Geometric Deep Learning: from grid to graph structured data

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
Federico Monti

under the supervision of
Prof. Michael M. Bronstein

January 2025

Dissertation Committee

Prof. Cesare Alippi	Università della Svizzera italiana, Lugano, Switzerland
Prof. Andrea-Emilio Rizzoli	Università della Svizzera italiana, Lugano, Switzerland
Prof. Xavier Bresson	National University of Singapore, Singapore

Dissertation accepted on 23 January 2025

Prof. Michael M. Bronstein
Research Advisor
Università della Svizzera italiana, Lugano, Switzerland

The PhD program Director
Prof. Walter Binder

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.

Federico Monti
Lugano, 23 January 2025

To my entire family.

Abstract

The success of Deep Learning architectures (e.g. Convolutional Neural Networks, Recurrent Neural Networks, Transformers, ...) and the increasing availability of graph/manifold structured data (e.g. social networks, sensor networks, molecules, 3D shapes, ...) motivated, in recent years, the development of a new class of Geometric Deep Learning (GDL) approaches aimed at extending traditional DL solutions to non-Euclidean domains.

In this thesis, we explore the realm of Graph Convolutional Neural Networks (GCNNs), a popular class of GDL architectures that rely on generalizations of the convolution operation to process the provided input data. Our contributions are organized in two main different parts.

In the first part of this manuscript, we focus on methodologies. We introduce in particular novel generalizations of convolution that are defined either in space or in the spectral domain. We present an attention mechanism able to generalize convolution through a generalization of the notion of pixel (MoNet), spectral filters able to process signals defined over multiple graphs (MGCNN), spectral filters with spectral zoom properties (CayleyNet), and a scalable inception-based architecture able to efficiently process graphs with millions of nodes and billions of edges (SIGN).

In the second part of our work, we direct our attention towards applications of GCNNs. First, we show how GCNNs allow to detect high-energy neutrinos by processing signals retrieved by the IceCube detector. Second, we introduce VeritasZero, a GCNN-based approach able to detect fake news based on the spreading patterns this forms on social media. Third, we present BP-IIG, a GCNN-based profiling attack that exploits the stability of people's interaction behavior to identify individuals in anonymous datasets.

Acknowledgements

Throughout my PhD studies I met a large amount of incredible people (both from the academic and industrial world), I had the opportunity to live in fabulous places and I consistently worked on cutting edge exciting projects that allowed me to grow beyond my most optimistic expectations. If there is one person that I have to thank for all of this is without a doubt my advisor. Michael has been a consistent source of inspiration during my days at USI. He provided me the opportunities that any student dreams of, and constantly pushed me to strive for excellence in every project we decided to face. I am a better scientist and a better engineer thanks to Michael, and for that, I will always be grateful.

Next in line, I would like to call out and thank all the amazing collaborators that I had the opportunity to work with in the past years: Davide Boscaini, Davide Eynard, Jan Svoboda, Jonathan Masci, Emanuele Rodolá, Or Litany, Xavier Bresson, Ron Levie, Karl Otness, Emanuele Rossi, Fabrizio Frasca, Ben Simpson, Damon Mannion, Ernesto Schmitt, Pablo Gainza, Freyr Sverrisson, Bruno Correia, Nicholas Choma, Lisa Gerhardt, Tomasz Palczewski, Zahra Ronaghi, Prabhat Prabhat, Wahid Bhimji, Spencer R Klein, Joan Bruna, Ana-Maria Creţu, Xiaowen Dong, Yan Leng and Yves-Alexandre de Montjoye. It was a true pleasure working with and learning from each one of you.

Among the above, a special thanks goes in particular to all the people that I had the honor to work with at Fabula. Working at Fabula has been the most exciting and fulfilling experience of my life, and it would have not been so special if I didn't have such an amazing group of people around me.

Last but not least, I want to thank my entire family, and in particular my parents and my grandparents, for assisting me in my studies and pushing me to reach such an important milestone in my life. I would have not got to this if I didn't have the support of all of you.

Contents

Figures

Tables

Chapter 1

Introduction

Over the past decade, we witnessed the explosion of Deep Artificial Neural Networks (DNNs) as an effective paradigm for solving prediction and generation tasks in a variety of different areas. Image classification [144], image super-resolution [47], text / image generation [102, 45], speech recognition [264], and candidate generation / ranking [65, 56] are just few of the many examples of successful applications that this specific class of techniques found in recent times. In the Computer Vision field, Convolutional Neural Networks (CNNs) received in particular a lot of attention for the ground-breaking results that they allowed to achieve on popular benchmarks (e.g. the ImageNet challenge). As a provided motif (e.g. an edge, a corner, a blob) has the same chance of appearing in every position of a provided input image [151], in CNNs the same feature detector (same artificial neuron) is applied indiscriminately throughout the entire domain by simply sliding the window of operation of said detector. This "sliding window" behavior is implemented in CNNs, and in particular in *convolutional layers*, through the so-called *convolution operation*, which gives the name to this class of approaches as it represents their fundamental building block. The ultimate effect of processing a provided input with a *convolutional layer* is that each single neuron gets to process a significantly larger amount of data with respect to what a classic dense layer would allow (as it is applied multiple times over the same domain), thus reducing the chances of overfitting and in turn boosting performance.

While CNNs found large success in the processing of images (as well as videos [47, 132, 250], and audio signals [235, 103, 169] - i.e. signals defined over regular grids), in many situations, one might need to process signals that are defined over graphs or manifolds (i.e. non-Euclidean domains). Typical examples of this class of data are for instance citation networks (where nodes are documents typically decorated with their text, [270]), social networks (which describe users and the way they connect among each other, [187]) or 3D shapes (discretized as triangular meshes [38, 174, 184]). Unfortunately, due to the irregularity and lack of a global reference system of these domains, the very same definition of convolution is not well defined on network / manifold data. As a result, classic CNNs cannot be directly applied with the same construction we use for processing signals defined over grids, and new mechanisms are required for analyzing the provided information.

Motivated by the lack of techniques capable of processing non-Euclidean structured data, in a similar manner as classic CNNs do for regular grids, *Graph Convolutional Neural Networks* (GCNNs) appeared on the scene in the last years [43]. GCNNs are a particular class of *Geometric Deep Learning* approaches, a broader movement that appeared in the literature in recent time,

which aims at extending classic Deep Learning approaches to non-Euclidean domains[1]. In order to analyze the provided input, GCNNs leverage generalizations of the convolution operation that replicate somehow the way in which convolution works over grids. Spectral GCNNs resort to Graph Fourier Transform and a generalization of Fourier Convolution Theorem for extending convolution, while spatial GCNNs generalize convolution by means of message passing schemes that operate directly in space (e.g. along the edges of the provided graph). While the field of GCNNs has been a niche area of research until even a few years ago, the popularity of these architectures has been significantly increasing in recent times thanks to the broad applicability they present [11]. Today, GCNNs find application in traffic prediction on Google Maps [74], item recommendation in e-commerce [272, 7], link prediction on social media [51], molecular property prediction [99], fraud detection [79], and many other areas, as we shall see.

1.1 Contributions

In this thesis, we present some of the contributions of the author to the Geometric Deep Learning research field. In the first part of this manuscript, we introduce a series of possible generalizations of the convolution operation for non-Euclidean domains (MoNet, MGCNN, CayleyNet and SIGN) with particular emphasis on graph structured data. We highlight the benefits of each proposed generalization, and we compare it to previously presented approaches. For each architecture, we also provide a commentary section highlighting how the literature evolved from our own work.

MoNet. The *Mixture Model Neural Network* [184] is the first unified framework for generalizing convolution on both graph and manifold structured data. The main contribution of our MoNet architecture is a patch operator that generalizes the notion of pixel to non-Euclidean domains, by using local pseudo coordinates that define the relationship between neighboring nodes. It is also the first attention based GCNN that was ever introduced in the literature.

MGCNN. The *Multi-Graph Convolutional Neural Network* [185] is a generalization of the spectral approach proposed in [72] for signals defined over multiple graphs. Thanks to a suitable polynomial parametrization of spectral coefficients, in [185] we showed how multi-graph spectral filters, which enjoy a linear complexity in the number of entries of the provided input signal, could be realized. Our MGCNN was the cornerstone for solving matrix-completion problems in [185]. Coupling MGCNN with a Recurrent Neural Network (e.g. LSTM [119]), we showed in particular how an architecture (RMGCNN) able to learn a diffusion process aimed at reconstructing the missing information could be constructed. A single graph simplification (with better complexity) of RMGCNN was additionally presented in the paper.

CayleyNet. In [153] we introduced *CayleyNet*, a spectral GCNN that is able to detect at training time a frequency band of interest and to realize filters that specialize on that. The core ingredient of our model is a new class of parametric rational complex functions (Cayley polynomials), which allow to efficiently compute spectral filters that enjoy some form of "spectral zoom" property. Thanks to this specific construction, CayleyNet outperformed in experimental

[1] Other examples of GDL architectures are for instance neural networks aimed at processing point clouds or sets [276, 207, 208].

evaluation previously presented approaches [72] in community detection, document classification, and matrix completion tasks.

SIGN. In *Scalable Inception Graph Neural Networks* [90] we propose a scalable GCNN able to deal with extremely large graphs (in the order of million of nodes and billion of edges). Instead of doing diffusion on a graph on the fly, in SIGN we pre-compute aggregated descriptors at pre-processing time using a predefined set of diffusion operators. The obtained descriptors are then fed in input to a multi-layer perceptron reproducing an inception like architecture, in order to produce the final predictions. Despite the simplicity of our architecture, in our experiments we showed how SIGN was able to achieve competitive performance with respect to previously presented solutions designed with scalability in mind, while requiring smaller training / inference times.

In the second part of the manuscript, we introduce instead some applications of GCNNs in the realms of High-Energy Physics, Social Network Data Analysis, and Data Privacy. Similarly to what done for the methodologies discussed in Part I, each chapter is decorated with a commentary section, where we discuss relevant methods that have been proposed following our work.

Neutrino detection. A neutrino is a lepton (a type of elementary particle) with extremely small mass and no charge. As a result of these properties, neutrinos do not participate in electromagnetic interactions, they are only weakly affected by gravity (the weakest force in nature), and they participate in nuclear interactions only through the so-called weak nuclear force (the second weakest force in nature) [3]. Because of this, neutrinos can travel across matter without having their trajectory largely affected, appearing as such as the ideal *cosmic messenger* to pinpoint the source of interesting astrophysical phenomena (e.g. supernova explosions). In [60], we studied how GCNNs can be used to detect the passage of high energy neutrinos by processing signals retrieved by the IceCube detector. Using a GCNN implemented with our MoNet convolutional layer, we were able to show how GCNNs can significantly achieve higher detection rates with respect to both classic CNNs and a physics baseline realized by the IceCube collaboration.

Fake-news detection. The proliferation of social media in the last years lead always more people to access news online. While social media offer on one hand an easy-access low cost method for news consumption, they also expose their audience to potentially fabricated information, which can be crafted with the idea of manipulating the public opinion to achieve a specific political agenda. In [255], it was shown how false and true information spread in different ways on Twitter, and in particular how misinformation appears to spread faster and deeper compared to reliable information. Inspired by this work, in [187] we constructed a fake news detection model (VeritasZero) capable of directly learning propagation patterns that could allow to distinguish fake news from true news. In our experiments, VeritasZero achieves significantly high accuracy (nearly 93% ROC AUC), it requires very short news spread times (just a few hours of propagation) to approach maximum performance, and performs well even when the model is trained on data distant in time from the testing data.

User identification. Fine-grained records of people's interactions, both offline and online, are collected at large scale everyday. These data contain sensitive information about whom we meet, talk to, and when. In [66], we showed how GCNNs can be used to identify individuals in datasets of psuedonymized interaction networks based on the weekly behavior of their k-hop neighborhood. On a mobile phone metadata dataset of more than 40,000 people, our behavioral profiling attack (BP-IIG) correctly identifies 52% of individuals based on fingerprints extracted from their 2-hop interaction graph, and 15% of the them using only their 1-hop graph. The profiles learned by our method are additionally stable over long periods of time, and 24% of people can still be identified after 20 weeks. On a Bluetooth close-proximity dataset of 587 students, our approach correctly identifies more than 26% of the individuals using only their 1-hop interaction graph. Our results provide evidence that the disconnected and even frequently re-pseudonymized interaction data of an individual can be linked together, raising the question of whether they satisfy the anonymization standard set forth in GDPR.

1.2 Publications

The content of this thesis is based on the following publications (* denotes equal contribution):

- 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 *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5115–5124, 2017

- Federico Monti, Michael Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent multi-graph neural networks. *Advances in neural information processing systems*, 30, 2017

- Ron Levie*, Federico Monti*, Xavier Bresson, and Michael M Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Transactions on Signal Processing*, 67(1):97–109, 2018

- Federico Monti, Karl Otness, and Michael M Bronstein. Motifnet: a motif-based graph convolutional network for directed graphs. In *2018 IEEE Data Science Workshop (DSW)*, pages 225–228. IEEE, 2018

- Fabrizio Frasca*, Emanuele Rossi*, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. *Graph Representation Learning and Beyond, ICML Workshop*, 2020

- Nicholas Choma, Federico Monti, Lisa Gerhardt, Tomasz Palczewski, Zahra Ronaghi, Prabhat Prabhat, Wahid Bhimji, Michael M Bronstein, Spencer R Klein, and Joan Bruna. Graph neural networks for icecube signal classification. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 386–391. IEEE, 2018

- Federico Monti, Fabrizio Frasca, Davide Eynard, Damon Mannion, and Michael M Bronstein. Fake news detection on social media using geometric deep learning. *Representation Learning on Graphs and Manifolds workshop*, 2019

- Ana-Maria Creţu, Federico Monti, Stefano Marrone, Xiaowen Dong, Michael Bronstein, and Yves-Alexandre de Montjoye. Interaction data are identifiable even across long periods of time. *Nature Communications*, 13(1):313, 2022

On top of the above, the author additionally contributed to the following publications during the course of his PhD studies:

- Jan Svoboda, Federico Monti, and Michael M Bronstein. Generative convolutional networks for latent fingerprint reconstruction. In *2017 IEEE International joint conference on biometrics (IJCB)*, pages 429–436. IEEE, 2017

- Federico Monti, Oleksandr Shchur, Aleksandar Bojchevski, Or Litany, Stephan Günnemann, and Michael M Bronstein. Dual-primal graph convolutional networks. *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2019

- Jan Svoboda, Jonathan Masci, Federico Monti, Michael M Bronstein, and Leonidas Guibas. Peernets: Exploiting peer wisdom against adversarial attacks. *7th International Conference on Learning Representations, ICLR*, 2019

- Kevin McCloskey, Ankur Taly, Federico Monti, Michael P Brenner, and Lucy J Colwell. Using attribution to decode binding mechanism in neural network models for chemistry. *Proceedings of the National Academy of Sciences*, 116(24):11624–11629, 2019

- Pablo Gainza, Freyr Sverrisson, Federico Monti, Emanuele Rodola, D Boscaini, Michael M Bronstein, and Bruno E Correia. Deciphering interaction fingerprints from protein molecular surfaces using geometric deep learning. *Nature Methods*, 17(2):184–192, 2020

- Emanuele Rossi, Federico Monti, Michael Bronstein, and Pietro Liò. ncrna classification with graph convolutional networks. *arXiv preprint arXiv:1905.06515*, 2019

- Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. *Graph Representation Learning and Beyond, ICML Workshop*, 2020

- Emanuele Rossi, Federico Monti, Yan Leng, Michael Bronstein, and Xiaowen Dong. Learning to infer structures of network games. In *International Conference on Machine Learning*, pages 18809–18827. PMLR, 2022

- Vanessa Cai, Pradeep Prabakar, Manuel Serrano Rebuelta, Lucas Rosen, Federico Monti, Katarzyna Janocha, Tomo Lazovich, Jeetu Raj, Yedendra Shrinivasan, Hao Li, et al. Twerc: High performance ensembled candidate generation for ads recommendation at twitter. *29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD*, 2023

Part I

Convolution on non-Euclidean domains

Chapter 2

Background

In the first chapter of this part, we introduce the fundamental concepts needed for understanding the contribution of the author. We will in particular review classic Convolutional Neural Networks (CNNs), introduce the basics of Graph Theory and Signal Processing on Graphs, and finally review some prior art that will be useful in the analysis of the proposed GCNNs as a term of comparison. For the sake of clarity, throughout this document we will describe a scalar as x, a vector as $\mathbf{x}$, a matrix as $\mathbf{X}$, functions / signals defined over a continuous Euclidean space as $f(.)$, functions / signals defined over a discrete grid as $f[.]$ and functions defined over a sequence of elements (e.g. nodes of a graph, feature maps produced by a convolutional layer, ...) as $f_.$.

2.1 Convolutional Neural Networks on Euclidean domains

Convolutional Neural Networks [151, 144, 110, 123] are a particular realization of Deep Neural Networks, which received a lot of attention in the last years thanks to the groundbreaking performance they were able to achieve on a variety of different tasks [144, 47, 133]. The success of Convolutional Neural Networks is owed to their capacity of exploiting statistical properties of the input signal - in particular stationarity and compositionality - to contain the number of parameters without sacrificing the representation power of the model [43]. Stationarity and compositionality are two statistical properties which can be easily found in natural images, videos and speech signals [89, 196, 233]. In this context, compositionality refers to the hierarchical nature of features which can be built by a composition of other (more localized) features at a lower level (e.g. in an image, a set of edges construct a border, a set of borders define the shape of a person, ...) and stationarity implies that same motifs have the same probability of appearing in every location of the domain where the signal is defined. To exploit these two fundamental properties to contain the number of parameters and limit overfitting, CNNs rely on the convolution operation for extrapolating relevant patterns. Formally, provided a signal f and a filter h, the convolution of f and h is defined as:

$$g(t) = (f \star h)(t) = \int_{-\infty}^{+\infty} f(t-\tau)h(\tau)d\tau. \tag{2.1}$$

For functions defined over a discrete domain (e.g. digital images), this in turn corresponds to:

$$g[t] = (f \star h)[t] = \sum_{k=-\lfloor K/2 \rfloor}^{\lfloor K/2 \rfloor} f[t-k]h[k]; \tag{2.2}$$

with $h[.]$ showing non-zero values only in $\{-\lfloor K/2 \rfloor, \dots, \lfloor K/2 \rfloor\}$[1]. From equation (2.2), it is easy to see how, for any t, the output $g[t]$ of the convolution between $f[.]$ and $h[.]$ corresponds to nothing more than a linear combination of the values of the (input) signal f in a neighborhood of t (i.e. a patch of f centered around t). The coefficients of the linear combination are additionally independent on t and defined only by (filter) h. Considering the coefficients of h as parameters of a Neural Network, it's straightforward to see how a neural layer defined as in equation (2.2) is well suited to extrapolate features from stationary input signals. The same localized feature detector (here represented by filter h) is indeed applied throughout the entire domain in a sliding window fashion, thus avoiding the need for the model to re-learn the same feature in multiple different positions (*weight sharing*). To provide a pictorial example of this, Figure 2.1 shows the result of filtering multiple Dirac deltas located at multiple positions in space. Shifting the input in any direction simply corresponds in the same output but translated in the same direction and of the same amount where the input was shifted. As translations of the input result in equivalent translations of the output, we say that $g[.]$ is the result of processing $f[.]$ with a *Linear Shift Equivariant* system defined by the convolution operation and $h[.]$.

Convolution and Fourier Transform. Provided an absolutely integrable function $f \in L^1(\mathbb{R})$, the *Fourier Transorm* (FT) $\hat{f}(\xi) = \mathcal{F}\{f\}(\xi)$ and the *Inverse Fourier Transform* (IFT) $f(t) = \mathcal{F}^{-1}\{\hat{f}\}(t)$ are defined as:

$$\hat{f}(\xi) = \mathcal{F}\{f\}(\xi) = \int_{-\infty}^{+\infty} f(t) e^{-i2\pi\xi t} dt; \tag{2.3}$$

$$f(t) = \mathcal{F}^{-1}\{\hat{f}\}(t) = \int_{-\infty}^{+\infty} \hat{f}(\xi) e^{i2\pi\xi t} d\xi; \tag{2.4}$$

where ξ corresponds to a given frequency value and i is the imaginary unit. For a discrete function $f[.]$ defined over $\{0, 1, \dots, T-1\}$, the *Discrete Fourier Transform* (DFT) $\hat{f}[k] = \mathcal{F}_D\{f\}[k]$ and *Inverse Discrete Fourier Transform* (IDFT) $f[t] = \mathcal{F}_D^{-1}\{\hat{f}\}[t]$ are instead defined as:

$$\hat{f}[k] = \mathcal{F}_D\{f\}[k] = \sum_{t=0}^{T-1} f[t] e^{-i2\pi \frac{k}{T} t}, \quad k \in \{0, 1, \dots, T-1\}; \tag{2.5}$$

$$f[t] = \mathcal{F}_D^{-1}\{\hat{f}\}[t] = \frac{1}{T} \sum_{k=0}^{T-1} \hat{f}[k] e^{i2\pi \frac{k}{T} t}, \quad t \in \{0, 1, \dots, T-1\}; \tag{2.6}$$

with $k \in \{0, 1, \dots, T-1\}$ describing the *index* of a given frequency[2]. For the sake of this manuscript, it is interesting to analyze the relationship that exists between the Fourier Transform and the convolution of two signals. Provided two continuous signals $f(.)$ and $h(.)$, for the *Fourier Convolution Theorem* we have:

[1] While we focus in our definition on 1D signals, a similar construction can be implemented for a generic n-dimensional case simply expanding the convolution operation to include all the dimensions of the input domain.

[2] For functions defined over a generic interval $\{-\lfloor T/2 \rfloor, \dots, \lfloor T/2 \rfloor\}$, with T odd, the two equations are just the same, simply with the index of the summation in (2.5), and the domain of $f[.]$ in (2.6), ranging from $-\lfloor T/2 \rfloor$ to $\lfloor T/2 \rfloor$.

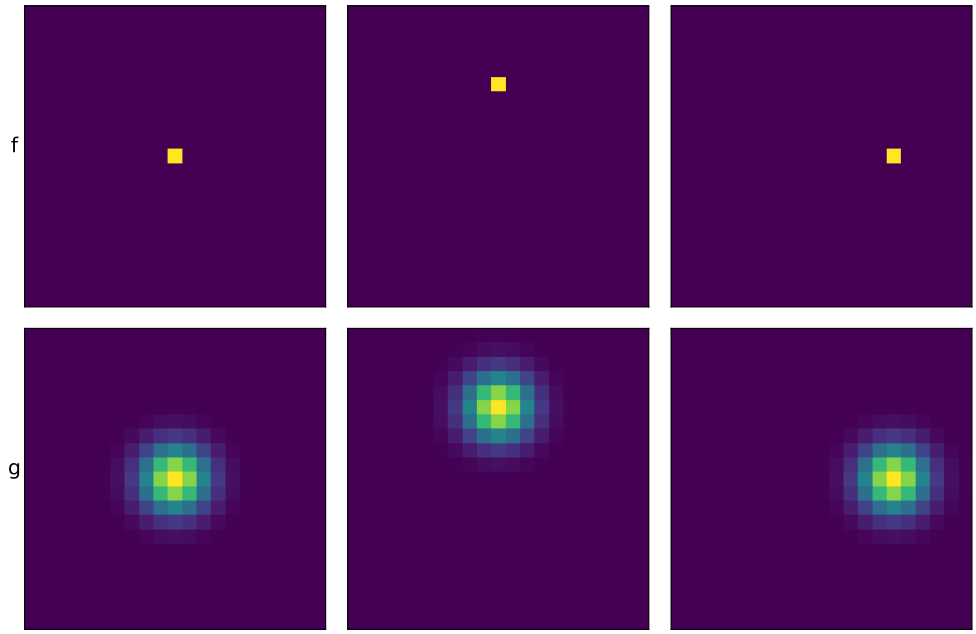


Figure 2.1. Top: multiple shifted version of input signal f. Bottom: the result of processing f with the convolution operation and a gaussian kernel ($g[.]$ was here cropped to have same size as $f[.]$ for clarity).

$$g(t) = (f \star h)(t) = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \cdot \mathcal{F}\{h\}\}. \tag{2.7}$$

Similarly, provided two discrete signals $f[.]$ and $h[.]$, with same length, and respectively defined over $\{0, \dots, T-1\}$ and $\{-\lfloor K/2 \rfloor, \dots, \lfloor K/2 \rfloor\}$:

$$g[t] = (f \circledast h)[t] = \mathcal{F}_D^{-1}\{\mathcal{F}_D\{f\} \cdot \mathcal{F}_D\{h\}\}; \tag{2.8}$$

where $\circledast$ is the *circular convolution operator*:

$$g[t] = (f \circledast h)[t] = \sum_{k=-\lfloor K/2 \rfloor}^{\lfloor K/2 \rfloor} f_T[t-k]h[k]. \tag{2.9}$$

Here, $f_T[t] = \sum_{m=-\infty}^{\infty} f[t - mT]$ is the periodic summation of signal $f[.]$ of length T, and we assumed $f[.]$ and $h[.]$ to have equal number of values in order to guarantee same number of frequencies, for both signals, in (2.8). From equation (2.9), it is possible to see how, if $f[.]$ and $h[.]$ are two signals of size generically equal to T and K (with K odd, and not necessarily equal to T), by appropriately zero padding $f[.]$ and $h[.]$ to make them of length $T+K-1$ and imposing that $g[.]$ has non zero value only in $[-\lfloor K/2 \rfloor, T + \lfloor K/2 \rfloor]$, the cropped circular convolution of the padded f and h reproduces the output of the convolution of the two original signals. It then follows from equation (2.8) (which provides a means for realizing circular convolution in the spectral domain) that convolution can also be realized as a multiplication of the spectrums obtained through DFT under analogous conditions.

The relationship between convolution and the Fourier Transform is typically of interest in Digital Signal Processing (DSP) for two main reasons. First of all, it allows to understand the behavior of a given filter (i.e. whether it is a low pass, band pass or high pass filter). Second of all, it allows to realize efficient implementations of filtering operations. Thanks indeed to the *Fast Fourier Transform* (FFT) algorithm, the DFT of a given signal $f[.]$ of size equal to T can be computed in just a $\mathcal{O}(Tlog_2T)$ operations. As a result of this, deciding whether it is more efficient to convolve two signals in the space or in the spectrum is a matter concerning the length of the two signals. The complexity of filtering a signal $f[.]$ of T values with a filter $h[.]$ of size equal to K is $\mathcal{O}(T \cdot K)$ in space and $\mathcal{O}((T+K-1) \cdot log_2(T+K-1))$ in the frequency domain. For filters of limited length (namely $K << log_2T$) the spatial construction is the preferable solution (as it is cheaper), while for filters showing a large size ($K >> log_2T$) the spectral construction is a more efficient choice.

Convolutional layers. A 1D convolutional layer is generally obtained in CNNs expanding equations (2.2) / (2.8) with a learnable bias b and a non-linearity σ (e.g. ReLU, LeakyRelu, sigmoid, tanh, ...), to allow the detection of high level features via the combination of lower level ones (*compositionality of representation*):

$$g_k^{(i)}[t] = \sigma(\sum_{k'}(g_{k'}^{(i-1)} \star h_{k,k'}^{(i)})[t] + b_k^{(i)}); \tag{2.10}$$

here $h_{k,k'}^{(i)}$ corresponds to the filter processing output signal k' from the $(i-1)$-th layer to produce feature k in layer i. As filters implemented in CNNs typically aim at extracting elementary motifs (since these can be frequently used for describing the behavior of multiple parts of a provided input, e.g. edges in an image), the support of filters implemented in convolutional layers is generally small (e.g. 3×3 / 5×5 on images with hundreds of pixels per dimension). For this reason, the spatial approach outlined in (2.2) is typically the solution of choice for implementing convolution in CNNs.

Pooling Layers

While convolutional layers allow to extract local re-usable features from the input signal (which can be effectively used for solving localized predictions problems such as image denoising [127]), in many instances one needs to solve a prediction problem at a global scale (e.g. image classification). In these situations, CNNs typically implement a second fundamental layer in their architecture, i.e. a pooling layer, to achieve robust predictions. Pooling layers (which are typically interleaved with convolutional layers) reduce the spatial resolution of each computed feature map by sliding a window of operation over the provided input signal with a stride larger than 1, and for each selected neighborhood compute an aggregated value:

$$\tilde{g}_k^{(i)}[t] = \square_{x \in \mathcal{N}_t} g_k^{(i)}[x]. \tag{2.11}$$

Here, t is the center of each window of points and $\square$ is the operation chosen for pooling (typically mean, sum or max). Figure 2.2 (top) provides an example of the result of applying max-pooling over a generic grey-scale image. Through this mechanism, pooling layers achieve two fundamental goals: (a) they build some level of invariance in the representation of the input signal with respect to small translations of the input (as only an approximate representation of

the input signal is provided to the next layer, figure 2.2 (bottom)); (b) they reduce the computational complexity of the entire architecture, as for each pooling layer that is introduced, only a fraction of the computed features are pushed through the remaining part of the architecture. If robustness with respect to translations is key for good performance, global pooling layers can additionally be introduced [157]. With global pooling layers, the features produced by convolutional layers are aggregated together in a unique representation that is completely independent on the absolute position of each computed feature:

$$\tilde{g}_k^{(I)} = \square_t g_k^{(I)}[t]. \tag{2.12}$$

This, in turn, makes the output of a CNN invariant to translations of the input, as (unless positional embeddings are used to describe the location of different points) no matter the position of a target in the domain, the final embedding $\tilde{\mathbf{g}}^{(I)}$ will always be the same. Since a vectorized representation with fixed length is computed no matter the support of the provided input signal, global pooling layers additionally make CNNs able to handle inputs of multiple sizes. As we shall see, this last property will come particularly in handy with GCNNs, as it often happens that graphs with variable number of vertices need to be processed with the same architecture.

2.2 Convolution and graph-structured data

In order to provide a full understanding to the reader of the topics discussed later in this thesis, we review here some fundamental concepts of Graph Theory and Signal Processing on Graphs (SGP).

2.2.1 Definitions

Undirected and directed graphs. A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ can be defined as a tuple consisting of the *node / vertex set* $\mathcal{V}$ and *edge set* $\mathcal{E}$. The node set defines the set of "items" that build the graph and the edges describe existing relations between pairs of nodes (edges in $\mathcal{G}$ can additionally be weighted to describe the strength of the connection). An edge and a node are called *incident* if the node is one of the end-points of the edge. Two edges are called *incident* if they share at least one end-point. Edges can be undirected in case of symmetric relations ("undirected edges" are simply referred to as *edges*) or directed if they are asymmetric ("directed edges" are typically referred to as *arcs*). If undirected edges are used in the definition of $\mathcal{G}$, the graph is said undirected and pairs of nodes in $\mathcal{E}$ are unordered. If directed edges are used instead, the graph is called directed and $\mathcal{E}$ is built by ordered pairs of nodes describing the direction of each relation (i.e. pair (i, j) describes an arc going from node j to node i). An edge connecting a node to itself is called a *self-loop*.

Neighborhoods on graphs. Two nodes are said *adjacent / neighbors* if there is at least one edge connecting them. The *neighborhood* $\mathcal{N}_i$ of a node i is the set of neighbors of i. In directed graphs, $\mathcal{N}_i$ can be divided between *in-neighborhood* $\mathcal{N}_i^- = \{j \in \mathcal{V} \,|\, (i,j) \in \mathcal{E}\}$ and *out-neighborhood* $\mathcal{N}_i^+ = \{j \in \mathcal{V} \,|\, (j,i) \in \mathcal{E}\}$. Nodes that are reachable in k steps from node i define the k-hop neighborhood $\mathcal{N}_i^{(k)}$ of i, said nodes are called k-hop neighbors of i.

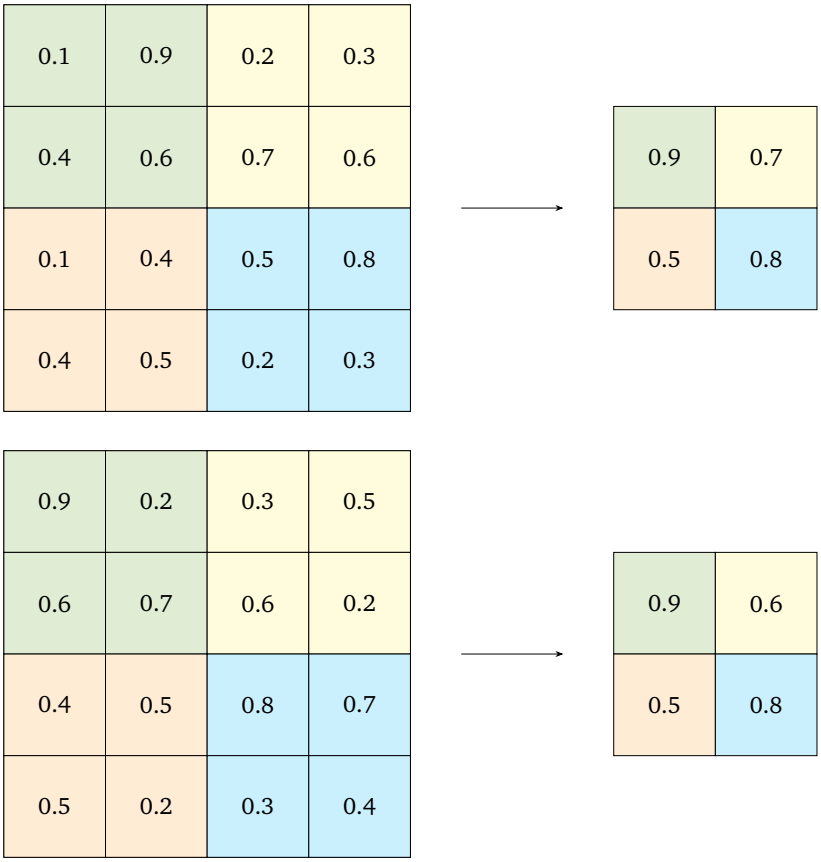


Figure 2.2. Top: Effect of max pooling over a portion of an image with windows of size 2×2 and stride equal to 2. Pixels with same color belong to the same neighborhood and are aggregated together. Bottom: Same pooling applied on a shifted version of the above image (shifted of 1 pixel to the left), only $1/4$ of the pixel values are changed after aggregation.

Connectivity. An undirected graph is called *connected* if from each node $i \in \mathcal{V}$ any other node $j \in \mathcal{V}$ can be reached "walking" along the edges. A undirected graph is called disconnected otherwise. A directed graph is called *strongly connected* if any node can be reached from any other by walking in the direction specified by the arcs. A directed graph is *weakly connected* if the associated undirected graph is connected. A directed graph is *disconnected* otherwise.

Adjacency matrix. A typical representation to describe the connectivity of a graph and the strength of the connections between vertices is the so-called *adjacency matrix* $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$. The adjacency matrix has a value $a_{i,j} \neq 0$ for entries $(i,j) \in \mathcal{E}$ and zero otherwise. For a given edge / arc (i,j), $a_{i,j} = 1$ when the associated graph is unweighted, $a_{i,j}$ can instead assume any other value in $\mathbb{R} \backslash \{0\}$ for weighted graphs. For undirected graphs, since the relations between nodes are symmetric, $a_{i,j} = a_{j,i}$, the adjacency matrix is symmetric as well (i.e. $\mathbf{A} = \mathbf{A}^T$).

The adjacency matrix of a graph can be normalized to describe the relative weight of an edge/arc with respect to its other incident edges (possibly computed considering only one given end-point). Popular normalized adjacency matrices in the literature are the *asymmet-*

ric normalized adjacency matrix $\tilde{\mathbf{A}}^{(asym)} = \mathbf{D}^{-1}\mathbf{A}$, and the *symmetric normalized adjacency matrix* $\tilde{\mathbf{A}}^{(sym)} = \mathbf{D}^{-0.5}\mathbf{A}\mathbf{D}^{-0.5}$ (unless specified otherwise, we describe with $\tilde{\mathbf{A}}$ the symmetric normalized adjacency matrix). $\mathbf{D} = diag(d_1, \dots, d_{|\mathcal{V}|})$ is the so-called *degree matrix* and each diagonal entry $d_i = \sum_{j \in \mathcal{V}} a_{i,j}$ is the degree of node i.

Signals on graphs. The nodes and edges of a given graph can be decorated with features in order to describe their respective properties. In social networks for instance, node-wise embeddings can be introduced to describe the past behavior of users (e.g. the tweets they might have liked or retweeted at some point); similarly, edge-wise features can be used to describe the frequency of interactions between two users and thus define the strength of a bond. Whenever a graph is equipped with this additional information, it is named *attributed*. Signals defined on the nodes / edges of a given graph are typically collected in matrix $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times F}$ / $\mathbf{E} \in \mathbb{R}^{|\mathcal{E}| \times \tilde{F}}$, where each row i represent a vertex / edge and each column a feature. The order of the rows of $\mathbf{X}$ is generally consistent with the one of $\mathbf{A}$ (or similar operators such as the graph Laplacian, see next paragraph) to allow an easy processing of the information.

Laplace operator. A relevant matrix for Signal Processing on Graphs is the (combinatorial) *Graph Laplacian* $\mathbf{\Delta} = \mathbf{D} - \mathbf{A}$ and its normalized version $\tilde{\mathbf{\Delta}} = \mathbf{D}^{-0.5}\mathbf{\Delta}\mathbf{D}^{-0.5} = \mathbf{I} - \tilde{\mathbf{A}}$. The Laplacian implements on graphs an equivalent version of the Laplace operator defined for Euclidean domains. The Laplace operator Δ is defined on grids as the diverge of the gradient of a scalar function f:

$$\Delta f(\mathbf{x}) = \nabla \cdot \nabla f(\mathbf{x}) = \sum_i \frac{\partial^2 f}{\partial x_i^2}(\mathbf{x}); \tag{2.13}$$

and it can be interpreted as an operator measuring the difference between the value of $f(.)$ in a point $\mathbf{x}$ and the average value of $f(.)$ in a neighborhood of $\mathbf{x}$[3]. For a generic signal $\mathbf{f}$ defined over the nodes of a graph, applying the Laplace operator on $\mathbf{f}$ simply corresponds with projecting $\mathbf{f}$ over the Graph Laplacian $\mathbf{\Delta}$ (or $\tilde{\mathbf{\Delta}}$): $\tilde{\mathbf{f}}_i = (\mathbf{\Delta}\mathbf{f})_i$.

2.2.2 Signal Processing on Graphs

As illustrated in Section 2.1, in order to extract re-usable features across a given grid, CNNs rely on the convolution operator for implementing filtering schemes. While the definition of convolution appears quite natural on grids (as it simply corresponds to a linear combination of signal values available on the neighbors of a target point), for signals defined over graphs things are not so straightforward. As graphs are not equipped with a global reference system, we don't have indeed a canonical way for sorting the neighbors of a given node. On top of this, the neighborhoods of different nodes are typically irregular, i.e. they generally show different sizes. Due to these two properties, the definition of convolution can't be directly applied for graph-structured data, and generalizations of such operator need to be sought to realize filters on these specific domains.

Broadly speaking, two main families of approaches can be identified in the literature, which aim at generalizing convolution for signals defined over graphs:

[3] Approximating the Laplace operator through finite differences, it is indeed possible to observe how: $\Delta f[\mathbf{x}] = \sum_i \frac{f[\mathbf{x}+h\cdot\mathbb{1}_i]-2f[\mathbf{x}]+f[\mathbf{x}-h\cdot\mathbb{1}_i]}{h^2}$; where $\mathbb{1}_i$ is a unit vector with zeros everywhere but in position i.

- *Spectral approaches*, which resort to a generalization of the Fourier Transform (the *Graph Fourier Transform*) and the Fourier Convolution Theorem.
- *Spatial approaches*, which generalize the notion of shift to graph structured data by means of message passing mechanisms.

In this subsection, we will review some of the most classic approaches for generalizing convolution on graphs both in the spectral and spatial domain. It should be noted that the separation between spectral and spatial generalizations of convolution is not necessarily well marked and, as we shall see, some approaches can be understood from both perspectives.

Spectral Graph Convolutional Neural Networks

Graph Fourier Transform. The first GCNN that made use of a spectral generalization of convolution for implementing filters over graph (at least that we are aware of) is the work of Bruna et al. [46]. Provided an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the vertex-set and $\mathcal{E}$ the edge set, $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ with $\mathbf{A}_{i,j} > 0 \iff (i,j) \in \mathcal{E}$ the (potentially weighted) adjacency matrix of $\mathcal{G}$, $\mathbf{D} = diag(\sum_j \mathbf{A}_{1,j}, \dots, \sum_j \mathbf{A}_{|\mathcal{V}|,j})$ the diagonal degree matrix and $\tilde{\mathbf{\Delta}} = \mathbf{I} - \mathbf{D}^{-0.5}\mathbf{A}\mathbf{D}^{-0.5} = \tilde{\mathbf{\Phi}}\tilde{\mathbf{\Lambda}}\tilde{\mathbf{\Phi}}^T$ the corresponding normalized graph Laplacian[4] (together with its orthogonal eigendecomposition), the *Graph Fourier Transform* (GFT) and *Inverse Graph Fourier Transform* (IGFT) of a generic signal $\mathbf{x}$ defined over $\mathcal{V}$ are defined as:

$$\hat{\mathbf{x}} = \tilde{\mathbf{\Phi}}^T\mathbf{x}; \qquad \mathbf{x} = \tilde{\mathbf{\Phi}}\hat{\mathbf{x}}. \tag{2.14}$$

The eigenfunctions of the Laplace operator corresponds in this scenario to a generalization of the Fourier basis used in spectral analsys and the eigenvalues $\tilde{\lambda}_i >= 0$ for $i \in \{1, .., |\mathcal{V}|\}$ describe the frequencies of the different functions building the basis. The eigenvectors of the graph Laplacian are typically considered a generalization of the Fourier basis for graphs due to the relationship that the complex exponential and the Laplace operator enjoy in Euclidean domains. The complex exponential corresponds in this sense to an eigenfunction of the Laplace operator on grids, and the corresponding eigenvalue provides a measure of frequency of the eigenfunction. As an example, in the 1D case we have:

$$\Delta e^{-i2\pi f x} = \frac{\partial^2 e^{-i2\pi f x}}{\partial x^2} = (2\pi f)^2 e^{-i2\pi f x}; \tag{2.15}$$

where $(2\pi f)^2$ is the eigenvalue of eigenfunction $e^{-i2\pi f x}$. On top of this, the eigenfunctions of the graph Laplacian and the complex exponential share similar properties over the respective domains. As the graph Laplacian is a symmetric operator, for the *Courant-Fisher theorem* we have:

$$\begin{aligned} \tilde{\lambda}_1 &= \min_{\substack{\mathbf{u}\in\mathbb{R}^{|\mathcal{V}|} \\ ||\mathbf{u}||_2=1}} \mathbf{u}^T\tilde{\mathbf{\Delta}}\mathbf{u}; \\ \tilde{\lambda}_k &= \min_{\substack{\mathbf{u}\in\mathbb{R}^{|\mathcal{V}|} \\ ||\mathbf{u}||_2=1 \\ \mathbf{u}\perp span\{\tilde{\boldsymbol{\phi}}_1,\dots,\tilde{\boldsymbol{\phi}}_{k-1}\}}} \mathbf{u}^T\tilde{\mathbf{\Delta}}\mathbf{u}; \end{aligned} \tag{2.16}$$

[4] Here we use the normalized Laplacian for the definition of a Graph Fourier Transform as it simplifies introducing follow up works. The combinatorial graph Laplacian could be used as well.

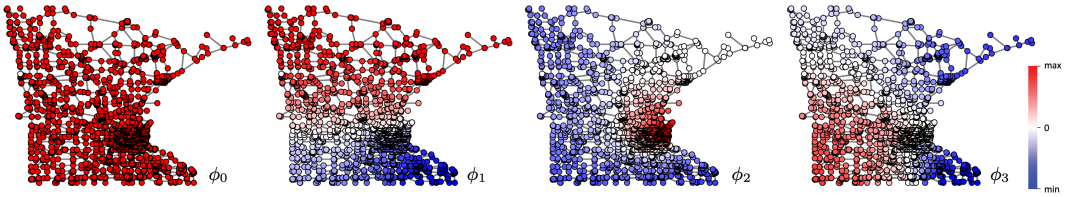


Figure 2.3. The first four eigenvectors of the graph Laplacian (visualized via a heat map) in increasing order of associated eigenvalue for the provided graph. The higher the value of λ, the lower the smoothness of the eigenvector (picture from [43]).

where $\mathbf{u}^T \tilde{\mathbf{\Delta}} \mathbf{u} = \sum_{(i,j)\in\mathcal{E}} a_{i,j}(\frac{u_i}{\sqrt{d_i}} - \frac{u_j}{\sqrt{d_j}})^2$ is the so-called *Dirichlet energy*[5] and $\tilde{\boldsymbol{\phi}}_k$ is the minimizer of the k-th problem. The eigenfunctions of $\tilde{\mathbf{\Delta}}$ thus correspond to the smoothest set of orthonormal functions that we can define over $\mathcal{V}$, where smoothness is defined in terms of Dirichlet energy. Each eigenvalue $\tilde{\lambda}_k$ corresponds to the Dirichlet energy of the associated eigenvector $\tilde{\boldsymbol{\phi}}_k$ and thus provides a measure of how rapidly $\tilde{\boldsymbol{\phi}}_k$ changes over $\mathcal{V}$. In Euclidean domains, two different complex exponential are orthogonal if associated to different frequencies[6] and the frequency f / the angular frequency $2\pi f$ provides a measure of how "variable" $e^{-i2\pi f x}$ is. Figure 2.3 depicts the first four eigenfunctions of the graph Laplacian of the Minnesota road network to provide an example of this set of functions.

Spectral convolution. Provided how to extend the Fourier Transform to graphs, convolution on $\mathcal{G}$ can then be defined in the spectral domain per analogy with the Fourier Convolution Theorem:

$$\tilde{\mathbf{x}} = \tilde{\mathbf{\Phi}} diag(\hat{\mathbf{h}}) \tilde{\mathbf{\Phi}}^T \mathbf{x}; \tag{2.17}$$

where $\hat{\mathbf{h}}$ is the vector of spectral coefficients of the filter we want to realize. Graph Convolutional layers implemented with (2.17) suffer however of a series of issues. First of all, the number of parameters required by the model is equal to the number of vertices of the provided graph, which could lead to overfitting. To provide an example of this, consider for instance a situation where one is trying to solve a node-wise prediction problem with labels provided only for some nodes of a graph. Provided a generic signal $\mathbf{x}$ defined over $\mathcal{V}$ with full spectrum (i.e. $|\hat{x}_i| > 0 \, \forall i$), the convolutional layer could simply learn a set of coefficients $\hat{\mathbf{h}} = \hat{\mathbf{y}}/\hat{\mathbf{x}}$, where $\hat{\mathbf{y}}$ is the spectrum of any signal showing the correct value on the given set of target nodes. As there are many signals $\mathbf{y}$ showing the correct value on the labeled vertices, it is easy to end up with poor generalization over the remaining nodes, and thus with poor inference performance.

A second second issue that affects equation (2.17) concerns the computational complexity of the approach. As matrix $\tilde{\mathbf{\Phi}}$ is a dense matrix and no FFT algorithm is available for graphs, the computational complexity of the GFT ($\tilde{\mathbf{\Phi}}^T \mathbf{x}$) and IGFT ($\tilde{\mathbf{\Phi}} \hat{\mathbf{x}}$) is quadratic in the number of vertices. This in turn limits the scalability of a GCNN with layers implemented as per (2.17), thus making it hard to apply such approaches to large graphs.

[5] Please note, in the case of the combinatorial Laplacian, the Dirichlet energy simplifies to: $\mathbf{u}^T \mathbf{\Delta} \mathbf{u} = \sum_{(i,j)\in\mathcal{E}} a_{i,j}(u_i - u_j)^2$.

[6] In the discrete case for two signals of N values: $< e^{-i2\pi n/Nx}, e^{-i2\pi m/Nx} > = \sum_{x=0}^{N-1} e^{-i2\pi(n-m)/Nx}$ which is equal to 0 for $n \neq m$ and equal to N for $n = m$ ($n \in \mathbb{Z}, m \in \mathbb{Z}$).

Finally, filters implemented as defined in (2.17) do not generalize well across graphs, since they depend on the specific eigendecomposition of the Laplace operator used for training. A typical example that is generally brought forward to explain this issue is the one of a filter trained on given graph $\mathcal{G}$, which shows a Laplacian eigenvalue $\bar{\lambda}$ with geometric multiplicity larger than 1. At inference time, even if we were using the model to process signals defined on the very same graph $\mathcal{G}$, unless the same exact eigendecomposition of $\tilde{\mathbf{\Delta}}$ was provided, for eigenvalue $\bar{\lambda}$ we could easily end up with a different set of eigenvectors that span the same eigenspace. This, in turn, could change the behavior of the GFT / IGFT, which would naturally affect the behavior of the filter on the given domain [43].

Spectral filters with splines. In order to reduce the amount of parameters required by the model, Bruna et al. [46] (and subsequently Henaff at al. [117]) proposed to parametrize the spectral coefficients of $\hat{\mathbf{h}}$ via cubic splines defined over the eigenvalues of $\tilde{\mathbf{\Delta}}$. While this potentially greatly reduces the number of parameters to learn and favors generalization across graphs (as the response of a cubic spline can be approximated with a sufficiently high order polynomial, which produces transferable filters across graphs - see next paragraph), such an approach still requires $\mathcal{O}(|\mathcal{V}|^2)$ operations due to the explicit GFT and IGFT.

ChebNet. To overcome all the aforementioned problems, Defferrard et al. [72] proposed to replace the cubic splines introduced in [46, 117] with *Chebyshev polynomials* applied over the eigenvalues of the graph Laplacian. In formula:

$$\tilde{\mathbf{x}} = \tilde{\mathbf{\Phi}} \left(\sum_{k=0}^{K} \omega_k T_k(\tilde{\mathbf{\Lambda}} - \mathbf{I}) \right) \tilde{\mathbf{\Phi}}^T \mathbf{x}; \tag{2.18}$$

with $T_k(.)$ the Chebyshev polynomial of degree k defined as $T_0(x) = 1$, $T_1(x) = x$ and $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$. Interestingly, as $\tilde{\mathbf{\Phi}} T_k(\tilde{\mathbf{\Lambda}} - \mathbf{I}) \tilde{\mathbf{\Phi}}^T = T_k(\tilde{\mathbf{\Phi}}(\tilde{\mathbf{\Lambda}} - \mathbf{I})\tilde{\mathbf{\Phi}}^T) = T_k(\tilde{\mathbf{\Delta}} - \mathbf{I}) = T_k(-\mathbf{D}^{-0.5}\mathbf{A}\mathbf{D}^{-0.5})$, equation (2.18) can be written as:

$$\tilde{\mathbf{x}} = \tilde{\mathbf{\Phi}} \left(\sum_{k=0}^{K} \omega_k T_k(\tilde{\mathbf{\Lambda}} - \mathbf{I}) \right) \tilde{\mathbf{\Phi}}^T \mathbf{x} = \sum_{k=0}^{K} \omega_k T_k(-\mathbf{D}^{-0.5}\mathbf{A}\mathbf{D}^{-0.5}) \mathbf{x}; \tag{2.19}$$

which doesn't require any form of eigen-decomposition and boils down to a series of sparse-dense matrix multiplications between the shifted graph Laplacian (i.e. the normalized adjacency matrix) and the input signal. While the authors do not describe in their paper the rationale behind shifting the Laplace operator with the identity matrix, a good reason for using such shift is to avoid possible numerical instabilities that might derive from exploding feature values, and exploding gradients[7] [138]. Thanks to efficient routines, equation (2.19) shows a complexity equal to just $\mathcal{O}(K|\mathcal{E}|)$, which is significantly smaller than $\mathcal{O}(|\mathcal{V}|^2)$ on sparse graphs. On top of this, filters are transferable across graphs. As it was shown in [154, 134], polynomial filters $g(\tilde{\mathbf{\Delta}})$ are stable stable under perturbations of the Laplace operator (i.e. $\|g(\tilde{\mathbf{\Delta}}) - g(\tilde{\mathbf{\Delta}}')\|_2 \leq \mathcal{O}(\|\tilde{\mathbf{\Delta}} - \tilde{\mathbf{\Delta}}'\|_2)$), and filters applied over similar graphs yield similar responses.

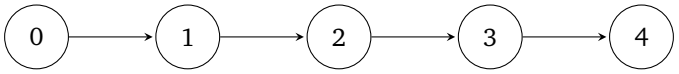


Figure 2.4. Pictorial representation of a directed path graph describing a 1D grid, with each node represented by its coordinate value. Walking to the left/right in the graph corresponds with moving towards the left/right in the grid.

Spatial Graph Convolutional Neural Networks

An alternative way of defining convolution on graphs is to work directly on the nodes and edges of a graph, generalizing the notion of shift that we naturally have in Euclidean domains. In this direction, assuming a filter that is symmetric around the origin for the sake of generality, equation (2.2) defining convolution (in space) over grids can be rewritten as:

$$g[t] = \sum_{k=-\lfloor K/2 \rfloor}^{\lfloor K/2 \rfloor} (\tau_k f)[t] h[k]; \tag{2.20}$$

where $\tau_k(.)$ implements a shift operator (i.e. $(\tau_k f)[t] = f[t-k]$). The convolution operation can thus be interpreted as nothing more than a linear combination of shifted versions of input $f[.]$, where each $(\tau_k f)[.]$ receives a coefficient $h[k]$ that depends on the magnitude and direction of the shift. Taking for simplicity a 1D signal defined over a grid and interpreting the domain as a directed *path graph* (Figure 2.4) with adjacency matrix $\check{\mathbf{A}}$ defined as $\check{a}_{ij} = 1 \iff j = i-1$ and 0 otherwise (namely $\check{a}_{ij} = 1$ if and only if i follows j in the grid), then equation (2.20) can be rewritten as:

$$g[t] = \sum_{k=0}^{\lfloor K/2 \rfloor} (\check{\mathbf{A}}^k \mathbf{f})[t] h[k] + \sum_{k=0}^{\lfloor K/2 \rfloor} \left((\check{\mathbf{A}}^T)^k \mathbf{f}\right)[t] h[-k]; \tag{2.21}$$

where matrices $\check{\mathbf{A}}$ and $\check{\mathbf{A}}^T$ operate in this scenario as two different shift operators (which shift the signal towards the right and the left respectively) and their exponent defines the magnitude of the shift. As this specific formulation is defined just in terms of powers of the adjacency matrix, we can think to apply it to any form of directed graph, where the operators $\check{\mathbf{A}}^k$ and $(\check{\mathbf{A}}^T)^k$ shifts to each node in the graph the values of the input signal that are available on nodes k steps apart. More generally, if no specific direction is provided, i.e. if the provided signal is defined on an undirected graph, equation (2.21) can be rewritten as:

$$g[t] = \sum_{k=0}^{K-1} (\mathbf{A}^k \mathbf{f})[x] h[t]; \tag{2.22}$$

where $\mathbf{A}$ is the undirected adjacency matrix, which operates in this case as a generalized shift operator [197]. As $(\mathbf{A}^k \mathbf{f})_i = \sum_{j \in \mathcal{N}_i^{(k)}} a_{i,j}^k f_j$ is a linear combination of signal values available in the k-hop neighborhood of node i (where each neighbor j of i receives a weight equal to the number of walks of length k connecting i to j), equation (2.22) can be understood as nothing more than a function of *neighborhood descriptors* that depicts the behavior of $\mathbf{f}$ in the surroundings of i. At a high level, this is the fundamental idea behind spatial GCNNs, where convolution can be

[7] As the eigenvalues of $\tilde{\mathbf{\Delta}}$ lies in the range $[0,2]$, $T_k(\tilde{\mathbf{\Delta}})\mathbf{x}$ can show large values when $\mathbf{x}$ aligns with some of the eigenvectors of $\tilde{\mathbf{\Delta}}$ associated with eigenvalues larger than 1.

defined as a function of signals that are exchanged and aggregated among neighboring nodes dependently on the relationship (e.g. distance between nodes, number of walks of a specific length, ...) that exists among them.

ChebNet as a spatial approach. From this perspective, it is interesting to re-analyze the work proposed by Defferrard et al. [72]. While we described ChebNet in the previous subsection as a spectral solution (which is the style chosen by the authors for presenting their work), equation (2.19) can also be interpreted from a spatial stand point. Indeed, a Chebyshev polynomial $T_k(\tilde{\mathbf{\Delta}} - \mathbf{I})$ of degree k simply corresponds to a linear combination of powers of the normalized adjacency matrix, which (as described above) shifts and aggregates signals from nodes that are k steps apart. The approach proposed in [72] can thus be intended as a spatial generalization of convolution for graph structured data that enjoys a spectral interpretation as well (and viceversa).

GCN. Building on top of the spatial interpretation of equation (2.19), Kipf & Welling [138] introduced a simplified version of ChebNet, which further limits the amount of parameters required by the filters. In particular, the authors proposed to limit the Chebyshev expansion to a degree $d = 1$, and to impose that the coefficients of the polynomials with degree 0 and 1 were just the same but with opposite sign. The convolutional layer obtained in [138] thus had the form:

$$\tilde{\mathbf{x}} = \omega + \mathbf{D}^{-0.5}\mathbf{A}\mathbf{D}^{-0.5}\mathbf{x}\omega = \left(\mathbf{I} + \mathbf{D}^{-0.5}\mathbf{A}\mathbf{D}^{-0.5}\right)\mathbf{x}\omega. \tag{2.23}$$

As it was the case for ChebNet, since matrix $\left(\mathbf{I} + \mathbf{D}^{-0.5}\mathbf{A}\mathbf{D}^{-0.5}\right)$ has eigenvalues in $[0, 2]$, repeated applications of this convolutional layer could lead to features with very large absolute values, which in turn could result in exploding gradients during training and numerical instabilities. To avoid this issue, Kipf & Welling proposed to combine the identity matrix and the adjacency matrix before the normalization took place. This *re-normalization trick* preserves the eigenvalues of the shift operator in the range $[-1, 1]$ as matrix $\bar{\mathbf{A}} = \mathbf{I}+\mathbf{A}$ is nothing more than the adjacency matrix of a new graph where a self-loop was added at each node. The convolutional layer obtained using $\bar{\mathbf{A}}$ for diffusion has now the form:

$$\tilde{\mathbf{x}} = \bar{\mathbf{D}}^{-0.5}\bar{\mathbf{A}}\bar{\mathbf{D}}^{-0.5}\mathbf{x}\omega; \tag{2.24}$$

with $\bar{\mathbf{D}} = diag(\sum_j \bar{\mathbf{A}}_{1,j}, \ldots, \sum_j \bar{\mathbf{A}}_{|\mathcal{V}|,j})$. Thanks to the particularly small amount of parameters required by this approach and the types of filters (2.24) can implement (i.e. low pass filters[8]), the solution proposed in [138] is well suited for learning on *homophilic* domains (where similar nodes tend to connect to each other and high frequencies are mostly associated with noise) with low amount of labeled data. It appeared for this, as the first GCNN able to achieve good success for transductive semi-supervised learning tasks on citation networks with extremely small labelled sets.

[8] Wu et al. [263] observed how adding self-loops to the adjacency matrix shrinks its spectrum (Theorem 1 of their paper) and makes only the eigenvalues associated with low frequencies to survive after several applications of the operator.

Model name	Message $\mathbf{m}_{j\to i}$	Update function
Spectral GCNN ([46])	$\left(\mathbf{\Phi} diag(\tilde{\mathbf{h}})\mathbf{\Phi}^T\right)_{ij} x_j$	$\sigma(\sum_{j\in V} m_{j\to i})$
ChebNet ([72])	$\sum_{k=0}^{K} T_k(\mathbf{\Delta}-\mathbf{I})_{ij} x_j \omega_k$	$\sigma(\sum_{j\in\mathcal{N}_i^{(K)}} m_{j\to i})$
GCN ([138])	$\left(\bar{A}_{ij}/\sqrt{\bar{d}_i\bar{d}_j}\right) x_j$	$\sigma(\sum_{j\in\mathcal{N}_i} m_{j\to i}\omega)$
GraphSAGE ([109])	$\left(\bar{A}_{ij}/\bar{d}_i\right) x_j$	$\sigma(x_i\omega_0 + \sum_{j\in\mathcal{N}_i} m_{j\to i}\omega_1)$
S-GCN ([263])	$\left(\bar{A}_{ij}/\sqrt{\bar{d}_i\bar{d}_j}\right) x_j$	$\sum_{j\in\mathcal{N}_i} m_{j\to i}\omega$

Table 2.1. Various GCNNs implemented with the message passing mechanism. We focused on layers receiving 1D signal as input and producing a 1D signal as output for simplicity. $\sigma(.)$ is a non-linearity and $\mathcal{N}_i^{(K)}$ is the K-hop neighborhood of node i. For GraphSAGE we use summation to implement the AGGREGATE(.) function.

GraphSAGE. In a follow up work, Hamilton et al. [109] additionally showed how the construction proposed by Kipf & Welling (and similar variations) could also be extended to prediction problems defined on large graphs via the introduction of a sampling operation prior aggregation. In short, by sampling the neighbors of a target node before convolution takes place, one can limit the computational complexity of a graph convolutional layer to a fixed $\mathcal{O}(|\mathcal{V}|S_l)$ where S_l is the number of neighbors sampled for each node at layer $l \in \{1,\dots,L\}$. In formula, a graph convolutional layer can be defined as:

$$\tilde{x}_i = x_i\omega_0 + \text{AGGREGATE}(x_j | j \in \mathcal{N}_i^{(S)})\omega_1; \tag{2.25}$$

where $\mathcal{N}_i^{(S)}$ is the sampled 1-hop neighborhood of node i at the current layer and AGGREGATE(.) is an aggregation operation such as mean, max or a Recurrent Neural Network such as LSTM [119]. In the case of RNNs, as the model processes the provided input data in a sequential manner, the authors in [109] used random permutations of the neighbors of each node to push the overall architecture to learn permutation invariant aggregation functions.

It should be noted that, while the sampling procedure introduced in [109] does allow to split the processing of an entire graph in smaller independent batches that require lower computation for a single forward pass, the overall number of operations required to process the entire domain increases from $\mathcal{O}(|\mathcal{E}|L)$ to a $\mathcal{O}(|\mathcal{V}|\prod_{l=1}^{L} S_l)$ [57]. This is due to the exponential number of neighbors that are required for producing the target nodes' embeddings over L different layers, and to the fact that, at each layer l, neighbors are sampled independently for each node whose embedding is required at layer $l+1$. Possible solutions to this problem entail sampling vertices from the whole graph in each Neural Network's layer (rather than independently sampling neighbors of previously selected nodes) [53], processing partitions of the input domain to avoid sampling all together [57, 277], or realizing diffusion only at pre-processing time to reduce the GCNN to a simple MLP (Chapter 6).

MPNNs. In the previous paragraphs we described how to generalize the convolution operator from a signal processing perspective (i.e. in terms of generalizations of the shift operator). A slightly different view was provided in [99], where the authors proposed a general formalism for implementing GCNNs that is based on the idea of *message passing*. A general graph convolutional layer can be seen as the composition of two different functions: a *message function*

$M(.)$ that determines the information that should be exchanged from one node to its neighbors, and an *update function* $U(.)$ that refines the feature of the target node once the exchange of information took place. In formula:

$$\mathbf{m}_{j\to i} = M(\mathbf{x}_i, \mathbf{x}_j, \mathbf{e}_{ij}); \qquad \tilde{\mathbf{x}}_i = U(\mathbf{x}_i, \square_{j\in\mathcal{N}_i}\mathbf{m}_{j\to i}); \tag{2.26}$$

where $\mathbf{m}_{j\to i}$ is the message sent from neighbor j to node i, $\mathbf{e}_{ij}$ are features defined over the (possibly directed) edge going from j to i and $\square_{j\in\mathcal{N}_i}$ is any permutation invariant operator[9]. As equation (2.26) operates only on the 1-hop neighborhood of a target node, multiple message passing layers are typically stacked on top of each other to allow information to spread in the graph beyond 1-hop neighborhoods. Thanks to the very flexible nature of this particular framework, many GCNNs can be seen as a special case of the work of [99] where suitable message, aggregation and update functions were selected (Table 2.1). As a result of this generality, the message passing formalism is probably today the most common way of thinking to GCNNs in the GDL community[10].

[9] In [99] the authors used summation to aggregate messages coming from the neighbors of the target node. For the sake of generality, equation (2.26) lists the construction of [259] where summation is replaced with any permutation invariant aggregation operator (e.g. mean, max, ...).

[10] For the sake of clarity, we would like to point out that, because of the broad class of functions equation (2.26) can realize, the message passing formalism as defined in this section actually admits implementations that, strictly speaking, do not generalize the convolution operation if applied over a grid (e.g. in the case where $\square_{j\in\mathcal{N}_i}$ is the max pooling operation). As a result of this, many architectures implementing a layer of the form (2.26) are often referred to as Graph Neural Networks (GNNs) rather than GCNNs in the literature. In this document we will use the term *graph convolutional layer* in a broad sense (i.e. with this including all variations of message passing), and as such we will also use the terms GCNN and GNN interchangeably.

Chapter 3

Mixture Model Neural Networks

This chapter is based on "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* Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5115–5124, 2017" (* denotes equal contribution).*

3.1 Methodology

Deep Learning on Manifolds. While in this document we focused so far on how convolution can be generalized to graphs (which was the main focus of the author), a related line of work in the GDL community concerns how convolution can be extended to signals defined on manifolds (e.g. 3D shapes). Let $\mathcal{X}$ be a d-dimensional manifold, possibly with boundary $\partial\mathcal{X}$. Around point $x \in \mathcal{X}$, the manifold is homeomorphic to a d-dimensional Euclidean space referred to as the *tangent space* and denoted by $T_x\mathcal{X}$. An inner product $\langle\cdot,\cdot\rangle_{T_x\mathcal{X}} : T_x\mathcal{X} \times T_x\mathcal{X} \to \mathbb{R}$ depending smoothly on x is called the *Riemannian metric*. In shape analysis, 3D shapes are modeled as 2-dimensional manifolds (surfaces), representing the boundaries of 3D volumes. In [174], Masci et al. introduced a generalization of CNNs (Geodesic CNN - herein referred to as GeoCNNs) on 2-dimensional manifolds, based on the definition of a local charting procedure in geodesic polar coordinates [142]. Such a construction, named the *patch operator*

$$(D(x)f)(\rho,\theta) = \int_{\mathcal{X}} w_{\rho,\theta}(x,y) f(y) dy \tag{3.1}$$

maps the values of the function $f : \mathcal{X} \to \mathbb{R}^Q$ in a neighborhood of point $x \in \mathcal{X}$ into the local polar coordinates ρ, θ via an aggregation procedure. Here dy denotes the area element induced by the Riemannian metric, and $w_{\rho,\theta}(x,y)$ is the activation of a weighting function (here computed for neighbor y of x) localized around ρ, θ (see examples in Figure 3.1). $D(x)f$ can be regarded as a *patch* on the manifold and the *geodesic convolution* operator introduced by the authors

$$(f \star g)(x) = \max_{\Delta\theta\in[0,2\pi)} \int_0^{2\pi}\int_0^{\rho_{\max}} g(\rho,\theta+\Delta\theta)(D(x)f)(\rho,\theta) d\rho d\theta \tag{3.2}$$

can be thought of as matching a (learnable) template $g(\rho,\theta)$ with the extracted patch at each point, where the maximum is taken over all possible rotations of the template in order to resolve the origin ambiguity in the angular coordinate.

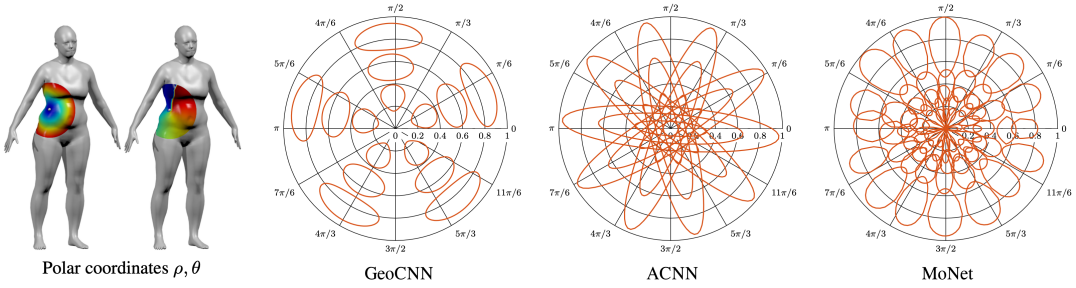


Figure 3.1. Left: intrinsic local polar coordinates ρ, θ. Right: patch operator weighting functions $w_i(\rho, \theta)$ used in different generalizations of convolution on manifolds.

In a subsequent work, Boscaini et al. [38] extended the work proposed in [174] using *anisotropic heat kernels* $h_{\alpha\theta t}(x, y)$ (which describe the amount of heat that is transferred from point x to point y at time t) as possible weighting functions that can be used for the construction of the patch operator (ACNN, Figure 3.1). The main novelty of this work was to consider the direction of maximum curvature of the manifold at x as a meaningful reference to orientate the patch operator. θ in the anisotropic heat kernel is indeed an angle in the tangent plane w.r.t. such direction, while α describes the elongation of the kernel (i.e. the degree of anisotropy along the direction defined by θ). Defining a canonical orientation of the kernel removed the need of max pooling the filter activations over all possible orientations of the filter, which in turn produced patch operators that were more robust and more efficient to compute.

MoNet. The main contribution of our work is a generic spatial-domain framework for deep learning on non-Euclidean domains which can be used for processing both signals defined on graphs and manifolds. We use x to denote, depending on context, a point on a manifold or a vertex of a graph, and consider points $y \in \mathcal{N}_x$ in the neighborhood of x. With each such y, we associate a d-dimensional vector of *pseudo-coordinates* $\mathbf{u}(x, y)$. In these coordinates, we define a weighting function (kernel) $\mathbf{w}_{\boldsymbol{\Theta}}(\mathbf{u}) = (w_1(\mathbf{u}), \dots, w_J(\mathbf{u}))$, which is parametrized by some learnable parameters $\boldsymbol{\Theta}$. The patch operator can therefore be written in the following general form:

$$D_j(x)f = \sum_{y \in \mathcal{N}_x} w_j(\mathbf{u}(x, y)) f_y, \qquad j = 1, \dots, J; \tag{3.3}$$

where the summation should be interpreted as an integral in the case we deal with a continuous manifold, and J represents the dimensionality of the extracted patch (namely the amount of weighting functions that define kernel $\mathbf{w}_{\boldsymbol{\Theta}}$). A spatial generalization of convolution on non-Euclidean domains is then given by:

$$(f \star g)(x) = \sum_{j=1}^{J} g_j D_j(x) f. \tag{3.4}$$

The two key choices in our construction are the pseudo-coordinates $\mathbf{u}$ and the weight functions $\mathbf{w}(\mathbf{u})$. Table 3.1 shows that other deep learning methods (including classic CNNs on Euclidean domains, GCN on graphs and GeoCNN/ACNN on manifolds) can be obtained as particular settings of our framework with appropriate definition of $\mathbf{u}$ and $\mathbf{w}(\mathbf{u})$. GeoCNN and ACNN boil down in particular to using Gaussian kernels on local polar geodesic coordinates ρ, θ on

Table 3.1. Several GDL methods as a particular setting of the MoNet framework. x denotes the reference point (center of the patch) and y a point within the patch. $\mathbf{e}$ denotes Euclidean coordinates on a regular grid. $\bar{\alpha}, \bar{\sigma}_\rho, \bar{\sigma}_\theta$ and $\bar{\mathbf{u}}_j, \bar{\theta}_j, j = 1, \dots, J$ denote fixed parameters of the weight functions. $\mathbf{r}(x,y) = (\cos(\theta(x,y)), \sin(\theta(x,y))$. $\mathbf{R}_{\bar{\theta}_j}$ is a rotation matrix defined by angle $\bar{\theta}_j$. $\mathbf{w}$ is a vector of learnable weights.

Method	Pseudo-coordinates	$\mathbf{u}(x,y)$	Weight function $w_j(\mathbf{u})$
CNN	Euclidean	$\mathbf{e}(y) - \mathbf{e}(x)$	$\delta(\mathbf{u} - \bar{\mathbf{u}}_j)$
GeoCNN	Polar geodesic	$\rho(x,y), \theta(x,y)$	$\exp(-\frac{1}{2}(\mathbf{u}-\bar{\mathbf{u}}_j)^T \begin{pmatrix} \bar{\sigma}_\rho^2 & \\ & \bar{\sigma}_\theta^2 \end{pmatrix}^{-1} (\mathbf{u}-\bar{\mathbf{u}}_j))$
ACNN	Polar geodesic	$\rho(x,y) \cdot \mathbf{r}(x,y)$	$\exp(-\frac{1}{2}\mathbf{u}^T \mathbf{R}_{\bar{\theta}_j} \begin{pmatrix} \bar{\alpha} & \\ & 1 \end{pmatrix} \mathbf{R}_{\bar{\theta}_j}^T \mathbf{u})$
GCN	Vertex degree	$\deg(x), \deg(y)$	$\left(1 - \lvert 1 - \frac{1}{\sqrt{u_1}} \rvert\right)\left(1 - \lvert 1 - \frac{1}{\sqrt{u_2}} \rvert\right)$
GAT	Node Features	$\mathbf{x}(y), \mathbf{x}(x)$	$\exp\left((\mathbf{x}(y) \lVert \mathbf{x}(x))^T \mathbf{w}\right)$

a manifold, and GCN can be interpreted as applying a triangular kernel on pseudo-coordinates given by the degree of the graph vertices. Differently from previous works, in MoNet rather than using fixed handcrafted weight functions (see [39, 174]) we consider parametric kernels with learnable parameters. In particular, a convenient choice is:

$$w_j(\mathbf{u}) = e^{-\frac{1}{2}(\mathbf{u}-\boldsymbol{\mu}_j)^T \boldsymbol{\Sigma}_j^{-1} (\mathbf{u}-\boldsymbol{\mu}_j)}; \tag{3.5}$$

where $\boldsymbol{\Sigma}_j$ and $\boldsymbol{\mu}_j$ are learnable $d \times d$ and $d \times 1$ covariance matrix and mean vector of a Gaussian kernel, respectively. Equations (3.3)–(3.4) can in this case be interpreted as a *gaussian mixture model* (GMM), where for a specific neighbor y a linear combination of J different gaussians (evaluated in $\mathbf{u}(x,y)$) is computed:

$$\begin{aligned}
(f \star g)(x) &= \sum_{j=1}^{J} g_j D_j(x) f \\
&= \sum_{j=1}^{J} g_j \left(\sum_{y \in \mathcal{N}_x} w_j(\mathbf{u}(x,y)) f_y \right) \\
&= \sum_{y \in \mathcal{N}_x} \underbrace{\left(\sum_{j=1}^{J} g_j \, e^{-\frac{1}{2}(\mathbf{u}(x,y)-\boldsymbol{\mu}_j)^T \boldsymbol{\Sigma}_j^{-1} (\mathbf{u}(x,y)-\boldsymbol{\mu}_j)} \right)}_{\text{Gaussian Mixture Model}} f_y .
\end{aligned} \tag{3.6}$$

From a different perspective, as our parametric kernel function allow the model to directly learn which neighbors should be used in the implementation of the filters dependently on the positions that these assume in the pseudo-coordinate space (thus effectively implementing a form of attention), our work can also be interpreted as the first attention-based [251] Graph Convolutional Neural Network that was ever introduced, and laid the foundations for subsequent works in this direction (Section 3.3). Veličković et al. [252] showed in particular how using the features of the nodes as pseduo-coordinates and replacing the gaussian kernel with a linear layer followed by a softmax activation function (Graph Attention Networks), even better

performance than the ones we originally achieved in our paper could be obtained for document classification tasks, while essentially retaining the same construction (Table 3.2).

3.2 Results

Shape correspondence. The first application of MoNet that we introduce is learning dense intrinsic correspondence between collections of 3D shapes represented as discrete manifolds. Correspondence between shapes can be casted as a labelling problem, where one tries to label each vertex of a given query shape $\mathcal{X}$ with the index of a corresponding point on some reference shape $\mathcal{Y}$ [217, 174, 38]. Let m denote the number of vertices in $\mathcal{Y}$. For a point x on a query shape, the last layer of a network produces an m-dimensional output that is interpreted as a probability distribution on $\mathcal{Y}$ (the probability of x mapped to y). Learning is typically done minimizing the standard log loss [38].

In our experiments, we reproduced verbatim the setting of [174, 38] on the FAUST humans dataset [37], comparing to the methods reported therein. The dataset consists of 100 meshes representing 10 different poses for 10 different subjects with exact ground-truth correspondence. Each shape was represented as a mesh with 6890 vertices; the first subject in first pose was used as the reference. For all the shapes, point-wise 544-dimensional SHOT descriptors (local histogram of normal vectors [248]) were used as input data. As our goal is to highlight the increased performance our new patch operator allows to achieve with respect to prior art, we implemented a MoNet architecture with 3 convolutional layers, replicating the architectures presented in [174] and [38]. First 80 subjects in all the poses were used for training (800 shapes in total); the remaining 20 subjects were used for testing. The output of the network was refined using the intrinsic Bayesian filter [254] in order to remove some local outliers. Correspondence quality was evaluated using the Princeton benchmark [135], plotting the percentage of matches that are at most r-geodesically distant from the groundtruth correspondence on the reference shape. For comparison, we report the performance of Geodesic CNN [174], ACNN [38], as well as blended maps [135], random forests [217] and ADD [39]. Figure 3.2 (left) depicts the evaluation results, showing that MoNet significantly outperforms the competing approaches. In particular, close to 90% of points have *zero* error, and for 99% of the points the error is below 4cm. To get a better idea of the correspondence quality, Figure 3.3 shows the point-wise geodesic correspondence error of our method. Figure 3.4 visualizes the obtained correspondence using texture transfer.

Range maps. On top of the above, we repeated the shape correspondence experiment on range maps synthetically generated from FAUST meshes. For each subject and pose, we produced 10 rangemaps in 100×180 resolution, covering shape rotations around the z-axis with increments of 36 degrees (total of 1,000 range maps), keeping the groundtruth correspondence. We used MoNet architecture with 3 convolutional layers and local SHOT descriptors as input data. Training and testing set splitting was done as previously. Figure 3.2 (right) shows the quality of correspondence computed using the Princeton protocol. For comparison, we show the performance of a standard Euclidean CNN in equivalent architecture (3 convolutional layers) applied on raw depth values and on SHOT descriptors. Our approach clearly shows a superior performance. Figure 3.5 shows the point-wise geodesic correspondence error. Figure 3.6 shows a qualitative visualization of correspondence using similar color code for corresponding vertices.

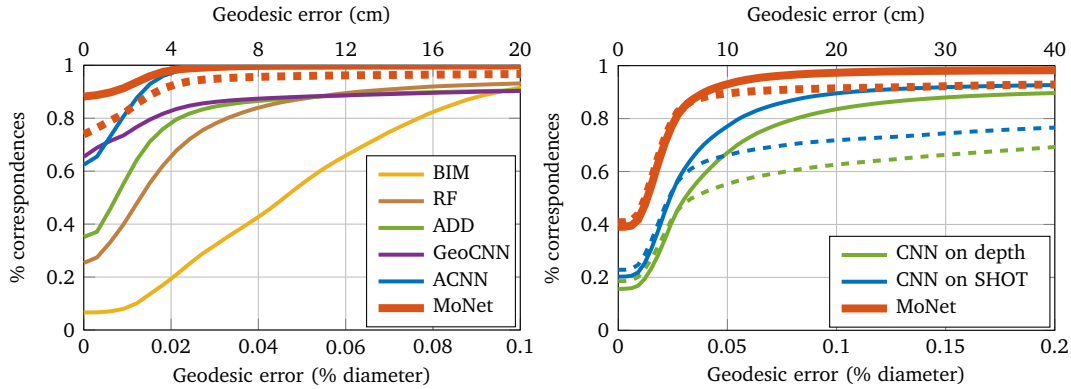


Figure 3.2. (Left) Shape correspondence quality obtained by different methods on the FAUST humans dataset. (Right) Shape correspondence quality obtained by different methods on FAUST range maps. The raw performance is shown as dotted curve in both plots.

Figure 3.3. Pointwise error (geodesic distance from groundtruth) of MoNet on the FAUST humans dataset. For visualization clarity, the error values are saturated at 7.5% of the geodesic diameter, which corresponds to approximately 15 cm. Hot colors represent large errors.

Figure 3.4. Examples of correspondence on the FAUST humans dataset obtained by the proposed MoNet method. Shown is the texture transferred from the leftmost reference shape to different subjects in different poses by means of our correspondence.

Document Classification. Moving our attention to graphs, one of the most classic applications of GCNNs in the literature is the task of document classification in citation networks [270, 138, 184, 252]. Provided a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where vertices are documents and edges are citations among them, the problem is to predict the category y_i of a document i based on i's text and the text of i's similar documents (which can be described as the documents that fall in a

Figure 3.5. Pointwise error (geodesic distance from groundtruth) of MoNet on FAUST range maps. For visualization clarity, the error values are saturated at 7.5% of the geodesic diameter, which corresponds to approximately 15 cm. Hot colors represent large errors.

Figure 3.6. Visualization of correspondence on FAUST range maps as color code (corresponding points are shown in the same color). Full reference shape is shown on the left. Bottom row show examples of additional shapes from SCAPE and TOSCA datasets.

Method	Cora	PubMed
ManiReg [25]	59.5%	70.7%
SemiEmb [261]	59.0%	71.1%
LP [283]	68.0%	63.0%
DeepWalk [202]	67.2%	65.3%
Planetoid [270]	75.7%	77.2%
GCN [138]	**81.6** ± 0.4%	**78.7** ± 0.3%
MoNet [184]	**81.7** ± 0.5%	**78.8** ± 0.4%
GAT [252]	**83.0** ± 0.7%	**79.0** ± 0.3%

Table 3.2. Vertex classification accuracy on the Cora and PubMed datasets following the split suggested in [270]. GCNNs are listed at the bottom of the table. In **black** / **red** / **blue** performance of the 1st / 2nd / 3rd best model on a given dataset.

neighborhood of i of radius r, $\mathcal{N}_i^{(r)}$). To contain the complexity of the problem, the text of each document is typically described with a fixed length representation $\mathbf{x}_i$, which can be interpreted

as signals defined on the vertices of $\mathcal{G}$. A typical choice for $\mathbf{x}_i$ is a bag-of-words representation [270], where each document is represented as a multi-set of its own words. Each entry k of $\mathbf{x}_i$ corresponds to a word and (excluding any possible normalization) the value $x_{i,k}$ matches the number of times that word k appears in i.

Training a GCNN for document classification in citation networks works slightly differently dependently on the setting of the problem. In the *transductive learning* setting (which has historically been the most common scenario for evaluating GCNNs), all documents of $\mathcal{G}$ (with their features) are available at training time but only a fraction of them have labels that we can use for training. In the *inductive learning* setting, test samples are not available at training time and are added (together with their features and edges) to $\mathcal{G}$ only when inference is performed [109]. With our MoNet architecture we focused only on the *transductive learning* setting. $\mathbf{\Theta}$ is the set of parameters of the chosen GCNN and it is optimized by minimizing (through gradient descent) a classic supervised loss (e.g. log loss) on the subset of labelled nodes available at training time.

Table 3.2 shows performance (measured as accuracy on the test set) of the proposed MoNet architecture when compared to previous approaches (plus the aforementioned Graph Attention Network, which followed our work) on the CORA and PubMed citation networks, using the so-called *planetoid split* of these datasets [270]. The training sets consisted of 20 samples per class (i.e. 140 training samples for CORA and 60 for PubMed); the validation and test sets consisted of 500 and 1000 disjoint vertices. For MoNet, we used the degrees of the nodes as the input pseudo-coordinates $\mathbf{u}(x,y) = (\frac{1}{\sqrt{\deg(x)}}, \frac{1}{\sqrt{\deg(y)}})^T$; these coordinates underwent an additional transformation in the form of a fully-connected neural network layer $\tilde{\mathbf{u}}(x,y) = \tanh(\mathbf{W}\mathbf{u}(x,y) + \mathbf{b})$ before being fed to the Gaussian kernel, where the $r \times 2$ matrix $\mathbf{W}$ and $r \times 1$ vector $\mathbf{b}$ were also learned (we used $r = 2$ for Cora and $r = 3$ for PubMed). We trained MoNet with $\mathbb{L}_2$-regularization to contain overfitting (regularization coefficient $\gamma = 10^{-2}$ and $\gamma = 5 \times 10^{-2}$ for Cora and PubMed, respectively). Adam [137] was used for training the model on both datasets, and the log loss was picked as our training objective. As we can see, our Mixture Model Neural Network achieves comparable performance to the solution proposed by Kipf & Welling on both datasets, learning to reproduce comparable filters from the provided pseudo-coordinates. Additionally, using node-features in the attention mechanism (as highlighted in [252]) appears beneficial for further increasing performance.

3.3 Discussion

The attentive reader might have noticed that the original paper describing our MoNet framework was presented back in 2017, and the state of the art naturally evolved since then (e.g. see Table 3.3 for the performance of a series of more recent methods on the Cora and PubMed datasets). Following up from our own work, several other researchers dedicated their efforts to develop new variants of attention-based models that could allow to maximize performance on a given prediction task. In this direction, two main classes of architectures can be generally identified in the literature today: *attention-based local GCNNs* (which propagate information among different nodes following the connectivity of the provided domain, and use a form of attention to condition the diffusion process) and *Graph Transformer Networks* (which decouple the computational graph from the input one by using positional / structural encodings and a global attention mechanism).

Table 3.3. Performance of multiple methods on the Planetoid split [270] of Cora and PubMed. Results taken from [88], and complemented with the ones reported in Table 3.2, Table 5.2 and in [54, 32, 35, 84, 95, 152]. Methods are sorted in ascending order of average performance on Cora. In **black** / **red** / **blue** performance of the 1st / 2nd / 3rd best model on a given dataset.

Method	Cora	PubMed
GraphSAGE [109]	78.9 ± 0.8%	77.8 ± 0.6%
S-GCN [263]	81.0 ± 0.0%	78.9 ± 0.0%
FastGCN [53]	81.4 ± 0.5%	77.6 ± 0.5%
GCN [138]	81.6 ± 0.4%	78.7 ± 0.3%
MoNet [184]	81.7 ± 0.5%	78.8 ± 0.4%
CayleyNet [153]	81.9 ± 0.7%	-
MixHop [16]	81.9 ± 0.4%	80.8 ± 0.6%
G^3NN [166]	82.5 ± 0.2%	77.9 ± 0.4%
GAT [252]	83.0 ± 0.7%	79.0 ± 0.3%
JKNet [267]	83.3%	79.2%
ARMA [32]	83.4 ± 0.6%	78.9 ± 0.3%
VBAT [73]	83.6 ± 0.5%	79.9 ± 0.4%
GMNN [210]	83.7%	**81.8%**
APPNP [97]	83.8 ± 0.3%	79.7 ± 0.3%
GraphMix [253]	83.9 ± 0.6%	**81.0 ± 0.6%**
AEROGNN [152]	83.9 ± 0.5%	80.59 ± 0.5%
FAGCN [35]	84.1 ± 0.5%	79.4 ± 0.3%
GraphNAS [96]	84.2 ± 1.0%	79.6 ± 0.4%
Graph U-Net [95]	84.4 ± 0.6%	79.6 ± 0.2%
DAGNN [95]	84.4 ± 0.5%	80.5 ± 0.5%
ωGAT [84]	84.8%	**81.8%**
GRAND [88]	**85.4 ± 0.4%**	**82.7 ± 0.6%**
GCNII [54]	**85.5 ± 0.5%**	80.3 ± 0.4%
ωGCN [84]	**85.9%**	81.1%

Attention-based local GCNNs. Broadly speaking, attention-based local GCNNs leverage an attention mechanism to achieve one of two different goals:

1. learning how to diffuse information on a given graph by inferring the relevance of neighboring nodes [184, 252, 42, 35, 84];
2. learning how information that was harvested from neighborhoods of different sizes should be combined together to produce meaningful descriptors [123, 267, 159].

In the first category of approaches, besides the MoNet framework and GATs that we mentioned earlier in this chapter, we find approaches such as *GATv2* [42] (which fixes an issue of the attention layer used in GAT that effectively made the ranking of attention scores to be independent on the features of the query node), *FAGCN* [35] (which uses an attention mechanism capable of producing scores in $[-1, 1]$, in order to leverage both the information of similar and dissimilar nodes in the construction of the filters), and *ωGAT* [84] (which learns per-channel smoothing /

sharpening operators, while requiring only one projection over a single attention matrix). The second category of approaches addresses instead an orthogonal direction from what discussed in the previous sections. More in detail, in [267] it was observed how the influence distribution[1] of a given node i changes for the same number of diffusion steps dependently on the structure of i's neighborhood. For example, if i is a node situated at the core of a given community, only few diffusion steps could be needed to reach most of the nodes belonging to such community. If on the other hand, i lies on the boundary of the community and it is connected to it through a tree-like neighborhood with bounded (and small) tree-width, more iterations could be required to reach a comparable number of community nodes [267]. With this idea in mind, Xu et al. defined *Jumping Knowledge Networks* (JK-Nets) [267], a family of GCNNs with convolutional layers that resembles the skip-connection module introduced in *DenseNets* [123]. The main intent of the paper was in this case to realize a GCNN with an *adaptable* radius of operation, that could be directly inferred dependently on the target node. To do so, JK-Nets extract a series of neighborhood descriptors with different levels of locality (via a sequence of convolutional layers that are stacked on top of each other), and learn how to combine them via a layer aggregation mechanism. An attention layer was, in this case, one of the aggregation mechanisms used in [267] to combine representations extracted at different scales. In a follow up work, Liu et al. further introduced *Deep Adaptive Graph Neural Network* (DAGNN) [159], a particular version of the framework presented in [267] (and similar in spirit to GPR-GNN [58] that we will discuss in Section 5.4), which decouples feature transformation from feature propagation, and achieves (in virtue of its parameter efficiency) good performance even when many diffusion steps are realized as part of the model.

Recently, there has also been some effort to unify in a single framework both the edge attention mechanism introduced in [184, 252] and the hop attention approach used in [267, 159]. In [152], Lee at al. introduced *AERO-GNN*, an attention based GCNN that learns to infer both the relevance of a given neighbor to a target node, as well as the importance of a particular diffusion step, from the combined node embeddings of the previous layers. Thanks to the combination of both approaches, AERO-GNN outperforms previously presented GCNNs, in the authors' experimental evaluation, on a variety of different datasets.

Graph Transformers. An alternative class of approaches that gained popularity in the last years is represented by *Graph Transformer Networks* (GTs) [82, 271, 211, 175, 193]. Inspired from the success that Transformers achieved in Natural Language Processing [251], Graph Transformers use positional (or structural) encodings to describe the position of a node[2] in the graph (or the structure that surrounds this), and apply an attention mechanism between each pair of nodes to let information flow in the domain. Inspired from Spectral Clustering, eigenvectors of the graph Laplacian are for instance a popular choice of positional encodings; the degree of nodes that we used in our MoNet framework can instead be understood as a form of structural encoding. This specific construction decouples the computational graph of the model from the one of the input graph, and allows to avoid issues such as *under-reaching* [23] and (potentially) *over-squashing* [20] that affect traditional GCNNs.

Under-reaching is generally defined as the inability of a model to reach information that is too far from a given target node. For GCNNs with 1-hop filters and L layers, the model can't

[1] The influence distribution of a node i can be informally defined as a measure depicting how much a change in the input features of neighbor j affects the final representation of i for a given model [267].

[2] Please note, for the sake of consistency with what described previously in this chapter, we consider as tokens in input to the Transformer architectures only nodes. Edges and sub-graphs can however be used as tokens as well [193].

for instance reach information which is further than L hops apart. Graph Tansformer Networks do not suffer from such issue, as every node can attend to any other.

Over-squashing is defined instead as the collapse of information coming from exponentially many nodes in a finite feature vector. This phenomenon generally affects GCNNs, as a model with L layers and 1-hop filters aggregates information coming from up to $O(d_{max}^L)$ neighbors, in order to make predictions (d_{max} is here the maximum node-degree in the given graph). While Graph Transformers do not provide a solution that allows to effectively aggregate information coming from large amounts of nodes in feature vectors of limited size, in problems exhibiting long-range dependencies [20, 83] they allow the model to focus only on the (hopefully) few distant neighbors that are relevant for a target node, and thus reduce the amount of information that needs to be aggregated together [193].

Despite the nice theoretical properties showed by Graph Transformers, it should be noted that such architectures are not necessarily the best choice for all prediction problems. Indeed, while the global attention mechanism allows to access information located in any possible region of the graph, it also removes the local inductive bias of traditional GCNNs, which can prove beneficial to achieve good generalization (e.g. in social networks, it is reasonable to assume that only the information localized in a small neighborhood of a given user is relevant for predicting their interests). On top of this, including edge features in the node refinement process requires a materialization of the full attention matrix (as linear transformers such as the Performer [61] cannot be used), which in turn forces a $\mathcal{O}(|\mathcal{V}|^2)$ time and space complexity. To overcome these limitations and achieve a best of both world solution, hybrid (local) GCNN + GT architectures appeared on the scene in the last few years [211, 175]. Thanks to the combination of both approaches, such methods allow to efficiently exploit edge features in the computation of the final node embeddings (via the local message-passing module), they provide a local inductive bias (again thanks to the presence of local message-passing layers), and they allow to capture long-range dependencies (via the transformer network).

Chapter 4

Recurrent Multi-Graph Convolutional Neural Networks

This chapter is based on "Federico Monti, Michael Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent multi-graph neural networks. Advances in neural information processing systems*, 30, 2017*".

4.1 Introduction

Being able to effectively recommend items of interests to users plays a major role for the success of virtually any e-commerce (e.g. Amazon), media / entertainment provider (e.g. Netflix, Spotify) and social media (e.g. Facebook, Twitter / X, LinkedIn). Mathematically, a recommendation task can be posed as a *matrix completion* problem [49], where the columns and rows of a sparse matrix $\mathbf{X}$ represent users and items, respectively, and matrix values represent scores determining whether a user would like an item or not. Given a small set of known elements of $\mathbf{X}$, the goal is to fill in the rest.

In the past years, there have been several attempts to incorporate geometric structure into matrix completion problems [164, 131, 212, 145] (e.g. in the form of regularizers aimed at minimizing the Dirichlet energy that columns and rows of $\mathbf{X}$ show on users and items similarity graphs [131]). In [185], we introduced a new multi-graph CNN architecture (MGCNN) that generalizes the approach outlined in [72] to multiple graphs. This new architecture is able to extract local stationary patterns from signals defined over multiple graphs (e.g. a matrix of user-item scores defined over a user-user and an item-item similarity network) and, thanks to the addition of a recurrent neural network, to fill in possible gaps in the input signal by learning a suitable diffusion process.

Please note, while using GCNNs for matrix completion was a novel application of this class of methods at the time our paper was presented, due to the core methodological innovation of our work (i.e. the first GCNN able to process signals defined over multiple graphs, at least to the best of our knowledge), we decided to illustrate both the method and the application we targeted with it in this first part of the manuscript. This avoided spreading related content over different chapters, and provides the required background for analyzing some of the results illustrated in Chapter 5.

4.2 Methodology

Multi-graph CNN. Provided a generic matrix $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}_r| \times |\mathcal{V}_c|}$, which is defined over the vertices of two different graphs $\mathcal{G}_r = (\mathcal{V}_r, \mathcal{E}_r)$ and $\mathcal{G}_c = (\mathcal{V}_c, \mathcal{E}_c)$, each of these equipped with its respective normalized Laplace operator $\tilde{\mathbf{\Delta}}_r = \tilde{\mathbf{\Phi}}_r \tilde{\mathbf{\Lambda}}_r \tilde{\mathbf{\Phi}}_r^T$ and $\tilde{\mathbf{\Delta}}_c = \tilde{\mathbf{\Phi}}_c \tilde{\mathbf{\Lambda}}_c \tilde{\mathbf{\Phi}}_c^T$. Similarly to how a multi-dimensional Fourier transform is defined in 2D (i.e. where we transform the rows of an image first and then its columns), a bi-graph Fourier transform can be defined as:

$$\hat{\mathbf{X}} = \tilde{\mathbf{\Phi}}_r^T \mathbf{X} \tilde{\mathbf{\Phi}}_c. \tag{4.1}$$

As it is the case for signals defined over a single graph, convolution with a multi-graph filter can be realized in the spectral domain simply multiplying the spectrum of matrix $\mathbf{X}$ with a matrix of spectral coefficients $\hat{\mathbf{Y}}$, and then projecting the signal back to its spatial domain:

$$\tilde{\mathbf{X}} = \mathbf{X} \star \mathbf{Y} = \tilde{\mathbf{\Phi}}_r \left(\hat{\mathbf{X}} \circ \hat{\mathbf{Y}} \right) \tilde{\mathbf{\Phi}}_c^T. \tag{4.2}$$

As we have seen for signals defined over a single graph, parametrizing multi-graph filters directly with spectral multipliers $\hat{\mathbf{Y}}$ yields however a number of parameters that is linear in the number of entries of matrix $\mathbf{X}$ (i.e. a $\mathcal{O}(|\mathcal{V}_r||\mathcal{V}_c|)$), which in turn can lead to overfitting. Following up on the work of Defferrard et al. [72], one possibility to overcome this issue is to impose that the spectral response of our filters corresponds to a function of both the row and column graph frequencies (i.e. $\hat{\mathbf{Y}}_{k,k'} = \tau(\tilde{\lambda}_{c,k}, \tilde{\lambda}_{r,k'})$). In particular, using Chebyshev polynomials of degree up to p[1], a spectral coefficient $\tau_{\mathbf{\Theta}}(\tilde{\lambda}_c, \tilde{\lambda}_r)$ can be defined as:

$$\tau_{\mathbf{\Theta}}(\tilde{\lambda}_c, \tilde{\lambda}_r) = \sum_{j,j'=0}^{p} \theta_{jj'} T_j(\tilde{\lambda}_c - 1) T_{j'}(\tilde{\lambda}_r - 1). \tag{4.3}$$

Similarly to what it was the case for ChebNet, such filters are parametrized by a $(p+1) \times (p+1)$ matrix of coefficients $\mathbf{\Theta} = (\theta_{jj'})$, which makes them independent on the size of the provided domains. The application of a multi-graph filter of the form (4.3) to matrix $\mathbf{X}$ can then be written as:

$$\tilde{\mathbf{X}} = \sum_{j,j'=0}^{p} \theta_{jj'} T_j(\tilde{\mathbf{\Delta}}_r - \mathbf{I}) \mathbf{X} T_{j'}(\tilde{\mathbf{\Delta}}_c - \mathbf{I}); \tag{4.4}$$

which incurs a $\mathcal{O}(|\mathcal{V}_r||\mathcal{V}_c|)$ computational complexity. If a tensor of q' input channels is provided as input instead of a singular matrix, a multi-graph convolutional layer can be realized using the parametrization of filters defined in (4.4), and applying one filter for each of the (q, q') output and input features of the layer:

$$\tilde{\mathbf{X}}_l = \sigma\left(\sum_{l'=1}^{q'} \mathbf{X}_{l'} \star \mathbf{Y}_{ll'} \right) = \sigma\left(\sum_{l'=1}^{q'} \sum_{j,j'=0}^{p} \theta_{jj',ll'} T_j(\tilde{\mathbf{\Delta}}_r - \mathbf{I}) \mathbf{X}_{l'} T_{j'}(\tilde{\mathbf{\Delta}}_c - \mathbf{I}) \right), \quad l = 1, \dots, q; \tag{4.5}$$

$\sigma(.)$ is here a non-linearity (e.g. ReLU, tanh, sigmoid, ...). We name such an architecture a *Multi-Graph Convolutional Neural Network* (MGCNN).

[1] For simplicity, we use the same degree p for row- and column frequencies.

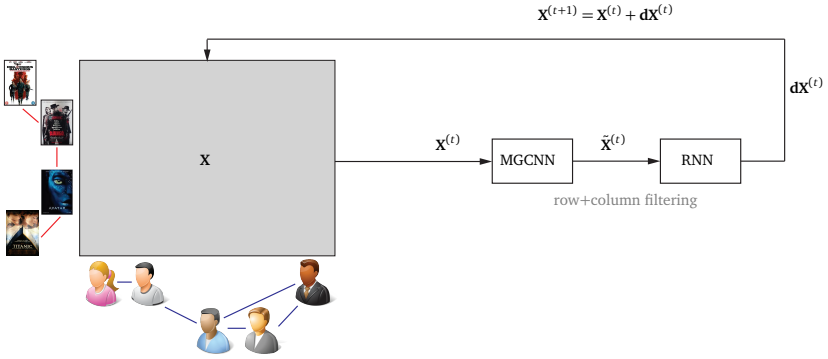


Figure 4.1. Recurrent MGCNN (RMGCNN) architecture using the full matrix completion model and operating simultaneously on the rows and columns of the matrix $\mathbf{X}$. Learning complexity is $\mathcal{O}(|\mathcal{V}_r||\mathcal{V}_c|)$.

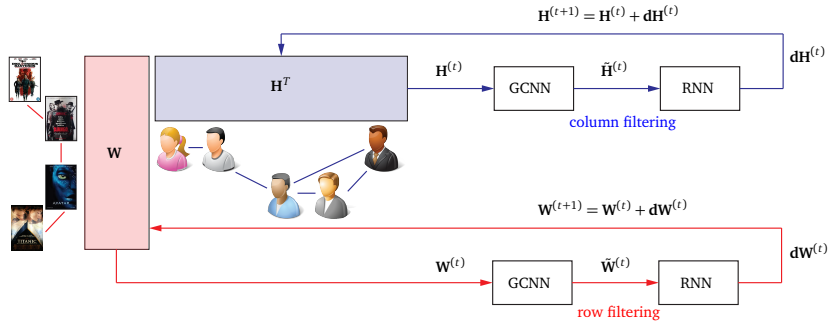


Figure 4.2. Separable Recurrent MGCNN (sRMGCNN) architecture using the factorized matrix completion model and operating separately on the rows and columns of the factors $\mathbf{W}$, $\mathbf{H}^T$. Learning complexity is $\mathcal{O}(|\mathcal{V}_r| + |\mathcal{V}_c|)$.

Separable convolution. As MGCNN layers require a $\mathcal{O}(|\mathcal{V}_r||\mathcal{V}_c|)$ operations to implement a convolutional filter, one may wonder whether a more efficient solution can be implemented to more easily process significantly large matrices. A simplification of what we presented above can be obtained considering a low-rank factorization of matrix $\mathbf{X} = \mathbf{W}\mathbf{H}^T$ (which is a popular assumption for matrix completion problems [236, 143, 164, 269, 212, 28]), and applying two independent 1D convolutional layers on the respective graph of each factor:

$$\tilde{\mathbf{w}}_l = \sum_{j=0}^{p} \theta_j^{(r)} T_j(\tilde{\mathbf{\Delta}}_r)\mathbf{w}_l, \qquad \tilde{\mathbf{h}}_l = \sum_{j'=0}^{p} \theta_{j'}^{(c)} T_{j'}(\tilde{\mathbf{\Delta}}_c)\mathbf{h}_l, \qquad l = 1, \dots, d. \tag{4.6}$$

Here, $\mathbf{w}_l, \mathbf{h}_l$ denote the lth columns of factors $\mathbf{W}$, $\mathbf{H}$ (which can be seen as signals defined over the row and column graph of $\mathbf{X}$), $d \ll \min(|\mathcal{V}_r|, |\mathcal{V}_c|)$ is the number of columns of $\mathbf{W}$ and $\mathbf{H}$, and $\boldsymbol{\theta}^{(r)} = (\theta_0^{(r)}, \dots, \theta_p^{(r)})$ and $\boldsymbol{\theta}^{(c)} = (\theta_0^{(c)}, \dots, \theta_p^{(c)})$ are the parameters of the row- and column- filters (a total of $2(p+1) = \mathcal{O}(1)$). Application of such filters to $\mathbf{W}$ and $\mathbf{H}$ incurs only a $\mathcal{O}(|\mathcal{V}_r| + |\mathcal{V}_c|)$ operations, thus dramatically reducing the cost of processing the information contained in $\mathbf{X}$. As it was the case for ChebNet, convolutional layers implemented with (4.6) take the form:

Algorithm 1 (RMGCNN)

Input: matrix $\mathbf{X}^{(0)} \in \mathbb{R}^{|\mathcal{V}_r|\times|\mathcal{V}_c|}$ containing initial values

1: **for** $t = 0 : T$ **do**
2: Apply the Multi-Graph CNN (4.5) on $\mathbf{X}^{(t)}$ producing an $|\mathcal{V}_r| \times |\mathcal{V}_c| \times q$ output $\tilde{\mathbf{X}}^{(t)}$.
3: **for** all elements (i, j) **do**
4: Apply RNN to q-dim $\tilde{\mathbf{x}}_{ij}^{(t)} = (\tilde{x}_{ij1}^{(t)}, \ldots, \tilde{x}_{ijq}^{(t)})$ producing incremental update $dx_{ij}^{(t)}$
5: **end for**
6: Update $\mathbf{X}^{(t+1)} = \mathbf{X}^{(t)} + \mathbf{dX}^{(t)}$
7: **end for**

Algorithm 2 (sRMGCNN)

Input: factor $\mathbf{H}^{(0)} \in \mathbb{R}^{|\mathcal{V}_c|\times d}$ and factor $\mathbf{W}^{(0)} \in \mathbb{R}^{|\mathcal{V}_r|\times d}$ of matrix $\mathbf{X}^{(0)}$

1: **for** $t = 0 : T$ **do**
2: Apply the Graph CNN (4.7) on $\mathbf{H}^{(t)}$ producing an $|\mathcal{V}_c| \times q$ output $\tilde{\mathbf{H}}^{(t)}$.
3: **for** $j = 1 : |\mathcal{V}_c|$ **do**
4: Apply RNN to q-dim $\tilde{\mathbf{h}}_j^{(t)} = (\tilde{h}_{j1}^{(t)}, \ldots, \tilde{h}_{jq}^{(t)})$ producing incremental update $dh_j^{(t)}$
5: **end for**
6: Update $\mathbf{H}^{(t+1)} = \mathbf{H}^{(t)} + \mathbf{dH}^{(t)}$
7: Repeat steps 2-6 for $\mathbf{W}^{(t+1)}$
8: **end for**

$$\tilde{\mathbf{w}}_l = \sigma\left(\sum_{l'=1}^{q'}\sum_{j=0}^{p}\theta_{j,ll'}^{(r)}T_j(\tilde{\mathbf{\Delta}}_r)\mathbf{w}_{l'}\right), \qquad \tilde{\mathbf{h}}_l = \sigma\left(\sum_{l'=1}^{q'}\sum_{j'=0}^{p}\theta_{j',ll'}^{(c)}T_{j'}(\tilde{\mathbf{\Delta}}_c)\mathbf{h}_{l'}\right), \quad l = 1, \ldots, q. \quad (4.7)$$

We call such an architecture a *separable MGCNN* or *sMGCNN*.

Learnable diffusion. The last step of our approach to reconstruct missing information in $\mathbf{X}$ is to feed the spatial features extracted by MGCNN or sMGCNN layers to a recurrent neural network (RNN) implementing a diffusion process (Figure 4.1 and 4.2). Modelling matrix completion as a diffusion process appears particularly suitable for realizing an architecture that is independent on the sparsity of the available information. In order to combine the few scores available in a sparse input matrix, a multi-layer GCNN would require very large filters or many layers to diffuse the score information across matrix domains. On the contrary, a diffusion-based approach allows to reconstruct the missing information just by imposing the proper amount of diffusion iterations. This gives the possibility to deal with extremely sparse data, without requiring at the same time excessive amounts of model parameters (Table 4.1 in the experimental section of this chapter shows some results in this direction).

For our experiments, we use the classic LSTM architecture [119], which has demonstrated to be efficient to learn complex non-linear diffusion processes, due to its ability to keep long-term internal states (limiting in particular the vanishing gradient problem that affects vanilla RNNs). The input of the LSTM gate is given by the static features extracted from a cascade of MGCNN/sMGCNN layers, which can be seen as a projection or dimensionality reduction of the original matrix in the space of the most meaningful and representative information. The output

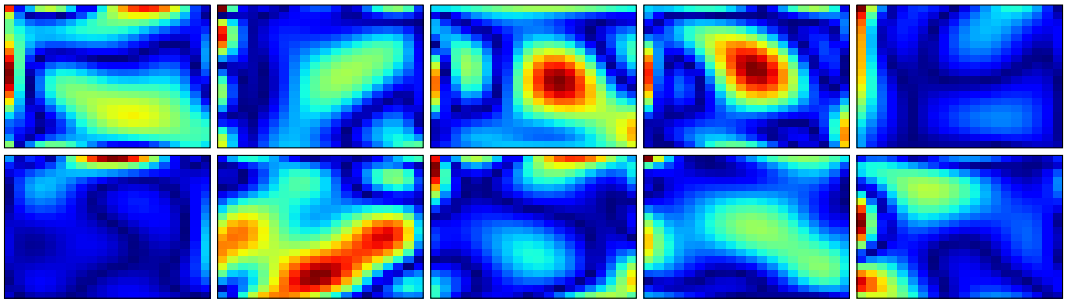


Figure 4.3. Absolute value $|\tau(\tilde{\lambda}_c, \tilde{\lambda}_r)|$ of the first ten spectral filters learnt by our MGCNN model. In each matrix, rows and columns represent frequencies $\tilde{\lambda}_r$ and $\tilde{\lambda}_c$ of the row and column graphs, respectively.

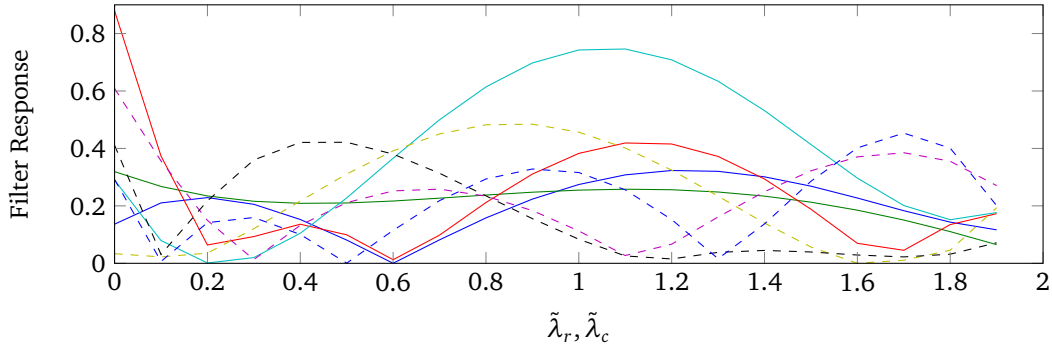


Figure 4.4. Absolute values $|\tau(\tilde{\lambda}_c)|$ and $|\tau(\tilde{\lambda}_r)|$ of the first four column (solid) and row (dashed) spectral filters learned by our sMGCNN model.

of the LSTM module is instead a small change $\mathbf{dX}$ of matrix $\mathbf{X}$ (or $\mathbf{dW}$, $\mathbf{dH}$ of factors $\mathbf{W}$, $\mathbf{H}$) that is summed to the input matrix to reconstruct the missing information.

Algorithms 1 and 2 summarize the proposed matrix completion architectures. We refer to the whole architecture combining MGCNN and RNN in the full matrix completion setting as *Recurrent Multi-Graph CNN (RMGCNN)*. The factorized version with separable MGCNN and RNN is referred to as *separable RMGCNN (sRMGCNN)*. The complexity of Algorithm 1 scales quadratically as $\mathcal{O}(|\mathcal{V}_r||\mathcal{V}_c|)$ due to the MGCNN layers. For large matrices, Algorithm 2 that processes the rows and columns separately with standard GCNNs (and scales linearly as a $\mathcal{O}(|\mathcal{V}_r| + |\mathcal{V}_c|)$) is probably preferable, being a more efficient solution.

Training and testing. Training of the networks is performed by minimizing loss:

$$\ell(\boldsymbol{\Theta}, \boldsymbol{\sigma}) = \|\mathbf{X}^{(T)}_{\boldsymbol{\Theta},\boldsymbol{\sigma}}\|^2_{\mathcal{G}_r} + \|\mathbf{X}^{(T)}_{\boldsymbol{\Theta},\boldsymbol{\sigma}}\|^2_{\mathcal{G}_c} + \frac{\mu}{2}\|\boldsymbol{\Omega} \circ (\mathbf{X}^{(T)}_{\boldsymbol{\Theta},\boldsymbol{\sigma}} - \mathbf{Y})\|^2_{\mathrm{F}}. \tag{4.8}$$

Here, $\|\mathbf{X}\|^2_{\mathcal{G}_r} = \mathrm{trace}(\mathbf{X}^\top \tilde{\boldsymbol{\Delta}}_r \mathbf{X})$ and $\|\mathbf{X}\|^2_{\mathcal{G}_c} = \mathrm{trace}(\mathbf{X}\tilde{\boldsymbol{\Delta}}_c \mathbf{X}^\top)$ are the Dirichlet energies of $\mathbf{X}$ on $\mathcal{G}_r$ and $\mathcal{G}_c$ respectively, $\boldsymbol{\Omega} \in \{0,1\}^{|\mathcal{V}_r| \times |\mathcal{V}_c|}$ is a matrix identifying which entries the model should aim to infer at training time (target scores) out of the ones available in the training set (data scores), and T denotes the number of diffusion iterations (applications of the RNN). Training

scores have been uniformly split between data scores and target scores in our experiments to provide a full coverage of the entire matrix to the model. At test time we initialize the input matrix only with the considered data scores, to provide the network the same conditions it observed at training time. We use the notation $\mathbf{X}^{(T)}_{\boldsymbol{\Theta},\boldsymbol{\sigma}}$ to emphasize that the matrix depends on the parameters of the MGCNN (Chebyshev polynomial coefficients $\boldsymbol{\Theta}$) and those of the LSTM (denoted by $\boldsymbol{\sigma}$). In the factorized setting, we use loss:

$$\ell(\boldsymbol{\theta}_r, \boldsymbol{\theta}_c, \boldsymbol{\sigma}) = \|\mathbf{W}^{(T)}_{\boldsymbol{\theta}_r,\boldsymbol{\sigma}}\|^2_{\mathcal{G}_r} + \|\mathbf{H}^{(T)}_{\boldsymbol{\theta}_c,\boldsymbol{\sigma}}\|^2_{\mathcal{G}_c} + \frac{\mu}{2}\|\boldsymbol{\Omega} \circ (\mathbf{W}^{(T)}_{\boldsymbol{\theta}_r,\boldsymbol{\sigma}}(\mathbf{H}^{(T)}_{\boldsymbol{\theta}_c,\boldsymbol{\sigma}})^T - \mathbf{Y})\|^2_{\mathrm{F}}; \tag{4.9}$$

where $\boldsymbol{\theta}_c, \boldsymbol{\theta}_r$ are the parameters of the two GCNNs. In experiments where only one graph of $\mathbf{X}$ is provided, only the Dirichlet energy of the provided graph is used in the loss function. For the factorized model with only one graph available, the term defined over the missing graph is simply considered as a learnable parameter of the model.

4.3 Results

Experimental setting. We closely followed the experimental setup of [212], using five standard datasets: Synthetic dataset from [131], MovieLens [180], Flixster [128], Douban [164], and YahooMusic [81]. We used disjoint training and test sets, the presented results are reported on test sets in all our experiments. As in [212], we evaluated MovieLens using only the first of the 5 provided data splits. For Flixster, Douban and YahooMusic, we evaluated on a reduced matrix of 3,000 users and items, considering 90% of the given scores as training set and the remaining as test set. Classical Matrix Completion (MC) [49], Inductive Matrix Completion (IMC) [126, 268], Geometric Matrix Completion (GMC) [131], and Graph Regularized Alternating Least Squares (GRALS) [212] were used as baseline methods. In all the experiments, we used the following settings for our RMGCNNs: Chebyshev polynomials of order $p = 4$, outputting $k = 32$-dimensional features, LSTM cells with 32 features and $T = 10$ diffusion steps (for both training and test). The number of diffusion steps T has been estimated on the MovieLens validation set and used in all our experiments. A better estimate of T can be done by cross-validation, and thus can potentially only improve the final results. Only one convolutional layer was used in our experiments for producing features in input to LSTM. All the models were implemented in Google TensorFlow, and trained using the Adam stochastic optimization algorithm [136] with learning rate 10^{-3}. In factorized models, ranks $d = 15$ and 10 were used for the synthetic and real datasets, respectively. For all methods, hyperparameters were chosen by cross-validation.

Synthetic data. We start the experimental evaluation showing the performance of our approach on a small synthetic dataset, in which the user and item graphs have strong communities structure and scores are smooth over such communities. Though rather simple, such a dataset allows to study the behavior of different algorithms in a controlled setting.

The performance of different matrix completion methods is reported in Table 4.2, along with their theoretical complexity. Our RMGCNN and sRMGCNN models achieve better accuracy than other methods with lower complexity. Different diffusion time steps of these two models are visualized in Figure 4.5. Figures 4.3 and 4.4 depict the spectral filters learnt by MGCNN and row- and column-GCNNs on such dataset.

To investigate the performance of our method in scenarios where only one graph might be available, we repeated the same experiment assuming only the column (users) graph to

Table 4.1. Reconstruction errors for the synthetic dataset between multiple convolutional layers architectures and the proposed architecture. Chebyshev polynomials of order 4 have been used for both users and movies graphs (q'MGCq denotes a multi-graph convolutional layer with q' input features and q output features). In **black** / **red** / **blue** performance of the 1st / 2nd / 3rd best model.

Method	Params	Architecture	RMSE
$\text{MGCNN}_{\text{3layers}}$	$9K$	1MGC32, 32MGC10, 10MGC1	0.0116
$\text{MGCNN}_{\text{4layers}}$	$53K$	1MGC32, 32MGC32 × 2, 32MGC1	
$\text{MGCNN}_{\text{5layers}}$	$78K$	1MGC32, 32MGC32 × 3, 32MGC1	0.0074
$\text{MGCNN}_{\text{6layers}}$	$104K$	1MGC32, 32MGC32 × 4, 32MGC1	**0.0064**
RMGCNN	$9K$	1MGC32 + LSTM	**0.0053**

Table 4.2. Comparison of different matrix completion methods using *users+items graphs* in terms of number of parameters (optimization variables) and computational complexity order (operations per iteration). Big-O notation is avoided for clarity reasons. Rightmost column shows the RMS error on Synthetic dataset. In **black** / **red** / **blue** performance of the 1st / 2nd / 3rd best model on a given dataset.

Method	Params	No. Op.	RMSE
GMC	$\lvert\mathcal{V}_r\rvert\lvert\mathcal{V}_c\rvert$	$\lvert\mathcal{V}_r\rvert\lvert\mathcal{V}_c\rvert$	0.3693
GRALS	$\lvert\mathcal{V}_r\rvert+\lvert\mathcal{V}_c\rvert$	$\lvert\mathcal{V}_r\rvert+\lvert\mathcal{V}_c\rvert$	**0.0114**
sRMGCNN	1	$\lvert\mathcal{V}_r\rvert+\lvert\mathcal{V}_c\rvert$	**0.0106**
RMGCNN	1	$\lvert\mathcal{V}_r\rvert\lvert\mathcal{V}_c\rvert$	**0.0053**

Table 4.3. Comparison of different matrix completion methods using *users graph only* in terms of number of parameters (optimization variables) and computational complexity order (operations per iteration). Big-O notation is avoided for clarity reasons. Rightmost column shows the RMS error on Synthetic dataset. In **black** performance of the best model.

Method	Params	No. Op.	RMSE
GRALS	$\lvert\mathcal{V}_r\rvert+\lvert\mathcal{V}_c\rvert$	$\lvert\mathcal{V}_r\rvert+\lvert\mathcal{V}_c\rvert$	0.0452
sRMGCNN	$\lvert\mathcal{V}_r\rvert$	$\lvert\mathcal{V}_r\rvert+\lvert\mathcal{V}_c\rvert$	**0.0362**

be given. In this setting, RMGCNN cannot be applied, while sRMGCNN has only one GCNN applied on the factor **H** (the other factor **W** is free). Table 4.3 summarizes the results of this experiment, again, showing that our approach performs the best.

Table 4.1 compares our RMGCNN with more classical multilayer MGCNNs. Our recurrent solutions outperforms deeper and more complex architectures, requiring at the same time a lower amount of parameters.

Real data. Following [212], we evaluated the proposed approach on the MovieLens, Flixster, Douban and YahooMusic datasets. For the MovieLens dataset we constructed the user and item (movie) graphs as unweighted 10-nearest neighbor graphs in the space of user and movie features, respectively. For Flixster, the user and item graphs were constructed from the scores of the original matrix. On this dataset, we also performed an experiment using only the users graph. For the Douban dataset, we used only the user graph (provided in the form of a social network). For the YahooMusic dataset, we used only the item graph, constructed with unweighted 10-nearest neighbors in the space of item features (artists, albums, and genres). For the latter

Table 4.4. Performance (RMS error) of different matrix completion methods on the MovieLens dataset. In **black** / **red** / **blue** performance of the 1st / 2nd / 3rd best model

METHOD	RMSE
GLOBAL MEAN	1.154
USER MEAN	1.063
MOVIE MEAN	1.033
MC [49]	0.973
IMC [126, 268]	1.653
GMC [131]	**0.996**
GRALS [212]	**0.945**
SRMGCNN	**0.929**

Table 4.5. Performance (RMS error) on several datasets. For Douban and YahooMusic, a single graph (of users and items respectively) was used. For Flixster, two settings are shown: users+items graphs / only users graph. In **black** performance of the best model.

METHOD	FLIXSTER	DOUBAN	YAHOOMUSIC
GRALS	1.3126 / 1.2447	0.8326	38.0423
SRMGCNN	**1.1788 / 0.9258**	**0.8012**	**22.4149**

three datasets, we used a sub-matrix of $3,000 \times 3,000$ entries for evaluating the performance. Tables 4.4 and 4.5 summarize the performance of different methods. sRMGCNN outperforms the competitors in all the experiments.

4.4 Discussion

To the best of our knowledge, RMGCNN and sRMGCNN were the first methods that leveraged GCNNs for addressing the recommendation problem in the literature. From the publication of our work, there has been increasingly more interest in this class of techniques for implementing recommender systems. Due to the scale at which the recommendation problem often needs to be addressed (think for instance to social media that need to recommend relevant items to billions of users on a daily basis), recommendation is often casted as a candidate generation plus ranking problem, where a small set of candidate items (e.g. hundreds) is first retrieved for a given user with efficient approaches, and a ranking of the returned candidates is constructed at a later stage with rich (but heavier) solutions. Full matrices describing the relevance of all items for all users are thus rarely computed in practice, and an encoder-decoder approach is typically exploited to produce user / item embeddings that one might need to solve the candidate generation / ranking phase[2]. While a complete overview of the application of GCNNs to recommender systems is beyond the scope of this work (the interested reader can refer to [94] for a comprehensive review), we provide here a few references to popular approaches that have been proposed in the last few years, which highlight the general direction the community took from the publication of our work[3].

[2]Please note, sRMGCNN can in principle be understood as an encoder-decoder approach, where the embeddings of user and items are obtained through diffusion on the similarity graphs, and the decoder is simply implemented with dot product.

[3]For the sake of brevity, we focus here only on models designed for static graphs. It should be noted however, that models designed for handling temporal networks (i.e. graphs with evolving node set, edge set and feature values) have been proposed in the literature as well (e.g. see [62] for a recent work processing dynamic sequential graphs for

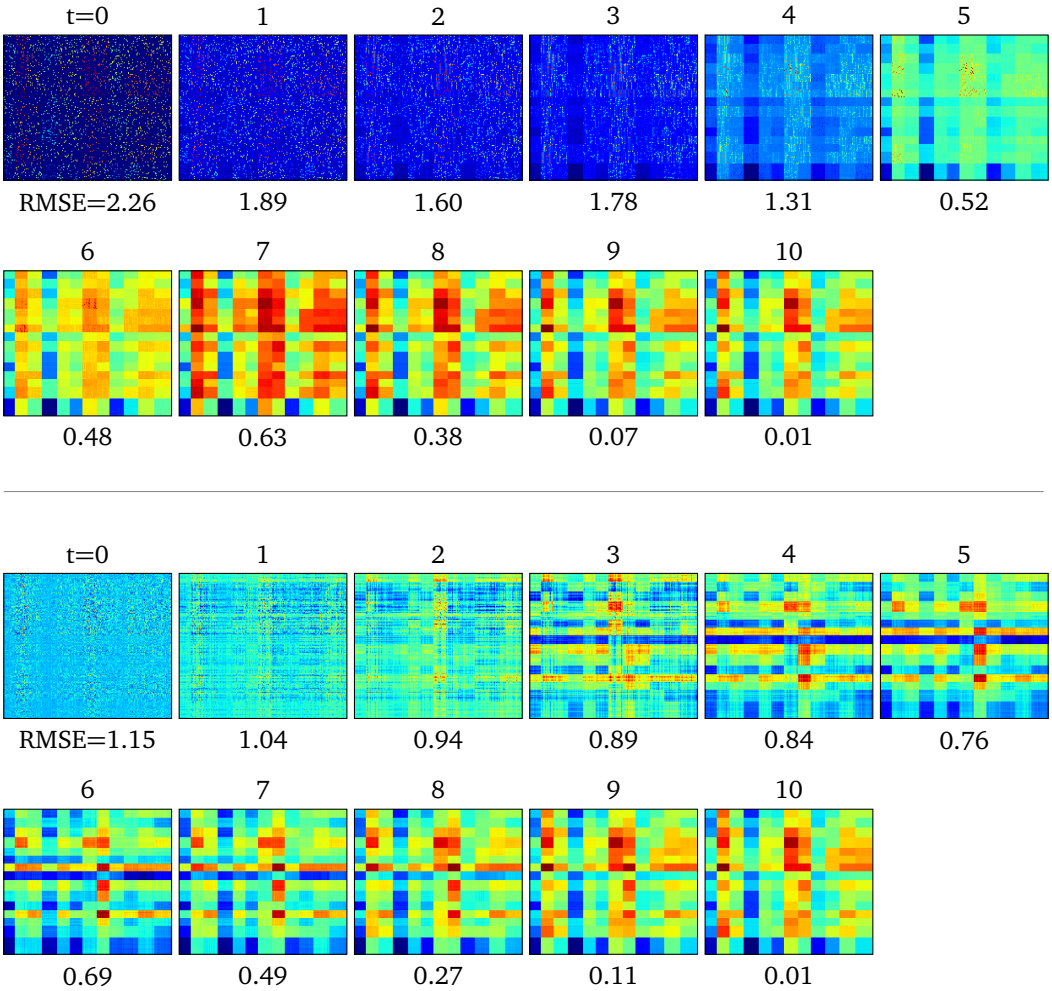


Figure 4.5. Evolution of the matrix $\mathbf{X}^{(t)}$ with our architecture using full matrix completion model RGCNN (top 2 rows) and factorized matrix completion model sRGCNN (bottom 2 rows). The diffusion time associated to each matrix is placed on top of the relative matrix, the RMSE with respect to the ground truth is instead on the bottom.

GCNNs on attributed graphs. In *GCMC* [29], van den Berg et al. proposed to predict the relevance of a given item for a particular user, resorting to a GCNN-based auto-encoder. The model receives as input a weighted bi-partite graph depicting the interactions between users and items (here the weight of an edge describes for instance the score a user assigned to a particular movie), it then embeds the nodes of the graph via a one-layer GCNN (diffusing information via $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ or $\mathbf{D}^{-1}\mathbf{A}$), and finally predicts the score of each (user, item) through a bilinear decoder. At training time, the model is trained to predict the weight of each observed interaction (using a classic cross-entropy loss), while at test time it is used to reconstruct the missing entries of the interaction matrix. As the matrix describing interactions between users

click-through rate prediction).

and items corresponds with the adjacency matrix of the bipartite user-item graph, predicting the relevance of each (user, item) corresponds to predict the weight of each possible edge. The recommendation problem is thus effectively cast as a link prediction task. As the features available in the experimental evaluation of [29] were not particularly representative of the users interests / items properties (e.g. in the context of the MovieLens dataset: age, gender and occupation were the only features available for the users; while the genre was the only property used to describe items), a one-hot encoding was used as input to the convolutional layer in [29], and the aforementioned information was injected in the model only in a dense layer afterwards. In a more general setting, where meaningful embeddings are available to describe the different nodes of the graph (e.g. as it was the case in [272] that we will see next), such information can be directly used as input to the convolutional layer, in order to produce an architecture with a number of parameters independent on the graph size.

Following from [29], to realize a solution able to efficiently produce embeddings for billions of users / items, *PinSAGE* [272] introduced an encoder-decoder architecture based on a variation of GraphSAGE that relies on importance sampling / pooling for realizing convolution[4]. Differently from what previously presented in this chapter, the goal in [272] was to retrieve similar pins provided a query one, and to populate the home feed of Pinterest's users via a (approximate) nearest neighbor search. To ensure the construction of embeddings that allow to return the few relevant items out of a catalog consisting of even billion of entries, a triplet loss (see Chapter 9) combined with curriculum learning was used for training. Embeddings describing the visual / textual content of pins were used as input features to the model, while the input graph was defined as a bi-partite network connecting pins with boards defined by the users. Dot product was used in [272] as decoding function. To achieve high GPU utilization in the learning phase (and thus optimize the consumption of the available resources), a producer-consumer mini-batch construction strategy was further implemented to sample the graph in parallel to a forward/backward pass. An implementation of convolution based on MapReduce was instead introduced to allow the model to scale to web-size graphs in production. To the best of our knowledge, PinSAGE was the first GCNN deployed in a production recommender system, and outperformed learnable content-based and non-learnable graph-based approaches in experimental evaluation.

In a follow up work, Gurukar et al. further introduced *MultiBiSAGE* [107], an extension of PinSAGE that processes heterogeneous networks describing multiple types of interactions among different types of entities (e.g. user clicks product, user follows board, pin contained in board, ...). The main intuition of the paper is that the information contained in different interactions is generally beneficial to provide an holistic view on the properties of the available pins, and it can thus be used to improve recommendations for a variety of engagements. In [107], heterogeneous networks are first decomposed in k bi-partite graphs, in order to re-utilize existing infrastructure developed for PinSAGE. For each extracted graph, a transformer architectures is subsequently applied on the one-hop neighborhood of a target node to compute k different node representations (a global token is used here to condition the behavior of the transformer dependently on the type of interactions where it is applied). Finally, the different embeddings are combined together through a second transformer architecture (the number of parameters of the model is thus independent on the amount of bi-partite graphs provided as input). Experimental evaluation on 8 different types of engagements highlight how embeddings

[4] Instead of resorting to a uniform sampling of the neighbors, the neighbors that are visited the most during a random walk are retained for diffusion. The retained neighbors receive a weight in the sampled graph that is proportional to the number of times these are visited by the walker.

computed on an heterogeneous network, built with six different types of interactions, yield better performance compared to the ones provided by PinSAGE on the sole pin-board graph.

Graph Collaborarive Filtering. One of the most classic methodologies for solving the recommendation problem in the literature is *Collaborative Filtering* (CF) [41]. The main intuition at the core of a CF approach is that: if two users u_1 and u_2 generally rate items similarly (or, more widely, have similar interests), then the items that are relevant for u_2 should most likely be relevant to u_1, and can thus be recommended to them. Observing that, in bi-partite interaction networks, nodes situated at a distance smaller or equal to k (with $k > 1$ but still relatively small, e.g. $k \in \{2, 3, 4, \ldots\}$) from a target user u correspond either with users that share similar interests with u (at least at some level), or items that such similar users interacted with[5] (and that an equivalent reasoning can be made taking a target item i), Wang et al. [256] proposed to capture the collaborative signal available in the graph, by making the embeddings of users / items a function of their k-hop neighbors interactions. In [256], the authors thus introduced *Neural Graph Collaborative Filtering* (NGCF), a GCNN-based approach for learning latent node representations, where the embedding function simply corresponds with a series of message passing layers that are applied on top of "raw" *learnable* node features (no additional information is assumed available to the model besides the interaction network). The dot product was used in the paper as a decoding function, and the *Bayesan Personalized Ranking Loss* [216] was used for training. Experiments on three different real-world benchmarks showed better performance of the proposed method with respect to a variety of baselines, including simple matrix factorization (MF), GCMC, and PinSAGE (embeddings pre-computed with MF were used as input features for the last two approaches). Better results were in particular achieved with relatively deep architectures (three or four convolutional layers) in [256] (as reference, GC-MC used only one convolutional layer in [29]), which emphasized the benefit of constructing user / item embeddings that directly depend on the behavior of *higher order neighbors* (i.e. neighbors situated at a distance larger than one) for the recommendation problem.

Due to the good performance shown in [256], NGCF spawned some level of interest in the community in the last years, and a series of follow up works appeared in the literature as a result. In [113], He et al. introduced *LightGCN*, a simplification of NGCF, where no non-linearities, nor matrices of weights are actually used in the diffusion layers. The main result highlighted by He et al. was that the good performance of NGCF mainly derived from the diffusion process that favored smooth embeddings across nearby nodes, rather than the extra logic provided by the aforementioned components. In [171], Mao et al. showed that in order to achieve a simple yet effective solution based on collaborative filtering, message passing operations are not strictly necessary, and a matrix factorization approach (*UltraGCN*), inspired from the properties of the steady state of an infinitely deep LightGCN, can be used to achieve good performance in practice. In [226], Shen et al. further introduced a general framework for collaborative filtering based on graph-signal processing, which highlights how previous approaches (e.g. low-rank matrix factorization, LightGCN, ...) can be seen as some form of low-pass filtering on an item-item similarity graphs. A CF approach (GF-CF) based on the combination of a linear and an ideal low-pass filter was additioanlly introduced in [226].

[5] Recursively, we can see that nodes situated at distance equal to 2 from u ($\mathcal{U}^{(2)}$) are users that share some interactions with u (and have, as such, somehow similar interests to u), nodes at distance equal to 3 from u are items ($\mathcal{I}^{(3)}$) that $\mathcal{U}^{(2)}$ interacted with (and are thus items that similar users to u interacted with), nodes at distance equal to 4 ($\mathcal{U}^{(4)}$) are users that share some interactions with $\mathcal{U}^{(2)}$ (and are thus users that are similar to users that have similar interests with u), etc.

Chapter 5

Graph Convolutional Neural Networks with Complex Rational Spectral Filters

This chapter is based on "Ron Levie, Federico Monti*, Xavier Bresson, and Michael M Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters.* IEEE Transactions on Signal Processing*, 67(1):97–109, 2018" (* denotes equal contribution).*

5.1 Methodology

In [153], we construct GCNNs employing an efficient spectral filtering scheme based on a new class of polynomials named *Cayley polynomials*. Filters implemented with Cayley polynomials enjoy similar properties to the ones showed by Chebyshev filters [72] (localization and linear complexity in the number of edges), while adding the capability of detecting narrow frequency bands of importance during training and to specialize on them. We define a *Cayley polynomial* of order r to be a real-valued function with complex coefficients:

$$g_{\mathbf{c},h}(\lambda) = c_0 + 2\,\mathfrak{Re}\Big\{\sum_{j=1}^{r} c_j (h\lambda - i)^j (h\lambda + i)^{-j}\Big\}; \tag{5.1}$$

where $\mathbf{c} = (c_0, \ldots, c_r)$ is a vector of a real coefficient c_o and r complex coefficients $c_i \forall i > 0, h > 0$ is what we call a *spectral zoom* parameter, and λ is an eigenvalue of the (possibly normalized) Laplace operator of the given graph. A *Cayley filter* $\mathbf{G}$ implemented with a Cayley polynomial is a spectral filter defined on real signals $\mathbf{x} \in \mathbb{R}^{|\mathcal{V}|}$ as:

$$\mathbf{G}\mathbf{x} = g_{\mathbf{c},h}(\mathbf{\Delta})\mathbf{x} = c_0\mathbf{x} + 2\,\mathfrak{Re}\{\sum_{j=1}^{r} c_j (h\mathbf{\Delta} - i\mathbf{I})^j (h\mathbf{\Delta} + i\mathbf{I})^{-j}\mathbf{x}\}, \tag{5.2}$$

where the parameters $\mathbf{c}$ and h are learnable parameters of the filter and are optimized during training. Similarly to Chebyshev filters, Cayley filters involve basic matrix operations such as powers, additions, multiplications by scalars, and also inversions. This implies that applications of the filter $\mathbf{G}\mathbf{x}$ can be performed without explicit expensive eigendecomposition of the Laplacian, thus scaling well also on big graphs (the matrix inversion can additionally be avoided resorting to an iterative process, paragraph "avoiding matrix inversion").

Interpreting Cayley filters. The simplest way to understand Cayley filters is probably through the so-called *Cayley transform*, from which their name derives. Denote by $e^{i\mathbb{R}} = \{e^{i\theta} : \theta \in \mathbb{R}\}$ the unit complex circle, the *Cayley transform* $\mathcal{C}(x) = \frac{x-i}{x+i}$ of a real value x is a smooth bijection between $\mathbb{R}$ and $e^{i\mathbb{R}} \setminus \{1\}$. Similarly, the Cayley transform of a matrix $\mathbf{\Delta} = \mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Phi}^T$ is $\mathcal{C}(\mathbf{\Delta}) = (\mathbf{\Delta} - i\mathbf{I})(\mathbf{\Delta} + i\mathbf{I})^{-1}$ and simply corresponds to the Cayley transform applied to each of its eigenvalues: $\mathcal{C}(\mathbf{\Delta}) = (\mathbf{\Delta} - i\mathbf{I})(\mathbf{\Delta} + i\mathbf{I})^{-1} = \mathbf{\Phi}(\mathbf{\Lambda} - i\mathbf{I})(\mathbf{\Lambda} + i\mathbf{I})^{-1}\mathbf{\Phi}^T$. As $z^{-1} = \overline{z}$ for $z \in e^{i\mathbb{R}}$ and $2\,\mathfrak{Re}\{z\} = z + \overline{z}$, we can write $c_j\mathcal{C}^j(h\mathbf{\Delta}) = \overline{c_j}\mathcal{C}^{-j}(h\mathbf{\Delta})$ and subsequently:

$$\begin{aligned}\mathbf{G} &= c_0\mathbf{I} + 2\,\mathfrak{Re}\{\sum_{j=1}^{r} c_j\mathcal{C}^j(h\mathbf{\Delta})\} = c_0\mathbf{I} + \sum_{j=1}^{r} c_j\mathcal{C}^j(h\mathbf{\Delta}) + \overline{c_j}\mathcal{C}^{-j}(h\mathbf{\Delta}) \\ &= \mathbf{\Phi}\left(c_0\mathbf{I} + \sum_{j=1}^{r} c_j\mathcal{C}^j(h\mathbf{\Lambda}) + \overline{c_j}\mathcal{C}^{-j}(h\mathbf{\Lambda})\right)\mathbf{\Phi}^T.\end{aligned} \tag{5.3}$$

Any Cayley filter can thus be written as a conjugate-even Laurent polynomial (coefficients for equal but opposite powers are the conjugate of one another) with respect to $\mathcal{C}(h\mathbf{\Delta})$ (and $\mathcal{C}(h\mathbf{\Lambda})$). Since in equation (5.3) the Laurent polynomial used for interpolating over the eigenvalues of the Laplacian corresponds to a trigonometric polynomial[1] (which are are complete on the unit circle), Cayley filters of sufficiently high order can implement any spectral filter on the provided graph. On top of this, as low order trigonometric polynomials generate a response that is smooth over the frequencies, filters implemented with small values of r will also enjoy some degree of locality on the given graph. In Euclidean domains, smoothness in the spectrum is associated with localization per Parseval's identity:

$$\int_{-\infty}^{\infty} |\frac{\partial^n \hat{f}}{\partial \xi^n}(\xi)| d\xi = \int_{-\infty}^{\infty} (2\pi x)^n |f(x)| dx; \tag{5.4}$$

similar results also hold for graphs via the generalization of a shift operator in the spectral domain (see [232], Section 4.4). Cayley filters can in particular be proved to have exponential decay in $\mathbb{L}_2$, as we'll see in paragraph "localization".

Spectral zoom. To understand the role of parameter h in Cayley filters, we need to look at how this affects the spectrum of $\mathcal{C}(h\mathbf{\Delta})$. Since $h\mathbf{\Delta} = \mathbf{\Phi}(h\mathbf{\Lambda})\mathbf{\Phi}^T$, muliplying h by $\mathbf{\Delta}$ just corresponds to dilating the Laplacian spectrum. In turn, as applying $\mathcal{C}(.)$ on $h\mathbf{\Delta}$ maps the non-negative spectrum to the lower complex half-circle (and the larger/smaller the value of a scaled eigenvalue $h\lambda_i$ the closer it gets mapped to 1/-1), the value of h allows to control how much low or high frequencies should be spread apart before being fed to the filter's polynomial (Figure 5.1). This allows to 'zoom' on different parts of the spectrum making it simpler for a filter to specialize on some specific frequency bands.

Avoiding matrix inversion. Let $\mathbf{y}_0, \ldots, \mathbf{y}_r$ denote the solutions of the following linear recursive system:

$$\mathbf{y}_0 = \mathbf{x}, \qquad (h\mathbf{\Delta} + i\mathbf{I})\mathbf{y}_j = (h\mathbf{\Delta} - i\mathbf{I})\mathbf{y}_{j-1}, \qquad j = 1, \ldots, r. \tag{5.5}$$

[1] If $c_j \in \mathbb{R} \forall j$ then equation (5.1) is an even cosine polynomial, if $c_j \in i\mathbb{R} \forall j$ then equation (5.1) is an odd sine polynomial, if $c_j \in \mathbb{C}$ then equation (5.1) is a full trigonometric polynomial of both sine and cosine.

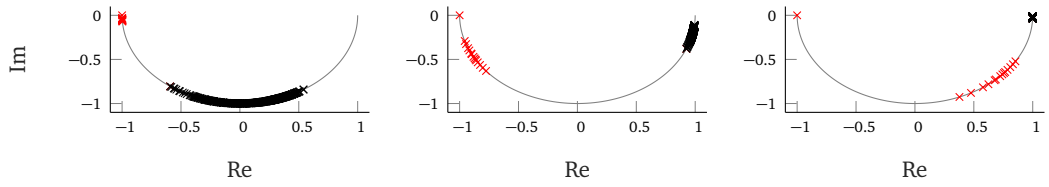


Figure 5.1. Eigenvalues of the scaled Laplacian $h\mathbf{\Delta}$ of a connected 15-communities graph (Appendix A, Figure 5.5 left) mapped on the complex unit half-circle with Cayley transform ($h = 0.1$, 1, and 10 left-to-right). The first 15 frequencies carrying information about the communities are marked in red.

A Cayley filter can be written as $\mathbf{Gx} = c_0\mathbf{y}_0 + 2\,\mathrm{Re}\{\sum_{j=1}^{r} c_j\mathbf{y}_j\}$. As the cost of inverting $h\mathbf{\Delta} + i\mathbf{I}$ is $\mathcal{O}(|\mathcal{V}|^3)$, computing the exact value of each $\mathbf{y}_j$ can be prohibitively expensive for sufficiently large $|\mathcal{V}|$ and thus hinders the applicability of the entire approach to large graphs (the complexity of computing an exact Cayley filter is $\mathcal{O}(|\mathcal{V}|^3 + r|\mathcal{V}|^2)$, where $\mathcal{O}(r|\mathcal{V}|^2)$ is due to the r projections of $\mathbf{x}$ on $\mathcal{C}(h\mathbf{\Delta})$). In order to avoid such an expensive operation, and allow the approach to scale well, an alternative solution is to use the *Jacobi method* for computing approximate solutions $\tilde{\mathbf{y}}_j \approx \mathbf{y}_j$ and subsequently constructing an *approximate Cayley filter*:

$$\widetilde{\mathbf{Gx}} = c_0\tilde{\mathbf{y}}_0 + 2\,\mathrm{Re}\{\sum_{j=1}^{r} c_j\tilde{\mathbf{y}}_j\}. \tag{5.6}$$

First of all, note that sequentially approximating $\mathbf{y}_j$ in equation (5.5) using the approximation of $\mathbf{y}_{j-1}$ in the right hand side is stable, since $\mathcal{C}(h\mathbf{\Delta})$ is unitary[2] and thus has condition number equal to 1 (errors in $\mathbf{y}_{j-1}$ are not amplified in $\mathbf{y}_j$ by the system of equations). Using the Jacobi method for solving a generic system of linear equations $\mathbf{Ax} = \mathbf{b}$ consists in decomposing $\mathbf{A} = \mathrm{Diag}(\mathbf{A}) + \mathrm{Off}(\mathbf{A})$ and obtaining the solution iteratively as

$$\mathbf{x}^{(k+1)} = -(\mathrm{Diag}(\mathbf{A}))^{-1}\mathrm{Off}(\mathbf{A})\mathbf{x}^{(k)} + (\mathrm{Diag}(\mathbf{A}))^{-1}\mathbf{b}, \tag{5.7}$$

until convergence is reached. In our case, fixing $\mathbf{J} = -(\mathrm{Diag}(h\mathbf{\Delta} + i\mathbf{I}))^{-1}\mathrm{Off}(h\mathbf{\Delta} + i\mathbf{I})$ to be the Jacobi iteration matrix, an iteration of the Jacobi method to approximate equation (5.5) for a given j has the form:

$$\begin{aligned} \tilde{\mathbf{y}}_j^{(k+1)} &= \mathbf{J}\tilde{\mathbf{y}}_j^{(k)} + \mathbf{b}_j; \\ \mathbf{b}_j &= (\mathrm{Diag}(h\mathbf{\Delta} + i\mathbf{I}))^{-1}(h\mathbf{\Delta} - i\mathbf{I})\tilde{\mathbf{y}}_{j-1}. \end{aligned} \tag{5.8}$$

Here, $\tilde{\mathbf{y}}_j^{(0)}$ is initialized as $\tilde{\mathbf{y}}_j^{(0)} = \mathbf{b}_j$. If the combinatorial Laplacian is used in the definition of the filter, matrix $h\mathbf{\Delta} + i\mathbf{I}$ is strictly diagonally dominant and the Jacobi method is guaranteed to converge. If the normalized Laplacian is used instead, convergence is guaranteed as the spectral radius of $\mathbf{J}$ ($\rho(\mathbf{J}) = \frac{h}{\sqrt{h^2+1}}$) is smaller than 1. It is worth noting how, since parameters c_j are learnable in the model, computing the exact values of $\mathbf{y}_j$ is typically not required as the model can learn to compensate potential noise in the representation and achieve good performance even for a small number of iterations K. As the Laplace operator for a sparse graph has $\mathcal{O}(|\mathcal{V}|)$ non-zero entries, the complexity of the approximate approach is only $\mathcal{O}(rK|\mathcal{V}|)$.

[2] $(\mathcal{C}(h\mathbf{\Delta}))^{-1} = \mathbf{\Phi}(h\mathbf{\Lambda} + i\mathbf{I})(h\mathbf{\Lambda} - i\mathbf{I})^{-1}\mathbf{\Phi}^T = (\mathcal{C}(h\mathbf{\Delta}))^*$, with $(\mathcal{C}(h\mathbf{\Delta}))^*$ the conjugate transpose of $\mathcal{C}(h\mathbf{\Delta})$.

Localization. Differently from filters implemented with Chebyshev polynomials (which are localized by construction on the given graph, i.e. a filter implemented with a polynomial of degree r will have a support of radius r), filters implemented with Cayley polynomials have global support due to the inversion of matrix $(h\mathbf{\Delta} + i\mathbf{I})$. Nonetheless, Cayley filters still show an exponential decay on the provided domain and for sufficiently small values of h it can be shown they are actually well localized (Theorem 5.1.1).

Definition 5.1.1. Let $\mathbf{x} \in \mathbb{R}^{|\mathcal{V}|}$ be a signal on the vertices of graph $\mathcal{G}$, $1 \leq p \leq \infty$, and $0 < \epsilon < 1$. Denote by $S \subseteq \{1, \dots, n\}$ a subset of the vertices and by S^c its complement. We say that the $\mathbb{L}_p$*-mass of* $\mathbf{x}$ *is supported in* S *up to* ϵ if $\|\mathbf{x}|_{S^c}\|_p \leq \epsilon \|\mathbf{x}\|_p$, where $\mathbf{x}|_{S^c} = (x_l)_{l \in S^c}$ is the restriction of $\mathbf{x}$ to S^c. We say that $\mathbf{x}$ has (graph) *exponential decay about vertex* m, if there exists some $\gamma \in (0, 1)$ and $c > 0$ such that for any k, the $\mathbb{L}_p$-mass of $\mathbf{x}$ is supported in $\mathcal{N}_m^{(k)}$ up to $c\gamma^k$.

Remark 1. Note that Definition 5.1.1 is analogous to classical exponential decay on Euclidean space: $|f(x)| \leq R\gamma^{-x}$ iff for every ball B_ρ of radius ρ about 0, $\|f|_{B_\rho^c}\|_\infty \leq c\gamma^{-\rho}\|f\|_\infty$ with $c = \frac{R}{\|f\|_\infty}$.

Theorem 5.1.1. Let $\mathbf{G}$ be a Cayley filter of order r. Then, $\mathbf{G}\boldsymbol{\delta}_m$ has exponential decay about m in $\mathbb{L}_2$, with constants $c = 4\overline{M}\frac{1}{\|\mathbf{G}\boldsymbol{\delta}_m\|_2}$, $\overline{M} = \sum_{j=1}^{r} M_j \left|c_j\right|$ and $\gamma = \kappa^{1/r}$. $\boldsymbol{\delta}_m$ is here an impulse localized on vertex m (i.e. $\delta_{m,i} = 0 \ \forall i \neq m$ and $\delta_{m,m} = 1$). For the combinatorial Laplacian, $M_j = j\sqrt{n}$ and $\kappa = \|\mathbf{J}\|_\infty = \frac{hd}{\sqrt{h^2d^2+1}} < 1$ (with $d = \max_j d_j$ the maximum node degree). For the normalized Laplacian, $M_j = j$ and $\kappa = \|\mathbf{J}\|_2 = \frac{h}{\sqrt{h^2+1}} < 1$.

The proof of Theorem 5.1.1 is provided in Section 5.3

5.2 Results

Experimental settings We test the proposed CayleyNets reproducing the experiments of [72, 138, 184] and using ChebNet [72] as our main baseline method across all experiments. GCN [138], MoNet [184] and GAT [72] are listed as well for document classification tasks, in line with what reported in Chapter 3. Pooling and graph coarsening was performed identically to [72] where required. The hyperparameters are identical to the original experiments, unless specified otherwise. All the methods were implemented in TensorFlow. The experiments were executed on a machine with a 3.5GHz Intel Core i7 CPU, 64GB of RAM, and NVIDIA Titan X GPU with 12GB of RAM. SGD+Momentum and Adam [136] optimization methods were used to train the models in MNIST and the rest of the experiments, respectively. Training and testing were always done on disjoint sets.

MNIST. Following [72], we start our experimental evaluation with a toy experiment to validate the performance of CayleyNet in a controlled setting. In this direction, we approached the classical MNIST digits classification as a learning problem on graphs. Each pixel of an image is a vertex of a graph (regular grid with 8-neighbor connectivity), and pixel color is a signal on

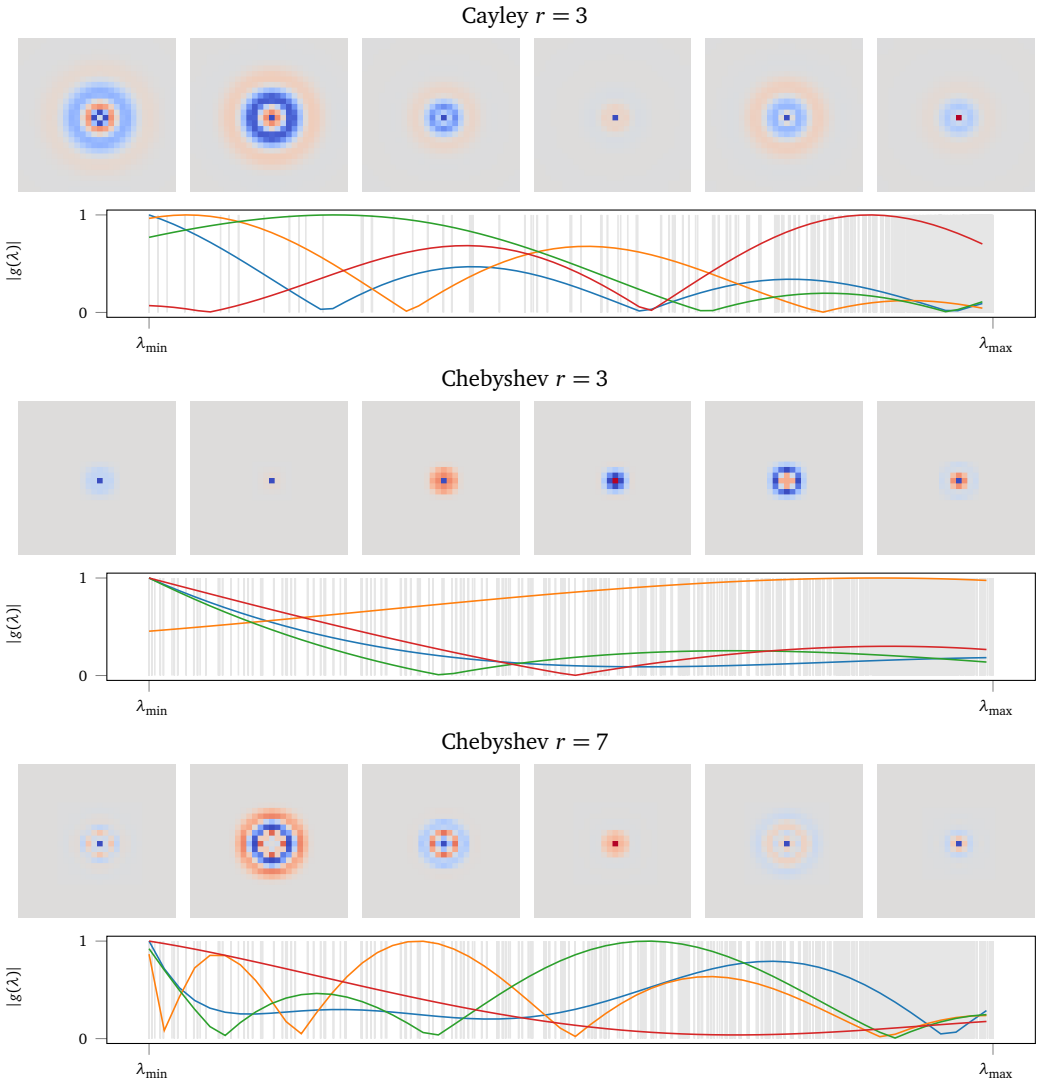


Figure 5.2. Filters (spatial domain, top and spectral domain, bottom) learned by CayleyNet (top) and ChebNet (center, bottom) on the MNIST dataset. Cayley filters are able to realize larger supports for the same order r.

the graph. We used a graph CNN architecture with two spectral convolutional layers based on Chebyshev and Cayley filters (producing 32 and 64 output features, respectively), interleaved with pooling layers performing 4-times graph coarsening using the Graclus algorithm [76], and finally a fully-connected layer (this architecture replicates the classical LeNet5 architecture [151], whose performance is shown in the results for comparison). SGD+Momentum with learning rate equal to 0.02, momentum $m = 0.9$, dropout probability $p = 0.5$ and weight decay coefficient $\gamma = 5 \cdot 10^{-4}$ have been applied as described in [72]. MNIST classification results are reported in Table 5.1. CayleyNet (11 Jacobi iterations) achieves the same (near perfect) accuracy as ChebNet with filters of lower order ($r = 12$ vs 25). Examples of filters learned by

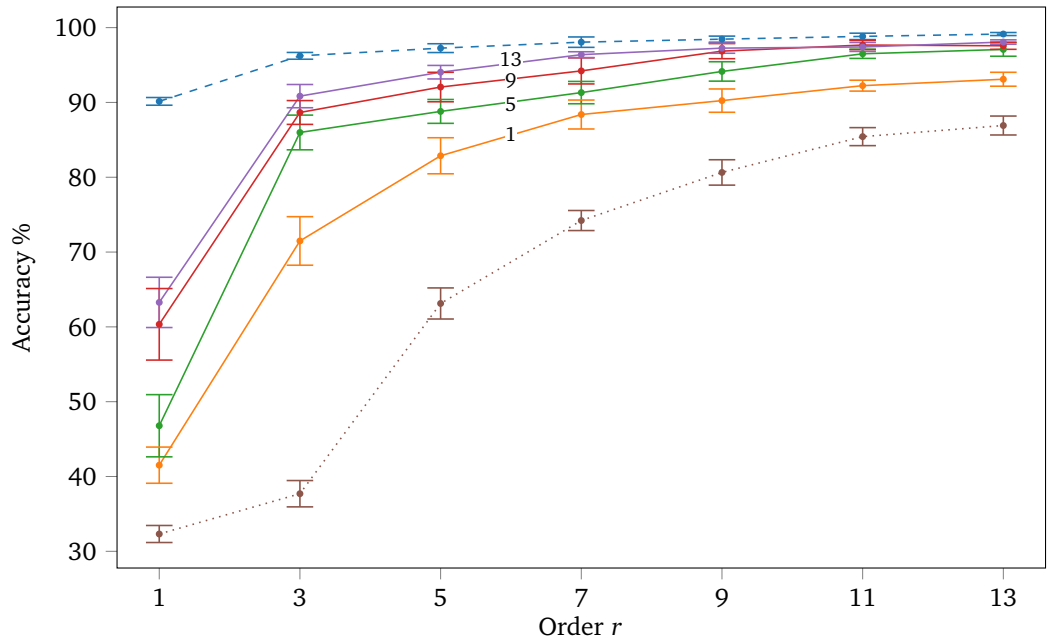


Figure 5.3. Community detection test accuracy as function of filter order r. Shown are exact matrix inversion (dashed) and approximate Jacobi with different number of iterations (colored). For reference, ChebNet is shown (dotted).

Table 5.1. Test accuracy obtained with different methods on the MNIST dataset. In **black** / **red** / **blue** performance of the 1st / 2nd / 3rd best model.

Model	Order	Accuracy	#Params
ChebNet	25	**99.14%**	1.66M
$\text{CayleyNet}_{11 \text{ Jacobi iter}}$	12	**99.18%**	1.66M
LeNet5	-	**99.33%**	1.66M

ChebNet and CayleyNet are shown in Figure 5.2. 0.1776 +/- 0.06079 sec and 0.0268 +/- 0.00841 sec are respectively required by CayleyNet and ChebNet for analyzing a batch of 100 images at test time.

Community detection. To demonstrate the ability of CayleyNet to focus only on particular frequency bands, we introduce here a second graph classification problem. Namely, a community detection task. We generated for this purpose a synthetic graph built by 15 different communities, vertices belonging to the same community are strongly connected to each other, and sparsely connected to other communities (Figure 5.5). On this graph, we generate noisy step signals, defined as $f_i = 1 + \sigma_i$ if i belongs to the community, and $f_i = \sigma_i$ otherwise, where $\sigma_i \sim \mathcal{N}(0, 0.3)$ is Gaussian i.i.d. noise. The goal is to classify each signal according to the community it belongs to. The neural network architecture used for this task consisted of a spectral

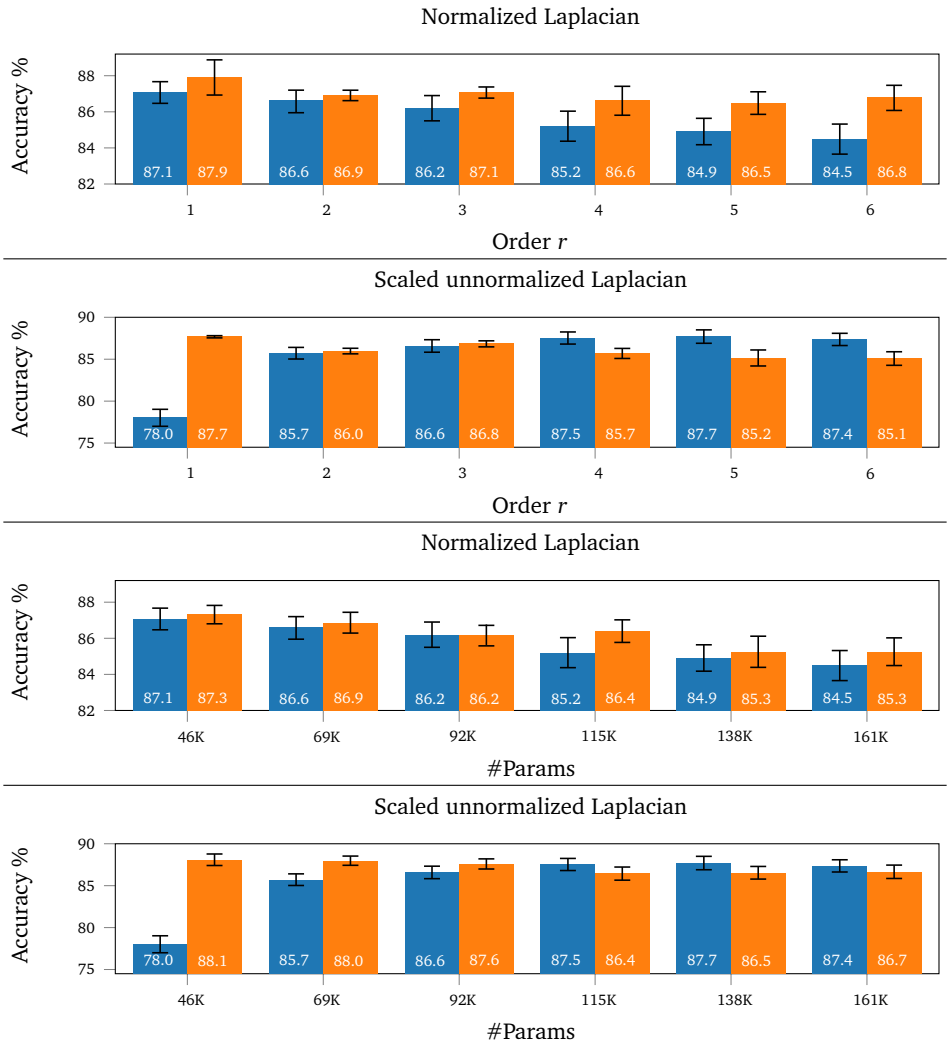


Figure 5.4. ChebNet (blue) and CayleyNet (orange) test accuracies obtained on the CORA dataset for different polynomial orders. Polynomials with complex coefficients (top two) and real coefficients (bottom two) have been exploited with CayleyNet in the two analysis. Orders 1 to 6 have been used in both comparisons.

convolutional layer (based on Chebyshev or Cayley filters) with 32 output features, a mean pooling layer, and a softmax classifier for producing the final classification into one of the 15 classes. No regularization has been exploited in this setting.

As the first 15 eigenfunctions of the Laplace operator are almost piece-wise constant over the different communities, spectral filters able to produce different coefficients for such eigenfunctions result in output signals with different behavior dependently on the community C where the input is localized. To understand why this might be the case, consider a filter that produces a null spectral response for all frequencies but frequency $t \in \{0, \dots, 15\}$. The behavior of such filter can be described by matrix $\boldsymbol{\phi}_t g(\lambda_t) \boldsymbol{\phi}_t^T$, and the output of a graph convolutional layer receiving in input the noisy step function $\mathbf{f}$ equals:

$$\tilde{\mathbf{f}} = \mathbf{G}\mathbf{f} = \boldsymbol{\phi}_t g(\lambda_t)\boldsymbol{\phi}_t^T \mathbf{f}. \tag{5.9}$$

It is easy to see how $\tilde{\mathbf{f}}$ simply correspond to a scaled version of $\boldsymbol{\phi}_t$, whose $\mathbb{L}_2$ norm depends on C (as $\boldsymbol{\phi}_t^T \mathbf{f} \simeq \sum_{i \in \mathcal{V}_C} \phi_{t,i} f_i \simeq |\mathcal{V}_C| \overline{\phi}_{t,C}$ with $\mathcal{V}_C$ the vertex set of the community where $\mathbf{f}$ is almost equal to 1, and $\overline{\phi}_{t,C}$ the average value of $\boldsymbol{\phi}_t$ on $\mathcal{V}_C$, we have $\tilde{\mathbf{f}} \simeq \boldsymbol{\phi}_t g(\lambda_t) |\mathcal{V}_c| \overline{\phi}_{t,C}$). Feeding in input $\tilde{\mathbf{f}}$ to a ReLU layer and averaging the signal over the entire graph yields in turn a scalar value that depends on the community C that supports $\mathbf{f}$, thus providing a meaningful descriptor for identifying which community is supporting $\mathbf{f}$. It is worth noting how such a behavior could not be possible with filters producing a similar spectral response for all the first 15 frequencies of the graph, as the spectrum of noisy step functions is almost entirely localized on the first 15 eigenfunctions, and the output of the convolutional layer would equal in this scenario a scaled version of $\mathbf{f}$ that depends only on the filter but not on the target community C (as $g(\lambda_t)$ would be almost constant for $0 \leq t \leq 15$).

As a result of the above, filters implemented with Chebyshev polynomials struggle at producing meaningful descriptors for the considered classification task (large polynomial orders are indeed required to produce different coefficients for the first 15 frequencies), while Cayley filters learn instead to specialize on the low-frequency band that discriminate well the communities and achieve good results even for low polynomial orders (Figure 5.5, second to the top). Figure 5.5 shows in the two bottom rows the spectral response of filters implemented with both Chebyshev and Cayley polynomials to illustrate the different behavior of the two classes of filters.

Finally, to evaluate the effect of the Jacobi method on the effectiveness of Cayley filters, we further measured the classification performance of CayleyNet with various numbers of Jacobi iterations on our community detection task (Figure 5.3). As it possible to see, CayleyNet consistently ouperforms ChebNet for any number of Jacobi iterations and for any possible polynomial order we considered (even just one Jacobi iteration is enough to consistently achieve better performance).

Complexity. We experimentally validated the computational complexity of our model applying filters of different order r to synthetic 15-community graphs of different size n using exact matrix inversion and approximation with different number of Jacobi iterations (Figure 5.6). All times have been computed running 30 times the considered models and averaging the final results. As expected, approximate inversion guarantees $\mathcal{O}(n)$ complexity.

Citation networks. To investigate the performance of CayleyNet on a more popular task in the literature, we next addressed the problem of vertex classification on graphs using the CORA citation graph that we introduced earlier for our MoNet architecture. As it was the case for MoNet, the task is to classify each vertex into one of 7 groundtruth classes. Since the setting of the problem is transductive learning, the features of all vertices are known, but labels are given just for a subset of the nodes. The model is trained by minimizing the label error at the nodes with known labels. After training, the model is tested over the nodes in which the labels were unknown during training.

We analyze the performance of our model in two different settings in our experiments: the classic semi-supervised problem presented in [138, 184, 252] with 140 training samples, 500 validation samples and 1,000 test samples and a relaxed version of this that exploits 1,708 vertices for training, 500 for validation and 500 for testing. We opted for a larger amount of

Table 5.2. Test accuracy of different methods on the planetoid split [270] of the CORA dataset. GCNNs are listed at the bottom of the table. In **black** / **red** / **blue** performance of the 1st / 2nd / 3rd best model.

Method	Accuracy	#Params
ManiReg [25]	59.5%	-
SemiEmb [261]	59.0%	-
LP [283]	68.0%	-
DeepWalk [202]	67.2%	-
Planetoid [270]	75.7%	-
CayleyNet$_{64 \text{ features}}$	81.0 ± 0.5 %	92K
GCN [138]	81.6 ± 0.4 %	23K
MoNet [184]	**81.7 ± 0.5 %**	23K
CayleyNet$_{16 \text{ features}}$	**81.9 ± 0.7 %**	23K
GAT [252]	**83.0 ± 0.7** %	92K

training samples in our second experiment, in order to provide an estimate of the quality of CayleyNet in a situation that is less prone to overfitting. Cayley operators with matrix inversion have been considered in both settings for our solution.

For the sake of simplicity, on the standard split, two versions of CayleyNet that reproduce the architectures used in [138, 184] and [252] have been implemented: a lightweight architecture exploiting two convolutional layers with 16 and 7 output features, and a heavier solution producing 64 and 7 output features from the two layers. The structure of our CayleyNets have thus been fixed a priori in these experiments, and it is possible that even better performance could be achieved tuning our architecture further. Normalized Laplacian has been used for building our spectral filters. Adam with learning rate equal to $5 \cdot 10^{-3}$, dropout probability $p = 0.6$ and weight decay coefficient $\gamma = 5 \cdot 10^{-4}$ have been used for training. Table 5.2 presents the results we obtained with our solution, average performance over 50 runs are reported to guarantee accurate estimates. Our lighter version of CayleyNet achieves similar performance to GCN and MoNet, while being defeated by GAT. Our heavier CayleyNet shows instead a drop in performance, likely because of overfitting.

On our extended split, we analyze the behavior of CayleyNet and ChebNet for a variety of different polynomial orders. Two spectral convolutional layers with 16 and 7 outputs features have been used for implementing the two architectures. Adam with learning rate equal to 10^{-3}, dropout probability $p = 0.5$ and weight decay with coefficient $\gamma = 5 \cdot 10^{-4}$ have been used for training. Figure 5.4 presents the results of our analysis. Since ChebNet requires Laplacians with spectra bounded in $[-1, 1]$, we consider both the normalized Laplacian, and the scaled combinatorial Laplacian ($2\mathbf{\Delta}/\lambda_{max} - \mathbf{I}$), where $\mathbf{\Delta}$ is the combinatorial Laplacian and λ_{max} is its largest eigenvalue. For fair comparison, we fix the order of the filters (top two figures) and the overall number of network parameters (bottom two figures). In the bottom figures, the Cayley filters are restricted to even cosine polynomials by considering only real filter coefficients. The best CayleyNets consistently outperform the best ChebNets requiring at the same time less parameters (CayleyNet with order r and complex coefficients requires a number of parameters equal to ChebNet with order $2r$).

Table 5.3. Performance (RMSE) of different matrix completion methods on the MovieLens dataset. In **black** / **red** / **blue** performance of the 1st / 2nd / 3rd best model.

Method	RMSE
Global Mean	1.154
User Mean	1.063
Movie Mean	1.033
MC [49]	0.973
IMC [126, 268]	1.653
GMC [131]	0.996
GRALS [212]	0.945
sRGCNN$_{\text{Cheby,r=4}}$ [185]	**0.929**
sRGCNN$_{\text{Cheby,r=8}}$ [185]	**0.925**
sRGCNN$_{\textbf{Cayley}}$	**0.922**

Recommender systems. In our final experiment, we applied CayleyNet for solving the matrix completion problem that we illustrated in Chapter 4. As before, given a sparsely sampled matrix of scores assigned by users (columns) to items (rows), the task is to fill in the missing scores. The similarities between users and items are given in the form of column and row graphs, respectively. To evaluate the performance of CayleyNet, we repeated verbatim the previously presented experiment on the MovieLens dataset [180], simply replacing Chebyshev filters with Cayley filters. We used sRGCNN with Cayley filters of order $r = 4$ employing 15 Jacobi iterations. Adam with learning rate equal to 10^{-3} and regularization coefficient $\gamma = 10^{-10}$ have been used for training. The results are reported in Table 5.3. To present a complete comparison, we further extended the experiments reported Chapter 4 by training sRGCNN with ChebNets of order 8. This provides an architecture with same number of parameters as the exploited CayleyNet (23k coefficients). The version of sRGCNN based on Cayley polynomials outperforms all the competing methods, including the previous result with Chebyshev filters. sRGCNNs with Chebyshev polynomials of order 4 and 8 respectively require 0.0698 +/- 0.00275 sec and 0.0877 +/- 0.00362 sec at test time, sRGCNN with Cayley polynomials of order 4 and 15 jacobi iterations requires 0.165 +/- 0.00332 sec.

5.3 Exponential decay of Cayley filters (proof)

In order to prove the exponential decay of Cayley filters over a generic vertex m, we will use a bound on the approximation error obtained estimating $\mathbf{G}\boldsymbol{\delta}_m$ with the Jacobi method, which we show here first.

Approximation error bound. Note the following classical result for the approximation of $\mathbf{A}\mathbf{x} = \mathbf{b}$ using the Jacobi method: if the initial condition is $\mathbf{x}^{(0)} = \mathbf{0}$, then $(\mathbf{x} - \mathbf{x}^{(k)}) = \mathbf{J}^k\mathbf{x}$. In our case, note that if we start with initial condition $\tilde{\mathbf{y}}_j^{(0)} = 0$, the next iteration gives $\tilde{\mathbf{y}}_j^{(1)} = \mathbf{b}_j$, which is the initial condition from our construction. Therefore, since we are approximating $\mathbf{y}_j = \mathcal{C}(h\boldsymbol{\Delta})\tilde{\mathbf{y}}_{j-1}$ by $\tilde{\mathbf{y}}_j = \tilde{\mathbf{y}}_j^{(K)}$, we have:

$$\mathcal{C}(h\boldsymbol{\Delta})\tilde{\mathbf{y}}_{j-1} - \tilde{\mathbf{y}}_j = \mathbf{J}^{K+1}\mathcal{C}(h\boldsymbol{\Delta})\tilde{\mathbf{y}}_{j-1}; \tag{5.10}$$

where $\mathbf{J}^{K+1}$ appears on the right hand side instead of $\mathbf{J}^K$ as we impose $\tilde{\mathbf{y}}_j^{(0)} = \mathbf{b}_j$ in our approach. Define the (relative) approximation error in $\mathcal{C}(h\boldsymbol{\Delta})^j\mathbf{x}$ by:

$$e_j = \frac{\left\|\mathcal{C}^j(h\boldsymbol{\Delta})\mathbf{x} - \tilde{\mathbf{y}}_j\right\|_2}{\|\mathcal{C}^j(h\boldsymbol{\Delta})\mathbf{x}\|_2}. \tag{5.11}$$

By the triangle inequality, by the fact that $\mathcal{C}(h\boldsymbol{\Delta})$ is unitary (thus $\|\mathcal{C}(h\boldsymbol{\Delta})\mathbf{x}\|_2 = \|\mathbf{x}\|_2$), and by (5.10):

$$\begin{aligned}
e_j \leq & \frac{\left\|\mathcal{C}^j(h\boldsymbol{\Delta})\mathbf{x} - \mathcal{C}(h\boldsymbol{\Delta})\tilde{\mathbf{y}}_{j-1}\right\|_2}{\|\mathcal{C}^j(h\boldsymbol{\Delta})\mathbf{x}\|_2} + \frac{\left\|\mathcal{C}(h\boldsymbol{\Delta})\tilde{\mathbf{y}}_{j-1} - \tilde{\mathbf{y}}_j\right\|_2}{\|\mathcal{C}^j(h\boldsymbol{\Delta})\mathbf{x}\|_2} \\
= & \frac{\left\|\mathcal{C}^{j-1}(h\boldsymbol{\Delta})\mathbf{x} - \tilde{\mathbf{y}}_{j-1}\right\|_2}{\|\mathcal{C}^{j-1}(h\boldsymbol{\Delta})\mathbf{x}\|_2} + \frac{\left\|\mathbf{J}^{K+1}\mathcal{C}(h\boldsymbol{\Delta})\tilde{\mathbf{y}}_{j-1}\right\|_2}{\|\mathbf{x}\|_2} \\
\leq & e_{j-1} + \left\|\mathbf{J}^{K+1}\right\|_2 \frac{\left\|\mathcal{C}(h\boldsymbol{\Delta})\tilde{\mathbf{y}}_{j-1}\right\|_2}{\|\mathbf{x}\|_2} \\
= & e_{j-1} + \left\|\mathbf{J}^{K+1}\right\|_2 \frac{\left\|\tilde{\mathbf{y}}_{j-1}\right\|_2}{\|\mathbf{x}\|_2} \\
\leq & e_{j-1} + \left\|\mathbf{J}^{K+1}\right\|_2 (1 + e_{j-1}) \\
= & (1 + \|\mathbf{J}^{K+1}\|_2) e_{j-1} + \|\mathbf{J}^{K+1}\|_2;
\end{aligned} \tag{5.12}$$

where the last inequality is due to:

$$\begin{aligned}
\left\|\tilde{\mathbf{y}}_{j-1}\right\|_2 &\leq \left\|\mathcal{C}^{j-1}(h\boldsymbol{\Delta})\mathbf{x}\right\|_2 + \left\|\mathcal{C}^{j-1}(h\boldsymbol{\Delta})\mathbf{x} - \tilde{\mathbf{y}}_{j-1}\right\|_2 \\
&= \|\mathbf{x}\|_2 + \|\mathbf{x}\|_2 \, e_{j-1}.
\end{aligned} \tag{5.13}$$

If we use the normalized Laplacian for implementing Cayley filters, $\|\mathbf{J}\|_2 = \frac{h}{\sqrt{h^2+1}} < 1$. Since $\|\mathbf{J}^{K+1}\|_2 \leq \|\mathbf{J}\|_2^{K+1}$, if we recursively unravel (5.12) and we fix $\kappa = \|\mathbf{J}\|_2$, we obtain:

$$e_j \leq j\kappa^{K+1} + \mathcal{O}(\kappa^{2K+2}). \tag{5.14}$$

If we additionally impose $\|\tilde{\mathbf{y}}_j\|_2 = \|\mathbf{x}\|_2$[3], the error bound further reduces to $e_j \leq e_{j-1} + \left\|\mathbf{J}^{K+1}\right\|_2$, which in turn yields:

$$e_j \leq j\kappa^{K+1}. \tag{5.15}$$

If we use the combinatorial Laplacian for the implementation, using standard norm bounds (i.e. $\left\|\mathbf{J}^{K+1}\right\|_2 \leq \sqrt{n}\left\|\mathbf{J}^{K+1}\right\|_\infty$) and fixing $\kappa = \|\mathbf{J}\|_\infty = \frac{hd}{\sqrt{h^2d^2+1}} < 1$ (with $d = \max_j d_j$), we obtain:

$$\begin{aligned}
e_j \leq & e_{j-1} + \sqrt{n}\,\|\mathbf{J}\|_\infty^{K+1}(1 + e_{j-1}) \\
= & (1 + \sqrt{n}\kappa^{K+1}) e_{j-1} + \sqrt{n}\kappa^{K+1}.
\end{aligned} \tag{5.16}$$

For sufficiently large values of K, $\sqrt{n}\kappa^{K+1} < 1$, and the solution of the recurrent sequence becomes:

[3] As $\tilde{\mathbf{y}}_j$ is an approximation of $\mathcal{C}(h\boldsymbol{\Delta})^j\mathbf{x}$ (whose norm is equal to the one of $\mathbf{x}$), one can impose $\|\tilde{\mathbf{y}}_j\|_2 = \|\mathbf{x}\|_2$ in order to recreate more similar conditions to what we would have with matrix inversion.

$$e_j \leq j\sqrt{n}\kappa^{K+1} + \mathcal{O}(n\kappa^{2K+2}). \tag{5.17}$$

As for the case of the normalized laplacian, if the normalized version of the algorithm is used, this bound further reduces to:

$$e_j \leq j\sqrt{n}\kappa^{K+1}. \tag{5.18}$$

Letting $M_j = j\sqrt{n}$ for the combinatorial Laplacian, $M_j = j$ for the normalized one, and imposing $\|\tilde{\mathbf{y}}_j\|_2 = \|\mathbf{x}\|_2$, by the triangle inequality we have:

$$\begin{aligned}\frac{\left\|\mathbf{G}\mathbf{x} - \widetilde{\mathbf{G}\mathbf{x}}\right\|_2}{\|\mathbf{x}\|_2} \leq & 2\sum_{j=1}^{r}\left|c_j\right| \frac{\left\|\mathcal{C}^j(h\boldsymbol{\Delta})\mathbf{x} - \tilde{\mathbf{y}}_j\right\|_2}{\|\mathcal{C}^j(h\boldsymbol{\Delta})\mathbf{x}\|_2} \\ = & 2\sum_{j=1}^{r}\left|c_j\right| e_j \leq 2\sum_{j=1}^{r} M_j\left|c_j\right| \ \kappa^{K+1};\end{aligned} \tag{5.19}$$

which shows how convergence of the approximation algorithm is guaranteed for a sufficiently large number of Jacobi iterations, and the role of parameter h in the convergence. The smaller the value of h, the smaller κ we have and thus the smaller the approximation error we obtain.

Proof of Theorem 5.1.1. In this proof we approximate $\mathbf{G}\boldsymbol{\delta}_m$ by $\widetilde{\mathbf{G}\boldsymbol{\delta}}_m$. Note that the signal $\boldsymbol{\delta}_m$ is supported on one vertex, and in the calculation of $\widetilde{\mathbf{G}\boldsymbol{\delta}}_m$, each Jacobi iteration increases the support of the signal by 1-hop. Therefore, the support of $\widetilde{\mathbf{G}\boldsymbol{\delta}}_m$ is the $r(K+1)$-hop neighborhood $\mathcal{N}_m^{(r(K+1))}$ of m. Denoting $l = r(K+1)$, we get:

$$\begin{aligned}\left\|\mathbf{G}\boldsymbol{\delta}_m - \mathbf{G}\boldsymbol{\delta}_m|_{\mathcal{N}_m^{(l)}}\right\|_2 \leq & \left\|\mathbf{G}\boldsymbol{\delta}_m - \widetilde{\mathbf{G}\boldsymbol{\delta}}_m\right\|_2 + \left\|\widetilde{\mathbf{G}\boldsymbol{\delta}}_m - \mathbf{G}\boldsymbol{\delta}_m|_{\mathcal{N}_m^{(l)}}\right\|_2 \\ \leq & \left\|\mathbf{G}\boldsymbol{\delta}_m - \widetilde{\mathbf{G}\boldsymbol{\delta}}_m\right\|_2 + \left\|\widetilde{\mathbf{G}\boldsymbol{\delta}}_m - \mathbf{G}\boldsymbol{\delta}_m\right\|_2 \\ = & 2\left\|\mathbf{G}\boldsymbol{\delta}_m - \widetilde{\mathbf{G}\boldsymbol{\delta}}_m\right\|_2 \leq 4\overline{M}\kappa^{K+1}\left\|\boldsymbol{\delta}_m\right\|_2 = 4\overline{M}(\kappa^{1/r})^l;\end{aligned} \tag{5.20}$$

with $\overline{M} = \sum_{j=1}^{r} M_j\left|c_j\right|$.

5.4 Discussion

As it was the case for the Mixture Model Neural Network and RMGCNN/sRMGCNN, our CayleyNet was introduced back in 2017, and it represented one of the early works on spectral GCNNs. Following from the publication of our original paper (and the one of Defferrard et al. [72] before ours), several publications appeared in the literature proposing alternative rational and polynomial parametrizations of spectral filters. In this section, we discuss some of the most relevant methods that have recently been proposed in this direction. For a broader review of spectral methods, the interested reader is invited to refer to [36].

Rational spectral filters. At writing time, there are at least two recent works that use rational spectral filters for processing signals defined on graphs: *ARMA* of Bianchi et al. [32], and *ResolvNet* of Koke et al. [141].

ARMA is a spectral GCNN that uses approximations of ARMA filters to implement convolution on the provided graph. ARMA filters correspond to a generic class of Infinite Impulse Response (IIR) graph filters, whose spectral response can be defined as:

$$g_r(\lambda) = \frac{\sum_{j=0}^{r-1} p_j \lambda^j}{1 + \sum_{j=1}^{r} q_j \lambda^j}; \tag{5.21}$$

where $\{p_j \in \mathbb{R} | j \in \{0, \dots, r-1\}\}$ and $\{q_j \in \mathbb{R} | j \in \{1, \dots, r\}\}$ are parameters of the model. As filters implemented with Cayley polynomials of order r can be written as the ratio of two real polynomials of order $2r$ (Section 5.4.1):

$$g_{\mathbf{c},h}(\lambda) = \frac{2\sum_{j=0}^{r}(h^2\lambda^2+1)^{r-j}\left(\sum_{t=0}^{j}\mathfrak{Re}\{c_j\}\binom{2j}{2t}(h\lambda)^{2j-2t}i^{2t} - \sum_{t=1}^{j}\mathfrak{Im}\{c_j\}\binom{2j}{2t-1}(h\lambda)^{2j-2t+1}i^{2t}\right)}{(h^2\lambda^2+1)^r}; \tag{5.22}$$

ARMA filters of order $2r+1$ (and higher) generalize Cayley filters of order r, and they allow to implement (at the cost of a larger amount of parameters) a broader family of functions. As it emerges from (5.21), differently from Cayley polynomials, which are stable by construction for undirected graphs (as the eigenvalues of the Laplace operator are real on such domains), ARMA filters can show diverging behaviors if the poles of (5.21) match at least one eigenvalue of $\mathbf{\Delta}$ (or $\tilde{\mathbf{\Delta}}$). At the same time (similarly to what discussed in Section 5.1), the matrix form of (5.21) requires a matrix inversion to be implemented (equation (7) in [32]). To avoid such an expensive operation and guarantee stable filters at the same time, Bianchi et al. proposed to approximate the spectral response depicted in (5.21) using sequences of layers of the form:

$$\mathbf{X}^{(k+1)} = \sigma((\mathbf{I} - \tilde{\mathbf{\Delta}})\mathbf{X}^{(k)}\mathbf{W} + \mathbf{X}\mathbf{V}); \tag{5.23}$$

where $\mathbf{W}$ and $\mathbf{V}$ are two matrices of learnable weights, σ is a non-linearity (e.g. ReLU) and $\mathbf{X}^{(k)}$ corresponds with the output of the k-th layer. The output of multiple columns of layers is averaged in [32] to achieve filters with a variety of different responses (low-pass, band-pass or high-pass filters). As it appears from (5.23), differently from our approximation of Cayley polynomials (which applies multiple Jacobi iterations, or diffusion steps, before multiplying the output with a matrix of weights), ARMA mixes the input channels (through the weight matrices $\mathbf{W}$ and $\mathbf{V}$) and applies the non-linearity σ at each step of diffusion. Thanks to the extra-richness this provides to the approximation process (and the fact that filters are produced averaging sequences of layers of the form (5.23), and not stacking them on top of each other), the solution proposed in [32] appears to require less consecutive projections on the diffusion operator, compared to CayleyNet, to produce meaningful spectral responses, and achieves better performance especially on graph classification tasks[4].

[4] If large amounts of Jacobi iterations, or large values of r, are used to compute Cayley polynomials, the implemented filters reduce to something akin a high-order polynomials of $\mathbf{J}$ [32]. As filters implemented with high-order polynomials can be more unstable than low-order ones across similar graphs [32][134, Theorem 2], this can result in less robust performance over unseen conditions.

Turning our attention to ResolvNet, in [141] Koke et al. studied the behavior of GCNNs on multi-scale domains[5], and highlighted in particular how architectures able to: i) avoid disconnected propagation graphs (which might appear if one uses, for instance, the symmetric normalized adjacency matrix for diffusion on multi-scale domains [141]), and ii) realize graph-wise embeddings that are similar for graphs denoting the same object at multiple resolutions; can be realized resorting to polynomials of the *resolvent* of the graph Laplacian (i.e. $R_z(\mathbf{\Delta}) = (\mathbf{\Delta} - z\mathbf{I})^{-1}$). The filters implemented in [141] thus take the form:

$$g_{z,\theta}(\mathbf{\Delta}) = \sum_{k=a}^{K} \theta_k [(\mathbf{\Delta} - z\mathbf{I})^{-1}]^k; \tag{5.24}$$

where both $z \in \mathbb{R}_{<0}$ and $a \in \{0, 1\}$ are hyperparameters of the model[6]. If different scales are well separated in the input domain (i.e. $\lambda_1^{(high)} >> \lambda_{max}^{(reg)}$), it can be shown that filters implemented as in (5.24) are close to filters implemented (with the same set of weights, and formulation) on a coarser version of the given graph that is obtained replacing clusters of densely connected points with single nodes. In this situation, signals defined on nodes of strongly connected clusters tend to be homogenized by $g_{z,\theta}(\mathbf{\Delta})$, and diffusion is realized across such subgraphs as if the domain had only one scale [141]. In the experimental evaluation of [141], this specific inductive bias allowed ResolvNet to favorably perform in node classification tasks defined on homophilic datasets (where similar, tightly connected, nodes are pushed to have similar embeddings), as well as in molecular prediction problems that depend on long range interactions (thanks to the ability of the model to propagate information across weakly connected clusters of nodes).

Polynomial spectral filters. Moving away from rational functions (and approximations of these), a variety of works appeared in the literature in the last few years that use some form of polynomial to realize spectral filters [58, 111, 112, 257, 106]. In [58], Chien et al. introduced *GPR-GNN*, a variation of APPNP [97] where a single polynomial filter of $\bar{\mathbf{A}}$ (Chapter 2), implemented with the monomial basis and with coefficients shared across feature channels, is applied on top of node-wise descriptors obtained refining the input node features with a shared MLP. As the number of parameters required by the diffusion process depends only on the number of propagation steps used in the model (as a single filter is applied on all feature channels in [58]), large polynomial orders can be used to produce rich spectral responses in GPR-GNNs without requiring a significant amount of coefficients. This, in turn, leads to an interesting balance between expressivity and robustness of the model, and allows GPR-GNN to achieve good performance at inference time, even when very few labels are provided for training.

In [111], He et al. introduced *BernNet*, a spectral GCNN that resorts to Bernstein polynomials for implementing spectral filters on graphs. Bernstein polynomials of order K can be defined over the domain of the normalized graph Laplacian eigenvalues (i.e. the interval $[0, 2]$) as $g(\tilde{\lambda}) = \sum_{k=0}^{K} \omega_k b_k^K(\tilde{\lambda})$, with $b_k^K(\tilde{\lambda}) = \frac{1}{2^K}\binom{K}{k}(2 - \tilde{\lambda})^{K-k}\tilde{\lambda}^k$. As each $b_k^K(.)$ corresponds to a "bump" centered around $\frac{2k}{K}$, the value of each ω_k directly affects the spectral response of the filter in a neighborhood of said frequency, and the shape of the spectral response can thus be controlled simply imposing some form of constraint on the value of $\boldsymbol{\omega}$. Filters with spectral

[5] A multi-scale graph can be informally defined as a domain with edges distributed over (at least) two scales: a large scale representing strong connections within clusters, and a regular scale denoting weaker connections across clusters. Two scale graphs are graphs whose Laplacian can be decomposed as $\mathbf{\Delta} = \mathbf{\Delta}_{high} + \mathbf{\Delta}_{reg}$, with $\lambda_1^{(high)} >> \lambda_{max}^{(reg)}$ [141].

[6] As it was the case for Cayley filters, filters implemented in [141] are a particular setting of (5.21).

response in $[0,1]$, which allow to produce signals optimizing a target energy function, were in particular introduced in [111], simply imposing $\omega_k \geq 0 \; \forall k \in \{0,\dots,K\}$ and applying a suitable normalization step.

The additional level of control, Bernstein polynomials allow to achieve, unfortunately comes at the cost of a quadratic complexity in K, as K projections over the Laplace operator need to be carried out for each $b_k^K(.)$ (equation 3 in [111]). In a follow up work [112], He et al. showed that filters enjoying a similar form of regularization, but better complexity (i.e. $\mathcal{O}(K^2+K|\mathcal{E}|)$), can also be obtained with Chebyshev polynomials, simply fixing the spectral response of the target filter $h(.)$ in correspondence of Chebyshev nodes $x_j = \cos(\frac{j+1/2}{K+1}\pi) \; \forall j = 0,\dots,K$, and resorting to Chebyshev interpolation for estimating the value of polynomial coefficients $\boldsymbol{\omega}$ (*ChebNetII*). Variations of GPR-GNN implemented with Chebyshev interpolation / Bernstein polynomials, and using the regularization we discussed above for BernNet, outperformed the original implementation based on monomials in [111, 112]. ChebNetII appeared in particular as the better performing solution of the two, possibly because of better convergence properties (see [257, 106] next). A scalable implementation of spectral filters, based on the same pre-processing step we leveraged a few years earlier in SIGN (Chapter 6), was additionally introduced in [112] to solve prediction tasks on web-scale graphs involving millions of nodes and billions of edges.

Finally, in [257] Wang et al. showed that polynomials that are orthonormal, w.r.t. a weight function that is dependent on the spectrum of a provided input signal, can allow linear GCNNs (i.e. GCNNs of the form $\mathbf{Y} = g(\boldsymbol{\Delta})\mathbf{XW}$, where $g(.)$ is a real-valued polynomial) to maximize the convergence rate of their filters' coefficients in the proximity of a loss minimum. In line with this, the authors introduced *JacobiConv*, a linear spectral GCNN with filters implemented through Jacobi polynomials (which are a generalization of Chebyshev polynomials). As Jacobi polynomials are orthogonal w.r.t. weight function $(1-\lambda)^a(1+\lambda)^b$ in $(-1,1)$ (and they can be made orthonormal via a suitable normalization [78]), they represent a flexible basis that can be adapted for a variety of input signals ($a \geq -1$ and $b \geq -1$ are hyperparameters of the model that can be suitably tuned here). It should be noted that, while Jacobi polynomials do provide a versatile family of functions for implementing spectral filters, there exists (of course) weight functions that they cannot adopt. In a follow up work [106], Guo et al. highlighted how *any* orthonormal polynomial basis can be obtained defining the elements of the basis via a three-term recurrence of the form:

$$\sqrt{\beta_{k+1}}p_{k+1}(x) = (x-\gamma_k)p_k(x) - \sqrt{\beta_k}p_{k-1}(x); \tag{5.25}$$

with $\beta_k \in \mathbb{R}^+$ and $\gamma_k \in \mathbb{R}$. Additionally, any series $\{p_0(.), p_1(.), \dots, p_K(.)\}$ that satisfies the above recurrence was shown to correspond with an orthonormal polynomial basis in [106] (*Favard's theorem*). As a result of this, the three-term relation highlighted above offers a continuous, and easily explorable, parameter space for implementing orthonormal polynomials, and a more general version of the work of Wang et al. (which is not bounded to polynomials that are orthonormal only to $(1-\lambda)^a(1+\lambda)^b$) can be obtained swapping Jacobi polynomials with polynomials satisfying (5.25). In line with this, the authors introduced *FavardGNN* and *OptBasisGNN*, two different GCNNs where the terms of the recurrence are either learned at training time together with the coefficients of the filters (FavardGNN), or are set to optimize the convergence rate, that a linear GCNN would have, in the proximity of a minimum of the square loss (OptBasisGNN). As it was the case for JacobiConv before them, instances of FavardGNN and OptBasisGNN achieved comparable or superior performance w.r.t. similar ar-

chitectures implemented with other polynomial bases. OptBasisGNN appeared in particular as the more scalable solution in [106], as (similarly to what was done in [112]) a pre-processing step analogous to the one used in SIGN can be implemented with such approach, to reduce the complexity of the model to the one of an MLP.

Spectral filters and directed graphs. So far in this document we addressed how spectral filters can be implemented and applied to signals defined on undirected domains. However, in many situations, one might actually need to process signals that are defined over directed domains (e.g. the user-user follow networks of Twitter / X and Instagram). In Chapter 2, we highlighted the connection between the eigenfunctions (and eigenvalues) of the graph Laplacian for undirected graphs and the modes (and their related frequencies) of the classic Fourier Transform. Unfortunately, due to the asymmetry of the Laplace operator, generalizing the Fourier Transform on directed domains is not as straightforward. The graph Laplacian (as we defined it in Section 2.2) can indeed be defective, and thus not admit a valid eigendecomposition. Additionally, even if $\mathbf{\Delta}$ is diagonalizable, the eigenvalues $\mathbf{\Lambda}$ are not necessarily real, and they do not provide a notion of how variable the associated eigenfunctions are, as it was the case for undirected graphs[7]. As a result of this, the connection between the eigendecomposition of the graph Laplacian and the classic Fourier basis is generally lost for directed networks, and we are left with the question on how to possibly generalize convolution in the spectrum on such domains. To solve this issue, Zhang et al. [281] proposed to resort to the so-called *magnetic Laplacian* [92], a generalized version of the combinatorial Laplacian for directed graphs (a normalized version of such operator is available as well). The magnetic Laplacian $\mathbf{\Delta}^{(mag)}$ is defined as:

$$\mathbf{\Delta}^{(mag)} = \mathbf{D}^{(s)} - \mathbf{A}^{(s)} \odot \exp(i\mathbf{\Theta}^{(q)}); \tag{5.26}$$

where $\mathbf{D}^{(s)}$ is the diagonal degree matrix of the associated undirected graph, $\mathbf{A}^{(s)}$ is its corresponding adjacency matrix, and $\mathbf{\Theta}^{(q)} = 2\pi q(\mathbf{A} - \mathbf{A}^T)$ (with $\mathbf{A}$ the directed adjacency matrix). It corresponds with a complex hermitian positive semi-definite matrix, which encodes the direction of the edges via the phase of each entry. $\mathbf{\Delta}^{(mag)}_{i,j}$ is real if an edge exists from i to j and viceversa, $\mathbf{\Delta}^{(mag)}_{i,j}$ has phase equal to $2\pi q$ if an edge exists from i to j but not from j to i, and $\mathbf{\Delta}^{(mag)}_{i,j}$ has phase equal to $-2\pi q$ if an edge exists from j to i but not from i to j. In virtue of being hermitian, $\mathbf{\Delta}^{(mag)}$ always admits an eigendecomposition, the eigenvectors of $\mathbf{\Delta}^{(mag)}$ are complex (and can be chosen to be orthogonal) and describe the direction of the edges, while the eigenvalues are real and provide a measure of how much the eigenvectors change (in modulo and phase) over the graph[8]. As a result of this, the constructions that we use for realizing spectral filters on undirected graphs (e.g. Chebyshev polynomials, Cayley polynomials, Bernstein polynomials, ...) can be extended to directed domains, while maintaining similar interpretations, simply swapping the combinatorial Laplacian for the magnetic one. In a follow up work [114], He et al. additionally showed how the magnetic Laplacian can be extended to signed

[7] Let $||\boldsymbol{\phi}_k||_2 = 1$, since $\mathbf{\Delta} \neq \mathbf{\Delta}^T$, $\lambda_k = <\boldsymbol{\phi}_k, \mathbf{\Delta}\boldsymbol{\phi}_k> = \sum_{(i,j)\in\mathcal{E}} a_{i,j}(|\phi_{k,i}|^2 - \bar{\phi}_{k,i}\phi_{k,j})$ cannot be reduced to a sum of squared differences of the values of $\boldsymbol{\phi}_k$ over the arcs of the graph. Additionally, the terms of the summation used to define λ_k (which measure how $\boldsymbol{\phi}_k$ varies over the graph) can point in different directions (the real or imaginary part can have different signs), and can thus compensate each other in the calculation of the eigenvalue.

[8] With a few derivations (Section 5.4.2) it can be shown that $\lambda_k = <\boldsymbol{\phi}_k, \mathbf{\Delta}^{(mag)}\boldsymbol{\phi}_k> = \frac{1}{2}\sum_{i,j=0}^{|\mathcal{V}|-1} a^{(s)}_{i,j}(|\phi_{k,i}|^2 + |\phi_{k,j}|^2 - 2|\phi_{k,i}||\phi_{k,j}|\cos(\theta^{(q)}_{i,j} - (\angle\phi_{k,i} - \angle\phi_{k,j})))$. An eigenvector $\boldsymbol{\phi}_k$ thus shows a small λ_k if it assumes complex values with similar magnitudes for adjacency nodes, and angles that reproduce the differences set in $\mathbf{\Theta}^{(q)}$.

directed networks (i.e. directed graphs built by positive and negative relations), in order to further generalize spectral approaches to this class of domains.

A generalization of spectral approaches to directed graphs, which use generalized Laplacian matrices computed resorting to page rank scores, has additionally been proposed in [168]. It should be noted however that the directional information of the edges is not preserved in the diffusion operator in such construction.

5.4.1 Cayley filters as real rational functions

Let $\tilde{c}_0 = \frac{c_0}{2}$ and $\tilde{c}_j = c_j \ \forall j \in \{1,\dots,r\}$, the spectral response of filters implemented with Cayley polynomials can be rewritten as the ratio of two real valued polynomials:

$$
\begin{aligned}
g_{\mathbf{c},h}(\lambda) &= c_0 + 2\mathfrak{Re}\{\sum_{j=1}^{r} c_j (h\lambda - i)^j (h\lambda + i)^{-j}\} \\
&= 2\mathfrak{Re}\{\sum_{j=0}^{r} \tilde{c}_j \left(\frac{h\lambda - i}{h\lambda + i}\right)^j\} \\
&= 2\mathfrak{Re}\{\sum_{j=0}^{r} \tilde{c}_j \left(\frac{(h\lambda - i)(h\lambda - i)}{(h\lambda + i)(h\lambda - i)}\right)^j\} \\
&= 2\mathfrak{Re}\{\sum_{j=0}^{r} \tilde{c}_j \frac{(h\lambda - i)^{2j}}{(h^2\lambda^2 + 1)^j}\} \\
&= 2\mathfrak{Re}\{\sum_{j=0}^{r} \tilde{c}_j \frac{(h\lambda - i)^{2j}(h^2\lambda^2 + 1)^{r-j}}{(h^2\lambda^2 + 1)^r}\} \\
&= 2\mathfrak{Re}\{\frac{\sum_{j=0}^{r}(h^2\lambda^2 + 1)^{r-j}\left(\sum_{t=0}^{2j} \tilde{c}_j \binom{2j}{t}(h\lambda)^{2j-t}(-i)^t\right)}{(h^2\lambda^2 + 1)^r}\} \\
&= \frac{2\sum_{j=0}^{r}(h^2\lambda^2 + 1)^{r-j}\left(\sum_{t=0}^{j} \mathfrak{Re}\{\tilde{c}_j\}\binom{2j}{2t}(h\lambda)^{2j-2t}(-i)^{2t} + \sum_{t=1}^{j} i\mathfrak{Im}\{\tilde{c}_j\}\binom{2j}{2t-1}(h\lambda)^{2j-2t+1}(-i)^{2t-1}\right)}{(h^2\lambda^2 + 1)^r} \\
&= \frac{2\sum_{j=0}^{r}(h^2\lambda^2 + 1)^{r-j}\left(\sum_{t=0}^{j} \mathfrak{Re}\{\tilde{c}_j\}\binom{2j}{2t}(h\lambda)^{2j-2t} i^{2t} - \sum_{t=1}^{j} \mathfrak{Im}\{\tilde{c}_j\}\binom{2j}{2t-1}(h\lambda)^{2j-2t+1} i^{2t}\right)}{(h^2\lambda^2 + 1)^r}.
\end{aligned}
\tag{5.27}
$$

5.4.2 Eigenvalues and eigenvectors of the magnetic Laplacian

Here we show how eigenvalues and eigenvectors of the magnetic Laplacian relate one to the other. In the derivation we use the classic result $\cos(\alpha-\beta) = \cos(\alpha)\cos(\beta) + \sin(\alpha)\sin(\beta)$, and the fact that $\theta^{(q)}_{i,j} = -\theta^{(q)}_{j,i}$. The eigenvector $\boldsymbol{\phi}_k$ is also assumed to have $\mathbb{L}_2$ norm equal to 1. For the sake of clarity, the subscript of λ_k and $\boldsymbol{\phi}_k$ is dropped in the proof.

$$
\begin{aligned}
\lambda &= <\boldsymbol{\phi}, \boldsymbol{\Delta}^{(mag)}\boldsymbol{\phi}> = \sum_{i,j=0}^{|\mathcal{V}|-1} a^{(s)}_{i,j}\left(|\phi_i|^2 - \bar{\phi}_i \phi_j e^{i\theta^{(q)}_{i,j}}\right) \\
&= \sum_{i,j|i\leq j} a^{(s)}_{i,j}\left(|\phi_i|^2 + |\phi_j|^2 - \bar{\phi}_i \phi_j e^{i\theta^{(q)}_{i,j}} - \bar{\phi}_j \phi_i e^{i\theta^{(q)}_{j,i}}\right)
\end{aligned}
\tag{5.28}
$$

$$= \sum_{i,j|i\leq j} a_{i,j}^{(s)} \left(|\phi_i|^2 + |\phi_j|^2 - |\phi_i||\phi_j| e^{-i(\angle\phi_i - \angle\phi_j)} e^{i\theta_{i,j}^{(q)}} - |\phi_i||\phi_j| e^{i(\angle\phi_i - \angle\phi_j)} e^{-i\theta_{i,j}^{(q)}} \right)$$

$$= \sum_{i,j|i\leq j} a_{i,j}^{(s)} \Bigg(|\phi_i|^2 + |\phi_j|^2 - |\phi_i||\phi_j| \left(\cos(\angle\phi_i - \angle\phi_j) - i\sin(\angle\phi_i - \angle\phi_j)\right)\left(\cos(\theta_{i,j}^{(q)}) + i\sin(\theta_{i,j}^{(q)})\right) +$$

$$- |\phi_i||\phi_j| \left(\cos(\angle\phi_i - \angle\phi_j) + i\sin(\angle\phi_i - \angle\phi_j)\right)\left(\cos(\theta_{i,j}^{(q)}) - i\sin(\theta_{i,j}^{(q)})\right) \Bigg)$$

$$= \sum_{i,j|i\leq j} a_{i,j}^{(s)} \left((|\phi_i|^2 + |\phi_j|^2 - 2|\phi_i||\phi_j| \left(\cos(\angle\phi_i - \angle\phi_j)\cos(\theta_{i,j}^{(q)}) + \sin(\angle\phi_i - \angle\phi_j)\sin(\theta_{i,j}^{(q)})\right) \right)$$

$$= \sum_{i,j|i\leq j} a_{i,j}^{(s)} \left(|\phi_i|^2 + |\phi_j|^2 - 2|\phi_i||\phi_j| \cos\left(\theta_{i,j}^{(q)} - (\angle\phi_i - \angle\phi_j)\right) \right)$$

$$= \frac{1}{2} \sum_{i,j} a_{i,j}^{(s)} \left(|\phi_i|^2 + |\phi_j|^2 - 2|\phi_i||\phi_j| \cos\left(\theta_{i,j}^{(q)} - (\angle\phi_i - \angle\phi_j)\right) \right).$$

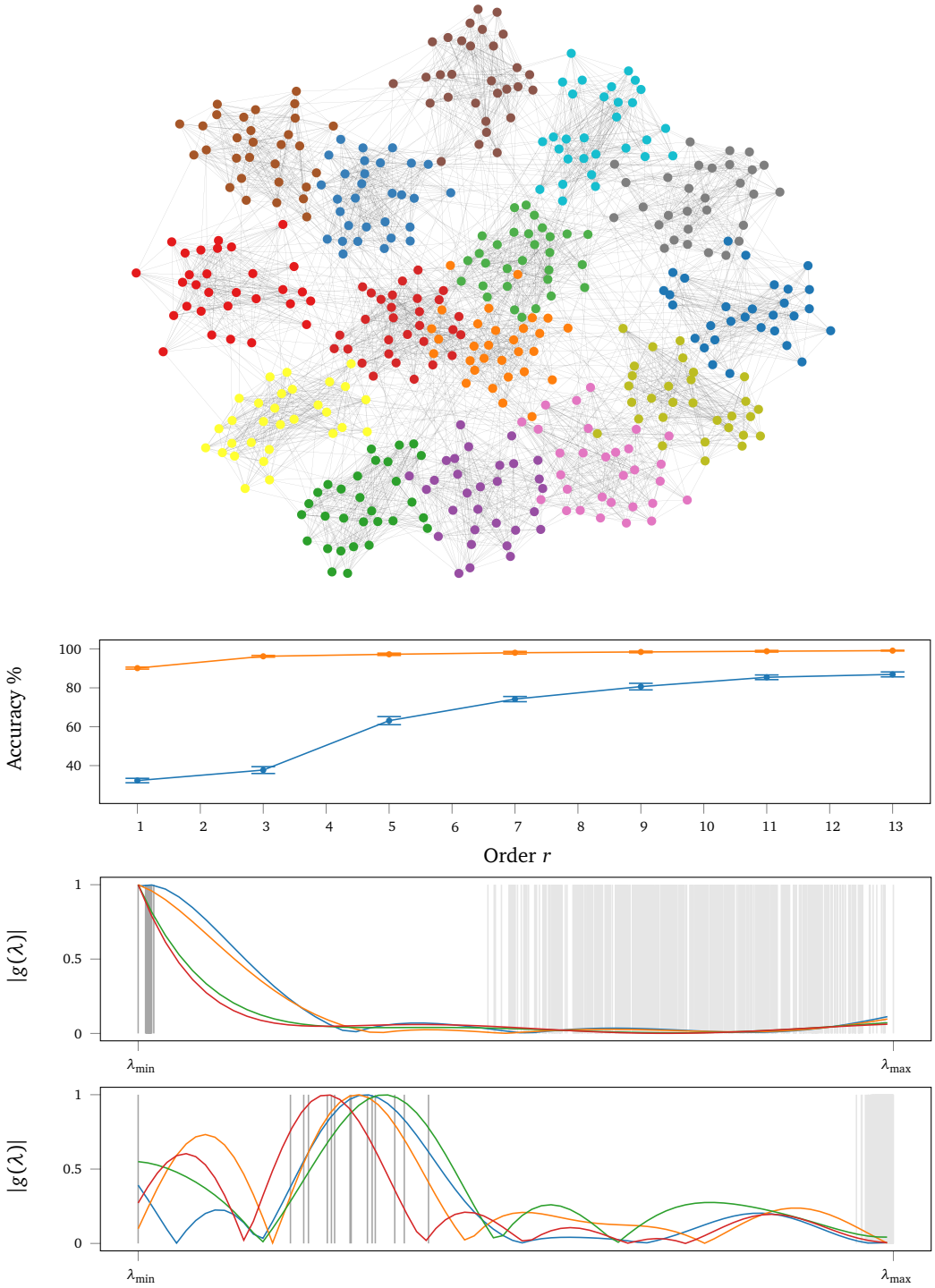


Figure 5.5. Top: synthetic 15-communities graph. Second to the top: community detection accuracy of ChebNet and CayleyNet. Bottom two: normalized responses of four different filters learned by ChebNet (top) and CayleyNet (bottom), each response is in a different color. Grey vertical lines represent the frequencies of the normalized Laplacian ($\tilde{\lambda} = 2\lambda_n^{-1}\lambda - 1$ for ChebNet and $C(\lambda) = (h\lambda - i)/(h\lambda + i)$ unrolled to a real line for CayleyNet). Note how, thanks to spectral zoom property, Cayley filters can focus on the band of low frequencies (dark grey lines) containing most of the information about communities.

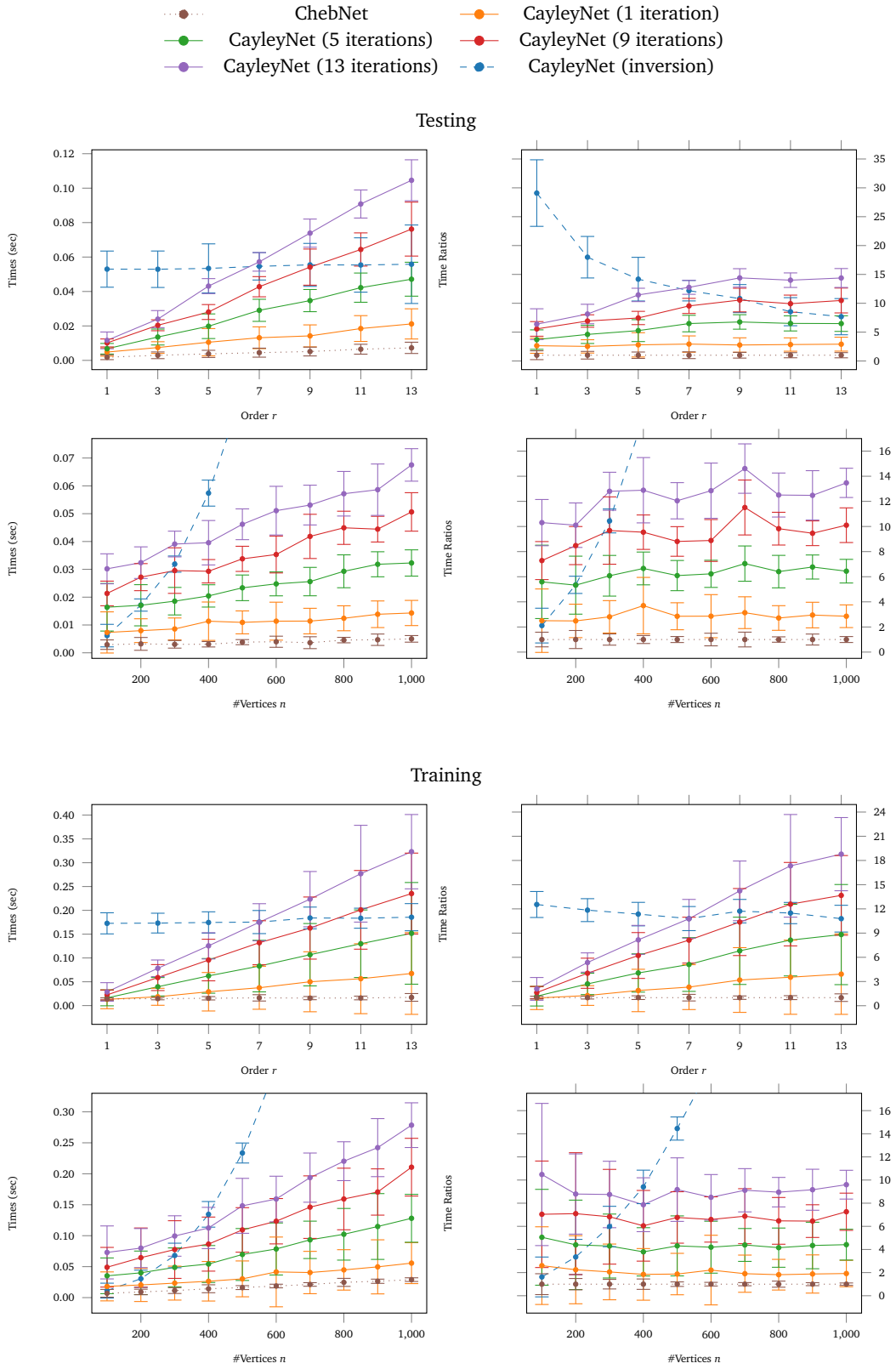


Figure 5.6. Test (above) and training (below) times with corresponding ratios (using ChebNet as reference) as function of filter order r and graph size n on our community detection dataset.

Chapter 6

Scalable Inception Graph Neural Networks

This chapter is based on "Fabrizio Frasca, Emanuele Rossi*, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks.* Graph Representation Learning and Beyond, ICML Workshop*, 2020*" *and "Federico Monti, Karl Otness, and Michael M Bronstein. Motifnet: a motif-based graph convolutional network for directed graphs. In* 2018 IEEE Data Science Workshop (DSW)*, pages 225–228. IEEE, 2018*" *(* denotes equal contribution). In the case of SIGN, the author collaborated with Emanuele Rossi and Fabrizio Frasca on the design of the architecture, and the design and review of the experiments.*

6.1 Introduction

As we have seen thus far, many of the early works in the literature designed and evaluated architectures on relatively small graphs (tens of thousands of vertices and edges). In many practical situations however, one might need to apply GCNNs on web-scale graphs such as Facebook's or Twitter/X's social networks. Since such networks can show even 10^9 nodes and 10^{11} edges, it can easily become computationally prohibitive to make predictions resorting to architectures that process the entire graph in one go. To limit the number of nodes and edges the model needs to process in a single forward / backward pass (and thus effectively apply GCNNs to such large networks), sampling and batching approaches have often been used in the literature [109, 272, 57, 277]. In [90], we took a different approach for scaling GCNNs. We showed in particular how moving the computation of neighborhood descriptors at pre-processing time, efficient GCNNs, that show the same complexity of MLPs, can be constructed while maintaining competitive performance. Such an architecture appears especially well suited for productionization in industrial settings, where heavy pre-processing steps can efficiently be realized at scale with distributed frameworks (e.g. Apache Spark, BigQuery, ...), and the model is just left with the task of handling compact representations at inference time.

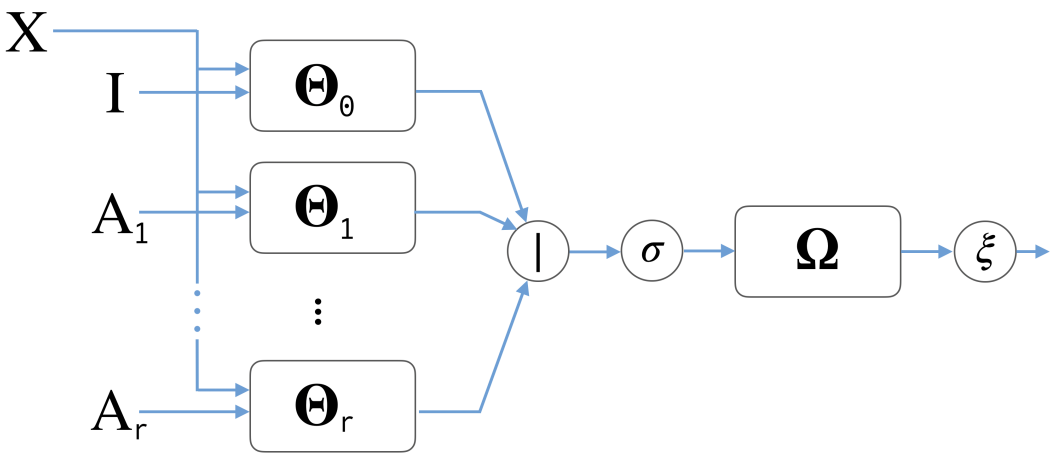


Figure 6.1. The SIGN architecture for r generic graph filtering operators. $\mathbf{\Theta}_k$ represents the k-th dense layer transforming node-wise features downstream the application of operator k, $|$ is the concatenation operation and $\mathbf{\Omega}$ refers to the dense layer used to compute final predictions.

6.2 Methodology

The key building block of our architecture is a set of linear diffusion / shift operators represented as $|\mathcal{V}| \times |\mathcal{V}|$ matrices $\mathbf{A}_1, \dots, \mathbf{A}_r$, whose application to the node-wise features can be pre-computed. For node-wise classification tasks, our architecture (that we named *Scalable Inception Graph Neural Network*, or *SIGN*, as it shows convolutional layers that resemble the Inception Module introduced in [245]) has the form:

$$\begin{aligned} \mathbf{Z} &= \sigma\left([\mathbf{X}\mathbf{\Theta}_0, \mathbf{A}_1\mathbf{X}\mathbf{\Theta}_1, \dots, \mathbf{A}_r\mathbf{X}\mathbf{\Theta}_r]\right), \\ \mathbf{Y} &= \xi\left(\mathbf{Z}\mathbf{\Omega}\right); \end{aligned} \tag{6.1}$$

where $\mathbf{\Theta}_0, \dots, \mathbf{\Theta}_r$ and $\mathbf{\Omega}$ are learnable matrices respectively of dimensions $d \times d'$ and $d'(r+1) \times c$ for c classes, and σ and ξ are non-linearities (the second one computing class probabilities, e.g. softmax or sigmoid function). Figure 6.1 provides a pictorial representation of our architecture.

As mentioned in the preamble of this chapter, in order to realize scalable filters able to process large graphs, we designed such architecture with the idea of being able to realize all the matrix products concerning graph diffusion (i.e. $\mathbf{A}_1\mathbf{X}, \dots, \mathbf{A}_r\mathbf{X}$,) at pre-processing time. In order to do so, all the diffusion operators are constrained to the first layer of the model, so that there is no dependency on the model parameters when exchanging information among neighboring nodes. This effectively reduces the computational complexity of the overall model to the one of a multi-layer perceptron (i.e. a $\mathcal{O}(L_{ff}|\mathcal{V}|d^2)$, where d is the number of features, $|\mathcal{V}|$ the number of nodes we need to process and L_{ff} the number of feed-forward layers), as the only information the model needs to process for a given node is the concatenation of the neighborhood descriptors extracted by the available shift operators. Table 6.1 provides a comparison of the complexity of SIGN with other architectures designed with scalability in mind (GraphSAGE, ClusterGCN and GraphSaint)[1].

[1] For an overview of GraphSAGE, please see Section 2.2.2. ClusterGCN [57] improves scalability of classic GCNNs by

Table 6.1. Theoretical time complexity where L_c, L_{ff} is the number of graph convolutional and feed-forward layers, r is the filter size, $|\mathcal{V}|$ the number of nodes (in training or inference), $|\mathcal{E}|$ the number of edges, and d the feature dimensionality (assumed fixed for all layers). For GraphSAGE, S is the number of neighbors sampled at each layer per node. For ClusterGCN and GraphSAINT, the cost of clustering and sampling (respectively) is ignored in the pre-processing phase as this depends on the chosen approach. Both pre-processing and forward pass complexities correspond to an entire epoch where all nodes are seen.

Method	Pre-processing	Forward Pass
GraphSAGE [109]	$\mathcal{O}(S^{L_c}\|\mathcal{V}\|)$	$\mathcal{O}(L_c S^{L_c}\|\mathcal{V}\|d^2)$
ClusterGCN [57]	$\mathcal{O}(\|\mathcal{E}\|)$	$\mathcal{O}(L_c\|\mathcal{E}\|d + L_c\|\mathcal{V}\|d^2)$
GraphSAINT [277]	$\mathcal{O}(\|\mathcal{E}\|)$	$\mathcal{O}(L_c\|\mathcal{E}\|d + L_c\|\mathcal{V}\|d^2)$
SIGN	$\mathcal{O}(r\|\mathcal{E}\|d)$	$\mathcal{O}(L_{ff}\|\mathcal{V}\|d^2)$

The different diffusion operators referenced in (6.1) can simply be powers of a given shift operator (e.g. the shifted normalized Laplacian used by ChebNet), as well as operators that diffuse in different ways information among neighboring nodes (and potentially combination of these). Generally speaking, the choice of the diffusion / shift operators to use in a given architecture to maximize performance likely depends on the task, the graph structure, and the features at our disposal. In [104], it was shown for instance how operators induced by triangles or cliques might help distinguishing edges representing weak or strong ties in social graphs, in [260] the same operator appeared to help detect the role that users play within communities in interaction networks, and in [140] diffusion operators based on personalized PageRank (PPR) or heat kernel allowed to boost performance in graphs with noisy connectivity. As a result of this, in [90] we decided to make the set of diffusion operators a hyperparameter of the model, and we determined which operators the architecture should use for each single (dataset, task) via a suitable grid search. For undirected graphs, we chose three specific types of operators for shifting information on the domain: "simple" (normalized) adjacency matrix, personalized PageRank-based adjacency matrix, and triangle-based adjacency matrices. As mentioned before, we also consider powers of each operator for accessing information available on neighbors situated multiple steps apart. We denote by *SIGN(p,s,t)*, with $r = p+s+t$, the configuration using up to p, s, and t powers of simple, PPR-based, and triangle-based adjacency matrices [186], respectively. In the case instead of directed graphs, we used the directed adjacency matrix $\mathbf{A}$, its transpose $\mathbf{A}^T$ and the adjacency matrix of the associated undirected graph $\mathbf{A}_u = \frac{1}{2}(\mathbf{A} + \mathbf{A}^T)$ for diffusion. We denote by *SIGN(p,d,f)*, with $r = p+d+f$, the configuration using up to p, d, and f powers of undirected, directed, and directed-transposed adjacency matrices, respectively.

In the following paragraphs we present a high level overview of PPR-based matrix and motif adjacency matrices to provide the reader an understanding of all the operators used in this work.

Personalized Page Rank. Personalized Page Rank (PPR) is a variation of the Page Rank algorithm [198] that allows to determine the importance of each node t with respect to a source node s. The PPR algorithm produces a probability distribution π_s that can be interpreted as

partitioning the provided domain with a clustering algorithm (e.g. Graclus [76]), and processing in a given forward pass only the nodes and edges belonging to one or more clusters. GraphSAINT processes instead samples of the domain obtained via a provided sampling mechanism, and resorts to a normalization step in order to avoid any bias in the estimated neighborhood descriptors.

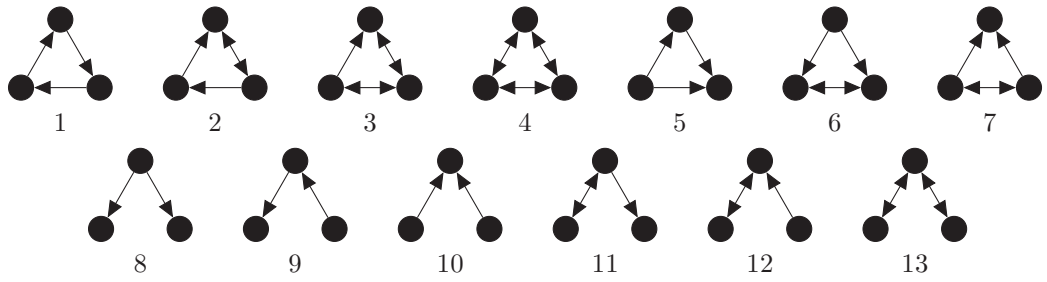


Figure 6.2. The thirteen connected 3-vertex graph motifs that can appear in directed graphs. In undirected graphs there are only two possible motifs of three nodes: a wedge (i.e. a length-two path) and a triangle (a length-three cycle).

the likelihood that a random surfer starting from s ends up on t after an infinite number of steps. The surfer at each step with probability $1-\alpha$ moves to one of the out-neighbors of the node where it currently stands, and with probability α tele-ports back to s (α is, for this reason, often referred to as *tele-port probability*). As the behavior of such surfer can be described by a Markov chain with transition probability equal to $(1-\alpha)\mathbf{A}\mathbf{D}^{-1}+\alpha\mathbf{E}_s$ (with $\mathbf{E}_s$ a matrix that is zero everywhere but in row s where it is filled with ones), $\boldsymbol{\pi}_s$ can be seen as the solution to the following linear system of equations:

$$\boldsymbol{\pi}_s = (1-\alpha)\mathbf{A}\mathbf{D}^{-1}\boldsymbol{\pi}_s + \alpha * \mathbf{e}_s. \tag{6.2}$$

Here, $\mathbf{e}_s$ is a vector of all zeros except in position s where it assumes value equal to one, and π_s is the stationary distribution of the Markov chain[2]. A PPR matrix is simply obtained stacking over multiple columns/rows the different π_s computed for all nodes in the graph.

Motif adjacency matrix. To quantify the importance of the neighbors of a target node i that are involved in triangular motifs with i, we resort to *motif adjacency matrices* following the construction proposed by Benson et al. [27]. Let $\mathcal{G}=\{\mathcal{V},\mathcal{E},\mathbf{A}\}$ be a (possibly weighted and/or directed) graph, and let $\mathcal{M}_1,\ldots,\mathcal{M}_K$ denote a collection of *graph motifs* (Figure 6.2). For each edge $(i,j)\in\mathcal{E}$ and each motif $\mathcal{M}_k$, a motif adjacency matrix $\tilde{\mathbf{A}}_k$ can be defined as $\tilde{a}_{k,ij}=u_{k,ij}a_{ij}$ for undirected graphs, and as $\tilde{a}_{k,ij}=u_{k,ij}(a_{ij}+a_{ji})$ for directed ones, where $u_{k,ij}$ denotes the number of times node i and j participate in $\mathcal{M}_k$. Each entry of $\tilde{\mathbf{A}}_k$ defines the strength of a connection between i and j, dependently on how frequently the two nodes engage in $\mathcal{M}_k$. While in principle it could be interesting to "anchor" a motif dependently on the role that i and j play in the sub-network (i.e. different motif adjacency matrices could be constructed dependently on which positions i and j assume in it), for efficiency reasons, in [90] we decided to limit ourselves to the construction proposed in [27], and we defined motif adjacency matrices only based on the number of times that two nodes appear in the given motif. For each motif adjacency matrix, a corresponding (normalized) *motif Laplacian* $\tilde{\boldsymbol{\Delta}}_k=\mathbf{I}-\tilde{\mathbf{D}}_k^{-1/2}\tilde{\mathbf{A}}_k\tilde{\mathbf{D}}_k^{-1/2}$ can be defined as well.

[2]Such distribution always exists, is unique, and can be reached through multiple projections of an initial probability distribution over matrix $(1-\alpha)\mathbf{A}\mathbf{D}^{-1}+\alpha\mathbf{E}_s$. These results directly follow from the fact that the Markov chain has only one closed communication class (i.e. only one subset of nodes that can be reached by each other but can't reach other nodes outside the subset), and such class is aperiodic (there is no state i that can be visited, starting from i, only at multiples of some $k\in\mathbb{N}^+\backslash\{1\}$) [2]. The single communication class for (6.2) is the one consisting of the strongly connected component including s [206].

While in [90] we limited ourselves to only one type of motif adjacency matrix (i.e. the one associated with triangles in undirected graphs), multiple different motif adjacency matrices could be used as part of the same model. In our own MotifNet [186], we showed for instance how anisotropic filters can be realized on graphs leveraging multi-variate polynomials obtained multiplying together different motif adjacency matrices. Such polynomials can be realized with (6.1), simply setting some of the shift operators to correspond with the aforementioned products. To limit the computational complexity of our experimental evaluation, we left an exploration of this research direction to future work.

6.3 Results

Datasets. We evaluated the proposed method on node-wise classification tasks, both in transductive and inductive settings. Inductive experiments are performed using four datasets describing undirected graphs: *Reddit* [109], *Flickr* [277], *Yelp* [277], and *PPI* [284]. Related tasks are multi-class node-wise classification (each node can be assigned only one of many possible classes) for Reddit and Flickr, and multi-label classification for Yelp and PPI (each node can be assigned zero or more labels). Transductive experiments were performed on the *ogbn-products* ($\sim$2.5 M nodes and $\sim$62 M undirected edges) and ogbn-papers100M datasets [122] ($\sim$111 M nodes and $\sim$1.6 B directed arcs). ogbn-products is an undirected network describing an Amazon product co-purchasing network [31], where the task is to predict the category of a product in a multi-class classification setup. ogbn-papers100M is a directed citation network where the task is to infer the labels (subject areas) of a smaller subset of ArXiv papers. Due to its size, ogbn-papers100M is an important test-bed for the scalability of SIGN and related methods. In addition to this, we tested the scalability of our method on *Wikipedia* links [10], a large-scale network of links between articles in the English version of Wikipedia. As no node features or labels are available for this dataset, only pre-processing, training and inference times were measured on such network (see next paragraph for more details on this). Statistics for all the datasets are reported in Table 6.2.

Experimental setting. For undirected graphs (Reddit, Flickr, Yelp, PPI, ogbn-products and Wikipedia), we tested several $SIGN(p,s,t)$ configurations. PPR-based operators are computed from a transition matrix in an approximated form (using the variation of the Andersen algorithm [22] implemented in PyTorch Geometric[3]), with a restart probability of $\alpha = 0.01$ for inductive datasets and $\alpha = 0.05$ in the transductive case. To allow for larger model capacity in the inception modules and in computing final model predictions, we replace the single-layer projections performed by $\mathbf{\Theta}_i$ and $\mathbf{\Omega}$ modules with multiple feedforward layers. Model parameters are found by minimizing the cross-entropy loss via minibatch gradient descent with the Adam optimizer. Early stopping is applied with a patience of 15. In order to limit overfitting, we apply weight decay and dropout to our model. Additionally, batch-normalization [125] was used in every layer to stabilize training and increase convergence speed. Architectural and optimization hyperparameters were estimated using Bayesian optimization with a tree Parzen estimator [30] over all inductive datasets. As for the the transductive setting, we employed standard exhaustive search on a predefined hyperparameter grid on ogbn-products. For the Wikipedia dataset, we randomly generated 100-dimensional node feature vectors and scalar targets and considered

[3] https://pytorch-geometric.readthedocs.io/en/latest/modules/utils.html#torch_geometric.utils.get_ppr

Table 6.2. Summary of (s)ingle and (m)ulti-label dataset statistics. Wikipedia is used, with random features, for timing purposes only.

Dataset	n	$\|\mathcal{E}\|$	*Avg. Deg.*	d	Classes	Train / Val / Test
ogbn-papers100M	111,059,956	1,615,685,872	30	128	172(s)	78% / 8% / 14%
Wikipedia	12,150,976	378,142,420	62	100	2(s)	100% / — / 100%
ogbn-products	2,449,029	61,859,140	51	100	47(s)	10% / 2% / 88%
Reddit	232,965	11,606,919	50	602	41(s)	66% / 10% / 24%
Yelp	716,847	6,977,410	10	300	100(m)	75% / 10% / 15%
Flickr	89,250	899,756	10	500	7(s)	50% / 25% / 25%
PPI	14,755	225,270	15	50	121(m)	66% / 12% / 22%

Table 6.3. Micro-averaged F1 score average and standard deviation over inductive datasets. For SIGN, we show the best performing configurations. The top three performance scores are highlighted as: **First**, **Second**, **Third**.

Method	Reddit	Flickr	PPI	Yelp
GCN [138]	0.933±0.000	0.492±0.003	0.515±0.006	0.378±0.001
FastGCN [53]	0.924±0.001	**0.504±0.001**	0.513±0.032	0.265±0.053
Stochastic-GCN [52]	**0.964±0.001**	0.482±0.003	**0.963±0.010**	**0.640±0.002**
AS-GCN [124]	0.958±0.001	**0.504±0.002**	0.687±0.012	—
GraphSAGE [109]	0.953±0.001	0.501±0.013	0.637±0.006	**0.634±0.006**
ClusterGCN [57]	0.954±0.001	0.481±0.005	0.875±0.004	0.609±0.005
GraphSAINT [277]	**0.966±0.001**	**0.511±0.001**	**0.981±0.004**	**0.653±0.003**
S-GCN [263]	0.949±0.000	0.502±0.001	0.892±0.015	0.358±0.006
SIGN	**0.968±0.000**	**0.514±0.001**	**0.970±0.003**	0.631±0.003
(p,s,t)	$(4,2,0)$	$(4,0,1)$	$(2,0,1)$	$(2,0,1)$

the whole network for both training and inference (powers of the normalized adjacency have been used as the only shift operator for implementing our approach).

On ogbn-papers100M (the only directed network of our analysis), due to its size, we fixed the structure of the neural network a priori and evaluated only three choices of diffusion operators. Being ogbn-papers100M a directed network, we experimented with operators built via asymmetric normalization of the original directed adjacency matrix and its transpose, as well as their powers.

Implementation. All experiments, including timings, were run on an AWS p2.8xlarge instance, with 8 NVIDIA K80 GPUs, 32 vCPUs, a processor Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz and 488GiB of RAM. SIGN is implemented using Pytorch.

Inductive. On the inductive datasets, we compare our method to *GCN* [138], *FastGCN* [53], *Stochastic-GCN* [52], *AS-GCN* [124], *GraphSAGE* [109], *S-GCN* [263], *ClusterGCN* [57], and *GraphSAINT* [277]. Table 6.3 presents the results of the various methods. In line with [277], we report the micro-averaged F1 score means and standard deviations computed over 10 runs. For each dataset we report the best performing SIGN configuration, specifying the maximum powers for each of the three employed operators. SIGN outperforms other methods on Reddit

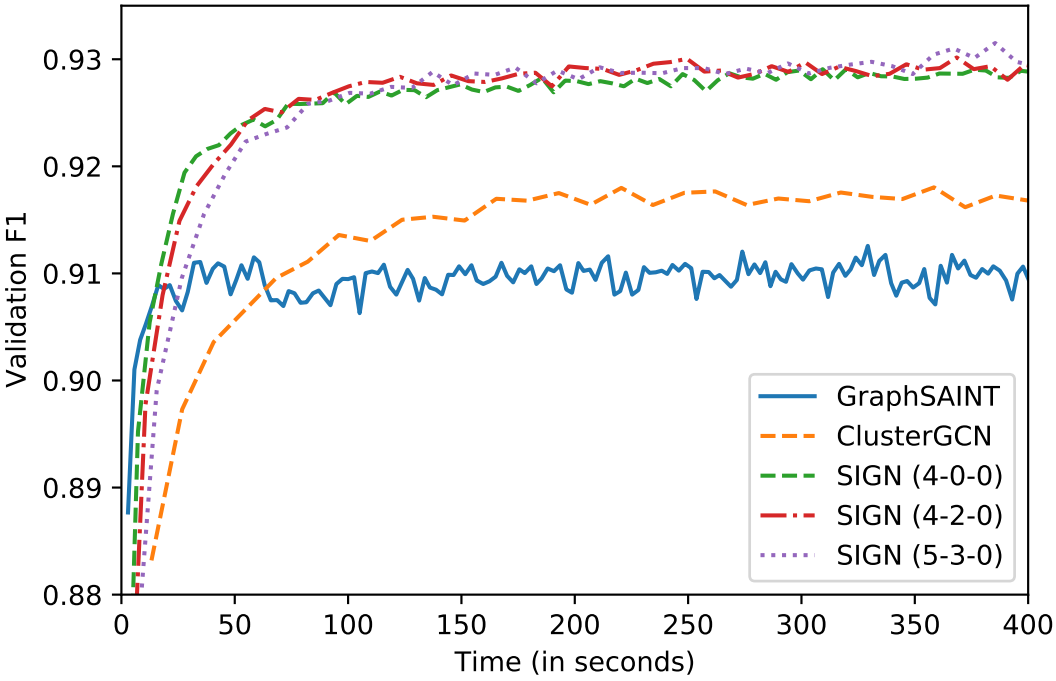


Figure 6.3. Convergence of different methods on ogbn-products.

and Flickr, and performs competitively to GraphSAINT on PPI (the best performing model in our analysis for that dataset). Our performance on Yelp is worse than in the other datasets; we hypothesize that a more tailored operators choice is required to better suit the characteristics of this dataset.

Transductive. On the massive ogbn-papers100M (Table 6.4), we report performance of a MLP applied on node features (MLP), a MLP that processes the concatenation of the raw node features and Node2Vec embeddings (Node2Vec), and S-GCN [263]. SIGN outperforms all other sampling-free competitors by at least 1.8%. On ogbn-products (Table 6.5) we additionally compare against the scalable *ClusterGCN* [57], and *GraphSAINT* [277], in line with what reported in [122]. While SIGN outperforms all other sampling-free methods (MLP, Node2Vec and S-GCN), sampling methods perform the best at test time (albeit being slower, see next paragraph) and appear to generally be more suitable on this dataset. We hypothesise that, on this particular task, sampling may implicitly act as a regularizer at training time, making these methods generalize better to the held-out test set, which is sampled from a different distribution w.r.t. training and validation nodes [122].

Run time. While performing competitively on most benchmarks w.r.t. previous methods, our architecture has the advantage of being significantly faster for large graphs. To showcase the efficiency of SIGN, we performed a timing evaluation on ogbn-products and Wikipedia datasets and report average training, inference, and pre-processing times in Table 6.7. For these experiments, we ran the implementations of ClusterGCN and GraphSAINT provided in the OGB code

Table 6.4. Results on ogbn-papers100M. SIGN(p,d,f) refers to a configuration using p, d, and f powers of simple undirected, directed and directed-transposed adjacency matrices. The top three performance scores are highlighted as: **First**, **Second**, **Third**.

Method	Training	Validation	Test
MLP	58.84±0.43	49.60±0.29	47.24±0.31
Node2Vec [105]	—	55.60±0.23	58.07±0.28
S-GCN (L=3) [263]	67.54±0.43	66.48±0.20	63.29±0.19
SIGN(3,0,0)	**70.18±0.37**	**67.57±0.14**	**64.28±0.14**
SIGN(3,1,1)	**72.24±0.32**	**67.76±0.09**	**64.39±0.18**
SIGN(3,3,3)	**73.94±0.72**	**68.60±0.04**	**65.11±0.14**

repository[4].

We used these datasets rather than ogbn-papers100M so we could compare to ClusterGCN and GraphSAINT, which, to the best of our knwoledge, are not scaled yet to ogbn-papers100M. For the sake of completeness, we report however that on ogbn-papers100M our best performing SIGN(3,3,3) model completes one evaluation pass on the validation set in 1.99 ± 0.05 seconds and on the test set in 3.34 ± 0.04 seconds (statistics are estimated over 10 runs and include the time required by device data transfers and by the computation of evalution metric).

On both ogbn-products and Wikipedia, our model is faster than ClusterGCN and of comparable speed with respect to GraphSAINT in training[5]. At inference time, SIGN is by far the fastest approach in our comparison, being always one order of magnitude faster than other methods (our largest architecture, 8 operators, requires no more than 30 seconds to perform inference on over 12M nodes). SIGN's preprocessing is slightly longer than other methods, but we notice that most of the calculations can be cast as sparse matrix multiplications and easily parallelized with frameworks for distributed computing. Finally, in order to also study the convergence behavior of our proposed model, in Figure 6.3 we plot the validation performance on ogbn-products from the start of the training as a function of run time for ClusterGCN, GraphSaint and several SIGN configurations. As it is possible to see, all methods exhibit comparable convergence speed, with SIGN being slightly faster than ClusterGCN (after 50 seconds of training, GraphSAINT appears to have converged, ClusterGCN is $\sim$ 1% away from peak performance, while SIGN is $\sim$ 0.5% from convergence).

Ablation study. To understand how different operator combinations affect the performance of SIGN, we report the results obtained for different values of p, s and t in Tables 6.5 and 6.6 for, respectively, the transductive ogbn-products and inductive datasets. We notice that best performance is obtained on each benchmark by a specific combination of operators, remarking the fact that each dataset features topological and content characteristics that can best be processed with different filters. Interestingly, we also observe while the PPR operators do not bring significant improvements in the inductive setting (being even harmful in certain cases), they are beneficial on the transductive ogbn-products. This finding is in accordance with [140], where the effectiveness of PPR diffusion operators in transductive settings has been extensively studied. Finally, we notice promising results attained in Flickr and PPI inductive settings by

[4] `https://github.com/snap-stanford/ogb/tree/master/examples/nodeproppred/products`

[5] Traning time is measured as forward-backward time to complete one epoch.

Table 6.5. Performance on ogbn-products. SIGN(p,s,t) refers to a configuration using p, s, and t powers of simple, PPR-based, and triangle-based adjacency matrices. The top three performance scores are highlighted as: **First**, **Second**, **Third**.

Method	Training	Validation	Test
MLP	84.03±0.93	75.54±0.14	61.06±0.08
Node2Vec [105]	93.39±0.10	90.32±0.06	72.49±0.10
S-GCN (L=5) [263]	92.54±0.09	91.38±0.07	74.87±0.25
ClusterGCN [57]	93.75±0.13	92.12±0.09	**78.97±0.33**
GraphSAINT [277]	92.71±0.14	91.62±0.08	**79.08±0.24**
SIGN(3,0,0)	96.21±0.31	**92.99±0.05**	76.52±0.14
SIGN(3,0,1)	**96.46±0.29**	92.93±0.04	75.73±0.20
SIGN(3,3,0)	**96.87±0.23**	**93.02±0.04**	77.13±0.10
SIGN(5,0,0)	95.99±0.69	92.98±0.18	76.83±0.39
SIGN(5,3,0)	**96.92±0.46**	**93.10±0.08**	**77.60±0.13**

Table 6.6. Impact of various operator combinations on inductive datasets. Best results are in **black**.

Method	Reddit	Flickr	PPI	Yelp
SIGN(2,0,0)	0.966±0.003	0.503±0.003	0.965±0.002	0.623±0.005
SIGN(2,0,1)	0.966±0.000	0.510±0.001	**0.970±0.003**	**0.631±0.003**
SIGN(2,2,0)	0.967±0.000	0.495±0.002	0.964±0.003	0.617±0.005
SIGN(4,0,0)	0.967±0.000	0.508±0.001	0.959±0.002	0.623±0.004
SIGN(4,0,1)	0.967±0.000	**0.514±0.001**	0.965±0.003	0.622±0.003
SIGN(4,2,0)	**0.968±0.000**	0.500±0.001	0.930±0.010	0.618±0.004
SIGN(4,2,1)	0.967±0.000	0.508±0.002	0.969±0.001	0.620±0.004

pairing standard adjacency matrices with a triangle-induced one.

6.4 Discussion

To the best of our knwoledge, SIGN was the first model that proposed combining multiple different neighborhood descriptors that were extracted at pre-processing time, in order to realize scalable yet effective GCNNs[6]. Following up on our work, other better performing approaches appeared however in the literature implementing similar architectural choices[7].

The closest work to ours is probably *Feature Selection Graph Neural Network* (FSGNN) of Maurya et al. [176]. As we have seen in Section 6.2, SIGN utilizes a variety of diffusion operators in order to extract a rich set of descriptors depicting the behavior of the neighborhood of each target node. As the set of diffusion operators $\{\mathbf{A}_1,\ldots,\mathbf{A}_r\}$ is fixed a priori, this gen-

[6]Please note, S-GCN [263] can be seen as a simplified version of our work where only one diffusion operator is used, and the MLP defined by matrices $\mathbf{\Theta}_1,\ldots,\mathbf{\Theta}_r$ and $\mathbf{\Omega}$ is replaced with a simple linear layer.

[7]The interested reader is invited to refer to `https://ogb.stanford.edu/docs/leader_nodeprop/` for the live leaderboards of ogbn-product and ogbn-papers100M, depicting the performance of the methods discussed in this section and others.

Table 6.7. Mean and standard deviation of preprocessing, training (one epoch) and inference times, in seconds, on OGBN-Product and Wikipedia datasets, computed over 10 runs. SIGN-r denotes architecture with r precomputed operators. Preprocessing and training times for ClusterGCN on Wikipedia are not reported due to the clustering algorithm failing to complete.

Method	ogbn-products			Wikipedia		
	Preprocessing	Training	Inference	Preprocessing	Training	Inference
ClusterGCN [57]	36.93 ± 0.52	13.34 ± 0.16	93.00 ± 0.68	—	—	183.76 ± 3.01
GraphSAINT [277]	52.06 ± 0.54	2.89 ± 0.05	94.76 ± 0.81	123.60 ± 1.60	135.73 ± 0.06	209.86 ± 4.73
SIGN-2	88.21 ± 1.33	1.04 ± 0.10	2.86 ± 0.10	192.88 ± 0.12	62.37 ± 0.17	13.40 ± 0.15
SIGN-4	160.16 ± 1.20	1.54 ± 0.04	3.79 ± 0.08	326.21 ± 1.14	93.84 ± 0.08	18.15 ± 0.05
SIGN-6	226.48 ± 1.43	2.05 ± 0.00	4.84 ± 0.08	459.24 ± 0.14	125.24 ± 0.03	22.94 ± 0.02
SIGN-8	297.92 ± 2.92	2.53 ± 0.04	5.88 ± 0.09	598.67 ± 0.82	154.73 ± 0.12	27.69 ± 0.11

erally requires some level of exploration, in the construction of the architecture, to identify a good configuration. To alleviate this procedure, similarly to what we previously presented in MotifNet [186], Maurya et al. proposed to introduce a soft feature selection step in the implementation of SIGN. Each neighborhood descriptor $\mathbf{A}_k\mathbf{X}\mathbf{\Theta}_k$ gets multiplied in FSGNN by a scalar value α_k, which can be interpreted as a particular form of attention. Each α_k is constrained by a softmax layer to be in $[0,1]$ and $\sum_{i=1}^{r}\alpha_i = 1$. Thanks to the addition of this "architectural attention layer" (and an additional $\mathbb{L}_2$ normalization layer applied on top of each $\mathbf{A}_k\mathbf{X}\mathbf{\Theta}_k$), FSGNN managed to achieve with the same set of neighborhood descriptors (obtained with powers of the symmetric normalized adjacency matrix and its version with added self-loops) consistently good performance across both homophilic and heterophilic datasets. FSGNN additionally outperformed SIGN on the large ogbn-papers100M dataset. While no ablation study is provided in the paper to explain whether this is due to extra normalization step, the architectural attention layer, or simply better hyperparameters, we conjecture that the vectors of weights $\boldsymbol{\alpha}$ could have played a role in the performance improvement, as it introduces a form of regularization provided that $\mathbf{\Omega}$ doesn't have scores (of potentially different magnitudes) that nullify the effect of the attention mechanism.

Moving now in a slightly different direction, another work strongly related to SIGN is *Scalable and Adaptive Graph Neural Network* (SAGN) of Sun et al [238]. Drawing inspiration from the Jumping Knowledge Networks we discussed in Section 3.3, SAGN introduces an attention mechanism between the first and the second layer of SIGN to reduce the information collected from multiple different hops to a single d-dimensional vector of features (powers of a unique diffusion operator $\bar{\mathbf{A}}$ are used in [238] to compute neighborhood descriptors, and the features of the target node are used as key for computing the attention scores). Thanks to the introduction of this attention layer, the number of parameters in the final classifier reduces from $\mathcal{O}(d'(r+1)\times c)$ to $\mathcal{O}(d'\times c)$, thus hopefully limiting the chance of overfitting (especially when many powers of $\bar{\mathbf{A}}$ are used in the pre-processing phase and only a small labelled set is available[8]). In addition to this, a label propagation [228] and a self-training [156] strategy is introduced in [238] to boost performance. In experimental evaluation, SAGN (especially when combined with label propagation and self-training) outperformed SIGN on Reddit, Flickr, PPI, Yelp and ogbn-product, and achieved slightly better performance than FSGNN on ogbn-papers100M.

The main intuition at the core of SIGN was to remove any dependency between model pa-

[8] A variation of SAGN that moves the attention layer at the beginning of the model, and thus admits a configuration with a number of parameters that is completely independent on r, was additionally presented in [280].

rameters and diffusion steps, in order to realize a GCNN with the same computational complexity of a MLP. A more extreme version of our architecture can be achieved completely removing any aggregated descriptor from the input to the model, and letting a MLP applied on the target node features to do all the heavy lifting (thus completely removing the need of any diffusion step on the input graph). While this might not always be feasible to achieve good performance (think for instance to a heterophilic graph where the labels to predict depend only on the behavior of the neighbors, and there is no dependency across nodes), Zhang et al. showed that performance close to the one of a teacher GCNN can be achieved in practice with a suitable student MLP (which is applied only on a given node features) that is trained with *Knowledge Distillation* (KD) [279]. In this setting, the teacher GCNN is trained first on the possibly small labeled set, then predictions $\mathbf{Y}^{(U)}_{obs}$ are computed with the trained teacher for each observed but unlabeled node in the graph, and finally $\mathbf{Y}^{(U)}_{obs}$ is used as soft labels for instructing the MLP in addition to the provided hard training labels. $\mathbf{Y}^{(U)}_{obs}$ effectively operates here as a regularizer, which aims at transferring the inductive bias of the teacher GCNN to the student MLP by training this on a much larger (and varied) labeled set, and hopefully limiting overfitting as a result. In experimental evaluation, Zhang et al. observed how using $\mathbf{Y}^{(U)}_{obs}$ as part of the training set consistently pushes the results of the MLP to match the ones of the teacher GCNN, while only requiring a fraction of the computational complexity. On ogbn-products, the approach proposed in [279] (named *Graph-Less Neural Networks*, GLNNs) achieved comparable performance to the one of SIGN that we reported in our original paper[9].

[9] At the time of writing, an implementation of SIGN outperforming the results showed by GLNNs in [279] is available in the leaderboard of ogbn-products. It should not be excluded, however, that GLNNs could be able to achieve similar (or better) results if trained using a more performing teacher model or a more suitable student architecture.

Part II

Applications of GCNNs

Chapter 7

Neutrino detection via IceCube Signal Classification

This chapter is based on "Nicholas Choma, Federico Monti, Lisa Gerhardt, Tomasz Palczewski, Zahra Ronaghi, Prabhat Prabhat, Wahid Bhimji, Michael M Bronstein, Spencer R Klein, and Joan Bruna. Graph neural networks for icecube signal classification. In 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 386–391. IEEE, 2018". Here, the author collaborated with Nicholas Choma on the design and implementation of the method, and the design and execution of the experiments.*

7.1 Introduction

In [60] we studied the application of GCNNs to the challenging problem of neutrino detection in the IceCube observatory. IceCube is a 1 km^3 neutrino observatory located at the South Pole [14]. Its primary purpose is to look for high-energy neutrinos (above 100 gigaelectronvolts (GeV)) that are produced by the same cosmic particle accelerators (e.g. supernovas) that produce ultra-high energy cosmic-rays [108]. Its 5,160 sensors (digital optical modules, or DOMs) detect the Cherenkov light that is produced by the relativistic charged particles (i.e. muons) resulting from high-energy neutrinos interacting in the Antarctic ice[1]. The Cherenkov light is emitted at a fixed angle [19] and may scatter before being observed by sensors that are typically 10 to 60 meters away from the track. Sixty sensors are deployed on each of 86 strings placed in holes drilled in the ice. Most of the strings are on a 125 m triangular grid, but 8 strings, forming the 'Deep Core' infill array, have much tighter spacing. On most strings, the DOMs are deployed every 17 m, from 1450 m to 2450 m below the surface; in Deep Core, most of the DOMs are deployed with a 7 m spacing between 2100 and 2450 m. IceCube includes an array of 81 surface stations called IceTop, designed to study cosmic ray interactions in the atmosphere. The schematic view of the IceCube detector is shown in Figure 7.1. The sensors record the photon arrival times

[1]When a neutrino interacts with a molecule of ice, it creates a secondary charged particle named *muon*, which travels nearly collinearly with the neutrino [19]. Electrically charged particles when they travel through a mean with a phase velocity faster than the one of light produce a radiation named *Cherenkov light*. The goal of the detector is to capture the Cherenkov light produced by the neutrino generated muons, and use this to infer the travelling direction, arrival time, and energy of astrophysical neutrinos [19, 18].

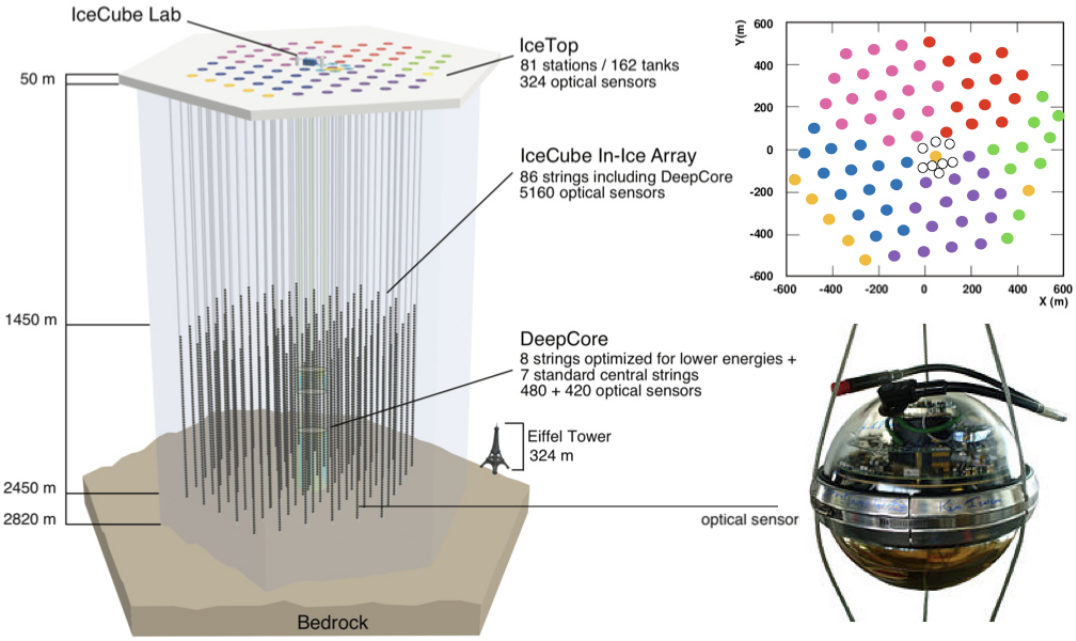


Figure 7.1. The IceCube Neutrino Observatory with the in-ice array, its sub-array DeepCore, and the cosmic-ray air shower array IceTop. The string color scheme represents different deployment seasons. The top-right insert presents the top view of the IceCube detector. The DeepCore sub-array is represented by open circles.

using waveform digitizers. Across the array, the relative arrival times are known to better than 3 ns [14].

IceCube observes two classes of events. *Contained events* occur when neutrinos interact with the ice of the detector. *Through-going events* are instead long-lived muons which are produced outside the detector and can travel many kilometers in the ice. They can be produced in neutrino interactions, or in cosmic-ray air showers that occur when high-energy cosmic-rays interact with molecules available in the upper atmosphere (producing down-going *atmospheric* muons).

In this work, we discuss methods to separate the signal (muons from neutrinos) from the background (muons from cosmic-ray showers). For the present purposes, the main difference between the signal and the background is the stochasticity of the energy deposition, which translates into clumpiness in the light emission from the track. Muons from neutrinos are single high-energy muons, which lose energy mainly through stochastic (random) processes [63, 15], leading to a very uneven light emission[2]. In contrast, muons from cosmic-ray interactions come in bundles containing from one (rarely) to hundreds of muons, and typically have relative low energy. As low energy muons lose energy smoothly [12, 63] (the stochasticity of energy losses for a given muon increases with its energy [63]), there is negligible fluctuation in their energy deposition, which leads to smoother patterns on the sensor network (Figure 7.2).

We generated two Monte Carlo datasets for our experimentation, one for signal and one for background, and used them for all three methods discussed in this section. In our signal

[2]The amount of energy lost can generally be considered proportional to the amount of emitted light [63].

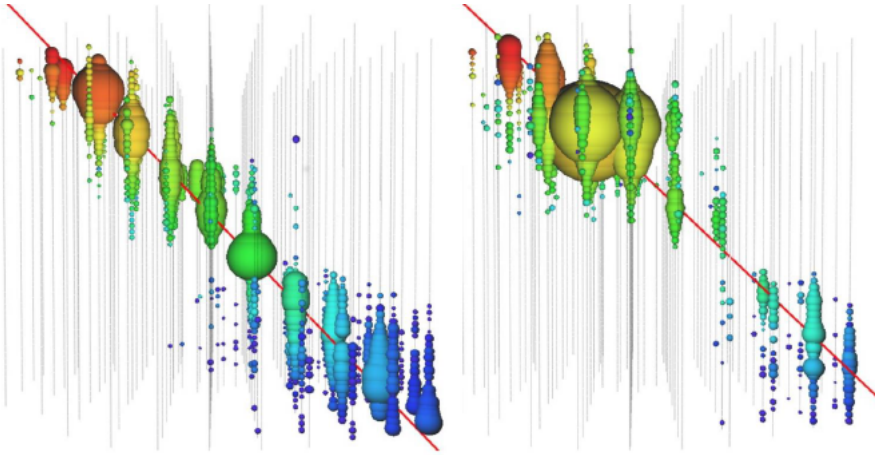


Figure 7.2. The characteristic pattern of light deposition for muon bundles (left) and a high-energy single muon with visible stochastic light emission along the track (right). The red line shows the muons track, while each colored bubble represents a DOM that saw light in the event. The colors indicate the relative light arrival time, from red (earliest) to blue (latest), while the size of the bubbles indicates the number of observed photons.

simulation set, neutrino energies range from 100 GeV to 10^8 GeV. The astrophysical neutrino data set was assumed to follow a power law ($\partial N_\nu/\partial E_\nu \propto E_\nu^{-2}$, with $\partial N_\nu(E_\nu)$ the number of neutrinos ν between E_ν and $E_\nu + \partial E$), while the cosmic-ray air shower background was generated with the CORSIKA simulation package [116]. The energy range of cosmic ray primary particles entering the atmosphere in our background simulation set is 600 GeV to 10^5 GeV. No other backgrounds have been taken into consideration in our experiments.

For both the signal and background classes, more energetic events are more likely to pass the event selection process [100]. To efficiently generate our samples, we sampled the signal and background events following a distribution that is more biased towards energetic events with respect to their real one (this allowed a more efficient use of computational resources as less samples were discarded by the filtering scheme). To maintain the real distribution of background and signal events, each event received a weight at generation time, so that the dataset's weighted histogram reproduced the (unweighted) histogram we would have had if the dataset was sampled from the original spectrum.

The particles produced by these simulations were then run through an IceCube-specific detector simulation, which included the generation of Cherenkov light, light propagation through the ice, and the detector response. The DeepCore array and IceTop stations were excluded during the pre-selection process. The resulting simulated data were run through the standard IceCube calibration and reconstruction packages.

To be usable by IceCube, the predictions of any detection model must have a reasonably high signal-to-noise ratio (SNR) in a given year. To evaluate the performance of different methods, we decided a priori to evaluate the methods on the basis of how many events they could find, subject to maintaining a 1:1 SNR (i.e. the best model is the one that retrieves the largest amount of positive events in a given year, while not retrieving more false positives than true positives). Since the background is many orders of magnitude larger than the signal [18], this requires a very high level of rejection.

7.2 Methodology

To classify the simulated signals dependently on the event they describe, we evaluated three different architectures: a physics baseline implemented by the IceCube collaboration, a classic 3D Convolutional Neural Network, and a GCNN implemented on the irregular grid of the detector using the MoNet patch operator [184].

7.2.1 Physics Baseline

IceCube has developed two conventional (i. e. not machine learning based) criteria to measure stochasticity and reject muon bundles. The first divides the muon track into 120 meter long segments, and determines the light output from the track in each segment [15]. This light output is then fit to a line, and a pseudo-χ^2 as the sum of squared residuals is calculated (residuals are defined in this case as the difference between the measured energy loss in a given segment and the expected one). Events with large pseudo-χ^2 are very stochastic (as they show strong variations w.r.t. the mean energy loss) and thus likely represent single muons from neutrinos. The second approach also reconstructs the light deposition along the track, by examining the light output in 50 meter long segments, apportioning light from each DOM to the nearest segment. Then, the largest muon's energy loss across all segments is divided by the median energy loss, giving a peak/median ratio [12]. The results from these two methods are correlated, but the correlation is low enough that there is value in using both. IceCube uses a combination of hand-tuned selections on these two variables to select single muon events [199].

7.2.2 3D Convolution Neural Networks

Since the IceCube detector has an irregular hexagonal shape, classic CNNs cannot be directly applied to signals defined over the sensor network. To evaluate the performance of this specific class of models, we thus mapped the DOMs of the detectors to voxels in a discrete $10 \times 20 \times 60$ 3D grids (with the last dimension matching the number of DOMs in each string of sensors). Deepcore strings were mapped on the 3D grid with strings {86, 81, 82} on one row and strings {85, 79, 84, 83, 80} on a different one. Zero padding was used to fill the input signals for voxels that were not associated with any DOM. To classify a provided input, we trained a ResNet [110] with 18 residual convolutional layers, which was identified with a grid search as a good architecture for classifying the input signals.

7.2.3 Graph Convolutional Neural Networks

Due to the irregular structure of the detector's sensor network (which presents different densities of sensors in different regions of space), mapping the sensors to a 3D grid doesn't necessarily well describe the sensors position unless very fine grained (and thus large) grids are used to describe the space. This, however, could require very large filters (or many convolutional layers) to learn meaningful local patterns with classic CNNs (due to the potentially large distance that pairs of sensors could show in such 3D grids), which could easily lead to overfitting. On top of this, as both background and target events are typically localized in the sensor network (Figure 7.2), vast regions of the grid typically do not carry meaningful information about the input signal. Resorting to classic CNNs to process the mapped information can thus lead to

a waste of computational resources, as all areas of the grid are processed by the architecture independently on whether they carry meaningful features or not.

Graph construction. An alternative approach to process the information retrieved by the DOMs is to describe the sensor network as an attributed directed graph. Each DOM corresponds in the graph with a node, and two nodes are connected by an arc if they fall within a given distance. The relative position of two sensors (defined by the tuple $(\Delta x, \Delta y, \Delta z)$) is described via attributes defined over the arcs of the graph itself. As no discretization takes place to describe the position of different sensors in this scenario, the size of the graph equals the number of DOMs in the detector, and the exact relative position of two neighboring sensors can be used to define accurate filters in space.

Graph convolutional layers. In our experiments, we constructed a GCNN resorting to the MoNet patch operator defined in Chapter 3 for learning meaningful patterns on the provided graph. To contain the computational complexity of the model, instead of processing for each event the whole sensor network, only the DOMs that were activated in the given event were considered for constructing the associated graph. Different events thus show graphs of different sizes in input to the model, and no zero-padding is required to fill in the gaps as it was the case for classic CNNs. Due to the small number of positive samples available for training, resorting to a single Gaussian kernel with zero mean (and thus realizing isotropic filters) turned out to be the architecture able to achieve best performance in our experiments. In our model, the relevance a_{ij} of a given neighbor j of a target node i was computed as:

$$a_{ij} = \frac{e^{d_{ij}}}{\sum_k e^{d_{ik}}}; \tag{7.1}$$

$$d_{ij} = -\frac{1}{2}\|\mathbf{x}_i - \mathbf{x}_j\|^2/\sigma^2; \tag{7.2}$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ denotes the spatial coordinates of the ith and jth DOM respectively, and σ is a learnable scalar parameter controlling the locality of the kernel and how fast information is allowed to spread across distant nodes.

Differently from equation (3.4), since we just use one kernel for aggregating information on the neighborhood of a target node i and no self-loops are used in the definition of our graph, convolutional filters are implemented in our architecture via a linear combination of the neighborhood features and the ones of i, to ensure that i's features are processed by the filter:

$$\mathrm{GConv}^{(l)}(\mathbf{X}^{(l)}) = (\mathbf{A}\mathbf{X}^{(l)} || \mathbf{X}^{(l)})\mathbf{w}^{(l)} + b^{(l)}. \tag{7.3}$$

Here $\mathbf{w}^{(l)}$ is a $2d^{(l)}$-dimensional vector of learnable weights, $b^{(l)}$ is a bias, and σ is the scaling parameter that is used in the definition of attention matrix $\mathbf{A}$ (which is shared across layers). Please note that matrix $\mathbf{A}$ can (and most likely is) different for different events, in virtue of the node sampling defined at the beginning of this paragraph. Each graph convolutional layer is obtained concatenating the output of two different set of filters, which use two different activation functions (ReLU and the identity function respectively):

$$\mathbf{X}^{(l+1)} = \mathrm{ReLU}\left(\mathrm{GConv}^{(l,1)}(\mathbf{X}^{(l)})\right) || \, \mathrm{GConv}^{(l,2)}(\mathbf{X}^{(l)}). \tag{7.4}$$

This was experimentally found beneficial to achieve better performance.

Table 7.1. Unweighted and weighted number of signal and background events within each dataset

	# Unweighted		# Weighted	
Dataset	Signal	Background	Signal	Background
Training	12624	54745	7.8	37275
Validation	6313	27373	3.9	18648
Test	6313	27373	3.9	18632
Final Evaluation	8487	366433	5.2	250982

Table 7.2. Performance of several methods in terms of expected number of signal and background events returned in a year. Our GCNN outperforms both the 3D CNN and the physics baseline both in terms of SNR and overall number of retrieved positive events.

	# events per year		
Method	Signal	Background	Signal:Noise
Physics Baseline	0.922	0.934	0.987
3D CNN	1.815	1.937	0.937
GCNN	**5.772**	1.937	**2.980**

Graph pooling. To predict the class of a given event, the output features of the last graph convolutional layer L are pooled together by summing over the vertices of the associated graph:

$$x_k^{(pool)} = \sum_{i=1}^{n} x_{ik}^{(L)}, \quad k = 1, \ldots, d^{(T)}; \tag{7.5}$$

here n is the number of vertices in the graph of the given event. This produces a $d^{(L)}$-dimensional vector $\mathbf{x}^{(pool)}$, which is then fed in input to a logistic regression classifier to predict the class of the given event:

$$\hat{y} = \text{sigmoid}((\mathbf{x}^{(pool)})^\top \mathbf{w}^{(pool)} + b^{(pool)}); \tag{7.6}$$

$\mathbf{w}^{(pool)}$ is again a $d^{(L)}$-dimensional vector of weights and $b^{(pool)}$ is the scalar bias parameter of the output layer.

7.3 Results

Model training, validation, and testing were completed using a signal and a background dataset containing 25,250 and 109,491 events, respectively. Each dataset was subdivided into 50% training, 25% validation, and 25% test sets. We trained each model up to 100 epochs, performing early stopping once performance was maximized on the validation set. For each considered architecture, the configuration that performed the best on the test set was selected for final evaluation on an additional evaluation dataset. Final model evaluation was performed using 8,487 and 366,433 signal and background events respectively.

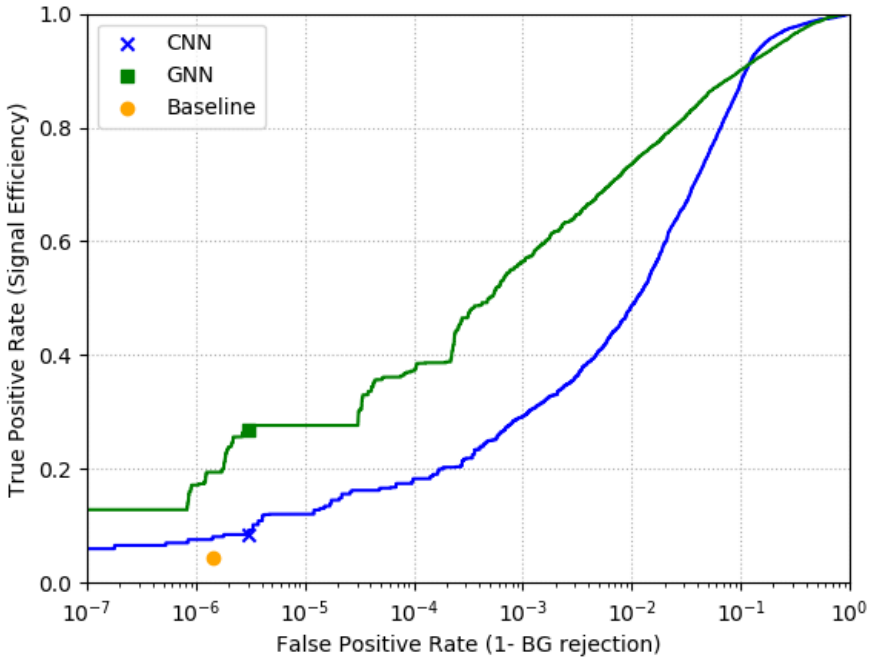


Figure 7.3. Receiver operating characteristic curve for various methods considered. The green square and blue X indicate the evaluation point for the GCNN and CNN, respectively.

As mentioned at the beginning of this work, each event is assigned a weight at data generation time that allows to reproduce the real distribution of both the positive and negative class. Training and evaluation was performed on weighted samples, in order to take into account the discrepancy in size between the two classes. Table 7.1 displays the number of events for both the signal (SG) and background (BG) classes within each dataset, with and without event weighting. The number of weighted events for a given class within a given dataset is the sum of all weights of events in the dataset which belong to that class. As it emerges from the table, while the class imbalance is large for the unweighted samples, it is even larger once event weighting is applied.

For each event, a 6-dimensional feature vector is associated to each DOM. Such vector contains the x, y, z position of its corresponding DOM, the sum of charge in the first pulse within the sensor, the sum of charge in all pulses within the sensor, and the time at which the first pulse crosses the DOM activation threshold[3]. 6 graph convolutional layers were used to implement our GCNN, while 18 layers were used in the implementation of ResNet as mentioned in Section 7.2.

Figure 7.3 displays receiver operating characteristic (ROC) curves for both the GCNN and

[3]Cherenkov photons are converted to electrons in DOMs through photomultiplier tubes (PMTs). When photons hit the photocathode of a PMT they release electrons that are amplified in the DOM, resulting in a measurable current. The current generated in the DOMs is then discretized in a series of pulses describing the time and amount of charge moved by the photons, to allow processing of the retrieved information with digital systems. A DOM records and discretizes the generated current only when this passes an activation threshold of 0.25 photoelectrons [100].

CNN models as evaluated on the final evaluation set. As it is possible to see, the GCNN outperforms both the physics baseline and the 3D CNN in almost all conditions (the 3D CNN achieves better performance for FPRs > 0.1, however such operative conditions are not interesting due to the large amount of FPs the models would retrieve in practice).

Table 7.2 displays the number of weighted signal and background events each model is expected to select per year. To determine the values displayed in the table we used the TPR and FPR as assessed on the final evaluation dataset, and we estimated the number of TPs and FPs the model is expected to return assuming that performance remain the same over a year (an estimate of ~ 21 positive events and $\sim 645,683$ background events per year have been used for calculation here). For our best model (i.e. the GCNN), we selected as evaluation point the one that maximizes the number of retrieved events while maintaining a SNR > 1.0. For the 3D CNN, we used as evaluation point the one showing the same FPR of the GCNN, in order to provide a meaningful comparison on the expected number of retrieved signals the two models allow to achieve for comparable number of FPs. Our GCNN outperforms both the physics baseline and the 3D CNN by identifying 6.3x and 3.2x more signal events respectively. Signal-to-noise ratio (SNR) is additionally 3x better for the GCNN with respect to both models at the selected evaluation points. As reference, if we were to select as evaluation point the one where the 3D CNN returns the same number of TPs as the GCNN, roughly 250x more FPs would be retrieved by the model.

7.4 Discussion

Future research. Despite having small sample sizes (especially for the positive class), our results shows the possibility of a marked improvement from application of GCNNs over the physics baseline and demonstrates a huge potential of this approach for signal classification within the IceCube detector. In future works, it would be interesting to analyze the performance that different graph convolutional layers (or layers able to deal with sets, e.g. [276]) could allow to achieve, as no other implementation besides the MoNet architecture was evaluated in this work. Investigating richer pooling mechanisms (e.g. DiffPool [273]) could also allow to boost performance further.

Related works. To the best of our knowledge, the work discussed in this chapter was the first application of GCNNs to neutrino detection. In follow up works, GCNNs have additionally been used for the classification of KM3NeT signals[4] [214], as well as for the reconstruction of events in IceCube[5] [182, 13, 241] and TRIDENT[6] [183]. It also should be noted that, as particle detectors often retrieve sparse information that naturally does not fit over grids or sequences (but rather appears as unordered sets of data possibly showing interesting pairwise relations) [229], several other works have been proposed in the last years that more broadly apply GCNNs in the realm of High Energy Physics. In this direction, GCNNs have been applied for jet tagging [118, 209, 189, 190] (i.e. identifying the type of particle that originated a jet[7]), pileup mitigation [172, 179] (i.e. identifying which particles belong to uninteresting

[4] KM3NeT is an alternative neutrino detector that is under construction at the bottom of the Mediterranean sea.

[5] The reconstruction of a neutrino event involves inferring the type of neutrino that caused the event as well as its direction and energy [4].

[6] TRIDENT is a new neutrino observatory that is being built in the South China Sea.

[7] A jet is a cone-shaped spray of particles that derives from the hadronization of isolated quarks or gluons that are created in high-energy proton-proton collisions.

interactions deriving from the collision of high density beams), particle tracking [86, 130] (i.e. identifying which hits in a detector correspond to which particles), and more. As a review of the aforementioned methods is beyond the scope of this work, we refer the interested reader to [229] for a more extensive revision of the literature, and to [75] for specific application of GCNNs at the Large Hadron Collider.

Chapter 8

Fake News Detection on Social Media

This chapter is based on "Federico Monti, Fabrizio Frasca, Davide Eynard, Damon Mannion, and Michael M Bronstein. Fake news detection on social media using geometric deep learning. Representation Learning on Graphs and Manifolds workshop*, 2019*".

8.1 Introduction

In the past decade, social media have become one of the main sources of information for people around the world. Yet, using social media for news consumption is a double-edged sword. On the one hand, it offers low cost, easy access, and rapid dissemination. On the other hand, it comes with the danger of exposure to 'fake news' containing poorly checked or even intentionally false information aimed at misleading and manipulating the readers to pursue certain political or economic agendas.

The extensive spread of fake news has recently become a global problem and threat to modern democracies. The diffusion of fake news before the United States 2016 presidential elections [40] and the Brexit vote in United Kingdom has become for instance the centerpiece of the controversy surrounding these political events and allegations of public opinion manipulation. Due the very high societal and economic cost of the phenomenon [121], fake news detection in social media has attracted enormous attention in the academic and industrial realms in recent times [150].

Prior works. Broadly speaking, existing approaches for fake news detection can be categorized into two main categories: *content-based* and *context-based* solutions[1].

Content-based approaches can be classified in two main sub-categories: *style-based* and *knowledge-based*. Style-based approaches rely on linguistic (lexical and syntactical) features that can capture deceptive cues or writing styles in the content of an article [17, 221, 213, 204, 201]. Knowledge-based approaches make predictions by comparing the facts contained in a given piece of news against a knowledge base / graph [85, 247, 227]. The main drawback of style-based approaches is that they can be defied by sufficiently sophisticated fake news that does not immediately appear as fake. Knowledge-based approaches might fail instead

[1] Please note, the two classes of approaches outlined in this section are not mutually exclusive, and hybrid solutions that use features from both classes can be found in the literature [231].

at processing news that contain new information that cannot be verified with the potentially outdated ground truth. Additionally, linguistic features are generally language-dependent in both cases, which limits the applicability of these approaches.

Context-based solutions, on the other hand, process the interactions that users might have with pieces of news, together with information about the news authors / publishers, to infer the veracity of a given article [222, 231]. An interesting class of context-based solutions is represented in particular by *propagation-based approaches* [50, 147, 167]. As it was shown by Vosoughi et al. in [255], false and true information appear to spread in different ways on social media. In order to predict the veracity of a given piece of news, propagation-based approaches thus leverage the patterns that form when the target news spread over time on a given social network. As propagation-based features can be content-agnostic (e.g. statistics describing the structure of trees of posts that respond to each other [255], or features capturing properties of the induced social network of users that share a given article [282]), they have the potential to generalize across different languages, and geographies, as opposed to content-based features that must be developed separately for each language. Furthermore, controlling the news spread patterns in a social network is generally beyond the capability of individual users. Propagation-based models could thus be hard to tamper with by adversarial attacks.

Main contribution. In [187], we proposed learning fake news specific propagation patterns exploiting GCNNs. Our model is trained in a supervised manner on a large set of annotated fake and true stories spread on Twitter in the period 2013-2018. We performed extensive testing of our model in different challenging settings, showing that it achieves significantly high accuracy (nearly 93% ROC AUC), it requires very short news spread times (just a few hours of propagation), and performs well when the model is trained on data distant in time from the testing data. The architecture, insights and experiments presented in this chapter are all the results of the work we did at Fabula AI (our own startup that was acquired by Twitter in 2019). The approach discussed herein represents therefore a possible application of GCNNs in an industrial setting.

8.2 Dataset

One of the key challenges in machine-learning based approaches is collecting a sufficiently large, rich, and reliably labelled dataset on which the algorithms can be trained and tested. As the notion of 'fake news' itself is rather vague and nuanced, an intuitive approach to gather a large dataset would be to exploit the notion of reliable or unreliable *sources* as a proxy for true or false *stories*. However, what constitute a "reliable source" may change dependently on the views of people, and in general not all articles from a trusted / untrusted source are necessarily true / fake news [255]. In our study, we opted for a data collection process in which each 'story' has an underlying article published on the web, and each such story is verified *individually*. In our classification of true or false statements we rely on professional non-profit fact-checking organizations such as Snopes,[2] PolitiFact,[3] and Buzzfeed.[4] We note that our use of the term *fake news*, though disliked in the social science research community for its abuse in the political discourse,

[2] https://www.snopes.com/
[3] https://www.politifact.com/
[4] https://www.buzzfeed.com/

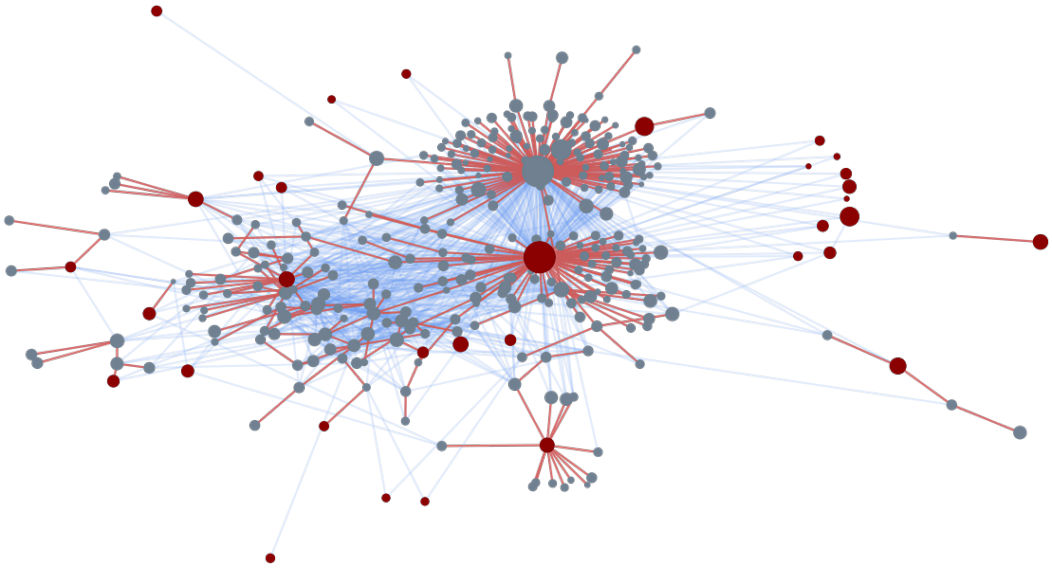


Figure 8.1. Example of a single news story spreading on a subset of the Twitter social network. Social connections between users are visualized as light-blue edges. A news URL is tweeted by multiple users (cascade roots denotes in red), each producing a cascade propagating over a subset of the social graph (red edges). Circle size represents the number of followers. Note that some cascades are small, containing only the root (the tweeting user) or just a few retweets.

refers to both misinformation and disinformation, i.e. unintentional (misinformation) as well as deliberate spread of misleading or wrong narrative or facts (disinformation).

Data collection protocol. Our data collection process was inspired by and largely followed the pioneering work of Vosoughi et al. [255]. We used a collection of news verified by fact-checking organizations with established reputation in debunking rumors; each source fact-checking organization provides an archive of news with an associated short *claim* (e.g. 'Actress Allison Mack confessed that she sold children to the Rothschilds and Clintons') and a *label* determining its veracity ('false' in the above example). First, we gathered the overall list of fact-checking articles from such archives and, for simplicity, discarded claims with ambiguous labels, such as 'mixed' or 'partially true/false'. Second, for each of the filtered articles we identified potentially related *URLs* referenced by the fact-checkers, filtering out all those not mentioned at least once on Twitter. Third, trained human annotators were employed to ascertain whether the web pages associated with the collected URLs were *matching* or *denying* the claim, or were simply unrelated to that claim. This provided a simple method to propagate truth-labels from fact-checking verdicts to URLs: if a URL matches a claim, then it directly inherits the verdict; if it denies a claim, it inherits the opposite of the verdict (e.g. URLs matching a true claim are labeled as true, URLs denying a true claim are labeled as false). URLs gathered from different sources, with same veracity and date of first-appearance on Twitter, were additionally inspected to ensure they referred to different articles.

The last part of the data collection process consisted of the retrieval of Twitter data related

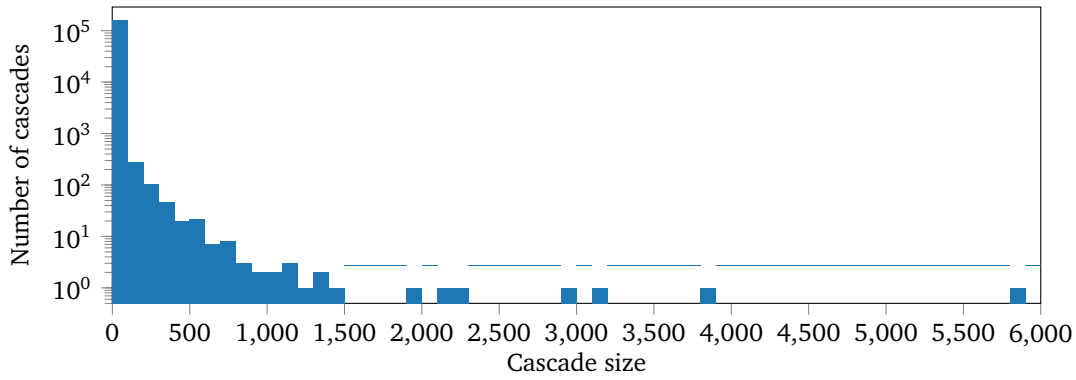


Figure 8.2. Distribution of cascade sizes (number of tweets per cascade) in our dataset.

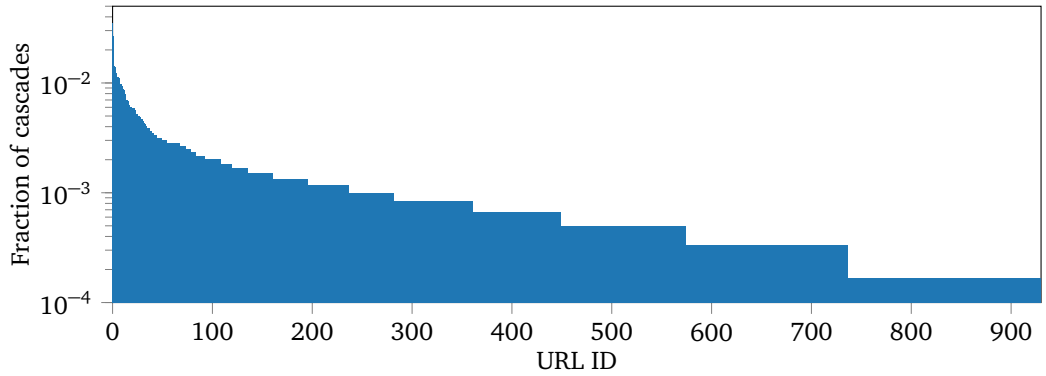


Figure 8.3. Distribution of cascades over the 930 URLs available in our dataset with at least six tweets per cascade, sorted by the number cascades in descending order. The first 15 URLs ($\sim$ 1.5% of the entire dataset) correspond to 20% of all the cascades.

to the propagation of news associated with a particular URL. Following the nomenclature of [255], we term as *cascade* the news diffusion tree produced by a *source* tweet referencing a URL and all of its retweets. For each URL, we searched for all the related cascades and enriched their Twitter-based characterization (i.e. users and tweet data) by drawing edges among users according to Twitter's social network (see example in Figure 8.1).

With regard to this last step of data collection, our approach is significantly different from the protocol of [255], where tweets linking to a fact-checking website were collected, thus essentially retrieving only cascades in which someone has posted a 'proof-link' with the veracity of the news. Though significantly more laborious, we believe that our data collection protocol produces a much cleaner dataset.

Statistics. Figures 8.2–8.3 depict the statistics of our dataset. Overall, our collection consisted of 1,084 labeled claims, spread on Twitter in 158,951 cascades covering the period from May 2013 till January 2018. The total number of unique users involved in the spreading was 202,375 and their respective social graph comprised 2,443,996 edges. As we gathered 1,129

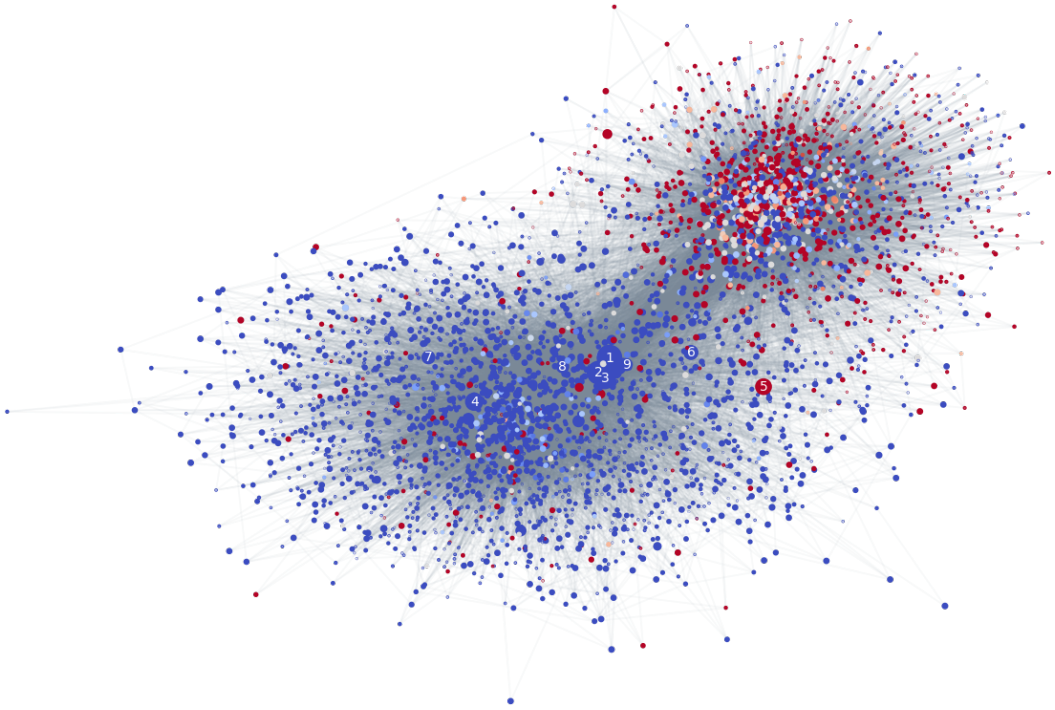


Figure 8.4. Subset of the Twitter network used in our study with estimated user credibility. Vertices represent users, gray edges the social connections. Vertex color and size encode the user credibility (blue = reliable, red = unreliable) and number of followers of each user, respectively. Numbers 1 to 9 represent the nine users with most followers.

URLs, the average number of article URLs per claim is around 1.04; as such, a URL can be considered as a good proxy for a claim in our dataset and we will thus use the two terms synonymously hereinafter. We also note that, similarly to [255], a large proportion of cascades were of small size (the average number of tweets and users in a cascade is 2.79, see also Figure 8.2 depicting the distribution of cascade sizes), which required to use a threshold on a minimum cascade size for classifying these independently in some experiments (see details in Section 8.4.1).

Features. We extracted the following features describing news, users, and their activity, grouped into four categories: *User profile* (geolocalization and profile settings, language, word embedding of user profile self-description, date of account creation, and whether it has been verified), *User activity* (number of favorites, lists, and statuses), *Network and spreading* (social connections between the users, number of followers / friends, cascade spreading tree, retweet timestamps, retweet source device, and number of replies / quotes / favorites / retweets for the source tweet), and *Content* (word embedding of the tweet textual content and included hashtags).

Credibility and polarization. The social network collected in our study manifests noticeable polarization depicted in Figure 8.4. Each user in this plot is assigned a credibility score in the range $[-1, +1]$ computed as the difference between the proportion of (re)tweeted true and fake news (negative values representing fake are depicted in red; more credible users are

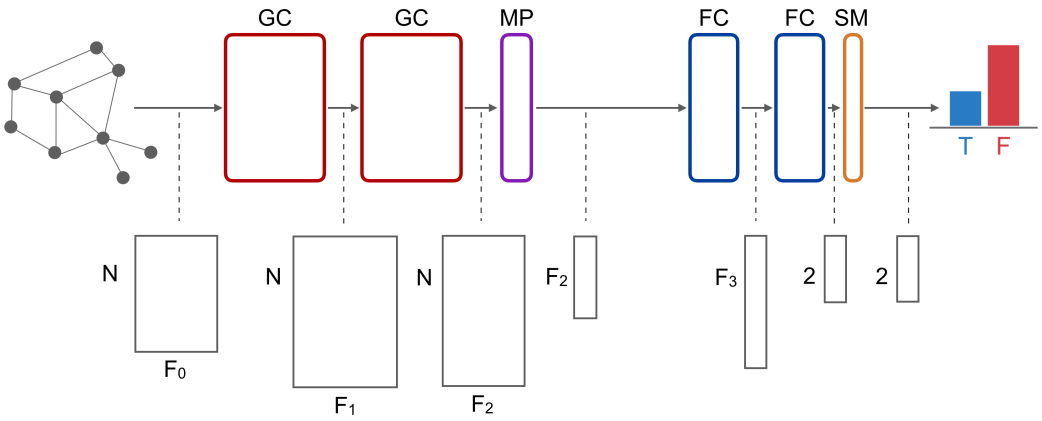


Figure 8.5. The architecture of our neural network model. Top row: GC = Graph Convolution, MP = Mean Pooling, FC = Fully Connected, SM = SoftMax layer. Bottom row: input/output tensors received/produced by each layer.

represented in blue). The node positions of the graph are determined by topological embedding computed via the Fruchterman-Reingold force-directed algorithm [91], grouping together nodes of the graph that are more strongly connected and mapping apart nodes that have weak connections. We observe that credible (blue) and non-credible (red) users tend to form two distinct communities, suggesting these two categories of tweeters prefer to have mostly homophilic interactions (i.e. interactions with similar users). Similar polarization has been observed before in social networks, e.g. in the context of political discourse [64] and might be related to 'echo chamber'[5] theories that attempt to explain the reasons for the difference in fake and true news propagation patterns.

8.3 Methodology

8.3.1 Architecture and training settings

To build our classifier (which we named *VeritasZero*), we used a four-layer GCNN with two convolutional layers (64-dimensional output features map in each) and two fully connected layers (producing 32- and 2-dimensional output features, respectively) to predict the fake/true class probabilities. Figure 8.5 depicts a block diagram of our model. One head of graph attention [252] was used in every convolutional layer to implement the filters together with mean-pooling for dimensionality reduction. We used *Scaled Exponential Linear Unit* (SELU) [139] as non-linearity throughout the entire network. Hinge loss was employed to train the neural network (we preferred hinge loss to the more commonly used cross entropy loss as it outperformed the latter in early experiments). No regularization was used with our model.

[5] An *echo chamber* is a closed environment where people are exposed only to information that reinforces (over and over) their beliefs, potentially resulting in polarization. Such a phenomenon is particularly prevalent on social media where users tend to follow only people with opinions similar to their own [230].

8.3.2 Input generation

Given a URL u (or a cascade c arising from u) with corresponding tweets $T_u = \{t_u^1, t_u^2, ..., t_u^N\}$ mentioning it, we describe u in terms of graph $\mathcal{G}_u$. $\mathcal{G}_u$ has tweets in T_u as nodes and estimated news diffusion paths plus social relations as edges. In other words, given two nodes i and j, edge $(i, j) \in \mathcal{G}_u$ iff at least one of the following holds: i *follows* j (i.e. the author of tweet i follows the author of tweet j), j *follows* i, news spreading occurs from i to j, or from j to i.

News diffusion paths defining *spreading trees* were estimated as in [255] by jointly considering the timestamps of involved (re)tweets and the social connections between their authors. In particular, given t_u^n (the retweet of a cascade related to URL u) authored by user a_u^n, and $\{t_u^0 \dots t_u^{n-1}\}$ the (re)tweets immediately preceding t_u^n belonging to the same cascade and authored by users $\{a_u^0, \dots, a_u^{n-1}\}$, then:

1. if a_u^n *follows* at least one user in $\{a_u^0, \dots, a_u^{n-1}\}$, we estimate news spreading to t_u^n from the very last tweet in $\{t_u^0 \dots t_u^{n-1}\}$ whose author is *followed* by a_u^n;
2. if a_u^n does not *follow* any of the users in $\{a_u^0, \dots, a_u^{n-1}\}$, we conservatively estimate news spreading to t_u^n from the user in $\{a_u^0, \dots, a_u^{n-1}\}$ having the largest number of followers (i.e. the most popular one).

Finally, nodes and edges of graph $\mathcal{G}_u$ have features describing them. Nodes, representing tweets and their authors, were characterized with all the features presented in Section 8.2[6]. As for edges, we used features representing the membership to each of the aforementioned four relations (i.e. *following* and *news spreading*, both directions). Our approach to defining graph connectivity and edge features allows, in graph convolution, to spread information independently of the relation direction while potentially giving different importance to the types of connections. Features of edge (i, j) are concatenated to those of nodes i and j in the attention projection layer to achieve such behavior.

8.4 Results

We considered two different settings of fake news detection: *URL-wise* and *cascade-wise*, using the same architecture for both settings. In the first setting, we attempted to predict the true/fake label of a URL containing a news story from all the Twitter cascades it generated. On average, each URL resulted in approximately 141 cascades. In the latter setting, which is significantly more challenging, we assumed to be given only one cascade arising from a URL and attempted to predict the label associated with that URL. Our assumption is that all the cascades associated with a URL inherit the label of the latter. While we checked this assumption to be true in most cases in our dataset, it is possible that an article is for example tweeted with a comment denying its content. We left the analysis of comments accompanying tweets/retweets as a future research direction.

8.4.1 Model performance

For URL-wise classification, we used five randomized training/test/validation splits. On average, the training, test, and validation sets contained 677, 226, and 226 URLs, respectively,

[6] For tweet content and user description embeddings we averaged together the embeddings of the constituent words (GloVe [200] 200-dimensional vectors pre-trained on Twitter).

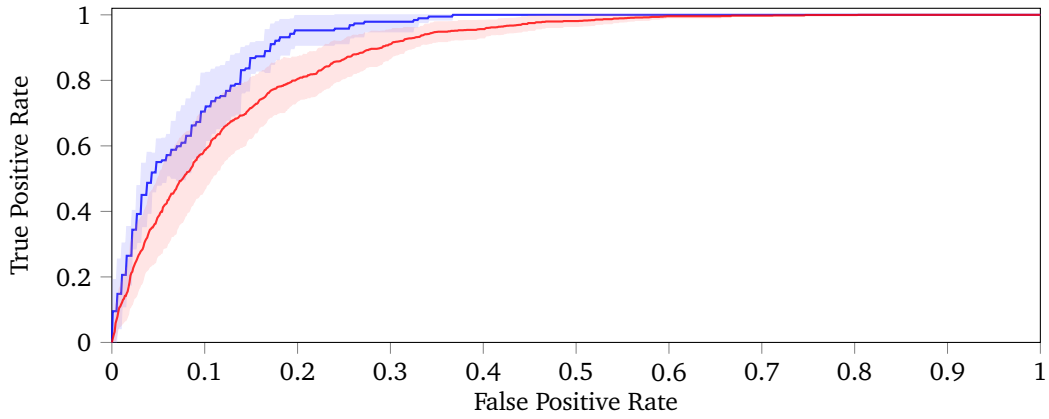


Figure 8.6. Performance of URL-wise (blue) and cascade-wise (red) fake news detection using 24hr-long diffusion time. Shown are ROC curves averaged on five folds (the shaded areas represent the standard deviations). ROC AUC is 92.70 ± 1.80% for URL-wise classification and 88.30 ± 2.74% for cascade-wise classification, respectively. Only cascades with at least 6 tweets were considered for cascade-wise classification.

with 83.26% true and 16.74% false labels (± 0.06% and 0.15% for training and validation/test set respectively). For cascade-wise classification we used the same split initially realized for URL-wise classification (i.e. all cascades originated by URL u are placed in the same fold as u). Cascades containing less than 6 tweets were discarded; the reason for the choice of this threshold is motivated below. Full cascade duration (24 hr) was used for both settings of this experiment. The training, validation, and test sets contained on average 3586, 1195, 1195 cascades, respectively, with 81.73% true and 18.27% false labels (± 3.25% and 6.50% for training and validation/test set respectively).

Our neural network was trained for 25,000 and 50,000 iterations in the URL-wise and cascade-wise settings, respectively, using AMSGrad [215] with learning rate of 5×10^{-4} and mini-batch of size one.

Figure 8.6 depicts the performance of URL-wise (blue) and cascade-wise (red) fake news classification represented as a tradeoff (ROC curve) between false positive rate (fraction of true news wrongly classified as fake) and true positive rate (fraction of fake news correctly classified as fake). We use *area under the ROC curve* (ROC AUC) as an aggregate measure of accuracy. On the above splits, our method achieved mean ROC AUC of 92.70 ± 1.80% and 88.30 ± 2.74% in the URL-wise and cascade-wise settings, respectively.

Figure 8.7 depicts a low-dimensional plot of the last graph convolutional layer vertex-wise features obtained using t-SNE embedding. The vertices are colored using the credibility score defined in Section 8.2. We observe clear clusters of reliable (blue) and unreliable (red) users, which is indicative of the neural network learning features that are useful for fake news classification.

Influence of minimum cascade size. One of the characteristics of our dataset (as well as the dataset in the study of [255]) is the abundance of small cascades containing just a few users (see Figure 8.2). Since our approach relies on the spreading of news across the Twitter social network, such examples may be hard to classify, as too small cascades may manifest no clear

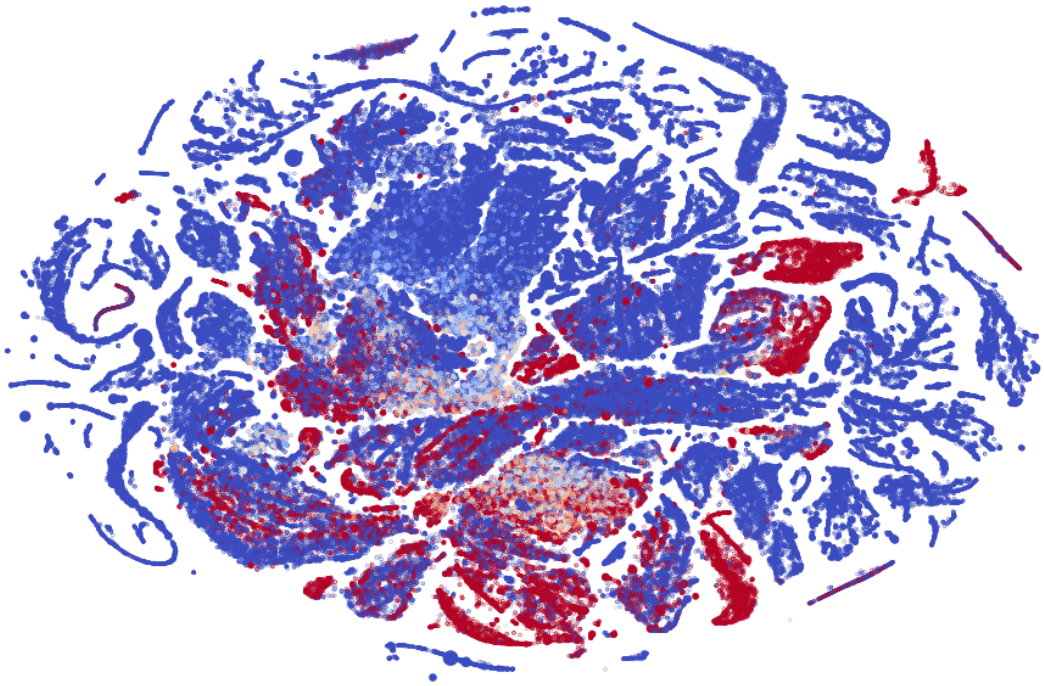


Figure 8.7. T-SNE embedding of the vertex-wise features produced by our neural network at the last convolutional layer representing all the users in our study, color-coded according to their credibility (blue = reliable, red = unreliable). Clusters of users with different credibility clearly emerge, indicative that the neural network learns features useful for fake news detection.

diffusion pattern. To identify the minimum useful cascade size, we investigated the performance of our model in the cascade-wise classification setting using cascades of various minimum sizes (Figure 8.8). As expected, the model performance increases with larger cascades, reaching saturation for cascades of at least 6 tweets (leaving a total of 5,976 samples). This experiment motivates our choice of using 6 tweets as the minimum cascade size in cascade-wise experiments in our study.

Ablation study. To further highlight the importance of the different categories of features provided as input to the model, we conducted an ablation study by means of backward-feature selection. We considered for this the four groups of features defined in Section 8.2: *user profile*, *user activity*, *network and spreading*, and *content*. The results of ablation experiment are shown in Figure 8.9 for the URL-wise (top) and cascade-wise (bottom) settings. In both settings, user-profile and network/spreading appear as the two most important feature groups, and allow achieving satisfactory classification results (near 90% ROC AUC) with the proposed model.

Interestingly, in the cascade-wise setting, while all features were positively contributing to the final predictions at URL-level, removing tweet content from the provided input improves performance by 4%. This seemingly contradictory result can be explained by looking at the distribution of cascades over all the available URLs (Figure 8.3): 20% of cascades are associated to the top 15 largest URLs in our dataset ($\sim$ 1.5% out of a total of 930). Since tweets

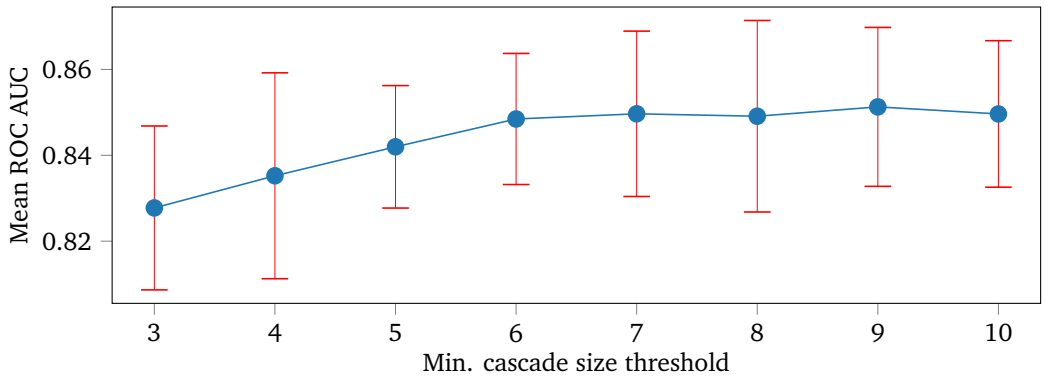


Figure 8.8. Performance of cascade-wise fake news detection (mean ROC AUC, averaged on five folds) using minimum cascade size threshold. Best performance is obtained by filtering out cascades smaller than 6 tweets.

citing the same URL typically present similar content, it is easy for the model to overfit on this particular feature. Proper regularization (e.g. dropout or weight decay) should thus be introduced to avoid overfitting and improve performance at test time (we left this further study for future research). For simplicity, by leveraging the capabilities of our model to classify fake news in a content-free scenario, we decided to completely ignore content-based descriptors (tweet word embeddings) for cascade-wise classification and let the model exploit only user- and propagation-related features.

8.4.2 News spreading over time

One of the key differentiators of propagation-based methods from their content-based counterparts, namely relying on the news spreading features, potentially raises the following question: for how much time do the news have to spread before we can classify them reliably? We conducted a series of experiments to study the extent to which this is the case with our approach.

For this purpose, we truncated the cascades after time t starting from the first tweet, with t varying from 0 (effectively considering only the initial tweet, i.e. the 'root' of each cascade) to 24 hours (the full cascade duration) with one hour increments. The model was trained separately for each value of t. Five-fold cross validation was used to reduce the bias of the estimations while containing the overall computational cost.

Figure 8.10 depicts the performance of the model (mean ROC AUC) as function of the cascade duration, for the URL-wise (top) and cascade-wise (bottom) settings. As expected, performance increases with the cascade duration, saturating roughly after 15 hours in the URL-wise setting and after 7 hours in the cascade-wise one, respectively. This different behavior is mainly due to the simpler topological patterns and shorter life of individual cascades. Seven hours of spreading encompass on average around 91% of the cascade size; for the URL-wise setting, the corresponding value is 68%. A similar level of coverage, 86%, is achieved after 15 hours of spreading in the URL-wise setting.

We also note that remarkably just a few (~ 2) hours of news spread are sufficient to achieve above 90% mean ROC AUC in URL-wise fake news classification. Furthermore, we observe a significant jump in performance from the 0 hr setting (effectively using only user profile, user ac-

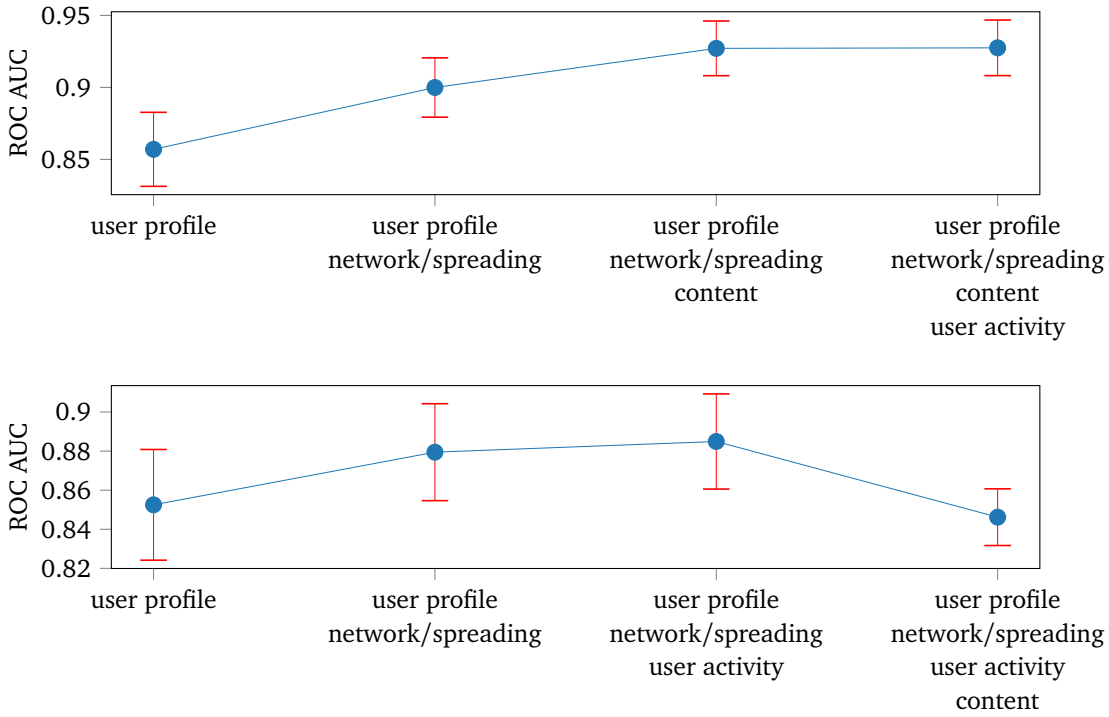


Figure 8.9. Ablation study result on URL-wise (top) / cascade-wise (bottom) fake news detection, using backward feature selection. Shown is performance (ROC AUC) for our model trained on subsets of features, grouped into four categories: user profile, network and spreading, content, and user activity. Groups are sorted for importance from left to right.

tivity, and content features) to ≥ 1 hr settings (considering additionally the news propagation), which we interpret as another indication of the importance of propagation-related features.

8.4.3 Model aging

We live in a dynamic world with constantly evolving political context. Since the social network, user preferences and activity, news topics and potentially also spreading patterns evolve in time, it is important to understand to what extent a model trained in the past can generalize to such new circumstances. In the final set of experiments of this chapter, we study how the model performance ages with time in the URL-wise and cascade-wise settings. These experiments aim to emulate a real-world scenario in which a model trained on historical data is applied to new tweets in real time.

For the URL-wise setting, we split our dataset into training/validation (80% of URLs) and test (20% of URLs) sets; the training/validation and test sets were disjoint and subsequent in time. We assessed the results of our model on subsets of the test set, designed as partially overlapping (mean intersection over union equal to 0.56 ± 0.15) time windows. Partial over-

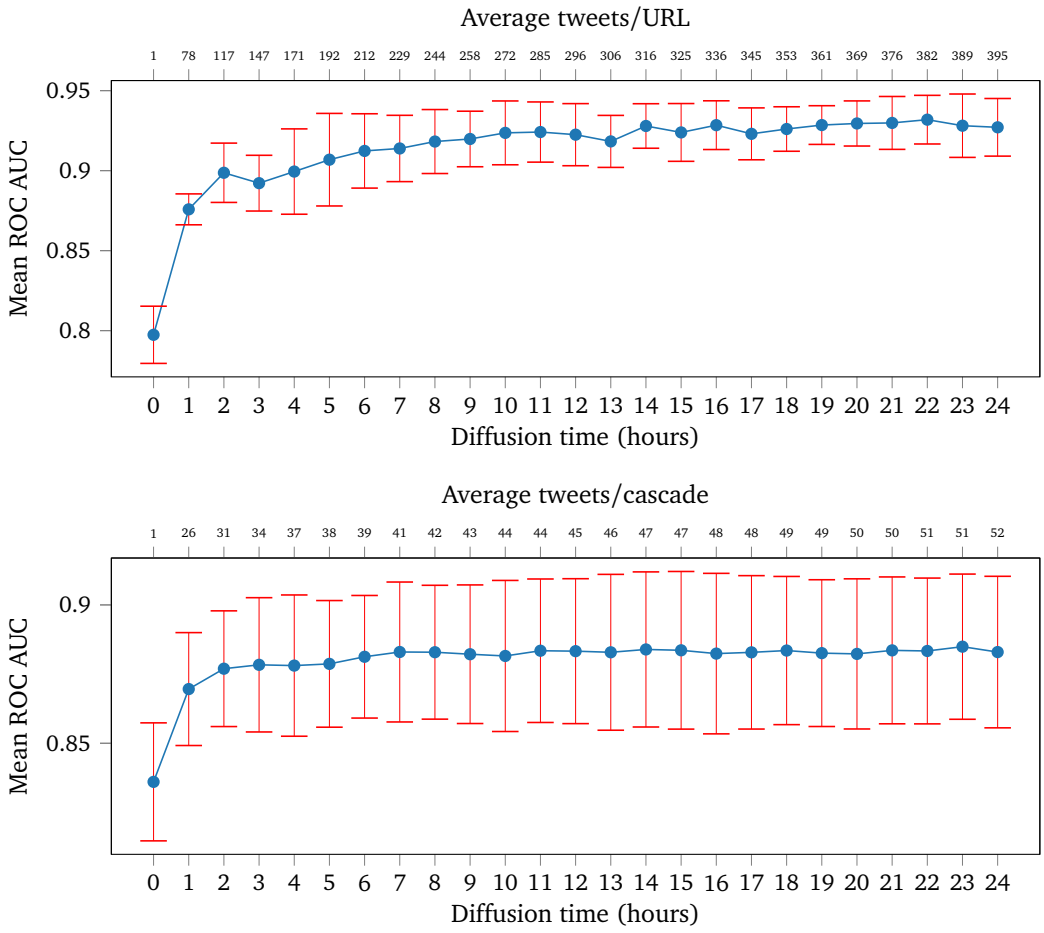


Figure 8.10. Performance of URL-wise (top) and cascade-wise (bottom) fake news detection (mean ROC AUC, averaged on five folds) as function of cascade diffusion time.

lap allowed us to work on larger subsets while preserving the ratio of positives vs negatives, providing at the same time smoother results as with moving average. This way, each window contained at least 24% of the test set (average number of URLs in a window was 73 ± 33.34) and the average dates of two consecutive windows were at least 14 days apart, progressively increasing.

Figure 8.11 (top) captures the variation in performance due to aging of the training set in the URL-wise setting. Our model exhibits a slight deterioration in performance only after 180 days. We attribute this deterioration to the change in the spreading patterns and the user activity profiles.

We repeated the same experiment in the cascade-wise setting. The split into training/validation and test sets and the generation of the time windows was done similarly to the URL-wise experiment. Each time window in the test set has an average size of 314 ± 148 cascades, and two consecutive windows had a mean overlap with intersection over union equal to 0.68 ± 0.21. Figure 8.11 (bottom) summarizes the performance of our model in the cascade-wise setting. In

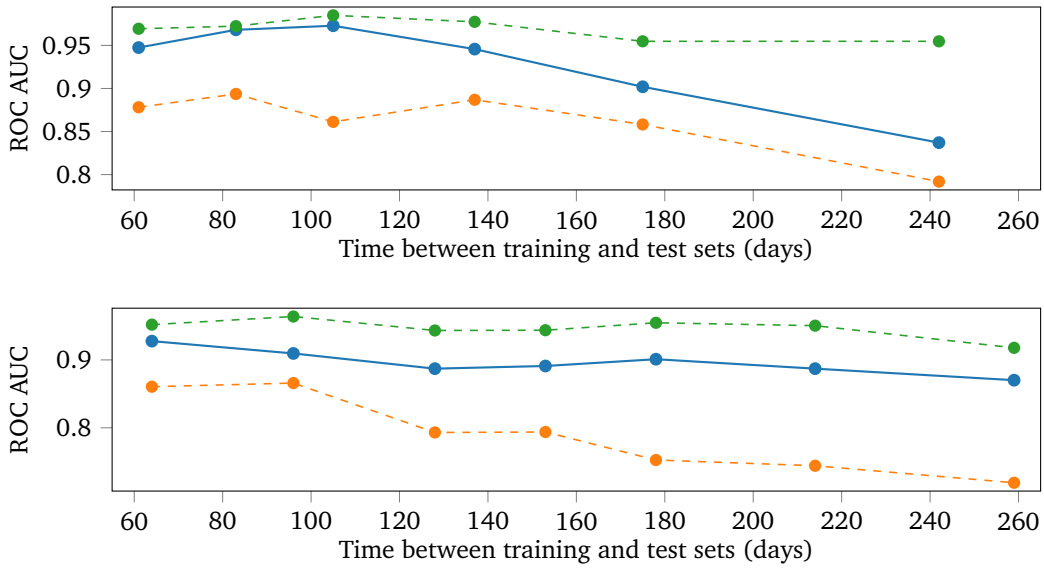


Figure 8.11. Effects of training set aging on the performance of URL-wise (top) and cascade-wise (bottom) fake news detection. Horizontal axis shows difference in days between average date of the training and test sets. Shown is the test performance obtained by our model with 24hrs diffusion (solid blue), test performance obtained with same model just using the first tweet of each piece of news (0hrs diffusion, dashed orange), and test performance obtained training on our original uniformly sampled five folds (veracity predictions are computed for each URL/cascade when this appears as a test sample in our 24hrs five fold cross-validation, dashed green).

this case, it shows a more robust behavior compared to the URL-wise setting, losing only 4% after 260 days.

This different behavior is likely due to the higher variability that characterizes cascades as opposed to URLs. As individual cascades are represented by smaller and simpler graphs, the likelihood of identifying recurrent rich structures between different training samples is lower compared to the URL-wise setting and, also, cascades may more easily involve users coming from different parts of the Twitter social network. In the cascade-wise setting, our propagation-based model is thus forced to learn simpler features that on the one hand are less discriminative (hence the lower overall performance), and on the other hand appear to be more robust to aging.

8.5 Discussion

Future research. There are multiple intriguing phenomena and hypotheses left for future research in this work. Of particular interest is the experimental validation of the conjecture that content agnostic features extracted from the propagation of a given article could allow to realize models that are geography-independent. The study of adversarial attacks is also of great interest, both from theoretical and practical viewpoints. We conjecture that attacks on

graph-based approaches require social network manipulations that are difficult to implement in practice. Finally, while our entire model was designed with fake news detection in mind, the proposed architecture is general and can be also applied to other applications in the realm of social network data analysis. Topic classification and virality prediction could in this sense be interesting research directions to explore with an architecture like ours.

Related work. As mentioned at the beginning of this chapter, the approach described herein was the core technology of Fabula AI. Our own startup that was acquired by Twitter back in 2019. To the best of our knowledge, Fabula AI was the first GDL-based startup that was ever acquired by a major technology company, and VeritasZero was one of the earliest solutions based on GCNNs for the detection of fake news that appeared in the literature[7]. From the publication of our original paper, several other researchers investigated how GCNNs can be used to mitigate misinformation [203, 101]. Here, we follow the categorization set forth in [101], and we briefly mention relevant publications that provide the general direction the field took from the publication of our original paper (for a more complete overview of the listed methods, as well as others, the reader is invited to refer to [203, 101]).

Generally speaking, two classes of approaches are mostly related to our work: *propagation-based methods*, and *global context-based methods*[8].

As previously described in this chapter, propagation-based solutions aim at classifying a piece of news dependently on how this spreads on social media. In this direction, today we find in the literature approaches that combine multiple types of interactions (e.g. retweets, likes or replies) to fully capture the diffusion of a given article / post on the social network [246], approaches that use contrastive learning to make a given architecture more robust to missing or fake interactions [115, 240, 163, 165], approaches that leverage the users historical behavior to better characterize their properties [80], and approaches that treat cascades as temporal networks to capture the evolution of their properties through time [59, 239].

Global context-based solutions, on the other hand, make predictions based on both the interactions that pieces of news receive, as well as how these relate to each other. In this class of methods, an heterogeneous graph depicting how multiple entities (e.g. articles, users, news sources, topics, ...) connect one to the other is built at first, and then predictions are made based on the behavior of a neighborhood of a target article [275, 195, 178]. The main benefit of this class of approaches is that, differently from propagation-based approaches that process articles "in a vacuum", global context-based solutions allow the exchange of information across related articles at prediction time (e.g. articles that are shared by the same user, refer to the same entities, or are authored by the same source). This in turn allows to exploit a larger amount of information for inference, which in turn can be beneficial for improving performance.

[7] The only work prior to ours that we are aware of is [167], where a RNN is used to diffuse information over cascades of posts for rumor detection.

[8] Please note, what we refer to as "global context-based methods" is what is described in [101] as "heterogenous social context-based methods". We rename this class of approaches for the sake of consistency with the definitions outlined at the beginning of this chapter (we also avoid the use of adjective "heterogenous", as heterogenous graphs can be used to represent both the spreading of multiple articles, and the the diffusion a single piece of news [181]).

Chapter 9

User identification in datasets of pseudonymized interaction networks

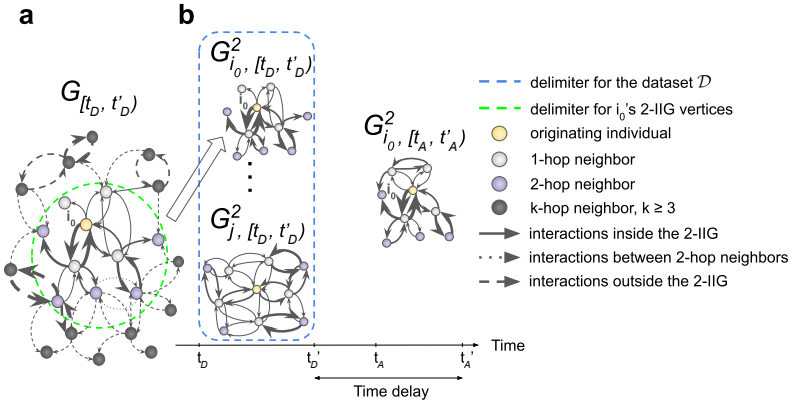
This chapter is based on "Ana-Maria Creţu, Federico Monti, Stefano Marrone, Xiaowen Dong, Michael Bronstein, and Yves-Alexandre de Montjoye. Interaction data are identifiable even across long periods of time. Nature Communications*, 13(1):313, 2022". Here, the author collaborated with Ana-Maria Creţu on the design of the method, and the design and review of the experiments.*

9.1 Introduction

An increasing fraction of our online and offline interactions is now captured by technology [149]. Large amounts of interaction data are collected everyday by messaging apps, mobile phone carriers, social media companies, and other apps to operate their service or for research purposes. Interaction data typically consist of the pseudonyms of the interaction parties, the timestamp of the interaction, and possibly further information describing more in detail the interaction (e.g. length or place of interaction). As such, interaction data are deeply personal and sensitive data. They record with high precision who we talk to or meet, at what time, and for how long or where. Sensitive information can furthermore be inferred from interaction data. Previous research showed in this sense how algorithms can predict from interaction data who a person's significant other is [21], their wealth [162, 34], demographics [87] or propensity to overspend [234].

Interaction data can be shared or sold to third parties without users' consent, so long as they are anonymized. According to data protection regulations, such as the European Union's General Data Protection Regulation (GDPR) [9], anonymized (or de-identified) data are indeed no longer considered as personal data. The European Data Protection Board (EDPB) predecessor (the Article 29 Working Party) defined anonymization as resistance to singling out, linkability, and inference attacks [8]. In particular, the linkability criterion refers to "the ability to link, at least, two records concerning the same data subject." While guidances are subject to the interpretation of the courts, matching identities between two pseudonymous datasets would likely mean that they are not anonymous under GDPR.

Matching attacks have long been used to identify individuals in datasets using matching auxiliary information, calling into question their anonymity. In [244], it was shown for instance



c

Party A	Party B	Timestamp	Type	Direction	Duration (secs)
aSG64X	rxJKc9	2020-11-05 20:00:05	call	in	325
gvuQjU	dPefYb	2020-11-06 10:23:11	text	in	-
gvuQjU	dPefYb	2020-11-06 10:25:13	text	out	-
gvuQjU	LUrKAk	2020-11-06 10:47:20	call	out	124
⋮	⋮	⋮	⋮	⋮	⋮

Figure 9.1. Setup of the behavioral profiling attack. (a) An example of a 2-IIG highlighted in the larger graph it comes from. The vertices of the 2-IIG (inside the dashed green circle) are respectively the originating individual (in yellow), 1-hop neighbors (in gray), and the 2-hop neighbors (in purple). In solid lines are the edges that are part of the 2-IIG. Edges are shown as a single directed edge of thickness proportional to the total number of interactions. (b) The data available to the attacker consist of (left) 2-IIGs coming from time period $[t_{\mathcal{D}}, t'_{\mathcal{D}})$, usually as part of an anonymized dataset, and (right) auxiliary 2-IIG data about a target individual ($\mathcal{G}^2_{i_0,[t_{\mathcal{A}},t'_{\mathcal{A}})}$). (c) An example of mobile phone interaction data.

how the Governor of Massachusetts of 1997 (William Weld) could be identified in a medical dataset just using his zip code, birth date and gender. In [194], the movies people had watched were used to identify users in the famous Netflix Prize dataset [26]. In 2013, de Montjoye et al. showed how four spatio-temporal points (describing approximate locations and times of individuals) were enough to uniquely identify someone in location data 95% of the time [69]. In [66], we proposed a *profiling attack* for interaction data based on GCNNs. While matching attacks rely on auxiliary information that is fairly stable over time (e.g. gender, zip code, etc.) or from the same time period (e.g. movies watched), profiling attacks use auxiliary information from one time period to profile and identify a person in another non-overlapping time period. This makes them more broadly applicable (and thus potentially appealing to an attacker), as the auxiliary data does not have to come from the same time period as the dataset.

Using a Graph Attention Network [252], we learn in particular an individual's behavioral profile by extracting a vector representation of their *weekly* k-hop interaction network. Our weekly profiles use only behavioral features, aggregating both node features and topological information typically present in interaction data, and are optimized for identification. In a mobile phone dataset of more than 40,000 people, our model was able to correctly identify

a person 52% of the time based on their 2-hop interaction network ($k = 2$). Using only a person's interactions with their direct contacts ($k = 1$), our model could still identify them 15% of the time. We further show that the accuracy of our model only decreases slowly as time passes with 24% of the people still being correctly identified after 20 weeks ($k = 2$), thus making identification a real risk in practice. In a Bluetooth close-proximity dataset of more than 500 people [223], our model is able to link together 1-hop interaction networks with 26% accuracy. Our results provide evidence that disconnected and even frequently re-pseudonymized interaction data (i.e. where users pseudonyms are changed often over time) remain identifiable even across long periods of time, and may thus not satisfy the anonymization standard set forth by the EDPB in particular with regard to the linkability criteria.

9.2 Experimental setup

9.2.1 Overview of the attack

Our attack exploits the stability over time of people's interaction patterns to identify individuals using k-hop interaction data.

We consider a service S collecting data about the interactions it is mediating. We denote by $\mathcal{I}$ the set of individuals taking part in the communications recorded by S. For example, $\mathcal{I}$ could be the subscribers of a mobile phone carrier and their contacts, the set of users of a contact tracing app, the users of a messaging application, etc. We call interaction data the record describing the interaction between two individuals using S, consisting of the pseudonyms of the two individuals, a timestamp, and potentially other information. We define a time period $\mathcal{T} = [t, t')$ as the set of all timestamps between a start t (inclusive) and end t' (exclusive). Given a time period $\mathcal{T}$, we define the interaction graph $\mathcal{G}_\mathcal{T}$ as the directed multi-graph with node set $\mathcal{I}$ and an edge between two nodes for each interaction between the corresponding individuals at a timestamp in the time period $\mathcal{T}$. Each edge between two nodes is equipped with additional data $\mathbf{e}$ describing the interaction. For example: if S is a mobile operator, $\mathbf{e}$ would be a tuple containing the timestamp, the type of interaction (call or text), its direction (i.e. which party initiated it), and the duration for calls; if S is a close-proximity app, $\mathbf{e}$ would be the timestamp and the strength of the signal; etc. Given a time period $\mathcal{T}$, $i \in \mathcal{I}$ an individual and $k = 1, 2, \ldots$, we define the *k-hop Individual Interaction Graph* (k-IIG) $\mathcal{G}^k_{i,\mathcal{T}}$ of user i as the sub-graph induced in $\mathcal{G}_\mathcal{T}$ by the set of nodes situated on paths of length at most k starting at i, excluding interactions between the k-hop neighbors themselves (Figure 9.1 shows an example of a 2-IIG).

Our attack model assumes that a malicious agent / attacker has access to (1) an anonymous dataset $\mathcal{D} = \{\mathcal{G}^k_{i,[t_\mathcal{D}, t'_\mathcal{D})} \mid i \in \mathcal{I}_\mathcal{D}\}$ consisting of the k-IIGs of people in $\mathcal{I}_\mathcal{D} \subset \mathcal{I}$ from time period $\mathcal{T}_\mathcal{D} = [t_\mathcal{D}, t'_\mathcal{D})$, as well as to (2) auxiliary data $\mathcal{G}^k_{i_0,\mathcal{T}_\mathcal{A}}$ consisting in the k-IIG of a known target individual $i_0 \in \mathcal{I}_\mathcal{D}$, coming from a disjoint time period $\mathcal{T}_\mathcal{A} = [t_\mathcal{A}, t'_\mathcal{A})$ (i.e. $t'_\mathcal{D} \leq t_\mathcal{A}$ or $t'_\mathcal{A} \leq t_\mathcal{D}$, we denote by time delay the quantity $D = t'_\mathcal{A} - t'_\mathcal{D}$). We further assume that the attacker knows, for each k-IIG, which node is at the center of the k-IIG (i.e. the originating node), and that the k-IIGs are pseudonymized, meaning that a node will have a different pseudonym in each graph it appears in. The attacker's goal is to find the target i_0 in $\mathcal{D}$, which in our setting translates in finding the $\mathcal{G}^k_{i,[t_\mathcal{D}, t'_\mathcal{D})} \in \mathcal{D}$ such that $i = i_0$. If successful, the attacker is said to have identified i_0 and is able to retrieve all their pseudonymized interactions from time period $[t_\mathcal{D}, t'_\mathcal{D})$.

To reproduce a realistic scenario, in our experiments we assumed that an attacker has only

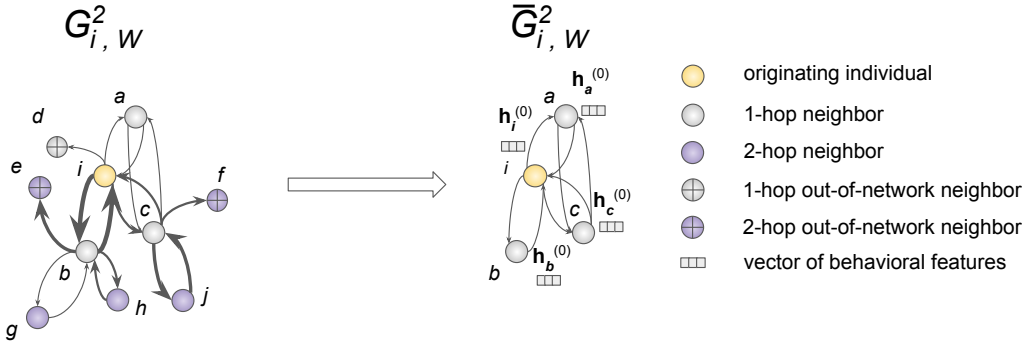


Figure 9.2. An example of a simplified 2-IIG. On the left, a 2-IIG $\mathcal{G}^2_{i,\mathcal{W}}$, with vertex set consisting of originating individual i (yellow), and its 1-hop (a, b, c and d, gray) and 2-hop (e, f, g, h and j, purple) neighbors. Edges between nodes are displayed as arrows with thickness proportional to the number of interactions. The nodes marked with + (d, e and f) can be considered as out-of-network, as they only have incoming edges in the 2-IIG. On the right, the simplified 2-IIG $\bar{\mathcal{G}}^2_{i,\mathcal{W}}$, consisting of the originating individual i and the 1-hop neighbors, with one edge between any two nodes if there was at least one edge in 2-IIG. In the simplified 2-IIG, all nodes are equipped with behavioral features.

access to weekly interactions in the auxiliary data (i.e. $[t_{\mathcal{A}}, t'_{\mathcal{A}})$ equals one week). In line with this, the attacker splits the k-IIGs from $\mathcal{D}$ by weeks to obtain multi-graphs describing time periods with same length of the one available in the auxiliary data: $\{\mathcal{G}^k_{i,\mathcal{W}_n} : i \in \mathcal{I}_{\mathcal{D}}, n \in \{1,\ldots,T'\}\}$. Here $\mathcal{T}_{\mathcal{D}} = \mathcal{W}_1 \cup \ldots \cup \mathcal{W}_{T'}$ and $\mathcal{W}_n = [t_n, t_{n+1})$ is the n-th week of the dataset ($\mathcal{W}_n \cap \mathcal{W}_{n'} = \emptyset$ if $n \neq n'$). The attacker singles out the *most recent week* $\mathcal{W}_{T'}$ in $\mathcal{T}_{\mathcal{D}}$, and uses this for identifying the target user. The remaining data in $\mathcal{T}_{\mathcal{D}}$ are used to train the profiles of k-IIGs. We used the most recent week in $\mathcal{T}_{\mathcal{D}}$ for identification as in our experiments $\mathcal{T}_{\mathcal{A}}$ always follows $\mathcal{T}_{\mathcal{D}}$. Under the assumption that user behavior changes gradually over time, the most recent week of interactions $\mathcal{W}_{T'}$ is the one that, most likely, better describes the behavior of the target user in $\mathcal{T}_{\mathcal{A}}$.

9.2.2 Preprocessing of a k-IIG

To train a model for identification, the attacker extracts behavioral features of each node available in a k-IIG, using the Bandicoot toolbox [70]. Bandicoot is an open-source Python library used to compute a set of behavioral features from an individual's list of mobile phone interactions. Bandicoot takes as input an individual's list of interactions, consisting of the other party's unique identifier, the interaction timestamp, type (*call* or *text*), direction (*in* or *out*), and duration (if a call). The features range from simple aggregated features (e.g. the number of voice and text contacts), to more sophisticated statistics (e.g. the percentage of an individual's contacts that account for 80% of their interactions).

Using Bandicoot, the attacker extracts a set of features (Table 9.3 and 9.4) for all nodes in a weekly k-IIG with out-degree ≥ 1 that are at most $k-1$ hops away from the originating individual. For Bluetooth close-proximity data, the behavior of Bandicoot is adapted by setting the type of interaction to *call*, the direction to *out*, and the call duration to the *negative*

RSSI recorded for the interaction[1]. The positive out-degree is here a proxy for a node being a subscriber to service S, which is the service providing the data. Being a subscriber to S is of relevance in our experiments as it determines whether we have all the interactions made by a specific user or not (e.g. all the interactions of the subscribers to a mobile phone carrier can be available in a dataset released by said carrier, however only interactions initiated / received by a subscriber to the carrier are available for non-subscribers). To the behavioral features extracted with Bandicoot, the attacker adds:

- for the mobile phone dataset, estimates[2] of the percentage of out-of-network calls, texts, call durations and contacts based on the information available in the k-IIG;
- for the Bluetooth-close proximity dataset, the number of empty device scans and percentage of out-of-network interactions[3].

The attacker further removes the featureless nodes from the k-IIG and collapses all directed edges between two remaining nodes into a single directed edge of the same direction. The attacker thus "simplifies" the (multi-graph) k-IIG $\mathcal{G}^k_{i,\mathcal{W}} = (\mathcal{V}, \mathcal{E})$ to obtain the *simplified k-IIG* $\bar{\mathcal{G}}^k_{i,\mathcal{W}} = (\bar{\mathcal{V}}, \bar{\mathcal{E}})$; a simple graph with $\bar{\mathcal{V}} = \{v \in \mathcal{V} \mid v \text{ is on a path of length at most } k-1 \text{ from node } i \wedge v\text{'s out-degree} \geq 1\}$ and $\bar{\mathcal{E}} = \{(v,w) \mid v \in \bar{\mathcal{V}} \wedge w \in \bar{\mathcal{V}} \wedge \exists (v,w,\mathbf{e}) \in \mathcal{E}\}$. Figure 9.2 shows an example of our reduction step.

9.3 Methodology

9.3.1 Model

To match an individual over different intervals of time, our *k-IIG-based Behavioral Profiling attack* (BP-IIG) first computes a time-dependent profile for each user in $\mathcal{I}_\mathcal{D}$ through a Neural Network (NN) applied over their corresponding simplified k-IIG. The structure of the network we use slightly changes dependently on the value of k. For $k \in \{2,3\}$, a Graph Attention Network (GAT) is used to compute embeddings that depend on the features of the IIG's nodes, and how these are connected. For $k = 1$, a multi-layer perceptron is used instead, as simplified 1-IIGs are graphs consisting of a single node and no edges, and a GAT would naturally reduce to a MLP in this scenario. In the case of the Graph Attention Network, for each node v of a provided simplified k-IIG $\bar{\mathcal{G}}^k_{i,\mathcal{W}}$, a feature embedding is computed applying multiple layers of the form:

$$\mathbf{h}_v^{(l)} = \text{MLP}^{(l)}\left(\left[\mathbf{h}_v^{(l-1)}, \sum_{w \in \mathcal{N}_v} \alpha_{v,w}^{(l)} \mathbf{h}_w^{(l-1)}\right]\right) \tag{9.1}$$

$$\alpha_{v,w}^{(l)} = \frac{e^{\text{LeakyReLU}\left((\mathbf{W}_a^{(l)}\mathbf{h}_v^{(l-1)} \| \mathbf{W}_a^{(l)}\mathbf{h}_w^{(l-1)})^T \mathbf{c}_\alpha^{(l)}\right)}}{\sum_{k \in \mathcal{N}_v} e^{\text{LeakyReLU}\left((\mathbf{W}_a^{(l)}\mathbf{h}_v^{(l-1)} \| \mathbf{W}_a^{(l)}\mathbf{h}_k^{(l-1)})^T \mathbf{c}_\alpha^{(l)}\right)}} \tag{9.2}$$

[1] RSSI (Received Signal Strength Indicator) is a measure of strength of the Bluetooth signal received by a device, which could be viewed as a proxy for distance.

[2] We refer to these statistics as estimates, as we can only guess which users are non-subscribers to S from our data.

[3] Interactions with out-of-network devices are marked appropriately in the Bluetooth dataset (ID = -2), as a result we do not refer to these statistics as estimates.

where the output $\mathbf{h}_v^{(l-1)}$ of layer $l-1$ is passed as the input to layer l, with $l \in \{1,\dots,L\}$ ($\mathbf{h}_v^{(0)}$ are here the input features of node v). For each layer $1 \leq l \leq L$, $\text{MLP}^{(l)}$ is a multi-layer perceptron with one hidden layer, followed by $\mathbb{L}_2$-normalization (which was found experimentally beneficial for maximizing performance). $\alpha_{v,w}^{(l)}$ denotes the attention weight computed as a non-linear function of the features of node v and its neighbor $w \in \mathcal{N}_v$. In the case of the MLP, $\mathbb{L}_2$-normalization is used after each layer for consistency.

Independently on the neural network we use for inferring user embeddings, the output features $\mathbf{h}_i^{(L)} = \text{NN}_{\boldsymbol{\Theta}}(\bar{\mathcal{G}}_{i,\mathcal{W}}^k)_i$ of the model for target individual i are used as the embedding of i for the attack ($\boldsymbol{\Theta}$ denotes here the parameters of the model). The network is trained to optimize the matching accuracy, using the triplet loss [224], which optimizes the profile embeddings of the same individual at different time periods (positive pair) to be closer to each other than to those of different individuals at any time period (negative pair). A triplet of simplified k-IIGs $(\bar{\mathcal{G}}_{i,\mathcal{W}}^k, \bar{\mathcal{G}}_{i,\mathcal{W}'}^k, \bar{\mathcal{G}}_{i',\mathcal{W}''}^k)$ contains data from two individuals i and i' (with $i \neq i'$), such that there are two k-IIGs for i, coming from two different weeks $\mathcal{W}$ and $\mathcal{W}'$, and a k-IIG for i' from a time period $\mathcal{W}''$ not necessarily different from $\mathcal{W}$ or $\mathcal{W}'$. Let $\mathbf{h}_{i,\mathcal{W}}(\boldsymbol{\Theta}) = \text{NN}_{\boldsymbol{\Theta}}(\bar{\mathcal{G}}_{i,\mathcal{W}}^k)$, $\mathbf{h}_{i,\mathcal{W}'}(\boldsymbol{\Theta}) = \text{NN}_{\boldsymbol{\Theta}}(\bar{\mathcal{G}}_{i,\mathcal{W}'}^k)$ and $\mathbf{h}_{i',\mathcal{W}''}(\boldsymbol{\Theta}) = \text{NN}_{\boldsymbol{\Theta}}(\bar{\mathcal{G}}_{i',\mathcal{W}''}^k)$ denote the respective embeddings, the triplet loss for our prediction problem is defined as:

$$\ell(\boldsymbol{\Theta}) = \max(0, \|\mathbf{h}_{i,\mathcal{W}}(\boldsymbol{\Theta}) - \mathbf{h}_{i,\mathcal{W}'}(\boldsymbol{\Theta})\|_2 - \|\mathbf{h}_{i,\mathcal{W}}(\boldsymbol{\Theta}) - \mathbf{h}_{i',\mathcal{W}''}(\boldsymbol{\Theta})\|_2 + \lambda). \tag{9.3}$$

As it appears from equation 9.3, minimizing the triplet loss pushes embeddings extracted for a given individual at different intervals of time to be close one to the other, and embeddings of different individuals to be distant one from the other. Once the model is trained on the simplified k-IIG of $\mathcal{D}$, the attacker tries to identify the target user i_0 by computing the Euclidean distance $d_{i_0,j} = \|\text{NN}_{\boldsymbol{\Theta}^*}(\bar{\mathcal{G}}_{i_0,\mathcal{T}_\mathcal{A}}^k) - \text{NN}_{\boldsymbol{\Theta}^*}(\bar{\mathcal{G}}_{j,\mathcal{W}_{\mathcal{T}'}}^k)\|_2$ between the profile of i_0 from target time period $\mathcal{T}_\mathcal{A}$ and the profiles of all the individuals $j \in \mathcal{I}_\mathcal{D}$ from reference time period $\mathcal{W}_{\mathcal{T}'}$. If the candidate with the smallest distance (or one of the R candidates with the smallest distance) is the target individual i_0, we say that the attacker has correctly identified the target.

9.3.2 Training setup

Mobile phone metadata dataset. For each possible value of $k \in \{1,2,3\}$, the attacker selects the best hyperparameters for our model using cross validation on the periods / weeks $\mathcal{W}_{1:T'}$ that build our training set. Each test fold is composed of two consecutive weeks: the first week of the test fold is used as reference week, while the auxiliary data about target individuals comes from the second week. With T' being odd, the $(T'-1)/2$ disjoint test folds are defined as $\{(\mathcal{W}_{2i+1}, \mathcal{W}_{2i+2}), 0 \leq i < (T'-1)/2\}$. For each test fold, the previous two periods (modulo $T'-1$) are used as validation weeks for early stopping, while the remaining weeks are used for training. Given the best hyperparameter set, the attacker trains the model on data from $\mathcal{W}_{1:T'-3}$, using validation weeks $(\mathcal{W}_{T'-2}, \mathcal{W}_{T'-1})$ for early stopping. The probability of identification within rank 1 (i.e. the percentage of identified individuals), herein named p_k, is the metric used for early stopping in our experiments.

Model parameters of our architecture are optimized with stochastic gradient descent with a mini-batch size of 64 and a weight decay coefficient of 10^{-3}. We use a margin value of $\lambda = 0.25$ for implementing the triplet loss. For a given mini-batch, triplets are built as follows:

1. one week t is sampled uniformly at random among the training weeks;

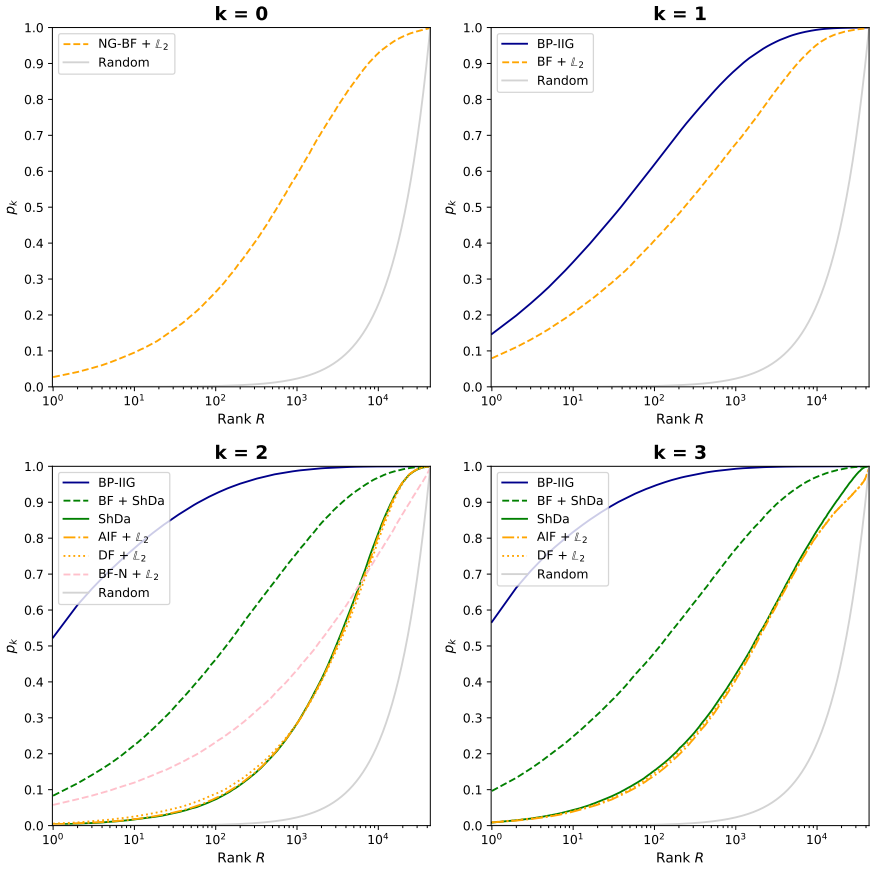


Figure 9.3. p_k, the probability of identification within rank R when the time gap is one week, $R \in \{1, \ldots, N\}$. For each $k \in \{1, 2, 3\}$, our method outperforms all the other approaches.

2. B individuals are sampled from $\mathcal{I}_{\mathcal{D}}$ and their simplified k-IIGs in week t are used as anchor examples, while their k-IIGs in weeks $t-1$ and $t+1$ (modulo the number of training weeks) are used as positive examples (2 positive samples per anchor are thus constructed at this step);

3. for each anchor selected at the previous step, a negative example is selected via mini-batch hard negative mining [224]. The k-IIG in the mini-batch with an embedding that is the closest to the one of the target anchor, but that is associated to a different individual, it's the one used for training.

We train for a maximum number of 100 epochs, decreasing the learning rate by a factor of 2 after 5 epochs of non-increasing probability of identification p_k computed over the validation weeks (an epoch is defined as a full pass over at least one anchor example of each individual in the training set). We stop training if the learning rate decreases below 10^{-5}. ReLU was the non-linearity of choice for our approach. The architecture configurations we obtained for $k \in \{1, 2, 3\}$ as a result of our grid search are detailed below:

Table 9.1. Examples of attention weight vectors for various intervals of the normalized entropy. The examples are sampled uniformly at random from the given interval for the first propagation step ($l = 1$). In each example, the weights are sorted decreasingly. In all cases, one or two neighbors have an attention weight at least twice as large as the lowest attention weight.

Interval	Normalized entropy	Attention weight vector
$[0.85, 0.90)$	0.8990	$[0.26, 0.24, 0.19, 0.17, 0.06, 0.04, 0.04]$
$[0.90, 0.95)$	0.9495	$[0.26, 0.19, 0.11, 0.10, 0.09, 0.09, 0.08, 0.08]$
$[0.95, 1.00]$	0.9827	$[0.24, 0.14, 0.13, 0.13, 0.12, 0.12, 0.12]$

- For $k = 1$, we use a MLP with input size $F = 23$, a hidden layer of size 128 and output size of 50.
- For $k \in \{2, 3\}$, we use $L = 2$ convolutional layers. $\text{MLP}^{(1)}$ has an input size of 46, one hidden layer of size 128 and an output size of 50. $\text{MLP}^{(2)}$ has an input size of 100, one hidden layer of size 128 and an output size of 50. For what concerns the parameters $\mathbf{c}_a^{(l)}$ and $\mathbf{W}_a^{(l)}$ ($l \in \{1, 2\}$) used to compute the attention weights, we use $F' = 20$ features for dimensionality reduction.

Bluetooth close-proximity dataset. Due to the small size of the available dataset, only $k = 1$ IIGs were considered in our experiments with Bluetooth data. A MLP, with one hidden layer of size 64 and output size of 25, was used for identification in this scenario. The remaining architectural and training details are the same described above for the mobile phone dataset.

9.4 Results

9.4.1 Mobile phone metadata dataset

The mobile phone interaction dataset we used in our experiments is composed of the 3-IIGs of $N = 43,606$ subscribers of a mobile carrier collected over a period of $T = 35$ consecutive weeks ($\mathcal{T} = \mathcal{W}_1 \cup \ldots \cup \mathcal{W}_T = \mathcal{W}_{1:T}$). We here consider the auxiliary profiling information available to the attacker to be the k-IIG of the target individual from a week $\mathcal{T}_\mathcal{A} \in \{\mathcal{W}_{T'+1}, \ldots, \mathcal{W}_T\}$ and the anonymous dataset to be the k-IIGs of all the N people from the first $T' = 15$ weeks of data ($\mathcal{T}_\mathcal{D} = \mathcal{W}_{1:T'}$). Unless specified otherwise, interactions from week $\mathcal{W}_{T'+1}$ are the ones used as auxiliary data. We report the probability of identification within rank R, defined as the fraction of people among the N subscribers who are correctly identified by one candidate in the first R positions (averaged over 10 runs).

Baselines. We compare our method with what is, to the best of our knowledge, the only attack designed for mobile call k-hop graphs that was proposed before our own approach [225]. The method (herein denoted as *ShDa*) uses a random forest binary classifier trained on hand-engineered node pair features to predict whether a pair of users represent the same individual or not. For a given node i, a feature vector $\mathbf{h}_i^{(1)} \in \mathbb{R}^{F_H}$ corresponding with the histogram of the

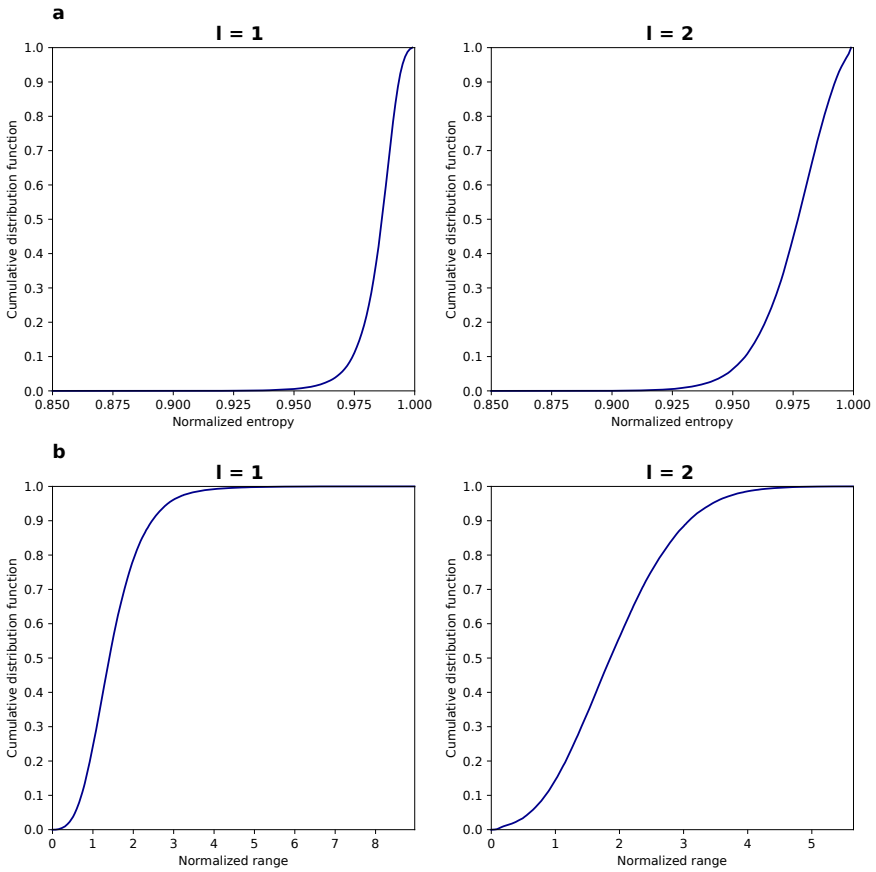


Figure 9.4. Cumulative distribution functions of the normalized entropy (a) and range (b) of the attention weights used to aggregate the features available on neighbors of each originating individual in the dataset. Two propagation layers are used in our model, each column showing the corresponding distributions.

degrees of i's 1-hop contacts is computed at first. For a pair of nodes (i, j), a number of F_H^2 features are then constructed from the available histograms and fed in input to the classifier. Each feature assumes a value equal to $\frac{|\mathbf{h}_{i,f}^{(1)} - \mathbf{h}_{j,f'}^{(1)}|}{\max(\mathbf{h}_{i,f}^{(1)}, \mathbf{h}_{j,f'}^{(1)}, 1)}$ for $1 \leq f \leq F_H$ and $1 \leq f' \leq F_H$. In our experiments, we used the out-degree (number of contacts) of a node's neighbors in the k-IIG for computing the feature vectors (this provided an indication on the behavior of the neighboring subscribers of the target node as well as the amount of non-subscribers available in the k-IIG), and $F_H = 21$ bins to compute our histograms. For a fair comparison with our approach, when the degrees of 2-hop neighbors are available (i.e. for a 3-IIG), we computed an additional feature vector $\mathbf{h}_i^{(2)}$ (consisting of the histogram of the degrees of the node's 2-hop neighbors), and we concatenated this to $\mathbf{h}_i^{(1)}$ before computing a pair descriptor. As our GCNN-based approach exploits behavioral features which were not considered in [225], to provide a comparison with an alternative learnable (but non-message passing) solution that relies on the same features used by BP-IIG, we additionally introduced in our experiments an

evolution of ShDA (herein referred to as *ShDa +BF*), which exploits our behavioral embeddings for computation. Provided a pair of nodes (i, j), we concatenated to the pair descriptor defined in [225] the vector: $(|b_{i,f} - b_{j,f}|)$ defined for $1 \leq f \leq F$. Here, $\mathbf{b}_i$ and $\mathbf{b}_j$ are the nodes' tuples of bandicoot features. Hyperparameter tuning was done for ShDA and ShDA+BF similarly to our GCNN-based approach.

On top of the method presented in [225], to provide reference performance for simpler (and more easily manageable) solutions, seven additional non-learning-based approaches have been considered in our comparison:

- Two baselines using the histogram of degrees $\mathbf{h}_i^{(1)}$ for $k = 2$, and $[\mathbf{h}_i^{(1)}, \mathbf{h}_i^{(2)}]$ for $k = 3$, with $\mathbb{L}_2$ distance for matching (denoted as *Degree Features (DF)* $+ \mathbb{L}_2$).
- Two baselines using the histogram of degrees computed for $k = 2$ and $k = 3$ from the number of interactions instead of the number of contacts, and $\mathbb{L}_2$ distance for matching (denoted *All Interaction Features (AIF)* $+ \mathbb{L}_2$).
- Three baselines using our behavioral features with $\mathbb{L}_2$ distance for matching:
  - *Non-Graph based Behavioural Features* (*NG-BF* $+ \mathbb{L}_2$), a baseline that works in a non graph-based scenario ($k = 0$) where a node's interaction list is available to the attacker, but the contacts' identities are not (an attacker knows for instance in this scenario how many calls a user has made but not who has called). *NG-BF* $+ \mathbb{L}_2$ uses only the behavioral features that do not exploit the graph information (i.e. features 2, 4, 6, 9-13, and 18-19 in Table 9.3).
  - *Behavioural Features* (*BF* $+ \mathbb{L}_2$), a baseline that uses all our behavioral features for the target node (to be compared with our BP-IIG, for $k = 1$).
  - *Neighborhood Behavioural Features* (*BF-N* $+ \mathbb{L}_2$), a baseline that concatenates the behavioral features of a target node to those of its top 5 neighbors (valid only for $k = 2$ as the neighbors interactions are needed as well). The neighbors are ordered decreasingly by the number of interactions, with tie-breaks decided by the total call duration.

Performance comparison. Figure 9.3 shows that our method ($k \in \{1, 2, 3\}$) vastly outperforms all the other approaches in the identification scenario. When $k = 2$ and 3, all baselines using only the graph structure (i.e. ShDa, AIF + $\mathbb{L}_2$ and DF + $\mathbb{L}_2$) perform very poorly, with a probability of identification ($p_{k=2}$ or $p_{k=3}$) within rank 1 of less than 1%. When we augment the random forest method with the bandicoot behavioral features, its performance increases to 8.3% for $k = 2$ and 9.6% for $k = 3$, suggesting that the behavioral features help in our prediction task. When $k = 1$, the BF + $\mathbb{L}_2$ baseline achieves a probability of identification ($p_{k=1}$) equal to 7.9%, which in turn outperforms the non-graph behavioral features-based approach (NG-BF + $\mathbb{L}_2$, $k = 0$), which only achieves 2.7%. On the other hand, BP-IIG correctly identifies people 14.7% of the times when $k = 1$, 52.4% of the times when $k = 2$, and 56.7% when $k = 3$. Higher ranks probabilities show similar behavior for different values of k. For $k = 1$, the probability of identifying the correct person among the top 10 candidates (rank 10) is 34.7%, while the rank 100 probability is 61.9% respectively. For $k = 2$, our model is able to rank the correct person among the top 10 candidates 77.2% of the time and among the top 100 candidates 92.4% of the time. For $k = 3$, probability of identification is 81.7% and 94.6% , for the top 10 and top 100 candidates respectively.

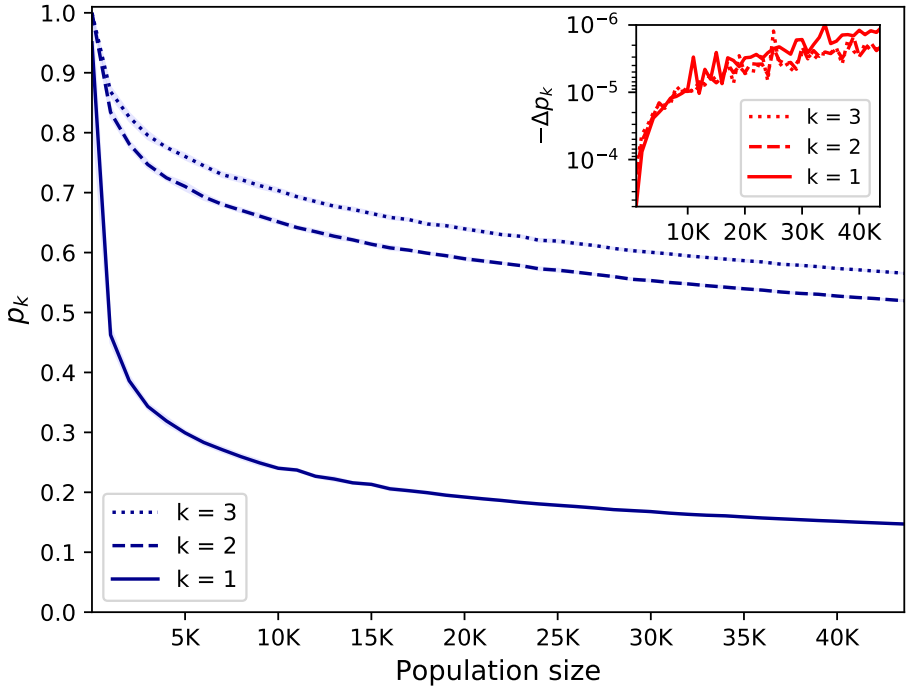


Figure 9.5. Attack's performance with increasing population size. We show $p_k(N')$, the probability of identification within rank 1 for $k \in \{1,2,3\}$ in a population of size N'. The 95% confidence interval is shown in light blue. (Inset) shows the negative difference quotient $-\Delta p_k(N') = -(p_k(N') - p_k(N' - \Delta(N))/\Delta N'$. The probability of identification decreases with the population size N', but at increasingly slower rates.

Other GCNNs. As an alternative to our attention-based GCNN, we evaluated in early experiments the performance that a Message Passing Neural Network (MPNN) [99] (implemented with summation for message aggregation) could achieve on our identification task. For MPNN, the message between a node and a neighbor is computed by applying a linear layer to the concatenation of their features followed by a ReLU non-linearity. After tuning, MPNN achieves comparable performance to GAT (for $k = 2$, $p_{k=2}^{(\text{GAT})} = 53.5\%$ vs $p_{k=2}^{(\text{MPNN})} = 53.0\%$ on the validation set, p-value: 0.11, 95% confidence intervals: [52.6, 53.4] for MPNN and [53.0, 53.9] for GAT using 10 runs). As GAT provides more easily interpretable filters than MPNN (i.e. the relevance of a neighbor can be determined by simply inspecting its attention weight), we decided to stick with this approach in our exploration.

Analysis of attention weights We perform an analysis of the attention weights to gain insights into our GCNN's capacity to assign different weights to different neighbors of a node in the aggregation step, and thus learn their relative importance. Figure 9.4(a) shows the cu-

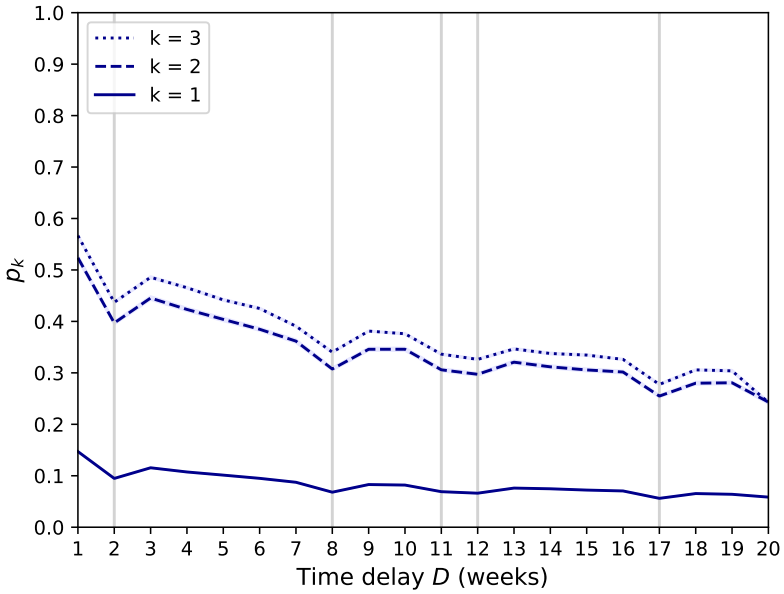


Figure 9.6. Probability of identification when the time delay increases. We plot p_k, the probability of identification within rank 1 for $k \in \{1, 2, 3\}$ when the time delay between the dataset and the attacker's auxiliary information is equal to D weeks. The auxiliary information is one week long. The 95% confidence interval is shown in light blue. The vertical grey lines correspond to holidays.

mulative distribution of the normalized entropy[4] of the attention weight vectors for both our convolutional layers. Table 9.1 shows instead three random samples for various entropy intervals. While entropy values tend to be high for our approach, there appears to be sufficient and meaningful variation in the attention weights, suggesting that the model is able to capture the relative importance of neighbors. This is further supported by Figure 9.4(b) which shows the cumulative distribution of the normalized range of the attention weights[5]. For both our layers, in more than $\sim 90\%$ of the cases, the normalized range is larger than 1, which means that the largest attention weight is at least twice as large as the smallest one(s). This further shows how attention weights help discriminate between the neighbors.

Increasing population size. The performance of a profiling attack like ours likely depends on the size of the population where one might want to identify a given target. To better understand how well the attack scales with the population size, we evaluated its performance when identifying $N' \leq N (N = 43,606)$ people among the same N' people. For this analysis, the final model trained on data from the N people was used. Ten random subsets of size N' were sampled uniformly without replacement and the average probability of identification $p_k(N')$ (defined as

[4] The normalized entropy of a vector $(\alpha_1, \ldots, \alpha_{|\mathcal{N}_i|})$ is defined as the ratio between the entropy $\sum_{1 \leq q \leq |\mathcal{N}_i|} -\alpha_q \ln(\alpha_q)$ of the vector and $\ln(|\mathcal{N}_i|)$ (i.e. the maximum entropy for a discrete probability distribution with $|\mathcal{N}_i|$ possible values).

[5] We define the normalized range of a vector $\boldsymbol{\alpha}$ as $(\max(\boldsymbol{\alpha}) - \min(\boldsymbol{\alpha}))/\min(\boldsymbol{\alpha})$.

the fraction of people among the N' that are correctly identified within rank 1) was computed. The probability of identification $p_k(N')$ was computed for $N' = 2$, for $N' \in [1,000; 43,000]$ using increments of $\Delta N = 1000$, and finally for $N' = N$. The difference quotient was computed as $\Delta p_k(N') = (p_k(N') - p_k(N' - \Delta N'))/\Delta N'$.

Figure 9.5 shows that the probability of identification decreases with the reference population size, but that $\Delta p_k(N')$ decreases fast as N' grows. The difference quotient seems to be still decreasing around $N' = N$, suggesting that the probability of identification would decrease at an even slower rate for larger values of N'.

Degradation through time. The accuracy of our behavioral model is likely to decrease as time passes: people change behavior, make new friends and lose contact with others. Figure 9.6 shows that, despite this, the probability of correct identification only slowly decreases with the time delay $D = t'_{\mathcal{A}} - t'_{\mathcal{D}}$. Even after 20 weeks, our model still correctly identifies people $p_{k=2} = 24.3\%$ of the time for $k = 2$. This suggests that the profiles our model extracts from the data capture key behavioral features of individuals. The probability of identification decreases similarly slowly with time for $k = 3$ and $k = 1$.

Interestingly, Figure 9.6 also shows that the probability of identification (p_k) visibly decreases when the time delay is of 8, 11, 12, and 17 weeks, respectively. In a post-hoc analysis, we found that they all correspond to weeks containing a national holiday. This further suggests that our model captures a person's routine weekly behavior, both weekdays and weekend, and consequently loses some accuracy when a user's behavior changes in response to external events.

More auxiliary data. We have so far assumed that the attacker has access to only a week of a target individual's data (i.e. their auxiliary information is the target individual's k-IIG from one week). In practice, an attacker might however have access to more weeks of data from an individual. In the D4D challenge[6] for instance, data were re-pseudonymized every two weeks [33]. To simply evaluate the extent to which more auxiliary data increase accuracy, we combine the predictions from growing sequences of target weeks used as auxiliary data. For $1 \leq L_{\mathcal{W}} \leq T - T'$ ($L_{\mathcal{W}}$ denotes the number of weeks in the auxiliary data or $\mathcal{T}_{\mathcal{A}}$), we combine the predictions from the $T'+1, \ldots, (T'+L_{\mathcal{W}})$-th target weeks using a majority vote: the candidate that was ranked first most of the times is the final prediction. The tie-breaks are decided by the lowest total Euclidean distance between the target individual embedding and the highest ranked candidates embeddings.

Figure 9.7 shows how having auxiliary data over several weeks further improves the performance of the attack. For $k = 2$, the probability of correct identification increases from $p_{k=2} = 52.4\%$ with one week of auxiliary data to $p_{k=2} = 66.0\%$ with $L_{\mathcal{W}} = 16$ weeks. Interestingly, the probability of correct identification for all values of k increases fast and then plateaus around $L_{\mathcal{W}} = 8$, even slightly decreasing after $L_{\mathcal{W}} = 16$ and $L_{\mathcal{W}} = 15$ for $k = 2$ and $k = 3$, respectively. While this might seem surprising at first, we hypothesize this to be due to small changes to people's behavior over time. This makes auxiliary data that are more distant in time less useful than closer one and sometimes slightly detrimental. The maximum probability

[6] Orange's "Data for Development" challenge is an open data challenge consisting of four mobile phone datasets (one of these describing a pseudonymized mobile interaction network of 5,000 customers in the Ivory Coast), which was released with the intent of fostering scientific research and collaborations with African scientists.

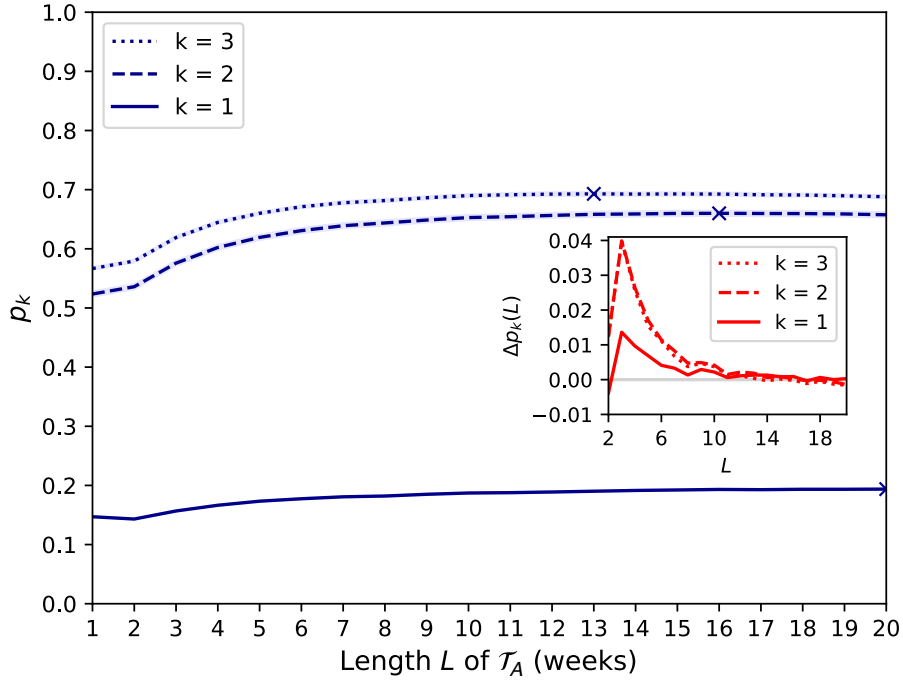


Figure 9.7. Probability of identification for increasing time period length of auxiliary data. For each $k \in \{1,2,3\}$, we plot p_k, the probability of correct identification ($R = 1$) when the attacker's auxiliary data $\mathcal{T}_A$ consist of $L_\mathcal{W}$ weeks, $1 \leq L_\mathcal{W} \leq 20$ (the largest value for each k is marked with an 'x' for each model). The 95% confidence interval is shown in light blue. (Inset) shows the difference quotient $\Delta p_k(L_\mathcal{W}) = p_k(L_\mathcal{W}) - p_k(L_\mathcal{W} - 1)$ for $2 \leq L \leq 20$.

for $k = 1$ is at $L_\mathcal{W} = 20$ weeks ($p_{k=1} = 19.4\%$), for $k = 2$ at $L_\mathcal{W} = 16$ weeks ($p_{k=2} = 66.0\%$), and for $k = 3$ at $L_\mathcal{W} = 13$ weeks ($p_{k=3} = 69.3\%$).

Generalization over unseen users and periods of time. In our last set of experiments, we validated that our attack generalizes by examining its performances when testing is performed on a set disjoint from the training set in the identities of the individuals, time periods used, or both. We use a dataset composed of weeks 1 to $T' + 1 = 16$ and all k-IIGs to design the following three different scenarios (Figure 9.8):

1. the testing, validation and training sets are disjoint in the time periods but not in the identities of the originating individuals of the k-IIGs (Figure 9.8(a) and 9.8(b)). This is the scenario considered in the previous paragraphs, with the data split as in Figure 9.8(a).

2. the testing, validation and training sets are disjoint in the identities of the originating individuals of the k-IIGs but not in time (Figure 9.8(c)). This scenario can be a transductive problem (dependently on the data used for testing) as some people's features used at inference time might have been seen during training. For example, Alice, the originating individual for a k-IIG in the training set, could be a neighbor of Bob, the originating individual for a k-IIG in the test set.

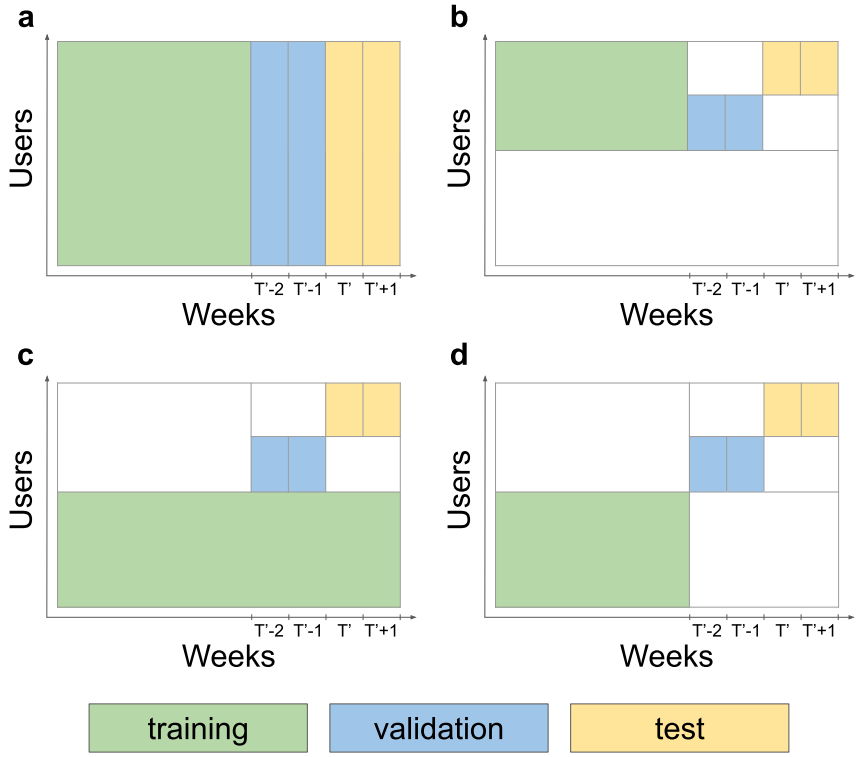


Figure 9.8. The various evaluation scenarios. Testing, validation and training are performed on sets disjoint in the time periods (a and b), the identities of the originating individuals of the k-IIGs (c) or the time periods and the identities of the originating individuals of the k-IIGs (d). The green, blue and yellow dataset parts are used for training, validation and testing, respectively. In the validation and test parts, the first week is used as reference week and the second one as target week.

3. the testing, validation and training sets are disjoint in the time periods, and in the identities of the originating individual of the k-IIGs (Figure 9.8(d)).

For a fair comparison of the three different configurations, the first scenario (Figure 9.8(b)) uses the first $T'-3$ weeks of data from half the k-IIGs for training, while the other two scenarios are trained on the other half of the users and all weeks (Figure 9.8(c)), or weeks 1 to $T'-3$ (Figure 9.8(d)), respectively. This ensures that the validation and test sets are always the same in our analysis. Table 9.2 shows that our attack is robust, and performs similarly across all the three scenarios.

9.4.2 Bluetooth close-proximity dataset

In the Bluetooth close-proximity dataset [223], only 4 weeks $\mathcal{T} = \mathcal{W}_1 \cup \ldots \cup \mathcal{W}_4 := \mathcal{W}_{1:4}$ are available for experimentation. The attacker uses the first two weeks of data for training, the second and third week of data for validation, and results are reported on the third and fourth

Table 9.2. The probability of identification p_k within rank 1 computed for individuals in the test set when compared with users from the reference week, when the time delay is one week for the three scenarios comparison, $k \in \{1, 2, 3\}$. By design, the test set is common across the three scenarios.

	Split by week (%)	Split by individuals (%)	Split by individuals and weeks (%)
$k = 1$	22.8	23.1	23.0
$k = 2$	61.5	62.2	60.5
$k = 3$	66.5	66.8	66.6

week of data (i.e. $T' = \mathcal{W}_3$ and $\mathcal{T}_{\mathcal{A}} = \mathcal{W}_4$). To increase the number of training samples per individual and limit the chances of overfitting, the attacker generates 8 overlapping weeks of data from the considered two training weeks. Because the training data contain a total of 14 days of interactions $d_1 \cup \ldots \cup d_{14}$, the attacker generates 8 overlapping weeks $\mathcal{W}'_1, \ldots, \mathcal{W}'_8$, with $\mathcal{W}'_i = d_i \cup \ldots \cup d_{i+6}$, $1 \leq i \leq 8$. As it was the case for mobile phone interactions, we report the probability of identification within rank R as a measure of performance, defined as the fraction of people among the N subscribers who are correctly identified by one candidate in the first R positions (averaged over 10 runs). Figure 9.9 shows that for $k = 1$ our approach is able to identify target individuals $p_{k=1} = 26.4\%$ of the time for $R = 1$, and $p_{k=1} = 60.1\%$ of the time for $R = 10$. These results provide evidence that our attack is general, and can deal with human-human interaction data that go beyond mobile phone interactions.

9.5 Discussion

Commentary. Our results provide evidence of the urgent need to consider profiling attacks when evaluating whether systems, protocols, or datasets satisfy the Article 29 WP's definition of anonymization [8]. In particular, they show how people's interaction patterns remain identifiable across long periods of time allowing an attacker to link together data coming from disjoint time periods with high accuracy even in large datasets. From a methodological perspective, in our study we evaluated the effectiveness of our method using only a MLP, a Graph Attention Network and a Message Passing Neural Network. Likely, even better performance could be achieved extending our grid search to a variety of different GDL architectures (for $k > 1$), which in turn would make an attack like ours to pose an even stronger threat.

For what concerns possible defenses against our attack, one may consider to add noise to the data to be released to mitigate the risk of identification (e.g. dropping interactions among users, adding fake interactions, changing features values, ...). However, unless the amount of noise introduced is significant (which likely makes any result derived from such a dataset meaningless), this would not necessarily protect well against future attacks [194, 249]. On the other hand, access control mechanisms and approaches that provide provable privacy guarantees (e.g. differentially private question-and-answer systems [129]) appear today as valuable solutions to enable data analysis on pseudonymized interaction data, while limiting the risk of identification [71].

Related works. Our approach for user identification in datasets of interaction networks is the most recent work listed in this manuscript (it was indeed published in Nature Communications

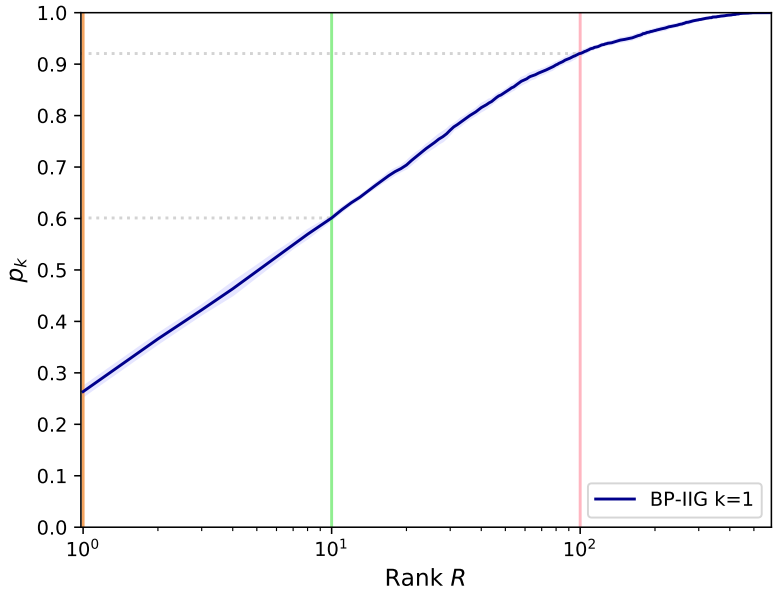


Figure 9.9. Probability of identification in a bluetooth close proximity network. We plot $p_{k=1}$, the probability of identification within rank R for $k = 1$. The 95% confidence interval is shown in light blue for BP-IIG. Our method correctly identifies people $p_{k=1} = 26.4\%$ of the time based on their 1-IIGs. Out of 10 people ($R = 10$), it is able to identify the right person $p_{k=1} = 60.1\%$ of the time.

in 2022). While, to the best of our knowledge, there are no other papers following our own that use GCNNs for introducing new possible attacks for interaction data[7], our work is cited in various recent publications, which stress the danger that attacks like ours pose for user identification [1, 170, 161, 205, 68]. As the primary goal of our paper was to raise awareness in the community on the risk posed by profiling attacks and the identifiability of interaction data, it appears that our efforts are actually having the positive effect we were hoping to achieve in the very first place.

[7] It should be noted that a non-GCNN based profiling attack, similar to ours, has been recently introduced in [67].

Table 9.3. The list of the 23 features used for the mobile phone interaction data. The bandicoot toolbox methods for computing the features are provided. A dash symbol (-) indicates that the feature was not computed using bandicoot.

	Feature description	Bandicoot method
1	Percentage of call contacts that account for 80% of the total call duration	`percent_pareto_durations`
2	Percentage of initiated calls	`percent_initiated_interactions`
3	Entropy of call contacts	`entropy_of_contacts`
4	Percentage of calls that occurred between 7PM and 7AM	`percent_nocturnal`
5	Percentage of contacts that account for 80% of the user's calls	`percent_pareto_interactions`
6	Number of outgoing calls	`number_of_interactions`
7	Number of call contacts	`number_of_contacts`
8	Number of text contacts	`number_of_contacts`
9	Number of incoming calls	`number_of_interactions`
10	Number of incoming texts	`number_of_interactions`
11	Number of days, between 1 and 7, when the user was active	`active_days`
12-13	Mean/standard deviation of the time between two consecutive calls recorded for the user	`intervent_time`
14-15	Mean/standard deviation of the balance of the user's call contacts, where the balance is the ratio between the number of outgoing calls with a contact and the total number of calls	`balance_of_contacts`
16-17	Mean/standard deviation of the number of calls with each contact	`interactions_per_contacts`
18-19	Mean/standard deviation of call durations	`call_duration`
20	Percentage of out-of-network calls	-
21	Percentage of out-of-network contacts	-
22	Percentage of out-of-network call duration	-
23	Percentage of out-of-network texts	-

Table 9.4. The list of the 16 features used for the Bluetooth close-proximity interaction data. A dash symbol (-) indicates that the feature was not computed using bandicoot.

	Feature description	Bandicoot method
1	Number of days, between 1 and 7, when the user was active	`active_days`
2	Number of contacts	`number_of_contacts`
3-4	Mean/standard deviation of the negative RSSI of the user's interactions	`call_duration`
5	Percentage of interactions that occurred between 7PM and 7AM	`percent_nocturnal`
6	Entropy of contacts	`entropy_of_contacts`
7-8	Mean/standard deviation of the balance of the user's contacts, where the balance is the ratio between the number of outgoing interactions with a contact and the total number of interactions.	`balance_of_contacts`
9-10	Mean/standard deviation of the time between two consecutive interactions recorded for the user	`interevent_time`
11-12	Mean/standard deviation of the number of interactions with each contact	`interactions_per_contacts`
13	Percent of contacts that account for 80% of the user's interactions	`percent_pareto_interactions`
14	Number of interactions	`number_of_interactions`
15	Number of empty scans (i.e. where the other node's ID is -1)	-
16	Percentage of out-of-network calls (i.e. where the other node's ID is -2)	-

Chapter 10

Conclusions and future works

In this thesis we introduced several different contributions to the Geometric Deep Learning field, involving both new methodologies and applications of Graph Convolutional Neural Networks.

In the first part of the manuscript, we discussed possible generalizations of convolution for graphs structured data (with our MoNet approach being able to deal with manifolds as well). In Chapter 3, we discussed how filters for graph-/manifold-structured data can be implemented resorting to an attention mechanism. In Chapter 4, we introduced MGCNN, a generalization of [72] for signals defined over multiple graphs. In the same chapter, we also dicussed how multi-graph GCNNs can be used for solving matrix completion tasks, and we introduced an architecture (RMGCNN) able to address this problem via a learnable diffusion process. In Chapter 5, we introduced CayleyNet, a spectral GCNN that enjoys spectral zoom properties. Thanks to their ability to learn filters that specialize on a particular frequency band, Cayleynets outperformed previously presented approaches on community detection, node classification and matrix completion problems. Finally, in Chapter 6 we introduced SIGN, a simple yet scalable GCNN that by moving the application of a given set of shift operators at pre-processing time, it is able to achieve competitive performance to prior art on very large graphs (in the order of even millions of nodes and billions of edges), while showing the same computational complexity of a MLP.

In the second part of the manuscript, we investigated instead possible applications that GCNNs can have in the realms of High Energy Physics, Social Network Data Analysis, and Data Privacy. In Chapter 7, we showed how GCNNs can be used to detect high-energy neutrinos by classifying signals retrived by the IceCube detector. Our approach outperformed in our analysis the performance achieved by both a physics-inspired baseline, and by classic CNNs. In Chapter 8, we discussed Fake News detection, and we showed in particular how GCNNs can be used to detect misinformation by classifying cascades of tweets / retweets that form on Twitter / X's social network when news spread. The method we dicussed in Chapter 8 corresponds to the main technology that was at the base of our start-up, Fabula AI, which was acquired by Twitter in 2019. Finally, in Chapter 9 we showed how GCNNs can be used for user identification in datasets of pseudonymized interaction networks. Our results highlight how people's interaction patterns are identifiable even over long periods of time, and how profiling attacks (like ours) should be taken into consideration when evaluating whether a given dataset or system satisfies the anonymization guidelines set forth in GDPR. Taken all together, the results we presented in the second part of this thesis (together with what discussed in Chapter 4) provide a glimpse of the wide applicability that GCNNs can have in a variety of different fields.

10.1 Future works

For what concerns areas of future research, there are two main directions that naturally emerge from our own work.

New methodologies. The first direction concerns the design and analysis of increasingly more effective approaches for processing graph-structured data. In this general area, there are in particular two recent lines of work that we would like to point out, as we believe can be a source of inspiration for future research. The first one considers the design of *provably more expressive GCNNs*. As it was shown by Morris et al. [191] and Xu et al. [266], message-passing neural networks have an expressivity that is bound by the 1-Weisfeiler Lehman (WL) isomorphism test, and are thus limited in the class of functions they can implement on a provided graph (e.g. MPNNs cannot count the number of induced subgraphs that are isomorphic to a pattern of size equal to 3 or higher [55]). As a result of this, many researchers devoted their efforts in the last years to develop new architectures that, by enriching the way in which message passing is realized, are able to achieve an expressivity that goes beyond the 1-WL test. Despite the literature on this topic is extensive (please refer to [278, 155] for some recent review papers), there are still challenges that need to be addressed in this area [192]. One research direction that we find especially interesting concerns the design of *efficient* expressive GCNNs. GCNNs with an expressive power beyond the 1-WL test generally come with a high computational cost compared to MPNNs (e.g. k-GNN [191], a GCNN with an expressive power matching the one of a k-WL test, requires a $\mathcal{O}(|\mathcal{V}|^{k+1})$ operations per layer), which hinders their applicability on sufficiently large domains. As of today, it is unclear however whether the GCNNs we have with a given level of expressivity are as efficient as possible, and more scalable designs could be feasible, especially for specific families of graphs [192]. In some recent efforts, Dimitrov et al. [77] and Bause et al. [24] highlighted in this direction how maximally expressive GCNNs, which show the same computational complexity of MPNNs, can actually be realized if we restrict the domain of application to planar and outer-planar graphs, respectively[1]. We believe that further studies on the complexity of expressive GCNNs would be valuable for the community, as they could unlock the design of more scalable architectures (for the whole family of graphs, or for specific classes of interest), and ultimately improve the applicability of this class of approaches in the real-world. The second line of work pertains instead to *dynamic / temporal graphs* (i.e. graphs that show nodes, edges or attributes that evolve over time). In this direction, despite the vast majority of approaches for learning representation on graphs assume the input domain to be static, several examples of real-world relational data actually show some form of temporal behavior (e.g. users in social networks build and loose connections over time, new user-item interactions constantly form on e-commerce websites, ...). While in principle it is possible to apply approaches designed for static graphs on dynamic graphs by ignoring the temporal evolution of the network, this has been shown to be sub-optimal [265], as in many cases it is the dynamics of the graph itself which carries meaningful information for solving the considered prediction problem [219]. Recently, a variety of architectures (including our own TGN [219]) have been proposed in the literature, which introduce different ways for producing representations that capture the evolution of temporal networks [219, 265, 146, 258, 274]. However, if compared to solutions designed for "classic" static graphs, methodologies designed

[1] Please note, [77] requires a one-off pre-processing step with a complexity equal to a $O(|\mathcal{V}|^2)$ to achieve, on planar graphs, a maximally expressive linear time GCNN. [24] requires instead a $O(|\mathcal{V}|)$ amount of operations at pre-processing time, and it is thus entirely linear in complexity w.r.t. the size of the vertex set.

for temporal graphs are still comparably few, and many problems appear under-explored in the community (e.g. predicting the time of a future event, clustering nodes or groups of dynamic graphs, characterizing the expressive power of different models, explaining the predictions of a given architecture, ...) [160]. The design and analysis of techniques able to effectively process dynamic graphs represents today one of the fastest growing areas of research in the GDL movement, for recent reviews on the matter we refer the reader to [6, 160].

GCNNs in the real-world. The second direction of research pertains instead to applications of GCNNs on interesting real-world problems. When we started our journey into learning on graphs, the GCNNs available in the literature were very few and they were predominantly applied on synthetic datasets, or citation networks, to showcase the performance they were able to achieve. As highlighted at the beginning of this manuscript, today Geometric Deep Learning is one of the most active topics in major machine learning conferences, and GCNNs are used for a variety of relevant tasks that range from traffic prediction [74] to weather forecasting [148]. We believe that the methods discussed in Part II, (as well as our approach for matrix completion - Chapter 4 -, and our work on protein-protein interactions [93]) contributed to fostering the popularity that GCNNs are enjoying today in numerous disciplines. Moving forward, we find particularly fascinating the applications that GCNNs can have for natural sciences. Many examples of "natural data" directly show in this sense a graph structure (e.g. molecules, protein-protein interaction networks, signals retrieved by sensor networks, ...) and are thus amenable to be processed (or possibly generated [120, 173]) with graph-based approaches. In this direction, besides the applications in High Energy Physics that we discussed in Chapter 7, several works recently appeared in the literature that apply GCNNs in the realm of *Structural Biology*. In this area, one problem that has been receiving special attention is in particular the task of *drug discovery*. As the space of synthesisable small molecules is extremely large (estimated to be $\sim 10^{60}$), the search of effective and safe to use drugs cannot be done experimentally, and scalable cost-efficient solutions are required to limit the time and money required for identifying promising candidates [44]. Graph machine learning models have recently been playing an increasingly more important role for the screening of promising molecules, thanks to their ability of rapidly exploring vast molecular spaces in silico. In [237] GCNNs have been used for instance to predict whether a molecule inhibits the growth of bacterium *Escherichia coli*, in [158] GCNNs were used to discover an antibacterial compound (*abaucin*) that targets a pathogen that shows resistence to multiple drugs (*Acinetobacter baumannii*), and in [262] GCNNs were used to identify a new structural class of antibiotics. As a deep overview of applications of GCNNs in Structural Biology, High Energy Physics, and other fields is beyond the scope of this work, we refer the interested reader to [5, 75, 98] for further details on the subject.

Bibliography

[1] What anonymization techniques can you trust? https://desfontain.es/privacy/trustworthy-anonymization.html.

[2] Mathematical models of social systems. https://users.ssc.wisc.edu/~jmontgom/376textbook.htm.

[3] Britannica, the editors of encyclopaedia. "neutrino". https://www.britannica.com/science/neutrino, .

[4] Machine learning method improves reconstruction and classification of low-energy icecube events. https://icecube.wisc.edu/news/research/2022/11/machine-learning-method-improves-reconstruction-and-classification-of-low-energy-icecube-events/, .

[5] Graph geometric ml in 2024: Where we are and what's next (part ii - applications). https://towardsdatascience.com/graph-geometric-ml-in-2024-where-we-are-and-whats-next-part-ii-applications-1ed786f7bf63, .

[6] Temporal graph learning in 2024. https://towardsdatascience.com/temporal-graph-learning-in-2024-feaa9371b8e2, .

[7] Food discovery with uber eats: Using graph learning to power recommendations. https://www.uber.com/en-GB/blog/uber-eats-graph-learning/.

[8] Article 29 data protection working party. opinion 05/2014 on anonymisation techniques. https://ec.europa.eu/justice/article-29/documentation/opinion-recommendation/files/2014/wp216_en.pdf, 2014.

[9] General data protection regulation. https://gdpr-info.eu/, 2016.

[10] Wikipedia links, english network dataset – KONECT, April 2017. URL http://konect.uni-koblenz.de/networks/wikipedia_link_en.

[11] What are graph neural networks?, 2022. URL https://blogs.nvidia.com/blog/what-are-graph-neural-networks/.

[12] MG Aartsen, K Abraham, M Ackermann, J Adams, JA Aguilar, M Ahlers, M Ahrens, D Altmann, T Anderson, M Archinger, et al. Characterization of the atmospheric muon flux in icecube. *Astroparticle physics*, 78:1–27, 2016.

[13] R Abbasi, M Ackermann, J Adams, N Aggarwal, JA Aguilar, M Ahlers, M Ahrens, JM Alameddine, AA Alves, NM Amin, et al. Graph neural networks for low-energy event classification & reconstruction in icecube. *Journal of Instrumentation*, 17(11):P11003, 2022.

[14] Rasha Abbasi et al. The IceCube data acquisition system: Signal capture, digitization, and timestamping. *Nucl. Instrum. Meth. A*, 601:294–316, 2009.

[15] Rasha Abbasi et al. An improved method for measuring muon energy using the truncated mean of dE/dx. *Nucl. Instrum. Meth. A*, 703:190–198, 2013.

[16] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*, pages 21–29. PMLR, 2019.

[17] Sadia Afroz, Michael Brennan, and Rachel Greenstadt. Detecting hoaxes, frauds, and deception in writing style online. In *Proc. IEEE Symp. Security and Privacy (SP)*, pages 461–475, 2012.

[18] M. Ahlers, K. Helbing, and C. Pérez de los Heros. Probing particle physics with icecube. *arXiv:1806.05696*, 2018.

[19] J Ahrens, X Bai, R Bay, SW Barwick, T Becka, JK Becker, K-H Becker, E Bernardini, D Bertrand, A Biron, et al. Muon track reconstruction and data selection techniques in amanda. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 524(1-3):169–194, 2004.

[20] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*, 2020.

[21] Yaniv Altshuler, Nadav Aharony, Micky Fire, Yuval Elovici, and Alex Pentland. Incremental learning with accuracy prediction of social and individual properties from mobile-phone data. In *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Conferenece on Social Computing*, pages 969–974. IEEE, 2012.

[22] Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 475–486. IEEE, 2006.

[23] Pablo Barceló, Egor V Kostylev, Mikael Monet, Jorge Pérez, Juan Reutter, and Juan-Pablo Silva. The logical expressiveness of graph neural networks. In *8th International Conference on Learning Representations (ICLR 2020)*, 2020.

[24] Franka Bause, Fabian Jogl, Patrick Indri, Tamara Drucks, David Penz, Nils Kriege, Thomas Gärtner, Pascal Welke, and Maximilian Thiessen. Maximally expressive gnns for outerplanar graphs. In *NeurIPS 2023 Workshop: New Frontiers in Graph Learning*, 2023.

[25] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *JMLR*, 7:2399–2434, 2006.

[26] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, 2007.

[27] A. R. Benson, D. F. Gleich, and J. Leskovec. Higher-order organization of complex networks. *Science*, 353(6295):163–166, 2016.

[28] K. Benzi, V. Kalofolias, X. Bresson, and P. Vandergheynst. Song recommendation with non-negative matrix factorization and graph total variation. In *Proc. ICASSP*, 2016.

[29] Rianne van den Berg, Thomas N Kipf, and Max Welling. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.

[30] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2546–2554. Curran Associates, Inc., 2011. URL `http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf`.

[31] K. Bhatia, K. Dahiya, H. Jain, A. Mittal, Y. Prabhu, and M. Varma. The extreme classification repository: Multi-label datasets and code, 2016. URL `http://manikvarma.org/downloads/XC/XMLRepository.html`.

[32] 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*, 44(7):3496–3507, 2021.

[33] Vincent D Blondel, Markus Esch, Connie Chan, Fabrice Clérot, Pierre Deville, Etienne Huens, Frédéric Morlot, Zbigniew Smoreda, and Cezary Ziemlicki. Data for development: the d4d challenge on mobile phone data. Preprint at `https://arxiv.org/abs/1210.0137`, 2012.

[34] Joshua Blumenstock, Gabriel Cadamuro, and Robert On. Predicting poverty and wealth from mobile phone metadata. *Science*, 350(6264):1073–1076, 2015.

[35] Deyu Bo, Xiao Wang, Chuan Shi, and Huawei Shen. Beyond low-frequency information in graph convolutional networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3950–3957, 2021.

[36] Deyu Bo, Xiao Wang, Yang Liu, Yuan Fang, Yawen Li, and Chuan Shi. A survey on spectral graph neural networks. *arXiv preprint arXiv:2302.05631*, 2023.

[37] Federica Bogo, Javier Romero, Matthew Loper, and Michael J Black. FAUST: Dataset and evaluation for 3D mesh registration. In *Proc. CVPR*, 2014.

[38] D. Boscaini, J. Masci, E. Rodolà, and M. M. Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. In *Proc. NIPS*, 2016.

[39] Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Michael M Bronstein, and Daniel Cremers. Anisotropic diffusion descriptors. In *Computer Graphics Forum*, volume 35, pages 431–441. Wiley Online Library, 2016.

[40] Alexandre Bovet and Hernán A Makse. Influence of fake news in Twitter during the 2016 US presidential election. *Nature Communications*, 10(1):7, 2019.

[41] J. Breese, D. Heckerman, and C. Kadie. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Proc. Uncertainty in Artificial Intelligence*, 1998.

[42] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

[43] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[44] Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

[45] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[46] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. *arXiv:1312.6203*, 2013.

[47] Jose Caballero, Christian Ledig, Andrew Aitken, Alejandro Acosta, Johannes Totz, Zehan Wang, and Wenzhe Shi. Real-time video super-resolution with spatio-temporal networks and motion compensation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4778–4787, 2017.

[48] Vanessa Cai, Pradeep Prabakar, Manuel Serrano Rebuelta, Lucas Rosen, Federico Monti, Katarzyna Janocha, Tomo Lazovich, Jeetu Raj, Yedendra Shrinivasan, Hao Li, et al. Twerc: High performance ensembled candidate generation for ads recommendation at twitter. *29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD*, 2023.

[49] E. Candes and B. Recht. Exact matrix completion via convex optimization. *Comm. ACM*, 55(6):111–119, 2012.

[50] Carlos Castillo, Marcelo Mendoza, and Barbara Poblete. Information credibility on twitter. In *Proceedings of the 20th international conference on World wide web*, pages 675–684, 2011.

[51] Benjamin Paul Chamberlain, Sergey Shirobokov, Emanuele Rossi, Fabrizio Frasca, Thomas Markovich, Nils Hammerla, Michael M Bronstein, and Max Hansmire. Graph neural networks for link prediction with subgraph sketching. *ICLR*, 2023.

[52] Jianfei Chen and Jun Zhu. Stochastic training of graph convolutional networks, 2018.

[53] Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast learning with graph convolutional networks via importance sampling. In *ICLR*, 2018.

[54] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International conference on machine learning*, pages 1725–1735. PMLR, 2020.

[55] Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. Can graph neural networks count substructures? *Advances in neural information processing systems*, 33:10383–10395, 2020.

[56] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10, 2016.

[57] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *KDD*, 2019.

[58] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. *arXiv preprint arXiv:2006.07988*, 2020.

[59] Jiho Choi, Taewook Ko, Younhyuk Choi, Hyungho Byun, and Chong-kwon Kim. Dynamic graph convolutional networks with attention mechanism for rumor detection on social media. *Plos one*, 16(8):e0256039, 2021.

[60] Nicholas Choma, Federico Monti, Lisa Gerhardt, Tomasz Palczewski, Zahra Ronaghi, Prabhat Prabhat, Wahid Bhimji, Michael M Bronstein, Spencer R Klein, and Joan Bruna. Graph neural networks for icecube signal classification. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 386–391. IEEE, 2018.

[61] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

[62] Yunfei Chu, Xiaofu Chang, Kunyang Jia, Jingzhen Zhou, and Hongxia Yang. Dynamic sequential graph learning for click-through rate prediction. *arXiv preprint arXiv:2109.12541*, 2021.

[63] IceCube Collaboration et al. The icecube neutrino observatory iii: Cosmic rays. *arXiv preprint arXiv:1111.2735*, 2011.

[64] Michael Conover, Jacob Ratkiewicz, Matthew R Francisco, Bruno Gonçalves, Filippo Menczer, and Alessandro Flammini. Political polarization on twitter. In *Proc. ICWSM*, 2011.

[65] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198, 2016.

[66] Ana-Maria Creţu, Federico Monti, Stefano Marrone, Xiaowen Dong, Michael Bronstein, and Yves-Alexandre de Montjoye. Interaction data are identifiable even across long periods of time. *Nature Communications*, 13(1):313, 2022.

[67] Ana-Maria Creţu, Miruna Rusu, and Yves-Alexandre de Montjoye. Re-pseudonymization strategies for smart meter data are not robust to deep learning profiling attacks. In *In Proceedings of the Fourteenth ACM Conference on Data and Application Security and Privacy (CODASPY â24)*, 2024.

[68] Danielle Movsowitz Davidow, Yacov Manevich, and Eran Toch. Privacy-preserving transactions with verifiable local differential privacy. In *5th Conference on Advances in Financial Technologies*, 2023.

[69] Yves-Alexandre de Montjoye, César A Hidalgo, Michel Verleysen, and Vincent D Blondel. Unique in the crowd: The privacy bounds of human mobility. *Scientific reports*, 3:1376, 2013.

[70] Yves-Alexandre de Montjoye, Luc Rocher, and Alex Sandy Pentland. bandicoot: A python toolbox for mobile phone metadata. *The Journal of Machine Learning Research*, 17(1): 6100–6104, 2016.

[71] Yves-Alexandre De Montjoye, Sébastien Gambs, Vincent Blondel, Geoffrey Canright, Nicolas De Cordes, Sébastien Deletaille, Kenth Engø-Monsen, Manuel Garcia-Herranz, Jake Kendall, Cameron Kerry, et al. On the privacy-conscientious use of mobile phone data. *Scientific data*, 5(1):1–6, 2018.

[72] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proc. NIPS*, 2016.

[73] Zhijie Deng, Yinpeng Dong, and Jun Zhu. Batch virtual adversarial training for graph convolutional networks. *AI Open*, 4:73–79, 2023.

[74] Austin Derrow-Pinion, Jennifer She, David Wong, Oliver Lange, Todd Hester, Luis Perez, Marc Nunkesser, Seongjae Lee, Xueying Guo, Brett Wiltshire, et al. Eta prediction with graph neural networks in google maps. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 3767–3776, 2021.

[75] Gage DeZoort, Peter W Battaglia, Catherine Biscarat, and Jean-Roch Vlimant. Graph neural networks at the large hadron collider. *Nature Reviews Physics*, pages 1–23, 2023.

[76] 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.

[77] Radoslav Dimitrov, Zeyang Zhao, Ralph Abboud, and Ismail Ceylan. Plane: representation learning over planar graphs. *Advances in Neural Information Processing Systems*, 36: 16028–16054, 2023.

[78] Eid H Doha, Ali H Bhrawy, Dumitru Baleanu, Samer S Ezz-Eldien, and Ramy M Hafez. An efficient numerical scheme based on the shifted orthonormal jacobi polynomials for solving fractional optimal control problems. *Advances in Difference Equations*, 2015:1–17, 2015.

[79] Yingtong Dou, Zhiwei Liu, Li Sun, Yutong Deng, Hao Peng, and Philip S Yu. Enhancing graph neural network-based fraud detectors against camouflaged fraudsters. In *Proceedings of the 29th ACM international conference on information & knowledge management*, pages 315–324, 2020.

[80] Yingtong Dou, Kai Shu, Congying Xia, Philip S Yu, and Lichao Sun. User preference-aware fake news detection. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*, pages 2051–2055, 2021.

[81] G. Dror, N. Koenigstein, Y. Koren, and M Weimer. The Yahoo! music dataset and KDD-Cup'11. In *KDD Cup*, 2012.

[82] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

[83] Vijay Prakash Dwivedi, Ladislav Rampášek, Michael Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long range graph benchmark. *Advances in Neural Information Processing Systems*, 35:22326–22340, 2022.

[84] Moshe Eliasof, Lars Ruthotto, and Eran Treister. Improving graph neural networks with learnable propagation operators. In *International Conference on Machine Learning*, pages 9224–9245. PMLR, 2023.

[85] Hugo Farinha and Joao P Carvalho. Towards computational fact-checking: Is the information checkable? In *2018 IEEE international conference on fuzzy systems (fuzz-IEEE)*, pages 1–7. IEEE, 2018.

[86] Steven Farrell, Paolo Calafiura, Mayur Mudigonda, Dustin Anderson, Jean-Roch Vlimant, Stephan Zheng, Josh Bendavid, Maria Spiropulu, Giuseppe Cerati, Lindsey Gray, et al. Novel deep learning methods for track reconstruction. *arXiv preprint arXiv:1810.06111*, 2018.

[87] Bjarke Felbo, PSundsøy, 'Sandy' Alex Pentland, Sune Lehmann, and Yves-Alexandre de Montjoye. Modeling the temporal nature of human behavior for demographics prediction. *Machine Learning and Knowledge Discovery in Databases. ECML PKDD 2017. Lecture Notes in Computer Science, vol 10536.*, 2017.

[88] Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. Graph random neural networks for semi-supervised learning on graphs. *Advances in neural information processing systems*, 33:22092–22103, 2020.

[89] David J Field. What the statistics of natural images tell us about visual coding. In *Human Vision, Visual Processing, and Digital Display*, volume 1077, pages 269–276. SPIE, 1989.

[90] Fabrizio Frasca*, Emanuele Rossi*, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. *Graph Representation Learning and Beyond, ICML Workshop*, 2020.

[91] Thomas MJ Fruchterman and Edward M Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.

[92] Satoshi Furutani, Toshiki Shibahara, Mitsuaki Akiyama, Kunio Hato, and Masaki Aida. Graph signal processing for directed graphs based on the hermitian laplacian. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part I*, pages 447–463. Springer, 2020.

[93] Pablo Gainza, Freyr Sverrisson, Federico Monti, Emanuele Rodola, D Boscaini, Michael M Bronstein, and Bruno E Correia. Deciphering interaction fingerprints from protein molecular surfaces using geometric deep learning. *Nature Methods*, 17(2):184–192, 2020.

[94] Chen Gao, Yu Zheng, Nian Li, Yinfeng Li, Yingrong Qin, Jinghua Piao, Yuhan Quan, Jianxin Chang, Depeng Jin, Xiangnan He, et al. A survey of graph neural networks for recommender systems: Challenges, methods, and directions. *ACM Transactions on Recommender Systems*, 1(1):1–51, 2023.

[95] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *international conference on machine learning*, pages 2083–2092. PMLR, 2019.

[96] Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. Graphnas: Graph neural architecture search with reinforcement learning. *arXiv preprint arXiv:1904.09981*, 2019.

[97] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.

[98] Thomas Gaudelet, Ben Day, Arian R Jamasb, Jyothish Soman, Cristian Regep, Gertrude Liu, Jeremy BR Hayter, Richard Vickers, Charles Roberts, Jian Tang, et al. Utilizing graph machine learning within drug discovery and development. *Briefings in bioinformatics*, 22 (6):bbab159, 2021.

[99] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272, 2017.

[100] Theo Glauch. *The Origin of High-Energy Cosmic Particles: IceCube Neutrinos and the Blazar Case*. PhD thesis, Technische Universität München, 2021.

[101] Shuzhi Gong, Richard O Sinnott, Jianzhong Qi, and Cecile Paris. Fake news detection through graph-based neural networks: A survey. *arXiv preprint arXiv:2307.12639*, 2023.

[102] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[103] Emad M Grais and Mark D Plumbley. Single channel audio source separation using convolutional denoising autoencoders. In *2017 IEEE global conference on signal and information processing (GlobalSIP)*, pages 1265–1269. IEEE, 2017.

[104] Mark Granovetter. The strength of weak ties: A network theory revisited. In *Sociological Theory*, pages 105–130, 1982.

[105] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[106] Yuhe Guo and Zhewei Wei. Graph neural networks with learnable and optimal polynomial bases. In *International Conference on Machine Learning*, pages 12077–12097. PMLR, 2023.

[107] Saket Gurukar, Nikil Pancha, Andrew Zhai, Eric Kim, Samson Hu, Srinivasan Parthasarathy, Charles Rosenberg, and Jure Leskovec. Multibisage: A web-scale recommendation system using multiple bipartite graphs at pinterest. *arXiv preprint arXiv:2205.10666*, 2022.

[108] Francis Halzen and Spencer R Klein. IceCube: an instrument for neutrino astronomy. *Rev. Sci. Instrum.*, 81:081–101, 2010.

[109] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proc. NIPS*, 2017.

[110] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[111] Mingguo He, Zhewei Wei, Hongteng Xu, et al. Bernnet: Learning arbitrary graph spectral filters via bernstein approximation. *Advances in Neural Information Processing Systems*, 34:14239–14251, 2021.

[112] Mingguo He, Zhewei Wei, and Ji-Rong Wen. Convolutional neural networks on graphs with chebyshev approximation, revisited. *Advances in neural information processing systems*, 35:7264–7276, 2022.

[113] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 639–648, 2020.

[114] Yixuan He, Michael Perlmutter, Gesine Reinert, and Mihai Cucuringu. Msgnn: A spectral graph neural network based on a novel magnetic signed laplacian. In *Learning on Graphs Conference*, pages 40–1. PMLR, 2022.

[115] Zhenyu He, Ce Li, Fan Zhou, and Yi Yang. Rumor detection on social media with event augmentations. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*, pages 2020–2024, 2021.

[116] Dieter Heck, G Schatz, J Knapp, T Thouw, and JN Capdevielle. CORSIKA: A Monte Carlo code to simulate extensive air showers. Technical Report FZKA-6019, 1998.

[117] M. Henaff, J. Bruna, and Y. LeCun. Deep convolutional networks on graph-structured data. *arXiv:1506.05163*, 2015.

[118] Isaac Henrion, Johann Brehmer, Joan Bruna, Kyunghyun Cho, Kyle Cranmer, Gilles Louppe, and Gaspar Rochette. Neural message passing for jet physics. 2017.

[119] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[120] Emiel Hoogeboom, Vıctor Garcia Satorras, Clément Vignac, and Max Welling. Equivariant diffusion for molecule generation in 3d. In *International conference on machine learning*, pages 8867–8887. PMLR, 2022.

[121] Lee Howell et al. Digital wildfires in a hyperconnected world. *WEF Report*, 3:15–94, 2013.

[122] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *CoRR*, abs/2005.00687, 2020. URL `https://arxiv.org/abs/2005.00687`.

[123] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[124] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive sampling towards fast graph representation learning. In *NIPS*, 2018.

[125] 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.

[126] P. Jain and I. S. Dhillon. Provable inductive matrix completion. *arXiv:1306.0626*, 2013.

[127] Viren Jain and Sebastian Seung. Natural image denoising with convolutional networks. *Advances in neural information processing systems*, 21, 2008.

[128] M. Jamali and M. Ester. A matrix factorization technique with trust propagation for recommendation in social networks. In *Proc. Recommender Systems*, 2010.

[129] Noah Johnson, Joseph P Near, and Dawn Song. Towards practical differential privacy for sql queries. *Proceedings of the VLDB Endowment*, 11(5):526–539, 2018.

[130] Xiangyang Ju, Steven Farrell, Paolo Calafiura, Daniel Murnane, Lindsey Gray, Thomas Klijnsma, Kevin Pedro, Giuseppe Cerati, Jim Kowalkowski, Gabriel Perdue, et al. Graph neural networks for particle reconstruction in high energy physics detectors. *arXiv preprint arXiv:2003.11603*, 2020.

[131] V. Kalofolias, X. Bresson, M. M. Bronstein, and P. Vandergheynst. Matrix completion on graphs. *arXiv:1408.1717*, 2014.

[132] Kai Kang, Wanli Ouyang, Hongsheng Li, and Xiaogang Wang. Object detection from video tubelets with convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 817–825, 2016.

[133] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019.

[134] Henry Kenlay, Dorina Thanou, and Xiaowen Dong. On the stability of polynomial spectral graph filters. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5350–5354. IEEE, 2020.

[135] Vladimir G Kim, Yaron Lipman, and Thomas Funkhouser. Blended intrinsic maps. In *ACM Transactions on Graphics (TOG)*, volume 30, page 79. ACM, 2011.

[136] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proc. ICLR*, 2015.

[137] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[138] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. 2017.

[139] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Proc. NIPS*, 2017.

[140] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[141] Christian Koke, Abhishek Saroha, Yuesong Shen, Marvin Eisenberger, and Daniel Cremers. Resolvnet: A graph convolutional network with multi-scale consistency. *arXiv preprint arXiv:2310.00431*, 2023.

[142] I. Kokkinos, M. Bronstein, R. Litman, and A. Bronstein. Intrinsic shape context descriptors for deformable shapes. In *Proc. CVPR*, 2012.

[143] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[144] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[145] D. Kuang, Z. Shi, S. Osher, and A. Bertozzi. A harmonic extension approach for collaborative ranking. *arXiv:1602.05127*, 2016.

[146] Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1269–1278, 2019.

[147] Sejeong Kwon, Meeyoung Cha, Kyomin Jung, Wei Chen, and Yajun Wang. Prominent features of rumor propagation in online social media. In *Proc. Conf. Data Mining*, pages 1103–1108, 2013.

[148] Remi Lam, Alvaro Sanchez-Gonzalez, Matthew Willson, Peter Wirnsberger, Meire Fortunato, Ferran Alet, Suman Ravuri, Timo Ewalds, Zach Eaton-Rosen, Weihua Hu, et al. Learning skillful medium-range global weather forecasting. *Science*, 382(6677):1416–1421, 2023.

[149] David Lazer, Alex Pentland, Lada Adamic, Sinan Aral, Albert-László Barabási, Devon Brewer, Nicholas Christakis, Noshir Contractor, James Fowler, Myron Gutmann, Tony Jebara, Gary King, Michael Macy, Deb Roy, and Marshall Van Alstyne. Computational social science. *Science*, 323(5915):721–723, 2009. ISSN 0036-8075. doi: 10.1126/science.1167742.

[150] David MJ Lazer, Matthew A Baum, Yochai Benkler, Adam J Berinsky, Kelly M Greenhill, Filippo Menczer, Miriam J Metzger, Brendan Nyhan, Gordon Pennycook, David Rothschild, et al. The science of fake news. *Science*, 359(6380):1094–1096, 2018.

[151] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[152] Soo Yong Lee, Fanchen Bu, Jaemin Yoo, and Kijung Shin. Towards deep attention in graph neural networks: Problems and remedies. *Proceedings of the 40th International Conference on Machine Learning*, 2023.

[153] Ron Levie*, Federico Monti*, Xavier Bresson, and Michael M Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Transactions on Signal Processing*, 67(1):97–109, 2018.

[154] Ron Levie, Elvin Isufi, and Gitta Kutyniok. On the transferability of spectral graph filters. In *2019 13th International conference on Sampling Theory and Applications (SampTA)*, pages 1–5. IEEE, 2019.

[155] Pan Li and Jure Leskovec. The expressive power of graph neural networks. In Lingfei Wu, Peng Cui, Jian Pei, and Liang Zhao, editors, *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 63–98. Springer Singapore, Singapore, 2022.

[156] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[157] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[158] Gary Liu, Denise B Catacutan, Khushi Rathod, Kyle Swanson, Wengong Jin, Jody C Mohammed, Anush Chiappino-Pepe, Saad A Syed, Meghan Fragis, Kenneth Rachwalski, et al. Deep learning-guided discovery of an antibiotic targeting acinetobacter baumannii. *Nature Chemical Biology*, 19(11):1342–1350, 2023.

[159] Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 338–348, 2020.

[160] Antonio Longa, Veronica Lachi, Gabriele Santin, Monica Bianchini, Bruno Lepri, Pietro Lio, Franco Scarselli, and Andrea Passerini. Graph neural networks for temporal graphs: State of the art, open challenges, and opportunities. *arXiv preprint arXiv:2302.01018*, 2023.

[161] Antonio Longa, Giulia Cencetti, Sune Lehmann, Andrea Passerini, and Bruno Lepri. Generating fine-grained surrogate temporal networks. *Communications Physics*, 7(1):22, 2024.

[162] Shaojun Luo, Flaviano Morone, Carlos Sarraute, MatÃas Travizano, and HernÃan A. Makse. Inferring personal economic status from social network location. *Nature Communications*, 8(1), May 2017. ISSN 2041-1723.

[163] Guanghui Ma, Chunming Hu, Ling Ge, Junfan Chen, Hong Zhang, and Richong Zhang. Towards robust false information detection on social networks with contrastive learning. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 1441–1450, 2022.

[164] H. Ma, D. Zhou, C. Liu, M. Lyu, and I. King. Recommender systems with social regularization. In *Proc. Web Search and Data Mining*, 2011.

[165] Jiachen Ma, Yong Liu, Meng Liu, and Meng Han. Curriculum contrastive learning for fake news detection. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 4309–4313, 2022.

[166] Jiaqi Ma, Weijing Tang, Ji Zhu, and Qiaozhu Mei. A flexible generative framework for graph-based semi-supervised learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[167] Jing Ma, Wei Gao, and Kam-Fai Wong. Rumor detection on twitter with tree-structured recursive neural networks. Association for Computational Linguistics, 2018.

[168] Yi Ma, Jianye Hao, Yaodong Yang, Han Li, Junqi Jin, and Guangyong Chen. Spectral-based graph convolutional network for directed graphs. *arXiv preprint arXiv:1907.08990*, 2019.

[169] Craig Macartney and Tillman Weyde. Improved speech enhancement with the wave-u-net. *arXiv preprint arXiv:1811.11307*, 2018.

[170] Abdul Majeed, Safiullah Khan, and Seong Oun Hwang. A comprehensive analysis of privacy-preserving solutions developed for online social networks. *Electronics*, 11(13): 1931, 2022.

[171] Kelong Mao, Jieming Zhu, Xi Xiao, Biao Lu, Zhaowei Wang, and Xiuqiang He. Ultragcn: ultra simplification of graph convolutional networks for recommendation. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 1253–1262, 2021.

[172] J Arjona Martínez, Olmo Cerri, Maria Spiropulu, JR Vlimant, and M Pierini. Pileup mitigation at the large hadron collider with graph neural networks. *The European Physical Journal Plus*, 134(7):333, 2019.

[173] Karolis Martinkus, Jan Ludwiczak, WEI-CHING LIANG, Julien Lafrance-Vanasse, Isidro Hotzel, Arvind Rajpal, Yan Wu, Kyunghyun Cho, Richard Bonneau, Vladimir Gligorijevic, et al. Abdiffuser: full-atom generation of in-vitro functioning antibodies. *Advances in Neural Information Processing Systems*, 36, 2024.

[174] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *Proc. 3dRR*, 2015.

[175] Dominic Masters, Josef Dean, Kerstin Klaser, Zhiyi Li, Sam Maddrell-Mander, Adam Sanders, Hatem Helal, Deniz Beker, Ladislav Rampášek, and Dominique Beaini. Gps++: An optimised hybrid mpnn/transformer for molecular property prediction. *arXiv preprint arXiv:2212.02229*, 2022.

[176] Sunil Kumar Maurya, Xin Liu, and Tsuyoshi Murata. Improving graph neural networks with simple architecture design. *arXiv preprint arXiv:2105.07634*, 2021.

[177] Kevin McCloskey, Ankur Taly, Federico Monti, Michael P Brenner, and Lucy J Colwell. Using attribution to decode binding mechanism in neural network models for chemistry. *Proceedings of the National Academy of Sciences*, 116(24):11624–11629, 2019.

[178] Nikhil Mehta, María Leonor Pacheco, and Dan Goldwasser. Tackling fake news detection by continually improving social context representations using graph neural networks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1363–1380, 2022.

[179] Vinicius Mikuni and Florencia Canelli. Abcnet: An attention-based method for particle tagging. *The European Physical Journal Plus*, 135:1–11, 2020.

[180] B. N. Miller et al. MovieLens unplugged: experiences with an occasionally connected recommender system. In *Proc. Intelligent User Interfaces*, 2003.

[181] Erxue Min, Yu Rong, Yatao Bian, Tingyang Xu, Peilin Zhao, Junzhou Huang, and Sophia Ananiadou. Divide-and-conquer: Post-user interaction network for fake news detection on social media. In *Proceedings of the ACM web conference 2022*, pages 1148–1158, 2022.

[182] Martin Ha Minh. Reconstruction of neutrino events in icecube using graph neural networks. *arXiv preprint arXiv:2107.12187*, 2021.

[183] Cen Mo, Fuyudi Zhang, and Liang Li. Neutrino reconstruction in trident based on graph neural network. In *BenchCouncil International Symposium on Intelligent Computers, Algorithms, and Applications*, pages 264–271. Springer, 2023.

[184] 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 *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5115–5124, 2017.

[185] Federico Monti, Michael Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent multi-graph neural networks. *Advances in neural information processing systems*, 30, 2017.

[186] Federico Monti, Karl Otness, and Michael M Bronstein. Motifnet: a motif-based graph convolutional network for directed graphs. In *2018 IEEE Data Science Workshop (DSW)*, pages 225–228. IEEE, 2018.

[187] Federico Monti, Fabrizio Frasca, Davide Eynard, Damon Mannion, and Michael M Bronstein. Fake news detection on social media using geometric deep learning. *Representation Learning on Graphs and Manifolds workshop*, 2019.

[188] Federico Monti, Oleksandr Shchur, Aleksandar Bojchevski, Or Litany, Stephan Günnemann, and Michael M Bronstein. Dual-primal graph convolutional networks. *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2019.

[189] Eric A Moreno, Olmo Cerri, Javier M Duarte, Harvey B Newman, Thong Q Nguyen, Avikar Periwal, Maurizio Pierini, Aidana Serikova, Maria Spiropulu, and Jean-Roch Vlimant. Jedi-net: a jet identification algorithm based on interaction networks. *The European Physical Journal C*, 80:1–15, 2020.

[190] Eric A Moreno, Thong Q Nguyen, Jean-Roch Vlimant, Olmo Cerri, Harvey B Newman, Avikar Periwal, Maria Spiropulu, Javier M Duarte, and Maurizio Pierini. Interaction networks for the identification of boosted hâ b b decays. *Physical Review D*, 102(1): 012010, 2020.

[191] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4602–4609, 2019.

[192] Christopher Morris, Nadav Dym, Haggai Maron, İsmail İlkan Ceylan, Fabrizio Frasca, Ron Levie, Derek Lim, Michael Bronstein, Martin Grohe, and Stefanie Jegelka. Future directions in foundations of graph machine learning. *arXiv preprint arXiv:2402.02287*, 2024.

[193] Luis Müller, Mikhail Galkin, Christopher Morris, and Ladislav Rampášek. Attending to graph transformers. *arXiv preprint arXiv:2302.04181*, 2023.

[194] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *2008 IEEE Symposium on Security and Privacy*, pages 111–125, 2008.

[195] Van-Hoang Nguyen, Kazunari Sugiyama, Preslav Nakov, and Min-Yen Kan. Fang: Leveraging social context for fake news detection using graph representation. In *Proceedings of the 29th ACM international conference on information & knowledge management*, pages 1165–1174, 2020.

[196] Bruno A Olshausen and David J Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision research*, 37(23):3311–3325, 1997.

[197] Antonio Ortega, Pascal Frossard, Jelena Kovačević, José MF Moura, and Pierre Vandergheynst. Graph signal processing: Overview, challenges, and applications. *Proceedings of the IEEE*, 106(5):808–828, 2018.

[198] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bring order to the web. Technical report, Technical report, stanford University, 1998.

[199] T Palczewski for IceCube Collaboration. Icecube study of down-going neutrinos for the spectral cutoff determination. In *Proc. Neutrino*, 2018.

[200] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proc. EMNLP*, 2014.

[201] Verónica Pérez-Rosas, Bennett Kleinberg, Alexandra Lefevre, and Rada Mihalcea. Automatic detection of fake news. *arXivarXiv:1708.07104*, 2017.

[202] B. Perozzi, R. Al-Rfou, and S. Skiena. DeepWalk: Online learning of social representations. In *Proc. KDD*, 2014.

[203] Huyen Trang Phan, Ngoc Thanh Nguyen, and Dosam Hwang. Fake news detection: A survey of graph neural network methods. *Applied Soft Computing*, page 110235, 2023.

[204] Martin Potthast, Johannes Kiesel, Kevin Reinartz, Janek Bevendorff, and Benno Stein. A stylometric inquiry into hyperpartisan and fake news. *arXiv:1702.05638*, 2017.

[205] Jovan Powar and Alastair R Beresford. Sok: Managing risks of linkage attacks on data privacy. *Proceedings on Privacy Enhancing Technologies*, 2023.

[206] Luca Pretto. A theoretical analysis of google's pagerank. In *String Processing and Information Retrieval: 9th International Symposium, SPIRE 2002 Lisbon, Portugal, September 11–13, 2002 Proceedings 9*, pages 131–144. Springer, 2002.

[207] 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.

[208] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017.

[209] Huilin Qu and Loukas Gouskos. Jet tagging via particle clouds. *Physical Review D*, 101 (5):056019, 2020.

[210] Meng Qu, Yoshua Bengio, and Jian Tang. Gmnn: Graph markov neural networks. In *International conference on machine learning*, pages 5241–5250. PMLR, 2019.

[211] Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 35:14501–14515, 2022.

[212] N. Rao, H.-F. Yu, P. K. Ravikumar, and I. S. Dhillon. Collaborative filtering with graph information: Consistency and scalable methods. In *Proc. NIPS*, 2015.

[213] Hannah Rashkin, Eunsol Choi, Jin Yea Jang, Svitlana Volkova, and Yejin Choi. Truth of varying shades: Analyzing language in fake news and political fact-checking. In *Proc. Empirical Methods in Natural Language Processing*, pages 2931–2937, 2017.

[214] S Reck, D Guderian, G Vermariën, A Domi, KM3NeT Collaboration, et al. Graph neural networks for reconstruction and classification in km3net. *Journal of Instrumentation*, 16 (10):C10011, 2021.

[215] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of ADAM and beyond. 2018.

[216] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618*, 2012.

[217] Emanuele Rodolà, Samuel Rota Bulo, Thomas Windheuser, Matthias Vestner, and Daniel Cremers. Dense non-rigid shape correspondence using random forests. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4177–4184, 2014.

[218] Emanuele Rossi, Federico Monti, Michael Bronstein, and Pietro Liò. ncrna classification with graph convolutional networks. *arXiv preprint arXiv:1905.06515*, 2019.

[219] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. *Graph Representation Learning and Beyond, ICML Workshop*, 2020.

[220] Emanuele Rossi, Federico Monti, Yan Leng, Michael Bronstein, and Xiaowen Dong. Learning to infer structures of network games. In *International Conference on Machine Learning*, pages 18809–18827. PMLR, 2022.

[221] Victoria Rubin, Niall Conroy, Yimin Chen, and Sarah Cornwell. Fake news or truth? using satirical cues to detect potentially misleading news. In *Proc. Computational Approaches to Deception Detection*, pages 7–17, 2016.

[222] Natali Ruchansky, Sungyong Seo, and Yan Liu. Csi: A hybrid deep model for fake news. *arXiv:1703.06959*, 2017.

[223] Piotr Sapiezynski, Arkadiusz Stopczynski, David Dreyer Lassen, and Sune Lehmann. Interaction data from the copenhagen networks study. *Scientific Data*, 6(1):315, 2019.

[224] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[225] Kumar Sharad and George Danezis. An automated social graph de-anonymization technique. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pages 47–58, 2014.

[226] Yifei Shen, Yongji Wu, Yao Zhang, Caihua Shan, Jun Zhang, B Khaled Letaief, and Dongsheng Li. How powerful is graph convolution for recommendation? In *Proceedings of the 30th ACM international conference on information & knowledge management*, pages 1619–1629, 2021.

[227] Baoxu Shi and Tim Weninger. Fact checking in heterogeneous information networks. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 101–102, 2016.

[228] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*, 2020.

[229] Jonathan Shlomi, Peter Battaglia, and Jean-Roch Vlimant. Graph neural networks in particle physics. *Machine Learning: Science and Technology*, 2(2):021001, 2020.

[230] Kai Shu, H Russell Bernard, and Huan Liu. Studying fake news via network analysis: detection and mitigation. In *Emerging Research Challenges and Opportunities in Computational Social Network Analysis and Mining*, pages 43–65. Springer, 2019.

[231] Kai Shu, Suhang Wang, and Huan Liu. Beyond news contents: The role of social context for fake news detection. In *Proc. Web Search and Data Mining*, 2019.

[232] David I Shuman, Benjamin Ricaud, and Pierre Vandergheynst. Vertex-frequency analysis on graphs. *Applied and Computational Harmonic Analysis*, 40(2):260–291, 2016.

[233] Eero P Simoncelli and Bruno A Olshausen. Natural image statistics and neural representation. *Annual review of neuroscience*, 24(1):1193–1216, 2001.

[234] K. Singh, Vivek, Laura Freeman, Bruno Lepri, and Alex Sandy Pentland. Predicting spending behavior using socio-mobile features. *IEEE International Conference on Social Computing*, 2013.

[235] Harsh Sinha, Vinayak Awasthi, and Pawan K Ajmera. Audio classification using braided convolutional neural networks. *IET Signal Processing*, 14(7):448–454, 2020.

[236] N. Srebro, J. Rennie, and T. Jaakkola. Maximum-Margin Matrix Factorization. In *Proc. NIPS*, 2004.

[237] Jonathan M Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina M Donghia, Craig R MacNair, Shawn French, Lindsey A Carfrae, Zohar Bloom-Ackermann, et al. A deep learning approach to antibiotic discovery. *Cell*, 180(4):688–702, 2020.

[238] Chuxiong Sun, Hongming Gu, and Jie Hu. Scalable and adaptive graph neural networks with self-label-enhanced training. *arXiv preprint arXiv:2104.09376*, 2021.

[239] Mengzhu Sun, Xi Zhang, Jiaqi Zheng, and Guixiang Ma. Ddgcn: Dual dynamic graph convolutional networks for rumor detection on social media. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pages 4611–4619, 2022.

[240] Tiening Sun, Zhong Qian, Sujun Dong, Peifeng Li, and Qiaoming Zhu. Rumor detection on social media with graph adversarial contrastive learning. In *Proceedings of the ACM Web Conference 2022*, pages 2789–2797, 2022.

[241] Yiming Sun, Song Zixing, and Irwin King. Score-based graph generative model for neutrino events classification and reconstruction. *NeurIPS Machine Learning and the physical sciences Workshop*, 2021.

[242] Jan Svoboda, Federico Monti, and Michael M Bronstein. Generative convolutional networks for latent fingerprint reconstruction. In *2017 IEEE International joint conference on biometrics (IJCB)*, pages 429–436. IEEE, 2017.

[243] Jan Svoboda, Jonathan Masci, Federico Monti, Michael M Bronstein, and Leonidas Guibas. Peernets: Exploiting peer wisdom against adversarial attacks. *7th International Conference on Learning Representations, ICLR*, 2019.

[244] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International journal of uncertainty, fuzziness and knowledge-based systems*, 10(05):557–570, 2002.

[245] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[246] Lin Tian, Xiuzhen Jenny Zhang, and Jey Han Lau. Duck: Rumour detection on social media by modelling user and comment propagation networks. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4939–4949, 2022.

[247] Thanassis Tiropanis, Wendy Hall, Nigel Shadbolt, David De Roure, Noshir Contractor, and Jim Hendler. Computational fact checking from knowledge networks. 2015.

[248] F. Tombari, S. Salti, and L. Di Stefano. Unique signatures of histograms for local surface description. In *Proc. ECCV*, 2010.

[249] Arnaud J Tournier and Yves-Alexandre De Montjoye. Expanding the attack surface: Robust profiling attacks threaten the privacy of sparse behavioral data. *Science Advances*, 8 (33):eabl6464, 2022.

[250] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A closer look at spatiotemporal convolutions for action recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 6450–6459, 2018.

[251] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[252] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *Proc. ICLR*, 2018.

[253] Vikas Verma, Meng Qu, Kenji Kawaguchi, Alex Lamb, Yoshua Bengio, Juho Kannala, and Jian Tang. Graphmix: Improved training of gnns for semi-supervised learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 10024–10032, 2021.

[254] M. Vestner, R. Litman, A. Bronstein, E. Rodolà, and D. Cremers. Bayesian inference of bijective non-rigid shape correspondence. *arXiv:1607.03425*, 2016.

[255] Soroush Vosoughi, Deb Roy, and Sinan Aral. The spread of true and false news online. *Science*, 359(6380):1146–1151, 2018.

[256] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, pages 165–174, 2019.

[257] Xiyuan Wang and Muhan Zhang. How powerful are spectral graph neural networks. In *International Conference on Machine Learning*, pages 23341–23362. PMLR, 2022.

[258] Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. Inductive representation learning in temporal networks via causal anonymous walks. *arXiv preprint arXiv:2101.05974*, 2021.

[259] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph CNN for learning on point clouds. *arXiv:1801.07829*, 2018.

[260] Howard T Welser, Eric Gleave, Danyel Fisher, and Marc Smith. Visualizing the signatures of social roles in online discussion groups. *Journal of social structure*, 8(2):1–32, 2007.

[261] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pages 639–655. 2012.

[262] Felix Wong, Erica J Zheng, Jacqueline A Valeri, Nina M Donghia, Melis N Anahtar, Satotaka Omori, Alicia Li, Andres Cubillos-Ruiz, Aarti Krishnan, Wengong Jin, et al. Discovery of a structural class of antibiotics with explainable deep learning. *Nature*, 626(7997): 177–185, 2024.

[263] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR, 2019.

[264] Wayne Xiong, Lingfeng Wu, Fil Alleva, Jasha Droppo, Xuedong Huang, and Andreas Stolcke. The microsoft 2017 conversational speech recognition system. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5934–5938. IEEE, 2018.

[265] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. *arXiv preprint arXiv:2002.07962*, 2020.

[266] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[267] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International conference on machine learning*, pages 5453–5462. PMLR, 2018.

[268] M. Xu, R Jin, and Z.-H. Zhou. Speedup matrix completion with side information: Application to multi-label learning. In *Proc. NIPS*, 2013.

[269] F. Yanez and F. Bach. Primal-dual algorithms for non-negative matrix factorization with the kullback-leibler divergence. *arXiv:1412.1788*, 2012.

[270] Z. Yang, W. Cohen, and R. Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *Proc. ICML*, 2016.

[271] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34:28877–28888, 2021.

[272] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, 2018.

[273] 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.

[274] Jiaxuan You, Tianyu Du, and Jure Leskovec. Roland: graph learning framework for dynamic graphs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2358–2366, 2022.

[275] Chunyuan Yuan, Qianwen Ma, Wei Zhou, Jizhong Han, and Songlin Hu. Jointly embedding the local and global relations of heterogeneous graph for rumor detection. In *2019 IEEE international conference on data mining (ICDM)*, pages 796–805. IEEE, 2019.

[276] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

[277] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor K. Prasanna. Graphsaint: Graph sampling based inductive learning method. *arXiv:1907.04931*, 2019.

[278] Bingxu Zhang, Changjun Fan, Shixuan Liu, Kuihua Huang, Xiang Zhao, Jincai Huang, and Zhong Liu. The expressive power of graph neural networks: A survey. *arXiv preprint arXiv:2308.08235*, 2023.

[279] Shichang Zhang, Yozen Liu, Yizhou Sun, and Neil Shah. Graph-less neural networks: Teaching old mlps new tricks via distillation. *arXiv preprint arXiv:2110.08727*, 2021.

[280] Wentao Zhang, Ziqi Yin, Zeang Sheng, Yang Li, Wen Ouyang, Xiaosen Li, Yangyu Tao, Zhi Yang, and Bin Cui. Graph attention multi-layer perceptron. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 4560–4570, 2022.

[281] Xitong Zhang, Yixuan He, Nathan Brugnone, Michael Perlmutter, and Matthew Hirn. Magnet: A neural network for directed graphs. *Advances in neural information processing systems*, 34:27003–27015, 2021.

[282] Xinyi Zhou and Reza Zafarani. Network-based fake news detection: A pattern-driven approach. *ACM SIGKDD explorations newsletter*, 21(2):48–60, 2019.

[283] Xiaojin Zhu, Zoubin Ghahramani, John Lafferty, et al. Semi-supervised learning using gaussian fields and harmonic functions. In *Proc. ICML*, 2003.

[284] Marinka Zitnik and Jure Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33(14):i190–i198, Jul 2017. ISSN 1460-2059. doi: 10.1093/bioinformatics/btx252. URL `http://dx.doi.org/10.1093/bioinformatics/btx252`.