eigenvectors $\mathbf{u}_{\max}$ and $\mathbf{v}_{\max}$, computed from the same graph. Since many values in $\mathbf{u}_{\max}$ are concentrated around zero, partitioning the nodes according to the sign of the entries in $\mathbf{u}_{\max}$ gives numerically unstable results. On the other hand, since the values in $\mathbf{v}_{\max}$ are distributed more evenly, the nodes can be partitioned more precisely.

Note that, even if the indexes of $\mathcal{V}^+$ are identified from the normalised Laplacian $\mathbf{L}_n$, the Kron reduction is still performed on the Laplacian $\mathbf{L}$. In Appendix B.2, we report numerical differences in the size of the cut obtained when using $\mathbf{u}_{\max}$ or $\mathbf{v}_{\max}$.

Evaluation of the approximate maximum cut solution Since computing the optimal maximum cut solution is NP-hard, it is generally not possible to evaluate the quality of the cut found by the proposed spectral method (*cf.* Section 4.3.1) in terms of discrepancy from the maximum cut.

Let MaxCut indicate the value of the maximum cut. To assess the quality of a solution we consider the following bounds:

$$0.5 \leq \frac{\text{MaxCut}}{|\mathcal{E}|} \leq \frac{\mu_{\max}}{2} \leq 1. \tag{4.25}$$

The value $\mu_{\max}/2$ is an upper-bound of MaxCut$/|\mathcal{E}|$, where $\mu_{\max}$ is the largest eigenvalue of $\mathbf{L}_n$ and $|\mathcal{E}| = \sum_{i,j} \mathbf{a}_{ij}$. The lower-bound 0.5 is given by the random cut, which uniformly assigns the nodes to the two sides of the partition.[2] The derivation of the upper bound is in Appendix B.2.

To quantify the size of a cut induced by a partition vector $\mathbf{z}$, such as the one in (4.20), we introduce the function

$$0 \leq \gamma(\mathbf{z}) = \frac{\mathbf{z}^\top \mathbf{L}\mathbf{z}}{2\sum_{i,j} a_{ij}} \leq \frac{\text{MaxCut}}{|\mathcal{E}|}, \tag{4.26}$$

which measures the proportion of edges cut by $\mathbf{z}$. Note that $\gamma(\cdot)$ depends also on $\mathbf{L}$, but we keep it implicit to simplify the notation.

Let us now consider the best- and worst-case scenarios. The best case is the bipartite graph, where the maximum cut is known and it cuts all the graph edges. The partition $\mathbf{z}$ found by our spectral algorithm on bipartite graphs is optimal, *i.e.*, $\gamma(\mathbf{z}) = \text{MaxCut}/|\mathcal{E}| = 1$. In graphs that are almost bipartite or, in general, that have very sparse and regular edges, a large percentage of edges can be cut

[2] A random cut $\mathbf{z}$ is, on average, at least 0.5 of the optimal cut $\mathbf{z}^*$: $\mathbb{E}[|\mathbf{z}|] = \sum_{(i,j)\in\mathcal{E}} \mathbb{E}[z_i z_j] = \sum_{(i,j)\in\mathcal{E}} \Pr[(i,j) \in \mathbf{z}] = \frac{|\mathcal{E}|}{2} \geq \frac{|\mathbf{z}^*|}{2}$

if the nodes are partitioned correctly. Indeed, for these graphs, the maximum cut is usually large and is closer to the upper bound in Equation (4.25). On the other hand, in very dense graphs the maximum cut is smaller, as well as the gap between the upper- and lower-bound in (4.25). Notably, the worst-case scenario is a complete graph where it is not possible to cut more than half of the edges, *i.e.*, MaxCut $= 0.5$. We note that in graphs made of a sparse regular part that is weakly connected to a large dense part, the gaps in Equation (4.25) can be arbitrarily large.

The proposed spectral algorithm is not designed to handle very dense graphs; an intuitive explanation is that $\mathbf{v}_{\max}$ can be interpreted as the graph signal with the highest possible frequency, since its sign changes as often as possible between adjacent nodes. While such oscillation in the sign is possible on bipartite graphs, for complete graphs it is not possible to find a signal that assumes an opposite sign on neighbouring nodes, because all nodes are connected with each other. Remarkably, the solution of Equation (4.20) found by the spectral algorithm on very dense graphs can be worse than the random cut. A theoretical result found by Trevisan [208] states that a spectral algorithm, like the one we propose, is guaranteed to yield a cut larger than the random partition only when $\mu_{\max} \geq 2(1-\tau) = 1.891$ (see Appendix B.3 for details).

To illustrate how the size of the cut found by the spectral algorithm changes between the best- and worst-case scenarios, we randomly add edges to a bipartite graph until it becomes complete. Figure 4.11 illustrates how the size of the cut $\gamma(\mathbf{z})$ induced by the spectral partition $\mathbf{z}$ changes as more edges are added and the original structure of the graph is corrupted (blue line). The figure also reports the size of the random cut (orange line) and the maximum cut upper bound from Equation (4.26) (green line). The black line indicates the threshold found by Trevisan [208], *i.e.*, the value of $\mu_{\max}^2/2$ below which the spectral cut is no longer guaranteed to be larger than the random cut. The graph used to generate the figure is a regular grid; however, similar results hold for other families of random graphs and are reported in the supplementary material.

Figure 4.11 shows that the spectral algorithm finds a better-than-random cut even when $\mu_{\max}/2 < 1-\tau$ (*i.e.*, when the result of Trevisan [208] does not hold), and only approaches the size of the random cut when the edge density is very high (70%-80%).

Importantly, when the size of the spectral partition becomes smaller than the random partition, the upper-bound $\mu_{\max}/2 \approx 0.5$, meaning that the random cut is very close to the maximum cut. To obtain a cut that is always at least as good as the random cut, we first compute the partition $\mathbf{z}$ as in Equation (4.20) and evaluate its size $\gamma(\mathbf{z})$: if $\gamma(\mathbf{z}) < 0.5$, we return a random partition instead.

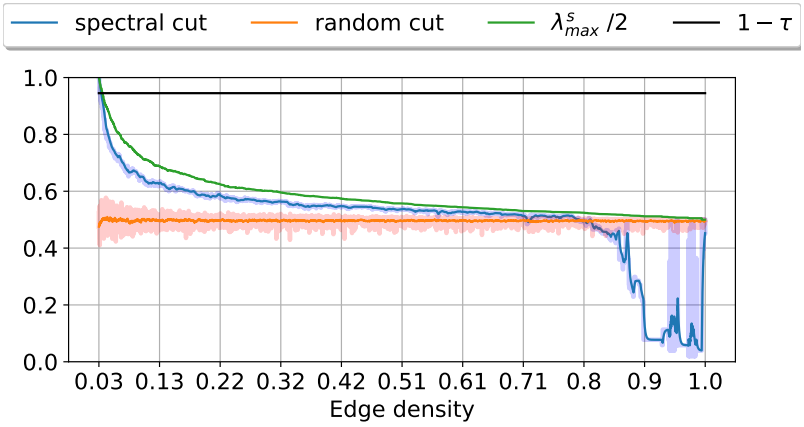


Figure 4.11. Blue line: fraction of edges cut by the partition given by the spectral algorithm. Orange line: fraction of edges removed by a random cut. Green line: the maximum cut upper bound as a function of the largest eigenvalue $\mu_{\max}$ of the symmetric normalised Laplacian. Black line: the threshold from [208] indicating the value of $\mu_{\max}/2$ below which one should switch to the random cut to obtain a solution guaranteed to be $\geq 0.53\cdot$MaxCut. The x-axis indicates the density of the graph connectivity, which increases by randomly adding edges.

We conclude by noting that, due to the smoothing effect of convolution, the nodes belonging to densely connected graph components are likely to have very similar representations computed by the GNN; it is, therefore, not important which of these nodes are dropped by a random cut. The random cut in these cases not only is optimal in terms of the maximum cut objective, but it also introduces randomness that provides robustness when training the GNN model.

Pseudo-code The procedure for generating the pyramid of coarsened adjacency matrices $\mathcal{A}$ and the selection matrices is reported in Algorithm 1. $\mathcal{L}$ is a list of positive integers indicating the levels in the pyramid of coarsened Laplacians that we want to compute. For instance, given levels $\mathcal{L} = [1, 3, 5]$ for a graph of N nodes, the algorithm will return the coarsened graphs with approximately $N/2$, $N/8$, and $N/32$ nodes (in general, $N/2^{l_i}$ for each l_i in $\mathcal{L}$). Matrix $\mathbf{R}$ is a buffer that accumulates the selection matrices when one or more coarsening levels are skipped.

Algorithm 2 shows the details of the pooling function, used in line 3 of Algorithm 1.

Algorithm 1: Graph coarsening

Input: adjacency matrix $\mathbf{A}$, coarsening levels $\mathcal{L}$, sparsification threshold ϵ
Output: coarsened adjacency matrices $\mathcal{A}$, selection matrices $\mathcal{S}$

$\mathbf{A}^{(0)} = \mathbf{A}$, $\mathbf{R} = \mathbf{I}_N$, $\mathcal{A} = \{\}$, $\mathcal{S} = \{\}$, $l = 0$
while $l \leq \max(\mathcal{L})$ **do**
  $\mathbf{A}^{(l+1)}, \mathbf{S}^{(l+1)} = \text{pool}(\mathbf{A}^{(l)})$
  if $l \in \mathcal{L}$ **then**
    $\mathcal{A} \leftarrow \mathcal{A} \cup \mathbf{A}^{(l+1)}$, $\mathcal{S} \leftarrow \mathcal{S} \cup \mathbf{S}^{(l+1)}\mathbf{R}$
    $\mathbf{R} \leftarrow \mathbf{I}_{N_l}$
  else
    $\mathbf{R} \leftarrow \mathbf{S}^{(l+1)}\mathbf{R}$
  $l \leftarrow l + 1$
$\mathcal{A} \leftarrow \{\bar{\mathbf{A}}^{(l)} \mid \bar{a}_{ij}^{(l)} = a_{ij}^{(l)}$ if $a_{ij}^{(l)} > \epsilon$ and 0 otherwise, $\forall \mathbf{A}^{(l)} \in \mathcal{A}\}$

Algorithm 2: pool$(\cdot)$ function

Input: adjacency matrix $\mathbf{A}^{(l)} \in \mathbb{R}^{N_l \times N_l}$
Output: coarsened adjacency matrix $\mathbf{A}^{(l+1)} \in \mathbb{R}^{N_{l+1} \times N_{l+1}}$, selection matrix $\mathbf{S}^{(l+1)} \in \mathbb{N}^{N_{l+1} \times N_l}$

get $\mathbf{L}^{(l)} = \mathbf{D}^{(l)} - \mathbf{A}^{(l)}$ and $\mathbf{L}_s^{(l)} = \mathbf{I} - (\mathbf{D}^{(l)})^{-\frac{1}{2}}\mathbf{A}^{(l)}(\mathbf{D}^{(l)})^{-\frac{1}{2}}$
compute the eigenvector $\mathbf{v}_{\max}$ of $\mathbf{L}_n^{(l)}$
partition vector $\mathbf{z}$ s.t. $z_i = 1$ if $\mathbf{v}_{\max}[i] \geq 0$, $z_i = -1$ if $\mathbf{v}_{\max}[i] < 0$
if $\gamma(\mathbf{z}) < 0.5$ **then**
  random sample $z_i \sim \{-1, 1\}, \forall i = 1, \ldots, N_l$ (random cut)
$\mathcal{V}^+ = \{i \mid z_i = 1\}$, $\mathcal{V}^- = \{i \mid z_i = -1\}$
$\mathbf{L}^{(l+1)} = \mathbf{L}^{(l)} \setminus \mathbf{L}^{(l)}_{\mathcal{V}^-,\mathcal{V}^-}$ (Kron reduction)
$\mathbf{A}^{(l+1)} = -\mathbf{L}^{(l+1)} + \text{diag}(\sum_{j \neq i} \mathbf{L}_{ij}^{(l+1)})$
$\mathbf{S}^{(l+1)} = [\mathbf{I}_{N_{l+1}}]_{\mathcal{V}^+,:}$

Computational cost analysis The most expensive operations in the NDP algorithm are i) the cost of computing the eigenvector $\mathbf{v}_{\max}$, and ii) the cost of inverting the sub-matrix $\mathbf{L}_{\mathcal{V}^-,\mathcal{V}^-}$ within the Kron reduction.

Computing all eigenvectors costs $O(N^3)$, where N is the number of nodes. However, computing only the eigenvector corresponding to the largest eigenvalue is fast when using the power method [217], which requires only a few iterations (usually 5-10), each one of complexity $O(N^2)$. The cost of inverting $\mathbf{L}_{\mathcal{V}^-,\mathcal{V}^-}$ is $O(|\mathcal{V}^-|^3)$, where $|\mathcal{V}^-|$ is the number of nodes that are dropped.

We notice that the coarsened graphs are pre-computed before training the GNN. Therefore, the computational time of graph coarsening is much lower compared to training the GNN for several epochs, since each convolution in the GNN has a cost $O(N^2)$.

Structure of the sparsified graphs When applying the sparsification, the spectrum of the resulting adjacency matrix $\bar{\mathbf{A}}$ is preserved, up to a small factor that depends on ϵ, w.r.t. the spectrum of $\mathbf{A}$.

Theorem 2. *Let* $\mathbf{Q}$ *be a matrix used to remove small values in the adjacency matrix* $\mathbf{A}$*, which is defined as*

$$\mathbf{Q} = \begin{cases} q_{ij} = -a_{ij}, & \text{if } |a_{ij}| \leq \epsilon \\ q_{ij} = 0, & \text{otherwise.} \end{cases} \tag{4.27}$$

Each eigenvalue $\bar{\alpha}_i$ *of the sparsified adjacency matrix* $\bar{\mathbf{A}} = \mathbf{A} + \mathbf{Q}$ *is bounded by*

$$\bar{\alpha}_i \leq \alpha_i + \mathbf{u}_i^\top \mathbf{Q} \mathbf{u}_i, \tag{4.28}$$

where α_i *and* $\mathbf{u}_i$ *are eigenvalue-eigenvector pairs of* $\mathbf{A}$*.*

Proof. Let $\mathbf{P}$ be a matrix with elements $\mathbf{p}_{ij} = \text{sign}(\mathbf{q}_{ij})$ and consider the perturbation $\mathbf{A} + \epsilon\mathbf{P}$, which changes the eigenvalue problem $\mathbf{A}\mathbf{u}_i = \alpha_i \mathbf{u}_i$ to

$$(\mathbf{A} + \epsilon\mathbf{P})(\mathbf{u}_i + \mathbf{u}_\epsilon) = (\alpha_i + \alpha_\epsilon)(\mathbf{u}_i + \mathbf{u}_\epsilon). \tag{4.29}$$

where α_ϵ is a small number and $\mathbf{u}_\epsilon$ a small vector, which are unknown and indicate a perturbation on α_i and $\mathbf{u}_i$, respectively. By expanding Equation (4.29), then cancelling the equation $\mathbf{A}\mathbf{u}_i = \alpha_i \mathbf{u}_i$ and the high order terms $O(\epsilon^2)$, one obtains

$$\mathbf{A}\mathbf{u}_\epsilon + \epsilon\mathbf{P}\mathbf{u}_i = \alpha_i \mathbf{u}_\epsilon + \alpha_\epsilon \mathbf{u}_i. \tag{4.30}$$

Since $\mathbf{A}$ is symmetric, its eigenvectors can be used as a basis to express the small vector $\mathbf{u}_\epsilon$

$$\mathbf{u}_\epsilon = \sum_{j=1}^{N} \delta_j \mathbf{u}_j, \tag{4.31}$$

where δ_j are (small) unknown coefficients. Substituting Equation (4.31) into Equation (4.30) and bringing $\mathbf{A}$ inside the summation we get

$$\sum_{j=1}^{N} \delta_j \mathbf{A}\mathbf{u}_j + \epsilon \mathbf{P}\mathbf{u}_i = \alpha_i \sum_{j=1}^{N} \delta_j \mathbf{u}_j + \alpha_\epsilon \mathbf{u}_i. \tag{4.32}$$

By considering the original eigenvalue problem that gives $\sum_{j=1}^{N} \delta_j \mathbf{A}\mathbf{u}_j = \sum_{j=1}^{N} \delta_j \alpha_j \mathbf{u}_j$ and by left-multiplying each term with $\mathbf{u}_i^\top$, Equation (4.32) becomes

$$\mathbf{u}_i^\top \sum_{j=1}^{N} \delta_j \alpha_j \mathbf{u}_j + \mathbf{u}_i^\top \epsilon \mathbf{P}\mathbf{u}_i = \mathbf{u}_i^\top \alpha_i \sum_{j=1}^{N} \delta_j \mathbf{u}_j + \mathbf{u}_i^\top \alpha_\epsilon \mathbf{u}_i. \tag{4.33}$$

Since eigenvectors are orthogonal, $\mathbf{u}_i^\top \mathbf{u}_j = 0, \forall j \neq i$ and $\mathbf{u}_i^\top \mathbf{u}_j = 1$, for $j = i$, Equation (4.33) becomes

$$\begin{gathered} \mathbf{u}_i^\top \delta_i \alpha_i \mathbf{u}_i + \mathbf{u}_i^\top \epsilon \mathbf{P}\mathbf{u}_i = \mathbf{u}_i^\top \alpha_i \delta_i \mathbf{u}_i + \mathbf{u}_i^\top \alpha_i \mathbf{u}_i, \\ \mathbf{u}_i^\top \epsilon \mathbf{P}\mathbf{u}_i = \mathbf{u}_i^\top \alpha_\epsilon \mathbf{u}_i = \alpha_\epsilon, \end{gathered} \tag{4.34}$$

which, in turn, gives

$$\alpha_\epsilon = \mathbf{u}_i^\top \epsilon \mathbf{P}\mathbf{u}_i \geq \mathbf{u}_i^\top \mathbf{Q}\mathbf{u}_i, \tag{4.35}$$

as $\mathbf{Q} \leq \epsilon \mathbf{P}$. □

A common way to measure the similarity of two graphs is to compare the spectrum of their Laplacians. To extend the results of Theorem 1 to the spectra of the Laplacians $\mathbf{L}$ and $\bar{\mathbf{L}}$, respectively associated with the original and sparsified adjacency matrices $\mathbf{A}$ and $\bar{\mathbf{A}}$, it is necessary to consider the relationships between the eigenvalues of $\mathbf{A}$ and $\mathbf{L}$. For a d-regular graph, the relationship $\lambda_i = d - \alpha_i$ links the i^{th} eigenvalue λ_i of $\mathbf{L}$ to the i^{th} eigenvalue α_i of $\mathbf{A}$ [131]. However, for a general graph it is only possible to derive a loose bound, $d_{\max} - \alpha_n \leq \lambda_n \leq d_{\max} - \alpha_1$, that depends on the maximum degree $d_{\max}$ of the graph [250, Lemma 2.21].

Therefore, we numerically compare the spectra of the Laplacians associated with the adjacency matrices before and after sparsification. In particular, Figure 4.12 (*top-left*) depicts the spectrum of the Laplacian associated to the original graph $\mathbf{A}^{(0)}$ (black dotted line) and the spectra $\Lambda(\mathbf{L}^{(1)})$, $\Lambda(\mathbf{L}^{(2)})$, $\Lambda(\mathbf{L}^{(3)})$ of the

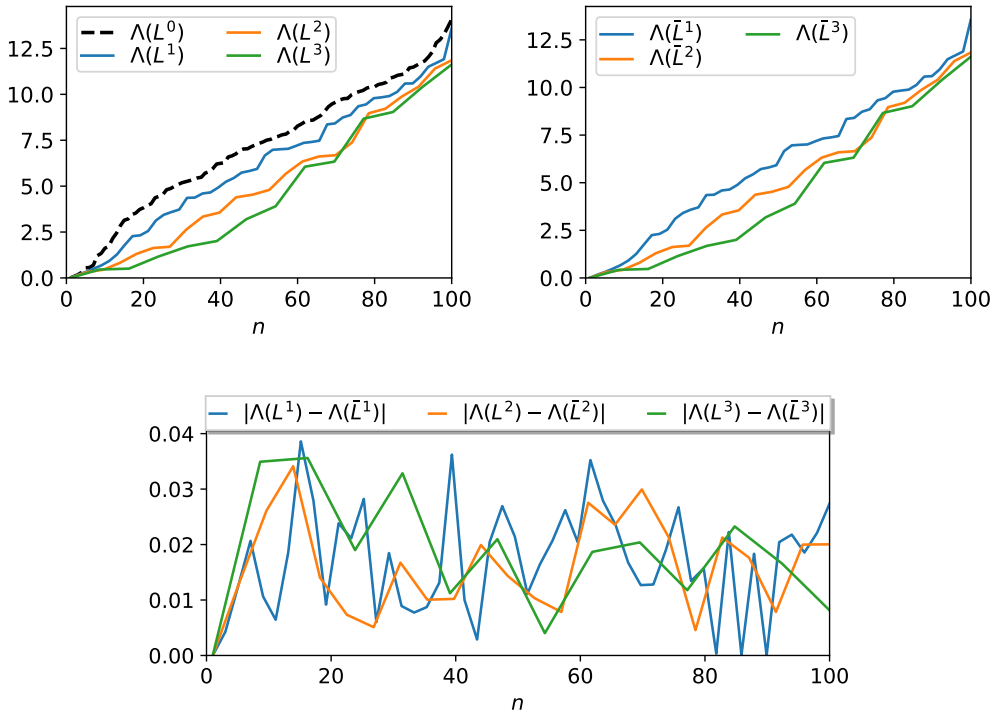


Figure 4.12. *Top-left*: Spectrum of the Laplacians associated with the original adjacency matrix $\mathbf{A}^{(0)}$ and its coarsened versions $\mathbf{A}^{(1)}$, $\mathbf{A}^{(2)}$, and $\mathbf{A}^{(3)}$ obtained with the NDP algorithm. *Top-right*: Spectrum of the Laplacians associated with the sparsified adjacency matrices $\bar{\mathbf{A}}^{(1)}$, $\bar{\mathbf{A}}^{(2)}$, and $\bar{\mathbf{A}}^{(3)}$. *Bottom*: Absolute difference between the spectra of the Laplacians.

Laplacians associated with $\mathbf{A}^{(1)}$, $\mathbf{A}^{(2)}$, and $\mathbf{A}^{(3)}$. Figure 4.12 (*top-right*) depicts the spectra of the Laplacians $\bar{\mathbf{L}}^{(1)}$, $\bar{\mathbf{L}}^{(2)}$, $\bar{\mathbf{L}}^{(3)}$ associated with the sparsified adjacency matrices $\bar{\mathbf{A}}^{(1)}$, $\bar{\mathbf{A}}^{(2)}$, and $\bar{\mathbf{A}}^{(3)}$. We observe that the spectra of $\mathbf{L}^{(l)}$ and $\bar{\mathbf{L}}^{(l)}$ are almost identical and therefore, to better visualise the differences, we show in Figure 4.12 (*bottom*) the absolute differences $|\Lambda(\mathbf{L}^{(l)}) - \Lambda(\bar{\mathbf{L}}^{(l)})|$, where $\Lambda(\mathbf{L})$ indicates the eigenvalues of $\mathbf{L}$. The graph used in Figure 4.12 is a random sensor network and the sparsification threshold is $\epsilon = 10^{-2}$.

To quantify how much the coarsened graph changes as a function of ϵ, we consider the *spectral distance* that measures the dissimilarity between the spectra of the Laplacians associated with $\mathbf{A}$ and $\bar{\mathbf{A}}$ [128]. The spectral distance is computed as

$$SD(\mathbf{L}, \bar{\mathbf{L}}; \epsilon) = \frac{1}{K} \sum_{k=2}^{K+1} \frac{|\bar{\lambda}_k(\epsilon) - \lambda_k|}{\lambda_k}, \tag{4.36}$$

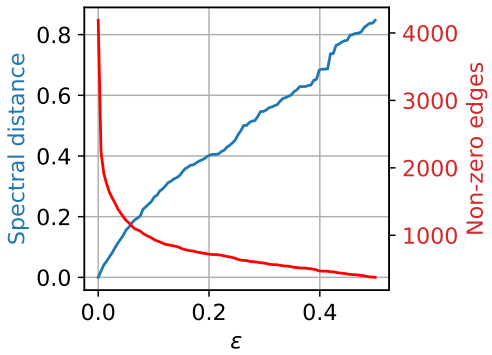


Figure 4.13. In blue, the variation of spectral distance between the Laplacian $\mathbf{L}$ and the Laplacian $\bar{\mathbf{L}}$, associated with the adjacency matrix $\mathbf{A}$ sparsified with threshold ϵ. In red, the number of edges that remain in $\bar{\mathbf{L}}$.

where $\{\lambda_k\}_{k=2}^{K+1}$ and $\{\bar{\lambda}_k(\epsilon)\}_{k=2}^{K+1}$ are, respectively, the K smallest non-zero eigenvalues of $\mathbf{L}$ and $\bar{\mathbf{L}}$.

Figure 4.13 depicts in blue the variation of spectral distance between $\mathbf{L}$ and $\bar{\mathbf{L}}$, as we increase the threshold ϵ used to compute $\bar{\mathbf{A}}$. The red line indicates the number of edges that remain in $\bar{\mathbf{A}}$ after sparsification. We see that, for small increments of ϵ, the spectral distance increases linearly while the number of edges in the graph drops exponentially. Therefore, with a small ϵ it is possible to discard a large number of edges with minimal changes in the graph spectrum.

The graph used to generate Figure 4.13 is a sensor network; results for other types of graph are in Appendix B.4.

4.4 Experiments with pooling methods

The contents of this section were adapted from "Understanding pooling in graph neural networks," Grattarola et al. [79].

One of the main issues in the GNN literature on graph pooling is how to fairly evaluate and compare different methods in order to identify the best one for a specific task. However, the literature has failed to provide a definitive answer on how to best perform this assessment, and currently resorts mostly to evaluating the downstream performance of GNNs equipped with different pooling operators on different graph classification and regression tasks. However, we argue that there is no single general-purpose measure to quantify the performance of a

graph pooling algorithm and the quality of a coarsened graph. In this section, we define three evaluation criteria for pooling operators and design experiments to test whether different classes of methods are able to meet them. In particular, we evaluate operators based on their ability to 1) preserve the information content of the node attributes, 2) preserve the topological structure, and 3) preserve the information required to solve various classification tasks.

To study the behaviour of different pooling methods according to the three main criteria, we perform an experimental comparison between eight hierarchical pooling methods representing the three remaining axes of the taxonomy (density, adaptability, trainability). The pooling methods that we consider are those listed in Table 4.1, which include the two methods that we proposed in Sections 4.2 and 4.3: MinCut [20] and DiffPool [237] are trainable, dense and fixed; Top-K [88, 32] and SAGPool [117] are trainable, sparse, and adaptive; NMF [7] and LaPool [155] are non-trainable, dense and adaptive; Graclus [49] and NDP [21] are non-trainable, sparse and adaptive. We defer implementation details to Appendix C.

4.4.1 Preserving node attributes

As a first experiment, we test the ability of pooling methods to preserve node information. We consider the task of reconstructing the original coordinates of a geometric point cloud from its pooled version. We configure a graph autoencoder to pool the node attributes and then lift them back to the original size using an appropriate lift operator for each method. Note that this experiment evaluates the quality of the pooling methods in compressing node information, but it does not test their generalisation capability since the autoencoder is independently fit on each point could.

Table 4.4 reports the average and standard deviation of the mean squared error (MSE) obtained by the eight methods on different point clouds from the PyGSP library [46] and the ModelNet40 dataset [228]. Figure 4.14 contrasts the original point cloud with the points reconstructed from each pooling method while the connectivity is unchanged; the figures for all point clouds are available in Appendix C. As baseline values for the MSE, we report the mean squared distance between adjacent points: $\gamma = (|\mathcal{E}|\, D_n)^{-1} \sum_{(i,j)\in\mathcal{E}} |\mathbf{x}_i - \mathbf{x}_j|_2^2$; the intuition behind this baseline is that reconstructed points with a MSE score larger than the reference γ cannot be matched, on average, with the original points. We observe that as the point clouds grow in size, many operators cannot achieve MSE $< \gamma$ (red entries). The two methods that stand out from Table 4.4 are the non-trainable NMF and NDP, as confirmed by their respective average ranks

Table 4.4. MSE (values in scale of 10^{-3}) in the autoencoder experiment.[3] The **Rank** row indicates the average ranking of the methods across all datasets.

	Ref. γ	DiffPool	MinCut	NMF	LaPool	TopK	SAGPool	NDP	Graclus
Grid2d	*7.812*	0.010 ±0.005	0.002 ±0.002	**0.000** ±0.000	0.002 ±0.001	18.86 ±3.923	16.61 ±3.270	**0.000** ±0.000	0.109 ±0.000
Ring	*4.815*	0.018 ±0.003	0.001 ±0.000	**0.000** ±0.000	0.052 ±0.046	132.2 ±4.133	148.5 ±30.10	**0.000** ±0.000	0.600 ±0.000
Bunny	*6.874*	3.901 ±0.275	**0.208** ±0.034	0.339 ±0.055	0.610 ±0.103	15.32 ±3.557	16.10 ±1.722	0.373 ±0.070	0.332 ±0.043
Airplane	*0.097*	0.094 ±0.022	0.005 ±0.002	0.020 ±0.000	**0.002** ±0.000	0.096 ±0.028	0.268 ±0.081	0.012 ±0.000	0.009 ±0.000
Car	*0.028*	0.143 ±0.127	0.535 ±0.200	0.016 ±0.001	OOR	0.229 ±0.023	0.204 ±0.029	**0.009** ±0.000	0.102 ±0.000
Guitar	*0.091*	0.101 ±0.025	0.313 ±0.000	0.007 ±0.000	OOR	0.056 ±0.051	0.060 ±0.044	**0.005** ±0.000	0.010 ±0.000
Person	*0.013*	0.077 ±0.041	0.301 ±0.000	**0.001** ±0.000	OOR	0.055 ±0.012	0.062 ±0.033	**0.001** ±0.000	**0.001** ±0.000
Rank		5.29	4.29	2.14	5.43	6.14	6.57	**1.86**	3.43

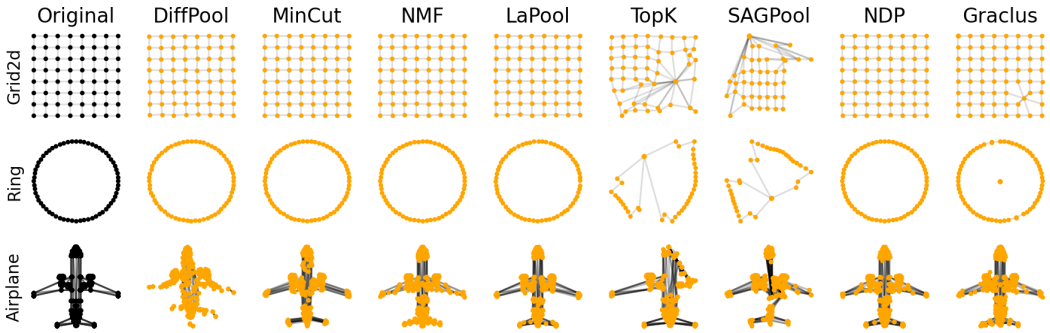


Figure 4.14. Node attributes (point locations) reconstructed with different operators in the autoencoder experiment.

across datasets.

Figure 4.15 depicts a variant of the coarsened graphs where the SEL and CON operations are the same as in Table 4.1, but the RED function is replaced by

$$\mathbf{X}' = \mathbf{S}^\top \mathbf{X}. \tag{4.37}$$

This modification is crucial to interpret the SEL operation because most of the pooling methods use message passing layers before the reduction (see the autoencoder architecture details), which makes the node feature space not directly comparable with the original 2 or 3-dimensional input space. Conversely, the reduction in (4.37) gives (weighted) averages of the supernodes with the benefits of maintaining points in the input space and locating them in the supernodes' centres of mass.

[3] "OOR" indicates Out Of Resources, *i.e.*, either we could not fit a batch size of 8 graphs on an Nvidia Titan V GPU or it took more than 24 hours to complete training. Values of 0.000 indicate any value $< 10^{-6}$.

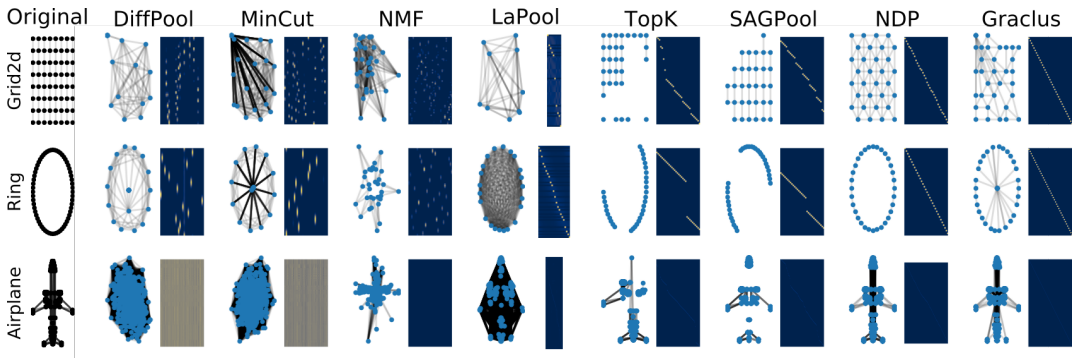


Figure 4.15. Graphs pooled with different operators in the autoencoder experiment with the modified RED function, and the associated selection matrices $\mathbf{S}$.

Two main patterns emerge. First, we see that the two non-trainable sparse methods (NDP and Graclus) perform a rather uniform node subsampling, in such a way that the reduced node features are representative of the original input, which may facilitate the reconstruction of the input node features, as confirmed by the low MSE in Table 4.4. Second, trainable and sparse methods (TopK and SAGPool) tend to cut off entire portions of the graphs, therefore discarding essential node information.

4.4.2 Preserving structure

In this experiment we study the structural similarity between the input and coarsened graphs $\mathcal{G}$ and $\mathcal{G}'$, respectively, by comparing the quadratic forms associated with their respective combinatorial Laplacian matrices $\mathbf{L}$ and $\mathbf{L}'$. This evaluation criterion has also been recently studied by Loukas [128], Hermsdorff and Gunderson [85], and Cai et al. [31], and allows us to compare graphs of different sizes. Specifically, we consider the quadratic loss $\mathcal{L}(\mathcal{G}, \mathcal{G}') = \sum_{i=0}^{D_n} \|\mathbf{X}_{:,i}^\top \mathbf{L} \mathbf{X}_{:,i} - \mathbf{X}'^\top_{:,i} \mathbf{L}' \mathbf{X}'_{:,i}\|$, where $\mathbf{X} \in \mathbb{R}^{N \times D_n}$ is an arbitrary graph signal and $\mathbf{X}'$ its reduction. In this experiment, we choose $\mathbf{X}$ to be the concatenation of the first 10 eigenvectors of $\mathbf{L}$ and the node coordinates of $\mathcal{G}$; all columns are normalised by their norm. For trainable methods, we directly minimise the loss as a self-supervised target. Table 4.5 reports the average loss obtained by the eight operators on different graphs from the PyGSP library, while Figure 4.16 shows examples of pooled graphs and their spectra. We show the result for Grid2d since it is easier to interpret visually; the figures for all graphs are available in Appendix C.

Trainable dense methods can generate coarsened graphs with a quadratic loss w.r.t. the original graph lower than their non-trainable or sparse counterparts.

Table 4.5. Average quadratic loss in the spectral similarity experiment.

	DiffPool	MinCut	NMF	LaPool	TopK	SAGPool	NDP	Graclus
Grid2d	**0.002** ±0.000	0.099 ±0.016	0.369 ±0.000	8.486 ±0.000	0.483 ±0.001	0.306 ±0.017	0.068 ±0.000	0.375 ±0.000
Ring	0.001 ±0.000	**0.000** ±0.000	0.050 ±0.000	5.603 ±0.000	0.067 ±0.000	0.034 ±0.001	0.002 ±0.000	0.058 ±0.000
Sensor	**0.010** ±0.000	0.155 ±0.005	1.177 ±0.000	28.99 ±0.000	1.306 ±0.001	0.721 ±0.077	0.486 ±0.000	1.027 ±0.000
Bunny	**0.011** ±0.003	0.272 ±0.013	40.48 ±0.000	$> 10^3$	1.251 ±0.000	0.708 ±0.138	0.156 ±0.000	1.228 ±0.000
Minnes.	**0.000** ±0.000	0.004 ±0.000	7.117 ±0.000	4.030 ±0.000	0.004 ±0.000	0.001 ±0.000	**0.000** ±0.000	0.080 ±0.000
Airfoil	**0.000** ±0.000	0.006 ±0.000	2.604 ±0.000	26.97 ±0.000	0.006 ±0.000	0.003 ±0.000	**0.000** ±0.000	0.048 ±0.000
Rank	**1.17**	2.83	6.33	7.83	5.83	3.67	2.00	5.67

Table 4.6. Density and median weight of the edges of the coarsened graphs in the spectral similarity experiment.

		Original	DiffPool	MinCut	NMF	LaPool	TopK	SAGPool	NDP	Graclus
Grid2d	**Density**	0.055	0.969	0.969	0.463	0.917	0.084	0.092	0.189	0.103
	Median	1.000	0.216	0.024	0.018	1.445	1.000	1.000	0.500	0.154
Minnes.	**Density**	$9.47{\cdot}10^{-4}$	0.999	0.999	0.010	0.999	0.002	0.002	0.003	0.002
	Median	1.000	0.004	$7.58{\cdot}10^{-4}$	0.014	0.013	1.000	1.000	0.333	0.204
Sensor	**Density**	0.159	0.969	0.969	0.844	0.875	0.273	0.230	0.529	0.227
	Median	0.742	0.463	$2.42{\cdot}10^{-4}$	0.005	6.147	0.765	0.756	0.201	0.103

Interestingly, from the bottom row of Figure 4.16 we see that a low quadratic loss does not necessarily imply a good alignment of the spectra. For example, on the regular grid in Figure 4.16, the excellent spectral alignment achieved by TopK and SAGPool is not reflected by a low quadratic loss value (0.596 and 0.361 respectively). While in principle this experiment focuses on comparing only SEL and CON, we are also evaluating RED since it affects the loss that depends on $\mathbf{X}'$. This can explain the discrepancy between the loss values and the eigenvalues plots.

From Figure 4.16, we see that dense methods (DiffPool, MinCut, NMF, LaPool) yield coarsened graphs that are densely connected. We can make a similar observation for the autoencoder experiment with modified RED operation in Equation (4.37), as shown in Figure 4.15. However, in these dense graphs, most of the edge weights are also small. This is quantitatively reported in Table 4.6, in which we compare the density of edges ($|\mathcal{E}'|/K^2$) and the median edge weight of the coarsened graphs for Grid2d, Minnesota and Sensor. An extended version of this table is reported in Appendix C.

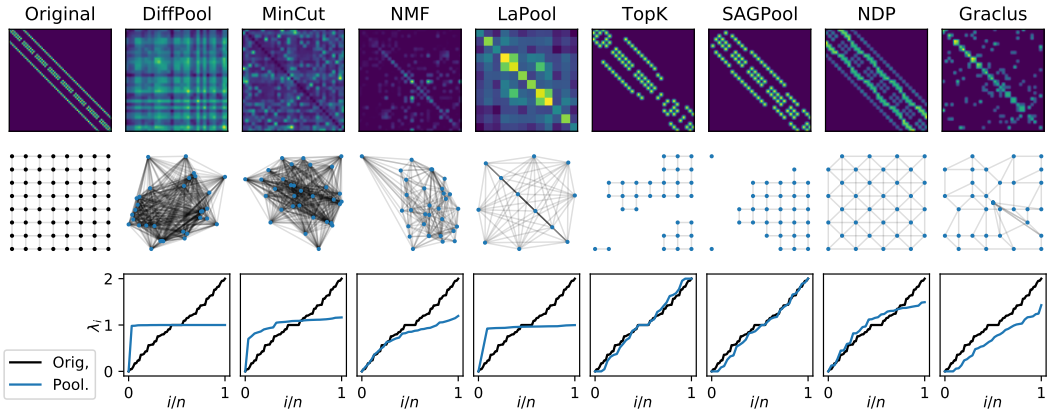


Figure 4.16. Results on a regular grid when optimising for spectral similarity. Top: the coarsened adjacency matrices. Middle: the coarsened graphs with modified RED function. Bottom: the eigenvalues of the normalised Laplacian before (black) and after (blue) pooling. The indices of the eigenvalues are rescaled to fill $[0, 1]$.

Table 4.7. Accuracy on the graph classification benchmarks.[3]

	No-pool	**DiffPool**	**MinCut**	**NMF**	**LaPool**	**TopK**	**SAGPool**	**NDP**	**Graclus**
Colors-3	40.8 ±2.1	55.2 ±1.5	**60.1 ±4.0**	29.7 ±1.7	44.9 ±1.0	26.9 ±4.0	34.4 ±5.2	25.4 ±1.8	29.5 ±2.0
Triangles	93.5 ±0.7	91.3 ±0.2	**95.3 ±0.5**	58.1 ±5.2	88.8 ±0.8	75.2 ±17.3	80.3 ±8.6	75.3 ±1.0	71.4 ±1.7
Proteins	68.8 ±2.8	70.0 ±0.6	**73.8 ±0.8**	68.9 ±3.4	72.9 ±2.0	71.3 ±0.8	73.7 ±0.8	68.4 ±3.4	72.6 ±1.1
Enzymes	83.6 ±2.0	72.4 ±3.9	83.6 ±0.6	32.4 ±8.1	85.0 ±1.2	81.0 ±0.4	68.8 ±18.8	84.8 ±3.2	**85.4 ±4.1**
DD	81.1 ±0.4	75.6 ±1.8	**82.5 ±0.9**	OOR	OOR	80.4 ±0.9	79.0 ±2.7	79.6 ±1.2	78.3 ±2.9
Mutagen.	**78.0 ±1.6**	76.2 ±1.4	73.9 ±1.6	70.3 ±1.6	75.3 ±0.1	75.8 ±1.4	76.9 ±1.4	76.9 ±1.0	74.2 ±0.5
ModelNet	81.0 ±0.5	70.4 ±2.4	75.9 ±1.2	OOR	OOR	74.1 ±3.0	71.9 ±2.6	77.1 ±2.6	**83.9 ±1.9**
Rank		4.43	**2.57**	7.14	4.29	4.71	3.86	4.29	4.29

4.4.3 Preserving task-specific information

In our final experiment, we consider several benchmarks of graph classification to test the third criterion. A high classification accuracy implies that an operator can selectively preserve information based on the requirements of the task at hand. We consider graph classification problems from the TUDataset [150], the ModelNet10 dataset [228], and the Colors-3 and Triangles datasets introduced by Knyazev et al. [108].

Table 4.7 reports the average and standard deviation of the classification accuracy on the test set, as well as the average ranking of the operators. We also report as a baseline the classification accuracy of a GNN with no pooling (No-

pool). We observe that, on the datasets considered here, the operators based on graph spectral properties (MinCut, NDP, Graclus) achieve the highest accuracy. However, we could not find strong evidence that one pooling operator (or even a class of operators) is systematically better than all others. For instance, on Triangles and Colors-3 we see that dense, trainable operators have a consistent advantage. However, the family of sparse and/or non-trainable methods performs better on Enzymes, Mutagenicity, and the large-scale ModelNet10 dataset. Finally, in datasets such as Mutagenicity, Proteins, and DD the performance gap is not very large. Table 4.7 also shows that some of the models with graph pooling operators achieve higher classification accuracy than the no-pool baseline (in green). In Mutagenicity, the baseline architecture achieves top performance, suggesting that graph pooling is not always beneficial in some graph classification tasks. Further discussion can be found in the recent work of Mesquita et al. [145].

4.4.4 Discussion

Overall, we showed that: 1) the choice of the best pooling operator, and whether performing graph pooling is necessary at all, highly depends on the problem at hand; 2) a comprehensive evaluation of pooling operators requires considering multiple criteria—such as those discussed in this section—to highlight all their fundamental properties and, in particular, it cannot be limited to measuring the downstream performance on small-scale benchmark datasets.

Part II

Architectures

Chapter 5

Adversarial autoencoders with constant-curvature latent manifolds

Performing statistical inference and analysis on graph datasets is a complicated task, due to the complexity added by the variable topology between samples (a variability that classical domains, like images and time signals, do not have). In particular, one of the key issues in dealing with graphs is to find representations that respect their underlying geometry, which is usually defined by application-specific distances that often do not satisfy the identity of indiscernibles or the triangular inequality [125, 222]. The use of metric distances, like graph alignment distances [93], only mitigates the problem, as they are computationally intractable and not useful in practical applications. However, recent research on the geometry of the domain of graphs [222, 94, 245] has highlighted that constant-curvature Riemannian manifolds (CCMs), like hyperspherical and hyperbolic spaces, are ideal for representing graphs and many other kinds of data [222, 154, 245, 80, 43].

Motivated by these observations, in this chapter we introduce the constant-curvature manifold adversarial autoencoder (CCM-AAE) [75, 76], a neural network architecture designed to learn representations on non-Euclidean manifolds in an unsupervised way. The proposed architecture is general and not necessarily limited to graph data, although we show that it leads to substantial improvements on many problems of statistical inference and analysis in the domain of graphs.

In reference [75], besides some benchmark applications, we consider the task of generating small molecules for *de novo* drug design. We show that the non-Euclidean geometry of the latent space promotes a more diverse generation (measured in terms of variety and novelty w.r.t. the training data). In particular,

we show that our approach achieves a significantly better generation than the state of the art (at the time of publication).

In reference [76], inspired by the work of Zambon *et al.* [244, 243, 246], we propose a change detection technique for sequences of graphs, based on the CCM embeddings provided by the autoencoder. Specifically, we develop two *ad hoc* change detection techniques that take into account the non-Euclidean geometry of the latent space. Interestingly, we show that for some applications changes emerge only on CCMs with specific curvatures. We apply our change detection pipeline to two real-world applications: 1) the detection of hostile behaviour in sequences of skeletal graphs representing humans, and 2) the detection of epileptic seizures in sequences of functional brain networks.

In the following sections, we first give the relevant background on adversarial autoencoders and CCMs, then we describe the proposed architecture, and finally, we go over the applications mentioned above.

5.1 Background

5.1.1 Adversarial autoencoders

Adversarial autoencoders (AAEs) are probabilistic models for performing variational inference, based on the framework of generative adversarial networks (GANs) [71]. In AAEs, the encoder network acts as the generator of a GAN, and the aggregated posterior of its latent representation is matched with a prior distribution, by training the model to fool the discriminator.

Training of AAEs occurs in two phases, namely *reconstruction* and *regularisation*. During the former, the AAE is trained to reconstruct samples from the data distribution. During the regularisation phase, the discriminator is trained to distinguish between samples coming from the encoder and samples coming from the prior. Finally, the encoder is updated to fool the discriminator. The repetition of these training steps results in a *min-max* game between the encoder and the discriminator, where both networks compete to improve at their respective tasks [71].

Let $p_{\text{data}}(\mathbf{x})$ be the true data distribution in some data space $\mathcal{X}$ and $p(\mathbf{z})$ the prior distribution in some latent space $\mathcal{Z}$. We indicate with $f_{\text{enc}} : \mathcal{X} \to \mathcal{Z}$ the encoder network of the AAE and with $f_{\text{dec}} : \mathcal{Z} \to \mathcal{X}$ the decoder network, so that the full autoencoder is $f_{\text{dec}} \circ f_{\text{enc}} : \mathcal{X} \to \mathcal{X}$. Finally, we indicate with $f_{\text{dis}} : \mathcal{Z} \to \mathbb{R}$ the discriminator network, which is typically a classifier.

First, during the reconstruction phase, the AAE is trained to minimise the

reconstruction error between the samples $\mathbf{x} \sim p_{\text{data}}(x)$ and the output of the autoencoder, $(f_{\text{dec}} \circ f_{\text{enc}})(\mathbf{x})$. Then, during the regularisation phase, the adversarial optimisation is formulated by Makhzani et al. [136] as:

$$\min_{f_{\text{enc}}} \max_{f_{\text{dis}}} \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} [\log f_{\text{dis}}(\mathbf{z})] + \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log(1 - (f_{\text{dis}} \circ f_{\text{enc}})(\mathbf{x})] . \tag{5.1}$$

The two training steps are then repeated iteratively until convergence.

AAEs are intuitively similar to variational autoencoders (VAEs), with the key difference that AAEs replace the Kullback-Leibler divergence penalty of VAEs with the adversarial training procedure outlined above. However, this means that AAEs do not need an exact functional form of the prior to perform backpropagation, but only a way to sample from it. This makes them more flexible in the choice of prior [136]. In our approach, we use this property of AAEs to constrain the latent space of the network to a CCM using a particular prior.

5.1.2 Constant-curvature manifolds

A d-dimensional CCM $\mathcal{M}$ is a Riemannian manifold characterised by a constant sectional curvature $\kappa \in \mathbb{R}$. We consider an extrinsic representation of $\mathcal{M}$ in its ambient space and define the CCM as

$$\mathcal{M} = \{\mathbf{x} \in \mathbb{R}^{d+1} \mid \langle \mathbf{x}, \mathbf{x} \rangle = \kappa^{-1}\}. \tag{5.2}$$

The scalar product $\langle \cdot, \cdot \rangle$ in Equation (5.2) defines the geometry of the CCM. For $\kappa > 0$, the geometry is said to be *spherical* and is defined by the inner product:

$$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^\top \mathbf{y}. \tag{5.3}$$

In this case, the geodesic distance between points is computed using Equation (5.3) as

$$\rho(\mathbf{x}, \mathbf{y}) = \arccos(\langle \mathbf{x}, \mathbf{y} \rangle). \tag{5.4}$$

For $\kappa < 0$, the geometry is said to be *hyperbolic* and the formulation of Equation (5.2) is the *hyperboloid* model. The geometry in this case is defined by the pseudo-Euclidean scalar product:

$$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^\top \begin{pmatrix} \mathbf{I}_d & 0 \\ 0 & -1 \end{pmatrix} \mathbf{y}. \tag{5.5}$$

Geodesics are computed from Equation (5.5) as

$$\rho(\mathbf{x}, \mathbf{y}) = \operatorname{arcosh}(\langle \mathbf{x}, \mathbf{y} \rangle). \tag{5.6}$$

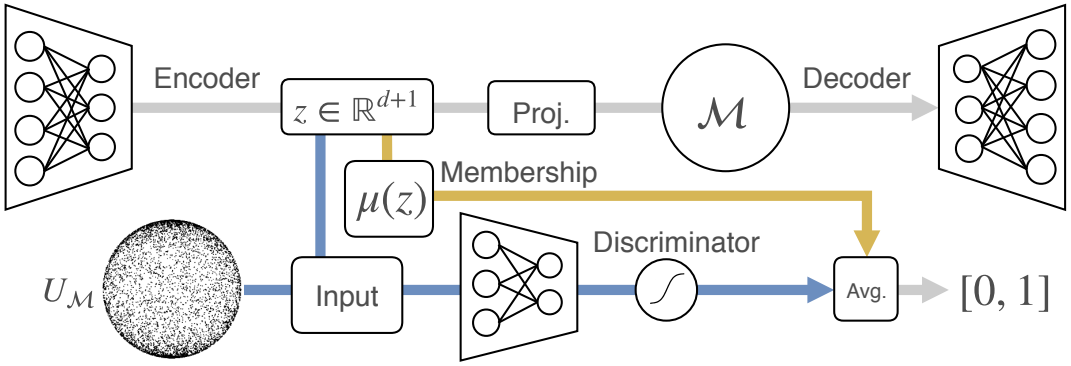


Figure 5.1. Schematic view of the spherical CCM-AAE. From left to right: the encoder produces embeddings $z \in \mathbf{R}^{d+1}$ in the ambient space, which are projected onto the CCM before being fed to the decoder. The discriminator distinguishes between the embeddings and the samples from the prior (blue path). Finally, the membership degree of the embeddings is averaged with the discriminator to compute the loss and update the encoder (yellow path)

5.1.3 Distributions on CCMs

To train the CCM-AAE and impose a geometric constraint on the representation, we need to define probability distributions on CCMs.

Let $P_{\mathcal{M}}(\boldsymbol{\theta})$ be a probability distribution with support on $\mathcal{M}$ and parametrised by vector $\boldsymbol{\theta}$. Given the tangent plane $T_{\mathbf{x}}\mathcal{M} \in \mathbb{R}^d$ at point $\mathbf{x}$, a general approach to compute $P_{\mathcal{M}}(\boldsymbol{\theta})$ is to take a probability distribution $P(\boldsymbol{\theta})$ with support on $T_{\mathbf{x}}\mathcal{M}$ and compute $P_{\mathcal{M}}(\boldsymbol{\theta})$ as the push-forward distribution of $P(\boldsymbol{\theta})$ through the Riemannian exponential map (exp-map) $\mathrm{Exp}_{\mathbf{x}}(\cdot)$ [202, 222]. Intuitively, we obtain a sample from $P_{\mathcal{M}}(\boldsymbol{\theta})$ by sampling from $P(\boldsymbol{\theta})$ and mapping the sample to $\mathcal{M}$ using the exp-map. This provides a way to compute a distribution on a CCM starting from any distribution on the Euclidean tangent space, but it is not the only possibility. For instance, mapping the uniform distribution to the spherical manifold via the exp-map leads to counter-intuitive results, whereas a correct way of computing a spherical uniform distribution is to L_2-normalise the samples from a Gaussian distribution in the ambient space.

5.2 Adversarial autoencoders on CCMs

We consider the problem of mapping a data distribution to a d-dimensional CCM $\mathcal{M}$ with curvature $\kappa \in \mathbb{R}$, as well as learning a map from $\mathcal{M}$ to the data space to generate new samples.

To achieve this, we introduce the *constant-curvature manifold adversarial autoencoder* (CCM-AAE). Using adversarial learning, we match the latent representation of the CCM-AAE to a prior distribution defined on the CCM, while jointly training the encoder to produce embeddings that lie on the CCM via an explicit regularisation term in the loss, penalising the encoder when the embeddings are far from the manifold. This facilitates the network in converging to the target manifold, making it easier to match the aggregated posterior to the prior. We show a schematic view of the proposed architecture in Figure 5.1.

The methodology presented here is independent of the type of neural network used to learn the representation of the data, and can be applied as a general regularisation technique. In this section, we provide an outline of the approach and leave the implementation details to the experiments section.

5.2.1 Method

We use the same notation adopted in Section 5.1.1, except that now the latent space $\mathcal{Z}$ is a CCM $\mathcal{M}$. The CCM-AAE is defined as the composition of two maps:

- $f_{\text{enc}} : \mathcal{X} \to \mathcal{M}$, mapping data to the manifold;
- $f_{\text{dec}} : \mathcal{M} \to \mathcal{X}$, mapping embeddings back the data space.

In practice, the latent space of the autoencoder is $\mathbb{R}^{(d+1)}$ and represents the ambient space of $\mathcal{M}$.

During the reconstruction phase, the autoencoder is trained to reconstruct samples from the data distribution, by minimising a loss function between $\mathbf{x} \in \mathcal{X}$ and $(f_{\text{dec}} \circ f_{\text{enc}})(\mathbf{x})$. During the regularisation phase, we train the discriminator to distinguish between samples from the encoder and samples from the prior $P_{\mathcal{M}}(\boldsymbol{\theta})$. Then, we update the encoder to fool the discriminator. By matching the aggregated posterior to $P_{\mathcal{M}}(\boldsymbol{\theta})$, the network is implicitly constrained to embed input data on the CCM, and the solution to the adversarial game can be obtained from Equation (5.1) as:

$$\min_{f_{\text{enc}}} \max_{f_{\text{dis}}} \mathbb{E}_{\mathbf{z} \sim P_{\mathcal{M}}(\boldsymbol{\theta})} [\log f_{\text{dis}}(\mathbf{z})] + \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log(1 - (f_{\text{dis}} \circ f_{\text{enc}})(\mathbf{x}))] . \tag{5.7}$$

However, this implicit optimisation may not be sufficient for the network to learn the non-Euclidean geometry of the CCM. Consequently, here we also train the encoder network to maximise the membership degree of the embeddings to $\mathcal{M}$, so that the CCM-AAE is steered towards computing embeddings that lie exactly on the CCM. For a manifold $\mathcal{M}$ with $\kappa \neq 0$, the membership degree of a

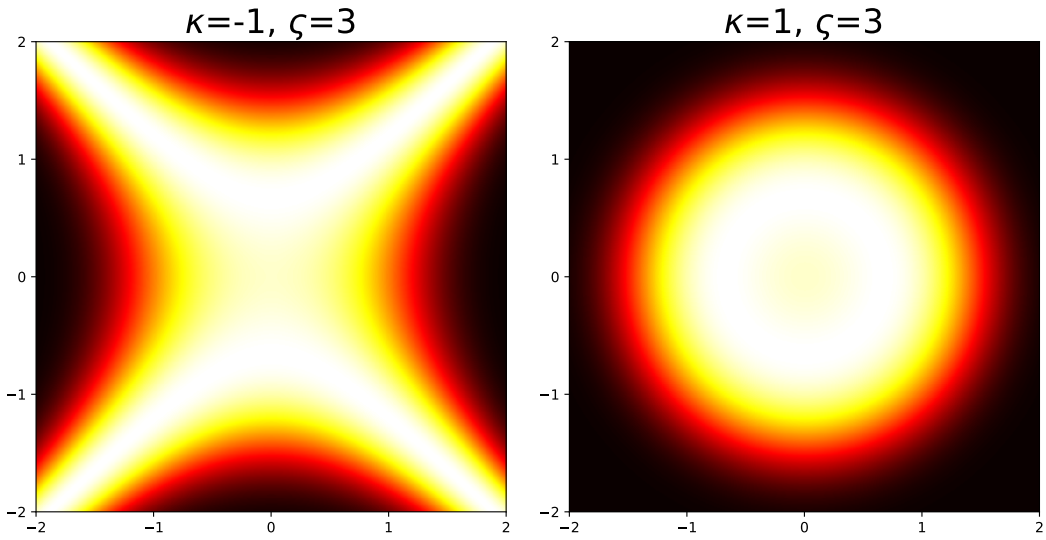


Figure 5.2. Membership function of a hyperbolic (left) and a spherical (right) CCM in the case of $d = 1$. Lighter colours indicate higher values (white = 1).

sample $\mathbf{z}$ is computed as:

$$\mu(\mathbf{z}) = \exp\left(\frac{-\left(\langle \mathbf{z}, \mathbf{z}\rangle - \frac{1}{\kappa}\right)^2}{2\varsigma^2}\right) \tag{5.8}$$

where $\varsigma \neq 0$ controls the width of the membership function (see Figure 5.2). In practice, we optimise both regularisation objectives in parallel, by taking the average of the discriminator's output and the membership degree of the embeddings when updating the encoder in the regularisation phase as:

$$\tilde{f_{\text{dis}}}(\mathbf{z}) = \frac{f_{\text{dis}}(\mathbf{z}) + \mu(\mathbf{z})}{2}. \tag{5.9}$$

The final form of the regularisation for the CCM-AAE is:

$$\min_{f_{\text{enc}}} \max_{f_{\text{dis}}} \mathbb{E}_{\mathbf{z}\sim P_{\mathcal{M}}(\boldsymbol{\theta})}\left[\log \tilde{f_{\text{dis}}}(\mathbf{z})\right] + \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\left[\log(1 - (\tilde{f_{\text{dis}}} \circ f_{\text{enc}})(\mathbf{x}))\right]. \tag{5.10}$$

Projection to the CCM When we need to compute exact operations on the CCM (*e.g.*, distances or sampling), we orthogonally project the embeddings onto $\mathcal{M}$ to compensate for the inevitable error in the model. The projection can be embedded as an operation in the computational graph of the CCM-AAE or computed only at test time. For instance, for KNN-based semi-supervised classification (*cf.*

Section 5.3), where we need to compute the pairwise distances of the embeddings, we only project the embeddings onto $\mathcal{M}$ at test time. Alternatively, when using the CCM-AAE for generating new samples in the data space, we ensure that the decoder network learns a meaningful map between $\mathcal{M}$ and the data space by always projecting the latent codes onto $\mathcal{M}$. This does not impact the regularisation of the encoder and has only a marginal effect on the convergence of the model.

5.2.2 Related works

Focusing on recent literature regarding unsupervised learning on graphs and Riemannian manifolds, we mention that graph autoencoders (GAE) are typically used for node-level prediction [107, 17, 187]; in this framework, an adversarially regularised GAE is proposed by Pan et al. [165]. For graph-level learning, instead, Simonovsky and Komodakis [196] propose a variational GAE for generating molecules.

Several works in the literature introduce different approaches to model the latent space geometry of generative models or study the geometry of the data distribution to facilitate the autoencoder in learning a non-Euclidean representation. Davidson et al. [43] introduce a variational autoencoder that uses the von Mises–Fisher distribution as prior, aimed at modelling the spherical geometry of the data. Korman [109] proposes to use the AAE framework to recover the manifold underlying a data distribution, without making assumptions on the geometry of the manifold. This is achieved by approximating the manifold as a set of charts, each represented by the latent space of a linear AAE trained to match a uniform prior. The Riemannian geometry of deep generative models is also studied by Shao et al. [191], Chen et al. [34], whereas Giryes et al. [66] study the metric-preserving properties of neural networks with random Gaussian weights. To capture the hierarchical structure of domains like natural language, Nickel and Kiela [154] develop a technique based on stochastic gradient descent on manifolds for embedding graph data on a Poincaré ball.

Advantages A key difference of our approach with other works in the literature is that we do not impose the non-Euclidean geometry on the latent space by simply projecting the embeddings onto the CCM, or otherwise explicitly limiting the latent space (*e.g.*, by sampling embeddings from the CCM prior [43]). The encoder must learn the latent manifold autonomously because the projection is not performed during the regularisation step. Moreover, as highlighted in previous sections, AAEs have the advantage w.r.t. to VAEs of not requiring the explicit

form of the prior to perform backpropagation. This is especially relevant when dealing with non-Euclidean geometry, where functional forms can be analytically complex. A relevant effect of this is discussed in Section 5.3.2, where we show that the spherical version of our model can deal with high-dimensional manifolds better than an equivalent VAE with spherical latent space. This results in a more stable performance when using high-dimensional manifolds on the considered applications (we show a specific example of this on a semi-supervised classification task on MNIST), since our model does not suffer from the same performance drop of the spherical VAE.

5.3 Benchmarks

Following the experimental methodology of Davidson et al. [43], we begin by reporting a performance comparison of different models on two relevant benchmarks of interest: semi-supervised image classification on MNIST and link prediction on citation networks.

For each experiment, we test two main configurations of CCM-AAE, *i.e.*, with spherical and hyperbolic geometry. The geometry of the CCM is dependent only on the sign of the curvature, whereas the absolute value of the curvature has only an effect on the scale of the representation. For this reason, in order to simplify the implementation of the CCM-AAE, here we only consider $\kappa = 1$ and $\kappa = -1$.

For the prior, we follow the methodology of Makhzani et al. [136] and adapt the standard normal distribution $\mathcal{N}(0,1)$ to our setting. For $\kappa = -1$ we use the push-forward standard normal $\mathcal{N}_{\mathcal{M}}(0,1)$, where the origin of the exp-map is taken as the point $\mathbf{x} \in \mathbb{R}^{d+1}$ such that $\mathbf{x}_i = 0$ for $i = 0, \dots, d-1$ and $\mathbf{x}_d = 1$ (the point is chosen to simplify some implementation details, but any other point on the manifold would be suitable).

For $\kappa = 1$ however, as the dimension d of the manifold grows, mapping $\mathcal{N}(0,1)$ to the CCM via the exp-map results in a uniform distribution on the sphere. A similar consideration was also noted by Davidson et al. [43] for the von Mises–Fisher distribution. Therefore, a better choice is to use directly the spherical uniform distribution as prior (*cf.* Section 5.1.3).

5.3.1 Semi-supervised image classification

Following the methodology of Kingma et al. [104], we evaluate the quality of the embeddings produced by the CCM-AAE on a semi-supervised classification task on the *dynamically binarised MNIST* dataset [176]. We train the CCM-AAE on a

Table 5.1. Accuracy of semi-supervised KNN classification on MNIST for 100, 600, and 1000 observed training labels per class w.r.t. the dimensionality of the latent manifold. We report mean and standard deviation computed over 10 runs.

Method	**d**	**l=100**	**l=600**	**l=1000**
VAE	10	89.1 ±0.6	92.7 ±0.5	93.3 ±0.5
$\mathcal{S}$-VAE	10	90.7 ±0.7	93.7 ±0.5	94.1 ±0.5
AAE	100	91.2 ±0.5	94.9 ±0.2	95.4 ±0.2
Ours ($\kappa = 1$)	20	91.4 ±0.4	95.0 ±0.5	95.6 ±0.3
Ours ($\kappa = -1$)	30	**91.5** ±0.3	**95.2** ±0.2	**95.8** ±0.2

random split of 55k samples for training, 10k for testing, and 5k for validation and model selection.
After training, we draw for each class $l = 100, 600, 1000$ pairs of samples and labels uniformly from the training set, and evaluate the test accuracy of a K-nearest neighbours (KNN) classifier on the embeddings produced by hyperbolic and hyperspherical CCM-AAEs.

Similar to the experimental setting of Davidson et al. [43], the encoder is a two-layer fully connected network of 256 and 128 neurons with ReLU activations, followed by a linear layer with $d+1$ neurons to produce the latent representation. The decoder has two ReLU layers with 128 and 256 units, followed by an output layer with sigmoid activations.

For hyperparameters, we perform a grid search using the validation loss to perform model selection. The tested values and final configuration are reported in Table 5.2. We adopt a fully connected discriminator with two layers of h units, leaky ReLU activation, and L2 regularisation, followed by an output layer with sigmoid activation. We train both networks using Adam until convergence, using early stopping on the autoencoder's validation loss with a look-ahead of 50 epochs (value taken from Davidson et al. [43]). For both networks, we optimise the cross-entropy between the inputs and reconstructed images.

The embeddings of the network are projected onto the manifold only at test time to compute the mutual geodesic distances for KNN (for which we set $K = 5$ as done by Davidson et al. [43]). We compare our results using the same network architecture and configuration to train a standard adversarial autoencoder with Gaussian prior (AAE) [136], a variational autoencoder with Gaussian prior (VAE) [103], and the hyperspherical variational autoencoder proposed by Davidson et al. [43] ($\mathcal{S}$-VAE). For $\mathcal{S}$-VAE, we use the open source implementation pro-

Table 5.2. Hyperparameter configuration of the CCM-AAE for MNIST. The **Searched** column indicates that the final value was found via grid search among the indicated values, using the validation loss for model selection. Alternatively, when we did not perform a grid search, we indicate how we chose the value (*Keras default* indicates that the value was the default setting for the popular deep learning library Keras, which we used for experiments). An empty final value in the **Value** column indicates that the grid search was repeated for each combination of dataset and κ.

Hyperparam.	**Value**	**Searched**
d	-	2, 5, 10, 20, 40, 60, 100
h	64	32, 64, 128
Leaky ReLU α	0.3	Keras default
L_2 reg. (for f_{dis})	0.01	Keras default
ς (for μ)	5	1, 2, 5, 10
Learning rate	0.001	0.001, 0.005, 0.01
Batch size	1024	Empirically

vided by the authors in the original paper.[1]

The best results obtained by each method are summarised in Table 5.1. We note that the adversarial setting consistently outperforms its variational counterparts, even when considering the non-Euclidean $\mathcal{S}$-VAE method. The CCM-AAE also performs slightly better on average w.r.t. the Euclidean AAE, with no significant statistical differences observed between spherical and hyperbolic embeddings. However, we note that the Euclidean model requires a significantly higher d in order to match the performance of the non-Euclidean ones, with the spherical model being the most efficient in this regard. This confirms an already observed fact in the literature [43], and a possible explanation for this phenomenon can be intuitively seen in Figure 5.3, where MNIST is shown to have a natural representation on a spherical domain.

Finally, we show in Figure 5.4 that the CCM-AAE can learn a sufficiently good representation of the data even at very low dimensions (pictured for $d = 2$ for visualisation purposes), with no substantial differences between hyperbolic and spherical geometries as also highlighted by the performance of KNN.

[1] https://github.com/nicola-decao/s-vae-tf

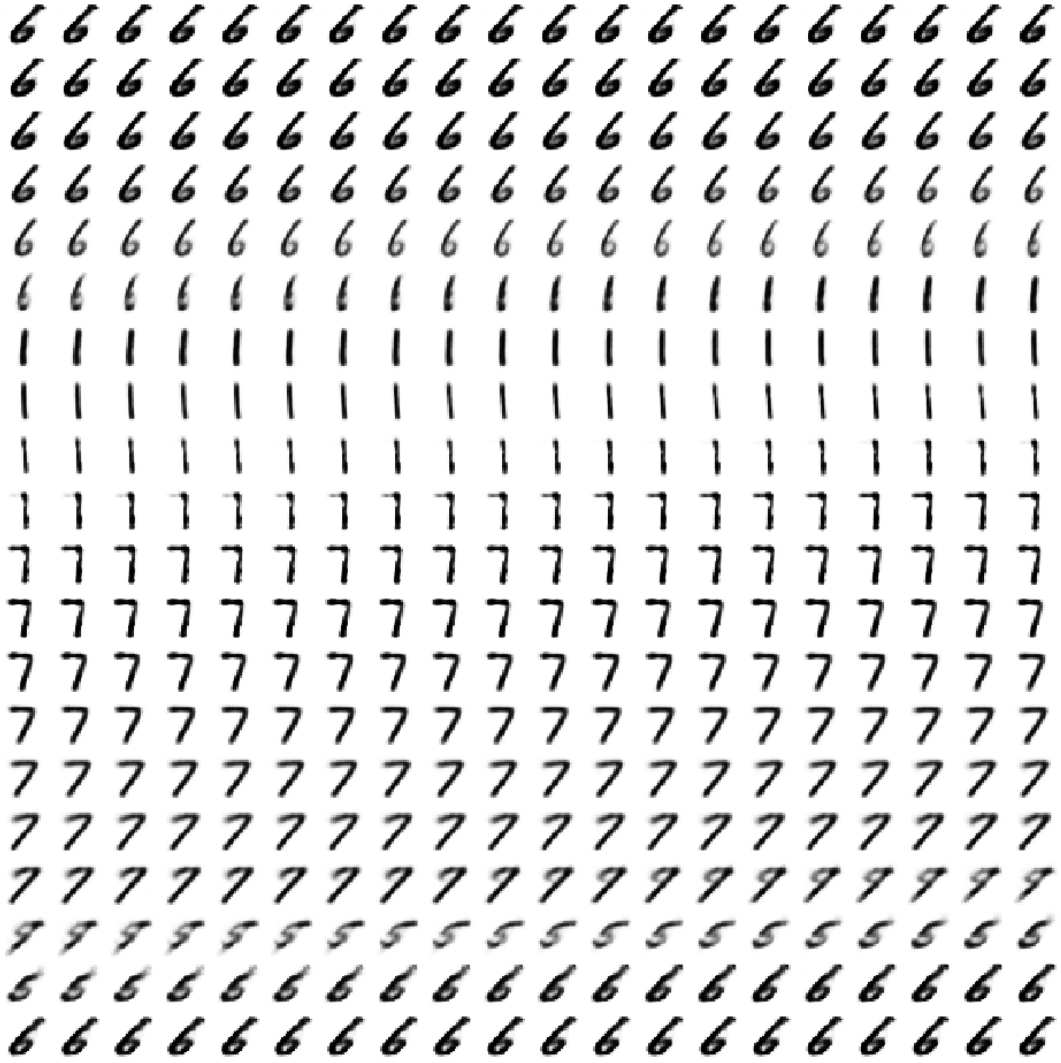


Figure 5.3. Traversing the latent space of a spherical CCM-AAE ($d = 2$) along an equator of the sphere (samples are arranged left-to-right, top-to-bottom). Note how the digits are smoothly represented in a circular way, suggesting how the data can be naturally encoded on a sphere.

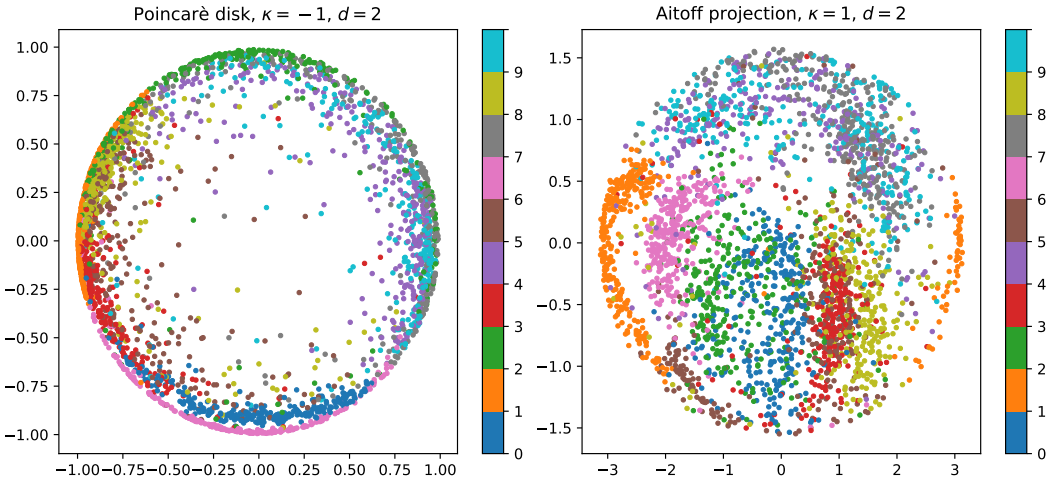


Figure 5.4. Embeddings produced by the CCM-AAE on MNIST, for $d = 2$. We report the Poincaré disk model of the latent CCM for $\kappa = -1$, and the Aitoff projection for $\kappa = 1$.

5.3.2 Link prediction

For the link prediction task, we follow the methodology of Kipf and Welling [107] and evaluate the performance of the CCM-AAE on the popular Cora and Citeseer citation network datasets.[2] In this task, we train the CCM-AAE to predict connections on a subset of the network and evaluate the area under the receiver operating characteristic curve (AUC) and average precision (AP) of the model in

[2]The Pubmed dataset is often considered alongside the other two, but its large number of nodes caused GPU memory issues with all algorithms and we could not report results for it.

Table 5.3. Best average AUC and AP of semi-supervised link-prediction on the Cora and Citeseer datasets. We report mean and standard deviation over 5 runs. Best results are not highlighted due to the differences between the best algorithms not being statistically significant.

	Cora			Citeseer		
Method	**d**	**AUC**	**AP**	**d**	**AUC**	**AP**
VGAE	20	91.8 ±0.8	92.9 ±0.6	16	90.6 ±1.3	91.7 ±1.1
AAE	64	93.4 ±0.6	93.8 ±0.7	64	94.0 ±0.8	94.6 ±1.0
Ours ($\kappa = 1$)	8	93.4 ±0.7	93.9 ±0.8	8	92.8 ±0.4	93.4 ±0.4
Ours ($\kappa = -1$)	64	89.4 ±0.9	90.4 ±1.0	8	91.0 ±0.4	91.6 ±0.4

predicting a test set of held-out links. We split the data randomly, using 10% of the links for testing and 5% for validation and model selection.

Cora and Citeseer are two popular network datasets representing citation links between documents, with sparse node attributes representing text features found in the documents. Each node is also associated with a class label, which we do not use here. We represent a network with N nodes and F-dimensional node attributes as a tuple (A, X), where $A \in \{0,1\}^{N \times N}$ is the symmetric adjacency matrix of the network, and $X \in \{0,1\}^{N \times F}$ is the node attribute matrix. For the Cora dataset we have $N = 2708$ and $F = 1433$, whereas for Citeseer we have $N = 3327$ and $F = 3703$. The networks have an average degree of 4 and 2.84, respectively.

The CCM-AAE has the same structure as the variational graph autoencoder (VGAE) [107], with a GCN encoder network followed by a scalar product decoder. The encoder consists of a graph convolutional layer with 32 channels and ReLU activations, followed by a $d+1$ dimensional graph convolutional linear layer. Dropout is applied before every layer.

For the decoder, we first project the latent representation onto the target manifold, and then we reconstruct the adjacency matrix by computing the scalar product between node embeddings, followed by an activation to normalise the output between 0 and 1. In the spherical case, the scalar product can be interpreted as computing a cosine similarity between embeddings, which we then normalise with a sigmoid activation. For the hyperbolic CCM-AAE, the pseudo-Euclidean scalar product assumes values in the $(-\infty, -1]$ range, so we normalise it to $(0,1]$ by applying a shifted exponential as activation to the decoder's output, $\sigma(x) = \exp(x+1)$. In principle, any normalisation function can be used here, but we leave further exploration of this matter as future work. For instance, an obvious way of normalising the output would be to add a final layer with sigmoid activations and let the network learn how to map the scalar product to a prediction in $(0,1)$. However, here we wanted to have the same number of parameters across all models to report a fair comparison.

We keep most of the configurations used for MNIST unvaried, but we perform a grid search over the dropout rate (0.0, 0.1, 0.2, 0.3, 0.4) for each dataset and geometry. Additionally, we repeat the grid search over the dimension d using similar values to those reported by Davidson et al. [43] (8, 16, 20, 32, 64, 128). Both networks are trained using Adam to optimise the cross-entropy (when training the autoencoder, we apply the same re-weighting technique used by Kipf and Welling [107]). We train the model using early stopping on the validation AUC with patience of 100 epochs (value taken from Kipf and Welling [107]).

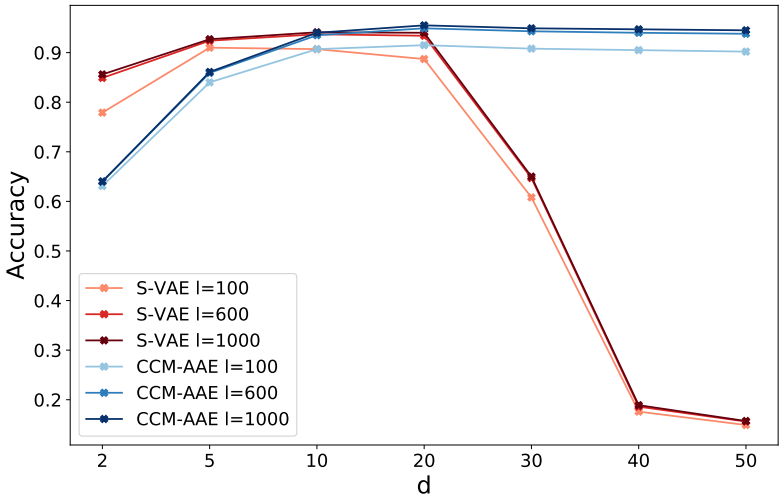


Figure 5.5. Comparison between the spherical CCM-AAE and $\mathcal{S}$-VAE considering the mean test accuracy on MNIST w.r.t. the manifold dimension. $\mathcal{S}$-VAE shows a performance collapse as reported by Davidson et al. [43], but the CCM-AAE maintains a stable performance even at higher dimensions.

We report results in Table 5.3. We compare our results against VGAE and an AAE, using the same network architecture for all models. The spherical CCM-AAE consistently outperforms VGAE in both tasks, but we highlight a drop in performance in the hyperbolic model. Once again, the Euclidean AAE performs comparably to the spherical one but requires a significantly higher-dimensional latent space.

While the code used to implement $\mathcal{S}$-VAE worked as intended on MNIST, on the link prediction task we encountered numerical issues that made it impossible to replicate the results of Davidson et al. [43] in our different experimental setting with different data splits, re-weighting technique for the loss, and hyperparameters searched. When further investigating the instability of $\mathcal{S}$-VAE, we observed a computational issue in the model, which would saturate the floating-point representation of the GPU, resulting in invalid gradients propagating through the network. The numerical instability derives from the sampling procedure of the von Mises–Fisher distribution on which the model is based, as the exponentially scaled modified Bessel function used for sampling causes a division by zero for higher values of d. On the other hand, we note that the performance of the proposed spherical CCM-AAE does not suffer from the same issue when using high-dimensional latent spaces (shown in Figure 5.5 for MNIST).

5.4 Molecule generation

Graph-based molecule generation is a recent area of research that is starting to draw the attention of the machine learning and cheminformatics communities [196, 45]. Unlike past approaches in molecule generation, most of which relied on the SMILES string representation of molecules, the graph-based approach represents atoms as nodes in a graph, with chemical bonds represented as attributed edges.

We compare the proposed CCM-AAE against several molecule generation models using the QM9 dataset of small molecules. We evaluate the performance of our model following the methodology of Simonovsky and Komodakis [196]. QM9 contains around 134.000 small molecules of up to nine heavy atoms, with four atomic numbers and three bond types. We split the data randomly using 10.000 samples for testing and 10.000 for validation and model selection.

To evaluate the quality of the model, we sample random points from the latent CCM and map them to the molecule space using the decoder. We then compute three metrics for the generated molecules: the *validity* measure indicates the fraction of chemically valid molecules, the *novelty* metric indicates the fraction of valid molecules that are not in the original QM9 dataset, and finally the *uniqueness* metric indicates the fraction of unique molecules among the valid ones [177]. Finally, to quantify the overall performance of a model, we aggregate the three metrics by multiplying them together. If we assume independence of uniqueness and novelty given validity, this is the probability of generating a valid, unique, and novel molecule. We use this aggregated *joint* metric to compare different models between them. We also empirically validated the assumption of independence among the metrics, by computing the true ratio of valid, unique, and novel molecules generated by our algorithm. The difference between the true ratio and the *joint* metric was not statistically significant in our experiments.

Molecules representation We represent molecules as attributed graphs following the approach of Simonovsky and Komodakis [196]. We use a one-hot binary representation for the attributes, where nodes are labelled with one of four possible atomic numbers and edges with one of three types of bonds. We also explicitly represent null node and edge types with a dedicated class label.

5.4.1 Setting

We structure the CCM-AAE according to the same architecture of *GraphVAE* [196], of which we consider the unconditional version for simplicity. The encoder is a

graph-convolutional network based on ECC [195] (*cf.* Section 2.3), composed of two layers of 32 and 64 channels, with ReLU activation, batch normalisation, and a kernel-generating network with a single linear layer. The convolutions are followed by a global gated attention readout [124] with 128 units and a linear layer with $d + 1$ units to map the representation to the ambient space of the CCM. The decoder is a fully connected network with three layers of 128, 256, and 512 neurons, with ReLU activation, and batch normalisation, followed by three parallel output layers to produce the reconstructed $\mathbf{A}$, $\mathbf{X}$, and $\mathbf{E}$ matrices of the graph (respectively, adjacency matrix, node features, and edge features). The first output layer has sigmoid activations, whereas the latter two have node- and edge-wise softmax activations, respectively. We project the embeddings to the CCM before feeding them to the decoder. The configuration of the discriminator and training procedure is again unvaried. The network is trained until convergence, monitoring the validation reconstruction loss with a look-ahead of 25 epochs (*i.e.*, the number of epochs used to train GraphVAE [196]).

Following Simonovsky and Komodakis [196], we train the CCM-AAE using graph matching to account for graphs with unidentified nodes. This consists of matching the input graphs to the generated outputs before computing the loss, so that the network learns a permutation-invariant representation. We apply the same max-pooling matching algorithm used for GraphVAE [35], with 75 iterations and the same affinity function. We also implement the same tricks to speed up training:

1. We impose the symmetry of the output matrices by removing those edges for which $\mathbf{a}_{ij} \neq \mathbf{a}_{ji}$;
2. We include in the prediction the maximum spanning tree on the set of probable nodes, *i.e.*, those for which $\mathbf{a}_{ii} \geq 0.5$;
3. We ignore hydrogen atoms and only add them as padding during chemical validation.[3]

Finally, we apply a re-weighting of the loss function to mitigate the importance of the null nodes and edges, and to improve the reconstruction of the rarer types of edges. We compute the weights from the dataset-wide inverse document frequency (IDF) score of each element in $\mathbf{A}$, $\mathbf{X}$ and $\mathbf{E}$. For instance, the IDF score of the adjacency matrix across a dataset $\mathcal{D}$ is:

$$\mathrm{IDF}_{ij} = \log\left(\frac{|\mathcal{D}|}{1 + \sum_{k=1}^{|\mathcal{D}|} \mathbf{a}_{ij}^{(k)}}\right) \tag{5.11}$$

[3] We used the RDKit framework for chemical validation and hydrogen padding.

Table 5.4. Validity, uniqueness, and novelty metrics on QM9, with and without graph matching. The **Joint** column shows the aggregated score computed as the product of the three metrics, providing a general idea of the overall performance of the models. Baseline results are taken from references [196, 45]. The best individual metrics and the model with the best *joint* score are highlighted in bold.

	Method	d	Valid	Uniq.	Novel	Joint
No match	GraphVAE	80	81.0	61.0	24.1	11.9
	CVAE	60	10.3	67.5	90.0	6.3
	GVAE	20	60.2	9.3	80.9	4.5
	MolGAN	-	**98.1**	10.4	**94.2**	9.6
	AAE	80	30.1	**92.7**	84.8	23.7
	Ours ($\kappa = 1$)	80	36.3	92.6	87.1	**29.2**
	Ours ($\kappa = -1$)	80	22.5	86.1	70.2	13.6
Match	GraphVAE	80	55.7	66.0	61.6	26.1
	GraphVAE/imp	40	56.2	42.0	75.8	17.9
	AAE	40	13.8	87.1	66.6	8.0
	Ours ($\kappa = 1$)	20	18.0	91.7	78.3	12.9
	Ours ($\kappa = -1$)	5	19.1	50.7	76.5	7.4

and the log-loss between $\mathbf{A}$ and its reconstruction $\mathbf{A}'$ is re-weighted as:

$$\log(\mathbf{A}' \mid \mathbf{A}) = \sum_{i,j} \left(1 + \mathbf{a}_{ij} \mathrm{IDF}_{ij}\right) \left(\mathbf{a}'_{ij} \log \mathbf{a}_{ij} + (1 - \mathbf{a}'_{ij}) \log(1 - \mathbf{a}_{ij})\right). \tag{5.12}$$

To account for the node permutations, the IDF weight matrices are matched to their respective target matrices before computing the loss.

5.4.2 Results

We report our results along with the results of several models for molecule generation, namely GraphVAE, MolGAN [45], the character-based CVAE [70], and the grammar-based GVAE [111] (the latter two use SMILES representations). We compare algorithms with and without the graph matching step, and we report the performance of our model in both cases.

We report a comparison of the models in Table 5.4. The spherical CCM-AAE without graph matching achieves the best performance for the *joint* metric, al-

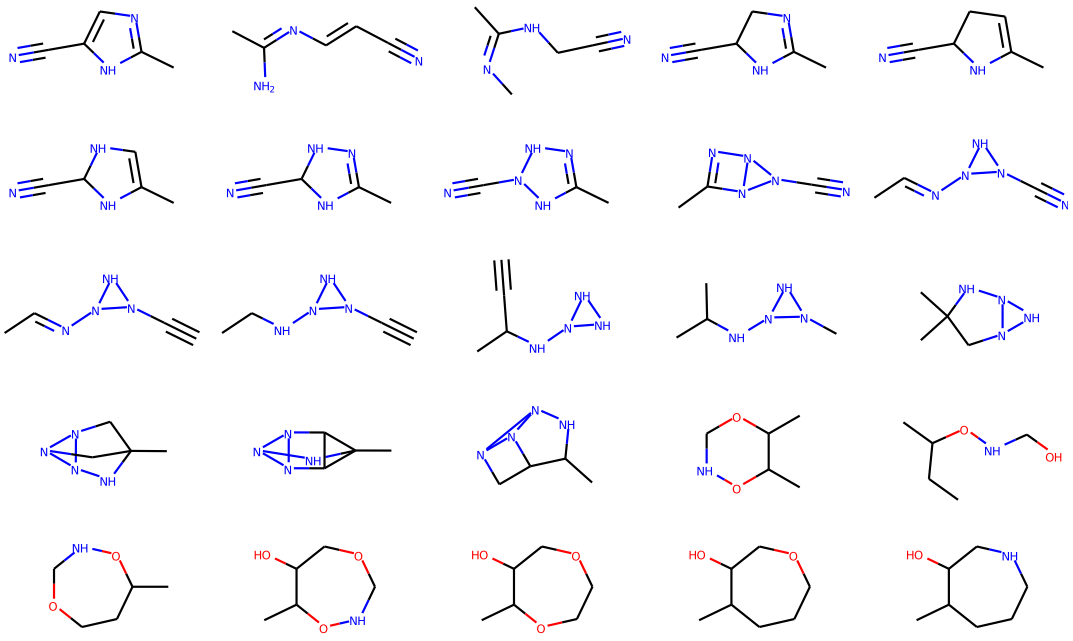


Figure 5.6. Traversing the latent space of a spherical CCM-AAE ($d = 80$) along an equator of the sphere (samples are arranged left-to-right, top-to-bottom). We show only valid, unique, and novel molecules.

though the performance on individual metrics is never better than the other models taken into account. All models have unbalanced performance across the three metrics (with GraphVAE being the most balanced), and we note that the CCM-AAE suffers from a low validity score. This confirms the effects of graph matching observed for GraphVAE, and explains the poor performance of the CCM-AAE when graph matching is considered (validity is halved in the spherical case). We note, however, that our model does not suffer from mode collapse, a problem commonly observed in generative adversarial networks as confirmed by the low uniqueness score of MolGAN. Finally, sampling from the latent manifold learned with the CCM-AAE produces a smooth transition in molecule space (Figure 5.6); however, the properties of the generated molecules are less interpretable than in the MNIST case, and we leave a more extended analysis of these results as future research.

5.5 Change detection

In many application scenarios, graphs are assumed to be generated by a stationary process, implying that the topology and graph attributes are drawn from

a fixed, albeit unknown, distribution [152]. However, the stationarity assumption does not always hold, with relevant examples including cyber-physical systems [5], functional networks associated with brain imaging [84], and many others, *e.g.*, see the works in references [52, 142, 121, 123].

In this section, we focus on the problem of detecting changes in stationarity in a sequence of attributed graphs, *i.e.*, monitoring whether the common assumption of i.i.d. observations breaks. We propose a methodology based on the CCM-AAE to represent a graph sequence as a sequence of points on a CCM, and we present two novel change detection tests (CDTs) operating on CCMs. The first CDT monitors the geodesic distances of each embedded graph w.r.t. the sample Fréchet mean observed in the nominal regime of the process. The resulting stream of distance values is processed by a CDT based on the Central Limit Theorem (CLT). The second method considers embeddings lying on the CCMs, and builds on a novel CDT based on the CLT for Riemannian manifolds [18].

We will show a comparative analysis of the proposed change detection mechanisms by testing on two real-world tasks:

1. The detection of ictal and preictal phases in epileptic seizures, using functional connectivity networks extracted from intracranial electroencephalography data;

2. The detection of hostility between two human subjects, using skeletal graphs extracted from video frames.

We show that our methodology can effectively exploit the non-Euclidean geometry of CCMs to detect changes in graph streams, consistently outperforming baseline algorithms.

5.5.1 Related works

The problem of detecting changes in graph sequences is relatively unexplored in the literature, with most works focusing on graphs with a fixed topology and without attributes [87]. Literature reviews on detecting changes, anomalies, and events in temporal networks can be found in the works of Akoglu and Faloutsos [3], Ranshous et al. [172], Akoglu et al. [4]. Some notable contributions include the matrix-decomposition-based algorithm of Sun et al. [203], the change point methods of Barnett and Onnela [11], Peel and Clauset [166] for large-scale and correlation networks, and the block-model of Wilson et al. [221] used to monitor a co-voting network over time. More recently, Zambon et al. [243] proposed a

theoretical framework for change detection in graph streams based on embedding techniques. To the best of our knowledge, their work was the first to address the problem by considering each graph in the stream as a random variable, which allowed them to perform change detection using classical statistically motivated methods. We note that none of the cited works applies modern deep learning methods to compute graph embeddings, resorting to either feature extraction or dissimilarity-based representations.

5.5.2 Method

Let $\mathcal{G}(t) = \{\mathcal{G}_1, \dots, \mathcal{G}_t, \dots\}$ indicate a random process that, at every time step t, generates a graph $\mathcal{G}_t$. We consider the task of determining whether the probability distribution underlying such a graph-generating process has changed, from the nominal distribution Q_0 to a non-nominal distribution Q_1. Our methodology consists of training a CCM-AAE to compute graph embeddings, and then running a change detection test in the non-Euclidean embedding space.

The algorithm has a training and an operational phase. During the training phase, we observe a finite stream of graphs, $G^{(\text{train})} = [\mathcal{G}_1, \dots, \mathcal{G}_T]$, generated from the nominal distribution Q_0. We map the training stream to the CCM using the encoder network, and we perform a statistical analysis to configure the CDT. In the operational phase, we monitor the graph-generating process, which is again mapped to the CCM using the encoder, with the aim of raising an alarm when a change in stationarity occurs.

The general test hypotheses considered for detecting a change in stationarity in the distribution of a sequence of i.i.d. graphs $\mathcal{G}(t)$ observed during the operational phase are

$$\begin{aligned} H_0 &: \mathcal{G}_t \sim Q_0, \ t = 1, 2, \dots \\ H_1 &: \mathcal{G}_t \sim \begin{cases} Q_0 & t < \tau \\ Q_1 & t \geq \tau, \end{cases} \end{aligned} \tag{5.13}$$

where τ indicates the change point of the sequence. Note that Q_0, Q_1, and τ are unknown. During the operational phase, we use the encoder network of the CCM-AAE to convert the incoming graph stream into a multivariate stream of embeddings $\mathbf{z}(t) \in \mathcal{M}_\kappa$, which we then monitor using a sequential statistical test to detect a possible change in the nominal distribution. Similarly, the training observations $G^{(\text{train})}$ are converted to a sequence of embeddings $Z^{(\text{train})}$.

Our change detection methodology builds on the CDT proposed by Zambon et al. [243], extending it to the case of CCMs. More in detail, the CDT considers a generic stream of vector points $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_t, \dots$, which we process in windows

of n points at a time, so that for each $w = 1, 2, 3, \ldots$, a window $[\mathbf{u}]_w$ containing $\mathbf{u}_{(w-1)n+1}, \ldots, \mathbf{u}_{wn}$ is generated, and a statistic S_w is computed by means of the accumulation process of the cumulative sums (CUSUM) chart [162]. Statistic S_w has a global role, as it recurrently accumulates information from *local* statistics $s_i = s([\mathbf{u}]_i)$, for $i = 1, \ldots, w$, as

$$S_w = \max\{0, S_{w-1} + s_w - q\}, \tag{5.14}$$

with $S_0 = 0$ and q a parameter for tuning the sensitivity of the test. The null hypothesis H_0 is rejected any time S_w exceeds a threshold h_w, and the algorithm raises an alarm indicating that a change has been detected; the accumulator S_w is then reset to 0. After the first alarm is raised, the change point is estimated as

$$\hat{\tau} = n \cdot \min\{w \mid S_w > h_w\}. \tag{5.15}$$

Threshold h_w is set according to a user-defined significance level α, by requiring, under the null hypothesis H_0, that

$$\mathbb{P}(S_w > h_w \mid H_0, S_i \leq h_i, i < w) = \alpha. \tag{5.16}$$

The threshold is set so that the probability of having a false alarm at the generic step w is α, allowing us to control the false-positive detection rate.

Note that the scoring function $s_w = s([\mathbf{u}]_w)$ entirely defines the behaviour of the CDT and that by knowing the distribution of s_w we can compute the threshold h_w given α. Here, we consider s_w to be the Mahalanobis distance

$$s_w = (\mathbb{E}[\mathbf{u}] - \overline{[\mathbf{u}]}_w)^\top \mathrm{Cov}[\mathbf{u}]^{-1} (\mathbb{E}[\mathbf{u}] - \overline{[\mathbf{u}]}_w), \tag{5.17}$$

between the sample mean $\overline{[\mathbf{u}]}_w$ of $[\mathbf{u}]_w$ and the expected value $\mathbb{E}[\mathbf{u}]$ of $\mathbf{u}$. In the stationary case, thanks to the CLT, it can be shown that $n \cdot s_w \sim \chi^2$.

We propose two different ways of computing the points $\mathbf{u}_i$, both exploiting the geometry of the CCMs. By monitoring the mean of the sequence, we can detect changes in the distribution driving the graph-generating process. Since we use graph convolutions in the encoder network, changes in the distribution of $\mathbf{A}$, $\mathbf{X}$, and $\mathbf{E}$ are all reflected on the embeddings and can be detected by the CDTs.

Distance-based CDT (D-CDT) The first proposed CDT considers the nominal distribution F_0 of the training stream of embeddings, derived as the push-forward

distribution of Q_0 through the encoder network. The Fréchet mean of F_0, denoted as μ_0, is estimated over the training sequence $Z^{(\text{train})}$ as

$$\mu_0 = \underset{\mathbf{z}\in\mathcal{M}_\kappa}{\text{ArgMin}} \sum_{\mathbf{z}_t\in Z^{(\text{train})}} \rho(\mathbf{z}_t, \mathbf{z})^2, \tag{5.18}$$

where $\rho(\cdot,\cdot)$ is the geodesic distance as defined in Section 5.1.2.

For each embedding $\mathbf{z}_t \in \mathcal{M}_\kappa$ in the operational stream, then, we consider $\mathbf{u}_t = \rho(\mu_0, \mathbf{z}_t)$. The resulting sequence $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_t, \ldots$ is finally monitored with the CDT presented above.

Riemannian CLT-based CDT (R-CDT) Our second implementation of the CDT builds on a Riemannian version of the CLT proposed by Bhattacharya and Lin [18], which adapts the Mahalanobis distance of Equation (5.17) to non-Euclidean manifolds. In this case, the operational stream of embeddings $\mathbf{z}_t \in \mathcal{M}_\kappa$ is mapped to the tangent space $T_{\mu_0}\mathcal{M}_\kappa$ with

$$\mathbf{u}_t = \text{Log}_{\mu_0}(\mathbf{z}_t), \tag{5.19}$$

and the usual CDT is applied using the modified local statistic s_w. In the case of $\kappa = 0$, the standard CLT applies directly to the embeddings without modifying s_w.

Setting CDT parameters The literature suggests setting q as half of the increase in $\mathbb{E}[s_w]$ that the designer expects to observe [148]. It is possible to show that the change detection procedure can identify any change of magnitude larger than q, independently from significance level α: to every α is associated a threshold h and the expected time of detection is

$$\hat{t} < \frac{h}{(\mathbb{E}[s_w \mid H_1] - q)}, \tag{5.20}$$

where $\mathbb{E}[s_w \mid H_1]$ is the expected value of s_w in the non-nominal conditions. Although in principle it is possible to detect arbitrarily small shifts by setting $q = \mathbb{E}[s_w \mid H_0]$, we suggest avoiding this setting because any (even small) bias introduced at training time in estimating $\mathbb{E}[s_w \mid H_0]$ will eventually trigger a false alarm.

Parameter α corresponds to type-I errors of the statistical test, that is, the probability of rejecting H_0 when H_0 is known to be true. Parameter α, is therefore directly related to the false alarm rate, with smaller values of α corresponding to fewer false alarms. However, we note that a smaller α corresponds also to larger

delays of detection under H_1. Depending on the application, the user should determine the best trade-off and the tolerable rate of false alarms.

Finally, the size n of the windows processed by the CDT should be large enough to consider $n \cdot s_w \sim \chi^2$, thus yielding the desired significance level α. However, processing larger windows of observations in the operational phase of the algorithm will result in a lower time resolution.

Ensemble of CCMs In most applications, we do not have prior information about the optimal CCM for embedding the data distribution and choosing the optimal CCM for a specific task may not be trivial. Therefore, here we propose to use an ensemble of CCMs, each characterised by a different curvature. We denote the ensemble of CCMs using the product space notation as $\mathcal{M}_* = \mathcal{M}_{\kappa_1} \times \cdots \times \mathcal{M}_{\kappa_i} \times \cdots \times \mathcal{M}_{\kappa_c}$. In practice, we consider each manifold separately and train the CCM-AAE to optimise the latent representation in parallel on each CCM. Adapting the CCM-AAE to the ensemble case is as simple as considering c parallel fully connected layers after the readout in the bottleneck, each producing a representation in a $(d+1)$-dimensional ambient space; when $\kappa = 0$, we assume that $\mathcal{M}_0$ has dimension $d+1$, rather than d. We concatenate the embeddings in a single $c(d+1)$-dimensional vector before feeding them to the discriminator. Similarly, the prior is the concatenation of c samples $\mathbf{z}_i \sim P_{\mathcal{M}_{\kappa_i}}(\theta_i)$, one for each CCM. To compute the membership function of Equation (5.8) given an embedding $\mathbf{z} = \mathbf{z}_1 \| \ldots \| \mathbf{z}_c$, we compute the membership for each CCM separately and take the average:

$$\mu_{\mathcal{M}_*}(\mathbf{z}) = \frac{1}{c} \sum_{i=1}^{c} \mu_{\mathcal{M}_{\kappa_i}}(\mathbf{z}_i). \tag{5.21}$$

We also perform the orthogonal projection of the embeddings separately for each CCM.

Accordingly, we also adapt the CDTs to consider the ensemble of CCMs. For D-CDT, we compute for each CCM the same distance-based representation as in the single CCM case. This results in a multivariate stream of c-dimensional vector of distances, which can be monitored by the base CDT. Similarly, we adapt R-CDT by applying it separately on each CCM $\mathcal{M}_{\kappa_i}$. The ensemble of statistical tests raises an alarm any time at least one of the individual tests detects a change. Since the tests are generally not independent, we apply a Bonferroni correction [24] to each R-CDT, so that the overall significance level is at least the user-defined level α.

5.5.3 Setting

We test our methodology by considering three different CCMs, namely the Euclidean $\mathcal{M}_0$, hyperspherical $\mathcal{M}_1$, and hyperbolic $\mathcal{M}_{-1}$ manifolds. For $\mathcal{M}_1$ and $\mathcal{M}_{-1}$, we take $d = 2$ and, accordingly, a three-dimensional ambient space. By choosing a low-dimensional manifold, we encourage the encoder to learn an abstract representation of the graphs and we can visualise the representations learned by the CCM-AAE for a qualitative assessment of the algorithm. For $\mathcal{M}_0$, we keep the architecture unchanged and consider a three-dimensional latent space. Since we are unable to identify *a priori* the best curvature for the problems taken into account, we also consider an ensemble composed of all three geometries, $\mathcal{M}_* = \mathcal{M}_{-1} \times \mathcal{M}_0 \times \mathcal{M}_1$. Note that the specific values of κ are only important for their sign, which determines the geometry of the CCMs. Since we are not interested in imposing any other constraint on the representation (*e.g.*, minimising the distortion introduced by the embedding process [245]), we can safely ignore the magnitude of the curvature as it only affects the scale of the representation. Thus, we choose $\kappa = -1, 0, 1$ to simplify the implementation of the experiments.

Unlike the previous experiments, here using the CCM-AAE to condition the distribution of the embeddings by matching the prior is not necessarily a good strategy, since preserving the original distribution of the graph stream is crucial in detecting a change. Therefore, we propose to train the CCM-AAE using only the membership function as regularisation, so that the discriminator of Equation (5.9) is effectively only Equation (5.8). We refer to this setting as the *geometric* discriminator. To validate our assumption, we run the experiments considering both kinds of discriminator and, for each setting, we run both D-CDT and R-CDT.

The reference baseline is that of Zambon et al. [243], for which we use the open-source implementation published by the authors.[4] There, we use a $(d+1)$-dimensional dissimilarity representation for the embedding.

More details on the experimental setting are reported in Appendix D.

Performance metric for CDTs To evaluate the detection performance of a CDT, we consider the predictions of the algorithm (*i.e.*, whether or not it raises an alarm) for each point of the operational stream, and compare them with the ground truth (*i.e.*, whether or not a change has occurred at a given time). In this setting, accuracy is not a fair performance indicator for the proposed CUSUM-

[4] https://github.com/dan-zam/cdg

based algorithms, because the detection delay of the CDT (due to the accumulation process) may result in low true positive rates even if the change is consistently detected by the algorithm. To avoid this issue, we consider the *run lengths* (RLs) of the CDT, defined as the number of time-steps between any two consecutive alarms. In the nominal regime, we configure the CDT to have a false positive rate of α and, accordingly, the average RL is $1/\alpha$. Conversely, in the non-nominal regime the detection rate should be significantly higher (ideally 1), and the average RL should be lower than the one under the nominal distribution. Therefore, by comparing the distributions of RLs in the two regimes, we can quantify the performance of the CDT.

We test whether nominal RLs are statistically larger than non-nominal ones according to the Mann-Whitney U test [137]. The resulting U statistic is then normalised to obtain the Area Under the receiver operating characteristic curve (AUC) score, which in our case measures the separability of the two RL distributions,

$$\mathrm{AUC}_{\mathrm{RL}} = \frac{U}{N_0 N_1}, \tag{5.22}$$

where N_0 and N_1 are the sample sizes of the observed RLs in the two regimes, respectively. This metric allows us to compare different algorithms operating on the graph streams and is easy to compute starting from the alarms raised by the CDTs over time.

5.5.4 Seizure detection

As a first real-world application to test our methodology, we consider iEEG data from Kaggle's *UPenn and Mayo Clinic's Seizure Detection Challenge* (SDC)[5] and the *American Epilepsy Society Seizure Prediction Challenge* (SPC).[6] We provide a summary of the SDC and SPC datasets in Table 5.5. In these datasets, iEEG signals are given as one-second clips in one of two different classes, namely the nominal *interictal* class and the non-nominal *ictal* class (or *preictal* in SPC). The datasets are collected from dogs and human patients, with a variable number of sensors for each patient, resulting in multivariate streams of different dimensions. For SDC, we consider only subjects with more than 1000 labelled clips, while for SPC we consider those with more than 500 (due to the datasets of SPC being smaller, with some patients having as little as 42 labelled clips). Functional networks (FNs) are widely used in neuroscience [12] to represent the coupling between

[5] https://www.kaggle.com/c/seizure-detection
[6] https://www.kaggle.com/c/seizure-prediction

the activity of different brain regions. By computing FNs over subsequent windows of iEEG data, we obtain a stream of attributed graphs with varying topology and attributes, with changes in the stream corresponding to the occurrence of seizures. We will cover the subject of FNs for detecting and localising seizures more in detail in Chapter 6.

Details We generate the training and operational streams for each patient using the labelled training clips in the datasets. We generate the streams by bootstrapping, sampling interictal graphs in the nominal regime and ictal (or preictal) graphs in the non-nominal regime.

We generate graphs from each one-second multivariate stream. As a first preprocessing step, we use a Butterworth filter to remove the baseline 60 Hz from the recording devices. The number of nodes N in the graphs corresponds to the number of channels in the iEEG recordings (see Table 5.5). We estimate functional connectivity between channels in the high-gamma band (70-100 Hz), such that $\mathbf{E} \in \mathbb{R}^{N \times N \times 1}$. We report experimental results using two different measures: 1) Pearson correlation and 2) the Directed Phase Lag Index (DPLI) [12]. Finally, to encode information about each channel in the node attributes, we consider the first four wavelet coefficients of the discrete wavelet transform of the related signals [12]. As a final preprocessing step, we sparsify the FNs by removing those edges with absolute connectivity below 0.1 (otherwise, the FNs would be fully connected, reducing the effectiveness of the graph convolutions). We consider a training stream of 5.000 graphs, and an operational stream of 20.000 graphs with $\tau = 10.000$.

Results The proposed method performs well on iEEG data, where the CCM ensemble with R-CDT and the geometric discriminator outperforms the single-curvature setting and the baseline on most patients. In Table 5.6, we report the results obtained with Pearson's correlation as the functional connectivity measure. Using DPLI results in slightly worse performance on average (see Table 5.7). Unlike correlation, DPLI is a directed measure of connectivity, indicating that undirected measures might be more suitable for detecting changes in this case. We note that the spherical CCM has a marginal advantage w.r.t. the other configurations on P1, indicating that single CCMs can be effective in some cases. We also note the poor performance achieved by all models on subject P2. Here, when considering non-nominal graphs, the representation learned by the encoder collapses around the mean value of the nominal regime (a phenomenon known as mode collapse), resulting in poor performance. Adding dropout be-

Table 5.5. Summary of the iEEG datasets. We report the ID that we use to identify subjects, the original ID from the datasets, the number of graphs in the nominal and non-nominal distributions, and the number of nodes for each subject.

Dataset	ID	Original ID	Graphs Q_0	Graphs Q_1	N
SDC	D1	Dog 2	1148	172	16
	D2	Dog 3	4760	480	16
	D3	Dog 4	2790	257	16
	D4	Patient 2	2990	151	16
	D5	Patient 5	2610	135	64
	D6	Patient 6	2772	225	30
	D7	Patient 7	3239	282	36
SPC	P1	Dog 2	500	42	16
	P2	Dog 3	1440	72	16
	P3	Dog 4	804	97	16

tween the ECC layers in the encoder mitigates the issue, but is still not sufficient to achieve results as good as for the other patients.

5.5.5 Detection of hostile behaviour

As a second application, we consider the task of action recognition using graphs that represent the skeletal structure of human beings. In line with our proposed method, the task is to detect when the stream of skeletal graphs changes from a nominal action performed by the subjects to a non-nominal one. Practical applications of this setting include the surveillance of public places for security purposes, the detection of a distracted driver, or the detection of incidents for people at risk (*e.g.*, children and the elderly). For this experiment, we focus on one of such tasks: detecting whether the interaction between two subjects changes from friendly to hostile. Because skeletal data provides information and constraints that are not explicitly encoded in the raw video, approaches based on GNNs have achieved state-of-the-art results in action recognition [235], surpassing the typical deep learning algorithms.

Details For this experiment, we consider the NTU RGB+D dataset for action recognition, a large collection of video samples containing RGB images, infrared

Table 5.6. AUC score on seizure detection, using Pearson's correlation as functional connectivity measure. Best results are in bold.

						SDC				SPC		
CCM	**CDT**	**Emb.**	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**P1**	**P2**	**P3**
$\mathcal{M}_*$	D-CDT	Geom.	0.99	0.99	0.99	0.93	0.99	0.66	0.99	0.22	0.19	0.87
		Full	0.98	0.99	0.99	0.92	0.99	0.43	0.98	0.10	0.15	0.83
	R-CDT	Geom.	**1.00**	**1.00**	**1.00**	0.94	**1.00**	0.74	**1.00**	0.54	0.51	0.97
		Full	**1.00**	**1.00**	**1.00**	**0.96**	**1.00**	0.79	0.98	0.40	0.45	**0.98**
$\mathcal{M}_{-1}$	D-CDT	Geom.	**1.00**	0.99	0.99	0.94	0.99	0.62	**1.00**	0.64	0.25	0.90
		Full	0.99	0.99	0.99	0.90	0.99	0.34	0.95	0.78	0.16	0.92
	R-CDT	Geom.	0.99	0.99	0.99	0.93	0.98	0.85	0.99	0.25	0.28	0.90
		Full	0.97	0.99	0.97	0.83	0.98	0.82	0.96	0.58	0.28	0.95
$\mathcal{M}_0$	D-CDT	Geom.	0.99	0.99	0.99	0.81	0.99	0.00	0.99	0.44	0.09	0.78
		Full	0.99	0.99	0.99	0.74	0.97	0.59	0.99	0.76	0.10	0.85
	R-CDT	Geom.	0.92	0.96	0.93	0.83	0.97	0.61	0.93	0.17	0.17	0.74
		Full	0.91	0.93	0.88	0.62	0.97	0.71	0.96	0.15	0.53	0.65
$\mathcal{M}_1$	D-CDT	Geom.	0.99	0.99	0.99	0.94	0.88	0.65	0.89	0.67	0.15	0.91
		Full	0.99	0.99	0.99	0.90	0.99	0.33	0.98	**0.87**	0.48	0.82
	R-CDT	Geom.	0.99	0.99	0.99	**0.96**	0.99	0.85	0.97	0.57	0.20	0.95
		Full	0.99	0.99	0.99	0.95	0.99	0.53	0.99	0.64	0.73	0.91
$\mathcal{M}_0$	R-CDT	[243]	0.92	0.74	0.84	0.90	0.90	**0.88**	0.79	0.73	**0.84**	0.90

Table 5.7. AUC score on seizure detection, using DPLI as functional connectivity measure. Best results are in bold.

						SDC					SPC	
CCM	**CDT**	**Emb.**	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**P1**	**P2**	**P3**
$\mathcal{M}_*$	D-CDT	Geom.	0.99	0.99	0.99	0.72	0.98	0.15	0.69	0.20	0.30	0.62
		Full	0.98	0.99	0.99	0.74	0.98	0.30	0.63	0.18	0.32	0.61
	R-CDT	Geom.	**1.00**	0.99	**1.00**	0.81	**0.99**	0.68	0.81	0.63	0.61	0.73
		Full	**1.00**	**1.00**	0.99	**0.82**	0.98	0.53	**0.85**	0.55	0.65	0.73
$\mathcal{M}_{-1}$	D-CDT	Geom.	0.99	0.99	0.99	0.38	0.98	0.44	0.12	0.00	0.22	0.51
		Full	0.99	0.99	0.99	0.51	0.98	0.18	0.30	0.00	0.21	0.63
	R-CDT	Geom.	0.91	0.96	0.90	0.61	0.84	0.29	0.51	0.37	0.50	0.56
		Full	0.89	0.91	0.88	0.52	0.81	0.54	0.55	0.48	0.47	0.55
$\mathcal{M}_0$	D-CDT	Geom.	**1.00**	**1.00**	**1.00**	0.79	**0.99**	0.34	0.84	0.30	0.37	0.57
		Full	0.97	0.99	0.96	0.61	**0.99**	0.70	0.32	0.35	0.37	0.66
	R-CDT	Geom.	0.99	0.99	0.99	0.81	**0.99**	0.51	0.84	0.24	0.49	0.63
		Full	0.94	0.99	0.98	0.70	0.97	0.51	0.68	0.41	0.39	0.66
$\mathcal{M}_1$	D-CDT	Geom.	0.99	0.99	0.99	0.70	0.94	0.38	0.72	0.31	0.50	0.48
		Full	0.90	**1.00**	0.97	0.58	0.96	0.41	0.41	0.41	0.52	0.50
	R-CDT	Geom.	0.99	0.99	0.99	0.72	0.96	0.28	0.84	0.27	0.51	0.65
		Full	0.99	0.99	0.98	0.70	0.96	0.42	0.56	0.44	0.54	0.51
$\mathcal{M}_0$	R-CDT	[243]	0.92	0.69	0.78	0.90	0.90	**0.82**	0.77	**0.73**	**0.78**	**0.90**

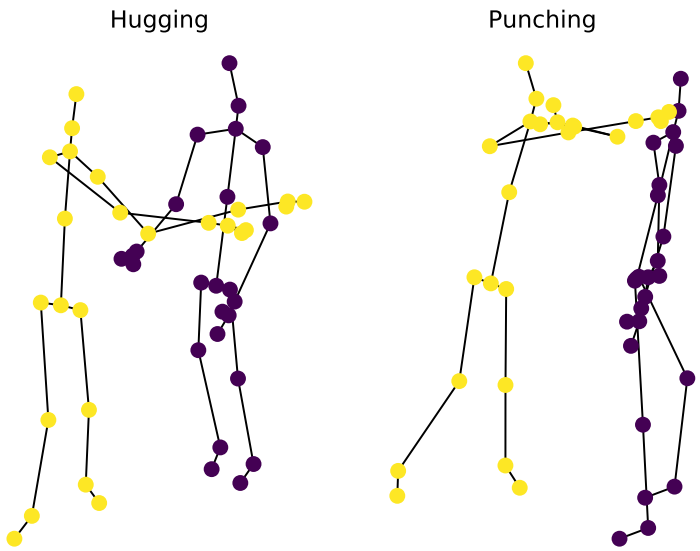


Figure 5.7. Examples of graphs from NTU RGB+D, from the *hugging* and *punching* classes. Different colours indicate different subjects.

depth maps, and skeletal data of 56880 action samples [190]. The dataset contains 60 different action types, including daily, mutual, and health-related actions. Actions are performed by 40 volunteers and each action is repeated twice in 17 different camera settings. The dataset consists of short clips of about 2 seconds sampled at 30Hz. Skeletal data are provided for each frame. Each subject is represented by 25 joints annotated with 3D coordinates and orientation (*i.e.*, position, direction and rotation of the joint in space), and 2D position w.r.t. the frame for both the RGB and infrared channels. Metadata regarding the confidence of the measurement (*i.e.*, whether the annotation is missing, inferred, or actually recorded) is also provided. The topological connections between pairs of joints are fixed and known *a priori*.

We consider a subset of NTU RGB+D containing mutual interactions between two subjects, namely the *hugging* and *punching* actions (see Figure 5.7), where each sample is the disjoint union of the two skeletal graphs. The task is to detect when the interaction between the two subjects changes from friendly (*hugging*) to hostile (*punching*).

The graph stream is generated, like in the previous experiment, by sampling graphs from the *hugging* distribution for the nominal regime, and then switching to the *punching* distribution to simulate a change. The assumption of station-

Table 5.8. Number of graphs sampled for the training and operational phases. Training graphs correspond to the first set of clips, while operational graphs are taken from the second set.

Phase	**Action**	**Graphs**
Train	Hugging	26818
Operational	Hugging	26166
	Punching	24512

Table 5.9. Performance of R-CDT and D-CDT on detection of hostile behaviour.

CCM	**Emb.**	**Emb.**	**AUC**	**ARL**
$\mathcal{M}_*$	Geom.	R-CDT	**0.999**	**1.04**
		D-CDT	0.965	3.27

arity of Q_0 here does not hold due to the high correlation between consecutive samples. To avoid this issue, we can decorrelate the observations by lowering the sampling rate of the clips. However, we note that lowering the sampling rate to obtain an i.i.d. graph stream is equivalent, in this controlled setting, to taking random permutations of the available data. This results in a stationary stream and allows us to test our method without wasting precious data. Therefore, we obtain the training data by randomly sampling graphs from the *hugging* distribution. Similarly, for the operational test stream, we randomly sample graphs from *hugging* first and then change to *punching*. While in principle it is not necessary to randomise the non-nominal regime, we keep the same setting to ensure that the CDT detects changes in the actual class rather than in the sampling technique. To make the training and operational streams independent, we sample them from different sets of clips (since every clip is recorded twice for each pair of subjects). We report in Table 5.8 the number of graphs for each regime in the training and operational streams, *i.e.*, the number of clips from the respective sets of repetitions.

Results We build on the results of the previous experiment for configuring the change detection pipeline, and we only report results with the ensemble of manifolds $\mathcal{M}_*$, the geometric discriminator, and R-CDT.

Our method achieved an AUC score of 0.999, *i.e.*, the algorithm is consistently able to identify changes in stationarity with a very short delay. The average run

length (ARL) in the non-nominal regime is 1.04, meaning that the algorithm raises an alarm almost at every window. By comparison, D-CDT shows a similarly good separability in the distributions of the run lengths, with AUC 0.965 (see Table 5.9). However, the ARL of D-CDT is significantly higher at 3.27, indicating a slower detection.

In Figure 5.8, we show the evolution of the accumulator S_w on the operational stream for each of the three CDTs run by R-CDT (one for each manifold). We see that the spherical component of the embeddings computed by the AAE clearly indicates a change in stationarity. A similar conclusion is also evident when comparing the distribution of the spherical embeddings to the other two geometries in Figure 5.9.

These results highlight the advantages of using an ensemble of CCMs as latent space and confirm the importance of representing graph-structured data in non-Euclidean domains.

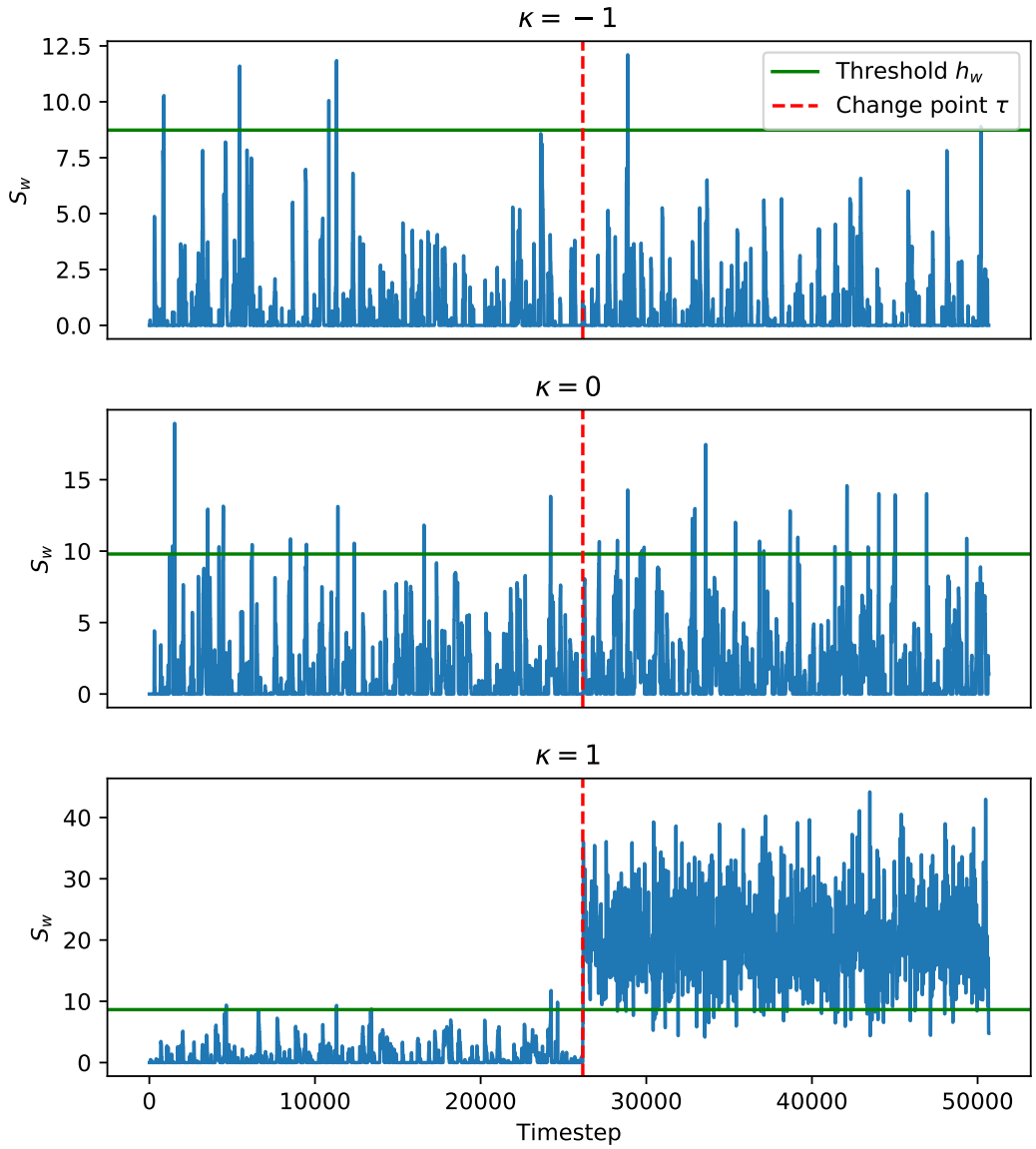


Figure 5.8. Accumulator S_w on the operational stream, for the three independent CDTs run by R-CDT on the ensemble of manifolds. The dashed red line indicates the change point. Whenever the accumulator exceeds the threshold (green line) an alarm is raised. The CDT on the spherical manifold ($\kappa = 1$) identifies the change in stationarity, while the Euclidean and hyperbolic embeddings do not show such a strong response to the change.

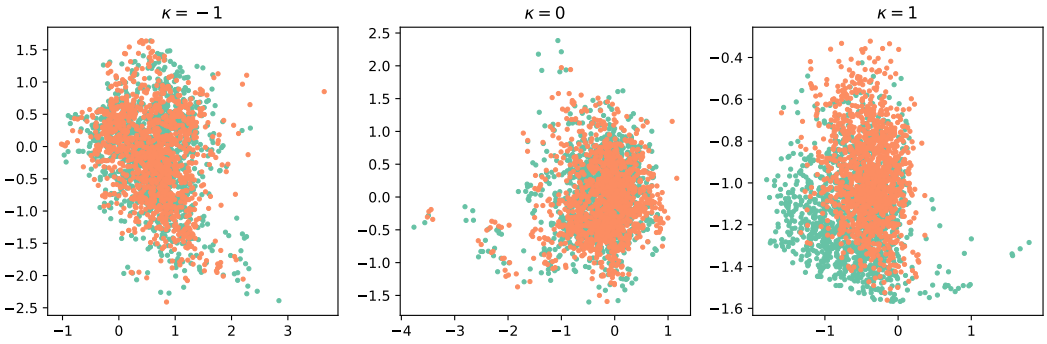


Figure 5.9. Embeddings learned by the AAE with latent ensemble of CCMs, $\mathbf{M}_*$. We show a planar projection for the hyperbolic and Euclidean CCMs, and the Hammer projection of the spherical CCM (*hugging* is in green, *punching* in orange).

Chapter 6

Explainable GNNs: a case study on seizure localisation

Epilepsy is a neurological disorder characterised by recurrent episodes of excessive neuronal firing [199]. In approximately a third of the patients, epilepsy cannot be treated with anti-seizure drugs and resective surgery can be considered as a possible treatment [112]. The outcome of surgery is crucially dependent on the successful localisation of the seizure onset zone (SOZ) [30, 209].

Electroencephalography (EEG) is the mainstay for studying and diagnosing epilepsy, and it is widely used to detect, classify, and localise seizures by recording and processing the electrical activity of groups of neurons [157]. However, due to their low spatial resolution, scalp EEG recordings in some cases are not informative enough to successfully localise seizures [188]. In these cases, intracranial EEG recordings (iEEG), in which electrodes are placed directly on or within the brain, provide better spatio-temporal resolution to capture the dynamics of seizure generation and propagation [83]. However, the high temporal resolution of iEEG and the complex functional interaction of distant brain areas, especially during seizures, make the interpretation and processing of raw iEEG data a non-trivial task for clinicians. For this reason, a significant branch of epilepsy research is concerned with summarising iEEG data by considering the pairwise (statistical) dependencies between the activity of different brain areas over time [209]. These dependencies are usually represented by *functional networks* (FNs), in which each node represents a sensor and edges are weighted by a *functional connectivity* (FC) metric [12].

In this chapter, we introduce a GNN-based methodology to automate the localisation of seizures, using FNs to efficiently represent brain states [77]. The core of our algorithm is a GNN equipped with an *attention-based readout*. By

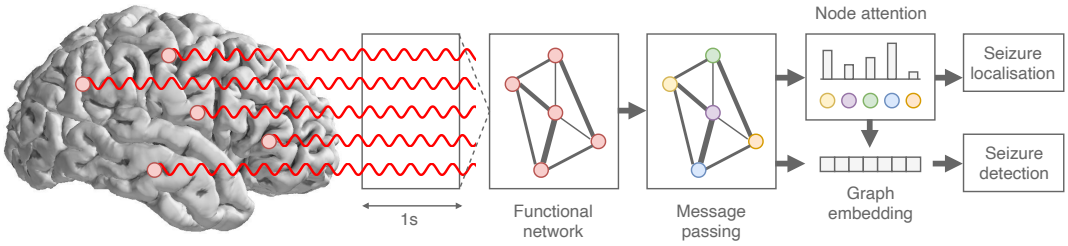


Figure 6.1. Schematic view of our GNN-based pipeline for seizure detection and localisation. Starting from raw iEEG data, we compute a functional network to represent the spatio-temporal dynamics of the signals compactly. The FN is then given as input to a GNN with an attention-based readout to compute a graph-level embedding. The embedding is then classified to perform seizure detection, while the attention scores are analysed to perform seizure localisation.

training such a GNN to perform seizure detection, the readout automatically learns to assign higher attention to those nodes that are more important for a correct classification. Then, we propose a simple and fast way of analysing the attention coefficients over time, so that we obtain a ranking of the nodes based on their overall importance in detecting a seizure. Crucially, our methodology does not require *a priori* information regarding the SOZ, but only weak supervision in the form of annotated seizure onsets and offsets. A schematic representation of our approach is shown in Figure 6.1.

Through a collaboration with the Krembil Research Institute and the Toronto Western Hospital, we were able to test our methodology using iEEG data collected from eight human subjects with refractory epilepsy, for which we also had ground-truth knowledge regarding the SOZ provided by professional electroencephalographers. The results obtained on real-world data (and verified on synthetic data from brain simulators) are extremely encouraging. First, we show that the attention coefficients learned by the GNN correlate with the clinically-identified SOZs and accurately predict the presence of ictal activity. Second, we show that, when electroencephalographers could not identify the SOZ from the iEEG data, the GNN also shows uncertainty in the localisation. This is a very desirable behaviour since a strong attention score in such cases would have raised concerns about the reliability of our method. Finally, we show that our methodology performs well on very imbalanced datasets, achieving a good localisation accuracy even on patients for which we observe as few as five seizures during training.

In the following sections, we first give the relevant background on seizure

localisation with functional networks and then we describe the proposed method and report our experimental results. More details and experiments are reported in Appendix E.

6.1 Background

FNs are a widespread tool to study seizure localisation, with early approaches dating back to the 1970s [64, 26]. Seizures have been observed to affect the functional organisation of brain activity at the mesoscale, both from a node-centric [30] and an edge-centric [99] perspective. In particular, Burns et al. [30] identified sets of brain states that emerge by clustering FNs, consistent in interictal and ictal periods for individual patients. They observed that changes in node centrality in FNs accurately predict the SOZ. Khambhati et al. [99] observed a strengthening of FC in the SOZ during seizures, also coinciding with a topological tightening of the connections (*i.e.*, strong connections also become physically closer). Khambhati et al. [100] proposed *virtual cortical resection*, *i.e.*, the removal of nodes from FNs, in order to study changes in network synchronizability, which is a known predictor for the spread of seizures [183]. Lopes et al. [126] also observed that the resection of brain areas associated with *rich-club* hubs in FNs correlates with a good postoperative outcome. Seizure localisation has also been studied in FNs obtained from functional magnetic resonance imaging (fMRI) [116, 219] and scalp EEG [200] data. Recent work by Covert et al. [40] introduced the use of spatio-temporal graph convolutional networks (ST-GCNs) [240] to perform seizure detection, and conducted an *ex-post* analysis similar to the one of Khambhati et al. [100] to quantify the importance of a node by observing the effect of its removal on the downstream detection accuracy. Gadgil et al. [61] also proposed a methodology based on ST-GCNs that allows identifying high-interest areas in fMRI by learning to estimate edge importance, although they did not apply it to seizure localisation. For a more in-depth review of approaches to seizure localisation with FNs, we refer the reader to Van Mierlo et al. [209].

6.2 Method

6.2.1 Functional networks

Choosing a suitable FC metric to model the pairwise interaction between brain areas is a non-trivial challenge, as there exist a large variety of methods with

their advantages and disadvantages. FC metrics can be characterised according to several properties, including whether they are in the time or frequency domain, whether they are directed or undirected (*i.e.*, if they model asymmetric or symmetric couplings), or whether they are model-free or model-based [12]. Here, we focus on undirected FC metrics to simplify the GNN computation, and on model-based approaches to reduce the computational costs of estimating the FC metrics directly from data. We do, however, consider two different metrics to highlight the practical differences that emerge between time- and frequency-domain metrics.

FNs are generated by computing a FC value for each pair of iEEG channels $x_a(t)$ and $x_b(t)$ over a time window of length T. For the time-domain metric, we consider Pearson's correlation coefficient:

$$\mathbf{e}_{a\to b} = \mathbf{e}_{b\to a} = \frac{\sum_{t=1}^{T}(x_a(t)-\bar{x}_a)(x_b(t)-\bar{x}_b)}{\sqrt{\sum_{t=1}^{T}(x_a(t)-\bar{x}_a)^2}\sqrt{\sum_{t=1}^{T}(x_b(t)-\bar{x}_b)^2}}, \tag{6.1}$$

where $\bar{x}_a = \frac{1}{T}\sum_{t=1}^{T} x_a(t)$ and analogously for $\bar{x}_b$. Correlation allows to quantify symmetric linear interactions, it is easy to compute and, as such, it is often used in the literature. For the frequency domain, we consider the phase-locking value (PLV) [113]:

$$\mathbf{e}_{a\to b} = \mathbf{e}_{b\to a} = \left|\frac{1}{T}\sum_{t=1}^{T} e^{i(\varphi_a(t)-\varphi_b(t))}\right|, \tag{6.2}$$

where $\varphi_a(t)$ indicates the instantaneous phase of signal $x_a(t)$ obtained via Hilbert transform (and similarly for $\varphi_b(t)$). A significant advantage of PLV over correlation is that it is less sensitive to artefacts in the iEEG signals (such as those caused by the patient's movements). After computing the FC metrics for each pair of channels, we sparsify the resulting FNs by removing those edges for which $|\mathbf{e}_{i\to j}| < 0.1$, *i.e.*, those indicating weak coupling. The choice of sparsification threshold is generally an important hyperparameter when studying FNs. For example, a principled way of computing a dynamic sparsification for each individual FN is described in the work of Kramer et al. [110]. However, in this case we are not interested in fine-tuning the threshold nor do we wish to devise a dynamic sparsification scheme to process each FN independently. Our only goal is to remove those connections that do not have enough statistical significance, and a threshold of 0.1 achieves the desired result. As long as the same threshold is consistently used for different FNs, then the GNN will learn to deal with the

resulting distribution of FNs. We report an additional discussion regarding the threshold in Appendix E.5.

We generate a dataset of FNs for each patient, dividing the FNs into ictal and interictal classes and proceeding in a per-seizure fashion. Let f_s be the sampling rate of the iEEG signal, L the duration of a seizure, t_0 the time indicating the seizure onset, $k \geq 1$ a subsampling factor, and T the length of the time windows. Additionally, let $y(t) \in \{0, 1\}$ be a binary signal indicating whether the patient is having a seizure at time t (*i.e.*, $y(t) = 1$ if $t \geq t_0$ and 0 otherwise). Note that we consider each seizure to end at time $t_0 + L$ and we do not compute FNs for the data immediately following a seizure offset.

Given a time window $[t - T, ..., t]$, we compute a FN $\mathcal{G}^{(t)}$ and label it with class

$$\mathcal{Y}^{(t)} = \begin{cases} 1, & \text{if } \sum_{\tau=t-T}^{\tau} y(\tau) > T/2 \\ 0, & \text{otherwise.} \end{cases} \tag{6.3}$$

To generate the FNs associated with seizures (class 1), we consider the data interval $[t_0 - T/2, ..., t_0 + L]$ and take overlapping windows of size T with a stride of $1/f_s$. For the interictal FNs (class 0), instead, we consider a longer period preceding the seizure onset, $[t_0 - kL, ..., t_0 + T/2]$, and we take windows at a larger stride of k/f_s. Here, we consider $k = 10$ and $T = 1$s for all experiments, although other values are possible.

This procedure to generate the FNs (summarised in Figure 6.2) results in a balanced dataset and has two advantages. First, it allows us to fully use all the available (and rare) ictal events. Second, it allows us to consider a more diverse sample for the interictal class. The small differences between consecutive FNs of the positive class, due to the small stride at which windows are taken, can be seen as a form of sample weighting to account for the class unbalance characterising the problem.

In order to have initial node features that can be processed by the GNN, we consider dummy attributes set to 1 for all nodes. Other choices that depend on the actual iEEG signals are possible (*e.g.*, the signal power or wavelet coefficients) but were not explored here.

6.2.2 Attention mechanism

Attention [8, 210] is a processing technique for neural networks to learn how to selectively focus on parts of the input. Originally developed for aligning sentences in neural machine translation [8, 210], the attention mechanism has been

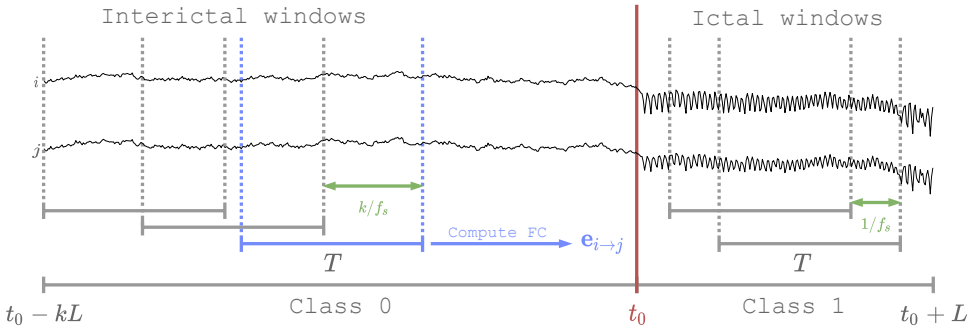


Figure 6.2. Schematic representation of the procedure used to generate FNs. For each seizure of length L starting at t_0 (marked in red), we consider an interictal interval of length kL. Interictal FNs are generated taking windows of length T at stride k/f_s, while ictal windows are taken with stride $1/f_s$ (in green). For each window and each pair of electrodes i and j, we compute the FC value $\mathbf{e}_{i\to j}$ (in blue) to obtain the full FN. This figure is only meant to represent the procedure and is not shown in any physical temporal scale.

used to achieve state-of-the-art results on different tasks like language modelling [28], image processing [233], and even learning on graphs [211].

Here, we focus on the concept of *self*-attention, which indicates a class of attention mechanisms that learn to attend to the output of a layer using the output itself (in contrast to classical attention, which uses the output of one layer to focus on the output of another—*e.g.*, the sentence of the source language is used to focus on the target language). At its core, self-attention consists of computing a compatibility score $\alpha_{ij} \in [0,1]$ between two vectors $\mathbf{h}_i, \mathbf{h}_j \in \mathbb{R}_n^D$ (both part of the same sequence, image, graph, etc.):

$$\alpha_{ij} = \text{Softmax}_j(b_{ij}) = \frac{\exp\left(b_{ij}\right)}{\sum_{k=1}^{N} \exp\left(b_{ik}\right)}, \tag{6.4}$$

where

$$b_{ij} = f_a(\mathbf{h}_i, \mathbf{h}_j) \tag{6.5}$$

and f_a is called an *alignment* model, which is usually learned end-to-end along with the other parameters of the neural network. The compatibility score is then used to compute a representation of element i as:

$$\mathbf{z}_i = \sum_j \alpha_{ij} \mathbf{h}_j. \tag{6.6}$$

Intuitively, the attention mechanism learns the importance of element j to describe element i, and computes score α_{ij} to quantify this importance. The alignment model can be seen as a similarity function between the two elements, which is then normalised via the softmax function. Different implementations of the alignment model are possible, although often it is implemented as a multi-layer perceptron.

Attention mechanisms are usually trained without direct supervision and automatically learn to focus on different parts of the data according to the loss of the given task. By optimising the overall task loss, the attention layers in a neural network learn to compute the optimal compatibility scores. This is a key aspect of our proposed methodology, where we use self-attention to automatically detect those brain areas (monitored via different iEEG channels) that are important to detect a seizure. Crucially, using attention allows us to perform localisation without providing our neural network with ground truth information on the SOZ.

6.2.3 Graph neural networks for seizure localisation

Our method for seizure localisation can be summarised as follows. First, we train a GNN with an attention-based readout to detect seizures from FNs. This is a graph-level classification problem where a label (ictal or interictal) is assigned to each FN. Then, we analyse the compatibility scores learned by the attentional mechanism to identify those nodes that the model consistently considers important. Although we train the GNN to do seizure detection in a supervised way, *i.e.*, it requires manually-annotated seizure onsets and offsets, the localisation is fully unsupervised. This is one of the main strengths of the proposed method, as significantly less manual work is required to annotate the temporal boundary for each seizure, rather than the SOZ.

There are two main components in our GNN architecture. First, the connectivity information is propagated to the node attributes via an edge-aware message-passing operation like ECC (*cf.* Section 2.3, Equation (2.36)). A single layer is sufficient because the input FNs are densely connected, and most nodes will receive information from the whole graph in a single step of message passing.

Then, we use a self-attentional mechanism to compute the graph readout:

$$\mathbf{z} = \textsc{Attn-RO}(\mathbf{h}) = \sum_{j=1}^{N} \alpha_j \mathbf{h}_j \tag{6.7}$$

where

$$\alpha_j = \frac{\exp\left(\mathbf{h}_j \cdot \mathbf{a}\right)}{\sum_{k=1}^{N} \exp\left(\mathbf{h}_k \cdot \mathbf{a}\right)}, \tag{6.8}$$

$\mathbf{h}_j \in \mathbf{R}^{F^{out}}$ is the embedding of the j-th node computed by the ECC layer, and $\mathbf{a} \in \mathbf{R}^{F^{out}}$ is a vector of learnable weights. Note that, compared to Equation (6.6), here index i is left implicit as the attention is only computed once for all nodes, to reduce the graph to a vector. This is also reflected in the fact that the alignment model is a function of only one node at a time, *e.g.*, $\mathbf{h}_j \cdot \mathbf{a}$.

Finally, a multi-layer perceptron with sigmoid activation computes the probability that the input FN represents an ictal window of iEEG data.

The full architecture is written as:

$$\hat{y} = \text{MLP}(\text{ATTN-RO}(\text{ECC}(\mathcal{G}))) \tag{6.9}$$

where $\mathcal{G}$ represents an input FN (*cf.* Figure 6.1).

By training the GNN to correctly distinguish the ictal FNs from the non-ictal ones, we also implicitly train the attentional readout ATTN-RO to assign higher attention to those nodes of the FNs that maximise the confidence in the prediction. We then analyse how the attention scores assigned to nodes change over time, and rank the nodes according to the overall amount of attention they receive before and during a seizure. The localisation procedure is described in the following section.

6.2.4 Localising the seizure onset zone

For each seizure in the data, we consider symmetric intervals of length $2L$ centred at the seizure onset, so that the first L timesteps are pre-ictal and the remaining L cover the beginning of the seizure. For each of the $2L$ timesteps, we compute a FN $\mathcal{G}^{(t)}$ from a $T = 1$s window ending at time t, obtaining a sequence of FNs $[\mathcal{G}^{(1)}, \ldots, \mathcal{G}^{(2L)}]$ (this is equivalent to how we generate the training datasets, except that the subsampling is set at $k = 1$). For each FN in the sequence, we use the GNN to compute the attention scores over the nodes according to Equation (6.8). We thus compute a sequence of attention scores $[\alpha_i^{(1)}, \ldots, \alpha_i^{(2L)}]$ for each node i.

We then sum the sequence of attention scores to obtain the overall *importance* of the node over the considered time interval:

$$\sigma_i = \sum_{t=1}^{2L} \alpha_i^{(t)}, \tag{6.10}$$

and normalise the importance scores to the $[0, 1]$ interval as:

$$s_i^{(s)} = \frac{\sigma_i^{(s)} - \min_{j \in \mathcal{V}} \sigma_j^{(s)}}{\max_{j \in \mathcal{V}} \sigma_j^{(s)} - \min_{j \in \mathcal{V}} \sigma_j^{(s)}}. \tag{6.11}$$

Finally, we rank the nodes according to their importance and predict the SOZ accordingly.

6.3 Results

We report the results obtained on real iEEG data collected from eight patients. Additional results on two brain activity simulators (a simple network model [16] and *The Virtual Brain* simulator [179]) and all experimental details regarding the GNN are reported in Appendix E.

6.3.1 Data collection and pre-processing

We used iEEG data recorded from eight human subjects with medically refractory epilepsy, the recordings obtained as part of their standard clinical pre-surgical investigations. The patients were selected among a larger pool of patients based on certain criteria, chiefly having at least five clinical seizures recorded in our database and having a recorded clinical history of at least two years.

The study was approved by the Research Ethics Board at the University Health Network (ID number 12-0413) and written consent for data collection was obtained from all participants. Each patient had a varying number of recorded clinical seizures and the number of electrodes also varied from patient to patient (*cf.* Table 6.1). The data was recorded from subdural or intracerebral depth electrodes at $f_s = 500$Hz over the course of several days per patient, and seizures were manually annotated by electroencephalographers, inspecting both raw iEEG and video recordings of the patient. The iEEG signal was notch-filtered at 60Hz and related harmonics to remove powerline trends, and then filtered with an order-3 low-pass filter at 100Hz to remove any high-frequency noise. Then, each electrode channel was independently re-referenced to have zero mean and rescaled to have unit variance.

Before pre-processing, we visually inspected the raw data of each patient and each seizure to assess the presence of bad channels: we considered symmetric windows around each labelled seizure onset and we removed from the data any channels that exhibited abnormal (*i.e.*, either flat or excessive) activity in at least one seizure.

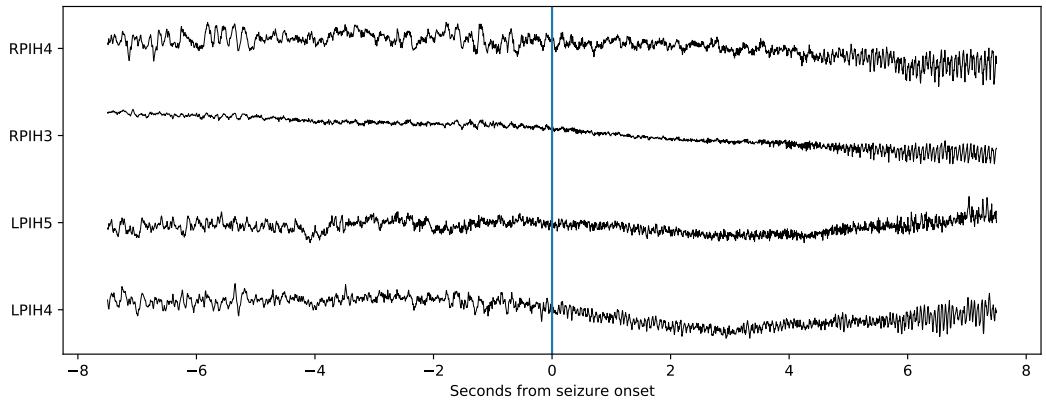
Table 6.1. Summary of the patients considered for this study. The columns indicate (left-to-right): the number of recorded seizures, the number of implanted electrodes, the presence of ictal activity (IA) marked by electroencephalographers on one or more channels, whether the patient had surgery, and the outcome of the surgery.

Patient	Seizures	Electrodes	IA identified	Surgery	Outcome
1	15	100	Yes, low confidence	No	-
2	9	96	Yes	Yes	Seizures reduced
3	10	23	Yes	No	-
4	5	74	No	No	-
5	11	38	Yes	Yes	Seizures reduced
6	18	45	Yes, poorly defined	No	-
7	5	45	Yes	No	-
8	16	69	Yes	No	-

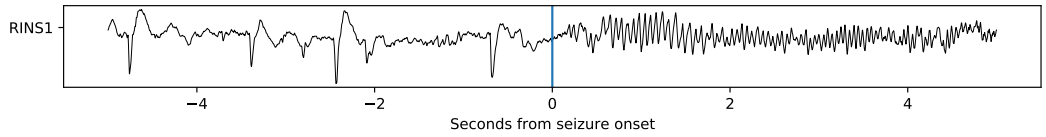
6.3.2 Per-patient analysis of the SOZ

This section reports the available clinical data for the patients considered in our study. For all patients, both the seizure onset time instants and the SOZ annotations were provided by electroencephalographers.

Patient 1 demonstrated ictal activity in both the left and right posterior interhemispheric regions (Figure 6.3a), with interictal epileptiform discharges recorded independently from the left anterior frontal and right middle frontal lobes. The patient did not undergo resective surgery due to low confidence in the identification of the SOZ. Patient 2 showed clear seizures originating in the right posterior insular region (Figure 6.3b). The patient underwent laser interstitial thermal therapy targeting a focal cortical dysplasia in the area. The patient continued to have some postoperative seizures, although these were reduced in frequency and intensity, indicating that the SOZ was identified correctly. Patient 3 had seizure onsets recorded independently from both temporal lobes and thus was not a candidate for surgery. Patient 4 had no clear ictal activity identified by electroencephalographers in the iEEG recordings and was thus not a candidate for surgery, the SOZ evidently not captured by the intracranial electrode placements. Patient 5 demonstrated ictal activity in the left hippocampal body and underwent a left anterior temporal resection. The patient continued to have seizures after the surgery, but of reduced frequency and intensity, indicating a successful localisation of the SOZ. Patient 6 had multiple seizures recorded with poorly defined, inconsistent ictal onsets over the temporoparietal sensory cortex



(a) Patient 1



(b) Patient 2

Figure 6.3. Examples of raw iEEG traces for patients 1 and 2. The two plots show the activity of electrodes that were identified as SOZs by electroencephalographers. The vertical line marks the seizure onset, as reported in the patients' clinical records.

and was deemed not a candidate for surgical resection due to uncertainty on the SOZ. Patient 7 had seizures recorded in the left hemisphere, with onsets involving a broad region of the temporal lobe neocortex. The patient was not subject to resection due to the epileptogenic zone being too large, and near eloquent language cortex. Patient 8 exhibited abnormal activity in the left amygdala and hippocampus. The patient had already undergone contralateral right anterior temporal resective surgery years prior to the collection of the iEEG data and was not a candidate for further resections.

Table 6.1 summarises the relevant details of the eight patients. In particular, six patients had clinically identified, well-defined information regarding the SOZ, whereas in two patients the SOZ could not be clearly identified in the iEEG data by electroencephalographers. Despite not having ground truth information related to the SOZ for these two patients, we still included them as part of our study to analyse the behaviour of our algorithm in such cases of high uncertainty. The question that we aim to answer with this analysis is: what does the GNN see when professional electroencephalographers are uncertain about the SOZ? A strong attention score in such cases would raise concerns about the soundness of our method. Instead, we observe in the following section that the GNN shows uncertainty in those cases where professionals are also uncertain.

6.3.3 Results on seizure detection and localisation

Table 6.2 reports the Area Under the Receiver Operating Characteristic Curve (ROC-AUC) and the Area Under the Precision-Recall Curve (PR-AUC) obtained by the GNN on the seizure detection task. We report the results obtained using both FC metrics (correlation and PLV) to generate the FNs. We also report the detection performance of a baseline convolutional neural network for time series classification (details in Appendix E.4). We repeat each experiment five times and, where appropriate, report the average and standard deviation of the results.

The GNN achieved an average ROC-AUC score of 79.56 and an average PR-AUC of 81.24 (the average is computed over all patients) when using correlation as FC metric. These results are aligned with the performance of the baseline, which our method slightly outperformed on average, and indicate that 1) our choice of architecture was reasonable and 2) using graph-structured data is an interesting direction for future research on efficient seizure detection. We also recall that the detection task is only meant to provide a weak supervision for the more interesting challenge of localisation, and that better detection results could be achieved by increasing the capacity of the GNN or collecting more training data.

Table 6.2. Average ROC-AUC score and average PR-AUC score for seizure detection on unseen test data. These scores represent the model's ability to correctly classify the FNs as interictal or ictal. The last row reports the average score over all patients. The highest ROC-AUC and PR-AUC scores are reported in bold for each patient. We report the average and standard deviation over all test seizures and all repetitions.

	Baseline		GNN Corr.		GNN PLV	
Patient	**ROC**	**PR**	**ROC**	**PR**	**ROC**	**PR**
1	62.54 ± 22.5	70.06 ± 17.8	68.63 ± 11.43	75.20 ± 10.30	**75.68 ± 23.3**	**77.51 ± 20.1**
2	80.19 ± 15.5	85.96 ± 10.6	**86.87 ± 9.07**	**89.04 ± 9.35**	65.36 ± 20.1	72.91 ± 14.8
3	82.32 ± 14.19	87.25 ± 9.24	**93.35 ± 3.12**	**94.34 ± 2.72**	71.50 ± 14.8	71.02 ± 16.3
4	67.81 ± 8.75	69.83 ± 13.12	**60.40 ± 14.41**	**61.11 ± 14.82**	53.83 ± 6.6	51.67 ± 6.4
5	76.18 ± 15.41	80.42 ± 14.26	**77.04 ± 11.98**	**76.39 ± 13.03**	71.46 ± 12.1	71.45 ± 12.9
6	**76.32 ± 17.2**	**80.94 ± 13.5**	73.72 ± 17.14	76.02 ± 14.53	63.81 ± 17.2	71.06 ± 12.4
7	76.46 ± 11.24	81.22 ± 7.65	**85.52 ± 10.95**	**85.92 ± 13.65**	69.32 ± 2.6	65.55 ± 1.8
8	85.60 ± 14.6	89.29 ± 10.7	**90.97 ± 5.51**	**91.89 ± 3.49**	77.69 ± 11.5	78.32 ± 11.3
Avg.	75.93 ± 7.06	80.62 ± 6.86	**79.56 ± 10.82**	**81.24 ± 10.37**	68.58 ± 7.08	69.94 ± 7.86

Tables 6.3 and 6.4 report the performance of the model on the patients with a known SOZ, respectively using correlation and PLV to generate FNs. In particular, we report three main performance measures:

(a) the average precision at K (AP@K) [178] obtained by the GNN when computing an average ranking of the electrodes. Each electrode is re-ranked by considering five models trained on the same data and taking the average score assigned to each electrode over all models and all seizures. This measure quantifies the GNN's ability to correctly identify the SOZ for a patient in general, which is the most clinically relevant scenario.

(b) The mean AP@K (MAP@K) obtained by the GNN on different individual seizures. In this case, the ranking for each seizure is compared to the ground truth independently of the others (*i.e.*, without averaging the scores), and the scores are averaged *a posteriori* (also considering five repetitions of the experiments). This measure quantifies the GNN's ability to correctly identify target electrodes in a given seizure.

(c) The MAP@K obtained by the GNN on different individual seizures, but considering groups of electrodes belonging to the same strip (implying spatial

Table 6.3. Localisation performance for patients with a known SOZ, when using Pearson's correlation as FC metric. We report: **(a)** the average precision at K for averaged rankings, which evaluates the localisation for the patient overall; **(b)** the mean average precision at K for single rankings, which evaluates the localisation for a given seizure; **(c)** the mean average precision at K for single rankings and groups of electrodes, which is equivalent to (b) but at a coarser scale. We report scores for $K = 2, 5, 10$. Bold indicates that the results are better than the ones obtained with PLV as FC metric (*cf.* Table 6.4).

	(a) AP@K - Avg. rank			**(b) MAP@K - Single**			**(c) MAP@K - Groups**		
Patient	$K=2$	$K=5$	$K=10$	$K=2$	$K=5$	$K=10$	$K=2$	$K=5$	$K=10$
1	**50.00**	**20.00**	**12.50**	**22.31**	**12.0**	**7.24**	**26.92**	**21.48**	31.64
2	**100.00**	**100.00**	**100.00**	**51.11**	**54.8**	**56.71**	**53.33**	**58.48**	**60.73**
3	0.00	16.67	38.96	20.37	26.51	28.98	36.11	45.09	50.07
5	**100.00**	**55.00**	**55.00**	**97.73**	**48.55**	**54.71**	**99.09**	**99.09**	**99.09**
7	25.00	20.00	10.00	22.00	20.56	16.76	78.00	72.03	82.70
8	0.00	6.67	5.56	**19.69**	**13.00**	**7.42**	**20.00**	**36.43**	**44.07**

locality of the electrodes). This allows us to evaluate the performance of the model at a coarser scale.

From the results we see that, while correlation was a clearly better metric for seizure detection, the localisation performance can vary depending on the particular FC metric used. In particular, the localisation for patients 1 and 5 was better when using correlation networks, but PLV yielded better results for patients 3, 7, and 8.

In general, however, we note that the (M)AP@5 score is positive for both FC metrics, for all performance measures and all patients, meaning that at least one SOZ-associated electrode was ranked in the top five every time. We also note that the GNN achieves a perfect AP@2 score (average rankings) in six out of eight cases when using PLV, indicating a high chance of localising at least two relevant electrodes per patient.

Remarkably, we see that these results were obtained even when considering small datasets, *e.g.*, down to only five seizures for patient 7 (*cf.* Table 6.1). While this result is encouraging and highlights the sample efficiency of our approach, we stress that a higher amount of training data can only improve the detection and, likely, localisation performance of our method, as well as giving a higher statistical certainty about the results.

Table 6.4. Localisation performance for patients with a known SOZ, when using PLV as FC metric. Bold indicates that the results are better than the ones obtained with correlation as FC metric (*cf.* Table 6.3).

	(a) AP@K - Avg. rank			**(b) MAP@K - Single**			**(c) MAP@K - Groups**		
Patient	$K=2$	$K=5$	$K=10$	$K=2$	$K=5$	$K=10$	$K=2$	$K=5$	$K=10$
1	0.00	5.00	5.83	8.67	5.97	4.67	16.67	18.82	**31.78**
2	**100.00**	**100.00**	**100.00**	50.00	54.33	56.65	50.00	58.04	60.21
3	**100.00**	**55.00**	**45.46**	**60.00**	**40.82**	**32.58**	**66.88**	**45.16**	**51.36**
5	**100.00**	40.00	48.57	66.82	38.28	45.27	91.82	93.48	93.48
7	**100.00**	**40.00**	**35.71**	**70.00**	**43.84**	**30.45**	**82.00**	**74.22**	**84.22**
8	**50.00**	**20.00**	**10.00**	15.62	9.15	6.43	16.56	20.07	32.30

6.3.4 Comparison with clinical information

Figure 6.5 shows a graphical visualisation of the scores and rankings used to compute the values in Tables 6.3 and 6.4. The figure summarises our results and provides an overview of the importance scores, their variability across different models and seizures, and their agreement with the ground truth. For every electrode, we report the average score and its standard deviation over all test seizures and all repetitions.

The results for patient 5 can be considered a complete success, with the highest AP@K scores among all patients and very little uncertainty in the ranking by the GNN. Crucially, the successful postoperative outcome confirms that the localisation of the SOZ for this patient was accurate and points to a strong localisation ability of the GNN. For patient 2, ictal activity was evident and well-localised on a specific depth electrode placed in the right insular complex (RINS1). The clinical localisation of the SOZ was therefore likely accurate, even if the outcome of the surgery was not completely successful. More importantly, we note that the GNN was strongly aligned with the human analysis given the same information, and similarly focused on the same electrode (which is ranked first using either of the FC metrics). Our methodology also confirms the conclusions reached by electroencephalographers for patients 3, 7 and 8, although further studies would be required to give a more precise interpretation of the results (including, possibly, the outcome of future surgeries). The results for patient 8 are particularly uncertain, despite the GNN achieving a good detection accuracy (*cf.* Table 6.2). In general, however, the rankings provided by the GNN show a high agreement with the medical assessment in those cases where the SOZ was successfully identified.

For patients with no known SOZ (4, 6), the GNN has a low detection per-

formance and the average attention scores assigned by the GNN are uniformly distributed across all electrodes around an average score of 0.5. On the contrary, patients with a known SOZ have a few electrodes that are assigned a majority of the attentional budget. This difference between the two cases is more clearly visualised in Figure 6.4, which shows the distribution of the scores given to different electrodes at the seizure onset (patient 5 is taken as representative of the case in which the SOZ is known).

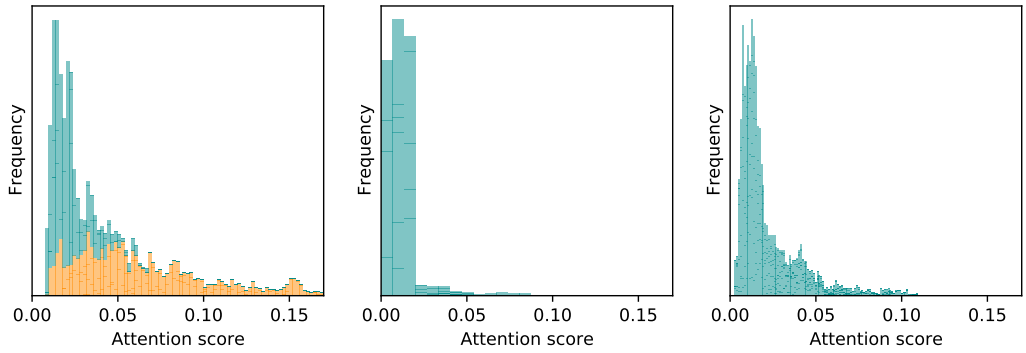
For patient 1, the GNN did not identify any particularly important regions despite there being some clinical evidence of ictal activity in the posterior interhemispheric region. Two posterior interhemispheric electrodes are indeed ranked in the top ten (averaged rankings) by the GNN when using correlation FNs, although with very high uncertainty. We note, however, that the uncertainty showed by the GNN was also reflected clinically in the electroencephalographers' interpretations and in the final decision to not operate on this patient.

Our analysis for patients 1, 4, and 6 shows that the uncertainty of the GNN correlates with uncertainty or inability on the part of electroencephalographers to identify the SOZ in iEEG, and can still be useful to support their decision making (*e.g.*, deciding to not operate a patient can be just as valuable as a successful localisation).

6.4 Discussion

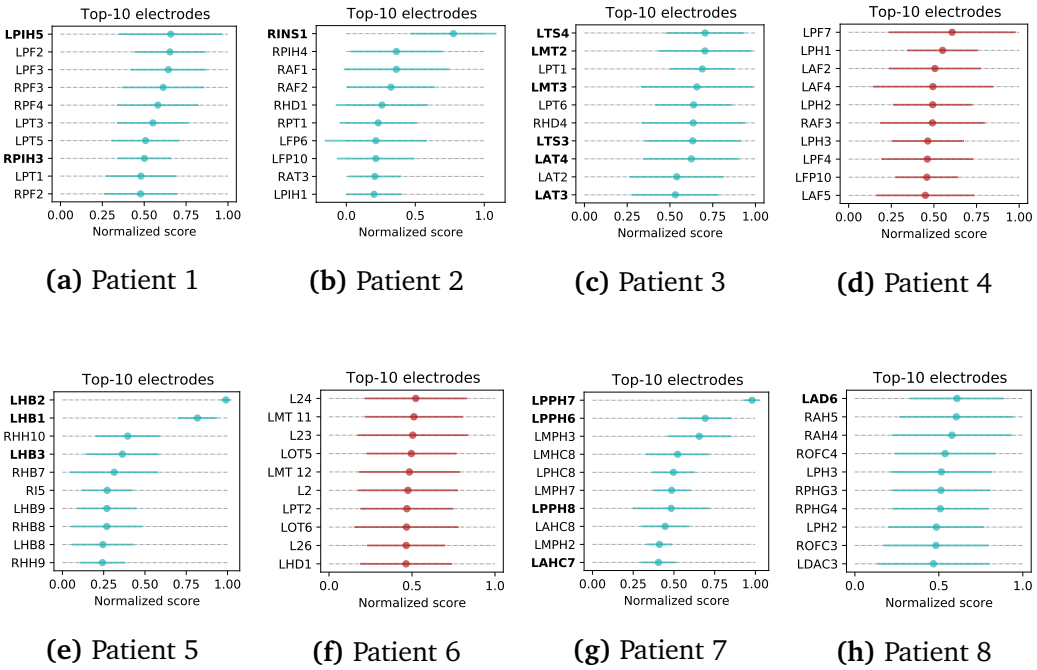
Our work introduces a methodology for automated seizure localisation using graph-based machine learning. Our approach does not require any manual annotation of the SOZ in order to work, making it cheaper to train and easier to scale to a larger number of patients. Our method is also data-efficient: we were able to provide a good—and clinically verified—localisation using as little as five annotated seizures per patient.

The goal of the proposed approach is to provide a support tool for clinicians to allocate precious resources in the analysis of iEEG data, and to improve the efficiency of the decision-making process. Crucially, in this regard, we note that our algorithm is conservative in scoring potential SOZ candidates. When the SOZ was not identifiable by electroencephalographers, the GNN also showed uncertainty in the scoring (rather than making high-confidence predictions). Contrarily, a high importance score consistently correlated with clinically-identified SOZs. With this premise, we believe that our approach could have practical value if deployed to epilepsy monitoring units to provide real-time analysis of iEEG recordings.



(a) Patient 5 **(b)** Patient 4 **(c)** Patient 6

Figure 6.4. Histograms of the attention scores over a 2-second window starting from a seizure onset. Each bin represents the frequency with which the corresponding attention score is assigned to ten randomly-selected electrodes. Figure **(a)** shows a patient with a known SOZ, while Figures **(b)** and **(c)** show patients without a known SOZ. For Figure **(a)**, the contribution to each bin of those electrodes that are part of the SOZ ground truth are highlighted in orange. Note how the score distribution for SOZ-associated electrodes is spread out towards higher values, while for patients with no known SOZ the scores are similar for all channels.



(a) Patient 1 **(b)** Patient 2 **(c)** Patient 3 **(d)** Patient 4

(e) Patient 5 **(f)** Patient 6 **(g)** Patient 7 **(h)** Patient 8

Figure 6.5. Top ten electrodes when considering the averaged rankings. We report the ranking obtained with the best-performing FC metric for each patient, according to the AP@10 score for average rankings reported in Tables 6.3 and 6.4. The two plots in red indicate those patients for which the SOZ was not identified clinically. Bold labels indicate that the corresponding electrode was marked as a potential SOZ by electroencephalographers. For every electrode, we report the average score and its standard deviation over all test seizures and all repetitions.

Chapter 7

Autoregressive models for graph sequences

Predicting the temporal evolution of a multivariate stochastic process is a widely explored problem in system identification and machine learning. However, the extension of this problem to the case of attributed graphs poses significant challenges due to the lack of basic operations to process the sequence directly in the domain of graphs. In an aim to address these issues, in this chapter we focus on the development and analysis of autoregressive (AR) models for learning to predict sequences of graphs [247].

We begin by formalising the idea of AR models for graph-valued variables, by making the classical AR formulation more general. Then, we propose an AR neural network to predict the next graph in a sequence. The architecture consists of a GNN block to map an input sequence of graphs to a sequence of vectors, followed by a recurrent neural network to predict the next vector in the sequence, and a graph decoder to map the predicted vector back to a graph.

We run experiments on two graph-based dynamical systems in which the attributes and connectivity can change at every time step. The results show that our approach is better than simpler baselines at predicting the evolution of the systems.

7.1 Autoregressive models for graphs

Several physical systems can be described using AR models in which, at every time step, the observation is modelled as the realisation of a random variable that depends on the previous observations.

In the traditional setting for AR models, each observation generated by the process is modelled as a vector $\mathbf{x}_{t+1} \in \mathbb{R}^d$ so that

$$\begin{cases} \mathbf{x}_{t+1} = f\left(\mathbf{x}_t, \mathbf{x}_{t-1}, \ldots, \mathbf{x}_{t-p+1}\right) + \boldsymbol{\epsilon}, \\ \mathbb{E}[\boldsymbol{\epsilon}_i] = 0 \\ \text{Var}[\boldsymbol{\epsilon}_i] = \sigma^2 < \infty, \end{cases} \tag{7.1}$$

where

$$f : \mathbb{R}^{p \times d} \to \mathbb{R}^d \tag{7.2}$$

and $\boldsymbol{\epsilon} \in \mathbb{R}^d$ is a random noise vector. Given the model in (7.1), the prediction of $\mathbf{x}_{t+1}$ is given by

$$\hat{\mathbf{x}}_{t+1} = \mathbb{E}_{\boldsymbol{\epsilon}}[\mathbf{x}_{t+1}] = f\left(\mathbf{x}_t, \mathbf{x}_{t-1}, \ldots, \mathbf{x}_{t-p+1}\right). \tag{7.3}$$

The predictor from Equation (7.3) is optimal when considering the L_2 norm between $\hat{\mathbf{x}}_{t+1}$ and $\mathbf{x}_{t+1}$.

Due to the lack of basic mathematical operators for graphs, the generalisation of model (7.1) to account for graph data is non-trivial. For instance, we have to deal with the fact that the sum between two graphs is generally not defined, with very few exceptions [15].

Let $\{\mathcal{G}_1, \ldots, \mathcal{G}_t, \ldots\}$ be a random process that generates a graph at every time step t. For simplicity of notation, we introduce the symbol $\mathfrak{G}$ to indicate the space of all possible attributed graphs. As for the numerical case, we model each observation of the process as the realisation of a random variable $\mathcal{G}_{t+1}$ through an AR function

$$\phi : \mathfrak{G}^p \to \mathfrak{G}. \tag{7.4}$$

Similar to Equation (7.1), we model $\mathcal{G}_{t+1}$ as

$$\mathcal{G}_{t+1} = H\left(\phi\left(\mathcal{G}_t, \ldots, \mathcal{G}_{t-p+1}\right), \eta\right), \tag{7.5}$$

where

$$H : \mathfrak{G} \to \mathfrak{G}. \tag{7.6}$$

is a function that models the effects of noise on the predicted graph. Function $H(\cdot, \cdot)$ is necessary because, even if we model η as a random graph variable in $\mathfrak{G}$, the sum between graphs is not generally defined.

The assumptions made on the noise in model (7.1) have to be adapted as well. The condition of unbiased noise in the classical formulation is $\mathbb{E}[\boldsymbol{\epsilon}_i] = 0$ or, equivalently

$$f\left(\mathbf{x}_t, \ldots, \mathbf{x}_{t-p-1}\right) = \mathbb{E}_{\boldsymbol{\epsilon}}[f\left(\mathbf{x}_t, \ldots, \mathbf{x}_{t-p-1}\right) + \boldsymbol{\epsilon}]. \tag{7.7}$$

In the case of graphs, the assumption of unbiased noise can be written as

$$\phi\left(\mathcal{G}_t,\dots,\mathcal{G}_{t-p+1}\right)\in\mathbb{E}^{\mathrm{f}}_{\eta}[H\left(\phi\left(\mathcal{G}_t,\dots,\mathcal{G}_{t-p+1}\right),\eta\right)], \tag{7.8}$$

where $\mathbb{E}^{\mathrm{f}}[\cdot]\in\mathfrak{G}$ is the set of mean graphs according to Fréchet [59]:

$$\mathbb{E}^{\mathrm{f}}[\mathcal{G}]=\operatorname*{ArgMin}_{\mathcal{G}'\in\mathfrak{G}}\int_{\mathfrak{G}}\mathrm{d}\left(\mathcal{G},\mathcal{G}'\right)^2\,dQ(\mathcal{G}) \tag{7.9}$$

where $\mathrm{d}(\cdot,\cdot)$ is a pre-metric graph distance, and Q is a graph distribution defined on the Borel sets of space $(\mathfrak{G},\mathrm{d})$. Possible choices for $\mathrm{d}(\cdot,\cdot)$ include the graph edit distances [2, 55, 25] and any distance derived from positive semi-definite kernel functions [186]. $\mathbb{E}^{\mathrm{f}}[\cdot]$ is defined as a set because, depending on the distribution Q, there can be more than one graph minimising Equation (7.9). Note that, for a sufficiently small Fréchet variation of the noise η (see Equation (7.11)) and a metric $\mathrm{d}(\cdot,\cdot)$, the Fréchet mean graph exists and is unique.

Equation (7.9) holds only when

$$\int_{\mathfrak{G}}\mathrm{d}\left(\mathcal{G},\mathcal{G}'\right)^2\,dQ(\mathcal{G})<\infty, \tag{7.10}$$

which can be interpreted as the graph counterpart of $\mathrm{Var}[\boldsymbol{\epsilon}_i]<\infty$ in model (7.1). The variance of a graph distribution can be expressed in terms of the Fréchet variation as

$$\mathrm{Var}^{\mathrm{f}}[\mathcal{G}]:=\min_{\mathcal{G}'\in\mathfrak{G}}\int_{\mathfrak{G}}\mathrm{d}\left(\mathcal{G},\mathcal{G}'\right)^2\,dQ(\mathcal{G}). \tag{7.11}$$

The final AR system model in the graph domain is

$$\begin{cases}\mathcal{G}_{t+1}=H\left(\phi\left(\mathcal{G}_t,\dots,\mathcal{G}_{t-p+1}\right),\eta\right),\\ \phi\left(\mathcal{G}_t,\dots,\mathcal{G}_{t-p+1}\right)\in\mathbb{E}^{\mathrm{f}}_{\eta}[H\left(\phi\left(\mathcal{G}_t,\dots,\mathcal{G}_{t-p+1}\right),\eta\right)],\\ \mathrm{Var}^{\mathrm{f}}[\eta]<\infty.\end{cases} \tag{7.12}$$

Note that the proposed graph AR model (7.12) is a proper generalisation of model (7.1). In fact, it can be shown that (7.12) reduces to (7.1), when considering $(\mathbb{R},\|\cdot\|)$—or more generally, $\left(\mathbb{R}^d,\|\cdot\|_2\right)$—instead of $(\mathfrak{G},\mathrm{d})$, and choosing $H(a,b)$ as the sum $a+b$ (see Appendix F.1 for a proof).

Given a system modelled by (7.12), we can predict graph at time $t+1$ as the graph $\hat{\mathcal{G}}_{t+1}$ minimising

$$\mathbb{E}\left[\mathrm{d}\left(\mathcal{G},\mathcal{G}_{t+1}\right)^2\right], \tag{7.13}$$

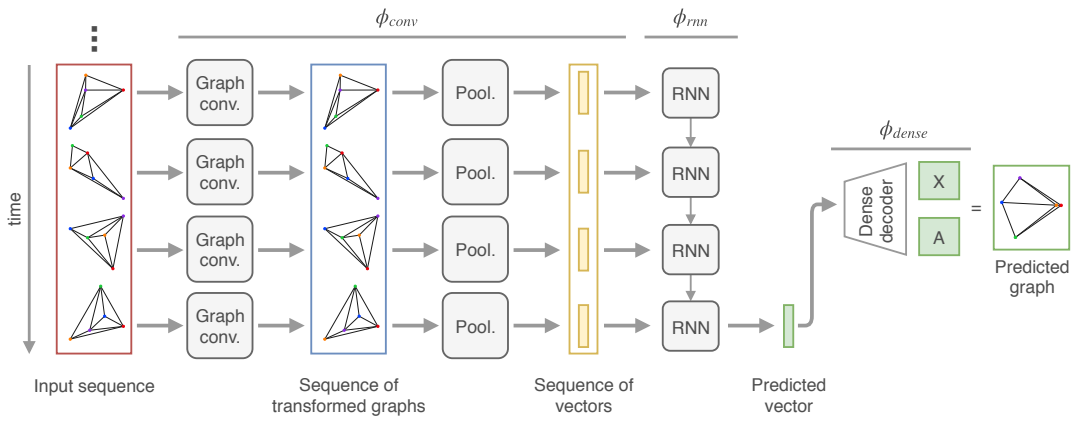


Figure 7.1. Schematic view of NGAR. The network takes as input a sequence of k graphs up to time t (red box) and predicts the graph at time $t+1$. The input is processed by a GNN, applied in parallel to all graphs, followed by a global pooling layer that compresses the graphs down to vectors. Then, a recurrent neural network predicts the representation of the next graph. Finally, the prediction is converted to a graph by a multi-layer perceptron with two outputs: one for the node features $\mathbf{X}_{t+1}$ and one for the adjacency matrix $\mathbf{A}_{t+1}$.

where the expectation is taken w.r.t. $\mathcal{G}_{t+1}$. Therefore, we have that the optimal prediction is

$$\hat{\mathcal{G}}_{t+1} = \operatorname*{ArgMin}_{\mathcal{G}\in\mathfrak{G}} \mathbb{E}\left[\mathrm{d}\left(\mathcal{G}, \mathcal{G}_{t+1}\right)^2\right] = \tag{7.14}$$

$$= \mathbb{E}^{\mathrm{f}}_{\eta}\left[H\left(\phi\left(\mathcal{G}_t, \ldots, \mathcal{G}_{t-p+1}\right), \eta\right)\right] = \tag{7.15}$$

$$= \phi\left(\mathcal{G}_t, \ldots, \mathcal{G}_{t-p+1}\right). \tag{7.16}$$

7.1.1 Neural graph recurrent autoregressive model

Given a graph-generating process described by an AR model of order p, the task of predicting the next graph in the sequence can be formulated as that of approximating $\phi(\cdot)$ in (7.12), since the optimal prediction is given by Equation (7.16). In order to approximate $\phi(\cdot)$ we propose to use a neural network, which can be seen as a family of parametric models

$$\phi_{\mathrm{nn}} : \mathfrak{G}^p \to \mathfrak{G} \tag{7.17}$$

that receive as input a regressor $G_{t,p} = [\mathcal{G}_t, \ldots, \mathcal{G}_{t-p+1}]$ and output the predicted graph

$$\hat{\mathcal{G}}_{t+1} = \phi_{\mathrm{nn}}(G_{t,p}). \tag{7.18}$$

Note that since the true order of the system is usually unknown, here the value p is a hyperparameter that must be chosen appropriately.

Our architecture for ϕ_{nn} is composed of three stages:

1. We map each graph in $G_{t,p}$ to a vector representation, using a GNN with a final readout layer;

2. We apply a recurrent neural network to the resulting vector sequence;

3. We obtain the predicted graph by mapping the predicted vector back to the graph domain.

The full model ϕ_{nn} is therefore obtained by the composition of three blocks, denoted ϕ_{gnn}, ϕ_{rnn}, and ϕ_{dec}. We refer to this model, schematised in Figure 7.1, as a *neural graph AR* (NGAR) model.

The first block of NGAR converts the input sequence $G_{t,p}$ to a sequence of l-dimensional vectors. This operation can be described by a map:

$$\phi_{\text{gnn}} : \mathfrak{G} \to \mathbb{R}^l, \tag{7.19}$$

which we implement using a GNN with a final readout step. By mapping graphs to vectors, we go back to the numerical setting of model (7.1). Also, by using a GNN we ensure that the graph embeddings will encode information related to both attributes and structure.

By applying ϕ_{gnn} to each graph in the regressor $G_{t,p}$, we obtain a sequence

$$Z_{t,p} = [\phi_{\text{gnn}}(\mathcal{G}_t), \ldots, \phi_{\text{gnn}}(\mathcal{G}_{t-p+1})] \tag{7.20}$$

of l-dimensional vectors, which we then process using a recurrent neural network:

$$\phi_{\text{rnn}} : \mathbb{R}^{p \times l} \to \mathbb{R}^l. \tag{7.21}$$

The role of this block is to produce the vector representation of the predicted graph, capturing the temporal dependencies in the input sequences. Here, we formulate the block as a recurrent network, but any method to map the sequence to a prediction is suitable (*e.g.*, 1-dimensional convolutional networks).

Finally, we convert the vector representation to the final prediction in the space of graphs, using a multi-head dense MLP similar to the ones proposed by Simonovsky and Komodakis [196] and De Cao and Kipf [45]:

$$\phi_{\text{dec}} : \mathbb{R}^l \to \mathfrak{G}. \tag{7.22}$$

7.1.2 Extensions of NGAR

Generating a graph by sampling its structure and attributes with a dense network has known limitations and implicitly assumes node identity and a maximum order of the graph. While this solution greatly simplifies the implementation of the GNN, other approaches can be used when dealing with more complex graphs, like the GraphRNN decoder proposed by You et al. [238]. In the follow-up work of reference [161], we have addressed these issues by introducing the Graph Edit Network (GEN), a general technique in which we train a GNN to edit the last observed graph in the sequence into the following one. The idea behind GEN is that, under reasonable operating conditions, the difference between two consecutive graphs will be small and, therefore, easy to model as a series of edit operations. GEN is a flexible technique that can be trained to model a variety of edits, including node and edge insertions and deletions, as well as changes in the attributes. In the paper, we prove that a mapping between the node sets of two consecutive graphs is sufficient to construct training data for a GEN and that an optimal mapping yields edit scripts that are almost as short as the graph edit distance between the graphs. We further provide a proof-of-concept empirical evaluation on several graph dynamical systems, which are difficult to learn for baselines from the literature. We refer the reader to the paper for more details [161].

7.2 Experiments with NGAR

In this section, we perform experiments on sequences of graphs to compare NGAR with other non-AR or non-graph-based baselines. We consider graphs with $N = 5$ identified nodes and variable topology. Each node is associated with a vector attribute of dimension $D_n = 2$, and no edge attributes. The sequences are produced by two synthetic graph-generating processes with a known memory order p, allowing us to have a ground truth for our analysis.

To have a fair comparison of the methods, we consider a graph edit distance (GED) [2]

$$\mathrm{d}(\mathcal{G}_{t+1}, \hat{\mathcal{G}}_{t+1}) \tag{7.23}$$

between the ground truth and the prediction made by the models. We also analyse the loss and accuracy of NGAR to show the relative performance of the model on problems of different complexity. Finally, we report a qualitative assessment of NGAR, by visualising the graphs predicted by the model and the true observations from the system.

7.2.1 Baseline methods

We consider four baselines commonly applied in the numerical case, that can be easily adapted to our setting. We indicate our proposed method as *NGAR*, while the four baselines as *Mean*, *Mart*, *Move*, and *VAR*, respectively.

Mean The first baseline (*Mean*) assumes that the system is stationary, with independent and identically distributed graphs. In this case, the optimal prediction $\hat{g}_{t+1}$ is the mean graph:

$$\hat{\mathcal{G}}_{t+1} = \mathbb{E}^{\mathrm{f}}[\mathcal{G}] \tag{7.24}$$

where $\mathcal{G} \sim Q$, and Q indicates here the distribution (supposed stationary) of the graphs.

Martingale The second baseline assumes that the process is a martingale (*Mart*), s.t. $\mathbb{E}^{\mathrm{f}}[\mathcal{G}_{t+1}] = \mathcal{G}_t$, and predicts:

$$\hat{\mathcal{G}}_{t+1} = \mathcal{G}_t, \tag{7.25}$$

i.e., the graph at the previous time step.

Moving average The third baseline (*Move*) considers the p last observed graphs $G_{t,p}$, and predicts $\mathcal{G}_{t+1}$ to be their Fréchet sample mean:

$$\hat{\mathcal{G}}_{t+1} = \operatorname*{ArgMin}_{\mathcal{G}'} \sum_{\mathcal{G}_i \in \vec{G}_{t,p}} \mathrm{d}(\mathcal{G}', \mathcal{G}_i)^2. \tag{7.26}$$

VAR The fourth baseline is a vector AR model (VAR) of order p, which treats each graph $\mathcal{G}_t$ as the vectorisation of its node features $\mathbf{X}_t$ and adjacency matrix $\mathbf{A}_t$, concatenated in a single vector:

$$\mathbf{u}_t = [\mathrm{vec}(\mathbf{X}_t)^\top, \mathrm{vec}(\mathbf{A}_t)^\top]^\top \in \mathbb{R}^{N \cdot D_n + N^2}. \tag{7.27}$$

We compute a prediction $\hat{\mathbf{u}}_{t+1}$ using the linear model

$$\hat{\mathbf{u}}_{t+1} = B_0 + \sum_{i=1}^{p} B_i \cdot \mathbf{u}_{t-i+1}, \tag{7.28}$$

with regressor $U_{t,p} = [\mathbf{u}_t, \ldots, \mathbf{u}_{t-p+1}]$ and, subsequently, we obtain the actual graph prediction $\hat{\mathcal{G}}_{t+1}$ by reshaping $\hat{\mathbf{u}}_{t+1}$.

Note that we can use the VAR baseline in these experiments only because here we are considering dynamical systems that produce graphs with a fixed size. In more general settings, this would not be possible.

7.2.2 Graph-generating processes

We consider two different systems, both based on a common framework where a multivariate AR model of the form

$$\mathbf{x}_{t+1} = f(X_{t,p}) + \epsilon, \tag{7.29}$$

produces a sequence of vectors $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_t, \dots\}$, which are then used to generate the graph sequence. From each $\mathbf{x}_t$, we obtain the node features $\mathbf{X}_t \in \mathbb{R}^{N \times D_n}$. The graph is the Delaunay triangulation of the rows of $\mathbf{X}_t$. ϵ is a noise term randomly drawn from a normal distribution $\mathcal{N}(0, \sigma^2)$.

The result of this process is a sequence of graphs $\{\mathcal{G}_1, \mathcal{G}_2, \dots, \mathcal{G}_t, \dots\}$, where each graph depends on the previous p graphs. By choosing different implementations of $f(X_{t,p})$, we can generate different graph-generating processes. Note that the noise η and the function $H(\cdot, \cdot)$ in model (7.12) are never made explicit in this procedure, but instead they are determined by the Gaussian noise perturbation introduced by ϵ, which affects the node attributes and causes possible changes in the topology.

Rotational model In our first model, we implement Equation (7.29) as:

$$f(X_{t,p}) = \mathbf{R}_{t,p} \cdot \mathbf{x}_t \tag{7.30}$$

where $\mathbf{x}_t \in \mathbb{R}^{N \cdot D_n}$ and $\mathbf{R}_{t,p} \in \mathbb{R}^{N \cdot D_n \times N \cdot D_n}$ is a block-diagonal rotation matrix with blocks $\mathbf{R}_{n,t,p} \in \mathbb{R}^{D_n \times D_n}$, $n = 1, \dots, N$, defined as:

$$\mathbf{R}_{n,t,p} = \begin{bmatrix} \cos(\omega) & \sin(\omega) \\ -\sin(\omega) & \cos(\omega) \end{bmatrix}, \tag{7.31}$$

where

$$\omega = c_n + \alpha \cdot \cos\left(\sum_{i=0}^{p-1} \mathbf{x}_{t-i}[2n-1] + \mathbf{x}_{t-i}[2n] \right). \tag{7.32}$$

The coefficients c_n are randomly sampled from a uniform distribution in $(-1, 1]$, while α is set to 0.01.

We obtain the node attributes by re-arranging each $\mathbf{x}_t$ in a $N \times D_n$ matrix. The regression function f can be interpreted as an independent rotation of each node feature (see Figure 7.2).

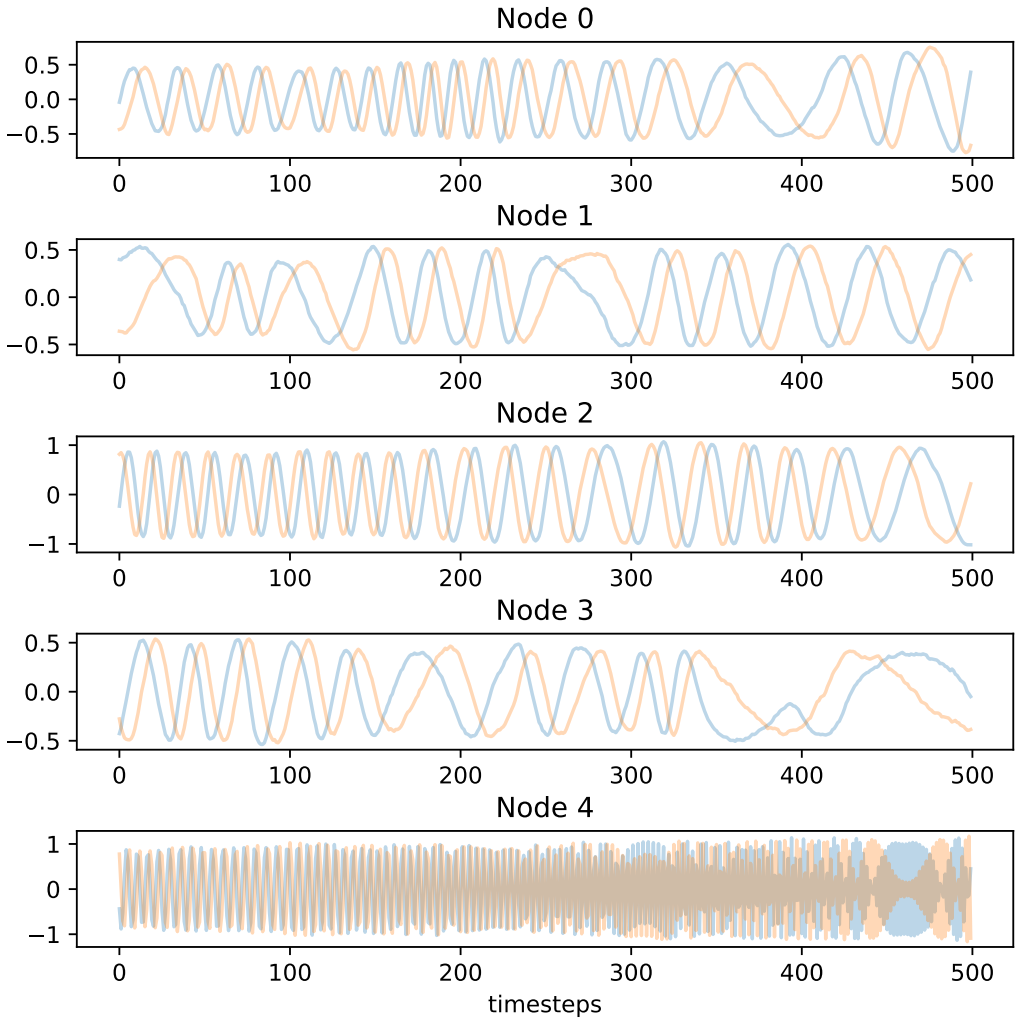


Figure 7.2. Evolution of the node attributes in the rotational model of order 10. Each plot shows the dynamics of the $D_n = 2$ attributes of each node in the graph.

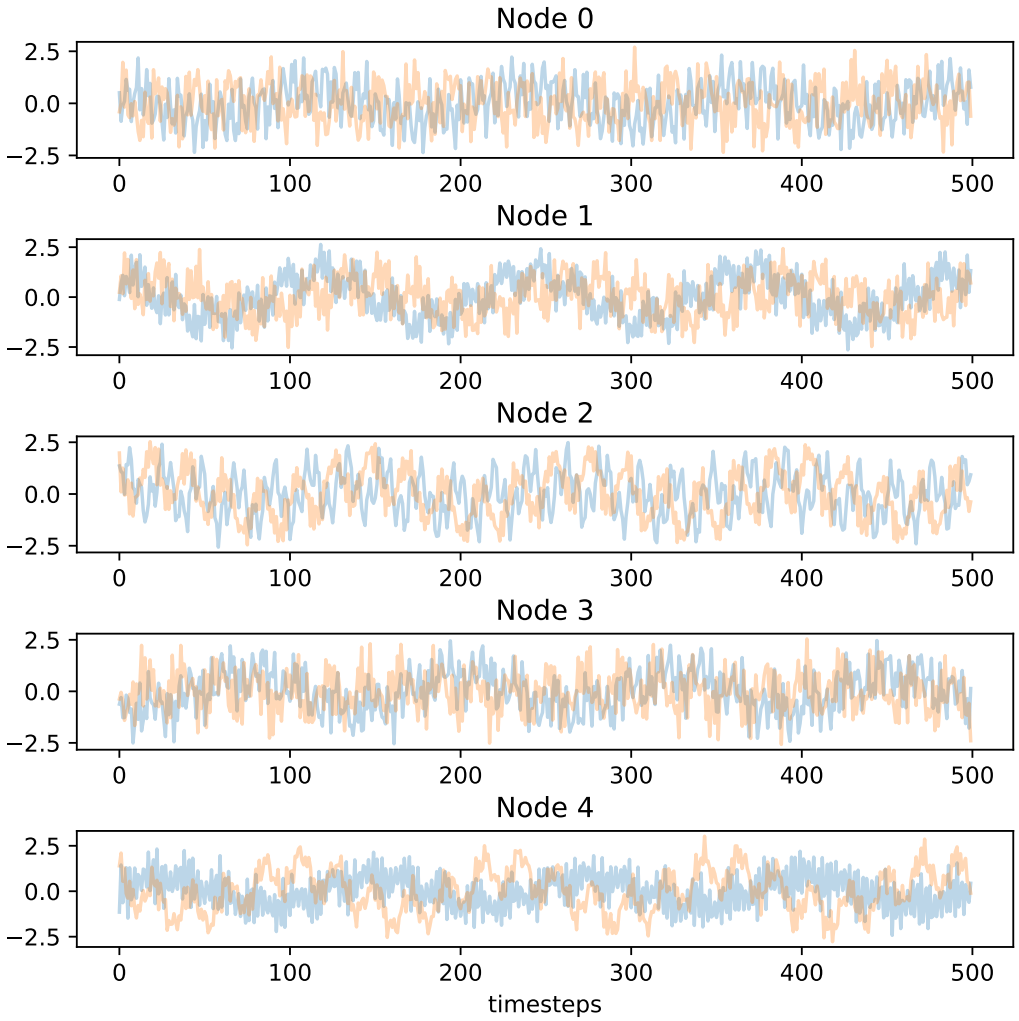


Figure 7.3. Temporal evolution of the node attributes in PMLDS(10). Each plot shows the dynamics of the $D_n = 2$ attributes of each node in the graph.

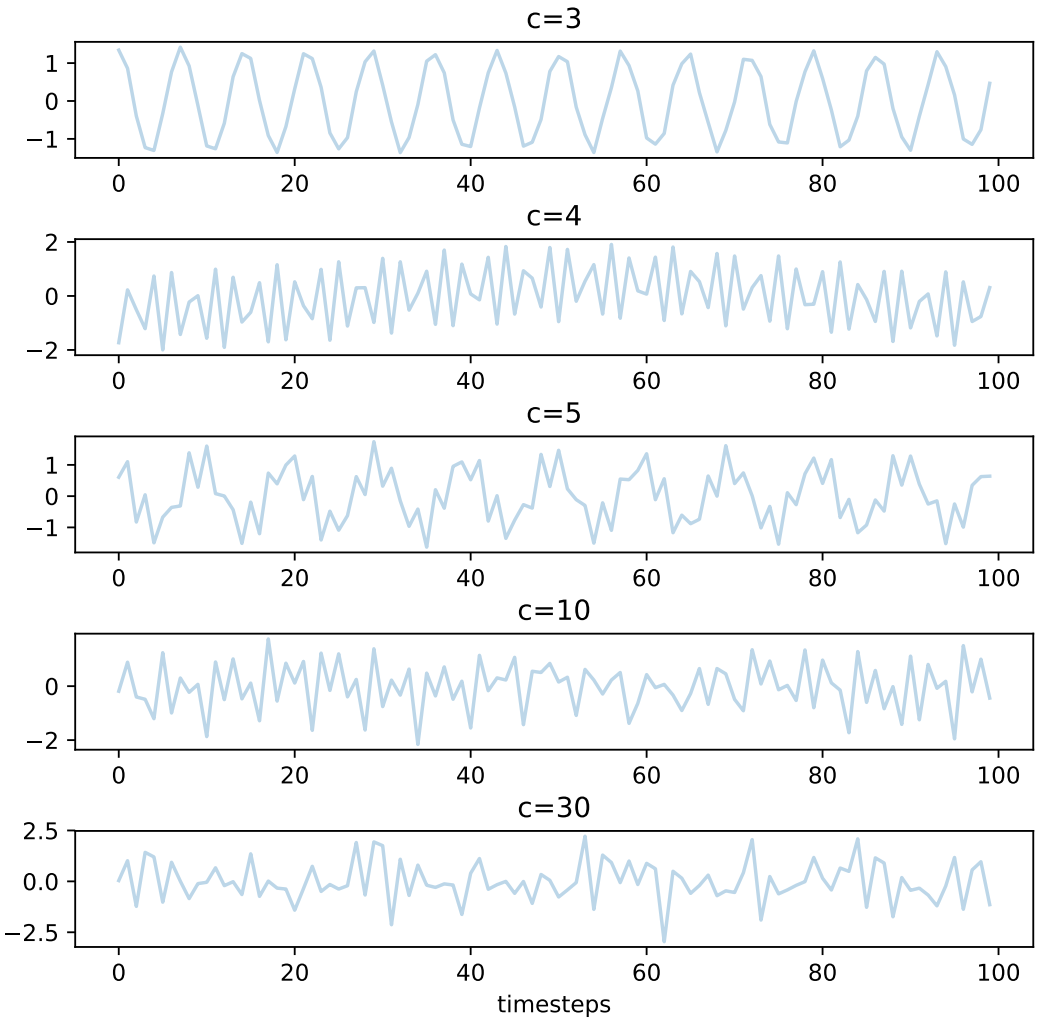


Figure 7.4. Temporal evolution of the first component of PMLDS, for different values of c. As c increases, it becomes more difficult to explain the dynamics.

Table 7.1. Hyperparameter configuration for the GNN used in the experiments.

Hyperparameter	Value
Weight for L_2	0.0005
Learning rate	0.001
Batch size	256
Early stopping	20 epochs

Partially Masked Linear Dynamical System We consider a discrete-time linear dynamical system

$$\mathbf{x}_{t+1} = \mathbf{R}\mathbf{x}_t, \tag{7.33}$$

where $\mathbf{x}_t \in \mathbb{R}^c$, and $\mathbf{R} \in \mathbb{R}^{c \times c}$ is an orthonormal random correlation matrix computed with the method proposed by Davies and Higham [44].

Although the dynamical system in Equation (7.33) depends on exactly one previous time step, the partial observation of the first $N \cdot D_n < c$ components of $\mathbf{x}_t$ results in a dynamical system of order $p \propto (c - N \cdot D_n)$ [160]. Similar to the rotational model, we obtain $\mathbf{X}_t$ by reshaping the masked vectors to $N \times D_n$ (see Figure 7.3). We refer to this setting as a *partially masked linear dynamical system* of complexity c (PMLDS(c)).

The size c of the original linear system is an index of complexity of the problem, as the system's memory is dependent on it: given N, a higher c will result in more complicated dynamics of $\mathbf{x}_t$ and *vice versa* (see Figure 7.4). However, because the system is controlled by hidden variables that the NGAR model never sees at training time, the problem is closer to real-world scenarios where processes are rarely fully observable.

7.2.3 Details

We use the same GNN architecture for both settings. All hyperparameters of ϕ_{nn} were found through a grid search of common values, using the validation loss as a metric. We report them in Table 7.1. All component networks were structured with two layers to provide sufficient expressivity and. In particular, ϕ_{gnn} and ϕ_{dec} have been shown to be effective architectures for processing small graphs [47, 196]. Our network consists of two convolutional layers with 128 channels, ReLU activations, and L_2 regularisation. For convolution, we use a first-order polynomial GCN (*cf.* Section 2.3.1), although any other method is suitable. The convolutions are followed by a gated global pooling layer with

Table 7.2. Test performance of NGAR on the rotational model of order p. We report the full prediction loss as well as its two terms for **X** and **A** separately. For **A**, we also report the accuracy achieved by the model.

p	**Loss**	**Loss X**	**Loss A**	**Acc. A**
1	1.108	0.715	0.334	0.86
5	0.341	0.076	0.227	0.92
10	0.326	0.090	0.197	0.92
20	0.336	0.105	0.194	0.92
50	0.479	0.193	0.244	0.90
100	0.541	0.250	0.246	0.90

128 channels [124]. We apply ϕ_{gnn} in parallel (*i.e.*, with shared weights) to all p graphs in the input sequence, and feed the resulting vector sequence to a 2-layer LSTM [86] with 256 units and hyperbolic tangent activations. Note that we set $p = 20$ for NGAR, Move, and VAR. The output of the LSTM is fed to a fully connected network of two layers with 256 and 512 units, ReLU activations, and L_2 regularisation. Finally, the network has two parallel output layers with $N \cdot N$ and $N \cdot D_n$ units respectively, to produce the adjacency matrix and node attributes of the predicted graph. The output layer for **A** has sigmoid activations, and the one for **X** is a linear layer. The network is trained until convergence using Adam [102], monitoring the validation loss on a held-out 10% of the training data for early stopping. We jointly minimise the mean squared error for the predicted node attributes and the log-loss for the adjacency matrix. For each problem, we evolve the systems for 10000 steps and test the model on 10% of the data.

7.2.4 Results

Tables 7.2 and 7.3 report the test performance of NGAR, in terms of prediction loss and accuracy. We see that, as the problem complexity increases (p and c), the test performance gets worse. In Table 7.2, for the rotational model with $p = 1$ we observe an unexpected high test loss, which might be due to overfitting. We also see from Figure 7.5 that the test performance is consistently better when the complexity of the problem is lower than the order of NGAR, *i.e.*, $p \leq k$ and $c - N \cdot D_n \leq k$ (*cf.* Sections 7.2.2 and 7.2.2).

The second part of the experimental analysis aims at comparing the NGAR method with the baselines in terms of prediction error, assessed as the GED be-

Table 7.3. Test performance of the NGAR model on PMLDS(c). We report the full prediction loss as well as its two terms for **X** and **A** separately. For **A**, we also report the accuracy achieved by the model.

c	**Loss**	**Loss X**	**Loss A**	**Acc. A**
11	0.200	0.018	0.154	0.95
15	0.278	0.041	0.195	0.92
20	0.283	0.038	0.204	0.92
30	0.341	0.081	0.222	0.90
60	1.366	0.983	0.345	0.85
110	4.432	3.950	0.418	0.82

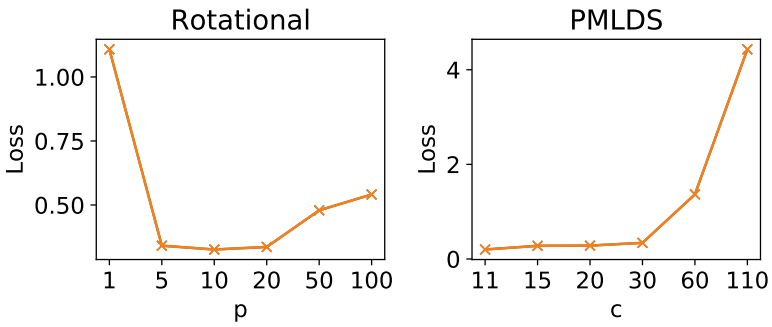


Figure 7.5. Test loss of NGAR for different levels of complexity on the rotational and PMLDS models.

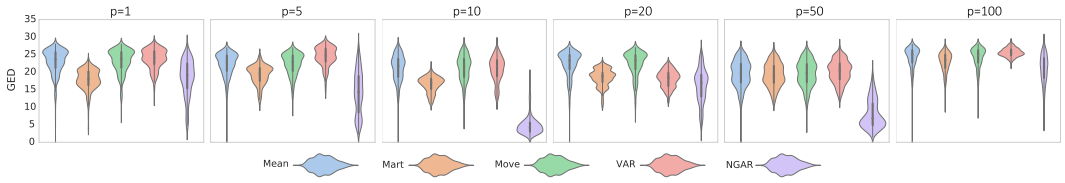


Figure 7.6. Comparison of Mean (blue), Mart (orange), Move (green), VAR (red), and NGAR (purple) on different rotational systems. Each plot shows the distribution of the residual GED between the targets in the test set and the graphs predicted by each model respectively.

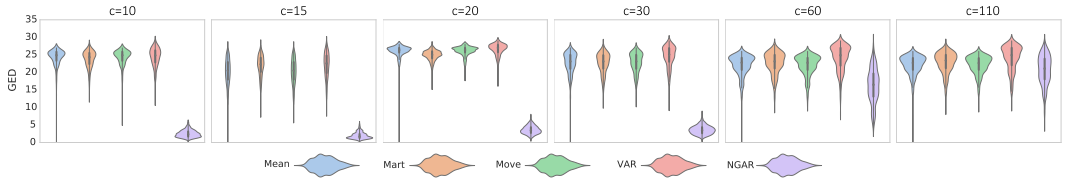


Figure 7.7. Comparison of Mean (blue), Mart (orange), Move (green), VAR (red), and NGAR (purple) on different PMLDS(c) settings. Each plot shows the distribution of the residual GED between the targets in the test set and the graphs predicted by each model respectively.

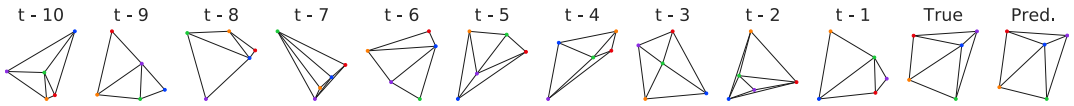


Figure 7.8. Comparison of a graph predicted by the NGAR model with the ground truth, on PMLDS(30).

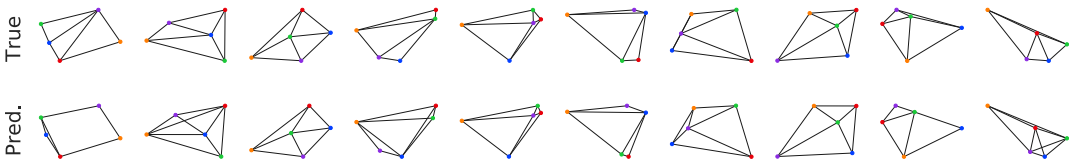


Figure 7.9. Comparison between randomly selected graphs in the test set of PMLDS(30) and the corresponding graphs predicted by the NGAR model. Upper row: samples from the ground truth. Bottom row: graphs predicted by the GNN.

tween predicted and ground-truth graphs; Figures 7.6 and 7.7 show a comparison of the baselines for different levels of complexity. NGAR consistently outperformed the baselines in almost all settings. On the other hand, among the baselines, we cannot identify one with significantly better performance. Given the performance of Mean and Mart for all p and c, we see that neither system is stationary or a martingale. Mean and Move performed almost the same in all experiments, while Mart performed significantly better than the other baselines in some cases, *e.g.*, see the rotational system with $p = 10$. Despite being widely used in multivariate AR problems, VAR was not able to compute meaningful predictions.

Finally, we show a qualitative assessment of the predictions in Figures 7.8 and 7.9, which highlight a good performance of NGAR even on sequences of graphs that cannot be predicted intuitively.

Chapter 8

Conclusion

8.1 Summary

In this thesis we have presented numerous contributions to the field of graph machine learning, organised along two main research directions.

In Part I, we have covered operators. Specifically, in Chapter 3, we have introduced a graph convolutional layer based on autoregressive moving-average graph filters, showing that it overcomes the modelling limitations of typical polynomial GNNs and that it outperforms many competitive baselines on tasks of node-level and graph-level learning.

In Chapter 4, we have studied pooling operators in detail. First, we have proposed a unifying and modular framework (SRC) to implement any pooling operator, leading us to organise the vast literature under a common taxonomy. We have also proposed two new pooling methods inspired by the complementary problems of finding the minimum and maximum cut on a graph: MinCut and Node Decimation Pooling (NDP). Finally, we have proposed a suite of evaluation benchmarks for pooling operators, based on measuring their information-preservation ability for attributes, structure, and task-specific information. The two methods proposed in this work achieve state-of-the-art results in their respective category of the taxonomy (respectively: MinCut is dense, trainable and fixed, NDP is sparse, non-trainable and adaptive).

In Part II, we have focused on architectures and their practical applications. In Chapter 5, we have presented the constant-curvature manifold adversarial autoencoder (CCM-AAE), a general family of autoencoders designed to learn representations on non-Euclidean manifolds (namely, hyperspherical and hyperbolic spaces). We have applied the CCM-AAE to the tasks of molecule generation and change detection. For the former, we have shown that the CCM-AAE achieves a

more balanced generation of small molecules compared to previous baselines. In particular, it is more likely that the CCM-AAE will generate a novel, chemically valid, and unique molecule (whereas previous methods tended to have much less variety in the generation). For change detection, we have proposed two novel algorithms for detecting changes in non-Euclidean spaces. We have focused on two tasks: detecting epileptic seizures in a stream of functional brain networks and detecting when the interaction between two subjects (represented by their skeletal graphs) changes from friendly to hostile. In all tasks, we have observed that modelling data in a non-Euclidean latent space leads to significant improvements in the final performance.

In Chapter 6, we have proposed a GNN architecture designed to be explainable. The model was specifically designed for the task of localising the seizure onset zone in patients with epilepsy. We trained our model, in a supervised way, to perform seizure detection, using functional brain networks extracted from electroencephalography data. By learning to solve the main task, the attention-based readout of the GNN also learned, implicitly, to place more attention on those areas of the brain that were most important to discriminate between interictal and ictal functional networks. By analysing the attention scores, we were able to identify the potential seizure onset zones of a subject. Empirically, we observed that the areas identified by the GNN were in agreement with the areas identified by professional electroencephalographers. The results were also validated on two simulators of brain activity.

Finally, in Chapter 7, we have discussed how to model stochastic sequences of graphs as autoregressive (AR) processes. First, we extended the typical AR model to account for graph-valued variables, generalising the concept of additive noise to the graph domain. Then, we propose an AR architecture to solve the prediction problem. The model consists of a GNN block to compute a vector representation of the input graphs, a recurrent neural network to predict the next code in the vector sequence, and finally a graph decoder to map the predicted vector back to the domain of graphs. The model showed competitive performance compared to simpler baselines.

8.2 Future work

The research presented in this thesis covers many different topics and applications of graph machine learning and GNNs and we believe that our work has uncovered some interesting opportunities for future research in the field.

Operators The design of more expressive methods for learning graph representations is an open challenge. The study of convolutional operators has now moved towards a more theoretical approach, in which their expressive power is measured based on the kinds of graphs that they are able to distinguish. The literature on the subject is incredibly rich and indicates a likely path forward in the study of GNNs [180]. Regarding our contributions in this area, the ARMA GNN was designed primarily as a method to learn node representations on static graphs with no edge attributes. However, the design of ARMA GNNs naturally lends itself to dealing with dynamical graphs and the extension of the ARMA GNN to account for edge attributes could also make the model more versatile on a large class of problems (*e.g.*, in chemistry or social network analysis).

We have also highlighted that the topic of graph pooling is in dire need of further exploration. While representing a significant step forward on the subject, the *Select, Reduce, Connect* framework, the taxonomy, and the suite of experimental benchmarks that we have presented are merely tools to accelerate future research. The experimental results of Section 4.4 are useful to provide general guidelines on how to choose a pooling operator in practice, but they also highlight that no single method is consistently better than all others. This leads to two considerations. First, a one-size-fits-all pooling operator has yet to be discovered, and we conjecture that it will likely combine the strengths of existing methods. Namely, we expect that an operator with a strong inductive bias for preserving structure will perform reliably across tasks. Also, the flexibility of dense and trainable operators is evident from our results and the literature, although a severe limitation of current methods is that they are fixed (whereas an adaptive pooling strategy is necessarily better). Our second consideration is that the question of whether having pooling operators in GNNs at all, which has been discussed in works like that of Mesquita et al. [145], might depend strongly on the task. It is still not clear what problems or types of data benefit from having hierarchical representations, but we argue that this underlying hierarchy should be evident to experts in the associated domain (*e.g.*, one field where hierarchy and pooling are universally acknowledged to be useful is that of computer vision on point clouds).

Architectures The future research directions that we envision from the perspective of architectures are tightly related to the specific application domains.

The CCM-AAE has proved to be a powerful architecture for learning unsupervised representations of all kinds of data, although its success is mostly due to the general idea of modelling data in non-Euclidean spaces. In this regard, the

literature on the subject is vast (*cf.* Section 5.1) and there are no doubts about the advantage of non-Euclidean representations. Any applications based on studying distances between graphs, or where the distance between graphs carries important meaning, can benefit from this kind of representation. We have applied it to molecule generation (small distance in the latent space means small semantic distance in molecule space) and change detection (measuring the distance from the nominal mean), but the potential applications are countless.

Our pipeline for seizure localisation using attention in GNNs has had success in identifying the seizure onset zones in real subjects with epilepsy and shows a promising future for applying GNNs in neuroscience. Our results are encouraging, but further research is obviously needed to assess the generality of our localisation method. For this, a larger sample of subjects and a fine-tuning of the model could validate our results further. Also, the results that we have shown in Chapter 6 are patient-specific, and a limitation of the model is that it must be trained specifically for every patient. A large-scale study would allow us to train models that can be transferred between subjects, making the deployment of our method easier and cheaper.

Finally, our work on modelling sequences of graphs is part of a larger corpus of literature about modelling time in the domain of graphs. Our contributions represent a first general formalisation of the problem. The NGAR model is a robust architecture for solving the prediction task in the domain of graphs but has evident limitations in the decoder. We have discussed a better approach to improve over NGAR in the work by Paassen et al. [161], although we believe that future work could explore the topic further. A key factor in the development of this topic is the availability of training data: most datasets of temporal networks usually have significant limitations, like a short time horizon or the absence of structural changes. The introduction of benchmarks to better understand the implications of modelling temporal graph sequences could give a significant boost to this research area.

8.3 Final remarks

In this thesis, we set out to study graph machine learning and to design operators and architectures to expand the state of the art of this vibrant field. We believe to have achieved this goal, even if our contributions represent only a small drop in a vast ocean of research.

The multifaceted nature of this work reflects the depth and variety of graph machine learning itself, and we hope that it will inspire the larger scientific com-

munity to design new artificial intelligence systems to solve real-world problems.

As we have discussed at the beginning of the thesis, the path to understanding general intelligence will likely require us to understand human intelligence first, the only example that we have. In this pursuit, designing our systems to model relations is a sensible choice that combines the strengths of the two main paradigms of artificial intelligence and reflects the nature of human thought.

From here, it is up to us to use these systems to understand the networks of our world.

Appendices

Appendix A

Experimental details for ARMA GNNs

A.1 Node classification

Table A.1 reports for each node classification dataset the number of nodes, number of edges, number of node attributes (size of the node feature vectors), the average shortest path of the graph (Avg. SP), and the number of classes that each node can be assigned to. The three citation networks (Cora, Citeseer, and Pubmed) are taken from `https://github.com/tkipf/gcn/raw/master/gcn/data/`, while the PPI dataset is taken from `http://snap.stanford.edu/graphsage/`.

Table A.2 describes the optimal hyperparameters used in GCN, Chebyshev, CayleyNet, and ARMA for each node classification dataset. For all GNN, we report the L_2 regularisation weight, the learning rate (lr) and dropout probability (p_{drop}). For GCN, we report the number of stacked graph convolutions (L). For Chebyshev, we report the polynomial order (K). For CayleyNet, we report the polynomial order (K) and the number of Jacobi iterations (T). For ARMA, we report the number of GCS stacks (K) and the stacks' depth (T). Additionally,

Table A.1. Summary of the node classification datasets.

Dataset	Nodes	Edges	Node attr.	Avg. SP	Node classes
Cora	2708	5429	1433	$5.87_{\pm 1.52}$	7 (single label)
Citeseer	3327	9228	3703	$6.31_{\pm 2.00}$	6 (single label)
Pubmed	19717	88651	500	$6.34_{\pm 1.22}$	3 (single label)
PPI	56944	818716	50	$2.76_{\pm 0.56}$	121 (multi-label)

Table A.2. Hyperparameters for node classification.

Dataset	L_2 reg.	p_{drop}	lr	**GCN** L	**Cheby.** K	**Cayley** K	T	**ARMA** K	T
Cora	5e-4	0.75	0.01	1	2	1	5	2	1
Citeseer	5e-4	0.75	0.01	1	3	1	5	3	1
Pubmed	5e-4	0.25	0.01	1	3	2	5	1	1
PPI	0.0	0.25	0.01	2	3	3	5	3	2

Table A.3. Summary of the graph regression dataset.

Samples	**Avg. nodes**	**Avg. edges**	**Node attr.**
133,885	8.79	27.61	1

we configured the MLP in GIN with 2 hidden layers and trained the parameter ϵ, while for GraphSAGE we used the *max* aggregator, to differentiate more its behaviour from GCN and GIN. Finally, GAT is configured with 8 attention heads and the same number of layers L as GCN.

Each model is trained for 2000 epochs with early stopping (based on the validation accuracy) at 50 epochs. We used full-batch training, *i.e.*, in each epoch the weights are updated one time, according to a single batch that includes all the training data.

A.2 Graph regression

The QM9 dataset used for graph regression is available at `http://quantum-machine.org/datasets/`, and its statistics are reported in Table A.3.

The hyperparameters are reported in Table A.4. Only for this task, CayleyNets use only 3 Jacobi iterations, since with more iterations we experienced numerical errors and the loss quickly diverged. All models are trained for 1000 epochs with early stopping at 50 epochs, using the Adam optimiser with learning rate 10^{-3}. We used batch size 64 and no L_2 regularisation.

Table A.4. Hyperparameters for graph regression.

Dataset	GCN	Cheby.	Cayley		ARMA		
	L	K	K	T	p_{drop}	K	T
QM9	3	3	3	3	0.75	3	3

Table A.5. Summary of the graph classification datasets.

Dataset	Samples	Classes	Avg. nodes	Avg. edges	Node attr.	Node labels
Bench-hard	1,800	3	148.32	572.32	–	yes
Enzymes	600	6	32.63	62.14	18	no
Proteins	1,113	2	39.06	72.82	1	no
D&D	1,178	2	284.32	715.66	–	yes
MUTAG	188	2	17.93	19.79	–	yes

A.3 Graph classification

The datasets Enzymes, Proteins, D&D, and MUTAG are taken from the Benchmark Data Sets for Graph Kernels `https://ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets`, while the dataset Bench-hard is taken from `https://github.com/FilippoMB/Benchmark_dataset_for_graph_classification`. The statistics of each graph classification dataset are summarised in Table A.5.

For all methods, we use a fixed architecture composed of three GNN layers, each with 32 output units, ReLU activations, and L_2 regularisation with a factor of 10^{-4}. All models are trained to convergence with Adam, using a learning rate of 10^{-3}, batch size of 32, and patience of 50 epochs. We summarise in Table A.6 the hyperparameters used for ARMA, Chebyshev, and CayleyNets on the different datasets.

A.4 Graph signal classification

To generate the datasets we used the code available at `github.com/mdeff/cnn_graph`. The models are trained for 20 epochs on each dataset. We used batches of size 32 for MNIST and 128 for 20news. In the 20news dataset, the word embeddings have size 200.

Table A.6. Hyperparameters for graph classification.

Dataset	GCN	Cheby	Cayley		ARMA		
	L	K	K	T	p_{drop}	K	T
Bench-hard	2	2	2	10	0.4	1	2
Enzymes	2	2	2	10	0.6	2	2
Proteins	4	4	4	10	0.6	4	4
D&D	4	4	4	10	0.0	4	4
MUTAG	4	4	4	10	0.0	4	4

Table A.7. Summary of the graph signal classification datasets.

Dataset	Nodes	Edges	Avg. SP	Class	Train	Val	Test
MNIST	784	5,928	$12.36_{\pm 5.45}$	10	55,000	5,000	10,000
20news	10,000	249,944	$4.21_{\pm 0.94}$	20	10,168	7,071	7,071

Table A.8. Hyperparameters for graph signal classification.

Dataset	L_2 reg.	lr	p_{drop}	GCN	Cheby.	Cayley		ARMA	
				L	K	K	T	K	T
MNIST	5e-4	1e-3	0.5	3	25	12	11	5	10
20news	1e-3	1e-3	0.7	1	5	5	10	1	1

Appendix B

Additional discussion on NDP

B.1 Kron reduction in graphs with self-loops

If $\mathbf{A}$ contains self loops, the existence of the strict inequality condition $\mathbf{L}_{ii} > \sum_{j=1,j\neq i}^{n} |\mathbf{L}_{ij}|$ discussed in Section 4.3.2 is no more guaranteed. However, it is sufficient to consider the loopy-Laplacian $\mathbf{Q} = \mathbf{D} - \mathbf{A} + 2\mathbf{I} \odot \mathbf{A}$. $\mathbf{Q}$ is now an irreducible matrix and $\mathbf{Q}_{ii} > \sum_{j=1,j\neq i}^{n} |\mathbf{Q}_{ij}| + \mathbf{A}_{ii}$ holds for at least least one vertex $i \in \mathcal{V}^{+}$. We note that the adjacency matrix can be univocally recovered: $\mathbf{A} = -\mathbf{Q} + \text{diag}(\{\sum_{j=1,j\neq i}^{N} \mathbf{Q}_{ij}\}_{i=1}^{N})$. Therefore, from the Kron reduction $\mathbf{Q}'$ of $\mathbf{Q}$ we can first recover $\mathbf{A}'$ and then compute the reduced Laplacian as $\mathbf{L}' = \mathbf{D}' - \mathbf{A}'$.

B.2 Derivation of the maximum cut upper bound

Let us consider the Rayleigh quotient

$$r(\mathbf{z}, \mathbf{L}) = \frac{\mathbf{z}^{\top}\mathbf{L}\mathbf{z}}{\mathbf{z}^{\top}\mathbf{z}}, \tag{B.1}$$

which assumes its maximum value $\lambda_{\max}$ when $\mathbf{z} = \mathbf{u}_{\max}$. When $\mathbf{z}$ is the partition vector in Equation (4.20), we have $r(\mathbf{z}, \mathbf{L}) \leq \lambda_{\max}$. As shown in Section 4.3.1,

the numerator in Equation (B.1) can be rewritten as

$$\mathbf{z}^\top \mathbf{L}\mathbf{z} = \sum_{i,j\in\mathcal{E}} \mathbf{a}_{ij}(z_i - z_j)^2 \tag{B.2}$$

$$= \sum_{i,j\in\mathcal{V}^+} \mathbf{a}_{ij}(z_i - z_j)^2 + \sum_{i,j\in\mathcal{V}^-} \mathbf{a}_{ij}(z_i - z_j)^2 + \sum_{i\in\mathcal{V}^+, j\in\mathcal{V}^-} \mathbf{a}_{ij}(z_i - z_j)^2 \tag{B.3}$$

$$= 0 + 0 + \sum_{i\in\mathcal{V}^+, j\in\mathcal{V}^-} \mathbf{a}_{ij} 2^2 \tag{B.4}$$

$$= 4 \cdot \text{cut}(\mathbf{z}), \tag{B.5}$$

since $z_i = 1$ if $i \in \mathcal{V}^+$ and $z_i = -1$ if $i \in \mathcal{V}^-$ according to Equation (4.20), and where cut($\mathbf{z}$) is the volume of edges crossing the partition induced by $\mathbf{z}$. From Equation (4.20), it also follows that the denominator in Equation (B.1) is $\mathbf{z}^\top\mathbf{z} = N$, since $z_i^2 = 1, \forall i$. By combining the results we obtain

$$\frac{4 \cdot \text{cut}(\mathbf{z})}{N} \leq \lambda_{\max}, \forall \mathbf{z} \in \mathbb{R}^N, \tag{B.6}$$

from which wee obtain a bound on the value of the maximum cut, MaxCut:

$$\text{MaxCut} \leq \lambda_{\max} \frac{\text{N}}{4}. \tag{B.7}$$

When considering the symmetric normalised Laplacian $\mathbf{L}_n$, we multiply Equation (B.1) on both sides by $\mathbf{D}^{-1/2}$, changing the denominator into:

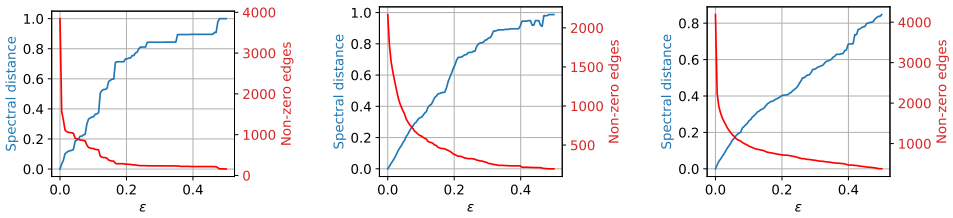
$$\mathbf{z}^\top \mathbf{D}\mathbf{z} = \sum_{i,i} d_{ii} z_i^2 = 2|\mathcal{E}|. \tag{B.8}$$

Replacing N with $2|\mathcal{E}|$ and $\lambda_{\max}$ with $\mu_{\max}$ in Equation (B.6), we get the bound

$$\text{MaxCut} \leq \mu_{\max} \frac{|\mathcal{E}|}{2}. \tag{B.9}$$

B.3 Relationship with Trevisan's spectral algorithm

Trevisan [208] shows that if $\mu_{\max} \geq 2(1-\tau)$, then there exist a set of vertices $\mathcal{V}$ and a partition $(\mathcal{V}^1, \mathcal{V}^2)$ of $\mathcal{V}$ so that $|e(\mathcal{V}^1, \mathcal{V}^2)| \geq \frac{1}{2}(1\sqrt{16\tau})\text{vol}(\mathcal{V})$, where $\text{vol}(\mathcal{V}) = \sum_{i\in\mathcal{V}} d_i$ and $e(\mathcal{V}^1, \mathcal{V}^2)$ are the edges with one endpoint in $\mathcal{V}^1$ and the other in $\mathcal{V}^2$. In cases where an optimal solution cuts $1-\tau$ fraction of the edges, a partition found by a recursive spectral algorithm will remove $1-4\sqrt{\tau}+8\tau$ of the edges. The optimal τ is value 0.0549 for which $\frac{1-4\sqrt{\tau}+8\tau}{1-\tau}$ reaches its minimum



(a) Regular grid **(b)** Community graph **(c)** Sensor network graph

Figure B.1. In blue, the variation of spectral distance between the Laplacian $\mathbf{L}$ associated with $\mathbf{A}$ and the Laplacian $\bar{\mathbf{L}}$ associated with the adjacency matrix $\bar{\mathbf{A}}$ sparsified with a varying threshold ϵ. In red, the number of edges that remain in $\bar{\mathbf{L}}$.

0.5311. When the largest eigenvalue $\mu_{\max}$ is too small, the expected random cut is larger than the solution found by the spectral algorithm. The analysis in [208] shows that the spectral cut is guaranteed to be larger than the random cut only when $\mu_{\max} \geq 2(1-\tau)$, *i.e.*, when $\mu_{\max} \geq 1.891$ given the optimal value $\tau = 0.0549$. Therefore, an algorithm that recursively cuts a fraction of edges according to the values in $\mathbf{v}_{\max}$ until $\mu_{\max} \geq 2(1-\tau)$ and then performs a random cut, finds a solution that is always larger than 0.5311 MaxCut.

B.4 Spectral similarity after sparsification

In Section 4.3.4, we introduced the spectral similarity distance to quantify how much the spectrum of the Laplacian associated with the sparsified adjacency matrix changes when edges smaller than ϵ are dropped. In Figure B.1 we show how the graph structure (in terms of spectral similarity) varies when the value of ϵ increases and more edges are dropped. In every example, for small values of ϵ the structure of the graphs changes only slightly while a large amount of edges is dropped. Notably, the spectral similarity increases almost linearly with ϵ, while the edge density decreases exponentially.

Appendix C

Experimental details for pooling operators

C.1 Experimental Details

C.1.1 Preliminaries

We use the same message-passing model in all of our experiments. The model is inspired by the work of You et al. [239] and has the following form:

$$\mathbf{x}'_i = \mathbf{x}_i \Big\| \sum_{j \in \mathcal{N}(i)} \text{ReLU}\left(\text{BN}\left(\mathbf{W}\mathbf{x}_j + \mathbf{b}\right)\right) \tag{C.1}$$

where BN indicates batch normalisation [91], ReLU is the rectified linear unit, $\mathbf{x}_i \in \mathbb{R}^{d_{in}}$, $\mathbf{x}'_i \in \mathbb{R}^{d_{out}}$, $\mathbf{W} \in \mathbb{R}^{d_{out} \times d_{in}}$ is a trainable matrix, $\mathbf{b} \in \mathbb{R}^{d_{out}}$ is a trainable bias, and $\Big\|$ indicates concatenation. We configure all layers to have $d_{out} = 256$.

We use GNN$(\mathbf{A}, \mathbf{X})$ to indicate the application of one layer as in Equation (C.1) to a graph described by $\mathbf{A}$ and $\mathbf{X}$.

We use MLP$(\mathbf{X})$ to indicate the application, to the features of each node, of one multi-layer perceptron (MLP) with 2 layers, 256 hidden units, ReLU activation and batch normalisation. The number of neurons of each layer is implied by the dimension of the data or, if not specified, is also set to 256 units (*e.g.*, for MLPs used as intermediate blocks).

All architectures and hyperparameters are also inspired by the work You et al. [239]. Specifically, we include a pre-processing and post-processing MLP as the first and last blocks of each architecture, respectively. The activation of the last layer of the post-processing MLP is implied by the task.

We use POOL$(\mathbf{A}, \mathbf{X})$ to indicate the application of a generic pooling layer, which is substituted in the architecture with the different operators that we compared in our experiments. Note that, for clarity, we change notation w.r.t. the main text and indicate with $\mathbf{X}_{pool}$, $\mathbf{A}_{pool}$ the output of a pooling layer.

C.1.2 Preserving node attributes

Architecture The bottleneck of the autoencoder consists of a single pooling layer to compress the graph signal, immediately followed by an *upscaling* layer. The overall architecture is:

$$\mathbf{X}, \mathbf{A} \leftarrow \mathcal{G} \tag{C.2}$$
$$\mathbf{X}_{in} \leftarrow \text{MLP}_{in}(\mathbf{X}) \tag{C.3}$$
$$\mathbf{X}_{in} \leftarrow \text{GNN}_{in}(\mathbf{A}, \mathbf{X}_{in}) \tag{C.4}$$
$$\mathbf{X}_{pool} \leftarrow \text{POOL}(\mathbf{A}, \mathbf{X}_{in}) \tag{C.5}$$
$$\mathbf{X}_{up} \leftarrow \text{UPSCALE}\left(\mathbf{X}_{pool}\right) \tag{C.6}$$
$$\mathbf{X}_{out} \leftarrow \text{GNN}_{out}\left(\mathbf{A}, \mathbf{X}_{up}\right) \tag{C.7}$$
$$\mathbf{X}_{out} \leftarrow \text{MLP}_{out}(\mathbf{X}_{out}). \tag{C.8}$$

All pooling layers are configured to reduce the graph to $K = \lfloor N/2 \rfloor$ nodes, except for LaPool which determines the number of output nodes autonomously.

The UPSCALE layer lifts the reduced node features $\mathbf{X}'$ of the coarsened graph $\mathcal{G}'$ back to the original data dimensionality of $\mathbf{X}_{in}$. For almost all methods, the upscaled node features $\mathbf{X}_{up}$ are constructed as $\mathbf{U}\mathbf{X}'$, where $\mathbf{U} = \mathbf{S}^{\mp}$ is the transposed pseudo-inverse of the selection matrix $\mathbf{S} = \text{SEL}(\mathbf{X}, \mathbf{A})$. This is the optimal choice when the reduction function is $\mathbf{X}' = \mathbf{S}^{\top}\mathbf{X}_{in}$. Conversely, for Top-$K$ and SAGPool we need to account also for the scaling factors $\sigma(\mathbf{y})$, so we have $\mathbf{U} = (\sigma(\mathbf{y}) \odot \mathbf{S})^{\mp}$. Finally, even though DiffPool's reduction is $\mathbf{X}' = \mathbf{S}^{\top}\text{GNN}(\mathbf{A}, \mathbf{X})$, we still employ $\mathbf{U}\mathbf{S}^{\mp}$ as upscaling operator and allow the output layer GNN_{out} to counteract the effects of the GNN.

Note that GNN_{out} takes as input the original adjacency matrix $\mathbf{A}$, to focus the experiment on the node features only and suppressing possible interference of the CON function.

Training All models are trained to convergence using Adam to minimise the mean squared error between the input and output features, with a learning rate 0.0005 and early stopping on the training loss with a patience of 1000 epochs and a tolerance of 10^{-6}. Each experiment is repeated 3 times.

Table C.1. Statistics of graphs used in the autoencoder and spectral similarity experiments.

Graph	Nodes	Edges	Avg. degree
Grid	64	112	3.5
Ring	64	64	2
Bunny	2503	65490	52.35
Airfoil	4253	12289	5.77
Sensor	64	313.7 ± 21.9	9.8 ± 0.8
Airplane	1333	2611	3.91
Car	1920	2372	2.47
Guitar	3125	5508	3.52
Person	3305	9055	5.47

Data The Grid, Ring, and Bunny graphs are generated using the PyGSP library [46]. The Airplane, Car, Person, and Guitar graphs are taken from the ModelNet40 dataset [228]. We selected one graph randomly from the training set of each category (with a threshold on the number of nodes). The ModelNet40 IDs of the selected graph are: sample 151 for Airplane, sample 75 for Car, sample 38 for Guitar, sample 83 for Person. Details on the size and average degrees of the graphs are reported in Table C.1. All datasets that were not generated programmatically are unlicensed.

C.1.3 Preserving structure

Architecture The architecture for this experiment consists only of a pooling layer, to ensure that the model is actually operating on the original coordinates and eigenvectors without transformations:

$$\mathbf{X}, \mathbf{A} \leftarrow \mathcal{G} \tag{C.9}$$

$$\mathbf{X}_{pool} \leftarrow \textsc{Pool}(\mathbf{A}, \mathbf{X}) \tag{C.10}$$

$$\tag{C.11}$$

All pooling layers are configured to reduce the graph to $K = \lfloor N/2 \rfloor$ nodes, except for LaPool which determines the number of output nodes autonomously.

Training For trainable models, we train them to convergence using Adam to minimise the quadratic loss described in the main text, with a learning rate 0.01

and early stopping on the training loss with a patience of 50 epochs and a tolerance of 10^{-6}. Each experiment is repeated 3 times.

Data All graphs are generated using the PyGSP library. In particular, Sensor is a random graph that is generated once per experiment and used for all models. Details on the size and average degrees of the graphs are reported in Table C.1. All datasets that were not generated programmatically are unlicensed.

C.1.4 Preserving task-specific information

Architecture We use the following architecture for all datasets:

$$\mathbf{X}, \mathbf{A} \leftarrow \mathcal{G} \tag{C.12}$$
$$\mathbf{X} \leftarrow \text{MLP}_1(\mathbf{X}) \tag{C.13}$$
$$\mathbf{X} \leftarrow \text{GNN}_1(\mathbf{A}, \mathbf{X}) \tag{C.14}$$
$$\mathbf{A}_{pool}, \mathbf{X}_{pool} \leftarrow \text{POOL}(\mathbf{A}, \mathbf{X}) \tag{C.15}$$
$$\mathbf{X}_{pool} \leftarrow \text{GNN}_2\left(\mathbf{A}_{pool}, \mathbf{X}_{pool}\right) \tag{C.16}$$
$$\mathbf{x}_{out} \leftarrow \sum_i \mathbf{x}_{pool,i} \tag{C.17}$$
$$\mathbf{x}_{out} \leftarrow \text{MLP}_2(\mathbf{x}_{out}) \tag{C.18}$$
$$\tag{C.19}$$

Adaptive pooling layers are configured to reduce the graph to $K = \lfloor N/2 \rfloor$ nodes, except for LaPool which determines the number of output nodes autonomously. Fixed pooling layers are configured to return $K = \left\lfloor \bar{N}/2 \right\rfloor$ nodes, where $\bar{N}$ is the average number of nodes in the training set.

Training All models are trained to convergence using Adam, with a batch size of 16, learning rate 0.0005 and early stopping on the validation loss with a patience of 50 epochs. Each experiment is repeated 3 times.

Data Proteins, Enzymes, Mutagenicity, DD, Colors-3 and Triangles are taken from the TUDataset collection [150]. The ModelNet10 dataset is taken from its original source [228]. All datasets are split randomly according to an 8:1:1 proportion between training, validation and test sets. The only exceptions are Colors-3 and Triangles, for which the data splits are described in [108], and ModelNet10, for which the data splits are given. The TUDatasets are unlicensed and ModelNet10 is provided freely for academic use.

Table C.2. Density and median weight of the edges of the coarsened graphs in the spectral similarity experiment.

		Original	DiffPool	MinCut	NMF	LaPool	TopK	SAGPool	NDP	Graclus
Grid2d	**Density**	0.055	0.969	0.969	0.463	0.917	0.084	0.092	0.189	0.103
	Median	1.000	0.216	0.024	0.018	1.445	1.000	1.000	0.500	0.154
Ring	**Density**	0.031	0.969	0.969	0.125	0.900	0.055	0.061	0.062	0.045
	Median	1.000	0.124	0.032	0.039	1.171	1.000	1.000	0.500	0.250
Bunny	**Density**	0.021	0.999	0.999	0.327	0.952	0.038	0.038	0.104	0.029
	Median	0.812	0.068	$7.55{\cdot}10^{-4}$	$9.99{\cdot}10^{-6}$	234.887	0.815	0.816	0.111	0.026
Minnesota	**Density**	$9.47{\cdot}10^{-4}$	0.999	0.999	0.010	0.999	0.002	0.002	0.003	0.002
	Median	1.000	0.004	$7.58{\cdot}10^{-4}$	0.014	0.013	1.000	1.000	0.333	0.204
Sensor	**Density**	0.159	0.969	0.969	0.844	0.875	0.273	0.230	0.529	0.227
	Median	0.742	0.463	$2.42{\cdot}10^{-4}$	0.005	6.147	0.765	0.756	0.201	0.103
Airfoil	**Density**	0.001	1.000	1.000	0.009	0.996	0.003	0.003	0.006	0.002
	Median	0.500	0.003	$4.70{\cdot}10^{-4}$	0.026	0.209	0.500	0.500	0.090	0.118

C.1.5 Memory usage

To produce Figure 3, we generated random Erdős-Rényi graphs with $p = 0.1$ and random features, and gave it as input to the trainable pooling methods (MinCutPool, DiffPool, Top-K and SAGPool). At each forward pass, we increased the number of nodes by 1000 until an out-of-memory exception was raised. We used a sparse tensor to represent the adjacency matrix of the input graphs (so that the cost of loading $\mathbf{A}$ into memory is linear in the number of edges). We repeated the experiments with node features of size $F = 1, 10, 100, 1000$ and found no significant differences in the results.

C.2 Additional results

C.2.1 Preserving node attributes

An extended version of Figure 4 in the main paper is reported in Figure C.1 here. An extended version of Figure 5 in the main paper is reported in Figures C.2 and C.3. Note that the missing plots for LaPool are due to the Out Of Memory exception as reported in Table 3 in the main paper.

C.2.2 Preserving structure

An extended version of Figure 6 in the main paper is reported in Figure C.4 here. An extended version of Table 5 is reported in Table C.2.

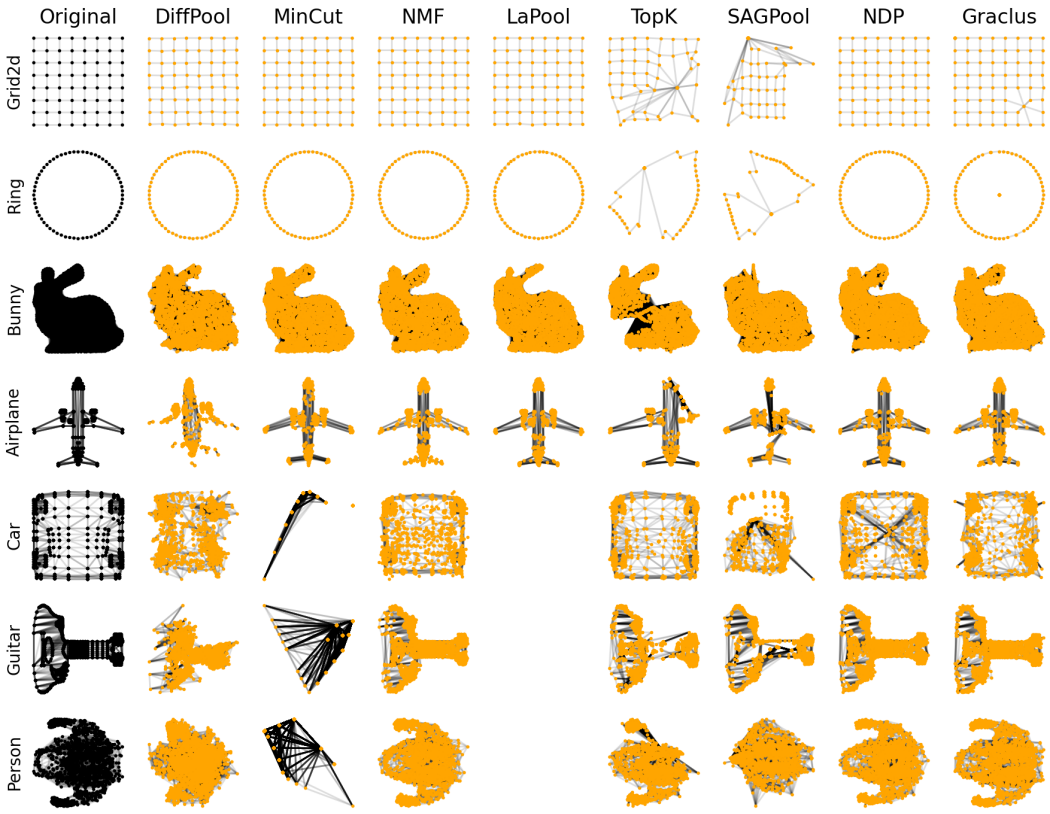


Figure C.1. Node attributes (point locations) reconstructed with different operators in the autoencoder experiment.

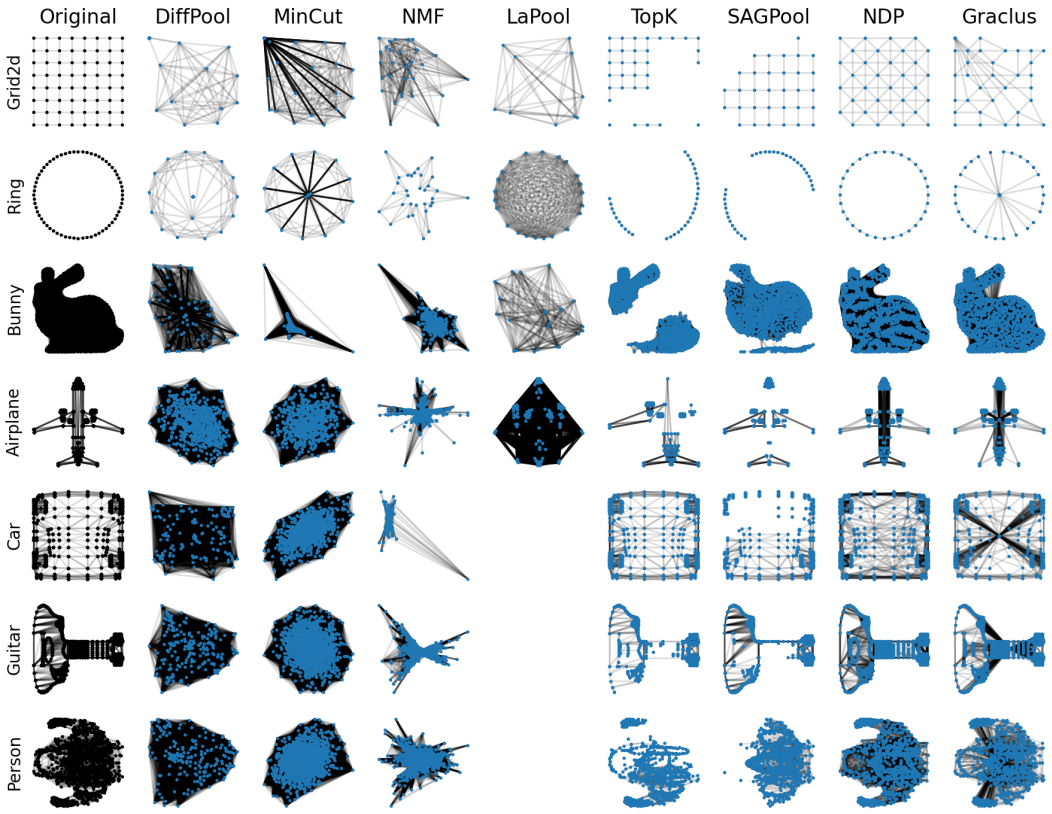


Figure C.2. Graphs pooled with different operators in the autoencoder experiment with the modified RED function.

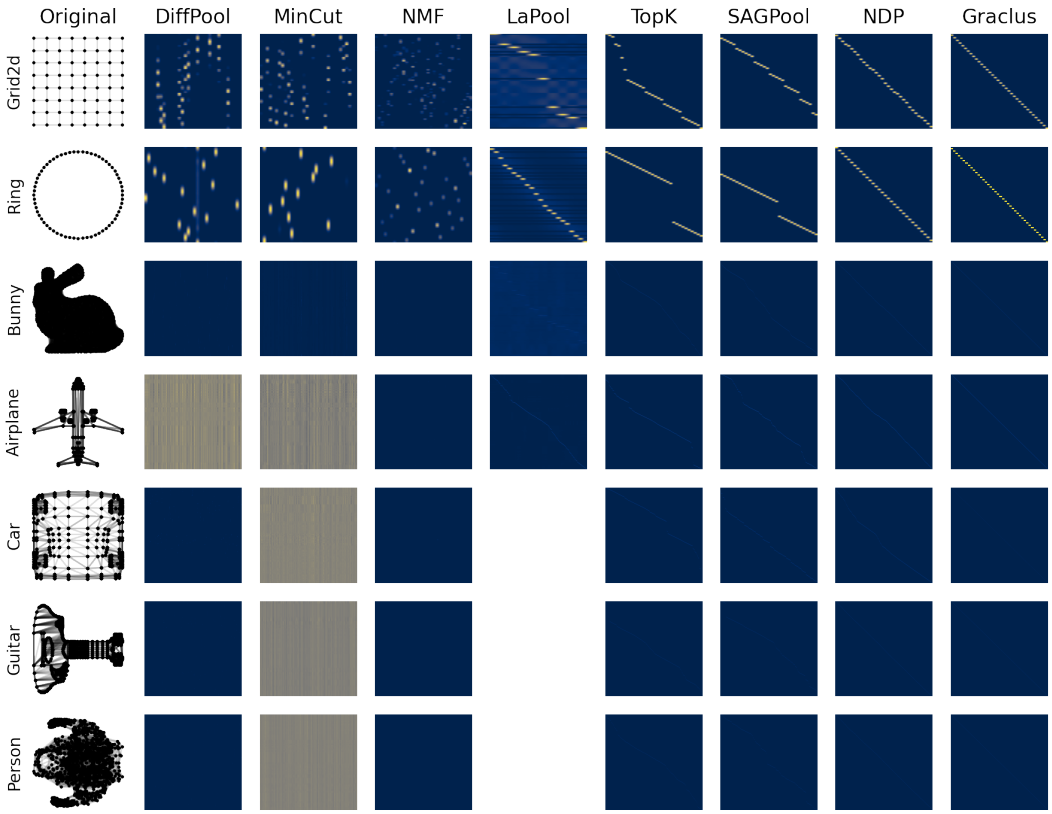
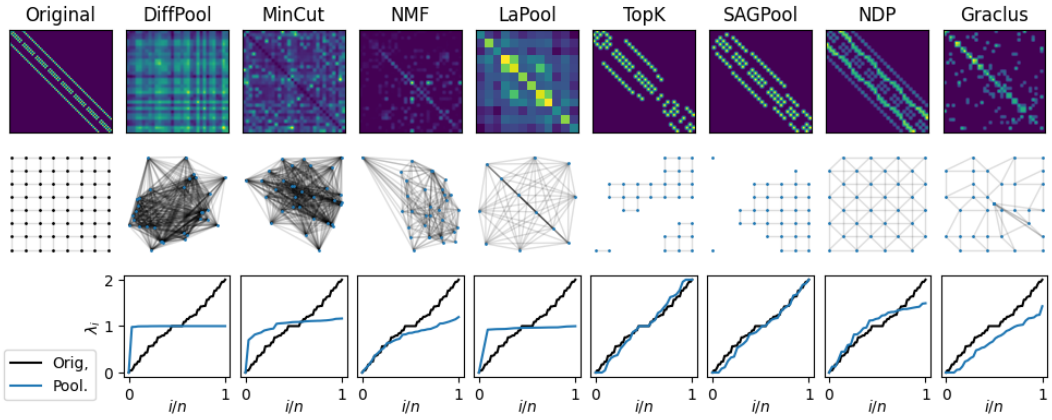
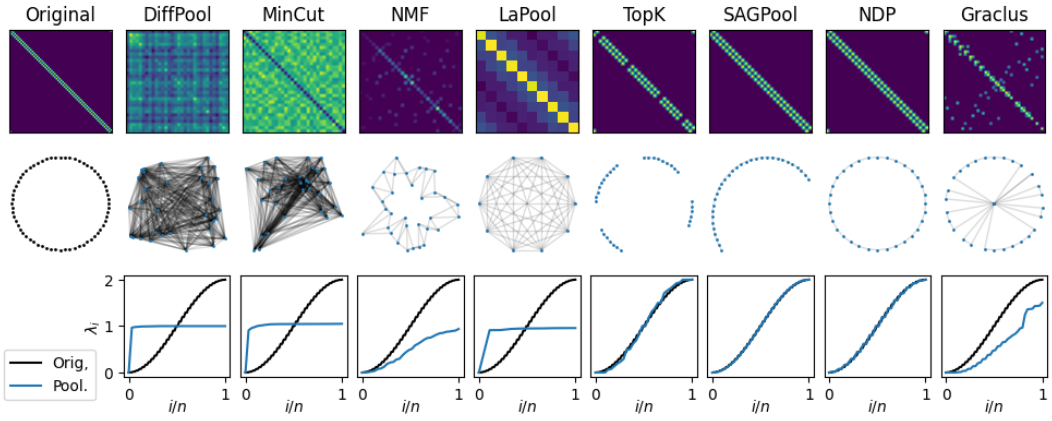


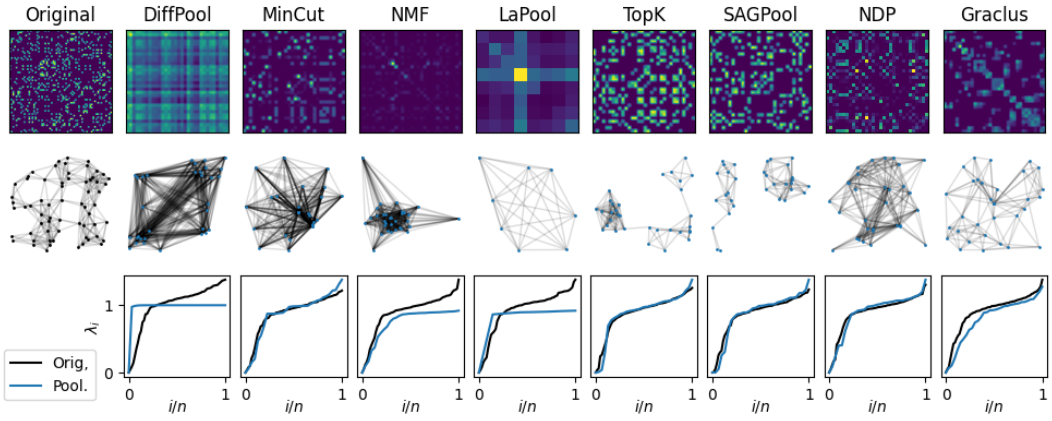
Figure C.3. Selection matrices computed with different operators in the autoencoder experiment.



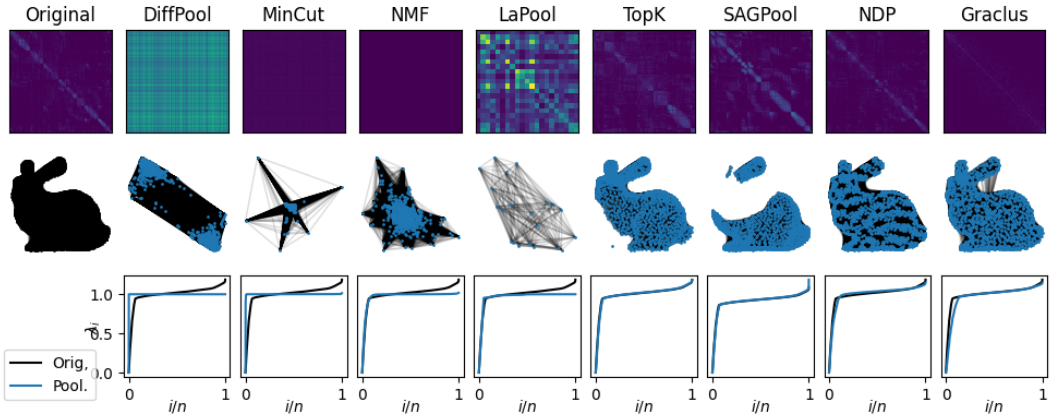
(a) Grid



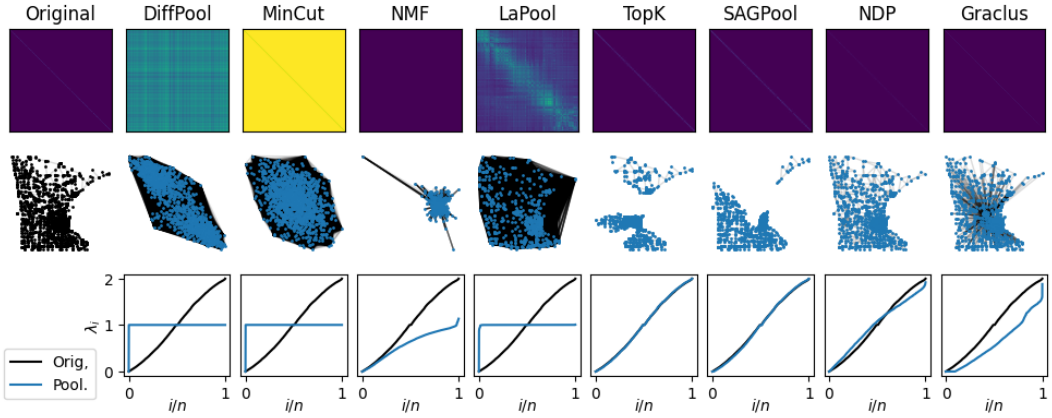
(b) Ring



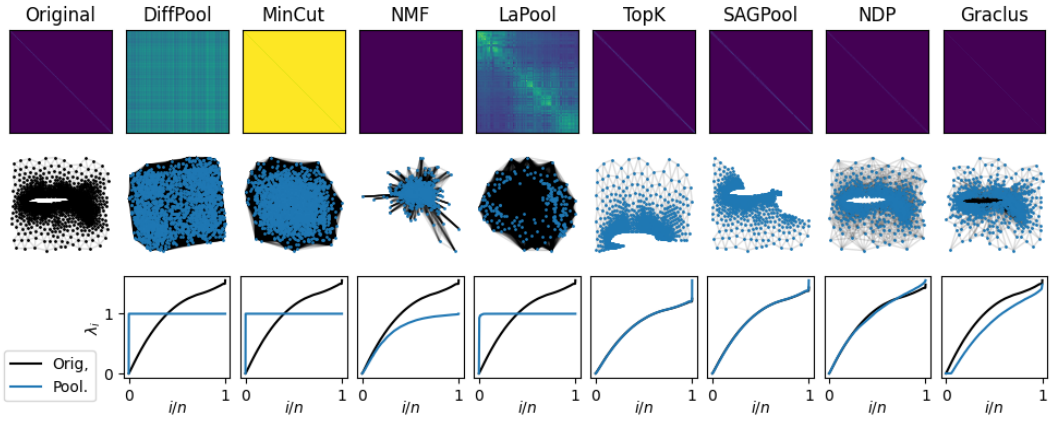
(c) Sensor



(d) Bunny



(e) Minnesota



(f) Airfoil

Figure C.4. Results for all graphs when optimising for spectral similarity. Top: the coarsened adjacency matrices. Middle: the coarsened graphs with modified SEL function. Bottom: the eigenvalues of the normalised Laplacian before (black) and after (blue) pooling. The indices of the eigenvalues are rescaled to fill $[0, 1]$.

Appendix D

Experimental details for the change detection experiments

The architecture of the AAE is closely inspired to GraphVAE [196], and we conduct a brief hyperparameter search for each experiment, using the validation loss of the network for model selection. As done by Simonovsky and Komodakis [196], the encoder consists of two graph convolutional layers with 32 and 64 channels respectively, with batch normalisation, ReLU, and L_2 regularisation (with a factor of $5 \cdot 10^{-4}$), followed by global attention pooling with 128 channels. When using ECC layers, the kernel-generating network consists of two fully connected ReLU layers of 128 units, with a linear output of $F_l \cdot F_{l-1}$ neurons. The latent representation is produced by a ReLU layer with 128 units followed by a linear layer with $d+1$ units (these last two layers are replicated in parallel when considering the ensemble of CCMs). The decoder is a fully connected three-layer network of 128, 256, and 512 neurons, with ReLU and batch normalisation, followed by three parallel output layers to reconstruct the graphs: a sigmoid layer for $\mathbf{A}$, and two layers for $\mathbf{X}$ and $\mathbf{E}$, with activations according to their specific domain (*e.g.*, for categorical attributes we could use a softmax activation).

We consider a discriminator network with three hidden ReLU layers of 128 units each. For the prior, we consider the commonly used Gaussian distribution $\mathcal{N}_{\mathcal{M}_{\kappa_i}}(0,1)$, adapted to have support on the CCM (*cf.* Section 5.1.3). When using the geometric discriminator we set $\varsigma = 5$. We train all networks using Adam [102] with a learning rate of 0.001 and a batch size of 128. We train to convergence, monitoring the validation loss with a patience of 20 epochs. We set aside 10% of the samples for testing, and 10% for validation and model selection. For each graph, $\mathbf{X}$ and $\mathbf{E}$ are normalised element-wise, by removing the mean and scaling to unit variance. For the CDTs we set $\alpha = 0.01$ and q to the

0.75 quantile of the χ^2 distribution of s_w. The size of the windows processed by CUSUM is set to 0.1% of the number of training samples, which we found to be enough to estimate the mean and variance of s_w in Equation 5.17.

Appendix E

Additional experiments and details on seizure localisation

E.1 Seizure generator from Benjamin et al. [16]

In this experiment we considered a simple network model of seizure initiation presented by Benjamin et al. [16], and also used by Lopes et al. [126, 127] to study the effect of network structure on the generation of seizures. The model consists of a network of N bi-stable oscillators

$$\dot{z} = f(z) = (\lambda - 1 + i\omega)z + 2z|z|^2 - z|z|^4 \tag{E.1}$$

where $z \in \mathbb{C}$. Equation (E.1) describes a dynamical system with a stable fixed point at the origin of the complex plane (which we consider as interictal), and an oscillating attractor with frequency ω (which we consider as ictal). Parameter λ controls the location of the oscillator in phase space. Nodes are interconnected in a graph described by adjacency matrix $\mathbf{A}$ with a coupling factor β, such that the dynamic of a single node reads:

$$dz_i(t) = \Big(f(z_i) + \beta \sum_{j \neq i} \mathbf{A}_{ji}(z_j - z_i)\Big) + \alpha\, dW_i(t) \tag{E.2}$$

where $W_i(t)$ is a stochastic Wiener process rescaled by a factor of α.

All nodes in the model are initialised at the fixed point and, due to the presence of noise and the interaction between nodes, eventually switch to the oscillation state. We identify the activity of the whole system as ictal if any of the nodes meets the condition $|\mathrm{Re}(z_i)| > 1$, and the SOZ as the first node that escapes the fixed regime.

Table E.1. Configuration used for the simulator by Benjamin et al. [16].

Param.	Value
N	3
ω	20
λ	0.5
β	0.1
α	0.05

We consider a complete graph without self-loops to describe the interaction of the nodes. The configuration of the parameters is summarised in Table E.1. The hyperparameters used for creating the FNs and training the GNN are the same ones that we used for the real iEEG data, and we only report results obtained using PLV as FC metric.

The GNN achieves an almost perfect detection score with a ROC-AUC of 99.61 ± 0.0 and a PR-AUC of 99.69 ± 0.0 (averaged over five runs, evaluated on hold-out test data). Figure E.1 compares the generated node activity with the attention scores assigned by the GNN over time. The SOZ channel (green) is assigned the highest attention since the beginning of the seizure until all nodes are simultaneously oscillating, at which point the attention scores converge to be evenly distributed. A similar even distribution is observed in the interictal state, indicating that the network has correctly learned to identify the SOZ electrode without defaulting to assign a high score to just one electrode. This behaviour is confirmed by the spikes in attention assigned to channels 0 and 1, which happen as soon as the node dynamics escape the fixed-point attractor.

E.2 The Virtual Brain simulator

In this experiment we use The Virtual Brain simulator (TVB) [179] to model a patient with temporal lobe epilepsy.

We follow the same approach described in TVB's documentation to configure the simulator.[1] We assign the Epileptor neural mass model [95] to all the controllable brain regions of TVB. We set the epileptogenicity of the right limbic areas (rHC, rPHC and rAMYG) to -1.6, the superior temporal cortex (rTCI) and the ventral temporal cortex (rTCV) to -1.8, while for all other areas to -2.2. The remaining parameters are kept as default. The hyperparameters used for

[1] https://github.com/the-virtual-brain/tvb-root/blob/master/tvb_documentation

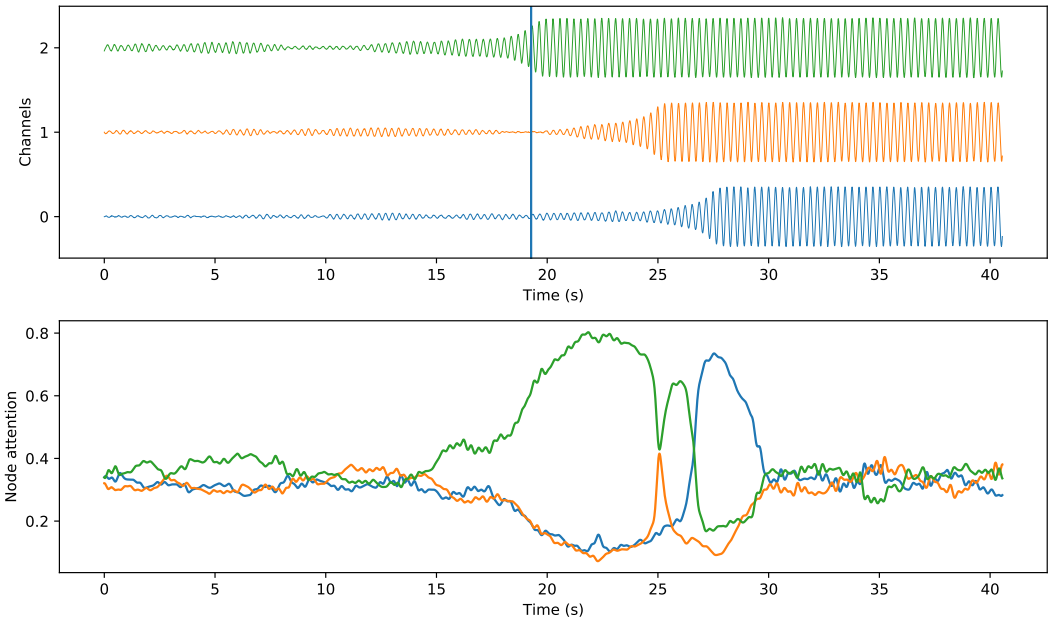


Figure E.1. Top: a clip showing the generated activity of a 3-node simulator, compared to the attention coefficient assigned by the GNN at each node over time. Colors indicate the same node in both plots.

creating the FNs and training the GNN are the same ones that we used for the real iEEG data.

We select a subset of 34 sEEG virtual sensors among the ones provided for the default subject of TVB. Of this subset, electrode 33 shows strong epileptogenic activity, while electrodes 18, 19, and 20 show mild activity. We generate clips of roughly 1 minute at 20Hz so that there is a simulated onset in the middle of each clip. An example of a generated clip is shown in Figure E.2.

The GNN achieved an average detection ROC-AUC of 98.87 ± 0.18 and an average PR-AUC of 99.18 ± 0.07 (averaged over five runs, evaluated on hold-out test data). The electrode with a strong ictal activity is consistently assigned a maximum score of 1 by all models and electrode 19 is also ranked in the top-5 electrodes (see Figure E.3).

E.3 GNN training details

We consider each patient separately and train a GNN from scratch to build patient-specific models. The GNN architecture is the one given in Equation (6.9). The

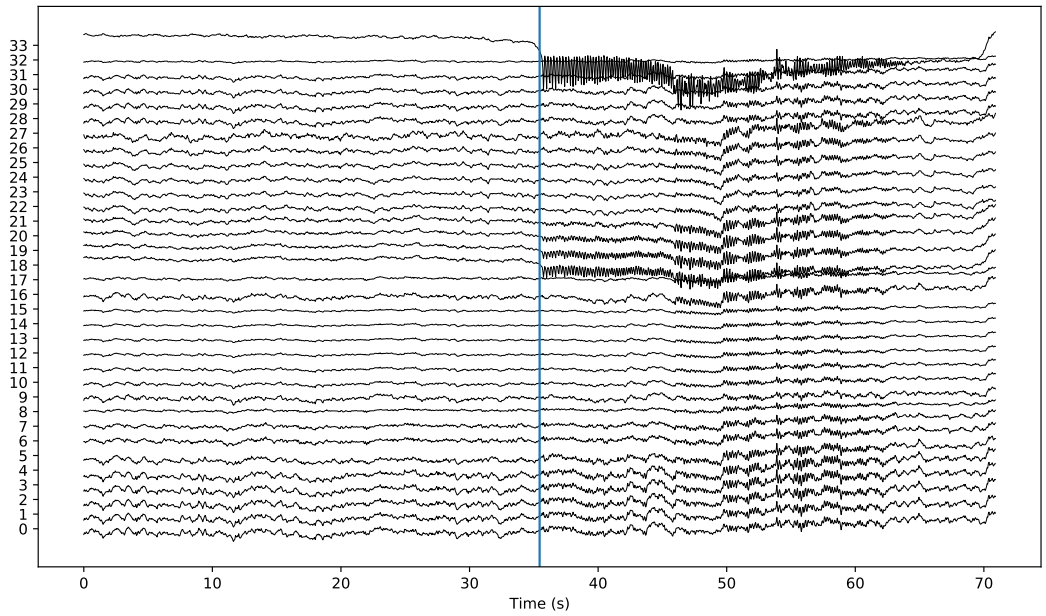


Figure E.2. A virtual seizure generated with TVB. The vertical line indicates the annotated seizure onset in time.

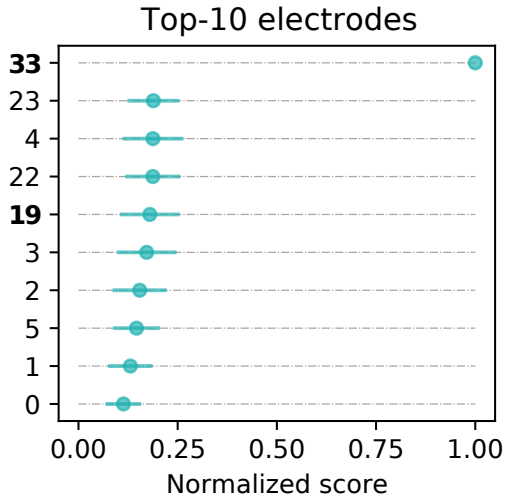


Figure E.3. Top-10 electrodes with averaged rankings. Bold labels indicate that the corresponding electrode showed ictal activity. As desired, electrode 33 shows strong epileptogenic activity.

ECC layer has 32 output units with ReLU activation and a kernel-generating network $f(\cdot)$ consisting of a two-layer MLP with 32 hidden units and ReLU activation. All parameters of the layer are regularised with an L_2 penalty with a factor of 10^{-5}.

The MLP classifier following the ATTN-RO readout has 2 layers, with the hidden one having 32 units and ReLU activation and with 25% dropout in-between. Both layers are regularised with an L_2 penalty with factor 10^{-5}.

The model is trained using Adam, with a learning rate of 10^{-3} and a batch size of 32 graphs. The model is trained to convergence with 10 epochs of patience, using the data from $\lceil 0.1 \cdot n \rceil$ seizures selected randomly (n being the overall number of seizures) for early stopping. We then test the model on a held-out set of $\lceil 0.1 \cdot n \rceil$ seizures. The remaining seizures are used for training. All experiments are repeated 5 times using different random data splits.

E.4 Baseline training details

The baseline is a simple 1D convolutional neural network (CNN) based on the architecture described by Wang et al. [216]. The CNN operates directly on iEEG time series and therefore does not take into account any graph-based representation for the data. Similarly to how we create the input-output pairs for the GNN, here we consider windows of size T taken at a stride of k/f_s for the interictal class and stride $1/f_s$ for the ictal class, and we associate to each window a class label corresponding to the majority class of $y(t)$ in the corresponding window.

In particular, we shrink the model to make it comparable in terms of number of parameters and depth to the GNN one, and also to prevent overfitting (which we experimentally encountered as a significant problem with the model). We consider a single convolutional layer with a kernel of size 3, 8 output channels, and ReLU activations, followed by a global average pooling and a single-layer MLP to output the classification decision. We train the model using Adam with learning rate 0.001, batch size of 32 and early stopping with a patience of 5 epochs.

E.5 Additional results

Detection A notable behaviour of the GNN can be observed from Figure E.4, which shows the output of the GNN (*i.e.*, the detection score outputted by the model) on a symmetrical window around the onset, for randomly sampled seizures

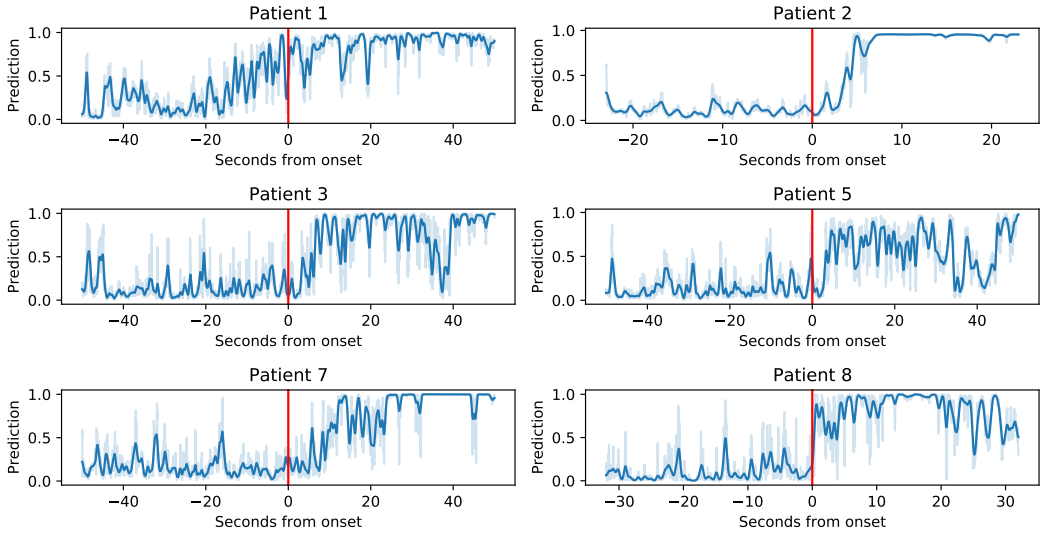


Figure E.4. Example of the detection score outputted by the GNN, for all patients with a known SOZ. We show a window of 50 seconds around the marked onset for random test seizures.

of the six patients with a known SOZ. We empirically observed that the model is robust to the onset labelling provided by electroencephalographers. Notably, by analysing the prediction of the GNN in the time instants prior to the seizure onset, we can see that the confidence with which the GNN detects a seizure starts to gradually increase towards the seizure onset, but does not always peak at the onset time marked by electroencephalographers. This suggests that the GNN is learning to detect the anomalous brain activity rather than overfitting to the known onset labels.

Localisation We show in Figures E.6 and E.7 the top 10 electrodes by AP@10 score for all patients, respectively when using correlation and PLV as FC metrics.

Threshold To demonstrate that our approach is robust to the choice of sparsification threshold for the FNs, as argued in Section 6.2.1, we report in Figure E.5 the average performance for detection and localisation over all patients and all metrics for different thresholds (that is, we average all the values reported in tables 6.2 and 6.3 after having re-computed the tables with different sparsification thresholds). While this is a coarse-grained analysis, it shows that there are no significant differences in the downstream performance for reasonable sparsification thresholds (*i.e.*, in the $[0.0, 0.5]$ interval). All results in Figure E.5 are not

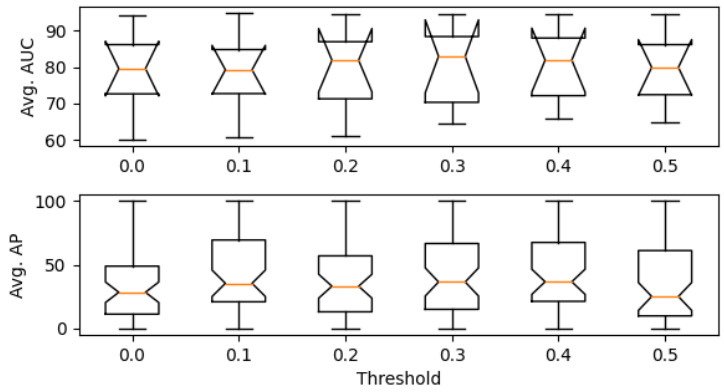
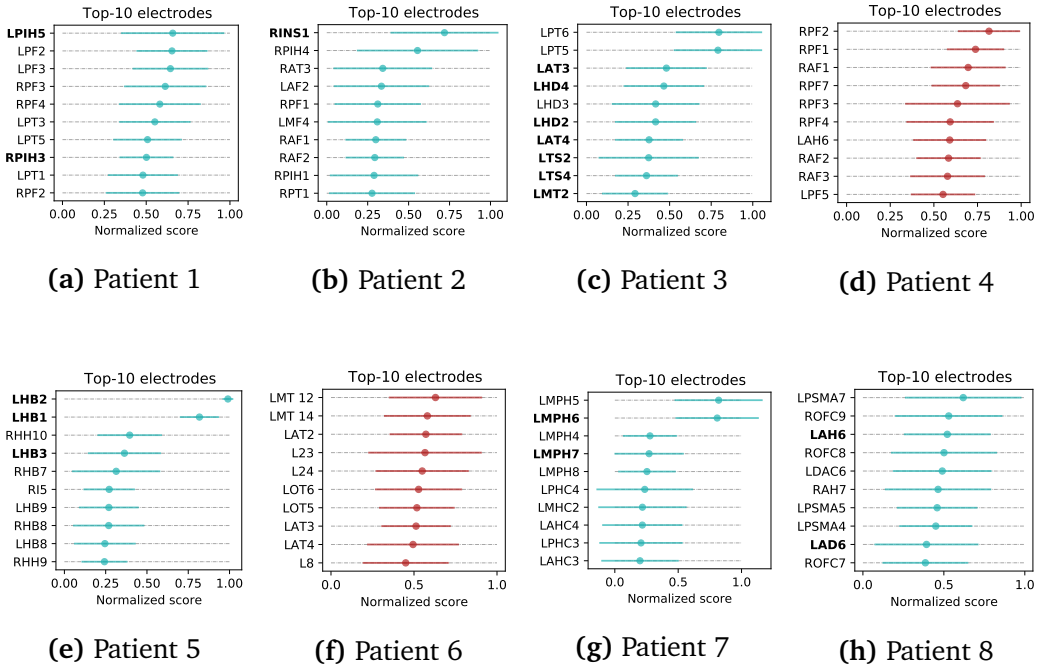


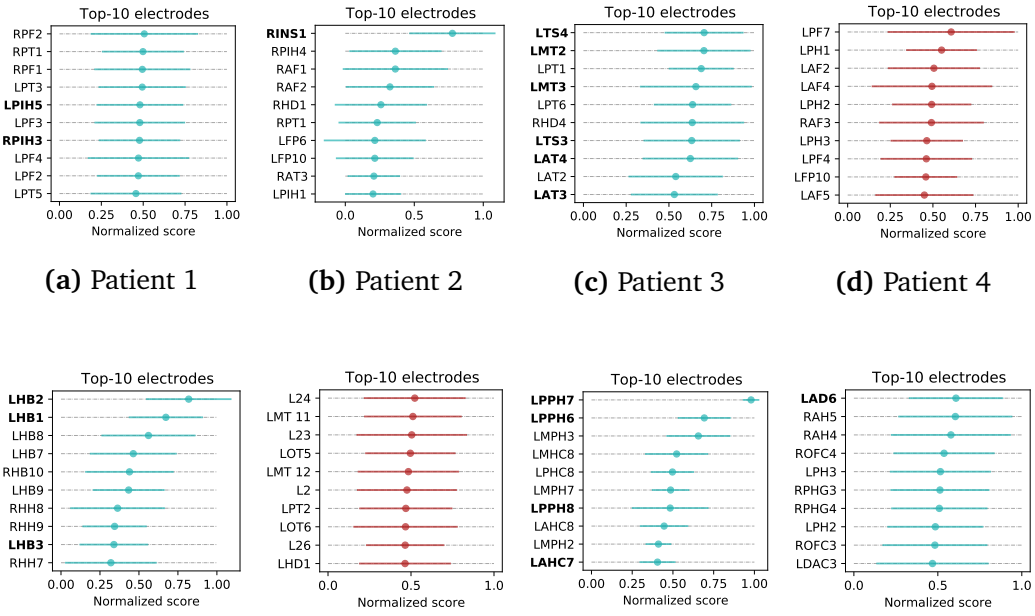
Figure E.5. Average detection and localisation performance as a function of the sparsification threshold. We report the average over all metrics and all patients, as reported in Tables II and III of the manuscript.



(a) Patient 1 **(b)** Patient 2 **(c)** Patient 3 **(d)** Patient 4

(e) Patient 5 **(f)** Patient 6 **(g)** Patient 7 **(h)** Patient 8

Figure E.6. Top ten electrodes by AP@10 score for the average rankings, using correlation as FC measure. The two plots in red indicate those patients for which the SOZ was not identified clinically. Bold labels indicate that the corresponding electrode was marked as a potential SOZ by electroencephalographers.



(a) Patient 1 **(b)** Patient 2 **(c)** Patient 3 **(d)** Patient 4

(e) Patient 5 **(f)** Patient 6 **(g)** Patient 7 **(h)** Patient 8

Figure E.7. Top ten electrodes by AP@10 score for the average rankings, using PLV as FC measure. The two plots in red indicate those patients for which the SOZ was not identified clinically. Bold labels indicate that the corresponding electrode was marked as a potential SOZ by electroencephalographers.

significantly different, with p-value of a two-sided t-test $p \gg 0.05$ for all pairs on both tasks (detection and localisation).

Appendix F

Additional details on autoregressive models

F.1 Equivalence between (7.1) and (7.12).

Let $(\mathfrak{G}, \mathrm{d})$ be the Euclidean space $(\mathbb{R}, \|\cdot\|)$, $H(a,b) = a + b$, and

$$G_{t,p} = [\mathcal{G}_t, \ldots, \mathcal{G}_{t-p+1}]. \tag{F.1}$$

Then

$$\mathcal{G}_{t+1} = H(\phi(G_{t,p}), \eta) = \phi(G_{t,p}) + \eta. \tag{F.2}$$

Regarding assumption (7.8), we see that the integral in Equation (7.9) (known as Fréchet function of Q) becomes

$$\int_{\mathfrak{G}} \mathrm{d}(\mathcal{G}, \mathcal{G}')^2 dQ(\mathcal{G}) = \int_{\mathbb{R}} \left\| \mathcal{G} - \mathcal{G}' \right\|^2 dQ(\mathcal{G}) \tag{F.3}$$

and

$$\int_{\mathfrak{G}} \mathrm{d}(H(\phi(G_{t,p}), \eta), \mathcal{G}')^2 \, dQ(\eta) = \int_{\mathbf{R}} \left\| \phi(G_{t,p}) + \eta - \mathcal{G}' \right\|^2 dQ(\eta) = \tag{F.4}$$

$$= \int_{\mathbf{R}} \left\{ \left\| \phi(G_{t,p}) - \mathcal{G}' \right\|^2 + \|\eta\|^2 - 2\eta(\phi(G_{t,p}) - \mathcal{G}') \right\} dQ(\eta) = \tag{F.5}$$

$$= \left\| \phi(G_{t,p}) - \mathcal{G}' \right\|^2 + \mathrm{Var}[\eta] - 2\mathbb{E}[\eta](\phi(G_{t,p}) - \mathcal{G}'). \tag{F.6}$$

We conclude that (7.1) and (7.12) are equivalent since

$$\phi(G_{t,p}) \in \mathbb{E}^{\mathrm{f}}[H(\phi(G_{t,p}), \eta)] \iff \mathbb{E}[\eta] = 0. \tag{F.7}$$

and

$$\mathrm{Var}[\eta] < \infty \iff \mathrm{Var}^{\mathrm{f}}[\eta] < \infty. \tag{F.8}$$

Appendix G

Spektral

In this Appendix, we report our efforts in the development of open-source software related to this thesis and GML in general.

One of the major challenges faced by researchers and software developers who wish to contribute to the larger scientific community is to make software both accessible and intuitive, so that even non-technical audiences can benefit from the advances carried by intelligent systems. In this spirit, Keras is an application programming interface (API) for creating neural networks, developed according to the guiding principle that "being able to go from idea to result with the least possible delay is key to doing good research" [36]. Keras is designed to reduce the cognitive load of end-users, shifting the focus away from the boilerplate implementation details and allowing instead to focus on the creation of models. As such, Keras is extremely beginner-friendly and, for many, an entry point to machine learning itself. At the same time, Keras integrates smoothly with its TensorFlow [1] backend and enables users to build any model that they could have implemented in pure TensorFlow. This flexibility makes Keras an excellent tool even for expert deep learning practitioners and has recently led to TensorFlow's adoption of Keras as the official interface to the framework.

Figure G.1. The logo of Spektral.

Motivated by the increasing interest in GNNs from the scientific community, and the consequent need for an easy-to-use and complete software library for GNNs, a core contribution of the doctorate was the development of Spektral, a Python library for building GNNs using TensorFlow and the Keras API. Spektral implements some of the most important papers from the GNN literature as Keras layers, and it integrates seamlessly within Keras models and with the most important features of Keras like the training loop, callbacks, distributed training, and automatic support for GPUs and TPUs. As such, Spektral inherits the philosophy of ease of use and flexibility that characterises Keras. The components of Spektral act as standard TensorFlow operations and can be easily used even in more advanced settings, integrating tightly with all the features of TensorFlow and allowing for easy deployment to production systems. For these reasons, Spektral is the ideal library to implement GNNs in the TensorFlow ecosystem, both for total beginners and experts alike. The documentation and source code of Spektral is available at the following URL: `https://graphneural.network`.

We have also presented the main features of Spektral in reference [74].

Appendix H

Hardware and software

All the experimental results reported in this thesis were obtained on a machine configured with the following hardware:

- 2 Intel Xeon Silver 4116 CPUs (48 threads in total);
- 126 GB of RAM;
- 1 NVIDIA Titan XP GPU;
- 2 Nvidia Titan V GPUs;
- 1 Nvidia GeForce RTX 2080 GPU.

The operating system for the machine was Ubuntu 16.04.1 LTS. All models were implemented in Python 3, and all machine learning code was based on TensorFlow 2 [1] and Spektral [74]. We have published the code for all experiments discussed in this thesis on `github.com` (see the relevant references for the exact URLs).

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Zeina Abu-Aisheh, Romain Raveaux, Jean-Yves Ramel, and Patrick Martineau. An exact graph edit distance algorithm for solving pattern recognition problems. In *4th International Conference on Pattern Recognition Applications and Methods 2015*, 2015.

[3] Leman Akoglu and Christos Faloutsos. Anomaly, event, and fraud detection in large network datasets. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 773–774. ACM, 2013.

[4] Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery*, 29(3):626–688, 2015.

[5] Cesare Alippi and Manuel Roveri. The (not) far-away path to smart cyber-physical systems: An information-centric framework. *Computer*, 50(4): 38–47, 2017.

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

[7] Davide Bacciu and Luigi Di Sotto. A non-negative factorization approach to node pooling in graph convolutional neural networks. In *Proceedings of the 18th International Conference of the Italian Association for Artificial Intelligence*. AIIA, 2019.

[8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1409.0473`.

[9] Yunsheng Bai, Hao Ding, Yang Qiao, Agustin Marinovic, Ken Gu, Ting Chen, Yizhou Sun, and Wei Wang. Unsupervised inductive graph-level representation learning via graph-graph proximity. *arXiv preprint arXiv:1904.01098*, 2019.

[10] Ivana Balazevic, Carl Allen, and Timothy Hospedales. Multi-relational poincaré graph embeddings. *Advances in Neural Information Processing Systems*, 32:4463–4473, 2019.

[11] Ian Barnett and Jukka-Pekka Onnela. Change point detection in correlation networks. *Scientific reports*, 6:18893, 2016.

[12] A M Bastos and J-M Schoffelen. A tutorial review of functional connectivity analysis methods and their interpretational pitfalls. *Frontiers in Systems Neuroscience*, 9:175, 2016.

[13] Joshua Batson, Daniel A Spielman, Nikhil Srivastava, and Shang-Hua Teng. Spectral sparsification of graphs: theory and algorithms. *Communications of the ACM*, 56(8):87–94, 2013.

[14] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[15] Lowell W Beineke, Robin J Wilson, and Peter J Cameron. *Topics in algebraic graph theory*, volume 102. Cambridge University Press, 2004.

[16] Oscar Benjamin, Thomas HB Fitzgerald, Peter Ashwin, Krasimira Tsaneva-Atanasova, Fahmida Chowdhury, Mark P Richardson, and John R Terry. A phenomenological model of seizure initiation suggests network structure may explain seizure frequency in idiopathic generalised epilepsy. *The Journal of Mathematical Neuroscience*, 2(1):1–30, 2012.

[17] Rianne van den Berg, Thomas N Kipf, and Max Welling. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.

[18] Rabi Bhattacharya and Lizhen Lin. Differential geometry for model independent analysis of images and other non-euclidean data: Recent developments. *arXiv preprint arXiv:1801.00898*, 2018.

[19] Filippo Maria Bianchi, Enrico Maiorino, Lorenzo Livi, Antonello Rizzi, and Alireza Sadeghian. An agent-based algorithm exploiting multiple local dissimilarities for clusters mining and knowledge discovery. *Soft Computing*, 21(5):1347–1369, 2017.

[20] Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral clustering with graph neural networks for graph pooling. In *International Conference on Machine Learning (ICML)*, 2020.

[21] Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Hierarchical representation learning in graph neural networks with node decimation pooling. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[22] Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Graph neural networks with convolutional arma filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[23] Cristian Bodnar, Catalina Cangea, and Pietro Liò. Deep graph mapper: Seeing graphs through the neural lens. *arXiv preprint arXiv:2002.03864*, 2020.

[24] C Bonferroni. Teoria statistica delle classi e calcolo delle probabilita. *Pubblicazioni del R. Istituto Superiore di Scienze Economiche e Commericiali di Firenze*, 8:3–62, 1936.

[25] Sebastien Bougleux, Luc Brun, Vincenzo Carletti, Pasquale Foggia, Benoit Gaüzère, and Mario Vento. Graph edit distance as a quadratic assignment problem. *Pattern Recognition Letters*, 87:38–46, 2017.

[26] Mary AB Brazier. Spread of seizure discharges in epilepsy: anatomical and electrophysiological considerations. *Experimental Neurology*, 36(2): 263–272, 1972.

[27] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[28] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[29] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[30] S P Burns, Sabato Santaniello, R B Yaffe, C C Jouny, N E Crone, G K Bergey, W S Anderson, and S V Sarma. Network dynamics of the brain and influence of the epileptic seizure onset zone. *Proceedings of the National Academy of Sciences*, 111(49):321–330, 2014. doi: 10.1073/pnas.1401752111.

[31] Chen Cai, Dingkang Wang, and Yusu Wang. Graph coarsening with neural networks. *arXiv preprint arXiv:2102.01350*, 2021.

[32] Catalina Cangea, Petar Veličković, Nikola Jovanović, Thomas Kipf, and Pietro Liò. Towards sparse hierarchical graph classifiers. *arXiv preprint arXiv:1811.01287*, 2018.

[33] Quentin Cappart, Didier Chételat, Elias Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. *arXiv preprint arXiv:2102.09544*, 2021.

[34] Nutan Chen, Alexej Klushyn, Richard Kurle, Xueyan Jiang, Justin Bayer, and Patrick van der Smagt. Metrics for deep generative models. In *AISTATS*, 2018.

[35] Minsu Cho, Jian Sun, Olivier Duchenne, and Jean Ponce. Finding matches in a haystack: A max-pooling strategy for graph matching in the presence of outliers. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2083–2090, 2014.

[36] Francois Chollet et al. Keras. `https://keras.io`, 2015.

[37] Adam Coates and Andrew Y Ng. Selecting receptive fields in deep networks. In *Advances in neural information processing systems*, pages 2528–2536, 2011.

[38] Maxwell D Collins, Ji Liu, Jia Xu, Lopamudra Mukherjee, and Vikas Singh. Spectral clustering with a convex regularizer on millions of images. In *European Conference on Computer Vision*, pages 282–298. Springer, 2014.

[39] Padraig Corcoran. Function space pooling for graph convolutional networks. *arXiv preprint arXiv:1905.06259*, 2019.

[40] Ian Covert, Balu Krishnan, Imad Najm, Jiening Zhan, Matthew Shore, John Hixson, and M J Po. Temporal Graph Convolutional Networks for Automatic Seizure Detection. *arXiv preprint arXiv:1905.01375*, 2019.

[41] Miles Cranmer, Alvaro Sanchez-Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho. Discovering symbolic models from deep learning with inductive biases. *arXiv preprint arXiv:2006.11287*, 2020.

[42] Anil Damle, Victor Minden, and Lexing Ying. Robust and efficient multiway spectral clustering. *arXiv preprint arXiv:1609.08251*, 2016.

[43] Tim R. Davidson, Luca Falorsi, Nicola De Cao, Thomas Kipf, and Jakub M. Tomczak. Hyperspherical variational auto-encoders. *34th Conference on Uncertainty in Artificial Intelligence (UAI-18)*, 2018.

[44] Philip I Davies and Nicholas J Higham. Numerically stable generation of correlation matrices and their factors. *BIT Numerical Mathematics*, 40(4): 640–651, 2000.

[45] Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018.

[46] Michaël Defferrard, Lionel Martin, Rodrigo Pena, and Nathanaël Perraudin. Pygsp: Graph signal processing in python. URL `https://github.com/epfl-lts2/pygsp/`.

[47] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

[48] Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 551–556. ACM, 2004.

[49] Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE transactions on pattern analysis and machine intelligence*, 29(11):1944–1957, 2007.

[50] Frederik Diehl. Edge contraction pooling for graph neural networks. *CoRR*, abs/1905.10990, 2019. URL `http://arxiv.org/abs/1905.10990`.

[51] Frederik Diehl, Thomas Brunner, Michael Truong Le, and Alois Knoll. Towards graph pooling by edge contraction. *ICML Workshop on Learning and Reasoning with Graph-Structured Representations*, 2019.

[52] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar. Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, 10(4): 12–25, Nov. 2015. ISSN 1556-603X. doi: 10.1109/MCI.2015.2471196.

[53] F. Dorfler and F. Bullo. Kron reduction of graphs with applications to electrical networks. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 60(1):150–163, Jan 2013. ISSN 1549-8328. doi: 10.1109/TCSI.2012.2215780.

[54] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

[55] Stefan Fankhauser, Kaspar Riesen, and Horst Bunke. Speeding up graph edit distance computation through fast bipartite matching. In *International Workshop on Graph-Based Representations in Pattern Recognition*, pages 102–111. Springer, 2011.

[56] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International journal of computer vision*, 59(2):167–181, 2004.

[57] Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 869–877, 2018.

[58] Alex M Fout. *Protein interface prediction using graph convolutional networks*. PhD thesis, Colorado State University, 2017.

[59] Maurice Fréchet. Les éléments aléatoires de nature quelconque dans un espace distancié. In *Annales de l'Institut Henri Poincaré*, volume 10, pages 215–310, 1948.

[60] Victor Fung, Jiaxin Zhang, Eric Juarez, and Bobby G Sumpter. Benchmarking graph neural networks for materials chemistry. *npj Computational Materials*, 7(1):1–8, 2021.

[61] Soham Gadgil, Qingyu Zhao, Ehsan Adeli, Adolf Pfefferbaum, Edith V Sullivan, and Kilian M Pohl. Spatio-temporal graph convolution for functional mri analysis. *arXiv preprint arXiv:2003.10613*, 2020.

[62] P Gainza, F Sverrisson, F Monti, E Rodolà, D Boscaini, MM Bronstein, and BE Correia. Deciphering interaction fingerprints from protein molecular surfaces using geometric deep learning. *Nature Methods*, 17(2):184–192, 2020.

[63] Claudio Gallicchio and Alessio Micheli. Fast and deep graph neural networks. In *AAAI*, pages 3898–3905, 2020.

[64] Will Gersch and GV Goddard. Epileptic focus location: spectral analysis method. *Science*, 169(3946):701–702, 1970.

[65] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017.

[66] R. Giryes, G. Sapiro, and A. M. Bronstein. Deep neural networks with random Gaussian weights: A universal classification strategy? *IEEE Transactions on Signal Processing*, 64(13):3444–3457, Jul. 2016. ISSN 1053-587X. doi: 10.1109/TSP.2016.2546221.

[67] Vladimir Gligorijević, P Douglas Renfrew, Tomasz Kosciolek, Julia Koehler Leman, Daniel Berenberg, Tommi Vatanen, Chris Chandler, Bryn C Taylor,

Ian M Fisk, Hera Vlamakis, et al. Structure-based protein function prediction using graph convolutional networks. *Nature communications*, 12(1): 1–14, 2021.

[68] K Goebel and WA Kirk. A fixed point theorem for asymptotically nonexpansive mappings. *Proceedings of the American Mathematical Society*, 35 (1):171–174, 1972.

[69] Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

[70] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

[71] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[72] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE, 2005.

[73] Leo J Grady and Jonathan R Polimeni. *Discrete calculus: Applied analysis on graphs for computational science*. Springer Science & Business Media, 2010.

[74] Daniele Grattarola and Cesare Alippi. Graph neural networks in tensorflow and keras with spektral. *IEEE Computational Intelligence Magazine*, 2021.

[75] Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Adversarial autoencoders with constant-curvature latent manifolds. *Applied Soft Computing*, 81:105511, 2019.

[76] Daniele Grattarola, Daniele Zambon, Cesare Alippi, and Lorenzo Livi. Change detection in graph streams by learning graph embeddings on

constant-curvature manifolds. *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

[77] Daniele Grattarola, Lorenzo Livi, Cesare Alippi, Richard Wennberg, and Taufik Valiante. Unsupervised seizure localisation with attention-based graph neural networks. *bioRxiv*, 2020.

[78] Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Learning graph cellular automata. In *Neural Information Processing Systems*, 2021.

[79] Daniele Grattarola, Daniele Zambon, Filippo Maria Bianchi, and Cesare Alippi. Understanding pooling in graph neural networks. *arXiv preprint arXiv:2110.05292*, 2021.

[80] Caglar Gulcehre, Misha Denil, Mateusz Malinowski, Ali Razavi, Razvan Pascanu, Karl Moritz Hermann, Peter Battaglia, Victor Bapst, David Raposo, Adam Santoro, et al. Hyperbolic attention networks. *arXiv preprint arXiv:1805.09786*, 2018.

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

[82] Yufei Han and Maurizio Filippone. Mini-batch spectral clustering. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 3888–3895. IEEE, 2017.

[83] Kimiaki Hashiguchi, Takato Morioka, Fumiaki Yoshida, Yasushi Miyagi, Shinji Nagata, Ayumi Sakata, and Tomio Sasaki. Correlation between scalp-recorded electroencephalographic and electrocorticographic activities during ictal period. *Seizure*, 16(3):238–247, 2007.

[84] Stewart Heitmann and Michael Breakspear. Putting the "dynamic" back into dynamic functional connectivity. *Network Neuroscience*, pages 1–61, 2017. doi: 10.1162/NETN_a_00041.

[85] Gecia Bravo Hermsdorff and Lee M Gunderson. A unifying framework for spectrum-preserving graph sparsification and coarsening. *arXiv preprint arXiv:1902.09702*, 2019.

[86] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[87] Petter Holme. Modern temporal network theory: a colloquium. *The European Physical Journal B*, 88(9):234, 2015.

[88] Gao Hongyang and Ji Shuiwang. Graph u-net. *Submitted to ICLR*, 2019.

[89] R A Horn and C R Johnson. *Matrix analysis*. Cambridge university press, 2012.

[90] Yasuhiko Ikebe, Toshiyuki Inagaki, and Sadaaki Miyamoto. The monotonicity theorem, cauchy's interlace theorem, and the courant-fischer theorem. *The American Mathematical Monthly*, 94(4):352–354, 1987.

[91] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[92] Elvin Isufi, Andreas Loukas, Andrea Simonetto, and Geert Leus. Autoregressive moving average graph filtering. *arXiv preprint arXiv:1602.04436*, 2016.

[93] B J Jain. On the geometry of graph spaces. *Discrete Applied Mathematics*, 214:126–144, 2016. doi: 10.1016/j.dam.2016.06.027.

[94] B J Jain. Statistical graph space analysis. *Pattern Recognition*, 60:802–812, 2016. doi: 10.1016/j.patcog.2016.06.023.

[95] Viktor K Jirsa, William C Stacey, Pascale P Quilichini, Anton I Ivanov, and Christophe Bernard. On the nature of seizure dynamics. *Brain*, 137(8): 2210–2230, 2014.

[96] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[97] George Karypis. Metis: Unstructured graph partitioning and sparse matrix ordering system. *Technical report*, 1997.

[98] Nicolas Keriven and Gabriel Peyré. Universal invariant and equivariant graph neural networks. In *Advances in Neural Information Processing Systems 32*, pages 7090–7099. Curran Associates, Inc., 2019. URL `http://papers.nips.cc/paper/8931-universal-invariant-and-equivariant-graph-neural-networks.pdf`.

[99] A N Khambhati, K A Davis, B S Oommen, S H Chen, T H Lucas, Brian Litt, and D S Bassett. Dynamic network drivers of seizure generation, propagation and termination in human neocortical epilepsy. *PLoS Computational biology*, 11(12):e1004608, 2015. doi: 10.1371/journal.pcbi.1004608.

[100] A N Khambhati, K A Davis, T H Lucas, Brian Litt, and D S Bassett. Virtual cortical resection reveals push-pull network control preceding seizure evolution. *Neuron*, 91(5):1170–1182, 2016. doi: 10.1016/j.neuron.2016.07.039.

[101] Daesik Kim, YoungJoon Yoo, Jee-Soo Kim, Sangkuk Lee, and Nojun Kwak. Dynamic graph generation network: Generating relational knowledge from diagrams. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4167–4175, 2018.

[102] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[103] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[104] Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pages 3581–3589, 2014.

[105] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. *arXiv preprint arXiv:1802.04687*, 2018.

[106] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[107] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[108] Boris Knyazev, Graham W. Taylor, and Mohamed R. Amer. Understanding attention in graph neural networks. *CoRR*, abs/1905.02850, 2019. URL `http://arxiv.org/abs/1905.02850`.

[109] E O Korman. Autoencoding topology. *arXiv preprint arXiv:1803.00156*, 2018.

[110] Mark A Kramer, Uri T Eden, Sydney S Cash, and Eric D Kolaczyk. Network inference with confidence from multivariate time series. *Physical Review E*, 79(6):061916, 2009.

[111] Matt J Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar variational autoencoder. In *International Conference on Machine Learning*, pages 1945–1954, 2017.

[112] Patrick Kwan and M J Brodie. Early identification of refractory epilepsy. *New England Journal of Medicine*, 342(5):314–319, 2000.

[113] Jean-Philippe Lachaux, Eugenio Rodriguez, Jacques Martinerie, and Francisco J Varela. Measuring phase synchrony in brain signals. *Human Brain Mapping*, 8(4):194–208, 1999.

[114] Marc T Law, Raquel Urtasun, and Richard S Zemel. Deep spectral clustering learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1985–1994. JMLR. org, 2017.

[115] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[116] H W Lee, Jagriti Arora, Xenophon Papademetris, Fuyuze Tokoglu, Michiro Negishi, Dustin Scheinost, Pue Farooque, Hal Blumenfeld, D D Spencer, and R T Constable. Altered functional connectivity in seizure onset zones revealed by fmri intrinsic connectivity. *Neurology*, 83(24):2269–2277, 2014.

[117] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. *arXiv preprint arXiv:1904.08082*, 2019.

[118] Huan Lei, Naveed Akhtar, and Ajmal Mian. Octree guided cnn with spherical kernels for 3d point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9631–9640, 2019.

[119] Ron Levie, Federico Monti, Xavier Bresson, and Michael M Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *arXiv preprint arXiv:1705.07664*, 2017.

[120] Ron Levie, Isufi Elvin, and Kutyniok Gitta. On the transferability of spectral graph filters. *arXiv preprint*, 2019.

[121] Aming Li, S P Cornelius, Y-Y Liu, Long Wang, and A-L Barabási. The fundamental advantages of temporal networks. *Science*, 358(6366):1042–1046, 2017. doi: 10.1126/science.aai7488.

[122] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. *arXiv preprint arXiv:1801.07606*, 2018.

[123] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *International Conference on Learning Representations*, 2018.

[124] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[125] L. Livi and A. Rizzi. The graph matching problem. *Pattern Analysis and Applications*, 16(3):253–283, 2013. ISSN 1433-7541. doi: 10.1007/s10044-012-0284-8.

[126] M A Lopes, M P Richardson, Eugenio Abela, Christian Rummel, Kaspar Schindler, Marc Goodfellow, and J R Terry. An optimal strategy for epilepsy surgery: Disruption of the rich-club? *PLoS Computational Biology*, 13(8): e1005637, 2017.

[127] M A Lopes, Leandro Junges, Wessel Woldman, Marc Goodfellow, and J R Terry. The role of excitability and network structure in the emergence of focal and generalized seizures. *Frontiers in Neurology*, 11:74, 2020.

[128] Andreas Loukas. Graph reduction with spectral and cut guarantees. *Journal of Machine Learning Research*, 20(116):1–42, 2019.

[129] Andreas Loukas, Andrea Simonetto, and Geert Leus. Distributed autoregressive moving average graph filters. *IEEE Signal Processing Letters*, 22 (11):1931–1935, 2015.

[130] Mantas Lukoševičius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009. doi: 10.1016/j.cosrev.2009.03.005.

[131] JF Lutzeyer and AT Walden. Comparing graph spectra of adjacency and laplacian matrices. *arXiv preprint arXiv:1712.03769*, 2017.

[132] Enxhell Luzhnica, Ben Day, and Pietro Lio. Clique pooling for graph classification. *International Conference of Learning Representations (ICLR) – Representation Learning on Graphs and Manifolds workshop*, 2019.

[133] Yao Ma, Suhang Wang, Charu C Aggarwal, and Jiliang Tang. Graph convolutional networks with eigenpooling. *arXiv preprint arXiv:1904.13107*, 2019.

[134] Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019.

[135] Jack B Maguire, Daniele Grattarola, Vikram Khipple Mulligan, Eugene Klyshko, and Hans Melo. Xenet: Using a new graph convolution to accelerate the timeline for protein design on quantum computers. *PLoS computational biology*, 17(9):e1009037, 2021.

[136] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian Goodfellow. Adversarial autoencoders. In *International Conference on Learning Representations*, 2016.

[137] Henry B Mann and Donald R Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, pages 50–60, 1947.

[138] Gary Marcus. The algebraic mind, 2001.

[139] Gary Marcus. Deep learning: A critical appraisal. *arXiv preprint arXiv:1801.00631*, 2018.

[140] Haggai Maron, Ethan Fetaya, Nimrod Segol, and Yaron Lipman. On the universality of invariant networks. In *International Conference on Machine Learning*, pages 4363–4371, 2019.

[141] A Marques, A Ribeiro, and S Segarra. Graph signal processing: Fundamentals and applications to diffusion processes. In *Proc. Int. Conf. Accoustic, Speech and Signal Processing,(ICASSP)*, 2017.

[142] N. Masuda and R. Lambiotte. *A Guide to Temporal Networks*. Series on Complexity Science. World Scientific Publishing Company, Singapore, 2016.

[143] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual review of sociology*, 27 (1):415–444, 2001.

[144] Donald Meagher. Geometric modeling using octree encoding. *Computer graphics and image processing*, 19(2):129–147, 1982.

[145] Diego Mesquita, Amauri Souza, and Samuel Kaski. Rethinking pooling in graph neural networks. *Advances in Neural Information Processing Systems*, 33, 2020.

[146] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *ICLR (Workshop)*, 2013.

[147] Gabriele Monfardini, Vincenzo Di Massa, Franco Scarselli, and Marco Gori. Graph neural networks for object localization. *Frontiers in Artificial Intelligence and Applications*, 141:665, 2006.

[148] D C Montgomery. *Introduction to Statistical Quality Control*. John Wiley & Sons, 2007.

[149] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proc. CVPR*, volume 1, page 3, 2017.

[150] Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020.

[151] Nicolò Navarin, Dinh Van Tran, and Alessandro Sperduti. Universal readout for graph convolutional neural networks. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2019.

[152] M E J Newman. *Networks: An Introduction*. Oxford University Press, Oxford, UK, 2010.

[153] Mark EJ Newman. Mixing patterns in networks. *Physical Review E*, 67(2): 026126, 2003.

[154] Maximillian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. In *Advances in Neural Information Processing Systems*, pages 6341–6350, 2017.

[155] Emmanuel Noutahi, Dominique Beani, Julien Horwood, and Prudencio Tossou. Towards interpretable sparse graph representation learning with laplacian pooling. *arXiv preprint arXiv:1905.11577*, 2019.

[156] Hoang Nt and Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019.

[157] Paul L Nunez, Ramesh Srinivasan, et al. *Electric fields of the brain: the neurophysics of EEG*. Oxford University Press, USA, 2006.

[158] Alan V Oppenheim, John R Buck, and Ronald W Schafer. *Discrete-time signal processing. Vol. 2*. Upper Saddle River, NJ: Prentice Hall, 2001.

[159] Antonio Ortega, Pascal Frossard, Jelena Kovacević, José MF Moura, and Pierre Vandergheynst. Graph signal processing: Overview, challenges, and applications. *Proceedings of the IEEE*, 106(5):808–828, 2018.

[160] Edward Ott. *Chaos in Dynamical Systems*. Cambridge University Press, 2002.

[161] Benjamin Paassen, Daniele Grattarola, Daniele Zambon, Cesare Alippi, and Barbara Eva Hammer. Graph edit networks. In *International Conference on Learning Representations*, 2021.

[162] Ewan S Page. Continuous inspection schemes. *Biometrika*, 41(1/2):100–115, 1954.

[163] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[164] Laura Palagi, Veronica Piccialli, Franz Rendl, Giovanni Rinaldi, and Angelika Wiegele. Computational approaches to max-cut. In *Handbook on semidefinite, conic and polynomial optimization*, pages 821–847. Springer, 2012.

[165] Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. Adversarially regularized graph autoencoder for graph embedding. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 2609–2615. International

Joint Conferences on Artificial Intelligence Organization, 7 2018. doi: 10.24963/ijcai.2018/362.

[166] Leto Peel and Aaron Clauset. Detecting change points in the large-scale structure of evolving networks. In *AAAI*, volume 15, pages 1–11, 2015.

[167] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[168] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108, 2017.

[169] Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. On the expressive power of deep neural networks. In *international conference on machine learning*, pages 2847–2854. PMLR, 2017.

[170] Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1:140022, 2014.

[171] Ekagra Ranjan, Soumya Sanyal, and Partha Pratim Talukdar. Asap: Adaptive structure aware pooling for learning hierarchical graph representations. *arXiv preprint arXiv:1911.07979*, 2019.

[172] Stephen Ranshous, Shitian Shen, Danai Koutra, Steve Harenberg, Christos Faloutsos, and Nagiza F Samatova. Anomaly detection in dynamic networks: a survey. *Wiley Interdisciplinary Reviews: Computational Statistics*, 7(3):223–247, 2015.

[173] Santanu Saha Ray. *Graph theory with algorithms and its applications: in applied science and technology*. Springer Science & Business Media, 2012.

[174] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3577–3586, 2017.

[175] Krzysztof Rusek, José Suárez-Varela, Albert Mestres, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Unveiling the potential of graph neural networks

for network modeling and optimization in sdn. In *Proceedings of the 2019 ACM Symposium on SDN Research*, pages 140–151, 2019.

[176] Ruslan Salakhutdinov and Iain Murray. On the quantitative analysis of deep belief networks. In *Proceedings of the 25th international conference on Machine learning*, pages 872–879. ACM, 2008.

[177] Bidisha Samanta, Abir De, Gourhari Jana, Vicenc Gómez, Pratim Kumar Chattaraj, Niloy Ganguly, and Manuel Gomez-Rodriguez. Nevae: A deep generative model for molecular graphs. *Journal of machine learning research. 2020 Apr; 21 (114): 1-33*, 2020.

[178] Mark Sanderson, Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. Introduction to information retrieval, cambridge university press. 2008. isbn-13 978-0-521-86571-5, xxi+ 482 pages. *Natural Language Engineering*, 16(1):100–103, 2010.

[179] Paula Sanz Leon, Stuart A Knock, M Marmaduke Woodman, Lia Domide, Jochen Mersmann, Anthony R McIntosh, and Viktor Jirsa. The virtual brain: a simulator of primate brain network dynamics. *Frontiers in Neuroinformatics*, 7:10, 2013.

[180] Ryoma Sato. A survey on the expressive power of graph neural networks. *arXiv preprint arXiv:2003.04078*, 2020.

[181] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[182] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[183] Kaspar A Schindler, Stephan Bialonski, Marie-Therese Horstmann, Christian E Elger, and Klaus Lehnertz. Evolving functional network properties and synchronizability during human epileptic seizures. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 18(3):033119, 2008.

[184] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer, 2018.

[185] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[186] Dino Sejdinovic, Bharath Sriperumbudur, Arthur Gretton, and Kenji Fukumizu. Equivalence of distance-based and rkhs-based statistics in hypothesis testing. *The Annals of Statistics*, pages 2263–2291, 2013.

[187] Raghavendra Selvan, Thomas Kipf, Max Welling, Jesper H Pedersen, Jens Petersen, and Marleen de Bruijne. Extraction of airways using graph neural networks. *arXiv preprint arXiv:1804.04436*, 2018.

[188] A K Shah and Sandeep Mittal. Invasive electroencephalography monitoring: Indications and presurgical planning. *Annals of Indian Academy of Neurology*, 17(Suppl 1):S89, 2014.

[189] Uri Shaham, Kelly Stanton, Henry Li, Boaz Nadler, Ronen Basri, and Yuval Kluger. Spectralnet: Spectral clustering using deep neural networks. *arXiv preprint arXiv:1801.01587*, 2018.

[190] Amir Shahroudy, Jun Liu, T-T Ng, and Gang Wang. NTU RGB+D: A large scale dataset for 3d human activity analysis. In *The IEEE Conference on Computer Vision and Pattern Recognition*, Las Vegas Valley, Nevada, United States, June 2016.

[191] Hang Shao, Abhishek Kumar, and P. Thomas Fletcher. The riemannian geometry of deep generative models. In *CVPR Workshops*, 2018.

[192] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Departmental Papers (CIS)*, page 107, 2000.

[193] Jonathan Shlomi, Peter Battaglia, and Jean-Roch Vlimant. Graph neural networks in particle physics. *Machine Learning: Science and Technology*, 2 (2):021001, 2020.

[194] David I Shuman, Mohammad Javad Faraji, and Pierre Vandergheynst. A multiscale pyramid transform for graph signals. *IEEE Transactions on Signal Processing*, 64(8):2119–2134, 2016.

[195] Martin Simonovsky and Nikos Komodakis. Dynamic edgeconditioned filters in convolutional neural networks on graphs. In *Proc. CVPR*, 2017.

[196] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. *arXiv preprint arXiv:1802.03480*, 2018.

[197] Gurjeet Singh, Facundo Mémoli, and Gunnar E Carlsson. Topological methods for the analysis of high dimensional data sets and 3d object recognition. In *SPBG*, pages 91–100, 2007.

[198] Alessandro Sperduti and Antonina Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.

[199] C E Stafstrom and Lionel Carmant. Seizures and epilepsy: An overview for neuroscientists. *Cold Spring Harbor Perspectives in Medicine*, 5(6): a022426, 2015.

[200] Willeke Staljanssens, Gregor Strobbe, Roel Van Holen, Gwénaël Birot, Markus Gschwind, Margitta Seeck, Stefaan Vandenberghe, Serge Vulliémoz, and Pieter van Mierlo. Seizure onset zone localization from ictal high-density eeg in refractory focal epilepsy. *Brain Topography*, 30(2): 257–271, 2017.

[201] X Yu Stella and Jianbo Shi. Multiclass spectral clustering. In *Computer Vision, IEEE International Conference on*, volume 2, pages 313–313. IEEE Computer Society, 2003.

[202] Julian Straub, Jason Chang, Oren Freifeld, and John Fisher III. A Dirichlet process mixture model for spherical data. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics*, volume 38, pages 930–938, San Diego, CA, USA, 2015.

[203] Jimeng Sun, Yinglian Xie, Hui Zhang, and Christos Faloutsos. Less is more: Sparse graph mining with compact matrix decomposition. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 1(1):6–22, 2008.

[204] Fei Tian, Bin Gao, Qing Cui, Enhong Chen, and Tie-Yan Liu. Learning deep representations for graph clustering. In *AAAI*, pages 1293–1299, 2014.

[205] Peter Tiňo. Dynamical systems as temporal feature spaces. *Journal of Machine Learning Research*, 21(44):1–42, 2020.

[206] Nicolas Tremblay, Paulo Goncalves, and Pierre Borgnat. Design of graph filters and filterbanks. In *Cooperative and Graph Signal Processing*, pages 299–324. Elsevier, 2018.

[207] Alain Trémeau and Philippe Colantoni. Regions adjacency graph applied to color image segmentation. *IEEE Transactions on image processing*, 9(4): 735–744, 2000.

[208] Luca Trevisan. Max cut and the smallest eigenvalue. *SIAM Journal on Computing*, 41(6):1769–1786, 2012.

[209] Pieter Van Mierlo, Margarita Papadopoulou, Evelien Carrette, Paul Boon, Stefaan Vandenberghe, Kristl Vonck, and Daniele Marinazzo. Functional brain connectivity from eeg in epilepsy: Seizure prediction and epileptogenic focus localization. *Progress in Neurobiology*, 121:19–35, 2014.

[210] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[211] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[212] Kirill Veselkov, Guadalupe Gonzalez, Shahad Aljifri, Dieter Galea, Reza Mirnezami, Jozef Youssef, Michael Bronstein, and Ivan Laponogov. Hyperfoods: Machine intelligent mapping of cancer-beating molecules in foods. *Scientific reports*, 9(1):1–12, 2019.

[213] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015.

[214] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[215] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds.(2018). *arXiv preprint arXiv:1801.07829*, 2018.

[216] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International joint conference on neural networks (IJCNN)*, pages 1578–1585. IEEE, 2017.

[217] David S Watkins. *Fundamentals of matrix computations*, volume 64. John Wiley & Sons, 2004.

[218] Duncan J Watts and Steven H Strogatz. Collective dynamics of small-world networks. *nature*, 393(6684):440–442, 1998.

[219] Kurt E Weaver, WA Chaovalitwongse, Edward J Novotny, Andrew Poliakov, Thomas J Grabowski Jr, and Jeffrey G Ojemann. Local functional connectivity as a pre-surgical tool for seizure focus identification in non-lesion, focal epilepsy. *Frontiers in Neurology*, 4:43, 2013.

[220] Zaiwen Wen and Wotao Yin. A feasible method for optimization with orthogonality constraints. *Mathematical Programming*, 142(1-2):397–434, 2013.

[221] James D Wilson, Nathaniel T Stevens, and William H Woodall. Modeling and detecting change in temporal networks via a dynamic degree corrected stochastic block model. *arXiv preprint arXiv:1605.04049*, 2016.

[222] R. C. Wilson, E. R. Hancock, E. Pekalska, and R. P. W. Duin. Spherical and hyperbolic Embeddings of data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2255–2269, Nov. 2014. ISSN 0162-8828. doi: 10.1109/TPAMI.2014.2316836.

[223] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. Simplifying graph convolutional networks. *arXiv preprint arXiv:1902.07153*, 2019.

[224] Jun Wu, Jingrui He, and Jiejun Xu. Net: Degree-specific graph neural networks for node and graph classification. *arXiv preprint arXiv:1906.02319*, 2019.

[225] Lingfei Wu, Yu Chen, Kai Shen, Xiaojie Guo, Hanning Gao, Shucheng Li, Jian Pei, and Bo Long. Graph neural networks for natural language processing: A survey. *arXiv preprint arXiv:2106.06090*, 2021.

[226] Shiwen Wu, Fei Sun, Wentao Zhang, and Bin Cui. Graph neural networks in recommender systems: a survey. *arXiv preprint arXiv:2011.02260*, 2020.

[227] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[228] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[229] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

[230] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[231] Tian Xie and Jeffrey C Grossman. Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties. *Physical review letters*, 120(14):145301, 2018.

[232] Yu Xie, Chuanyu Yao, Maoguo Gong, Cheng Chen, and AK Qin. Graph convolutional networks with multi-level coarsening for graph classification. *Knowledge-Based Systems*, page 105578, 2020.

[233] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

[234] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[235] Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *AAAI*, 2018.

[236] Li Yi, Hao Su, Xingwen Guo, and Leonidas J Guibas. Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2282–2290, 2017.

[237] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *arXiv preprint arXiv:1806.08804*, 2018.

[238] Jiaxuan You, Rex Ying, Xiang Ren, William L Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. *arXiv preprint arXiv:1802.08773*, 2018.

[239] Jiaxuan You, Zhitao Ying, and Jure Leskovec. Design space for graph neural networks. *Advances in Neural Information Processing Systems*, 33, 2020.

[240] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875*, 2017.

[241] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in neural information processing systems*, pages 3391–3401, 2017.

[242] Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, et al. Relational deep reinforcement learning. *arXiv preprint arXiv:1806.01830*, 2018.

[243] D. Zambon, C. Alippi, and L. Livi. Concept drift and anomaly detection in graph streams. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–14, Mar. 2018. doi: 10.1109/TNNLS.2018.2804443.

[244] Daniele Zambon, Lorenzo Livi, and Cesare Alippi. Detecting changes in sequences of attributed graphs. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7. IEEE, 2017.

[245] Daniele Zambon, Lorenzo Livi, and Cesare Alippi. Anomaly and change detection in graph streams through constant-curvature manifold embeddings. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2018.

[246] Daniele Zambon, Cesare Alippi, and Lorenzo Livi. Change-point methods on a sequence of graphs. *IEEE Transactions on Signal Processing*, 67(24): 6327–6341, 2019.

[247] Daniele Zambon, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Autoregressive models for sequences of graphs. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.

[248] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of AAAI Conference on Artificial Inteligence*, 2018.

[249] Denny Zhou, Olivier Bousquet, Thomas N Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. In *Advances in neural information processing systems*, pages 321–328, 2004.

[250] Philipp Zumstein. Comparison of spectral methods through the adjacency matrix and the laplacian of a graph. *TH Diploma, ETH Zürich*, 2005.